entropy

MDPI

Article

Three Efficient All-Erasure Decoding Methods for Blaum–Roth Codes

Weijie Zhou [1,†] and Hanxu Hou [2,*,†]

[1] School of Computer Science and Technology, Dongguan University of Technology, Dongguan 523820, China
[2] School of Electrical Engineering and Intelligentization, Dongguan University of Technology, Dongguan 523820, China
* Correspondence: houhanxu@163.com
† These authors contributed equally to this work.

Abstract: Blaum–Roth Codes are binary maximum distance separable (MDS) array codes over the binary quotient ring $\mathbb{F}_2[x]/(M_p(x))$, where $M_p(x) = 1 + x + \cdots + x^{p-1}$, and p is a prime number. Two existing all-erasure decoding methods for Blaum–Roth codes are the syndrome-based decoding method and the interpolation-based decoding method. In this paper, we propose a modified syndrome-based decoding method and a modified interpolation-based decoding method that have lower decoding complexity than the syndrome-based decoding method and the interpolation-based decoding method, respectively. Moreover, we present a fast decoding method for Blaum–Roth codes based on the LU decomposition of the Vandermonde matrix that has a lower decoding complexity than the two modified decoding methods for most of the parameters.

Keywords: distributed storage; Blaum–Roth codes; all-erasure decoding; decoding complexity

check for updates

Citation: Zhou, W.; Hou, H. Three Efficient All-Erasure Decoding Methods for Blaum–Roth Codes. *Entropy* **2022**, *24*, 1499. https://doi.org/10.3390/e24101499

Academic Editors: Shenghao Yang and Kenneth Shum

Received: 5 September 2022
Accepted: 17 October 2022
Published: 20 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

Redundancy is necessary in storage systems in order to provide high data reliability in case of disk failures [1]. Replication and erasure codes are two main ways of including redundancy. The idea of replication is that the data in one disk are copied to multiple disks. The storage system replaces damaged disks with their copies when some disks are erased. It is fast to repair the erased disks but requires a lot of storage space. In contrast, erasure codes provide higher data reliability with a small storage cost.

Maximum distance separable (MDS) codes [2] are typical erasure codes that have optimal tradeoff between storage cost and data reliability, i.e., they can achieve the minimum storage cost given a level of data reliability. Binary MDS codes are special MDS codes that have lower computational complexity in the encoding/decoding procedures, since only XORs and cyclic-shift operations are involved. Some existing constructions of binary MDS codes are EVENODD codes [3,4], RDP codes [5], and X-codes [6,7], which can correct any two-column (we use "column" and "disk" interchangeably in this paper) erasures. RTP codes [8], Star codes [9,10], and extended EVENODD codes [11–14] can correct any three-column erasures. With the rapid increase in the data scale in storage systems [15], we need to design binary MDS codes that can correct any number of erasures as well as efficient encoding/decoding methods. Graftage codes [16] can achieve various tradeoffs between storage and repair bandwidth, while we focus on efficient decoding methods of binary MDS codes. Blaum–Roth codes [17] are this type of code, which are designed over the ring $\mathcal{R}_p = \mathbb{F}_2[x]/(M_p(x))$, where $M_p(x) = 1 + x + \cdots + x^{p-1}$, and p is a prime number.

When some columns are erased, the syndrome-based decoding method [17] and the interpolation-based decoding method [18] have been proposed to recover the erased columns. In the decoding methods [17,18], there are three basic operations over the ring $\mathcal{R}_p$: (i) addition, (ii) multiplication of a power of x and a polynomial, and (iii) division of factor $1 + x^b$ with $1 \leq b \leq p - 1$. It is shown in the decoding methods [17,18] that we can first take

the operations (i) and (ii) modulo $1+x^p$ and then take the results of modulo $M_p(x)$, while operation (iii) in the decoding methods [17,18] is directly taken as modulo $M_p(x)$.

In this paper, we show that we can also compute operation (iii) as modulo $1+x^p$, which has lower computational complexity than modulo $M_p(x)$. We propose modified decoding methods for the two existing decoding methods [17,18] that have a lower decoding complexity than the original decoding methods by computing operation (iii) as modulo $1+x^p$ instead of modulo $M_p(x)$. The reason our modified decoding methods have much lower decoding complexity than the decoding methods [17,18] is twofold. First, all the operations in our decoding methods are taken as modulo $1+x^p$, while the existing decoding methods execute the divisions as modulo $M_p(x)$. Second, we propose new algorithms in the decoding procedure to reduce the number of operations. Please refer to Section 3 for our two modified decoding methods. Moreover, the efficient LU decoding method [19] proposed for extended EVENODD codes decoding can also be employed to recover the erased columns of Blaum–Roth codes. We show that the LU decoding method has lower decoding complexity than the two modified decoding methods for most of the parameters. We define the decoding complexity as the total number of XORs required to recover the erased columns.

2. Blaum–Roth Codes

In this section, we first review the construction of Blaum–Roth codes [17] and then show the efficient operations over the ring $\mathbb{F}_2[x]/(1+x^p)$. Finally, we present an algorithm to compute multiple multiplications, which have two nonzero terms over $\mathbb{F}_2[x]/(1+x^p)$ with lower complexity.

2.1. Construction of Blaum–Roth Codes [17]

The codeword of Blaum–Roth codes [17] is a $(p-1)\times n$ array $[c_{i,j}]_{i=0,j=0}^{p-2,n-1}$ that is encoded from the $(p-1)k$ information bits, where $c_{i,j}\in\mathbb{F}_2$ and $n\le p$. We can view any k columns of the $(p-1)\times n$ array as information columns that store the $(p-1)k$ information bits and the other $r=n-k$ columns as parity columns that store the $(p-1)r$ parity bits. For $j=0,1,\ldots,n-1$, we represent the $p-1$ bits in column j by a polynomial $c_j(x)=\sum_{i=0}^{p-2}c_{i,j}x^i$. The $(p-1)\times n$ array of Blaum–Roth codes is defined as

$$\begin{pmatrix} c_0(x) & c_1(x) & \cdots & c_{n-1}(x)\end{pmatrix}\cdot\mathbf{H}_{r\times n}^T\equiv\mathbf{0}\pmod{M_p(x)},$$

where $\mathbf{H}_{r\times n}$ is the $r\times n$ parity-check matrix

$$\mathbf{H}_{r\times n}=\begin{bmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & x & x^2 & \cdots & x^{n-1}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & x^{(r-1)} & x^{(r-1)2} & \cdots & x^{(r-1)(n-1)}\end{bmatrix},$$

and $\mathbf{0}$ is the all-zero row of length r. We denote the Blaum–Roth codes defined above as $\mathcal{C}(p,n,r)$. When $p\ge n$ and p is a prime number, we can always retrieve all the information bits from any k out of the n polynomials [17], i.e., $\mathcal{C}(p,n,r)$ are MDS codes.

If we let $c_{p-1,j}=0$ for all $j=0,1,\ldots,n-1$, then $\mathcal{C}(p,n,r)$ can be equivalently defined as the following $p\cdot r$ linear constraints. (The subscripts are taken as modulo p unless otherwise specified.)

$$\sum_{j=0}^{n-1}c_{\langle m-\ell\cdot j\rangle_p,j}=0,$$

where $0\le m\le p-1$ and $0\le\ell\le r-1$.

Suppose that the λ columns $\{e_i\}_{i=0}^{\lambda-1}$ are erased, where $\lambda\ge 2$ and $0\le e_0<\cdots<e_{\lambda-1}<n$. Let the $\delta=n-\lambda$ surviving columns be $\{h_j\}_{j=0}^{\delta-1}$, where $0\le h_0<\cdots<h_{\delta-1}<n$ and $\{e_i\}_{i=0}^{\lambda-1}\cup\{h_j\}_{j=0}^{\delta-1}=\{0,1,\ldots,n-1\}$. We have

$$\begin{pmatrix} c_{e_0}(x) & c_{e_1}(x) & \cdots & c_{e_{\lambda-1}}(x) \end{pmatrix} \cdot \mathbf{V}_{\lambda\times\lambda}^T = \mathbf{S}, \tag{1}$$

over the ring $\mathcal{R}_p$, where $\mathbf{V}_{\lambda\times\lambda}$ is the $\lambda \times \lambda$ square

$$\mathbf{V}_{\lambda\times\lambda} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x^{e_0} & x^{e_1} & x^{e_2} & \cdots & x^{e_{\lambda-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x^{(\lambda-1)e_0} & x^{(\lambda-1)e_1} & x^{(\lambda-1)e_2} & \cdots & x^{(\lambda-1)e_{\lambda-1}} \end{bmatrix},$$

and $\mathbf{S} = \begin{pmatrix} S_0(x) & S_1(x) & \cdots & S_{\lambda-1}(x) \end{pmatrix}$, where the λ syndrome polynomials are

$$S_\ell(x) = \sum_{j=0}^{\delta-1} x^{\ell\cdot h_j} c_{h_j}(x) \text{ for } 0 \le \ell \le \lambda - 1. \tag{2}$$

In this paper, we present three efficient decoding methods to solve the linear systems in Equation (1) over the ring $\mathbb{F}_2[x]/(1+x^p)$.

2.2. Efficient Operations over $\mathbb{F}_2[x]/(1+x^p)$

It is more efficient to compute the multiplication of a power of x and division of the factor $1+x^b$ over the ring $\mathbb{F}_2[x]/(1+x^p)$ rather than over the ring $\mathcal{R}_p$: (i) Let $a(x) \in \mathcal{R}_p$, and the multiplication $x^i \cdot a(x)$ over the ring $\mathcal{R}_p$ in [17] (Equation (19)) takes $p-1$ XORs, while the multiplication $x^i \cdot a(x)$ over the ring $\mathbb{F}_2[x]/(1+x^p)$ takes no XORs [20]. (ii) Let $g(x), f(x) \in \mathbb{F}_2[x]/(1+x^p)$, where d is a positive integer, which is coprime with p. Consider the equation

$$(1+x^d)g(x) \equiv f(x) \pmod{1+x^p}, \tag{3}$$

where $f(x)$ has an even number of nonzero terms. Given such $f(x)$ and d, we can compute $g(x)$ by Lemma 1.

Lemma 1. *[Lemma 8] in [21] The coefficients of $g(x)$ in Equation (3) are given by*

$$g_{p-1} = 0,\ g_{p-d-1} = f_{p-1},\ g_{d-1} = f_{d-1},$$
$$g_{p-(\ell+1)d-1} = g_{p-\ell d-1} + f_{p-\ell d-1} \text{ for } \ell = 1, 2, \ldots, p-3.$$

By Lemma 1, computing the division $\frac{f(x)}{1+x^d}$ takes $p-3$ XORs, but we are not sure whether $g(x)$ has an even number of nonzero terms or not. If we want to guarantee that $g(x)$ has an even number of nonzero terms, we should use Lemma 2 to compute the division $\frac{f(x)}{1+x^d}$.

Lemma 2. *[Lemma 13] in [20] The coefficients of $g(x)$ in Equation (3) are given by*

$$g_0 = f_{2d} + f_{4d} + \cdots + f_{(p-1)d},$$
$$g_{\ell d} = g_{(\ell-1)d} + f_{\ell d} \text{ for } \ell = 1, 2, \ldots, p-1.$$

By Lemma 2, the division $\frac{f(x)}{1+x^d}$ takes $\frac{3p-5}{2}$ XORs, and $g(x)$ has an even number of nonzero terms. However, computing the division $\frac{f(x)}{1+x^d}$ in [Corollary 2] in [17] takes $2(p-1)$ XORs over the ring $\mathcal{R}_p$, which is strictly larger than the decoding methods in Lemmas 1 and 2. It is shown in [Theorem 5] in [19] that we can always solve the equations in Equation (1) over the ring $\mathbb{F}_2[x]/(1+x^p)$ of which all the solutions are congruent to each other after modulo $M_p(x)$. Therefore, we can first solve the equations in Equation (1) over the ring $\mathbb{F}_2[x]/(1+x^p)$ and then obtain the unique solution by taking modulo $M_p(x)$ to reduce the computational complexity.

2.3. Multiple Multiplications over $\mathbb{F}_2[x]/(1+x^p)$

Note that in our modified syndrome-based decoding method and the modified interpolation-based decoding method, we need to compute multiple polynomial multiplications, where each polynomial has two nonzero terms. Suppose that we want to compute the following m multiplications

$$\mathcal{L}(x^\tau) = \prod_{i=0}^{m-1}(x^\tau - x^{\xi_i}) \pmod{1+x^p}, \tag{4}$$

where m is a positive integer, $0 \le \tau \le p-1$ such that $\tau \notin \{\xi_0, \xi_1, \ldots, \xi_{m-1}\}$, and $0 \le \xi_0 < \cdots < \xi_{m-1} < n$.

We can derive from Equation (4) that

$$\mathcal{L}(x^\tau) = x^\pi \cdot \prod_{i=0}^{m-1}(1+x^{d_i}) \pmod{1+x^p}, \tag{5}$$

where $\pi = \sum_{i=0}^{m-1} \min(\tau, \xi_i)$ modulo p and $d_i = |\tau - \xi_i|$ for $i = 0, 1, \ldots, m-1$.

Algorithm 1 presents a method to simplify the multiplications in Equation (4). In Algorithm 1, we use Γ_ℓ to denote the number of the polynomial $1+x^\ell$ in the multiplication $\mathcal{L}(x^\tau)$. Note that we only need to count the number of $1+x^\ell$ for $1 \le \ell \le \frac{p-1}{2}$, because the equation $1+x^\ell \equiv x^\ell \cdot (1+x^{p-\ell})$ modulo $1+x^p$ holds for $\frac{p-1}{2} < \ell < n$. If $\Gamma_\ell > 1$, then we have $(1+x^\ell)^{\Gamma_\ell} = (1+x^\ell)^{\Gamma_\ell - 2\lfloor \frac{\Gamma_\ell}{2} \rfloor} \cdot (1+x^{2\ell})^{\lfloor \frac{\Gamma_\ell}{2} \rfloor}$. Therefore, we can always merge Γ_ℓ multiplications $(1+x^\ell)^{\Gamma_\ell}$ into $\Gamma_\ell - \lfloor \frac{\Gamma_\ell}{2} \rfloor$ multiplications and the computational complexity can be reduced with Algorithm 1. When Algorithm 1 is executed, all elements of count-array Γ should be zero or one, and the length η of the final $\mathcal{L}(x^\tau)$ is between 1 and m.

Algorithm 1: Simplify the multiple multiplications.

Data: $\mathcal{L}(x^\tau) = \prod_{i=0}^{m-1}(x^\tau - x^{\xi_i})$
$\pi \leftarrow \sum_{i=0}^{m-1} \min(\tau, \xi_i) \pmod{p}$;
count-array $\Gamma[\frac{p-1}{2}] = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_{\frac{p-1}{2}}\} \leftarrow \{\mathbf{0}\}$;
for $i \leftarrow 0$ **to** $m-1$ **do** // Hash.
  $d_i \leftarrow |\tau - \xi_i| \bmod p$;
  if $d_i \le \frac{p-1}{2}$ **then** $\Gamma_{d_i} \leftarrow \Gamma_{d_i} + 1$;
  else // Use $1 + x^{d_i} \equiv x^{d_i} \cdot (1 + x^{p-d_i})$
    $\pi \leftarrow (\pi + d_i) \bmod p$;
    $\Gamma_{p-d_i} \leftarrow \Gamma_{p-d_i} + 1$;
$\omega \leftarrow 0$;
while $\omega \ne \frac{p-1}{2}$ **do**
  for $\ell \leftarrow 1$ **to** $\frac{p-1}{2}$ **do** // Merge Multiplications $(1+x^\ell)^{\Gamma_\ell}$
    if $\Gamma_\ell \le 1$ **then Continue**;
    if $2\ell \le \frac{p-1}{2}$ **then** $\Gamma_{2\ell} \leftarrow \Gamma_{2\ell} + \lfloor \frac{\Gamma_\ell}{2} \rfloor$;
    else // Use $1 + x^{2\ell} \equiv x^{2\ell} \cdot (1 + x^{p-2\ell})$
      $\pi \leftarrow (\pi + 2\lfloor \frac{\Gamma_\ell}{2} \rfloor \ell) \bmod p$;
      $\Gamma_{p-2\ell} \leftarrow \Gamma_{p-2\ell} + \lfloor \frac{\Gamma_\ell}{2} \rfloor$;
    $\Gamma_\ell \leftarrow \Gamma_\ell - 2\lfloor \frac{\Gamma_\ell}{2} \rfloor$;
  $\omega \leftarrow$ the amount of elements no greater than one in count-array Γ;
$\{\xi_i\}_{i=0}^{\eta-1} \leftarrow$ the subscript of one in count-array Γ, i.e., $\Gamma_{\xi_i} = 1$;
return $x^\pi \cdot \prod_{i=0}^{\eta-1}(1+x^{\xi_i})$;

3. Decoding Algorithm

In this section, we present two decoding methods over the ring $\mathbb{F}_2[x]/(1+x^p)$ by modifying two existing decoding methods [17,18] that can reduce the decoding complexity.

Recall that the λ erased columns are λ columns $\{e_i\}_{i=0}^{\lambda-1}$, and the $\delta = n - \lambda$ surviving columns are δ columns $\{h_j\}_{j=0}^{\delta-1}$.

3.1. Modified Syndrome-Based Method

We define the function of the indeterminate z

$$G_i(z) = \prod_{s=0,\neq i}^{\lambda-1}(1 - x^{e_s}z) = \sum_{\ell=0}^{\lambda-1} G_{i,\ell}(x)z^\ell,$$

and the syndrome function $S(z) = \sum_{\ell=0}^{r-1} S_\ell(x)z^\ell$, where $0 \le i \le \lambda - 1$ and $S_\ell(x)$ is given in Equation (2). We can obtain in [Equation (18)] in [17] that

$$\begin{aligned}\prod_{s=0,\neq i}^{\lambda-1}(x^{e_i} - x^{e_s})c_{e_i}(x) &\equiv \sum_{\ell=0}^{\lambda-1} G_{i,\lambda-1-\ell}(x)S_\ell(x)\\ &\equiv \sigma_i(x) \pmod{M_p(x)}.\end{aligned}$$

Therefore, the $\sigma_i(x)$ can be regarded as the coefficient of $z^{\lambda-1}$ of the polynomial $G_i(z)S(z)$. Then, the erased column $c_{e_i}(x)$ is given by $\frac{\sigma_i(x)}{\prod_{s=0,\neq i}^{\lambda-1}(x^{e_i}-x^{e_s})}$, where $0 \le i \le \lambda - 1$.

Note that the terms of set $\{S_\ell(x)z^\ell\}_{\ell=\lambda}^{r-1}$ are not involved in computing the coefficient of $z^{\lambda-1}$ of the polynomial $G_i(z)S(z)$. Thus, we can just consider the first λ terms (the λ coefficients of degrees less than λ) of $S(z)$ when computing these coefficients, but all the r terms of $S(z)$ are calculated in [Step 1] in [17]. This is one essential way our modified syndrome-based decoding method obtains a lower decoding complexity than the original method in [17].

Moreover, the syndrome polynomials $S_\ell(x)$ satisfy

$$S_0(1) = S_1(1) = \cdots = S_{\lambda-1}(1), \tag{6}$$

i.e., the λ syndrome polynomials $S_\ell(x)$ either all have an even number of nonzero terms, or they all have an odd number of nonzero terms, from the definition of Equation (2).

Let $G(z) = (1 - x^{e_i}z)G_i(z)$ and $Q(z) = G(z)S(z)$. Then, we have

$$\begin{aligned}Q(z) &= (1 - x^{e_i}z)\prod_{s=0,\neq i}^{\lambda-1}(1 - x^{e_s}z)S(z)\\ &= \prod_{s=0}^{\lambda-1}(1 - x^{e_s}z)S(z) = \sum_{\ell=0}^{r+\lambda-1} Q_\ell(x)z^\ell.\end{aligned} \tag{7}$$

Thus, $Q(z)$ is independent of the erasure index i, and we only need to compute $Q(z)$ once in the decoding procedure. Recall that $\sigma_i(x)$ is the coefficient of $z^{\lambda-1}$ of the polynomial $G_i(z)S(z)$; then, the $\sigma_i(x)$ is also the coefficient of $z^{\lambda-1}$ of the polynomial $\frac{Q(z)}{(1-x^{e_i}z)} = \frac{(1-x^{e_i}z)G_i(z)S(z)}{(1-x^{e_i}z)}$ for all $0 \le i \le \lambda - 1$. Suppose that

$$\frac{Q(z)}{(1 - x^{e_i}z)} = f_0^i(x) + f_1^i(x)z + \cdots + f_{\lambda-1}^i(x)z^{\lambda-1} + \cdots,$$

we can derive the recurrence formula

$$f_\ell^i(x) = \begin{cases} Q_0(x), & \ell = 0;\\ x^{e_i} \cdot f_{\ell-1}^i(x) + Q_\ell(x), & \ell > 0;\end{cases} \tag{8}$$

where $0 \leq i \leq \lambda - 1$. Notice that $\sigma_i(x) = f^i_{\lambda-1}(x)$ holds. Similar to $S(z)$, we only compute the first λ terms (the λ coefficients of degrees less than λ) of $Q(z)$, since the other coefficients of $Q(z)$ are not needed, but all the $r + \lambda$ terms of $Q(z)$ are calculated in [Step 2] in [17]. This is another way our modified syndrome-based decoding method obtains a lower decoding complexity than the original method in [17]. Algorithm 2 shows our modified syndrome-based decoding method over the ring $\mathbb{F}_2[x]/(1 + x^p)$.

The following Lemma shows that we can always compute the divisions in steps 11–12 of Algorithm 2 by Lemmas 1 and 2 when $\lambda \geq 2$.

Lemma 3. *In steps 11–12 of Algorithm 2, the $\sigma_i(x)$ has an even number of nonzero terms for all $0 \leq i \leq \lambda - 1$, and we can employ Lemmas 1 and 2 to compute the divisions.*

Proof. From Equation (8) and steps 7–10 of Algorithm 2, we obtain

$$\sigma_i(x) = x^{(\lambda-1)e_i} Q_0(x) + x^{(\lambda-2)e_i} Q_1(x) + \cdots + Q_{\lambda-1}(x),$$

where $0 \leq i \leq \lambda - 1$. If the number of polynomials in the set $\{Q_j(x)\}_{j=0}^{\lambda-1}$, which has an odd number of nonzero terms, is an even number, then the $\sigma_i(x)$ has an even number of nonzero terms for $0 \leq i \leq \lambda - 1$. In the following, we will show this is true. According to Equation (6) and step 3 of Algorithm 2, $Q_0(1) = \cdots = Q_{\lambda-1}(1)$ holds.

Firstly, we consider $Q_0(1) = \cdots = Q_{\lambda-1}(1) = 1$. We denote the λ polynomials $\{Q_j(x)\}_{j=0}^{\lambda-1}$ with $\varepsilon = 0, 1, \ldots, \lambda$ as $\{Q_j^\varepsilon(x)\}_{j=0}^{\lambda-1}$. Let $Q_j^0(x)$ be the initial $Q_j(x)$ for $0 \leq j \leq \lambda - 1$.

To prove that the number of polynomials with an odd number of nonzero terms in the set $\{Q_j^\varepsilon(x)\}_{j=0}^{\lambda-1}$ is even, it is equivalent to prove that $\sum_{j=0}^{\lambda-1} Q_j^\varepsilon(1) = 0$.

Algorithm 2: Modified syndrome-based decoding method.

Input: The λ erased columns $\{e_i\}_{i=0}^{\lambda-1}$ and the $\delta = n - \lambda$ surviving columns $\{h_j\}_{j=0}^{\delta-1}$.

1 **for** $\ell \leftarrow 0$ **to** $\lambda - 1$ **do** // Use Equation (2) and subscript ℓ means slope.
2 $\quad S_\ell(x) \leftarrow \sum_{j=0}^{\delta-1} x^{\ell \cdot h_j} c_{h_j}(x)$;
3 $Q(z) = \sum_{\ell=0}^{\lambda-1} Q_\ell(x) z^\ell \leftarrow S(z) = \sum_{\ell=0}^{\lambda-1} S_\ell(x) z^\ell$;
4 **for** $s \leftarrow 0$ **to** $\lambda - 1$ **do** // Use Equation (7).
5 $\quad$ **for** $\ell \leftarrow \lambda - 1$ **down to** 1 **do** // Calculate $Q_\ell(x)$ by backward additions.
6 $\quad\quad Q_\ell(x) \leftarrow x^{e_s} \cdot Q_{\ell-1}(x) + Q_\ell(x)$;
7 **for** $i \leftarrow 0$ **to** $\lambda - 1$ **do** // Use Equation (8).
8 $\quad \sigma_i(x) \leftarrow Q_0(x)$;
9 $\quad$ **for** $\ell \leftarrow 1$ **to** $\lambda - 1$ **do**
10 $\quad\quad \sigma_i(x) \leftarrow x^{e_i} \cdot \sigma_i(x) + Q_\ell(x)$;
11 **for** $i \leftarrow 0$ **to** $\lambda - 1$ **do** // Apply Algorithm 1.
12 $\quad c_{e_i}(x) \leftarrow \frac{\sigma_i(x)}{\prod_{s=0,\neq i}^{\lambda-1}(x^{e_i} - x^{e_s})}$;

Output: The erased columns $\{c_{e_i}(x)\}_{i=0}^{\lambda-1}$.

According to Equation (7) and steps 4–6 of Algorithm 2, we have

$$Q_j^\varepsilon(1) = \begin{cases} Q_j^{\varepsilon-1}(1), & j = 0; \\ Q_{j-1}^{\varepsilon-1}(1) + Q_j^{\varepsilon-1}(1), & 1 \leq j \leq \lambda - 1; \end{cases} \tag{9}$$

where $\varepsilon = 1, 2, \ldots, \lambda$. The $Q_j^1(1) = 0$ holds for all $j \geq 1$. We can obtain by induction

$$Q_j^{\varepsilon}(1) = Q_{j-1}^{\varepsilon-1}(1) + Q_j^{\varepsilon-1}(1) = 0 \text{ for all } j \geq \varepsilon \geq 1. \tag{10}$$

Note that $\sum_{j=0}^{\lambda-1} Q_j^2(1) = 0$; we can suppose that there are an even number of polynomials in the set $\{Q_j^{\varepsilon}(x)\}_{j=0}^{\lambda-1}$, which has an odd number of nonzero terms, when $\varepsilon = y \geq 2$, i.e., $\sum_{j=0}^{\lambda-1} Q_j^y(1) = 0$ first. We have $\sum_{j=0}^{\lambda-1} Q_j^{y+1}(1)$; so,

$$\begin{aligned} \sum_{j=0}^{\lambda-1} Q_j^{y+1}(1) &= Q_0^y(1) + \sum_{j=1}^{\lambda-1} \left(Q_{j-1}^y(1) + Q_j^y(1)\right) \\ &= \sum_{j=0}^{\lambda-1} Q_j^y(1) + \sum_{j=0}^{\lambda-2} Q_j^y(1) \\ &= Q_{\lambda-1}^y(1) = 0. \end{aligned} \tag{11}$$

Equation (11) comes from Equation (10) with $j = \lambda - 1$. Therefore, there are an even number of polynomials in the set $\{Q_j^{y+1}(x)\}_{j=0}^{\lambda-1}$, which has an odd number of nonzero terms.

Secondly, when $Q_0(1) = \cdots = Q_{\lambda-1}(1) = 0$, the argument is similar. This completes the proof. □

According to Lemma 3, we can use Lemmas 1 and 2 to compute the divisions in step 12. The number of divisions required in step 12 is recorded as L_i, which ranges from 1 to $\lambda - 1$ for $i = 0, 1, \ldots, \lambda - 1$. So, we can obtain $c_{e_i}(x)$ in step 12 by recursively computing the division L_i times, while the number of nonzero terms of the polynomial resulting from the first $L_i - 1$ divisions is even. Therefore, we can execute these divisions by Lemma 2 and execute the last division by Lemma 1. The computational complexity T_D in steps 11–12 of Algorithm 2 is

$$T_D = \sum_{i=0}^{\lambda-1} \left((L_i - 1)\frac{3p-5}{2} + p - 3\right), \tag{12}$$

where $\lambda(p-3) \leq T_D \leq \lambda(\lambda-2)\frac{3p-5}{2} + \lambda(p-3)$.

In steps 11-12 of Algorithm 2, we take the $\lambda(\lambda-1)$ division without Algorithm 1, in which λ divisions are executed by Lemma 1 and $\lambda(\lambda-2)$ divisions are executed by Lemma 2; however, the number of the divisions can be reduced with Algorithm 1. In Table 1, we show the average number of divisions in steps 11–12 of Algorithm 2 executed by Lemma 1 and Lemma 2 with Algorithm 1 for $(p, n) \in \{(5,5), (7,7)\}$.

Table 1. The average number of XORs involved in steps 11–12 of Algorithm 2.

p, n	λ	**Without Algorithm 1**			**Apply Algorithm 1**			**Improvement(%)**
		Lemma 2	**Lemma 1**	**XORs**	**Lemma 2**	**Lemma 1**	**XORs**	
(5,5)	2	0	2	4	0	2	4	0%
	3	3	3	21	2	3	16	23.81%
	4	8	4	48	0	4	8	83.33%
(7,7)	2	0	2	8	0	2	8	0%
	3	3	3	36	2.4	3	31.2	13.33%
	4	8	4	80	4.4	4	51.2	36%
	5	15	5	140	1	5	28	80%
	6	24	6	216	6	6	72	66.67%

We specify the computational complexity of Algorithm 2 as follows:

- Steps 1–2 take $\lambda(\delta-1)p = \lambda(n-\lambda-1)p$ XORs.

- Steps 3–6 take $\lambda(\lambda-1)p$ XORs.
- Steps 7–10 take $\lambda(\lambda-1)p$ XORs.
- Steps 11–12 take T_D XORs by Equation (12).

Then, the computational complexity $T_{Alg\ 2}$ of Algorithm 2 is

$$T_{Alg\ 2} = \lambda(n+\lambda-3)p + T_D, \tag{13}$$

where

$$p\lambda^2+\big((n-2)p-3\big)\lambda \le T_{Alg\ 2} \le \frac{5(p-1)}{2}\lambda^2+\big((n-5)p+2\big)\lambda.$$

Recall that the computational complexity of the decoding method in [17] is

$$\frac{7p-4}{2}\lambda^2 - \frac{7p-2}{2}\lambda + r(n-1)p.$$

which is strictly larger than $T_{Alg\ 2}$.

Table 2 evaluates the computational complexity of the decoding method in [17] and Algorithm 2 for some parameters. The results in Table 2 demonstrate that Algorithm 2 has much lower decoding complexity, compared with the original decoding method in [17]. For example, Algorithm 2 has 40.60% less decoding complexity than the decoding method in [17] when $(p,n,r)=(7,7,4), \lambda=3$.

Table 2. Decoding complexity of method in [17] and Algorithm 2.

p, n, r	λ	XORs in [17]	XORs of $T_{Alg\ 2}$	Improvement(%)
(5,5,3)	2	89	44	50.56%
	3	150	91	39.33%
(7,7,4)	2	211	92	56.40%
	3	300	178.2	40.60%
	4	434	275.2	36.59%

The reason why Algorithm 2 has lower decoding complexity than the decoding method in [17] can be summarized as the following three points.

Firstly, we only consider the first λ terms (the λ coefficients of degrees less than λ) for both $S(z)$ and $Q(z)$ in computing the coefficients of $z^{\lambda-1}$, while all r terms of $S(z)$ and all $r+\lambda$ terms of $Q(z)$ are calculated in the decoding method in [17], where $r \ge \lambda$.

Secondly, all the divisions in Algorithm 2 are executed over the ring $\mathbb{F}_2[x]/(1+x^p)$ by Lemmas 1 and 2, which takes $p-3$ XORs and $\frac{3p-5}{2}$ XORs for each division, respectively. In addition, the division in [17] is executed over the ring $\mathcal{R}_p$, which takes $2(p-1)$ XORs [17] (Corollary 2).

Thirdly, we apply Algorithm 1 to steps 11–12 of Algorithm 2, which can significantly reduce the number of divisions, thus reducing the number of XORs required.

3.2. Modified Interpolation-Based Decoding Method

According to the decoding method in [18], we can recover the erased column $c_{e_i}(x)$ with $0 \le i \le \lambda-1$ by

$$c_{e_i}(x) = \sum_{j=0}^{\delta-1} c_{h_j}(x)\frac{f_i(x^{h_j})}{f_i(x^{e_i})} \pmod{M_p(x)}, \tag{14}$$

where $f_i(y) = \prod_{s=0,\neq i}^{\lambda-1}(y-x^{e_s})$ and $f(y) = \prod_{s=0}^{\lambda-1}(y-x^{e_s})$. Let

$$a_j(x) = c_{h_j}(x)\cdot f(x^{h_j}) = \prod_{s=0}^{\lambda-1}(x^{h_j}-x^{e_s})\cdot c_{h_j}(x) \pmod{M_p(x)}, \tag{15}$$

where $0 \leq j \leq \delta - 1$. Then, $a_j(x)$ has an even number of nonzero terms, and we only need to compute once for $a_j(x)$ in the decoding procedure, since $a_j(x)$ is independent of the erasure index i. Let

$$b_i(x) = \sum_{j=0}^{\delta-1} \frac{a_j(x)}{x^{h_j} - x^{e_i}} \pmod{M_p(x)}, \tag{16}$$

$$c_{e_i}(x) = \frac{b_i(x)}{f_i(x^{e_i})} = \frac{b_i(x)}{\prod_{s=0,\neq i}^{\lambda-1}(x^{e_i} - x^{e_s})} \pmod{M_p(x)}, \tag{17}$$

where $0 \leq i \leq \lambda - 1$, and $M_p(x) = 1 + x + \cdots + x^{p-1}$. Algorithm 3 shows our modified interpolation-based method over the ring $\mathbb{F}_2[x]/(1 + x^p)$.

After using Algorithm 1, the number of polynomial multiplications in step 2 ranges from 1 to λ. Thus, the computational complexity T_M in steps 1–2 of Algorithm 3 is

$$(n - \lambda)p \leq T_M \leq (n - \lambda)\lambda p. \tag{18}$$

Algorithm 3: Modified interpolation-based method.

Input: The λ erased columns $\{e_i\}_{i=0}^{\lambda-1}$ and the $\delta = n - \lambda$ surviving columns $\{h_j\}_{j=0}^{\delta-1}$.

1 **for** $j \leftarrow 0$ **to** $\delta - 1$ **do** // Use Equation (15) and apply Algorithm 1

2 $\quad a_j(x) \leftarrow f(x^{h_j}) \cdot c_{h_j}(x) = \prod_{s=0}^{\lambda-1}(x^{h_j} - x^{e_s}) \cdot c_{h_j}(x)$;

3 **for** $i \leftarrow 0$ **to** $\lambda - 1$ **do** // Use Equation (16)

4 $\quad b_i(x) \leftarrow \sum_{j=0}^{\delta-1} \frac{a_j(x)}{x^{h_j} - x^{e_i}}$;

5 **for** $i \leftarrow 0$ **to** $\lambda - 1$ **do** // Use Equation (17) and apply Algorithm 1

6 $\quad c_{e_i}(x) \leftarrow \frac{b_i(x)}{f_i(x^{e_i})} = \frac{b_i(x)}{\prod_{s=0,\neq i}^{\lambda-1}(x^{e_i} - x^{e_s})}$;

Output: The erased columns $\{c_{e_i}(x)\}_{i=0}^{\lambda-1}$.

In steps 1–2, we need to take λ multiplications without Algorithm 1, which takes $(n - \lambda)\lambda p$ XORs; however, with Algorithm 1, the number of multiplications involved in steps 1–2 can be reduced. In Table 3, we show the average number of XORs involved in steps 1–2 of Algorithm 3 with Algorithm 1 for $(p, n) \in \{(5,5), (7,7)\}$. The results in Table 3 show that we can reduce the number of XORs with Algorithm 1, especially for a large value of λ.

Table 3. The average number of XORs involved in steps 1–2 of Algorithm 3.

p, n	λ	Without Algorithm 1		Apply Algorithm 1		Improvement(%)
		Multiplication	XORs	Multiplication	XORs	
(5,5)	2	6	30	5	25	16.67%
	3	6	30	2	10	66.67%
	4	4	20	2	10	50%
(7,7)	2	10	70	9	63	10%
	3	12	84	8.4	58.8	30%
	4	12	84	3.6	25.2	70%
	5	10	70	4	28	60%
	6	6	42	3	21	50%

Only steps 4 and 6 of Algorithm 3 are needed to compute the division. We should employ Lemma 2 to execute the divisions in steps 3–4 in Algorithm 3, since $b_i(x)$ in step 6 of Algorithm 3 should have an even number of nonzero terms. Notice that steps 5–6 of Algorithm 3 are exactly the same as steps 11–12 of Algorithm 2.

We specify the computational complexity of Algorithm 3 as follows:

- Steps 1–2 require T_M XORs by Equation (18).
- Steps 3–4 need $\lambda(\delta-1)$ additions and $\lambda\delta$ divisions by Lemma 2, which require $\lambda(n-\lambda-1)p+\lambda(n-\lambda)\frac{3p-5}{2}$ XORs in total.
- Steps 5–6 require T_D XORs by Equation (12).

Then, the computational complexity of Algorithm 3 is

$$T_{Alg\ 3} = T_M + \lambda(n-\lambda-1)p + \lambda(n-\lambda)\frac{3p-5}{2} + T_D, \tag{19}$$

where

$$-\frac{5(p-1)}{2}\lambda^2 + (\frac{5n-2}{2}p - \frac{5}{2}n - 3)\lambda + np \le T_{Alg\ 3} \le -2p\lambda^2 + (\frac{7n-6}{2}p - \frac{5}{2}n + 2)\lambda.$$

Recall that the computational complexity of the decoding method in [18] is

$$(-2p+1)\lambda^2 + \big(4(n-1)p - 3n + 4\big)\lambda + n(p-1),$$

which is larger than that of our Algorithm 3.

Table 4 evaluates the computational complexity of the decoding method in [18] and Algorithm 3 for some parameters. The results in Table 4 demonstrate that our Algorithm 3 had much lower decoding complexity, compared with the original decoding method in [18]. For example, Algorithm 3 had a 34.13% lower decoding complexity than the decoding method in [18], when $(p,n,r) = (7,7,4), \lambda = 3$.

Table 4. Decoding complexities of the decoding method in [18] and our Algorithm 3.

p, n, r	λ	**XORs in [18]**	**XORs of $T_{Alg\ 3}$**	**Improvement(%)**
$(5,5,3)$	2	122	79	35.25%
	3	146	71	51.37%
$(7,7,4)$	2	292	207	29.11%
	3	378	249	34.13%
	4	438	228.4	47.85%

The reason why Algorithm 3 has a lower decoding complexity than that of the decoding method in [18] is summarized as follows.

Firstly, all the divisions in Algorithm 3 were executed over the ring $\mathbb{F}_2[x]/(1+x^p)$ by Lemmas 1 and 2, which used $p-3$ XORs and $(3p-5)/2$ XORs for each division, respectively. The division in the decoding method in [18] was executed over the ring $\mathcal{R}_p$, which used $2(p-1)$ XORs.

Secondly, we applied our Algorithm 1 to steps 1–2 and steps 5–6, which significantly reduced the number of multiplications, thus reducing the number of XORs required.

4. LU Decomposition-Based Method

The LU factorization of a matrix [22] is to express the matrix as a product of a lower triangular matrix **L** and an upper triangular matrix **U**. According to the LU factorization of the Vandermonde matrix [23], we can express a Vandermonde matrix as a product of several lower triangular matrices and several upper triangular matrices. Therefore, we can solve the Vandermonde linear equations by first solving the linear equations with the encoding matrices that are the upper triangular matrices and then solving the linear equations with the encoding matrices that are the lower triangular matrices.

Suppose that the λ erased columns are λ columns $\{e_i\}_{i=0}^{\lambda-1}$ and the $\delta = n-\lambda$ surviving columns are $\{h_j\}_{j=0}^{\delta-1}$. Algorithm 4 shows our LU decomposition-based method over the ring $\mathbb{F}_2[x]/(1+x^p)$.

According to [Theorem 8] in [19], Equation (1) can be factorized into

$$\begin{pmatrix} c_{e_0}(x) & c_{e_1}(x) & \cdots & c_{e_{\lambda-1}}(x) \end{pmatrix} \cdot (\mathbf{L}_\lambda^{(1)}\mathbf{L}_\lambda^{(2)}\cdots\mathbf{L}_\lambda^{(\lambda-1)}) \cdot (\mathbf{U}_\lambda^{(\lambda-1)}\mathbf{U}_\lambda^{(\lambda-2)}\cdots\mathbf{U}_\lambda^{(1)}) = \mathbf{S}, \quad (20)$$

over the ring $\mathcal{R}_p$, where $\mathbf{U}_\lambda^{(\theta)}$ is the upper triangle matrix

$$\mathbf{U}_\lambda^{(\theta)} = \left[\begin{array}{c|cccccc} \mathbf{I}_{\lambda-\theta-1} & & & \mathbf{0} & & & \\ & 1 & x^{e_0} & 0 & \cdots & 0 & 0 \\ & 0 & 1 & x^{e_1} & \cdots & 0 & 0 \\ ine\mathbf{0} & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ & 0 & 0 & 0 & \cdots & 1 & x^{e_{\theta-1}} \\ & 0 & 0 & 0 & \cdots & 0 & 1 \end{array}\right], \quad (21)$$

and $\mathbf{L}_\lambda^{(\theta)}$ is the lower triangle matrix

$$\mathbf{L}_\lambda^{(\theta)} = \left[\begin{array}{c|cccc} \mathbf{I}_{\lambda-\theta-1} & & \mathbf{0} & & \\ & 1 & 0 & \cdots & 0 & 0 \\ & 1 & x^{e_{\lambda-\theta}} + x^{e_{\lambda-\theta-1}} & \cdots & 0 & 0 \\ ine\mathbf{0} & \vdots & \vdots & \ddots & \vdots & \vdots \\ & 0 & 0 & \cdots & x^{e_{\lambda-2}} + x^{e_{\lambda-\theta-1}} & 0 \\ & 0 & 0 & \cdots & 1 & x^{e_{\lambda-1}} + x^{e_{\lambda-\theta-1}} \end{array}\right], \quad (22)$$

for $\theta = 1, 2, \ldots, \lambda - 1$.

Algorithm 4: LU decomposition-based method.

Input: The λ erased columns $\{e_i\}_{i=0}^{\lambda-1}$ and the $\delta = n - \lambda$ surviving columns $\{h_j\}_{j=0}^{\delta-1}$.

1 **for** $\ell \leftarrow 0$ **to** $\lambda - 1$ **do** `// use Equation (2) and the subscript` ℓ `means slope.`

2 $\quad S_\ell(x) \leftarrow \sum_{j=0}^{\delta-1} x^{\ell \cdot h_j} c_{h_j}(x)$;

3 $\begin{pmatrix} c_{e_0}(x) & c_{e_1}(x) & \cdots & c_{e_{\lambda-1}}(x) \end{pmatrix} \leftarrow \begin{pmatrix} S_0(x) & S_1(x) & \cdots & S_{\lambda-1}(x) \end{pmatrix}$;

`// Eliminate` $\lambda - 1$ `upper triangular matrices` $\mathbf{U}_\lambda^{(1)}, \mathbf{U}_\lambda^{(2)}, \ldots, \mathbf{U}_\lambda^{(\lambda-1)}$.

4 **for** $\theta \leftarrow 1$ **to** $\lambda - 1$ **do**

$\quad$ `// Eliminate upper triangular matrix` $\mathbf{U}_\lambda^{(\theta)}$ `by forward additions.`

5 $\quad$ **for** $i \leftarrow \lambda - \theta$ **to** $\lambda - 1$ **do**

6 $\quad\quad c_{e_i}(x) \leftarrow c_{e_i}(x) + x^{e_{\theta+i-\lambda}} \cdot c_{e_{i-1}}(x)$;

`// Eliminate` $\lambda - 1$ `lower triangular matrix` $\mathbf{L}_\lambda^{(\lambda-1)}, \mathbf{L}_\lambda^{(\lambda-2)}, \ldots, \mathbf{L}_\lambda^{(1)}$.

7 **for** $\theta \leftarrow \lambda - 1$ **down to** 1 **do**

$\quad$ `// Eliminate lower triangular matrix` $\mathbf{L}_\lambda^{(\theta)}$ `by backward additions.`

8 $\quad$ Solve $c_{e_{\lambda-1}}(x)$ from $c_{e_{\lambda-1}}(x) = \frac{c_{e_{\lambda-1}}(x)}{x^{e_{\lambda-1}} + x^{e_{\lambda-\theta-1}}}$ by Lemma 2 and Lemma 1(only when $\theta = 1$);

9 $\quad$ **for** $i \leftarrow \lambda - 2$ **down to** $\lambda - \theta$ **do**

10 $\quad\quad$ Solve $c_{e_i}(x)$ from $c_{e_i}(x) = \frac{c_{e_i}(x) - c_{e_{i+1}}(x)}{x^{e_i} + x^{e_{\lambda-\theta-1}}}$ by Lemma 2 and Lemma 1(only when $i = \lambda - \theta$);

11 $\quad c_{e_{\lambda-\theta-1}}(x) \leftarrow c_{e_{\lambda-\theta-1}}(x) - c_{e_{\lambda-\theta}}(x)$;

Output: The erased columns $\{c_{e_i}(x)\}_{i=0}^{\lambda-1}$.

We specify the computational complexity of Algorithm 4 as follows:

- Steps 1–2 require $\lambda(\delta - 1)p = \lambda(n - \lambda - 1)p$ XORs.

- Steps 3–11 require $\lambda(\lambda-1)p+(\lambda-1)(p-3)+(\lambda-1)(\lambda-2)(3p-5)/4$ XORs at most, according to [Theorem 10] in [19].

  Then, the computational complexity of Algorithm 4 is

$$T_{Alg\ 4}=\frac{3p-5}{4}\lambda^2+\frac{(4n-13)p+3}{4}\lambda+\frac{p+1}{2}. \tag{23}$$

5. Comparison and Conclusions

Table 5 evaluates the decoding complexity of Algorithm 2–4 for some parameters. The results of Table 5 demonstrate that Algorithm 2 performs better than Algorithm 3 if $\lambda \leq \frac{n}{2}$; otherwise, if $\lambda > \frac{n}{2}$, then Algorithm 3 has less decoding complexity. Algorithm 4 has less decoding complexity than both Algorithms 2 and 3, when λ is small. However, when λ is large, Algorithm 3 is more efficient than Algorithm 4. For example, compared with Algorithm 2–4 have 21.98% and 40.66% less decoding complexity, respectively, when $(p,n,r)=(5,5,4), \lambda=3$.

Table 5. Decoding complexities of the proposed three decoding methods.

p, n, r	λ	total XORs			$\frac{T_{Alg\ 2}-T_{Alg\ 3}}{T_{Alg\ 2}}$	$\frac{T_{Alg\ 2}-T_{Alg\ 4}}{T_{Alg\ 2}}$
		$T_{Alg\ 2}$	$T_{Alg\ 3}$	$T_{Alg\ 4}$		
$(5,5,4)$	2	44	79	32	−79.55%	27.27%
	3	91	71	54	21.98%	40.66%
	4	128	38	81	70.31%	36.72%
$(7,7,6)$	2	92	207	74	−125%	19.57%
	3	178.2	249	121	−39.73%	32.10%
	4	275.2	228.4	176	17.01%	36.05%
	5	343	171	239	50.15%	30.32%
	6	492	141	310	71.34%	36.99%

In this paper, we presented three efficient decoding methods for the erasures of Blaum–Roth codes that all have lower decoding complexity than the existing decoding methods. The efficient implementation of the proposed decoding methods in practical storage systems is one of our future works.

Author Contributions: Funding acquisition, H.H. methodology, H.H.; writing—original draft preparation, W.Z.; writing—review and editing, H.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China grant number 62071121 and National Key R&D Program of China grant number 2020YFA0712300.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: All data generated or analysed during this study are included in this published article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Peng, P. Redundancy Allocation in Distributed Systems. Ph.D. Thesis, Rutgers The State University of New Jersey, School of Graduate Studies, New Brunswick, NJ, USA, 2022.
2. MacWilliams, F.J.; Sloane, N.J.A. *The Theory of Error Correcting Codes*; Elsevier: Amsterdam, The Netherlands, 1977; Volume 16.
3. Blaum, M.; Brady, J.; Bruck, J.; Menon, J. EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures. *IEEE Trans. Comput.* **1995**, *44*, 192–202. [CrossRef]
4. Hou, H.; Lee, P.P.C. A New Construction of EVENODD Codes With Lower Computational Complexity. *IEEE Commun. Lett.* **2018**, *22*, 1120–1123. [CrossRef]

5. Corbett, P.; English, B.; Goel, A.; Grcanac, T.; Kleiman, S.; Leong, J.; Sankar, S. Row-diagonal Parity for Double Disk Failure Correction. In Proceedings of the 3rd USENIX Conference on File and Storage Technologies, San Francisco, CA, USA, 31 March–4 April 2004; pp. 1–14.
6. Xu, L.; Bruck, J. X-code: MDS Array Codes with Optimal Encoding. *IEEE Trans. Inf. Theory* **1999**, *45*, 272–276.
7. Tsunoda, Y.; Fujiwara, Y.; Ando, H.; Vandendriessche, P. Bounds on separating redundancy of linear codes and rates of X-codes. *IEEE Trans. Inf. Theory* **2018**, *64*, 7577–7593. [CrossRef]
8. Goel, A.; Corbett, P. RAID Triple Parity. *ACM SIGOPS Oper. Syst. Rev.* **2012**, *46*, 41–49. [CrossRef]
9. Huang, C.; Xu, L. STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures. *IEEE Trans. Comput.* **2008**, *57*, 889–901. [CrossRef]
10. Hou, H.; Lee, P.P.C. STAR+ Codes: Triple-Fault-Tolerant Codes with Asymptotically Optimal Updates and Efficient Encoding/Decoding. In Proceedings of the 2021 IEEE Information Theory Workshop (ITW 2021), Kanazawa, Japan, 17–21 October 2021.
11. Blaum, M.; Brady, J.; Bruck, J.; Jai Menon, J.; Vardy, A. The EVENODD Code and its Generalization: An Effcient Scheme for Tolerating Multiple Disk Failures in RAID Architectures. In *High Performance Mass Storage and Parallel I/O*; Wiley-IEEE Press: Hoboken, NJ, USA, 2002; Chapter 8, pp. 187–208.
12. Blaum, M.; Bruck, J.; Vardy, A. MDS Array Codes With Independent Parity Symbols. *IEEE Trans. Inf. Theory* **1996**, *42*, 529–542. [CrossRef]
13. Hou, H.; Shum, K.W.; Chen, M.; Li, H. New MDS Array Code Correcting Multiple Disk Failures. In Proceedings of the 2014 IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 2369–2374.
14. Fu, H.; Hou, H.; Zhang, L. Extended EVENODD+ Codes with Asymptotically Optimal Updates and Efficient Encoding/Decoding. In Proceedings of the 2021 XVII International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY), Moscow, Russia, 25–29 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
15. Chiniah, A.; Mungur, A. On the Adoption of Erasure Code for Cloud Storage by Major Distributed Storage Systems. *EAI Endorsed Trans. Cloud Syst.* **2022**, *7*, e1. [CrossRef]
16. Rui, J.; Huang, Q.; Wang, Z. Graftage Coding for Distributed Storage Systems. *IEEE Trans. Inf. Theory* **2021**, *67*, 2192–2205. [CrossRef]
17. Blaum, M.; Roth, R.M. New Array Codes for Multiple Phased Burst Correction. *IEEE Trans. Inf. Theory* **1993**, *39*, 66–77. [CrossRef]
18. Guo, Q.; Kan, H. On Systematic Encoding for Blaum-Roth Codes. In Proceedings of the 2011 IEEE International Symposium on Information Theory Proceedings, St. Petersburg, Russia, 31 July–5 August 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 2353–2357.
19. Hou, H.; Han, Y.S.; Shum, K.W.; Li, H. A Unified Form of EVENODD and RDP Codes and Their Efficient Decoding. *IEEE Trans. Commun.* **2018**, *66*, 5053–5066. [CrossRef]
20. Hou, H.; Shum, K.W.; Chen, M.; Li, H. BASIC Codes: Low-complexity Regenerating Codes for Distributed Storage Systems. *IEEE Trans. Inf. Theory* **2016**, *62*, 3053–3069. [CrossRef]
21. Hou, H.; Han, Y.S. A New Construction and An Efficient Decoding Method for Rabin-like Codes. *IEEE Trans. Commun.* **2017**, *66*, 521–533. [CrossRef]
22. Strang, G.; Strang, G.; Strang, G.; Strang, G. *Introduction to Linear Algebra*; Wellesley-Cambridge Press: Wellesley, MA, USA, 1993; Volume 3.
23. Yang, S.l. On the LU factorization of the Vandermonde matrix. *Discret. Appl. Math.* **2005**, *146*, 102–105. [CrossRef]