2014 IEEE Annual Symposium on Foundations of Computer Science

Novel Polynomial Basis and Its Application to Reed-Solomon Erasure Codes

Sian-Jheng Lin, Wei-Ho Chung
Research Center for Information Technology Innovation
Academia Sinica
Taipei City, Taiwan
Email: sjhenglin@gmail.com; whc@citi.sinica.edu.tw

Yunghsiang S. Han
Department of Electrical Engineering
National Taiwan University of Science and Technology
Taipei City, Taiwan
Email: yshan@mail.ntust.edu.tw

***Abstract*—In this paper, we present a new basis of polynomial over finite fields of characteristic two and then apply it to the encoding/decoding of Reed-Solomon erasure codes. The proposed polynomial basis allows that h-point polynomial evaluation can be computed in $O(h\log_2(h))$ finite field operations with small leading constant. As compared with the canonical polynomial basis, the proposed basis improves the arithmetic complexity of addition, multiplication, and the determination of polynomial degree from $O(h\log_2(h)\log_2\log_2(h))$ to $O(h\log_2(h))$. Based on this basis, we then develop the encoding and erasure decoding algorithms for the $(n=2^r,k)$ Reed-Solomon codes. Thanks to the efficiency of transform based on the polynomial basis, the encoding can be completed in $O(n\log_2(k))$ finite field operations, and the erasure decoding in $O(n\log_2(n))$ finite field operations. To the best of our knowledge, this is the first approach supporting Reed-Solomon erasure codes over characteristic-2 finite fields while achieving a complexity of $O(n\log_2(n))$, in both additive and multiplicative complexities. As the complexity leading factor is small, the algorithms are advantageous in practical applications.**

I. Introduction

For a positive integer $r\geq 1$, let $\mathbb{F}_{2^r}$ denote a characteristic-2 finite field containing 2^r elements. A polynomial over $\mathbb{F}_{2^r}$ is defined as

$$a(x)=a_0+a_1x+a_2x^2+\cdots+a_{h-1}x^{h-1},$$

where each $a_i\in\mathbb{F}_{2^r}$. A fundamental issue is to reduce the computational complexities of arithmetic operations over polynomials. Many fast polynomial-related algorithms, such as Reed-Solomon codes, are based on fast Fourier transforms (FFT). However, it is algorithmically harder as the traditional fast Fourier transform (FFT) cannot be applied directly over a characteristic-2 finite fields. To the best of our knowledge, no existing algorithm for characteristic-2 finite field FFT/polynomial multiplication has provably achieved $O(h\lg(h))$ operations[1] (see Section VII for more details).

In algorithmic viewpoint, FFT is a polynomial evaluations at a period of consecutive points, where the polynomial is in monomial basis. This viewpoint gives us the ability to design fast polynomial-related algorithms. In this paper, we present a new polynomial basis in the polynomial ring $\mathbb{F}_{2^r}[x]/(x^{2^r}-x)$. Then a transform in the new basis is defined to compute the polynomial evaluations. The new basis possesses a recursive structure which can be exploited to compute the polynomial evaluations at a period of h consecutive points in time $O(h\lg(h))$ with small leading constant. Furthermore, the recursive structure also works in formal derivative with time complexity $O(h\lg(h))$.

An application of the proposed polynomial basis is in erasure codes, that is an error-correcting code by converting a message of k symbols into a codeword with n symbols such that the original message can be recovered from a subset of the n symbols. An (n,k) erasure code is called Maximum Distance Separable (MDS) if any k out of the n codeword symbols are sufficient to reconstruct the original message. A typical class of MDS codes is Reed-Solomon (RS) codes [1]. Nowadays, RS codes have been applied to many applications, such as RAID systems [2, 3], distributed storage codes [4, 5], and data carousel [6]. Hence, the computational complexity of RS erasure code is considered crucial and has attracted substantial research attention. Based on the new polynomial basis, this paper presents the encoding/decoding algorithms for RS erasure codes. The proposed algorithms use the structure [7] that requires evaluating a polynomial and it's derivatives, while the polynomial used in the structure is in the new polynomial basis, rather than the monomial basis.

The rest of this paper is organized as follows. The proposed polynomial basis is defined in Section II. Section III gives the definition and algorithm of the transform to compute the polynomial evaluations based on the proposed polynomial basis. Section IV shows the formal derivative of polynomial. Section V presents the encoding and erasure decoding algorithm for Reed-Solomon codes. The discussions and comparisons are placed in Section VI. SectionVII reviews some related literature. Concluding remarks are provided in Section VIII.

II. A new polynomial basis over $\mathbb{F}_{2^r}$

A. Finite field arithmetic

Let $\mathbb{F}_{2^r}$ be an extension finite field with dimension r over $\mathbb{F}_2$. The elements of $\mathbb{F}_{2^r}$ are represented as a set $\{\omega_i\}_{i=0}^{2^r-1}$.

[1] Throughout this paper, the notation $\lg(x)$ represents the logarithm to the base 2.

0272-5428/14 $31.00 © 2014 IEEE
DOI 10.1109/FOCS.2014.41

We order those elements as follows. Assume that V be the r-dimensional vector space spanned by $v_0, v_1, \dots, v_{r-1} \in \mathbb{F}_{2^r}$ over $\mathbb{F}_2$. For any $0 \le i < 2^r$, its binary representation is given as

$$i = i_0 + i_1 \cdot 2 + i_2 \cdot 2^2 + \dots + i_{r-1} \cdot 2^{r-1}, \forall i_j \in \{0,1\}. \quad (1)$$

Then ω_i is defined as

$$\omega_i = i_0 \cdot v_0 + i_1 \cdot v_1 + i_2 \cdot v_2 + \dots + i_{r-1} \cdot v_{r-1}.$$

A polynomial $f(x)$ defined over $\mathbb{F}_{2^r}$ is a polynomial whose coefficients are from $\mathbb{F}_{2^r}$.

B. Subspace vanishing polynomial

The subspace vanishing polynomial defined in [8–10] is expressed as

$$W_j(x) = \prod_{i=0}^{2^j-1} (x + \omega_i), \quad (2)$$

where $0 \le j \le r-1$. It can be seen that $deg(W_j(x)) = 2^j$.

Next we present properties of $W_j(x)$ without proof.

Lemma 1 ([9])**.** *$W_j(x)$ is an $\mathbb{F}_2$-linearlized polynomial for which*

$$W_j(x) = \sum_{i=0}^{j} a_{j,i} x^{2^i}, \quad (3)$$

where each $a_{j,i} \in \mathbb{F}_{2^r}$ is a constant. Furthermore,

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

C. Polynomial basis

In this work, we consider the polynomial ring $\mathbb{F}_{2^r}[x]/(x^{2^r} - x)$. A form of polynomial basis we work with is denoted as $\mathbb{X}(x) = (X_0(x), X_1(x), \dots, X_{2^r-1}(x))$ over $\mathbb{F}_{2^r}$. Each polynomial $X_i(x)$ is defined as the product of subspace vanishing polynomials. For each polynomial $X_i(x)$, i is written in binary representation as

$$i = i_0 + i_1 \cdot 2 + \dots + i_{r-1} \cdot 2^{r-1}, \forall i_j \in \{0,1\}. \quad (5)$$

The polynomial $X_i(x)$ is then defined as

$$X_i(x) = \prod_{j=0}^{r-1} \left(\frac{W_j(x)}{W_j(\omega_{2^j})} \right)^{i_j}, \quad (6)$$

for $0 \le i < 2^r$. Notice that $\left(\frac{W_j(x)}{W_j(\omega_{2^j})} \right)^{i_j} = 1$, if $i_j = 0$. It can be seen that $deg(X_i(x)) = i$.

Then a form of polynomial expression $[\bullet](x)$ is given as follows.

Definition 1. *A form of polynomial expression over $\mathbb{F}_{2^r}$ is defined as*

$$[D_h](x) = \sum_{i=0}^{h-1} d_i X_i(x), \quad (7)$$

where

$$D_h = (d_0, d_1, \dots, d_{h-1}) \quad (8)$$

is an h-element vector denoting the polynomial coefficients and $h \le 2^r$. Consequently, $deg([D_h](x)) \le h-1$.

III. Fast transform $\Psi_h^l[\bullet]$

In this section, we define a h-point transformation $\Psi_h^l[\bullet]$ that computes the evaluations of $[\bullet](x)$ at h successive points, for h a power of two. Given a h-element input vector D_h, the polynomial $[D_h](x)$ can be constructed accordingly. The transform outputs a h-element vector

$$\hat{D}_h^l = \Psi_h^l[D_h],$$

where

$$\hat{D}_h^l = ([D_h](\omega_0 + \omega_l), [D_h](\omega_1 + \omega_l), \dots, [D_h](\omega_{h-1} + \omega_l)),$$

and l denotes the amount of shift in the transform.

Oppositely, the inversion, denoted as $(\Psi_h^l)^{-1}[\bullet]$, can convert $\hat{D}_h^l$ into D_h, and we have $(\Psi_h^l)^{-1}[\hat{D}_h^l] = D_h$. Here, we omit to provide the close form for inversion. Instead, an algorithm for transform $\Psi_h^l[\bullet]$ and the inverse algorithm will be presented later.

A. Recursive structure in polynomial basis

This subsection shows that the polynomial $[D_h](x)$ can be formulated as a recursive function $[D_h](x) = \Delta_0^0(x)$, where the function $\Delta_i^m(x)$ is defined as

$$\Delta_i^m(x) = \Delta_{i+1}^m(x) + \frac{W_i(x)}{W_i(\omega_{2^i})} \Delta_{i+1}^{m+2^i}(x) \quad , \text{ for } 0 \le i \le \lg(h) - 1; \quad (9)$$

$$\Delta_{\lg(h)}^m(x) = d_m, \text{ for } 0 \le m \le h-1. \quad (10)$$

Note that m in $\Delta_i^m(x)$ represents a $\lg(h)$-bits binary integer

$$m = m_0 + m_1 \cdot 2 + \dots + m_{i-1} \cdot 2^i, \forall m_j \in \{0,1\}. \quad (11)$$

By induction, it can be seen that $deg(\Delta_i^m(x)) \le h/2^i - 1$. For example, if $h = 8$, we have

$$\begin{aligned}
[D_8](x) = &\sum_{i=0}^{7} d_i X_i(x) \\
=&d_0 + d_1 \frac{W_0(x)}{W_0(\omega_1)} + d_2 \frac{W_1(x)}{W_1(\omega_2)} + d_3 \frac{W_0(x)}{W_0(\omega_1)} \frac{W_1(x)}{W_1(\omega_2)} \\
&+ d_4 \frac{W_2(x)}{W_2(\omega_4)} + d_5 \frac{W_0(x)}{W_0(\omega_1)} \frac{W_2(x)}{W_2(\omega_4)} + d_6 \frac{W_1(x)}{W_1(\omega_2)} \frac{W_2(x)}{W_2(\omega_4)} \\
&+ d_7 \frac{W_0(x)}{W_0(\omega_1)} \frac{W_1(x)}{W_1(\omega_2)} \frac{W_2(x)}{W_2(\omega_4)} \\
=&\left(d_0 + d_4 \frac{W_2(x)}{W_2(\omega_4)} + \frac{W_1(x)}{W_1(\omega_2)} \left(d_2 + d_6 \frac{W_2(x)}{W_2(\omega_4)} \right) \right) \\
&+ \frac{W_0(x)}{W_0(\omega_1)} \left(d_1 + d_5 \frac{W_2(x)}{W_2(\omega_4)} + \frac{W_1(x)}{W_1(\omega_2)} \left(d_3 + d_7 \frac{W_2(x)}{W_2(\omega_4)} \right) \right) \\
=&\left(\Delta_2^0(x) + \frac{W_1(x)}{W_1(\omega_2)} \Delta_2^2(x) \right) \\
&+ \frac{W_0(x)}{W_0(\omega_1)} \left(\Delta_2^1(x) + \frac{W_1(x)}{W_1(\omega_2)} \Delta_2^3(x) \right) \\
=&\Delta_1^0(x) + \frac{W_0(x)}{W_0(\omega_1)} \Delta_1^1(x) = \Delta_0^0(x).
\end{aligned} \quad (12)$$

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

Lemma 2.

$$\Delta_i^m(x+y)=\Delta_i^m(x), \forall y \in \{\omega_b\}_{b=0}^{2^i-1}. \tag{13}$$

Proof: By Lemma 1, we have

$$W_i(x+y)=W_i(x)+W_i(y)=W_i(x), \forall y \in \{\omega_b\}_{b=0}^{2^i-1}. \tag{14}$$

The proof follows mathematical induction on i*. In the base case, we consider* (9) *at* $i=\lg(h)-1$*:*

$$\begin{aligned}
&\Delta_{\lg(h)-1}^m(x)\\
=&\Delta_{\lg(h)}^m(x)+\frac{W_{\lg(h)-1}(x)}{W_{\lg(h)-1}(\omega_{2^{\lg(h)-1}})}\Delta_{\lg(h)}^{m+2^{\lg(h)-1}}(x)\\
=&d_m+\frac{W_{\lg(h)-1}(x)}{W_{\lg(h)-1}(\omega_{2^{\lg(h)-1}})}d_{m+2^{\lg(h)-1}}.
\end{aligned}$$

From (14)*, we have*

$$\begin{aligned}
&\Delta_{\lg(h)-1}^m(x+y)\\
=&d_m+\frac{W_{\lg(h)-1}(x+y)}{W_{\lg(h)-1}(\omega_{2^{\lg(h)-1}})}d_{m+2^{\lg(h)-1}}\\
=&d_m+\frac{W_{\lg(h)-1}(x)}{W_{\lg(h)-1}(\omega_{2^{\lg(h)-1}})}d_{m+2^{\lg(h)-1}}\\
=&\Delta_{\lg(h)-1}^m(x), \forall y \in \{\omega_b\}_{b=0}^{h/2-1}.
\end{aligned}$$

Thus (13) *holds for* $i=\lg(h)-1$.

Assume (13) *holds for* $i=c+1$*. When* $i=c$*, we have*

$$\begin{aligned}
&\Delta_c^m(x+y)\\
=&\Delta_{c+1}^m(x+y)+\frac{W_c(x+y)}{W_c(\omega_{2^c})}\Delta_{c+1}^{m+2^c}(x+y)\\
=&\Delta_{c+1}^m(x+y)+\frac{W_c(x)}{W_c(\omega_{2^c})}\Delta_{c+1}^{m+2^c}(x+y)\\
=&\Delta_{c+1}^m(x)+\frac{W_c(x)}{W_c(\omega_{2^c})}\Delta_{c+1}^{m+2^c}(x)\\
=&\Delta_c^m(x), \forall y \in \{\omega_b\}_{b=0}^{2^c-1}.
\end{aligned}$$

This completes the proof. ■

B. Proposed algorithm

Let

$$\Psi(i,m,l)=\{\Delta_i^m(\omega_c+\omega_l)|c\in\{b\cdot 2^i\}_{b=0}^{h/2^i-1}\} \\ \text{, for } 0\le i\le \lg(h)-1; \tag{15}$$

$$\Psi(\lg(h),m,l)=\{d_m\}. \tag{16}$$

The objective of algorithm is to compute the values in set $\Psi(0,0,l)$. In the following, we rearrange the set $\Psi(i,m,l)$ into two parts: $\Psi(i+1,m,l)$ and $\Psi(i+1,m+2^i,l)$, by taking around $h/2^i$ additions and $h/2^{i+1}$ multiplications.

In (15), $\Psi(i,m,l)$ can be divided into two individual subsets:

$$\{\Delta_i^m(\omega_c+\omega_l)|c\in\{b\cdot 2^{i+1}\}_{b=0}^{h/2^{i+1}-1}\} \tag{17}$$

and

$$\{\Delta_i^m(\omega_c+\omega_l+\omega_{2^i})|c\in\{b\cdot 2^{i+1}\}_{b=0}^{h/2^{i+1}-1}\}. \tag{18}$$

In (17), we have

$$\begin{aligned}
&\Delta_i^m(\omega_c+\omega_l)\\
=&\Delta_{i+1}^m(\omega_c+\omega_l)+\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l).
\end{aligned} \tag{19}$$

It can be seen that $\Delta_{i+1}^m(\omega_c+\omega_l)\in\Psi(i+1,m,l)$, and $\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)\in\Psi(i+1,m+2^i,l)$. The factor $\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}$ can be precomputed and stored. Hence, for each element of the set given in (17), the calculation requires a multiplication and an addition. Note that when $\omega_c+\omega_l=0$, we have

$$\Delta_i^m(0)=\Delta_{i+1}^m(0), \tag{20}$$

which does not involve any arithmetic operations.

Next we consider the computation in (18), and we have

$$\begin{aligned}
&\Delta_i^m(\omega_c+\omega_l+\omega_{2^i})=\Delta_{i+1}^m(\omega_c+\omega_l+\omega_{2^i})\\
&+\frac{W_i(\omega_c+\omega_l+\omega_{2^i})}{W_i(\omega_{2^i})}\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l+\omega_{2^i}).
\end{aligned} \tag{21}$$

By Lemma 2, we have

$$\Delta_{i+1}^m(\omega_c+\omega_l+\omega_{2^i})=\Delta_{i+1}^m(\omega_c+\omega_l);$$

$$\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l+\omega_{2^i})=\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l).$$

Furthermore, the factor can be rewritten as

$$\begin{aligned}
&\frac{W_i(\omega_c+\omega_l+\omega_{2^i})}{W_i(\omega_{2^i})}\\
=&\frac{W_i(\omega_c+\omega_l)+W_i(\omega_{2^i})}{W_i(\omega_{2^i})}\\
=&\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}+1.
\end{aligned}$$

With above results, (21) can be rewritten as

$$\begin{aligned}
&\Delta_i^m(\omega_c+\omega_l+\omega_{2^i})\\
=&\Delta_{i+1}^m(\omega_c+\omega_l)+\left(\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}+1\right)\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)\\
=&\Delta_{i+1}^m(\omega_c+\omega_l)+\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)\\
&+\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)\\
=&\Delta_i^m(\omega_c+\omega_l)+\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l).
\end{aligned} \tag{22}$$

Hence, the element requires an addition.

C. Inverse transform

The inversion is a transform converts $\Psi(i,m,l)$ into polynomial coefficients $\{d_m\}_{m=0}^{h-1}$. The inversion can be done through backtracking the transform algorithm. As mentioned previously, $\Psi(i,m,l)$ can be rearranged into two parts: $\Psi(i+1,m,l)$ and $\Psi(i+1,m+2^i,l)$. Assume the set $\Psi(i,m,l)$ is given, we present the method to compute $\Psi(i+1,m,l)$ and $\Psi(i+1,m+2^i,l)$, respectively.

To construct $\Psi(i+1,m+2^i,l)$, (22) is reformulated as

$$\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)=\Delta_i^m(\omega_c+\omega_l)+\Delta_i^m(\omega_c+\omega_l+\omega_{2^i}). \quad (23)$$

Since $\Delta_i^m(\omega_c+\omega_l),\Delta_i^m(\omega_c+\omega_l+\omega_{2^i})\in\Psi(i,m,l)$, each $\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)\in\Psi(i+1,m+2^i,l)$ can be calculated with taking an addition.

To construct $\Psi(i+1,m,l)$, (19) is reformulated as

$$\begin{aligned}&\Delta_{i+1}^m(\omega_c+\omega_l)\\ =&\Delta_i^m(\omega_c+\omega_l)+\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l).\end{aligned} \quad (24)$$

Since $\Delta_i^m(\omega_c+\omega_l)\in\Psi(i,m,l)$ and $\Delta_{i+1}^{m+2^i}(\omega_c+\omega_l)\in\Psi(i+1,m+2^i,l)$ are known, each $\Delta_{i+1}^m(\omega_c+\omega_l)\in\Psi(i+1,m,l)$ can be calculated with taking an addition and a multiplication.

Figure 1 depicts an example of the proposed transform $\Psi_h^l[\bullet]$ of length $h=8$. Figure 1(a) shows the flow graph of the transform. The dotted line arrow denotes that the element should be multiplied with a scalar factor $\hat{W}_i^j$ upon adding together with other element, where the scalar factor is denoted as

$$\hat{W}_i^j=\frac{W_i(\omega_j)}{W_i(\omega_{2^i})}.$$

Figure 1(b) shows the flow graph of inversion. Also, it would be of interest to compare Figure 1 with the butterfly diagram of radix-2 FFT.

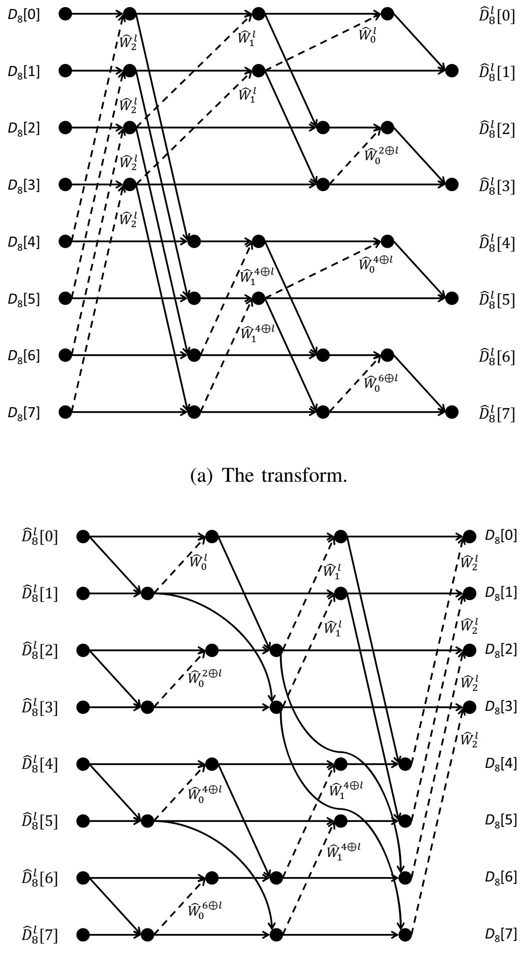
D. Computational complexity

Clearly, the proposed transform and its inversion have the same computational complexity. Thus, we only consider the computational complexity on transform. By the recursive structure, the number of arithmetic operations can be formulated as recursive functions. Let $A(h)$ and $M(h)$ respectively denote the number of additions and multiplications used in the algorithm. By (19) and (22), the recursive formula is given by

$$A(h)=2A(h/2)+h;A(1)=0;$$
$$M(h)=2M(h/2)+h/2;M(1)=0.$$

The solution is

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



(a) The transform.

(b) The inverse transform.

Figure 1. Data flow diagram of proposed transform of length $h=8$.

Notice that when the amount of shift $\omega_l=0$, the number of operations can be reduced slightly (see (20)). In this case, we have

$$A_0(h)=h\lg(h)-h+1;\ M_0(h)=\frac{h}{2}\lg(h)-h+1.$$

E. Space complexity

In a h-point transform, we need h units of space for the input data and an array to store the factors used in the computation of (17). From (19), the factors are

$$\frac{W_i(\omega_c+\omega_l)}{W_i(\omega_{2^i})}=\frac{W_i(\omega_c)}{W_i(\omega_{2^i})}+\frac{W_i(\omega_l)}{W_i(\omega_{2^i})},\forall c\in\{b\cdot2^{i+1}\}_{b=0}^{h/2^{i+1}-1}.$$

As $0\le i\le\lg(h)$, a h-point transform requires a total of

$$\frac{h}{2}+\frac{h}{4}+\cdots+\frac{h}{h}=h-1$$

units of space to store the factors. Hence, the space complexity is $O(h)$.

IV. Formal derivative

In this section, we consider the formal derivative over the proposed basis. Section IV-A gives the closed form of the formal derivative. SectionIV-B presents a computation method that has lower multiplicative complexity than the original approach.

A. Closed-form expression of formal derivative of $[D_h](x)$

Lemma 3. *The formal derivative of $W_i(x)$ is a constant given by*

$$W_i'(x) = \prod_{j=1}^{2^i-1} \omega_j. \tag{25}$$

Proof: Let

$$C(x) = c \cdot x^j,$$

where $c \in \mathbb{F}_{2^r}$. Its formal derivative is defined as

$$C'(x) = \begin{cases} 0 & \textit{if } j \textit{ is even;} \\ cx^{j-1} & \textit{otherwise.} \end{cases}$$

From Lemma 1, $W_i(x)$ has terms in the degrees of $1, 2, 4, \dots, 2^i$, so the formal derivative of $W_i(x)$ is a constant that is the coefficient of $W_i(x)$ at degree 1. The value is

$$\sum_{l=0}^{2^i-1} \prod_{j\neq l} \omega_j = \prod_{j=1}^{2^i-1} \omega_j.$$

This completes the proof. ■

By Lemma 3, the formal derivative of $X_i(x)$ is shown to be

$$\begin{aligned} X_i(x) =& \sum_{l=0}^{r-1} i_l \frac{W_l'(x)}{W_l(\omega_{2^l})} \prod_{j\neq l} \left(\frac{W_j(x)}{W_j(\omega_{2^j})} \right)^{i_j} \\ =& \sum_{l\in I_i} W_l' \cdot X_{i-2^l}(x), \end{aligned} \tag{26}$$

where

$$W_l' = \frac{W_l'(x)}{W_l(\omega_{2^l})} = \frac{\prod_{j=1}^{2^l-1} \omega_j}{W_l(\omega_{2^l})}, \tag{27}$$

and I_i is a set including all the non-zero indices in the binary representation of i, given by

$$I_i = \{ j | i_j = 1, j = 0, 1, \dots, r-1 \}.$$

For example, if $i = 13 = 2^0 + 2^2 + 2^3$, we have

$$\begin{aligned} & X_{13}'(x) \\ =& W_0' W_2(x) W_3(x) + W_2' W_0(x) W_3(x) + W_3' W_0(x) W_2(x) \\ =& W_0' X_{12}(x) + W_2' X_9(x) + W_3' X_5(x). \end{aligned} \tag{28}$$

From (26), the formal derivative of $[D_h](x)$ is given by

$$[D_h]'(x) = \sum_{i=0}^{h-1} d_i \sum_{l\in I_i} W_l' \cdot X_{i-2^l}(x), \tag{29}$$

We move the term $X_j(x)$ out of the summation operator to get

$$[D_h]'(x) = \sum_{j=0}^{h-1} X_j(x) \sum_{l\in I_j^c} W_l' \cdot d_{j+2^l}, \tag{30}$$

where I_j^c is the complement of I_j defined as

$$I_j^c = \{i\}_{i=0}^{\lg(h)-1} \setminus I_j.$$

From (30), when W_l' given in (27) are pre-computed and stored, each coefficient of $X_j(x)$ requires at most $\lg(h)-1$ additions and $\lg(h)$ multiplications. Thus a native way to compute the formal derivation of $[D_h](x)$ requires $O(h\lg(h))$ operations, in both additive complexity and multiplicative complexity.

B. Computation method with lower multiplicative complexity

We present an alternative approach whose multiplicative complexity is lower than the above approach. Define

$$d_i^{\mathrm{d}} = d_i \prod_{j\in I_i} W_j', \tag{31}$$

for $0 \leq i \leq h-1$. By substituting (31) into (30), we have

$$[D_h]'(x) = \sum_{j=0}^{h-1} X_j(x) \sum_{l\in I_j^c} \frac{W_l' \cdot d_{j+2^l}^{\mathrm{d}}}{\prod_{m\in I_{j+2^l}} W_m'}. \tag{32}$$

As

$$\prod_{m\in I_{j+2^l}} W_m' = W_l' \prod_{m\in I_j} W_m',$$

(32) can be rewritten as

$$\begin{aligned} [D_h]'(x) &= \sum_{j=0}^{h-1} X_j(x) \sum_{l\in I_j^c} \frac{d_{j+2^l}^{\mathrm{d}}}{\prod_{m\in I_j} W_m'} \\ &= \sum_{j=0}^{h-1} X_j(x) \frac{\sum_{l\in I_j^c} d_{j+2^l}^{\mathrm{d}}}{\prod_{m\in I_j} W_m'}. \end{aligned} \tag{33}$$

By the above formulas, the method of computing $[D_h]'(x)$ consists of two steps. In the first step, we compute (31). Here, the set of factors

$$B = \{ \prod_{j\in I_i} W_j' | i = 0, 1, \dots, h-1 \} \tag{34}$$

can be pre-computed and stored, and this step only requires h multiplications. In the second step, we compute the coefficients through (33). Notice that the denominator is an element of B. Thus, this step needs around $\frac{1}{2}h\lg(h)$ additions and h multiplications.

Next we use an example to demonstrate how to obtain $[D_h]'(x)$. If $h = 8$ and the set B includes 8 elements defined as

$$B_0 = 1; B_1 = W_0'; B_2 = W_1'; B_3 = W_0'W_1';$$
$$B_4 = W_2'; B_5 = W_0'W_2'; B_6 = W_1'W_2'; B_7 = W_0'W_1'W_2'.$$

From (31), each $d_i, 0 \leq i \leq 7$ is computed via

$$d_i^{\mathrm{d}} = d_i B_i.$$

From (33), the formal derivative of $[D_8](x)$ is shown to be

$$\begin{aligned}&[D_8]'(x)\\ =&X_0(x)\frac{d_1^{\mathrm{d}}+d_2^{\mathrm{d}}+d_4^{\mathrm{d}}}{B_0}+X_1(x)\frac{d_3^{\mathrm{d}}+d_5^{\mathrm{d}}}{B_1}+X_2(x)\frac{d_3^{\mathrm{d}}+d_6^{\mathrm{d}}}{B_2}\\ &+X_3(x)\frac{d_7^{\mathrm{d}}}{B_3}+X_4(x)\frac{d_5^{\mathrm{d}}+d_6^{\mathrm{d}}}{B_4}+X_5(x)\frac{d_7^{\mathrm{d}}}{B_5}+X_6(x)\frac{d_7^{\mathrm{d}}}{B_6}.\end{aligned}$$

V. Algorithms of Reed-Solomon erasure codes

Based on the new polynomial basis, this section presents the encoding and decoding algorithms for (n, k) Reed-Solomon (RS) erasure codes over characteristic-2 fields. There are two major approaches on the encoding of 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 $F(x)$ of degree less than k. Let

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

denote the vector of message, for each $m_i \in \mathbb{F}_{2^r}$. In the systematic construction, $F(x)$ is a polynomial of degree less than k such that

$$F(\omega_i) = m_i, \text{ for } 0 \leq i \leq k-1. \tag{35}$$

By the set of equations (35), $F(x)$ can be uniquely constructed via polynomial interpolation. Then we use this $F(x)$ to calculate the codeword

$$F_n = (F(\omega_0), F(\omega_1), \ldots, F(\omega_{n-1})).$$

In decoding, assume the received codeword has $n-k$ erasures $\{F(y) : y \in E\}$, where E denotes the set of evaluation points of erasures. With the k un-erased symbols, $F(x)$ can be uniquely reconstructed via polynomial interpolation, and thus the erasures can be computed accordingly.

In the following, we illustrate the algorithms of encoding and erasure decoding for Reed-Solomon codes. The proposed algorithm is for k a power of two, and $n = 2^r$. The codes for other k can be derived through code shortening strategy; i.e., appending zeros to message vector so that the length of the vector is power of two.

A. Encoding algorithm

Algorithm 1 illustrates the pseudocode of the (n, k) RS encoding algorithm. In Line 1, we compute the vector

$$\bar{M}_k = (\bar{m}_0, \bar{m}_1, \ldots, \bar{m}_{k-1}),$$

which can be formed as a polynomial

$$[\bar{M}_k](x) = \sum_{i=0}^{k-1} \bar{m}_i X_i(x).$$

Algorithm 1 Reed-Solomon encoding algorithm.

```
Input: A $k$-element message vector $M_k$ over $\mathbb{F}_{2^r}$.
Output: An $n$-element systematic codeword $F_n$.
1: $\bar{M}_k = (\Psi_k^0)^{-1}[M_k]$
2: for $i = 1$ to $(n/k - 1)$ do
3:     $\bar{F}_i = \Psi_k^{i\cdot k}[\bar{M}_k]$
4: end for
5: return $F_n = (M_k, \bar{F}_1, \bar{F}_2, \ldots, \bar{F}_{\lceil n/k \rceil - 1})$.
```

Since $deg([\bar{M}_k](x)) \leq k-1$ and

$$[\bar{M}_k](\omega_i) = m_i, \text{ for } 0 \leq i \leq k-1 \tag{36}$$

we conclude that $[\bar{M}_k](x) = F(x)$. Thus, the parity-check symbols can be computed by applying the proposed transform on $\bar{M}_k$ (see Lines 2-4). The parity-check symbols are obtained in blocks with size k and there are $n/k - 1$ blocks.[2] For each block, the vector $\bar{F}_i$ includes k elements and each element is

$$\bar{F}_i[j] = [\bar{M}_k](\omega_{j+(i\cdot k)}) = [\bar{M}_k](\omega_j + \omega_{i\cdot k}), \text{ for } 0 \leq j \leq k-1.$$

In Line 5, we assemble those vectors to get the codeword vector F_n.

In summary, the encoding algorithm requires a k-element inversion $(\Psi_k^0)^{-1}[\bullet]$ and $(n/k - 1)$ times of k-element transform $\Psi_k^i[\bullet]$. Thus, the encoding algorithm has the complexity

$$O((n/k)k \lg(k)) = O(n \lg(k)).$$

B. Erasure decoding algorithm

The decoding algorithm follows our previous work [7] that requires evaluating a polynomial and it's derivatives. The code proposed in [7] is based on Fermat number transforms (FNT). In this paper, we replace the role of FNT over $\mathbb{F}_{2^r+1}$ with the proposed transform over $\mathbb{F}_{2^r}$. However, since the proposed transform is not Fourier transform, some arithmetic operations involved in the transform should be modified accordingly.

Assume the received codeword $\bar{F}_n$ has $n-k$ erasures. The set of evaluation points of erasures are denoted as

$$E = \{\omega_{e_i}\}_{i=0}^{n-k-1}.$$

Let

$$\Pi(x) = \prod_{y \in E}(x+y)$$

denote the error locator polynomial having zeros at all erased symbols. It can be seen that $\Pi(j) = 0, \forall j \in E$. Define

$$\hat{F}(x) = F(x)\Pi(x),$$

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

Algorithm 2 Framework of Reed-Solomon erasure decoding algorithm.

```
Input: Received codeword $\bar{F}_n$, and the positions of erasures
    $E = \{e_i\}_{i=0}^{n-k-1}$.
Output: The erasures $\{F(j) | j \in E\}$.
1: Compute two sets of values $\bar{\Pi}$ and $\Pi'$, defined in (40)
    and (42).
2: From (39), compute
    $\Phi = (\hat{F}(\omega_0), \hat{F}(\omega_1), \dots, \hat{F}(\omega_{n-1}))$.
3: Apply $n$-point fast inverse transform on $\Phi$ to get
    $\bar{\Phi}_n = (\Psi_n^0)^{-1}[\Phi]$.
4: Compute the formal derivative of $\bar{\Phi}_n$. The result is
    denoted as $\bar{\Phi}_n^{\mathrm{d}}$.
5: Apply $n$-point fast transform on $\bar{\Phi}_n^{\mathrm{d}}$ to get
    $\Phi_n^{\mathrm{d}} = \Psi_n^0[\bar{\Phi}_n^{\mathrm{d}}]$.
6: Compute the erasures via
    $F(j) = \frac{\Phi_n^{\mathrm{d}}[j]}{\Pi'(j)}, \forall j \in E$.
```

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

$$\hat{F}'(x) = F'(x)\Pi(x) + F(x)\Pi'(x). \tag{37}$$

By substituting $x = j \in E$ into (37), we have

$$\hat{F}'(j) = F(j)\Pi'(j), \forall j \in E.$$

Hence the erasures can be computed by

$$F(j) = \frac{\hat{F}'(j)}{\Pi'(j)}, \forall j \in E. \tag{38}$$

Based on 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 by (38). The details are elaborated as follows.

In the first stage, we need to compute the coefficients of $\hat{F}(x)$. It can be shown that

$$\hat{F}(j) = F(j)\Pi(j) = \begin{cases} 0 & \forall j \in E; \\ F(j)\Pi(j) & \text{otherwise.} \end{cases} \tag{39}$$

Here, we define

$$\bar{\Pi} = \{\Pi(j) | j \in \mathbb{F}_{2^r} \backslash E\}. \tag{40}$$

Appendix shows the algorithm of computing $\bar{\Pi}$ proposed by [11]. Since $F(j)$, $j \in \mathbb{F}_{2^r} \backslash E$ are elements of the received vector, the result of (39) can computed with n multiplications after $\bar{\Pi}$ is obtained and is denoted as a vector

$$\Phi = (\hat{F}(\omega_0), \hat{F}(\omega_1), \dots, \hat{F}(\omega_{n-1})).$$

Then we compute

$$\bar{\Phi}_n = (\Psi_n^0)^{-1}[\Phi]. \tag{41}$$

Here, the resulting vector $\bar{\Phi}_n = (\bar{\phi}_0, \bar{\phi}_1, \dots, \bar{\phi}_{n-1})$ can be formed as a polynomial

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

where $[\bar{\Phi}_n](\omega_j) = \hat{F}(\omega_j)$, for $0 \leq j \leq n-1$. That is, $[\bar{\Phi}_n](\omega_j) - \hat{F}(\omega_j) = 0$, for $0 \leq j \leq n-1$. Since the degree of $[\bar{\Phi}_n](x) - \hat{F}(x)$ is at most $n-1$, it must be the zero polynomial when it has n roots. Hence, we conclude $[\bar{\Phi}_n](x) = \hat{F}(x)$.

The second stage is to compute the formal derivative of $\hat{F}(x)$. Since $[\bar{\Phi}_n](x)$ is under the polynomial basis given by Definition 1, we compute the formal derivative of $[\bar{\Phi}_n](x)$ by the method presented in Section IV. Then we can obtain the result vector $\bar{\Phi}_n^{\mathrm{d}} = (\bar{\phi}_0^{\mathrm{d}}, \bar{\phi}_1^{\mathrm{d}}, \dots, \bar{\phi}_{n-1}^{\mathrm{d}})$, and the polynomial

$$[\bar{\Phi}_n^{\mathrm{d}}](x) = \sum_{i=0}^{n-1} \bar{\phi}_i^{\mathrm{d}} X_i(x)$$

is the formal derivative of $[\bar{\Phi}_n](x)$.

In the final stage, we need to compute the erasures via (38). Here, the denominators in (38) are defined as a set

$$\Pi' = \{\Pi'(j) | j \in E\}, \tag{42}$$

which can be constructed by the algorithm introduced in Appendix. We then compute

$$\Phi_n^{\mathrm{d}} = \Psi_n^0[\bar{\Phi}_n^{\mathrm{d}}], \tag{43}$$

where the resulting vector includes the evaluations of $\hat{F}'(j)$ for $j \in \mathbb{F}_{2^r}$; i.e., the Φ_n^{d} is denoted as

$$\Phi_n^{\mathrm{d}} = (\hat{F}'(\omega_0), \hat{F}'(\omega_1), \dots, \hat{F}'(\omega_{n-1})).$$

Then the erasures can be computed through (38).

The decoding procedure is summarized in Algorithm 2. The complexity of this algorithm is dominated by Steps 1, 3, 4 and 5, whereas Steps 2 and 6 only require $O(n)$ multiplications. By the proposed fast transform algorithm, Steps 3 and 5 can be done with $O(n \lg(n))$ additions and $O(n \lg(n))$ multiplications. By the method in Section IV, Step 4 requires $O(n \lg(n))$ additions and $O(n)$ multiplications. In Step 1, we use the algorithm shown in Appendix, and it can be done with $O(n \lg(n))$ modulus operations. In summary, the proposed decoding algorithm has the complexity of order $O(n \lg(n))$.

VI. Discussions and comparisons

A. Complexities of operations in polynomial basis

We consider some polynomial operations in this section. Table I tabulates the complexities of some polynomial operations in the monomial basis and the proposed basis

Table I
COMPLEXITIES OF OPERATIONS IN POLYNOMIAL BASIS OVER CHARACTERISTIC-2 FINITE FIELDS

Operations	Monomial basis	Proposed basis
Addition	$O(h)$	$O(h)$
Multiplication	$O(h\lg(h)\lg\lg(h))$	$O(h\lg(h))$
Polynomial degree	$O(h)$	$O(h)$
Formal derivative	$O(h)$	$O(h\lg(h))$

over characteristic-2 finite fields. In particular, the polynomial addition is simple by adding the coefficients of two polynomials. Hence, the complexity is $O(h)$ in both basis. For the polynomial multiplication, an algorithm with order $O(h\lg(h)\lg\lg(h))$ is proposed by [12], in 1977. To compute $[A_h](x)\times[B_h](x)$ in the proposed basis, the result polynomial is computed via

$$(\Psi^l_{2h})^{-1}[\Psi^l_{2h}[A_{2h}]\star\Psi^l_{2h}[B_{2h}]],$$

where A_{2h}(and B_{2h}) is $2h$-point vector by appending zeros to A_h(and B_h), and $\star$ denotes the operation of pairwise multiplication. Hence, the complexity is $O(h\lg(h))$.

To determine the degree polynomial in proposed basis, we scan the coefficients of $[D_h](x)$ to determine the highest degree term $d_jX_j(x), d_j\neq 0$, and thus the complexity is $O(h\lg(h))$; and so does the polynomial in monomial basis.

The formal derivative in proposed basis requires $O(h\lg(h))$ field operations shown in Section IV. In contrast, the formal derivative in monomial basis only requires $O(h)$ operations.

B. Comparisons with Didier's approach

In 2009, Didier [11] present an erasure decoding algorithm for Reed-Solomon codes based on fast Walsh-Hadamard transforms. The algorithm [11] consists of two major parts: the first part is to compute the polynomial evaluations of error locator polynomial, and the second part is to decompose the Lagrange polynomial into several logical convolutions, which are then respectively computed with fast Walsh-Hadamard transforms. The first part requires $O(n\lg(n))$ time, and the second part requires $O(n\lg^2(n))$ time, so the complexity [11] is $O(n\lg^2(n))$. In contrast, the proposed approach employs the first part in [11]; in the second part, we design another decoding structure based on the proposed basis. The proposed transform only requires $O(n\lg(n))$ time, so that the proposed approach can reduce the complexity from $O(n\lg^2(n))$ to $O(n\lg(n))$.

We also implement the proposed algorithm in C and run the program on 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 [11] written by the author on the same platform. In our simulation, the program [11] took about 52.91 seconds in both encoding and erasure decoding under the same parameter configuration. Thus, the proposed erasure decoding is around 17 times faster than [11], while $n = 2^{16}$.

VII. LITERATURE REVIEW

In the original view of [1], the codeword of the RS code is a sequence of evaluation values of a polynomial interpreted by message. By this viewpoint, the encoding process can be treated as an oversampling process through discrete Fourier transform (DFT) over finite fields. Some studies [13–15] indicate that, if a $O(n\lg(n))$ finite field FFT is available, the error-correction decoding can be reduced to $O(n\lg^2(n))$. An n-point radix-2 FFT butterfly diagram requires $n\lg(n)$ additions and $\frac{n}{2}\lg(n)$ multiplications. This FFT butterfly diagram can be directly applied on Fermat prime fields $\mathbb{F}_{2^r+1}, r\in\{1,2,4,8,16\}$. In this case, the transform, referred to as Fermat number transform (FNT), requires $n\lg(n)$ finite field additions and $\frac{n}{2}\lg(n)$ finite field multiplications. By employing FNT, a number of fast approaches [13, 16, 17] had been presented to reduce the complexity of encoding and decoding of RS codes. Some FNT-based RS erasure decoding algorithms had been proposed [7, 18, 19] in $O(n\lg(n))$ finite 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 it needs more space to store one extra symbol in practical implementation such that the FNT-based codes are impractical in general applications.

On the other hand, FFTs over characteristic-2 finite fields require higher complexities than $O(n\lg(n))$. Table II tabulates the arithmetic complexities of FFT algorithms over characteristic-2 finite fields. As shown in Table II, Gao and Mateer [10] gave two versions of additive FFTs over $\mathbb{F}_{2^r}$ that are most likely the most efficient FFTs by far. The first is for arbitrary r, and the second is for r a power of two. Notably, Wu's approach [20] has very low multiplicative complexity $O(n\lg^{\lg(3/2)}(n))$, but the additive complexity is higher with complexity $O(n^2/\lg^{\lg(8/3)}(n))$. This implies that when the polynomial representation in RS codes are in monomial basis, the complexity will fail to reach $O(n\lg(n))$.

There exist faster encoding and erasure decoding approaches in some non-MDS codes. Such codes, termed as fountain codes [6], require a little more than k codeword symbols to recover the original message. Two famous classes of fountain codes are LT code [21] and Raptor code [22]. Due to the low complexity, fountain codes have significant merits in many applications. However, because of the randomly generated generator matrices, the hardware parallelization of fountain code is not trivial.

VIII. CONCLUDING REMARKS

Based on the proposed polynomial basis, we can compute the polynomial evaluations in the complexity of order

Table II
COMPLEXITIES OF n-POINT FFT ALGORITHMS OVER $\mathbb{F}_{2^r}$, WHERE $n = 2^r - 1$

Algorithm	Restriction	Additive complexity	Multiplicative complexity
Gao [10]	r is a power of two	$O(n \lg(n) \lg \lg(n))$	$O(n \lg(n))$
Cantor [8]	r is a power of two	$O(n \lg^{\lg(3)}(n))$	$O(n \lg(n))$
Gao [10]		$O(n \lg^2(n))$	$O(n \lg(n))$
Wang [23], Gathen [9]		$O(n \lg^2(n))$	$O(n \lg^2(n))$
Pollard [24]	r is even	$O(n^{3/2})$	$O(n^{3/2})$
Wu [20]		$O(n^2 / \lg^{\lg(8/3)}(n))$	$O(n \lg^{\lg(3/2)}(n))$
Sarwate [25]		$O(n^2)$	$O(n \lg(n))$
Naive approach		$O(n^2)$	$O(n^2)$

$O(h \lg(h))$ with a small leading constant. This enables our capability to encode/erasure decode (n, k) Reed-Solomon codes over characteristic-2 finite field in $O(n \lg(n))$ time. As the complexity leading factor is small, the algorithms are advantageous in practical applications. To the best of our knowledge, this is the first approach supporting Reed-Solomon erasure codes on characteristic-2 finite fields to achieve complexity of $O(n \lg(n))$. In addition, all the transforms employed in the Reed-Solomon algorithms can be easily implemented in parallel processing. Hence, the proposed algorithms substantially facilitate practical applications. While this paper has demonstrated the polynomial basis and operations over characteristic-2 finite fields, it is of interest to consider the case over fields with arbitrary characteristics.

ACKNOWLEDGMENT

This work was supported in part by Ministry of Science and Technology of Taiwan, under grants NSC 101-2221-E-011-069-MY3, NSC 102-2221-E-001-006-MY2, and MOST 103-3113-E-110-002.

APPENDIX

In [11], Didier present an efficient algorithm to compute the elements in two sets (40) and (42). The method is presented here for the purpose of completeness. Consider the construction of Π'. The formal derivative of $\Pi(x)$ is given by

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

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

$$\Pi'(j) = \prod_{y \in E, y \neq j} (j + y) = \prod_{y \in \mathbb{F}_{2^r}, y \neq j} (j + y)^{R(y)}, \quad (44)$$

where $R(x)$ is a function defined as

$$R(y) = \begin{cases} 1 & \text{if } y \in E; \\ 0 & \text{otherwise.} \end{cases} \quad (45)$$

Define $Log(x)$ as the discrete logarithm function. For each $i \in \mathbb{F}^*_{2^r}$, we denote $Log(i) = j$ iff $i = \alpha^j$, where α is the primitive element of $\mathbb{F}^*_{2^r}$. Then (44) can be reformulated as

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

Note that the symbol $\biguplus$ means the summation with normal additions. By setting $Log(0) = 0$, the above equation can be rewritten as

$$Log(\Pi'(j)) = \biguplus_{y \in \mathbb{F}_{2^r}} R(y) Log(j + y), \forall j \in E. \quad (46)$$

Upon describing the algorithm to compute (46), we consider the construction of another set Π. In the same way, the elements of Π can be formulated as

$$Log(\Pi(j)) = \biguplus_{y \in \mathbb{F}_{2^r}} R(y) Log(j + y), \forall j \in \mathbb{F}_{2^r} \setminus E. \quad (47)$$

With combining (46) and (47), the objective of algorithm is to compute

$$Log(\Pi(j)) = \biguplus_{y \in \mathbb{F}_{2^r}} R(y) Log(j + y), \forall j \in \mathbb{F}_{2^r}. \quad (48)$$

In (48), the operation $+$ is the $\mathbb{F}_{2^r}$ addition, that can be treated as exclusive-or operation. Hence, (48) is namely the logical convolution [26][27], that can be efficiently computed with fast Walsh-Hadamard transform [28]. The algorithm is elaborated as follows.

Let $FWT_h[\bullet]$ denote the h-point fast Walsh-Hadamard transform (FWHT). A h-point FWHT requires $h \lg(h)$ additions. Define

$$R_{2^r} = (R(0), R(1), \dots, R(2^r - 1)),$$

$$L_{2^r} = (0, Log(\omega_1), Log(\omega_2), \dots, Log(\omega_{2^r - 1})).$$

The result of (48) is computed by

$$R^{\mathrm{w}}_{2^r} = \mathrm{FWHT}_{2^r}[\mathrm{FWHT}_{2^r}[R_{2^r}] \star \mathrm{FWHT}_{2^r}[L_{2^r}]], \quad (49)$$

where the operation $\star$ denotes pairwise multiplication. To further reduce the complexity, the $\mathrm{FWHT}_{2^r}[L_{2^r}]$ can be pre-computed and stored, and thus (49) can be done with performing two fast Walsh transforms of length 2^r. We remark that all the above computation can be performed over modulo $2^r - 1$. After obtaining $R^{\mathrm{w}}_{2^r}$, we compute the exponent for each element of $R^{\mathrm{w}}_{2^r}$, and this step can be done via table lookup. In summary, the algorithm requires $O(2^r \lg(2^r))$ modulus additions, $O(2^r)$ modulus multiplications, and $O(2^r)$ exponentiations.

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] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *SIGMOD Rec.*, vol. 17, no. 3, pp. 109–116, Jun. 1988.

[3] A. Aliev and P. Trifonov, "Redundant disk encoding via erasure decoding," Jan. 2 2014, US Patent App. 13/784,000. [Online]. Available: http://www.google.com/patents/US20140006850

[4] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, Boston, MA, 2012, p. 2.

[5] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," *Proc. VLDB Endow.*, vol. 6, no. 5, pp. 325–336, Mar. 2013.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] F. Didier, "Efficient erasure decoding of Reed-Solomon codes," *CoRR*, vol. abs/0901.1886, 2009.

[12] 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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

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

[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.