Novel Polynomial Basis with Fast Fourier Transform and Its Application to Reed-Solomon Erasure Codes

Sian-Jheng Lin, *Member, IEEE,* Tareq Y. Al-Naffouri, *Member, IEEE,* Yunghsiang S. Han, *Fellow, IEEE,* and Wei-Ho Chung, *Member, IEEE*

***Abstract*—In this paper, we present a fast Fourier transform (FFT) algorithm over extension binary fields, where the polynomial is represented in a non-standard basis. The proposed Fourier-like transform requires $\mathcal{O}(h \lg(h))$ field operations, where h is the number of evaluation points. Based on the proposed Fourier-like algorithm, we then develop the encoding/decoding algorithms for $(n = 2^m, k)$ Reed-Solomon erasure codes. The proposed encoding/erasure decoding algorithm requires $\mathcal{O}(n \lg(n))$, in both additive and multiplicative complexities. As the complexity leading factor is small, the proposed algorithms are advantageous in practical applications. Finally, the approaches to convert the basis between the monomial basis and the new basis are proposed.**

***Index Terms*—Fast Fourier transform, polynomial basis, finite field, Reed-Solomon code.**

I. Introduction

LET $\mathbb{F}_q$, $q = p^m$, denote an extension finite field of dimension m over $\mathbb{F}_p$. A polynomial $a(x) \in \mathbb{F}_q[x]$ of degree less than $h < q$ in the monomial basis is written by

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{h-1} x^{h-1},$$

with each $a_i \in \mathbb{F}_q$. Given a set of evaluation points $E = \{e_i\}_{i=0}^{h-1}, \forall e_i \in \mathbb{F}_q$, the multipoint polynomial evaluation is the task of evaluating $a(x)$ at E. A primitive algorithm requires $\mathcal{O}(h^2)$ field operations of $\mathbb{F}_q$. However, the task can be completed faster if we carefully choose the set of evaluation points. Assume h is a divisor of $q-1$. The discrete Fourier transform (DFT) is the algorithm of evaluating $a(x)$ at $E = \{\omega^i\}_{i=0}^{h-1}$, where ω is the h-th root of unity. We refer to this class of DFTs as multiplicative DFT, as those evaluation points form a multiplicative group. Fast Fourier transforms (FFT) are the algorithms for performing DFT with lower arithmetic complexities.

FFT over finite fields is a traditional algebra problem. Specifically, a significant application of FFTs over finite fields is the coding algorithms of algebraic codes such as Reed-Solomon (RS) codes. As the codes are usually constructed over extension binary fields $\mathbb{F}_{2^m}, m \in \mathbb{N}$, FFTs over extension binary fields naturally receive higher attentions than over other fields. In 1971, Pollard [1] showed that if $q-1$ is a smooth number (the number that can be factored into small primes), there exists an FFT algorithm with the complexity $\mathcal{O}(h \lg(h))$. However, as $2^m - 1$ usually cannot be factorized into the product of small primes, this approach is inapplicable for fileds $\mathbb{F}_{2^m}$. Currently, the asymptotically fastest approach is based on Bluestein's FFT [2] (or Rader's FFT), and the convolution in the algorithm is computed by Schönhage's polynomial multiplication [3]. This requires $\mathcal{O}(h \lg(h) \lg\lg(h))$ with huge leading constant.

If the set of evaluation points E forms an additive group, the transform is termed as additive DFT. Additive FFT over fields was firstly invented by Wang and Zhu [4] in 1988. Later, the faster approaches were proposed by [5] and [6]. Currently, the asymptotically fastest approach is proposed by Gao and Mateer [7]. They gave an $\mathcal{O}(h \lg^2(h))$ approach for arbitrary m, as well as an $\mathcal{O}(h \lg(h) \lg\lg(h))$ approach for m a power of two.

The traditional definition of FFTs, as well as most polynomial arithmetic, presumes that the input polynomials are written in the monomial basis. An important property of the monomial basis is the total order in degrees. In particular, for a polynomial basis $\mathbf{G}(x) = \{g_0(x), g_1(x), \ldots, g_{n-1}(x)\}$ ordered by degrees, $deg(g_i(x)) \leq \deg(g_{i+1}(x))$, we define that $\mathbf{G}(x)$ is with full degree when $\deg(g_i(x)) = i$. This basis possesses a property that, for a polynomial

$$b(x) = \sum_{i=0}^{n-1} b_i g_i(x),$$

we have $b_i = 0$ for $i > \deg(b(x))$. Thus, the degree of $b(x)$ can be determined by using $\mathcal{O}(n)$.

In this paper, a full-degree basis is introduced for additive DFTs. In the first part of this paper, an $\mathcal{O}(h \lg(h))$ additive Fourier-like transform is proposed, where the basis to represent the input polynomial is nonstandard. The existing works on

A preliminary version of this work has been presented at FOCS'14. This work was supported in part by the CAS Pioneer Hundred Talents Program, the National Science of Council (NSC) of Taiwan under Grants NSC 102-2221-E-011-006-MY3, the Ministry of Science and Technology under Grant MOST 105-2221-E-001-009-MY3, and the Academia Sinica Thematic Project under Grant AS-104-TP-A05. Sian-Jheng Lin is with CAS Key Laboratory of Electro-magnetic Space Information, the School of Information Science and Technology, University of Science and Technology of China (USTC), China.(e-mail: sjlin@ustc.edu.cn); Tareq Al-Naffouri is with the Computer, Electrical, Mathematical Sciences and Engineering (CEMSE) Division at King Abdullah University of Science and Technology (KAUST), Saudi Arabia. (e-mail: tareq.alnaffouri@kaust.edu.sa); Yunghsiang S. Han is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taiwan. (e-mail: yshan@mail.ntust.edu.tw); Wei-Ho Chung is with the Research Center for Information Technology Innovation (CITI), Academia Sinica, Taiwan. (e-mail: whc@iis.sincia.edu.tw)

FFTs over extension binary fields are tabulated in Table II. To show the applicability of the new basis, the second part of this paper applies the new basis to the (n, k) RS erasure codes over $\mathbb{F}_{2^m}$, resulting in an $\mathcal{O}(n \lg(k))$ encoding algorithm, and an $\mathcal{O}(n \lg(n))$ erasure decoding algorithm.

In the final part of this paper, the basis conversion algorithms for the polynomials are proposed. For arbitrary m, two approaches are devised. Both approaches use $\mathcal{O}(n \lg^2(n))$ field operations. For m a power of two, a faster approach requiring $\mathcal{O}(n \lg(n) \lg\lg(n))$ is presented. Further, we also generalize the new basis to over $\mathbb{F}_{p^m}$. With the new basis, the complexity of the proposed Fourier-like transform is $\mathcal{O}(n \cdot p \log_p(n))$. By letting p a constant, the complexity can be written as $\mathcal{O}(n \log(n))$. Also Notice that the new basis can also be used to improve the complexities of RS error decoding algorithms [8].

The rest of this paper is organized as follows. Section II gives the definition of the new polynomial basis. In Section III, a Fourier-like algorithm is proposed based on the new basis. Section IV presents the fast approach to perform the formal derivative in the new basis. Based on above results, Section V presents the encoding and erasure decoding for Reed-Solomon codes. The discussions are placed in Section VI. SectionVII reviews some related literature. The basis conversion algorithms are presented in Section VIII. Section IX concludes this work. The generalization of the new basis over $\mathbb{F}_{p^m}$ is addressed in Appendix.

II. Polynomial basis

This section introduces a new polynomial basis. Section II-A reviews the definition of subspace polynomials and the polynomial basis is introduced in Section II-B.

A. Subspace polynomial

Let $\mathbb{F}_{p^m}$ denote an binary extension field. Let

$$\boldsymbol{v}_m = (v_0, v_1, \ldots, v_{m-1}) \tag{1}$$

denote a basis of $\mathbb{F}_{p^m}$, whereas all $v_i \in \mathbb{F}_{p^m}$ are linearly independent over $\mathbb{F}_p$. Let $\{\omega_i\}_{i=0}^{p^m-1}$ denote the set of elements of $\mathbb{F}_{p^m}$. Each ω_i is defined as

$$\omega_i = i_0 v_0 + i_1 v_1 + \cdots + i_{m-1} v_{m-1}, \tag{2}$$

where $i_j \in \mathbb{F}_p$, and

$$i = i_0 + i_1 \cdot p + i_2 \cdot p^2 + \cdots + i_{m-1} \cdot p^{m-1}.$$

Let

$$V_k = \mathrm{Span}(v_0, v_1, \ldots, v_{k-1}) = \{\sum_{j=0}^{k-1} a_j \cdot v_j | a_j \in \mathbb{F}_p\} \tag{3}$$

denote a k-dimensional subspace in $\mathbb{F}_{p^m}$, where $k \leq m$. These V_k form a strictly ascending chain as

$$\{0\} = V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m = \mathbb{F}_{p^m}.$$

The subspace polynomial [5, 7] is defined as

Definition 1 (Subspace polynomial [5, 7])**.** *Given a subspace V_k of $\mathbb{F}_{p^m}$, the subspace polynomial is defined as*

$$s_k^{V_k}(x) = \prod_{w \in V_k} (x - w). \tag{4}$$

Clearly, $\deg(s_k^{V_k}(x)) = p^k$.

The subspace polynomial depends on the chosen subspace of the field. If not specified, throughout this paper, $s_k(x) = s_k^{V_k}(x)$ denotes the subspace polynomial to the subspace V_k defined in (3).

The properties of subspace polynomials are given as follows.

Lemma 1 ([5, 6])**.** *The subspace polynomial can be written as a recursive form:*

$$\begin{aligned} &s_0(x) = x; \\ &s_j(x) = \prod_{i \in \mathbb{F}_p} s_{j-1}(x - i \cdot v_{j-1}) \qquad j = 1, 2, \ldots, m. \end{aligned} \tag{5}$$

Lemma 2 ([6, 7, 9, 10])**.** *$s_k(x)$ is an $\mathbb{F}_p$-linearlized polynomial for which*
(1). $s_k(x)$ is a sparse polynomial with no more than $k+1$ non-zero terms. That is,

$$s_k(x) = \sum_{i=0}^{k} s_{k,i} x^{p^i}, \tag{6}$$

where $s_{k,i} \in \mathbb{F}_{p^m}$ for $0 \leq i \leq k$.
(2).

$$s_k(x + y) = s_k(x) + s_k(y), \qquad \forall x, y \in \mathbb{F}_{p^m}. \tag{7}$$

(3). Given a basis $\boldsymbol{v}_j = (v_0, v_1, \ldots, v_{j-1})$ of a subspace

$$V_j = \mathrm{Span}(\boldsymbol{v}_j),$$

a subspace W_{j-1} is defined as

$$W_{j-1} = \mathrm{Span}(\boldsymbol{w}_{j-1}),$$

where

$$\boldsymbol{w}_{j-1} = (s_1^{V_1}(v_1), s_1^{V_1}(v_2), \ldots, s_1^{V_1}(v_{j-1})),$$

and

$$s_1^{V_1}(x) = x(x - v_0).$$

Then, the subspace polynomials for V_j and W_{j-1} satisfy

$$s_j^{V_j}(x) = s_{j-1}^{W_{j-1}}(s_1^{V_1}(x)). \tag{8}$$

B. Polynomial basis

In this subsection we only consider the case $p = 2$, and the general case for arbitrary prime p is given in Appendix D. Based on the subspace polynomials, a polynomial basis is defined as

$$\mathbb{X}^{\boldsymbol{v}_m} = \left(X_0^{\boldsymbol{v}_m}(x), X_1^{\boldsymbol{v}_m}(x), \ldots, X_{2^m-1}^{\boldsymbol{v}_m}(x)\right)$$

in $\mathbb{F}_{2^m}[x]/(x^{2^m} - x)$. For simplicity, we continually use $s_j(x)$ to indicate the subspace polynomials in $\mathbb{F}_{2^m}[x]/(x^{2^m} - x)$. Let

$$\bar{s}_j(x) = \frac{s_j(x)}{s_j(v_j)},$$

where v_j is defined in (1), and hence

$$\bar{s}_j(v_j) = 1. \tag{9}$$

$X_i^{\boldsymbol{v}_m}(x)$ is defined as

$$X_i^{\boldsymbol{v}_m}(x) = \prod_{j=0}^{m-1} (\bar{s}_j(x))^{i_j}, \tag{10}$$

where $i = i_0 + i_1 \cdot 2 + i_2 \cdot 2^2 + \cdots + i_{m-1} \cdot 2^{m-1}$, for $i_j \in \{0, 1\}$. Notice that $(\bar{s}_j(x))^0 = 1$. It can be seen that this basis is full degree $\deg(X_i(x)) = i$, and thus it can determine all polynomials of $\mathbb{F}_{2^m}[x]/(x^{2^m} - x)$. If not specified, throughout this paper,

$$\mathbb{X} = (X_0(x), X_1(x), \ldots, X_{2^m-1}(x))$$

denotes the polynomial basis with respective to the basis $\boldsymbol{v}$ defined in (1). Note that X_i can be treated as the binary representation of i with basis $\bar{s}_j(x)$, $j = 0, 1, \ldots, m-1$.

A polynomial in $\mathbb{X}$ is written by

$$D(x) = \sum_{i=0}^{2^m-1} d_i X_i(x),$$

with each $d_i \in \mathbb{F}_{2^m}$. In this paper, the coefficients of $D(x)$ is denoted as a vector $D = (d_0, d_1, \ldots, d_{2^m-1})$.

III. Transform in the new basis

Given a polynomial $D(x) = \sum_{i=0}^{2^k-1} d_i X_i(x)$ of degree $\deg(D(x)) < 2^k = h$ in the basis $\mathbb{X}$ and $\beta \in \mathbb{F}_{2^m}$, this section presents a algorithm to compute $\{D(\omega) | \omega \in V_k + \beta\}$.

A. Recursive structure in polynomial basis

The polynomial $D(x)$ can be formulated as a recursive function. Let

$$D(x) = \Delta_0^0(x).$$

The recursive function is defined as

$$\Delta_i^r(x) = \Delta_{i+1}^r(x) + \bar{s}_i(x)\Delta_{i+1}^{r+2^i}(x), \text{ for } 0 \le i \le k-1; \tag{11}$$

$$\Delta_k^r(x) = d_r, \text{ for } 0 \le r \le 2^k - 1. \tag{12}$$

By induction, the coefficients of $\Delta_i^r(x)$ are denoted as

$$\Delta_i^r = \{d_{j \cdot 2^i + r} | j = 0, \ldots, 2^{k-i} - 1)\}. \tag{13}$$

Lemma 3. *From* (11) *and* (12)*, we have* $D(x) = \Delta_0^0(x)$.

Proof. Assume

$$\Delta_i^r(x) = \sum_{j=0}^{2^{k-i}-1} d_{j \cdot 2^i + r} X_{j \cdot 2^i}(x), \tag{14}$$

for $0 \le i \le k-1$ and $0 \le r \le 2^i - 1$. From (14), it can be verified that $D(x) = \Delta_0^0(x)$, and (13) is correct. In the following, the validity of (14) is proved by mathematical inductions with decreasing index. For the basis case, we consider $i = k$ in (14), that gives

$$\Delta_k^r(x) = \sum_{j=0}^{0} d_{j \cdot 2^k + r} X_{j \cdot 2^k}(x) = d_r X_0(x) = d_r, \tag{15}$$

and thus (12) holds.

Assume (14) is valid for $i = \ell + 1$. When $i = \ell$, (11) can be written as

$$\begin{aligned}
&\Delta_\ell^r(x) \\
=&\Delta_{\ell+1}^r(x) + \bar{s}_\ell(x)\Delta_{\ell+1}^{r+2^\ell}(x) \\
=& \sum_{j=0}^{2^{k-\ell-1}-1} d_{j \cdot 2^{\ell+1} + r} X_{j \cdot 2^{\ell+1}}(x) \\
&+ \bar{s}_\ell(x) \sum_{j=0}^{2^{k-\ell-1}-1} d_{j \cdot 2^{\ell+1} + 2^\ell + r} X_{j \cdot 2^{\ell+1}}(x) \\
=& \sum_{j=0}^{2^{k-\ell-1}-1} d_{(2j) \cdot 2^\ell + r} X_{(2j) \cdot 2^\ell}(x) \\
&+ \bar{s}_\ell(x) \sum_{j=0}^{2^{k-\ell-1}-1} d_{(2j+1) \cdot 2^\ell + r} X_{(2j) \cdot 2^\ell}(x) \\
=& \sum_{j=0}^{2^{k-\ell-1}-1} d_{(2j) \cdot 2^\ell + r} X_{(2j) \cdot 2^\ell}(x) \\
&+ \sum_{j=0}^{2^{k-\ell-1}-1} d_{(2j+1) \cdot 2^\ell + r} X_{(2j+1) \cdot 2^\ell}(x) \\
=& \sum_{j=0}^{2^{k-\ell}-1} d_{j \cdot 2^\ell + r} X_{j \cdot 2^\ell}(x),
\end{aligned} \tag{16}$$

and thus (14) is valid for $i = \ell$. This completes the proof. □

As mentioned previously, X_i can be treated as the binary representation of i with basis $\bar{s}_j(x)$, $j = 0, 1, \ldots, m-1$. The idea behind the recursion is that first combining the terms with only difference in $\bar{s}_1(x)$ and then combing the resultant terms with only difference in $\bar{s}_2(x)$, and so on. In the following, we demonstrate this idea by an example. For example, if $h = 8$, we have

$$\begin{aligned}
&D(x) = \sum_{i=0}^{7} d_i X_i(x) \\
=&d_0 + d_1\bar{s}_0(x) + d_2\bar{s}_1(x) + d_3\bar{s}_0(x)\bar{s}_1(x) + d_4\bar{s}_2(x) \\
&+ d_5\bar{s}_0(x)\bar{s}_2(x) + d_6\bar{s}_1(x)\bar{s}_2(x) + d_7\bar{s}_0(x)\bar{s}_1(x)\bar{s}_2(x) \\
=&\left(d_0 + d_4\bar{s}_2(x) + \bar{s}_1(x)\left(d_2 + d_6\bar{s}_2(x)\right)\right) \\
&+ \bar{s}_0(x)\left(d_1 + d_5\bar{s}_2(x) + \bar{s}_1(x)\left(d_3 + d_7\bar{s}_2(x)\right)\right) \\
=&\left(\Delta_2^0(x) + \bar{s}_1(x)\Delta_2^2(x)\right) + \bar{s}_0(x)\left(\Delta_2^1(x) + \bar{s}_1(x)\Delta_2^3(x)\right) \\
=&\Delta_1^0(x) + \bar{s}_0(x)\Delta_1^1(x) = \Delta_0^0(x).
\end{aligned} \tag{17}$$

$\Delta_i^m(x)$ possesses the following equality that will be utilized in the algorithm:

Lemma 4. $\forall a \in V_m = \mathbb{F}_{p^m}$ *and* $\forall b \in V_i$, $0 \le i \le k-1$,

$$\Delta_i^m(a+b) = \Delta_i^m(a). \tag{18}$$

Proof. By definition, $\bar{s}_i(b) = 0$, $\forall b \in V_i$. From Lemma 2, we have

$$\bar{s}_i(a+b) = \bar{s}_i(a) + \bar{s}_i(b) = \bar{s}_i(a), \qquad \forall b \in V_i. \tag{19}$$

The proof is based on the mathematical induction on i. In the base case $i = k-1$, (11) can be written as

$$\begin{aligned}\Delta_{k-1}^{m}(x) =&\Delta_{k}^{m}(x) + \bar{s}_{k-1}(x)\Delta_{k}^{m+2^{k-1}}(x)\\ =&d_m + \bar{s}_{k-1}(x)d_{m+2^{k-1}}.\end{aligned} \tag{20}$$

From (19), we have

$$\begin{aligned}&\Delta_{k-1}^{m}(a+b)\\ =&d_m + \bar{s}_{k-1}(a+b)d_{m+2^{k-1}}\\ =&d_m + \bar{s}_{k-1}(a)d_{m+2^{k-1}}\\ =&\Delta_{k-1}^{m}(a), \qquad \forall b \in V_{k-1}.\end{aligned} \tag{21}$$

Thus (18) holds for $i = k-1$.

Assume (18) holds for $i = j+1$. When $i = j$, we have

$$\begin{aligned}&\Delta_{j}^{m}(a+b)\\ =&\Delta_{j+1}^{m}(a+b) + \bar{s}_{j}(a+b)\Delta_{j+1}^{m+2^{j}}(a+b)\\ =&\Delta_{j+1}^{m}(a) + \bar{s}_{j}(a)\Delta_{j+1}^{m+2^{j}}(a)\\ =&\Delta_{j}^{m}(a), \qquad \forall b \in V_{j}.\end{aligned} \tag{22}$$

This completes the proof. □

B. Proposed algorithm

The proposed FFT algorithm is similar to that for complex fields. The algorithm is with a divide-and-conquer fashion. Hence, we need to determine the recursive equation for each iteration. Let

$$V_j^k = \mathrm{Span}(v_j, v_{j+1}, \ldots, v_{k-1}), \quad 0 \le j \le k \le m-1. \tag{23}$$

denote an $(k-j)$-dimensional subspace in $\mathbb{F}_{2^m}$. These subspaces form a strictly ascending chain as

$$\{0\} = V_k^k \subset V_{k-1}^k \subset V_{k-2}^k \subset \cdots \subset V_0^k.$$

Let

$$\Psi_\beta(i,r) = \{\Delta_i^r(\omega) | \omega \in V_i^k + \beta\}, \qquad i = 0, \ldots, k-1; \tag{24}$$

$$\Psi_\beta(k,r) = \{d_r\}. \tag{25}$$

The objective of algorithm is to compute the values in set $\Psi_\beta(0,0)$. In the following, we rearrange $\Psi_\beta(i,r)$ into two parts: $\Psi_\beta(i+1,r)$ and $\Psi_\beta(i+1,r+2^i)$, by taking at most 2^{k-i} additions and 2^{k-i-1} multiplications.

As

$$V_i^k = V_{i+1}^k \cup (V_{i+1}^k + v_i),$$

(24) can be divided into two individual subsets

$$\Psi_\beta^{(0)}(i,r) = \{\Delta_i^r(\omega) | \omega \in V_{i+1}^k + \beta\}, \tag{26}$$

and

$$\Psi_\beta^{(1)}(i,r) = \{\Delta_i^r(\omega + v_i) | \omega \in V_{i+1}^k + \beta\}. \tag{27}$$

To evaluate each element of $\Psi_\beta^{(0)}(i,r)$, by recursive function given in (11), we have

$$\Delta_i^r(\omega) = \Delta_{i+1}^r(\omega) + \bar{s}_i(\omega)\Delta_{i+1}^{r+2^i}(\omega) \quad \forall \omega \in V_{i+1}^k + \beta. \tag{28}$$

It can be seen that $\Delta_{i+1}^r(\omega) \in \Psi_\beta(i+1,r)$, and $\Delta_{i+1}^{r+2^i}(\omega) \in \Psi_\beta(i+1,r+2^i)$, for $\omega \in V_{i+1}^k + \beta$. The constant factor $\bar{s}_i(\omega)$ can be precomputed and stored. Hence, for each element of $\Psi_\beta^{(0)}(i,r)$, the calculation requires a multiplication and an addition.

To evaluate each element of $\Psi_\beta^{(1)}(i,r)$, we have

$$\Delta_i^r(\omega + v_i) = \Delta_{i+1}^r(\omega + v_i) + \bar{s}_i(\omega + v_i)\Delta_{i+1}^{r+2^i}(\omega + v_i), \tag{29}$$

for $\omega \in V_{i+1}^k + \beta$. By Lemma 4, we have

$$\Delta_{i+1}^r(\omega + v_i) = \Delta_{i+1}^r(\omega);$$

$$\Delta_{i+1}^{r+2^i}(\omega + v_i) = \Delta_{i+1}^{r+2^i}(\omega).$$

Furthermore, the factor can be rewritten as

$$\begin{aligned}\bar{s}_i(\omega + v_i) =& \frac{s_i(\omega + v_i)}{s_i(v_i)} = \frac{s_i(\omega) + s_i(v_i)}{s_i(v_i)}\\ =&\frac{s_i(\omega)}{s_i(v_i)} + 1 = \bar{s}_i(\omega) + 1.\end{aligned} \tag{30}$$

With above results, (29) can be rewritten as

$$\begin{aligned}&\Delta_i^r(\omega + v_i)\\ =&\Delta_{i+1}^r(\omega) + (\bar{s}_i(\omega) + 1)\,\Delta_{i+1}^{r+2^i}(\omega)\\ =&\Delta_{i+1}^r(\omega) + \bar{s}_i(\omega)\Delta_{i+1}^{r+2^i}(\omega) + \Delta_{i+1}^{r+2^i}(\omega)\\ =&\Delta_i^r(\omega) + \Delta_{i+1}^{r+2^i}(\omega).\end{aligned} \tag{31}$$

It can be seen that $\Delta_i^r(\omega) \in \Psi_\beta^{(0)}(i,r)$, and $\Delta_{i+1}^{r+2^i}(\omega) \in \Psi_\beta(i+1,r+2^i)$, for $\omega \in V_{i+1}^k + \beta$. Hence, it requires an addition. Algorithm 1 depicts the steps of the algorithm. We call the procedure by the following instruction

$$\Psi_\beta(0,0) \leftarrow \mathrm{FFT}_h(\Delta_0^0, \beta, 0, 0),$$

where $\Psi_\beta(0,0)$ is the desired output, and Δ_0^0 is the set including the coefficients of the input polynomial $D(x)$.

Algorithm 1 Transform of the basis X

Input: $\mathrm{FFT}_h(\Delta_i^r, \beta, i, r)$: Δ_i^r is defined in (13), $h = 2^k$ denotes the size of the transform, and $\beta \in \mathbb{F}_{2^m}$

Output: $\Psi_\beta(i,r) = \{\Delta_i^r(\omega) | \omega \in V_i^k + \beta\}$

1: **if** $i = k$ **then return** d_r
2: **end if**
3: Call $\Psi_\beta(i+1,r) \leftarrow \mathrm{FFT}_{h/2}(\Delta_{i+1}^r, \beta, i+1, r)$, where $\Delta_{i+1}^r \subset \Delta_i^r$
4: Call $\Psi_\beta(i+1, r+2^i) \leftarrow \mathrm{FFT}_{h/2}(\Delta_{i+1}^{r+2^i}, \beta, i+1, r+2^i)$, where $\Delta_{i+1}^{r+2^i} \subset \Delta_i^r$
5: **for** $j = 0, \ldots, 2^{k-i-1} - 1$ **do**
6:

$$\begin{aligned}&\Delta_i^r(\omega_{j2^{i+1}})\\ \leftarrow&\Delta_{i+1}^r(\omega_{j2^{i+1}}) + \bar{s}_i(\omega_{j2^{i+1}})\Delta_{i+1}^{r+2^i}(\omega_{j2^{i+1}})\end{aligned}$$

7: $\Delta_i^r(\omega_{j2^{i+1}+2^i}) \leftarrow \Delta_i^r(\omega_{j2^{i+1}}) + \Delta_{i+1}^{r+2^i}(\omega_{j2^{i+1}})$
8: **end for**
9: **return**

$$\begin{aligned}\Psi_\beta(i,r) =&\{\Delta_i^r(\omega_{j2^i}) | i = 0, \ldots, 2^{k-i-1} - 1\}\\ &\cup \{\Delta_i^r(\omega_{j2^i} + v_i) | i = 0, \ldots, 2^{k-i-1} - 1\}\end{aligned}$$

C. Complexity

We start to discuss the computational and space complexities of the proposed Fourier-like transform. For the computational complexity, let $A(h)$ and $M(h)$ respectively denote the number of additions and multiplications used in the algorithm. From (110) and (31), the recurrence relation is formulated as

$$\begin{aligned} A(h) &= 2 \times A(h/2) + h; \qquad A(1) = 0; \\ M(h) &= 2 \times M(h/2) + h/2; \qquad M(1) = 0. \end{aligned} \tag{32}$$

The solution is given by

$$A(h) = h \lg(h); \ M(h) = \frac{h}{2} \lg(h).$$

Note that if $\omega+\beta = 0$ in (110), the formula can be simplified to

$$\Delta_i^r(\omega + \beta) = \Delta_{i+1}^r(\omega + \beta), \tag{33}$$

which does not involve any arithmetic operations. This case occurs when the set of evaluation points is defined as V_k. In this case, the number of saved operations is less than h in both additions and multiplications, and thus the big-O complexity is unchanged.

Assume the factor $\bar{s}_i(\omega)$ in (110) is pre-computed and stored. We consider the number of factors to be stored. In (110), the set of the factors is

$$\{\bar{s}_i(\omega) | \omega \in V_{i+1}^k + \beta\},$$

that has 2^{k-i-1} elements.

As shown in the algorithm, $\Psi_\beta(i, r)$ is divided into two parts: $\Psi_\beta^{(0)}(i, r)$ and $\Psi_\beta^{(1)}(i, r)$, where $\Psi_\beta^{(0)}(i, r)$ involves the constant factors but $\Psi_\beta^{(1)}(i, r)$ does not. Let $N(h)$, with $h = 2^{k-i}$, denote the number of constant factors used in the computation of $\Psi_\beta(i, r)$. The recurrence relation is given by

$$N(h) = 2N(h/2) + h/2, \qquad N(1) = 0,$$

and the solution is $N(h) = h - 1$, which is linear complexity.

D. Inverse transform

In the inverse transform, the input is $\{D(\omega) | \omega \in V_k + \beta\}$, and the objective is to calculate the coefficients of $D(x)$. The inversion can be done by backtracking the transform. In the inverse transform, $\Psi_\beta(i, r)$ is given, and the objective is to compute $\Psi_\beta(i + 1, r)$ and $\Psi_\beta(i + 1, r + 2^i)$.

For $\Psi_\beta(i + 1, r + 2^i)$, (31) is reformulated as

$$\Delta_{i+1}^{r+2^i}(\omega) = \Delta_i^r(\omega) + \Delta_i^r(\omega + v_i), \tag{34}$$

where $\Delta_i^r(\omega) \in \Psi_\beta^{(0)}(i, r)$, and $\Delta_i^r(\omega + v_i) \in \Psi_\beta^{(1)}(i, r)$. This takes an addition. For $\Psi_\beta(i + 1, r)$, (110) is reformulated as

$$\Delta_{i+1}^r(\omega) = \Delta_i^r(\omega) + \bar{s}_i(\omega)\Delta_{i+1}^{r+2^i}(\omega), \tag{35}$$

where $\Delta_i^r(\omega) \in \Psi_\beta^{(0)}(i, r)$, and $\Delta_{i+1}^{r+2^i}(\omega) \in \Psi_\beta(i+1, r+2^i)$. This takes an addition and a multiplication. Consequently, the inverse algorithm has the same computational complexity with the transform. The steps are shown in Algorithm 2.

Figure 1 depicts an example of the proposed transform of length $h = 8$. Figure 1(a) shows the transform $\mathrm{FFT}(\Delta_0^0, \beta, 0, 0)$. The dotted line arrow denotes that the element should be multiplied with the factor $\bar{s}_j(\bullet)$ upon adding together with other element. For simplicity, we use the notations

$$\Delta_{i,j}^r = \Delta_i^r(\omega_j + \beta), \qquad \bar{s}_i^j = \bar{s}_i(\omega_j).$$

The two gray blocks indicate the calls $\mathrm{FFT}(\Delta_1^0, \beta, 1, 0)$ and $\mathrm{FFT}(\Delta_1^1, \beta, 1, 1)$ in Algorithm 1. The inversion is shown in Figure 1(b).

Algorithm 2 Inverse transform of the basis X

Input: $\mathrm{IFFT}_h(\Psi_\beta(i, r), \beta, i, r)$: $\Psi_\beta(i, r) = \{\Delta_i^r(\omega) | \omega \in V_i^k + \beta\}$, $h = 2^k$ denotes the size of the transform, and $\beta \in \mathbb{F}_{2^m}$
Output: Δ_i^r defined in (13)

```
if i = k then return d_r
end if
for j = 0, ..., 2^{k-i-1} - 1 do
    Δ^{r+2^i}_{i+1}(ω_{j2^{i+1}}) ← Δ^r_i(ω_{j2^{i+1}}) + Δ^r_i(ω_{j2^{i+1}+2^i})
    Δ^r_{i+1}(ω_{j2^{i+1}}) ← Δ^r_i(ω_{j2^{i+1}}) + s̄_i(ω_{j2^{i+1}})Δ^{r+2^i}_{i+1}(ω_{j2^{i+1}})
end for
Call Δ^r_{i+1} ← IFFT_{h/2}(Ψ_β(i+1, r), β, i+1, r)
Call Δ^{r+2^i}_{i+1} ← IFFT_{h/2}(Ψ_β(i+1, r+2^i), β, i+1, r+2^i)
return Δ^r_i = Δ^r_{i+1} ∪ Δ^{r+2^i}_{i+1}
```

IV. Formal derivative

In this section, the algorithm for the formal derivative in the new basis is proposed. Section IV-A gives the closed form. By directly following the formula, the formal derivative requires $\mathcal{O}(h \lg(h))$ in both additive and multiplicative complexity. Section IV-B presents an improved approach, that requires $\mathcal{O}(h \lg(h))$ additions and $\mathcal{O}(h)$ multiplications.

A. The closed form

First we present a Lemma that will be used in the closed-form.

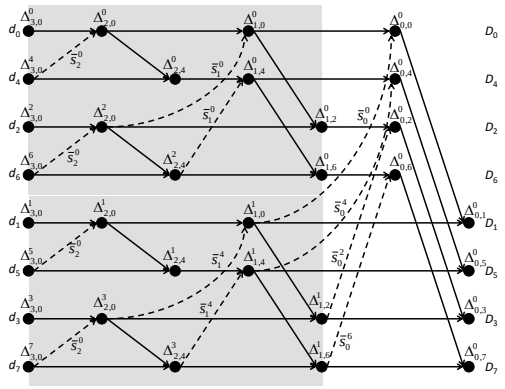
Lemma 5. *The formal derivative of $s_k(x)$ is a constant given by*

$$s_k'(x) = \prod_{w \in V_k \setminus \{0\}} w. \tag{36}$$

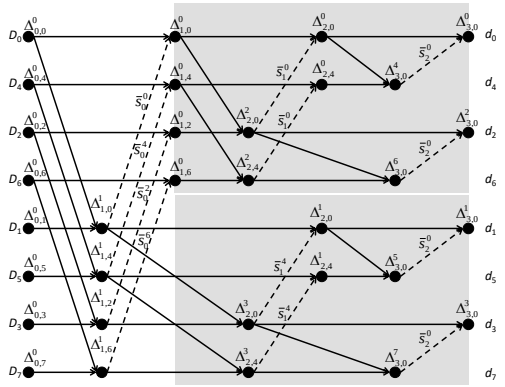
Proof. To begin with, we recall the definition of the formal derivative. Let $B(x) = b \cdot x^j$ denote a polynomial of $\mathbb{F}_{2^m}[x]$. It is well known if j is even, then $B'(x) = 0$, or else $B'(x) = b \cdot x^{j-1}$. From Lemma 2, the non-zero terms of $s_k(x)$ are among the degrees of $1, 2, 4, \ldots, 2^k$. Thus, $s_k'(x)$ is a constant, which is the coefficient of $s_k(x)$ with degree 1. The value is

$$\sum_{\ell \in V_k} \prod_{w \neq \ell} w = \prod_{w \in V_k \setminus \{0\}} w.$$

This completes the proof. □



(a) The Fourier-like transform



(b) The inverse Fourier-like transform

Fig. 1. Data flow diagram of proposed Fourier-like transform and its inversion of length $h = 8$.

Since $s'_k(x)$ is a constant, we define $\bar{s}'_k = s'_k(x)/s_k(v_k)$. From Lemma 5, the formal derivative of $X_i(x)$ becomes

$$\begin{aligned} X'_i(x) &= \sum_{\ell=0}^{m-1} i_\ell \bar{s}'_\ell(x) \prod_{j\neq\ell} (\bar{s}_j(x))^{i_j} \\ &= \sum_{\ell=0}^{m-1} i_\ell \bar{s}'_\ell(x) X_{i-2^\ell}(x) \\ &= \sum_{\ell\in I_i} \bar{s}'_\ell X_{i-2^\ell}(x), \end{aligned} \tag{37}$$

where

$$I_i = \{j | j \in \{0, 1, \ldots, m-1\}, i_j = 1\}$$

includes the non-zero indices of the binary representation of i. For example, if $i = 13 = 2^0+2^2+2^3$, we have $I_{13} = \{0, 2, 3\}$, and

$$\begin{aligned} &X'_{13}(x) \\ =&\bar{s}'_0 \bar{s}_2(x)\bar{s}_3(x) + \bar{s}'_2 \bar{s}_0(x)\bar{s}_3(x) + \bar{s}'_3 \bar{s}_0(x)\bar{s}_2(x) \\ =&\bar{s}'_0 X_{12}(x) + \bar{s}'_2 X_9(x) + \bar{s}'_3 X_5(x). \end{aligned} \tag{38}$$

From (37), the formal derivative of $D(x)$ is given by

$$D'(x) = \sum_{i=0}^{h-1} d_i X'_i(x) = \sum_{i=0}^{h-1} d_i \sum_{\ell\in I_i} \bar{s}'_\ell X_{i-2^\ell}(x). \tag{39}$$

In (39), for a specified X_j it can come from X_{i-2^ℓ} when $i - 2^\ell = j$ and ℓ does not belong to I_j. Recall that $2^k = h$, i.e. $\log h = k$. Hence, (39) can be further rearranged as

$$D'(x) = \sum_{j=0}^{h-1} X_j(x) \sum_{\ell\in I_j^c} \bar{s}'_\ell \cdot d_{j+2^\ell}, \tag{40}$$

where I_j^c is the complement of I_j defined as

$$I_j^c = \mathbb{Z}_k \setminus I_j.$$

From (40), each coefficient of $D'(x)$ requires at most $k-1$ additions and k multiplications, whereas the set of constants $\{\bar{s}'_\ell\}_{\ell=0}^{k-1}$ can be precomputed and stored. Since $h = 2^k$, this requires $O(h \lg(h))$ operations, in both additive complexity and multiplicative complexity.

B. Algorithm with lower multiplicative complexity

This subsection presents an improved approach on performing formal derivative in $\mathcal{O}(h \lg(h))$ additions and $\mathcal{O}(h)$ multiplications. Let

$$B = \left\{ B_i = \prod_{j\in I_i} \bar{s}'_j \right\}_{i=0}^{h-1}, \tag{41}$$

and

$$d_i^{\mathrm{d}} = d_i \cdot B_i, \qquad i = 0, 1, \ldots h-1. \tag{42}$$

By plugging $d_i = d_i^{\mathrm{d}} B_i^{-1}$ to (40), we have

$$D'(x) = \sum_{j=0}^{h-1} X_j(x) \sum_{\ell\in I_j^c} \frac{d^{\mathrm{d}}_{j+2^\ell} \bar{s}'_\ell}{B_{j+2^\ell}}. \tag{43}$$

As

$$B_{j+2^\ell} = \prod_{m\in I_{j+2^\ell}} \bar{s}'_m = \bar{s}'_\ell \prod_{m\in I_j} \bar{s}'_m = \bar{s}'_\ell B_j,$$

(43) can be rewritten as

$$\begin{aligned} D'(x) &= \sum_{j=0}^{h-1} X_j(x) \sum_{\ell\in I_j^c} \frac{d^{\mathrm{d}}_{j+2^\ell} \bar{s}'_\ell}{\bar{s}'_\ell B_j} \\ &= \sum_{j=0}^{h-1} X_j(x) \sum_{\ell\in I_j^c} \frac{d^{\mathrm{d}}_{j+2^\ell}}{B_j} \\ &= \sum_{j=0}^{h-1} X_j(x) \frac{\sum_{\ell\in I_j^c} d^{\mathrm{d}}_{j+2^\ell}}{B_j} \\ &= \sum_{j=0}^{h-1} X_j(x) \frac{d_j^{\mathrm{s}}}{B_j}, \\ &= \sum_{j=0}^{h-1} X_j(x) d'_j, \end{aligned} \tag{44}$$

where

$$d_j^{\mathrm{s}} = \sum_{\ell\in I_j^c} d^{\mathrm{d}}_{j+2^\ell} \qquad j = 0, 1, \ldots, h-1, \tag{45}$$

$$d'_j = d_j^{\mathrm{s}}/B_j \qquad i = 0, 1, \ldots, h-1. \tag{46}$$

Based on the above formulas, the approach consists of three steps. Assume the set B was precomputed and stored. The first step is to compute $\{d_i^{\rm d}\}_{i=0}^{h-1}$ defined in (42). The second step is to compute $\{d_j^{\rm s}\}_{j=0}^{h-1}$ defined in (45), and the final step is to compute the result $\{d'_j\}_{j=0}^{h-1}$ defined in (46). For the complexity, the first and the third steps require h multiplications. In the second step, it takes around $\frac{1}{2}h\lg(h)$ additions.

Next an example is given to demonstrate how to obtain $D'(x)$. If $h=8$ and the set B includes 8 elements defined as

$$\begin{aligned} &B_0=1; \quad B_1=\bar{s}'_0; B_2=\bar{s}'_1; \quad B_3=\bar{s}'_0\bar{s}'_1;\\ &B_4=\bar{s}'_2; \quad B_5=\bar{s}'_0\bar{s}'_2; B_6=\bar{s}'_1\bar{s}'_2; \quad B_7=\bar{s}'_0\bar{s}'_1\bar{s}'_2. \end{aligned} \tag{47}$$

In the first step, each $d_i^{\rm d}$ is computed by

$$d_i^{\rm d}=d_iB_i, \qquad i=0,1,\ldots,7.$$

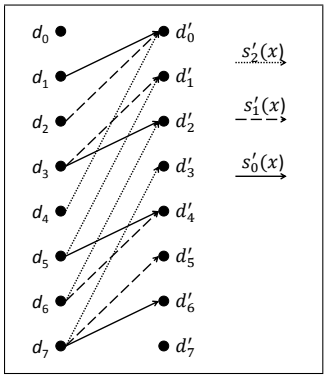
In the second step,

$$\begin{aligned} d_0^{\rm s}&=d_1^{\rm d}+d_2^{\rm d}+d_4^{\rm d}; & d_1^{\rm s}&=d_3^{\rm d}+d_5^{\rm d};\\ d_2^{\rm s}&=d_3^{\rm d}+d_6^{\rm d}; & d_3^{\rm s}&=d_7^{\rm d};\\ d_4^{\rm s}&=d_5^{\rm d}+d_6^{\rm d}; & d_5^{\rm s}&=d_7^{\rm d};\\ d_6^{\rm s}&=d_7^{\rm d}; & d_7^{\rm s}&=0. \end{aligned} \tag{48}$$

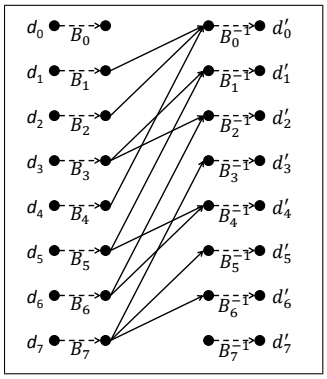
In the final step, $D'_8(x)$ is computed by

$$\begin{aligned} D'_8(x)=&X_0(x)\frac{d_0^{\rm s}}{B_0}+X_1(x)\frac{d_1^{\rm s}}{B_1}+X_2(x)\frac{d_2^{\rm s}}{B_2}\\ &+X_3(x)\frac{d_3^{\rm s}}{B_3}+X_4(x)\frac{d_4^{\rm s}}{B_4}+X_5(x)\frac{d_5^{\rm s}}{B_5}+X_6(x)\frac{d_6^{\rm s}}{B_6}. \end{aligned}$$

Figure 2(b) shows the improved version for $h=8$ in graphical diagrams.



(a) Direct approach.



(b) Improved approach.

Fig. 2. Data flow diagram of formal derivative of length $h=8$.

V. Reed-Solomon erasure codes

In this section, we propose the encoding and erasure decoding algorithms for $(n=2^m,k)$ single extended Reed-Solomon (RS) codes over binary extension fields. There exist two major viewpoints for the Reed-Solomon codes, termed as polynomial evaluation approach and generator polynomial approach. In this paper, we follow the polynomial evaluation approach, which treats the codeword symbols as the evaluation values of a polynomial $f(x)\in\mathbb{F}_{2^m}[x]$ of degree $k\le 2^m-1$. The codeword is defined as

$$f=(f(\omega_0),f(\omega_1),\ldots,f(\omega_{2^m-1})),$$

where $\omega_i\in\mathbb{F}_{2^m}$ defined in (2). The message is denoted as

$$m=(m_0,m_1,\ldots,m_{k-1})$$

for each $m_i\in\mathbb{F}_{2^m}$. In the systematic construction, we require

$$f(\omega_i)=m_i, \qquad i=0,1,\ldots,k-1. \tag{49}$$

In decoding, when any k out of $n=2^m$ symbols are received, one can uniquely reconstruct $f(x)$ via polynomial interpolation, and thus the erasures can be computed accordingly.

In the following, the algorithms for encoding and erasure decoding are proposed. The proposed encoding algorithm is designed only when k is a power of two and $n=2^m$. If the given k is not a power of two, there are two methods to perform the encoding procedure. In the first method, the code can be obtained by using shortening technique. Precisely, some zeros are appended to the message vector such that the size is a power of two. Let

$$\bar{m}=(m_0,m_1,\ldots,m_{k-1},m_k,\ldots,m_{k'-1}),$$

with $m_k=\cdots=m_{k'-1}=0$ and $k'=2^{\lceil\log_2(k)\rceil}$. Then $\bar{m}$ is coded by (n,k') RS encoding algorithm. After obtaining the codeword, the zero symbols $\{m_i\}_{i=k}^{k'-1}$ are removed. In decoding, the received codeword is decoded by (n,k') RS erasure decoding algorithms by appending the removed zero symbols to the received codeword.

In the second method, the erasure decoding algorithm can be applied to compute the parities of a codeword. Precisely, we create a "received codeword" by filling the message part with message symbols, and the parity part are marked with erasures. Then the erasure decoding algorithm is applied on this "received codeword" to get the values of the erasures that are the parity symbols.

A. Encoding algorithm

Given $\beta\in\mathbb{F}_{2^m}$ and $D(x)\in\mathbb{F}_{2^m}[x]/(x^{2^m}-x)$ with $\deg(D(x))<h$, the notation $\mathrm{FFT}_h^\beta(\bullet)$ denotes that the proposed transform with shifting β is applied to the input vector $\bullet$ of size h. Precisely, the transform

$$\bar{D}\leftarrow\mathrm{FFT}_h^\beta(D)$$

returns a vector

$$\bar{D}=(D(\omega_0+\beta),D(\omega_1+\beta),\ldots,D(\omega_{h-1}+\beta)).$$

Algorithm 3 Reed-Solomon encoding algorithm

Input: a k-element message vector m
Output: an n-element systematic codeword f
1: $\bar{m} \leftarrow \mathrm{IFFT}_k^0(m)$
2: **for** $i \leftarrow 1, \ldots, (n/k-1)$ **do**
3: $\quad \bar{m}_i \leftarrow \mathrm{FFT}_k^{i \cdot k}(\bar{m})$
4: **end forreturn** $f \leftarrow (m, \bar{m}_1, \bar{m}_2, \ldots, \bar{m}_{n/k-1})$

In contrast, the inverse transform is denoted as

$$D \leftarrow \mathrm{IFFT}_h^{\beta}(\bar{D}).$$

Algorithm 3 illustrates the pseudocode of the encoding algorithm. Line 1 computes the coefficients of

$$\bar{m}(x) = \sum_{i=0}^{k-1} \bar{m}_i X_i(x). \tag{50}$$

It is clear that $\deg(\bar{m}(x)) \leq k-1$ and

$$\bar{m}(\omega_i) = m_i, \qquad i = 0, 1, \ldots, k-1. \tag{51}$$

Thus, we conclude that $\bar{m}(x) = f(x)$, and the parity-check symbols can be computed by applying the transforms on $\bar{m}$ (see Lines 2-4). The parity-check symbols are obtained in blocks with size k and there are $n/k-1$ blocks.[1] In Line 5, those vectors are assembled to get the codeword vector f.

For the computational complexity, the proposed encoding algorithm requires a k-element IFFT and $(n/k-1)$ times of k-element FFT. Thus, the encoding algorithm has the complexity

$$\mathcal{O}((n/k)k \lg(k)) = \mathcal{O}(n \lg(k)).$$

B. Erasure decoding algorithm

The decoding algorithm is based on our previous work [11], that requires the polynomial evaluations and it's derivatives. The code considered in [11] is based on Fermat number transforms (FNT) over $\mathbb{F}_{2^m+1}$. In this paper, the FNT [11] is replaced with the proposed transform. However, since the transform is in a different basis, the formula should be reformulated to fit the format.

Assume the received codeword $\bar{f}$ has $n-k$ erasures. The error locator polynomial is defined as

$$\Pi(x) = \prod_{\omega \in R} (x + \omega),$$

where

$$R = \{r_i\}_{i=0}^{n-k-1}$$

denotes the set of evaluation points for erasures.

Let

$$\hat{f}(x) = f(x)\Pi(x)$$

of degree $\deg(\hat{f}(x)) = \deg(f(x)) + \deg(\Pi(x)) \leq n-1$. The formal derivative of $\hat{f}(x)$ is

$$\hat{f}'(x) = f'(x)\Pi(x) + f(x)\Pi'(x). \tag{52}$$

[1]Since k and n are both powers of 2, n is divisible by k.

By substituting $x = \omega \in R$ into (52), we have

$$\begin{aligned} \hat{f}'(\omega) &= f(\omega)\Pi'(\omega) \\ \Rightarrow f(\omega) &= \frac{\hat{f}'(\omega)}{\Pi'(\omega)}, \qquad \forall \omega \in R. \end{aligned} \tag{53}$$

Based on the above formulas, the decoding procedure consists of three major stages: First, compute the coefficients of $\hat{f}(x)$; second, compute the formal derivative of $\hat{f}(x)$; and third, compute the erasures through (53). The details are given in Algorithm 4.

Line 1 computes two sets $\bar{\Pi}$ and Π', where

$$\bar{\Pi} = \{\Pi(\omega) | \omega \in \mathbb{F}_{2^m} \backslash R\} \tag{54}$$

and

$$\Pi' = \{\Pi'(\omega) | \omega \in R\}. \tag{55}$$

Notice that this step does not use the codeword symbols. Thus, if we have many codewords with the same locations of erasures, $\bar{\Pi}$ and Π' can be computed once and used in each codeword. Line 2 computes the evaluations of $\hat{f}(x)$ at $\mathbb{F}_{2^m}$, in which the factor $\Pi(r)$ is taken from $\bar{\Pi}$. Line 3 applies IFFT on Φ to obtain $\bar{\Phi} = (\bar{\phi}_0, \bar{\phi}_1, \ldots, \bar{\phi}_{n-1})$, which forms a polynomial

$$\bar{\Phi}(x) = \sum_{i=0}^{n-1} \bar{\phi}_i X_i(x),$$

such that $\bar{\Phi}(\omega_j) = \hat{f}(\omega_j)$, for $0 \leq j \leq n-1$. Thus, we conclude $\bar{\Phi}(x) = \hat{f}(x)$. In Line 4, the formal derivative of $\bar{\Phi}(x)$ can be computed by the method in Section IV, resulting in $\bar{\Phi}' = (\bar{\phi}'_0, \bar{\phi}'_1, \ldots, \bar{\phi}'_{n-1})$ forming a polynomial

$$\bar{\Phi}'(x) = \sum_{i=0}^{n-1} \bar{\phi}'_i X_i(x).$$

Line 5 applies FFT on $\bar{\Phi}'$ to obtain Φ^{d}, which is a vector consists of $\{\hat{f}'(r) | r \in \mathbb{F}_{2^m}\}$. In Line 6, Φ^{d}_j is an element of Φ^{d}, and $\Pi'(j) \in R$.

The complexity of this algorithm is dominated by Steps 1, 3, 4 and 5, whereas Step 2 only takes k multiplications and Step 6 only takes $n-k$ divisions. By the proposed FFT and IFFT algorithms, Step 3 and Step 5 can be performed in $\mathcal{O}(n \lg(n))$ field operations. By Section IV, Step 4 requires $\mathcal{O}(n \lg(n))$ field additions and $\mathcal{O}(n)$ field multiplications. In Step 1, we employ the algorithm shown in Appendix, which requires $\mathcal{O}(n \lg(n))$ additions in modulo (2^m-1). In summary, this algorithm has the complexity of $\mathcal{O}(n \lg(n))$.

VI. Discussions and Comparisons

A. Complexities of polynomial operations

By using the proposed Fourier-like transforms, the fast polynomial multiplications in the proposed basis can be derived. Table I tabulates the complexities of some polynomial operations in the monomial basis and the proposed basis over binary extension fields. In particular, the polynomial addition is simple by adding the coefficients of two polynomials, so the complexity is $\mathcal{O}(h)$ in both basis. The formal derivative in proposed basis requires $\mathcal{O}(h \lg(h))$ field operations. In

Algorithm 4 Reed-Solomon erasure decoding algorithm

Input: Received codeword $\bar{f}$, and the positions of erasures $R = \{r_i\}_{i=0}^{n-k-1}$

Output: The erasures $\{f(j)|j \in R\}$

1: Compute

$$\bar{\Pi} \leftarrow \{\Pi(\omega)|\omega \in \mathbb{F}_{2^m} \backslash R\} \tag{56}$$

$$\Pi' \leftarrow \{\Pi'(\omega)|\omega \in R\} \tag{57}$$

2: Compute $\Phi \leftarrow (\hat{f}(\omega_0), \hat{f}(\omega_1), \ldots, \hat{f}(\omega_{n-1}))$ by

$$\hat{f}(r) = \begin{cases} 0 & \forall r \in R \\ f(r)\Pi(r) & \text{otherwise} \end{cases} \tag{58}$$

3: Compute

$$\bar{\Phi} \leftarrow \mathrm{IFFT}_n^0(\Phi) \tag{59}$$

4: Compute the formal derivative of $\bar{\Phi}$, denoted as $\bar{\Phi}'$

5: Compute

$$\Phi^{\mathrm{d}} \leftarrow \mathrm{FFT}_n^0(\bar{\Phi}') \tag{60}$$

6: Compute the erasures via

$$f(j) = \frac{\Phi_j^{\mathrm{d}}}{\Pi'(j)}, \qquad \forall j \in R$$

TABLE I
COMPLEXITIES OF OPERATIONS IN POLYNOMIAL BASIS OVER $\mathbb{F}_{2^m}$

Operations	monomial basis	proposed basis
Addition	$\mathcal{O}(h)$	$\mathcal{O}(h)$
Multiplication	$\mathcal{O}(h \lg(h) \lg\lg(h))$	$\mathcal{O}(h \lg(h))$
Formal derivative	$\mathcal{O}(h)$	$\mathcal{O}(h \lg(h))$

contrast, the formal derivative in monomial basis only requires $\mathcal{O}(h)$.

Next we consider the polynomial multiplication. For then monomial basis, the asymptotically fastest algorithm was proposed by Schönhage [3], in 1977. The algorithm takes $\mathcal{O}(h \lg(h) \lg\lg(h))$ field operations. For the proposed basis, the fast approach is based on the Fourier-like transform. Let $a(x) = \sum_{i=0}^{h-1} a_i \cdot X_i(x)$ and $b(x) = \sum_{i=0}^{h-1} b_i \cdot X_i(x)$. Its product $c(x) = a(x) \cdot b(x)$ can be computed as

$$c = \mathrm{IFFT}_\beta(\mathrm{FFT}_\beta(a) \otimes \mathrm{FFT}_\beta(b)),$$

where $\otimes$ performs pairwise multiplication on two vectors. a (b) is the vector of coefficients of $a(x)$ ($b(x)$) by appending zeros on high degrees such that its length is up to 2^j, where 2^j is the smallest integer that is larger than or equal to $2h-1$. This requires one 2^j-point IFFT, two 2^j-point FFTs and 2^j multiplications, and thus the complexity is $\mathcal{O}(2^j \lg(2^j)) = \mathcal{O}(h \lg(h))$.

B. Discussions about RS algorithms

Traditionally, the polynomials for RS codes are represented in the monomial basis. However, Algorithm 4 uses the new basis to represent $\hat{f}(x)$. Assume the basis of $\hat{f}(x)$ is settled as the monomial basis. In this case, Lines 3, 4, 5 shall be replaced with the arithmetic algorithms for the monomial basis. Particularly, the formal derivative in Line 4 takes $\mathcal{O}(n)$. For Line 3 and Line 5, we shall choose the finite field FFT in the monomial basis, that takes $\mathcal{O}(n \lg(n) \lg\lg(n))$ field operations. Thus, the complexity of this algorithm is $\mathcal{O}(n \lg(n) \lg\lg(n))$. The same result is concluded in Algorithm 3. If $\bar{m}(x)$ in (50) is represented in the monomial basis, the encoding algorithm shall take $\mathcal{O}(n \lg(n) \lg\lg(n))$.

As the proposed algorithm employs a portion of the algorithm proposed by Didier [12], we briefly introduce Didier's approach as follows. In 2009, Didier [12] proposed an erasure decoding algorithm for Reed-Solomon codes based on fast Walsh-Hadamard transforms. The algorithm [12] consists of two major parts: the first part is to compute the polynomial evaluations of the error locator polynomial (see Appendix). The second part is decomposing the Lagrange polynomial into several logical convolutions, which are then respectively computed with fast Walsh-Hadamard transforms. The first part requires $\mathcal{O}(n \lg(n))$, and the second part requires $\mathcal{O}(n \lg^2(n))$, so the overall complexity is $\mathcal{O}(n \lg^2(n))$. For the proposed algorithm, the first part of algorithm given in [12] is employed. Furthermore, we design another decoding structure based on the proposed basis. The proposed transform only requires $\mathcal{O}(n \lg(n))$, so that the proposed approach can reduce the complexity from $\mathcal{O}(n \lg^2(n))$ to $\mathcal{O}(n \lg(n))$.

To demonstrate the real performance, the proposed algorithm was implemented in C and was run on a PC with Intel core i7-950 CPU. While $n = 2^{16}$, $k/n = 1/2$, the program took about 1.12 seconds to generate a codeword, and 3.06 seconds to decode an erased codeword on average. On the other hand, we also ran the program written by the author in [12] on the same platform. In our simulation, the program implemented the algorithm given in [12] took about 52.91 seconds in both encoding and erasure decoding under the same parameter setting. Thus, the proposed erasure decoding is around 17 times faster than that given in[12] for $n = 2^{16}$.

VII. LITERATURE REVIEW

In [13], the codewords of RS codes are treated as a sequence of evaluations of polynomials interpreted by the messages. From this viewpoint, the encoding process can be treated as an oversampling process through discrete Fourier transform over finite fields. Some studies [14–16] indicated that, if an $\mathcal{O}(n \lg(n))$ finite field FFT is available, the error-correction decoding can be reduced to $\mathcal{O}(n \lg^2(n))$. An n-point radix-2 FFT butterfly diagram requires $n \lg(n)$ additions and $\frac{1}{2} n \lg(n)$ multiplications. This FFT butterfly diagram can be directly applied on Fermat prime fields $\mathbb{F}_{2^m+1}, m \in \{1, 2, 4, 8, 16\}$. In this case, the transform, referred to as Fermat number transform (FNT), also requires $n \lg(n)$ field additions and $\frac{1}{2} n \lg(n)$ field multiplications. By employing FNTs, a number of fast approaches [14, 17, 18] had been presented to reduce the complexity of encoding and decoding of RS codes. Some FNT-based RS erasure decoding algorithms had been proposed [11, 19, 20] in $\mathcal{O}(n \lg(n))$ field operations. Thus far, no existing algorithm for (n, k) RS codes has decoding complexity achieving lower than $\Omega(n \lg(n))$ operations, in a context of a fixed coding rate k/n. However, the major drawback of FNT is that the number of possible values of each symbol is $2^m + 1$,

that cannot be losslessly stored in m-bit binary digit. Thus, the FNT-based codes are impractical in applications.

In real applications, RS codes are usually constructed over extension binary fields. This arises the attentions of FFTs over extension binary fields. Table II tabulates the arithmetic complexities of FFT algorithms over extension binary fields. Table II shows that no FFT algorithm has achieved $\mathcal{O}(n\lg(n))$ in both additive and multiplicative complexities. This implies that when the polynomials in RS codes are in monomial basis, the complexity will fail to reach $\mathcal{O}(n\lg(n))$.

There exist faster approaches for some non-MDS codes. Such codes, termed as fountain codes [21], require a little more than k codeword symbols to recover the original message. Two remarkable classes of fountain codes are LT code [22] and Raptor code [23]. Due to the low complexity, fountain codes have significant merits in many applications. However, MDS codes have some irreplaceable properties, so that RS codes are still used nowadays.

VIII. Basis conversion

This section gives two approaches to convert the representation of a polynomial from the monomial basis $\mathbf{m}(x) = \{1, x, \ldots, x^{2^m-1}\}$ to $\mathbb{X}$. Specifically, the input is defined as $D(x) = \sum_{i=0}^{n-1} a_i x^i$, where the size of input is a power of two $n \in \{2^j\}_{j=1}^{m}$. The conversion algorithm outputs $\{\bar{a}_i\}_{i=0}^{n-1}$ such that $D(x) = \sum_{i=0}^{n-1} \bar{a}_i X_i(x)$. In particular, the first approach is for arbitrary m with $\mathcal{O}(n\lg^2(n))$ field operations. The second approach is only for m a power of two, and its complexity is $\mathcal{O}(n\lg(n)\lg\lg(n))$. The ideas of both algorithms are based on [7]. In particular, the FFTs in [7] are algebraically similar to the approach by combining the proposed transforms with the basis conversion algorithms in this section.

A. First approach (For arbitrary m)

Given $D(x) = \sum_{i=0}^{n-1} d_i x^i$, we want to find out $\{(d_{i,0}, d_{i,1})\}_{i=0}^{n/2-1}$ such that

$$D(x) = \sum_{i=0}^{n/2-1} (d_{i,0} + d_{i,1}\frac{x}{v_0})(s_1(x))^i. \tag{61}$$

Notably, $x/v_0 = s_0(x)/s_0(v_0)$ and $s_1(x) = x^2 + v_0 x$. Then (61) can be rewritten as

$$\begin{aligned} D(x) &= \sum_{i=0}^{n/2-1} d_{i,0}(s_1(x))^i + \frac{x}{v_0}\sum_{i=0}^{n/2-1} d_{i,1}(s_1(x))^i \\ &= D_0(s_1(x)) + \frac{x}{v_0}\cdot D_1(s_1(x)), \end{aligned} \tag{62}$$

where

$$D_j(x) = \sum_{i=0}^{n/2-1} d_{i,j}x^j \qquad j = 0, 1. \tag{63}$$

Lemma 6. *Let* $\boldsymbol{v}_m = (v_0, v_1, \ldots, v_{m-1})$ *and*

$$\begin{aligned} \boldsymbol{w}_{m-1} &= (w_1, w_2, \ldots, w_{m-1}) \\ &= (s_1^{V_1}(v_1), s_1^{V_1}(v_2), \ldots, s_1^{V_1}(v_{m-1})), \end{aligned} \tag{64}$$

where $s_1^{V_1}(x) = x(x - v_0)$. *Let*

$$W_j = \mathrm{Span}(w_1, w_2, \ldots, w_{j-1}) \qquad j = 1, 2, \ldots, m-1.$$

A polynomial basis with respective to $\boldsymbol{w}_{m-1}$ *is denoted as*

$$\mathbb{X}^{\boldsymbol{w}_{m-1}} = \left(X_0^{\boldsymbol{w}_{m-1}}(x), X_1^{\boldsymbol{w}_{m-1}}(x), \ldots, X_{2^{m-1}-1}^{\boldsymbol{w}_{m-1}}(x)\right),$$

where

$$X_i^{\boldsymbol{w}_{m-1}}(x) = \prod_{j=0}^{m-2}\left(\bar{s}_j^{W_j}(x)\right)^{i_j} \tag{65}$$

and

$$\bar{s}_j^{W_j}(x) = \frac{s_j^{W_j}(x)}{s_j^{W_j}(w_{j+1})}. \tag{66}$$

Then $X_{2i}(x) = X_i^{\boldsymbol{w}_{m-1}}(s_1(x))$, *for* $i = 0, 1, \ldots, 2^{m-1} - 1$.

Proof. For simplicity, the subspace polynomial $s_j^{V_j}(x)$ with respective to $V_j = \mathrm{Span}(v_0, \ldots, v_{j-1})$ is denoted as $s_j(x)$. From (8), we have

$$s_j^{W_j}(w_{j+1}) = s_j^{W_j}(s_1(v_{j+1})) = s_{j+1}(v_{j+1}),$$

and thus (66) can be written as

$$\bar{s}_j^{W_j}(x) = \frac{s_j^{W_j}(x)}{s_j^{W_j}(w_{j+1})} = \frac{s_j^{W_j}(x)}{s_{j+1}(v_{j+1})}. \tag{67}$$

By substituting $s_1(x)$ into (67), we have

$$\begin{aligned} \bar{s}_j^{W_j}(s_1(x)) =& \frac{s_j^{W_j}(s_1(x))}{s_{j+1}(v_{j+1})} \\ =& \frac{s_{j+1}(x)}{s_{j+1}(v_{j+1})} \quad \text{(By (8))} \\ =& \bar{s}_{j+1}(x). \end{aligned} \tag{68}$$

By substituting $s_1(x)$ into (65), we have

$$\begin{aligned} X_i^{\boldsymbol{w}_{m-1}}(s_1(x)) &= \prod_{j=0}^{m-2}\left(\bar{s}_j^{W_j}(s_1(x))\right)^{i_j} \\ &= \prod_{j=0}^{m-2}(\bar{s}_{j+1}(x))^{i_j} \qquad \text{(By (68))} \\ &= X_J(x), \end{aligned} \tag{69}$$

where

$$J = i_0 \cdot 2 + i_1 \cdot 2^2 + i_2 \cdot 2^3 + \cdots + i_{m-2} \cdot 2^{m-1}.$$

Thus,

$$J = 2(i_0 + i_1 \cdot 2 + \cdots + i_{m-2} \cdot 2^{m-2}) = 2i,$$

and (69) becomes $X_{2i}(x) = X_i^{\boldsymbol{w}_{m-1}}(s_1(x))$. This completes the proof. □

Assume $D_j(x)$, $j = 0, 1$, in the basis $\mathbb{X}$ is denoted as

$$D_j(x) = \sum_{i=0}^{n/2-1} \bar{d}_{i,j} X_i^w(x) \qquad j = 0, 1, \tag{70}$$

TABLE II
COMPLEXITIES OF n-POINT FFTS OVER $\mathbb{F}_{2^m}$

Algorithm	Restriction	Add. comp.	Multi. comp.
Schönhage [3]		$\mathcal{O}(n\lg(n)\lg\lg(n))$	$\mathcal{O}(n\lg(n))$
Gao and Mateer [7]	m is a power of two	$\mathcal{O}(n\lg(n)\lg\lg(n))$	$\mathcal{O}(n\lg(n))$
Cantor [5]	m is a power of two	$\mathcal{O}(n\lg^{\lg(3)}(n))$	$\mathcal{O}(n\lg(n))$
Gao and Mateer [7]		$\mathcal{O}(n\lg^2(n))$	$\mathcal{O}(n\lg(n))$
Wang and Zhu [4] Gathen and Gerhard [6]		$\mathcal{O}(n\lg^2(n))$	$\mathcal{O}(n\lg^2(n))$
Pollard [1]	m is even	$\mathcal{O}(n^{3/2})$	$\mathcal{O}(n^{3/2})$
Wang and Yan [24]		$\mathcal{O}(n^2/\lg^{\lg(8/3)}(n))$	$\mathcal{O}(n\lg^{\lg(3/2)}(n))$
Sarwate [25]		$\mathcal{O}(n^2)$	$\mathcal{O}(n\lg(n))$
Naive approach		$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

where w is defined in Lemma 6. Then (62) can be rewritten as

$$\begin{aligned} &D(x)\\ &= \sum_{i=0}^{n/2-1} \bar{d}_{i,0}X_i^w(s_1(x)) + \frac{x}{v_0}\sum_{i=0}^{n/2-1} \bar{d}_{i,1}X_i^{\mathbb{W}}(s_1(x))\\ &= \sum_{i=0}^{n/2-1} \bar{d}_{i,0}X_{2i}(x) + \frac{x}{v_0}\sum_{i=0}^{n/2-1} \bar{d}_{i,1}X_{2i}(x) \qquad \text{(By Lemma 6)}\\ &= \sum_{i=0}^{n/2-1} \bar{d}_{i,0}X_{2i}(x) + \sum_{i=0}^{n/2-1} \bar{d}_{i,1}X_{2i+1}(x)\\ &= \sum_{i=0}^{n-1} \bar{a}_i X_i(x). \end{aligned} \tag{71}$$

Thus, the coefficients in (70) are the desired results. To convert $D_j(x)$ from the monomial basis (63) to $\mathbb{X}$ in (70), the same approach (61) can be applied recursively to each $D_j(x)$. The method to calculate (61) is addressed as follows.

1) Computation of (61)*:* To solve (61), the Taylor expansion in Appendix B can be applied. In order to reduce multiplications while applying Taylor expansions, we need to twist polynomial $D(x)$. By substituting $x = v_0 \cdot y$ into (61), we have

$$\begin{aligned} D(v_0\cdot y) &= \sum_{i=0}^{n/2-1}(d_{i,0}+d_{i,1}\cdot y)(s_1(v_0\cdot y))^i\\ &= \sum_{i=0}^{n/2-1}(d_{i,0}+d_{i,1}\cdot y)(v_0^2(y+y^2))^i\\ &= \sum_{i=0}^{n/2-1}(d_{i,0}v_0^{2i}+d_{i,1}v_0^{2i}\cdot y)(y+y^2)^i\\ &= D_0(v_0^2(y+y^2)) + y\cdot D_1(v_0^2(y+y^2)), \end{aligned} \tag{72}$$

where

$$\begin{aligned} D_j(v_0^2x) &= \sum_{i=0}^{n/2-1} d_{i,j}(v_0^2x)^i\\ &= \sum_{i=0}^{n/2-1} d'_{i,j}x^i, \qquad i=0,1. \end{aligned} \tag{73}$$

From (72), Taylor expansions can be applied to the twisted polynomial $D(v_0\cdot y) = \sum_{i=0}^{n-1}(d_i v_0^i)y^i$, an the result forms two polynomials $D_0(v_0^2x)$ and $D_1(v_0^2x)$. The detailed steps are summarized in Algorithm 5.

Algorithm 5 First approach of basis conversion algorithm

Input: $\mathbf{B}_2(D,\gamma,n,v)$, where D is the coefficients of $D(\gamma\cdot x)=\sum_{i=0}^{n-1}d_ix^i$, n is a power of two, and v is the basis
Output: $(\bar{d}_0,\bar{d}_1,\ldots,\bar{d}_{n-1})$, such that $D(x) = \sum_{j=0}^{n-1}\bar{d}_jX_j(x)$

1: **if** $n=1$ **then return** d_0
2: **end if**
3: Compute $D(v_0\cdot x)\leftarrow\sum_{i=0}^{n-1}d_i\cdot(v_0/\gamma)^i\cdot x^i$
4: Call Taylor expansion (Appendix B) to find out $\{(d'_{i,0},d'_{i,1})\}_{i=0}^{n/2-1}$ such that

$$D(v_0\cdot x)\leftarrow\sum_{i=0}^{n/2-1}(d'_{i,0}+d'_{i,1}\cdot y)(x+x^2)^i$$

5: Obtain $D_0(v_0^2x)$ and $D_1(v_0^2x)$ as in (73)
6: Compute w as in (64)
7: Call

$$\begin{aligned} U_0 &\leftarrow \mathbf{B}_2(D_0,v_0^2,n/2,\bar{v})\\ U_1 &\leftarrow \mathbf{B}_2(D_1,v_0^2,n/2,\bar{v}), \end{aligned}$$

to obtain

$$U_i=(u_{0,i},u_{1,i},\ldots,u_{n/2-1,i}), \qquad \forall i=0,1$$

return $(u_{0,0},u_{0,1},u_{1,0},u_{1,1},\ldots,u_{n/2-1,0},u_{n/2-1,1})$

2) Complexity: Based on Appendix B and the above discussions, computing (61) requires $\mathcal{O}(n\lg(n))$ additions and $\mathcal{O}(n)$ multiplications. This leads the recurrence relation

$$A(n)=2\cdot A(n/2)+\mathcal{O}(n\lg(n)), \qquad M(n)=2\cdot M(n/2)+\mathcal{O}(n).$$

Thus $A(h)=\mathcal{O}(n\lg^2(n))$ and $M(h)=\mathcal{O}(n\lg(n))$.

B. Second approach (For m a power of two)

This approach is based on Canter basis [5]. Thus, upon describing the algorithm, the definition as well as the properties of the Cantor basis is addressed below.

Definition 2 (Cantor basis [5]). *For m a power of 2, the Cantor basis $C = (c_0, c_1, \ldots, c_{m-1})$, where each $c_i \in \mathbb{F}_{2^m}$, is constructed by following*

$$\begin{aligned} c_0 &= 1; \\ c_i^2 - c_i &= c_{i-1} \qquad i = 1, 2, \ldots, m-1. \end{aligned} \tag{74}$$

Let $s_i^{\mathrm{C}}(x)$ denote the subspace polynomial in the Cantor basis C. $s_i^{\mathrm{C}}(x)$ possesses the following properties.

Lemma 7 ([5]). *(i). Let $s(x) = x^2 - x$. Then $s_i^{\mathrm{C}}(x)$ possesses a recursive form given by*

$$\begin{aligned} s_0^{\mathrm{C}}(x) &= x; \\ s_i^{\mathrm{C}}(x) &= s(s_{i-1}^{\mathrm{C}}(x)) \qquad i = 1, 2, \ldots, m. \end{aligned} \tag{75}$$

(ii). For i a power of 2 and $i \leq m$,

$$s_i^{\mathrm{C}}(x) = x^{2^i} + x. \tag{76}$$

The polynomial basis over the Cantor basis is denoted as

$$\mathbb{X}^{\mathrm{C}} = \left(X_0^{\mathrm{C}}(x), X_1^{\mathrm{C}}(x), \ldots, X_{2^m-1}^{\mathrm{C}}(x)\right),$$

where

$$X_i^{\mathrm{C}}(x) = \prod_{j=0}^{m-1} \left(\frac{s_j^{\mathrm{C}}(x)}{s_j^{\mathrm{C}}(c_j)} \right)^{i_j} = \prod_{j=0}^{m-1} \left(s_j^{\mathrm{C}}(x)\right)^{i_j}. \tag{77}$$

Given $D(x) = \sum_{i=0}^{n-1} a_i x^i$, the objective of the basis conversion is to determine $\{\bar{a}_i\}$ such that $D(x) = \sum_{i=0}^{n-1} \bar{a}_i X_i^{\mathrm{C}}(x)$.

Let $\ell = 2^{\lceil \lg \lg(n) \rceil - 1}$, and $L = 2^\ell$. Given $D(x)$ in the monomial basis, we compute $\{D_i(x)\}_{i=0}^{k-1}$ such that

$$D(x) = \sum_{i=0}^{k-1} D_i(x) (s_\ell^{\mathrm{C}}(x))^i, \tag{78}$$

where $k = n/L$, and each

$$D_i(x) = \sum_{j=0}^{L-1} a_{i,j} x^j \qquad i = 0, 1, \ldots, k-1. \tag{79}$$

From (76), $s_\ell^{\mathrm{C}}(x) = x^L + x$, and hence (78) can be written as

$$D(x) = \sum_{i=0}^{k-1} D_i(x) (x^L + x)^i, \tag{80}$$

that can be computed with the Taylor expansion in Appendix B.

Assume $D_i(x)$ in the basis $\mathbb{X}^{\mathrm{C}}$ is denoted as

$$D_i(x) = \sum_{j=0}^{L-1} \bar{a}_{i,j} X_j^{\mathrm{C}}(x) \qquad i = 0, 1, \ldots, k-1. \tag{81}$$

Then (78) can be written as

$$\begin{aligned} D(x) &= \sum_{i=0}^{k-1} \sum_{j=0}^{L-1} \bar{a}_{i,j} X_j^{\mathrm{C}}(x) (s_\ell^{\mathrm{C}}(x))^i \\ &= \sum_{j=0}^{L-1} X_j^{\mathrm{C}}(x) \sum_{i=0}^{k-1} \bar{a}_{i,j} (s_\ell^{\mathrm{C}}(x))^i \\ &= \sum_{j=0}^{L-1} X_j^{\mathrm{C}}(x) E_j(s_\ell^{\mathrm{C}}(x)), \end{aligned} \tag{82}$$

Algorithm 6 Second approach of basis conversion algorithm

Input: $\mathbf{B}_3(D, n)$, where $D(x) = \sum_{i=0}^{n-1} d_i x^i$, and n is a power of two

Output: $(\bar{d}_0, \bar{d}_1, \ldots, \bar{d}_{n-1})$ such that $D(x) = \sum_{j=0}^{n-1} \bar{d}_j X_j(x)$

1: **if** $n = 1$ **then return** d_0
2: **end if**
3: Let $\ell = 2^{\lceil \lg \lg(n) \rceil - 1}$, and $L = 2^\ell$
4: Call Taylor expansion (Appendix B) to find $\{D_i(x)\}_{i=0}^{k-1}$ such that (80) holds
5: **for** $i = 0, \ldots, k-1$ **do**
6: Call $\bar{D}_i \leftarrow \mathbf{B}_3(D_i, L)$
7: **end for**
8: Obtain $E_j(x)$, $j = 0, 1, \ldots, L-1$, as in (82)
9: **for** $j = 0, \ldots, L-1$ **do**
10: Call $\bar{E}_j \leftarrow \mathbf{B}_3(E_j, k)$
11: **end forreturn** $\{\bar{E}_j\}_{j=0}^{L-1}$, where the order of elements is as in (85)

where each

$$E_j(x) = \sum_{i=0}^{k-1} \bar{a}_{i,j} x^i \qquad j = 0, 1, \ldots, L-1. \tag{83}$$

Assume $E_j(x)$ in the basis $\mathbb{X}^{\mathrm{C}}$ is denoted as

$$E_j(x) = \sum_{i=0}^{k-1} \bar{e}_{i,j} X_i^{\mathrm{C}}(x) \qquad j = 0, 1, \ldots, L-1. \tag{84}$$

Then (82) can be written as

$$\begin{aligned} D(x) &= \sum_{j=0}^{L-1} X_j^{\mathrm{C}}(x) \sum_{i=0}^{k-1} \bar{e}_{i,j} X_i^{\mathrm{C}}(s_\ell^{\mathrm{C}}(x)) \\ &= \sum_{j=0}^{L-1} X_j^{\mathrm{C}}(x) \sum_{i=0}^{k-1} \bar{e}_{i,j} X_{i \cdot L}^{\mathrm{C}}(x) \\ &= \sum_{j=0}^{L-1} \sum_{i=0}^{k-1} \bar{e}_{i,j} X_{i \cdot L + j}^{\mathrm{C}}(x), \end{aligned} \tag{85}$$

that is the desired result.

In this approach, we have to convert the basis of $D_i(x)$ (see (79) and (81)) and $E_j(x)$ (see (83) and (84)). Both conversions can be finished by applying the approach recursively.

1) Complexity: Since the approach only takes additions, the number of multiplications is zero ($M(n) = 0$). The recurrence relation is formulated as

$$A(n) = \frac{n}{L} \cdot A(L) + L \cdot A(\frac{n}{L}) + \mathcal{O}(n \lg(\frac{n}{L})). \tag{86}$$

Since it is not straightforward to see the close form of $A(n)$, the details are addressed in Appendix C, thereby the result $A(n) = \mathcal{O}(n \lg(n) \lg \lg(n))$.

C. Inverse algorithm

The inverse algorithm is the approach converting the basis of the given polynomial $D(x)$ from $\mathbb{X}$ to the monomial basis. It is straightforward to devise the inverse algorithm by backtracking

the steps of the proposed basis conversion approaches. In the following, the inverse algorithms for the two approaches are described. In general, the inverse algorithm has the same complexity with the corresponding conversion approach, and thus we do not take much space to describe those inverse algorithms.

For the inversion of the second approach, we want to solve (61) in each iteration, where the set of coefficients $\{(d_{i,0}, d_{i,1})\}_{i=0}^{n/2-1}$ is known, and the objective is to calculate $D(x) = \sum_{i=0}^{n-1} d_i x^i$. (61) can be converted to a form of Taylor expansion with $\mathcal{O}(n)$ multiplications. Then the inverse Taylor expansion, that is presented in Appendix B, is employed.

For the inversion of the third approach, we want to solve (78) in each iteration, where $\{D_i(x)\}_{i=0}^{k-1}$ is known, and the objective is to compute $D(x)$. With the inverse algorithm of Taylor expansion, (78) can be solved with $\mathcal{O}(n \lg(n/L))$ operations. The complexity of this approach takes $\mathcal{O}(n \lg(n) \lg\lg(n))$, the same as the conversion algorithm.

IX. Conclusions

In this paper, we proposed an additive FFT over extension binary fields (as well as fields of constant characteristic p), where the input polynomial is represented in a new basis. Based on the proposed FFTs, the encoding/erasure decoding algorithms for Reed-Solomon codes are proposed. The encoding is in $\mathcal{O}(n \lg(k))$ field operations, and the erasure decoding is in $\mathcal{O}(n \lg(n))$ field operations. The basis conversion approaches are also proposed. In particular, for arbitrary m, the conversion algorithms require $\mathcal{O}(n \lg^2(n))$ field operations. For m a power of two, the complexity is $\mathcal{O}(n \lg(n) \lg\lg(n))$.

Appendix A
Evaluating error-locator polynomials with fast Walsh-Hadamard transforms

In [12], Didier presented an efficient algorithm to compute (54) and (55). The method is presented here for the purpose of completeness. The formal derivative of $\Pi(x)$ is given by

$$\Pi'(x) = \sum_{j \in R} \prod_{y \in R, y \neq j} (x + y).$$

By substituting $x = j \in R$ into $\Pi'(x)$, we have

$$\Pi'(j) = \prod_{y \in R, y \neq j} (j + y) = \prod_{y \in \mathbb{F}_{2^m}, y \neq j} (j + y)^{R_y}, \qquad (87)$$

where $\{R_y | y \in \mathbb{F}_{2^m}\}$ is defined as

$$R_y = \begin{cases} 1 & \text{if } y \in R; \\ 0 & \text{otherwise.} \end{cases} \qquad (88)$$

Let $\mathrm{Log}(x)$ denote the discrete logarithm function of $\mathbb{F}_{2^m}^*$, where $\mathbb{F}_{2^m}^*$ contains all nonzero elements in $\mathbb{F}_{2^m}$. Precisely, for each $i \in \mathbb{F}_{2^m}^*$, we have $\mathrm{Log}(i) = j$ iff $i = \alpha^j$, where α is a primitive element of $\mathbb{F}_{2^m}^*$. Then (87) can be reformulated as

$$\mathrm{Log}(\Pi'(j)) = \biguplus_{y \in \mathbb{F}_{2^m}, y \neq j} R_y \mathrm{Log}(j + y), \qquad \forall j \in R.$$

Note that the symbol $\biguplus$ means the summation with normal additions, rather than the additions in extension binary fields. By letting $\mathrm{Log}(0) = 0$, the above equation can be rewritten as

$$\mathrm{Log}(\Pi'(j)) = \biguplus_{y \in \mathbb{F}_{2^m}} R_y \mathrm{Log}(j + y), \qquad \forall j \in R. \qquad (89)$$

Then we consider the construction of Π. In the same way, the elements of Π can be formulated as

$$\mathrm{Log}(\Pi(j)) = \biguplus_{y \in \mathbb{F}_{2^m}} R_y \mathrm{Log}(j + y), \qquad \forall j \in \mathbb{F}_{2^m} \setminus R. \qquad (90)$$

From (89) and (90), we have

$$\biguplus_{y \in \mathbb{F}_{2^m}} R_y \mathrm{Log}(j + y) = \begin{cases} \mathrm{Log}(\Pi'(j)) & \text{if } j \in R; \\ \mathrm{Log}(\Pi(j)) & \text{otherwise.} \end{cases} \qquad (91)$$

In (91), $+$ is the addition in $\mathbb{F}_{2^m}$ and it can be treated as exclusive-or operation. Hence, (91) is namely the logical convolution [26][27] that can be efficiently computed with fast Walsh-Hadamard transforms [28]. The steps of the algorithm are elaborated as follows.

Let $\mathrm{FWT}(\bullet)$ denote the h-point fast Walsh-Hadamard transform (FWHT). An h-point FWHT requires $h \lg(h)$ additions. Define

$$\tilde{R} = (R_0, R_1, \ldots, R_{2^m-1}),$$

$$\tilde{L} = (0, \mathrm{Log}(\omega_1), \mathrm{Log}(\omega_2), \ldots, \mathrm{Log}(\omega_{2^m-1})).$$

(91) is computed by

$$R_{\mathrm{w}} = \mathrm{FWT}(\mathrm{FWT}(\tilde{R}) \times \mathrm{FWT}(\tilde{L})), \qquad (92)$$

where $\times$ denotes pairwise integer multiplication. Notably, $\mathrm{FWT}(\tilde{L})$ can be precomputed and stored, and thus (92) can be computed with performing two fast Walsh transforms of length 2^r. We remark that all the above computation can be performed over modulo $2^m - 1$. Also note that R_{w} is the logarithm of the desired values, and thus the exponent for each element is computed. In summary, the algorithm requires $\mathcal{O}(n \lg(n))$ modulus additions, $\mathcal{O}(n)$ modulus multiplications, and $\mathcal{O}(n)$ exponentiations for $n = 2^m$.

Appendix B
Taylor expansion

Given $D(x) = \sum_{i=0}^{n-1} a_i x^i \in \mathbb{F}_{2^m}[x]$ and an integer $t > 1$, [7] introduced an algorithm to find $\{\bar{a}_{0,i}, \bar{a}_{1,i}\}_{i=0}^{m-1}$ such that

$$D(x) = \sum_{i=0}^{m-1} (\bar{a}_{0,i} + \bar{a}_{1,i} x)(x + x^t)^i,$$

where $m = \lceil n/t \rceil$. Firstly, $D(x)$ is divided with $(x + x^t)^k$ to obtain

$$D(x) = D_0(x) + (x + x^t)^k D_1(x), \qquad (93)$$

with $k = 2^{\lceil \lg(n/t) \rceil - 1}$. Then the polynomial division is recursively applied to $D_0(x)$ and $D_1(x)$, until $k = 0$. Clearly, the result is the desired output.

To perform the division, the following identity over $\mathbb{F}_{2^m}$ is utilized:

$$(x + x^t)^k = x^k + x^{tk}.$$

Thus, the division in (93) requires only $\mathcal{O}(n)$ additions. Hence, the Taylor expansion takes a total of $\mathcal{O}(n \lg(n/t))$ field additions.

The inverse approach is straightforward by backtracking the original algorithm. In (93), we have $D_0(x)$ and $D_1(x)$, and the objective is to calculate $D(x)$, that requires $\mathcal{O}(n)$ additions. Thus the inverse of Taylor expansion also requires $\mathcal{O}(n \lg(n/t))$ field additions.

APPENDIX C
COMPLEXITY OF THE SECOND APPROACH OF THE BASIS CONVERSION

In this appendix, the close form of $A(n)$ in (86) is derived, for $n = 2^N$. The complexity analysis consists of two parts. The first part discusses the case for N a power of two. In the second part, the case for arbitrary N is considered. recall that $L = 2^\ell$, where $\ell = 2^{\lceil \lg \lg(n) \rceil - 1}$. Hence, for N a power of two, we have $L = n/L = \sqrt{n}$. (86) can then be formulated as

$$A(n) = 2\sqrt{n} \cdot A(\sqrt{n}) + \mathcal{O}(n \lg(n)).$$

By induction, it can be seen that $A(n) = \mathcal{O}(n \lg(n) \lg \lg(n))$.

The case for arbitrary N is considered below. Since $L = 2^\ell$, where $\ell = 2^{\lceil \lg \lg(n) \rceil - 1}$ is a power of two, we know $A(L) = \mathcal{O}(L \lg(L) \lg \lg(L))$, and $L \geq n/L$. That can be substituted into (86) to obtain

$$\begin{aligned} &A(n) = \mathcal{O}(n \lg(L) \lg \lg(L)) + L \cdot A(\frac{n}{L}) + \mathcal{O}(n \lg(\frac{n}{L})) \\ \Rightarrow &A(n) = L \cdot A(\frac{n}{L}) + \mathcal{O}(n \lg(L) \lg \lg(L)). \end{aligned} \tag{94}$$

To prove the complexity, assume the big-O term in (94) is smaller than

$$\mathcal{O}(n \lg(L) \lg \lg(L)) \leq c_0 n \lg(L) \lg \lg(L),$$

with a constant c_0. Further, assume $A(n) \leq c_1 \cdot n \lg(n) \lg \lg(n)$, for c_1 a constant and $c_1 \geq c_0$. Then (94) gives

$$\begin{aligned} &L \cdot A(\frac{n}{L}) + \mathcal{O}(n \lg(L) \lg \lg(L)) \\ \leq &c_1 \cdot n \lg(\frac{n}{L}) \lg \lg(\frac{n}{L}) + c_0 \cdot n \lg(L) \lg \lg(L) \\ \leq &c_1 \cdot n \lg(\frac{n}{L}) \lg \lg(L) + c_0 \cdot n \lg(L) \lg \lg(L) \\ = &c_1 \cdot n \lg(n) \lg \lg(L) - c_1 \cdot n \lg(L) \lg \lg(L) \\ &+ c_0 \cdot n \lg(L) \lg \lg(L) \\ \leq &c_1 \cdot n \lg(n) \lg \lg(L) \qquad (\text{As } c_1 \geq c_0) \\ \leq &A(n), \end{aligned} \tag{95}$$

and this proofs the assumption.

APPENDIX D
POLYNOMIAL BASIS OVER A FINITE FIELD OF CHARACTERISTIC p

In this appendix, we extend the basis to finite fields of characteristic p. The algorithm is similar to the approach shown in Section III. The polynomial basis is for $\mathbb{F}_{p^r}[x]/(x^{p^r} - x)$, such that additive FFT in this basis leads to $\mathcal{O}(n \cdot p \log_p(n))$ field operations. By fixing p a constant, the complexity can be written as $\mathcal{O}(n \log(n))$. The basis is based on the subspace polynomial over a finite field of characteristic p.

Let $v = (v_0, v_1, \ldots, v_{m-1})$ denote a basis of $\mathbb{F}_{p^m}$. The proposed basis $\mathbb{X} = (X_0(x), X_1(x), \ldots, X_{p^m-1}(x))$ is respectively defined as

$$X_i(x) = \prod_{j=0}^{k-1} (s_j(x))^{i_j}, \tag{96}$$

where i is a base-p integer expressed as

$$i = i_0 + i_1 \cdot p + \cdots + i_{m-1} \cdot p^{m-1}, \qquad 0 \leq i_j \leq p-1. \tag{97}$$

Notice that $(s(x))^0 = 1$, and $\deg(X_i(x)) = i$.

For any $D(x) \in \mathbb{F}_{p^m}[x]/(x^{p^m} - x)$, it can be represented in the basis $\mathbb{X}$ given by

$$D(x) = \sum_{i=0}^{p^m-1} a_i X_i(x).$$

In the following, we propose an $\mathcal{O}(h \cdot p \log_p(h))$ algorithm to compute $\{D(a+\beta) | a \in V_k\}$, where $\deg(D(x)) < h = p^k$, and $\beta \in \mathbb{F}_{p^m}$.

Given a polynomial $D(x) = \sum_{i=0}^{2^k-1} d_i X_i(x)$ of degree $\deg(D(x)) < 2^k = h$ in the basis $\mathbb{X}$ and $\beta \in \mathbb{F}_{2^m}$, this section presents a algorithm to compute $\{D(\omega) | \omega \in V_k + \beta\}$.

A. Recursive structure in polynomial basis

The polynomial $D(x)$ can be formulated as a recursive function. Let

$$D(x) = \Delta_0^0(x).$$

The recursive function is defined as

$$\Delta_i^r(x) = \sum_{j=0}^{p-1} (s_i(x))^j \Delta_{i+1}^{r+p^{ij}}(x), \text{ for } 0 \leq i \leq k-1; \tag{98}$$

$$\Delta_k^r(x) = d_r, \text{ for } 0 \leq r \leq 2^k - 1. \tag{99}$$

By induction, the coefficients of $\Delta_i^r(x)$ are denoted as

$$\Delta_i^r = \{d_{j \cdot p^i + r} | j = 0, \ldots, p^{k-i} - 1)\}. \tag{100}$$

$\Delta_i^m(x)$ possesses the following equality that will be utilized in the algorithm:

Lemma 8. *$\forall a \in V_m = \mathbb{F}_{p^m}$ and $\forall b \in V_i$, $0 \leq i \leq k-1$,*

$$\Delta_i^r(a+b) = \Delta_i^r(a). \tag{101}$$

Proof. By definition, $\bar{s}_i(b) = 0$, $\forall b \in V_i$. From Lemma 2, we have

$$s_i(a+b) = s_i(a) + s_i(b) = s_i(a), \qquad \forall b \in V_i. \tag{102}$$

The proof is based on the mathematical induction on i. In the base case $i = k-1$, (98) can be written as

$$\begin{aligned} &\Delta_{k-1}^r(x) \\ = &\sum_{j=0}^{p-1} (s_{k-1}(x))^j \Delta_k^{r+p^{(k-1)j}}(x) \\ = &\sum_{j=0}^{p-1} (s_{k-1}(x))^j d_{r+p^{(k-1)j}}. \end{aligned} \tag{103}$$

From (102), we have

$$\begin{aligned}&\Delta_{k-1}^r(a+b)\\ =&\sum_{j=0}^{p-1}(s_{k-1}(a+b))^j d_{r+p^{(k-1)j}}\\ =&\sum_{j=0}^{p-1}(s_{k-1}(a))^j d_{r+p^{(k-1)j}}\\ =&\Delta_{k-1}^r(a), \qquad \forall b\in V_{k-1}.\end{aligned} \tag{104}$$

Thus (101) holds for $i=k-1$. Assume (101) holds for $i=\ell+1$. When $i=\ell$, we have

$$\begin{aligned}&\Delta_{\ell}^r(a+b)\\ =&\sum_{j=0}^{p-1}(s_{\ell}(a+b))^j \Delta_{\ell+1}^{r+p^{\ell j}}(a+b)\\ =&\sum_{j=0}^{p-1}(s_{\ell}(a))^j \Delta_{\ell+1}^{r+p^{\ell j}}(a)\\ =&\Delta_{\ell}^r(a), \qquad \forall b\in V_{\ell}.\end{aligned} \tag{105}$$

This completes the proof. □

B. Proposed algorithm

Let

$$V_j^k=\mathrm{Span}(v_j,v_{j+1},\ldots,v_{k-1}), \qquad 0\le j\le k\le m-1, \tag{106}$$

denote an $(k-j)$-dimensional subspace in $\mathbb{F}_{p^m}$. Let

$$\Psi_\beta(i,r)=\{\Delta_i^r(\omega)|\omega\in V_i^k+\beta\}, \qquad i=0,\ldots,k-1; \tag{107}$$

$$\Psi_\beta(k,r)=\{d_r\}. \tag{108}$$

The objective of algorithm is to find out $\Psi_\beta(0,0)$. In the following, we rearrange $\Psi_\beta(i,r)$ into p parts $\Psi_\beta(i+1,r+p^{ij})$, for $j=0,1,\ldots,p-1$.

As

$$V_i^k=V_{i+1}^k\cup(V_{i+1}^k+v_i),$$

(107) can be divided into p individual subsets

$$\Psi_\beta^{(j)}(i,r)=\{\Delta_i^r(\omega+j\cdot v_i)|\omega\in V_{i+1}^k+\beta\} \qquad j=0,1,\ldots,p-1. \tag{109}$$

To evaluate each element of $\Psi_\beta^{(j)}(i,r)$, from (98), we have

$$\begin{aligned}&\Delta_i^r(\omega+j\cdot v_i)\\ =&\sum_{j=0}^{p-1}(s_i(\omega+j\cdot v_i))^j \Delta_{i+1}^{r+p^{ij}}(\omega+j\cdot v_i)\\ =&\sum_{j=0}^{p-1}(s_i(\omega+j\cdot v_i))^j \Delta_{i+1}^{r+p^{ij}}(\omega) \qquad \forall\omega\in V_{i+1}^k+\beta.\end{aligned} \tag{110}$$

It can be seen that $\Delta_{i+1}^{r+p^{ij}}(\omega)\in\Psi_\beta(i+1,r+p^{ij})$, for $j=0,1,\ldots,p-1$. Hence, for each element of $\Psi_\beta^{(0)}(i,r)$, the calculation requires $p-1$ multiplications and $p-1$ additions.

Let $A(h)$ and $M(h)$ respectively denote the number of additions and multiplications used in the algorithm. The recurrence relation is formulated as

$$\begin{aligned}A(h)&=p\times A(h/p)+\mathcal{O}(hp);\\ M(h)&=p\times M(h/p)+\mathcal{O}(hp).\end{aligned} \tag{111}$$

The solution is given by

$$A(h)=M(h)=\mathcal{O}(hp\log_p(h)).$$

REFERENCES

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

[2] L. I. Bluestein, "A linear filtering approach to the computation of discrete fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. 18, no. 4, pp. 451–455, Dec 1970.

[3] A. Schönhage, "Schnelle multiplikation von polynomen über körpern der charakteristik 2," *Acta Informatica*, vol. 7, no. 4, pp. 395–398, 1977. [Online]. Available: http://dx.doi.org/10.1007/BF00289470

[4] Y. Wang and X. Zhu, "A fast algorithm for the fourier transform over finite fields and its VLSI implementation," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 3, pp. 572–577, Apr 1988.

[5] D. G. Cantor, "On arithmetical algorithms over finite fields," *Journal of Combinatorial Theory, Series A*, vol. 50, no. 2, pp. 285–300, 1989.

[6] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomial over F_2," in *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, Zurich, Switzerland, 1996, pp. 1–9.

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

[8] S.-J. Lin, T. Y. Al-Naffouri, and Y. S. Han, "Fft algorithm for binary extension finite fields and its application to reed-solomon codes," *IEEE Trans. Inf. Theory*, submitted manuscript under review.

[9] O. Ore, "On a special class of polynomials," *Trans. Amer. Math. Soc.*, vol. 35, no. 11, pp. 559–584, Nov 1933.

[10] ——, "Contributions to the theory of finite fields," *Trans. Amer. Math. Soc.*, vol. 36, no. 2, pp. 243–274, Apr 1934.

[11] S. J. Lin and W. H. Chung, "An efficient (n, k) information dispersal algorithm based on fermat number transforms," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 8, pp. 1371–1383, 2013.

[12] F. Didier, "Efficient erasure decoding of reed-solomon codes," *CoRR*, vol. abs/0901.1886, 2009.

[13] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[14] J. Justesen, "On the complexity of decoding Reed-Solomon codes (corresp.)," *IEEE Trans. Inf. Theory*, vol. 22, no. 2, pp. 237–238, Mar 1976.

[15] R. Blahut, "A recursive berlekamp-massey algorithm," in *Theory and practice of error control codes*. Boston: Addison-Wesley, 1983, ch. 11.7, pp. 336–340.

[16] F. MacWilliams and N. Sloane, "Generalized bch codes," in *The Theory of Error-correcting Codes*. Oxford: North-Holland Publishing Company, 1977, ch. 12, pp. 332–369.

[17] I. S. Reed, T. K. Truong, and L. R. Welch, "The fast decoding of reed-solomon codes using number theoretic transforms," in *The Deep Space Network 42-35*, Jet Propulsion Laboratory, Pasadena, CA, July 1976, pp. 64–78.

[18] I. S. Reed, R. Scholtz, T.-K. Truong, and L. Welch, "The fast decoding of reed-solomon codes using fermat theoretic transforms and continued fractions," *IEEE Trans. Inf. Theory*, vol. 24, no. 1, pp. 100–106, Jan 1978.

[19] A. Sora and J. Lacan, "FNT-based reed-solomon erasure codes," in *Proceedings of the 7th IEEE Conference on Consumer Communications and Networking Conference*, Las Vegas, Nevada, USA, 2010, pp. 466–470.

[20] S. J. Lin and W. H. Chung, "An efficient (n, k) information dispersal algorithm for high code rate system over fermat fields," *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 2036–2039, December 2012.

[21] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 56–67, Oct. 1998.

[22] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–280.

[23] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, June 2006.

[24] X. Wu, Y. Wang, and Z. Yan, "On algorithms and complexities of cyclotomic fast fourier transforms over arbitrary finite fields," *IEEE Trans. Signal Process.*, vol. 60, no. 3, pp. 1149–1158, March 2012.

[25] D. Sarwate, "Semi-fast fourier transforms over $\mathrm{GF}(2^m)$." *IEEE Trans. Comput.*, vol. C-27, no. 3, pp. 283–285, March 1978.

[26] J. E. Gibbs and F. Pichler, "Comments on transformation of "fourier" power spectra into "walsh" power spectra," *IEEE Trans. Audio Electroacoust.*, vol. EMC-13, no. 3, pp. 51–54, Aug 1971.

[27] G. Robinson, "Logical convolution and discrete walsh and fourier power spectra," *IEEE Trans. Audio Electroacoust.*, vol. 20, no. 4, pp. 271–280, Oct 1972.

[28] B. Fino and V. Algazi, "Unified matrix treatment of the fast walsh-hadamard transform," *IEEE Trans. Comput.*, vol. C-25, no. 11, pp. 1142–1146, Nov 1976.

Sian-Jheng Lin (M'16) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2004, 2006, and 2010, respectively. From 2010 to 2014, he was a postdoc with the Research Center for Information Technology Innovation, Academia Sinica. From 2014 to 2016, He was a postdoc with the Electrial Engineering Department at King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia. He was a part-time lecturer at Yuanpei University from 2007 to 2008, and at Hsuan Chuang University From 2008 to 2010. He is currently a project researcher with the School of Information Science and Technology at University of Science and Technology of China (USTC), Hefei, China. In recent years, his research focus on the algorithms of MDS codes and its applications to storage systems.

Tareq Al-Naffouri (M'10) received the B.S. degrees in mathematics and electrical engineering (with first honors) from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, the M.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, in 1998, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2004.

He was a visiting scholar at California Institute of Technology, Pasadena, CA, from January to August 2005 and during summer 2006. He was a Fulbright scholar at the University of Southern California from February to September 2008. He has held internship positions at NEC Research Labs, Tokyo, Japan, in 1998, Adaptive Systems Lab, University of California at Los Angeles in 1999, National Semiconductor, Santa Clara, CA, in 2001 and 2002, and Beceem Communications Santa Clara, CA, in 2004. He is currently an Associate at the Electrical Engineering Department, King Abdullah University of Science and Technology (KAUST). His research interests lie in the areas of sparse, adaptive, and statistical signal processing and their applications and in network information theory. He has over 150 publications in journal and conference proceedings, 9 standard contributions, 10 issued patents, and 6 pending.

Dr. Al-Naffouri is the recipient of the IEEE Education Society Chapter Achievement Award in 2008 and Al-Marai Award for innovative research in communication in 2009. Dr. Al-Naffouri has also been serving as an Associate Editor of Transactions on Signal Processing since August 2013.

Yunghsiang S. Han (S'90-M'93-SM'08-F'11) was born in Taipei, Taiwan, 1962. He received B.Sc. and M.Sc. degrees in electrical engineering from the National Tsing Hua University, Hsinchu, Taiwan, in 1984 and 1986, respectively, and a Ph.D. degree from the School of Computer and Information Science, Syracuse University, Syracuse, NY, in 1993. He was from 1986 to 1988 a lecturer at Ming-Hsin Engineering College, Hsinchu, Taiwan. He was a teaching assistant from 1989 to 1992, and a research associate in the School of Computer and Information Science, Syracuse University from 1992 to 1993. He was, from 1993 to 1997, an Associate Professor in the Department of Electronic Engineering at Hua Fan College of Humanities and Technology, Taipei Hsien, Taiwan. He was with the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou, Taiwan from 1997 to 2004. He was promoted to Professor in 1998. He was a visiting scholar in the Department of Electrical Engineering at University of Hawaii at Manoa, HI from June to October 2001, the SUPRIA visiting research scholar in the Department of Electrical Engineering and Computer Science and CASE center at Syracuse University, NY from September 2002 to January 2004 and July 2012 to June 2013, and the visiting scholar in the Department of Electrical and Computer Engineering at University of Texas at Austin, TX from August 2008 to June 2009. He was with the Graduate Institute of Communication Engineering at National Taipei University, Taipei, Taiwan from August 2004 to July 2010. From August 2010, he is with the Department of Electrical Engineering at National Taiwan University of Science and Technology as Chair Professor. He is also a Chair Professor at National Taipei University from February 2015. His research interests are in error-control coding, wireless networks, and security.

Dr. Han was a winner of the 1994 Syracuse University Doctoral Prize and a Fellow of IEEE. One of his papers won the prestigious 2013 ACM CCS Test-of-Time Award in cybersecurity.

Wei-Ho Chung received the B.Sc. and M.Sc. degrees in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in Electrical Engineering from the University of California, Los Angeles, in 2009. From 2002 to 2005, he was a system engineer at ChungHwa Telecommunications Company, where he worked on data networks. In 2008, he worked on CDMA systems at Qualcomm, Inc., San Diego, CA. His research interests include communications, signal processing, and networks. Dr. Chung received the Taiwan Merit Scholarship from 2005 to 2009 and the Best Paper Award in IEEE WCNC 2012, and has published over 40 journal articles and over 50 conference papers. Since January 2010, Dr. Chung has been an assistant research fellow, and promoted to the rank of associate research fellow in January 2014 in Academia Sinica. He leads the Wireless Communications Lab in the Research Center for Information Technology Innovation, Academia Sinica, Taiwan.