FFT Algorithm for Binary Extension Finite Fields and its Application to Reed-Solomon Codes

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

Abstract—Recently, a new polynomial basis over binary extension fields was proposed such that the fast Fourier transform (FFT) over such fields can be computed in the complexity of order $\mathcal{O}(n \lg(n))$, where n is the number of points evaluated in FFT. In this work, we reformulate this FFT algorithm such that it can be easier understood and be extended to develop frequency-domain decoding algorithms for $(n = 2^m, k)$ systematic Reed-Solomon (RS) codes over $\mathbb{F}_{2^m}, m \in \mathbb{Z}^+$, with $n - k$ a power of two. First, the basis of syndrome polynomials is reformulated in the decoding procedure so that the new transforms can be applied to the decoding procedure. A fast extended Euclidean algorithm is developed to determine the error locator polynomial. The computational complexity of the proposed decoding algorithm is $\mathcal{O}(n \lg(n - k) + (n - k) \lg^2(n - k))$, improving upon the best currently available decoding complexity $\mathcal{O}(n \lg^2(n) \lg \lg(n))$, and reaching the best known complexity bound that was established by Justesen in 1976. However, Justesen's approach is only for the codes over some specific fields, which can apply Cooley-Tucky FFTs. As revealed by the computer simulations, the proposed decoding algorithm is 50 times faster than the conventional one for the $(2^{16}, 2^{15})$ RS code over $\mathbb{F}_{2^{16}}$.

I. INTRODUCTION

Reed-Solomon (RS) codes are a class of block error-correcting codes that were invented by Reed and Solomon [1] in 1960. An (n, k) RS code is constructed over $\mathbb{F}_q$, for $n = q - 1$. Its extended version, called extended Reed-Solomon codes [2], admits a codeword length of up to $n = q$ or $n = q + 1$. The systematic version of (n, k) RS code appends $n - k$ parity symbols to the k message symbols, forming a codeword of length n. RS codes are maximum distance separable (MDS). (n, k) RS codes can correct up to $\lfloor (n - k)/2 \rfloor$ erroneous symbols. Nowadays, RS codes have numerous important applications, including barcodes (such as QR codes), storage devices (such as Blu-ray Discs), digital television (such as DVB and ATSC), and data transmission technologies (such as DSL and WiMAX). RS codes are also used to design other forward error correction codes, such as

regenerating codes [3][4] and local reconstruction codes [5]. The wide range of applications of RS codes raises an important issue concerning their computational complexity. More specifically, since the practical implementations of RS codes are typically over binary extension finite fields, the complexity of RS codes over those fields has received more attentions than that over others [6][7].

The conventional syndrome-based RS decoding algorithm has quadratic complexities. Some fast approaches [8][9] are based on FFTs or fast polynomial arithmetic techniques. However, the structures of FFTs over finite fields vary with the sizes of fields $\mathbb{F}_q$. When $q - 1$ is a smooth number, meaning that $q - 1$ can be factorized into many small primes, the Cooley-Tucky FFT in $\mathcal{O}(n \lg(n))$ field additions and field multiplications can be applied. A conventional case involves choosing Fermat primes $q \in \{2^m + 1 | m = 1, 2, 4, 8, 16\}$. Based on such FFTs, Justesen [8] gave an $\mathcal{O}(n \lg^2(n))$ approach for decoding (n, k) RS code over $\mathbb{F}_{2^m+1}$. Another approach to solve the key equations of BCH codes was proposed by Pan [10], and it reduces a factor of $\lg n$ when the characteristic of the field is large enough. However, the algorithm [10] does not have improvement for the codes over binary extension fields. If $q - 1$ is not smooth, Cooley-Tucky FFTs are inapplicable. In this case, the FFTs over arbitrary fields [11][12] can be applied and it requires $\mathcal{O}(n \lg(n) \lg \lg(n))$ field operations. Gao [9] presented an $\mathcal{O}(n \lg^2(n) \lg \lg(n))$ RS decoding algorithm over arbitrary fields, by utilizing fast polynomial multiplications [13]. Further, for the codes over $\mathbb{F}_{2^m}$, the additive FFT [14], that requires $\mathcal{O}(n \lg(n) \lg \lg(n))$ operations, can be applied to reduce the leading constant further. To authors' knowledge, the additive FFT [14] is the fastest algorithm over $\mathbb{F}_{2^m}$ so far.

As RS codes are typically constructed over binary extension fields, we consider this case in this paper. Clearly, if one wants to remove the extra factor $\lg \lg(n)$ in the RS algorithms over binary extension fields, the FFTs in $\mathcal{O}(n \lg(n))$ are required. Recently, Lin et al. [15] showed a new way to solve aforementioned FFT problem. The paper [15] defined a new polynomial basis based on subspace polynomials over $\mathbb{F}_{2^m}$. For a polynomial of degree less than h in this new basis, the h-point multipoint evaluations can be made in $\mathcal{O}(h \lg(h))$ field operations. Based on the multipoint evaluation algorithm, encoding/erasure decoding algorithms for (n, k) RS codes [15] were proposed to achieve $\mathcal{O}(n \lg(n))$. However, the error-correction RS decoding algorithm based on the new basis was not yet provided.

This paper develops an error correction decoding algorithm for $(n = 2^m, k)$ RS codes over $\mathbb{F}_{2^m}$, for $k/n \geq 0.5$ and $(n-k)$

a power of two.[1] In practice RS codes usually have rates $k/n \geq 0.5$. The complexity of the proposed algorithm is given by $\mathcal{O}(n \lg(n-k) + (n-k) \lg^2(n-k))$. Holding constant the code rate k/n yields a complexity $\mathcal{O}(n \lg^2(n))$, which is better than the best existing complexity of $\mathcal{O}(n \lg^2(n) \lg \lg(n))$, that was achieved by Gao [9] in 2002. The algorithm is based on the non-standard polynomial basis [15]. To embed the new basis into the decoding algorithm, we reformulate the decoding formulas such that all arithmetics are performed on the new basis. The key equation is solved by the Euclidean algorithm, and thus the fast polynomial divisions, as well as the Euclidean algorithm in the new basis are proposed. Finally, we combine those algorithms, resulting in a fast error-correction RS decoding algorithm. The major contributions of this paper are summarized as follows.

1) An alternative description of the algorithms [15] for the new polynomial basis is presented.
2) An $\mathcal{O}(h \lg(h))$ fast polynomial division in the new basis is derived.
3) An $\mathcal{O}(h \lg^2(h))$ fast half-GCD algorithm in the new basis is presented.
4) An $\mathcal{O}(n \lg(n-k))$ RS encoding algorithm is presented, for $n-k$ a power of two.
5) A syndrome-based RS decoding algorithm that is based on the new basis is demonstrated.
6) An $\mathcal{O}(n \lg(n-k) + (n-k) \lg^2(n-k))$ RS decoding algorithm is presented, for $n-k$ a power of two.

Notably, [15] gave the encoding algorithms for RS codes with the complexity $O(n \lg(k))$, for k a power of two. The encoding algorithm [15] is suitable for coding rate $k/n \leq 0.5$; however, the proposed encoding algorithm in this work is suitable for $k/n \geq 0.5$.

The rest of this paper is organized as follows. Section II reviews the definitions of the polynomial basis. The multi-point evaluation algorithm is provided in Sec. III. Section IV provides an alternative polynomial basis that is constructed using monic polynomials. The polynomial operations that are used in the encoding/decoding of RS codes are explicated. Section V presents the fast extended Euclidean algorithm that is based on the half-GCD method. Section VI and Section VII introduce the algorithms for encoding and decoding RS codes. Section VIII presents simulations and draws conclusions.

II. POLYNOMIAL BASIS IN $\mathbb{F}_{2^m}[x]/x^{2^m} - x$

This section reviews the subspace polynomials over $\mathbb{F}_{2^m}$, and the polynomial basis defined in [15].

A. Subspace polynomial

Let $\mathbb{F}_{2^m}$ denote an extension finite field with dimension m over $\mathbb{F}_2$. Let $\overline{v} = (v_0, v_1, \ldots, v_{m-1})$ denote a basis of $\mathbb{F}_{2^m}$. That is, all $v_i \in \mathbb{F}_{2^m}$ are linearly independent over $\mathbb{F}_2$. A k-dimensional space V_k of $\mathbb{F}_{2^m}$ is defined as

$$V_k = \text{Span}(\overline{v}_k)$$
$$= \{i_0 \cdot v_0 + i_1 \cdot v_1 + \cdots + i_{k-1} \cdot v_{k-1} | \forall i_j \in \{0,1\}\}, \quad (1)$$

[1]There are many (n, k) can be chosen when $n = 2^m$, $k = 2^m - 2^t$, where $t < m$.

where $\overline{v}_k = (v_0, v_1, \ldots, v_{k-1})$ is a basis of space V_k, and $k \leq m$. We can form a strictly ascending chain of subspaces given by

$$\{0\} = V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m = \mathbb{F}_{2^m}.$$

Let $\{\omega_i\}_{i=0}^{2^m-1}$ denote the elements of $\mathbb{F}_{2^m}$. Each element is defined as

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

where $i_j \in \{0, 1\}$ is the binary representation of i. That is,

$$i = i_0 + i_1 \cdot 2 + \cdots + i_{m-1} \cdot 2^{m-1}, \forall i_j \in \{0, 1\}.$$

This implies that $V_k = \{\omega_i\}_{i=0}^{2^k-1}$, for $k = 0, 1, \ldots, m$. Note that $\omega_0 = 0$ is the additive identity in the filed. In this work, ω_0 and 0 will be used interchangeably when there is no confusion.

The subspace polynomial [14, 16, 17] of V_k is defined as

$$s_k(x) = \prod_{a \in V_k} (x - a), \quad (2)$$

and it is clear to see that $\deg(s_k(x)) = 2^k$. For example, $s_0(x) = x$, and $s_2(x) = x(x - v_0)(x - v_1)(x - v_0 - v_1)$. The properties of $s_k(x)$ are given in [16, 18].

Theorem 1 ([16, 18])**.** *(i). $s_k(x)$ is an $\mathbb{F}_2$-linearlized polynomial for which*

$$s_k(x) = \sum_{i=0}^{k} s_{k,i} x^{2^i}, \quad (3)$$

with each $s_{k,i} \in \mathbb{F}_{2^m}$. This implies that

$$s_k(x + y) = s_k(x) + s_k(y), \forall x, y \in \mathbb{F}_{2^m}. \quad (4)$$

(ii). The formal derivative of $s_k(x)$ is a constant

$$s'_k(x) = \prod_{a \in V_k \setminus \{0\}} a. \quad (5)$$

The recursive form [16] of subspace polynomials is given by

$$s_0(x) = x; \quad (6)$$

$$s_j(x) = s_{j-1}(x) s_{j-1}(x - v_{j-1})$$
$$= (s_{j-1}(x))^2 - s_{j-1}(v_{j-1}) s_{j-1}(x) \quad j = 1, 2, \ldots, m. \quad (7)$$

B. Polynomial basis

Let $\bar{\mathbb{X}} = \{\bar{X}_0(x), \bar{X}_1(x), \ldots, \bar{X}_{2^m-1}(x)\}$ denote a basis of $\mathbb{F}_{2^m}[x]/(x^{2^m} - x)$. Each $\bar{X}_i(x)$ is defined as

$$\bar{X}_i(x) = X_i(x)/p_i, \quad (8)$$

where

$$X_i(x) = \prod_{j=0}^{m-1} (s_j(x))^{i_j}, \qquad p_i = \prod_{j=0}^{m-1} (s_j(v_j))^{i_j}, \quad (9)$$

and each $i_j \in \{0, 1\}$ is the binary representation of i. Notice that $(s_j(x))^0 = (s_j(v_j))^0 = 1$. For example, $\bar{X}_0(x) = 1$, and $\bar{X}_3(x) = X_3(x)/p_3 = (s_0(x)s_1(x))/(s_0(v_0)s_1(v_1))$. It can be seen that $\deg(\bar{X}_i(x)) = i$, and thus the basis $\bar{\mathbb{X}}$ can represent all elements in $\mathbb{F}_{2^m}[x]/(x^{2^m} - x)$.

2

Algorithm 1 Transform of the basis $\bar{X}$

Input: $\text{FFT}_{\bar{X}}(\bar{D}_{2^k}, k, \beta)$: $\bar{D}_{2^k} = (\bar{d}_0, \bar{d}_1, \ldots, \bar{d}_{2^k-1})$, k is the binary logarithm of size, and $\beta \in \mathbb{F}_{2^m}$
Output: 2^k evaluations $\underline{D}_{2^k} = (\underline{d}_0, \underline{d}_1, \ldots, \underline{d}_{2^k-1})$, where each $\underline{d}_i = \bar{D}_{2^k}(\omega_i + \beta)$
  1: **if** $k = 0$ **then return** $\bar{d}_0$
  2: **end if**
  3: **for** $i = 0, \ldots, 2^{k-1} - 1$ **do**
  4:     $g_i^{(0)} \leftarrow \bar{d}_i + \frac{s_{k-1}(\beta)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}}$
  5:     $g_i^{(1)} \leftarrow g_i^{(0)} + \bar{d}_{i+2^{k-1}}$
  6: **end for**
  7: Call $V_0 \leftarrow \text{FFT}_{\bar{X}}(\bar{D}_{2^{k-1}}^{(0)}, k-1, \beta)$, where $\bar{D}_{2^{k-1}}^{(0)} = (g_0^{(0)}, \ldots, g_{2^{k-1}-1}^{(0)})$ and $V_0 = (\underline{d}_0, \ldots, \underline{d}_{2^{k-1}-1})$
  8: Call $V_1 \leftarrow \text{FFT}_{\bar{X}}(\bar{D}_{2^{k-1}}^{(1)}, k-1, v_{k-1}+\beta)$, where $\bar{D}_{2^{k-1}}^{(1)} = (g_0^{(1)}, \ldots, g_{2^{k-1}-1}^{(1)})$ and $V_1 = (\underline{d}_{2^k}, \ldots, \underline{d}_{2^k-1})$
  9: **return** $\underline{D}_{2^k} = (\underline{d}_0, \underline{d}_1, \ldots, \underline{d}_{2^k-1})$

Algorithm 2 Inverse transform of the basis $\bar{X}$

Input: $\text{IFFT}_{\bar{X}}(\underline{D}_{2^k}, k, \beta)$: $\underline{D}_{2^k} = (\underline{d}_0, \underline{d}_1, \ldots, \underline{d}_{2^k-1})$, where each $\underline{d}_i = \bar{D}_{2^k}(\omega_i + \beta)$, k is the binary logarithm of size, and $\beta \in \mathbb{F}_{2^m}$
Output: $\bar{D}_{2^k} = (\bar{d}_0, \bar{d}_1, \ldots, \bar{d}_{2^k-1})$, the coefficients of $\bar{D}_{2^k}(x)$
  1: **if** $k = 0$ **then return** $\underline{d}_0$
  2: **end if**
  3: Call $\bar{D}_{2^{k-1}}^{(0)} \leftarrow \text{IFFT}_{\bar{X}}(V_0, k-1, \beta)$, where $V_0 = (\underline{d}_0, \ldots, \underline{d}_{2^{k-1}-1})$ and $\bar{D}_{2^{k-1}}^{(0)} = (g_0^{(0)}, \ldots, g_{2^{k-1}-1}^{(0)})$
  4: Call $\bar{D}_{2^{k-1}}^{(1)} \leftarrow \text{IFFT}_{\bar{X}}(V_1, k-1, v_{k-1}+\beta)$, where $V_1 = (\underline{d}_{2^k}, \ldots, \underline{d}_{2^k-1})$ and $\bar{D}_{2^{k-1}}^{(1)} = (g_0^{(1)}, \ldots, g_{2^{k-1}-1}^{(1)})$
  5: **for** $i = 0, \ldots, 2^{k-1} - 1$ **do**
  6:     $\bar{d}_{i+2^{k-1}} \leftarrow g_i^{(0)} + g_i^{(1)}$
  7:     $\bar{d}_i \leftarrow g_i^{(0)} + \frac{s_{k-1}(\beta)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}}$
  8: **end for**
  9: **return** $\bar{D}_{2^k} = (\bar{d}_0, \bar{d}_1, \ldots, \bar{d}_{2^k-1})$

A polynomial $\bar{D}_h(x)$ of degree h in the basis $\bar{\mathbb{X}}$ is represented as

$$\bar{D}_h(x) = \sum_{i=0}^{h-1} \bar{d}_i \bar{X}_i(x), \tag{10}$$

with each $\bar{d}_i \in \mathbb{F}_{2^m}$. Throughout this paper, $\bar{D}_h = (\bar{d}_0, \bar{d}_1, \ldots, \bar{d}_{h-1})$ is used to indicate the vector of the coefficients of $\bar{D}_h(x)$. Due to the fact $\deg(\bar{X}_i(x)) = i$, the new basis possesses the following properties.

Corollary 1. *Given a polynomial $f(x) \in \mathbb{F}_{2^m}[x]/(x^{2^m} - x)$ respectively expressed in the monmial basis and $\bar{\mathbb{X}}$*

$$f(x) = \sum_{i=0}^{2^m-1} f_i^{(0)} x^i = \sum_{i=0}^{2^m-1} f_i^{(1)} \bar{X}_i(x),$$

the following properties hold.
  1) $f_i^{(0)} = f_i^{(1)} = 0$, *for* $i \geq h+1$.
  2) $f_h^{(1)} = f_h^{(0)} \cdot p_h$.
  3) *For* $0 \leq j \leq h$, $(f_j^{(1)}, f_{j+1}^{(1)}, \ldots, f_h^{(1)})$ *is determined by* $(f_j^{(0)}, f_{j+1}^{(0)}, \ldots, f_h^{(0)})$, *and vice versa.*

III. MULTIPOINT EVALUATIONS AT V_k

For any polynomial $f(x)$ and a set V, let the notation $f(V)$ denote a set of evaluation values $f(V) = \{f(a)|\forall a \in V\}$. [15] gave a recursive algorithm in $O(2^k \lg(2^k))$ to calculate $\bar{D}_{2^k}(V_k + \beta)$, where

$$V_k + \beta = \{a + \beta | a \in V_k\} \text{ for any } \beta \in \mathbb{F}_{2^m}.$$

In this section, we describe the algorithm [15] in another viewpoint, which helps us to develop encoding/decoding algorithm for RS codes.

The set of evaluation points can be divided into two individual subsets

$$V_k + \beta = (V_{k-1} + \beta) \cup (V_{k-1} + v_{k-1} + \beta),$$

where $(V_{k-1} + v_{k-1} + \beta)$ is the coset of $(V_{k-1} + \beta)$ by adding v_{k-1}. Accordingly, the set of polynomial evaluations can be divided into two subsets

$$\bar{D}_{2^k}(V_k + \beta) = \bar{D}_{2^k}(V_{k-1} + \beta) \cup \bar{D}_{2^k}(V_{k-1} + v_{k-1} + \beta). \tag{11}$$

The algorithm relied on the following lemma.

Lemma 1. *Given $\gamma \in \mathbb{F}_{2^m}$ and a polynomial $\bar{D}_{2^k}(x) \in \mathbb{F}_{2^m}[x]/(x^{2^m} - x)$ in the basis $\bar{\mathbb{X}}$, we have*

$$\bar{D}_{2^k}(a + \gamma)$$
$$= \sum_{i=0}^{2^{k-1}-1} (\bar{d}_i + \frac{s_{k-1}(\gamma)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}}) \bar{X}_i(a + \gamma), \tag{12}$$

for each $a \in V_{k-1}$.

Based on Lemma 1, the algorithm to compute (11) is described below. By substituting $\gamma = \beta$ into (12), we obtain

$$\bar{D}_{2^k}(a + \beta)$$
$$= \sum_{i=0}^{2^{k-1}-1} (\bar{d}_i + \frac{s_{k-1}(\beta)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}}) \bar{X}_i(a + \beta)$$
$$= \sum_{i=0}^{2^{k-1}-1} g_i^{(0)} \bar{X}_i(a + \beta) = D_{2^{k-1}}^{(0)}(a + \beta) \qquad \forall a \in V_{k-1}, \tag{13}$$

where each

$$g_i^{(0)} = \bar{d}_i + \frac{s_{k-1}(\beta)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}} \qquad i = 0, 1, \ldots, 2^{k-1} - 1. \tag{14}$$

This converts $\bar{D}_{2^k}(V_{k-1} + \beta)$ into $D_{2^{k-1}}^{(0)}(V_{k-1} + \beta)$. Furthermore, by substituting $\gamma = v_{k-1} + \beta$ into (12), we obtain (15), where each

$$g_i^{(1)} = g_i^{(0)} + \bar{d}_{i+2^{k-1}} \qquad i = 0, 1, \ldots, 2^{k-1} - 1. \tag{16}$$

This converts $\bar{D}_{2^k}(V_{k-1}+v_{k-1}+\beta)$ into $D_{2^{k-1}}^{(1)}(V_{k-1}+v_{k-1}+\beta)$.

From (13) (15), the set of evaluation points (11) can be expressed as

$$\bar{D}_{2^k}(V_k + \beta)$$
$$= \bar{D}_{2^{k-1}}^{(0)}(V_{k-1} + \beta) \cup \bar{D}_{2^{k-1}}^{(1)}(V_{k-1} + v_{k-1} + \beta). \tag{17}$$

3

(a) The transform

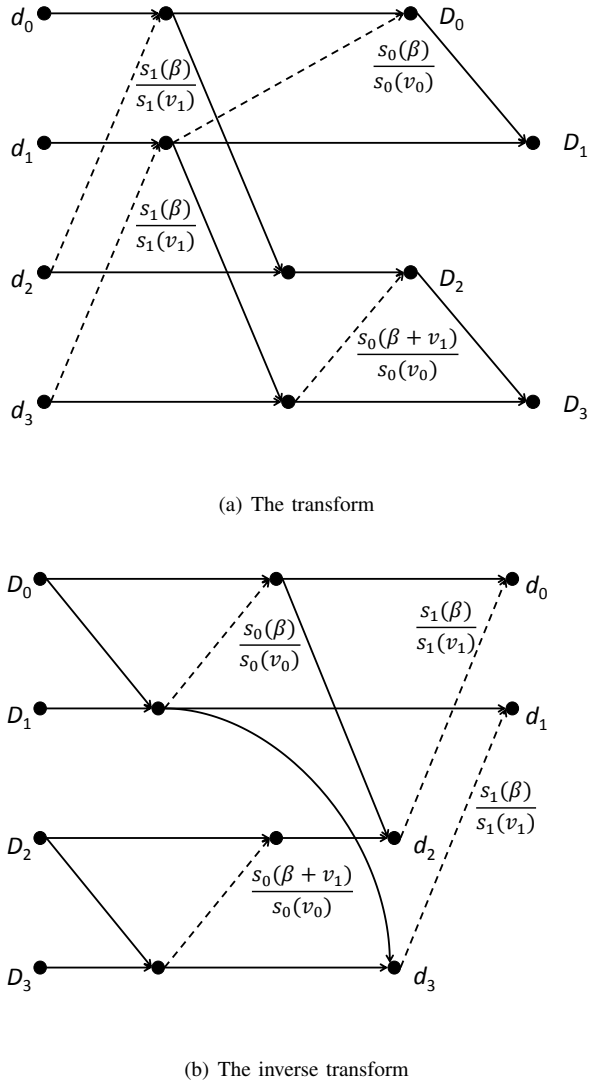(b) The inverse transform

Fig. 1.   Data flow diagram of proposed 4-poiont transform and its inversion.

By comparing (11) and (17), the degrees of both polynomials are reduced one-half (the number of terms are reduced from 2^k to 2^{k-1}). The complexity of obtaining both polynomials are discussed below. In (13), each coefficient g_i takes an addition and a multiplication, except if $s_{k-1}(\beta) = 0$, then $g_i = \bar{d}_i$ without any arithmetic operations. However, we do not consider this exception here, because the reduction from those exceptions is limited. As $D_{2^{k-1}}^{(0)}(x)$ has 2^{k-1} coefficients, it takes a total of 2^{k-1} additions and 2^{k-1} multiplications to obtain them. In (15), calculating each coefficient $g_i + \bar{d}_{i+2^{k-1}}$ takes an addition, so it takes a total of 2^{k-1} additions to obtain the coefficients of $\bar{D}_{2^{k-1}}^{(1)}(x)$.

This procedure can be applied recursively to each set $\bar{D}_{2^{k-1}}^{(0)}(V_{k-1} + \beta)$ and $\bar{D}_{2^{k-1}}^{(1)}(V_{k-1} + v_{k-1} + \beta)$ until the size of each set is one. With the divide-and-conquer strategy, the additive complexity and the multiplicative complexity are respectively written as

$$A(h) = 2 \times A(h/2) + h, \qquad M(h) = 2 \times M(h/2) + h/2,$$

and the result is $A(h) = h\lg(h)$ and $M(h) = h/2\lg(h)$.

Algorithm 1 depicts the details of the recursive approach, denoted as $\mathrm{FFT}_{\bar{\mathbb{X}}}(\bullet, k, \beta)$.

The inverse FFT can be obtained by backtracking FFT given above. As opposite to (17), the inverse transform get the coefficients of $\bar{D}_{2^{k-1}}^{(0)}(x)$ and $\bar{D}_{2^{k-1}}^{(1)}(x)$, and the objective is to find the coefficients of $\bar{D}_{2^k}(x)$. We reformulate (16) and (14) as

$$\bar{d}_{i+2^{k-1}} = g_i^{(0)} + g_i^{(1)},$$
$$\bar{d}_i = g_i^{(0)} + \frac{s_{k-1}(\beta)}{s_{k-1}(v_{k-1})}\bar{d}_{i+2^{k-1}} \qquad i = 0, 1, \ldots, 2^{k-1} - 1.$$
(18)

From (18), we can compute the coefficients of $\bar{D}_{2^k}(x)$. The coefficients of $\bar{D}_{2^{k-1}}^{(0)}(x)$ and $\bar{D}_{2^{k-1}}^{(1)}(x)$ can be obtained by applying the inverse transform recursively. The details are shown in Algorithm 2. Note that $\mathrm{IFFT}_{\bar{\mathbb{X}}}(\bullet, k, \beta)$ denotes the inverse transform. Algorithms 1 and 2 use the same notations such that one can follow them easily. It is clear that both algorithms have the same number of arithmetic operations. Figure 1 showed an example of the proposed algorithm and its inversion. The input polynomial is defined as $\bar{D}(x) = \sum_{i=0}^{3} d_i \bar{X}_i(x)$, and the output is given by $D_i = \bar{D}(\omega_i + \beta)$, for $i = 0, 1, 2, 3$.

IV. POLYNOMIAL BASIS WITH MONIC POLYNOMIALS AND ITS OPERATIONS

In this section, we define an alternative version of the polynomial basis, and its algorithms to perform multiplications, formal derivatives, and divisions on the new basis. All these operations will be used in the coding algorithms. The alternative basis is defined as

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

in $\mathbb{F}_{2^m}[x]/(x^{2^m} - x)$, where each $X_i(x)$ is given in (9). This implies that each $X_i(x)$ is a monic polynomial. For any $D_{h-1}(x) \in \mathbb{F}_{2^m}[x]/(x^{2^m} - x)$, the basis conversion between $\mathbb{X}$ and $\bar{\mathbb{X}}$ requires only h multiplications/divisions:

$$D_{h-1}(x) = \sum_{i=0}^{h-1} d_i \cdot \bar{X}_i(x) = \sum_{i=0}^{h-1} \frac{d_i}{p_i} X_i(x). \qquad (19)$$

With the linear-time basis conversion, the multipoint evaluation in $\bar{\mathbb{X}}$ (Algorithm 1) can also be applied on $\mathbb{X}$, and the complexity is unchanged.

To simplify the notations, in the rest of this paper, the polynomials are represented in $\mathbb{X}$. For $D_{2^k}(x)$ in $\mathbb{X}$, the evaluations at $V_k + \beta = \{\omega_i + \beta\}_{i=0}^{2^k-1}$ is denoted as

$$\mathrm{FFT}_{\mathbb{X}}(D_{2^k}, k, \beta)$$
$$= (D_{2^k}(\omega_0 + \beta), D_{2^k}(\omega_1 + \beta), \ldots, D_{2^k}(\omega_{2^k-1} + \beta)), \qquad (20)$$

and the inversion is denoted as $\mathrm{IFFT}_{\mathbb{X}}(D_{2^k}, k, \beta)$. Based on Algorithm 1, the transforms are defined as

$$\mathrm{FFT}_{\mathbb{X}}(D_{2^k}, k, \beta) = \mathrm{FFT}_{\bar{\mathbb{X}}}(D_{2^k} \otimes P_{2^k}, k, \beta),$$
$$\mathrm{IFFT}_{\mathbb{X}}(D_{2^k}, k, \beta) = \mathrm{IFFT}_{\bar{\mathbb{X}}}(D_{2^k}, k, \beta) \oslash P_{2^k}, \qquad (21)$$

where $P_{2^k} = (p_0, p_1, \ldots, p_{2^k-1})$. The operation $\otimes$ is the pairwise multiplication on two vectors, and the operation $\oslash$ is the pairwise division. Since the multiplication and formal

4

derivative in $\mathbb{X}$ are similar to those given in [15], we summarize them in Appendix A for completeness. Next we present the algorithm for polynomial division that is essential for decoding of RS codes.

A. Polynomial Division

In this subsection, we proposed an $\mathcal{O}(h \lg(h))$ polynomial division in the basis $\mathbb{X}$. The proposed algorithm is based on Newton iteration approach that was used by the fast division algorithms in the standard basis [13] with $\mathcal{O}(h \lg(h))$, if $\mathcal{O}(h \lg(h))$ FFT exists. However, since our basis is different from the standard basis, some moderate modifications are required.

As compared with the conventional fast division [13], the proposed approach has two major differences. First, the conventional fast division shall reverse the coefficients of the divisor $B(x)$ upon performing the Newton iteration. However, in our basis $\mathbb{X}$, the polynomial reversion cannot be applied. Thus, the proposed algorithm does not reverse the polynomials, and all operations are performed on the polynomials without reversions. Second, the proposed algorithm includes some specific multiplications that are not required in the conventional approach, such as $X_y(x)$ in (28) and $s_1(x)$ in (45). The objective of these multiplications are to align the results such that the desired polynomial can be extracted properly.

Let $\mathbb{Q}(A(x), i)$ denote the quotient of dividing $A(x)$ by $s_i(x)$, where $A(x)$ is in the basis $\mathbb{X}$ and $\deg(A(x)) < 2^{i+1}$. Precisely, for a polynomial of degree $h < 2^{i+1}$,

$$
\begin{aligned}
A(x) &= \sum_{l=0}^{h-1} a_l X_l(x) = \sum_{l=0}^{2^i-1} a_l X_l(x) + \sum_{l=2^i}^{h-1} a_l X_l(x) \\
&= \sum_{l=0}^{2^i-1} a_l X_l(x) + s_i(x) \sum_{l=2^i}^{h-1} a_l X_{l-2^i}(x).
\end{aligned} \tag{22}
$$

The quotient of dividing $A(x)$ by $s_i(x)$ is then

$$
\mathbb{Q}(A(x), i) = \sum_{l=0}^{h-1-2^i} a_{l+2^i} X_l(x).
$$

In general, given a dividend $a(x)$ and a divisor $b(x)$, the division is to determine the quotient $Q(x)$ and the remainder $r(x)$ such that

$$
a(x) = Q(x) \cdot b(x) + r(x), \tag{23}
$$

where $\deg(r(x)) \leq \deg(b(x)) - 1$. Without loss of generality, we consider the case

$$
\deg(a(x)) > \deg(b(x)) \geq 0. \tag{24}
$$

The proposed algorithm firstly finds out the quotient $Q(x)$, and then the remainder is calculated by

$$
r(x) = a(x) - Q(x) \cdot b(x). \tag{25}
$$

In the following, we focus on the algorithm to determine $Q(x)$.

Let

$$
y = 2^{D_\ell} - \deg(b(x)) - 1, \tag{26}
$$

where

$$
D_\ell = \lceil \lg(\deg(a(x)) + 1) \rceil. \tag{27}
$$

To begin with, (25) is multiplied by $X_y(x)$ to obtain

$$
r(x) \cdot X_y(x) = a(x) \cdot X_y(x) - Q(x) \cdot b(x) \cdot X_y(x). \tag{28}
$$

To simplify the notations, let

$$
\begin{aligned}
R(x) &= r(x) \cdot X_y(x), \\
A(x) &= a(x) \cdot X_y(x), \\
B(x) &= b(x) \cdot X_y(x).
\end{aligned} \tag{29}
$$

Then we have

$$
R(x) = A(x) - Q(x) \cdot B(x). \tag{30}
$$

Next we present a method to determine $Q(x)$ from (30).

Assume that there exists a polynomial $\Lambda(x)$ such that

$$
\Lambda(x) \cdot s_1(x) \cdot B(x) = s_{D_a}(x) + H(x), \tag{31}
$$

where

$$
\deg(H(x)) \leq \deg(B(x)) + 1 = 2^{D_\ell}, \tag{32}
$$

and

$$
D_a = \lceil \lg(\deg(A(x)) + 1) \rceil. \tag{33}
$$

$$
\begin{aligned}
& \bar{D}_{2^k}(a + v_{k-1} + \beta) \\
&= \sum_{i=0}^{2^{k-1}-1} (\bar{d}_i + \frac{s_{k-1}(v_{k-1} + \beta)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}}) \bar{X}_i(a + v_{k-1} + \beta) \\
&= \sum_{i=0}^{2^{k-1}-1} (\bar{d}_i + \frac{s_{k-1}(\beta)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}} + \bar{d}_{i+2^{k-1}}) \bar{X}_i(a + v_{k-1} + \beta) \\
&= \sum_{i=0}^{2^{k-1}-1} (g_i^{(0)} + \bar{d}_{i+2^{k-1}}) \bar{X}_i(a + v_{k-1} + \beta) \\
&= \sum_{i=0}^{2^{k-1}-1} g_i^{(1)} \bar{X}_i(a + v_{k-1} + \beta) = \bar{D}_{2^{k-1}}^{(1)}(a + v_{k-1} + \beta) \qquad \forall a \in V_{k-1},
\end{aligned} \tag{15}
$$

The algorithm to find out $\Lambda(x)$ will be addressed in Section IV-B. Before determining $Q(x)$, we first present two lemmas whose proofs are given in Appendix B.

Lemma 2.
$$D_a = D_\ell + 1. \tag{34}$$

From (31) and Lemma 2, the degree of $\Lambda(x)$ is thus

$$
\begin{aligned}
&\deg(\Lambda(x)) \\
&= \deg(s_{D_a}(x)) - \deg(B(x)) - \deg(s_1(x)) \\
&= 2^{D_a} - (2^{D_\ell} - 1) - 2 = 2^{D_\ell} - 1.
\end{aligned} \tag{35}
$$

After obtaining $\Lambda(x)$, (30) is multiplied by $\Lambda(x) \cdot s_1(x)$ to obtain

$$
\begin{aligned}
&R(x) \cdot \Lambda(x) \cdot s_1(x) \\
&= A(x) \cdot \Lambda(x) \cdot s_1(x) - Q(x) \cdot B(x) \cdot \Lambda(x) \cdot s_1(x)
\end{aligned} \tag{36}
$$

By (31), we have

$$
\begin{aligned}
&R(x) \cdot \Lambda(x) \cdot s_1(x) \\
&= A(x) \cdot \Lambda(x) \cdot s_1(x) - Q(x) \cdot (s_{D_a}(x) + H(x))
\end{aligned}
$$

and then

$$
\begin{aligned}
&Q(x) \cdot H(x) + R(x) \cdot \Lambda(x) \cdot s_1(x) \\
&= A(x) \cdot \Lambda(x) \cdot s_1(x) - Q(x) \cdot s_{D_a}(x).
\end{aligned} \tag{37}
$$

Lemma 3. *The left-hand side of* (37) *has degree*

$$\deg(Q(x) \cdot H(x) + R(x) \cdot \Lambda(x) \cdot s_1(x)) \le 2^{D_a} - 1. \tag{38}$$

In (37), $Q(x) \cdot s_{D_a}(x)$ is a polynomial where the coefficients of $Q(x)$ starts from $X_{2^{D_a}}(x) = s_{D_a}(x)$. By Lemma 3, the degree of the left-hand side is no more than $2^{D_a} - 1$. Thus, $A(x) \cdot \Lambda(x) \cdot s_1(x)$ has quotient $Q(x)$ starting on degree 2^{D_a}, and hence the quotient can be obtained by

$$Q(x) = \mathbb{Q}(A(x) \cdot \Lambda(x) \cdot s_1(x), D_a), \tag{39}$$

In (39), we have

$$\deg(A(x) \cdot \Lambda(x) \cdot s_1(x)) \le 2^{D_a} + 2^{D_\ell - 1} + 2 = 3 \cdot 2^{D_\ell} + 1. \tag{40}$$

Algorithm 3 shows the steps of the division algorithm. The complexity is analyzed below. In Step 1, as $\deg(A(x)) = \deg(a(x)) + \deg(y(x)) < 2^{D_\ell + 1}$ and $\deg(B(x)) = \deg(b(x)) + \deg(y(x)) = 2^{D_\ell} - 1$, the complexity is $\mathcal{O}(2^{D_\ell + 1} \lg(2^{D_\ell + 1})) = \mathcal{O}(2^{D_\ell} \lg(2^{D_\ell}))$. In Step 2, we will show that $\mathcal{O}(2^{D_\ell} \lg(2^{D_\ell}))$ suffice in Section IV-B. In Step 3, (40) shows that the complexity is $\mathcal{O}(2^{D_\ell} \lg(2^{D_\ell}))$. In Step 4, as the degrees of polynomials are less than 2^{D_ℓ}, the complexity is no more than $\mathcal{O}(2^{D_\ell} \lg(2^{D_\ell}))$. In summary, Algorithm 3 has the complexity $\mathcal{O}(2^{D_\ell} \lg(2^{D_\ell})) = \mathcal{O}(\deg(a(x)) \lg(\deg(a(x))))$.

B. Determining $\Lambda(x)$ given in (31)

Given $B(x) = \sum_{j=0}^{d_B} b_j X_j(x)$ with $b_{d_B} \ne 0$, this subsection presents a method to find out $\Lambda(x)$ in (31). Notice that $d_B = \deg(\Lambda(x)) = 2^{D_\ell} - 1$. The proposed method can be seen as a modified version of the division with Newton iterations [13][19].

The method iteratively computes the coefficients of $\Lambda(x)$ from highest degree to lowest degree. For $i = 0, 1, \ldots, D_\ell$,

---

Algorithm 3 Polynomial divisions in $\mathbb{X}$

Input: A dividend $a(x)$ and a divisor $b(x)$, with $\deg(a(x)) > \deg(b(x)) \ge 0$

Output: A quotient $Q(x)$ and a remainder $r(x)$, such that

$$a(x) = Q(x) \cdot b(x) + r(x).$$

1: Compute
$$
\begin{aligned}
A(x) &= a(x) \cdot X_y(x), \\
B(x) &= b(x) \cdot X_y(x),
\end{aligned}
$$
where y is defined as (26).
2: Find $\Lambda(x)$ such that (31) holds.
3: Compute $Q(x)$ by (39).
4: Compute $r(x)$ by (25).
5: **return** $Q(x)$ and $r(x)$.

---

the updated polynomial $\Lambda_i(x)$ of degree $2^i - 1$ is calculated from $\Lambda_{i-1}(x)$. The initial polynomial is

$$\Lambda_0(x) = b_{d_B}^{-1}. \tag{41}$$

Let $B_{D_\ell}(x) = B(x)$, and

$$B_i(x) = \mathbb{Q}(B_{i+1}(x), i) \qquad i = 0, 1, \ldots, D_\ell - 1. \tag{42}$$

(42) can be rewritten as

$$B_{i+1}(x) = B_i(x) \cdot s_i(x) + \bar{B}_i(x), \tag{43}$$

where $\bar{B}_i(x)$, $\deg(\bar{B}_i) \le 2^i - 1$, is the residual. Clearly, $\deg(B_i(x)) = 2^i - 1$.

For $i = 1, 2, \ldots, D_\ell$, $\Lambda_i(x)$ is defined as

$$\Lambda_i(x) = \mathbb{Q}((s_{i-1}(x))^2 \cdot \bar{\Lambda}_i(x), i + 1), \tag{44}$$

where

$$\bar{\Lambda}_i(x) = (\Lambda_{i-1}(x))^2 \cdot B_i(x) \cdot s_1(x). \tag{45}$$

It can be verified that $\deg(\bar{\Lambda}_i(x)) = 2^{i+1} - 1$ and $\deg(\Lambda_i(x)) = 2^i - 1$ holds. The validity of $\Lambda(x) = \Lambda_{D_\ell}(x)$ is supported as follows, where all proofs are given in Appendix B.

Lemma 4. $\Lambda_i(x)$ *possesses the following equality:*

$$\Lambda_i(x) \cdot B_i(x) \cdot s_1(x) = s_{i+1}(x) + \bar{r}_i(x), \tag{46}$$

with $\deg(\bar{r}_i(x)) \le 2^i$.

The following reformulation of (44), that contains no polynomial multiplications, can be used to determine the complexity of calculating (44).

Lemma 5. (44) *can be rewritten as*

$$\Lambda_i(x) = \Lambda_i^{(1)}(x) + \mathbb{Q}(\Lambda_i^{(1)}(x), i - 1) \cdot s_{i-1}(v_{i-1}), \tag{47}$$

where

$$\Lambda_i^{(1)}(x) = \mathbb{Q}(\bar{\Lambda}_i(x), i).$$

Algorithm 4 depicts the steps. The algorithm repeats performing (45) and (44) (or (47)) to obtain $\Lambda_{D_\ell}(x)$, which is the desired output $\Lambda(x)$. For the complexity, each iteration (lines 3-4) calculates (45) and (44). In (45), as $\deg(\Lambda_{i-1}(x)) =$

6

Algorithm 4 $\Lambda(x)$ computation

Input: A polynomial $B(x)$
Output: A polynomial $\Lambda(x)$ such that (31) holds, where
$\deg(\Lambda(x)) = \deg(B(x)) = 2^{D_\ell} - 1$.
  1: Let $\Lambda_0(x) = b_{d_B}^{-1}$.
  2: **for** $i = 1, 2, \ldots, D_\ell$ **do**
  3:     Compute (45).
  4:     Compute (44) (or (47), equivalently).
  5: **end for**
  6: **return** $\Lambda_{D_\ell}(x)$.

$2^{i-1} - 1$, $\deg(B_i(x)) = 2^i - 1$ and $\deg(s_i(x)) = 2$, the multiplications (45) requires $\mathcal{O}(2^i \lg(2^i))$. In (44), Lemma 5 showed that the computation can be reduced to $\mathcal{O}(2^i)$ without polynomial multiplications. Thus, each iteration takes $\mathcal{O}(2^i \lg(2^i))$ operations, and the complexity for the loop (line 2-5) takes

$$\sum_{i=1}^{D_\ell} \mathcal{O}(2^i \lg(2^i)) = \mathcal{O}(2^{D_\ell} \lg(2^{D_\ell})).$$

V. EXTENDED EUCLIDEAN ALGORITHM BASED ON HALF-GCD APPROACH

This section introduces the extended Euclidean algorithm that will be used in the decoding of RS codes. Given two polynomials $a(x) = r_{-1}(x)$, $b(x) = r_0(x)$, and

$$\deg(b(x)) \leq \deg(a(x)) < 2^g, \tag{52}$$

Euclidean algorithm is a procedure to recursively divide $r_{k-2}(x)$ by $r_{k-1}(x)$ to get

$$r_{k-2}(x) = q_k(x) \cdot r_{k-1}(x) + r_k(x),$$

with $\deg(r_k) < \deg(r_{k-1})$. The procedure stops at $r_N(x) = 0$, and $r_{N-1}(x)$ is the greatest common divisor (gcd) of $a(x)$ and $b(x)$. An extension version, namely extended Euclidean algorithm, calculates $r_k(x)$ with a pair of polynomials $(u_k(x), v_k(x))$ in each iteration such that

$$a(x) \cdot u_k(x) + b(x) \cdot v_k(x) = r_k(x).$$

The $(k-1)$-th step of extended Euclidean algorithm can be expressed as a matrix form

$$\begin{bmatrix} r_{k-2}(x) \\ r_{k-1}(x) \end{bmatrix} = \begin{bmatrix} u_{k-2}(x) & v_{k-2}(x) \\ u_{k-1}(x) & v_{k-1}(x) \end{bmatrix} \cdot \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}. \tag{53}$$

The next step is shown as

$$\begin{bmatrix} r_{k-1}(x) \\ r_k(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_k(x) \end{bmatrix} \cdot \begin{bmatrix} u_{k-2}(x) & v_{k-2}(x) \\ u_{k-1}(x) & v_{k-1}(x) \end{bmatrix} \cdot \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}. \tag{54}$$

The half-GCD algorithm [20][13] calculates the temporal result of extended Euclidean algorithm at s-th step such that

$$\deg(r_s(x)) \leq 2^{g-1} - 1. \tag{55}$$

In this section, we present a half-GCD algorithm in basis $\mathbb{X}$. This approach will be performed to solve the error locator polynomial (see (76)) in the decoding procedure of RS codes.

Algorithm 5 Half-GCD algorithm

Input: $\text{HGCD}(a(x), b(x), g)$, where $a(x), b(x) \in \mathbb{F}_{2^m}[x]/x^{2^m} - x$ in basis $\mathbb{X}$, and $\deg(b(x)) \leq \deg(a(x))$, $2^{g-1} \leq \deg(a(x)) \leq 2^g - 1$
Output: Two matrices (Z, M) given in (58)
  1: **if** $\deg(b(x)) < 2^{g-1}$ **then return**

$$Z = \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}, \qquad M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{48}$$

  2: **end if**
  3: $(Z_H, M_H) \leftarrow \text{HGCD}(a_H(x), b_H(x), g-1)$
  4: Compute

$$\begin{bmatrix} z_{M0}(x) \\ z_{M1}(x) \end{bmatrix} = Z_H \cdot s_{g-1}(x) + M_H \begin{bmatrix} a_L(x) \\ b_L(x) \end{bmatrix}. \tag{49}$$

  5: **if** $\deg(z_{Ml}(x)) \leq 2^{g-1} - 1$ **then return**

$$\left(Z_M = \begin{bmatrix} z_{M0}(x) \\ z_{Ml}(x) \end{bmatrix}, M_H \right).$$

  6: **end if**
  7: $z_{M0}(x)$ is divided by $z_{M1}(x)$ to get

$$z_{M0}(x) = q_M(x) \cdot z_{M1}(x) + r_M(x) \tag{50}$$

  with $\deg(r_M(x)) < \deg(z_{M1}(x)) \leq 2^{g-1} + 2^{g-2} - 1$.
  8: $z_{M1}(x)$ and $r_M(x)$ are divided into three polynomials, denoted as

$$z_{M1}(x)$$
$$= z_{M1LL}(x) + s_{g-2}(x) z_{M1LH}(x) + s_{g-1}(x) z_{M1H}(x),$$
$$r_M(x)$$
$$= r_{MLL}(x) + s_{g-2}(x) r_{MLH}(x) + s_{g-1}(x) r_{MH}(x).$$

  Compute

$$z_{M1M}(x)$$
$$= z_{M1LH}(x) + (s_{g-2}(x) + s_{g-2}(v_{g-2})) z_{M1H}(x),$$
$$r_{MM}(x)$$
$$= r_{MLH}(x) + (s_{g-2}(x) + s_{g-2}(v_{g-2})) r_{MH}(x).$$

  9: $(Y_M, M_M) \leftarrow \text{HGCD}(z_{M1M}(x), r_{MM}(x), g-1)$
 10: **return** (Z_R, M_R), where

$$M_R = M_M \begin{bmatrix} 0 & 1 \\ 1 & -q_M(x) \end{bmatrix} M_H,$$
$$Z_R = Y_M \cdot s_{g-2}(x) + M_M \begin{bmatrix} z_{M1LL}(x) \\ r_{MLL}(x) \end{bmatrix}. \tag{51}$$

For polynomials in the monomial basis, there exist fast approaches in $\mathcal{O}(M(h) \lg(h))$ operations, where $M(h)$ denotes the complexity of multiplying two polynomials of degrees $h/2$ (see [13, Algorithm 11.6] or [19, Figure 8.3]). The idea comes from an observation that, the quotient $q_k(x)$ in (54) is determined by the upper degree part of $r_{k-1}(x)$ and $r_k(x)$, and the lower degree part of $r_{k-1}(x)$ and $r_k(x)$ are not necessary. Fortunately, this observation is also applicable to our basis $\mathbb{X}$.

From the observation, we partition the inputs $a(x)$ (and $b(x)$) into several portions, so that the procedure can be applied

on the portions of higher degrees. For the algorithms on monomial basis, it is simple to make such partitions. For basis $\mathbb{X}$, we have to choose partition points at degrees $X_{2^{g-2}}(x)$ and $X_{2^{g-1}}(x)$. Precisely, $a(x)$ is divided into three polynomials $a_{\mathrm{LL}}(x)$, $a_{\mathrm{LH}}(x)$ and $a_{\mathrm{H}}(x)$ at $s_{g-2}(x)$ and $s_{g-1}(x)$, respectively. The representation is given by

$$
\begin{aligned}
a(x) =& a_{\mathrm{L}}(x) + s_{g-1}(x)a_{\mathrm{H}}(x) \\
=& a_{\mathrm{LL}}(x) + s_{g-2}(x)a_{\mathrm{LH}}(x) + s_{g-1}(x)a_{\mathrm{H}}(x).
\end{aligned}
\tag{56}
$$

Similarly, $b(x)$ is partitioned in the same manner:

$$
\begin{aligned}
b(x) =& b_{\mathrm{L}}(x) + s_{g-1}(x)b_{\mathrm{H}}(x) \\
=& b_{\mathrm{LL}}(x) + s_{g-2}(x)b_{\mathrm{LH}}(x) + s_{g-1}(x)b_{\mathrm{H}}(x).
\end{aligned}
\tag{57}
$$

Algorithm 5 depicts the proposed algorithm $\mathrm{HGCD}(a(x), b(x), g)$, with $\deg(b(x)) \leq \deg(a(x))$ and $2^{g-1} \leq \deg(a(x)) \leq 2^g - 1$. The algorithm outputs two matrices

$$
Z = \begin{bmatrix} z_0(x) \\ z_1(x) \end{bmatrix} \text{ and } M = \begin{bmatrix} m_{00}(x) & m_{01}(x) \\ m_{10}(x) & m_{11}(x) \end{bmatrix}
\tag{58}
$$

such that

1)
$$
Z = M \cdot \begin{bmatrix} a(x) \\ b(x) \end{bmatrix};
\tag{59}
$$

2)
$$
\begin{aligned}
\deg(z_0(x)) &\geq 2^{g-1}, \\
\deg(z_1(x)) &\leq 2^{g-1} - 1;
\end{aligned}
\tag{60}
$$

3)
$$
\deg(m_{11}(x)) \leq \deg(a(x)) - \deg(z_0(x));
\tag{61}
$$

4)
$$
\begin{aligned}
\deg(m_{i0}(x)) &\leq \deg(m_{i1}(x)), \\
\deg(m_{0i}(x)) &\leq \deg(m_{1i}(x)), \qquad i = 0, 1.
\end{aligned}
\tag{62}
$$

Before proving the validity of Algorithm 5, we give the following Lemmas whose proofs are given in Appendix B.

Lemma 6. *Algorithm 5 always outputs Z and M given in (58) that satisfy (59).*

Lemma 7. *The recursive calls in $\mathrm{HGCD}(a(x), b(x), g)$ meet the requirements $\deg(b(x)) < \deg(a(x))$ and $2^{g-1} \leq \deg(a(x)) \leq 2^g - 1$.*

Lemma 8. *Algorithm 5 always outputs Z and M given in (58) that satisfies (60).*

Lemma 9. *Algorithm 5 always outputs Z and M given in (58) that satisfy (61) and (62).*

By the above Lemmas, we have

Theorem 2. *Algorithm 5 is valid. That is, Algorithm 5 always outputs Z and M given in (58) that satisfy the above four conditions.*

We determine the computational complexity as follows. The algorithm complexity is denoted as $T(h)$ of polynomial degrees $h = 2^g$. In step 3 and step 9, the algorithm shall call the routine twice, and it takes $2 \cdot T(h/2)$. line 7 is the

polynomial division, and this requires $\mathcal{O}(h \lg(h))$ by using the fast division approach in Sec. IV. Line 4 and line 10 have polynomial additions and polynomials multiplications. As those polynomials have degrees less than h, the complexity is $\mathcal{O}(h \lg(h))$ by the results given in Appendix A. In summary, the overall complexity is

$$
T(h) = 2T(h/2) + \mathcal{O}(h \lg(h)), \text{ and } T(h) = \mathcal{O}(h \lg^2(h)).
$$

VI. REED-SOLOMON ENCODING ALGORITHM

This section introduces an $\mathcal{O}(n \lg(n - k))$ encoding algorithm for $(n = 2^m, k)$ RS codes over $\mathbb{F}_{2^m}$, with $T = 2^t = n - k$ a power of two. There exist two viewpoints for the constructions of RS codes, termed as the polynomial evaluation approach and the generator polynomial approach. For the polynomial evaluation approach, the message is interpreted as a polynomial $\mathbf{u}(x) \in \mathbb{F}_{2^m}[x]/(x^{2^m} - x)$ of degree less than k. The codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is defined as the evaluations of $\mathbf{u}(x)$ at n distinct points.

Assume $\mathbf{u}(x)$ is in the basis $\bar{\mathbb{X}}$, and thus $\mathbf{u}(x) = \sum_{i=0}^{k-1} u_i \bar{X}_i(x)$. The vector of coefficients is denoted as

$$
\mathbf{u} = (u_0, u_1, \ldots, u_{k-1}, \underbrace{\omega_0, \omega_0, \ldots, \omega_0}_{T}),
\tag{63}
$$

with T ω_0s in the high degree part. Then the codeword can be computed via Algorithm 1:

$$
\mathbf{v} = \mathrm{FFT}_{\bar{\mathbb{X}}}(\mathbf{u}, m, \omega_0).
\tag{64}
$$

However, (64) requires $\mathcal{O}(n \lg(n))$ operations, and the generated codeword is not systematic. In the following, another formula with complexity $\mathcal{O}(n \lg(n - k))$ is given, and the generated codeword is systematic. The inversion of (64) is given by

$$
\mathbf{u} = \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}, m, \omega_0).
\tag{65}
$$

Note that, in (65), $\mathbf{u}$ has T ω_0s in the high degree part (see (63)). To begin with, $\mathbf{v}$ is divided into a number of sub-vectors

$$
\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{n/T-1}),
\tag{66}
$$

where each $\mathbf{v}_i$ has T elements defined as

$$
\mathbf{v}_i = (v_{i \cdot T}, v_{1+i \cdot T}, \ldots, v_{T-1+i \cdot T}) \qquad i = 0, 1, \ldots, n/T - 1.
$$

Those sub-vectors can be proved to possess the equality given in the following lemma, whose proof is given in Appendix B.

Lemma 10. *The following equality is hold:*

$$
\begin{aligned}
\omega_0 =& \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}_0, t, \omega_0) + \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}_1, t, \omega_T) + \ldots \\
& + \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}_{n/T-1}, t, \omega_k),
\end{aligned}
\tag{67}
$$

where $+$ is the addition for vectors.

(67) plays the core transform of the proposed algorithm. Assume $\mathbf{v}_0$ includes the parity symbols, and others $\{\mathbf{v}_i\}_{i=1}^{n/T-1}$ are the message symbols. From (67), the parity is computed via

$$
\begin{aligned}
\mathbf{v}_0' =& \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}_1, t, \omega_T) + \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}_2, t, \omega_{2T}) + \ldots \\
& + \mathrm{IFFT}_{\bar{\mathbb{X}}}(\mathbf{v}_{n/T-1}, t, \omega_k), \\
\mathbf{v}_0 =& \mathrm{FFT}_{\bar{\mathbb{X}}}(\mathbf{v}_0', t, \omega_0).
\end{aligned}
\tag{68}
$$

8

This algorithm requires a T-point FFT and $(n/T - 1)$ times of T-point IFFT. Hence, the complexity of the encoding algorithm is

$$\mathcal{O}(T \lg(T)) + (n/T - 1)\mathcal{O}(T \lg(T)) = \mathcal{O}(n \lg(n - k)).$$

VII. REED-SOLOMON DECODING ALGORITHM

This section shows a decoding algorithm for $(n = 2^m, k)$ RS codes over $\mathbb{F}_{2^m}$, where the codeword $\mathbf{v} = (v_0, \ldots, v_{n-1}) = (\mathbf{u}(\omega_0), \ldots, \mathbf{u}(\omega_{n-1}))$ is generated by Section VI. The proposed algorithm follows the syndrome-based decoding process. Let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1}) = \mathbf{v} + \mathbf{e}$ denote the received vector with error pattern $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$. Hence,

$$r_i = \mathbf{u}(\omega_i) + e_i. \tag{69}$$

If $e_i \neq 0$, r_i is an erroneous symbol. Suppose $\mathbf{e}$ contains $v \leq (n - k)/2 = T/2$ non-zero symbols. Let

$$E = \{\omega_i \in \mathbf{F}_{2^m} | e_i \neq 0\} \tag{70}$$

denote the set of ω_i corresponding to locations of errors. Then, error-locator polynomial is defined as

$$\lambda(x) = \prod_{\omega_i \in E} (x - \omega_i). \tag{71}$$

Let $\bar{\mathbf{r}}(x)$ denote a polynomial of degree less than 2^m, with $\bar{\mathbf{r}}(\omega_i) = r_i, \forall \omega_i \in \mathbb{F}_{2^m}$. It is clear to see that

$$\bar{\mathbf{r}}(\omega_i) \cdot \lambda(\omega_i) = \begin{cases} 0 & \text{if } \omega_i \in E; \\ r_i \cdot \lambda(\omega_i) & \text{if } \omega_i \in \mathbb{F}_{2^m} \setminus E. \end{cases}$$

The above formula leads to

$$\mathbf{u}(\omega_i) \cdot \lambda(\omega_i) = \bar{\mathbf{r}}(\omega_i) \cdot \lambda(\omega_i)$$
$$\Rightarrow \mathbf{u}(x) \cdot \lambda(x) = \bar{\mathbf{r}}(x) \cdot \lambda(x) \pmod{x - \omega_i} \qquad \forall \omega_i \in \mathbb{F}_{2^m}. \tag{72}$$

Due to

$$\prod_{i=0}^{2^m - 1} (x - \omega_i) = x^{2^m} - x = s_m(x),$$

(72) implies that

$$\mathbf{u}(x) \cdot \lambda(x) \equiv \bar{\mathbf{r}}(x) \cdot \lambda(x) \pmod{s_m(x)}$$
$$\Rightarrow \mathbf{u}(x) \cdot \lambda(x) = \bar{\mathbf{r}}(x) \cdot \lambda(x) + \mathbf{q}(x) \cdot s_m(x), \tag{73}$$

with $\deg(\mathbf{q}(x)) < v \leq T/2$. Given $\bar{\mathbf{r}}(x)$, (73) is the key equation [21][9] to find out $\lambda(x)$, by applying the Euclidean algorithm on $s_m(x)$ and $\bar{\mathbf{r}}(x)$. However, though (73) is similar to the key equation of the syndrome decoding, $\bar{\mathbf{r}}(x)$ is not the syndrome polynomial. To obtain the syndrome decoding, the new key formula is the quotients of dividing $\lambda(x)$ and $s_m(x)$ by $X_k(x)$.

In this case, $\bar{\mathbf{r}}(x)$ is divided into two parts

$$\bar{\mathbf{r}}(x) = \bar{\mathbf{r}}_0(x) + X_k(x)\mathbf{s}(x), \tag{74}$$

where $\bar{\mathbf{r}}_0(x)$ denotes the residual. Notably, if no error occurs, $\bar{\mathbf{r}}(x) = \mathbf{u}(x)$ of degree less than k, and hence $\mathbf{s}(x) = 0$. Thus we can take $\mathbf{s}(x)$ as the syndrome polynomial.

For $s_m(x)$, the polynomial is recursively decomposed by (7) to obtain (75).

In (75), the degree of each term is less than $k = n - T = 2^m - 2^t$, except for the last term $X_{2^m - 2^t}(x)s_t(x)$. Thus, the quotient of dividing $s_m(x)$ by $X_k(x)$ would be $s_t(x)$.

Based on above results, the new key formula is

$$\mathbf{z}_0(x) = \mathbf{s}(x)\lambda(x) + \mathbf{q}(x)s_t(x), \tag{76}$$

with $\deg(\mathbf{z}_0(x)) \leq T/2$. (76) is the key equation to find the error locator polynomial.

To find $\lambda(x)$, extended Euclidean algorithm is applied on $s_t(x)$ and $\mathbf{s}(x)$. The extended Euclidean algorithm stops when the remainder has degree less than $T/2$. After obtaining $\lambda(x)$, the next step is to find out the locations of errors E defined in (70), that is the set of roots of $\lambda(x)$.

After obtaining E, the final step is to calculate the error values. The formal derivative of (73) is

$$\mathbf{u}'(x) \cdot \lambda(x) + \mathbf{u}(x) \cdot \lambda'(x)$$
$$= \bar{\mathbf{r}}'(x) \cdot \lambda(x) + \bar{\mathbf{r}}(x) \cdot \lambda'(x) + \mathbf{q}'(x) \cdot s_m(x) + \mathbf{q}(x). \tag{77}$$

By substituting $\omega_i \in E$ into (77), the error value is given by

$$\mathbf{u}(\omega_i) \cdot \lambda'(\omega_i) = \bar{\mathbf{r}}(\omega_i) \cdot \lambda'(\omega_i) + \mathbf{q}(\omega_i)$$
$$\Rightarrow \mathbf{u}(\omega_i) - \bar{\mathbf{r}}(\omega_i) = \frac{\mathbf{q}(\omega_i)}{\lambda'(\omega_i)}, \qquad \forall \omega_i \in E. \tag{78}$$

Notice that (78) uses $\mathbf{q}(x)$ to compute the error values, rather than $\mathbf{z}_0(x)$ used in Forney's formula. In summary, the decoding algorithm consists of four steps:

1) Calculate syndrome polynomial $\mathbf{s}(x)$.
2) Determine the error-locator polynomial $\lambda(x)$ from (76) by extended Euclidean algorithm.
3) Find the error locations E.
4) Calculate the error values via (78).

The details of each step is described below. In the first step, $\mathbf{s}(x)$ is the high degree part of applying IFFT on the received codeword $\mathbf{r}$. However, since the high degree part is required only, we follow the same idea of the encoding formula (67). In particular, the received codeword is divided into several individual parts $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{n/T-1})$, where each $\mathbf{r}_i$ has $T = 2^t = n - k$ elements. Then the syndrome polynomial is calculated by

$$\mathbf{s} = \text{IFFT}_{\mathbb{X}}(\mathbf{r}_0, t, \omega_0) + \text{IFFT}_{\mathbb{X}}(\mathbf{r}_1, t, \omega_T) + \ldots$$
$$+ \text{IFFT}_{\mathbb{X}}(\mathbf{r}_{n/T-1}, t, \omega_k).$$

In the second step, the fast Euclidean algorithm (Algorithm 5) is applied on $s_t(x)$ and $\mathbf{s}(x)$. Upon performing the Euclidean algorithm, we go a step by dividing $s_t(x)$ with $\mathbf{s}(x)$, resulting in

$$s_t(x) = q_{\text{t}}(x) \cdot \mathbf{s}(x) + r_{\text{t}}(x).$$

Then call Algorithm 5 with inputs $\mathbf{s}(x)$ and $r_{\text{t}}(x)$ to obtain

$$\left(\begin{bmatrix} z_0(x) \\ z_1(x) \end{bmatrix}, \begin{bmatrix} u_0(x) & v_0(x) \\ u_1(x) & v_1(x) \end{bmatrix} \right) \leftarrow \text{HGCD}(\mathbf{s}(x), r_{\text{t}}(x), T).$$

Then we have

$$z_1(x) = u_1(x)\mathbf{s}(x) + v_1(x)r_{\text{t}}(x)$$
$$\Rightarrow z_1(x) = u_1(x)\mathbf{s}(x) + v_1(x)(s_t(x) - q_{\text{t}}(x) \cdot \mathbf{s}(x)) \tag{79}$$
$$\Rightarrow z_1(x) = v_1(x)s_t(x) + (u_1(x) - v_1(x)q_{\text{t}}(x))\mathbf{s}(x),$$

9

and thus the error locator polynomial is given by

$$\lambda(x) = u_1(x) - v_1(x)q_t(x).$$

In the third step, the roots of $\lambda(x)$ can be searched via FFTs. The transform

$$\mathrm{FFT}_{\mathbb{X}}(\lambda, T, \omega_{i \cdot T}) \qquad (80)$$

is to evaluate $\lambda(x)$ at $V_t + \omega_{i \cdot T}$. If the result vector contains zeros, then $\lambda(x)$ has some roots at the corresponding points. (80) is performed at $i = 0, 1, \ldots, n/T - 1$ to search the roots in $\mathbb{F}_{2^m}$. Notably, if $\deg(\lambda(x))$ is larger than the number of found roots, the decoding procedure shall be terminated. This situation occurs when the number of errors exceeds $T/2$.

In the final step, we compute $\mathrm{FFT}_{\mathbb{X}}(\mathbf{q}, T, \omega_{i \cdot T})$ and $\mathrm{FFT}_{\mathbb{X}}(\lambda', T, \omega_{i \cdot T})$ (computing $\lambda'(x)$ is given in Appendix A), for $i = 0, 1, \ldots, n/T - 1$. Then the error values are calculated via (78).

To determine the computational complexity, the first step requires (n/T) times of T-point IFFT such that the complexity is $n/T \cdot \mathcal{O}(T \lg(T)) = \mathcal{O}(n \lg(n-k))$. The second step takes $\mathcal{O}((n-k) \lg^2(n-k))$ operations. The third step requires (n/T) times of T-point FFT, and thus the complexity is $\mathcal{O}(n \lg(n-k))$. The final step requires a formal derivative of polynomial degree T, and at most $2(n/T)$ times of T-point FFT. Thus, the complexity is $\mathcal{O}(n \lg(n-k))$. In summary, the proposed decoding algorithm requires $\mathcal{O}(n \lg(n-k) + (n-k) \lg^2(n-k))$.

VIII. Concluding remarks

In the simulations, we implemented the algorithm in C and compiled it in 64-bit GCC compiler on Intel Xeon X5650 and Windows 7 platform. For $(n, k) = (2^{16}, 2^{15})$ RS codes over $\mathbb{F}_{2^{16}}$, the program took about 2.22×10^{-3} second to produce a codeword. We tested a codeword with $(n-k)/2$ errors, and the decoding takes about 0.401 seconds. As for a comparison, we also ran the standard RS decoding algorithm [22], that took about 22.014 seconds to decode a codeword. Thus, the proposed decoding is around 50 times faster than the traditional approach under the parameter configurations described above. In our simulations, the proposed RS algorithm is suitable for long RS codes.

In this paper, we developed fast decoding algorithms for $(n = 2^m, k)$ systematic Reed-Solomon (RS) codes over fields $\mathbb{F}_{2^m}, m \in \mathbb{Z}^+$. The proposed algorithms are formed on a new basis $\mathbb{X}$ [15]. We reformulated the formulas of the syndrome-based decoding algorithm, such that the FFTs for the new basis can be applied. Further, the fast polynomial division algorithm is proposed. We made some modifications such that the Newton iteration can be applied to the new basis. The fast

Euclidean algorithm was also given in this paper. Combining these algorithms, a fast RS decoding algorithm is proposed, to achieve the complexity $\mathcal{O}(n \lg(n-k) + (n-k) \lg^2(n-k))$. By letting k/n a constant, the complexity can be written as $\mathcal{O}(n \lg^2(n))$, that improves upon the best currently available decoding complexity of $\mathcal{O}(n \lg^2(n) \lg \lg(n))$ [9]. Although Justesen [8] had given the algorithm with the same complexity in 1976, it does not include the field $\mathbb{F}_{2^m}$, that can be recognized as the most important case in the real applications.

The following we address some potential future works: 1. To remove the constraint $(n - k)$ a power of two in the encoding/decoding algorithms. This will increase the values of n and k to be selected. 2. To generalize the algorithm to handle both errors and erasures. 3. To reduce the leading constant of the FFT approach. This will make the proposed algorithm more competitive for short codes.

References

[1] 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.

[2] J. K. Wolf, "Adding two information symbols to certain nonbinary bch codes and some applications," *Bell System Technical Journal*, vol. 48, no. 7, pp. 2405–2424, Sept 1969.

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

[4] S.-J. Lin, W.-H. Chung, Y. S. Han, and T. Y. Al-Naffouri, "A unified form of exact-msr codes via product-matrix frameworks," *IEEE Trans. Inf. Theory*, vol. 61, no. 2, pp. 873–886, Feb 2015.

[5] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX, 2012, pp. 15–26.

[6] N. Chen and Z. Yan, "Complexity analysis of reed-solomon decoding over $GF(2^m)$ without using syndromes," *EURASIP J. Wirel. Commun. Netw.*, vol. 2008, pp. 16:1–16:11, Jan. 2008.

[7] T. Truong, P. Chen, L. Wang, and T. Cheng, "Fast transform for decoding both errors and erasures of reed-solomon codes over $GF(2^m)$ for $8 \le m \le 10$," *IEEE*

$$
\begin{aligned}
s_m(x) =& s_{m-1}(x)\left(s_{m-1}(v_{m-1}) + s_{m-1}(x)\right) \\
=& s_{m-1}(x)\left(s_{m-1}(v_{m-1}) + s_{m-2}(x)\left(s_{m-2}(v_{m-2}) + s_{m-2}(x)\right)\right) \\
=& \cdots \\
=& s_{m-1}(x)s_{m-1}(v_{m-1}) + s_{m-1}(x)s_{m-2}(x)s_{m-2}(v_{m-2}) + \ldots \\
& + s_{m-1}(x)s_{m-2}(x)\ldots s_t(x)s_t(v_t) + s_{m-1}(x)s_{m-2}(x)\ldots s_t(x)s_t(x) \\
=& X_{2^m-2^{m-1}}(x)s_{m-1}(v_{m-1}) + X_{2^m-2^{m-2}}(x)s_{m-2}(v_{m-2}) + \cdots + X_{2^m-2^t}(x)s_t(v_t) + X_{2^m-2^t}(x)s_t(x).
\end{aligned}
\qquad (75)
$$

Trans. Veh. Commun., vol. 54, no. 2, pp. 181–186, Feb 2006.

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

[9] S. Gao, "A new algorithm for decoding Reed-Solomon codes," in *Communications, Information and Network Security*. Kluwer, 2002, pp. 55–68.

[10] V. Y. Pan, "Faster solution of the key equation for decoding bch error-correcting codes," in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 168–175.

[11] 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

[12] D. G. Cantor and E. Kaltofen, "On fast multiplication of polynomials over arbitrary algebras," *Acta Informatica*, vol. 28, no. 7, pp. 693–701, 1991.

[13] J. V. Z. Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed. New York, NY, USA: Cambridge University Press, 2013.

[14] 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.

[15] S. J. Lin, W. H. Chung, and Y. S. Han, "Novel polynomial basis and its application to reed-solomon erasure codes," in *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, Oct 2014, pp. 316–325.

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

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

[18] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomial over F₂," in *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, Zurich, Switzerland, 1996, pp. 1–9.

[19] T. Mateer, "Fast Fourier transform algorithms with applications," Ph.D. dissertation, Clemson, SC, USA, 2008.

[20] R. T. Moenck, "Fast computation of GCDs," in *ACM Symposium on Theory of Computing (STOC)*, 1973, pp. 142–151.

[21] A. Shiozaki, "Decoding of redundant residue polynomial codes using euclid's algorithm," *IEEE Trans. Inf. Theory*, vol. 34, no. 5, pp. 1351–1354, Sep 1988.

[22] S. Rockliff. (1989) Reed-Solomon (RS) codes. [Online]. Available: http://www.eccpage.com/

APPENDIX A
POLYNOMIAL MULIPLICATION AND FORMAL DERIVATIVE ON NEW BASIS

[15] showed the polynomial multiplication and formal derivative in $\bar{\mathbb{X}}$. We take the similar procedure to show the corresponding operations in $\mathbb{X}$.

A. Muliplication

To multiply two polynomials, there exists a well-known fast approach based on FFT techniques. This approach can also be applied on the basis $\mathbb{X}$ over finite fields $\mathbb{F}_{2^m}$. Let $a(x) = \sum_{i=0}^{h-1} a_i \cdot X_i(x)$ and $b(x) = \sum_{i=0}^{h-1} b_i \cdot X_i(x)$ denote the two polynomials in $\mathbb{X}$. Its product $a(x) \cdot b(x) \pmod{s_i(x)}$ can be computed as

$$\mathrm{IFFT}_{\mathbb{X}}(\mathrm{FFT}_{\mathbb{X}}(a, \beta) \otimes \mathrm{FFT}_{\mathbb{X}}(b, \beta), \beta),$$

where $a = (a_0, a_1, \ldots, a_{h-1}, 0, \ldots, 0)$ is a 2^i-point vector represents the coefficients of $a(x)$ up to degree $2^i - 1$. Similarly, b is defined accordingly. The operation $\otimes$ performs pairwise multiplication on two vectors. This requires one 2^i-point IFFT, two 2^i-point FFTs and 2^i multiplications, and thus the complexity is $\mathcal{O}(2^i \lg(2^i))$.

B. Formal derivative

For a polynomial $D_{2^k}(x)$ in $\mathbb{X}$, we have

$$
\begin{aligned}
D_{2^k}(x) &= \sum_{i=0}^{2^k-1} d_i X_i(x) \\
&= \sum_{i=0}^{2^{k-1}-1} d_i X_i(x) + \sum_{i=2^{k-1}}^{2^k-1} d_i X_i(x) \\
&= \sum_{i=0}^{2^{k-1}-1} d_i X_i(x) + s_{k-1}(x) \sum_{i=0}^{2^{k-1}-1} d_{i+2^{k-1}} X_i(x) \\
&= D_{2^{k-1}}^{(0)}(x) + s_{k-1}(x) D_{2^{k-1}}^{(1)}(x).
\end{aligned}
\tag{81}
$$

The formal derivative of $D_{2^k}(x)$ is given by

$$
\begin{aligned}
D'_{2^k}(x) = &[D_{2^{k-1}}^{(0)}]'(x) + s'_{k-1}(x) D_{2^{k-1}}^{(1)}(x) \\
&+ s_{k-1}(x) [D_{2^{k-1}}^{(1)}]'(x).
\end{aligned}
\tag{82}
$$

From Theorem 1, $s'_{k-1}(x)$ is a constant. $[D_{2^{k-1}}^{(0)}]'(x)$ and $s_{k-1}(x)[D_{2^{k-1}}^{(1)}]'(x)$ can be computed recursively. Let $h = 2^k$, and the recursive form of the complexity is written by $T(h) = 2 \cdot T(h/2) + \mathcal{O}(h)$ and then $T(h) = \mathcal{O}(h \lg(h))$.

11

APPENDIX B
PROOF OF LEMMAS

A. Proof of Lemma 1

Proof. From the definition, $\bar{D}_{2^k}(x)$ can be reformulated as

$$
\begin{aligned}
\bar{D}_{2^k}(x) &= \sum_{i=0}^{2^k-1} \bar{d}_i \bar{X}_i(x) \\
&= \sum_{i=0}^{2^{k-1}-1} \bar{d}_i \bar{X}_i(x) + \sum_{i=2^{k-1}}^{2^k-1} \bar{d}_i \bar{X}_i(x) \\
&= \sum_{i=0}^{2^{k-1}-1} \bar{d}_i \bar{X}_i(x) + \frac{s_{k-1}(x)}{s_{k-1}(v_{k-1})} \sum_{i=0}^{2^{k-1}-1} \bar{d}_{i+2^{k-1}} \bar{X}_i(x) \\
&= \sum_{i=0}^{2^{k-1}-1} \left(\bar{d}_i + \frac{s_{k-1}(x)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}} \right) \bar{X}_i(x).
\end{aligned}
\tag{83}
$$

From Theorem 1, given $\gamma \in \mathbb{F}_{2^m}$, we have

$$
\begin{aligned}
s_{k-1}(a+\gamma) &= s_{k-1}(a) + s_{k-1}(\gamma) \\
&= s_{k-1}(\gamma) \qquad \forall a \in V_{k-1}.
\end{aligned}
\tag{84}
$$

From (83) and (84), we have

$$
\begin{aligned}
&\bar{D}_{2^k}(a+\gamma) \\
&= \sum_{i=0}^{2^{k-1}-1} \left(\bar{d}_i + \frac{s_{k-1}(a+\gamma)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}} \right) \bar{X}_i(a+\gamma) \\
&= \sum_{i=0}^{2^{k-1}-1} \left(\bar{d}_i + \frac{s_{k-1}(\gamma)}{s_{k-1}(v_{k-1})} \bar{d}_{i+2^{k-1}} \right) \bar{X}_i(a+\gamma),
\end{aligned}
\tag{85}
$$

for each $a \in V_{k-1}$. This completes the proof. $\square$

B. Proof of Lemma 2

Proof. From (33), we have

$$
\begin{aligned}
D_a &= \lceil \lg(\deg(a(x)) + y + 1) \rceil \\
&= \lceil \lg(\deg(a(x)) + 2^{D_\ell} - \deg(b(x))) \rceil \\
&\geq \lceil \lg(2^{D_\ell} + 1) \rceil \qquad \text{(From (24))} \\
&= D_\ell + 1.
\end{aligned}
\tag{86}
$$

Moreover,

$$
\begin{aligned}
D_a &= \lceil \lg(\deg(a(x)) + 2^{D_\ell} - \deg(b(x))) \rceil \\
&\leq \lceil \lg(\deg(a(x)) + 2^{D_\ell}) \rceil \\
&\leq \lceil \lg(2^{D_\ell} + 2^{D_\ell}) \rceil = D_\ell + 1.
\end{aligned}
\tag{87}
$$

(86)(87) concludes that $D_a = D_\ell + 1$. This completes the proof. $\square$

C. Proof of Lemma 3

Proof. (38) is a summation of two terms. For the first term, we have

$$
\begin{aligned}
&\deg(Q(x) \cdot H(x)) \\
&\leq \deg(Q(x)) + \deg(B(x)) + 1 \quad \text{(From (32))} \\
&= \deg(A(x)) + 1 \quad \text{(From (30))} \\
&= \deg(a(x)) + y + 1 \quad \text{(From (29))} \\
&= \deg(a(x)) + 2^{D_\ell} - \deg(b(x)) \quad \text{(From (26))} \\
&\leq \deg(a(x)) + 2^{D_\ell} \\
&\leq 2^{D_\ell} - 1 + 2^{D_\ell} \quad \text{(From (27))} \\
&= 2^{D_a} - 1.
\end{aligned}
\tag{88}
$$

For the second term, we have

$$
\begin{aligned}
&\deg(R(x) \cdot \Lambda(x) \cdot s_1(x)) \\
&= \deg(r(x)) + y + 2^{D_\ell} + 1 \\
&= \deg(r(x)) + (2^{D_\ell} - \deg(b(x)) - 1) + 2^{D_\ell} + 1 \quad \text{(From (26))} \\
&= 2^{D_a} + \deg(r(x)) - \deg(b(x)) \\
&\leq 2^{D_a} - 1.
\end{aligned}
\tag{89}
$$

This completes the proof. $\square$

D. Proof of Lemma 4

Proof. The proof follows mathematical induction. For the based case $i = 0$, (41) shows the following holds.

$$
\Lambda_0(x) \cdot B_0(x) \cdot s_1(x) = s_1(x),
$$

and $\bar{r}_0(x) = 0$.

Assume (46) holds at $i = j$. That is,

$$
\Lambda_j(x) \cdot B_j(x) \cdot s_1(x) = s_{j+1}(x) + \bar{r}_j(x),
\tag{90}
$$

which is multiplied by $(s_j(x))^2$ to get

$$
\begin{aligned}
&(s_j(x))^2 \cdot \Lambda_j(x) \cdot B_j(x) \cdot s_1(x) \\
&= (s_j(x))^2 \cdot s_{j+1}(x) + (s_j(x))^2 \cdot \bar{r}_j(x).
\end{aligned}
\tag{91}
$$

From (7), we have

$$
\begin{aligned}
&(s_j(x))^2 \cdot s_{j+1}(x) + (s_j(x))^2 \cdot \bar{r}_j(x) \\
&= (s_{j+1}(x))^2 + s_j(v_j)s_j(x)s_{j+1}(x) + (s_j(x))^2 \cdot \bar{r}_j(x) \\
&= s_{j+2}(x) + s_{j+1}(v_{j+1})s_{j+1}(x) + s_j(v_j)s_j(x)s_{j+1}(x) \\
&\quad + (s_j(x))^2 \cdot \bar{r}_j(x).
\end{aligned}
\tag{92}
$$

By (92)(43), (91) can be rewritten as

$$
s_j(x) \cdot \Lambda_j(x) \cdot B_{j+1}(x) \cdot s_1(x) = s_{j+2}(x) + \hat{r}_j(x), \tag{93}
$$

where

$$
\begin{aligned}
\hat{r}_j(x) &= s_{j+1}(v_{j+1})s_{j+1}(x) + s_j(v_j)s_j(x)s_{j+1}(x) \\
&\quad + (s_j(x))^2 \cdot \bar{r}_j(x) \\
&\quad + s_j(x) \cdot \Lambda_j(x) \cdot \bar{B}_j(x) \cdot s_1(x).
\end{aligned}
\tag{94}
$$

The degree of each term of $\hat{r}_j(x)$ is

$$
\deg(s_{j+1}(v_{j+1})s_{j+1}(x)) = 2^{j+1},
$$

$$
\deg(s_j(v_j)s_j(x)s_{j+1}(x)) = 2^j + 2^{j+1},
$$

12

$$\deg((s_j(x))^2 \cdot \bar{r}_j(x)) \leq 2^{j+1} + 2^j,$$

$$\deg(s_j(x) \cdot \Lambda_j(x) \cdot \bar{B}_j(x) \cdot s_1(x))$$
$$\leq 2^j + (2^j - 1) + (2^j - 1) + 2.$$

Thus, we have $\deg(\hat{r}_j(x)) \leq 2^{j+1} + 2^j$.

When $i = j + 1$, from (45)(44), we have

$$\Lambda_{j+1}(x) = \mathbb{Q}((s_j(x) \cdot \Lambda_j(x))^2 \cdot B_{j+1}(x) \cdot s_1(x), j+2).$$

The above equation can be rewritten as

$$\Lambda_{j+1}(x) \cdot s_{j+2}(x) + \check{r}_{j+2}(x) \tag{95}$$
$$= (s_j(x) \cdot \Lambda_j(x))^2 \cdot B_{j+1}(x) \cdot s_1(x),$$

where $\deg(\check{r}_{j+2}(x)) \leq 2^{j+2} - 1$. We then multiply (95) by $B_{j+1}(x) \cdot s_1(x)$ to obtain

$$\begin{aligned}
& \Lambda_{j+1}(x) \cdot B_{j+1}(x) \cdot s_1(x) \cdot s_{j+2}(x) \\
& + \check{r}_{j+2}(x) \cdot B_{j+1}(x) \cdot s_1(x) \\
= & (s_j(x) \cdot \Lambda_j(x) \cdot B_{j+1}(x) \cdot s_1(x))^2 \\
= & (s_{j+2}(x) + \hat{r}_j(x))^2 \qquad \text{(By (93))} \\
= & (s_{j+2}(x))^2 + (\hat{r}_j(x))^2.
\end{aligned} \tag{96}$$

(96) is then divided by $s_{j+2}(x)$ to get

$$\Lambda_{j+1}(x) \cdot B_{j+1}(x) \cdot s_1(x) = s_{j+2}(x) + \bar{r}_{j+1}(x), \tag{97}$$

where

$$\bar{r}_{j+1}(x) = \frac{(\hat{r}_j(x))^2 - \check{r}_{j+2}(x) \cdot B_{j+1}(x) \cdot s_1(x)}{s_{j+2}(x)}.$$

In (97), the degree of each term of $\deg(\bar{r}_{j+1}(x))$ is as follows:

$$\begin{aligned}
\deg((\hat{r}_j)^2(x)) & \leq 2 \cdot (2^{j+1} + 2^j), \\
\deg(\check{r}_{j+2}(x) \cdot B_{j+1}(x) \cdot s_1(x)) & \\
\leq (2^{j+2} - 1) & + (2^{j+1} - 1) + 2, \\
\deg(s_{j+2}(x)) & = 2^{j+2}.
\end{aligned}$$

Thus, $\deg(\bar{r}_{j+1}(x)) \leq 2^{j+1}$. This completes the proof. $\square$

E. Proof of Lemma 5

Proof. From (44), we have

$$\begin{aligned}
& \Lambda_i(x) \\
= & \mathbb{Q}((s_{i-1}(x))^2 \cdot \bar{\Lambda}_i(x), i+1) \\
= & \mathbb{Q}((s_i(x) + s_{i-1}(v_{i-1})s_{i-1}(x)) \cdot \bar{\Lambda}_i(x), i+1) \quad \text{(From (7))} \\
= & \mathbb{Q}(s_i(x)\bar{\Lambda}_i(x), i+1) \\
& + s_{i-1}(v_{i-1}) \cdot \mathbb{Q}(s_{i-1}(x)\bar{\Lambda}_i(x), i+1)
\end{aligned} \tag{98}$$

(98) has two terms, and we recalled that $\deg(\bar{\Lambda}_i(x)) = 2^{i+1} - 1$. Let

$$\bar{\Lambda}_i(x) = \Lambda_i^{(0)}(x) + s_i(x)\Lambda_i^{(1)}(x),$$

where both $\Lambda_i^{(0)}(x)$ and $\Lambda_i^{(1)}(x)$ have degrees no more than $2^i - 1$. Then

$$\begin{aligned}
& s_i(x)\bar{\Lambda}_i(x) \\
= & s_i(x)(\Lambda_i^{(0)}(x) + s_i(x)\Lambda_i^{(1)}(x)) \\
= & s_i(x)\Lambda_i^{(0)}(x) + (s_i(x))^2 \Lambda_i^{(1)}(x) \\
= & s_i(x)\Lambda_i^{(0)}(x) + s_i(v_i)s_i(x)\Lambda_i^{(1)}(x) \\
& + s_{i+1}(x)\Lambda_i^{(1)}(x).
\end{aligned} \tag{99}$$

From (99), the first term in (98) can be reformulated as

$$\mathbb{Q}(s_i(x)\bar{\Lambda}_i(x), i+1) = \Lambda_i^{(1)}(x) = \mathbb{Q}(\bar{\Lambda}_i(x), i).$$

With the similar step, it can be shown that the second stem can be formulated as

$$\begin{aligned}
& s_{i-1}(v_{i-1})\mathbb{Q}(s_{i-1}(x)\bar{\Lambda}_i(x), i+1) \\
= & \mathbb{Q}(\Lambda_i^{(1)}(x), i-1) \cdot s_{i-1}(v_{i-1}).
\end{aligned}$$

This completes the proof. $\square$

F. Proof of Lemma 6

Proof. For the based case $\deg(b(x)) < 2^{g-1}$ (see Algorithm 5, line 1), it is clear that (48) holds.

Assume Algorithm 5 is valid for $\text{HGCD}(a(x), b(x), j)$ with $j \leq g-1$. When $j = g$, the degree of $a(x)$ is between $2^{g-1} \leq \deg(a(x)) \leq 2^g - 1$. In this case, both $a(x)$ and $b(x)$ are divided into three individual polynomials as expressed in (56) and (57). In line 3, $\text{HGCD}(a_H(x), b_H(x), g-1)$ is called to obtain (Z_H, M_H), that possesses

$$Z_H = M_H \cdot \begin{bmatrix} a_H(x) \\ b_H(x) \end{bmatrix}. \tag{100}$$

Multiplying (100) by $s_{g-1}(x)$ to obtain

$$Z_H \cdot s_{g-1}(x) = M_H \cdot s_{g-1}(x) \begin{bmatrix} a_H(x) \\ b_H(x) \end{bmatrix} \tag{101}$$

which is equivalent to

$$\begin{aligned}
& Z_H \cdot s_{g-1}(x) + M_H \begin{bmatrix} a_L(x) \\ b_L(x) \end{bmatrix} \\
= & M_H \begin{bmatrix} s_{g-1}(x) \cdot a_H(x) + a_L(x) \\ s_{g-1}(x) \cdot b_H(x) + b_L(x) \end{bmatrix}.
\end{aligned} \tag{102}$$

By (56) and (57), (102) becomes

$$Z_H \cdot s_{g-1}(x) + M_H \begin{bmatrix} a_L(x) \\ b_L(x) \end{bmatrix} = M_H \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}. \tag{103}$$

Then we have

$$\begin{bmatrix} z_{M0}(x) \\ z_{M1}(x) \end{bmatrix} = M_H \begin{bmatrix} a(x) \\ b(x) \end{bmatrix} \quad \text{and} \quad Z_M = M_H \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}. \tag{104}$$

Note that $z_{M0}(x)$ and $z_{M1}(x)$ are computed in line 4. (104) shows that (Z_M, M_H) satisfies the equality, and thus the return in line 5 is valid.

In line 7, $z_{M0}(x)$ is divided by $z_{M1}(x)$ to get

$$z_{M0}(x) = q_M(x) \cdot z_{M1}(x) + r_M(x), \tag{105}$$

with

$$\deg(r_M(x)) < \deg(z_{M1}(x)). \tag{106}$$

The matrix form of (105) can be reformulated as

$$\begin{bmatrix} z_{M1}(x) \\ r_M(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_M(x) \end{bmatrix} \begin{bmatrix} z_{M0}(x) \\ z_{M1}(x) \end{bmatrix}. \tag{107}$$

13

Then $z_{\mathrm{M1}}(x)$ and $r_{\mathrm{M}}(x)$ are decomposed into several polynomials as

$$
\begin{aligned}
&z_{\mathrm{M1}}(x)\\
&=z_{\mathrm{M1LL}}(x)+s_{g-2}(x)z_{\mathrm{M1LH}}(x)+s_{g-1}(x)z_{\mathrm{M1H}}(x)\\
&=z_{\mathrm{M1LL}}(x)+s_{g-2}(x)z_{\mathrm{M1LH}}(x)\\
&\quad+s_{g-2}(x)(s_{g-2}(x)+s_{g-2}(v_{g-2}))z_{\mathrm{M1H}}(x)\\
&=z_{\mathrm{M1LL}}(x)+s_{g-2}(x)z_{\mathrm{M1M}}(x),
\end{aligned}
\tag{108}
$$

where

$$
\begin{aligned}
&z_{\mathrm{M1M}}(x)\\
&=z_{\mathrm{M1LH}}(x)+(s_{g-2}(x)+s_{g-2}(v_{g-2}))z_{\mathrm{M1H}}(x).
\end{aligned}
\tag{109}
$$

Similarly,

$$
r_{\mathrm{M}}(x)=r_{\mathrm{MLL}}(x)+s_{g-2}(x)r_{\mathrm{MM}}(x),
\tag{110}
$$

where

$$
\begin{aligned}
&r_{\mathrm{MM}}(x)\\
&=r_{\mathrm{MLH}}(x)+(s_{g-2}(x)+s_{g-2}(v_{g-2}))r_{\mathrm{MH}}(x).
\end{aligned}
\tag{111}
$$

$z_{\mathrm{M1M}}(x)$ (and $r_{\mathrm{MM}}(x)$) can be treated as the quotient of dividing $z_{\mathrm{M1}}(x)$ (and $r_{\mathrm{M}}(x)$) by $s_{g-2}(x)$. By (106), this implies

$$
\deg(r_{\mathrm{MM}}(x))<\deg(z_{\mathrm{M1M}}(x)).
$$

Line 9 calls $\mathrm{HGCD}(z_{\mathrm{M1M}}(x),r_{\mathrm{MM}}(x),g-1)$ to obtain $(Y_{\mathrm{M}},M_{\mathrm{M}})$ possessing

$$
Y_{\mathrm{M}}=M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1M}}(x)\\ r_{\mathrm{MM}}(x)\end{bmatrix}.
\tag{112}
$$

Multiplying (112) by $s_{g-2}(x)$ to obtain

$$
Y_{\mathrm{M}}\cdot s_{g-2}(x)=M_{\mathrm{M}}\cdot s_{g-2}(x)\begin{bmatrix}z_{\mathrm{M1M}}(x)\\ r_{\mathrm{MM}}(x).\end{bmatrix}
\tag{113}
$$

By adding $\begin{bmatrix}z_{\mathrm{M1LL}}(x)\\ r_{\mathrm{MLL}}(x)\end{bmatrix}$ to both side of (113), we have

$$
\begin{aligned}
&Y_{\mathrm{M}}\cdot s_{g-2}(x)+M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1LL}}(x)\\ r_{\mathrm{MLL}}(x)\end{bmatrix}\\
&=M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1LL}}(x)+z_{\mathrm{M1M}}(x)\cdot s_{g-2}(x)\\ r_{\mathrm{MLL}}(x)+r_{\mathrm{MM}}(x)\cdot s_{g-2}(x)\end{bmatrix}
\end{aligned}
\tag{114}
$$

which is equivalent to

$$
Y_{\mathrm{M}}\cdot s_{g-2}(x)+M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1LL}}(x)\\ r_{\mathrm{MLL}}(x)\end{bmatrix}=M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1}}(x)\\ r_{\mathrm{M}}(x)\end{bmatrix}.
\tag{115}
$$

Substituting (107) and (104) into (115) to obtain

$$
\begin{aligned}
&Y_{\mathrm{M}}\cdot s_{g-2}(x)+M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1LL}}(x)\\ r_{\mathrm{MLL}}(x)\end{bmatrix}\\
&=M_{\mathrm{M}}\begin{bmatrix}0&1\\ 1&-q_{\mathrm{M}}(x)\end{bmatrix}M_{\mathrm{H}}\begin{bmatrix}a(x)\\ b(x)\end{bmatrix}.
\end{aligned}
\tag{116}
$$

Hence, we have

$$
Z_{\mathrm{R}}=M_{\mathrm{R}}\begin{bmatrix}a(x)\\ b(x)\end{bmatrix},
\tag{117}
$$

where

$$
M_{\mathrm{R}}=M_{\mathrm{M}}\begin{bmatrix}0&1\\ 1&-q_{\mathrm{M}}(x)\end{bmatrix}M_{\mathrm{H}}
\tag{118}
$$

and

$$
Z_{\mathrm{R}}=Y_{\mathrm{M}}\cdot s_{g-2}(x)+M_{\mathrm{M}}\begin{bmatrix}z_{\mathrm{M1LL}}(x)\\ r_{\mathrm{MLL}}(x)\end{bmatrix}
\tag{119}
$$

are the return results in Line 10.   □

G. Proof of Lemma 7

Proof. Assume $\mathrm{HGCD}(a(x),b(x),i)$ is valid for $i\le g-1$, i.e., the recursive calls in line 3 (and line 9) are valid. Assume $i=q$. It is clear that the call at line 3 satisfies the condition, since $a_{\mathrm{H}}(x)$ and $b_{\mathrm{H}}(x)$ are the high degree portions of $a(x)$ and $b(x)$, respectively.

For the call at line 9, we first consider the degree of $z_{\mathrm{M1}}(x)$. For simplicity, $(Z_{\mathrm{H}},M_{\mathrm{H}})$ is denoted as

$$
Z_{\mathrm{H}}=\begin{bmatrix}z_{\mathrm{H0}}(x)\\ z_{\mathrm{H1}}(x)\end{bmatrix},M_{\mathrm{H}}=\begin{bmatrix}m_{\mathrm{H00}}(x)&m_{\mathrm{H01}}(x)\\ m_{\mathrm{H10}}(x)&m_{\mathrm{H11}}(x)\end{bmatrix}.
$$

Because $\deg(a_{\mathrm{L}}(x))\le 2^{q-2}-1$, $\deg(b_{\mathrm{L}}(x))\le 2^{q-2}-1$, and $\deg(z_{\mathrm{H1}}(x))\le 2^{q-2}-1$, from the assumption, we have

$$
\begin{aligned}
\deg(z_{\mathrm{M1}}(x))\le\max\{&(2^{q-2}-1)+2^{g-1},\\
&\deg(m_{\mathrm{H10}}(x))+(2^{q-2}-1),\\
&\deg(m_{\mathrm{H11}}(x))+(2^{q-2}-1)\}.
\end{aligned}
\tag{120}
$$

From the assumption,

$$
\deg(m_{\mathrm{H10}}(x))\le\deg(m_{\mathrm{H11}}(x))\le\deg(a_{\mathrm{H}}(x))-\deg(z_{\mathrm{H0}}(x)).
$$

As $\deg(a_{\mathrm{H}}(x))\le 2^{q-1}-1$ and $\deg(z_{\mathrm{H0}}(x))\ge 2^{q-2}$, we have

$$
\deg(m_{\mathrm{H10}}(x))\le\deg(m_{\mathrm{H11}}(x))\le 2^{q-2}-1.
$$

Then (120) gives

$$
\begin{aligned}
&\deg(z_{\mathrm{M1}}(x))\\
&\le\max\{(2^{g-2}-1)+2^{g-1},(2^{q-2}-1)+(2^{q-2}-1)\}\\
&=(2^{g-2}-1)+2^{g-1}.
\end{aligned}
\tag{121}
$$

Thus, the inequality

$$
\deg(r_{\mathrm{M}}(x))<\deg(z_{\mathrm{M1}}(x))\le 2^{g-1}+2^{g-2}-1
$$

in line 7 is valid. In line 8, $z_{\mathrm{M1M}}(x)$ and $r_{\mathrm{MM}}(x)$ are the quotients of dividing $r_{\mathrm{M}}(x)$ and $z_{\mathrm{M1}}(x)$ by $s_{g-2}(x)$, and then

$$
\deg(r_{\mathrm{MM}}(x))<\deg(z_{\mathrm{M1M}}(x))\le 2^{g-1}-1.
\tag{122}
$$

Further, due to the if condition in line 5, we have

$$
\deg(z_{\mathrm{M1}}(x))\ge 2^{g-1}
$$

after line 7. This implies

$$
\deg(z_{\mathrm{M1M}}(x))\ge 2^{g-2}.
\tag{123}
$$

By (122) and (123), the requirements of the call in line 9 are verified.   □

H. Proof of Lemma 8

Proof. Algorithm 5 has three returns at lines 1, 5 and 10. Assume that the recursive call HGCD in line 3 and line 10 outputs the valid results. For line 1, it is clear to see it. For line 5, (101)-(104) show that the degree of $z_{\mathrm{M0}}(x)$ is at least

$$
\deg(z_{\mathrm{M0}}(x))\ge\deg(z_{\mathrm{H0}})+2^{g-1}.
$$

By the assumption, $\deg(z_{\mathrm{H0}})\ge 2^{g-2}$ and we have

$$
\deg(z_{\mathrm{M0}}(x))\ge 2^{g-2}+2^{g-1}.
\tag{124}
$$

14

By (124) and the if condition in line 5, the first condition holds.

Let us consider line 10. Z_R, Y_M and M_M can be denoted as

$$Z_R = \begin{bmatrix} z_{R0}(x) \\ z_{R1}(x) \end{bmatrix}, Y_M = \begin{bmatrix} y_{M0}(x) \\ y_{M1}(x) \end{bmatrix},$$
$$M_M = \begin{bmatrix} m_{M00}(x) & m_{M01}(x) \\ m_{M10}(x) & m_{M11}(x) \end{bmatrix}.$$

The degree of $z_{R0}(x)$ is at least

$$\deg(z_{R0}(x)) \geq \deg(y_{M0}(x)) + 2^{g-2}$$
$$\geq 2^{g-2} + 2^{g-2} = 2^{g-1}.$$

Further,

$$\deg(z_{R1}(x)) = \max\{ \deg(y_{M1}(x)) + 2^{g-2},$$
$$\deg(m_{M10}(x)z_{M1LL}(x)), \quad (125)$$
$$\deg(m_{M11}(x)r_{MLL}(x))\}.$$

By assumptions, we have

$$\deg(m_{M10}(x)) \leq \deg(m_{M11}(x))$$
$$\leq \deg(z_{M1M}(x)) - \deg(y_{M0}(x))$$
$$\leq (2^{q-1} - 1) - 2^{q-2} = 2^{q-2} - 1.$$

Then (125) becomes

$$\deg(z_{R1}(x))$$
$$\leq \max\{(2^{g-2} - 1) + 2^{g-2}, (2^{q-2} - 1) + (2^{q-2} - 1)\}$$
$$= 2^{g-1} - 1,$$
$$(126)$$

as, by assumptions, $\deg(z_{M1LL}(x)) \leq 2^{q-2} - 1$ and $\deg(r_{MLL}(x)) \leq 2^{q-2} - 1$, and $\deg(y_{M1}(x)) \leq 2^{g-2} - 1$. This verifies (60). $\qquad\square$

I. Proof of Lemma 9

Proof. Assume that the recursive call HGCD in line 3 and line 10 outputs the valid results. For the base case in line 1, it is clear that the condition holds. Notice that $\deg(0)$ is a special case, we can treat $\deg(0) = 0$ in this case. For line 5, the objective is to prove

$$\deg(m_{H11}(x)) \leq \deg(a(x)) - \deg(z_{M0}(x)), \quad (127)$$

and

$$\deg(m_{Hi0}(x)) \leq \deg(m_{Hi1}(x)),$$
$$\deg(m_{H0i}(x)) \leq \deg(m_{H1i}(x)), \quad i = 0, 1. \quad (128)$$

By assumptions, line 3 of the algorithm gives

$$\deg(m_{H11}(x)) \leq \deg(a_H(x)) - \deg(z_{H0}(x)), \quad (129)$$

and

$$\deg(m_{Hi0}(x)) \leq \deg(m_{Hi1}(x)),$$
$$\deg(m_{H0i}(x)) \leq \deg(m_{H1i}(x)), \quad i = 0, 1. \quad (130)$$

(130) verifies that (128) is true. Further, (101)-(104) show that (129) can be reformed as

$$\deg(m_{H11}(x))$$
$$\leq \deg(a_H(x)s_{g-1}(x)) - \deg(z_{H0}(x)s_{g-1}(x)) \quad (131)$$
$$= \deg(a(x)) - \deg(z_{M0}(x)).$$

This verifies (127).

Let us consider line 10. Z_R, Y_M M_M, and M_R can be denoted as

$$Z_R = \begin{bmatrix} z_{R0}(x) \\ z_{R1}(x) \end{bmatrix}, Y_M = \begin{bmatrix} y_{M0}(x) \\ y_{M1}(x) \end{bmatrix},$$
$$M_M = \begin{bmatrix} m_{M00}(x) & m_{M01}(x) \\ m_{M10}(x) & m_{M11}(x) \end{bmatrix}. M_R = \begin{bmatrix} m_{R00}(x) & m_{R01}(x) \\ m_{R10}(x) & m_{R11}(x) \end{bmatrix}.$$

The objective is to prove

$$\deg(m_{R11}(x)) \leq \deg(a(x)) - \deg(z_{R0}(x)), \quad (132)$$

and

$$\deg(m_{Ri0}(x)) \leq \deg(m_{Ri1}(x)),$$
$$\deg(m_{R0i}(x)) \leq \deg(m_{R1i}(x)), \quad i = 0, 1. \quad (133)$$

By assumptions, line 9 of the algorithm gives

$$\deg(m_{M11}(x)) \leq \deg(z_{M1M}(x)) - \deg(y_{M0}(x)), \quad (134)$$

and

$$\deg(m_{Mi0}(x)) \leq \deg(m_{Mi1}(x)),$$
$$\deg(m_{M0i}(x)) \leq \deg(m_{M1i}(x)), \quad i = 0, 1. \quad (135)$$

To verify (133), (118) can be reformed as

$$M_R$$
$$= \begin{bmatrix} m_{M00}(x) & m_{M01}(x) \\ m_{M10}(x) & m_{M11}(x) \end{bmatrix} \begin{bmatrix} m_{H10}(x) & m_{H11}(x) \\ m_{H00}(x) & m_{H01}(x) \end{bmatrix} \quad (136)$$
$$- q_M(x) \begin{bmatrix} m_{M01}(x) \\ m_{M11}(x) \end{bmatrix} \begin{bmatrix} m_{H10}(x) & m_{H11}(x) \end{bmatrix}.$$

Based on assumptions (130) and (133), it can be seen that the degrees of the elements of M_R are determined by the second term. Precisely,

$$\deg(m_{R00}(x)) = \deg(q_M(x)) + \deg(m_{M01}(x)) + \deg(m_{H10}(x)),$$
$$\deg(m_{R01}(x)) = \deg(q_M(x)) + \deg(m_{M01}(x)) + \deg(m_{H11}(x)),$$
$$\deg(m_{R10}(x)) = \deg(q_M(x)) + \deg(m_{M11}(x)) + \deg(m_{H10}(x)),$$
$$\deg(m_{R11}(x)) = \deg(q_M(x)) + \deg(m_{M11}(x)) + \deg(m_{H11}(x)).$$
$$(137)$$

From the assumptions $\deg(m_{M01}(x)) \leq \deg(m_{M11}(x))$ and $\deg(m_{H10}(x)) \leq \deg(m_{H11}(x))$, (133) can be verified.

The verification of (132) is considered as follows. From (111), (134) can be reformed as

$$\deg(m_{M11}(x))$$
$$\leq \deg(z_{M1M}(x)s_{g-2}(x)) - \deg(y_{M0}(x)s_{g-2}(x)) \quad (138)$$
$$= \deg(z_{M1}(x)) - \deg(z_{R0}(x)).$$

(138) is summed by (131), resulting in (139), and thus (132) is verified. $\qquad\square$

J. Proof of Lemma 10

Proof. The proof follows mathematical induction. We pick $T = n/2$ as the base case. Then from (63), $\mathbf{u} = (\mathbf{u}_0, \omega_0)$ has $n/2$ ω_0s in the high degree part. From (66), $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1)$ is divided into two equal sub-vectors. Then (65) can be written as

$$(\mathbf{u}_0, \omega_0) = \text{IFFT}_{\bar{\mathbb{X}}}((\mathbf{v}_0, \mathbf{v}_1), m, \omega_0).$$

15

In Algorithm 2, Line 3 computes $D^{(0)} = \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_0, m - 1, \omega_0)$, and Line 4 computes $D^{(1)} = \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_1, m-1, \omega_{n/2})$. The vector ω_0 is calculated by line 6, and $\mathbf{u}_0$ is computed by line 7. As line 6 only requires pointwise additions, which can be written as a vector addition:

$$\omega_0 = D^{(0)} + D^{(1)}$$
$$=\text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_0, m - 1, \omega_0) + \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_1, m-1, \omega_{n/2}). \quad (140)$$

Assume (67) holds at $T = S = 2^s$, and thus

$$\omega_0 = \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_0, s, \omega_0) + \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_1, s, \omega_S) + \ldots$$
$$+ \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_{n/S-1}, s, \omega_{n-S}). \quad (141)$$

When $T = S/2 = 2^{s-1}$, (141) becomes

$$(\mathbf{u}_{n/(S/2)-2}, \omega_0)$$
$$=\text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_0, s, \omega_0) + \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_1, s, \omega_S) + \ldots \quad (142)$$
$$+ \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_{n/S-1}.s, \omega_{n-S}).$$

We can extract the computations regarding ω_0 in (142). Similarly, this decomposes each s-point IFFT into two $(s/2)$-point IFFTs, resulting in

$$\omega_0 = \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_0, s - 1, \omega_0) + \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_1, s - 1, \omega_{S/2}) + \ldots$$
$$+ \text{IFFT}_{\overline{\mathbb{X}}}(\mathbf{v}_{n/S-1}, s - 1, \omega_{n-S/2}), \quad (143)$$

This completes the proof. $\square$

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.

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.

$$\deg(m_{\text{M11}}(x)) + \deg(m_{\text{H11}}(x)) \leq \deg(z_{\text{M1}}(x)) - \deg(z_{\text{R0}}(x)) + \deg(a(x)) - \deg(z_{\text{M0}}(x))$$
$$\Rightarrow \deg(m_{\text{M11}}(x)) + \deg(m_{\text{H11}}(x)) + \deg(z_{\text{M0}}(x)) - \deg(z_{\text{M1}}(x)) \leq \deg(a(x)) - \deg(z_{\text{R0}}(x))$$
$$\Rightarrow \deg(m_{\text{M11}}(x)) + \deg(m_{\text{H11}}(x)) + \deg(q_{\text{M}}(x)) \leq \deg(a(x)) - \deg(z_{\text{R0}}(x)) \quad (By~(105)) \quad (139)$$
$$\Rightarrow \deg(m_{\text{R11}}(x)) \leq \deg(a(x)) - \deg(z_{\text{R0}}(x)) \quad (By~(137)),$$