# Efficient Exact Regenerating Codes for Byzantine Fault Tolerance in Distributed Networked Storage

Yunghsiang S. Han, *Fellow, IEEE,* Hung-Ta Pai, *Senior Member, IEEE,* Rong Zheng, *Senior Member, IEEE*, and Wai Ho Mow, *Senior Member, IEEE*

***Abstract*—Today's large-scale distributed storage systems are commonly built using commodity software and hardware. As a result, crash-stop and Byzantine failures in such systems become more and more prevalent. In the literature, regenerating codes have been shown to be a more efficient way to disperse information across multiple storage nodes and recover from crash-stop failures. In this paper, we propose a novel decoding design of product-matrix constructed regenerating codes in conjunction with integrity check that allows exact regeneration of failed nodes and data reconstruction in the presence of Byzantine failures. A progressive decoding mechanism is incorporated in both procedures to leverage computation performed thus far. Unlike previous works, our new regenerating code decoding has the advantage that its building blocks, such as Reed-Solomon codes and standard cryptographic hash functions, are relatively well-understood because of their widespread applications. The fault tolerance and security properties of the proposed schemes are also analyzed. In addition, the performance of the proposed schemes, in terms of the average number of access nodes and the reconstruction failure probability versus the node failure probability, are also evaluated by Monte Carlo simulations.**

***Index Terms*—Network storage, regenerating code, Byzantine failures, Reed-Solomon code, error-detection code.**

## I. Introduction

STORAGE is becoming a commodity due to the emergence of new storage media and the ever decreasing cost of conventional storage devices. Reliability, on the other hand, continues to pose challenges in the design of large-scale distributed storage systems such as data centers. Today's data centers operate on commodity hardware and software, where both crash-stop and Byzantine failures (as a result of software bugs and malicious attacks) are likely to be the norm. To achieve persistent storage, one common approach is to disperse information pertaining to a data file (the message) across nodes in a network. For instance, with $[n, k]$ maximum-distance-separable (MDS) codes, such as Reed-Solomon (RS) codes, data are encoded and stored across $n$ nodes, while an end user or a data collector can retrieve the original data file by accessing *any* $k$ of the storage nodes, a process referred to as *data reconstruction*.

Upon the failure of any storage node, data stored in the failed node need to be regenerated (recovered) to maintain the functionality of the system. A straightforward way for data recovery is to first reconstruct the original data and then regenerate the data stored in the failed node. However, it is wasteful to retrieve the entire $B$ symbols of the original file, just to recover a fraction of that stored in the failed node. A more efficient way is to use the so-called *regenerating codes* which was introduced in the pioneering works by Dimakis *et al.* in [1], [2]. In data regeneration, each storage node stores $\alpha$ symbol and a total of $d$ surviving nodes are accessed to retrieve $\beta \leq \alpha$ symbols from each node. A trade-off can be made between the storage overhead and the repair bandwidth needed for regeneration. Minimum Storage Regenerating (MSR) codes minimize first, the amount of data stored per node, and then the repair bandwidth, while Minimum Bandwidth Regenerating (MBR) codes carry out the minimization in the reverse order. The design of regenerating codes have received much attention in recent years [3]–[11]. Most notably, Rashmi *et al.* proposed optimal exact-regenerating codes using a product-matrix reconstruction that recovers exactly the same stored data of the failed node (and thus the name exact-regenerating) [11].

Existing work assumes crash-stop failures on storage nodes. However, with Byzantine failures, the stored data may be tampered resulting in erroneous data reconstruction and regeneration. In [12], the code capability and resilience were presented for error-correcting regenerating codes. The authors also stated that it is possible to decode an $[n, k, d]$ MBR code up to $\lfloor \frac{n-k+1}{2} \rfloor$ errors, and further claimed that any $[n, k, d \geq 2k-2]$ MSR code can decode up to $\lfloor \frac{n-k+1}{2} \rfloor$ errors. However, no explicit decoding procedure was provided. Thus it remains an open problem as to whether practical decoding algorithms can be designed with such an error correction capability.[1]

Manuscript received June 29, 2013; revised September 27 and December 8, 2013. The editor coordinating the review of this paper and approving it for publication was M. Xiao.

Y. S. Han is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taiwan, R.O.C. (e-mail: yshan@mail.ntust.edu.tw).

H.-T. Pai is with the Department of Communication Engineering, National Taipei University, Taiwan, R.O.C. (e-mail: htpai@mail.ntpu.edu.tw).

R. Zheng is with the Department of Computing and Software, McMaster University, Canada (e-mail: rzheng@mcmaster.ca).

W. H. Mow is with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong (e-mail: w.mow@ieee.org).

This work was supported in part by the National Science Council, R.O.C., under grants NSC-99-2221-E-011-157 and NSC-101-2221-E-305-003, the AoE Grant E-02/08 from the University Grants Committee of the Hong Kong Special Administration Region, China, the US National Science Foundation under contract #1117560 and #0832084, and the Canadian Natural Sciences and Engineering Research Council (NSERC) Discovery grant (435601-2013). Part of Y. S. Han's work was completed during his visit to the Department of Electrical Engineering and Computer Science at Syracuse University.

Part of this work was presented at the 31st Annual IEEE International Conference on Computer Communications, Orlando, March 2012.

Digital Object Identifier 10.1109/TCOMM.2013.122313.130492

[1]In fact, Dimakis *et al.* stated several open problems in the end of their survey on regenerating codes [13]. Specifically, Open Problems 2 and 4 stated in [13] explicitly call for ways to "repair" Reed-Solomon codes in order to fully realize the potential advantages of reduced update complexity and efficient decoding under errors in distributed storage applications. These open problems are partially settled by the results in this paper.

0090-6778/14$31.00 © 2014 IEEE

Specifically, as inspired by [11], we consider the problem of exact regeneration for Byzantine fault tolerance in distributed storage networks and design practical decoding algorithms for RS-code-based MBR and MSR codes that can tolerate Byzantine faults. Two challenging issues arise when nodes may fail arbitrarily besides erasure faults. First, we need to verify whether the regenerated or reconstructed data are correct. Second, efficient algorithms are needed that *incrementally* retrieve additional stored data and perform data reconstruction and regeneration when errors have been detected. Cryptographic hash function is adopted to verify the integrity of stored data.[2] In particular, for data reconstruction, the hash value is coded along with the original data and distributed among storage nodes. For data regeneration, hash values for each storage node are stored distributively to ensure the correctness of verification in face of node failures. Incremental retrieval, reconstruction and regeneration are made possible by the use of a progressive decoding procedure. Finally, we would like to emphasize that it is of practical significance to be able to tackle a new challenge using a new solution approach with well-understood ingredients (i.e., decoders of RS codes in this case). The proposed decoding schemes are conceptually novel and practically significant.

The rest of the paper is organized as follows. Related work is briefly surveyed in Section II. We give an overview of regenerating codes and RS codes in Section III to prepare the readers with necessary background. The design of error-correcting exact regenerating codes for the MSR points and MBR points are presented in Section IV and Section V, respectively. Section VI provides performance evaluation of the proposed schemes. Finally, we conclude the paper in Section VII.

## II. Related Work

Regenerating codes were introduced in the pioneer works by Dimakis *et al.* in [1], [2]. In these works, the so-called cut-set bound was derived which is the fundamental limit for designing regenerating codes. In these works, the data reconstruction and regeneration problems were formulated as a multicast network coding problem. From the cut-set bound between the source and the destination, the parameters of the regenerating codes were shown to satisfy the bound, which reveals the trade-off between storage and repair bandwidth. Those parameters satisfying the cut-set bound with equality were also derived.

The regeneration codes with parameters satisfying the cut-set bound with equality were proposed in [3], [4]. In [3] a deterministic construction of the regenerating codes with $d = n-1$ was presented. In [4], the network coding approach was adopted to design the regenerating codes. Both constructions achieved functional regeneration but not exact regeneration. Recently, a decentralized minimum-cost repair scheme was proposed in [14].

Exact regeneration was considered in [5]–[7]. In [5], a search algorithm was proposed to search for exact-regenerating MSR codes with $d = n-1$; however, no systematic construction method was provided. In [6], the MSR codes with $k = 2, d = n-1$ were constructed by using the concept of interference alignment, which was borrowed from the context of wireless communications. In [7], the authors provided an explicit method to construct the MBR codes with $d = n-1$. No computation is required for these codes during the regeneration of a failed node. Explicit construction of the MSR codes with $d = k+1$ was also provided. However, these codes can perform exact regeneration only for a subset of failed storage nodes.

In [15], the authors proved that exact regeneration is impossible for MSR codes with $[n, k, d < 2k-3]$ when $\beta = 1$. Based on interference alignment approach, a code construction was provided for the MSR codes with $[n = d+1, k, d \geq 2k-1]$. In [11], the explicit constructions for optimal MSR codes with $[n, k, d \geq 2k-2]$ and optimal MBR codes were proposed. The construction was based on the product of two matrices: information matrix and encoding matrix. The information matrix (or its sub-matrices) is symmetric in order to have exact-regeneration property. However, the authors only considered crash-stop failures of storage nodes. In this work, we extend the code design in [11] to devise an efficient decoding scheme that can resist not only crash-stop failures but also Byzantine failures of storage nodes.

The progressive decoding mechanism for distributed storage was first introduced in [16]. The scheme retrieved just enough data from surviving storage nodes to recover the original data in the presence of crash-stop and Byzantine failures. The decoding was performed incrementally such that both the communication and computation costs can be minimized. In [16], each data generating node generates the coded data without communicating with other data generating nodes, and distributes the coded data to the storage nodes. Only data reconstruction was considered in [16]. No regenerating scheme was proposed therein.

The problem of security on regenerating codes were considered in [8]–[10]. In [8], the authors considered the security problem against eavesdropping and adversarial attackers during the regeneration process. They derived upper bounds on the maximum amount of information that can be stored safely. An explicit code construction was given for $d = n-1$ in the bandwidth-limited regime. Later, the code based on product-matrix framework was proposed against the eavesdropping attack [9]. It can achieve the information-theoretic secrecy capacity. The problem of Byzantine fault tolerance for regenerating codes was considered in [10]. The authors studied the resilience of regenerating codes which support multi-repairs. By using collaboration among newcomers, upper bounds on the resilience capability of regenerating codes were derived. Even though our work also deals with the Byzantine failures, it does not need to have multiple newcomers to recover the failures. In [17], the regenerating codes for erasure networks was considered. The fundamental bandwidth-storage tradeoffs was derived when erasure probability of channels was known. The maximum probability of successful data regeneration was also derived over erasure channels. In [18], the problem of a minimum-cost repair for multi-hop storage networks with known topology and known link costs was investigated, where the repair cost was defined to be the total link costs required

[2]Data integrity refers to assuring the consistency and accuracy of data after any data process or transmission. Integrity check (verification) is a mechanism or process for checking data integrity.

in the regeneration process. A lower bound of the repair cost was derived and strategies for cooperation among surviving nodes were devised for tandem, star, grid and fully connected networks. Both [17] and [18] focused on functional regeneration, but not exact regeneration. Kurihara and Kuwakado [19] analyzed the secrecy capacity of MBR codes, and proposed $[n, k, d, m]$ secure regenerating codes that prevent information leakage even when an eavesdropper has data of $m$ storage nodes or repairing data for $m$ failed nodes.

The coding for errors and erasures for random linear network coding has been considered in [20]. In this work a Reed-Solomon-like code was constructed and its decoding scheme was proposed.

In our earlier conference version [21], we have proposed an MSR code to handle Byzantine failures for $d = 2k - 2$. In comparison, in this paper, we obtain a much more general result for exact regeneration with $d$ up to $n-1$. Additionally, instead of using cyclic redundancy check (CRC), we apply hash functions for data integrity, which is more tamper-resistant. Finally, simulation study is carried out to study the performance of the proposed decoding mechanisms.

## III. Preliminaries

### A. Regenerating Codes

Regenerating codes achieve bandwidth efficiency in the regeneration process by storing additional symbols in each storage node or accessing more storage nodes. Let $\alpha$ be the number of symbols over a finite field $GF(q)$ stored in each storage node and $\beta \leq \alpha$ the number of symbols downloaded from each storage during regeneration. To repair the stored data in the failed node, a newcomer accesses $d$ surviving nodes with the total repair bandwidth $d\beta$. In general, the total repair bandwidth is much less than $B$. A regenerating code can be used not only to regenerate coded data but also to reconstruct the original data symbols. Let the number of storage nodes be $n$. An $[n, k, d]$ regenerating code requires at least $k$ and $d$ surviving nodes to ensure successful data reconstruction and regeneration [11], respectively. Clearly, $k \leq d \leq n-1$. An example of regeneration taken from [13] is given in Fig. 1.

The main results given in [2], [3] are the so-called cut-set bound on the repair bandwidth. It states that any regenerating code must satisfy the following inequality:

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

Minimizing $\alpha$ in (1) results in a regenerating code with minimum storage requirement; and minimizing $\beta$ results in that with minimum repair bandwidth [11]. It is impossible to have minimum values both on $\alpha$ and $\beta$ concurrently, and thus there exists a trade-off between storage and repair bandwidth. The two extreme points in (1) are referred to as the minimum storage regeneration (MSR) and minimum bandwidth regeneration (MBR) points, respectively. The values of $\alpha$ and $\beta$ for MSR point can be obtained by first minimizing $\alpha$ and then minimizing $\beta$:

$$\begin{aligned} \alpha &= \frac{B}{k} \\ \beta &= \frac{B}{k(d-k+1)} \ . \end{aligned} \tag{2}$$

Reversing the order of minimization we have $\beta$ and $\alpha$ for MBR as

$$\begin{aligned} \beta &= \frac{2B}{k(2d-k+1)} \\ \alpha &= \frac{2dB}{k(2d-k+1)} \ . \end{aligned} \tag{3}$$

As defined in [11], an $[n, k, d]$ regenerating code with parameters $(\alpha, \beta, B)$ is optimal if i) it satisfies the cut-set bound with equality, and ii) neither $\alpha$ nor $\beta$ can be reduced unilaterally without violating the cut-set bound. Clearly, both MSR and MBR codes are optimal regenerating codes.

It has been proved that when designing $[n, k, d]$ MSR for $k/(n+1) \leq 1/2$ or MBR codes, it suffices to consider those with $\beta = 1$ [11]. Throughout this paper, we assume that $\beta = 1$ for code design. Hence (2) and (3) become

$$\begin{aligned} \alpha &= d-k+1 \\ B &= k(d-k+1) = k\alpha \end{aligned} \tag{4}$$

and

$$\begin{aligned} \alpha &= d \\ B &= kd - k(k-1)/2 \ , \end{aligned} \tag{5}$$

respectively, when $\beta = 1$.

There are two ways to regenerate data for a failed node. If the replacement data generated is exactly the same as those stored in the failed node, we call it the *exact regeneration*. If the replacement data generated is only to guarantee the data reconstruction and regeneration properties, it is called *functional regeneration*. In practice, exact regeneration is more desired since there is no need to inform each node in the network regarding the replacement. In addition, it is easy to keep the code systematic via exact regeneration, where partial data can be retrieved without accessing $k$ nodes. Throughout this paper, we only consider exact regeneration and design exact-regenerating codes with error-correction capabilities. We target for all possible $n, k$ for the MBR points and $k/(n+1) \leq 1/2$ for the MSR points.

### B. Reed-Solomon codes

Since Reed-Solomon (RS) codes will be used in the design of regenerating codes, we briefly describe the encoding and decoding mechanisms of RS codes.

RS codes are well studied error-correction codes. They not only can recover data when nodes fail, but also can guarantee recovery when a subset of nodes are Byzantine. RS codes operate on symbols of $m$ bits, where all symbols are from the finite field $GF(2^m)$. An $[\tilde{n}, k]$ RS code is a linear code, with parameters $\tilde{n} = 2^m - 1$ and $\tilde{n} - k = 2t$, where $\tilde{n}$ is the total number of symbols in a codeword, $k$ is the total number

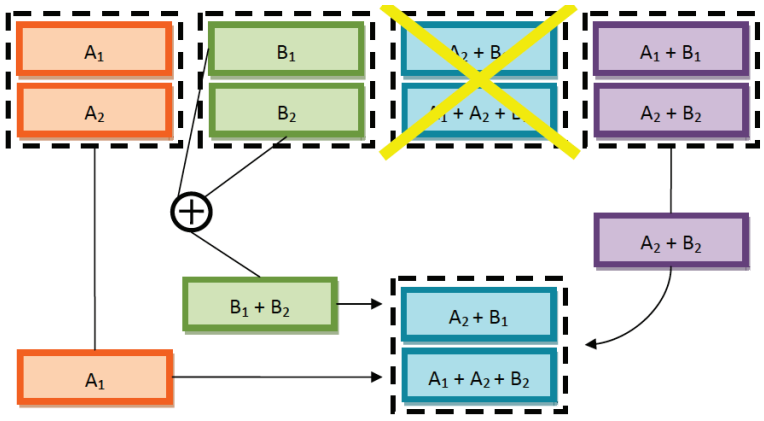



Fig. 1. An example of regeneration from [13].

of information symbols, and $t$ is the symbol-error-correction capability of the code.[3]

*Encoding:* Let the sequence of $k$ information symbols in $GF(2^m)$ be $\boldsymbol{u} = [u_0, u_1, \ldots, u_{k-1}]$ and $u(x)$ be the information polynomial of $\boldsymbol{u}$ represented as[4]

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1} \ .$$

The codeword polynomial, $c(x)$, corresponding to $u(x)$ can be encoded as [22]

$$c(x) = u(a^0) + u(a^1)x + u(a^2)x^2 + \cdots + u(a^{\tilde{n}-1})x^{\tilde{n}-1} \ ,$$

where $a$ is a generator (or a primitive element) in $GF(2^m)$. In the matrix form, the codeword

$$\begin{aligned} \mathbf{c} &= [c_0, c_1, \ldots, c_{\tilde{n}-1}] \\ &= [u(a^0), u(a^1), \ldots, u(a^{\tilde{n}-1})] \end{aligned}$$

is encoded as

$$\mathbf{c} = \boldsymbol{u} G \ ,$$

where

$$G = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{\tilde{n}-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{\tilde{n}-1})^2 \\ & & \vdots & \\ (a^0)^{k-1} & (a^1)^{k-1} & \cdots & (a^{\tilde{n}-1})^{k-1} \end{bmatrix} .$$

Let

$$\begin{aligned} g(x) &= (x-a)(x-a^2)\cdots(x-a^{2t}) \\ &= g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t} x^{2t} \ , \end{aligned} \tag{6}$$

for $g_i \in GF(2^m)$. It can be proved that $c(x)$ is divisible by $g(x)$, i.e., $a, a^2, \ldots, a^{2t}$ are roots of $c(x)$.

[3]The number of storage nodes, $n$, might be less than the length of the RS code, $\tilde{n}$. In this case, a shortened RS code with length $n < \tilde{n}$ is employed in our proposed scheme.

[4]We use polynomial and vectorized representations of information symbols, codewords, received symbols and errors interchangeably in this work.

*Decoding:* The decoding process of RS codes is more complex. A complete description can be found in [23].

Let $r(x)$ be the received polynomial and $r(x) = c(x) + e(x) + \gamma(x) = c(x) + \lambda(x)$, where $e(x) = \sum_{j=0}^{\tilde{n}-1} e_j x^j$ is the error polynomial, $\gamma(x) = \sum_{j=0}^{\tilde{n}-1} \gamma_j x^j$ the erasure polynomial, and $\lambda(x) = \sum_{j=0}^{\tilde{n}-1} \lambda_j x^j = e(x) + \gamma(x)$ the errata polynomial. Note that $g(x)$ and (hence) $c(x)$ have $a, a^2, \ldots, a^{2t}$ as roots. This property is used to determine the error locations and recover the information symbols.

The RS codes are optimal in terms of the minimum Hamming distance as it meets the Singleton bound [23]. An $[\tilde{n}, k]$ RS code can recover from any $v$ errors as long as $v \leq \lfloor \frac{\tilde{n}-k-s}{2} \rfloor$, where $s$ is the number of erasure (or irretrievable symbols). The decoding that handles both error and erasure is called the error-erasure decoding.

In $GF(2^m)$, addition is equivalent to bit-wise exclusive-or (XOR), and multiplication is typically implemented with multiplication tables or discrete logarithm tables. To reduce the complexity of multiplication, Cauchy Reed-Solomon (CRS) codes [24] have been proposed to use a different construction of the generator matrix, and convert multiplications to XOR operations for erasure. However, CRS codes incur the same complexity as RS codes for error correction.

## IV. Encoding and Decoding of Error-Correcting Exact-Regenerating Codes for the MSR Points

In this section, we demonstrate how to perform error correction on MSR codes designed to handle Byzantine failures by extending the code construction in [11]. The MSR code $\boldsymbol{C}$ with parameters $[n, k, d]$ for any $2k - 2 \leq d \leq n - 1$ can be constructed from an MSR code $\boldsymbol{C}'$ with parameters $[n' = n + \ell, k' = k + \ell, d' = d + \ell]$, where $d' = 2k' - 2$ and $\ell = d - (2k - 2)$. We call $\boldsymbol{C}'$ the mother code of $\boldsymbol{C}$. Note that the encoding and decoding of $\boldsymbol{C}$ are performed through $\boldsymbol{C}'$. By (4), we have $\alpha = d - k + 1 = d' - k' + 1$, $B = k\alpha$, and $B' = k'\alpha = B + \ell\alpha$. Hence, both codes share the same $\alpha$ and only differ in $\ell\alpha$ information symbols. If we have a systematic version of $\boldsymbol{C}'$ and set the last $\ell$ columns of information matrix $U$ to zeros, then the actual number of storage nodes for $\boldsymbol{C}'$ becomes $n' - \ell = n$, where $U$ is an $\alpha \times k'$ matrix. In such case, $\boldsymbol{C}'$ can be used to encode and decode the $B$ symbols. It

is clear that $d' = 2k' - 2$ and we have

$$\alpha = d' - k' + 1 = k' - 1 = d'/2$$

and

$$B' = k'\alpha = \alpha(\alpha + 1) \ .$$

We assume that the symbols in data are elements from $GF(2^m)$. Hence, the size of the data is $mB$ bits for $\beta = 1$. Note that since $d \leq n - 1$ it is clear that $2k - 2 \leq n - 1$ and then $k/(n + 1) \leq 1/2$.

### A. Verification for Data Reconstruction

Since we need to design codes with Byzantine fault tolerance it is necessary to perform integrity check after the original data are reconstructed. Two common verification mechanisms can be used: CRC and cryptographic hash function. Both methods add redundancy to the original data before they are encoded. In order to enhance the strength of verification on storage data, cryptographic hash function [25] is utlized in this work.

A cryptographic hash function is a one-way function that takes an arbitrary block of data and returns a fixed-size binary string, namely, the hash value. Modifying the input data without changing the hash value is computationally infeasible [25, Chapter 9]. Furthermore, it is also infeasible to generate the input data given the hash value alone. The design of cryptographic hash functions is usually based on block ciphers such as AES. Since the length of input data of a block cipher is fixed to some numbers, the Merkle-Damgård construction is adopted [26], [27]. A common length of the hash value is 224 bits as in SHA-224 [28].[5] Since the size of the original data is usually large, the redundancy added by imposing a cryptographic hash function is acceptable. For example, for a $[100, 20, 38]$ MSR code with $\alpha = 19,\ B = 19 \times 20 = 380$, we need to operate on $GF(2^{11})$ such that the size of the original data is $4180$ bits.[6] The redundancy added by the a cryptographic hash function is $224/4180 = 5.4\%$. Hence, in the following, we assume that the hash value has been added to the original data and the resultant size is $B$ symbols.

### B. Encoding

We arrange the information sequence

$$\boldsymbol{m} = [m_0, m_1, \ldots, m_{B-1}]$$

into an information matrix $U$ with size $\alpha \times k'$ such that the first $k$ columns contain the $B$ symbols of the information sequence and the last $\ell$ columns of $U$ are zeros. Let the generator matrix, $G$, of the MSR code is chosen as

$$G = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{n'-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{n'-1})^2 \\ & & \vdots & \\ (a^0)^{d'-1} & (a^1)^{d'-1} & \cdots & (a^{n'-1})^{d'-1} \end{bmatrix},$$

[5]A shorter lengh of hash value (160 bits) can be used such as in SHA-1; however, the security strength becomes weaker.

[6]The reason to choose $GF(2^{11})$ will be clear in the next section.

$d' = 2\alpha$, and $a$ is a generator of $GF(2^m)$. Note that $G$ is a generator matrix of the $[n', d']$ RS code. One important property of $G$ is that any $k'$ columns are linearly independent. The information matrix $U'$ to be encoded is obtained as

$$U' G_{k'} = U,$$

where

$$U' = [Z_1 Z_2] \, ,$$

$Z_j$'s are symmetric matrices with dimension $\alpha \times \alpha$ for $j = 1, 2$, and $G_{k'}$ is the last $k'$ columns of $G$. Given $U$, we need to solve for $U'$. In this encoding, each row of the information matrix $U'$ produces a codeword of length $n'$. Since the last $\ell$ symbols are zeros, the actual length of each codeword is $n$. Since the $[n', d']$ RS code is adopted to construct the MSR code, for the $i$th row of $U'$, the corresponding codeword is

$$[p_i(a^0 = 1), p_i(a^1), \ldots, p_i(a^{n-k-1}), u_{i1}, u_{i2,}, \ldots, u_{i,k}] \ , \quad (7)$$

where $p_i(x)$ is a polynomial with all elements in the $i$th row of $U'$ as its coefficients, that is, $p_i(x) = \sum_{j=0}^{d'-1} u_{ij} x^j$ and $u_{ij}$, $1 \leq j \leq k$, are the first $k$ elements in the $i$th row of $U$. In the matrix form, we have

$$U' \cdot G = [C|\mathbf{0}],$$

where $\mathbf{0}$ is all zero matrix with size $\alpha \times \ell$ and $C$ is the codeword vector with dimension $(\alpha \times n)$. Note that each row of $C$ is decoded separately in all decoding procedures. Finally, the $i$th column of $C$ is distributed to storage node $i$ for $1 \leq i \leq n$.

The generator matrix $G$ of the RS code can be reformulated as

$$G = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{n'-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{n'-1})^2 \\ & & \vdots & \\ (a^0)^{\alpha-1} & (a^1)^{\alpha-1} & \cdots & (a^{n'-1})^{\alpha-1} \\ (a^0)^\alpha 1 & (a^1)^\alpha 1 & \cdots & (a^{n'-1})^\alpha 1 \\ (a^0)^\alpha a^0 & (a^1)^\alpha a^1 & \cdots & (a^{n'-1})^\alpha a^{n'-1} \\ (a^0)^\alpha (a^0)^2 & (a^1)^\alpha (a^1)^2 & \cdots & (a^{n'-1})^\alpha (a^{n'-1})^2 \\ & & \vdots & \\ (a^0)^\alpha (a^0)^{\alpha-1} & (a^1)^\alpha (a^1)^{\alpha-1} & \cdots & (a^{n'-1})^\alpha (a^{n'-1})^{\alpha-1} \end{bmatrix} = \begin{bmatrix} \bar{G} \\ \bar{G}\Delta \end{bmatrix}, \quad (8)$$

where $\bar{G}$ contains the first $\alpha$ rows in $G$ and $\Delta$ is a diagonal matrix with $(a^0)^\alpha,\ (a^1)^\alpha,\ (a^2)^\alpha, \ldots,\ (a^{n'-1})^\alpha$ as diagonal elements. We require $(a^0)^\alpha,\ (a^1)^\alpha,\ (a^2)^\alpha, \ldots,\ (a^{n'-1})^\alpha$ to be all distinct. This can be guaranteed if this code is over $GF(2^m)$ for $m \geq \lceil \log_2 n' \rceil$ and $\gcd(2^m - 1, \alpha) = 1$.

It is easy to see that the $\alpha$ symbols stored in storage node $i$ is

$$U' \cdot \begin{bmatrix} \boldsymbol{g}_i^T \\ (a^{i-1})^\alpha \boldsymbol{g}_i^T \end{bmatrix} = Z_1 \boldsymbol{g}_i^T + (a^{i-1})^\alpha Z_2 \boldsymbol{g}_i^T,$$

where $\boldsymbol{g}_i^T$ is the $i$th column in $\bar{G}$.

Next we present the process to calculate the entries of $U'$ from $U$. The process is similar to the decoding process given

in [11]. Recall that $U'G_{k'} = U$, where $G_{k'}$ is the last $k'$ columns of $G$. That is

$$\begin{aligned} U'G_{k'} &= [Z_1 Z_2] \begin{bmatrix} \bar{G}_{k'} \\ \bar{G}_{k'}\Delta \end{bmatrix} \\ &= [Z_1\bar{G}_{k'} + Z_2\bar{G}_{k'}\Delta] \ . \end{aligned} \tag{9}$$

Multiplying $\bar{G}_{k'}^T$ to (9), we have

$$\begin{aligned} \bar{G}_{k'}^T Z_1 \bar{G}_{k'} + \bar{G}_{k'}^T Z_2 \bar{G}_{k'}\Delta &= P + Q\Delta \\ &= \bar{G}_{k'}^T U \ , \end{aligned} \tag{10}$$

where $P = \bar{G}_{k'}^T Z_1 \bar{G}_{k'}$ and $Q = \bar{G}_{k'}^T Z_2 \bar{G}_{k'}$. Since $Z_i$, $i = 1, 2$, are symmetric, $P$ and $Q$ are too. The $(i, j)$th, $1 \le i, j \le k'$ and $i \ne j$, element of $P + Q\Delta$ is

$$p_{ij} + q_{ij} a^{(j-1)\alpha} \ , \tag{11}$$

and the $(j, i)$th element is given by

$$p_{ji} + q_{ji} a^{(i-1)\alpha} \ . \tag{12}$$

Combining (11) and (12) we can obtain the values of $p_{ij}$ and $q_{ij}$ since $a^{(j-1)\alpha} \ne a^{(i-1)\alpha}$ for all $i \ne j$, $p_{ij} = p_{ji}$, and $q_{ij} = q_{ji}$.

Now consider $P = \bar{G}_{k'}^T Z_1 \bar{G}_{k'}$. We re-index $\boldsymbol{g}_{n'-k'+i}$ as $\bar{\boldsymbol{g}}_i$ for $1 \le i \le k'$. Since the only unknown elements in $P$ are those on diagonal and

$$\bar{G}_{k'}^T Z_1 \bar{G}_{k'} = \begin{bmatrix} \bar{\boldsymbol{g}}_1 \\ \bar{\boldsymbol{g}}_2 \\ \vdots \\ \bar{\boldsymbol{g}}_{k'} \end{bmatrix} Z_1 \begin{bmatrix} \bar{\boldsymbol{g}}_1^T & \bar{\boldsymbol{g}}_2^T & \cdots & \bar{\boldsymbol{g}}_{k'}^T \end{bmatrix} ,$$

we have $i$th row of $P$ excluding the element on diagonal as

$$\bar{\boldsymbol{g}}_i Z_1 \begin{bmatrix} \bar{\boldsymbol{g}}_1^T & \bar{\boldsymbol{g}}_2^T & \cdots & \bar{\boldsymbol{g}}_{i-1}^T & \bar{\boldsymbol{g}}_{i+1}^T & \cdots & \bar{\boldsymbol{g}}_{k'}^T \end{bmatrix} .$$

Since $k' = \alpha + 1$ and

$$\begin{bmatrix} \bar{\boldsymbol{g}}_1^T & \bar{\boldsymbol{g}}_2^T & \cdots & \bar{\boldsymbol{g}}_{i-1}^T & \bar{\boldsymbol{g}}_{i+1}^T & \cdots & \bar{\boldsymbol{g}}_{k'}^T \end{bmatrix}$$

is invertible due to construction, we can obtain $\bar{\boldsymbol{g}}_i Z_1$ for $1 \le i \le k'$. Selecting the first $\alpha$ of them yields

$$\begin{bmatrix} \bar{\boldsymbol{g}}_1 \\ \bar{\boldsymbol{g}}_2 \\ \vdots \\ \bar{\boldsymbol{g}}_{\alpha} \end{bmatrix} Z_1 \ .$$

Since

$$\begin{bmatrix} \bar{\boldsymbol{g}}_1 \\ \bar{\boldsymbol{g}}_2 \\ \vdots \\ \bar{\boldsymbol{g}}_{\alpha} \end{bmatrix}$$

is invertible, we can solve for $Z_1$. $Z_2$ can be obtained similarly from $Q$.

A final remark is that each column in $G$ can be generated by knowing the index of the column and the generator $a$. Therefore, each storage node does not need to store the entire $G$ to perform exact regeneration.

### C. Decoding for Data Reconstruction

The generator polynomial of the $[n', d']$ RS code encoded by (7) has $a^{n-d}, a^{n-d-1}, \ldots, a$ as roots [23].[7] Without loss of generality, we assume that the data collector retrieves encoded symbols from $k$ storage nodes $j_0, j_1, \ldots, j_{k-1}$ and forms the received matrix $Y_{\alpha\times k}$. First, the information sequence $\boldsymbol{m}$ is recovered by the procedure given in Section IV-B as follows.

The data collector pads $\ell$ columns of zeros to $Y_{\alpha\times k}$ to form $Y_{\alpha\times k'}$. It then collects the $k$ columns of $G$ corresponding to storage nodes $j_0,\ j_1, \ldots,\ j_{k-1}$ as the first $k$ columns of $\bar{G}_{k'}$ and the last $\ell$ columns from the last $\ell$ columns of $G$. The data collector calculates $\left(\bar{G}_{k'}\right)^T Y_{\alpha\times k'}$ and obtains $P$ and $Q$ from $\left(\bar{G}_{k'}\right)^T Y_{\alpha\times k'}$ via (11) and (12). Denote $\bar{G}_{k'}$ as $[\bar{\boldsymbol{g}}_i^T]_{1\le i\le k'}$. For $1 \le j \le k'-1$, it obtains $\bar{\boldsymbol{p}}_j$ by taking the $j$th row of $P$ and deleting the $j$th element in it. It calculates $\left[\bar{\boldsymbol{p}}_j \left([\bar{\boldsymbol{g}}_i^T]_{\substack{1\le i\le k'-1 \\ i\ne j}}\right)^{-1}\right]_{1\le j\le k'-1}$ and then obtains

$$Z_1 = \left([\bar{\boldsymbol{g}}_j]_{1\le j\le k'-1}\right)^{-1} \left(\left[\bar{\boldsymbol{p}}_j \left([\bar{\boldsymbol{g}}_i^T]_{\substack{1\le i\le k'-1 \\ i\ne j}}\right)^{-1}\right]_{1\le j\le k'-1}\right) .$$

$Z_2$ can be obtained by replacing $P$ with $Q$ in the above procedure. Then multiplying $U' = [Z_1\ Z_2]$ by $\bar{G}_{k'}$ to obtain the original $U$ and the information sequence $\boldsymbol{m}$.

If the recovered information sequence does not pass the integrity check, we need to perform the error-erasure decoding. In addition to the received encoded symbols from $k$ storage nodes, the data collector needs to retrieve the encoded symbols from $d + 2 - k$ storage nodes of the remaining storage nodes. The data collector then performs error-erasure decoding to obtain $\boldsymbol{m}$.[8] If the recovered information sequence passes the integrity check, the process finishes; otherwise, two more symbols need to be retrieved. The data collector continues the decoding process until it successfully recovers the correct information sequence or no more storage nodes can be accessed. In each step, the progressive decoding that we proposed in [16] is applied to reduce the computation complexity. Note that the RS code used is capable of correcting up to $\lfloor (n-d)/2 \rfloor$ errors.

The decoding algorithm with integrity check is summarized in Algorithm 1. Note that, in practice, Algorithm 1 will be repeated $\beta$ times for each retrieved symbol when $\beta > 1$.[9]

### D. Verification for Regeneration

To verify whether the recovered data are the same as those stored in the failed node, integrity check is needed. However, such a check should be performed based on information stored on nodes *other than* the failed node. We consider two mechanisms for verification.

In the first scheme, each storage node keeps the hash values for the remaining $n - 1$ storage nodes. When the newcomer accesses $d$ surviving storage nodes, it also asks for the hash values for the failed node from them. Using the majority vote on all received hash values, the newcomer can

[7] Recall that $n' - d' = n - d$.

[8] The information sequence is now part of the codeword due to the construction of $U'$ from $U$.

[9] By definition, $\beta$ is the number of symbols sent out from each of the $d$ surviving nodes. Hence, it should be kept as an integer by modifying $B$.

**Algorithm 1:** Decoding of MSR Codes for Data Reconstruction

**begin**

The data collector randomly chooses $k$ storage nodes and retrieves encoded data, $Y_{\alpha\times k}$; Pad $\ell$ columns of zeros to $Y_{\alpha\times k}$ to form $Y_{\alpha\times k'}$;

Collect the $k$ columns of $G$ corresponding to storage nodes $j_0,\ j_1,\ldots,\ j_{k-1}$ as the first $k$ columns of $\bar{G}_{k'}$ and the last $\ell$ columns from the last $\ell$ columns of $G$;

Calculate $\left(\bar{G}_{k'}\right)^T Y_{\alpha\times k'}$; Obtain $P$ and $Q$ from $\left(\bar{G}_{k'}\right)^T Y_{\alpha\times k'}$ via (11) and (12);

Denote $\bar{G}_{k'}$ as $[\bar{\boldsymbol{g}}_i^T]_{1\le i\le k'}$; For $1\le j\le k'-1$, obtain $\bar{\boldsymbol{p}}_j$ by taking $j$th row of $P$ and deleting $j$th element in it; Calculate $\left[\bar{\boldsymbol{p}}_j\left([\bar{\boldsymbol{g}}_i^T]_{\substack{1\le i\le k'-1\\ i\ne j}}\right)^{-1}\right]_{1\le j\le k'-1}$ and then obtain $Z_1 = \left([\bar{\boldsymbol{g}}_j]_{1\le j\le k'-1}\right)^{-1}\left(\left[\bar{\boldsymbol{p}}_j\left([\bar{\boldsymbol{g}}_i^T]_{\substack{1\le i\le k'-1\\ i\ne j}}\right)^{-1}\right]_{1\le j\le k'-1}\right)$;

Obtain $Z_2$ by replacing $P$ with $Q$ in the above procedure;

Obtain $\tilde{\boldsymbol{m}}$ in $U$ by multiplying $U'=[Z_1\ Z_2]$ with $\bar{G}_{k'}$;

**if** $HashTest(\tilde{\boldsymbol{m}}) = SUCCESS$ **then**

  **return** $\tilde{\boldsymbol{m}}$;

**else**

  Retrieve $d-k$ more encoded data from remaining storage nodes and merge them into $Y_{\alpha\times d}$; $j\leftarrow d'$;

  **while** $j\le n'-2$ **do**

    $j\leftarrow j+2$;

    Retrieve two more encoded data from remaining storage nodes and merge them into $Y_{\alpha\times j}$ and assign zeros to the last $\ell$ columns of $Y_{\alpha\times j}$;

    Perform progressive error-erasure decoding on each row in $Y_{\alpha\times j}$ to recover $\tilde{\boldsymbol{c}}$. Then obtain the $\tilde{\boldsymbol{m}}$ directly from $\tilde{\boldsymbol{c}}$;

    **if** $HashTest(\tilde{\boldsymbol{m}}) = SUCCESS$ **then**

      **return** $\tilde{\boldsymbol{m}}$;

  **return** FAIL;

obtain the correct hash value if no more than $\lfloor(d-1)/2\rfloor$ accessed storage nodes are compromised. To see the storage complexity of this scheme, let us take a numerical example. Consider a [100, 20, 38] MSR code with $\beta = 1000, \alpha = 19\times\beta = 19000, B = 4.18\text{Mb}$. The total number of bits stored in each node is then $19000\times 11 = 209000$. If a 224-bit hash value is added to each storage node, the redundancy is $r(n-1)/(\alpha m) = 224\times 99/209000 \approx 10\%$ and the extra bandwidth for transmitting the hash values is around $r/(\beta m) = 224/11000 \approx 2\%$, where $r$ is the size of the hash value. Hence, both redundancy for storage and bandwidth are manageable for large $\beta$'s.

When $\beta$ is small, we adopt an error-correcting code to encode the $r$-bit hash value. This can improve the storage and bandwidth efficiency. First we select the operating finite field $GF(2^{m'})$ such that $2^{m'}\ge n-1$. Then an $[n-1,\hat{k}]$ RS code with $\hat{k} = \lceil r/m'\rceil$ is used to encode the hash value. Note that this code is different from the RS code used for MSR data regeneration. In encoding the hash value of a storage node into $n-1$ symbols and distributing them to the $n-1$ other storage nodes, extra $(n-1)m'$ bits are needed on each storage node. When the newcomer accesses $d$ storage nodes to repair the failed node $i$, these nodes also send out the symbols associated with the hash value for node $i$. The newcomer then can perform error-erasure decoding to recover the hash value. The maximum number of compromised storage nodes among the accessed $d$ nodes that can be handled by this approach is $\lfloor(d-\hat{k})/2\rfloor$ and the extra bandwidth is $dm'$. Since $m'$ is much smaller than $n-1$ and $r$, the redundancy for storage and bandwidth can be reduced. Taking the previous numerical example, we have $m'=8$ and $\hat{k}=28$. Note that after the newcomer regenerates the data stored in the failed node, it requests the hash values from the other $n-1$ nodes.

### *E. Decoding for Regeneration*

Let node $i$ be the failed node to be recovered. During regeneration, the newcomer accesses $s$ surviving storage nodes, where $d\le s\le n-1$. Without loss of generality, we assume that the storage nodes accessed are $j_0,\ j_1,\ldots,\ j_{s-1}$. Every accessed node takes the inner product between its $\alpha$ symbols and

$$\boldsymbol{g}_i = [1, (a^{i-1})^1, (a^{i-1})^2, \ldots, (a^{i-1})^{\alpha-1}]\ , \tag{13}$$

where $\boldsymbol{g}_i$ can be generated by index $i$ and the generator $a$, and sends the resultant symbol to the newcomer. Since the MSR code is a linear code, the resultant symbols transmitted, $y_{j_0},\ y_{j_1},\ y_{j_2},\ldots,\ y_{j_{s-1}}$, that are appended $\ell$ 0's, can be decoded to the codeword $\boldsymbol{c}$, where

$$\begin{aligned}[\boldsymbol{c}|\boldsymbol{0}] &= \boldsymbol{g}_i\cdot(U'\cdot G)\\ &= (\boldsymbol{g}_i\cdot U')\cdot G\ ,\end{aligned}$$

if $(n-s)+2e < n-d+1$, where $e$ is the number of errors among the $s$ resultant symbols. Multiplying $[\boldsymbol{c}|\boldsymbol{0}]$ by the inverse of the first $d'$ columns of $G$, i.e., $\hat{G}_{d'}^{-1}$, one can recover

$$\boldsymbol{g}_i\cdot U'$$

which is equivalent to

$$\boldsymbol{g}_i\cdot[Z_1\ Z_2] = [\boldsymbol{g}_i\cdot Z_1\ \boldsymbol{g}_i\cdot Z_2]\ .$$

Recall that $\boldsymbol{g}_i$ is the transpose of $i$th column of $\bar{G}$, the first $\alpha$ rows in $G$. Since $Z_j$, for $j=1,2$, are symmetric matrices, $(\boldsymbol{g}_i Z_j)^T = Z_j\boldsymbol{g}_i^T$. The $\alpha$ symbols stored in the failed node $i$ can then be calculated as

$$(\boldsymbol{g}_i Z_1)^T + (a^{i-1})^\alpha(\boldsymbol{g}_i Z_2)^T\ . \tag{14}$$

The progressive decoding procedure in [16] can be applied in decoding $y_{j_0},\ y_{j_1},\ y_{j_2},\ldots,\ y_{j_{s-1}}$. First, the newcomer accesses $d$ storage nodes and decodes $y_{j_0},\ y_{j_1},\ y_{j_2},\ldots,\ y_{j_{d-1}}$ to obtain $\boldsymbol{c}$ and $\alpha$ symbols by (14). Then, it verifies the hash value. If the integrity check is passed, the regeneration is successful; otherwise, two more surviving storage nodes need to be accessed. Then the newcomer decodes the received $y_{j_0},\ y_{j_1},\ y_{j_2},\ldots,\ y_{j_{d+1}}$ to obtain $\boldsymbol{c}$ and recover $\alpha$ symbols.

The process repeats until a sufficient number of correctly stored data have been retrieved to recover the failed node. Again, in practice, when $\beta > 1$, the decoding needs to be performed $\beta$ times to recover $\beta\alpha$ symbols before verifying the hash value. The data regenerating algorithm with integrity check is summarized in Algorithm 2.

**Algorithm 2:** Decoding of MSR Codes for Regeneration

**begin**
- Assume node $i$ is failed.
- The newcomer randomly chooses $d$ storage nodes;
- Each chosen storage node combines its symbols as a $(\beta \times \alpha)$ matrix and multiply it by $\boldsymbol{g}_i$ in (13);
- The newcomer collects these resultant vectors as the first $d$ columns in a $(\beta \times d')$ matrix $Y$. Assign zeros to the last $\ell$ columns of $Y$.
- The newcomer obtains the hash value for node $i$;
- $j \leftarrow d'$;
- **repeat**
  - Perform progressive error-erasure decoding on each row in $Y$ to recover $\tilde{C}$ (error-erasure decoding performs $\beta$ times);
  - $M = [\tilde{C}|\mathbf{0}]\hat{G}_{d'}^{-1}$, where $\hat{G}_{d'}^{-1}$ is the inverse of the first $d'$ columns of $G$;
  - Obtain the $\beta\alpha$ information symbols, $\boldsymbol{s}$, from $M$ by the method given in (14);
  - **if** $HashTest(\boldsymbol{s}) = SUCCESS$ **then**
    - **return** $\boldsymbol{s}$;
  - **else**
    - $j \leftarrow j + 2$;
    - The newcomer accesses two more remaining storage nodes;
    - Each chosen storage node combines its symbols as a $(\beta \times \alpha)$ matrix and multiply it by $\boldsymbol{g}_i$ given in (13);
    - The newcomer merges the resultant vectors into $Y_{\beta\times j}$;
- **until** $j \geq n' - 2$;
- **return** FAIL;

## V. Encoding and Decoding of Error-Correcting Exact-Regenerating Codes for the MBR Points

In this section we demonstrate that by selecting the same RS codes as that for MSR codes and designing a proper decoding procedure, the MBR codes in [11] can be extended to handle Byzantine failures. Since the verification procedure for MBR codes is the same as that of MSR codes, it is omitted.

### A. Encoding

Let the information sequence $\boldsymbol{m} = [m_0, m_1, \ldots, m_{B-1}]$ be arranged into an information matrix $U$ with size $\alpha \times d$ such that

$$u_{ij} = \begin{cases} u_{ji} = m_{k_1} & \text{for } i \leq j \leq k \\ u_{ji} = m_{k_2} & \text{for } k+1 \leq i \leq d,\ 1 \leq j \leq k \\ 0 & \text{otherwise} \end{cases},$$

where $k_1 = (i-1)(k+1) - i(i+1)/2 + j$ and $k_2 = (i-k-1)k + k(k+1)/2 + j$. In the matrix form, we have

$$U = \begin{bmatrix} A_1 & A_2^T \\ A_2 & \mathbf{0} \end{bmatrix}, \tag{15}$$

where $A_1$ is a $k \times k$ matrix, $A_2$ a $(d-k) \times k$ matrix, $\mathbf{0}$ is the $(d-k) \times (d-k)$ zero matrix. Both $A_1$ and $U$ are symmetric. It is clear that $U$ has a dimension $d \times d$ (or $\alpha \times d$).

We apply an $[n, d]$ RS code to encode each row of $U$. Let $p_i(x)$ be the polynomial with all elements in the $i$th row of $U$ as its coefficients. That is, $p_i(x) = \sum_{j=0}^{d-1} u_{ij}x^j$. The corresponding codeword of $p_i(x)$ is thus

$$[p_i(a^0 = 1), p_i(a^1), \ldots, p_i(a^{n-1})]\ . \tag{16}$$

Recall that $a$ is a generator of $GF(2^m)$. In the matrix form, we have

$$U \cdot G = C,$$

where

$$G = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{n-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{n-1})^2 \\ & & \vdots & \\ (a^0)^{k-1} & (a^1)^{k-1} & \cdots & (a^{n-1})^{k-1} \\ (a^0)^k & (a^1)^k & \cdots & (a^{n-1})^k \\ & & \vdots & \\ (a^0)^{d-1} & (a^1)^{d-1} & \cdots & (a^{n-1})^{d-1} \end{bmatrix},$$

and $C$ is the codeword vector with dimension $(\alpha \times n)$. $G$ is called the generator matrix of the $[n, d]$ RS code. $G$ can be divided into two sub-matrices as

$$G = \begin{bmatrix} G_k \\ B \end{bmatrix},$$

where

$$G_k = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{n-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{n-1})^2 \\ & & \vdots & \\ (a^0)^{k-1} & (a^1)^{k-1} & \cdots & (a^{n-1})^{k-1} \end{bmatrix} \tag{17}$$

and

$$B = \begin{bmatrix} (a^0)^k & (a^1)^k & \cdots & (a^{n-1})^k \\ & & \vdots & \\ (a^0)^{d-1} & (a^1)^{d-1} & \cdots & (a^{n-1})^{d-1} \end{bmatrix}.$$

Note that $G_k$ is a generator matrix of the $[n, k]$ RS code and it will be used in the decoding process for data reconstruction.

### B. Decoding for Data Reconstruction

The generator polynomial of the RS code encoded by (17) has $a^{n-k}, a^{n-k-1}, \ldots, a$ as roots [23]. Hence, the progressive decoding scheme given in [16] can be applied to decode the proposed code if there are errors in the retrieved data. Unlike the decoding procedure given in Section IV-C, where an $[n, d]$ RS decoder is applied, we need an $[n, k]$ RS decoder for MBR codes.

Without loss of generality, we assume that the data collector retrieves encoded symbols from $s$ storage nodes $j_0,\ j_1, \ldots,\ j_{s-1}$, $k \leq s \leq n$. Recall that $\alpha = d$ in MBR. Hence, the data collector receives $d$ vectors where each vector has $s$ symbols. Collecting the first $k$ vectors as $Y_k$ and the remaining $d-k$ vectors as $Y_{d-k}$. From (15), we can view the codewords in the last $d-k$ rows of $C$ as being encoded by $G_k$ instead of $G$. Hence, the decoding procedure of $[n,k]$ RS codes can be applied on $Y_{d-k}$ to recover the codewords in the last $d-k$ rows of $C$. Let $\hat{G}_k$ be the first $k$ columns of $G_k$ and $\tilde{C}_{d-k}$ be the recovered codewords in the last $d-k$ rows of $C$. $A_2$ in $U$ can be recovered as

$$\tilde{A}_2 = \tilde{C}_{d-k} \cdot \hat{G_k}^{-1} \ . \tag{18}$$

We then calculate $\tilde{A}_2^T \cdot B$ and only keep the $j_0$th, $j_1$th, …, $j_{s-1}$th columns of the resultant matrix as $E$, and subtract $E$ from $Y_k$:

$$Y_k' = Y_k - E \ . \tag{19}$$

Applying the RS decoding algorithm again on $Y_k'$ we can recover $A_1$ as

$$\tilde{A}_1 = \tilde{C}_k \cdot \hat{G_k}^{-1} \ . \tag{20}$$

Hash value is computed on the decoded information sequence to verify the recovered data. If the integrity check is passed, the data reconstruction is successful; otherwise the progressive decoding procedure is applied, where two more storage nodes need to be accessed from the remaining storage nodes in each round until no further errors are detected. The detailed data reconstruction algorithm can be found in our conference version [21].

### C. Decoding for Regeneration

Decoding for regeneration with MBR is very similar to that with MSR. After obtaining $g_i \cdot U$, we take its transposition. Since $U$ is symmetric, we have $U^T = U$ and

$$U^T \cdot g_i^T = U \cdot g_i^T \ .$$

A integrity check is performed on all $\beta\alpha$ symbols. If the integrity check is passed, the $\beta\alpha$ symbols are the data stored in the failed node; otherwise, the progressive decoding procedure is applied.

## VI. Performance Evaluation

In this section, we first analyze the fault-tolerance capability of the proposed codes in presence of crash-stop and Byzantine failures, and then carry out numerical simulations to evaluate the performance for proposed schemes.

### A. Fault-tolerance capability

In analyzing the fault-tolerance capability, we consider two types of failures, namely, crash-stop failures and Byzantine failures. Nodes are assumed to fail independently (as opposed in a coordinated fashion). In both cases, the fault-tolerance capability is measured by the maximum number of failures that the system can handle to maintain correctness.

*Crash-stop failure:* Crash-stop failures can be viewed as an erasure in the codeword. Since at least $k$ nodes need to be available for data reconstruction, it can be shown that the maximum number of crash-stop failures that can be tolerated in data reconstruction is $n-k$. For regeneration, $d$ nodes need to be accessed. Thus, the fault-tolerance capability is $n-d-1$. Note that since all live nodes contain correct data, the associated hash values are also correct.

*Byzantine failure:* In general, in RS codes, two additional correct code fragments are needed to correct one erroneous code fragment. However, in the case of data regeneration, the capability of the newcomer to obtain the correct hash value also matters. In the analysis, we assume that the majority rule scheme is used in the process to obtain the correct hash value. Data regeneration fails if the newcomer cannot obtain the correct hash value even when the number of compromised nodes is less than the maximum number of faults the RS code can handle. Hence, we must take the minimum of the capability of the RS code (in MBR and MSR) and the capability to recover the correct hash value. Thus, with MSR and MBR codes, $\lfloor \frac{n-d}{2} \rfloor$ and $\lfloor \frac{n-k}{2} \rfloor$ erroneous nodes can be tolerated in data reconstruction. On the other hand, the fault-tolerance capability of MSR and MBR codes for data regeneration are both $\min\left\{ \lfloor \frac{n-d-1}{2} \rfloor, \lfloor \frac{d-1}{2} \rfloor \right\}$.

Table I summarizes the quantitative results of the fault-tolerance capability of MSR and MBR codes.

### B. Simulation study

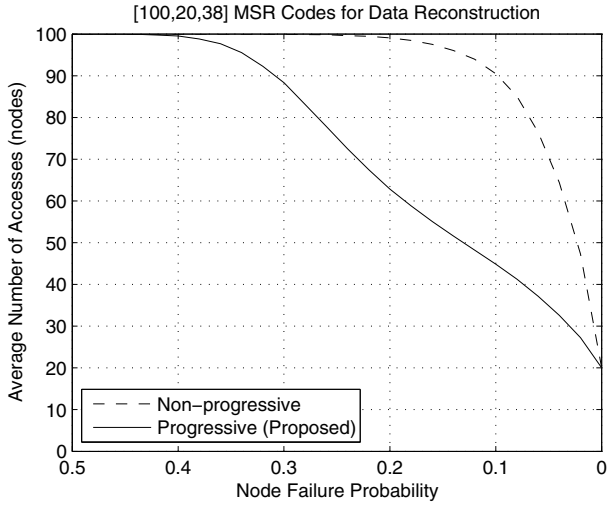
The performance of the proposed decoding scheme for MSR and MBR codes are evaluated by Monte Carlo simulations. For comparison, the performance of a traditional decoding scheme that is non-progressive is also provided. For the non-progressive decoding scheme, after $k$ nodes are accessed, if the integrity check fails, the data collector will access all remaining $n-k$ nodes in data reconstruction. Similarly, after $d$ nodes are accessed, if the integrity check fails, the newcomer will access all remaining surviving $n-d-1$ nodes in data regeneration. Each data point is generated from $10^4$ simulation results. Storage nodes may fail arbitrarily with the Byzantine failure probability ranging from 0 to 0.5. In both schemes, $[n,k,d]$ is chosen to be $[100,20,38]$.[10] Thus $\alpha = 19$, and the largest possible $n' = n+\ell = 100+100-1-2\times 20+2 = 241$. For MSR codes, as discussed in Section IV-B, for the finite field $GF(2^m)$, $m$ needs to satisfy $m \geq \lceil \log_2 n' \rceil$ and $\gcd(2^m-1, \alpha) = 1$. This gives $m = 8$ as $2^8 = 256 \geq 241$ is co-prime to 19. The finite field associated with MBR can be made smaller (e.g., $m = 7$). For simplicity, in the simulations, we select the same operating finite field $GF(2^8)$ for both MSR and MBR codes.

Figure 2 and Figure 3 show the performance of the MSR code in data reconstruction and regeneration, respectively. As shown in Fig. 2, as the failure probability of each node reduces, the average number of nodes accessed and the failure rate of reconstruction decrease. When the failure probability reduces to 0, there are $k = 20$ nodes that need to be accessed. This is due to the fact that at least $k$ nodes need to be

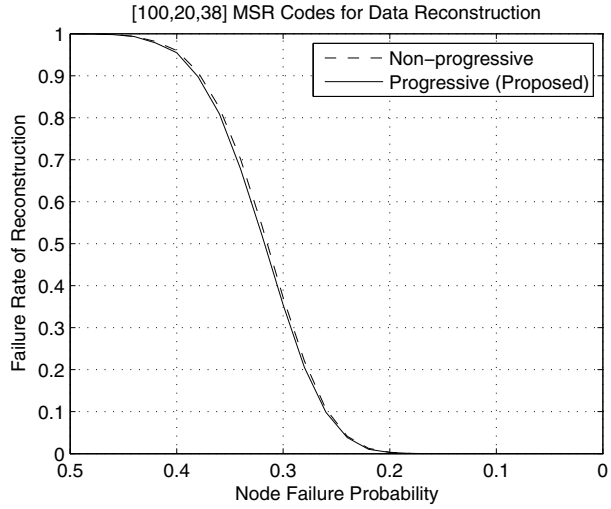
[10]Since the commonly used finite field size corresponds to $m = 8$ bits, the code $[100,20,38]$ is chosen to operate on $GF(2^8)$.

TABLE I
FAULT-TOLERACE CAPABILITY OF MSR AND MBR CODES

| | MSR code | | MBR code | |
|---|---|---|---|---|
| | Data reconstruction | Regeneration | Data reconstruction | Regeneration |
| Crash-stop failures | $n-k$ | $n-d$ | $n-k$ | $n-d$ |
| Byzantine faults | $\lfloor \frac{n-d}{2} \rfloor$ | $\min\{\lfloor \frac{n-d-1}{2} \rfloor, \lfloor \frac{d-1}{2} \rfloor\}$ | $\lfloor \frac{n-k}{2} \rfloor$ | $\min\{\lfloor \frac{n-d-1}{2} \rfloor, \lfloor \frac{d-1}{2} \rfloor\}$ |

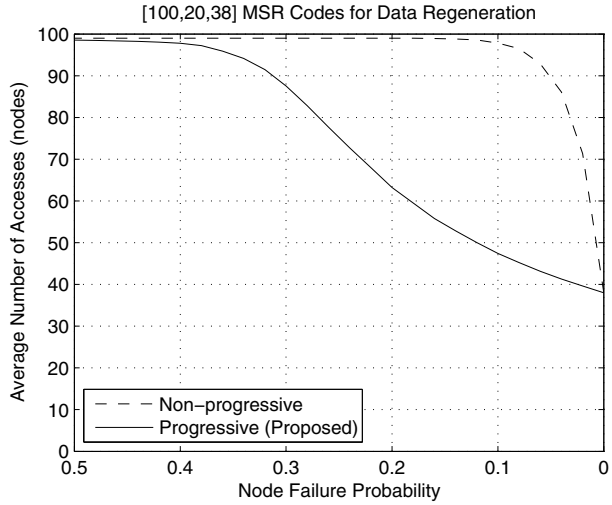



(a) Average number of accessed nodes

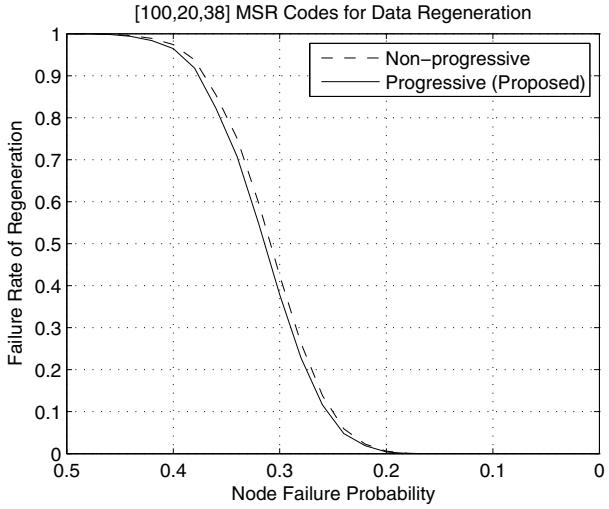



(b) Failure rate of data reconstruction

Fig. 2. The comparison between proposed progressive decoding scheme and the traditional non-progressive approach on varying node failure probability for data reconstruction using the $[100, 20, 38]$ MSR code.




(a) Average number of accessed nodes

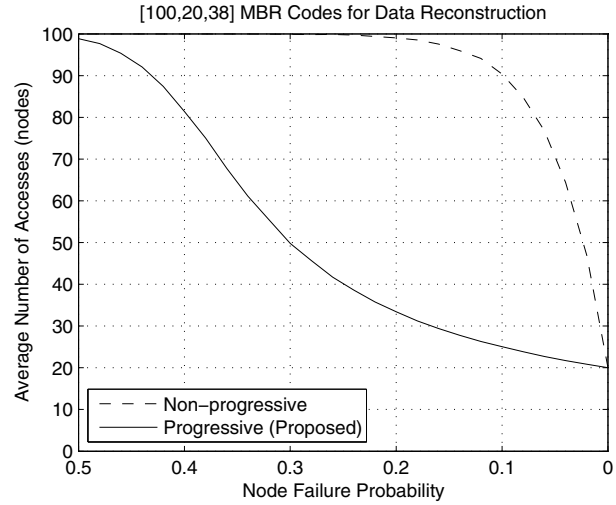



(b) Failure rate of data regeneration

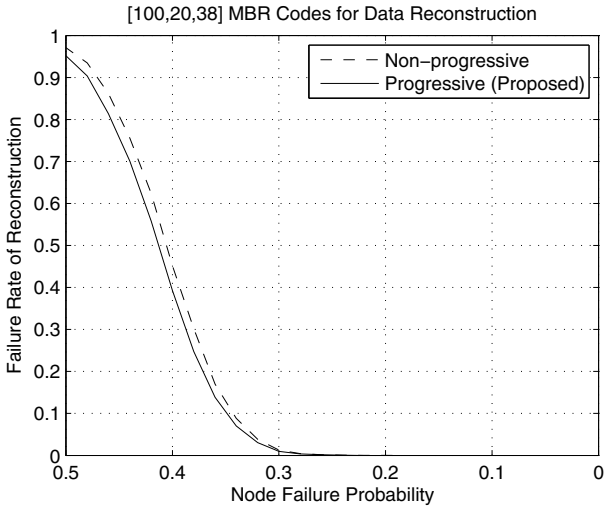
Fig. 3. The comparison between proposed progressive decoding scheme and the traditional non-progressive approach on varying node failure probability for data regeneration using the $[100, 20, 38]$ MSR code.

accessed in data reconstruction for the $[100, 20, 38]$ code. When the failure probability is 0.4, or roughly 40 nodes failed, one needs to retrieve data from all 100 nodes. Meanwhile, when the actual number of failure nodes is greater than 31, reconstruction is less feasible with MSR.[11] As a result, when the failure probability is around 0.31, roughly only half of the time, the reconstruction is successful. Similar observations can be made for regeneration as shown in Fig. 3. The main differences are i) when the failure probability is 0, $d = 38$ nodes need to be accessed, and ii) the maximum number of accessible nodes is $n - 1 = 99$ in presence of a single failure node. It is not surprising that the proposed decoding scheme has a smaller average number of accesses than the traditional one. Moreover, the advantage is more pronounced in presence of Byzantine faults. Specifically, the proposed scheme has a lower failure rate than the traditional one because the failure nodes may be not accessed in the progressive decoding when

[11]The error-erasure decoding adopted in the progressive decoding procedure in [16] can decode up to 80 errors; however, its decoding capability depends on the order of accessing extra nodes.

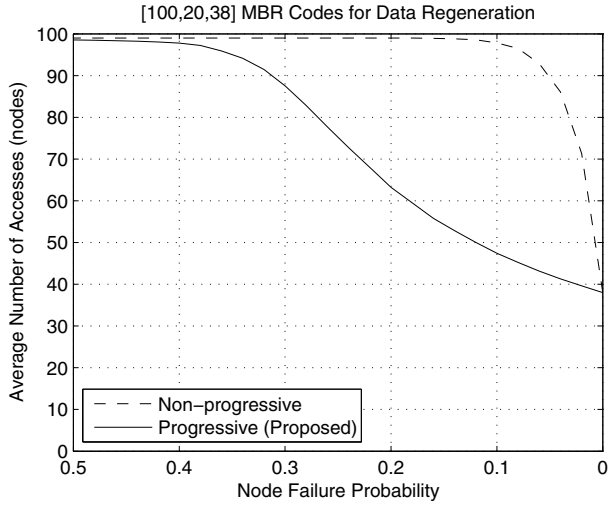



(a) Average number of accessed nodes

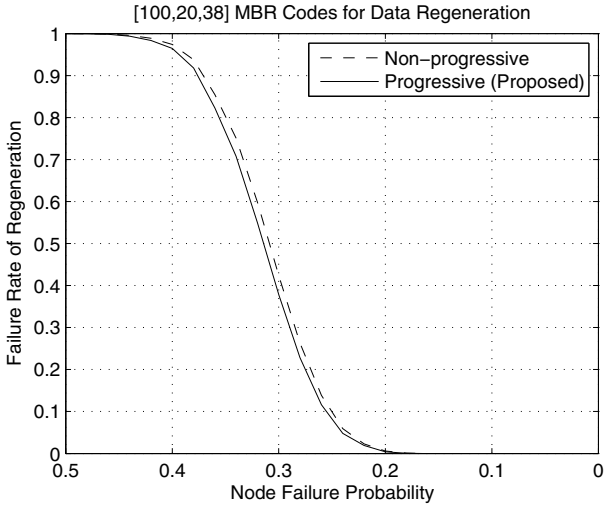



(b) Failure rate of data reconstruction

Fig. 4. The comparison between the proposed progressive decoding scheme and the traditional non-progressive approach on varying node failure probability for data reconstruction using the $[100, 20, 38]$ MBR code.




(a) Average number of accessed nodes




(b) Failure rate of data regeneration

Fig. 5. The comparison between proposed progressive decoding scheme and the traditional non-progressive approach on varying node failure probability for data regeneration using the $[100, 20, 38]$ MBR code.

the number of the failure nodes is larger than the fault-tolerance capability.

Figure 4 and Figure 5 show the performance of the MBR code in data reconstruction and regeneration, respectively. Again, as the failure probability reduces, the total number of nodes accessed and the failure rate of reconstruction and regeneration decrease. When comparing Fig. 4 with Fig. 2, we see a slower growth in both qualities as the failure probability increases. This is because in MBR, if there exists an error in the first $k = 20$ nodes accessed, only two extra nodes are needed for each error, while with MSR codes, $d - k = 18$ nodes need to be accessed first and two extra nodes are needed for each additional error (Algorithm 1). In contrast, the performance of the MBR code for code regeneration is identical to that of the MSR code. The advantage of the proposed scheme over the traditional non-progressive decoding scheme is similar to that in the case of the MSR code.

## VII. Conclusions

In this paper, the problem of exact regeneration with error correction capability for Byzantine fault tolerance in distributed storage networks has been addressed. We showed that the Reed-Solomon codes combined with cryptographic hash functions can be used for both data reconstruction and regeneration, and can realize both MSR and MBR. Progressive decoding can be applied in both application scenarios to reduce the computation complexity in presence of erroneous data. Finally, we analyzed the fault-tolerance capability of the proposed schemes and carried out Monte Carlo simulation studies. We found that both the average number of access nodes and the reconstruction failure probability were significantly lower with the the proposed schemes for MSR and MBR. The proposed MSR and MBR codes generalized our earlier results for MSR codes. By utilizing well-understood components such as RS codes and hash functions, our proposed schemes hold promises in practical implementation in both software and hardware.

The decoding schemes proposed here are for MSR and MBR codes based on the product-matrix construction. As future work, it will be interesting to see whether the similar idea can be extended to codes based on the interference alignment construction. Furthermore, it has been shown that exact regeneration is impossible for MSR codes with $[n, k, d < 2k - 3]$ when $\beta = 1$. That is, the rates of codes considered in this work are roughly less than $1/2$. In the future work, we would like to examine high-rate codes with functional regeneration in their Byzantine fault tolerance.

## REFERENCES


[1] A. G. Dimakis, P. B. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proc. 2007 IEEE International Conference on Computer Communications*, pp. 2000–2008.

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

[3] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Proc. 2007 Allerton Conference on Control, Computing, and Communication.*.

[4] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE J. Sel. Areas Commun.*, vol. 28, pp. 277–288, Feb. 2010.

[5] D. F. Cullina, "Searching for minimum storage regenerating codes," California Institute of Technology Senior Thesis, 2009.

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

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

[8] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," arXiv:1009.2556v2 [cs.IT] 27 Apr 2011.

[9] N. Shah, K. V. Rashmi, and P. Kumar, "Information-theoretically secure regenerating codes for distributed storage," in *Proc. 2011 IEEE Global Telecommunications Conference*, pp. 1–5.

[10] F. Oggier and A. Datta, "Byzantine fault tolerance of regenerating codes," arXiv:1106.2275v1 [cs.DC] 12 Jun 2011.

[11] 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, pp. 5227–5239, Aug. 2011.

[12] K. Rashmi, N. Shah, K. Ramchandran, and P. Kumar, "Regenerating codes for errors and erasures in distributed storage," in *Proc. 2012 IEEE International Symposium on Information Theory.*

[13] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, pp. 476–489, Mar. 2011.

[14] M. Gerami, M. Xiao, C. Fischione, and M. Skoglund, "Decentralized minimum-cost repair for distributed storage systems," in *Proc. 2013 IEEE International Conference on Communications*, pp. 1910–1914.

[15] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: necessity and code constructions," arXiv:1005.1634v2[cs.IT] 13 Sep 2010.

[16] Y. S. Han, S. Omiwade, and R. Zheng, "Progressive data retrieval for distributed networked storage," *IEEE Trans. Parallel and Distrib. Syst.*, pp. 2303–2314, Dec. 2012.

[17] M. Gerami and M. Xiao, "Repair for distributed storage systems with erasure channels," in *Proc. 2013 IEEE International Conference on Communications*, pp. 4058–4062.

[18] M. Gerami, M. Xiao, and M. Skoglund, "Optimal-cost repair in multi-hop distributed storage system," in *Proc. 2011 IEEE International Symposium on Information Theory*, pp. 1437–1441.

[19] M. Kurihara and H. Kuwakado, "Secure regenerating codes based on Rashmi-Shah-Kumar MBR codes," *IEICE Trans. Fundamentals of Electron., Commun. and Computer Sciences*, vol. E96-A, no. 2, pp. 635–648, 2013.

[20] R. Kötter and F. R. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Trans. Inf. Theory*, vol. 54, pp. 3579–3591, Aug. 2008.

[21] Y. S. Han, R. Zheng, and W. H. Mow, "Exact regenerating codes for Byzantine fault tolerance in distributed storage," in *Proc. 2012 IEEE INFOCOM.*

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

[23] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. John Wiley & Sons, Inc., 2005.

[24] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," ICSI Technical Report TR-95-048, 1995.

[25] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[26] R. C. Merkle, "A certified digital signature," in *Proc. 1989 CRYPTO*, pp. 218–238.

[27] I. Damgård, "A design principle for hash functions," in *Proc. 1989 CRYPTO*, pp. 416–427.

[28] "Secure Hash Standard (SHS)," FIPS PUB 180-4.


**Yunghsiang S. Han** (S'90–M'93–SM'08–F'11) was born in Taipei, Taiwan, 1962. He received B.Sc. and M.Sc. degrees in electrical engineering from the National Tsing Hua University, Hsinchu, Taiwan, in 1984 and 1986, respectively, and a Ph.D. degree from the School of Computer and Information Science, Syracuse University, Syracuse, NY, in 1993.

He was from 1986 to 1988 a lecturer at Ming-Hsin Engineering College, Hsinchu, Taiwan. He was a teaching assistant from 1989 to 1992, and a research associate in the School of Computer and Information Science, Syracuse University from 1992 to 1993. He was, from 1993 to 1997, an Associate Professor in the Department of Electronic Engineering at Hua Fan College of Humanities and Technology, Taipei Hsien, Taiwan. He was with the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou, Taiwan from 1997 to 2004. He was promoted to Professor in 1998. He was a visiting scholar in the Department of Electrical Engineering at University of Hawaii at Manoa, HI from June to October 2001, the SUPRIA visiting research scholar in the Department of Electrical Engineering and Computer Science and CASE center at Syracuse University, NY from September 2002 to January 2004 and July 2012 to June 2013, and the visiting scholar in the Department of Electrical and Computer Engineering at University of Texas at Austin, TX from August 2008 to June 2009. He was with the Graduate Institute of Communication Engineering at National Taipei University, Taipei, Taiwan from August 2004 to July 2010. From August 2010, he is with the Department of Electrical Engineering at National Taiwan University of Science and Technology as Chair professor. His research interests are in error-control coding, wireless networks, and security.

Dr. Han was a winner of the 1994 Syracuse University Doctoral Prize and a Fellow of IEEE.

**Hung-Ta Pai** (S'91–M'99–SM'11) was born in Taichung, Taiwan. He received a B.Sc. degree from National Tsing Hua University, Taiwan, in 1992, and the M.Sc. and Ph.D. degrees from the University of Texas at Austin, Texas, in 1996 and 1999, respectively, all in electrical engineering. Hung-Ta Pai served as an Army officer from 1992 to 1994. He worked as a senior design engineer at Silicon Integrated Systems Corp from 1999 to 2001. He was, from 2001 to 2002, an assistant professor at the Department of Electrical Engineering, Tatung University, Taiwan. He was an assistant professor at the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taiwan from 2002 to 2004. He was a visiting scholar in the Department of Electrical and Computer Engineering at the University of Texas at Austin from August 2010 to July 2011. He has since August 2004 been with National Taipei University, Taiwan where he is currently a professor of the Department of Communication Engineering.

His research interests include communication systems and signal processing.

**Rong Zheng** (S'03-M'04-SM'10) received her Ph.D. degree from Dept. of Computer Science, University of Illinois at Urbana-Champaign and earned her M.E. and B.E. in Electrical Engineering from Tsinghua University, P.R. China. She is on the faculty of the Department of Computing and Software, McMaster University. She was with University of Houston between 2004 and 2012. Rong Zheng's research interests include network monitoring and diagnosis, cyber physical systems, and sequential learning and decision theory. She received the National Science Foundation CAREER Award in 2006. She serves on the technical program committees of leading networking conferences including INFOCOM, ICDCS, ICNP, etc. She served as a guest editor for *EURASIP Journal on Advances in Signal Processing*, Special issue on wireless location estimation and tracking, *Elsevlers Computer Communications* Special Issue on Cyber Physical Systems; and Program co-chair of WASA'12 and CPSCom'12.

**Wai Ho Mow** (S'89-M'93-SM'99) received his M.Phil. and Ph.D. degrees in information engineering from the Chinese University of Hong Kong in 1991 and 1993, respectively. From 1997 to 1999, he was with the Nanyang Technological University, Singapore. He has been with the Hong Kong University of Science and Technology since March 2000. He was the recipient of seven research/exchange fellowships from ve countries, including the Humboldt Research Fellowship. His research interests are in the areas of communications, coding, and information theory. He pioneered the lattice approach to signal detection problems (such as sphere decoding and complex lattice reduction-aided detection) and unied all known constructions of perfect roots-of-unity (or CAZAC) sequences (widely used as packet preambles and radar signals). Since 2002, he has been the principal investigator of over 16 funded research projects. He published one book, and coauthored over 30 filed patent applications and over 160 technical publications, among which he is the sole author of over 40. His coauthored papers won the APCC'2013 Best Paper Award and the ISITA'2002 Paper Award for Young Researchers. The mobile app PiCode developed by his barcode team won the first ever Best Mobile App Award at ACM MobiCom'2013. In 2005, he chaired the Hong Kong Chapter of the IEEE Information Theory Society, which won the 2006 ITSoc Chapter of the Year Award. He was the Technical Program Co-Chair of five conferences, and served the technical program committees of numerous conferences, such as ICC, Globecom, WCNC, ITW, ISITA, VTC and APCC. He was a Guest Associate Editor for numerous special issues of the *IEICE Transactions on Fundamentals*. He was a member of the Radio Spectrum Advisory Committee, Office of the Telecommunications Authority, Hong Kong S.A.R. Government from 2003 to 2008, and was a past consultant of Huawei, ZTE, and Magnotech Ltd.