Low-complexity Codes for Distributed Storage Systems

HOU, Hanxu

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Doctor of Philosophy

in

Information Engineering

The Chinese University of Hong Kong

September 2015

Abstract of thesis entitled:

Low-complexity Codes for Distributed Storage Systems

Submitted by HOU, Hanxu

for the degree of Doctor of Philosophy

at The Chinese University of Hong Kong in September 2015

Distributed storage systems are composed by many unreliable distributed storage nodes. A data file is stored in multiple storage nodes redundantly to provide high reliability. Erasure codes are being increasingly employed in distributed storage systems to combat the cost of reliably storing larger amounts of data, with optimal storage efficiency. Regenerating codes form a class of erasure codes which can achieve the optimal trade-off between the storage capacity and the bandwidth needed to repair a failed node. However, one of the critical drawbacks of existing regenerating codes in general is the high coding and repair complexities, since the coding and repair processes involve expensive multiplication operations in a finite field.

This thesis proposes a framework of linear codes with the binary parity-check code as the alphabet, named Binary Addition and Shift Implementable Cyclic-convolutional (BASIC) codes. When the encoding matrix is composed by the identity matrix and a rectangular Vandermonde matrix, the proposed BASIC codes reduce to BASIC array codes. We give a sufficient condition of Maximum-Distance Separable (MDS) property for the BASIC array codes with any number of parity columns. The proposed BASIC array code provides a larger spectrum of parameters, with comparable encoding complexity when compared with existing 2-erasure or 3-erasure MDS array codes, such as the row diagonal parity (RDP) code with 2 parity columns and the Star code with 3 parity columns. An efficient decoding method is presented to show that the decoding complexity of the proposed BASIC array code is less than that of the existing 4-erasure correcting MDS array codes.

A new family of regenerating codes is constructed. The proposed codes are called BASIC regenerating codes, which can be regarded as a concatenation coding scheme with the outer code being a binary parity-check code, and the inner code a regenerating code utilizing the binary parity-check code as the alphabet. We show that the proposed functional repair BASIC regenerating codes

can achieve the fundamental trade-off curve between the storage and repair bandwidth asymptotically of functional repair regenerating codes with less computational complexity. Furthermore, we demonstrate that some existing exact repair regenerating codes can be modified to exact repair BASIC regenerating codes with much less encoding, repair and decoding complexity.

摘要

分布式存儲系統是由很多不可靠的存儲節點組成。為了保證存儲數據的高可靠性，我們需要把數據文件的冗余數據存儲在多個節點。糾刪碼因其擁有最優的存儲效率而被廣泛的用於分布式存儲系統以提高數據可靠性。再生碼是一類可以達到存儲容量和修復一個失效節點所需帶寬的最優權衡。然而，再生碼的編解碼和修復過程中涉及很多計算量大的有限域操作，因此現有再生碼的一個關鍵缺陷為較高的編解碼以及修復**複雜度**。

我們提出了以二進制奇偶校驗碼為字母表的線性碼的編碼框架，我們稱之為BASIC碼。當BASIC碼的編碼矩陣由一個單位矩陣和范德蒙德矩陣組成時，我們稱BASIC碼為BASIC陣列碼。我們給出了有任意數量校驗列的BASIC陣列碼滿足最大距離可分離特征的充分條件。相較於現有的滿足最大距離可分離特征的擁有2個校驗列或者3個校驗列陣列碼，比如RDP碼和STAR碼，我們提出的BASIC陣列碼可以提供更多的參數選擇且編解碼**複雜度**相差不多。我們提出了BASIC 陣列碼有效的解碼算法，並證明BASIC陣列碼的解碼**複雜度**低於現有的滿足最大距離可分離特征的擁有4個校驗列或者以上校驗列陣列碼。

我們使用BASIC碼構造了一類新的再生碼，稱之為BASIC再生碼。我們提出了功能修復BASIC再生碼的一般

構造過程並證明功能修復BASIC再生碼可以漸進的達到存儲容量與修復帶寬的最優權衡。而相比於現有的功能修復再生碼，我們提出的功能修復BASIC再生碼的計算複雜度低。另外，我們證明一些現有的精確修復再生碼均可以修改為相應的精確修復BASIC再生碼，而編解碼計算複雜度和修復計算複雜度可以大幅度的降低。

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisors Prof. Minghua Chen and Prof. Kenneth Shum. I have learned a lot from them and they have given me a lot of professional knowledge, always encouraged me, and provided close guidance throughout the PhD period. This work would not have been finished without their continuous support of my PhD studies. Minghua and Kenneth always help me in my research, public speaking and career choices. I could not have imagined for better advisors.

I also want to thank the generous financial support of my supervisors, the Department of Information Engineering at the Chinese University of Hong Kong and the Institute of Network Coding. Without them I would not have been able to accomplish my PhD journey.

I would like to thank Prof. Kenneth Shum for the patient instructions and discussions. Besides, some results in Chapter 3 are

proved and written by Prof. Kenneth Shum. I want to thank Mr. Gareth Jone for his kindly proofreading throughout the thesis. I am sincerely grateful to my supervisor of Peking University, Prof. Hui Li who has supported me all the way. I also extend many thanks to my coauthors: Huanle Xu and Jun Chen, for their insightful comments and helpful discussions. I thank my labmates and friends in CUHK: Shaoquan Zhang, Lian Lu, Jinlong Tu, Ziyu Shao, Jincheng Zhang, Guanglin Zhang, Wenjie Zhang, Sheng Cai, Yang Yang, Benedit Max, Lei Deng, Ying Zhang, Hanling Yi, Mohammad Hassan Hajiesmaili, Daxiang Li, Xiangyu Sun, Minglong Zhang and Tao Guo. They make my PhD life full of fun.

I wish to thank my parents and my parents-in-law for all their love, encouragement, and for believing in me in good and bad times. Finally, I want to give my special appreciation and thanks to my lovely wife, whose endless love and care always helped and encouraged me during my challenging PhD period.

This work is dedicated to my family.

Contents

List of Figures

List of Tables

Table 1: Notations.

Symbol	Meaning
$\mathbb{F}_q$	Finite field of size q
$\mathbb{F}_2[z]$	Polynomials in variable z with coefficients in $\mathbb{F}_2$
$\mathcal{R}_m := \mathbb{F}_2[z]/(1+z^m)$	The quotient ring of polynomials with binary coefficients modulo $1+z^m$
$\mathcal{C}_m = \{a(z)(1+z) : a(z) \in \mathcal{R}_m\}$	A subset of $\mathcal{R}_m$ consists of polynomials in $\mathcal{R}_m$with even number of non-zero coefficients
$h(z) = 1+z+\cdots+z^{m-1}$	The check polynomial of $\mathcal{C}_m$
$\mathbb{F}_2[z]/(h(z))$	The quotient ring of polynomials with binary coefficients modulo $h(z)$
$\mathbf{G}$	Generator matrix of $\mathcal{C}_m$
$\mathbf{I}$	Identity matrix

Chapter 1

Introduction

Distributed storage systems usually consist of a large number of inexpensive and individually unreliable storage nodes interconnected via a large distributed network. We need to introduce redundancy to maintain system reliability, as the system has to tolerate different failures arising from unreliable storage nodes, software glitches, machine reboots and maintenance operations. Two kinds of redundancy are widely employed in commercial distributed storage systems. The most straightforward way is replication, in which data are duplicated and stored in multiple storage nodes. For example, the Google File System [1] and Hadoop Distributed File System (HDFS) [2] maintain three copies of each data. Although storage node seems inexpensive today, replication of the entire data is infeasible for massive scales of data.

As a result, most large-scale distributed storage systems, such as Google's Colossus File System [3] and Facebook's HDFS [4], are transiting to the use of erasure codes, which provide better storage efficiency and higher reliability.

Meanwhile, to maintain a target high reliability across time, the system is repaired whenever a storage node fails by replacing it with a new one. As the node failures in distributed storage systems are the "norm rather than exceptional" [1], keeping the repair process efficient by minimizing the amount of data that a new storage node needs to repair the system, which is called *repair bandwidth*, is a critical system design objective.

Regenerating codes (RGC), which were introduced by Dimakis *et al*. in [5] based on the concept of network coding, are erasure codes with the aim of minimizing the repair bandwidth. We differentiate two modes of repair. The first one is called exact repair and the second one is functional repair. In exact repair, the content of the new node is required to be the same as in the failed node. In functional repair, the content of the new node needs not be the same as in the failed one, but the property that any k nodes are sufficient in decoding the original file should be maintained. A fundamental trade-off between the storage amount

per node and the repair bandwidth is established in [5]. It is shown in [6] that all points on the trade-off curve can be achieved by linear network codes with bounded field size. The construction relies on the arithmetic of a finite field, and as in the application of linear network codes to a single-source multi-cast problem in general, the underlying finite field must be sufficiently large. However, multiplication and division in a finite field are costly to implement in software or hardware.

In a practical distributed storage system, higher computational complexity implies longer processing time, and larger energy consumption. High computational complexity suffers from high encoding, repairing and decoding time. The computational penalty cost of encoding and decoding time has been practically demonstrated on a testbed [7]. In this work, the authors show that in general, encoding over $\mathbb{F}_2$ is approximately 8 times faster than encoding over $\mathbb{F}_{256}$. Similarly, decoding over $\mathbb{F}_2$ is approximately 6 times faster than decoding over $\mathbb{F}_{256}$ on the same testbed. The importance of designing low-complexity RGC codes is highlighted in [8,9]. In the literature of coding for disk arrays, the computational complexity is reduced by replacing the arithmetic finite field with simple bit-wise operations. For example, in [10], a maximal-distance separable

(MDS) code with a convolutional code as alphabet is introduced by Piret and Krol. The constructed MDS convolutional code has lower decoding complexity, compared with Reed-Solomon block code. In [11], Blaum and Roth proposed a construction of array codes based on the ring of polynomials with binary coefficients modulo $1 + z + \cdots + z^{m-1}$ for some prime number m. A similar approach was considered by Xiao *et al.* in [12].

In this thesis, we introduce another class of linear storage codes which enables coding and repair by XOR and bit-wise *cyclic-shifts*. The new class of storage codes is called BASIC (Binary Addition and Shift Implementable Cyclic-convolutional) codes. The reduction on computational complexity is made possible by replacing the base field by a ring with a cyclic structure. A similar methodology in reducing computational complexity in network coding problems can be found in [13–15]. In [13], field multiplications are replaced by the operations of permuting the symbols, and in [14, 15], field multiplications are replaced by rotating the symbols. More generally, network codes over rings are discussed in [16]. The contributions of this thesis are summarized as follows:

1. BASIC codes is proposed in chapter 3 that is a new framework

of linear codes with the binary parity-check code as the alphabet. A necessary and sufficient condition for decodability of BASIC codes is given the chapter.

2. In chapter 4, we consider a special case of BASIC codes with the encoding matrix being composed by the identity matrix and a rectangular Vandermonde matrix, the special BASIC codes reduce to BASIC array codes. We propose a general construction of BASIC array codes correcting up to any disk erasures, and give a sufficient MDS condition for the proposed BASIC array codes. The proposed BASIC array code provides a larger spectrum of parameters, in comparison with existing MDS array codes, such as the RDP code and the EVENODD code.

3. A general decoding method is proposed in chapter 5 to recover some erasures for BASIC array codes. In addition, we present a fast decoding method using an LU factorization of the Vandermonde matrix to recover the information erasures. By employing the proposed fast decoding method in the decoding process, we show that the decoding complexity of BASIC array codes proposed in the previous chapter is lower than

those of existing 4-erasure correcting MDS array codes, such as the well-known Rabin-Like codes in [17] which are based on circular permutation matrices and BBV (Blaum, Bruck and Vardy) codes in [18] which are based on the quotient ring of polynomials modulo $1 + z + \cdots + z^{m-1}$.

4. In chapter 6, we give a general construction of functional repair BASIC regenerating codes and show that the presented functional repair BASIC regenerating codes can achieve all the benefits of functional repair RGC asymptotically.

5. In chapter 7, we show that the existing exact repair RGC can be modified to exact repair BASIC regenerating codes. Although in this chapter we only give the conversion of the product-matrix construction in [19], all the proposed exact repair RGC in [8, 19–23] can be converted to the exact repair BASIC codes.

6. The computational complexity of functional repair BASIC codes constructed in chapter 6 and the BASIC product-matrix RGC constructed in chapter 7 are evaluated in chapter 8. The results show that functional repair BASIC codes have less complexity in coding and repair processes in comparison with functional repair RGC over a finite field in [6], and the

coding and repair complexities of BASIC product-matrix RGC is much less than that of the product-matrix RGC in [19].

□ **End of chapter.**

Chapter 2

Background Study

2.1 Erasure Codes

Erasure codes, most prominently Reed Solomon (RS) codes, have been increasingly embraced by distributed storage systems as an alternative for replication. RS codes have existed for decades and are widely used in communication and storage systems, e.g., OceanStore [24] and TotalRecall [25] to name a few. In 1995, Blomer *et al.* presented an important performance improvement of coding complexity to RS codes using a Cauchy distribution matrix rather than the standard Vandermonde distribution matrix, termed *Cauchy Reed-Solomon* (CRS) codes [26]. With the proliferation of RS codes in storage applications, there has been a corresponding rise in the exploration of novel variants of RS codes targeting different

requirements of practical storage systems. Specific aspects that have been investigated in designing such coding techniques to reduce:

1. Coding complexity, binary MDS array code that only involve binary addition and shift in the encoding and decoding processes, such as EVENODD codes [27] and RDP codes [28].

2. Repair bandwidth, by applying network coding techniques that is called regenerating codes [19, 29, 30].

3. Disk I/O cost, some explicit constructions of codes such as Zigzag codes in [23] and Dress codes in [22] are proposed to minimize the I/O cost.

2.2 Binary MDS Array Codes

Binary MDS array codes are employed in storage systems, such as Redundant Arrays of Inexpensive Disks (RAID) [31], for the purpose of enhancing data reliability. The EVENODD [27] and RDP [28] codes are two important families of binary MDS array codes correcting double disk failures. The EVENODD code was extended by Blaum, Bruck and Vardy [18] for three or more parity-check disks, with the additional assumption that the multiplicative order of 2 mod m is equal to $m - 1$. The extended EVENODD

code in [18] is termed as BBV (Blaum, Bruck and Vardy) code, it is proved that BBV code is always MDS for three parity-check disks, but may not be MDS for four or more party-check disks. A necessary and sufficient condition for four parity-check disks to be MDS is given in [18], and some scattered results for more than four parity-check disks are provided. The STAR code extends the EVENODD code to three parity-check disks [32], and the Triple-Star code improves the encoding/decoding efficiency of STAR code [33]. Based on circular permutation matrices and Reed-Solomon (RS) code, a construction of MDS array codes tolerating three disk failures is presented in [34]. The authors of [34] modified their construction by replacing RS code with Rabin code, and produced a class of Rabin-like array codes, which can tolerate four or more disk erasures [17].

2.3 Regenerating Codes

In regenerating codes, a data file of B symbols over the finite field $\mathbb{F}_{2^w}$ is encoded into $n\alpha$ symbols and distributed to n storage nodes, with each node storing α symbols such that the file can be decoded from any set of k nodes. Furthermore, upon the failure of

a storage node, we want to repair the failed node by downloading β symbols from each of the d surviving nodes, with the amount of data sent to the new node being as little as possible.

It is shown in [5] that, the minimization of repair bandwidth for functional repair is closely related to the single-source multi-cast problem in network coding theory. After formulating the problem using an information flow graph, a fundamental trade-off between the amount of storage per node and the repair bandwidth is established as follows,

$$B \leq \sum_{i=1}^{k} \min\{(d-i+1)\beta, \alpha\}. \tag{2.1}$$

A storage code attains the fundamental optimal trade-off between the amount of storage per node and the repair bandwidth in (2.1) is termed as regenerating codes (RGC). If we fix the value of file size B, we obtain a trade-off curve on storage α and repair bandwidth β. The two extreme points in this trade-off are termed the minimum storage regeneration (MSR) and minimum bandwidth regeneration (MBR) points respectively. The MSR point corresponds to

$$\alpha_{\text{MSR}} = \frac{B}{k}, \ \beta_{\text{MSR}} = \frac{B}{k(d-k+1)},$$

and the MBR point corresponds to

$$\alpha_{\text{MBR}} = \frac{2dB}{k(2d-k+1)}, \ \beta_{\text{MBR}} = \frac{2B}{k(2d-k+1)}.$$

The problem of exact repair RGC was investigated in [8, 19–23, 35–37], all of which address either the MBR case or the MSR case. The paper [19] presents the optimal explicit constructions of MBR codes for all feasible values of parameters $k \leq d \leq n-1$ and MSR codes for the parameters $2k-2 \leq d \leq n-1$, using the proposed product-matrix framework. The concept of uncoded repair is introduced in [20, 22]. RGC with uncoded repair does not require any arithmetic operation during the repair process; a helper node merely reads out the symbols from the memory and sends them to the new node. This minimizes the computational complexity of repair. Some explicit constructions of RGC at the MBR point with uncoded repair can be found in [8, 20–22]. It is shown in [37] that it is not possible to construct the uncoded repair MBR codes when $d \neq n-1$ for exact repair. On the MSR point, uncoded repair RGC for functional repair is discussed in [35, 36]. However, the code parameters considered in [35, 36] are restricted to $k = 2$ and $k = n-2$.

Recently, zigzag code [23] was constructed on the MSR point to achieve the optimal exact repair. The code parameters considered in [23] are relaxed to $k+1 \leq d \leq n-1$, at a cost of a very high level of sub-symbolization. This is because zigzag code is a

vector-linear code, while the codes in [19, 35, 36] are scalar-linear codes. Although the problem of determining the rate region for exact repair RGC in general remains open, some recent results on the fundamental limit on repair bandwidth can be found in [38, 39].

In [6], existence of linear network codes achieving all points on the fundamental trade-off curve for functional repair RGC is shown. The construction relies on arithmetic of a finite field, and as in the application of linear network codes to a single-source multi-cast problem in general, the underlying finite field must be sufficiently large.

□ **End of chapter.**

Chapter 3

BASIC Codes

In this chapter, we first propose the mathematical framework of BASIC codes, and then give a necessary and sufficient condition for decodability of BASIC codes.

3.1 Mathematical Framework of BASIC Codes

In this section, we will introduce the necessary algebra and mathematical framework used for BASIC codes.

3.1.1 Binary Cyclic Code

In this subsection we review some facts on binary cyclic codes [40, Chapters 7]. A linear code $\mathcal{C}$ over $\mathbb{F}_2$ is called a *binary cyclic code* if, whenever $\boldsymbol{c} = (c_0, c_1, \ldots, c_{m-1})$ is in $\mathcal{C}$, then $\boldsymbol{c}' =$

$(c_{m-1}, c_0, \ldots, c_{m-2})$ is also in $\mathcal{C}$. The codeword $\boldsymbol{c}'$ is obtained by cyclically shifting the components of the codeword $\boldsymbol{c}$ one place to the right. Let m be a positive odd number and let $\mathcal{R}_m$ be the ring

$$\mathcal{R}_m := \mathbb{F}_2[z]/(1+z^m). \tag{3.1}$$

The element of $\mathcal{R}_m$ will be referred to as a *polynomial* in the sequel. The vector $(a_0, a_1, \ldots, a_{m-1}) \in \mathbb{F}_2^m$ is the codeword corresponding to the polynomial $\sum_{i=0}^{m-1} a_i z^i$. The indeterminate z represents the *cyclic-right-shift* operator on the codewords. A subset of $\mathcal{R}_m$ is a binary cyclic code of length m if the subset is closed under addition and closed under multiplication by z.

In this thesis, we consider the simple parity-check code, $\mathcal{C}_m$, which consists of polynomials in $\mathcal{R}_m$ with an even number of non-zero coefficients,

$$\mathcal{C}_m = \{a(z)(1+z) : \ a(z) \in \mathcal{R}_m\}. \tag{3.2}$$

The dimension of $\mathcal{C}_m$ over $\mathbb{F}_2$ is $m-1$, and the *check polynomial* of $\mathcal{C}_m$ is $h(z) := 1 + z + \cdots + z^{m-1}$.

3.1.2 Design Framework of BASIC Code

We define *BASIC* (Binary Addition and Shift Implementable Cyclic-convolutional) code as follows.

Definition 1. *A BASIC code is an $\mathcal{R}_m$-linear code with the binary parity-check code $\mathcal{C}_m$ as the alphabet.*

Given an odd number m and positive integers κ and ν, the encoding of a BASIC code is mapped from $\mathbb{F}_2^{(m-1)\kappa}$ to $\mathcal{C}_m^{\nu}$, specified by a $\kappa \times \nu$ *generator matrix* $\mathbf{G}$ over $\mathcal{R}_m$. The encoding can be performed in two steps. Firstly, we divide the $(m-1)\kappa$ bits into κ groups, with each group containing $m-1$ bits. To each group of bits, we append a parity-check bit and form a polynomial in $\mathcal{C}_m$. We put the resulting polynomials together and form a κ-tuple $\mathbf{w} = (s_1(z), s_2(z), \ldots, s_\kappa(z)) \in \mathcal{C}_m^{\kappa}$. The codeword in the BASIC code corresponding to the $(m-1)\kappa$ input bits is obtained by multiplying $\mathbf{w}$ and $\mathbf{G}$.

Henceforth, we will call a polynomial in $\mathcal{C}_m$ a *source packet* or a *data packet*. A component in $\mathbf{wG}$ will be called a *coded packet*. A coded packet is thus an $\mathcal{R}_m$-linear combination of the κ data packets, with elements from $\mathcal{R}_m$ as the coefficients.

Remarks: There is an alternate description of BASIC codes in terms of group algebra and module. The ring $\mathcal{R}_m$ defined in (3.1) is isomorphic to the group algebra $\mathbb{F}_2\mathbb{Z}_m$, where $\mathbb{Z}_m$ is the cyclic group of size m, and the ring $\mathcal{C}_m$ defined in (3.2) is isomorphic to a subring of $\mathbb{F}_2\mathbb{Z}_m$. A BASIC code is a sub-module of the free $\mathbb{F}_2\mathbb{Z}_m$-

Table 3.1: An example of storage code for four nodes.

Node 1	Node 2	Node 3	Node 4
$s_{1,0}$	$s_{2,0}$	$s_{3,0} = s_{1,0} + s_{2,0}$	$s_{4,0} = s_{1,6} + s_{2,0}$
$s_{1,1}$	$s_{2,1}$	$s_{3,1} = s_{1,1} + s_{2,1}$	$s_{4,1} = s_{1,0} + s_{2,1}$
$s_{1,2}$	$s_{2,2}$	$s_{3,2} = s_{1,2} + s_{2,2}$	$s_{4,2} = s_{1,1} + s_{2,2}$
$s_{1,3}$	$s_{2,3}$	$s_{3,3} = s_{1,3} + s_{2,3}$	$s_{4,3} = s_{1,2} + s_{2,3}$
$s_{1,4}$	$s_{2,4}$	$s_{3,4} = s_{1,4} + s_{2,4}$	$s_{4,4} = s_{1,3} + s_{2,4}$
$s_{1,5}$	$s_{2,5}$	$s_{3,5} = s_{1,5} + s_{2,5}$	$s_{4,5} = s_{1,4} + s_{2,5}$
$s_{1,6} = \sum_{j=0}^{5} s_{1,j}$	$s_{2,6} = \sum_{j=0}^{5} s_{2,j}$	$s_{3,6} = s_{1,6} + s_{2,6}$	$s_{4,6} = s_{1,5} + s_{2,6}$

module $\mathcal{C}_m^n$. A BASIC code can be regarded as a quasi-cyclic code (See e.g. [41, 42]). Nonetheless, the quasi-cyclic codes considered in [41, 42] are submodules of the free $\mathbb{F}_2\mathbb{Z}_m$-module $(\mathbb{F}_2\mathbb{Z}_m)^m$, and the objective is to maximize the the minimum distance as a code of length mn over a base field. In this thesis, BASIC codes are considered a code of length n over the alphabet $\mathcal{C}_m$.

Example: Consider an example of BASIC code with parameters $m = 7$, $\kappa = 2$ and $\nu = 4$. The two data packets are $s_i(z) = \sum_{j=0}^{6} s_{i,j} z^j$, for $i = 1, 2$, and the generator matrix is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & z \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

The example is illustrated in Table 3.1. The bit

$$s_{i,6} := \sum_{j=0}^{5} s_{i,j}$$

in the last row in Table 3.1 is the *parity-check bit* of bits $s_{i,0}, \ldots, s_{i,5}$, $i = 1, 2, 3, 4$. We note that the redundant bits in node 3 are computed by adding the bits stored in nodes 1 and 2, meanwhile, the redundant bits in node 4 are computed by adding the bits in node 1 and a cyclically shifted version of the bits in node 2.

Recall that $h(z) = 1 + z + \cdots + z^{m-1}$ and is the check polynomial of $\mathcal{C}_m$. By its definition, $c(z)h(z) = 0$ for all $c(z) \in \mathcal{C}_m$, a coded packet can be obtained as an $\mathcal{R}_m$-linear combination of the data packets in more than one way. We can add the check polynomial $h(z)$ to any entry in $\mathbf{G}$ without modifying the encoding function. For example, we can pick

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & & z & \\ 0 & 1 & 1 & z + z^2 + \cdots + z^{m-1} & & \end{bmatrix}.$$

as the generator matrix of the example.

We remark that the last bit $s_{i,m-1}$ does not need to be stored in practical implementation. As the last bit in each node can be obtained by summing the first $m - 1$ bits, we can compute the last bit $s_{i,m-1}$ if necessary.

3.1.3 Erasure Decoding

A collection of κ coded packets is said to be *decodable* or *information set* if we can recover the source packets from these κ coded packets. In this subsection, we give a necessary and sufficient condition for decodability. Before that, we need to first introduce a definition.

Definition 2. *A polynomial $f(z)$ in $\mathcal{R}_m$ is called $\mathcal{C}_m$-*invertible *if we can find a polynomial $\tilde{f}(z) \in \mathcal{R}_m$ such that $f(z)\tilde{f}(z)$ is equal to either 1 or $1 + h(z)$.*

For a subset $\mathcal{I} \subseteq \{1, 2, \ldots, \nu\}$ with $|\mathcal{I}| = \kappa$, we let $\mathbf{G}_{\mathcal{I}}$ be the $\kappa \times \kappa$ submatrix of $\mathbf{G}$ obtained by retaining the columns indexed by $\mathcal{I}$.

Theorem 1. *Let $\mathcal{I} \subseteq \{1, 2, \ldots, \nu\}$ be an index set with cardinality κ. The coded packets indexed by $\mathcal{I}$ are decodable if $\det(\mathbf{G}_{\mathcal{I}})$ is $\mathcal{C}_m$-invertible.*

Proof. Let $s_1(z), \ldots, s_\kappa(z)$ be the data packets, and $p_1(z), \ldots, p_\kappa(z)$ be the coded packets indexed by $\mathcal{I}$,

$$(p_1(z), \ldots, p_\kappa(z)) = (s_1(z), \ldots, s_\kappa(z)) \cdot \mathbf{G}_{\mathcal{I}}.$$

Suppose that the determinant of $\mathbf{G}_{\mathcal{I}}$ is $\mathcal{C}_m$-invertible. Let $\delta(z)$ be a polynomial in $\mathcal{R}_m$ such that $\delta(z)\det(\mathbf{G}_{\mathcal{I}})$ is equal to 1 or $1 +$

$h(z)$. We can recover the data packets from the coded packets by

$$\begin{aligned}
&(p_1(z), \ldots, p_\kappa(z)) \cdot adj(\mathbf{G}_\mathcal{I}) \cdot \delta(z) \\
&= (s_1(z), \ldots, s_\kappa(z)) \cdot \mathbf{G}_\mathcal{I} \cdot adj(\mathbf{G}_\mathcal{I}) \cdot \delta(z) \\
&= (s_1(z), \ldots, s_\kappa(z)) \cdot \det(\mathbf{G}_\mathcal{I}) \cdot \delta(z) \\
&= (s_1(z), \ldots, s_\kappa(z)),
\end{aligned}$$

where $adj(\mathbf{G}_\mathcal{I})$ denotes the adjoint of $\mathbf{G}_\mathcal{I}$ [43, p.20]. In the last step, we have used the fact that $s_i(z)(1+h(z)) = s_i(z)$ if $s_i(z) \in \mathcal{C}_m$. □

We next give a criterion for checking whether a polynomial in the ring $\mathcal{R}_m$ is $\mathcal{C}_m$-invertible. Let $f_1(z), f_2(z), \ldots, f_L(z)$ be the prime factorization of the check polynomial $h(z)$ over $\mathbb{F}_2$. The irreducible polynomials $f_1(z)$ to $f_L(z)$ are distinct as they are divisors of $1+z^m$ and m is an odd number. We recall that in a general commutative ring R with identity, an element $u \in R$ is called a *unit* if we can find an element $\tilde{u} \in R$ such that $u\tilde{u}$ is equal to the identity element in R.

Theorem 2. *Suppose that $f_1(z)$, $f_2(z), \ldots, f_L(z)$ are the irreducible factors of the check polynomial $h(z)$. Let $a(z)$ be a polynomial in $\mathcal{R}_m$. The following are equivalents:*

1. *$a(z)$ is $\mathcal{C}_m$-invertible.*

2. $a(z) \bmod h(z)$ *is a unit in* $\mathbb{F}_2[z]/(h(z))$.

3. $a(z) \bmod f_\ell(z)$ *is a unit in* $\mathbb{F}_2[z]/(f_\ell(z))$ *for all* $\ell = 1, 2, \ldots, L$.

Proof. (1) $\Leftrightarrow$ (2). Define $f_0(z)$ as the polynomial $1 + z$. The polynomial $1 + z^m \in \mathbb{F}_2[z]$ can be factored into $f_0(z)$ and $h(z)$. Using the Chinese remainder theorem, it can be shown that the ring $\mathcal{R}_m$ is isomorphic to the direct sum

$$\mathcal{R}'_m := \mathbb{F}_2[z]/(f_0(z)) \oplus \mathbb{F}_2[z]/(h(z)).$$

Indeed, the mapping $\phi : \mathcal{R}_m \to \mathcal{R}'_m$ defined by

$$a(z) \mapsto (a(z) \bmod 1 + z,\ a(z) \bmod h(z)),$$

and the mapping $\phi' : \mathcal{R}'_m \to \mathcal{R}_m$ defined by

$$(a_0(z), a_1(z)) \mapsto h(z)a_0(z) + (1 + h(z))a_1(z) \bmod 1 + z^m$$

are inverse of each other. Suppose $a(z) \bmod h(z)$ is a unit in $\mathbb{F}_2[z]/(h(z))$, i.e., there is a polynomial $d(z)$ such that $\phi(a(z)d(z)) = (a, 1)$, where a is either 0 or 1. Hence $a(z)d(z)$ is equal to either $\phi'((0, 1)) = 1 + h(z)$ or $\phi'((1, 1)) = 1$. This proves that $a(z)$ is $\mathcal{C}_m$-invertible.

Conversely, suppose that $a(z)$ is $\mathcal{C}_m$-invertible. There is a polynomial $\tilde{a}(z) \in \mathcal{R}_m$ such that $a(z)\tilde{a}(z)$ is equal to 1 or $1 + h(z)$.

If we apply the mapping ϕ to $a(z)\tilde{a}(z)$, then we have $\phi(a(z)\tilde{a}(z)) = (a, 1)$, for some $a \in \mathbb{F}_2$. Therefore $a(z) \bmod h(z)$ is a unit.

(2) $\Leftrightarrow$ (3). Using the fact that $h(z)$ can be factorized into $f_1(z)f_2(z)\cdots f_L(z)$, the equivalence between the second and third conditions in the theorem can be shown by another application of Chinese remainder theorem. □

A necessary and sufficient condition for decodability is summarized in the following Corollary.

Corollary 3. *Consider a BASIC code with $\kappa \times \nu$ generator matrix $\mathbf{G}$, and an index set $\mathcal{I} \subseteq \{1, 2, \ldots, \nu\}$ with cardinality κ. The coded packets indexed by $\mathcal{I}$ is decodable if and only if $\det(\mathbf{G}_\mathcal{I})$ is $\mathcal{C}_m$-invertible.*

Proof. We have already shown the "if" part in Theorem 1. In the reverse direction, suppose that $\det(\mathbf{G}_\mathcal{I})$ is not $\mathcal{C}_m$-invertible. Using the same notation as in Theorem 2, we have $\det(\mathbf{G}_\mathcal{I}) = 0 \bmod f_{\ell_0}(z)$ for some $\ell_0 \in \{1, 2, \ldots, L\}$. If we reduce the matrix $\mathbf{G}_\mathcal{I}$ modulo $f_{\ell_0}(z)$ entry-wise, the resulting matrix is singular as a matrix over the finite field $\mathbb{F}_2[z]/(f_{\ell_0}(z))$. We can find a non-zero vector $\bar{\mathbf{a}} = (\bar{a}_1(z), \ldots, \bar{a}_\kappa(z)$, with each component belonging to $\mathbb{F}_2[z]/(f_{\ell_0}(z))$, such that $\bar{\mathbf{a}}\mathbf{G}_\mathcal{I} \bmod f_{\ell_0}(z)$ is the zero vector. For

$j = 1, 2, \ldots, \kappa$, choose $a_j(z) \in \mathcal{C}_m$ such that

$$a_j(z) = \begin{cases} \bar{a}_j(z) \bmod f_\ell(z) & \text{for } \ell = \ell_0 \\ 0 \bmod f_\ell(z) & \text{for } \ell \neq \ell_0. \end{cases}$$

If we take $a_j(z)$'s as the source packets, then ν-tuple obtained by $(a_1(z), a_2(z), \ldots, a_\kappa(z))\mathbf{G}_\mathcal{I}$ is the zero ν-tuple. The encoding map is not injective and therefore the coded packets indexed by $\mathcal{I}$ are not decodable. □

Continuing the example of $\nu = 4$, $\kappa = 2$ and $m = 7$, the polynomial $1 + z^7$ can be factorized as a product of $f_0(z) = 1 + z$, $f_1(z) = 1 + z + z^2$ and $f_2(z) = 1 + z^2 + z^3$. We can check that any two coded packets are decodable. For instance, if the index set is $\mathcal{I} = \{3, 4\}$, the determinant $\det(\mathbf{G}_\mathcal{I}) = 1 + z$ is not divisible by $f_1(z)$ and $f_2(z)$. Indeed, $1 + z$ is $\mathcal{C}_m$-invertible because

$$(1 + z)(z + z^3 + z^5) = z + z^2 + \cdots + z^6 = 1 + h(z).$$

We can thus compute the two data packets from node 3 and node 4 according to Theorem 1.

Some remarks on implementation are in order. In software implementation, we can implement a cyclic-shift by using a pointer. We store the m bits consecutively in the memory, and use a pointer

to store the beginning address of the packet. A cyclic-shift can be done by modifying the pointer only, without modifying the packet itself. We can also modify BASIC codes and replace bit-wise cyclic-shift by byte-wise cyclic-shift, which is more amenable to software implementation. In hardware implementation, a cyclic-shift can easily be done by having the bits cyclically shifted in a shift register.

Remarks: If the parameter m is a positive even number, then the polynomial $1 + z$ is a factor of the check polynomial $h(z)$. From the decodability condition of BASIC code in Theorem 2 and Corollary 3, if a BASIC code is decodable, then the determinants of all the $\nu \times \nu$ submatrices of the generator matrix do not have the factor $1 + z$, which is hard to satisfy in general. This is why we choose m to be a positive odd number. Someone may try to add more parity-check bits to formulate repeated-root cyclic code [44], when m is a positive even number.

□ **End of chapter.**

Chapter 4

BASIC Array Codes

In this chapter, we consider BASIC array codes, a class of BASIC codes with encoding matrix being composed by the identity matrix and a rectangular Vandermonde matrix.

4.1 Construction of BASIC Array Codes

In this section we consider a $(m-1) \times (k+r)$ BASIC array code, termed $\mathcal{C}(k, r, m)$, where m is an odd prime and $m \geq \max\{k, r\}$. The columns are identified with the disks. Columns 0 to $k-1$ are called the *information columns*, which store the information bits. Columns k to $k+r-1$ are called the *parity columns*, which store the redundant bits.

For $i = 0, 1, \ldots, m-2$ and $j = 0, 1, \ldots, k-1$, we let the i-th

information bit in the j-th information column be denoted by $s_{i,j}$. For notational convenience, we let $s_{m-1,j}$ be the *parity-check bit*

$$s_{m-1,j} \triangleq s_{0,j} + s_{1,j} + \cdots + s_{m-2,j},$$

associated with column j. This parity-check bit is not stored in the array code, but is essential in encoding and decoding.

The r parity columns are computed by XOR-ing the information bits. For $i = 0, 1, \ldots, m-2$ and $j = 0, 1, \ldots, r-1$, let the i-th redundant bit stored in the j-th parity column be defined as

$$c_{i,j} \triangleq \sum_{\ell=0}^{k-1} s_{\langle i-j\ell \rangle_m, \ell} \tag{4.1}$$

where $\langle x \rangle_m$ in the subscript means that we divide the integer x by m and take the remainder. We sometimes omit the brackets "$\langle\ \rangle_m$" for notation simplicity. The first parity column, i.e., the column containing the redundant bits $c_{0,1}, \ldots, c_{0,m-2}$, is named as *row parity column*. The other $r-1$ parity columns are called the *diagonal parity columns*.

For $j = 0, 1, \ldots, r-1$, we let $c_{p-1,j}$ to be the parity-check bit

$$c_{p-1,j} \triangleq c_{0,j} + c_{1,j} + \cdots + c_{m-2,j},$$

associated to the j-th parity column. For $j = 0, 1, \ldots, k-1$, we

represent the m information bits $s_{0,j}, \ldots, s_{m-1,j}$ by the polynomial

$$s_j(z) \triangleq s_{0,j} + s_{1,j}z + \cdots + s_{m-2,j}z^{m-2} + s_{m-1,j}z^{m-1}. \quad (4.2)$$

Similarly, we represent the m bits $c_{0,j}, \ldots, c_{m-1,j}$ by the polynomial

$$c_j(z) \triangleq c_{0,j} + c_{1,j}z + \cdots + c_{p-2,j}z^{p-2} + c_{p-1,j}z^{p-1}, \quad (4.3)$$

for $j = 0, 1, \ldots, r-1$.

The encoding can be now performed by taking the product

$$(s_0(z), s_1(z), \cdots, s_{k-1}(z)) \cdot \mathbf{G},$$

where $\mathbf{G}$ is the $k \times (k+r)$ generator matrix of BASIC array code $\mathcal{C}(k, r, m)$,

$$\mathbf{G} = \left[\mathbf{I} \mid \mathbf{P}\right].$$

The first k columns of $\mathbf{G}$ form the $k \times k$ identity matrix $\mathbf{I}$, and the last r columns form a rectangular Vandermonde matrix $\mathbf{P}$ given by

$$\mathbf{P} \triangleq \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & z & z^2 & \cdots & z^{r-1} \\ 1 & z^2 & z^4 & \cdots & z^{2(r-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z^{k-1} & z^{2(k-1)} & \cdots & z^{(r-1)(k-1)} \end{bmatrix}. \quad (4.4)$$

All the arithmetic operations are performed in the ring $\mathcal{R}_m$.

Table 4.1: The BASIC array code $\mathcal{C}(4,3,5)$ ($k=4$, $r=3$, $m=5$).

Disk 4	Disk 5	Disk 6
$s_{0,0}+s_{0,1}+s_{0,2}+s_{0,3}$	$s_{0,0}+\mathbf{s_{4,1}}+s_{3,2}+s_{2,3}$	$s_{0,0}+s_{3,1}+s_{1,2}+\mathbf{s_{4,3}}$
$s_{1,0}+s_{1,1}+s_{1,2}+s_{1,3}$	$s_{1,0}+s_{0,1}+\mathbf{s_{4,2}}+s_{3,3}$	$s_{1,0}+\mathbf{s_{4,1}}+s_{2,2}+s_{0,3}$
$s_{2,0}+s_{2,1}+s_{2,2}+s_{2,3}$	$s_{2,0}+s_{1,1}+s_{0,2}+\mathbf{s_{4,3}}$	$s_{2,0}+s_{0,1}+s_{3,2}+s_{1,3}$
$s_{3,0}+s_{3,1}+s_{3,2}+s_{3,3}$	$s_{3,0}+s_{2,1}+s_{1,2}+s_{0,3}$	$s_{3,0}+s_{1,1}+\mathbf{s_{4,2}}+s_{2,3}$

The encoding procedure can be described as follows. Given $k(m-1)$ information bits, we append k parity-check bits and obtain the k data polynomials $s_0(z), s_1(z), \cdots, s_{k-1}(z)$. The r coded polynomials can be computed by taking the product

$$(s_0(z), s_1(z), \cdots, s_{k-1}(z)) \cdot \mathbf{P}.$$

We ignore the terms with degree $m-1$ and store the coefficients of the terms in the polynomials of degrees from 0 to $m-2$.

Table 4.1 shows the three parity columns of BASIC array code $\mathcal{C}(4,3,5)$. In Table 4.1, the terms in boldface are the parity-check bits associated with columns 0 to 3. The coded packets in disk 4 to disk 6 are computed by adding some cyclically shifted version of the 4 data packets. The example of Table 3.1 in Section 3.1 is in fact a BASIC array code $\mathcal{C}(2,2,7)$.

We want to mention a very similar code of $\mathcal{C}(k,r,m)$, BBV

code in [18]. BBV code is defined over arrays with dimensions $(m-1) \times (m+r)$, with m information columns and r parity columns. Each array column of length $m-1$ in BBV is viewed as polynomial of degree $\leq m-2$ over the finite field $\mathbb{F}_2$. The main difference of the proposed BASIC array code $\mathcal{C}(k,r,m)$ and the BBV code in [18] is as follows. The code in [18] is constructed based on the ring of polynomials with binary coefficients modulo $h(z) = 1 + z + \cdots + z^{m-1}$ for some prime number m, and BASIC array code $\mathcal{C}(k,r,m)$ is based on the ring of polynomials with binary coefficients modulo $1 + z^m$. We will show in the next chapter that BASIC array code $\mathcal{C}(k,r,m)$ has more efficient decoding algorithm than that of BBV code, because of the difference.

4.2 The MDS Property

In the following section, we are going to prove that the BASIC array code constructed in the last section satisfies the MDS property under a mild condition. In the rest of the chapter, the prime m is chosen such that the multiplication order of 2 in the ring $\langle \mathbb{Z}_m, +, \cdot \rangle$ is equal to $m-1$. For this case, the ring $\mathcal{C}_m$ defined in (3.2) is isomorphic to the finite field $\mathbb{F}_{2^{m-1}}$ [40, p. 197]. By Corollary 3,

we have that if the determinant of any $k \times k$ submatrix of the generator matrix $\mathbf{G}$ is $\mathcal{C}_m$-invertible, i.e., is a nonzero polynomial in $\mathbb{F}_2[z]/h(z)$, then the array code $\mathcal{C}(k, r, m)$ satisfies the MDS property. In other words, the array code $\mathcal{C}(k, r, m)$ is MDS if, for all $\ell = 1, 2, \ldots, \min\{k, r\}$, the determinant of each $\ell \times \ell$ submatrix of the matrix $\mathbf{P}$, regarded as a polynomial in $\mathbb{F}_2[z]$, is not divisible by $1 + z^m$.

Theorem 4. *If any $k \times k$ sub-matrix of the generator matrix $\mathbf{G}$, after reduction modulo $h(z)$, is a non-singular matrix over $\mathbb{F}_2[z]/(h(z))$, then the array codes $\mathcal{C}(k, r, m)$ satisfy the MDS property.*

Proof. Let $\mathbf{A}$ be a $k \times k$ sub-matrix of the generator matrix $\mathbf{G}$, and $\bar{\mathbf{A}}$ be the matrix obtained by reducing each entry of $\mathbf{A}$ mod $h(z)$. By the Chinese Remainder Theorem, $\bar{\mathbf{A}}$ can be regarded as a matrix over the finite field $\mathbb{F}_2[z]/(h(z))$. Since the matrix $\bar{\mathbf{A}}$ is non-singular over $\mathbb{F}_2[z]/(h(z))$, we can find the inverse of $\bar{\mathbf{A}}$. Let $\bar{\mathbf{A}}^{-1}$ be the inverse of $\bar{\mathbf{A}}$, then we can compute the inverse matrix of $\mathbf{A}$ over the ring $\mathcal{R}_m$ by using the Chinese Remainder Theorem for each entry of $\bar{\mathbf{A}}^{-1}$. □

Let $\mathbf{G}_k$ be any $k \times k$ submatrix of the generator matrix $\mathbf{G}$. Therefore, we only need to show that the matrix $\mathbf{G}_k$ is non-singular

over $\mathbb{F}_2[z]/(h(z))$. By employing the result of [18] and the above theorem, we prove the MDS condition given in the following theorem when $r \leq 8$.

Theorem 5. *Let m be a prime such that the multiplicative order of 2 mod m is equal to $m-1$. We have that:*

1. *The array code $\mathcal{C}(k, r, m)$ is MDS for $m \geq 5$ and $r \leq 5$.*
2. *The array code $\mathcal{C}(k, 6, m)$ is MDS for $m \neq 3, 5, 13$.*
3. *The array code $\mathcal{C}(k, 7, m)$ is MDS for $m > 13$.*
4. *The array code $\mathcal{C}(k, 8, m)$ is MDS for $m > 29$.*

Proof. The proof follows the exact steps of the proof of Theorem 2.6 in [18], by showing that any $k \times k$ submatrix $\mathbf{G}_k$ is non-singular over the ring $\mathbb{F}_2[z]/(h(z))$. Then we can prove the results by invoking Theorem 4. □

Now we discuss the existence of BASIC array code with large r. For this purpose, we need to prove that the matrix $\mathbf{G}_k$ consisting of any ℓ rows of the identity matrix and any $k-\ell$ rows of the Vandermonde matrix is non-singular over $\mathbb{F}_2[z]/(h(z))$. The result is summarized in the following theorem.

Theorem 6. *Let m be a prime number such that 2 is primitive in $\mathbb{F}_m$. If the value of $m-1$ is larger than*

$$(\min\{k,r\}-4)(kr+\frac{(\min\{k,r\}-3)(\min\{k,r\}+3\max\{k,r\}+7)}{6}).$$

Then, the proposed BASIC array codes $\mathcal{C}(k,r,m)$ are MDS for $r \geq 9$ and $k \geq 5$.

Proof. See Appendix A.1. □

Table 4.2: The minimum value of m for $k = 20$ and r ranges from 9 to 14.

r	9	10	11	12	13	14
m	1283	1741	2269	2909	3547	4349

According to Theorem 4, the codes $\mathcal{C}(k,r,m)$ are MDS if the determinant of any $k \times k$ sub-matrix $\mathbf{G}_k$ is a unit over the ring $\mathbb{F}_2[z]/(h(z))$. As m is restricted to be a prime such that 2 is primitive in $\mathbb{F}_m$, we have the ring $\mathbb{F}_2[z]/(h(z))$ is actually a field of size 2^{m-1}. Therefore, if the determinant of any $k\times k$ sub-matrix $\mathbf{G}_k$ is a nonzero polynomial with degree less than $m-1$, or a polynomial with degree equal or larger than $m-1$ but is not divisible by $h(z)$, then the codes $\mathcal{C}(k,r,m)$ are MDS. Apparently, the determinant of $\mathbf{G}_k$ is a polynomial with degree depending on k and r. The lower bound

of $m-1$ given in Theorem 6 is a polynomial about k, r of degree 3, which is easy to satisfy and there are infinite many such prime number m under Artin's conjecture [45]. The minimum value of such prime number m is summarized in Table 4.2 for $k = 20$ and r ranges from 9 to 14.

We remark that if we replace the generator matrix of BASIC array code by a $k \times n$ Vandermonde matrix, the resulting BASIC array code can be viewed as a generalization of Reed-Solomon code.

□ **End of chapter.**

Chapter 5

Decoding Method of BASIC Array Codes

In this chapter, I consider the decoding method to recover some erasures. I will first present a general decoding method for BASIC array codes when there are some erasure columns. A fast decoding algorithm is given in the next for BASIC array codes when some information columns fail. Then, I will evaluate the encoding and decoding complexity for BASIC array codes $\mathcal{C}(k, r, m)$ with the proposed fast decoding algorithm for the information erasures, as well as other MDS array codes (such as RDP [28] and EVENODD [27] of $r = 2$, Star [32] and Triple-Star [33] of $r = 3$, (BBV) [18] and Rabin-Like [17] of $r \geq 4$). Finally, I will present an efficient decoding method for some cases of $r = 4$.

5.1 Decoding Method Using Cramer's Rule

Assume that the parameters m, k, r of BASIC array codes satisfy the MDS condition in Theorem 5 or Theorem 6. Let us consider the general case. Suppose that γ information columns $f_{i_1}, f_{i_2}, \ldots, f_{i_\gamma}$ and δ parity columns $f_{j_1}, f_{j_2}, \ldots, f_{j_\delta}$ erased with $0 \leq f_{i_1} < f_{i_2} < \ldots < f_{i_\gamma} \leq k-1$ and $0 \leq f_{j_1} < f_{j_2} < \ldots < f_{j_\delta} \leq r-1$, where $k \geq \gamma \geq 0$, $r \geq \delta \geq 0$ and $\gamma + \delta \leq r$. Let

$$\mathcal{A} := \{0, 1, \ldots, k-1\} \setminus \{f_{i_1}, f_{i_2}, \ldots, f_{i_\gamma}\}$$

be set of the indices of the available information columns, and let

$$\mathcal{B} := \{0, 1, \ldots, r-1\} \setminus \{f_{j_1} < f_{j_2} < \ldots < f_{j_\delta}\}$$

be set of the indices of the available parity columns.

We want to first recover the lost information columns by reading $k-\gamma$ information columns with indices $i_1, i_2, \ldots, i_{k-\gamma} \in \mathcal{A}$, and γ parity columns with indices $\ell_1, \ell_2, \ldots, \ell_\gamma \in \mathcal{B}$, and then recover the failure parity column by multiplying the corresponding encoding vector and the k data polynomials.

Let $p_{\ell_1}(z), p_{\ell_2}(z), \ldots, p_{\ell_\gamma}(z)$ be the polynomials by subtracting the chosen $k-\gamma$ data polynomials $s_{i_1}(z), s_{i_2}(z), \ldots, s_{i_{k-\gamma}}(z)$ from

γ coded polynomials $c_{\ell_1}(z), c_{\ell_2}(z), \ldots, c_{\ell_\gamma}(z)$, i.e.,

$$p_{\ell_h}(z) = c_{\ell_h}(z) + z^{i_1\ell_h} s_{i_1}(z) + z^{i_2\ell_h} s_{i_2}(z) + \ldots + z^{i_{k-\gamma}\ell_h} s_{i_{k-\gamma}}(z),$$

for $h = 1, 2, \ldots, \gamma$. We can obtain the γ information erasures by solving the following system of linear equations

$$\begin{bmatrix} z^{\ell_1 f_{i_1}} & z^{\ell_1 f_{i_2}} & \ldots & z^{\ell_1 f_{i_\gamma}} \\ z^{\ell_2 f_{i_1}} & z^{\ell_2 f_{i_2}} & \ldots & z^{\ell_2 f_{i_\gamma}} \\ \vdots & \vdots & \ddots & \vdots \\ z^{\ell_\gamma f_{i_1}} & z^{\ell_\gamma f_{i_2}} & \ldots & z^{\ell_\gamma f_{i_\gamma}} \end{bmatrix} \begin{bmatrix} s_{f_{i_1}}(z) \\ s_{f_{i_2}}(z) \\ \vdots \\ s_{f_{i_\gamma}}(z) \end{bmatrix} = \begin{bmatrix} p_{\ell_1}(z) \\ p_{\ell_2}(z) \\ \vdots \\ p_{\ell_\gamma}(z) \end{bmatrix}.$$

The determinant of the above matrix to the left is regarded as an element in the ring $\mathcal{R}_m$ and we can find the inverse polynomial of the determinant over the ring $\mathcal{R}_m$. Therefore, we can solve for the γ failure polynomials by Cramer's rule.

In the next section, I will first present a fast decoding algorithm using an LU factorization of Vandermonde matrix when there are some information failures, and then consider some special cases of $r = 4$ for three information columns and one parity column erasures.

5.2 Decoding Algorithm for Information Erasures

In this section, we consider some information erasures. Let m be a positive prime number such that $m \geq \{k, r\}$. Before giving

the decoding algorithm, I first introduce an LU factorization of the Vandermonde matrix. The efficient decoding method can recover up to $\min\{r, k\}$ information erasures. Alternately, this is a method of decoding all information bits from any $k - \ell$ information columns and ℓ parity columns.

5.2.1 Fast Decoding Algorithm by LU Factorization of Vandermonde Matrix

In the following, we give a fast decoding method using an LU factorization of the Vandermonde matrix. Expressing a matrix as a product of a lower triangular matrix L and an upper triangular matrix U is called an *LU factorization*. Let's review some results on an LU factorization of the Vandermonde matrix.

Given a vector $\mathbf{a} = (a_0, a_1, \ldots, a_\nu)$ with $\nu + 1$ components, we define the square Vandermonde matrix

$$\mathbf{V}_\nu = \mathbf{V}_\nu(\mathbf{a}) := \begin{bmatrix} 1 & a_0 & \cdots & a_0^\nu \\ 1 & a_1 & \cdots & a_1^\nu \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_\nu & \cdots & a_\nu^\nu \end{bmatrix} \tag{5.1}$$

with the second column equal to $\mathbf{a}$. Using symmetric functions and linear algebra, the author in [46] proved the result on an LU

factorization of the Vandermonde matrix, and further simplified the L matrix and U matrix into 1-banded matrices.

Theorem 7. *[46] The Vandermonde matrix $\mathbf{V}_\nu$ can be factorized into ν 1-lower banded matrices $L_\nu^{(1)}, L_\nu^{(2)}, \cdots, L_\nu^{(\nu)}$ and ν 1-upper banded matrices $U_\nu^{(1)}, U_\nu^{(2)}, \cdots, U_\nu^{(\nu)}$ such that*

$$\mathbf{V}_\nu = L_\nu^{(1)} L_\nu^{(2)} \cdots L_\nu^{(\nu)} U_\nu^{(\nu)} \cdots U_\nu^{(2)} U_\nu^{(1)}, \tag{5.2}$$

where the i-th row and the j-th column entry $L_\nu^\ell(i,j)$ and $U_\nu^\ell(i,j)$ of the banded matrix are as follows,

$$L_\nu^\ell(i,j) = \begin{cases} 1 & \text{if } j = i, i \le \nu - \ell \text{ or } i = j+1, i \ge \nu - \ell + 1, \\ a_j - a_{\nu-\ell} & \text{if } i = j, i > \nu - \ell, \\ 0 & \text{otherwise}, \end{cases}$$

$$U_\nu^\ell(i,j) = \begin{cases} 1 & \text{if } j = i, \\ a_{i-\nu+\ell} & \text{if } j = i+1, j \ge \nu - \ell + 1, \\ 0 & \text{otherwise}, \end{cases}$$

for $0 \le i, j \le \nu$ and $1 \le \ell \le \nu$.

For example, consider the case $\nu = 3$, the matrix $\mathbf{V}_3$ can be

factorized into three 1-lower banded matrices

$$L_3^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & a_3 - a_2 \end{pmatrix}, L_3^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & a_2 - a_1 & 0 \\ 0 & 0 & 1 & a_3 - a_1 \end{pmatrix},$$

$$L_3^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & a_1 - a_0 & 0 & 0 \\ 0 & 1 & a_2 - a_0 & 0 \\ 0 & 0 & 1 & a_3 - a_0 \end{pmatrix},$$

and three 1-upper banded matrices

$$U_3^{(3)} = \begin{pmatrix} 1 & a_0 & 0 & 0 \\ 0 & 1 & a_1 & 0 \\ 0 & 0 & 1 & a_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}, U_3^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & a_0 & 0 \\ 0 & 0 & 1 & a_1 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$U_3^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

A proof of Theorem 7 can be found in [46].

Given a $(\nu + 1) \times (\nu + 1)$ linear system in Vandermonde matrix form $\mathbf{V}_\nu \mathbf{x} = \mathbf{b}$, where $\mathbf{x} = (x_0, x_1, \ldots, x_\nu)^t$ and $\mathbf{b} =$

$(b_0, b_1, \ldots, b_\nu)^t$ is both a column vector of length $\nu + 1$. We want to solve the equation for $\mathbf{x}$ given $\mathbf{V}_\nu$ and $\mathbf{b}$. We propose a fast decoding method to solve the $\nu + 1$ linear equations $\mathbf{V}_\nu \mathbf{x} = \mathbf{b}$ using the 1-banded LU factorization of Vandermonde matrix, which is stated in Algorithm 1.

Algorithm 1 Solving $\mathbf{V}_\nu \mathbf{x} = \mathbf{b}$

Input: Column vector $\mathbf{b}$ and Vandermonde matrix $\mathbf{V}_\nu$ that is invertible.

1: Let $\mathbf{y}_\ell$ be column vector of length $\nu + 1$, $\ell = 1, 2, \ldots, \nu + 1$. Let $\mathbf{y}_1 = \mathbf{b}$.

2: **for** each $\ell = 1, 2, \ldots, \nu$ **do**

3: Solve the equation $L_\nu^{(\ell)} \mathbf{y}_{\ell+1} = \mathbf{y}_\ell$ for $\mathbf{y}_{\ell+1}$

4: **for** each $\ell = \nu, \nu - 1, \ldots, 1$ **do**

5: Solve the equation $U_\nu^{(\ell)} \mathbf{y}_\ell = \mathbf{y}_{\ell+1}$ for $\mathbf{y}_\ell$

Output: $\mathbf{y}_1$

Note that in the above Algorithm 1, we are dealing with 1-banded triangular matrices which can be solved directly by *forward or backward substitution* without using the Gaussian elimination process. From Theorem 7, we can see that in the 1-lower banded matrix $L_\nu^{(\ell)}$, there are ℓ rows that have 2 non-zero entries (one is 1 and the other is $a_i - a_{\nu-\ell}$ for $\nu - \ell \leq j \leq \nu$) and the other $\nu - \ell + 1$ rows have one non-zero entry. Therefore, computing the values for $\mathbf{y}_{\ell+1}$ from $L_\nu^{(\ell)} \mathbf{y}_{\ell+1} = \mathbf{y}_\ell$ takes ℓ multiplications and ℓ additions. Similarly in the 1-upper banded matrix $U_\nu^{(\ell)}$, there are

ℓ rows that have 2 non-zero entries (one is 1 and the other is a_j for $0 \leq j \leq \nu-1$) and the other $\nu-\ell+1$ rows have one non-zero entry, and the diagonal entry of $U_\nu^{(\ell)}$ is 1. We can count that computing the equation $U_\nu^{(\ell)}\mathbf{y}_\ell = \mathbf{y}_{\ell+1}$ for $\mathbf{y}_\ell$ takes ℓ multiplications and ℓ additions. Therefore, solving the $\nu+1$ linear equations $\mathbf{V}_\nu\mathbf{x} = \mathbf{b}$ by employing Algorithm 1 takes $\nu(\nu+1)$ multiplications and $\nu(\nu+1)$ additions.

In the following, we want to evaluate the decoding complexity of BASIC array codes, using the above Algorithm 1. We need the following lemma about how to compute the data packet $s(z)$ from $(1+z^b)s(z) = c(z)$ in $\mathcal{R}_m$ for $s(z), c(z) \in \mathcal{C}_m$.

Lemma 8. *Given the equation* $(1+z^b)s(z) = c(z)$, *where* b *is a positive integer such that* $(b,m) = 1$ *and* $s(z), c(z) \in \mathcal{C}_m$, *we can represent a coefficient* s_{m-b} *of* $s(z)$ *as*

$$s_{m-b} = c_b + c_{3b} + c_{5b} + \cdots + c_{(m-2)b},$$

where $s(z) = \sum_{i=0}^{m-1} s_i z^i$ *and* $c(z) = \sum_{i=0}^{m-1} c_i z^i$.

Proof. We can check that in the ring $\mathcal{R}_m$,

$$\begin{aligned}
&c_b + c_{3b} + \cdots + c_{(m-2)b} + s_{m-b} \\
=&(s_0 + s_b) + (s_{2b} + s_{3b}) + \cdots + (s_{(m-3)b} + s_{(m-2)b}) + s_{m-b} \\
=&s_0 + s_b + s_{2b} + \cdots + s_{(m-2)b} + s_{(m-1)b} \\
=&s_0 + s_1 + s_2 + \cdots + s_{m-2} + s_{m-1} \\
=&0.
\end{aligned}$$

In the equations above, the indices are taken modulo m. The second last equality follows from the fact that $\ell b \neq 0 \mod m$ for $(b, m) = 1$ and $1 \leq \ell \leq m-1$. □

The other coefficients of $s(z)$ can be computed recursively by

$$c_{m-b\ell} = s_{m-b\ell} + s_{m-b\ell-b}$$

for $\ell = 1, 2, \ldots, m-1$. Thus, there are $\frac{3(m-1)}{2}$ XORs involved in the solving $s(z)$ from $(1 + z^b)s(z) = c(z)$.

Theorem 9. *In BASIC codes with $\nu + 1$ data polynomials, let a $(\nu+1) \times (\nu+1)$ submatrix of the generator matrix be*

$$\begin{bmatrix} 1 & z^{i_0} & \cdots & z^{i_0\nu} \\ 1 & z^{i_1} & \cdots & z^{i_1\nu} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z^{i_\nu} & \cdots & z^{i_\nu\nu} \end{bmatrix}, \tag{5.3}$$

where $i_0, i_1, \ldots, i_\nu$ are distinct positive integer numbers that are no larger than the prime number m. If we use Algorithm 1 to decode the linear systems with the encoding matrix in (5.3)*, the decoding complexity is at most $\frac{7}{4}\nu(\nu+1)m$.*

Proof. See Appendix A.2. □

Note that the proposed Algorithm 1 can also be employed in the decoding process of BBV code. In the following, we discuss two basic operations that are involved in the decoding process of BBV code, when we use Algorithm 1. The first one is the multiplication $za(z)$ and the last one is to solve $a(z)$ from the equation $(1+z^b)a(z) = c(z)$, where $a(z)$ and $c(z)$ are two polynomials in the ring $\mathbb{F}_2[z]/h(z)$.

For a polynomial $a(z) = \sum_{i=0}^{m-2} a_i z^i$ in the ring $\mathbb{F}_2[z]/h(z)$, the multiplication $za(z)$ takes $m-1$ XORs, as we have

$$\begin{aligned} za(z) &= a_0 z + a_1 z^2 + \cdots + a_{m-3} z^{m-2} + a_{m-2} z^{m-1} \mod h(z) \\ &\equiv a_{m-2} + (a_0 + a_{m-2})z + \cdots + (a_{m-3} + a_{m-2}) z^{m-2}. \end{aligned}$$

Let $a(z) = \sum_{i=0}^{m-2} a_i z^i$ and $c(z) = \sum_{i=0}^{m-2} c_i z^i$ be two polynomials in $\mathbb{F}_2[z]/h(z)$ such that $(1+z^b)a(z) = c(z)$. After expanding the

parentheses, we have

$$
\begin{aligned}
c_0 &= a_0 + a_{m-b} + a_{m-b-1} \\
c_1 &= a_1 + a_{m-b+1} + a_{m-b-1} \\
&\vdots \\
c_{b-2} &= a_{b-2} + a_{m-2} + a_{m-b-1} \\
c_{b-1} &= a_{b-1} + a_{m-b-1} \\
c_b &= a_b + a_0 + a_{m-b-1} \\
c_{b+1} &= a_{b+1} + a_1 + a_{m-b-1} \\
&\vdots \\
c_{m-2} &= a_{m-2} + a_{m-b-2} + a_{m-b-1}.
\end{aligned}
$$

Table 5.1: Number of XORs of two basic operations for $\mathcal{C}(k, r, m)$ and BBV code.

Basic operation	$\mathcal{C}(k, r, m)$	BBV code
$zs(z)$	0	$m-1$
$\frac{s(z)}{1+z^b}$	$\frac{3(m-1)}{2}$	$2m-5$

We can decode the polynomial $a(z)$ by first computing $c_i + c_{i+b}$ for $i = 0, 1, \ldots, m-2$, and then using the result of Lemma 8. The computation of solving $a(z)$ from the equation $(1+z^b)a(z) = c(z)$ in the ring $\mathbb{F}_2[z]/h(z)$ is $2m-5$ XORs. For the two basic operations

$zs(z)$ and $\frac{s(z)}{1+z^b}$, Table 5.1 summarizes the number of XORs involved for $\mathcal{C}(k, r, m)$ and BBV code. By the same argument of Theorem 5.3, we can count that the decoding complexity is at most $\frac{5m-9}{2}\nu(\nu+1)$ XORs, if we use Algorithm 1 to decode the linear systems with the encoding matrix in (5.3) over the ring $\mathbb{F}_2[z]/h(z)$.

5.2.2 Computational Complexity

In the following, we evaluate the encoding/decoding complexities for the proposed $\mathcal{C}(k, r, m)$ and other existing MDS array codes, such as RDP and EVENODD of $r = 2$, Star and Triple-Star of $r = 3$, BBV and Rabin-Like of $r \geq 4$. Here the encoding/decoding complexities are defined as the number of XORs involved in the encoding/decoding processes of the codes. We determine the *normalized encoding complexity* as the ratio of the encoding complexity to the number of information bits, and *normalized decoding complexity* as the ratio of the decoding complexity to the number of information bits.

Encoding Complexity

In the $m-1 \times n$ array of $\mathcal{C}(k, r, m)$, there are k information columns, and $k-1$ XORs are required to reduce the k information

bits per row to the row parity column. The row parity thus requires $(k-1)(m-1)$ XORs. Computing the parity-check bit for information column 1 to $k-1$ takes $(k-1)(m-2)$ XORs. Each diagonal column contains a total of $m-1$ diagonal parity bits, requiring $k-1$ XORs to reduce one diagonal parity bit. Therefore, one diagonal parity column requires $(k-1)(m-1)$ XORs. The total number of XORs required for construction r parities are $(k-1)(m-2)+((k-1)(m-1))r$, and the normalized encoding complexity is

$$\frac{(k-1)(m-2)+((k-1)(m-1))r}{k(m-1)}.$$

In the $(m-1)\times(m+r)$ BBV codes, $r(m-1)$ redundant bits stored in r parity columns are calculated from the $m(m-1)$ information bits. The computation of computing the $m-1$ bits stored in the first parity column is $(m-1)^2$ XORs, and computing the $m-1$ redundant bits for each diagonal parity column takes $(m-1)^2+m-2$ XORs. Therefore, the encoding complexity of BBV code is $rm^2-rm-r-m+2$ and the normalized encoding complexity is

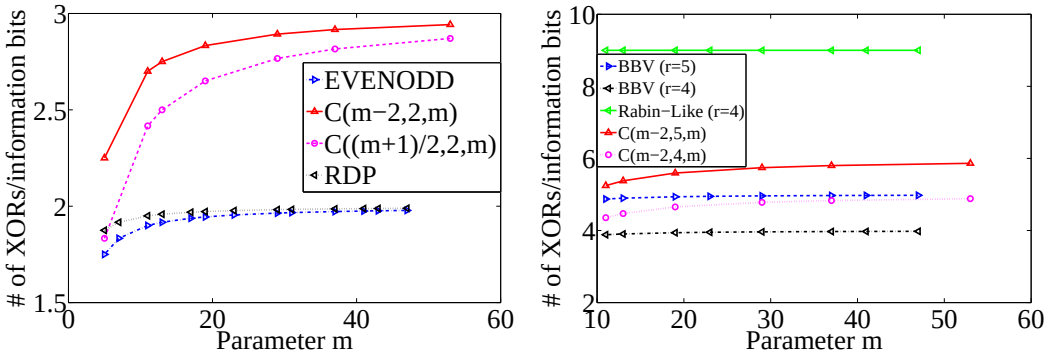
$$\frac{rm^2-rm-r-m+2}{(m-1)m}.$$

For the $(m-1)\times(k+4)$ Rabin-Like codes, the encoding complexity

is $9(m-1)k$ [17]. The normalized encoding complexity of RDP and EVENODD are $2-\frac{2}{m-1}$ and $2-\frac{1}{(m-1)}$ [28] respectively.

Table 5.2: Normalized encoding complexity of MDS array codes.

MDS array codes	Normalized encoding complexity
$\mathcal{C}(k,r,m)$	$\frac{(k-1)(m-2)+((k-1)(m-1))r}{k(m-1)}$
BBV	$\frac{rm^2-rm-r-m+2}{(m-1)m}$
Rabin-Like $(r=4)$	$9(m-1)k$
RDP $(r=2)$	$2-\frac{2}{m-1}$
EVENODD $(r=2)$	$2-\frac{1}{(m-1)}$



(a) $r=2$. (b) $r\geq 4$.

Figure 5.1: The normalized encoding complexity.

The normalized encoding complexities of $\mathcal{C}(k,r,m)$, RDP, EVENODD, BBV, Rabin-Like and CRS are summarized in Table 5.2 and Figure 5.1 for some fixed values of m. The results show that the normalized encoding complexity of $\mathcal{C}(k,2,m)$ is larger than that

of RDP and EVENODD. The reason is that we need to compute the parity-check bit for information columns from 1 to $k-1$. When $r \geq 4$, the encoding complexity of the BBV code and $\mathcal{C}(k,2,m)$ is almost the same, where both of them are smaller than that of Rabin-Like code.

Decoding Complexity

Without loss of generality, we assume that $k \geq r$ and suppose there are r information erasures, we want to decode the k data packets by reading the other $k-r$ information columns and the first r parity columns.

Theorem 10. *In BASIC array code $\mathcal{C}(k,r,m)$, if there are r information erasures and we employ Algorithm 1 to decode the information bits of the r information erasures by reading the bits in the remaining k columns, the decoding complexity is at most*

$$(k-r)(m-1)r + r(m-2) + \frac{7}{4}r(r-1)m.$$

Proof. First, we subtract the $(k-r)(m-1)$ information bits in the information column from the redundant bits in r parity columns, which takes $(k-r)(m-1)r$ XORs. Then, we add the parity-check bit for the r parity columns to formulate r coded polynomials, which

takes $r(m-2)$ XORs. We can decode the r erasure data polynomials by using Algorithm 1 to solve a $r \times r$ linear system, which takes at most $\frac{7}{4}r(r-1)m$ XORs by Theorem 9. Therefore, the decoding complexity of $\mathcal{C}(k,r,m)$ is at most

$$(k-r)(m-1)r + r(m-2) + \frac{7}{4}r(r-1)m.$$

□

It is shown in [47] that the computational complexity of correcting r information erasures for BBV codes is

$$rm^2 + 3.5r^2m - 2.5rm - 2r^2 + 2r.$$

However, we can first subtract the information bits from the redundant bits in r parity columns, then solve the resulting $r \times r$ linear system using Algorithm 1, and the decoding complexity of BBV codes can be reduced to

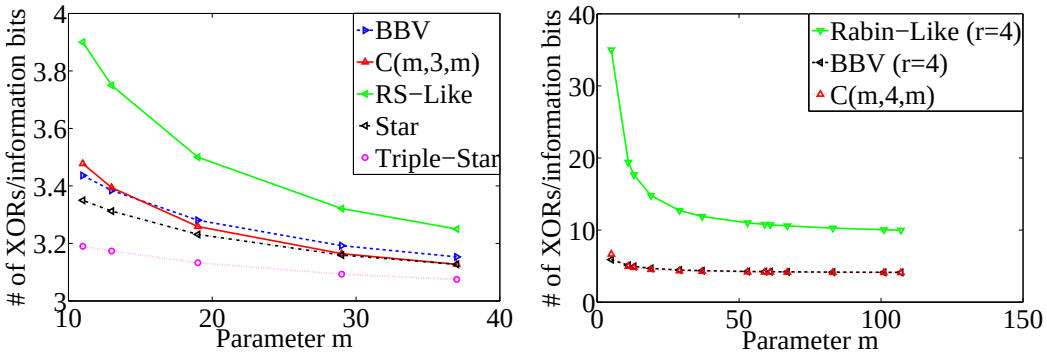
$$r(m-r)(m-1) + \frac{5m-9}{2}(r^2-r).$$

So we will evaluate the decoding complexity of BBV codes by employing Algorithm 1 in the following comparison. Table 5.3 summarizes the decoding complexity of r information erasures for $\mathcal{C}(k,r,m)$ and BBV codes, $r = 2,3,4,5,6,7,8$.

Table 5.3: Decoding complexity of BASIC array codes and BBV codes.

♯ of information erasures	$\mathcal{C}(k,r,m)$	BBV codes
2	$2km-2k+1.5m$	$2m^2-m-5$
3	$3km-3k+4.5m+3$	$3m^2+3m-18$
4	$4km-4k+9m+8$	$4m^2+10m-32$
5	$5km-5k+15m+15$	$5m^2+20m-65$
6	$6km-6k+22.5m+24$	$6m^2+33m-99$
7	$7km-7k+31.5m+35$	$7m^2+49m-140$
8	$8km-8k+42m+48$	$8m^2+68m-188$

For the case of two information columns failures, the decoding complexity of $\mathcal{C}(k,r,m)$ is $2km-2k+1.5m$, while the number of XORs of computing the lost two information columns in RDP [28] is $2(m-1)(m-2)$ and the reconstruction algorithm described in the EVENODD paper [27] requires more XORs. The decoding algorithms for two information erasures in the generalized EVENODD, such as STAR [32], Triple-Star [33] are the same as that in EVENODD and the repairing algorithm of extension RDP [48] is the same as RDP. For a fair comparison, we let $k=m-1$ in $\mathcal{C}(k,r,m)$. We can thus have the normalized decoding complexity of the proposed $\mathcal{C}(k,r,m)$ and RDP are $\frac{2m^2-2.5m+2}{m^2-2m+1}$ and $\frac{2(m-2)}{m-1}$



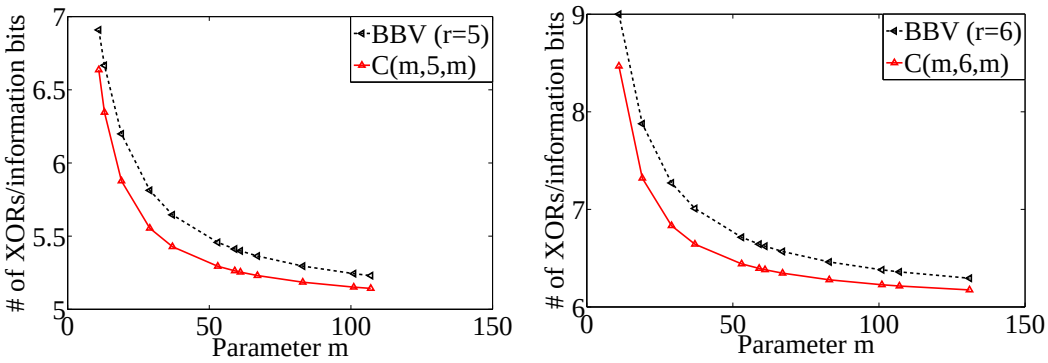
(a) $r = 3$. (b) $r = 4$.

Figure 5.2: The normalized decoding complexity of $r = 3, 4$.

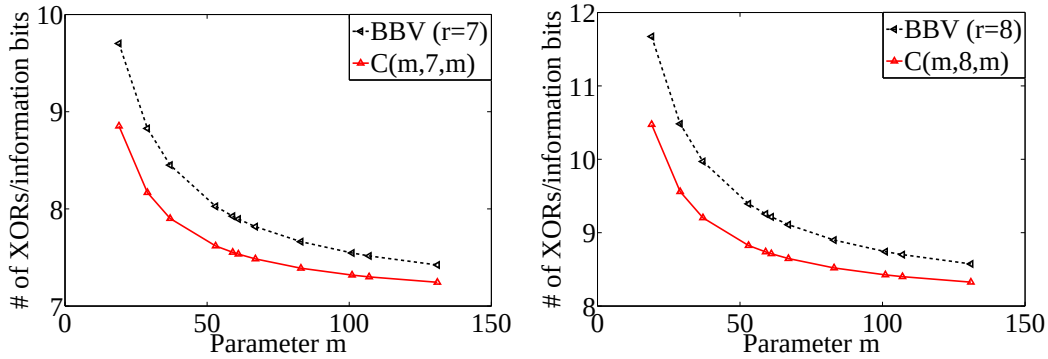
respectively, which are almost the same.

The normalized decoding complexity of the MDS array codes shows in Figure 5.2 and Figure 5.3. Among MDS codes supporting three disk failures, Triple-Star code has the lowest decoding complexity, and RS-Like code has the highest decoding complexity. Compared to BBV codes, the proposed array codes is more efficient in decoding when the parameter r is large, from Figure 5.2(a).

In the proposed codes $\mathcal{C}(k, r, m)$ with $r \geq 4$, we let $k = m$ for fair comparison. For 4 simultaneous information failures, the decoding complexity of Rabin-Like code, BBV code and the proposed $\mathcal{C}(m, 4, m)$ are $9m^2 + 86m$ [17], $4m^2 + 62m - 28$ [47] and $4m^2 + 16m + 12$ respectively. When $r = 4$, the proposed array codes have slightly less decoding complexity of BBV codes, and Rabin-Like code has lowest performance in terms of decoding complexity



(a) $r = 5$. (b) $r = 6$.



(c) $r = 7$. (d) $r = 8$.

Figure 5.3: The normalized decoding complexity of $r = 5, 6, 7, 8$.

among three MDS array codes. From Figure 5.3(a) to Figure 5.3(d), we have that the proposed array codes have an advantage in the decoding process compared to BBV codes, and that this advantage increases when r is large. For some cases of $r = 8$, the decoding complexity of $\mathcal{C}(k, r, m)$ has roughly 9% reduction of BBV codes. Therefore, we can conclude that if the number of parity columns r is large, and if we want fast decoding, it is better to use the proposed $\mathcal{C}(k, r, m)$.

When $r \geq 4$, the decoding complexity of the proposed array codes is less than that of BBV code. The essential reason is as follows. Recall that BBV code is constructed over the ring $\mathbb{F}_2[z]/h(z)$, while $\mathcal{C}(k, r, m)$ is $\mathcal{R}_m$. First, in $\mathcal{R}_m$, multiplication by z can be interpreted as a cyclic shift, there is no XOR operation in the multiplication. While in $\mathbb{F}_2[z]/h(z)$, the multiplication of z and a polynomial takes $m-2$ XORs. Second, in $\mathcal{R}_m$, there are at most $\frac{3(m-1)}{2}$ XORs involved in solving $s(z)$ from $(1+z^b)s(z) = c(z)$. While in $\mathbb{F}_2[z]/h(z)$, solve the equation $(1+z^b)s(z) = c(z)$ takes $2m-5$ XORs.

In conclusion, Triple-Star code has the most efficient decoding algorithm among the compared 3-erasure MDS array codes. When $r \geq 4$, the proposed BASIC array codes $\mathcal{C}(k, r, m)$ has the lowest decoding complexity in the compared MDS array codes.

5.2.3 Other Efficient Decoding Method

I want to mention that we can also employ the Lagrange interpolation polynomial method [49] to solve the Vandermonde system of BASIC array code. Both the Lagrange interpolation polynomial method and the LU factorization method can reduce the decoding complexity, compared to the traditional Cramer's rule

method. However, both the two methods are applicable only for the information erasures. The differences of two methods are as follows. First, the decoding complexity of BASIC array code with Lagrange interpolation polynomial method is larger than that with LU factorization method. Second, we can parallelly solve the Vandermonde system with Lagrange interpolation polynomial method, while we can not solve it parallelly for the LU factorization method. I refer the readers to [49] on Lagrange interpolation polynomial for more details.

5.3 Decoding Method for Four Parity Columns

In this section, we focus on $r = 4$ and consider decoding method to recover one parity column and three information columns. We assume the parameter m to be an odd number and $m \geq 5$. Suppose that three information columns f_1, f_2, f_3 and one parity column j are erased. We want to recover the lost data bits in columns f_1, f_2, f_3 by reading information columns i, for $i \in \mathcal{A}$, and three parity columns $a, b, c \in \mathcal{B}$.

First, we add the parity-check bit for each of the $k - 3$ information columns and 3 parity columns, which takes $k(m - 2)$

XORs. Then, we subtract the known values of $s_i(z)$, for $i \in \mathcal{A}$, from $c_\ell(z)$, for $\ell = a, b, c$.

If the failed parity column is the first parity column or the last parity column, then can recover the the information failures by first subtracting the remaining $k-3$ information columns from the other three parity columns, and then solving the 3×3 Vandermonde system by employing the LU method, which takes $10.5m$ XORs by Theorem 9. At last, we re-encode the failed parity column. The decoding complexity is upper bounded by

$$3m^2 + 1.5m + k(m-2) + (k-1)(m-1).$$

In the following, we consider the case that the failed parity column is the second parity column or the third parity column.

After subtracting the known $k-3$ information columns from three parity columns, we are sufficient to solve the following 3×3 linear equations

$$\begin{bmatrix} 1 & 1 & 1 \\ z^{a\alpha} & z^{b\alpha} & z^{c\alpha} \\ z^{3a} & z^{3b} & z^{3c} \end{bmatrix} \begin{bmatrix} s_a(z) \\ s_b(z) \\ s_c(z) \end{bmatrix} = \begin{bmatrix} p_0(z) \\ p_\alpha(z) \\ p_3(z) \end{bmatrix},$$

where $0 \le a < b < c \le k-1$ and $\alpha = 1$ or 2. The determinant of

the above matrix is

$$(z^a + z^b + z^c)(z^a + z^b)(z^a + z^c)(z^b + z^c)$$

when $\alpha = 1$, and

$$(z^{a+b} + z^{a+c} + z^{b+c})(z^a + z^b)(z^a + z^c)(z^b + z^c)$$

when $\alpha = 2$.

By Cramer's rule, we can compute $s_a(z)$ by

$$\frac{p_0(z)(z^{b\alpha+3c} + z^{c\alpha+3b}) + p_\alpha(z)(z^{3c} + z^{3b}) + p_3(z)(z^{b\alpha} + z^{c\alpha})}{f(z)(z^a + z^b)(z^a + z^c)(z^b + z^c)},$$

where $f(z) = z^a + z^b + z^c$ when $\alpha = 1$ and $f(z) = z^{a+b} + z^{a+c} + z^{b+c}$ when $\alpha = 2$. So, the trick of solving $s_a(z)$ is how to compute the inverse of $f(z)$.

In general, we can compute the inverse of $f(z)$ using the extended Euclidean algorithm, as $f(z)$ and $1+z^m$ is relatively prime. In the next, we show an efficient method of computing the inverse of $f(z)$ for some cases.

5.3.1 Complexity Reduction on Some Special Cases

For the three information failures f_1, f_2, f_3, let $a = f_2 - f_1$, $b = f_3 - f_2$ and we can write $b \equiv ia \mod m$. We will give an

efficient decoding method for $i = 2, 3, 4, 5$. We first consider the case of $i = 2$.

Case of $i = 2$. The following lemma gives an efficient method about how to compute the division $\frac{c(z)}{1+z^a+z^{2a}}$ over the ring $\mathcal{R}_m$, for $c(z) \in \mathcal{C}_m$.

Lemma 11. *Given the equation* $(1 + z^a + z^{2a})s(z) = c(z)$*, where* $0 < a \leq m - 1$ *and* $s(z), c(z) \in \mathcal{C}_m$*, we have*

$$(1 + z^a)c(z) = (1 + z^{3a})s(z).$$

Proof. We have in the ring $\mathcal{R}_m$,

$$\begin{aligned}(1 + z^a)c(z) &= (1 + z^a)(1 + z^a + z^{2a})s(z) \\ &= (1 + z^{3a})s(z).\end{aligned}$$

□

By combining Lemma 8 and Lemma 11, we can count that the computation of solving $s(z)$ takes $2.5m - 1.5$ XORs at most.

Case of $i = 3$. The following lemma shows how to transform the equation $(1 + z^a + z^{3a})s(z) = c(z)$ to the form in Lemma 8.

Lemma 12. *Given the equation* $(1 + z^a + z^{3a})s(z) = c(z)$*, where* $0 < a \leq m - 1$ *and* $s(z), c(z) \in \mathcal{C}_m$*, we can have*

$$(1 + z^a + z^{2a} + z^{4a})c(z) = (1 + z^{7a})s(z).$$

Lemma 12 can be proved by multiplying both sides of the equation $(1 + z^a + z^{3a})s(z) = c(z)$ by $(1 + z^a + z^{2a} + z^{4a})$.

We can decode the polynomial $s(z)$ from $(1 + z^a + z^{3a})s(z) = c(z)$ by combining Lemma 8 and Lemma 12.

For the cases of $i = 4$ and $i = 5$, we summarize the main results as follows.

Lemma 13. *Given the equation* $(1 + z^a + z^{4a})s(z) = c(z)$, *where* $0 < a \leq m - 1$ *and* $s(z), c(z) \in \mathcal{C}_m$, *we can have*

$$(1 + z^a + z^{2a} + z^{3a} + z^{5a} + z^{7a} + z^{8a} + z^{11a})c(z)$$
$$= (1 + z^{15a})s(z).$$

Lemma 13 can be proved by multiplying both sides of the equation $(1 + z^a + z^{4a})s(z) = c(z)$ by $(1 + z^a + z^{2a} + z^{3a} + z^{5a} + z^{7a} + z^{8a} + z^{11a})$.

Lemma 14. *Given the equation* $(1 + z^a + z^{5a})s(z) = c(z)$, *where* $0 < a \leq m - 1$ *and* $s(z), c(z) \in \mathcal{C}_m$, *we can have*

$$(1 + z^a + z^{2a} + z^{3a} + z^{4a} + z^{6a} + z^{8a} + z^{11a} + z^{12a} + z^{16a})c(z)$$
$$= (1 + z^{21a})s(z).$$

Lemma 14 can be proved by multiplying both sides of the equation $(1 + z^a + z^{5a})s(z) = c(z)$ by $(1 + z^a + z^{2a} + z^{3a} + z^{4a} + z^{6a} + z^{8a} + z^{11a} + z^{12a} + z^{16a})$.

□ **End of chapter.**

Chapter 6

Functional Repair BASIC RGC

In the rest of this thesis, we consider *BASIC regenerating codes*, which is defined as a class of BASIC codes such that the parameters n, k, d, α, β achieve the optimal trade-off curve in (2.1) asymptotically. We present a general construction of functional repair BASIC regenerating codes in this chapter. Exact repair BASIC regenerating codes will be given in the next chapter. First, we review the general construction of functional repair regenerating codes over a finite field.

6.1 Functional Repair RGC over Finite Field

Let B be the total file size measured in terms of the number of symbols in a finite field $\mathbb{F}_{2^w}$ of size 2^w. The data file is divided

into chunks of B *data symbols* and stored across n nodes with each node stores α *coded symbols* in $\mathbb{F}_{2^w}$, such that the data file can be recovered by connecting to any k nodes. Each coded symbol is a linear combination of the B data symbols in $\mathbb{F}_{2^w}$. The coefficients of the linear combination form the *global encoding vector* of the corresponding coded symbol.

In a repair process, a new node is created and replaces the failure node by connecting to an arbitrary set of d of the remaining nodes. The storage nodes which participate in the repair process are also called the *helpers*. Each of the helper nodes transmits β symbols to the new node, and each of these symbols is a linear combination of the α symbols stored in the node. The coefficients of the linear combination are called *local encoding coefficients* of the corresponding symbol downloaded to repair the failure. The new node will generate α new symbols, with each new symbol created by doing a linear combination of the receiving $d\beta$ symbols. The process is termed as the *repair process* and the total amount of $d\beta$ of data downloaded in a repair process is called *repair bandwidth*. Note that the α symbols stored in the new node need not be the same as the failures, as long as the property that any k nodes are sufficient in decoding the original file is maintained. We call this property the

(n, k) *recovery property*.

A major result in the field of RGC is that the parameters of a regenerating code must necessarily satisfy the inequality in (2.1). The general construction of functional repair RGC over a finite field that achieves the optimal trade-off in (2.1) is presented in [6] by Wu. It is shown that the functional repair RGC can be constructed over a finite field whose size is independent of how many failures/repairs can happen. The proof is established in [6] by first formulating the existence condition as a product of multivariate polynomials, then showing each polynomial is non-zero, and finally applying the Schwartz-Zippel lemma (see e.g. [50, p. 224])

Lemma 15 (Schwartz-Zippel)**.** *Let $\mathbb{F}$ be a finite field and $\mathcal{S}$ be a subset of elements in $\mathbb{F}$. Let f be a non-zero multivariate polynomial in $\mathbb{F}[X_1, X_2, \ldots, X_N]$ of degree e. Then the polynomial f has at most $e|\mathcal{S}|^{N-1}$ roots in $\mathcal{S}^N$.*

The key concept used in the repair process is the *information flow graph*, which represents the evolution of information flow as nodes join and leave. To ensure the (n, k) recovery property after each repair, the author in [6] characterized a *capacitated data collector* with a length-n *characteristic vector* $\mathbf{h}$ that indicates the allowed access capacities from the storage nodes, here data collector

corresponding to one request to reconstruct the original data. The entry of $\mathbf{h}$ refers to the information that the data collector can get from the storage node.

Given the values of system parameters n, k, d, α and β, let the size of data file, B, be

$$B := \sum_{i=1}^{k} \min\{(d-i+1)\beta, \alpha\}. \tag{6.1}$$

For $i = 1, 2, \ldots, k$, let s_i be the i-th term in the above summation in (6.1),

$$s_i := \min\{(d-i+1)\beta, \alpha\},$$

and for $i = k+1, k+2, \ldots, n$, let $s_i = 0$. Define $\mathcal{H}$ as the set of vectors of length n, whose components are non-negative integers, which are majorized by the vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$. In other words, if we sort the components of a vector $\mathbf{h} \in \mathbb{Z}_+^n$ in non-increasing order as $h_{[1]} \geq h_{[2]} \geq \cdots \geq h_{[n]}$, then $\mathbf{h}$ is in $\mathcal{H}$ if and only if

$$\sum_{i=1}^{\mu} h_{[i]} \begin{cases} \leq \sum_{i=1}^{\mu} s_i & \text{for } \mu = 1, 2, \ldots, n-1, \\ = B & \text{for } \mu = n. \end{cases}$$

We refer the readers to [51] for more details on majorization theory.

The existence proof assumes that any data collector with characteristic $\mathbf{h}' \in \mathcal{H}$ can recover the original file before a failure,

and shows that the (n, k) recovery property can be satisfied after the repair. The main result in [6] is summarized in the following theorem.

Theorem 16. *Let $\mathbb{F}_{2^w}$ be a finite field whose size is greater than*

$$B \cdot \max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}. \tag{6.2}$$

Then, there exists a functional repair regenerating code defined in $\mathbb{F}_{2^w}$ that achieves the optimal trade-off point in (2.1).

6.2 Functional Repair BASIC Regenerating Codes

In this section, we illustrate how to adapt the results in [6] to functional repair BASIC regenerating code.

We assume that a data file contains $B(m-1)$ bits, for the ease of presentation. In the encoding process, the data file is divided into B groups. Each group of $m-1$ bits is encoded to a codeword of the binary parity-check code $\mathcal{C}_m$. We let $s_1(z)$, $s_2(z), \ldots, s_B(z) \in \mathcal{C}_m$ be the resulting codewords. We call these B codewords the *data packets* or *source packets*.

We store α coded packets in each node. Each coded packet is an $\mathcal{R}_m$-linear combination of the B data packets, with the corresponding global encoding vector. When we choose the global

encoding vectors, the (n, k) recovery property should be satisfied. When a node fails, we connect to an arbitrary set of d helper nodes of the remaining nodes and download β coded packets from each helper node.

The repair process of functional repair BASIC RGC, which is different from functional repair RGC over a finite field, is stated as follows: each of the d helper nodes transmit β packets to the new node, and each of these packets is an $\mathcal{R}_m$-linear combination of the α encoded packets in the memory. The local encoding coefficients are polynomials in $\mathcal{R}_m$. Upon receiving the $d\beta$ packets from the helpers, the new node computes and stores α packets. Each packet stored in the new node is an $\mathcal{R}_m$-linear combination of the $d\beta$ received packets, with coefficients being polynomials in $\mathcal{R}_m$. The computations required during the repair process are just cyclic shifts and binary additions. The global encoding vectors of the new packets are also computed and stored. We want to show that by choosing the values of local encoding coefficients to be polynomials in $\mathcal{R}_m$, we can maintain the (n, k) recovery property.

It can be proved by modifying the argument in [6] on the existence of RGC over a finite field, and invoking a Schwartz-Zippel lemma over a specific ring $\mathcal{C}_m$.

Let $g(X_1, X_2, \ldots, X_N)$ be a non-zero multivariate polynomial in $\mathcal{R}_m[X_1, X_2, \ldots, X_N]$, with coefficients in the ring $\mathcal{R}_m$. For $\ell \in \{1, 2, \ldots, N\}$, let $r_\ell \in \mathcal{R}_m$, we define the N-tuple $(r_1, r_2, \ldots, r_N)$ as $\mathcal{C}_m$-*root* of the polynomial $g(X_1, X_2, \ldots, X_N)$, if $g(r_1, r_2, \ldots, r_N)$ in the ring $\mathcal{R}_m$ is not $\mathcal{C}_m$-invertible.

Lemma 17 (Schwartz-Zippel lemma over the ring $\mathcal{C}_m$)**.** *Suppose that $f_1(z), f_2(z), \ldots, f_L(z)$ are the irreducible factors of the check polynomial $h(z)$. Let $\mathcal{S}$ be a subset of $\mathcal{R}_m$ such that the function $\theta_\ell : \mathcal{S} \to \mathbb{F}_2[z]/f_\ell(z)$, defined as*

$$\theta_\ell(a(z)) := a(z) \mod f_\ell(z),$$

is injective $\forall \ell = 1, 2, \ldots, L$, where $a(z)$ can assume any value in $\mathcal{S}$. Then the polynomial $g(X_1, X_2, \ldots, X_N)$ has at most $L \cdot e \cdot |\mathcal{S}|^{N-1}$ $\mathcal{C}_m$-roots in $\mathcal{S}^N$, where e is the degree of the polynomial $g(X_1, X_2, \ldots, X_N)$.

Proof. See Appendix A.3. □

In [6], the existence of RGC over a finite field is proved by showing that we can choose the local encoding coefficient such that a collection of determinants are all evaluated to be non-zero. In the case of BASIC regenerating codes, we want to restrict the local encoding coefficients to be polynomials in $\mathcal{S}$, and the

collection of determinants are evaluated to be non-zero in several finite fields. Note that when we choose the polynomials for the set $\mathcal{S}$, the mapping θ_ℓ defined in Lemma 17 should be injective, $\forall \ell = 1, 2, \ldots, L$. The cardinality of $\mathcal{S}$ thus can not exceed the size of the smallest field in $\{\mathbb{F}_2(z)/f_1(z), \mathbb{F}_2(z)/f_2(z), \cdots, \mathbb{F}_2(z)/f_L(z)\}$, i.e.,

$$|\mathcal{S}| \leq \min(2^{\deg(f_1(z))}, \cdots, 2^{\deg(f_L(z))}) \leq 2^{\frac{m-1}{L}}. \tag{6.3}$$

With these modification, the requirement on the cardinality of $\mathcal{S}$ is stated in the next theorem.

Theorem 18. *Let n, k, d, α and β be fixed system parameters of a distributed storage system. Let m be an odd number, and $f_1(z)f_2(z)\cdots f_L(z)$ be the prime factorization of the check polynomial $h(z)$ over $\mathbb{F}_2$. If we can find a subset $\mathcal{S}$ of $\mathcal{R}_m$ such that (i) the mapping θ_ℓ defined in Lemma 17 is injective, $\forall \ell = 1, 2, \ldots, L$, and (ii) if $|\mathcal{S}|$ is larger than*

$$L \cdot B \cdot \max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}, \tag{6.4}$$

then there exists a functional repair BASIC regenerating code, which supports the file size

$$B = \sum_{i=1}^{k} \min\{(d-i+1)\beta, \alpha\},$$

with local encoding coefficients drawn from the subset $\mathcal{S}$.

Proof. See Appendix A.4. □

Theorem 18 says that when the cardinality of $\mathcal{S}$ is larger than (6.4), the proposed BASIC regenerating codes can achieve all the points on the optimal trade-off curve between storage and repair bandwidth asymptotically. Note that the coding scheme proposed in this thesis has an additional 1 bit per $m-1$ bits, and this leads to a slight increase in storage and repair bandwidth by a factor of $m/(m-1)$, this is what "asymptotically" means. The key difference of BASIC regenerating codes presented in this section and other RGC in the literature is that, the packets in BASIC regenerating codes assume value in $\mathcal{C}_m$, and the local encoding coefficients are polynomials in $\mathcal{S}$.

We may choose the parameter m to be a prime number such that the multiplicative order of 2 mod m is $m-1$. In this case, the polynomial $1+z^m$ is factorized as a product of two irreducible polynomials, namely, $1+z$ and the check polynomial $h(z) = 1 + z + \cdots + z^{m-1}$. Under the Artin's conjecture on primitive roots, there are infinitely many such prime number m [45]. In this case, we can let the set $\mathcal{S}$ to be the polynomials in $\mathcal{R}_m$ with non-zero coefficients less than or equal to $(m-1)/2$, and $|\mathcal{S}| = 2^{m-1}$. We can check that

the function $\theta_1 : \mathcal{S} \to \mathbb{F}_2(z)/h(z)$, defined as

$$\theta_1(a(z)) := a(z) \mod h(z),$$

is injective, for any polynomial $a(z)$ in $\mathcal{S}$. The following Corollary is a direct result of Theorem 18.

Corollary 19. *Let m be a prime number such that the multiplication order of $2 \mod m$ is equal to $m-1$. There exists a functional repair BASIC regenerating code for a file size*

$$\frac{m-1}{m} \sum_{i=1}^{k} \min\{(d-i+1)\beta, \alpha\},$$

if

$$m > \log_2 \left(B \cdot \max \left\{ \binom{n\alpha}{B}, 2|\mathcal{H}| \right\} \right) + 1. \tag{6.5}$$

□ **End of chapter.**

Chapter 7

Exact Repair BASIC RGC

In exact repair regenerating codes, a failed node is replaced by a new node that stores exactly the same data as was stored in the failed node. To construct exact repair RGC is more difficult than to construct functional repair RGC, and all the constructions in [8,19–22] for exact repair RGC have been focused on the MSR point and MBR point. A general explicit construction of exact repair MBR codes for all feasible values of n, k, d and exact repair MSR codes for all $n, k, d \leq 2k - 2$ is firstly presented in [19]. The construction is of a product-matrix nature that is shown to significantly simplify operation of the distributed storage network.

This chapter will starts from the product-matrix construction of exact repair RGC, and then research the conversion of product matrix RGC in [19] to product-matrix BASIC regenerating codes.

7.1 Product-Matrix Regenerating Codes

As the product-matrix construction is based on a finite field, throughout this subsection, we consider all symbols to belong to $\mathbb{F}_{2^w}$ of size 2^w. A regenerating code is represented by the product $\mathbf{\Psi M}$ of an $n \times d$ *encoding matrix* $\mathbf{\Psi}$ and an $d \times \alpha$ *message matrix* $\mathbf{M}$. The entries of $\mathbf{\Psi}$ are elements of the finite field $\mathbb{F}_{2^w}$ and are independent of the message symbols. The message matrix $\mathbf{M}$ is filled by the B message symbols, with some submatrices of $\mathbf{M}$ being symmetric. The i-th row of $\mathbf{\Psi}$ is referred to as the encoding vector $\boldsymbol{\psi}_i$ of node i. For $i = 1, 2, \ldots, n$, node i stores the i-th row of the product $\mathbf{\Psi M}$.

Let $\{\ell_1, \ell_2, \ldots, \ell_k\}$ be the index set of k storage nodes that a data collector connects to. The data collector can thus obtains $k\alpha$ symbols in the product matrix $\mathbf{\Psi}_{DC}\mathbf{M}$, where $\mathbf{\Psi}_{DC}$ is the submatrix of $\mathbf{\Psi}$ consisting of the k rows indexed by $\{\ell_1, \ell_2, \ldots, \ell_k\}$. We let Φ_k be a submatrix of $\mathbf{\Psi}$ consisting by the first k columns. There is a requirement that any k rows of Φ_k are independent when we choose the value of the encoding matrix $\mathbf{\Psi}$, to maintain the (n, k) recovery property.

If we assume node f fails, a new node replacing the failed node connects to an arbitrary subset $\{h_1, \ldots, h_d\}$ of d helper nodes. Each

helper node sends the inner product of the α symbols stored in it with the encoding vector $\boldsymbol{\psi}_f$, to the new node. The new node will thus receive the product matrix $\boldsymbol{\Psi}_{repair}\mathbf{M}\boldsymbol{\psi}_f$, where $\boldsymbol{\Psi}_{repair}$ is the submatrix of $\boldsymbol{\Psi}$ consisting of the d rows $\{h_1, \ldots, h_d\}$. From this it turns out that we can recover the failed symbols exactly if the matrix $\boldsymbol{\Psi}_{repair}$ is invertible and the message matrix $\mathbf{M}$ satisfies some properties.

7.2 BASIC Product-Matrix Regenerating Code

If we replace the symbol of product-matrix RGC over a finite field by a codeword of binary parity-check code $\mathcal{C}_m$, then the corresponding codes are BASIC product-matrix regenerating codes. In this section, we will convert the product-matrix RGC in [19] to BASIC product-matrix regenerating codes.

Let $\mathbf{u} = [s_1(z)\ s_2(z)\ \ldots\ s_B(z)]^T$ be a column vector of length B containing the source packets. Each source packet is a codeword of the binary parity-check code $\mathcal{C}_m$, which contains m bits. The entries of the encoding matrix $\boldsymbol{\Psi}$ are fixed polynomials in $\mathcal{R}_m$ and independent of the source packets. The entries of $\mathbf{M}$ are the source packets in $\mathcal{C}_m$.

7.2.1 BASIC Product-Matrix MSR Code

In the following, we construct the BASIC-PM (product-matrix) MSR code for $d = 2k - 2$. As in [19], the construction can be extended naturally to $d \geq 2k - 2$, but we will only discuss the basic case for

$$d = 2k - 2,\ \alpha = k - 1,\ B = k\alpha = k(k - 1).$$

Divide the data file into B parts, each of $m - 1$ bits, and generate B source packets in $\mathcal{C}_m$. Divide each of the B source packets into two equal groups. For each group, create an $(k - 1) \times (k - 1)$ symmetric matrix by filling the upper-triangular part of the matrix by the $k(k-1)/2$ source packets in the group, and obtain the lower-triangular part by reflection. Let the symmetric matrix obtained from group j be denoted by $\mathbf{S}_j$, for $j = 1, 2$, and let $\mathbf{M}$ be the $d \times (k - 1)$ matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{S}_1 \\ \mathbf{S}_2 \end{bmatrix}.$$

Define the encoding matrix $\mathbf{\Psi}$ to be the $n \times d$ Vandermonde matrix, with the i-th row defined as

$$\boldsymbol{\psi}_i^t := \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & \dots & z^{(d-1)(i-1)} \end{bmatrix}, \tag{7.1}$$

for $i = 1, 2, \dots, n$. The i-th node stores $\alpha = k - 1$ packets in $\boldsymbol{\psi}_i^t \mathbf{M}$.

Let Φ be the $n \times \alpha$ Vandermonde matrix such that the i-th row is

$$\phi_i^t := \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & \dots & z^{(\alpha-1)(i-1)} \end{bmatrix}, \tag{7.2}$$

for $i = 1, 2, \dots, n$, and let Λ be the $n \times n$ diagonal matrix with diagonal elements equal to $1, z^{\alpha}, \dots, z^{\alpha(n-1)}$. We have that $\mathbf{\Psi} = \begin{bmatrix} \Phi & \Lambda\Phi \end{bmatrix}$. There is a requirement when we choose the value of m if we want to maintain the (n, k) recovery property that is both any d rows of $\mathbf{\Psi}$ and any α rows of Φ are linear independent over $\mathcal{R}_m$, i.e., the determinants of any $d \times d$ submatrices of $\mathbf{\Psi}$ and any $\alpha \times \alpha$ submatrices of Φ are $\mathcal{C}_m$-invertible. The requirement can be met by checking the condition of decodability given in Theorem 2 and Corollary 3.

Lemma 20. *Let $\mathbf{\Psi}$ be the encoding matrix, which is composed by the encoding vector given in* (7.1)*. If $n - 1$ is strictly less than all divisors of m which are not equal to 1, then the determinants of any $d \times d$ submatrices of $\mathbf{\Psi}$ and any $\ell \times \ell$ submatrices of Φ are $\mathcal{C}_m$-invertible, for $1 \leq \ell < \alpha$.*

Proof. See Appendix A.5. □

If m is a prime number and $m \geq n$, then the determinant of any $d \times d$ submatrix of $\mathbf{\Psi}$ is $\mathcal{C}_m$-invertible according to the above lemma.

For example of $m = 7$, $n = 6$ and $d = 3$, the determinant of any 3×3 submatrix of $\mathbf{\Psi}$ can be written as $(z^a + z^b)(z^a + z^c)(z^b + z^c)$, where $0 \leq a < b < c \leq 5$. The check polynomial $1 + z + \cdots + z^6$ can be factorized into $f_1(z) = 1 + z + z^3$ and $f_2(z) = 1 + z^2 + z^3$. We can check that the polynomial $1 + z^\ell$ is not divisible by $f_1(z)$ and $f_2(z)$ for $\ell = 1, 2, 3, 4, 5$.

The protocol of repairing a failed node is the same as in [19], except that we are now working over $\mathcal{R}_m$ instead of a finite field. The following two theorems summarize the exact repair and data reconstruction properties of BASIC-PM MSR code.

Theorem 21. *Suppose that the parameter m satisfies the requirement in Lemma 20 and suppose there is a failure node, we can repair the α packets in the failure node by downloading one packet each from any $d = 2k - 2$ of the remaining nodes*

Proof. We assume that the node f fails and let $\boldsymbol{\psi}_f^t$ be the f-th row of $\mathbf{\Psi}$ corresponding to the failure node. So the α packets in the node f are $\boldsymbol{\psi}_f^t \mathbf{M}$. The new node which is created to replace the failure node and connects to any d helper nodes $h_1, h_2, \ldots, h_d$. The helper node h_j computes a packet $\boldsymbol{\psi}_{h_j}^t \mathbf{M} \boldsymbol{\phi}_f$ and sends it to the new node. The new node thus obtains d packets $\mathbf{\Psi}_{repair} \mathbf{M} \boldsymbol{\phi}_f$ from the d helper

nodes, where

$$\mathbf{\Psi}_{repair} = \begin{bmatrix} \boldsymbol{\psi}_{h_1}^t \\ \boldsymbol{\psi}_{h_2}^t \\ \vdots \\ \boldsymbol{\psi}_{h_d}^t \end{bmatrix}.$$

By Lemma 20, the square matrix $\mathbf{\Psi}_{repair}$ is $\mathcal{C}_m$-invertible. Therefore, the new node can computes d packets $\mathbf{M}\phi_f$, i.e., $\mathbf{S}_1\phi_f$ and $\mathbf{S}_2\phi_f$. As $\mathbf{S}_1$ and $\mathbf{S}_2$ are symmetric matrices, the new node thus can obtains $\phi_f^t\mathbf{S}_1$ and $\phi_f^t\mathbf{S}_2$. Then the new node can computes

$$\phi_f^t\mathbf{S}_1 + z^{\alpha(f-1)}\phi_f^t\mathbf{S}_2.$$

One can check that the above α packets are precisely the packets stored in the failure node. □

Theorem 22. *In BASIC-PM MSR code, we can reconstruct all the B source packets by connecting to any k nodes, if the parameter m satisfies the requirement in Lemma 20.*

Proof. For an arbitrary set $\{\ell_i | i = 1, 2, \ldots, k\}$ of k nodes, we let $\mathbf{\Psi}_k$, Φ_k and Λ_k be the submatrix of $\mathbf{\Psi}$, Φ and Λ with rows indexed by $\{\ell_i | i = 1, 2, \ldots, k\}$ respectively. We obtain the packets $\mathbf{\Psi}_k\mathbf{M} = \begin{bmatrix} \Phi_k\mathbf{S}_1 + \Lambda_k\Phi_k\mathbf{S}_2 \end{bmatrix}$ and then get

$$\begin{bmatrix} \Phi_k\mathbf{S}_1\Phi_k^t + \Lambda_k\Phi_k\mathbf{S}_2\Phi_k^t \end{bmatrix}$$

by multiplying $\mathbf{\Psi}_k\mathbf{M}$ and Φ_k^t. For notation simplicity, let the two matrices P and Q to denote $\Phi_k\mathbf{S}_1\Phi_k^t$ and $\Phi_k\mathbf{S}_2\Phi_k^t$ respectively. Note that the matrices P and Q are symmetric, as $\mathbf{S}_1$ and $\mathbf{S}_2$ are symmetric.

In the following, I will show how to recover $\mathbf{S}_1$ and $\mathbf{S}_2$ from the matrix $P + \Lambda_k Q$. The i-th row and the j-th column entry of matrix $P + \Lambda_k Q$ is

$$P_{i,j} + z^{\alpha(\ell_i-1)}Q_{i,j}, \tag{7.3}$$

while the j-th row and the i-th column entry is

$$P_{j,i} + z^{\alpha(\ell_j-1)}Q_{j,i} = P_{i,j} + z^{\alpha(\ell_j-1)}Q_{i,j}, \tag{7.4}$$

the above equation follows from the symmetry of P and Q. Therefore, we can compute $P_{i,j}$ and $Q_{i,j}$ for $i \neq j$.

Let's first consider the matrix P. Up to now, all the non-diagonal elements of P are known. The elements in the i-th row (excluding the diagonal element) are given by

$$\phi_{\ell_i}^t\mathbf{S}_1 \begin{bmatrix} \phi_{\ell_1} & \cdots & \phi_{\ell_{i-1}} & \phi_{\ell_{i+1}} & \cdots & \phi_{\ell_{\alpha+1}} \end{bmatrix}.$$

We note that the matrix to the right is a Vandermonde matrix and we can obtain $\phi_{\ell_i}^t\mathbf{S}_1$ by Lemma 20.

Therefore, we can compute

$$\begin{bmatrix} \phi_{\ell_1}^t \\ \vdots \\ \phi_{\ell_\alpha}^t \end{bmatrix} \mathbf{S}_1.$$

The matrix in the above is also a Vandermonde matrix and we can recover $\mathbf{S}_1$ by Lemma 20. Similarly, we can recover the matrix $\mathbf{S}_2$ from the matrix Q.

□

7.2.2 BASIC Product-Matrix MBR Code

We divide the data file into

$$B = \frac{k(k+1)}{2} + k(d-k) \tag{7.5}$$

parts, each of size $m-1$ bits. Encode the B parts to the B source packets by generating a codeword of $\mathcal{C}_m$ for each part. Each node stores $\alpha = d$ coded packets. Create an $d \times d$ matrix

$$\mathbf{M} := \begin{bmatrix} \mathbf{S} & \mathbf{T} \\ \mathbf{T}^t & \mathbf{0} \end{bmatrix}.$$

The matrix $\mathbf{S}$ is a symmetric $k \times k$ matrix obtained by first filling the upper-triangular part by source packets $s_j(z)$, for $j = 1, 2, \ldots, k(k+1)/2$, and then obtain the lower-triangular part by

reflection along the diagonal. The rectangular matrix $\mathbf{T}$ has size $k \times (d-k)$, and the entries in $\mathbf{T}$ are source packets $s_j(z)$, $j = k(k+1)/2+1, \ldots, B$, listed in some fixed but arbitrary order. The matrix $\mathbf{T}^t$ is the transpose of $\mathbf{T}$ and the matrix $\mathbf{0}$ is an $(d-k)\times(d-k)$ all-zero matrix. For $i = 1, 2, \ldots, n$, let the encoding vector of node i be defined as in (7.1). Node i stores the d packets in $\boldsymbol{\psi}_i^t\mathbf{M}$.

Similar to the case of MSR code, we need to carefully choose the value of m. If we want to maintain the (n, k) recovery property, we need to make sure that the determinants of any $d \times d$ submatrices of $\boldsymbol{\Psi}$ are $\mathcal{C}_m$-invertible. The repair process and decoding process of BASIC-PM MBR codes are presented in the following two theorems respectively.

Theorem 23. *Assume that the parameter m satisfies the requirement in Lemma 20. Suppose node f fails, where f is a positive integer ranges from 1 to n, we can repair the packets of node f by downloading one packet from each of any d remaining nodes.*

Proof. The d coded packets stored in the failed node f are $\boldsymbol{\psi}_f^t\mathbf{M}$. The new node connects to an arbitrary set $\{h_j | j = 1, 2, \ldots, d\}$ of d helper nodes. Upon being contacted by the new node, the helper node h_j computes the inner product $\boldsymbol{\psi}_{h_j}^t\mathbf{M}\boldsymbol{\psi}_f$ and sends the product to the new node. The new node thus obtains the d coded packets

$\mathbf{\Psi}_{\text{repair}}\mathbf{M}\boldsymbol{\psi}_f$ from the d helper nodes, where

$$\mathbf{\Psi}_{\text{repair}} = \begin{bmatrix} \boldsymbol{\psi}_{h_1}^t \\ \boldsymbol{\psi}_{h_2}^t \\ \vdots \\ \boldsymbol{\psi}_{h_d}^t \end{bmatrix}.$$

By construction, the matrix $\mathbf{\Psi}_{\text{repair}}$ is a Vandermonde matrix and the determinant is $\mathcal{C}_m$-invertible by hypothesis. Thus, the new node recovers $\mathbf{M}\boldsymbol{\psi}_f$ through multiplication on the left by $\mathbf{\Psi}_{\text{repair}}^{-1}$. Since $\mathbf{M}$ is symmetric, we have $(\mathbf{M}\boldsymbol{\psi}_f)^t = \boldsymbol{\psi}_f\mathbf{M}$, and this is precisely the data previously stored in the failed node. □

Theorem 24. *For the constructed BASIC-PM MBR codes with the requirement in Lemma 20, we can reconstruct the B source packets from any k nodes.*

Proof. For any set of k nodes $\ell_1, \ell_2, \ldots, \ell_k$, we can solve for $\mathbf{T}$ from $\Phi_k\mathbf{T}$, where

$$\Phi_k = \begin{bmatrix} 1 & z^{\ell_1-1} & z^{2(\ell_1-1)} & \ldots & z^{(k-1)(\ell_1-1)} \\ 1 & z^{\ell_2-1} & z^{2(\ell_2-1)} & \ldots & z^{(k-1)(\ell_2-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z^{\ell_k-1} & z^{2(\ell_k-1)} & \ldots & z^{(k-1)(\ell_k-1)} \end{bmatrix}$$

is a Vandermonde matrix and invertible by Lemma 20. After subtracting the source packets in $\mathbf{T}$ from the first k columns of

$\mathbf{\Psi}_{\mathrm{k}}\mathbf{M}$, where $\mathbf{\Psi}_{\mathrm{k}}$ is the submatrix of $\mathbf{\Psi}$ with rows indexed by $\{\ell_i | i = 1, 2, \ldots, k\}$, we obtain $\Phi_k\mathbf{S}$ and can solve all the sources packets in $\mathbf{S}$ from $\Phi_k\mathbf{S}$. □

7.2.3 Example of BASIC Product-Matrix MBR Codes

In the following, we give an example for $n = 5$, $k = 3$, $d = 4$ and $m = 11$ of BASIC MBR code. This example contains all the essential feature of BASIC-PM MBR code.

There are $B = 9$ source packets $s_1(z)$ to $s_9(z)$. The matrix

$$\mathbf{M} = \left[\begin{array}{ccc|c} s_1(z) & s_2(z) & s_3(z) & s_7(z) \\ s_2(z) & s_4(z) & s_5(z) & s_8(z) \\ s_3(z) & s_5(z) & s_6(z) & s_9(z) \\ \hline s_7(z) & s_8(z) & s_9(z) & 0 \end{array}\right]$$

is a symmetric matrix with entries taken from $\mathbb{F}_2[z]/(1 + z^{11})$. For $i = 1, 2, \ldots, 5$, the encoding vector of node i is

$$\boldsymbol{\psi}_i^t = \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & z^{3(i-1)} \end{bmatrix}. \tag{7.6}$$

For $i = 1, 2, \ldots, 5$, node i stores the four packets in the i-th

row of $\mathbf{\Psi M}$, namely

$$s_1(z) + z^{i-1}s_2(z) + z^{2(i-1)}s_3(z) + z^{3(i-1)}s_7(z),$$
$$s_2(z) + z^{i-1}s_4(z) + z^{2(i-1)}s_5(z) + z^{3(i-1)}s_8(z),$$
$$s_3(z) + z^{i-1}s_5(z) + z^{2(i-1)}s_6(z) + z^{3(i-1)}s_9(z),$$
$$s_7(z) + z^{i-1}s_8(z) + z^{2(i-1)}s_9(z).$$

Each of the coded packets can be obtained by cyclic-right-shifting and adding the source packets appropriately.

Suppose that a data collector connects to nodes 1, 2 and 3. We can solve for $s_7(z)$, $s_8(z)$ and $s_9(z)$ from

$$\begin{bmatrix} s_7(z) + s_8(z) + s_9(z) \\ s_7(z) + zs_8(z) + z^2s_9(z) \\ s_7(z) + z^2s_8(z) + z^4s_9(z) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & z & z^2 \\ 1 & z^2 & z^4 \end{bmatrix} \begin{bmatrix} s_7(z) \\ s_8(z) \\ s_9(z) \end{bmatrix}.$$

As the above encoding matrix is invertible by Lemma 20, we can thus decode $s_1(z)$ to $s_6(z)$ from

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & z & z^2 \\ 1 & z^2 & z^4 \end{bmatrix} \begin{bmatrix} s_1(z) & s_2(z) & s_3(z) \\ s_2(z) & s_4(z) & s_5(z) \\ s_3(z) & s_5(z) & s_6(z) \end{bmatrix}.$$

Suppose node 5 fails and we want to regenerate it from node 1, 2, 3 and 4. The coded packet sent from helper node i to the

newcomer is $\boldsymbol{\psi}_i^t \mathbf{M} \boldsymbol{\psi}_5$. If we put the packets received by the new node as a column vector, then the column vector can be written as

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 \\ 1 & z^2 & z^6 & z^8 \\ 1 & z^3 & z^6 & z^9 \end{bmatrix} \cdot \mathbf{M} \cdot \boldsymbol{\psi}_5.$$

Since the matrix on the left is invertible by the result in Lemma 20, we can compute $\mathbf{M} \cdot \boldsymbol{\psi}_5$, as $\mathbf{M}$ is symmetric, this is exactly equal to the content of the failed node.

The repair of other nodes can be done similarly. During the repair process of a failed node, each of the helper nodes cyclic-shifts the four packets in their memory according to the encoding vector of the failure node, and then add the shifted version. Each bit transmitted from the helping nodes is obtained by merely XORing four bits.

Although we only give the conversion of the product-matrix construction in [19], it is easy to check that we can convert all the proposed exact repair RGC in [8, 19–23] to the exact repair BASIC codes.

In the repair and decoding processes, although we show that we can decode the packets by multiplying the inverse matrix and the

received packets, we can employ a more efficient decoding method, Algorithm 1, which is given in Section 5.2.1.

□ **End of chapter.**

Chapter 8

Computational Complexity of BASIC RGC

In this chapter, we evaluate computational complexity of BASIC regenerating codes and RGC over a finite field, both for functional repair and exact repair. In the following, we first present the polynomial representation of finite field to give an accurate complexity of RGC over a finite field. Then we demonstrate that the coding and repair computational complexity of functional repair BASIC codes is less than that of functional repair RGC over a finite field. For exact repair BASIC-PM code, we show that the coding and repair complexity is much less than that of RGC-PM code over a finite field, by employing the Algorithm 1 given in Section 5.2.1.

8.1 Polynomial Representation of Finite Field

We represent a finite field of size 2^w as the quotient ring $\mathbb{F}_2[z]/(g(z))$ for an irreducible polynomial $g(z)$ of degree w, and use a polynomial basis to represent a finite field element. Addition is bit-wise XOR and multiplication in the field is multiplication modulo the irreducible polynomial $g(z)$. Generally, a multiplication in the field $\mathbb{F}_{2^w}$ takes $O(w^2)$ bit operations. (See e.g. [52, Chp. 11].)

There is a wide range of multiplication methods whose efficiency and level of sophistication increase with the size of operands. The easiest field multiplication in current software implementation is typically performed by using pre-calculated lookup tables for the full multiplication result [53], which requires a table of size $2^w \times 2^w \times w$ bits. Therefore, this method is only suitable for small field ($w \leq 8$), due to the limitation of memory. Another approach to perform a modular multiplication is to compute the product first and then reduce it independently. This is especially effective for large fields where it is worth using advanced multiplication techniques, such as Karatsuba-Ofman algorithm [54, 55] and Fast Fourier Transform (FFT) [56–58]. In the field $\mathbb{F}_{2^w}$, the field multiplication complexity may be improved to $O(w^{\log_2 3})$ using

Karatsuba-Ofman algorithm [54]. The most efficient FFT algorithm is proposed in [58], which has a multiplication complexity of $O(w \log_2 w)$. Moreover, all the advanced multiplication techniques are also suitable for the multiplication of the binary cyclic codes, and the multiplication complexity of binary cyclic codes can also reduced to $O(m^{\log_2 3})$ using Karatsuba-Ofman algorithm and $O(w \log_2 w)$ by FFT algorithm in [58]. So, for fair comparison, we implement the finite field multiplication by first computing the product and then reducing the irreducible polynomial, do not employ the advanced multiplication techniques.

In the following, we give an upper bound of XORs involved in the multiplication over the finite field $\mathbb{F}_2[z]/(g(z))$, where $g(z)$ is an irreducible polynomial in $\mathbb{F}_2[z]$ of degree w. Define the number of non-zero terms of polynomial $f(z)$ as the *weight* of $f(z)$, which is denoted as $\|f(z)\|_0$. For example, the weight of polynomial $1+z+z^2$ is 3, as it has three non-zero terms. In the multiplication of $a(z)$ and $b(z)$ over $\mathbb{F}_2[z]/(g(z))$, we first compute the *product* of $c(z) := a(z)b(z)$,

$$c(z) = a_0 b(z) + a_1 z b(z) + \cdots + a_{w-1} z^{w-1} b(z),$$

where $c(z) = \sum_{i=0}^{2w-2} c_i z^i$. The product of $a(z)$ and $b(z)$ takes at

most w^2 XORs, as $\|a(z)\|_0 \leq w, \|b(z)\|_0 \leq w$. The average number of XORs involved in the product is $0.5w^2$.

Then, we reduce the polynomial $c(z)$ by $g(z)$,

1. If $\deg c(z) \geq w$, let $\ell = \deg c(z)$.
2. Remove the term $c_\ell z^\ell$ of $c(z)$, and add $c_\ell(g(z) - z^w)z^{\ell-w}$ to $c(z)$,
$$c(z) = \sum_{i=0}^{\ell-1} c_i z^i + (g(z) - z^w)(c_\ell z^{\ell-w}).$$
3. Repeat the above until $\deg c(z) < w$.

After each iteration in the above, the degree of $c(z)$ is decreased by at least one and the degree of $a(z)b(z)$ is at most $2w-2$. So we need to go through the iteration at most $w-1$ times. In each iteration, we need to replace the term and update the polynomial $c(z)$, which takes $\|g(z)\|_0$ XORs. Therefore, we can count that the average number of XORs of the field multiplication $a(z)b(z)$ is at most

$$(0.5w + \|g(z)\|_0)w. \tag{8.1}$$

In the ring $\mathcal{R}_m$, we let $a(z) \in \mathcal{R}_m$ and $b(z) \in \mathcal{C}_m$. The multiplication $a(z)b(z)$ is simply the convolutional product of the coefficient vectors:

$$a(z)b(z) = \sum_{\ell=0}^{m-1} \Big(\sum_{i \oplus_m j = \ell} a_i b_j \Big) z^\ell,$$

where the symbol "$\oplus_m$" in the above stands for addition modulo m. Recall that $h(z) = 1 + z + \cdots + z^{m-1}$ is the check polynomial of $\mathcal{C}_m$. For any polynomial $a(z)$ in the ring $\mathcal{R}_m$ and $\forall b(z) \in \mathcal{C}_m$, we have that $a(z)b(z) = (a(z) + h(z))b(z)$. If the number of non-zero terms of $a(z)$ is larger than $(m-1)/2$, then we can compute $(a(z) + h(z))b(z)$ instead of $a(z)b(z)$ and the number of non-zero terms is less than or equal to $(m-1)/2$. Therefore, we can assume that $\|a(z)\|_0 \leq (m-1)/2$ without loss of generality. The number of XORs of multiplication $a(z)b(z)$ in $\mathcal{R}_m$ is thus at most $(m-1)m/2$. Therefore, the computational complexity of one multiplication over the ring $\mathcal{R}_m$ is much less than that of finite field multiplication, if we let $m - 1 = w$. The essential reason is that the multiplication $a(z)b(z)$ over $\mathcal{R}_m$ is a summing of at most $(m-1)/2$ cyclic-shifted versions of $b(z)$, while the multiplication over finite field not only need to compute $(m-1)/2$ shifted versions of $b(z)$ on average, but also modulo the irreducible polynomial $g(z)$.

8.2 Computational Complexity of Functional Repair BASIC RGC and RGC over Finite Field

For the purpose of easy presentation, we only consider the MSR case, i.e., each set of k nodes contains just enough information to decode the original data file. The equation in (6.1) holds with $B = k\alpha$ when $(d-i+1)\beta \geq \alpha$ for all $i = 1, 2, \ldots, k$. The minimum value of β is thus $\beta = \alpha/(d-k+1) = B/(k(d-k+1))$. In the remainder of this section, the parameters B, α and β are set to $B = k(d-k+1)$, $\alpha = d-k+1$ and $\beta = 1$. In the following, we consider the parameter m to be a prime number such that the multiplicative order of $2 \mod m$ is $m-1$ and the inequality (6.5) holds with $L = 1$.

8.2.1 BASIC Regenerating Codes

Without loss of generality we assume that the data file contains $\kappa B(m-1) = \kappa k\alpha(m-1)$ bits, where κ is a positive integer. For the ease of comparison, we will normalize the complexity by the file size.

Theorem 25. *Let m be a prime number such that the multiplicative order of $2 \mod m$ is $m-1$. In the repair process, the local*

encoding coefficients are restricted to be polynomials in $\mathcal{R}_m$ with number of non-zero term less than or equal to $(m-1)/2$. The normalized encoding complexity, repair complexity and decoding complexity of functional repair BASIC regenerating codes are at most $\frac{n\alpha m}{2}$, $\frac{d\beta m}{k}$ and $\frac{Bm}{2}$ respectively.

Proof. See Appendix A.6. □

8.2.2 Regenerating Codes Over Finite Field

Table 8.1: Comparison of functional repair.

	BASIC RGC	RGC
Normalized redundancy	$\frac{m}{m-1} \cdot \frac{n}{k}$	$\frac{n}{k}$
Normalized repair bandwidth	$\frac{m}{m-1} \cdot \frac{d}{k\alpha}$	$\frac{d}{k\alpha}$
Normalized encoding complexity	$\frac{n\alpha m}{2}$	$n\alpha(0.5m + \|\|g(z)\|\|_0)$
Normalized repair complexity	$\frac{d\beta m}{k}$	$\frac{2d\beta(0.5m+\|\|g(z)\|\|_0)}{k}$
Normalized decoding complexity	$\frac{k\alpha m}{2}$	$k\alpha(0.5m + \|\|g(z)\|\|_0)$

Consider functional repair RGC over the finite field $\mathbb{F}_{2^w}$, where w is an integer larger than

$$\log_2 \left(B \cdot \max \left\{ \binom{n\alpha}{B}, 2|\mathcal{H}| \right\} \right). \tag{8.2}$$

As the upper bounds of $m+1$ and w are the same from the inequalities in (6.5) and (8.2), we can let $m = w - 1$, when we

compare the computational complexity for RGC over a finite field and BASIC regenerating codes. The coding and repair processes of RGC over a finite field is similar to that of BASIC codes. The main difference is that we replace the binary parity-check code by the field element. The field element is viewed as a polynomial of degree at most $m-1$, and the coefficient is either 1 or 0.

Suppose that the data file contains $B(m-1)$ bits without loss of generality. In the encoding process, the file is divided into B source packets, we need to generate $n\alpha$ coded packets. Each coded packet is obtained by taking an $\mathbb{F}_{2^{m-1}}$-linear combination of the B source packets. The computation of such a linear combination is dominated by B multiplications and $B-1$ additions. One addition in the field takes $m-1$ XORs, and one multiplication takes $(m-1)(0.5(m-1)+||g(z)||_0)$ XORs at most by the equation (8.1). We can thus have that one coded packet takes $B(m-1)(0.5(m-1)+||g(z)||_0)+(B-1)(m-1)$ XORs. The normalized encoding complexity is at most $\kappa n\alpha B(m-1)(0.5(m-1)+||g(z)||_0)/(\kappa B(m-1)) = n\alpha(0.5(m-1)+||g(z)||_0)$. Likewise, the repair and decoding complexity of RGC over finite field can be computed.

The comparison of computational complexity is summarized in Table 8.1. The first row is the performance metric of the proposed

functional repair BASIC regenerating codes, and the second row is the functional repair RGC using a finite field as alphabet. The *normalized redundancy* is defined as the total number of bits in the storage system divided by the number of bits in the data file. As we are comparing at the MSR point, the storage efficiency is $n\alpha/B = n/k$ for RGC over a finite field. The coding scheme proposed in this thesis has an additional 1 bit per $m-1$ bits, and this leads to a slight increase in normalized redundancy by a factor of $m/(m-1)$. Similarly, there is a factor of $m/(m-1)$ in the normalized repair bandwidth of BASIC RGC. The storage efficiency and normalized repair bandwidth of the two coding schemes are approximately the same when m is large. The results of Table 8.1 show that the normalized computational complexity of RGC over a finite field is lager than that of BASIC regenerating codes, for both coding and repair processes.

In this subsection, we consider a class of special prime number m such that the multiplicative order of $2 \mod m$ is $m-1$. For this special prime m, the ring $\mathcal{C}_m$ is in fact isomorphic to a finite field of size 2^{m-1}. A method of fast multiplication in $\mathcal{R}_m$ is described in [59], which shows that multiplication in $\mathcal{R}_m$ is approximately twice as efficient as multiplication in $\mathbb{F}_{2^{m-1}}$ with the polynomial

basis representations.

In the above, we only consider the complexity of functional repair BASIC regenerating codes of $L = 1$. If $L > 1$, let $f_1(z)f_2(z)\cdots f_L(z)$ be the prime factorization of the check polynomial $h(z)$ and $\deg(f_1(z)) \leq \deg(f_\ell(z))\ \forall 2 \leq \ell \leq L$. Let the set $\mathcal{S}$ be equal to $\mathbb{F}_2[z]/f_1(z)$, we can check that the the function θ_ℓ is injective $\forall 1 \leq \ell \leq L$. The weight of local encoding coefficient is less than or equal to $\deg(f_1(z))$, and from Theorem 18, we have

$$\deg(f_1(z)) > \log_2\left(L \cdot B \cdot \max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}\right). \tag{8.3}$$

Note that the repair complexity increases as the weight of the local encoding coefficients increases, and the decoding complexity of increases along with the increase of m, where $m-1 \geq L\deg(f_1(z))$. Then the repair complexity is much less than that of functional repair RGC over a finite field, while the decoding complexity may be larger than that of functional repair RGC over a finite field.

8.3 Computational Complexity of BASIC Product-Matrix Codes

In this section, we estimate the computational complexity of encoding, repair and decoding, in terms of the number of XORs

for exact repair BASIC-PM codes and RGC-PM codes. Since the derivations of the complexity of the BASIC-PM MSR and MBR are similar, we will only consider the MBR case. Let m be a positive odd number such that $n-1$ is strictly less than all divisors of m which are not equal to 1. The encoding vector of node i is

$$\begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & \dots & z^{(d-1)(i-1)} \end{bmatrix},$$

for $i = 1, 2, \dots, n$.

Theorem 26. *Let m be a positive odd number such that $n-1$ is strictly less than all divisors of m which are not equal to 1. If we use Algorithm 1 to decode the linear systems in the repair and decoding processes of BASIC-PM MBR code, the normalized encoding complexity, repair complexity and decoding complexity of BASIC-PM MBR codes are $\frac{2n\alpha^2}{k(2d-k+1)}$, $\frac{5.5d^2}{k(2d-k+1)}$ and $\frac{k(kd-k^2+4.5d-3k)}{(2d-k+1)}$ respectively.*

Proof. See Appendix A.7. □

Recall that the $\nu+1$ linear equations in Vandermonde matrix form $\mathbf{V}_\nu \mathbf{x} = \mathbf{b}$ can be solved by employing Algorithm 1, which takes $\nu(\nu+1)$ multiplications and $\nu(\nu+1)$ additions. For the $\nu+1$ linear equations $\mathbf{V}_\nu \mathbf{x} = \mathbf{b}$ over a finite field $\mathbb{F}_{2^w}$, as the number of XORs of a field multiplication is at most $w(w + \|g(z)\|_0)$ from

Table 8.2: Computational complexity of exact repair with algorithm 1.

	BASIC-PM MBR	RGC-PM MBR
Normalized encoding complexity	$\frac{2nd^2}{k(2d-k+1)}$	$\frac{2(w+\|g(z)\|_0)nd^2}{k(2d-k+1)}$
Normalized repair complexity	$\frac{5.5d^2}{k(2d-k+1)}$	$\frac{4d^2(w+\|g(z)\|_0)}{k(2d-k+1)}$
Normalized decoding complexity	$\frac{k(kd-k^2+4.5d-3k)}{(2d-k+1)}$	$\frac{k(kd-k^2+3d-2k)(w+\|g(z)\|_0)}{(2d-k+1)}$

(8.1), we can count that the computation of solving the $\nu+1$ linear equations by Algorithm 1 is at most $\nu(\nu+1)w(w+\|g(z)\|_0+1)$ XORs. If we replace the binary cyclic code $\mathcal{C}_m$ with the field element in $\mathbb{F}_{2^w}$, the computational complexity of RGC-PM MBR code with Algorithm 1 can be calculated and is summarized in Table 8.2.

From Table 8.2, we can see that the encoding, repair and decoding complexity normalized the file size of BASIC-PM MBR code does not depend on the packet size m. The normalized encoding complexity of RGC-PM MBR code over a finite field $\mathbb{F}_{2^w}$ is $(w+\|g(z)\|_0)$ times of that of BASIC-PM MBR code, and the normalized repair/decoding complexities of RGC-PM MBR code are roughly $(w+\|g(z)\|_0)$ times of the normalized repair/decoding complexities of BASIC-PM MBR code. In RGC-PM MBR code over a finite field $\mathbb{F}_{2^w}$, the parameters have to satisfy $w > \log_2 n$. When the system parameter n is very large, the computational

complexity of BASIC-PM MBR code is thus much less than that of RGC-PM MBR code, for encoding, repair and decoding processes.

Consider the example of BASIC-PM MBR code given in Subsection 7.2.3 with parameters $n = 5$, $k = 3$ and $d = 4$. We can count that the normalized encoding complexity, normalized decoding complexity and normalized repair complexity is roughly 8.9 XORs, 6 XORs and 4.9 XORs respectively. While for the same parameters of RGC-PM MBR code with the same encoding matrix over the finite field $\mathbb{F}_{2^3}$, the generater polynomial of the field $\mathbb{F}_{2^3}$ is $1 + z + z^3$. By Table 8.2, the normalized encoding complexity, normalized decoding complexity and normalized repair complexity is approximately equal to 44 XORs, 22.5 XORs and 17.8 XORs respectively. We can see BASIC-PM MBR code has only 20% encoding complexity, 26.7% decoding complexity and 27.5% repair complexity of that of RGC-PM MBR code.

Table 8.3: Normalized computation of three operations in $\mathcal{R}_m$ and field $\mathbb{F}_{2^w}$.

Operation	$\mathcal{R}_m$	$\mathbb{F}_{2^w}$
$a(z) + b(z)$	1	1
Solve $s(z)$ from $z^i s(z)$	0	$w + \|g(z)\|_0$
Solve $s(z)$ from $(z^i + z^j)s(z)$	$\frac{3(m-1)}{2m}$	$w + \|g(z)\|_0$

The normalized decoding complexity of BASIC-PM MBR code is significantly less than that of RGC-PM MBR code, when we employ Algorithm 1 for both of them. The essential reason is as follows. By employing Algorithm 1, the decoding process of both BASIC-PM MBR code and RGC-PM MBR code can be partitioned to three operations: (1) compute the addition $a(z)+b(z)$, (2) solve the polynomial $s(z)$ from $z^i s(z)$, (3) solve the polynomial $s(z)$ from $(z^i + z^j)s(z)$. All the operations of BASIC-PM MBR code are over $\mathcal{R}_m$, while the operations of RGC-PM MBR code are the field $\mathbb{F}_{2^w}$. Table 8.3 summarizes the normalized computation of the three operations over $\mathcal{R}_m$ and the three operations over $\mathbb{F}_{2^w}$. In the Table 8.3, the normalized computation is the number of XORs normalized the value m for $\mathcal{R}_m$ or normalized w for $\mathbb{F}_{2^w}$. We can efficiently decode the polynomial $s(z)$ from $z^i s(z)$ or from $(z^i + z^j)s(z)$ for $0 \leq i, \neq j < m$ in BASIC-PM MBR code. While the computational complexity of a finite field multiplication and division is much higher in polynomial basis representation, compared with the multiplication $z^i s(z)$ and division $c(z)/(z^i + z^j)$ in the ring $\mathcal{R}_m$.

□ **End of chapter.**

Chapter 9

Implementation

We implement the proposed BASIC codes, includes BASIC array codes and BASIC-PM MBR codes, and evaluate their encoding, decoding and repair performances in order to validate our theoretical analysis.

In our implementation of BASIC codes, each bit or coefficient of a polynomial in $\mathcal{C}_m$ corresponds to a *chunk* and we fix the chunk size to be 4 KBytes, which is the default disk block size in existing Linux extended file systems. The file in all the experiments is of size 80 MBytes. The machine for testing has a 1GHz single-core processor, 1GB of RAM and 20GB of Hard Disk. It runs CentOS Linux, version 5.6 on VMware Workstation. Each data point in the graphs that follow is the average of one thousand runs.

9.1 BASIC Array Codes

The decoding performances of BASIC array codes are measured and compared to the publicly available implementation of CRS codes, Jerasure 2.0 in [60]. We compare three decoding algorithms: (i) the LU decoding algorithm in Algorithm 1 for four information erasures, (ii) the Cramer's rule method with extended Euclidean Algorithm for three information and one parity erasures, and (iii) the Cramer's rule method with Lemma 11 for some cases of three information and one parity erasures.

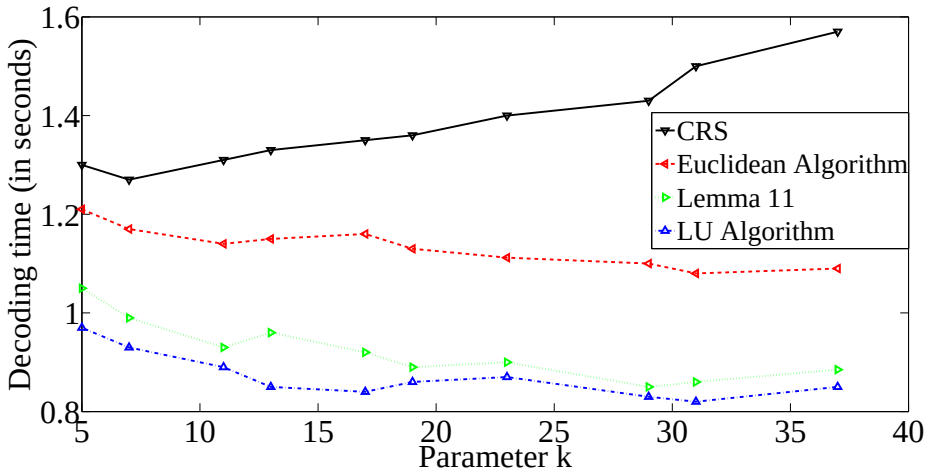


Figure 9.1: The decoding time of BASIC array codes $\mathcal{C}(m, 4, m)$ and Cauchy Reed-Solomon codes.

For BASIC array code, we let $k = m$. The results are shown in Figure 9.1, where the parameter k varies from 5 to 37. In the experiments of CRS codes, we also choose the chunk size to be 4

KBytes, as like BASIC array codes. It is clear that the decoding performances of all the three decoding algorithms of BASIC array codes $\mathcal{C}(m, 3, m)$ outperform that of CRS code. Among the three decoding algorithms of BASIC array codes, the LU method has the best decoding performance, reduces the decoding time of the conventional Cramer's rule method with extended Euclidean Algorithm roughly 20 percent. The Cramer's rule method with Lemma 11 has a slightly advantage of the conventional Cramer's rule method with extended Euclidean Algorithm in terms of decoding performance.

9.2 BASIC-PM MBR Codes

We implement BASIC-PM MBR codes and product-matrix MBR codes over finite field to verify the advantage of BASIC-PM MBR codes in terms of computation cost. Product-matrix MBR codes over finite field in [19] are implemented using Jerasure 2.0 [60]. In order to reduce the computational cost of product-matrix MBR codes, we choose the encoding matrix to be an $n \times d$ Cauchy matrix. In our experiments of two codes, the parameters are fixed to $d = \alpha = k$, $n = k + 3$ and k ranges from 6 to 20. For BASIC-PM MBR code, we choose value of the parameter m to be 23. It is easy

to check that this value satisfies the requirement in Lemma 20 for the given parameters.

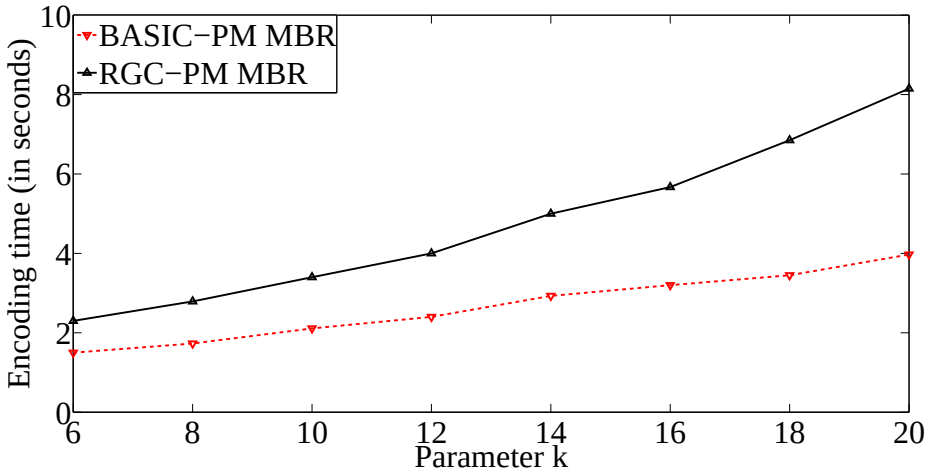


Figure 9.2: The encoding time of BASIC-PM MBR code and RGC-PM MBR code.

We first evaluate the encoding performance, which is shown in Figure 9.2. We observe that in both two codes, the encoding time increases as the value of k increases, mainly because the normalized encoding complexity increases as the parameter k increases. For all the values of parameter k, the encoding time of BASIC-PM MBR code is much less than that of RGC-PM MBR code.

We now evaluate the repair time. Figure 9.3 shows the repair time when a single node fails. We observe that the repair time of BASIC-PM MBR code is almost the same for different values of k, as like the relation between the normalized repair complexity and

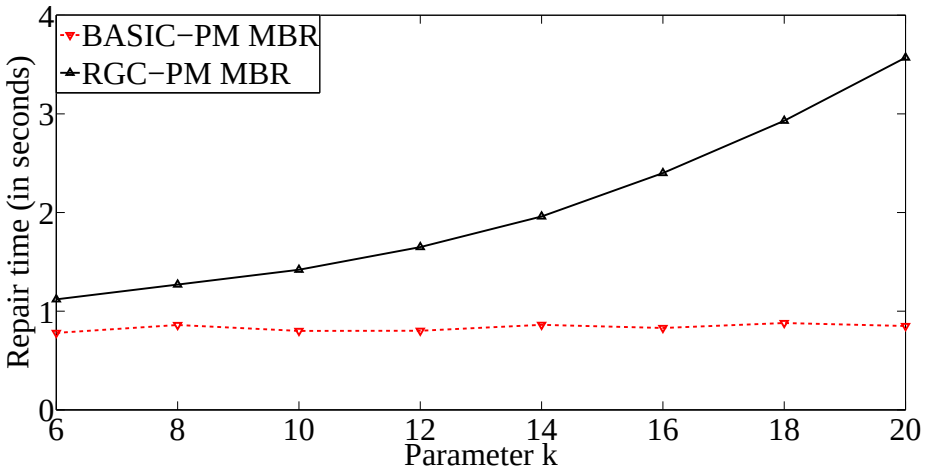


Figure 9.3: The repair time of BASIC-PM MBR code and RGC-PM MBR code.

the parameter k. However, the repair time of RGC-PM MBR code increases along with the parameter k increase, as the normalized repair complexity is directly proportional to k. In general, the repair time of RGC-PM MBR code is much larger than that of BASIC-PM MBR code, and the difference becomes bigger when k becomes bigger.

We now evaluate the decoding time for the two codes. Here decoding time is the time of reconstructing the original data file from any k storage nodes. Figure 9.4 shows the decoding time. Similar to the encoding performance, the decoding time of both two codes increases with the parameter k increases, as the normalized decoding complexity increase along with the parameter k increase. BASIC-PM MBR code can reduces the decoding time of RGC-PM MBR

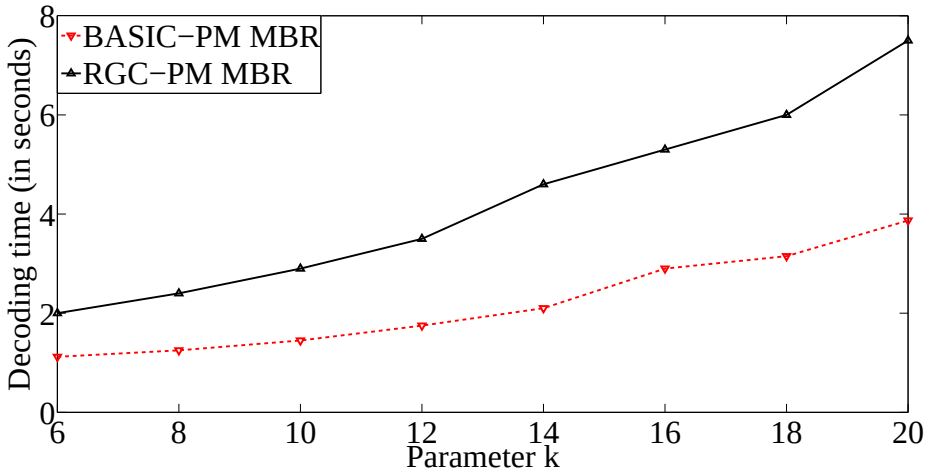


Figure 9.4: The decoding time of BASIC-PM MBR code and RGC-PM MBR code.

code greatly for all the evaluated parameters.

□ **End of chapter.**

Chapter 10

Conclusion

In this thesis, we investigated the erasure codes of distributed storage systems with the goal of designing low complexity storage codes. Based on the insights of the existing methodology in reducing computational complexity in network coding problems in [13–15], we propose a framework of designing low-complexity linear codes that employ XOR and bit-wise cyclic shifts called BASIC codes. However, how to reduce the repair bandwidth of BASIC array codes or how to determine the optimal repair bandwidth of BASIC array codes is still an open problem, which I plan to continue researching in the future.

□ **End of chapter.**

Appendix A

Proofs

A.1 Proof of Theorem 6

Before giving the proof, we first need to investigate the determinant of generalized Vandermonde matrix.

A *generalized Vandermonde matrix* is a square submatrix of some Vandermonde matrix. We will need two types of generalized Vandermonde matrix in this proof. For $i = 1, 2, \ldots, k$ and variables $a_1, a_2, \ldots, a_k$, let

$$\mathbf{V}_i(\mathbf{a}) \triangleq \begin{bmatrix} 1 & a_1 & \cdots & a_1^{i-1} & a_1^{i+1} & \cdots & a_1^k \\ 1 & a_2 & \cdots & a_2^{i-1} & a_2^{i+1} & \cdots & a_2^k \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k & \cdots & a_k^{i-1} & a_k^{i+1} & \cdots & a_k^k \end{bmatrix}.$$

and for $1 \le i < j \le k+1$, let $\mathbf{V}_{i,j}(\mathbf{a})$ be the matrix

$$\mathbf{V}_{i,j}(\mathbf{a}) \triangleq \begin{bmatrix} 1 & \cdots & a_1^{i-1} & a_1^{i+1} & \cdots & a_1^{j-1} & a_1^{j+1} & \cdots & a_1^{k+1} \\ 1 & \cdots & a_2^{i-1} & a_2^{i+1} & \cdots & a_2^{j-1} & a_2^{j+1} & \cdots & a_2^{k+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & a_k^{i-1} & a_k^{i+1} & \cdots & a_k^{j-1} & a_k^{j+1} & \cdots & a_k^{k+1} \end{bmatrix}.$$

Note that when $i = k$ and $j = k+1$, $\mathbf{V}_i(\mathbf{a})$ and $\mathbf{V}_{i,j}(\mathbf{a})$ reduce to the square Vandermonde matrix, and we denote the square Vandermonde matrix as $\mathbf{V}(\mathbf{a})$.

Lemma 27. *Let* $\mathbf{a}$ *denote the* k*-dimensional vector* $(a_1, \ldots, a_k)$*. For* $i = 1, 2, \ldots, k$*, we have*

$$\det \mathbf{V}_i(\mathbf{a}) = \sigma_{k-i}(\mathbf{a}) \det \mathbf{V}(\mathbf{a}),$$

and for $1 \le i < j \le k+1$*,*

$$\det \mathbf{V}_{i,j}(\mathbf{a}) = \begin{vmatrix} \sigma_{k-i}(\mathbf{a}) & \sigma_{k-j}(\mathbf{a}) \\ \sigma_{k+1-i}(\mathbf{a}) & \sigma_{k+1-j}(\mathbf{a}) \end{vmatrix} \cdot \det \mathbf{V}(\mathbf{a}),$$

where $\sigma_\ell(\mathbf{a})$ *denotes the* ℓ*-th* elementary symmetric polynomials

$$\sigma_\ell(\mathbf{a}) \triangleq \sum_{1 \le j_i < j_2 < \cdots < j_\ell \le k} a_{j_1} a_{j_2} \cdots a_{j_\ell}.$$

Proof. For $\det \mathbf{V}_i(\mathbf{a})$, consider the following $(k+1) \times (k+1)$ determinant

$$\begin{vmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^k \\ 1 & a_2 & a_2^2 & \cdots & a_2^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k & a_k^2 & \cdots & a_k^k \\ 1 & Z & Z^2 & \cdots & Z^k \end{vmatrix} = \det \mathbf{V}(\mathbf{a}) \prod_{m=1}^{k} (Z - a_m)$$

as a polynomial with variable Z. By expanding the determinant along the last row, we see that the coefficient of Z^i is $\sigma_{k-i}(\mathbf{a}) \det \mathbf{V}(\mathbf{a})$, for $i = 0, 1, \ldots, k$. The first part of the theorem follows from comparing the coefficients on both sides of the above equation.

For the second part of the theorem, consider the following $(k+1) \times (k+1)$ determinant as a polynomial in Y and Z,

$$\begin{vmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{k+1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{k+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k & a_k^2 & \cdots & a_k^{k+1} \\ 1 & Y & Y^2 & \cdots & Y^{k+1} \\ 1 & Z & Z^2 & \cdots & Z^{k+1} \end{vmatrix},$$

and note that the value of $\det \mathbf{V}_{i,j}(\mathbf{a})$ is equal to

$$(-1)^{i+j+1}(\text{coeff. of } Z^j Y^i - \text{coeff. of } Z^i Y^j).$$

After expanding and re-writing the determinant as

$$\begin{aligned}&\det(\mathbf{V}(\mathbf{a})) \cdot (Y^k - Y^{k-1}\sigma_1(\mathbf{a}) + \cdots + (-1)^k \sigma_k(\mathbf{a})) \\ &\quad \cdot (Z^k - Z^{k-1}\sigma_1(\mathbf{a}) + \cdots + (-1)^k \sigma_k(\mathbf{a})) \cdot (Z - Y),\end{aligned}$$

we can express $\det \mathbf{V}_{i,j}(\mathbf{a})$ in terms of $\det \mathbf{V}(\mathbf{a})$ and the elementary symmetric polynomials in $\mathbf{a}$ by comparing coefficients. □

We are now ready to prove the theorem. In Theorem 4, we state that if the determinant of any $k \times k$ submatrix of the generator matrix $\mathbf{G}$, after reduction modulo $h(z)$, is a nonzero polynomial in $\mathbb{F}_2[z]/(h(z))$, then the BASIC code $\mathcal{C}(k, r, m)$ is MDS. In other words, the array code $\mathcal{C}(k, r, m)$ is MDS if, for all $\ell = 1, 2, \ldots, \min\{k, r\}$, the determinant of each $\ell \times \ell$ submatrix of the matrix $\mathbf{P}$, regarded as a polynomial in $\mathbb{F}_2[z]$, is not divisible by $1 + z^m$.

The determinant of an $\ell \times \ell$ sub-matrix of the $\mathbf{P}$ defined in (4.4)

can be written as

$$\begin{vmatrix} z^{i_1 j_1} & z^{i_1 j_2} & \dots & z^{i_1 j_\ell} \\ z^{i_2 j_1} & z^{i_2 j_2} & \dots & z^{i_1 j_\ell} \\ \vdots & \vdots & \ddots & \vdots \\ z^{i_\ell j_1} & z^{i_\ell j_2} & \dots & z^{i_\ell j_\ell} \end{vmatrix}$$

with $0 \leq i_1 < \cdots < i_\ell \leq k-1$ and $0 \leq j_1 < \cdots < j_\ell \leq r-1$. By factoring out powers of z, it is sufficient to show that a determinant in the following form

$$\begin{vmatrix} 1 & z^{i_1(j_2-j_1)} & \dots & z^{i_1(j_\ell-j_1)} \\ 1 & z^{i_2(j_2-j_1)} & \dots & z^{i_2(j_\ell-j_1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z^{i_\ell(j_2-j_1)} & \dots & z^{i_\ell(j_\ell-j_1)} \end{vmatrix} \tag{A.1}$$

is not divisible by $1 + z^m$ in the polynomial ring $\mathbb{F}_2[z]$. We distinguish seven cases. For ease of presentation, we assume $r \geq k$ without loss of generality in the following discussion.

Case $\ell = 1$. The determinant of size 1×1 in (A.1) is equal to 1, and hence cannot be divisible by $1 + z^m$.

Case $\ell = 2$. The determinant in (A.1) is equal to $z^{i_2(j_2-j_1)} + z^{i_1(j_2-j_1)}$. It is divisible by $1 + z^m$ if and only if

$$i_2(j_2 - j_1) \equiv i_1(j_2 - j_1) \bmod m. \tag{A.2}$$

Since $j_2 - j_1$ is strictly smaller than m and $0 \leq i_1 < i_2 \leq k-1$, the condition in (A.2) implies that $i_1 = i_2$, which contradicts the fact that $i_1 < i_2$.

Case $\ell = 3$. Re-write the determinant in (A.1) as

$$\begin{vmatrix} 1 & z^{a\alpha} & z^{a\beta} \\ 1 & z^{b\alpha} & z^{b\beta} \\ 1 & z^{c\alpha} & z^{c\beta} \end{vmatrix} = z^{b\alpha+c\beta} + z^{c\alpha+b\beta} + z^{a\alpha+c\beta} + z^{c\alpha+a\beta} + z^{a\alpha+b\beta} + z^{b\alpha+a\beta}.$$

with $0 \leq a < b < c \leq k-1$ and $1 \leq \alpha < \beta \leq r-1$. If we view this determinant as a polynomial in $\mathbb{F}_2[z]$, then the degree of the polynomial is $b\alpha + c\beta \leq 2kr - 3k - 3r + 5$, which is less than m. Thus, the determinant in $\mathbb{F}_2[z]$, is not divisible by $1 + z^m$.

Case $\ell = k-2$. The $(k-2) \times (k-2)$ determinant in (A.1) is the determinant of a generalized Vandermonde matrix. Let $\mathbf{z}$ denote the vector $(z^{i_1}, z^{i_2}, \ldots, z^{i_{k-2}})$ such that $0 \leq i_1 < i_2 < \cdots < i_{k-2} \leq k-1$. The determinant is equal to either $\sigma_i(\mathbf{z}) \cdot \det \mathbf{V}(\mathbf{z})$, with $0 \leq i \leq k-3$, or

$$(\sigma_{k-3-i}(\mathbf{z})\sigma_{k-2-j}(\mathbf{z}) - \sigma_{k-3-j}(\mathbf{z})\sigma_{k-2-i}(\mathbf{z})) \cdot \det \mathbf{V}(\mathbf{z})$$

with $1 \leq i < j \leq k-3$.

We first note that all irreducible factors of $\det \mathbf{V}(\mathbf{z})$ have

degrees strictly less than $m-1$, since the values of $i_1, i_2, \ldots, i_{k-2}$ are distinct and less than m, and the factors $z^{i_2} - z^{i_1}$, $z^{i_3} - z^{i_1}$, $z^{i_4} - z^{i_1}$ *etc.* are divisible by $1 + z$ but not by $1 + z^m$. Hence $\det \mathbf{V}(\mathbf{z})$ is not divisible by $1 + z^m$, and it suffices to show that $\sigma_i(\mathbf{z})$, for $0 \leq i \leq k-3$, and $\sigma_{k-3-i}(\mathbf{z})\sigma_{k-2-j}(\mathbf{z}) - \sigma_{k-3-j}(\mathbf{z})\sigma_{k-2-i}(\mathbf{z})$, for $1 \leq i < j \leq k-3$ are not divisible by $1 + z^m$.

Let $\sigma_i(\mathbf{z})$ and

$$\sigma_{k-3-i}(\mathbf{z})\sigma_{k-2-j}(\mathbf{z}) - \sigma_{k-3-j}(\mathbf{z})\sigma_{k-2-i}(\mathbf{z})$$

be polynomials in $\mathbb{F}_2[z]$. We have the maximum degree of the above polynomials is $(k+3)(k-4)$, which is less than m. Therefore, the polynomials $\sigma_i(\mathbf{z})$ for $0 \leq i \leq k-3$ and

$$\sigma_{k-3-i}(\mathbf{z})\sigma_{k-2-j}(\mathbf{z}) - \sigma_{k-3-j}(\mathbf{z})\sigma_{k-2-i}(\mathbf{z})$$

for $1 \leq i < j \leq k-3$ are not divisible by $1 + z^m$.

Case $\ell = k-1$. Let $\mathbf{z}$ denote the vector $(z^{i_1}, z^{i_2}, \ldots, z^{i_{k-1}})$. The $(k-1) \times (k-1)$ determinant in (A.1) is equal to $\sigma_i(\mathbf{z}) \cdot \det \mathbf{V}(\mathbf{z})$, with $1 \leq i \leq k-2$.

It is sufficient to prove that $\sigma_i(\mathbf{z})$ for $1 \leq i \leq k-2$ are not divisible by $1 + z^m$. As the maximum degree of the polynomials $\sigma_i(\mathbf{z})$ is $\frac{(k+1)(k-2)}{2} < m$, we have $\sigma_i(\mathbf{z})$ is not divisible by $1 + z^m$.

Case $\ell = k$. The $k \times k$ determinant in (A.1) is the determinant

of a Vandermonde matrix. It cannot be divisible by $1 + z^m$ as $i_1, i_2, \ldots, i_k$ are all less than m.

Case $k-3 \geq \ell \geq 4$. Consider the other cases of $k-3 \geq \ell \geq 4$. The $\ell \times \ell$ determinant in (A.1) is a polynomial over $\mathbb{F}_2[z]$. Note that the polynomial $h(z)$ is irreducible and thus is not divisible by any polynomial over $\mathbb{F}_2[z]$ with a degree larger than zero but smaller than $m-1$. Therefore, if the maximum degree of the $\ell \times \ell$ determinant in (A.1) is less than $m-1$, the degree of $h(z)$, then the square matrix $\mathbf{G}_\ell$ is non-singular over $\mathbb{F}_2[z]/(h(z))$. Therefore, the main problem is to calculate the maximum degree of the $\ell \times \ell$ determinant.

Note that the maximum degree of the $(k-3) \times (k-3)$ determinant is always larger than the maximum degree of the $\ell \times \ell$ for $k-4 \geq \ell \geq 4$. Therefore in the following, we evaluate the maximum degree of the $(k-3) \times (k-3)$ determinant.

The $(k-3) \times (k-3)$ determinant can be re-written as

$$\begin{vmatrix} 1 & z^{i_1 j_1} & \ldots & z^{i_1 j_{k-4}} \\ 1 & z^{i_2 j_1} & \ldots & z^{i_2 j_{k-4}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z^{i_{k-3} j_1} & \ldots & z^{i_{k-3} j_{k-4}} \end{vmatrix}, \tag{A.3}$$

where $0 \leq i_1 < i_2 < \cdots < i_{k-3} \leq k-1$ and $1 \leq j_1 < j_2 < \cdots < j_{k-4} \leq r-1$. In the following, we use induction to prove that the

degree of polynomial in (A.3) is

$$D_{max} = i_2 j_1 + i_3 j_2 + \cdots + i_{k-3} j_{k-4}. \tag{A.4}$$

Obviously, (A.4) holds if $k = 5$. In such case, the polynomial in (A.3) is $z^{i_2 j_1} - z^{i_1 j_1}$, so the degree is $i_2 j_1$. We assume that for $k - 3 = N \geq 2$ the equation (A.4) holds. Now we prove that (A.4) is true for $k - 3 = N + 1$.

By the Leibniz formula, the $(k-3) \times (k-3)$ determinant is the sum of all possible products of elements in different rows and columns. Now we show that the largest power of z for these products is by the elements in the diagonal of the $(k-3) \times (k-3)$ matrix. For the product that includes the element $x^{i_{N+1}+j_N}$, the other elements are from the first N row and the first N columns. Thus, the product from the diagonal elements has the largest power, by the induction assumption. Then, the product is calculated as $z^{i_2 j_1 + i_3 j_2 + \cdots + i_{N+1} j_N}$.

If a product does not include the element $z^{i_{N+1}+j_N}$, it must include another element in the final row ($N+1$-th row). Without loss of generality, we assume that it includes $z^{i_{N+1}+j_\ell}$, for $\ell < N$. By removing the $N+1$-th row and $\ell+1$-th column, we get an $N \times N$ matrix. By the induction assumption, the product with the largest power for this matrix is composed of its diagonal elements.

Then the product including $z^{i_N+j_\ell}$ with the largest power is

$$z^{i_2j_1+i_3j_2+\cdots+i_\ell j_{\ell-1}+i_{\ell+1}j_{\ell+1}+\cdots+i_Nj_N+i_{N+1}j_\ell}.$$

Now we show that

$$i_2j_1+i_3j_2+\cdots+i_{N+1}j_N > i_2j_1+\cdots+i_\ell j_{\ell-1}+i_{\ell+1}j_{\ell+1}+\cdots+i_Nj_N+i_{N+1}j_\ell.$$

This is equivalently to show $i_{\ell+1}(j_{\ell+1}-j_\ell)+i_{\ell+2}(j_{\ell+2}-j_{\ell+1})+\cdots+i_N(j_N-j_{N-1}) < i_{N+1}(j_N-j_\ell)$. Since $i_{\ell+1} < i_{\ell+2} < \cdots < i_N < i_{N+1}$, and $j_N-j_\ell = (j_N-j_{N-1})+(j_{N-1}-j_{N-2})+\cdots+(j_{\ell+1}-j_\ell)$, so the above inequality holds. Thus it is true for $k-3 = N+1$. Hence, the product of diagonal elements of the $(k-3)\times(k-3)$ determinant has the largest power.

Clearly, D_{max} have the maximum if all $i_{\ell+1}$ and j_ℓ have the maximum values ($\ell = 1, 2, \cdots, k-4$). Thus, when $i_{k-3} = k-1, i_{k-4} = k-2, \cdots, i_2 = 4$ and $j_{k-4} = r-1, j_{k-5} = r-2, \cdots, j_1 = r-k+4$, the maximum degree is achieved, which is

$$\sum_{i=1}^{k-4}(k-i)(r-i) = (k-4)(kr-\frac{(k-3)(k+3r+7)}{6}).$$

Similarly, the maximum degree of the determinant is

$$(r-4)(kr-\frac{(r-3)(3k+r+7)}{6}).$$

when $k \geq r$.

As

$$(\min\{k,r\}-4)(kr+\frac{(\min\{k,r\}-3)(\min\{k,r\}+3\max\{k,r\}+7)}{6})$$

is no less than the value of $9k-13$, k^2-k-12 and $\frac{1}{2}(k^2-k-2)$, for $r \geq 9$ and $k \geq 5$. From the above discussion, we have when $m-1$ is larger than

$$(\min\{k,r\}-4)(kr+\frac{(\min\{k,r\}-3)(\min\{k,r\}+3\max\{k,r\}+7)}{6}),$$

all the $k \times k$ submatrices $\mathbf{G}_k$ are non-singular over $\mathbb{F}_2[z]/(h(z))$. The proposed MDS array code thus satisfies MDS property for $r \geq 9$.

A.2 Proof of Theorem 9

As the determinant of matrix in (5.3) is $\mathcal{C}_m$-invertible, we can employ Algorithm 1 to decode the ν data polynomials of BASIC codes. In the process of solving $\mathbf{y}_{\ell+1}$ from $L_\nu^{(\ell)}\mathbf{y}_{\ell+1} = \mathbf{y}_\ell$ for $\ell = 1, 2, \ldots, \nu$ in Algorithm 1, we need to calculate $\frac{\nu(\nu+1)}{2}$ equations like solving $s(z)$ from $(1+z^b)s(z) = c(z)$, and $\frac{\nu(\nu+1)}{2}$ additions. So, the computation of solving $\mathbf{y}_{\ell+1}$ for $\ell = 1, 2, \ldots, \nu$ is no larger than $\frac{5\nu(\nu+1)m}{4}$ XORs by Lemma 8. In the process of solving $\mathbf{y}_\ell$ from $U_\nu^{(\ell)}\mathbf{y}_\ell = \mathbf{y}_{\ell+1}$ in Algorithm 1, we can count that solving $\mathbf{y}_\ell$ for $\ell = \nu, \nu-1, \ldots, 1$ takes $\frac{\nu(\nu+1)}{2}$ additions, i.e., $\frac{\nu(\nu+1)m}{2}$ XORs.

Therefore, the total computation of Algorithm 1 with operations over $\mathcal{R}_m$ is at most $\frac{7}{4}\nu(\nu+1)m$ XORs.

A.3 Proof of Lemma 17

Note that the ring $\mathcal{C}_m$ is isomorphic to $\mathbb{F}_2(z)/h(z)$ with the Chinese remainder theorem. Furthermore, the ring $\mathbb{F}_2(z)/h(z)$ is isomorphic to the direct sum of the finite fields $\mathbb{F}_2(z)/f_\ell(z)$, for $\ell = 1, 2, \ldots, L$. As θ_ℓ is injective, we have that the set

$$\{\theta_\ell(a(z)) : a(z) \in \mathcal{S}\}$$

is a subset of the field $\mathbb{F}_2(z)/f_\ell(z)$ with cardinality $|\mathcal{S}|$, $\ell = 1, 2, \ldots, L$.

For $\ell = 1, 2, \ldots, L$, let $g_\ell(X_1, X_2, \ldots, X_N)$ be the polynomial of $g(X_1, X_2, \ldots, X_N)$ with coefficients of $g(X_1, X_2, \ldots, X_N)$ reduced modulo $f_\ell(z)$. Let $\Re$ be the set of $\mathcal{C}_m$-roots of the polynomial $g(X_1, X_2, \ldots, X_N)$ in $\mathcal{S}^N$ and let $\Re_\ell$ be a subset of $\mathcal{S}^N$ such that

$$g_\ell(\theta_\ell(a_1(z)), \theta_\ell(a_2(z)), \ldots, \theta_\ell(a_N(z))) = 0$$

in the field $\mathbb{F}_2[z]/f_\ell(z)$, $\forall(a_1(z), a_2(z), \ldots, a_N(z)) \in \Re_\ell$ and $\ell =$

$1, 2, \ldots, L$. We have

$$\begin{aligned} |\Re| &= |\Re_1 \bigcup \Re_2 \bigcup \cdots \bigcup \Re_L| \\ &\leq \sum_{\ell=1}^{L} |\Re_\ell| \\ &\leq L \cdot e|\mathcal{S}|^{N-1}, \end{aligned}$$

where in the last inequality, we use the result of Lemma 15.

A.4 Proof of Theorem 18

The proof is basically the same as in [6]. The encoding coefficients in the global encoding vector when we initialize the storage system are polynomials in $\mathcal{R}_m$. The local encoding coefficients in each repair process are chose from polynomials in the set $\mathcal{S}$ such that a collection of sets of k packets are decodable. For each set of k packets in this collection, we need to guarantee that the decodability by invoking Theorem 1 in the previous section. In the application of Lemma 17, the requirement about the set $\mathcal{S}$, which is stated in Lemma 17 should be satisfied.

In the proof of the existence of functional repair RGC over a finite field in [6], the local encoding coefficients are chosen in the field. They evaluate a set of polynomials to be non-zero over the

finite field, and show that if the field size is larger than the value given in (6.2), then there exists a regenerating code defined in the field. For functional repair BASIC regenerating codes, we need to evaluate the same set of polynomials to be non-zero simultaneously over L fields rather than one field, and the local coefficients are limited in the set $\mathcal{S}$. Thus, the value of $|\mathcal{S}|$ should be greater than the value in (6.4).

A.5 Proof of Lemma 20

Note that both the matrices $\mathbf{\Psi}$ and Φ are Vandermonde matrices. Therefore, we only need to consider the case of the matrix $\mathbf{\Psi}$. Consider the matrix $\mathbf{\Psi}$, for any d distinct rows indexed by $i_1, i_2, \cdots, i_d$ between 1 to n, the corresponding encoding vectors $\boldsymbol{\psi}_{i_1}^t, \boldsymbol{\psi}_{i_2}^t, \cdots, \boldsymbol{\psi}_{i_d}^t$ form a non-singular $d \times d$ Vandermonde matrix. So the determinant is

$$\prod_{j<\ell}(z^{\ell-1} + z^{j-1}), \tag{A.5}$$

where $j, \ell \in \{i_1, i_2, \cdots, i_d\}$.

Let $f_1(z)f_2(z)\cdots f_L(z)$ be the prime factorization of the check polynomial $h(z)$ over $\mathbb{F}_2$. Suppose that the above determinant is $\mathcal{C}_m$-invertible, then by Theorem 2, $z^{j-1} + z^{\ell-1}$ is a unit in $\mathbb{F}_2[z]/f_i(z)$,

$\forall i \in \{1, 2, \cdots, L\}$ and $1 \leq j < \ell \leq n$. This is equivalent to the condition that $1 + z^a$ is a unit in $\mathbb{F}_2[z]/f_i(z)$, i.e., z^a is not congruent to $1 \mod f_i(z)$, $\forall i \in \{1, 2, \cdots, L\}$ and $1 \leq a \leq n-1$. Note that $f_i(z)$ is a factor of $1 + z^m$. If $1 + z^a$ is divisible by $f_i(z)$, then a must be a divisor of m. If $n-1$ is strictly less than all divisors of m which are not equal to 1, we thus have that $1 + z^a$ is not divisible by $f_i(z)$, $\forall i \in \{1, 2, \cdots, L\}$ and $1 \leq a \leq n-1$, and then the determinant in (A.5) is $\mathcal{C}_m$-invertible.

A.6 Proof of Theorem 25

Encode. To each piece of data, we first append the parity-check bits after each $m-1$ bits to obtain B codewords in $\mathcal{C}_m$. The calculation of the B parity-check bits in one piece of data requires $B(m-2)$ XOR operations. There are n storage nodes and each node stores $\kappa\alpha$ coded packets, with each coded packet being a $\mathcal{R}_m$-linear combination of the B source packets. The complexity of computing one coded packet is directly proportional to the number of terms in the coefficients, and in the worst case, there are $(m-1)/2$ terms in each of them (see the last paragraph of Subsection 8.1). The computational complexity of calculating one coded packet is thus

at most $Bm(m-1)/2$ XOR operations. Hence, the total number of XORs in encoding is $\kappa B(m-2) + \kappa n\alpha Bm(m-1)/2$. The normalized computational complexity of encoding is $(\kappa B(m-2) + \kappa n\alpha Bm(m-1)/2)/(\kappa B(m-1)) \approx n\alpha m/2$.

Repair. Each of the helping node generates $\kappa\beta$ coded packets, with each coded packet by a $\mathcal{R}_m$-linear combination of α packets in its memory. As the local encoding coefficients are polynomials in $\mathcal{S}$, i.e., the polynomials with non-zero terms are less than or equal to $(m-1)/2$. The total number of XORs in generating one packet to be sent to the new node is at most $\alpha m(m-1)/2$. The total number of bit operations from the transmitting side is at most $\kappa d\beta\alpha m(m-1)/2$.

The new node generates $\kappa\alpha$ coded packets. Each of them is obtained by combining the $d\beta$ received packets. The required number of XORs is at most $\kappa\alpha d\beta m(m-1)/2$. The normalized computational complexity of the repair of a failed node is at most $(\kappa\alpha d\beta m(m-1))/(\kappa B(m-1)) \approx \frac{d\beta m}{k}$.

Decode. A data collector recovers each piece of the data file by linearly combining $k\alpha$ coded packets. The coefficients in the linear combination are polynomial in $\mathcal{R}_m$ and are obtained by solving some system of linear equations. We ignore the computational complexity in calculating these coefficients as it is negligible

asymptotically when κ is large. The number of XOR's in recovering one source packet is therefore at most $k\alpha m(m-1)/2$. The normalized computational complexity of decoding the data file is at most $(\kappa Bk\alpha m(m-1)/2)/(\kappa B(m-1)) = \frac{Bm}{2}$.

A.7 Proof of Theorem 26

Recall that we assume the data file contains $B(m-1)$ bits, where B is given in (7.5). First, we generate B source packets by encoding each group of $m-1$ bits to a codeword of the binary cyclic code $\mathcal{C}_m$, which takes $B(m-2)$ XORs. Each node stores $\alpha = d$ coded packets, where each coded packet is a linear combination of α source packets. As the encoding coefficients are powers of z, we have that each coded packet is computed by adding α shifted versions of source packets, which takes αm XORs. Therefore, the encoding complexity is $B(m-2) + n\alpha^2 m$ and the encoding complexity normalized by the file size is $\frac{2n\alpha^2}{k(2d-k+1)}$.

In the repair process, each of the d helper nodes sends one coded packet by adding the d shifted packets stored in the helper node. The total number of XORs of adding the d packets in each node is dm. The new node needs to compute a $d \times d$ linear system

with the encoding matrix being a Vandermonde matrix, which can be solved using the proposed Algorithm 1, with $\frac{7}{4}d(d-1)m$ XORs involved. The normalized repair complexity is $\frac{d^2m+3d(d-1)m}{B(m-1)} \approx \frac{5.5d^2}{k(2d-k+1)}$.

Algorithm 2 Solving the source packets of $\mathbf{S}$ for BASIC-PM MBR codes

Input: The $k \times k$ symmetric matrix $\Phi_k\mathbf{S}$, where Φ_k is a $k \times k$ Vandermonde matrix and $\mathbf{S}$ is a $k \times k$ symmetric matrix.

1: **for** $i = 1, 2, \ldots, k-1$ **do**

2: Solve $k-i+1$ source packets in the ith column of $\Phi_k\mathbf{S}$ by Algorithm 1. Subtract the first i known source packets from the first $k-i$ coded packets in the $i+1$-th column of $\Phi_k\mathbf{S}$.

The decoding complexity is composed by three parts. First one is the complexity of solving the packets in $\mathbf{T}$, and we denote the complexity as $N_{\mathbf{T}}$. The second part is the complexity of subtracting the known packets of $\mathbf{T}$ from the other coded packets, which is denoted as N_{sub}. The last one is the complexity of solving the packets in $\mathbf{S}$, and is denoted as $N_{\mathbf{S}}$.

For any k nodes $\ell_1, \ell_2, \ldots, \ell_k$, we can solve the $(d-k)k$ source packets in $\mathbf{T}$ by solving the $d-k$ Vandermonde systems with Algorithm 1. The complexity of the first part thus is $N_{\mathbf{T}} = \frac{7}{4}(d-k)k(k-1)m$. After subtracting the $k(d-k)$ source packets from the first k coded packets for each of the k nodes, and we obtain

the $k \times k$ symmetric matrix $\Phi_k \mathbf{S}$, where Φ_k is

$$\Phi_k = \begin{bmatrix} 1 & z^{\ell_1-1} & z^{2(\ell_1-1)} & \dots & z^{(k-1)(\ell_1-1)} \\ 1 & z^{\ell_2-1} & z^{2(\ell_2-1)} & \dots & z^{(k-1)(\ell_2-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z^{\ell_k-1} & z^{2(\ell_k-1)} & \dots & z^{(k-1)(\ell_k-1)} \end{bmatrix}.$$

Therefore $N_{sub} = k^2(d-k)(k+1)m/2$.

We can recursively solve the $k \times (k+1)/2$ source packets by Algorithm 1, and the decoding logic is given in Algorithm 2. The computational complexity of calculating the $k \times (k+1)/2$ source packets in Algorithm 2 is

$$\begin{aligned} N_{\mathbf{S}} =& \sum_{i=1}^{k-1} \frac{7}{4} i(i+1)m + \sum_{i=1}^{k-1} i(k-i)m \\ =& \frac{1}{8}(k-1)k(2k-1)m + \frac{7}{8}(k-1)km + (k-1)k^2m/2. \end{aligned}$$

The normalized decoding complexity of BASIC-PM MBR codes is

$$\begin{aligned} & \frac{N_{\mathbf{T}} + N_{sub} + N_{\mathbf{S}}}{B(m-1)} \\ \approx & \frac{k(kd - k^2 + 4.5d - 3k)}{(2d-k+1)}. \end{aligned}$$

□ **End of chapter.**

Bibliography

[1] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43, 2003.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[3] Colossus, successor of Google File System. *Available: http://www.highlyscalablesystems.com/3202/colossus-successor-to-google-file-system-gfs/.*

[4] Facebook's erasure coded hadoop distributed file system (HDFS-RAID). *Available: https://github.com/facebookarchive/hadoop-20.*

[5] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems.

IEEE Transactions on Information Theory, 56(9):4539–4551, 2010.

[6] Y. Wu. Existence and construction of capacity-achieving network codes for distributed storage. *IEEE Journal on Selected Areas in Communications*, 28(2):277–288, 2010.

[7] P. Vingelmann, M. Pedersen, F. Fitzek, and J. Heide. Multimedia distribution using network coding on the iphone platform. In *Proc. of the 2010 ACM multimedia workshop on Mobile cloud media computing*, pages 3–6, 2010.

[8] N. B. Shah, K.V. Rashmi, P. V. Kumar, and K. Ramchandran. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff. *IEEE Transactions on Information Theory*, 58(3):1837–1852, 2012.

[9] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing elephants: Novel erasure codes for big data. In *Proc. of the 39th Int. Conf. on Very Large Data Bases*, Trento, August 2013.

[10] P. Piret and T. Krol. MDS convolutional codes. *IEEE Transactions on Information Theory*, 29(2):224–232, 1983.

[11] M. Blaum and R. M. Roth. New array codes for multiple phased burst correction. *IEEE Transactions on Information Theory*, 39(1):66–77, 1993.

[12] M. Xiao, T. Aulin, and M. Médard. Systematic binary deterministic rateless codes. In *Proc. IEEE Int. Symp. Inf. Theory*, pages 2066–2070, Toronto, July 2008.

[13] S. Jaggi, Y. Cassuto, and M. Effros. Low complexity encoding for network codes. In *Proc. IEEE Int. Symp. Inf. Theory*, pages 40–44, Seattle, July 2006.

[14] A. Keshavarz-Haddad and M. A. Khojastepour. Rotate-and-add coding: A novel algebraic network coding scheme. In *Proc. IEEE Information Theory Workshop*, 2010.

[15] M. A. Khojastepour, A. Keshavarz-Haddad, and A. S. Golsefidy. On capacity achieving property of rotational coding for acyclic deterministic wireless networks. In *Proc. of the 8th Int. Symp. on Modeling and Optimization in Mobile, Ad Hoc*

and Wireless Networks (WiOpt), pages 313–317, Avignon, June 2010.

[16] S.-Y. Li and Q. T. Sun. Network coding theory via commutative algebra. *IEEE Transactions on Information Theory*, 57(1):403–415, 2011.

[17] G. 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 (= 4) disk failures. *IEEE Transactions on Computers*, 54(12):1473–1483, 2005.

[18] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory*, 42(2):529–542, 1996.

[19] 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 Transactions on Information Theory*, 57(8):5227–5239, 2011.

[20] S. El Rouayheb and K. Ramchandran. Fractional repetition codes for repair in distributed storage systems. In *48th*

Annual Allerton Conference on Communication, Control, and Computing (Allerton), pages 1510–1517, 2010.

[21] O. Olmez and A. Ramamoorthy. Repairable replication-based storage systems using resolvable designs. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 1174–1181, 2012.

[22] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran. Dress codes for the storage cloud: Simple randomized constructions. In *Proc. IEEE Int. Symp. Inf. Theory*, pages 2338–2342, 2011.

[23] I. Tamo, Z. Wang, and J. Bruck. Zigzag codes: MDS array codes with optimal rebuilding. *IEEE Transactions on Information Theory*, 59(3):1597–1616, 2013.

[24] J. Kubiatowicz, D. Bindel, et al. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.

[25] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *NSDI*, volume 4, pages 337–350, 2004.

[26] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. 1995.

[27] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on Computers*, 44(2):192–202, 1995.

[28] 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. of the 3rd USENIX Conf. on File and Storage Technologies (FAST)*, pages 1–14, 2004.

[29] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, 2011.

[30] A.-M. Kermarrec, N. Le Scouarnec, and G. Straub. Repairing multiple failures with coordinated and adaptive regenerating codes. In *Proc. Int. Symp. on Network Coding (Netcod)*, pages 88–93, Beijing, July 2011.

[31] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz. Introduction to Redundant Arrays of Inexpensive Disks (RAID). In *Proc. IEEE COMPCON*, volume 89, pages 112–117, 1989.

[32] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57(7):889–901, 2008.

[33] Y. Wang, G. Li, and X. Zhong. Triple-Star: A coding scheme with optimal encoding complexity for tolerating triple disk failures in RAID. *International Journal of Innovative Computing, Information and Control*, 3:1731–1472, 2012.

[34] G. Feng, R. H. Deng, F. Bao, and J-C Shen. New efficient MDS array codes for RAID. Part I. Reed-Solomon-like codes for tolerating three disk failures. *IEEE Transactions on Computers*, 54(9):1071–1080, 2005.

[35] Y. Hu, P. C. Lee, and K. W. Shum. Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems. In *INFOCOM, 2013 Proceedings IEEE*, pages 2355–2363, 2013.

[36] K. W. Shum and Y. Hu. Functional-repair-by-transfer regenerating codes. In *Proc. IEEE Int. Symp. Inf. Theory*, pages 1192–1196, Cambridge, July 2012.

[37] N. B. Shah. On minimizing data-read and download for storage-node recovery. *IEEE Comm. Letters*, 17(5):964–967, May 2013.

[38] C. Tian. Rate region of the (4, 3, 3) exact-repair regenerating codes. In *Proc. IEEE Int. Symp. Inf. Theory*, pages 1426–1430, Istanbul, July 2013.

[39] B. Sasidharan, K. Senthoor, and P. V. Kumar. An improved outer bound on the storage-repair-bandwidth tradeoff of exact-repair regenerating codes. In *Proc. IEEE Int. Symp. Inf. Theory*, pages 2430–2434, Honolulu, July 2014.

[40] F. J. MacWilliams and N. J. A. Sloane. *The Theory Of Error-correcting Codes*. Elsevier science publishers, 1977.

[41] C. L. Chen, W. W. Peterson, and E. J. Weldon Jr. Some results on quasi-cyclic codes. *Information and Control*, 15(5):407–423, November 1969.

[42] S. Ling and p. Solé. On the algebraic structure of quasi-cyclic codes I: Finite fields. *IEEE Transactions on Information Theory*, 47(7):2751–2760, November 2001.

[43] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambrdige, 1985.

[44] G. Castagnoli, J. L. Massey, P. Schoeller, and N. Von Seemann. On repeated-root cyclic codes. *IEEE Transactions on Information Theory*, 37(2):337–342, 1991.

[45] M. R. Murty. Artin's conjecture for primitive roots. *Math. Intelligencer*, 10(4):59–67, 1988.

[46] S. L. Yang. On the LU factorization of the Vandermonde matrix. *Discrete applied mathematics*, 146(1):102–105, 2005.

[47] P. Subedi and X. He. A comprehensive analysis of XOR-based erasure codes tolerating 3 or more concurrent failures. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 1528–1537, 2013.

[48] M. Blaum. A family of MDS array codes with minimal number of encoding operations. In *IEEE Int. Symp. on Inf. Theory*, pages 2784–2788, 2006.

[49] W. Werner. Polynomial interpolation: Lagrange versus Newton. *Mathematics of computation*, pages 205–217, 1984.

[50] S. Jukna. *Extremal combinatorics – with applications in computer science*. Springer-Verlag, Berlin, 2nd edition, 2011.

[51] A. W. Marshall and I. Olkin. Theory of majorization and its applications. *Academic, New York*, 1979.

[52] H. Cohen and G. Frey, editors. *Handbook Of Elliptic And Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2006.

[53] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in cryptology*, pages 95–107. Springer, 1994.

[54] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. In *Soviet Physics Doklady*, volume 7, pages 595–596, 1963.

[55] A. Weimerskirch and C. Paar. Generalizations of the Karatsuba algorithm for efficient implementations. *Available: https://eprint.iacr.org/2006/224.pdf*, 2006.

[56] R. Crandall and C. Pomerance. *Prime numbers: a computational perspective*. New York, 2001.

[57] J. M. Pollard. The fast Fourier transform in a finite field. *Mathematics of computation*, 25(114):365–374, 1971.

[58] S. Gao and T. Mateer. Additive fast Fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010.

[59] J. H. Silverman. Fast multiplication in finite fields GF (2^n). In *Cryptographic Hardware and Embedded Systems*, pages 122–134. Springer, 1999.

[60] J. S. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A library in C/C++ facilitating erasure coding for storage applications-version 1.2. *University of Tennessee, Tech. Rep. CS-08-627*, 23, 2008.