Decoding and Repair Schemes for Shift-XOR Regenerating Codes

Ximing Fu, Shenghao Yang, *Member, IEEE*, and Zhiqing Xiao

***Abstract*— Decoding and repair schemes are proposed for shift-exclusive-or (shift-XOR) product-matrix (PM) regenerating codes, which outperform the existing schemes in terms of both communication and computation costs. In particular, for the shift-XOR minimum bandwidth regenerating (MBR) codes, our decoding and repair schemes have the optimal transmission bandwidth and can be implemented in-place without extra storage space for intermediate XOR results. Technically, our schemes involve an in-place algorithm for solving a system of shift-XOR equations, called *shift-XOR elimination*, which does not have the bandwidth overhead generated by shift operations as in the previous zigzag algorithm and has lower computation complexities compared with the zigzag algorithm. The decoding and repair of shift-XOR MBR/MSR codes are decomposed into a sequence of systems of shift-XOR equations, and hence can be solved by a sequence of calls to the shift-XOR elimination. As the decompositions of the decoding and repair depend only on the PM construction, but not the specific shift and XOR operations, our decoding and repair schemes can be extended to other MBR/MSR codes using the PM construction. Due to its fundamental role, the shift-XOR elimination is of independent interest.**

***Index Terms*— Regenerating codes, shift-XOR regenerating codes, product-matrix construction, decoding, repair.**

I. Introduction

DISTRIBUTED storage systems with potential node failures usually use redundancy to ensure the reliability of the stored data. Compared with repetition, erasure coding is a more efficient approach to introduce redundancy. Using an $[n, k]$ maximum-distance separable (MDS) erasure code, a data file of kL bits is divided into k sequences, each of L bits. The k sequences are encoded into n coded sequences and stored in n nodes, each storing one coded sequence. A decoder can decode the data file from any k out of the n coded sequences. An $[n, k]$ MDS erasure code can tolerate at most $n-k$ node failures. Reed-Solomon codes [1] are widely used MDS erasure codes, where the encoding and decoding operations are over finite fields, and have high computation costs.

Towards low complexity codes, exclusive-or (XOR) and cyclic-shift operations have been employed to replace finite-field operations. One family of such MDS storage codes includes EVENODD codes [2] and RDP codes [3] for tolerating double node failures, and their extensions [4]–[6] for tolerating triple or more node failures. These codes share a common cyclic-shift Vandermonde generator matrix and the decoding complexity of these codes has been improved by LU factorization of Vandermonde matrix [7]. Another family of MDS storage codes using XOR and cyclic-shift is based on Cauchy generator matrices, including Cauchy Reed-Solomon [8], [9] and Rabin-like codes [10], where the decoding method is improved in [11].

In this article, we focus on a class of storage codes based on (non-cyclic) shift and XOR operations, called shift-XOR codes. Sung and Gong [12] presented a class of storage codes for any valid pair $[n, k]$ using shift and XOR operations, where the generator matrix satisfies the *increasing difference property*. Using an $[n, k]$ shift-XOR storage code, the kL bits of the data file can be decoded from any k out of the n coded sequences using the zigzag decoding algorithm [12]. Due to shift operations, the coded sequences are usually longer than L bits so that the shift-XOR storage codes are not strictly MDS, where the extra bits are called the *storage overhead*. An $[n, k]$ shift-XOR storage code is asymptotic MDS when L is large. Moreover, the total number of bits retrieved by the zigzag decoder is more than kL bits, where the extra bits are called the *(decoding) bandwidth overhead*.

Shift-XOR codes have attracted more research interests recently due to the potential low encoding/decoding computation costs. The shift-XOR storage codes with zigzag decoding have lower encoding and decoding complexities than Cauchy Reed-Solomon codes in a wide range of coding parameters [13]. A fountain code based on shift and XOR outperforms Raptor code in terms of the transmission overhead [14]. Efficient repair schemes for shift-XOR storage codes have been studied in [15]. Moreover, shift and XOR operations can also be used to construct network codes [16] and *regenerating codes* [17], [18].

Manuscript received July 3, 2019; revised April 30, 2020; accepted August 19, 2020. Date of publication August 24, 2020; date of current version November 20, 2020. This work was supported in part by NSFC under Grant 61471215 and in part by the Shenzhen Science and Technology Innovation Committee under Grant JCYJ20170817150000267, Grant ZDSYS20170725140921348, and Grant JCYJ20180508162604311. This article was presented in part at the 2014 IEEE TrustCom and in part at the 2015 IEEE ISIT. *(Corresponding author: Shenghao Yang.)*

Ximing Fu is with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen 518172, China, and also with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230052, China (e-mail: fuxm07@foxmail.com).

Shenghao Yang is with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen 518172, China, also with the Shenzhen Key Laboratory of IoT Intelligent Systems and Wireless Network Technology, The Chinese University of Hong Kong, Shenzhen 518172, China, and also with the Shenzhen Research Institute of Big Data, Shenzhen 518172, China (e-mail: shyang@cuhk.edu.cn).

Zhiqing Xiao was with Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: xzq.xiaozhiqing@gmail.com).

Communicated by A. Jiang, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2020.3019168

TABLE I

COMPARISON OF THE ALGORITHMS FOR DECODING A SHIFT-XOR STORAGE CODE IN TERMS OF BANDWIDTH OVERHEAD, DECODING AUXILIARY SPACE, AND DECODING COMPLEXITIES. HERE L IS THE MESSAGE SEQUENCE LENGTH, AND k IS THE NUMBER OF MESSAGE SEQUENCES. A VANDERMONDE GENERATOR MATRIX IS ASSUMED

Decoding Algorithm	Bandwidth Overhead	Decoding Auxiliary Space	Decoding XOR Operation	Decoding Integer Operation
shift-XOR elimination (Section III)	0	$O(1)$ integers	$< k(k-1)L$	$O(k^2L)$
zigzag decoding [12]	$i\,(k-1)$ bits for node i	$O\,(kL)$ integers and kL bits	$k(k+1)L$	$O(k^2L)$

For distributed storage system, it is also worthwhile to consider the *repair of failed nodes*. Dimakis *et al.* formulated regenerating codes to address this issue [19]. In an $[n, k, d]$ regenerating code, a data file of BL bits is divided into B sequences, each containing L bits. The sequences are encoded into $n\alpha$ sequences each of L bits and distributed to n storage nodes, each storing α sequences of L bits. The data file can be decoded from any k storage nodes, and a failed node can be repaired from any other d surviving nodes. There are two kinds of repair [19]: *exact repair* and *functional repair*. In exact repair, the sequences stored in the failed node can be exactly reconstructed in the new node. In functional repair, the sequences reconstructed in the new node may be different from those in the failed node as long as the new node and the other nodes form an $[n, k, d]$ regenerating code.

The tradeoff between the storage in a node and the repair bandwidth is characterized in [19]. Two classes of codes that achieve the optimal storage-repair-bandwidth tradeoff are of particular interests, i.e., the minimum bandwidth regenerating (MBR) codes and minimum storage regenerating (MSR) codes. Rashmi *et al.* [20] proposed product-matrix (PM) constructions of MBR codes for all valid tuples $[n, k, d]$ and MSR codes for $d \geq 2k-2$ with exact repair algorithms. In [20], PM MSR codes with $d > 2k-2$ are constructed based on the construction of the $d = 2k-2$ case. Two unified constructions of MSR codes for $d \geq 2k-2$ were proposed in [21], [22]. Based on special parameterized codes such as determinant codes with $k = d$ [23], Cascade codes were proposed to achieve the MSR tradeoff point with arbitrary feasible combinations of n, k, d [24]. Some MDS codes with sub-packetization were proposed to construct MSR codes [25]–[27].

The PM MBR/MSR codes require matrix operations over finite fields for encoding, decoding and repair, which have the high computation cost as Reed-Solomon codes. To achieve lower complexity, Hou *et al.* [17] proposed the regenerating codes using shift and XOR operations based on the PM construction. The shift-XOR MBR/MSR codes have the storage overhead and the decoding/repair bandwidth overhead. For example, for the shift-XOR MBR codes, extra $\frac{1}{2}k(k-1)$ sequences are retrieved for decoding, where the sequence length is at least L. Hou *et al.* [28] proposed another class of regenerating codes using cyclic-shift and XOR based on the PM construction, called *BASIC codes*, and demonstrated the lower computation costs than the finite-field PM regenerating codes. The BASIC MBR codes have the similar decoding bandwidth issue as the shift-XOR regenerating codes.

In [17], a general sufficient condition is provided such that a system of shift-XOR equations is uniquely solvable, which induces an adjoint matrix based approach to solve a system of shift-XOR equations. However, the adjoint matrix based approach has a high computation cost so that the decoding and repair computation costs of the shift-XOR regenerating codes in [17] are higher than those of the BASIC codes in [28] (see the comparison in Table II). In this article, we will show that it is possible to reduce the decoding/repair complexities of the shift-XOR regenerating codes to be better than or similar to those of BASIC codes.

A. Our Contributions

In this article, we first study solving a system of shift-XOR equations, where the generator matrix satisfies a refined version of the increasing difference property (see Section II). The *refined increasing difference (RID)* property relaxes the original one in [12] so that the storage overheads can be smaller. The RID property is satisfied by, for example, Vandermonde matrices. We propose an algorithm, called *shift-XOR elimination*, for solving such a system of shift-XOR equations (see Section III). Our algorithm has the following properties:

- Bandwidth overhead free: using the shift-XOR elimination to decode a shift-XOR storage code, only subsequences stored in a storage node are needed, and the number of bits retrieved from the storage nodes is equal to the number of bits to decode. In other words, the bandwidth costs of the shift-XOR elimination is optimal.
- Lower computational space and time complexities: the shift-XOR elimination has a smaller number of XOR operations and smaller auxiliary space compared with the zigzag algorithm. The number of XOR operations used by the shift-XOR elimination is the same as the number of XOR operations used to generate the input subsequences from the message sequences. The shift-XOR elimination only needs a constant number of auxiliary variables, and can be implemented in-place: the results of the intermediate XOR operations and the output bits are all stored at the same storage space of the input binary sequences. In other words, the algorithm uses an auxiliary space of $O(1)$ integers per kL bits to solve, and hence has an asymptotically optimal space cost as kL is large.

In Table I, the shift-XOR elimination and the zigzag algorithm are compared for decoding a shift-XOR storage code. The shift-XOR elimination can be used in other shift-XOR codes, e.g., [14], [15]. In this article, we focus on its application in regenerating codes.

TABLE II

COMPARISON AMONG DECODING AND REPAIR SCHEMES OF (CYCLIC-)SHIFT-XOR MBR/MSR CODES. HERE B IS THE NUMBER OF MESSAGE SEQUENCES, L IS THE MESSAGE SEQUENCE LENGTH, k IS THE NUMBER OF NODES FOR DECODING, n IS THE NUMBER OF STORAGE NODES, d IS THE NUMBER OF HELPER NODES FOR REPAIR, AND i IS THE NODE TO REPAIR. FOR MSR CODES, $d = 2k - 2$. A VANDERMONDE GENERATOR MATRIX IS ASSUMED

Algorithm		Bandwidth	Auxiliary Space for XOR results	XOR Operation
MBR Decoding	this paper (Section IV-A)	BL	0	$O(dk^2L)$
	[17]	$> BL + \frac{k(k-1)}{2}L$	$> BL + \frac{k(k-1)}{2}L$	$> d^2k^3nL$
	[28]	$> BL + \frac{k(k-1)}{2}L$	$> BL + \frac{k(k-1)}{2}L$	$O(dk^3L)$
MBR Repair	this paper (Section IV-B)	$d(L + (i-1)(d-1))$	0	$2d(d-1)L + O(nd^3)$
	[17]	$d(L + (i-1)(d-1)) + O(nd^2)$	$> dL$	$> d^4nL$
	[28]	dL	dL	$\left(\frac{15}{4}d - \frac{3}{4}\right)dL$
MSR Decoding	this paper (Section V-A2)	$k(k-1)L + O(nk^2d)$	$(k-1)(k-2)L + O(nk^2d)$	$O(k^3L + nk^3d)$
	[17]	$k(k-1)L + O(nk^2d)$	$k(k-1)L + O(nk^2d)$	$> dk^4nL$
	[28]	$k(k-1)L$	$k(k-1)L$	$O(k^4L)$
MSR Repair	this paper (Section V-B)	$dL + O(nd^2)$	$\frac{1}{2}dL + O(nd^2)$	$\frac{3}{2}(d-1)dL + O(nd^3)$
	[17]	$dL + O(nd^2)$	$dL + O(nd^2)$	$> d^4nL$
	[28]	dL	dL	$\left(\frac{23}{16}d + \frac{5}{8}\right)dL$

In Section IV and V, we study decoding and repair of the shift-XOR PM regenerating codes proposed in [17]. Our decoding/repair schemes transform the decoding/repair problem to a sequence of subproblems of shift-XOR equations, which can be solved using the shift-XOR elimination. Benefit from the advantages of the shift-XOR elimination, our schemes in general have lower computation and bandwidth costs than those in [17] (see Table II).

In particular, for the shift-XOR MBR codes, our decoding and repair schemes retrieve the same number of bits as the sequences to decode or repair. The decoding and repair schemes can be implemented in-place with only $O(1)$ integer auxiliary variables, but without any auxiliary variables to store the intermediate XOR results. Moreover, for decoding, the number of XOR operations is the same as the number of XOR operations for generating the input subsequences from the message sequences. For the shift-XOR MSR codes, our decoding and repair schemes have the same bandwidth cost, smaller auxiliary space for intermediate XOR results, and lower order of the number of XOR operations than those of [17].

With our decoding and repair schemes, shift-XOR regenerating codes can have better or similar performance compared with the BASIC codes [28], which use cyclic-shift and XOR. In particular, for MBR decoding and repair and MSR repair, our schemes have lower auxiliary spaces; for MBR decoding, our scheme has a smaller number of XOR operations. When L is sufficiently larger than nd, for MBR repair and MSR decoding and repair, our schemes have a similar or smaller number of XOR operations. See the comparison in Table II.

In Section VI, we discuss how to extend our decoding and repair schemes to some other MBR/MSR codes based on the PM construction, so that these codes can gain certain advantages we have for shift-XOR codes. For codes based on the PM construction in [20], [28], the decoding and repair schemes are the same except that the shift-XOR elimination is replaced by certain sub-processes for finite fields and cyclic-shift respectively.

II. SHIFT-XOR STORAGE CODES

In this section, we formulate shift-XOR storage codes [12], [17] after introducing some notations.

A. Notations

A range of integers from i to j is denoted by $i : j$. When $i > j$, $i : j$ is the empty set. A (binary) sequence is denoted by a bold lowercase letter, e.g., $\mathbf{a}$. The i-th entry of a sequence $\mathbf{a}$ is denoted by $\mathbf{a}[i]$. The subsequence of $\mathbf{a}$ from the i-th entry to the j-th entry is denoted by $\mathbf{a}[i : j]$.

For a sequence $\mathbf{a}$ of L bits and a natural number t, the *shift operator* z^t pads t zeros in front of $\mathbf{a}$, so that $z^t\mathbf{a}$ has $L + t$ bits and

$$\left(z^t\mathbf{a}\right)[l] = \begin{cases} 0, & 1 \le l \le t, \\ \mathbf{a}\left[l - t\right], & t < l \le L + t. \end{cases}$$

We use the convention that $\mathbf{a}[l] = 0$ for $l \notin 1 : L$, with which we can write

$$(z^t\mathbf{a})[l] = \mathbf{a}[l - t], \quad l = 1, \ldots, L + t.$$

Let $\mathbf{a}$ and $\mathbf{a}'$ be two sequences of length L and L', respectively. The *addition* of $\mathbf{a}$ and $\mathbf{a}'$, denoted by $\mathbf{a} + \mathbf{a}'$, is a sequence of $\max\{L, L'\}$ bits obtained by bit-wise exclusive-or (XOR), i.e., for $l \in 1 : \max\{L, L'\}$,

$$\left(\mathbf{a} + \mathbf{a}'\right)[l] = \mathbf{a}\left[l\right] \oplus \mathbf{a}'\left[l\right],$$

where we also use the convention that $\mathbf{a}[l] = 0$ for $l > L$ and $\mathbf{a}'[l] = 0$ for $l > L'$.

B. General Shift-XOR Storage Codes

We describe a general shift-XOR storage code and discuss special instances in the next subsection. The code has parameters B, L, d, n and α, which are positive integers. Consider a storage system of n storage nodes employing a shift-XOR storage code. A message is formed by B binary sequences, each of L bits. These B message sequences are organized

as a $d \times \alpha$ *message matrix* $\mathbf{M} = (\mathbf{m}_{i,j})$ in certain way to be described subsequently, where two entries may share the same message sequence, and certain entries may be the all-zero sequences. The *generator matrix* used for encoding the message is an $n \times d$ matrix $\mathbf{\Psi} = (z^{t_{i,j}})$, where $t_{i,j}$ are nonnegative integers to be explained further soon. For $1 \le i \le n, 1 \le j \le \alpha$, let

$$\mathbf{y}_{i,j} = \sum_{u=1}^{d} z^{t_{i,u}} \mathbf{m}_{u,j},$$

called the *coded sequence*. Denote $\mathbf{Y} = (\mathbf{y}_{i,j})$ as the $n \times \alpha$ coded matrix of sequences $\mathbf{y}_{i,j}$, which can be written in the matrix form

$$\mathbf{Y} = \mathbf{\Psi}\mathbf{M}. \tag{1}$$

The α sequences in the i-th row of $\mathbf{Y}$ are stored at the i-th node (also called node i) of the storage system.

Definition 1 (Refined Increasing Difference (RID) Property): Matrix $\mathbf{\Psi} = (z^{t_{i,j}})_{1 \le i \le n, 1 \le j \le d}$ is said to satisfy the *refined increasing difference (RID) property* if the following conditions hold: For any i, i', j, and j' such that $i < i'$ and $j < j'$,

$$0 \le t_{i,j'} - t_{i,j} < t_{i',j'} - t_{i',j},$$

where equality in the first inequality holds only when $i = 1$. We also say the numbers $t_{i,j}$, $1 \le i \le n$, $1 \le j \le d$ satisfy the RID property when they satisfy the above inequalities.

To guarantee certain efficient decoding algorithms, in this article, we require the generator matrix $\mathbf{\Psi}$ of a shift-XOR code satisfying the RID property. Different from the increasing difference property in [12], the RID property allows $t_{1,j'} - t_{1,j} = 0$ such that less storage at each node is required. Suppose $\alpha = 1$ and $B = d$, i.e., the d entries of $\mathbf{M}$ are independent message sequences. The shift-XOR storage codes of this case have been studied in [12], and the zigzag algorithm can decode the d message sequences from any d rows of $\mathbf{Y}$.

Due to shift operations, the length of a coded sequence can be more than L bits. In particular, the length of $\mathbf{y}_{i,j}$ is $L + t_{i,d}$. So the total number of bits stored at node i is $\alpha(L + t_{i,d})$. The extra $\alpha t_{i,d}$ bits stored at node i using a shift-XOR storage code is called the *storage overhead*. Under the constraint of the RID property, it can be argued that the generator matrix minimizing the storage overhead is [29]

$$\mathbf{\Psi} = (z^{(i-1)(j-1)}),$$

which is a *Vandermonde matrix*. In our analysis, we suppose $t_{i,j} = O(nd)$, which is feasible as we have a choice of $t_{i,j} = (i-1)(j-1) < nd$.

C. Shift-XOR Regenerating Codes

We discuss two classes of shift-XOR product-matrix (PM) regenerating codes [17]. The two constructions of the message matrix $\mathbf{M}$ are the same as those of the (finite-field) PM MBR and MSR codes [20]. According to the storage overhead discussed in the preceding section, these codes achieve the MBR/MSR tradeoff asymptotically when $L \to \infty$, so the constructed codes are called *shift-XOR MBR* codes and *shift-XOR MSR* codes, respectively.

In contrast, regenerating codes using cyclic-shift-and-XOR operations [28] and finite-field operations [20] do not have storage overhead. Though with the storage overhead, the shift-XOR codes have the potential of low encoding/decoding complexity, to be demonstrated by the schemes of this article.

1) Shift-XOR MBR Codes: Fix an integer k with $k \le d$. Consider $\alpha = d$ and

$$B = \frac{1}{2}(k+1)k + k(d-k). \tag{2}$$

The message matrix $\mathbf{M}$ is of the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{S} & \mathbf{T} \\ \mathbf{T}^\top & \mathbf{O} \end{bmatrix}, \tag{3}$$

where $\mathbf{S}$ is a $k \times k$ symmetric matrix of the first $\frac{1}{2}(k+1)k$ message sequences, $\mathbf{T}$ is a $k \times (d-k)$ matrix of the remaining $k(d-k)$ message sequences, $\mathbf{O}$ is a $(d-k) \times (d-k)$ matrix of the zero sequence $\mathbf{0}$, and $\mathbf{T}^\top$ is the transpose of $\mathbf{T}$. By (2), all the B message sequences are used in $\mathbf{M}$.

The shift-XOR MBR code has the coded sequences $\mathbf{Y} = \mathbf{\Psi}\mathbf{M}$, where $\mathbf{\Psi} = (z^{t_{i,j}})$ is an $n \times d$ matrix satisfying the RID property. A shift-XOR MBR code has the parameters n, k, d and L, and is usually referred to as an $[n, k, d]$ code.

Example 1 ([6,3,4] Shift-XOR MBR Code): For a shift-XOR MBR code with $n = 6$, $k = 3$, $d = 4$, the message matrix is of the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_7 \\ \mathbf{x}_2 & \mathbf{x}_4 & \mathbf{x}_5 & \mathbf{x}_8 \\ \mathbf{x}_3 & \mathbf{x}_5 & \mathbf{x}_6 & \mathbf{x}_9 \\ \mathbf{x}_7 & \mathbf{x}_8 & \mathbf{x}_9 & \mathbf{0} \end{bmatrix},$$

where $\mathbf{x}_i$ is a binary sequence of L bits. Using the Vandermonde generator matrix

$$\mathbf{\Psi} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 \\ 1 & z^2 & z^4 & z^6 \\ 1 & z^3 & z^6 & z^9 \\ 1 & z^4 & z^8 & z^{12} \\ 1 & z^5 & z^{10} & z^{15} \end{bmatrix},$$

the coded sequences stored at node i ($1 \le i \le 6$) are

$$\begin{cases} \mathbf{y}_{i,1} = \mathbf{x}_1 + z^{i-1}\mathbf{x}_2 + z^{2(i-1)}\mathbf{x}_3 + z^{3(i-1)}\mathbf{x}_7, \\ \mathbf{y}_{i,2} = \mathbf{x}_2 + z^{i-1}\mathbf{x}_4 + z^{2(i-1)}\mathbf{x}_5 + z^{3(i-1)}\mathbf{x}_8, \\ \mathbf{y}_{i,3} = \mathbf{x}_3 + z^{i-1}\mathbf{x}_5 + z^{2(i-1)}\mathbf{x}_6 + z^{3(i-1)}\mathbf{x}_9, \\ \mathbf{y}_{i,4} = \mathbf{x}_7 + z^{i-1}\mathbf{x}_8 + z^{2(i-1)}\mathbf{x}_9. \end{cases}$$

2) Shift-XOR MSR Codes: Here we only discuss shift-XOR MSR codes with $d = 2k - 2$ and $\alpha = k - 1$. Codes with $d \ge 2k-1$ can be constructed using the method in [20] based on the codes with $d = 2k - 2$ and $\alpha = k - 1$. Let $B = k\alpha = (\alpha + 1)\alpha$. The message matrix $\mathbf{M}$ is of the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{S} \\ \mathbf{T} \end{bmatrix}, \tag{4}$$

where $\mathbf{S}$ is an $\alpha \times \alpha$ symmetric matrix of the first $\frac{1}{2}\alpha(\alpha+1)$ message sequences, and $\mathbf{T}$ is another $\alpha \times \alpha$ symmetric matrix of the remaining $\frac{1}{2}\alpha(\alpha+1)$ message sequences.

The generator matrix $\mathbf{\Psi}$ is of the form

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{\Phi} & \mathbf{\Lambda}\mathbf{\Phi} \end{bmatrix}, \tag{5}$$

where $\mathbf{\Phi} = (z^{t_{i,j}})$ is an $n \times \alpha$ matrix satisfying the RID property, and $\mathbf{\Lambda}$ is an $n \times n$ diagonal matrix with diagonal entries $z^{\lambda_1}, z^{\lambda_2}, \ldots, z^{\lambda_n}$ such that $\mathbf{\Psi}$ satisfies the RID property. When $\mathbf{\Phi} = \left(z^{(i-1)(j-1)}\right)$ and $\lambda_i = (i-1)\alpha$, $\mathbf{\Psi}$ is a Vandermonde matrix, for which the storage overheads are minimal as we have discussed.

The shift-XOR MSR code described above has the coded sequences $\mathbf{Y} = \mathbf{\Psi}\mathbf{M}$, and is usually referred to as an $[n, k, d]$ code.

Example 2 ([6,3,4] Shift-XOR MSR Code): For a shift-XOR MSR code with $n = 6$, $k = 3$, $d = 4$ and $\alpha = k - 1 = 2$, the message matrix is

$$\mathbf{M} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \\ \mathbf{x}_2 & \mathbf{x}_3 \\ \mathbf{x}_4 & \mathbf{x}_5 \\ \mathbf{x}_5 & \mathbf{x}_6 \end{bmatrix},$$

where $\mathbf{x}_i$ is a binary sequence of L bits. Let

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 1 \\ 1 & z \\ 1 & z^2 \\ 1 & z^3 \\ 1 & z^4 \\ 1 & z^5 \end{bmatrix}.$$

and $\mathbf{\Lambda} = \operatorname{diag}\left\{1, z^2, z^4, z^6, z^8, z^{10}\right\}$. Then the generator matrix $\mathbf{\Psi}$ is

$$\mathbf{\Psi} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 \\ 1 & z^2 & z^4 & z^6 \\ 1 & z^3 & z^6 & z^9 \\ 1 & z^4 & z^8 & z^{12} \\ 1 & z^5 & z^{10} & z^{15} \end{bmatrix}.$$

The coded sequences stored at node i $(1 \le i \le 6)$ are

$$\begin{cases} \mathbf{y}_{i,1} = \mathbf{x}_1 + z^{i-1}\mathbf{x}_2 + z^{2(i-1)}\mathbf{x}_4 + z^{3(i-1)}\mathbf{x}_5, \\ \mathbf{y}_{i,2} = \mathbf{x}_2 + z^{i-1}\mathbf{x}_3 + z^{2(i-1)}\mathbf{x}_5 + z^{3(i-1)}\mathbf{x}_6. \end{cases}$$

III. Solving a System of Shift-XOR Equations

Before introducing the decoding and repair schemes of the shift-XOR regenerating codes, we give an algorithm for solving a system of shift-XOR equations, called *shift-XOR elimination*. This algorithm will be used as a sub-process of our subsequent decoding and repair schemes, and is of independent interest due to its fundamental role.

A $k \times k$ system of shift-XOR equations is given by

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_k \end{bmatrix} = \mathbf{\Psi} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_k \end{bmatrix}, \tag{6}$$

where $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ are binary sequences of L bits, and matrix $\mathbf{\Psi} = (z^{t_{i,j}})$ satisfies the RID property given in Definition 1. The problem of solving a system of shift-XOR equations (6) is to calculate $\mathbf{x}_i$, $i = 1, \ldots, k$ for given $\mathbf{y}_i$, $i = 1, \ldots, k$ and $\mathbf{\Psi}$.

A. Zigzag Algorithm

One approach to solve the system is the zigzag algorithm [12], which performs successive cancellation. We use an example to illustrate the idea of the zigzag algorithm.

Example 3: Consider the 3×3 system

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \end{bmatrix} = \begin{bmatrix} 1 & z & z^2 \\ 1 & z^2 & z^4 \\ 1 & z^3 & z^6 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix}. \tag{7}$$

Table III illustrates how the bits in $\mathbf{y}_i$ are aligned with bits in $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. We see that $\mathbf{x}_1[1] = \mathbf{y}_i[1]$, for $i = 1, 2, 3$, and hence $\mathbf{x}_1[1]$ is solvable. Next, we see that $\mathbf{x}_1[2] = \mathbf{y}_i[2]$, for $i = 2, 3$ and hence $\mathbf{x}_1[2]$ is solvable. Substituting $\mathbf{x}_1[2]$ back into $\mathbf{y}_1$, we further obtain $\mathbf{x}_2[1] = \mathbf{y}_1[2] - \mathbf{x}_1[2]$. This process can be repeated to solve all the bits in $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$: in every iteration, a solvable bit is found and is substituted back to all the equations it involves in. When there are more than one solvable bits in an iteration, one of them is chosen for substitution.

The zigzag algorithm in [12] implements the above idea to solve any $k \times k$ system of shift-XOR equations as defined in (6). The zigzag algorithm, however, is not optimal in several aspects. First, the zigzag algorithm needs all the $kL + \sum_{i=1}^{k} t_{i,k}$ bits of $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k$ to solve the kL bits of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$. The extra $\sum_{i=1}^{k} t_{i,k}$ bits consumed by the algorithm is called the *communication overhead* or *bandwidth overhead*. The ideal case is to have zero bandwidth overhead, same as solving a full-rank $k \times k$ system of linear equations over a finite field.

Second, the space and time computation complexities of the zigzag algorithm have room to improve. The zigzag algorithm [12] runs in kL iterations to solve all the kL bits. In each iteration, a solvable bit is found to back substitute into all the related equations. In [12], two approaches for searching the solvable bit are discussed. The first approach takes $O(k^2)$ comparisons to find a solvable bit and results in totally $O(k^3 L)$ time complexity. To reduce the time complexity, the second approach uses a pre-calculated array of $O(kL)$ integers to assist the searching process. Searching and updating the array takes $O(k)$ integer operations in each iteration, so that the time complexity is $O(k^2 L)$. In addition to the input and output sequences, the second approach requires $O(kL)$ auxiliary space to store the integer array. Without otherwise specified, we refer to the second approach as the zigzag algorithm.

B. Shift-XOR Elimination

Here we propose an algorithm to solve systems of shift-XOR equations, called *shift-XOR elimination*, which improves the zigzag algorithm in terms of both bandwidth overhead and computation complexities. First, only a subsequence of L bits of $\mathbf{y}_i$ $(i = 1, \ldots, k)$ is used so that the shift-XOR elimination has no bandwidth overhead. Second, the order of the bits to solve follows a regular pattern so that the shift-XOR elimination has lower computation time and space costs than the zigzag algorithm. We use an example to illustrate our algorithm.

TABLE III

THE THREE TABLES ILLUSTRATE HOW $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$ ARE FORMED BY $\mathbf{x}_1$, $\mathbf{x}_2$ AND $\mathbf{x}_3$. FOR NUMBER l IN THE ROW OF $\mathbf{y}_i$, LET l_j BE THE NUMBER IN THE SAME COLUMN OF l AND IN THE ROW OF $\mathbf{x}_j$. THEN THE TABLE TELLS THAT $\mathbf{y}_i[l] = \mathbf{x}_1[l_1] + \mathbf{x}_2[l_2] + \mathbf{x}_3[l_3]$. FOR EXAMPLE, THE 2ND, 3RD AND 4TH COLUMNS OF TABLE IIIa MEAN $\mathbf{y}_1[1] = \mathbf{x}_1[1]$, $\mathbf{y}_1[2] = \mathbf{x}_1[2] + \mathbf{x}_2[1]$ AND $\mathbf{y}_1[3] = \mathbf{x}_1[3] + \mathbf{x}_2[2] + \mathbf{x}_3[1]$, RESPECTIVELY

(a) $\mathbf{y}_1 = \mathbf{x}_1 + z\mathbf{x}_2 + z^2\mathbf{x}_3$

$\mathbf{y}_1$	1	2	3	4	5	$\cdots$	L	$L+1$	$L+2$
$\mathbf{x}_1$	1	2	3	4	5	$\cdots$	L		
$\mathbf{x}_2$		1	2	3	4	$\cdots$	$L-1$	L	
$\mathbf{x}_3$			1	2	3	$\cdots$	$L-2$	$L-1$	L

(b) $\mathbf{y}_2 = \mathbf{x}_1 + z^2\mathbf{x}_2 + z^4\mathbf{x}_3$

$\mathbf{y}_2$	1	2	3	4	5	$\cdots$	L	$L+1$	$L+2$	$L+3$	$L+4$
$\mathbf{x}_1$	1	2	3	4	5	$\cdots$	L				
$\mathbf{x}_2$			1	2	3	$\cdots$	$L-2$	$L-1$	L		
$\mathbf{x}_3$					1	$\cdots$	$L-4$	$L-3$	$L-2$	$L-1$	L

(c) $\mathbf{y}_3 = \mathbf{x}_1 + z^3\mathbf{x}_2 + z^6\mathbf{x}_3$

$\mathbf{y}_3$	1	2	3	4	5	6	7	$\cdots$	L	$L+1$	$L+2$	$L+3$	$\cdots$
$\mathbf{x}_1$	1	2	3	4	5	6	7	$\cdots$	L				
$\mathbf{x}_2$				1	2	3	4	$\cdots$	$L-3$	$L-2$	$L-1$	L	
$\mathbf{x}_3$							1	$\cdots$	$L-6$	$L-5$	$L-4$	$L-3$	$\cdots$

Example 4: Consider the system in (7). As illustrated in Table III, $\mathbf{y}_1[1]$, $\mathbf{y}_2[1]$ and $\mathbf{y}_3[1]$ are all equal to $\mathbf{x}_1[1]$ and hence one of them is sufficient for solving $\mathbf{x}_1[1]$ and other two are redundant. Similarly, $\mathbf{y}_2[2]$ and $\mathbf{y}_3[2]$ are the same, and one of them is redundant. Define subsequences

$$\hat{\mathbf{x}}_1 = \mathbf{y}_3[1:L],$$
$$\hat{\mathbf{x}}_2 = \mathbf{y}_2[3:(L+2)],$$
$$\hat{\mathbf{x}}_3 = \mathbf{y}_1[3:(L+2)].$$

Table IV illustrates how $\hat{\mathbf{x}}_i$ is formed by $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. In particular, for $l \in 1:L$,

$$\hat{\mathbf{x}}_1[l] = \mathbf{x}_1[l] + \mathbf{x}_2[l-3] + \mathbf{x}_3[l-6],$$
$$\hat{\mathbf{x}}_2[l] = \mathbf{x}_2[l] + \mathbf{x}_1[l+2] + \mathbf{x}_3[l-2],$$
$$\hat{\mathbf{x}}_3[l] = \mathbf{x}_3[l] + \mathbf{x}_1[l+2] + \mathbf{x}_2[l+1].$$

We see that $\hat{\mathbf{x}}_i$ involves all the bits in $\mathbf{x}_i$.

Let's illustrate how to solve the system using $\hat{\mathbf{x}}_i$, $i = 1, 2, 3$. The system is solved in multiple iterations indexed by $s = 1, 2, \ldots$, which can be further separated into three phases:

1) For each iteration $s = 1, 2$, one bit in $\mathbf{x}_1$ is solved.
2) For the iteration $s = 3$, one bit is solved in $\mathbf{x}_1$ and one bit is solved in $\mathbf{x}_2$ sequentially.
3) For each iteration $s \geq 4$, one bit is solved from each of $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$ sequentially. (When $l > L$, $\mathbf{x}_i[l]$ is supposed to be solvable.)

Here, a bit is solved implies that it is also back substituted into the equations it involves. Table IV illustrates this order of bit solving. The first two iterations form the first phase, where $\mathbf{x}_1[1]$ and $\mathbf{x}_1[2]$ are solved. The third iteration forms the second phase, where $\mathbf{x}_1[3]$ is solved first and then $\mathbf{x}_2[1]$ is solved by substituting $\mathbf{x}_1[3]$. Other iterations form the third phase. At iteration 4, for example, $\mathbf{x}_1[4]$ is first solved by substituting $\mathbf{x}_2[1]$; $\mathbf{x}_2[2]$ is then solved by substituting $\mathbf{x}_1[4]$; last, $\mathbf{x}_3[1]$ is solved by substituting $\mathbf{x}_1[3]$ and $\mathbf{x}_2[2]$.

TABLE IV

SOLVING SYSTEM (7) BY THE SHIFT-XOR ELIMINATION FOR THE FIRST 10 ITERATIONS. THE FIRST ROW GIVES THE THREE PHASES OF THE ITERATIONS, AND THE SECOND ROW GIVES THE ITERATIONS s. THE THREE ROWS FOLLOWING $\hat{\mathbf{x}}_i$ $(i = 1, 2, 3)$ SHOW HOW $\hat{\mathbf{x}}_i$ IS FORMED BY $\mathbf{x}_j$, $j = 1, 2, 3$. FOR NUMBER l IN THE ROW OF $\hat{\mathbf{x}}_i$, LET l_j BE THE NUMBER IN THE SAME COLUMN OF l AND IN THE ROW OF $\mathbf{x}_j$ FOLLOWING $\hat{\mathbf{x}}_i$. THEN THE TABLE TELLS THAT $\hat{\mathbf{x}}_i[l] = \mathbf{x}_1[l_1] + \mathbf{x}_2[l_2] + \mathbf{x}_3[l_3]$. FOR EACH ITERATION, THE BITS DECODED ARE SPECIFIED BY THE ENTRIES IN THE SAME COLUMN AND IN THE GRAY ROWS. FOR EXAMPLE, FROM THE COLUMN INDEXED BY $s = 4$, THE THREE GRAY ENTRIES ARE 4, 2 AND 1, WHERE THE ENTRY 4 IN THE ROW OF $\hat{\mathbf{x}}_1$ MEANS THAT $\mathbf{x}_1[4]$ CAN BE SOLVED BY SUBSTITUTING THE PREVIOUS SOLVED BITS INTO $\hat{\mathbf{x}}_1[4]$

phase	1		2	3							
iteration s	1	2	3	4	5	6	7	8	9	10	$\cdots$
$\hat{\mathbf{x}}_1$	1	2	3	4	5	6	7	8	9	10	$\cdots$
$\mathbf{x}_1$	1	2	3	4	5	6	7	8	9	10	$\cdots$
$\mathbf{x}_2$				1	2	3	4	5	6	7	$\cdots$
$\mathbf{x}_3$							1	2	3	4	$\cdots$
$\hat{\mathbf{x}}_2$			1	2	3	4	5	6	7	8	$\cdots$
$\mathbf{x}_1$			3	4	5	6	7	8	9	10	$\cdots$
$\mathbf{x}_2$			1	2	3	4	5	6	7	8	$\cdots$
$\mathbf{x}_3$					1	2	3	4	5	6	$\cdots$
$\hat{\mathbf{x}}_3$				1	2	3	4	5	6	7	$\cdots$
$\mathbf{x}_1$				3	4	5	6	7	8	9	$\cdots$
$\mathbf{x}_2$				2	3	4	5	6	7	8	$\cdots$
$\mathbf{x}_3$				1	2	3	4	5	6	7	$\cdots$

Now we introduce the general shift-XOR elimination for solving (6). The algorithm uses the subsequence $\hat{\mathbf{x}}_i$, $i = 1, \ldots, k$ defined as

$$\hat{\mathbf{x}}_i = \mathbf{y}_{k+1-i}\left[t_{k+1-i,i} + (1:L)\right], \tag{8}$$

where $t_{k+1-i,i}+(1:L)$ denotes $(t_{k+1-i,i}+1):(t_{k+1-i,i}+L)$. As $\hat{\mathbf{x}}_i$ has exactly L bits, our algorithm needs exactly kL input bits and hence achieves zero bandwidth overhead.

Substituting (6) into (8), we have for $1 \le l \le L$,

$$\hat{\mathbf{x}}_i[l] = \mathbf{x}_i[l] + \sum_{j \neq i} \mathbf{x}_j[l - t_{k+1-i,j} + t_{k+1-i,i}], \qquad (9)$$

where we use the convention that $\mathbf{x}_i[l] = 0$ for $l \le 0$ and $l > L$.

The shift-XOR elimination solves $\hat{\mathbf{x}}_i$, $i = 1, \ldots, k$ as follows. The algorithm runs in a number of iterations indexed by $s = 1, 2 \ldots$, which are partitioned into k phases. For $b = 1, 2, \ldots, k$, define

$$L_b = \begin{cases} t_{k-b,b+1} - t_{k-b,b}, & 1 \le b < k, \\ L, & b = k, \end{cases} \qquad (10)$$

and define $L_{1:b} = \sum_{b'=1}^{b} L_{b'}$. The b-th phase ($b \in 1 : k$) has L_b iterations. The operations in each iteration are specified as follows:

- For each iteration s in phase 1 (i.e., $s \in 1 : L_1$), $\mathbf{x}_1[s]$ is solved (using $\hat{\mathbf{x}}_1[s]$).
- For each iteration s in phase $b = 2, \ldots, k$ (i.e., $s \in L_{1:(b-1)} + (1 : L_b)$), $\mathbf{x}_i[s - L_{1:(i-1)}]$ is solved (using $\hat{\mathbf{x}}_i[s - L_{1:(i-1)}]$ and the previously solved bits) sequentially for $i = 1, \ldots, b$.

In the above process, i) a bit is solved implies that it is also back substituted into the equations it involves in; ii) for $\mathbf{x}_i[l]$ with $l > L$, $\mathbf{x}_i[l]$ is supposed to be solvable as 0, and hence do not need to be substituted; iii) the total number of iterations is $L_{1:k}$.

Algorithm 1 Shift-XOR Elimination With in-Place Implementation. After the Execution, the Value $\mathbf{x}_i[l]$ Is Stored at the Same Storage Space as $\hat{\mathbf{x}}_i[l]$

Input: sequences $\hat{\mathbf{x}}_i$, $1 \le i \le k$.
Output: solved sequences $\mathbf{x}_i$, $1 \le i \le k$.
1: Initialize $s \leftarrow 0$
2: **for** $b \leftarrow 1 : k$ **do**
3: **for** L_b iterations **do**
4: $s \leftarrow s + 1$;
5: **for** $i \leftarrow 1 : b$ **do**
6: $l \leftarrow s - L_{1:(i-1)}$; (the value of $\mathbf{x}_i[l]$ is stored at the same place of $\hat{\mathbf{x}}_i[l]$)
7: **for** $j \leftarrow 1, 2, \ldots, i-1, i+1, \ldots, k$ **do**
8: **if** $0 < l + t_{k+1-j,i} - t_{k+1-j,j} \le L$ **then**
9: $\hat{\mathbf{x}}_j[l + t_{k+1-j,i} - t_{k+1-j,j}] \oplus \leftarrow \mathbf{x}_i[l]$; (in-place back substitution)

A pseudocode of the shift-XOR elimination is given in Algorithm 1, which also demonstrates an *in-place* implementation of the shift-XOR elimination. The loop started at Line 2 enumerates all the phases $b = 1, \ldots, k$. The operations in each iteration are given from Line 3 to 9: In Line 6, one more bit is marked to be solved, and the following three lines perform back substitution. For in-place implementation, the back substitution result $\hat{\mathbf{x}}_v[l + t_{k+1-v,i} - t_{k+1-v,v}] \oplus \mathbf{x}_i[l]$ is stored at the same place of $\hat{\mathbf{x}}_v[l + t_{k+1-v,i} - t_{k+1-v,v}]$. After the execution of the algorithm, the value $\mathbf{x}_i[l]$ is stored at the same storage space as $\hat{\mathbf{x}}_i[l]$.

To prove the correctness of the shift-XOR elimination, we only need to show that each bit chosen to solve during the execution of the algorithm can be expressed as $\hat{\mathbf{x}}_i$, $i = 1, \ldots, k$ and the previously solved bits. Theorem 1, proved in Appendix, justifies the shift-XOR elimination.

Theorem 1: Consider a $k \times k$ system of shift-XOR equations $(\mathbf{y}_1 \ \cdots \ \mathbf{y}_k)^\top = \mathbf{\Psi}(\mathbf{x}_1 \ \cdots \ \mathbf{x}_k)^\top$ with $\mathbf{\Psi}$ satisfying the RID property. The shift-XOR elimination can successfully solve $\mathbf{x}_i$, $i = 1, \ldots, k$ using

$$\hat{\mathbf{x}}_i = \mathbf{y}_{k+1-i}\left[t_{k+1-i,i} + (1 : L)\right], i = 1, \ldots, k.$$

We summarize the bandwidth and computation costs of the shift-XOR elimination:

- First, the algorithm has no bandwidth overhead as the number of input bits is the same as the number of bits to solve. In contrast, the bandwidth overhead of the zigzag algorithm has $\sum_{i=1}^{k} t_{i,k}$ bits.
- Second, as the algorithm can be implemented in-place, no extra storage space is required to store the intermediate XOR results. The algorithm only needs a small constant number (independent of k and L) of intermediate integer variables. (The values $t_{i,j}$, L_b and $L_{1:b}$ are constants that included as a part of the program that implements the algorithm.) Therefore, the shift-XOR elimination uses $O(1)$ auxiliary integer variables. In contrast, the zigzag algorithm needs $O(kL)$ auxiliary integer variables, and kL bits to store the intermediate XOR results.
- Third, the number of XOR operations used by Algorithm 1 is the same as the number of XOR operations used to generate $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_k$ from $\mathbf{x}_1, \ldots, \mathbf{x}_k$, and is less than $k(k-1)L$. The number of integer operations used in the algorithm (for calculating back substitution positions) is $O(k^2 L)$. Similarly, the zigzag algorithm needs $k(k+1)L$ XOR operations and $O(k^2 L)$ integer operations.

To conclude this section, we remark that there are other choices of the subsequences that can guarantee the solvability. But different subsequences may result in different order of the bits to solve, which we would not explore in this article. For example, if we use $\mathbf{y}_u[t_{u,u} + (1 : L)]$, $u = 1, \ldots, k$, each sequence $\mathbf{x}_i$ can be solved from the last bit to the first bit.

IV. Decoding and Repair Schemes of Shift-XOR MBR Codes

In this section, we discuss the decoding and repair schemes for the shift-XOR MBR codes described in Section II-C.1. Our schemes decompose the decoding/repair problem into a sequence of systems of shift-XOR equations, each of which can be solved efficiently using the shift-XOR elimination discussed in the last section.

Let m, n be positive integers, and let $\mathbf{A} = (a_{i,j})$ be an $m \times n$ matrix. We define some notations to represent submatrices of $\mathbf{A}$. For two subsets $I \subset \{1, \ldots, n\}$ and $J \subset \{1, \ldots, m\}$, let $\mathbf{A}_I$ (resp. $\mathbf{A}^J$) be the submatrix of $\mathbf{A}$ formed by all the columns (resp. rows) with indices in I (resp. J). Following these notations, $\mathbf{A}_I^J$ is the submatrix of $\mathbf{A}$ formed by the entries on the rows in J and columns in I. When $I = \{i\}$

(resp. $J=\{j\}$), we also write $\mathbf{A}_i$ (resp. $\mathbf{A}^j$) for convenience. These submatrix notations should not be confused with the matrix entries (e.g., $a_{i,j}$), which are specified case by case before using.

A. Decoding Scheme of Shift-XOR MBR Codes

Consider an $[n,k,d]$ shift-XOR MBR codes. According to the encoding (1) with the symmetric matrix $\mathbf{M}=(\mathbf{m}_{i,j})$ in (3), we get

$$\mathbf{Y}=\mathbf{\Psi}\begin{bmatrix}\mathbf{S} & \mathbf{T}\\ \mathbf{T}^\top & \mathbf{O}\end{bmatrix}, \tag{11}$$

where $\mathbf{\Psi}=(z^{t_{i,j}})$ is an $n\times d$ matrix satisfying the RID property. We see that for $1\le i\le j\le k$, the (i,j) entry of the symmetric matrix $\mathbf{S}$ is $\mathbf{m}_{i,j}$ and for $1\le i\le k, 1\le j\le d-k$, the (i,j) entry of $\mathbf{T}$ is $\mathbf{m}_{i,j+k}$. Due to the symmetry of $\mathbf{S}$, there are totally $B=\frac{1}{2}(k+1)k+k(d-k)$ message sequences to decode. Substituting the entries $\mathbf{m}_{i,j}$ into (11), the (i,j) entry of $\mathbf{Y}$ is

$$\mathbf{y}_{v,u}[l]=\begin{cases}\sum_{j=1}^{u}\mathbf{m}_{j,u}[l-t_{v,j}]\\ \quad+\sum_{j=u+1}^{d}\mathbf{m}_{u,j}[l-t_{v,j}], & 1\le u\le k\\ \sum_{j=1}^{k}\mathbf{m}_{j,u}[l-t_{v,j}], & k<u\le d\end{cases} \tag{12}$$

where $1\le l\le L+t_{v,d}$.

1) Decomposition of Decoding Problem: Our scheme decodes the message matrix $\mathbf{M}$ by first decoding the $k\times(d-k)$ submatrix $\mathbf{T}$ and then decoding the $k\times k$ symmetric submatrix $\mathbf{S}$. For each $j=1,\dots,d-k$, we have the system of shift-XOR equations

$$\mathbf{Y}_{j+k}=\mathbf{\Psi}_{1:k}\mathbf{T}_j, \tag{13}$$

where $\mathbf{\Psi}_{1:k}$ satisfies the RID property. Using the shift-XOR elimination, $\mathbf{T}_j$ can be decoded from any k rows of (13). The $d-k$ columns of $\mathbf{T}$ can be decoded one-by-one or in parallel.

After decoding $\mathbf{T}$, we continue to decode the k columns of $\mathbf{S}$ sequentially with the column indices in descending order. Fix u with $1\le u\le k$. We have, according to (11),

$$\begin{aligned}\mathbf{Y}_u&=\mathbf{\Psi}_{1:k}\mathbf{S}_u+\mathbf{\Psi}_{(k+1):d}(\mathbf{T}^u)^\top\\ &=\mathbf{\Psi}_{1:u}\mathbf{S}_u^{1:u}+\mathbf{\Psi}_{(u+1):k}\mathbf{S}_u^{(u+1):k}+\mathbf{\Psi}_{(k+1):d}(\mathbf{T}^u)^\top.\end{aligned}$$

Then we have

$$\mathbf{Y}_u-\mathbf{\Psi}_{(u+1):k}\mathbf{S}_u^{(u+1):k}-\mathbf{\Psi}_{(k+1):d}(\mathbf{T}^u)^\top=\mathbf{\Psi}_{1:u}\mathbf{S}_u^{1:u}. \tag{14}$$

As $\mathbf{\Psi}_{1:u}$ satisfies the RID property, (14) can be viewed as a system of shift-XOR equations generated by $\mathbf{\Psi}_{1:u}$ of the input $\mathbf{S}_u^{1:u}$. When $u=k$, $\mathbf{S}_k$ can be decoded using the shift-XOR elimination on any k rows of $\mathbf{Y}_u-\mathbf{\Psi}_{(k+1):d}(\mathbf{T}^u)^\top$. When $u<k$, suppose that the columns of $\mathbf{S}$ with indices larger than u have all been decoded. Then $\mathbf{S}_u^{1:u}$ can be decoded using the shift-XOR elimination on any u rows of (14), where $\mathbf{S}_u^{(u+1):k}=\left(\mathbf{S}^u_{(u+1):k}\right)^\top$ have been decoded.

2) Decoding Scheme: Our decoding scheme is able to decode the message sequences by retrieving data from any k out of the n nodes. Now we give the details of our decoding scheme, which consists of two stages: the transmission stage and the decoding stage. In the transmission stage, k storage nodes are chosen. Let the indices of the k nodes be $i_1,i_2,\dots,i_k$, where $i_1>i_2>\cdots>i_k$. For $v\in 1:k$ and $u\in v:d$, define subsequence

$$\hat{\mathbf{m}}_{v,u}=\mathbf{y}_{i_v,u}\left[t_{i_v,v}+(1:L)\right], \tag{15}$$

which is of L bits. For $v\in 1:k$, node i_v transmits the subsequences $\hat{\mathbf{m}}_{v,u}$, $u=v,\dots,d$ to the decoder. Substituting (12) into (15), we have for $1\le l\le L$,

$$\hat{\mathbf{m}}_{v,u}[l]=\begin{cases}\mathbf{m}_{v,u}[l]+\sum_{j=1,j\neq v}^{u}\mathbf{m}_{j,u}[l+t_{i_v,v}-t_{i_v,j}]+\\ \sum_{j=u+1}^{d}\mathbf{m}_{u,j}[l+t_{i_v,v}-t_{i_v,j}], 1\le v\le u\le k,\\ \mathbf{m}_{v,u}[l]+\sum_{j=1,j\neq v}^{k}\mathbf{m}_{j,u}[l+t_{i_v,v}-t_{i_v,j}],\\ 1\le v\le k, k<u\le d,\end{cases} \tag{16}$$

where we see that $\hat{\mathbf{m}}_{v,u}$ involves all the bits in $\mathbf{m}_{v,u}$.

The decoding stage consists of two steps. In the first step, matrix $\mathbf{T}$ is decoded using $\hat{\mathbf{m}}_{v,u}$, $1\le v\le k, k+1\le u\le d$. In the second step, matrix $\mathbf{S}$ is decoded using $\hat{\mathbf{m}}_{v,u}$, $1\le v\le u\le k$. A pseudocode of the decoding scheme is shown in Algorithm 2, which also demonstrates an in-place implementation of the decoding algorithm.

Step 1: This step contains $d-k$ iterations. For each $u\in(k+1):d$, performing the shift-XOR elimination on $\hat{\mathbf{m}}_{1,u},\hat{\mathbf{m}}_{2,u},\dots,\hat{\mathbf{m}}_{k,u}$, we can decode $\mathbf{m}_{1,u},\mathbf{m}_{2,u},\dots,\mathbf{m}_{k,u}$ (ref. Line 2 in Algorithm 2). For in-place implementation, $\mathbf{m}_{v,u}$ is stored at the same storage place of storing $\hat{\mathbf{m}}_{v,u}$. For any v with $1\le v\le k$, $\mathbf{m}_{v,u}$ is involved in generating $\hat{\mathbf{m}}_{w,v}$, $1\le w\le v$ as in (16). So the value of $\mathbf{m}_{v,u}$ is substituted in $\hat{\mathbf{m}}_{w,v}$ (ref. Line 3–5 in Algorithm 2). After the substitution, $\hat{\mathbf{m}}_{w,v}$ with $1\le w\le v\le k$ is only related to $\mathbf{m}_{v',u'}$, $1\le v'\le u'\le k$.

Step 2: This step contains $k-1$ iterations. For iteration u from k down to 2, performing the shift-XOR elimination of $\hat{\mathbf{m}}_{1,u},\hat{\mathbf{m}}_{2,u},\dots,\hat{\mathbf{m}}_{u,u}$ to decode $\mathbf{m}_{1,u},\mathbf{m}_{2,u},\dots,\mathbf{m}_{u,u}$ (ref. Line 7 in Algorithm 2). Since $\mathbf{m}_{v,u}$ $(1\le v\le u-1)$ is involved in generating the bits in $\hat{\mathbf{m}}_{w,v}$ $(1\le w\le v)$, the sequence $\mathbf{m}_{v,u}$ $(1\le v\le u-1)$ is then substituted into $\hat{\mathbf{m}}_{w,v}$ $(1\le w\le v)$, according to (16) (ref. Line 8–10 in Algorithm 2). After the substitution, $\hat{\mathbf{m}}_{w,v}$ $(1\le w\le v\le u-1)$ is only related to $\mathbf{m}_{v',u'}, 1\le v'\le u'\le u-1$.

After Step 2, $\mathbf{m}_{v,u}$ $(1\le v\le k, v\le u\le d)$ is decoded and stored at the same storage place as $\hat{\mathbf{m}}_{v,u}$.

Example 5: Consider the $[6,3,4]$ shift-XOR MBR code in Example 1. Suppose the transmission stage chooses nodes 1, 3 and 4, i.e., $i_1=4$, $i_2=3$, and $i_3=1$. Node $i_1=4$ transmits

Algorithm 2 Decoding Algorithm for Shift-XOR MBR Codes With in-Place Implementation. After the Execution, the Output Value $\mathbf{m}_{v,u}[l]$ Is Stored at the Same Storage Space as $\hat{\mathbf{m}}_{v,u}[l]$

Input: coded sequences $\hat{\mathbf{m}}_{v,u}$, $1 \le v \le k, v \le u \le d$, and corresponding node indices i_j $(1 \le j \le k)$, which satisfies $i_1 > i_2 > \cdots > i_k$.

Output: decoded message sequences $\mathbf{m}_{v,u}$, $1 \le v \le k, v \le u \le d$.

(**Step 1:** Decode $\mathbf{T}$)

1: **for** $u \leftarrow d$ down to $k+1$ **do**
2: Decode $(\mathbf{m}_{v,u}, 1 \le v \le k)$ by executing Algorithm 1 on $\hat{\mathbf{m}}_{1,u}, \hat{\mathbf{m}}_{2,u}, \ldots, \hat{\mathbf{m}}_{k,u}$ with indices $i_1, i_2, \ldots, i_k$.
3: **for** $v \leftarrow 1 : k$ **do**
4: **for** $w \leftarrow 1 : v$ **do**
5: $\hat{\mathbf{m}}_{w,v}\left[(t_{i_w,u} - t_{i_w,w} + 1) : L\right] \oplus \mathbf{m}_{v,u}\left[1 : (L - t_{i_w,u} + t_{i_w,w})\right]$. ← (in-place back substitution)

(**Step 2:** Decode $\mathbf{S}$)

6: **for** $u \leftarrow k$ down to 2 **do**
7: Decode $(\mathbf{m}_{1,u}, \mathbf{m}_{2,u}, \ldots, \mathbf{m}_{u,u})$ executing Algorithm 1 on $\hat{\mathbf{m}}_{1,u}, \hat{\mathbf{m}}_{2,u}, \ldots, \hat{\mathbf{m}}_{u,u}$ with indices $i_1, i_2, \ldots, i_u$.
8: **for** $v \leftarrow 1 : (u-1)$ **do**
9: **for** $w \leftarrow 1 : v$ **do**
10: $\hat{\mathbf{m}}_{w,v}\left[(t_{i_w,u} - t_{i_w,w} + 1) : L\right] \oplus \mathbf{m}_{v,u}\left[1 : (L - t_{i_w,u} + t_{i_w,w})\right]$. ← (in-place back substitution)

$\hat{\mathbf{m}}_{1,i} = \mathbf{y}_{4,i}[1:L]$, $i = 1, \ldots, 4$ to the decoder. Node $i_2 = 3$ transmits $\hat{\mathbf{m}}_{2,i} = \mathbf{y}_{3,i}[3:(L+2)]$, $i = 2, 3, 4$ to the decoder. Node $i_3 = 1$ transmits $\hat{\mathbf{m}}_{3,i} = \mathbf{y}_{1,i}[1:L]$, $i = 3, 4$ to the decoder.

Since $d = k + 1$, Step 1 of Algorithm 2 has only one iteration. According to (16), we have for $1 \le l \le L$

$$\begin{aligned}
\hat{\mathbf{m}}_{1,4}[l] &= \mathbf{m}_{1,4}[l] + \mathbf{m}_{2,4}[l-3] + \mathbf{m}_{3,4}[l-3],\\
\hat{\mathbf{m}}_{2,4}[l] &= \mathbf{m}_{2,4}[l] + \mathbf{m}_{1,4}[l+2] + \mathbf{m}_{3,4}[l-2],\\
\hat{\mathbf{m}}_{3,4}[l] &= \mathbf{m}_{3,4}[l] + \mathbf{m}_{1,4}[l] + \mathbf{m}_{2,4}[l].
\end{aligned}$$

Performing the shift-XOR elimination on $\hat{\mathbf{m}}_{1,4}$, $\hat{\mathbf{m}}_{2,4}$ and $\hat{\mathbf{m}}_{3,4}$ we obtain $\mathbf{m}_{1,4}$, $\mathbf{m}_{2,4}$ and $\mathbf{m}_{3,4}$. For $1 \le l \le L$,

$$\hat{\mathbf{m}}_{1,1}[l] = \mathbf{m}_{1,1}[l] + \mathbf{m}_{1,2}[l-3] + \mathbf{m}_{1,3}[l-6] + \mathbf{m}_{1,4}[l-9],$$
$$\hat{\mathbf{m}}_{1,2}[l] = \mathbf{m}_{1,2}[l] + \mathbf{m}_{2,2}[l-3] + \mathbf{m}_{2,3}[l-6] + \mathbf{m}_{2,4}[l-9],$$
$$\hat{\mathbf{m}}_{1,3}[l] = \mathbf{m}_{1,3}[l] + \mathbf{m}_{2,3}[l-3] + \mathbf{m}_{3,3}[l-6] + \mathbf{m}_{3,4}[l-9],$$
$$\hat{\mathbf{m}}_{2,2}[l] = \mathbf{m}_{2,2}[l] + \mathbf{m}_{1,2}[l+2] + \mathbf{m}_{2,3}[l-2] + \mathbf{m}_{2,4}[l-4],$$
$$\hat{\mathbf{m}}_{2,3}[l] = \mathbf{m}_{2,3}[l] + \mathbf{m}_{1,3}[l+2] + \mathbf{m}_{3,3}[l-2] + \mathbf{m}_{3,4}[l-4],$$
$$\hat{\mathbf{m}}_{3,3}[l] = \mathbf{m}_{3,3}[l] + \mathbf{m}_{1,3}[l] + \mathbf{m}_{2,3}[l] + \mathbf{m}_{3,4}[l].$$

The decoder further substitutes $\mathbf{m}_{1,4}$ into $\hat{\mathbf{m}}_{1,1}$, substitutes $\mathbf{m}_{2,4}$ into $\hat{\mathbf{m}}_{1,2}$ and $\hat{\mathbf{m}}_{2,2}$, and substitutes $\mathbf{m}_{3,4}$ into $\hat{\mathbf{m}}_{1,3}$, $\hat{\mathbf{m}}_{2,3}$ and $\hat{\mathbf{m}}_{3,3}$ correspondingly. We use the same notation to denote the sequences after substitution.

Step 2 of Algorithm 2 has two iterations with $u = 3$ and $u = 2$ respectively. In the $u = 3$ iteration, we have for $1 \le l \le L$

$$\begin{aligned}
\hat{\mathbf{m}}_{1,3}[l] &= \mathbf{m}_{1,3}[l] + \mathbf{m}_{2,3}[l-3] + \mathbf{m}_{3,3}[l-6],\\
\hat{\mathbf{m}}_{2,3}[l] &= \mathbf{m}_{2,3}[l] + \mathbf{m}_{1,3}[l+2] + \mathbf{m}_{3,3}[l-2],\\
\hat{\mathbf{m}}_{3,3}[l] &= \mathbf{m}_{3,3}[l] + \mathbf{m}_{1,3}[l] + \mathbf{m}_{2,3}[l],
\end{aligned}$$

which can be solved by shift-XOR elimination to obtain $\mathbf{m}_{1,3}$, $\mathbf{m}_{2,3}$ and $\mathbf{m}_{3,3}$. Similarly, the decoder substitutes $\mathbf{m}_{1,3}$ into $\hat{\mathbf{m}}_{1,1}$, substitutes $\mathbf{m}_{2,3}$ into $\hat{\mathbf{m}}_{1,2}$ and $\hat{\mathbf{m}}_{2,2}$. In the $u = 2$ iteration, the shift-XOR elimination is performed on $\hat{\mathbf{m}}_{1,2}$ and $\hat{\mathbf{m}}_{2,2}$ to decode $\mathbf{m}_{1,2}$ and $\mathbf{m}_{2,2}$. The decoder then substitutes $\mathbf{m}_{1,2}$ into $\hat{\mathbf{m}}_{1,1}$, which becomes $\mathbf{m}_{1,1}$.

3) Complexity Analysis: **Time Complexity:** In Step 1, decoding each column of $\mathbf{T}$ costs $k(k-1)L$ XOR operations by the shift-XOR elimination (Line 2 in Algorithm 2). Noting that in Line 5 of Algorithm 2, the sequences for back substitution are shorter than L, and hence substituting each column of $\mathbf{T}$ into other retrieved sequences takes less than $\frac{1}{2}k(k+1)L$ XOR operations (Line 3 – 5 in Algorithm 2). There are $(d-k)$ iterations in this step, so the number of XOR operations T_1 required in Step 1 satisfies

$$\begin{aligned}
T_1 &< (d-k)\left(k(k-1) + \frac{1}{2}k(k+1)\right)L\\
&= \frac{1}{2}(d-k)(3k-1)kL.
\end{aligned}$$

In Step 2, similarly, an iteration $u \in 2 : k$ needs $u(u-1)L$ XOR operations to execute Algorithm 1 and less than $\frac{1}{2}u(u-1)L$ XOR operations to substitute the decoded sequences. Therefore, the number of XOR operations T_2 in the second step satisfies

$$\begin{aligned}
T_2 &< \sum_{u=2}^{k}\left(u(u-1)L + \frac{1}{2}u(u-1)L\right)\\
&= \frac{1}{2}(k-1)k(k+1)L.
\end{aligned}$$

Therefore, the total number of XOR operations is

$$T_1 + T_2 < \left(\left(\frac{3}{2}d - k\right)k - \frac{1}{2}(d-k+1)\right)kL.$$

So the time complexity is $O(dk^2L)$.

Space Complexity: Same as the shift-XOR elimination, Algorithm 2 can be implemented in-place, so that the output sequences take the same storage space as the input sequences. No extra space is required to store the intermediate XOR results. Only $O(1)$ integer auxiliary variables are required by Algorithm 2.

Bandwidth Overhead: Our algorithm consumes exactly BL bits from the storage nodes to decode the BL bits of the message sequences. Therefore, our decoding algorithm has zero bandwidth overhead.

B. Repair Scheme for Shift-XOR MBR Codes

This section introduces the repair scheme for the $[n, k, d]$ shift-XOR MBR code described in Section II-C.1. Suppose that node i fails. Our repair scheme generates a new storage node that stores d sequences

$$\mathbf{Y}^i = \mathbf{\Psi}^i\mathbf{M} = [\mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,d}], \tag{17}$$

same as the d sequences stored at node i. Note that a sequence in $\mathbf{Y}^i$ has $L+t_{i,d}$ bits. Recall that each storage node $j \in 1:n$ stores the sequences $\mathbf{\Psi}^j\mathbf{M}$, and hence can compute locally the sequence

$$\mathbf{r}_j = \mathbf{\Psi}^j\mathbf{M}\left(\mathbf{\Psi}^i\right)^\top = \mathbf{\Psi}^j(\mathbf{\Psi}^i\mathbf{M})^\top = \mathbf{\Psi}^j(\mathbf{Y}^i)^\top, \tag{18}$$

which is a shift-XOR equation of $\mathbf{Y}^i$. Therefore, using the shift-XOR elimination, $\mathbf{Y}^i$ can be decoded from $\mathbf{r}_j$ of any d nodes j.

Specifically, the repair scheme includes two stages: the transmission stage and the decoding stage. In the transmission stage, d *helper nodes* are chosen to repair node i, which have the indices $i_1, i_2, \ldots, i_d$ where $i_1 > i_2 > \cdots > i_d$. Each node i_v $(1 \le v \le d)$ transmits a subsequence of $\mathbf{r}_{i_v}$ (defined in (18))

$$\hat{\mathbf{m}}_v = \mathbf{r}_{i_v}[(1+t_{i_v,v}) : (L+t_{i,d}+t_{i_v,v})] \tag{19}$$

to the new node i for repairing. Note that $\hat{\mathbf{m}}_v$ has exactly the same number of bits as the sequences to repair. In the decoding stage, the new node i has received sequences $\hat{\mathbf{m}}_v$, $v = 1, \ldots, d$, and performs the shift-XOR elimination to decode $\mathbf{Y}^i$.

Example 6: Consider again $[6, 3, 4]$ MBR code in Example 1, where the message matrix and generator matrix can be found. This example will show the transmission stage and decoding stage to repair node 3 by connecting helper nodes 1, 2, 4 and 5, i.e., $i_1 = 5$, $i_2 = 4$, $i_3 = 2$, $i_4 = 1$. So

$$\begin{aligned}
\mathbf{r}_5 &= \mathbf{y}_{3,1} + z^4\mathbf{y}_{3,2} + z^8\mathbf{y}_{3,3} + z^{12}\mathbf{y}_{3,4},\\
\mathbf{r}_4 &= \mathbf{y}_{3,1} + z^3\mathbf{y}_{3,2} + z^6\mathbf{y}_{3,3} + z^9\mathbf{y}_{3,4},\\
\mathbf{r}_2 &= \mathbf{y}_{3,1} + z^1\mathbf{y}_{3,2} + z^2\mathbf{y}_{3,3} + z^3\mathbf{y}_{3,4},\\
\mathbf{r}_1 &= \mathbf{y}_{3,1} + \mathbf{y}_{3,2} + \mathbf{y}_{3,3} + \mathbf{y}_{3,4}.
\end{aligned}$$

Then the sequences transmitted to the new node 3 are

$$\begin{aligned}
\hat{\mathbf{m}}_1 &= \mathbf{r}_5[1 : (L+6)]\\
\hat{\mathbf{m}}_2 &= \mathbf{r}_4[4 : (L+9)]\\
\hat{\mathbf{m}}_3 &= \mathbf{r}_2[3 : (L+8)]\\
\hat{\mathbf{m}}_4 &= \mathbf{r}_1.
\end{aligned}$$

Applying Algorithm 1 on $\hat{\mathbf{m}}_1$, $\hat{\mathbf{m}}_2$, $\hat{\mathbf{m}}_3$ and $\hat{\mathbf{m}}_4$, we repair $\mathbf{y}_{3,1}$, $\mathbf{y}_{3,2}$, $\mathbf{y}_{3,3}$ and $\mathbf{y}_{3,4}$ at the new node 3.

Time Complexity: The repair computation cost involves two parts: The first part is the computation at the d helper nodes, and the second part is the computation at the repaired node. At each helper node, $(d-1)(L+t_{i,d})$ XOR operations are used. So the total number of XOR operations at all the helper nodes is $d(d-1)(L+t_{i,d})$. The number of XOR operations of the second part is $d(d-1)(L+t_{i,d})$, according to the analysis of Algorithm 1. Totally, the number of XOR operations of MBR codes repair is $2d(d-1)(L+t_{i,d}) = 2d(d-1)L + O(nd^3)$.

Space Complexity: In the repaired node, one shift-XOR elimination is performed and $O(1)$ auxiliary integer variables are required.

Bandwidth: Our algorithm consumes exactly $L + t_{i,d}$ bits from d helper nodes to repair the $d(L+t_{i,d})$ bits of node i, and hence has zero bandwidth overhead.

V. Decoding and Repair Schemes of Shift-XOR MSR Codes

In this section, we discuss the decoding and repair schemes for the shift-XOR MSR codes described in Section II-C.2. Similar to those of the shift-XOR MBR codes, our schemes decompose the decoding/repair problem into a sequence of systems of shift-XOR equations, each of which can be solved efficiently using the shift-XOR elimination.

A. Decoding Scheme of Shift-XOR MSR Codes

Consider an $[n, k, d]$ shift-XOR MSR code, where $d = 2k-2$ and $\alpha = k-1$. Substituting the message matrix $\mathbf{M}$ in (4) and the generator matrix $\mathbf{\Psi}$ in (5) into the general encoding formula (1), we obtain

$$\mathbf{Y}^i = \mathbf{\Phi}^i\mathbf{S} + z^{\lambda_i}\mathbf{\Phi}^i\mathbf{T}, \tag{20}$$

which is the i-th row of $\mathbf{Y}$ stored at storage node i. Denote the (v,u) entry of $\mathbf{S}$ and $\mathbf{T}$ as $\mathbf{s}_{v,u}$ and $\mathbf{t}_{v,u}$, respectively. Due to symmetry, the decoding problem is to solve the message sequences $\mathbf{s}_{v,u}$, $\mathbf{t}_{v,u}$, $1 \le v \le u \le \alpha$, totally $B = \alpha(\alpha+1) = (k-1)k$ sequences.

1) Decomposition of Decoding Problem: Suppose the indices of the k nodes chosen for decoding are $i_1, i_2, \ldots, i_k$, where $i_1 > i_2 > \cdots > i_k$, so that the decoder can retrieve $\mathbf{Y}^{i_u}$, $u = 1, \ldots, k$. Denote for $1 \le u \ne v \le k$,

$$\mathbf{c}_{u,v} = \mathbf{Y}^{i_u}(\mathbf{\Phi}^{i_v})^\top, \tag{21}$$
$$\mathbf{p}_{u,v} = \mathbf{\Phi}^{i_u}\mathbf{S}(\mathbf{\Phi}^{i_v})^\top, \tag{22}$$
$$\mathbf{q}_{u,v} = \mathbf{\Phi}^{i_u}\mathbf{T}(\mathbf{\Phi}^{i_v})^\top. \tag{23}$$

Here $\mathbf{c}_{u,v}$ can be computed by the decoder using the sequences it retrieved. Due to the symmetry of $\mathbf{S}$ and $\mathbf{T}$, we have $\mathbf{p}_{u,v} = \mathbf{p}_{v,u}$ and $\mathbf{q}_{u,v} = \mathbf{p}_{v,u}$. Briefly, the decoding problem is decomposed into the following two steps:

- First, $\mathbf{p}_{u,v}$, $\mathbf{q}_{u,v}$, $1 \le u < v \le k$ are solved using $\mathbf{c}_{u,v}$, $1 \le u \ne v \le k$.
- Second, $\mathbf{S}$ is solved using $\mathbf{p}_{u,v}$, $1 \le u < v \le k$, and $\mathbf{T}$ is solved using $\mathbf{q}_{u,v}$, $1 \le u < v \le k$.

Let us elaborate these two steps.

Step 1: For $1 \le v < u \le k$, we have the shift-XOR equations (obtained by (20) – (23))

$$\begin{aligned}
\begin{bmatrix}\mathbf{c}_{u,v}\\ \mathbf{c}_{v,u}\end{bmatrix} &= \begin{bmatrix}\mathbf{p}_{u,v} + z^{\lambda_{i_u}}\mathbf{q}_{u,v}\\ \mathbf{p}_{v,u} + z^{\lambda_{i_v}}\mathbf{q}_{v,u}\end{bmatrix}\\
&= \begin{bmatrix}\mathbf{p}_{u,v} + z^{\lambda_{i_u}}\mathbf{q}_{u,v}\\ \mathbf{p}_{u,v} + z^{\lambda_{i_v}}\mathbf{q}_{u,v}\end{bmatrix}\\
&= \begin{bmatrix}1 & z^{\lambda_{i_u}}\\ 1 & z^{\lambda_{i_v}}\end{bmatrix}\begin{bmatrix}\mathbf{p}_{u,v}\\ \mathbf{q}_{u,v}\end{bmatrix}.
\end{aligned} \tag{24}$$

Due to the RID property of $[\mathbf{\Phi}\ \mathbf{\Lambda}\mathbf{\Phi}]$, we have $\lambda_{i_u}t_{i_u,1} - t_{i_u,a} < \lambda_{i_v}t_{i_v,1} - t_{i_v,a}$ and $t_{i_u,a} - t_{i_u,1} < t_{i_v,a} - t_{i_v,1}$, which implies $\lambda_{i_u} \ne \lambda_{i_v}$. Hence, the shift-XOR elimination can be performed on $\mathbf{c}_{u,v}$ and $\mathbf{c}_{v,u}$ to solve $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$, which are of $L + t_{i_u,\alpha} + t_{i_v,\alpha}$ bits. According to the discussion in Section III, only the subsequences

$$\hat{\mathbf{c}}_{u,v} := \mathbf{c}_{u,v}[1 : (L + t_{i_u,\alpha} + t_{i_v,\alpha})] \tag{25}$$
$$\hat{\mathbf{c}}_{v,u} := \mathbf{c}_{v,u}[\lambda_{i_v} + (1 : (L + t_{i_u,\alpha} + t_{i_v,\alpha}))] \tag{26}$$

are needed for the shift-XOR elimination.

Step 2: Define an $\alpha \times \alpha$ matrix $\tilde{\mathbf{S}} = (\tilde{\mathbf{s}}_{v,u})$ as

$$\tilde{\mathbf{S}} = \begin{bmatrix} \mathbf{\Phi}^{i_1} \\ \vdots \\ \mathbf{\Phi}^{i_\alpha} \end{bmatrix} \mathbf{S}. \tag{27}$$

Due to the symmetry of $\mathbf{S}$, $(\tilde{\mathbf{S}}^v)^\top = \mathbf{S}(\mathbf{\Phi}^{i_v})^\top$. For each $v \in 1:\alpha$, form the $\alpha \times \alpha$ system of shift-XOR equations (by (22) and (27))

$$\begin{bmatrix} \mathbf{p}_{1,v} \\ \vdots \\ \mathbf{p}_{v-1,v} \\ \mathbf{p}_{v+1,v} \\ \vdots \\ \mathbf{p}_{k,v} \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^{i_1} \\ \vdots \\ \mathbf{\Phi}^{i_{v-1}} \\ \mathbf{\Phi}^{i_{v+1}} \\ \vdots \\ \mathbf{\Phi}^{i_k} \end{bmatrix} (\tilde{\mathbf{S}}^v)^\top, \tag{28}$$

so that $(\tilde{\mathbf{S}}^v)^\top$ can be solved by performing the shift-XOR elimination on the LHS of (28). By (27), we see the maximum length of sequences in $\tilde{\mathbf{S}}^v$ is $L'_v = L + t_{i_v,\alpha}$. So the shift-XOR elimination only needs the subsequences

$$\hat{\mathbf{p}}_{u,v} := \begin{cases} \mathbf{p}_{u,v}[t_{i_u,u} + (1:L'_v)], & u = 1,\ldots,v-1 \\ \mathbf{p}_{u,v}[t_{i_u,u-1} + (1:L'_v)], & u = v+1,\ldots,k. \end{cases}$$

After solving $\tilde{\mathbf{S}}$, we further solve the system of shift-XOR equations (27), so that $\mathbf{S}_u$ can be decoded by the shift-XOR elimination on

$$\hat{\mathbf{s}}_{v,u} := \tilde{\mathbf{s}}_{v,u}[t_{i_v,v} + (1:L)], \quad v = 1,\ldots,\alpha.$$

The procedure for solving $\mathbf{T}$ is the same and hence is omitted.

2) Decoding Scheme: Our decoding scheme is able to decode the message sequences by retrieving data from any k out of the n nodes. Now we give the details of our decoding scheme, which consists of two stages: the transmission stage and the decoding stage. In the transmission stage, k storage nodes are chosen. Let the indices of the k nodes be $i_1, i_2, \ldots, i_k$, where $i_1 > i_2 > \cdots > i_k$. For $u \in 1:k$, node i_u transmits $\mathbf{Y}^{i_u}$ to the decoder.

The decoding stage includes the two steps described above, with the pseudocode in Algorithm 3. The algorithm inputs $\mathbf{Y}^{i_u}$, $u = 1,\ldots,k$ (totally B sequences) and outputs the B message sequences. But different from the decoding of shift-XOR MBR codes, this algorithm needs extra storage space for the intermediate XOR results.

In Step 1, the algorithm first calculates $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$, $1 \le u < v \le k$. From (25) and (26), we see the length of $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$ is $L + O(nd)$. To storage $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$, $1 \le u < v \le k$, a space of $\alpha(\alpha+1)(L+O(nd))$ bits is needed. In Line 4, the shift-XOR elimination is applied on $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$ to solve $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$ for $1 \le v < u \le k$. As the shift-XOR elimination is in-place, $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$ can take exactly the same storage space as $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$ for $1 \le v < u \le k$, respectively. Then another space of $\alpha(\alpha-1)$ sequences is needed to store $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$, $1 \le u < v \le \alpha$. Therefore, after Line 7, totally $2\alpha^2(L+O(nd))$ bits space is needed.

In Step 2, from Line 8 to 9, the shift-XOR elimination is applied on $\hat{\mathbf{p}}_{u,v}$, $u = 1,\ldots,v-1,v+1,\ldots,k$ to solve $\tilde{\mathbf{S}}^v$ for $v = 1,\ldots,\alpha$. From Line 10 to 11, the shift-XOR elimination is applied on $\hat{\mathbf{s}}_{u,v}$, $v = 1,2,\ldots,\alpha$ to solve $\mathbf{S}_u$ for $u = 1,\ldots,\alpha$. Due to the in-place property of the shift-XOR elimination, no additional space is needed.

Last, $\mathbf{T}$ is solved by repeating the above process on $\hat{\mathbf{q}}_{u,v}$ instead of $\hat{\mathbf{p}}_{u,v}$.

Algorithm 3 Decoding Algorithm for Shift-XOR MSR Codes

Input: coded sequences $\mathbf{Y}^{i_u}$, $u = 1,\ldots,k$.
Output: message sequences $\mathbf{s}_{u,v}$ and $\mathbf{t}_{u,v}$, $1 \le u \le \alpha, 1 \le v \le \alpha$.
(**Step 1:** Solve $\mathbf{p}_{v,u}$ and $\mathbf{q}_{v,u}$)
1: Calculate $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$ for $1 \le u < v \le k$.
2: **for** $u \leftarrow 2:k$ **do**
3: **for** $v \leftarrow 1:u-1$ **do**
4: Apply Algorithm 1 on $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$ to solve $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$.
5: **if** $u < k$ **then**
6: $\mathbf{p}_{v,u} \leftarrow \mathbf{p}_{u,v}$.
7: $\mathbf{q}_{v,u} \leftarrow \mathbf{q}_{u,v}$.
(**Step 2:** Decode $\mathbf{S}$ and $\mathbf{T}$)
8: **for** $v \leftarrow 1:\alpha$ **do**
9: Apply Algorithm 1 on $\hat{\mathbf{p}}_{u,v}$, $u = 1,\ldots,v-1,v+1,\ldots,k$ to solve $\tilde{\mathbf{S}}^v$.
10: **for** $u \leftarrow 1:\alpha$ **do**
11: Apply Algorithm 1 on $\hat{\mathbf{s}}_{u,v}$, $v = 1,\ldots,\alpha$ to solve $\mathbf{S}_u$. ($\hat{\mathbf{s}}_{u,v}$ is a subsequence of the (u,v) entry of $\tilde{\mathbf{S}}$)
12: Repeat Line 8 – 11 on $\hat{\mathbf{q}}_{u,v}$ instead of $\hat{\mathbf{p}}_{u,v}$ to solve $\mathbf{T}$.

Example 7: Consider the example of $[6,3,4]$ MSR code in Example 2, where $\alpha = 2$. Consider decoding from nodes 1, 3 and 4, i.e., $i_1 = 4$, $i_2 = 3$ and $i_3 = 1$. The decoder retrieves the sequences $\mathbf{Y}^4 = [\mathbf{y}_{4,1}, \mathbf{y}_{4,2}]$, $\mathbf{Y}^3 = [\mathbf{y}_{3,1}, \mathbf{y}_{3,2}]$ and $\mathbf{Y}^1 = [\mathbf{y}_{1,1}, \mathbf{y}_{1,2}]$, and obtains the shift-XOR equations

$$\begin{aligned} \mathbf{y}_{4,1} &= \mathbf{x}_1 + z^3\mathbf{x}_2 + z^6\mathbf{x}_4 + z^9\mathbf{x}_5 \\ \mathbf{y}_{4,2} &= \mathbf{x}_2 + z^3\mathbf{x}_3 + z^6\mathbf{x}_5 + z^9\mathbf{x}_6 \\ \mathbf{y}_{3,1} &= \mathbf{x}_1 + z^2\mathbf{x}_2 + z^4\mathbf{x}_4 + z^6\mathbf{x}_5 \\ \mathbf{y}_{3,2} &= \mathbf{x}_2 + z^2\mathbf{x}_3 + z^4\mathbf{x}_5 + z^6\mathbf{x}_6 \\ \mathbf{y}_{1,1} &= \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 \\ \mathbf{y}_{1,2} &= \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_5 + \mathbf{x}_6. \end{aligned}$$

The above system cannot be solved directly using the shift-XOR elimination as it does not satisfy the RID property. We apply Algorithm 3 to solve the system.

First, the decoder uses the above sequences to calculate $\hat{\mathbf{c}}_{u,v}$ ($1 \le u \ne v \le 3$), where

$$\begin{aligned} \hat{\mathbf{c}}_{1,2} &= \left(\mathbf{Y}^4(\mathbf{\Phi}^3)^\top\right)[1:(L+5)], \\ \hat{\mathbf{c}}_{1,3} &= \left(\mathbf{Y}^4(\mathbf{\Phi}^1)^\top\right)[1:(L+3)], \\ \hat{\mathbf{c}}_{2,1} &= \mathbf{Y}^3(\mathbf{\Phi}^4)^\top[5:(L+9)], \\ \hat{\mathbf{c}}_{2,3} &= \left(\mathbf{Y}^3(\mathbf{\Phi}^1)^\top\right)[1:(L+2)], \\ \hat{\mathbf{c}}_{3,1} &= \mathbf{Y}^1(\mathbf{\Phi}^4)^\top[3:(L+5)], \\ \hat{\mathbf{c}}_{3,2} &= \mathbf{Y}^1(\mathbf{\Phi}^3)^\top[3:(L+4)]. \end{aligned}$$

By (24), we have the system of shift-XOR equations

$$\begin{bmatrix}\mathbf{c}_{2,1}\\ \mathbf{c}_{1,2}\end{bmatrix}=\begin{bmatrix}1 & z^{\lambda_{i_2}}\\ 1 & z^{\lambda_{i_1}}\end{bmatrix}\begin{bmatrix}\mathbf{p}_{2,1}\\ \mathbf{q}_{2,1}\end{bmatrix}.$$

Performing the shift-XOR elimination on $\hat{\mathbf{c}}_{2,1}$ and $\hat{\mathbf{c}}_{1,2}$, we obtain $\mathbf{p}_{2,1}$ and $\mathbf{q}_{2,1}$. Similarly, we obtain $\mathbf{p}_{3,1}$ and $\mathbf{q}_{3,1}$ from $\hat{\mathbf{c}}_{1,3}$ and $\hat{\mathbf{c}}_{3,1}$, and obtain $\mathbf{p}_{3,2}$ and $\mathbf{q}_{3,2}$ from $\hat{\mathbf{c}}_{2,3}$ and $\hat{\mathbf{c}}_{3,2}$. We further generate $\mathbf{p}_{v,u}$ and $\mathbf{q}_{v,u}$ for $1\le v<u\le 3$ by symmetry.

By (28), we can form two systems

$$\begin{bmatrix}\mathbf{p}_{2,1}\\ \mathbf{p}_{3,1}\end{bmatrix}=\begin{bmatrix}\mathbf{\Phi}^{i_2}\\ \mathbf{\Phi}^{i_3}\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{s}}_{1,1}\\ \tilde{\mathbf{s}}_{1,2}\end{bmatrix},$$

and

$$\begin{bmatrix}\mathbf{p}_{1,2}\\ \mathbf{p}_{3,2}\end{bmatrix}=\begin{bmatrix}\mathbf{\Phi}^{i_1}\\ \mathbf{\Phi}^{i_3}\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{s}}_{2,1}\\ \tilde{\mathbf{s}}_{2,2}\end{bmatrix},$$

solving of which give us $\tilde{\mathbf{s}}_{v,u}$, $1\le v,u\le 2$. Then by (27), we have the system

$$\begin{bmatrix}\tilde{\mathbf{s}}_{1,1} & \tilde{\mathbf{s}}_{1,2}\\ \tilde{\mathbf{s}}_{2,1} & \tilde{\mathbf{s}}_{2,2}\end{bmatrix}=\begin{bmatrix}\mathbf{\Phi}^{i_1}\\ \mathbf{\Phi}^{i_2}\end{bmatrix}\begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2\\ \mathbf{x}_2 & \mathbf{x}_3\end{bmatrix}.$$

Applying the shift-XOR elimination on $\hat{\mathbf{s}}_{1,1}$ and $\hat{\mathbf{s}}_{2,1}$, we obtain $\mathbf{x}_1$ and $\mathbf{x}_2$. Applying the shift-XOR elimination on $\hat{\mathbf{s}}_{1,2}$ and $\hat{\mathbf{s}}_{2,2}$, we obtain $\mathbf{x}_2$ and $\mathbf{x}_3$.

Executing the same process above using $\mathbf{q}_{u,v}$ in place of $\mathbf{p}_{u,v}$, we can solve $\mathbf{x}_4,\mathbf{x}_5,\mathbf{x}_6$.

3) Complexity Analysis: **Time Complexity:** The time complexity of Algorithm 3 can be calculated based on the time complexity of shift-XOR eliminations performed in the algorithm. First, $k\alpha(\alpha-1)(L+O(nd))=k(k-1)(k-2)(L+O(nd))$ XOR operations are needed for computing $\hat{\mathbf{c}}_{u,v}$ and $\hat{\mathbf{c}}_{v,u}$ for $1\le u<v\le k$. Solving $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$ for $u=2,\ldots,k$ and $v=1,\ldots,u-1$ costs $\alpha(\alpha+1)(L+O(nd))=k(k-1)(L+O(nd))$ XOR operations. Then solving $\mathbf{S}$ and $\mathbf{T}$ costs $4\alpha^2(\alpha-1)(L+O(nd))=4(k-1)^2(k-2)(L+O(nd))$ XOR operations. Totally, the time complexity T for decoding of MSR codes is

$$\begin{aligned}T &= k\,(k-1)\,(k-2)\,(L+O(nd))+k\,(k-1)\\ &\quad\times(L+O(nd))+4(k-1)^2(k-2)(L+O(nd))\\ &= (k-1)^2\,(5k-8)\,L+O(nk^3d).\end{aligned}$$

Space Complexity: In Algorithm 3, the $2\alpha^2$ sequences $\mathbf{p}_{u,v}$, $\mathbf{p}_{v,u}$ take $2\alpha^2(L+O(nd))$ bits storage, which is the largest space cost of the algorithm during an execution. The message sequences has $\alpha(\alpha+1)L$ bits, so the auxiliary space is $\alpha(\alpha-1)L+O(nd^3)$ bits.

Bandwidth: As the number of bits transmitted to the decoder from node i_j is $\alpha(L+t_{i_j,\alpha}+\lambda_{i_j})$, the total number of transmitted bits is $k\alpha L+\alpha\sum_{j=1}^{k}(t_{i_j,\alpha}+\lambda_{i_j})$. There are $k\alpha L$ bits in the message sequences, and hence the bandwidth overhead is $\alpha\sum_{j=1}^{k}(t_{i_j,\alpha}+\lambda_{i_j})=O(nk^2d)$.

B. Repair Scheme of Shift-XOR MSR Codes

This section introduces our repair scheme for an $[n,k,d]$ shift-XOR MBR code, where $d=2k-2$ and $\alpha=k-1$. Define an $\alpha\times d$ matrix $\mathbf{X}=(\mathbf{x}_{i,j})$ with $\mathbf{x}_{i,j}=\mathbf{\Phi}^i\mathbf{S}_j$ and $\mathbf{x}_{i,j+\alpha}=\mathbf{\Phi}^i\mathbf{T}_j$ for $1\le i,j\le\alpha$. Suppose that node i fails. Our repair scheme generates a new storage node that stores the same α sequences at node i:

$$\mathbf{Y}^i=\mathbf{\Psi}^i\mathbf{M}=\mathbf{\Phi}^i\mathbf{S}+z^{\lambda_i}\mathbf{\Phi}^i\mathbf{T}=\mathbf{X}^i_{1:\alpha}+z^{\lambda_i}\mathbf{X}^i_{(\alpha+1):2\alpha}. \quad (29)$$

Recall that each storage node $j\in 1:n$ stores the sequences $\mathbf{\Psi}^j\mathbf{M}$, and hence can compute locally the sequence

$$\mathbf{r}_j=\mathbf{\Psi}^j\mathbf{M}\left(\mathbf{\Phi}^i\right)^\top=\mathbf{\Psi}^j(\mathbf{X}^i)^\top, \quad (30)$$

which is a shift-XOR equation of $\mathbf{X}^i$. By $\mathbf{r}_j$ from any d nodes $j\ne i$, we can solve $\mathbf{X}^i$ using the shift-XOR elimination, and then calculate $\mathbf{Y}^i$ by (29).

Specifically, the repair scheme includes two stages: the transmission stage and the decoding stage. In the transmission stage, d helper nodes are chosen to repair node i, which have the indices $i_1,i_2,\ldots,i_d$ where $i_1>i_2>\cdots>i_d$. Each helper node i_v $(1\le v\le d)$ transmits

$$\hat{\mathbf{r}}_v=\mathbf{r}_{i_v}[t_{i_v,v}+(1:(L+t_{i,\alpha}))] \quad (31)$$

to the new node i for repairing, where the sequence $\mathbf{r}_{i_v}$ is defined in (30). In the decoding stage, the new node i performs the shift-XOR elimination on $\hat{\mathbf{r}}_v$, $v=1,\ldots,d$ to decode $\mathbf{X}^i$, and then calculate $\mathbf{Y}^i$ by (29).

Example 8: Consider the $[6,3,4]$ MSR Code studied in Example 2. This example will show the transmission and decoding stages to repair node 3 from helper nodes 1, 2, 4 and 5, i.e., $i_1=5$, $i_2=4$, $i_3=2$, $i_4=1$ and $i=3$. The sequences transmitted to node i from the 4 helper nodes are

$$\begin{aligned}\hat{\mathbf{r}}_1 &= \mathbf{r}_5\,[1:(L+2)]=(\mathbf{Y}^5(\mathbf{\Phi}^3)^\top)\,[1:(L+2)]\,, &(32)\\ \hat{\mathbf{r}}_2 &= \mathbf{r}_4\,[4:(L+5)]=(\mathbf{Y}^4(\mathbf{\Phi}^3)^\top)\,[4:(L+5)]\,, &(33)\\ \hat{\mathbf{r}}_3 &= \mathbf{r}_2\,[3:(L+4)]=(\mathbf{Y}^2(\mathbf{\Phi}^3)^\top)\,[3:(L+4)]\,, &(34)\\ \hat{\mathbf{r}}_4 &= \mathbf{r}_1=\mathbf{Y}^1(\mathbf{\Phi}^3)^\top. &(35)\end{aligned}$$

By (30), we have the system

$$\begin{bmatrix}\mathbf{r}_5\\ \mathbf{r}_4\\ \mathbf{r}_2\\ \mathbf{r}_1\end{bmatrix}=\begin{bmatrix}\mathbf{\Phi}^5\\ \mathbf{\Phi}^4\\ \mathbf{\Phi}^2\\ \mathbf{\Phi}^1\end{bmatrix}\left(\mathbf{X}^3\right)^\top.$$

Applying the shift-XOR elimination on $\hat{\mathbf{r}}_1$, $\hat{\mathbf{r}}_2$, $\hat{\mathbf{r}}_3$ and $\hat{\mathbf{r}}_4$, we can obtain $\mathbf{X}^3$, and hence solve $\mathbf{Y}^3$ by (29).

Time Complexity: At each helper node i_v $(v=1,\ldots,d)$, computing $\hat{\mathbf{r}}_{i_v}$ costs $(\alpha-1)(L+O(nd))$ XOR operations. So there are totally $d\,(\alpha-1)\,(L+O(nd))=d(\frac{d}{2}-1)(L+O(nd))$ XOR operations at all the helper nodes. At the new node i, solving $\mathbf{X}^i$ costs $d(d-1)(L+O(nd))$ XOR operations by the shift-XOR elimination. Computing $\mathbf{Y}^i$ costs $\alpha(L+O(n\alpha))=\frac{d}{2}(L+O(n\alpha))$ XOR operations. So the total time complexity at the new node i is $d(d-1/2)(L+O(nd))$. The overall time complexity among all the involved nodes is $\frac{3}{2}(d-1)\,dL+O(nd^3)$.

Bandwidth: As the number of bits transmitted from node i_v is $L+t_{i,\alpha}$, so the total number of bits transmitted is $d(L+t_{i,\alpha})=dL+O(nd^2)$.

Space Complexity: The storage space of $d(L+t_{i,\alpha})=d(L+O(nd))$ bits is required to store the bits retrieved from the helper nodes. As the shift-XOR elimination can be implemented in-place, no extra storage space is required

to store the intermediate results $\mathbf{X}^i$. The total length of the repaired sequences is $\alpha(L+\lambda_i+t_{i,\alpha}) = \frac{d}{2}(L+O(nd))$. So the auxiliary space required for intermediate XOR results is $\frac{d}{2}(L+O(nd))$.

VI. Extensions to Other PM-Constructed Codes

The decompositions of our decoding and repair schemes discussed in the previous two sections depend mostly on the PM construction, and have little correlation to the shift and XOR operations. Therefore, similar decomposition may be possible for the decoding and repair of other regenerating codes based on the PM construction. In this section, we study the extensions of our decoding and repair schemes to the finite-field PM codes [20], [21] and the cyclic-shift PM codes [28].

A. Extension to Finite-Field PM Codes

The finite-field PM codes in [20] use finite field operations. Suppose the entries of a sequence are elements from a finite field $\mathbb{F}$. Same as the setting in Section II-B, using finite field operations, we define

$$\mathbf{y}_{i,j} = \sum_{u=1}^{d} \psi_{i,u}\mathbf{m}_{u,j}, \quad 1 \le i \le n, 1 \le j \le \alpha,$$

where $\psi_{i,u} \in \mathbb{F}$, and $\mathbf{y}_{i,j}$ and $\mathbf{m}_{u,j}$ are sequences of L bits, or L symbols from $\mathbb{F}$. Denoting $\mathbf{\Psi} = (\psi_{i,j})$, called the generator matrix, the encoding follows the same form of (1):

$$\mathbf{Y} = \mathbf{\Psi}\mathbf{M}. \tag{36}$$

As Gaussian elimination can solve systems of linear equations over finite fields, we can use Gaussian elimination in place of shift-XOR elimination to build decode/repair schemes for the finite-field PM codes.

1) Decoding Scheme of Finite-Field MBR Codes: For the finite-field MBR codes in [20], $\mathbf{\Psi}$ in (36) is of the form

$$\mathbf{\Psi} = [\mathbf{\Phi} \quad \mathbf{\Delta}],$$

where $\mathbf{\Phi} = (\phi_{i,j})$ and $\mathbf{\Delta}$ are $n \times k$ and $n \times (d-k)$ matrices respectively and satisfy: 1) any d rows of $\mathbf{\Psi}$ are linearly independent; 2) any k rows of $\mathbf{\Phi}$ are linearly independent. Specifically, the Vandermonde matrix and Cauchy matrix satisfy the above two conditions [20]. The message matrix is of the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{S} & \mathbf{T} \\ \mathbf{T}^\top & \mathbf{O} \end{bmatrix},$$

where $\mathbf{S}$ is a $k \times k$ symmetric matrix and $\mathbf{T}$ is a $k \times (d-k)$ matrix. There are totally $\frac{1}{2}(k+1)k + k(d-k)$ message sequences. In the decoding algorithm of [20], the dk sequences stored at k nodes are retrieved for decoding, so that the decoding bandwidth overhead is $\frac{1}{2}k(k-1)$ sequences.

Due to the same matrix form as the shift-XOR MBR codes, we may wonder whether it is possible to derive a similar decoding algorithm as in Section IV-A with zero bandwidth overhead. We show it is possible when $\mathbf{\Phi}$ satisfies the further requirement that for any k rows of $\mathbf{\Phi}$, all the leading principal submatrices are full rank. It is noted that the Vandermonde matrix and Cauchy matrix used in [20] also satisfy the requirement.

To illustrate the algorithm, suppose the first k storage nodes are used for decoding. The decoding using other choice of nodes is similar. The decoder first retrieves

$$\mathbf{Y}_{(k+1):d}^{1:k} = \mathbf{\Phi}^{1:k}\mathbf{T}.$$

As any k rows of $\mathbf{\Phi}$ are linearly independent, $\mathbf{T}$ can be decoded using Gaussian elimination. After decoding $\mathbf{T}$, we continue to decode the k columns of $\mathbf{S}$ sequentially with the column indices in descending order. For $u = k, k-1, \ldots, 1$,

$$\mathbf{Y}_u^{1:u} = \mathbf{\Phi}_{1:u}^{1:u}\mathbf{S}_u^{1:u} + \mathbf{\Phi}_{(u+1):k}^{1:u}\mathbf{S}_u^{(u+1):k} + \mathbf{\Delta}^{1:u}(\mathbf{T}^u)^\top,$$

i.e.,

$$\mathbf{Y}_u^{1:u} - \mathbf{\Phi}_{(u+1):k}^{1:u}\mathbf{S}_u^{(u+1):k} - \mathbf{\Delta}^{1:u}(\mathbf{T}^u)^\top = \mathbf{\Phi}_{1:u}^{1:u}\mathbf{S}_u^{1:u}.$$

After substituting $\mathbf{T}$ and $\mathbf{S}_u^{(u+1):k} = \mathbf{S}_{(u+1):k}^u$, the LHS of the above equation is known and $\mathbf{S}_u^{1:u}$ can be decoded by Gaussian elimination as $\mathbf{\Phi}_{1:u}^{1:u}$ has rank u. Hence, $\mathbf{S}$ can be decoded column by column.

The above decoding scheme has the same asymptotic time/space/bandwidth complexity as the scheme for the shift-XOR MBR codes, and the decoder retrieves exactly the same number of bits as the message sequences.

2) Decoding Scheme of Finite-Field MSR Codes: For finite-field MSR codes with $d = 2k-2$ and $\alpha = k-1$ in [20], the generator matrix $\mathbf{\Psi}$ is of the form

$$\mathbf{\Psi} = [\mathbf{\Phi} \quad \mathbf{\Lambda}\mathbf{\Phi}],$$

where $\mathbf{\Phi}$ is an $n \times \alpha$ matrix and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$ is an $n \times n$ diagonal matrix and satisfy: 1) any d rows of $\mathbf{\Psi}$ are linearly independent; 2) any α rows of $\mathbf{\Phi}$ are linearly independent; (3) the n diagonal elements in $\mathbf{\Lambda}$ are distinct. The message matrix is of the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{S} \\ \mathbf{T} \end{bmatrix}.$$

The coded sequences stored are obtained by

$$\mathbf{Y} = \mathbf{\Psi}\mathbf{M}.$$

The i-th row of $\mathbf{Y}$ are stored at node i, i.e., $\mathbf{Y}^i = \mathbf{\Phi}^i\mathbf{S} + \lambda_i\mathbf{\Phi}^i\mathbf{T}$.

Assume that the decoder has access to k nodes i_u for $1 \le u \le k$. For decoding, node i_u transmits $\mathbf{Y}^{i_u}$ to the decoder. As discussed in Section V-A.2, denote for $1 \le u \ne v \le k$,

$$\begin{aligned} \mathbf{c}_{u,v} &= \mathbf{Y}^{i_u}(\mathbf{\Phi}^{i_v})^\top, \\ \mathbf{p}_{u,v} &= \mathbf{\Phi}^{i_u}\mathbf{S}(\mathbf{\Phi}^{i_v})^\top, \\ \mathbf{q}_{u,v} &= \mathbf{\Phi}^{i_u}\mathbf{T}(\mathbf{\Phi}^{i_v})^\top. \end{aligned}$$

$\mathbf{S}$ and $\mathbf{T}$ can be solved by the following two steps.

Step 1: For $1 \le v < u \le k$, we have the linear systems

$$\begin{aligned} \begin{bmatrix} \mathbf{c}_{u,v} \\ \mathbf{c}_{v,u} \end{bmatrix} &= \begin{bmatrix} \mathbf{p}_{u,v} + \lambda_{i_u}\mathbf{q}_{u,v} \\ \mathbf{p}_{v,u} + \lambda_{i_v}\mathbf{q}_{v,u} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_{u,v} + \lambda_{i_u}\mathbf{q}_{u,v} \\ \mathbf{p}_{u,v} + \lambda_{i_v}\mathbf{q}_{u,v} \end{bmatrix} \\ &= \begin{bmatrix} 1 & \lambda_{i_u} \\ 1 & \lambda_{i_v} \end{bmatrix} \begin{bmatrix} \mathbf{p}_{u,v} \\ \mathbf{q}_{u,v} \end{bmatrix}. \end{aligned}$$

Gaussian elimination is performed on $\mathbf{c}_{u,v}$ and $\mathbf{c}_{v,u}$ to solve $\mathbf{p}_{u,v}$ and $\mathbf{q}_{u,v}$.

Step 2: Define an $\alpha \times \alpha$ matrix $\tilde{\mathbf{S}} = (\tilde{\mathbf{s}}_{v,u})$ as

$$\tilde{\mathbf{S}} = \begin{bmatrix} \mathbf{\Phi}^{i_1} \\ \vdots \\ \mathbf{\Phi}^{i_\alpha} \end{bmatrix} \mathbf{S}. \tag{37}$$

Due to the symmetry of $\mathbf{S}$, $(\tilde{\mathbf{S}}^v)^\top = \mathbf{S}(\mathbf{\Phi}^{i_v})^\top$. For each $v \in 1:\alpha$, form the $\alpha \times \alpha$ linear system

$$\begin{bmatrix} \mathbf{p}_{1,v} \\ \vdots \\ \mathbf{p}_{v-1,v} \\ \mathbf{p}_{v+1,v} \\ \vdots \\ \mathbf{p}_{k,v} \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^{i_1} \\ \vdots \\ \mathbf{\Phi}^{i_{v-1}} \\ \mathbf{\Phi}^{i_{v+1}} \\ \vdots \\ \mathbf{\Phi}^{i_k} \end{bmatrix} (\tilde{\mathbf{S}}^v)^\top,$$

so that $(\tilde{\mathbf{S}}^v)^\top$ can be solved by performing Gaussian elimination. After solving $\tilde{\mathbf{S}}$, we further solve the linear system (37). Hence, $\mathbf{S}_u$ can be decoded by Gaussian elimination on $\tilde{\mathbf{s}}_{v,u}, v = 1, \ldots, \alpha$.

$\mathbf{T}$ can be solved by the same procedure and hence is omitted.

For our method, storage space of $2\alpha \times \alpha L = 2(k-1)^2 L$ symbols are needed; while in [20] the decoding needs space of $2k(k-1)L$ symbols. Hence, we can reduce space of $2(k-1)L$ symbols. In addition, the computation of $\mathbf{Y}^{i_u}(\mathbf{\Phi}^{i_u})^\top$ are omitted in our method, such that α multiplications of a vector and a sequence can be reduced.

In [21], finite-field MSR codes for $d \geq 2k-2$ are constructed, where the decoding procedure involves the decoding of the finite-field PM MSR codes with $d = 2k-2$. Therefore, our decoding scheme mentioned above can also be substituted into the decoding of these MSR codes for $d \geq 2k-2$, and reduce the corresponding decoding complexity.

B. Extension to Cyclic-Shift Regenerating Codes

The cyclic-shift regenerating codes in [28] employ a cyclic-shift operation defined as

$$\left(z_c^t \mathbf{a}\right)[l] = \begin{cases} \mathbf{a}\left[l + L - t\right], & 1 \leq l \leq t, \\ \mathbf{a}\left[l - t\right], & t < l \leq L. \end{cases}$$

Same as the setting in Section II-B, let

$$\mathbf{y}_{i,j} = \sum_{u=1}^{d} z_c^{t_{i,u}} \mathbf{m}_{u,j}, \quad 1 \leq i \leq n, 1 \leq j \leq \alpha,$$

where $t_{i,u} \geq 0$ are integers. Denoting $\mathbf{\Psi} = (z_c^{t_{i,j}})$, the encoding follows the same form as (1).

A system of cyclic-shift equations can be expressed as

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_k \end{bmatrix} = \mathbf{\Psi} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_k \end{bmatrix},$$

where $\det(\mathbf{\Psi})$ has an inverse element in $\mathbb{F}_2[z]/(1 + z + \cdots + z^{L-1})$. When $\mathbf{\Psi}$ is a Vandermonde matrix with $k-1$ strictly less than all divisors of L which are not equal to 1, the system can be solved using the LU method [28].

Similar to shift-XOR codes in Section IV and Section V, the decoding and repair of the cyclic-shift codes can be decomposed into a sequence of systems of cyclic-shift equations. When $n-1$ (where n is the number of storage nodes) is strictly less than all divisors of L which are not equal to 1, the sequence of systems can be solved by the LU method. The decoding and repair schemes built in this way have the same asymptotic complexity as that of our shift-XOR codes.

VII. Concluding Remarks

One technical contribution of this article is an efficient algorithm called shift-XOR elimination to solve a system of shift-XOR equations satisfying the RID property. Our algorithm consumes the exactly same number of XOR operations for decoding as encoding the input subsequences, and can be implemented in-place with only a small constant number of auxiliary integer variables. The shift-XOR elimination has the potential to be applied to and simplified the decoding costs of a range of codes based on shift-XOR operations.

For shift-XOR regenerating codes, the decoding/repair schemes are decomposed into a sequence of systems of shift-XOR equations. Our decoding/require schemes have much lower computation costs than the existing schemes for the shift-XOR regenerating codes, and demonstrate better or similar computation costs compared with the regenerating codes based on cyclic-shift and XOR operations. Our results provide a further evidence that shift and XOR operations can help to design codes with low computation costs.

Though we only studied the bit-wise shifts in this article, our algorithms can be extended to byte-wise or word-wise shifts to utilize multi-bit computation devices in parallel.

We are motivated to further explore the potential of shift-XOR codes. In one direction, we may extend the code constructions based on finite-field/cyclic-shift operations (e.g., [21]–[27]) to ones using shift and XOR. In another direction, we may investigate non-RID generator matrices, which may have lower storage overhead.

Appendix
Proof of Theorem 1

Here we prove Theorem 1, which concerns a $k \times k$ system of shift-XOR equations (6), where $\mathbf{\Psi} = (z^{t_{i,j}})$ satisfies the RID property in Definition 1. Recall L_b defined in (10).

Lemma 1: For integers $1 \leq u < v < k$,

$$t_{k-v,v+1} - t_{k-v,u} < \sum_{b=u}^{v} L_b < t_{k-u,v+1} - t_{k-u,u}.$$

Proof: The lemma can be proved by applying the RID property. On the one hand,

$$\begin{aligned}\sum_{b=u}^{v} L_b &= \sum_{b=u}^{v}(t_{k-b,b+1} - t_{k-b,b}) \\ &< \sum_{b=u}^{v}(t_{k-u,b+1} - t_{k-u,b}) \\ &= t_{k-u,v+1} - t_{k-u,u}.\end{aligned}$$

On the other hand,

$$\begin{aligned}\sum_{b=u}^{v} L_b &= \sum_{b=u}^{v}(t_{k-b,b+1} - t_{k-b,b}) \\ &> \sum_{b=u}^{v}(t_{k-v,b+1} - t_{k-v,b}) \\ &= t_{k-v,v+1} - t_{k-v,u}.\end{aligned}$$

□

Now we start to prove Theorem 1. We inductively show that all the bits to solve in each iteration depend on only the previous solved bits. We use l_i to denote the number of bits solved in $\mathbf{x}_i$, which are zero initially. For $k = 1$, the shift-XOR elimination is successful without using back substitution. We consider $k > 1$ in the following proof.

Firstly, for an iteration s in $1 : L_1$, we see that

$$\mathbf{x}_1[s] = \hat{\mathbf{x}}_1[s] + \sum_{u=2}^{k} \mathbf{x}_u[s + t_{k,1} - t_{k,u}]$$

As $s \le L_1 = t_{k-1,2} - t_{k-1,1}$, we have $s + t_{k,1} - t_{k,u} < t_{k,2} - t_{k,1} + t_{k,1} - t_{k,u} = t_{k,2} - t_{k,u} \le 0$ for $u \ge 2$ due to the RID property. Hence $\mathbf{x}_1[s] = \hat{\mathbf{x}}_1[s]$ so that $\mathbf{x}_1[s]$ can be solved. After iteration L_1, we have $l_1 = L_1$ and $l_i = 0$ for $i > 1$.

For certain $2 \le b \le k$, fix an iteration s in $\sum_{b'=1}^{b-1} L_{b'} + (1 : L_b)$ and an index i in $1 : b$. We assume that the algorithm runs successfully to iteration s with $\mathbf{x}_u[s - \sum_{b'=1}^{u-1} L_{b'}]$, for all $u < i$ solved, i.e.,

$$l_u = \begin{cases} s - \sum_{b'=1}^{u-1} L_{b'}, & 1 \le u < i, \\ s - 1 - \sum_{b'=1}^{u-1} L_{b'}, & i \le u \le b, \\ 0 & u > b. \end{cases} \tag{38}$$

To check whether $\mathbf{x}_i[l_i + 1]$ can be solved or not, we write by (9)

$$\mathbf{x}_i[l_i + 1] = \hat{\mathbf{x}}_i[l_i + 1] + \sum_{u \ne i} \mathbf{x}_u[l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u}]. \tag{39}$$

We can check the second term on the RHS is solved as follows:

1) For $1 \le u \le i - 1$, $\mathbf{x}_u[l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u}]$ has been solved as

$$\begin{aligned}&l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u} \\ &= s - 1 - \sum_{b'=1}^{i-1} L_{b'} + 1 + t_{k-i+1,i} - t_{k-i+1,u} \\ &= l_u - \sum_{b'=u}^{i-1} L_{b'} + t_{k-i+1,i} - t_{k-i+1,u} \\ &\le l_u,\end{aligned}$$

where the first two equalities are obtained by substituting the formula in (38), and the inequality is obtained by Lemma 1.

2) For $i + 1 \le u \le b$, $\mathbf{x}_u[l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u}]$ has been solved as

$$\begin{aligned}&l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u} \\ &= s - 1 - \sum_{b'=1}^{i-1} L_{b'} + 1 + t_{k-i+1,i} - t_{k-i+1,u} \\ &= l_u + 1 + \sum_{b'=i}^{u-1} L_{b'} + t_{k-i+1,i} - t_{k-i+1,u} \\ &\le l_u + t_{k-i,u} - t_{k-i,i} + t_{k-i+1,i} - t_{k-i+1,u} \\ &< l_u,\end{aligned}$$

where the first two equalities are obtained by substituting the formula in (38), the first inequality is obtained by Lemma 1, and the last inequality follows from the RID property.

3) For $b < u \le k$, $\mathbf{x}_u[l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u}] = 0$ as

$$\begin{aligned}&l_i + 1 + t_{k-i+1,i} - t_{k-i+1,u} \\ &= s - \sum_{b'=1}^{i-1} L_{b'} + t_{k-i+1,i} - t_{k-i+1,u} \\ &\le \sum_{b'=1}^{u-1} L_{b'} - \sum_{b'=1}^{i-1} L_{b'} + t_{k-i+1,i} - t_{k-i+1,u} \\ &= \sum_{i'=i}^{u-1} L_{i'} + t_{k-i+1,i} - t_{k-i+1,u} \\ &< 0,\end{aligned}$$

where the first equality follows from (38), the first inequality follows from $s \le \sum_{b'=1}^{b} L_{b'} \le \sum_{b'=1}^{u-1} L_{b'}$, and the second inequality is obtained by Lemma 1 and the RID property.

Therefore, all terms on the RHS of (39) are known and hence $\mathbf{x}_i[l_i+1]$ can be solved. The proof of the theorem is completed.

REFERENCES

[1] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[2] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.

[3] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. USENIX Conf. File Storage Technol.*, Mar. 2004, pp. 1–14.

[4] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 529–542, Mar. 1996.
[5] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, "The EVENODD code and its generalization," in *High Perform. Mass Storage Parallel I/O: Technologies and Applications*. Wiley-IEEE Press, Nov. 2001, pp. 187–208.
[6] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 889–901, Jul. 2008.
[7] H. Hou, Y. S. Han, K. W. Shum, and H. Li, "A unified form of EVENODD and RDP codes and their efficient decoding," *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5053–5066, Nov. 2018.
[8] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," ICSI Tech. Rep. TR-95-048, 1995.
[9] J. S. Plank and L. Xu, "Optimizing Cauchy Reed–Solomon codes for fault-tolerant network storage applications," in *Proc. 5th IEEE Int. Symp. Netw. Comput. Appl. (NCA)*, Jul. 2006, pp. 173–180.
[10] G.-L. Feng, R. H. Deng, F. Bao, and J.-C. Shen, "New efficient MDS array codes for RAID.Part II: Rabin-like codes for tolerating multiple (greater than or equal to 4) disk failures," *IEEE Trans. Comput.*, vol. 54, no. 12, pp. 1473–1483, Dec. 2005.
[11] H. Hou and Y. S. Han, "A new construction and an efficient decoding method for Rabin-like codes," *IEEE Trans. Commun.*, vol. 66, no. 2, pp. 521–533, Feb. 2018.
[12] C. W. Sung and X. Gong, "A zigzag-decodable code with the MDS property for distributed storage systems," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 341,–345.
[13] X. Gong and C. W. Sung, "ZigZag decodable codes: Linear-time erasure codes with applications to data storage," *J. Comput. Syst. Sci.*, vol. 89, pp. 190–208, Nov. 2017.
[14] T. Nozaki, "Fountain codes based on zigzag decodable coding," in *Proc. Int. Symp. Inf. Theory Appl.*, Oct. 2014, pp. 274–278.
[15] M. Dai, C. W. Sung, H. Wang, X. Gong, and Z. Lu, "A new zigzag-decodable code with efficient repair in wireless distributed storage," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1218–1230, May 2017.
[16] C. W. Sung and X. Gong, "Combination network coding: Alphabet size and zigzag decoding," in *Proc. Int. Symp. Inf. Theory Appl.*, Oct. 2014, pp. 699–703.
[17] H. Hou, K. W. Shum, M. Chen, and H. Li, "BASIC regenerating code: Binary addition and shift for exact repair," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 1621–1625.
[18] H. Hou, P. P. C. Lee, and Y. S. Han, "ZigZag-decodable reconstruction codes with asymptotically optimal repair for all nodes," *IEEE Trans. Commun.*, early access, Jul. 24, 2020, doi: 10.1109/TCOMM.2020.3011718.
[19] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proc. IEEE INFOCOM-26th IEEE Int. Conf. Comput. Commun.*, May 2007, pp. 2000–2008.
[20] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.
[21] S.-J. Lin, W.-H. Chung, Y. S. Han, and T. Y. Al-Naffouri, "A unified form of exact-MSR codes via product-matrix frameworks," *IEEE Trans. Inf. Theory*, vol. 61, no. 2, pp. 873–886, Feb. 2015.
[22] M. Kurihara and H. Kuwakado, "Generalization of Rashmi-Shah-Kumar minimum-storage-regenerating codes," 2013, *arXiv:1309.6701*. [Online]. Available: https://arxiv.org/abs/1309.6701
[23] M. Elyasi and S. Mohajer, "Determinant codes with helper-independent repair for single and multiple failures," *IEEE Trans. Inf. Theory*, vol. 65, no. 9, pp. 5469–5483, Sep. 2019.
[24] M. Elyasi and S. Mohajer, "A cascade code construction for (n, k, d) distributed storage systems," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2018, pp. 1241–1245.
[25] M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2001–2014, Apr. 2017.
[26] M. Ye and A. Barg, "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization," *IEEE Trans. Inf. Theory*, vol. 63, no. 10, pp. 6307–6317, Oct. 2017.
[27] J. Li, X. Tang, and C. Tian, "A generic transformation to enable optimal repair in MDS codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 64, no. 9, pp. 6257–6267, Sep. 2018.
[28] H. Hou, K. W. Shum, M. Chen, and H. Li, "BASIC codes: Low-complexity regenerating codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 3053–3069, Jun. 2016.
[29] Y. Guo, X. Fu, S. Yang, and K. W. Shum, "Shift-and-XOR storage code," CUHK(SZ) Tech. Rep., May 2020, pp. 1–6.

Ximing Fu received the B.S. and Ph.D. degrees from Tsinghua University in 2011 and 2018, respectively. He is currently a Post-Doctoral Fellow with The Chinese University of Hong Kong and the University of Science and Technology of China. His research interests include coding theory, network coding, storage codes, and cryptography.

Shenghao Yang (Member, IEEE) received the B.S. degree from Nankai University in 2001, the M.S. degree from Peking University in 2004, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2008.

He was a Visiting Student with the Department of Informatics, University of Bergen, Norway, in Spring 2017. He was a Post-Doctoral Fellow with the University of Waterloo from 2008 to 2009 and the Institute of Network Coding, The Chinese University of Hong Kong, from 2010 to 2012. He was with the Institute for Interdisciplinary Information Sciences, Tsinghua University, from 2012 to 2015. He is currently an Assistant Professor with The Chinese University of Hong Kong. His research interests include network coding, information theory, coding theory, and quantum information.

Zhiqing Xiao received the Ph.D. degree from Tsinghua University in 2016. He has been a Quantitative Researcher in an Investment Bank since 2016. His research interests include the application of probability theory, information theory, and machine learning.