AIMS Mathematics, 11(8): 26337–26358.
DOI: 10.3934/math.20261057
Received: 09 April 2026
Revised: 11 July 2026
Accepted: 05 August 2026
Published: 24 August 2026

https://www.aimspress.com/journal/Math

Research article

Array codes with local properties: Explicit constructions, erasure decoding, and MSSS application

Xian Lian[1], **Jingjie Lv**[1,*], **Hongwei Zhu**[2], **Shu-Tao Xia**[3] **and Hanxu Hou**[1,4,5]

[1] School of Telecommunications Engineering & Intelligentization, Dongguan University of Technology, Dongguan 523808, China

[2] School of Mathematics and Information Science, Guangzhou University, Guangzhou, Guangdong 510006, China

[3] Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China

[4] Institute of Network Coding, The Chinese University of Hong Kong, Hong Kong 999077, China

[5] Shenzhen University of Advanced Technology, Shenzhen 518107, China

* **Correspondence:** Email: juxianljj@163.com.

Abstract: This paper presents two constructions of q-ary array codes that possess column local properties; that is, any erasure can be recovered using only the remaining symbols within a single column of a codeword. The first construction generalizes the extended Blaum-Roth codes as a special case. An explicit formulation of Construction I is further developed, which enables the code length to grow exponentially as 2^{κ}, where κ denotes the minimum degree among the irreducible factors of $m(x)$. Leveraging the recursive structure of the associated polynomials and the Reed-Muller transform, we devise two fast erasure decoding algorithms based on recursive branching: one tailored for redundancy $r \leq 3$ and the other applicable to arbitrary r. The second family of array codes is applied to the modified Shamir secret sharing (MSSS) scheme. Compared with the existing BR/GRDP-based MSSS scheme, the proposed approach offers the following advantages: (i) support for a larger number of participants, (ii) self-repairability of every share, and (iii) lower encoding complexity.

Keywords: array codes; column local properties; erasure decoding; Shamir secret sharing
Mathematics Subject Classification: 11T71, 94B05

1. Introduction

Modern storage systems, ranging from cloud data centers to edge computing clusters, demand ever-increasing reliability, scalability, and I/O efficiency. Array codes, which organize data into

two-dimensional arrays and rely predominantly on exclusive or (XOR) operations, have become a cornerstone of redundant storage architectures such as redundant arrays of inexpensive disks (RAID) [1]. Their row-column parity structure naturally supports parallel processing and low-complexity encoding/decoding, making them highly attractive for large-scale storage systems [2].

Over the past decades, a rich family of array codes has been developed. Maximum distance separable (MDS) array codes achieve the Singleton bound, providing optimal redundancy for a given failure tolerance. For double-column failures, EVENODD [3] and row-diagonal parity (RDP) [4] codes are well-known efficient solutions. For triple-column failures, Reed-Solomon-like codes [5] and STAR codes [6] have been proposed. To tolerate more than three column failures, generalized RDP (GRDP) codes [7], independent parity (IP) codes [8], Blaum-Roth (BR) codes [9], and more recent constructions [10–12] have been introduced. However, when the redundancy $r > 3$, GRDP and IP codes are not always MDS.

A critical limitation of the aforementioned early array codes is that their code length is bounded by the sub-packetization parameter m. For instance, EVENODD and RDP codes can typically achieve a code length of at most $m + r$. To break this barrier, recent constructions have achieved significantly longer code lengths. Lv et al. [10] proposed a new family of q-ary MDS array codes whose length scales exponentially in m. Subsequently, Fang et al. [11] applied similar constructions to locally repairable array codes while maintaining sparse parity-check matrices (PCMs). More recently, Zhai et al. [12] constructed another family of MDS array codes that achieve a code length that is a multiple of m. Moreover, the latter three families [10–12] support a much wider range of parameters than classical constructions: They no longer require m to be prime, can realize MDS properties under more flexible configurations, and maintain sparse parity structures for efficient encoding and repair. These advances make long length, high-rate MDS array codes practically deployable in large-scale storage systems.

In addition to MDS array codes, another important direction is array codes with local repair capabilities. A related but distinct concept is (ℓ, k)-recoverable: any k out of ℓ columns can reconstruct the whole codeword. Although MDS codes satisfy this, the converse need not hold because the parity-check submatrix may be singular [7]. To support local repairs, Blaum et al. [13] introduced array codes with column local properties over polynomial rings, where each column forms a cyclic code and can be repaired independently. Hou et al. [14] generalized this framework and proved that such codes are (ℓ, k)-recoverable. However, when sub-packetization is fixed to m, the maximum code length is limited to m. Thus, we are eager to design long (ℓ, k)-recoverable array codes with column local properties.

Beyond this line of work, other locally repairable array codes (LRACs) have also been proposed [11, 15], which reduce repair bandwidth and I/O by recovering a failed node from a small group of surviving nodes. This is crucial for cloud storage, edge computing, and low-latency systems. It is important to distinguish them from array codes with local properties: The latter confine repair to a single column (each column is a cyclic code), incurring minimal cross-column communication; LRACs allow repair groups to span multiple columns, offering flexibility at the cost of accessing several columns. Because this paper focuses on array codes with local properties, we do not further explore LRACs, as they represent a different design philosophy.

In parallel with the construction of array codes, efficient decoding algorithms are crucial for practical storage sharing systems. Decoding array codes typically involves syndrome computation and solving linear equations derived from the parity-check matrix. Under high redundancy (high-rate), syndrome computation dominates the encoding/decoding computational overhead. Efficient decoding

is essential for array codes in storage and secret sharing. Decoding typically involves syndrome computation and solving parity-check equations. Extensive research has focused on efficient division over rings for MDS array codes, such as extended BR (EBR) codes [13], IP codes [8], and generalized EBR codes [14]. Using recursive branching approaches to compute syndromes [15–17] can reduce the asymptotic complexity of encoding and decoding to $\lfloor \log_2 r \rfloor + 1$ XORs per data bit. In practice, multiplications over $\mathbb{F}_q^m$ can be implemented with cyclic shifts and XORs, whereas multiplications over the ring $\mathbb{F}_q[x]/\langle x^m + 1\rangle$ require only cyclic shifts. Thus, array codes naturally achieve lower encoding/decoding complexity than Reed-Solomon (RS) codes. In [10], one proposed a fast syndrome algorithm for MDS array codes with redundancy $r \leq 3$ that approaches the asymptotic lower bound of 2 XORs per data bit. In this paper, we extend that recursive branching fast syndrome computation to array codes with local properties for arbitrary redundancy r.

Beyond storage systems, array codes have found promising applications in secret sharing. A perfect $(k, n-1)$-threshold secret sharing (TSS) scheme allows a secret to be reconstructed from $n-1$ shares, with any k shares being sufficient and any $k-1$ shares revealing no information about the secret [18]. Traditional TSS schemes, such as the Shamir secret sharing (SSS) scheme based on RS codes [19], rely on finite field arithmetic and face significant scalability and efficiency challenges when the number of participants becomes large (e.g., in blockchain sharding or distributed ledgers). To mitigate these issues, several works have explored replacing RS codes with XOR-based array codes in secret sharing [20–22]. Nevertheless, these alternatives still suffer from three major limitations. The first is limited scalability. Most schemes based on array codes [20] limit code lengths due to polynomial degree constraints or limited field sizes, hindering applications that require thousands of participants (such as blockchain sharding). The second is global repair overhead. Traditional erasing correction requires access to k participants for reconstruction [21], resulting in high latency in geographically distributed systems. The third is coding bottleneck. RS codes and even XOR-based array codes [22] usually rely on the dense PCMs, which limit the computational efficiency of long codes. To address existing limitations, this paper proposes a MSSS scheme based on a new q-ary array codes with local properties.

The rest of the paper is organized as follows. Section 2 reviews preliminaries on array codes, circulant matrices, and the MSSS scheme. Section 3 presents the generic construction of array codes with local properties (Construction I) and its explicit form. Section 4 develops an efficient erasure decoding algorithm for the proposed array codes, including syndrome calculation based on recursive branching approaches and the solution of Vandermonde linear equations. Section 5 extends the construction to an MSSS scheme (Construction II), provides encoding examples, and compares with other MSSS schemes. Finally, Section 6 concludes the paper.

2. Preliminaries

2.1. Array codes

Let $\mathbb{F}_q$ denote the finite field with q elements, where q is a power of a prime. An array code $\mathbf{C}$ of shape $m \times \ell$ is a K-dimensional linear subspace of the space of $m \times \ell$ matrices over $\mathbb{F}_q$. The parameters m, ℓ, and K are called the sub-packetization, the *code length*, and the dimension of $\mathbf{C}$, respectively. When $m \mid K$, we define the normalized dimension $k = K/m$ and the redundancy $r = n - k$. Each

codeword is written as

$$\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{\ell-1}),$$

where each $\mathbf{c}_i \in \mathbb{F}_q^m$ is a column vector of length m. The column weight of $\mathbf{c}$ is the number of nonzero columns, denoted by $\mathrm{wt}_A(\mathbf{c})$, and the minimum column distance $d_A(\mathbf{C})$ is the minimum column weight over all nonzero codewords. Such a code is denoted by the compact notation $(\ell, k, d_A; m)_q$.

When the entries of $\mathbf{c}$ are read column by column (and within each column from top to bottom), the array can be flattened into a row vector of length $m\ell$, thereby relating array codes to ordinary linear codes over $\mathbb{F}_q$.

Example 2.1. Consider a 4-ary array code $\mathbf{C}$ with shape 4×6. The following array is a codeword:

1	0	ω	ω^2	1	0
0	1	ω^2	ω	0	1
ω	ω^2	1	0	ω	ω^2
ω^2	ω	0	1	ω^2	ω

where $\omega \in \mathbb{F}_4$ satisfies $\omega^2 = \omega + 1$ (assuming a suitable representation). Flattening column by column yields the row vector

$$(1, 0, \omega, \omega^2,\ 0, 1, \omega^2, \omega,\ \omega, \omega^2, 1, 0,\ \omega^2, \omega, 0, 1,\ 1, 0, \omega, \omega^2,\ 0, 1, \omega^2, \omega).$$

This representation illustrates that an array code can be viewed as a linear code of length $m\ell$, though the two codes generally have different distance metrics.

2.2. Cyclic codes and circulant matrices

A cyclic code C of length m over $\mathbb{F}_q$ is a linear subspace of $\mathbb{F}_q^m$ that is closed under the cyclic shift: If $(c_0, c_1, \ldots, c_{m-1}) \in C$, then $(c_{m-1}, c_0, \ldots, c_{m-2}) \in C$. It is well-known that cyclic codes correspond to ideals in the residue ring $\mathbb{R}_m = \mathbb{F}_q[x]/\langle x^m + 1\rangle$. Specifically, there exists an $\mathbb{F}_q$-linear isomorphism between $\mathbb{F}_q^m$ and $\mathbb{R}_m$. Consequently, every cyclic code C is generated by a monic divisor $p(x)$ of $x^m + 1$, that is, $C = \langle p(x)\rangle$. The minimum Hamming distance $d_H(C)$ determines the error/erasure correction capability: C can correct up to $d_H - 1$ erasures or a burst of $\deg(p(x))$ erasures [23].

A closely related structure is that of circulant matrices. For a polynomial $a(x) = a_0 + a_1 x + \cdots + a_{m-1}x^{m-1} \in \mathbb{R}_m$, define A^t as the $m \times m$ circulant matrix whose first row is the coefficient vector of $a(x)^t \bmod (x^m + 1)$ (with $a(x)^0 = 1$),

$$A^t = \begin{pmatrix} u_0 & u_1 & \cdots & u_{m-1} \\ u_{m-1} & u_0 & \cdots & u_{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ u_1 & u_2 & \cdots & u_0 \end{pmatrix}_{m\times m}, \tag{2.1}$$

where $a(x)^t \equiv u_0 + u_1 x + \cdots + u_{m-1}x^{m-1} \bmod (x^m + 1)$. Let $\{A\}_m$ be the set of such matrices.

Throughout this paper, we use the term associated to describe pairwise correlations between vectors, polynomials, and circulant matrices.

Lemma 2.2. *There exists an $\mathbb{F}_q[x]$-module isomorphism between $\{A\}_m$ and $\mathbb{R}_m$. Hence, a vector $\mathbf{a}$, its polynomial $a(x)$, and its circulant matrix A are equivalent.*

Proof. Clearly, the mapping $\Phi : \mathbb{F}_q^m \to \mathbb{R}_m$ is a natural bijection $(a_0, \ldots, a_{m-1}) \mapsto \sum_{i=0}^{m-1} a_i x^i$. Define $\Psi : \{A\}_m \to \mathbb{R}_m$ by $\Psi(A) = \Phi(\mathbf{a})$, where $\mathbf{a}$ is the first row of A. Then, Ψ is bijective and $\mathbb{F}_q$-linear, and $\Psi(AB) = a(x)b(x) \bmod (x^m + 1) = \Psi(A)\Psi(B)$ (because circulant matrix multiplication corresponds to convolution with $x^m = -1$). Thus, Ψ is an $\mathbb{F}_q[x]$-module isomorphism. □

For an positive integer τ less than m, define the punctured circulant matrix $A^{[t,\tau]}$ as the $(m-\tau)$-order submatrix obtained by deleting the last τ rows and last τ columns of A^t:

$$A^{[t,\tau]} = \begin{pmatrix} u_0 & u_1 & \cdots & u_{m-\tau-1} \\ u_{m-1} & u_0 & \cdots & u_{m-\tau-2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{\tau+1} & u_{\tau+2} & \cdots & u_0 \end{pmatrix}_{(m-\tau)\times(m-\tau)}.$$

This puncturing operation reduces the sub-packetization level while preserving the algebraic structure, enabling MDS array codes with tunable sub-packetization [10, 24].

2.3. Modified Shamir secret sharing (MSSS) scheme

To reduce the encoding complexity of traditional Shamir secret sharing, Hineman et al. [20] proposed the modified Shamir secret sharing (MSSS) scheme. In MSSS, all parity symbols are made public, but only the $n-1$ data symbols (excluding the secret) are distributed to participants. Let C be an $[n, k]$ MDS code over $\mathbb{F}_q$ with parity-check matrix $\mathbf{H}_{(n-k)\times n}$. Define the systematic parity-check matrix

$$\mathbf{H}'_{(n-k)\times(2n-k)} = (\mathbf{H}_{(n-k)\times n} \mid \mathbf{I}_{n-k}), \tag{2.2}$$

where $\mathbf{I}_{n-k}$ is the identity matrix. For a secret $c_0 \in \mathbb{F}_q$ and $n-1$ random symbols $c_1, \ldots, c_{n-1}$, the public parity symbols $(c_n, \ldots, c_{2n-k-1})$ are computed as

$$(c_n, \ldots, c_{2n-k-1})^{\mathsf{T}} = \mathbf{H}_{(n-k)\times n} \cdot (c_0, c_1, \ldots, c_{n-1})^{\mathsf{T}}.$$

Any k participants can recover c_0 via erasure decoding, and fewer than k gain no information.

2.3.1. BR-based MSSS scheme (IP code)

Let p be a prime, $k < n \le p$. Over the ring $\mathbb{F}_q[x]/\langle M_p(x)\rangle$ with $M_p(x) = 1 + x + \cdots + x^{p-1}$, let α be a root of $M_p(x)$. Equation (2.2) directly gives the following BR-based MSSS scheme:

$$\mathbf{H}'_{BR} = (\mathbf{H}_{BR} \mid \mathbf{I}_{n-k}) = \left(\begin{array}{cccc|cccc} 1 & 1 & \cdots & 1 & 1 & 0 & \cdots & 0 \\ 1 & \alpha & \cdots & \alpha^{n-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k-1} & \cdots & \alpha^{(n-1)(n-k-1)} & 0 & 0 & \cdots & 1 \end{array}\right)_{(n-k)\times(2n-k)}.$$

The public parities are $(c_n, \ldots, c_{2n-k-1}) = \mathbf{H}_{BR} \cdot (c_0, \ldots, c_{n-1})^{\mathsf{T}}$.

2.3.2. GRDP-based MSSS scheme

We arrange the data as a $(p-1)\times(n-1)$ array, where the first column is the secret, and the next $n-2$ columns are random. Then, we encode it into a $(p-1)\times(2n-k-1)$ GRDP array as follows:

$$c_{i,n-1}=\sum_{j=0}^{n-2}c_{i,j}\ \text{(horizontal parity)},\quad c_{i,n-1+t}=\sum_{l=0}^{n-1}c_{i-tl,\,l}\ \text{(diagonal parity)}\quad \text{for}\ \begin{cases}0\le i\le n-2\\ 1\le t\le n-k-1,\end{cases}$$

where indices modulo p. Distribute the $n-1$ columns (secret, $p-2$ random, horizontal parity) to $p-1$ participants. Make the remaining $p-k-1$ diagonal parity columns public. The parity-check matrix of the corresponding systematic GRDP code is

$$\mathbf{H}'_{GRDP}=\begin{pmatrix}1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0\\ 1 & \alpha & \cdots & \alpha^{n-1} & 1 & 0 & \cdots & 0\\ 1 & \alpha^2 & \cdots & \alpha^{2(n-1)} & 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \alpha^{n-k-1} & \cdots & \alpha^{(n-k-1)(n-1)} & 0 & 0 & \cdots & 1\end{pmatrix}_{(n-k)\times(2n-k-1)}.$$

The GRDP-based MSSS scheme uses this matrix to compute public parities, achieving near-optimal XOR complexity (fewer operations than IP codes) [7, 8].

2.3.3. Definition of main symbols

We default all mentioned vectors to column vectors and $x^m+1=(1+x)g(x)m(x)$ unless otherwise stated. For the reader's convenience, we list the main symbols used in this paper:

$\mathbb{F}_q$, $\mathbb{N}$	Finite field with size q, the set of natural numbers
$\mathbb{R}_m$	Polynomial ring $\mathbb{R}_m:=\mathbb{F}_q[x]/\langle x^m+1\rangle$
$\mathbb{F}_q^m$	m-dimensional linear vector space over $\mathbb{F}_q$
κ	$\min\{\deg(m_i(x))\}$, where $m_i(x)$ are irreducible factors of $m(x)$
m,ℓ	Sub-packetization, shape of array code is $m\times\ell$
r,k	Number of parity columns (redundancy), number of information columns
$\mathbf{0}_m(\mathbf{I}_m)$	m-order full-zero (identity) matrix
H_1,H_2	Parity-check matrices of Construction I and Construction II
$\mathbf{C}_1,\mathbf{C}_2$	Proposed array codes in Construction I and Construction II
$\mathbf{a}^\top$, $A^\top$	Transposition of vector $\mathbf{a}$ (matrix A)
$[n]$, $\langle a\rangle_m$	Sets $\{0,1,\ldots,n-1\}$, the entry $(a \bmod m)\in[m]$
$\lfloor a\rfloor(\lceil a\rceil)$	Maximum (smallest) integer not greater (less) than a
$b(\cdot)$	Denotes number of 1's in the binary representation of the element
$\mathcal{T}_d$	Downward-cyclic operator $\mathcal{T}_d(\boldsymbol{v})=(v_{m-1},v_0,\ldots,v_{m-2})^\top$

3. New constructions of q-ary array codes with local properties

In this section, we develop a new family of q-ary long array codes with local properties. Our approach exploits the parity-check matrix formulation of linear codes, but unlike classical

constructions, it imposes a GCD-Constraint on the involved polynomials, which simultaneously guarantees local properties and is (ℓ, k)-recoverable. Throughout, we assume $\gcd(m, q) = 1$.

We first recall a standard form of the Chinese remainder theorem (CRT), which is key to our proposed construction.

Lemma 3.1. *[25] Let $x^m + 1 = d(x) \cdot m(x)$ with $\gcd(d(x), m(x)) = 1$ in $\mathbb{F}_q[x]$. Then, the ring $\mathbb{R}_m = \mathbb{F}_q[x]/\langle x^m + 1\rangle$ is isomorphic to the direct product*

$$\mathbb{R}_m \cong \mathbb{F}_q[x]/\langle d(x)\rangle \otimes \mathbb{F}_q[x]/\langle m(x)\rangle. \tag{3.1}$$

Consequently, any polynomial $t(x) \in \mathbb{R}_m$ corresponds uniquely to a pair $(t_d(x), t_m(x))$ via $t_d(x) = t(x) \bmod d(x)$, $t_m(x) = t(x) \bmod m(x)$. Moreover, there exists a polynomial $u(x)$ such that $u(x)d(x) \equiv 1 \bmod m(x)$, and the inverse isomorphism ϕ^{-1} is given by

$$(t_d(x), t_m(x)) \longmapsto t_d(x) \cdot u(x)d(x) + t_m(x) \cdot v(x)m(x) \bmod (x^m + 1), \tag{3.2}$$

where $v(x)$ satisfies $v(x)m(x) \equiv 1 \bmod d(x)$.

Definition 3.2. (GCD-Constraint) Let $s \geq 2$ be an integer and $a_0(x), a_1(x), \ldots, a_{s-1}(x)$ be s pairwise different polynomials of $\mathbb{R}_m$. For all $0 \leq i < j \leq s-1$, if it always holds that

$$\gcd(a_i(x) + a_j(x), x^m + 1) = (1 + x)g(x), \tag{3.3}$$

then we say that these polynomials $a_0(x), a_1(x), \ldots, a_{s-1}(x)$ meet the GCD-Constraint.

Generic Construction I [26]: Let $\mathbf{C}_1$ be an $(\ell, k, d_A; m, g(x))_q$-array code with shape $m \times \ell$ that satisfies:

i) Its parity-check matrix H_1 is in the form

$$H_1 = \begin{pmatrix} \mathbf{I}_m & \mathbf{I}_m & \cdots & \mathbf{I}_m \\ A_0 & A_1 & \cdots & A_{\ell-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_0^{r-1} & A_1^{r-1} & \cdots & A_{\ell-1}^{r-1} \end{pmatrix}_{mr \times m\ell}, \tag{3.4}$$

where $\mathbf{I}_m$ is the m-order identity matrix, and A_s^t is the circulant matrix associated to polynomial $a_s^t(x) \bmod (x^m + 1)$, $s \in [\ell]$, $1 \leq t \leq r-1$, $1 \leq r \leq \ell - 1$;

ii) These polynomials $a_0(x), \ldots, a_{\ell-1}(x)$ of $\mathbb{R}_m$ meet the GCD-Constraint provided in Definition 3.2;

iii) Each column of every codeword of $\mathbf{C}_1$ is located in the cyclic code $\langle (1 + x)g(x)\rangle$.

Next, we need to prove the existence of the array code mentioned above, where each column is in the cyclic code $\langle (1 + x)g(x)\rangle$.

Proof. Let $a(x) = a_0 + a_1 x + \cdots + a_{m-1}x^{m-1} \in \mathbb{R}_m$, and let $\boldsymbol{a} = (a_0, a_1, \ldots, a_{m-1}) \in \mathbb{F}_q^m$. By Lemma 2.2, the vector $\boldsymbol{a}$ and the circulant matrix A in (2.1) are equivalent representations of the polynomial $a(x)$. Hence, we may rewrite (3.4) in polynomial form as

$$H_1 = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ a_0(x) & a_1(x) & \cdots & a_{\ell-1}(x) \\ \vdots & \vdots & \ddots & \vdots \\ a_0^{r-1}(x) & a_1^{r-1}(x) & \cdots & a_{\ell-1}^{r-1}(x) \end{pmatrix}_{r \times \ell}. \tag{3.5}$$

Any $r \times r$ submatrix of H_1 is a Vandermonde matrix, and the determinant can be written as the multiplication of power of r different factors $a_i(x) + a_j(x)$, where $0 \le i \ne j \le \ell - 1$.

Thus, it is sufficient to show that the equation $(a_i(x) + a_j(x))s(x) = c(x) \mod (x^m + 1)$ has a unique solution $s(x) \in \langle (1 + x)g(x) \rangle$, where $c(x) \in \langle (1 + x)g(x) \rangle$.

Since $s(x)$ and $c(x)$ are both multiples of $(1 + x)g(x)$, we have

$$\begin{aligned} &(a_i(x) + a_j(x))s(x) = c(x) \mod (x^m + 1) \\ \Leftrightarrow &(a_i(x) + a_j(x))u(x)(1 + x)g(x) = v(x)(1 + x)g(x) \mod (1 + x)g(x)m(x) \\ \Leftrightarrow &(a_i(x) + a_j(x))u(x) = v(x) \mod m(x). \end{aligned}$$

By Eq (3.3), we can deduce that $\gcd(a_i(x) + a_j(x), m(x)) = 1$, that is, the inverse of $(a_i(x) + a_j(x))$ exists in $\mathbb{F}_q[x]/\langle m(x) \rangle$. This completes the proof. □

Remark 3.3. When the array code $\mathbf{C}_1$ is deployed in a distributed storage system, each column of a codeword resides on a distinct storage node. If the cyclic code $\langle (1 + x)g(x) \rangle$ has minimum Hamming distance d_H, then every node can independently correct up to $d_H - 1$ erasures or a burst of length $1 + \deg(g(x))$. This is the column local property.

Explicit Construction I: Since $\gcd(m, q) = 1$, then the polynomial $m(x) = \frac{x^m+1}{(1+x)g(x)}$ in $\mathbb{F}_q[x]$ can be completely factorized as

$$m(x) = m_1(x)m_2(x) \cdots m_t(x). \tag{3.6}$$

Assume that $\min\{\deg(m_i(x))\} = \kappa$, $i \in [t]^+$. For $j \in [\kappa]$, $0 \le i \le 2^j - 1$, define recursively

$$a_0(x) = 0, \quad a_{i+2^j}(x) = a_i(x) + (x^j + x^{j+1})g(x). \tag{3.7}$$

Choosing the parity-check matrix H_1 as in (3.4) with these polynomials yields an $(\ell, k, d_A; m, g(x))$-array code C_1 whose code length ℓ can reach 2^κ.

To better illustrate the flexibility and the long code length of our construction, we provide an explicit example based on the recursive polynomial generation outlined in explicit Construction I.

Example 3.4. Let $m = 9$ and $q = 2$. The polynomial $x^9 + 1$ over $\mathbb{F}_2[x]$ can be completely factorized as

$$x^9 + 1 = (1 + x)(x^2 + x + 1)(x^6 + x^3 + 1).$$

Choose $g(x) = x^2 + x + 1$, so that $(1 + x)g(x) = 1 + x^3$. Then, $m(x) = \frac{x^9+1}{(1+x)g(x)} = x^6 + x^3 + 1$, which is irreducible of degree $\kappa = 6$. Consequently, the explicit construction allows a maximum code length of $2^6 = 64$.

According to the recursive definition Eq (3.7) with $g(x) = x^2 + x + 1$, the first four polynomials are:

$$\begin{aligned} a_0(x) &= 0, & a_1(x) &= 1 + x^3, \\ a_2(x) &= x + x^4, & a_3(x) &= 1 + x + x^3 + x^4. \end{aligned}$$

Their coefficient vectors of length 9 are

Polynomial	Coefficient vector
$a_0(x)$	$(0, 0, 0, 0, 0, 0, 0, 0, 0)$
$a_1(x)$	$(1, 0, 0, 1, 0, 0, 0, 0, 0)$
$a_2(x)$	$(0, 1, 0, 0, 1, 0, 0, 0, 0)$
$a_3(x)$	$(1, 1, 0, 1, 1, 0, 0, 0, 0)$

One verifies that these polynomials satisfy the GCD-Constraint. Substituting the associated 9×9 circulant matrices A_i into Eq (3.4) yields the parity-check matrix H_1 of the $(4, 2, d_A; 9, x^2 + x + 1)$-array code C_1 of shape 9×4.

As a concrete codeword, take information columns

$$\boldsymbol{c}_0 = (1, 0, 0, 1, 0, 0, 0, 0, 0)^\top, \quad \boldsymbol{c}_1 = (1, 1, 0, 1, 1, 0, 0, 0, 0)^\top.$$

Then, the parity columns are obtained by solving $\boldsymbol{c}_2 + \boldsymbol{c}_3 = \boldsymbol{c}_0 + \boldsymbol{c}_1$ and $A_2\boldsymbol{c}_2 + A_3\boldsymbol{c}_3 = A_0\boldsymbol{c}_0 + A_1\boldsymbol{c}_1$ (for the detailed procedure, please refer to Section 4 on decoding). The resulting codeword, written columnwise, is

$$\begin{aligned}
\boldsymbol{c}_0 &= (1, 0, 0, 1, 0, 0, 0, 0, 0)^\top, \\
\boldsymbol{c}_1 &= (1, 1, 0, 1, 1, 0, 0, 0, 0)^\top, \\
\boldsymbol{c}_2 &= (1, 0, 1, 1, 0, 1, 0, 0, 0)^\top, \\
\boldsymbol{c}_3 &= (1, 1, 1, 1, 1, 1, 0, 0, 0)^\top.
\end{aligned}$$

Each column belongs to the cyclic code $\langle 1 + x^3 \rangle$ (the sum of entries at positions $i, i+3, i+6$ modulo 9 is zero), confirming the column local property.

Remark 3.5. The extended Blaum–Roth (EBR) codes introduced in [13] can be recovered as a special case of Generic Construction I by taking $a_i(x) = x^{(m-i) \bmod m}$ with m prime. That choice restricts the code length to $\ell \leq m$, whereas our explicit construction removes this limitation and allows the length to grow exponentially with κ, the degree of an irreducible factor of $m(x)$. Example 3.4 illustrates a case where $\kappa = 6$, enabling code lengths up to 64 while preserving column local properties.

Definition 3.6. An array code $\mathbf{C}$ of length ℓ over $\mathbb{F}_q$ is called (ℓ, k)-recoverable if any k columns of a codeword suffice to reconstruct the whole codeword, where k is the number of information columns.

To prove that array code $\mathbf{C}_1$ is (ℓ, k)-recoverable, we need the following lemma.

Lemma 3.7. *Let $\overline{\mathbf{C}}_1$ be an $(\ell, k, d_A; m - \tau)_q$-array code whose parity-check matrix is given by*

$$\overline{H}_1 = \begin{pmatrix} \mathbf{I}_{m-\tau} & \mathbf{I}_{m-\tau} & \cdots & \mathbf{I}_{m-\tau} \\ A_0^{[1,\tau]} & A_1^{[1,\tau]} & \cdots & A_{\ell-1}^{[1,\tau]} \\ \vdots & \vdots & \ddots & \vdots \\ A_0^{[r-1,\tau]} & A_1^{[r-1,\tau]} & \cdots & A_{\ell-1}^{[r-1,\tau]} \end{pmatrix}_{r(m-\tau)\times(m-\tau)\ell}, \tag{3.8}$$

where the polynomials $a_0(x), \ldots, a_{\ell-1}(x)$ of $\mathbb{R}_m$ satisfy the GCD-Constraint given in Definition 3.2. Here, $A_s^{[t,\tau]}$ denotes the $(m-\tau)$-order matrix obtained by puncturing the circulant matrix A_s^t associated with $a_s^t(x) \bmod (x^m + 1)$, with $\tau = 1 + \deg(g(x))$, $s \in [\ell]$, $1 \leq t \leq r - 1$, and $1 \leq r \leq \ell - 1$.

There is a one-to-one mapping relationship between the code $\mathbf{C}_1$ (Generic Construction I) and $\overline{\mathbf{C}}_1$.

Proof. Suppose that codewords $\overline{\boldsymbol{c}} \in \overline{\mathbf{C}_1}$ and $\boldsymbol{c} \in \mathbf{C}_1$ satisfy $\sum_{k=0}^{\ell-1} A_k^i \boldsymbol{c}_k = \mathbf{0}_{m\times 1}$. Define $\boldsymbol{c}_i^* = (\overline{\boldsymbol{c}}_i, 0, \ldots, 0) \in \mathbb{F}_q^m$. Let $x^m + 1 = (1 + x)g(x)m(x) = d(x)m(x)$, so $\deg(d(x)) = \tau$. By the GCD-Constraint, each $a_k(x)$ can be written as

$$a_k(x) = v(x) + d(x)b_k(x)$$

for some common $v(x) \in \mathbb{R}_m$ and $b_k(x) \in \mathbb{R}_m$. Let V, D, and B_k be the circulant matrices of $v(x)$, $d(x)$ and $b_k(x)$, respectively. Then, by Lemma 2.2, $A_k^t = (V + DB_k)^t = V^t + DQ_k$ for some matrix Q_k.

From the first block row of H_1, we have $\sum_{k=0}^{\ell-1} \overline{\boldsymbol{c}}_k = 0$. Hence, $\sum_{k=0}^{\ell-1} \boldsymbol{c}_k^* = 0$ in $\mathbb{F}_q^m$.

Assume that the parity-check equation $\sum_{k=0}^{\ell-1} A_k^{[t,\tau]} \overline{c}_k = 0$ holds. Lifting to the row-punctured form (deleting the last τ rows of the circulant matrix) gives

$$\sum_{k=0}^{\ell-1} A_k^{\{t,\tau\}} \boldsymbol{c}_k^* = 0.$$

Substituting $A_k^t = V^t + DQ_k$ and deleting the last τ rows, we obtain

$$\sum_{k=0}^{\ell-1} (V^{\{t,\tau\}} + D^{\{\tau\}} Q_k) \boldsymbol{c}_k^* = V^{\{t,\tau\}} \sum_{k=0}^{\ell-1} \boldsymbol{c}_k^* + D^{\{\tau\}} \sum_{k=0}^{\ell-1} Q_k \boldsymbol{c}_k^* D^{\{\tau\}} \left(\sum_{k=0}^{\ell-1} Q_k \boldsymbol{c}_k^* \right) = 0.$$

Since $d(x) \mid x^m + 1$ and $\deg(d(x)) = \tau$, we have $\operatorname{rank}(D^{\{\tau\}}) = m - \tau$. Because $D^{\{\tau\}}$ has full row rank, we obtain $\sum_{k=0}^{\ell-1} Q_k \boldsymbol{c}_k^* = 0$. Combining $\sum_{k=0}^{\ell-1} \boldsymbol{c}_k^* = 0$ and $\sum_{k=0}^{\ell-1} Q_k \boldsymbol{c}_k^* = 0$, we can directly obtain

$$\sum_{k=0}^{\ell-1} A_k^t \boldsymbol{c}_k^* = V^t \sum_{k=0}^{\ell-1} \boldsymbol{c}_k^* + D \sum_{k=0}^{\ell-1} Q_k \boldsymbol{c}_k^* = 0.$$

That is,

$$\sum_{k=0}^{\ell-1} A_k^{[i,\tau]} \overline{\boldsymbol{c}}_k = \mathbf{0}_{(m-\tau)\times 1} \Longleftrightarrow \sum_{k=0}^{\ell-1} A_k^i \boldsymbol{c}_k^* = \mathbf{0}_{m\times 1}. \tag{3.9}$$

Equation (3.9) indicates that these two codes satisfy the same constraint, implying a special mapping between their codewords. Next, we regard each column of the array codes as a polynomial over $\mathbb{R}_m$ to obtain the explicit mapping.

Let $x^m + 1 = (1 + x)g(x)m(x)$, $a(x) \in \langle (1 + x)g(x) \rangle$, and $s(x) \in \mathbb{F}_q[x]/\langle m(x) \rangle$. Since $x^m + 1$ has no repeated factors, then by Lemma 3.1, there exists an isomorphism θ from $\mathbb{F}_q[x]/\langle m(x) \rangle$ to $\langle (1 + x)g(x) \rangle$, which is defined by

$$\begin{aligned} &\theta(s(x)) = t(x)(1 + x)g(x)s(x) \mod (x^m + 1), \\ &\theta^{-1}(a(x)) = a(x) \mod m(x), \end{aligned} \tag{3.10}$$

where $t(x) \in \mathbb{R}_m$ meets $t(x)(1 + x)g(x) = 1 \bmod m(x)$. □

Example 3.8. Consider the binary array (left) $\boldsymbol{c} \in C_1$ (an $(6, 3, 4; 5, 1)$-array code) in Explicit Construction I. Regarding each column as a polynomial over $\mathbb{R}_5$, one can verify that the transformed array (right) in the $(6, 3, 4; 4)$-MDS array code $\overline{C_1}$ as given by Eq (3.10):

$$\begin{array}{|c|c|c|c|c|c|} \hline 1&1&0&1&0&1 \\ \hline 1&0&1&1&1&0 \\ \hline 1&0&0&0&0&1 \\ \hline 1&1&0&1&0&1 \\ \hline 0&0&1&1&1&1 \\ \hline \end{array} \quad \begin{array}{c} \theta^{-1}(a(x)) \\ \rightleftharpoons \\ \theta(s(x)) \end{array} \quad \begin{array}{|c|c|c|c|c|c|} \hline 1&1&1&0&1&0 \\ \hline 1&0&0&0&0&1 \\ \hline 1&0&1&1&1&0 \\ \hline 1&1&1&0&1&0 \\ \hline 0&0&0&0&0&0 \\ \hline \end{array} .$$

The isomorphism θ from $\mathbb{F}_2[x]/\langle m(x) \rangle$ to $\langle 1 + x \rangle$ is given by

$$\theta(s(x)) = (x + x^3)(1 + x)s(x) = (x + x^2 + x^3 + x^4)s(x) \bmod (x^5 + 1),$$

$$\theta^{-1}(a(x)) = a(x) \bmod (1 + x + x^2 + x^3 + x^4).$$

The array on the left is a codeword of $\mathbf{C}_1$ with shape 5×6, where each column is a polynomial in $\langle 1 + x \rangle$. Applying θ^{-1} to each column (that is, reducing modulo $1 + x + x^2 + x^3 + x^4$) yields the array on the right, which is a codeword of the punctured code $\overline{\mathbf{C}_1}$ with shape 4×6. Conversely, applying θ to each column of the punctured codeword recovers the original codeword in $\mathbf{C}_1$.

Theorem 3.9. *The $(\ell, k, d_A; m, g(x))_q$-array code $\mathbf{C}_1$ in Generic Construction I is (ℓ, k)-recoverable, where ℓ is its code length and k is the number of information columns.*

Proof. By Theorem 8 of Ref. [10], it is easy to see that the $(\ell, k, \ell - k; m - \tau)_q$-array code $\overline{\mathbf{C}_1}$ in Lemma 3.7 is MDS. Since there exists a one-to-one mapping between $\mathbf{C}_1$ and $\overline{\mathbf{C}_1}$, then $\mathbf{C}_1$ is (ℓ, k)-recoverable. □

4. Erasure decoding algorithm for array codes with local properties

Erasure is a common type of error in storage systems, and encoding can in fact be regarded as a fixed pattern of erasure decoding. Building on this observation, we first describe a decoder for array codes with local properties. To facilitate the description, in this section, we adopt a polynomial representation to characterize both the array codes and their parity-check matrices.

Let $\mathbf{C}$ be an $(\ell, k, d_A; m, g(x))_q$-array code with parity-check matrix $H_1 = (h_0, h_1, \ldots, h_{\ell-1})$ in Eq (3.5), where each h_i is an $r \times 1$ matrix over $\mathbb{R}_m$. A codeword of $\mathbf{C}$ is denoted as $\boldsymbol{c} = (c_i)_{i\in[\ell]}$, where $c_i \in \mathbb{R}_m$. Denote the received message as $\boldsymbol{r} = (r_i)_{i\in[\ell]}$, where $r_i \in \mathbb{R}_m$. For $k \in \mathcal{J}$, we have $r_k = 0$, whereas for the remaining indices, $r_k = c_k$. Assume that $\mathcal{J} = \{j_0, j_1, \ldots, j_\rho\} \subseteq [\ell]$ is the erased positions with position with $\rho \leq r$. Define the error vector as $\boldsymbol{e} = \boldsymbol{c} - \boldsymbol{r} = (e_i)_{i\in[\ell]}$. Thus, our goal for the decoder is to recover the erased vectors e_k for $k \in \mathcal{J}$.

Let H_{dec} be the ρ-order matrix formed by the first ρ rows of $(h_i)_{i\in\mathcal{J}}$ and the syndrome of $\boldsymbol{r}$ be

$$\boldsymbol{s} = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{\rho-1} \end{pmatrix} = H_1\boldsymbol{r}^\top = H_1(\boldsymbol{c}^\top - \boldsymbol{e}^\top) = -H_1\boldsymbol{e}^\top = -H_{dec}((e_k)_{k\in\mathcal{J}})^\top . \tag{4.1}$$

From Eq (4.1), we can consider that the decoding of array codes with local properties consists of the following two steps.

Step 1: Calculate syndrome $\boldsymbol{s} = H_1 \cdot \boldsymbol{r}^\top$.

Step 2: Find out the solution of $-H_{dec}((e_k)_{k\in\mathcal{J}})^\top = \boldsymbol{s}$.

Notably, as mentioned in Section 3, the second step always has a unique solution in $\langle (1 + x)g(x) \rangle$.

4.1. The fast syndrome calculation for array codes in explicit construction

Let $\mathbf{C}$ be an $(\ell, k, d_A; m, g(x))_2$-array code in Explicit Construction I. The syndrome $\boldsymbol{s} = (s_j)$, $j \in [r]$, $s_j \in \mathbb{R}_m$ in Eq (4.1) can be computed as

$$s_j = \sum_{i=0}^{\ell-1} a_i^j(x) r_i \, , \tag{4.2}$$

where the multiplication of polynomials over $\mathbb{R}_m$ is implemented via cyclic shifts.

Define $\delta = \lceil \log_2 \ell \rceil$ and $L = 2^\delta$. Obviously, If j is a power of two or $j = 0$, $\omega \in [\delta]$ and $i \in [2^\omega]$, then

$$a_0^j(x) = 0, \quad a_{i+2^\omega}^j(x) = a_i^j(x) + ((x^\omega + x^{\omega+1})g(x))^j \bmod (x^m + 1) . \tag{4.3}$$

For convenience, use $g_\omega(x)$ to represent $(x^\omega + x^{\omega+1})g(x)$. Then Eq (4.2) can be rewritten as

$$s_j(r) = \sum_{i=0}^{L/2-1} (a_i^j(x)r_i + a_{i+L/2}^j(x)r_{i+L/2}) = \sum_{i=0}^{L/2-1} (a_i^j(x)(r_i + r_{i+L/2}) + (g_{\delta-1}(x))^j r_{i+L/2}). \tag{4.4}$$

For $\omega \in [\delta]$, we recursively define

$$\begin{cases} \widetilde{r_{\delta-1-\omega}} = \sum_{i=0}^{N/2^{\omega+1}-1} r_{i+N/2^{\omega+1}}^{[\omega-1]}, \\ r^{[\omega]} = (r_0^{[\omega-1]} + r_{N/2^{\omega+1}}^{[\omega-1]}, r_1^{[\omega-1]} + r_{N/2^{\omega+1}+1}^{[\omega-1]}, \cdots, r_{N/2^{\omega+1}-1}^{[\omega-1]} + r_{N/2^\omega-1}^{[\omega-1]}). \end{cases} \tag{4.5}$$

Combining Eq (4.3) with Eq (4.5), we can simplify the syndrome computation in Eq (4.4) as

$$\begin{aligned} s_j(r) &= \sum_{i=0}^{L/2-1} a_i^j(x)r_i^{[0]} + (g_{\delta-1}(x))^j \widetilde{r_{\delta-1}} = s_j(r^{[0]}) + (g_{\delta-1}(x))^j \widetilde{r_{\delta-1}} \\ &= \cdots = a_0^j(x)r^{[\delta-1]} + \sum_{\omega=0}^{\delta-1} (g_\omega(x))^j \widetilde{r_\omega} = \sum_{\omega=0}^{\delta-1} (g_\omega(x))^j \widetilde{r_\omega} . \end{aligned} \tag{4.6}$$

In particular, for $j = 0$ or $j = 2^n$, the syndromes s_j can be calculated in Algorithm 1, where the downward-cyclic operator $\mathcal{T}_d$ acts on an m-length vector $\boldsymbol{v} = (v_0, v_1, \ldots, v_{m-1})^\top$ as $\mathcal{T}_d(\boldsymbol{v}) = (v_{m-1}, v_0, \ldots, v_{m-2})^\top$.

Algorithm 1 Computing syndromes s_j based on Eq (4.6).

Require: $\boldsymbol{r} = (r_0, r_1, \ldots, r_{\ell-1})$, $r_i \in \mathbb{R}_m$, let $\boldsymbol{r}_i = \Phi(r_i)$ (Using Lemma 2.2);
$g(x) = g_0 + g_1 x + \cdots + g_\nu x^\nu$ with $g_0 = 1$.
Ensure: $(\boldsymbol{s}_j)$, $j = 0$ or $j = 2^n$.

$\delta \leftarrow \lceil \log_2 \ell \rceil$, $L \leftarrow 2^\delta$
if $\ell < L$ **then**
  for $k = \ell$ **to** $L - 1$ **do**
    $\boldsymbol{r}_k \leftarrow \boldsymbol{0}_{m\times 1}$
  end for
end if
$\omega \leftarrow \delta - 1$, $L \leftarrow 2^{\omega+1}$
$\widetilde{\boldsymbol{r}_\omega} \leftarrow \sum_{i=0}^{\ell-L/2-1} \boldsymbol{r}_{L/2+i}$
for $i = 0$ **to** $\ell - L/2 - 1$ **do**
  $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_i + \boldsymbol{r}_{L/2+i}$
end for
while $\omega \geq 1$ **do**
  $\omega \leftarrow \omega - 1$, $L \leftarrow 2^{\omega+1}$
  $\widetilde{\boldsymbol{r}_\omega} \leftarrow \sum_{i=0}^{L/2-1} \boldsymbol{r}_{L/2+i}$
  for $i = 0$ **to** $L/2 - 1$ **do**
    $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_i + \boldsymbol{r}_{L/2+i}$
  end for
end while
$\widetilde{\boldsymbol{s}_j} \leftarrow \widetilde{\boldsymbol{r}_0}$
for $\omega = 1$ **to** $\delta - 1$ **do**
  $\widetilde{\boldsymbol{s}_j} \leftarrow \widetilde{\boldsymbol{s}_j} + \mathcal{T}_d^{\langle j\omega\rangle_m}(\widetilde{\boldsymbol{r}_\omega})$
end for
$q_0 \leftarrow 1$, $q_{\nu+1} \leftarrow g_\nu$
for $i = 1$ **to** ν **do**
  $q_i \leftarrow g_{i-1} + g_i$
end for
for $k = 1$ **to** $\nu + 1$ **do**
  $\widetilde{\boldsymbol{s}_j} \leftarrow \widetilde{\boldsymbol{s}_j} + q_k \cdot \mathcal{T}_d^{\langle jk\rangle_m}(\widetilde{\boldsymbol{s}_j})$
end for
return $(\boldsymbol{s}_0 \leftarrow \boldsymbol{r}_0, \boldsymbol{s}_j \leftarrow \widetilde{\boldsymbol{s}_j})$

However, for other values of j that are not powers of two, $a^j(x) = (a_i(x) + (x^\omega + x^{\omega+1})g(x))^j$ will generate complicated intermediate terms (more than just the two simple terms in Eq (4.3)). In fact, we can also expand this equation and rewrite it as a sum of two terms so as to exploit the idea of recursive branching. Below, we illustrate this with a toy example.

Example 4.1. For an integer $i \in \mathbb{N}$, let $(b_{n-1} \cdots b_0)_2$ be the binary representation of i without leading zeros, where n is the number of bits. We define

$$I_i^* = \left\{(x_{n-1} \cdots x_0)_2 \ : \ x_j \in \{0, 1\},\ x_j \leq b_j \text{ for every } j, \text{ and } (x_{n-1} \cdots x_0)_2 \neq i\right\}.$$

In other words, I_i^* consists of all binary strings of length n obtained from the binary expansion of i by changing some (possibly all) of the 1-bits to 0, but excluding the string that equals i itself. For example, $I_4^* = \{(000)_2\}$, $I_5^* = \{(000)_2, (001)_2, (100)_2\}$, $I_6^* = \{(000)_2, (010)_2, (100)_2\}$.

In the algebraic structure with characteristic 2, the expansion of $(a+b)^n$ can be expressed using the binary representation of n. Specifically, let the binary expansion of n be

$$n = 2^{t_1} + 2^{t_2} + \cdots + 2^{t_s},$$

where $t_1 > t_2 > \cdots > t_s$. Then we have

$$(a+b)^n = \prod_{i=1}^{s}(a+b)^{2^{t_i}} = \prod_{i=1}^{s}(a^{2^{t_i}} + b^{2^{t_i}}) = a^n + \sum_{i \in I_n^*} a^i b^{n-i}. \tag{4.7}$$

For example, if we take $n = 7$, by Eq (4.7), we obtain that

$$(a+b)^7 = (a+b)^4(a+b)^2(a+b) = \sum_{i \in I_7^*} a^i b^{7-i} = a^7 + a^6 b + a^5 b^2 + a^4 b^3 + a^3 b^4 + a^2 b^5 + a b^6 + b^7.$$

Example 4.1 reveals that $a^j(x)$ in $\mathbb{R}_m$ decomposes into a sum over all binary subterms, which naturally aligns with the recursive structure of the polynomial representation. Motivated by this observation, we can rewrite Eq (3.7) from a basis vector perspective as follows,

$$a_i(x) = (i_0 + i_1 \cdot x + \cdots + i_{\kappa-1} \cdot x^{\kappa-1})(1+x)g(x), \tag{4.8}$$

where $\kappa = \min\{\deg(m_i(x))\}$ in Eq (3.6) and $i = \sum_{j=0}^{n-1} i_j \cdot 2^j$, $i_j \in \mathbb{F}_2$.

Proposition 4.2. *When L is a power of 2 and $L \leq 2^n$, the Vandermonde matrix $H_{r\times L}$ takes the form of Eq* (3.5)*, where the polynomials $a_i(x)$ are defined in Eq* (4.8)*, $i \in [L]$. Then H_1 can be decomposed as*

$$H_{r\times L} = S_{r\times L} \cdot R_L, \tag{4.9}$$

where $S_{r\times L}$ is an $r \times L$ sparse matrix and R_L is an $L \times L$ matrix. R_L is an Reed-Muller matrix defined by

$$R_L = \begin{pmatrix} R_{L/2} & R_{L/2} \\ \mathbf{0}_{L/2} & R_{L/2} \end{pmatrix}, \text{with } R_1 = 1.$$

In addition, the entry of row i and column j in the matrix $S_{r\times L}$ is denoted by

$$S_{i,j} = \begin{cases} 1 & \text{if } i = 0, j = 0 \\ x^{i\cdot j} \cdot ((1+x)g(x))^i & \text{if } b(i) \geq 1, b(j) = 1 \\ f_{i,j} \cdot ((1+x)g(x))^i & \text{if } b(i) \geq b(j) > 1 \\ 0 & \text{otherwise} \end{cases},$$

where $f_{i,j}$ is a nonzero value that depends only on the indexes i and j.

Proof. Let $m_l(x)$ be an irreducible polynomial of degree κ over $\mathbb{F}_2[x]$, and let $(1, x, \ldots, x^{\kappa-1})$ be a basis of the finite field $\mathbb{F}_2[x]/\langle m_l(x)\rangle$. The elements of $\mathbb{F}_2[x]/\langle m_l(x)\rangle$ are denoted as $\{w_i(x)\}_{i=0}^{2^\kappa-1}$, where each $\sum_{j=0}^{\kappa-1} i_j \cdot x^j$ and $i = \sum_{j=0}^{n-1} i_j \cdot 2^j$ (the binary expansion of i), $i_j \in \mathbb{F}_2$.

By Lemma 1 in Ref. [17], a Vandermonde matrix H over $\mathbb{F}_2[x]/\langle m_l(x)\rangle$ can be decomposed as $H = E_{r\times L} \cdot R_L$, where $E_{r\times L}$ is a sparse matrix. Since $\gcd(m, q) = 1$, according to Lemma 3.1, we can obtain $H_1 = \Lambda_{r\times r} \cdot H$, where $\Lambda_{r\times r} = \mathrm{Diag}((1+x)g(x))^0, \ldots, (1+x)g(x))^{r-1})$. □

Applying Proposition 4.2, Eq (4.1) can be rewritten as

$$\boldsymbol{s} = H_1 \boldsymbol{r}^\top = S_{r\times L} \cdot R_L \cdot \boldsymbol{r}^\top = S_{r\times L} \cdot \boldsymbol{y}. \tag{4.10}$$

For completeness of decoding, we supplement the Algorithm 2 for computing $\boldsymbol{y} = R_N \cdot \boldsymbol{r}^\top$.

Remark 4.3. Note that after executing Algorithm 2, the output is not $\boldsymbol{y} = \{\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N-1}\}$ in sequential order; instead, all elements of $\boldsymbol{y}$ are listed in ascending order of the corresponding values of $b(i)$, which we denote as $\{\boldsymbol{y}_b^i\}_{i=0}^{depth}$. For example, taking $L = 8$ and $r = 4$, the output of Algorithm 2 is $\{\boldsymbol{y}_b^i\}_{i=0}^{2} = \{\boldsymbol{y}_b^0, \boldsymbol{y}_b^1, \boldsymbol{y}_b^2\} = \{\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_4, \boldsymbol{y}_3, \boldsymbol{y}_5, \boldsymbol{y}_6\}$.

Remark 4.4. With the redundancy $r \le 3$, we can use Algorithm 1 to compute syndromes in Eq (4.1). Both Algorithm 1 and Algorithm 2 employ a recursive strategy, and we collectively refer to them as the scheduled algorithm. Furthermore, Lemma 6 in Ref. [17] provides an upper bound on the complexity of Algorithm 2 (Reed-Muller transform), that is, $XORs \le m(t+1)(L-1) = m(\lfloor \log_2 r \rfloor + 1)(L-1)$.

Algorithm 2 $\mathcal{RM}(L, \boldsymbol{r}, depth, shift)$ [17].

Require: $L = 2^n$, $\boldsymbol{r} = (\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{L-1})$, $shift = 0$, $t = \lfloor \log_2 r \rfloor$, and $depth = t$
Ensure: $\{\boldsymbol{y}_b^i\}_{i=0}^{depth}$

```
if L = 1 then
    Return y_shift = r_0
end if
Let r_i ← r_i + r_{i+L/2}, for i = 0, 1, ..., L/2 − 1
Recall RM(L/2, r(0 : L/2), depth, shift)
if depth = 1 then
    Let y_{2^{n−1}+shift} = Σ_{i=L/2}^{L−1} r_i
else if depth > 1 then
    Recall RM(L/2, r(L/2 : L), depth − 1, L/2 + shift)
end if
Return
```

4.2. Solving Vandermonde linear equations

In Step 2, we use the LU decomposition presented in [27] to solve the Vandermonde linear equations $\boldsymbol{s} = -H_{dec}((e_k)_{k\in\mathcal{J}})^\top$. This step requires $r(r-1)$ additions, $r(r-1)/2$ multiplications, and $r(r-1)/2$ divisions by factors of the form $a(x)$, where $a(x) \in \langle (1+x)g(x) \rangle$. When r is much smaller than ℓ, the overall complexity is dominated by Step 1, resulting in that the complexity of this part can be ignored.

Before applying this LU-based decoding method, we must device an efficient division by $a(x)$ over $\mathbb{R}_m$, where $\gcd(a(x), m(x)) = 1$. Specifically, we aim to find $r(x) \in \langle (1+x)g(x) \rangle$ satisfying the equation

$$a(x)r(x) = s(x) \mod (x^m + 1), \quad a(x), s(x) \in \langle (1+x)g(x) \rangle . \tag{4.11}$$

Lemma 4.5. *Denote* $\mathcal{D} = \langle (1+x)g(x) \rangle \subseteq \mathbb{R}_m$. *For any* $a(x), s(x) \in \mathcal{D}$ *satisfying* $\gcd(a(x), m(x)) = 1$, *Eq* (4.11) *has a unique solution* $r(x) \in \mathcal{D}$, *given by*

$$r(x) = \big((a(x) \bmod m(x))^{-1} \cdot (s(x) \bmod m(x))\big) \cdot (1+x)g(x) \bmod (x^m + 1) .$$

Proof. When $\gcd(m, q) = 1$ and $x^m + 1 = (1 + x)g(x)m(x) = d(x)m(x)$, we have $\gcd(d(x), m(x)) = 1$. By Lemma 3.1, Eq (4.11) can be rewritten as

$$(a(x) \bmod m(x)) \cdot (r(x) \bmod m(x)) \equiv s(x) \bmod m(x) \pmod{m(x)}.$$

Define $a'(x) = a(x) \bmod m(x)$, $s'(x) = s(x) \bmod m(x)$, and $r'(x) = r(x) \bmod m(x)$. Because $\gcd(a(x), m(x)) = 1$, we have $r'(x) = (a'(x))^{-1}s'(x)$. Consequently, $r(x) = \phi^{-1}(r'(x)) = r'(x) \cdot d(x) \bmod (x^m + 1)$, where ϕ^{-1} is the inverse mapping from Eq (3.2). □

5. Array codes applied in the MSSS scheme

In this section, we first extend Generic Construction I to obtain a family of array codes specifically designed for an MSSS scheme. Based on this code family, we then propose a new MSSS scheme that supports a large number of participants. In this scheme, every share has self-reparability, which is enabled by the column local properties of the underlying codes.

Generic Construction II: Let $\mathbf{C}_2$ denote an $(n, k, d_A; m, g(x))_q$-array code with shape $m \times n$ ($n = \ell + r - 1$) whose parity-check matrix takes the block form

$$H_2 = \left(\begin{array}{c|ccc} H_1 & \mathbf{0}_m & \cdots & \mathbf{0}_m \\ & \mathbf{I}_m & \cdots & \mathbf{0}_m \\ & \vdots & \ddots & \vdots \\ & \mathbf{0}_m & \cdots & \mathbf{I}_m \end{array}\right)_{mr \times m(\ell+r-1)}, \tag{5.1}$$

where H_1 is the $r \times \ell$ parity-check matrix defined in (3.4). The additional $r - 1$ columns contain identity matrices that make the overall code systematic with respect to the parity symbols.

Definition 5.1. When we reconstruct a secret from $n - 1$ shares, if one can ensure that any k shares are sufficient for refactoring without $k - 1$ revealing any information, then it is said to be a perfect $(k, n - 1)$-TSS scheme [18].

Proposition 5.2. *(An MSSS scheme based on $\mathbf{C}_2$) Let $a_0(x) = 0$. Then, the parity-check matrix H_2 of $\mathbf{C}_2$ in Generic Construction II can be rewritten as*

$$H_2 = \begin{pmatrix} \mathbf{I}_m & \mathbf{I}_m & \cdots & \mathbf{I}_m & \mathbf{0}_m & \cdots & \mathbf{0}_m \\ \mathbf{0}_m & A_1 & \cdots & A_{\ell-1} & \mathbf{I}_m & \cdots & \mathbf{0}_m \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_m & A_1^{r-1} & \cdots & A_{\ell-1}^{r-1} & \mathbf{0}_m & \cdots & \mathbf{I}_m \end{pmatrix}_{mr \times m(\ell+r-1)}.$$

Let a codeword of $\mathbf{C}_2$ be written as $(\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\ell-1}, \boldsymbol{c}_\ell, \ldots, \boldsymbol{c}_{\ell+r-2})$. Designate $\boldsymbol{c}_1$ as the secret, assign the $\ell - 2$ vectors $\boldsymbol{c}_2, \boldsymbol{c}_3, \ldots, \boldsymbol{c}_{\ell-1}$ to distinct participants, and make the remaining r vectors $\boldsymbol{c}_0, \boldsymbol{c}_\ell, \ldots, \boldsymbol{c}_{\ell+r-2}$ publicly available. Then the scheme is a perfect $(\ell - r - 1, \ell - 2)$-TSS and every share has self-reparability.

Proof. Assume that $H_2 = (h_0, h_1, \cdots, h_{\ell-1}, h_\ell, \cdots, h_{\ell+r-2})$, where h_i is column i of H_2. By Theorem 3.9, we obtain that any $r \times r$ submatrix of $H_2' = (h_{i_0}, h_{i_1}, \cdots, h_{i_{r-1}})$ is invertible, where $i_j \subseteq [\ell]$. When

$a_0(x) = 0$, by Definition 5.1, it is easy to see that the proposition constructs a perfect $(\ell - r - 1, \ell - 2)$-TSS scheme from the array code $\mathbf{C}_2$.

Moreover, by the structure of H_2, we know that for $1 \le i \le \ell - 1$, $\boldsymbol{c}_i \in \langle (1 + x)g(x) \rangle$, which gives that every share has self-reparability, that is, it can recover $d_H - 1$ erasures or a burst of $1 + \deg(g(x))$ erasures, where d_H is the minimum Hamming distance of $\langle (1 + x)g(x) \rangle$. □

Obviously, with the increase of code length of the array code $\mathbf{C}_2$, the new MSSS scheme can support more participants, which can be applied in large-scale scenes such as blockchain sharding. Similar to Explicit Construction I, we can give the following construction of array codes.

Explicit Construction II: Let C_2 be the $(n, k, d_A; m, g(x))_q$-array code with shape $m \times (\ell + r - 1)$ defined by the parity-check matrix Eq (5.1), where the polynomials $a_0(x), a_1(x), \ldots, a_{\ell-1}(x)$ are generated according to the recurrence Eq (3.7).

Since $a_0(x) = 0$, Proposition 5.2 applies directly, yielding a perfect $(\ell - r - 1, \ell - 2)$-TSS scheme. By setting $\ell = 2^\kappa$, the number of participants reaches $2^\kappa - 2$, which is the exponent of the lowest degree κ in the irreducible factors of $m(x)$.

In practical deployments, participants must periodically refresh their shares to counter potential exposure. This motivates the need for an encoding procedure with low computational cost. Thanks to the recursive structure of the polynomials $a_i(x)$, the encoder for C_2 can be implemented using the scheduled algorithms (Algorithm 1 or Algorithm 2), resulting in complexity that scales favorably with the redundancy r.

We now illustrate the encoding process of the $(n, k, d_A; m, g(x))_q$-array code C_2 with a concrete example, showing how the public parities and the participants' shares are generated.

Example 5.3. Choose parameters $m = 5$, $r = 3$, $q = 2$, and $g(x) = 1$. According to Explicit Construction II, the $(7, 5, d_A; 5, 1)_2$-array code C_2 has shape 5×7 (since $n = \ell + r - 1 = 7$). The polynomials $a_i(x)$ for $i = 0, \ldots, 4$ are

$$a_0(x) = 0,\ a_1(x) = 1 + x,\ a_2(x) = x + x^2,\ a_3(x) = 1 + x^2,\ a_4(x) = x^2 + x^3 \pmod{x^5 + 1}.$$

The corresponding circulant matrices A_i are used to form H_1, and the full parity-check matrix H_2 is constructed as in (5.1) with two additional identity blocks.

Let a secret column $\boldsymbol{c}_1$ and three random information columns $\boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4$ be

$$\boldsymbol{c}_1 = (1, 0, 1, 0, 0)^\top, \quad \boldsymbol{c}_2 = (0, 1, 0, 1, 0)^\top, \quad \boldsymbol{c}_3 = (1, 1, 0, 0, 0)^\top, \quad \boldsymbol{c}_4 = (0, 0, 1, 1, 0)^\top.$$

(The last symbol of each column is chosen so that every column has even parity, that is, it belongs to $\langle 1 + x \rangle$, which is the local code.)

The encoder must compute the public column $\boldsymbol{c}_0$ and the two parity columns $\boldsymbol{c}_5, \boldsymbol{c}_6$ (corresponding to $\boldsymbol{c}_\ell$ and $\boldsymbol{c}_{\ell+1}$). These are determined by the linear constraints $H_2 \cdot (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4, \boldsymbol{c}_5, \boldsymbol{c}_6)^\top = \mathbf{0}$.

We perform the encoding using Algorithm 1. Set $L = 8$ and define the input vector as

$$\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_7) = (\mathbf{0}, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4, \mathbf{0}, \mathbf{0}, \mathbf{0}),$$

where $\mathbf{0}$ denotes the zero vector of length 5. The algorithm proceeds in three levels of recursion ($\delta = 3$ because $\lceil \log_2 5 \rceil = 3$):

Level 2 ($\omega = 2$):

$$\widetilde{\boldsymbol{x}_2} = \boldsymbol{x}_4 + \boldsymbol{x}_5 + \boldsymbol{x}_6 + \boldsymbol{x}_7 = \boldsymbol{c}_4 + \mathbf{0} = (0,0,1,1,0)^\top,$$

$$\boldsymbol{x}^{[0]} = (\boldsymbol{x}_0^{[0]}, \boldsymbol{x}_1^{[0]}, \boldsymbol{x}_2^{[0]}, \boldsymbol{x}_3^{[0]}) = (\boldsymbol{x}_0 + \boldsymbol{x}_4,\ \boldsymbol{x}_1 + \boldsymbol{x}_5,\ \boldsymbol{x}_2 + \boldsymbol{x}_6,\ \boldsymbol{x}_3 + \boldsymbol{x}_7) = (\boldsymbol{c}_4,\ \boldsymbol{c}_1,\ \boldsymbol{c}_2,\ \boldsymbol{c}_3).$$

Level 1 ($\omega = 1$):

$$\widetilde{\boldsymbol{x}_1} = \boldsymbol{x}_2^{[0]} + \boldsymbol{x}_3^{[0]} = \boldsymbol{c}_2 + \boldsymbol{c}_3 = (0,1,0,1,0)^\top + (1,1,0,0,0)^\top = (1,0,0,1,0)^\top,$$

$$\boldsymbol{x}^{[1]} = (\boldsymbol{x}_0^{[1]}, \boldsymbol{x}_1^{[1]}) = (\boldsymbol{x}_0^{[0]} + \boldsymbol{x}_2^{[0]},\ \boldsymbol{x}_1^{[0]} + \boldsymbol{x}_3^{[0]}) = (\boldsymbol{c}_4 + \boldsymbol{c}_2,\ \boldsymbol{c}_1 + \boldsymbol{c}_3).$$

Calculating these sums:

$$\boldsymbol{x}_0^{[1]} = (0,0,1,1,0)^\top + (0,1,0,1,0)^\top = (0,1,1,0,0)^\top,$$

$$\boldsymbol{x}_1^{[1]} = (1,0,1,0,0)^\top + (1,1,0,0,0)^\top = (0,1,1,0,0)^\top.$$

Level 0 ($\omega = 0$):

$$\widetilde{\boldsymbol{x}_0} = \boldsymbol{x}_1^{[1]} = (0,1,1,0,0)^\top,$$

$$\boldsymbol{x}^{[2]} = \boldsymbol{x}_0^{[1]} + \boldsymbol{x}_1^{[1]} = (0,1,1,0,0)^\top + (0,1,1,0,0)^\top = (0,0,0,0,0)^\top.$$

The public column $\boldsymbol{c}_0$ is exactly $\boldsymbol{x}^{[2]}$: $\boldsymbol{c}_0 = (0,0,0,0,0)^\top$.

Next, we compute the intermediate syndrome vectors $\widetilde{\boldsymbol{s}_1}$ and $\widetilde{\boldsymbol{s}_2}$ using the formula in Algorithm 1:

$$\widetilde{\boldsymbol{s}_1} = \widetilde{\boldsymbol{x}_0} + \mathcal{T}_d(\widetilde{\boldsymbol{x}_1}) + \mathcal{T}_d^2(\widetilde{\boldsymbol{x}_2}),$$

$$\widetilde{\boldsymbol{s}_2} = \widetilde{\boldsymbol{x}_0} + \mathcal{T}_d^2(\widetilde{\boldsymbol{x}_1}) + \mathcal{T}_d^4(\widetilde{\boldsymbol{x}_2}).$$

Recall that $\mathcal{T}_d$ is the downward cyclic shift: $\mathcal{T}_d((v_0, v_1, v_2, v_3, v_4)^\top) = (v_4, v_0, v_1, v_2, v_3)^\top$.

Applying these shifts:

Vector	Value	Shifted versions
$\widetilde{\boldsymbol{x}_0}$	$(0,1,1,0,0)^\top$	–
$\widetilde{\boldsymbol{x}_1}$	$(1,0,0,1,0)^\top$	$\mathcal{T}_d(\widetilde{\boldsymbol{x}_1}) = (0,1,0,0,1)^\top,\ \mathcal{T}_d^2(\widetilde{\boldsymbol{x}_1}) = (1,0,1,0,0)^\top$
$\widetilde{\boldsymbol{x}_2}$	$(0,0,1,1,0)^\top$	$\mathcal{T}_d^2(\widetilde{\boldsymbol{x}_2}) = (1,0,0,0,1)^\top,\ \mathcal{T}_d^4(\widetilde{\boldsymbol{x}_2}) = (0,1,1,0,0)^\top$

Then,

$$\widetilde{\boldsymbol{s}_1} = (0,1,1,0,0)^\top + (0,1,0,0,1)^\top + (1,0,1,0,0)^\top = (1,0,0,0,1)^\top,$$

$$\widetilde{\boldsymbol{s}_2} = (0,1,1,0,0)^\top + (1,0,0,0,1)^\top + (0,1,1,0,0)^\top = (1,0,0,0,1)^\top.$$

Finally, the parity columns are given by

$$\boldsymbol{p}_1 = \boldsymbol{c}_5 = \widetilde{\boldsymbol{s}_1} + \mathcal{T}_d(\widetilde{\boldsymbol{s}_1}), \qquad \boldsymbol{p}_2 = \boldsymbol{c}_6 = \widetilde{\boldsymbol{s}_2} + \mathcal{T}_d^2(\widetilde{\boldsymbol{s}_2}).$$

Computing these:

$$\mathcal{T}_d((1,0,0,0,1))^\top = (1,1,0,0,0)^\top \implies \boldsymbol{c}_5 = (0,1,0,0,1)^\top,$$

$$\mathcal{T}_d^2((1,0,0,0,1))^\top = (0,1,1,0,0)^\top \implies \boldsymbol{c}_6 = (1,1,1,0,1)^\top.$$

Thus, the complete codeword is

$$\boldsymbol{c}_0 = (0,0,0,0,0)^\top, \quad \boldsymbol{c}_1 = (1,0,1,0,0)^\top, \quad \boldsymbol{c}_2 = (0,1,0,1,0)^\top, \quad \boldsymbol{c}_3 = (1,1,0,0,0)^\top,$$
$$\boldsymbol{c}_4 = (0,0,1,1,0)^\top, \quad \boldsymbol{c}_5 = (0,1,0,0,1)^\top, \quad \boldsymbol{c}_6 = (1,1,1,0,1)^\top.$$

One can verify that $H_2 \cdot (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4, \boldsymbol{c}_5, \boldsymbol{c}_6)^\top = \mathbf{0}$ over $\mathbb{R}_5$.

According to Proposition 5.2, $\boldsymbol{c}_1$ serves as the secret, and $\boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4$ are distributed to three participants, and $\boldsymbol{c}_0, \boldsymbol{c}_5, \boldsymbol{c}_6$ are made public. The scheme constitutes a perfect $(1,3)$-TSS because $\ell - r - 1 = 1$, which implies that any single share, together with the public columns, can recover the secret. Moreover, each share is self-repairable: For instance, if one symbol in $\boldsymbol{c}_2$ is lost, it can be reconstructed using the even-parity constraint within that column.

Remark 5.4. In the encoding procedure of Example 5.3, the local parity symbols for the information columns are computed using $4 \times 4 = 16$ XORs (one per column to enforce even parity). The recursive steps involve five vector additions at level 2, four at level 1, and two at level 0, together with the necessary shifts and final combinations. With a sub-packetization of $m = 5$, the total number of XOR operations is approximately 85. For larger ℓ and small r, the encoding complexity scales as $O(m\ell \log r)$, which is significantly lower than that of traditional BR-based or GRDP-based MSSS schemes. A detailed complexity comparison is provided in Table 1.

For the convenience of calculating cyclic codes, only the simplest case, MSSS in C_2, is considered. According to Algorithm 2 and Remark 4.4, for general redundancy r, the encoding of the $(n, k, d_A; m, 1)_2$-array code C_2 requires about $XORs = km(\lfloor \log_2 r \rfloor + 1) + k(m-2)$, where $k = \ell - r - 1$. Some numbers are given in Table 1, in which the encoding complexity of our scheme is significantly lower than other schemes in both high code rate ($m < 2k$) and low code rate ($m > 2k$) scenarios.

Table 1. Number of XORs at encoding for the three schemes, all cases corresponding to a perfect $(k, m-1)$-TSS and same subpacketization $m - 1$.

m	k	$(k, m-1)$-TSS	Ours	MSSS GRDP [20]	MSSS BR [20]
23	15	(15,22)	1695	3696	3843
31	15	(15,30)	2760	13920	14355
71	50	(50,70)	21200	101430	102810
83	45	(45,82)	26055	252396	255393
89	40	(40,88)	24840	375144	379320
127	100	(100,126)	76000	425250	428500

The encoding complexity of BR and GRDP codes is asymptotically r XORs per data bit, where r denotes the redundancy. Thus, as the redundancy r grows, our C_2-based MSSS scheme distributes secrets faster than BR/GRDP-based schemes. Moreover, whereas BR/GRDP schemes require m to be prime, ours only requires m odd. For fixed m, BR/GRDP schemes support at most $m - 1$ participants, whereas our scheme can support up to $2^\kappa - 2$ participants, an exponential improvement relative to m. Additionally, among the three schemes, only each share in our scheme is locally repairable (self-recoverable), which can effectively safeguard the secret against accidental partial data loss.

6. Conclusions and future work

In this paper, we systematically studied the construction, decoding, and secret-sharing application of q-ary array codes with local locality. First, we proposed a generic construction (Construction I) over $\mathbb{R}_m$, where a GCD-Constraint ensures recoverability. An explicit construction was also given, allowing the code length to grow exponentially with the subpacketization parameter. Second, we designed a low-complexity erasure decoding algorithm for the explicit construction, comprising recursive-branching syndrome calculation and an efficient method for solving Vandermonde equations over $\mathbb{R}_m$ using the Chinese remainder theorem. Third, we embedded the array codes into a modified Shamir secret sharing scheme (Construction II). This new MSSS scheme offers three key advantages: It supports an exponential number of participants (from $O(m)$ to $O(2^\kappa)$), every share is locally repairable, and the encoding complexity is significantly lower than that of BR/GRDP-based schemes.

Finally, we note that the quantitative comparison of repair bandwidth with general LRACs is an important future direction. Although our codes possess column local properties that confine repair within a single column, LRACs allow repair groups to span multiple columns, which incurs different system assumptions. In future work, we plan to investigate the single-node repair bandwidth for array codes derived from our construction, which would bridge the gap between column local repair and LRACs.

Author contributions

Xian Lian: Writing – original draft, formal analysis, validation; Jingjie Lv: Conceptualization, methodology, investigation; Hongwei Zhu: Conceptualization, data curation; Shu-Tao Xia: Resources, validation, funding acquisition; Hanxu Hou: Formal analysis, funding acquisition, supervision. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This research is supported in part by the National Key Research and Development Program of China under Grant No. 2025YFA1017200, the China Postdoctoral Science Foundation No. 2025M783088, the National Natural Science Foundation of China under Grant No. 62401144, the Natural Science Foundation of Guangdong Province under Grant No. 2026A1515012796, and the National Natural Science Foundation of China under Grants Nos. 62171248 and 62571298. This paper has been partly published in ISIT 2026 [26].

Conflict of interest

The authors declare no conflict of interest.

References

1. D. A. Patterson, P. Chen, G. Gibson, R. H. Katz, Introduction to redundant arrays of inexpensive disks (RAID), In: *Thirty-fourth IEEE computer society international conference: Intellectual leverage*, San Francisco: IEEE, 1989, 112–117. https://doi.org/10.1109/CMPCON.1989.301912
2. M. Blaum, P. G. Farrell, H. C. A. van Tilborg, Chapter on array codes, In: *Handbook of coding theory*, Elsevier, **2** (1998), 1855–1909.
3. H. Hou, P. P. C. Lee, A new construction of EVENODD codes with lower computational complexity, *IEEE Commun. Lett.*, **22** (2018), 1120–1123. https://doi.org/10.1109/LCOMM.2018.2820007
4. P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, et al., Row-diagonal parity for double disk failure correction, In: *Proceedings of the 3rd USENIX conference on file and storage technologies*, 2004.
5. G. L. Feng, R. H. Deng, F. Bao, J. C. Shen, New efficient MDS array codes for RAID. Part I: Reed-Solomon-like codes for tolerating three disk failures, *IEEE Trans. Comput.*, **54** (2005), 1071–1080. https://doi.org/10.1109/TC.2005.150
6. C. Huang, L. Xu, STAR: An efficient coding scheme for correcting triple storage node failures, *IEEE Trans. Comput.*, **57** (2008), 889–901. https://doi.org/10.1109/TC.2007.70830
7. M. Blaum, A family of MDS array codes with minimal number of encoding operations, In: *2006 IEEE international symposium on information theory (ISIT)*, Seattle: IEEE, 2006, 2784–2788. https://doi.org/10.1109/ISIT.2006.261569
8. M. Blaum, J. Bruck, A. Vardy, MDS array codes with independent parity symbols, *IEEE Trans. Inf. Theory.*, **42** (1996), 529–542. https://doi.org/10.1109/18.485722
9. M. Blaum, R. M. Roth, New array codes for multiple phased burst correction, *IEEE Trans. Inf. Theory*, **39** (1993), 66–77. https://doi.org/10.1109/18.179343
10. J. Lv, W. Fang, X. Chen, J. Yang, S. T. Xia, New constructions of q-ary MDS array codes with multiple parities and their effective decoding, *IEEE Trans. Inf. Theory*, **69** (2023), 7082–7096. https://doi.org/10.1109/TIT.2023.3300919
11. W. Fang, J. Lv, B. Chen, S. T. Xia, X. Chen, New constructions of MDS array codes and optimal locally repairable array codes, *IEEE Trans. Inf. Theory*, **70** (2024), 1806–1822. https://doi.org/10.1109/TIT.2024.3353111
12. Z. Zhai, S. Jin, Q. T. Sun, S. Liu, X. Chen, Z. Li, New construction of MDS array codes and explicit characterization of decoding matrices, *IEEE Trans. Commun.*, **73** (2025), 5592–5606. https://doi.org/10.1109/TCOMM.2025.3541085
13. M. Blaum, S. R. Hetzler, Array codes with local properties, *IEEE Trans. Inf. Theory*, **66** (2020), 3675–3690. https://doi.org/10.1109/TIT.2019.2951693
14. H. Hou, Y. S. Han, P. P. C. Lee, Y. Wu, G. Han, M. Blaum, A generalization of array codes with local properties and efficient encoding/decoding, *IEEE Trans. Inf. Theory*, **69** (2023), 107–125. https://doi.org/10.1109/TIT.2022.3202140

15. Y. Tian, F. W. Fu, A fast algorithm of Syndrome computations for binary optimal locally repairable array codes, *IEEE Trans. Commun.*, **73** (2025), 13117–13129. https://doi.org/10.1109/TCOMM.2025.3615764

16. S. J. Lin, An encoding algorithm of triply extended Reed–Solomon codes with asymptotically optimal complexities, *IEEE Trans. Commun.*, **66** (2018), 3235–3244. https://doi.org/10.1109/TCOMM.2017.2737441

17. L. Yu, S. J. Lin, H. Hou, Z. Li, Reed-Solomon coding algorithms based on Reed-Muller transform for any number of parities, *IEEE Trans. Comput.*, **72** (2023), 2677–2688. https://doi.org/10.1109/TC.2023.3262922 .

18. J. Kurihara, S. Kiyomoto, K. Fukushima, T. Tanaka, A new (k, n)-threshold secret sharing scheme and its extension, In: *Information security. ISC 2008*, Berlin: Springer, 2008, 455–470. https://doi.org/10.1007/978-3-540-85886-7_31

19. A. Shamir, How to share a secret, *Commun. ACM*, **22** (1979), 612–613. https://doi.org/10.1145/359168.359176

20. A. Hineman, M. Blaum, A modified Shamir secret sharing scheme with efficient encoding, *IEEE Commun. Lett.*, **26** (2022), 758–762. https://doi.org/10.1109/LCOMM.2022.3144375

21. Y. Wang, Y. Desmedt, Efficient secret sharing schemes achieving optimal information rate, In: *2014 IEEE information theory workshop (ITW 2014)*, Hobart: IEEE, 2014, 516–520. https://doi.org/10.1109/ITW.2014.6970885

22. L. Chen, T. M. Laing, K. M. Martin, Efficient, XOR-based, ideal (t, n)-threshold schemes, In: *Cryptology and network security. CANS 2016.*, Cham: Springer, 2016, 467–483. https://doi.org/10.1007/978-3-319-48965-0_28

23. F. J. MacWilliams, N. J. A. Sloane, The theory of error-correcting codes, In: *North-Holland mathematical library*, Elsevier, **16** (1977), 1–762.

24. J. Lv, H. Zhu, W. Fang, H. Hou, S. T. Xia, New constructions of q-ary MDS array codes derived from $\mathbb{F}_q[x]/\langle x^m + \lambda\rangle$ and their efficient erasure encoding/decoding, *IEEE Trans. Inf. Theory*, **71** (2025), 8429–8446. https://doi.org/10.1109/TIT.2025.3610497

25. C. Ding, D. Pei, A. Salomaa, *Chinese remainder theorem: Applications in computing, coding, cryptography*, United States: World Scientific Publishing Co., Inc., 1996.

26. X. Lian, J. Lv, H. Zhu, S. Xia, H. Hou, Array codes with local properties and their application to Shamir secret sharing scheme, In: *IEEE International symposium on information theory (ISIT)*, Guangzhou: IEEE, 2026, in press.

27. S. L. Yang, On the LU factorization of the Vandermonde matrix, *Discrete Appl. Math.*, **146** (2005), 102–105. https://doi.org/10.1016/j.dam.2004.08.003

AIMS Press