BASIC Codes: Low-Complexity Regenerating Codes for Distributed Storage Systems

Hanxu Hou, Kenneth W. Shum, Minghua Chen and Hui Li*

***Abstract*— In distributed storage systems, regenerating codes can achieve the optimal tradeoff between storage capacity and repair bandwidth. However, a critical drawback of existing regenerating codes in general is the high coding and repair complexity, since the coding and repair processes involve expensive multiplication operations in finite field. In this paper, we present a design framework of regenerating codes which employ binary addition and bit-wise cyclic shift as the elemental operations, named BASIC regenerating codes. The proposed BASIC regenerating codes can be regarded as a concatenation coding scheme with the outer code being a binary parity-check code, and the inner code being 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 tradeoff curve between the storage and repair bandwidth asymptotically of functional-repair regenerating codes with less computational complexity. Furthermore, we demonstrate that the existing exact-repair product-matrix construction of regenerating codes can be modified to exact-repair BASIC product-matrix regenerating codes with much less encoding, repair and decoding complexity from theoretical analysis, and with less encoding time, repair time and decoding time from the implementation results.**

***Index Terms*—Regenerating codes, distributed storage systems, low complexity, binary parity-check code.**

I. Introduction

Distributed storage systems achieve high reliability by storing the data redundantly in many connected unreliable storage nodes. Maximum-distance-separable (MDS) codes such as Reed-Solomon (RS) codes is one common approach to provide redundancy. With an (n, k) RS code, a data file is encoded and stored across n nodes such that a data collector can retrieve the original data file from any k nodes.

Upon the failure of a node, we need to regenerate the data stored in the failed node in order to maintain the same level of reliability. Dimakis *et al.* in [2] formulated the repair problem and proposed the *regenerating codes* (RGC) with the aim of efficient repair of the failed node. In the pioneer work in [2], a data file with B symbols over the finite field $\mathbb{F}_{2^w}$ is encoded into $n\alpha$ symbols and distributed to n nodes, with each node storing α symbols such that the original data file can be recovered from any k nodes. When a node is failed, a new node is created and downloads β symbols from each of d surviving nodes. The total number of symbols downloaded from the surviving nodes during the repair process is coined the *repair bandwidth*.

Two main versions of repair are introduced in [2]: *exact repair* and *functional repair*. In exact repair, the symbols stored in the failed node are exactly reproduced in the new node. In functional repair, the requirement is relaxed: the new node may contain different symbols from that in the failed node as long as the repaired system maintains the (n, k) *recovery property* that any k nodes are sufficient in decoding the original data file. It is shown in [2] 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 tradeoff between the amount of storage per node and the repair bandwidth is established as follows,

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

If we fix the parameter B, there is a tradeoff between storage α and repair bandwidth β. The two extreme points in this tradeoff 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 [3]–[6], all of which address either the MBR case or the MSR case. The paper [7] 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 product-matrix framework. The concept of uncoded repair was originally introduced in [6]. 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 [3], [6]. It is shown in [5] that it is not possible to construct the uncoded repair MBR codes when $d \neq n-1$ for exact repair. At the MSR point, uncoded repair RGC for

This paper was presented in part in [1] at the IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 2014. This work was partially supported by the National Basic Research Program of China (No.2012CB315904, No. 2013CB336700), SZJCYJ20150331100723974, SZJCYJ20140417144423192, S2013020012822, by the University Grants Committee of the Hong Kong Special Administrative Region, China (Area of Excellence Grant Project No. AoE/E-02/08 and General Research Fund No. 14209115, No. CUHK14209515).

* Corresponding author.

functional repair is discussed in [8], [9]. However, the code parameters considered in [8], [9] are restricted to $k = 2$ and $k = n - 2$.

Recently, zigzag code [4] was constructed on the MSR point to achieve the optimal exact repair. The code parameters considered in [4] are relaxed to $n \geq k + 2$ and $d = n - 1$, at a cost of a very high level of sub-symbolization. The reason is that zigzag code is a vector-linear code, while the codes in [7]–[9] are scalar-linear codes. Although the problem of determining the rate region for exact-repair RGC in general remains open, some recent results can be found in [10], [11].

In [12], existence of linear network codes achieving all points on the fundamental tradeoff curve for functional-repair RGC is shown. The construction relies on arithmetic of finite field, and as in the application of linear network code to single-source multi-cast problem in general, the underlying finite field must be sufficiently large. However, multiplication and division in finite field are costly to implement in software or hardware. In the literature of coding for disk arrays, the computational complexity is reduced by replacing finite field arithmetic by simple bit-wise operations. For example, in [13], MDS code with a convolutional code as alphabet is introduced by Piret and Krol. In [14], Blaum and Roth proposed a construction of array codes based on the ring of polynomials with binary coefficients modulo $1 + z + \cdots + z^{p-1}$ for some prime number p. Similar approach was considered by Xiao *et al.* in [15].

The objective of this paper is to introduce another class of RGC which enables coding and repair by XOR and bit-wise *cyclic-shift*. The new class of codes is called BASIC (Binary Addition and Shift Implementable Cyclic-convolutional) regenerating codes. The reduction on computational complexity is made possible by replacing the finite field multiplications in RGC by bit-wise cyclic-shifts, and replacing the base field by a ring with cyclic structure.

Similar methodology in reducing computational complexity in network coding problems can be found in [16]–[18]. In [16], "permute-and-add" linear network codes are proposed with local encoding matrix being a permutation matrix. For such network codes, the encoding operation is equivalent to first permuting the incoming symbols and then summing the permuted symbols. Although the authors in [17], [18] only considered the encoding process, the essential ideas behind "rotate-and-add" network codes and BASIC codes are the same. In more detail, "rotate-and-add" network codes first append one zero bit for each $m - 1$ bits to form a packet, shift the received packets and then sum them. BASIC codes first append the parity-check bit for each $m-1$ bits to form a packet, do some shifts for the formulated packets and then sum the shifted packets. We do not need to store the last bit for BASIC codes, as we can compute it, which is the summation of the first $m - 1$ bits, when necessary. However, all the m bits of a packet in "rotate-and-add" network codes should be stored or transmitted. More generally, network codes over rings are discussed in [19]. Compared with the existing low complexity network codes in [16]–[18], this paper mainly has three contributions, which are summarized as follows:

1) We propose a new framework of linear codes with the binary parity-check code as the alphabet, named BASIC codes.
2) We give a general construction of functional-repair BASIC-RGC and show that the presented functional-repair BASIC-RGC can achieve all the benefits of functional-repair RGC asymptotically with less complexity in coding and repair processes. As there is an additional 1 bit per $m-1$ bits in the storage and repair bandwidth, this is what "asymptotic" means. The constructed functional-repair BASIC-RGC are existential.
3) We show that the existing exact-repair RGC can be modified to exact-repair BASIC-RGC. An efficient decoding method with LU factorization of Vandermonde matrix is proposed to show that exact-repair BASIC-RGC have much less complexity in encoding, repair and decoding processes. Although in this paper we only give the conversion of the product-matrix construction in [7], all the constructed exact-repair RGC in [3], [4], [6], [7] can be converted to the exact-repair BASIC-RGC.

This paper is organized as follows. A motivating example that illustrates the main ideas is given in Section II. After reviewing some facts on binary cyclic codes in Section III, we propose the design framework of BASIC codes and show that we can operate arbitrarily close to the fundamental tradeoff curve between storage and repair bandwidth by functional-repair BASIC-RGC in Section IV. In Section V, we show how the exact-repair product-matrix RGC in [7] are adapted to exact-repair BASIC-RGC. In Section VI, we compare the computational complexity with functional-repair RGC over finite field. The computational complexity of exact-repair BASIC product-matrix RGC as well as the product-matrix RGC over finite field in [7] is also evaluated in Section VI. Some results of the implementation of BASIC product-matrix RGC and product-matrix RGC over finite field are shown in Section VII. The last section concludes this paper.

II. A Motivating Example

The following example of storage code illustrates the main ideas. Suppose that we want to store some information bits to four storage nodes, such that we can recover the information bits from any two nodes. The information bits are divided into groups of $4(m-1)$ bits, for some positive and *odd* integer m. Each group of $4(m - 1)$ bits is called a data chunk. As the data chunks are processed in the same manner, we focus on one data chunk.

We divide the $4(m - 1)$ *information bits* into four equal parts and represent each of them by $s_{i,0}, s_{i,1}, \cdots, s_{i,m-2}$, for $i = 1, 2, 3, 4$. We append the *parity-check bit*

$$s_{i,m-1} := \sum_{j=0}^{m-2} s_{i,j}$$

for the $m-1$ information bits $s_{i,0}, s_{i,1}, \cdots, s_{i,m-2}$, and denote

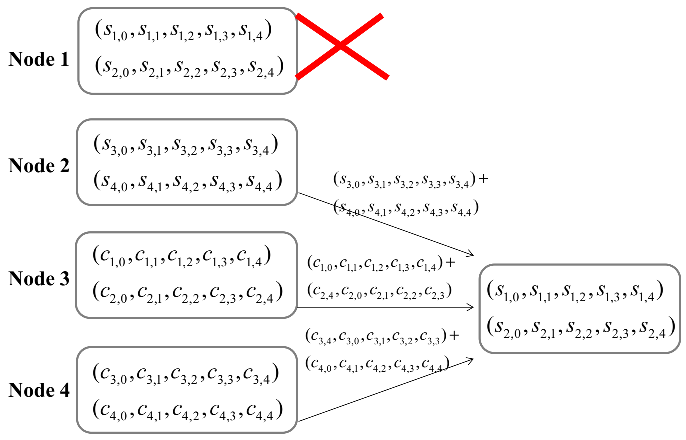
Node 1
$(s_{1,0},s_{1,1},s_{1,2},s_{1,3},s_{1,4})$
$(s_{2,0},s_{2,1},s_{2,2},s_{2,3},s_{2,4})$
Node 2
$(s_{3,0},s_{3,1},s_{3,2},s_{3,3},s_{3,4})$
$(s_{4,0},s_{4,1},s_{4,2},s_{4,3},s_{4,4})$
Node 3
$(c_{1,0},c_{1,1},c_{1,2},c_{1,3},c_{1,4})$
$(c_{2,0},c_{2,1},c_{2,2},c_{2,3},c_{2,4})$
Node 4
$(c_{3,0},c_{3,1},c_{3,2},c_{3,3},c_{3,4})$
$(c_{4,0},c_{4,1},c_{4,2},c_{4,3},c_{4,4})$
$(s_{3,0},s_{3,1},s_{3,2},s_{3,3},s_{3,4})+$
$(s_{4,0},s_{4,1},s_{4,2},s_{4,3},s_{4,4})$
$(c_{1,0},c_{1,1},c_{1,2},c_{1,3},c_{1,4})+$
$(c_{2,4},c_{2,0},c_{2,1},c_{2,2},c_{2,3})$
$(c_{3,4},c_{3,0},c_{3,1},c_{3,2},c_{3,3})+$
$(c_{4,0},c_{4,1},c_{4,2},c_{4,3},c_{4,4})$
$(s_{1,0},s_{1,1},s_{1,2},s_{1,3},s_{1,4})$
$(s_{2,0},s_{2,1},s_{2,2},s_{2,3},s_{2,4})$

Fig. 1: An example of storage code for four nodes. When node 1 fails, the bits sent to the new node are shown as the labels of the edges.

them as the vector

$$\mathbf{s}_i := (s_{i,0}, s_{i,1}, \cdots, s_{i,m-2}, s_{i,m-1}).$$

The summation $\mathbf{s}_i + \mathbf{s}_j$ of two vectors $\mathbf{s}_i$ and $\mathbf{s}_j$ is defined as

$$\mathbf{s}_i + \mathbf{s}_j := (s_{i,0} + s_{j,0}, s_{i,1} + s_{j,1}, \cdots, s_{i,m-1} + s_{j,m-1}).$$

In this storage code, we store $2m$ bits in each node. Nodes 1 and 2 are called *information nodes*, and each node stores the $2m$ bits of two vectors. The redundant bits are represented by

$$\mathbf{c}_i = (c_{i,0}, c_{i,1}, \cdots, c_{i,m-2}, c_{i,m-1})$$

for $i = 1, 2, 3, 4$, which are stored in two *redundant nodes*, nodes 3 and 4. Node 3 stores

$$c_{1,j} := s_{1,j} + s_{3,j} \text{ and } c_{2,j} := s_{2,j} + s_{4,j \oplus_m 1},$$

for $j = 0, 1, \dots, m-1$. The symbol "$\oplus_m$" in the above line stands for addition modulo m. For $j = 0, 1, \dots, m-1$, the redundant bits

$$c_{3,j} := s_{1,j} + s_{3,j\oplus_m 1} \text{ and } c_{4,j} := s_{2,j} + s_{4,j},$$

are stored in node 4. An example for $m = 5$ is illustrated in Fig. 1. We note that the redundant bits in nodes 3 and 4 are computed by either adding the bits of two vectors, or by adding the bits of one vector and a cyclically shifted version of bits of another vector.

We claim that we can recover all the information bits from any two nodes. From node 1 and node 2, we can obtain the information bits directly. We can verify that the information bits can be obtained from any one of the information nodes and any one of the redundant nodes. Finally, suppose that we want to decode the information bits from node 3 and node 4. In the following, we show how to decode the information bits of vectors $\mathbf{s}_1$ and $\mathbf{s}_3$ from redundant bits $c_{1,j}, c_{3,j}$ for $j = 0, 1, \dots, m-1$. The decoding method for information bits of vectors $\mathbf{s}_2$ and $\mathbf{s}_4$ is similar. We can recover $s_{3,0}$ by computing

$$\sum_{\ell=0}^{(m-3)/2} c_{1,2\ell\oplus_m 1} + c_{3,2\ell\oplus_m 1} = s_{3,1} + s_{3,2} + \cdots + s_{3,m-1} = s_{3,0}.$$

Once the value of $s_{3,0}$ is known, we can get $s_{1,0}$ by $c_{1,0} + s_{3,0} = s_{1,0}$, and get $s_{3,1}$ by $c_{3,0} + s_{1,0} = s_{3,1}$. The remaining information bits can be decoded iteratively. This proves the claim.

Suppose that node 1 fails, and we want to repair it by downloading one vector from each of the surviving nodes. The repair process of $m = 5$ is shown in Fig. 1. We define a right cyclic-shift of $\mathbf{s}_i$ as

$$z\mathbf{s}_i := (s_{i,m-1}, s_{i,0}, \cdots, s_{i,m-3}, s_{i,m-2}).$$

Node 2 sends the summation of two vectors $\mathbf{s}_3 + \mathbf{s}_4$ to the new node. Node 3 shifts the second vector to the right by one bit, adds to the first vector, and sends the resulting vector $\mathbf{s}_1 + z\mathbf{s}_2 + \mathbf{s}_3 + \mathbf{s}_4$ to the new node. Likewise, node 4 adds the second vector and a right cyclic-shift of the first vector, and sends the summation $z\mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3 + \mathbf{s}_4$ to the new node. The new node can obtain two vectors $\mathbf{s}_1 + z\mathbf{s}_2$ and $z\mathbf{s}_1 + \mathbf{s}_2$ by subtracting the vector $\mathbf{s}_3 + \mathbf{s}_4$ from the receiving two vectors of node 3 and node 4 respectively. It is sufficient to recover the vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ from

$$s_{1,0} + s_{2,4}, s_{1,1} + s_{2,0}, s_{1,2} + s_{2,1}, s_{1,3} + s_{2,2}, s_{1,4} + s_{2,3},$$
$$s_{1,0} + s_{2,1}, s_{1,1} + s_{2,2}, s_{1,2} + s_{2,3}, s_{1,3} + s_{2,4}, s_{1,4} + s_{2,0}.$$

We can recover $s_{2,3}$ by computing

$$\begin{aligned}&(s_{1,0} + s_{2,4}) + (s_{1,0} + s_{2,1}) + (s_{1,1} + s_{2,0}) + (s_{1,1} + s_{2,2})\\ =&s_{2,4} + s_{2,1} + s_{2,0} + s_{2,2}\\ =&s_{2,3}.\end{aligned}$$

The remaining bits of $\mathbf{s}_1$ and $z\mathbf{s}_2$ can be decoded iteratively. If node 2 fails, it can be repaired in a similar manner.

If a redundant node fails, suppose node 3 fails without loss of generality, we can alternately treat node 3 and node 1 as the information nodes, and treat nodes 2 and 4 as the redundant nodes. The bits of node 3 can also be repaired in an analogous manner.

The assumption that the parameter m to be an odd integer is essential. If m is an even integer and if we flip all the information bits $s_{i,j}$ from 1 to 0 or from 0 to 1, for $i = 1, 2, 3, 4$ and $j = 0, 1, \dots, m-2$, then the content of node 3 and node 4 will not change. The mapping from the information bits in nodes 1 and 2 to the redundancy bits in nodes 3 and 4 is a two-to-one map in this case. So there is no way to recover the information bits from nodes 3 and 4.

We remark that

1) If the last bit $s_{i,m-1}$ or $c_{i,m-1}$ is stored, then the storage-bandwidth tradeoff is not optimal, but the accessed/read bits are the same as the transmitted bits.
2) If the last bit $s_{i,m-1}$ or $c_{i,m-1}$ is not stored, then the storage-bandwidth is optimal, all the bits need to be read, and the transmitted bits need to be computed from them.

III. Mathematical Framework of BASIC Codes

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

A. Binary Cyclic Code

In this subsection we review some facts on binary cyclic codes [20, Chapters 7]. Let m be a positive odd number and let $\mathcal{R}_m$ be the ring

$$\mathcal{R}_m := \mathbb{F}_2[z]/(1+z^m). \tag{2}$$

The element of $\mathcal{R}_m$ will be referred to as *polynomial* in the sequel. The vector $(a_0, a_1, \dots, a_{m-1}) \in \mathbb{F}_2^m$ 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 codeword. A *binary cyclic code* of length m is a subset of $\mathcal{R}_m$ closed under addition and multiplication by z.

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

$$\mathcal{C}_m = \{a(z)(1+z) : a(z) \in \mathcal{R}_m\}. \tag{3}$$

The dimension of $\mathcal{C}_m$ over $\mathbb{F}_2$ is $m-1$, and the *check polynomial* of $\mathcal{C}_m$ is $h(z) := 1 + z + \cdots + z^{m-1}$. We can check that the multiplication of $h(z)$ and any polynomial in $\mathcal{C}_m$ is zero. For a polynomial $c(z) = \sum_{i=0}^{m-1} c_i z^i$ in $\mathcal{C}_m$, we call $c_0, c_1, \dots, c_{m-2}$ as the first $m-1$ bits of polynomial $c(z)$ and c_{m-1} as the parity-check bit of $c_0, c_1, \dots, c_{m-2}$, as we have $c_{m-1} = \sum_{i=0}^{m-2} c_i$.

B. Design Framework of BASIC Code

Let ν be a positive integer, *BASIC* code is defined as a subspace of $\mathcal{C}_m^\nu$, i.e., 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 such that $\kappa \le \nu$, the encoding of a BASIC is a mapping 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 $\mathbf{w} = (s_1(z), s_2(z), \dots, s_\kappa(z)) \in \mathcal{C}_m^\kappa$. The codeword in the BASIC code corresponding to the $(m-1)\kappa$ input bits is obtained by multiplying $\mathbf{wG}$.

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 (2) 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) is isomorphic to a subring of $\mathbb{F}_2\mathbb{Z}_m$. A BASIC code is a submodule of the free $\mathbb{F}_2\mathbb{Z}_m$-module $\mathcal{C}_m^n$. A BASIC code can be regarded as a quasi-cyclic code (See e.g. [21], [22]). Nonetheless, the quasi-cyclic codes considered in [21], [22] 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 paper, BASIC codes are considered a code of length n over the alphabet $\mathcal{C}_m$.

Example: The code in the previous section is an example of BASIC code with parameters $m=5$, $\kappa=4$ and $\nu=8$. The four data packets are $s_i(z) = \sum_{j=0}^4 s_{i,j} z^j$, for $i = 1,2,3,4$, and the generator matrix is

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

C. Erasure Decoding

A collection of κ coded packets is said to be an *information set*, or *decodable*, if we can recover the source packets from these κ coded packets. In this subsection, we give a necessary and sufficient condition for decodability. To this end, we introduce some notations. 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 $\mathcal{I} \subseteq \{1, 2, \dots, \nu\}$ 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 $\mathcal{I} \subseteq \{1, 2, \dots, \nu\}$ 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 $s_1(z), \dots, s_\kappa(z)$ be the data packets, and $p_1(z), \dots, p_\kappa(z)$ be the coded packets indexed by $\mathcal{I}$,

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

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

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

where $adj(\mathbf{G}_\mathcal{I})$ denotes the adjoint of $\mathbf{G}_\mathcal{I}$ [23, 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 $\mathcal{R}_m$ is $\mathcal{C}_m$-invertible. Let $f_1(z)$, $f_2(z)$, …, $f_L(z)$ 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 $f_1(z)$, $f_2(z), \dots, f_L(z)$ are the irreducible factors of the check polynomial $h(z)$. Let $a(z)$ be a polynomial in $\mathcal{R}_m$. The followings are equivalent:*

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* $\ell = 1, 2, \dots, L$.

Proof. (1) $\Leftrightarrow$ (2). Define $f_0(z)$ be the polynomial $1 + z$. By the Chinese remainder theorem, 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 that $a(z) \bmod h(z)$ is a unit in $\mathbb{F}_2[z]/(h(z))$, i.e., suppose that 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. $\square$

Corollary 3. *Consider a BASIC code with* $\kappa \times \nu$ *generator matrix* $\mathbf{G}$. *For any subset* $\mathcal{I} \subseteq \{1, 2, \dots, \nu\}$ *of size* κ *such that* $\mathcal{I}$ *corresponds to the packets residing in* κ *nodes, the coded packets indexed by* $\mathcal{I}$ *are 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 $\ell_0 \in \{1, 2, \dots, L\}$. 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 $\bar{\mathbf{a}} = (\bar{a}_1(z), \dots, \bar{a}_\kappa(z))$, 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 $j = 1, 2, \dots, \kappa$, 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 $(a_1(z), a_2(z), \dots, a_\kappa(z))\mathbf{G}_\mathcal{I}$ is the zero ν-tuple. The encoding map is not injective and therefore the coded packets indexed by $\mathcal{I}$ are not decodable. $\square$

Example (continued): The polynomial $1 + z^5$ can be factorized as a product of $f_0(z) = 1 + z$ and $f_1(z) = 1 + z + z^2 + z^3 + z^4$. We can check that the four coded packets in any two nodes are decodable. For instance, for nodes 3 and 4, the index set is $\mathcal{I} = \{5, 6, 7, 8\}$, the determinant $\det(\mathbf{G}_\mathcal{I}) = 1 + z^3$ is not divisible by $f_1(z)$. Indeed, $1 + z^3$ is $\mathcal{C}_5$-invertible because

$$(1 + z^3)(z^3 + z^4) = z + z^2 + z^3 + z^4 = 1 + h(z).$$

Some remarks on implementation are in order. In software implementation, we can implement a cyclic-shift by using 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.

A remark on multiplication in $\mathcal{R}_m$ is in order. 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 $(m - 1)/2$. So we will assume that the number of non-zero terms of $a(z)$ is less than or equal to $(m - 1)/2$, when we evaluate the computational complexity of BASIC codes.

IV. Functional-Repair BASIC Regenerating Codes

In the rest of this paper, we consider *BASIC regenerating codes* (BASIC-RGC), which is defined as a class of BASIC codes such that the parameters n, k, d, α, β achieve the optimal tradeoff curve in (1) or asymptotically. Exact-repair BASIC-RGC will be given in the next section. First, we review the general construction of functional-repair RGC over finite field.

A. Functional-Repair RGC over Finite Field

The data file is divided into B *data symbols* and stored across n nodes with each node storing 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 repair process, a new node is created and replaces the failed node, and connects 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 node 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. Then the new node generates α new symbols, with each new symbol created by doing a linear combination of the receiving $d\beta$ symbols. The process is termed as *repair process* and the total amount of $d\beta$ 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 of the failures, as long as the (n,k) recovery property that any k nodes are sufficient in decoding the original file should be maintained, after each repair.

A major result in the field of RGC is that the parameters of a regenerating code must necessarily satisfy the inequality in (1). The general construction of functional-repair RGC over finite field that achieve the optimal tradeoff in (1) is presented in [12] 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 [12] 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. [24, p. 224]).

Lemma 4 (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 $\mathbb{F}[X_1, X_2, \dots, X_N]$ 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 node join and leave. To ensure the (n,k) recovery property after each repair, the author in [12] 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.

For $i = 1, 2, \dots, k$, let τ_i be defined as $\tau_i := \min\{(d - i + 1)\beta, \alpha\}$, and for $i = k+1, k+2, \dots, n$, let $\tau_i = 0$. Define $\mathcal{H}$ as the set of vectors of length n, whose components are non-negative integers, which are majorized by the vector $(\tau_1, \tau_2, \dots, \tau_n)$. The main result in [12] is summarized in the following theorem.

Theorem 5. *Let $\mathbb{F}_{2^w}$ be a finite field whose size is greater than*

$$B \cdot \max\left\{ \binom{n\alpha}{B}, 2|\mathcal{H}| \right\}. \tag{4}$$

Then, there exists a functional-repair regenerating code defined in $\mathbb{F}_{2^w}$ that achieves the optimal tradeoff point in (1).

B. Functional-Repair BASIC-RGC

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 $s_1(z)$, $s_2(z), \dots, s_B(z) \in \mathcal{C}_m$ 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 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 finite field, is stated as follows. Each of the d helper nodes transmits 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.

We can prove this by modifying the argument in [12] on the existence of RGC over finite field, and invoking a Schwartz-Zippel lemma over a specific ring $\mathcal{C}_m$.

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

Lemma 6 (Schwartz-Zippel lemma over the ring $\mathcal{C}_m$)**.** *Suppose that $f_1(z), f_2(z), \dots, f_L(z)$ 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 $\forall \ell = 1, 2, \dots, L$, where $a(z)$ can assume any value in $\mathcal{S}$. Then the polynomial $g(X_1, X_2, \dots, X_N)$ 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 $g(X_1, X_2, \dots, X_N)$.

Proof. Note that the ring $\mathcal{C}_m$ is isomorphic to $\mathbb{F}_2(z)/h(z)$ by 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 $\ell = 1, 2, \dots, L$. 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 $|\mathcal{S}|$, $\ell = 1, 2, \dots, L$.

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

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

in the field $\mathbb{F}_2[z]/f_\ell(z)$, $\forall (a_1(z), a_2(z), \dots, a_N(z)) \in \Re_\ell$ 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 4. □

In [12], 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-RGC, 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 6 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 $\{\mathbb{F}_2(z)/f_1(z), \mathbb{F}_2(z)/f_2(z), \cdots, \mathbb{F}_2(z)/f_L(z)\}$. With these modification, the requirement on the cardinality of $\mathcal{S}$ is stated in the next theorem.

Theorem 7. *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 6 is injective, $\forall \ell = 1, 2, \ldots, L$, and (ii) if $|\mathcal{S}|$ is larger than*

$$L \cdot B \cdot \max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}, \tag{5}$$

then there exists a functional-repair BASIC-RGC, 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. The proof is essentially the same as in [12]. The encoding coefficients in global encoding vector when we initialize the storage system are polynomials in $\mathcal{R}_m$. While the local encoding coefficients in each repair process are 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 6, the requirement about the set $\mathcal{S}$, which is stated in Lemma 6 should be satisfied.

In the proof of the existence of functional-repair RGC over a finite field in [12], 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 (4), then there exists a regenerating code defined in the field. For functional-repair BASIC-RGC, 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 (5). □

Theorem 7 says that when the cardinality of $\mathcal{S}$ is larger than (5), the proposed BASIC-RGC can achieve all the points on the optimal tradeoff curve between storage and repair bandwidth asymptotically. Note that the coding scheme proposed in this paper 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-RGC presented in this paper and other RGC in the literature is that, the packets in BASIC-RGC assume values 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 2 is primitive in $\mathbb{F}_m$. 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 [25]. In this case, we can let the set $\mathcal{S}$ to be the polynomials in $\mathcal{R}_m$ with non-zero term 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 7.

Corollary 8. *Let m be a prime number such that 2 is primitive in $\mathbb{F}_m$. There exists a functional-repair BASIC-RGC for a file of size*

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

if

$$m > \log_2 \left(B \cdot \max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}\right) + 1. \tag{6}$$

Note that there is one additional bit per $m-1$ bits, because we store the parity-check bit. However, the parity-check bit is not necessary to be stored in practical system, as pointed out in Section II. So, there are two types in BASIC-RGC, first one is with the last bit being stored that can achieve the optimal trade-off asymptotically, another is with the last bit not being stored that has more computation and can achieve the optimal trade-off. For the second type of BASIC-RGC, we can say that functional-repair BASIC-RGC can exactly achieve the optimal trade-off curve in (1). We choose the first type for functional-repair BASIC-RGC only for the ease of presentation, there is no essential difference for the two types. While for exact-repair BASIC-RGC, we do not store the parity-check bit, but we need to compute it in the repair process and decoding process.

V. Exact-Repair BASIC Regenerating Codes

In exact-repair RGC, a failed node is replaced by a new node that stores exactly the same data as was stored in the failed node. Constructing exact-repair RGC is more difficult than constructing functional-repair RGC, and all the constructions in [3]–[7] 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 [7]. The construction is of a product-matrix nature that is shown to significantly simplify operation of the distributed storage network.

In this section, we first briefly describe the product-matrix construction of exact-repair RGC. Then, we give the conversion of product-matrix RGC in [7] to BASIC product-matrix RGC.

A. Product-Matrix Construction of Regenerating Codes

As the product-matrix construction is based on a finite field, throughout this subsection, we consider all symbols to belong to $\mathbb{F}_{2^w}$. 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 $\mathbb{F}_{2^w}$ and are independent of the message symbols. The 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 $\mathbf{\Psi M}$.

The data collector can obtain $k\alpha$ symbols from any k storage nodes. There is a requirement when we choose the value of the encoding matrix $\mathbf{\Psi}$, to maintain the (n, k) recovery property.

Assume node f fails, a new node replacing the failed node connects to any 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 thus receives the product matrix $\mathbf{\Psi}_{repair}\mathbf{M}\boldsymbol{\psi}_f$, where $\mathbf{\Psi}_{repair}$ is the submatrix of $\mathbf{\Psi}$ consisting of the encoding vectors of the d helper nodes. From this it turns out that we can recover the failed symbols exactly, if the matrix $\mathbf{\Psi}_{repair}$ is invertible and the message matrix $\mathbf{M}$ satisfies some properties.

B. Product-Matrix Construction of BASIC-RGC

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 RGC.

Let $s_1(z), s_2(z), \ldots, s_B(z)$ be the B source packets, which are codewords of the binary parity-check code $\mathcal{C}_m$. The entries of the encoding matrix $\mathbf{\Psi}$ are fixed polynomials in $\mathcal{R}_m$ and independent of the source packets. The entries of $\mathbf{M}$ are the source packets. Unlike functional-repair BASIC-RGC, we do not store the parity-check bits for BASIC product-matrix RGC. Therefore, our BASIC product-matrix RGC codes can achieve the optimal MSR point and MBR point, not asymptotically.

1) BASIC Product-Matrix MSR Code: In the following, we construct the BASIC-PM (product-matrix) MSR code for $d = 2k-2$. As in [7], the construction can be extended naturally to $d \geq 2k-2$, but we will only discuss the primary 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 $\mathcal{C}_m$ by appending a parity-check bit for each part. Divide each of the B source packets into two equal groups. For each group, create a $(k-1) \times (k-1)$ 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 a $n \times d$ Vandermonde matrix, with the i-th row defined as

$$\boldsymbol{\psi}_i^t := \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & \cdots & z^{(d-1)(i-1)} \end{bmatrix}, \qquad (7)$$

for $i = 1, 2, \ldots, n$. The i-th node stores the first $m-1$ bits of each $\alpha = k-1$ packets in $\boldsymbol{\psi}_i^t \mathbf{M}$.

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

$$\phi_i^t := \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & \cdots & z^{(\alpha-1)(i-1)} \end{bmatrix}, \qquad (8)$$

for $i = 1, 2, \ldots, n$, and let Λ be the $n \times n$ diagonal matrix with diagonal elements equal to $1, z^{\alpha}, \ldots, z^{\alpha(n-1)}$. 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 9. *Let $\mathbf{\Psi}$ be the encoding matrix, which is composed by the encoding vector given in (7). 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. Note that both the matrices $\mathbf{\Psi}$ and Φ are Vandermonde matrices. Therefore, we only need to consider the matrix $\mathbf{\Psi}$. For any d distinct rows of $\mathbf{\Psi}$ 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

$$\prod_{j<\ell} (z^{\ell-1} + z^{j-1}), \text{ for } j, \ell \in \{i_1, i_2, \cdots, i_d\}. \qquad (9)$$

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. As $n-1$ 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 $1 \le a \le n-1$, and then the determinant in (9) is $\mathcal{C}_m$-invertible. $\square$

The protocol of repairing a failed node is the same as in [7], 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 10. *Suppose that the parameter m satisfies the requirement in Lemma 9, and suppose there is a failed node. We can repair the α packets in the failed node by downloading the first $m-1$ bits of one packet each from any $d = 2k-2$ of the remaining nodes.*

Proof. We assume that the node f fails and the α packets in node f are $\boldsymbol{\psi}_f^t\mathbf{M}$, where $\boldsymbol{\psi}_f^t$ is the encoding vector of node f. The new node which is created to replace the failed node and connects to any d helper nodes $h_1, h_2, \ldots, h_d$. The helper node h_j first appends a parity-check bit for each $m-1$ bits to formulate the α packets $\boldsymbol{\psi}_{h_j}^t\mathbf{M}$ and then computes a packet $\boldsymbol{\psi}_{h_j}^t\mathbf{M}\phi_f$ and sends the first $m-1$ bits of packet $\boldsymbol{\psi}_{h_j}^t\mathbf{M}\phi_f$ to the new node. The new node thus obtains the first $m-1$ bits of each d packets $\boldsymbol{\Psi}_{repair}\mathbf{M}\phi_f$ from the d helper nodes, where

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

By Lemma 9, the square matrix $\boldsymbol{\Psi}_{repair}$ is $\mathcal{C}_m$-invertible. Therefore, the new node can first compute the parity-check bit for each received packet and then compute 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 obtain $\phi_f^t\mathbf{S}_1$ and $\phi_f^t\mathbf{S}_2$. Then the new node can compute

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

One can check that the above α packets are precisely the packets stored in the failed node. $\square$

Theorem 11. *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 9.*

Proof. For an arbitrary set $\{\ell_i | i = 1,2,\ldots,k\}$ of k nodes, we let $\boldsymbol{\Psi}_k$, Φ_k and Λ_k be the submatrix of $\boldsymbol{\Psi}$, Φ and Λ with rows indexed by $\{\ell_i | i = 1,2,\ldots,k\}$ respectively. We obtain the packets $\boldsymbol{\Psi}_k\mathbf{M} = \left[\Phi_k\mathbf{S}_1 + \Lambda_k\Phi_k\mathbf{S}_2\right]$ by appending the parity-check bits for the packets in the k nodes and then get

$$\left[\Phi_k\mathbf{S}_1\Phi_k^t + \Lambda_k\Phi_k\mathbf{S}_2\Phi_k^t\right]$$

by multiplying $\boldsymbol{\Psi}_k\mathbf{M}$ and Φ_k^t. For notation simplicity, let 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. The matrices P and Q are symmetric, as $\mathbf{S}_1$ and $\mathbf{S}_2$ are symmetric.

In the following, we 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

$$P_{i,j} + z^{\alpha(\ell_i - 1)}Q_{i,j}, \tag{10}$$

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

$$P_{j,i} + z^{\alpha(\ell_j - 1)}Q_{j,i} = P_{i,j} + z^{\alpha(\ell_j - 1)}Q_{i,j}, \tag{11}$$

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

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

Note that the matrix to the right is a Vandermonde matrix and we can obtain $\phi_{\ell_i}^t\mathbf{S}_1$ by Lemma 9.

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 9. Similarly, we can recover the matrix $\mathbf{S}_2$ from the matrix Q.

$\square$

2) BASIC Product-Matrix MBR Code: We divide the data file into

$$B = \frac{k(k+1)}{2} + k(d-k) \tag{12}$$

parts, each of size $m-1$ bits. Generate B source packets in $\mathcal{C}_m$ by appending the parity-check bit for each B parts. 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). Node i stores the first $m-1$ bits of each 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 12. *If m satisfies the requirement in Lemma 9, we can repair the packets of a failed node by downloading the first $m-1$ bits of 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 generates d coded packets $\boldsymbol{\psi}_{h_j}^t\mathbf{M}$ and sends the first $m-1$ bits of each the inner product $\boldsymbol{\psi}_{h_j}^t\mathbf{M}\boldsymbol{\psi}_f$ to the new node. The new node thus computes the d coded packets $\boldsymbol{\Psi}_{\text{repair}}\mathbf{M}\boldsymbol{\psi}_f$ by appending the parity-check bit for each received $m-1$ bits from the d helper nodes, where $\boldsymbol{\Psi}_{\text{repair}}$ is the square matrix with row consisting by $\boldsymbol{\psi}_{h_j}^t$ for $j = 1,2,\ldots,d$. By construction, the matrix $\boldsymbol{\Psi}_{\text{repair}}$ 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 $\boldsymbol{\Psi}_{\text{repair}}^{-1}$. Since $\mathbf{M}$ is symmetric, we have $(\mathbf{M}\boldsymbol{\psi}_f)^t = \boldsymbol{\psi}_f\mathbf{M}$, and the first $m-1$ bits of each $\boldsymbol{\psi}_f\mathbf{M}$ is precisely the data previously stored in the failed node. □

Theorem 13. *For the constructed BASIC-PM MBR codes with the requirement in Lemma 9, 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

$$\Phi_k = \begin{bmatrix} 1 & z^{\ell_1-1} & z^{2(\ell_1-1)} & \ldots & z^{(k-1)(\ell_1-1)} \\ 1 & z^{\ell_2-1} & z^{2(\ell_2-1)} & \ldots & z^{(k-1)(\ell_2-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z^{\ell_k-1} & z^{2(\ell_k-1)} & \ldots & z^{(k-1)(\ell_k-1)} \end{bmatrix} \tag{13}$$

is a Vandermonde matrix and invertible by Lemma 9. After subtracting the source packets in $\mathbf{T}$ from the first k columns of $\boldsymbol{\Psi}_{\text{k}}\mathbf{M}$, where $\boldsymbol{\Psi}_{\text{k}}$ is the submatrix of $\boldsymbol{\Psi}$ 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 $\Phi_k\mathbf{S}$. □

Note that we do not store the parity-check bit, and we can compute the last parity-check bit when necessary in the repair process and decoding process, as pointed out in the repair and decoding processes of BASIC-PM RGC. In the repair process, the helper node needs to compute the α parity-check bits to formulate the α polynomials before combining the coded polynomial, and the new node has to append the parity-check bit for each of the received $m-1$ bits from the helper nodes. While in the decoding process, the data collector should first computes the parity-check bits and then solves the Vandermonde system.

C. Example of BASIC-PM MBR Codes

In the following, we give an example for $n = 5$, $k = 3$, $d = 4$ and $m = 11$ of BASIC-PM 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 $\mathbb{F}_2[z]/(1+z^{11})$. The encoding vector of node i is

$$\boldsymbol{\psi}_i^t = \begin{bmatrix} 1 & z^{i-1} & z^{2(i-1)} & z^{3(i-1)} \end{bmatrix}, \tag{14}$$

for $i = 1,2,\ldots,5$, and node i stores the first 10 bits of polynomial in $\boldsymbol{\psi}_i^t\mathbf{M}$, namely

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

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 first add the parity-check bit for each packet and then 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 9, 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. After computing the parity-check bits for the packets in node 1, 2, 3 and 4, the first 10 bits of each the coded packet $\boldsymbol{\psi}_i^t\mathbf{M}\boldsymbol{\psi}_5$ are sent from helper node i to the new node. The new node can thus compute the packets as follows, by appending the parity-check bit for each of the $m-1$ received bits.

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

Since the matrix on the left is invertible by the result in Lemma 9, 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 failed 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 [7], it is easy to check that we can convert all the exact-repair RGC in [3], [4], [6].

VI. Computational Complexity

In this section, we compare computational complexity of BASIC-RGC and RGC over 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 finite field. Then we demonstrate that the coding and repair computational complexity of functional-repair BASIC-RGC is less than that of functional-repair RGC over finite field. For exact-repair BASIC-PM RGC, we show that the coding and repair complexity is much less than that of RGC-PM over finite field, by employing the LU decomposition of Vandermonde matrix.

A. Polynomial Representation of Finite Field

We represent the 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 field element. Addition is bit-wise XOR and multiplication in the field is multiplication modulo $g(z)$. Generally, a multiplication in the field $\mathbb{F}_{2^w}$ takes $O(w^2)$ bit operations. (See e.g. [26, 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 [27], 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 (KOA) [28], [29] and Fast Fourier Transform (FFT) [30]–[32]. The field multiplication complexity in $\mathbb{F}_{2^w}$ may be improved to $O(w^{\log_2 3})$ using KOA. The most efficient FFT algorithm was proposed in [32], 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 KOA and $O(m \log_2 m)$ by FFT algorithm in [32]. Although our binary cyclic codes has the same order of field multiplication complexity, if we apply an advanced multiplication technique for both of them, there is no need to transform the frequency domain to time domain to compute the reduction for binary cyclic codes. 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.

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 the multiplication of $a(z)b(z)$ over $\mathbb{F}_2[z]/(g(z))$, we first compute the *product* of $c(z) := a(z)b(z)$ over the ring $\mathbb{F}_2[z]$,

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

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, and the average number of XORs is thus $0.5w^2$. 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 $\deg c(z) < w$.

As the degree $\deg c(z)$ is at most $2w-2$, and after each iteration in the above, $\deg c(z)$ is decreased by at least one. 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, the average number of XORs of the field multiplication $a(z)b(z)$ is at most

$$\mu w, \text{ where } \mu = 0.5w + ||g(z)||_0. \tag{15}$$

For two polynomials $a(z), b(z)$ in the ring $\mathcal{R}_m$, the multiplication 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. Since $||a(z)||_0 \leq (m-1)/2$ (see the remark at the end of Subsection III-C), the number of XORs of the multiplication in $\mathcal{R}_m$ is thus at most $(m-1)m/2$. Therefore, the multiplication complexity over the ring $\mathcal{R}_m$ is much less than that of field multiplication, for $m-1 = w$. 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 field multiplication not only needs to compute $(m-1)/2$ shifted versions of $b(z)$ on average, but also modulo the irreducible polynomial $g(z)$.

B. Computational Complexity of Functional-Repair BASIC-RGC and RGC over Finite Field

For the purpose of easy presentation, we only consider the primary case of MSR code, i.e., the parameters B, α and β are set to $B = k(d-k+1)$, $\alpha = d-k+1$ and $\beta = 1$. The parameter m is chosen to be a prime number such that 2 is primitive in $\mathbb{F}_m$ and the inequality (6) holds.

1) BASIC-RGC: For the ease of comparison, we normalize the complexity by the file size. We separate the repair complexity into the number of XORs required in a helper node and the number of XORs required in the new node, which are called *repair complexity in helper node* and *repair complexity in new node* respectively.

Theorem 14. *Let m be a prime number such that 2 is primitive in $\mathbb{F}_m$, and let the set $\mathcal{S}$ to be the 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 in helper node, repair complexity in new node and decoding complexity of functional-repair BASIC-RGC are at most $\frac{n\alpha m}{2}$, $\frac{\beta m}{2k}$, $\frac{d\beta m}{2k}$ and $\frac{Bm}{2}$ respectively.*

Proof. **Encode.** Without loss of generality we assume that the data file contains $B(m-1) = k\alpha(m-1)$ bits. The data file is divided into B parts, each of $m-1$ bits. 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 requires $B(m-2)$ XORs. There are n storage nodes and each node stores α 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 remark at the end of Subsection III-C. The computational complexity of calculating one coded packet is thus at most $Bm(m-1)/2$ XORs. Hence the total number of XORs in encoding is $B(m-2)+n\alpha Bm(m-1)/2$. The normalized computational complexity of encoding is $(B(m-2)+n\alpha Bm(m-1)/2)/(B(m-1)) \approx n\alpha m/2$.

Repair. Each helper node generates β 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 term 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 normalized repair complexity in helper node is at most $\beta m/2k$. The new node generates α coded packets, each of them is obtained by combining the $d\beta$ received packets. The required number of XORs is at most $\alpha d\beta m(m-1)/2$. The normalized repair complexity in new node is $d\beta m/2k$.

Decode. A data collector recovers 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 the file size is large. The number of XORs in recovering one source packet is at most $k\alpha m(m-1)/2$. The normalized decoding complexity is therefore at most $(Bk\alpha m(m-1)/2)/(B(m-1)) = \frac{Bm}{2}$. $\square$

2) RGC Over Finite Field: Consider functional-repair RGC over the field $\mathbb{F}_{2^w}$ such that

$$w > \log_2\left(B\cdot\max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}\right). \tag{16}$$

As the upper bounds of $m+1$ and w are the same from the inequalities in (6) and (16), we let $m = w-1$, for fair comparison.

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 symbols in $\mathbb{F}_{2^{m-1}}$, we need to generate $n\alpha$ coded symbols. Each coded symbol is obtained by taking an linear combination of the B source symbols over the field $\mathbb{F}_{2^{m-1}}$. 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 $(m-1)\mu$ XORs at most by the equation (15). Therefore, one coded packet takes $B(m-1)\mu+(B-1)(m-1)$ XORs. The normalized encoding complexity is at most $n\alpha B(m-1)\mu/(B(m-1)) = n\alpha\mu$. Likewise, the repair and decoding complexity of RGC over finite field can be computed.

The comparison of computational complexity is summarized in Table I. The first row is the performance metric of the proposed functional-repair BASIC-RGC, 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 finite field. The coding scheme proposed in this paper has an additional 1 bit per $m-1$ bits, and this leads to a slight increase of 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 I show that the normalized computational complexity of RGC over finite field is larger than that of BASIC-RGC, for both coding and repair processes, if we replace μ by $(0.5m+||g(z)||_0)$.

Note that the computational complexity in Table I is for BASIC-RGC which store the parity-check bit. If we do not store the parity-check bit for functional-repair BASIC-RGC, we can check that the normalized computational complexity are the same.

In the above, we consider a class of prime number m such that 2 is primitive in $\mathbb{F}_m$. For such 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 [33], 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.

If $L > 1$, let $f_1(z)f_2(z)\cdots f_L(z)$ be the prime factorization of $h(z)$ such that $\deg(f_1(z)) \le \deg(f_\ell(z))\ \forall 2 \le \ell \le L$. Let the set $\mathcal{S}$ be equal to $\mathbb{F}_2[z]/f_1(z)$, we can check that the function θ_ℓ is injective $\forall 1 \le \ell \le L$. According to Theorem 7, we have

$$\deg(f_1(z)) > \log_2\left(L\cdot B\cdot\max\left\{\binom{n\alpha}{B}, 2|\mathcal{H}|\right\}\right). \tag{17}$$

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 $m-1 \ge L\deg(f_1(z))$. As the weight of local encoding coefficient is less than or equal to $\deg(f_1(z))$, so the repair complexity is much less than that of functional-repair RGC over finite field, while the decoding complexity may be larger than that of functional-repair RGC over finite field.

C. Computational Complexity of Exact-Repair BASIC-PM RGC

We estimate the computational complexity of encoding, repair and decoding, in terms of the number of XORs for exact-repair BASIC-PM RGC and RGC-PM over finite field in this section. 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.

TABLE I: Comparison of functional-repair.

	Normalized redundancy	Normalized repair bandwidth	Encoding complexity	Repair complexity in helper node	Repair complexity in new node	Decoding complexity
BASIC-RGC	$\frac{m}{m-1} \cdot \frac{n}{k}$	$\frac{m}{m-1} \cdot \frac{d}{k\alpha}$	$\frac{n\alpha m}{2}$	$\frac{\beta m}{2k}$	$\frac{d\beta m}{2k}$	$\frac{k\alpha m}{2}$
RGC	$\frac{n}{k}$	$\frac{d}{k\alpha}$	$n\alpha\mu$	$\frac{d\beta\mu}{2k}$	$\frac{d\beta\mu}{2k}$	$k\alpha\mu$

1) Decoding Method with LU Factorization of Vandermonde Matrix: In the following, we first give a fast decoding method using an LU factorization of the Vandermonde matrix, and then evaluate the computational complexity for BASIC-PM MBR codes. Expressing a matrix as a product of a lower triangular matrix L and an upper triangular matrix U is called an *LU factorization*. We first review some results on the LU factorization of the Vandermonde matrix.

Given a vector $\mathbf{a} = (a_0, a_1, \dots, a_\nu)$, we define the square Vandermonde matrix

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

with the second column equals to $\mathbf{a}$. Using symmetric functions and linear algebra, the author in [34] proved the result on the LU factorization of the Vandermonde matrix, and further simplified the L matrix and U matrix into 1-banded matrices.

Theorem 15. *[34] 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*

$$\mathbf{V}_\nu = L_\nu^{(1)} L_\nu^{(2)} \cdots L_\nu^{(\nu)} U_\nu^{(\nu)} \cdots U_\nu^{(2)} U_\nu^{(1)}, \qquad (18)$$

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 $1 \le \ell \le \nu$.

All the entries of $L_\nu^{(i)}$ and $U_\nu^{(i)}$ are either 0, 1, a_i, or $a_i - a_j$, $i > j$. A proof of Theorem 15 can be found in [34].

Given a $(\nu+1) \times (\nu+1)$ Vandermonde matrix $\mathbf{V}_\nu$ and $\mathbf{b} = (b_0, b_1, \dots, b_\nu)^t$, we can solve the linear system $\mathbf{V}_\nu \mathbf{x} = \mathbf{b}$ by solving

$$L_\nu^{(1)} L_\nu^{(2)} \cdots L_\nu^{(\nu)} U_\nu^{(\nu)} \cdots U_\nu^{(2)} U_\nu^{(1)} \mathbf{x} = \mathbf{b}.$$

We call the method by solving the above equation as *LU method.* According to Theorem 15, we are dealing with 1-banded triangular matrices which can be solved directly by *forward or backward substitution* without using the Gaussian elimination process. We can count that solving the 1-lower banded matrix $L_\nu^{(\ell)}$ system takes ℓ divisions and ℓ additions. Similarly, solving the 1-upper banded matrix $U_\nu^{(\ell)}$ system takes ℓ divisions and ℓ additions.

2) Computational Complexity of BASIC-PM MBR Codes: In the following, we evaluate the encoding, repair and decoding complexity of BASIC-PM MBR codes. We need the following lemma about how to compute the data packet $s(z)$ from $(1+z^b)s(z) = c(z)$ for $s(z), c(z) \in \mathcal{C}_m$.

Lemma 16. *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 $\mathcal{R}_m$,

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

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 $1 \le \ell \le m-1$. □

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 $\ell = 1, 2, \dots, m-1$. Thus, there are $\frac{3m-5}{2}$ XORs involved in the solving $s(z)$ from $(1+z^b)s(z) = c(z)$.

Recall that we do not store the parity-check bit for BASIC-PM RGC, and BASIC-PM MBR codes can exactly achieve the optimal MBR point. The normalized computational complexity is stated in the following theorem.

Theorem 17. *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. For $i = 1, 2, \dots, n$, the encoding vector of node i is*

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

If we use the LU method to decode the linear systems in the repair and decoding processes, the normalized encoding complexity, repair complexity in helper node, repair complexity in new node and decoding complexity of BASIC-PM MBR codes *are* $\frac{2n\alpha^2}{k(2d-k+1)}$, $\frac{4d}{k(2d-k+1)}$, $\frac{3.5d^2-1.5d}{k(2d-k+1)}$ *and* $\frac{k(kd-k^2+4.5d-3k)}{(2d-k+1)}$ *respectively.*

Proof. When we employ the LU method to solve a linear system in the repair and decoding processes of BASIC-PM MBR, the variable a_i is replaced by a power of z for $i = 0, 1, \dots, n$. In the process of solving the matrix $L_\nu^{(\ell)}$ system for $\ell = 1, 2, \dots, \nu$, we need to calculate $\frac{\nu(\nu+1)}{2}$ divisions by factor of the form $1 + z^b$, and $\frac{\nu(\nu+1)}{2}$ additions. So, the computation of solving the matrix $L_\nu^{(\ell)}$ system for $\ell = 1, 2, \dots, \nu$ is no larger than $\frac{5\nu(\nu+1)m}{4}$ XORs by Lemma 16, and solving the matrix $U_\nu^{(\ell)}$ system takes $\frac{\nu(\nu+1)}{2}$ additions, i.e., $\frac{\nu(\nu+1)m}{2}$ XORs. Therefore, the total computation of the LU method with operations over $\mathcal{R}_m$ is at most $\frac{7}{4}\nu(\nu+1)m$ XORs.

Assume that the data file contains $B(m-1)$ bits, where B is given in (12). First, we generate B source packets by encoding each group of $m-1$ bits to a codeword of $\mathcal{C}_m$, which takes $B(m-2)$ XORs. Each node stores $\alpha = d$ coded 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 nodes first computes the parity-check bits to get d packets and then sends one coded packet by adding the d shifted packets. The number of XORs of computing the d parity-check bits and adding the d packets in each node are $d(m-2)$ and dm respectively. The normalized repair complexity in helper node is $\frac{4d}{k(2d-k+1)}$. The new node first needs to add a parity-check bit for each received $m-1$ bits to obtain d coded packets, which takes $d(m-2)$ XORs. Then the new node computes a $d \times d$ linear system that can be solved using the LU method, with $\frac{7}{4}d(d-1)m$ XORs involved. The normalized repair complexity in new node is

$$\frac{d(m-2) + 1.75d(d-1)m}{B(m-1)} \approx \frac{3.5d^2 - 1.5d}{k(2d-k+1)}.$$

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

For any k nodes $\ell_1, \ell_2, \dots, \ell_k$, we can first add the parity-check bits for the packets in the k nodes and then solve the $(d-k)k$ source packets in $\mathbf{T}$ by solving the $d-k$ Vandermonde systems with the LU method. The complexity of the first part thus is $N_\mathbf{T} = kd(m-2)+\frac{7}{4}(d-k)k(k-1)m$. 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 defined in (13). Therefore $N_{sub} = k^2(d-k)(k+1)m/2$.

We can recursively solve the $k \times (k+1)/2$ source packets by the LU method as follows.

1) For $i = 1, 2, \dots, k-1$.
2) Solve $k-i+1$ source packets in the ith column of $\Phi_k\mathbf{S}$ by the LU method.
3) 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 computational complexity of calculating the $k \times (k+1)/2$ source packets 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

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

□

When we employ the LU method to solve a $k \times k$ linear system over finite field $\mathbb{F}_{2^w}$, the computation is $k(k-1)$ multiplications and $k(k-1)$ additions, i.e., $k(k-1)w\mu$ XORs. The computational complexity of exact-repair BASIC-PM MBR code and RGC-PM MBR code with the LU method is summarized in Table II. We can see that BASIC-PM MBR code only has $\frac{1}{\mu}$, $\frac{2}{\mu}$, $\frac{3.5}{2\mu}$ and $\frac{1}{\mu}$ complexity in encoding, repair in helper node, repair in new node and decoding of that of RGC-PM MBR code respectively. In RGC-PM MBR code over 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 an example of RGC-PM MBR code over $\mathbb{F}_{2^3}$ with $n = 5$, $k = 3$, $d = 4$. According to Table II, we can compute that BASIC-PM MBR code with the same parameters has only 22.2% encoding complexity, 37.5% decoding complexity, 66.0% repair complexity in helper node and 38.9% repair complexity in new node of that of RGC-PM MBR code.

TABLE III: 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	μ
Solve $s(z)$ from $(z^i + z^j)s(z)$	$\frac{3m-5}{2m}$	μ

The normalized decoding complexity of BASIC-PM MBR code is significantly less than that of RGC-PM MBR code, when we employ the LU method for both of them. The essential reason is as follows. With the LU method, 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 over the field $\mathbb{F}_{2^w}$. Table III summarizes the normalized computation of the three

TABLE II: Normalized computational complexity of exact-repair with the LU method.

	Encoding complexity	Repair complexity in helper node	Repair complexity in new node	Decoding complexity
BASIC-PM MBR	$\frac{2nd^2}{k(2d-k+1)}$	$\frac{4d}{k(2d-k+1)}$	$\frac{3.5d^2-1.5d}{k(2d-k+1)}$	$\frac{k(kd-k^2+4.5d-3k)}{(2d-k+1)}$
RGC-PM MBR	$\frac{2\mu nd^2}{k(2d-k+1)}$	$\frac{2d\mu}{k(2d-k+1)}$	$\frac{2d^2\mu}{k(2d-k+1)}$	$\frac{k(kd-k^2+3d-2k)\mu}{(2d-k+1)}$

operations. 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.

VII. Implementation

In this section, we present the implementation for the BASIC-PM MBR codes, and evaluate their encoding, decoding and repair performances in order to validate our theoretical analysis. The performances of BASIC-PM MBR codes are measured and compared to RGC-PM MBR codes over finite field using the publicly available implementation Jerasure 1.2 in [35]. We select the field size of RGC-PM MBR codes to be 2^8. Note that the encoding matrix of RGC-PM MBR codes is chosen to be an $n \times d$ Cauchy matrix in order to reduce the computational cost. Our BASIC-PM MBR prototype is written in C++ on Linux.

For RGC-PM MBR codes over finite field $\mathbb{F}_{2^w}$, the data file is divided into many pieces, each of Bw bits, where B is given in (12). Each piece is divided into B data symbols with the same size w bits. The B data symbols in each piece are used to generate $n\alpha$ coded symbols with encoding matrix being an $n \times d$ Cauchy matrix. The $n \times d$ Cauchy matrix over $\mathbb{F}_{2^w}$ is converted into a $wn \times wd$ *binary distribution matrix* using a projection defined by a primitive polynomial of $\mathbb{F}_{2^w}$ [36]. With binary distribution matrix, one may create a coded symbol as the XOR of some data symbols whose corresponding columns of the binary distribution matrix have ones. Note that the expensive field multiplication is replaced by binary addition. So this is a great improvement over standard field multiplication. For more information about the encoding and decoding process of Jerasure, we refer the reader to [36].

In our implementation of both codes, the data file is randomly generated with 100 MBytes. The file is divided into many *chunks* with the same size, and each chunk is partitioned to B *blocks* of the same size. For BASIC-PM MBR codes, a polynomial in $\mathcal{C}_m$ corresponds to a block and we fix the block size to be 4 KBytes, which is the default disk block size in existing Linux extended file systems. The block size of RGC-PM MBR codes is also chosen to be 4KBytes. The machine for testing has an Intel Core i3-4170 3.70GHz double-core CPU, 8GB RAM and 8GB Hard Disk. It runs Ubuntu 12.04. Each data point in the graphs that follow is the average of one thousand runs.

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 9 for the given parameters.

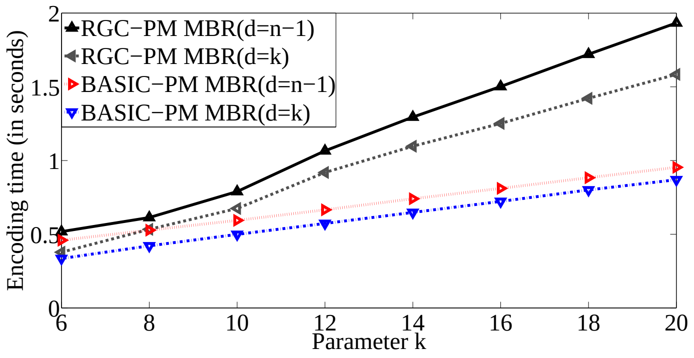

Fig. 2: The encoding time of BASIC-PM MBR code and RGC-PM MBR code.

We first evaluate the encoding performance, which is shown in Fig. 2. It is obvious that as k increases the encoding time increases, because the amount of data to be encoded is increased. 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.

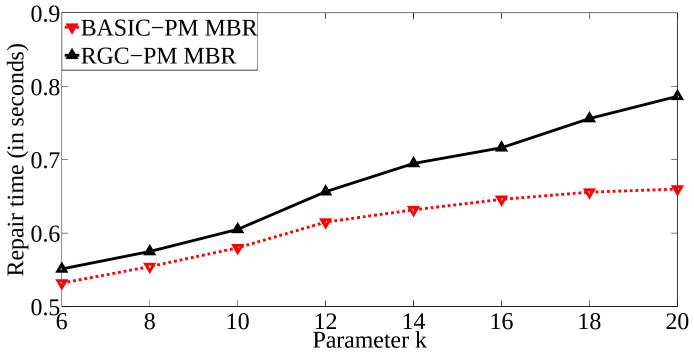

Fig. 3: The repair time of BASIC-PM MBR code and RGC-PM MBR code.

The repair time is shown in Fig. 3. We observe that the repair time of BASIC-PM MBR code increases as k increases when k is small, while when $k \geq 16$, the repair time of BASIC-PM MBR code is almost the same for different values of k, because the difference of normalized repair complexity can be ignored for the cases of $k = d$. 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 larger than that of BASIC-PM MBR code, and the difference becomes bigger when k becomes bigger.

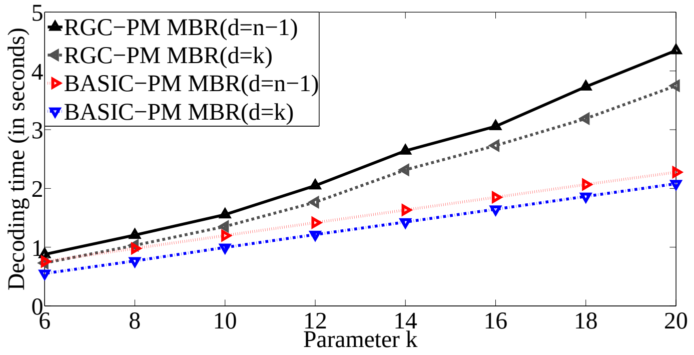

Fig. 4: The decoding time of BASIC-PM MBR code and RGC-PM MBR code.

We now compare the decoding time for the two codes. Here decoding time is the time of reconstructing the original data file from any k storage nodes. Fig. 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 reduce the decoding time of RGC-PM MBR code for all the evaluated parameters.

We notice that the advantage of BASIC-PM MBR codes on encoding/repair/decoding time is not so large as the normalized encoding/repair/decoding complexity, because of two reasons. First, the encoding/repair/decoding time includes not only the time of encoding/repair/decoding process, but also the I/O time. Second, the performance of Jerasure 1.2 in [35] is improved by choosing the binary distribution matrix which has the minimum number of ones.

VIII. Conclusion

We propose a framework of designing low complexity linear codes which employ XOR and bit-wise cyclic shifts, which is called BASIC codes. We give a general construction of functional-repair BASIC-RGC and show that the presented functional-repair BASIC-RGC can achieve all the fundamental tradeoff curve between storage and repair bandwidth of functional-repair RGC asymptotically with less complexity in coding and repair. We show that the product-matrix RGC in [7] can be converted to the exact-repair BASIC-RGC, with only $\frac{1}{\mu}$ encoding complexity, $\frac{7.5}{4\mu}$ repair complexity and $\frac{1}{\mu}$ decoding complexity. We implement BASIC-PM MBR codes and RGC-PM MBR codes over finite field based on the Jerasure 1.2 [35], our experiment results show that the encoding/repair/decoding time of BASIC-PM MBR codes is less than that of RGC-PM MBR codes.

References

[1] K. W. Shum, H. Hou, M. Chen, H. Xu, and H. Li, "Regenerating codes over a binary cyclic code," in *Proc. IEEE Int. Symp. Inf. Theory*, Honolulu, July 2014, pp. 1046–1050.

[2] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage system," in *Proc. IEEE INFOCOM*, Anchorage, Alaska, May 2007, pp. 2000–2008.

[3] 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 Trans. Inf. Theory*, vol. 58, no. 3, pp. 1837–1852, 2012.

[4] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, May 2013.

[5] N. B. Shah, "On minimizing data-read and download for storage-node recovery," *IEEE Comm. Letters*, vol. 17, no. 5, pp. 964–967, May 2013.

[6] K. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2009, pp. 1243–1249.

[7] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, August 2011.

[8] Y. Hu, P. P. C. Lee, and K. W. Shum, "Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems," in *Proc. IEEE INFOCOM*, Turin, April 2013.

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

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

[11] 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*, Honolulu, July 2014, pp. 2430–2434.

[12] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE J. Selected Areas in Communications*, vol. 28, no. 2, pp. 277–288, February 2010.

[13] P. Piret and T. Krol, "MDS convolutional codes," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 224–232, March 1983.

[14] M. Blaum and R. M. Roth, "New array codes for multiple phased burst correction," *IEEE Trans. Inf. Theory*, vol. 39, no. 1, pp. 66–77, January 1993.

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

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

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

[18] 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)*, Avignon, June 2010, pp. 313–317.

[19] S.-Y. R. Li and Q. T. Sun, "Network coding theory via commutative algebra," *IEEE Trans. Inf. Theory*, vol. 57, no. 1, pp. 403–415, January 2010.

[20] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier science publishers, 1977.

[21] C. L. Chen, W. W. Peterson, and E. J. Weldon Jr., "Some results on quasi-cyclic codes," *Information and Control*, vol. 15, no. 5, pp. 407–423, November 1969.

[22] S. Ling and p. Solé, "On the algebraic structure of quasi-cyclic codes I: Finite fields," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2751–2760, November 2001.

[23] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambrdige: Cambridge University Press, 1985.

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

[25] M. R. Murty, "Artin's conjecture for primitive roots," *Math. Intelligencer*, vol. 10, no. 4, pp. 59–67, 1988.

[26] H. Cohen and G. Frey, Eds., *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman & Hall/CRC, 2006.

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

[28] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet Physics Doklady*, vol. 7, 1963, pp. 595–596.

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

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

[31] J. M. Pollard, "The fast Fourier transform in a finite field," *Mathematics of computation*, vol. 25, no. 114, pp. 365–374, 1971.

[32] S. Gao and T. Mateer, "Additive fast Fourier transforms over finite fields," *IEEE Trans. Inf. Theory*, vol. 56, no. 12, pp. 6265–6272, 2010.

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

[34] S.-L. Yang, "On the LU factorization of the Vandermonde matrix," *Discrete applied mathematics*, vol. 146, no. 1, pp. 102–105, 2005.

[35] 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*, vol. 23, 2008.

[36] J. S. Plank, "Optimizing Cauchy Reed-Solomon codes for fault-tolerant storage applications," *University of Tennessee, Tech. Rep. CS-05-569*, 2005.