An Efficient (n, k) Information Dispersal Algorithm based on Fermat Number Transforms

Sian-Jheng Lin and Wei-Ho Chung*

***Abstract*—The (n, k) information dispersal algorithm (IDA) is a coding technique converting a digital source file into n small digital files (shadows), and the receipt of any k out of the n shadows can losslessly reconstruct the source file. This paper presents an encoding and two decoding algorithms of (n, k) IDA via the fast Fermat number transform (FNT). The proposed encoding algorithm requires $O(n \log k)$ arithmetic operations, and the two decoding algorithms have complexities $O(n \log k)$ and $O(k \log^2 k)$ for a reasonably large file. As compared with existing work, the proposed algorithms generate significant improvement in the throughput in the low code rate $k/n \leq 1/2$ settings.**

***Index Terms*—Erasure codes, fast Fourier transforms, Galois fields, information dispersal algorithm (IDA), Reed-Solomon codes.**

I. Introduction

Information dispersal algorithm (IDA) was first introduced by Rabin [1], [2] in 1989. The (n, k) IDA transforms a digital source file into n smaller files (shadows), and the receipt of any k out of the n shadows can reconstruct the source file. By such coding technology, the robustness and fault-tolerance of important files can be improved in communication and storage systems. The idea of IDA is similar to the polynomial secret sharing [35], [36], [37]. However, IDA does not consider the security issue in the design of coding system. In a well-designed IDA, the length of each shadow is (asymptotically) equal to one k-th of the length of source file. Precisely, the length of each shadow achieves the theoretical lower bound [3], under the condition of maximal entropy in the source file. The IDA had been applied to many applications, e.g., distributed data storage [4], RAID codes [5], peer-to-peer techniques [6], multicast [7], and secret sharing [33]. A remarkable coding technique, namely the fountain code [34], is the alternative technique to disperse the source file in the network environment.

Conceptually, the (n, k) IDA can be treated as the (n, k) erasure code. When the receiver acquires k shadows, the scenario is equivalent to erasing the corresponding $n-k$ non-received symbols in an n-symbol codeword. Thus, the optimal erasure codes, such as Maximum Distance Separable (MDS) codes [15], [17], [18] or Reed-Solomon(RS) codes [19], [20], [22], can serve as the applicable coding systems for the (n, k) IDA. The standard implementation of (n, k) RS erasure codes require $\mathrm{O}(nk)$ operations in encoding and $\mathrm{O}(k^2)$ operations in decoding via ordinary matrix multiplications.

The computational complexity is a challenge in IDA. The fast IDA can improve the throughput of the real-time systems by demanding large amount of encoding and decoding operations. Various IDAs have been reported in previous literature. In many cases, the erasure codes (or IDAs) can be formulated as the matrix-product forms, so the encoding and decoding complexities depend on the overhead of computing the matrix products. The conventional approaches of (n, k) IDA, such as [1], do not utilize the fast techniques on encoding and decoding processes. Therefore, the conventional encoding algorithm requires $\mathrm{O}(nk)$ operations and the conventional decoding algorithm requires $\mathrm{O}(k^2)$ operations. To further reduce the computational overhead, the fast Fourier transforms (FFT) over finite field with characteristic two [8], [9], [10], [11] or the fast Fermat number transforms (FNT) are employed in the coding algorithms. For example, Preparata [12] presented the realization of the coding schemes using FFT over finite fields, and the computational complexities are $\mathrm{O}(n \log n)$ in encoding and $\mathrm{O}(k(n-k+\log k))$ in decoding. Dianat and Marvasti [13] presented the systematic and nonsystematic codes of puncturing RS codes based on FFT over finite fields. Soro and Lacan [14] proposed a Reed-Solomon erasure coding algorithm with complexity $\mathrm{O}(n \log n)$ in both encoding and decoding. Lacan and Fimes [15] investigated a systematic MDS erasure coding algorithm based on Vandermonde matrices. For the case $k/n \geq 1/2$, Lin and Chung [38] present a (n, k) IDA with complexities $O(n \log(n-k))$ in encoding and decoding. For the finite field $GF(2^r)$ with characteristic two, Didier [16] presented an decoding algorithm for RS erasure codes $\mathrm{O}(2^r \log^2 2^r)$ via fast Walsh transforms. In Section VII, we compare the proposed IDA with those existing methods.

There exist works for erasure codes. G. L. Feng et al. [17], [18] proposed (n, k) MDS codes based on exclusive-OR (XOR) operations. Truong et al. [19] proposed a fast RS decoding algorithm for correcting both errors and erasures. By FFT over finite field with characteristic two [10], [11], the FFT version of RS decoding algorithm had been proposed [20]. Lin et al. [21] proposed a fast algorithm for computing the syndromes of RS codes. Note that there exist several non-optimal erasure codes, such as [34] and [23], [24]. In general, those non-optimal erasure codes require lower computational complexities than optimal erasure codes. However, non-optimal erasure codes cannot guarantee successful decoding from arbitrary k out of the n codeword symbols, as opposed

Sian-Jheng Lin and Wei-Ho Chung are with the Research Center for Information Technology Innovation, Academia Sinica, Taipei City, Taiwan (e-mail: sjlin@citi.sinica.edu.tw; whc@citi.sinica.edu.tw). This work was supported by National Science Council of Taiwan, under grant number NSC101-2221-E-001-002, NSC101-2221-E-001-008, NSC101-2622-E-001-001-CC3, and 32T-1010721-1C.

to the guaranteed successful decoding in the optimal erasure codes.

By our survey on the (n, k) erasure coding algorithms over Fermat field, the best records of encoding algorithm take $\mathrm{O}(n \log n)$ operations [12], [13], [14] and the decoding algorithms take $\mathrm{O}(n \log n)$ [14] or $\mathrm{O}(k \log^2 k)$ [15] operations. In this paper, we propose a fast (n, k) IDA based on erasure Reed-Solomon coding systems over Fermat fields. Given k source symbols, the proposed encoding algorithm requires complexity $\mathrm{O}(n \log k)$, which is lower than the existing work $\mathrm{O}(n \log n)$. In decoding, we present two algorithms. Given k shadow symbols, the main procedure of first decoding algorithm requires complexity $\mathrm{O}(n \log k)$, which is lower than the existing work $\mathrm{O}(n \log n)$ [12], [13], [14]. The main procedure of second decoding algorithm requires complexity $O(k \log^2 k)$ with smaller leading constant than the fast polynomial interpolation algorithm [15], [25]. By the computational complexities of the two proposed decoding algorithms, the first decoding algorithm is applicable for $n \leq k \log^2 k$, and otherwise the second decoding algorithm is to be adopted. The criteria of choosing one of the two decoding algorithms are discussed in Section VIII-B. It is noted that the [38] presents the IDA for the high code rates $k/n \geq 1/2$. In contrast, the proposed IDA is suitable for the low code rates $k/n \leq 1/2$, and the detailed comparisons between [38] and the proposed method are detailed in Section VII. The potential applications of the low-rate codes are in the communication systems. For example, the [39] indicates that the low-rate codes are applicable to code-division multiple-access (CDMA) systems.

The rest of the paper is organized as follows. Section II introduces the definition of coding system, and the concept of the proposed algorithms. Sections III, IV, and V respectively present the proposed encoding and two decoding algorithms under the settings where k is a power of two, and n is the multiple of k. Section VI analyses the complexities of the three proposed algorithms. In Section VII, we compare the proposed IDA with existing work in theoretical and experimental aspects. Certain related important issues on the proposed algorithm are discussed in Section VIII, including the generalization of the parameters (n, k), the decision criteria of choosing the suitable decoding algorithm, the partial FFT on second decoding algorithm, the strategy of storing the shadow elements in binary data format, and the generalized IDA over Proth field. Finally, Section IX concludes this work.

II. System description

The coding system is a modified version of the systematic Reed-Solomon codes over Fermat field $GF(2^r + 1)$. Thus far the known Fermat primes are for $r = \{1, 2, 4, 8, 16\}$. Throughout this paper, the arithmetic operation is conducted over Fermat field unless otherwise specified. The coding system follows the polynomial evaluation approach of (n, k) Reed-Solomon codes [22]. The source file is denoted as a $(2^r + 1)$-ary integer sequence F. This F is divided into a certain number of non-overlapping sub-vectors of length k each. The encoding process is then individually operated on each k-element sub-vector $a = (a_0, a_1 \dots a_{k-1})$, where

Table of Notations

For any polynomial defined in this paper, the vector formulation of the polynomial is denoted by the coefficients of the function. For example, the vector formulation of a polynomial $f(x) = \sum_{i=0}^{k-1} f_i x^i$ is denoted as $f = (f_0, f_1 \dots f_{k-1})$.

Big-O	$f(x) = O(g(x))$ iff there exist two positive constants c and x_0 such that $f(x) \leq c \times g(x)$ for $x \geq x_0$.
$GF(2^r + 1)$	Finite field of size $2^r + 1$, and $r = \{1, 2, 4, 8, 16\}$.
F	A sequence of data symbols over Fermat field.
n	Number of generated shadows.
k	Threshold number of shadows to reconstruct the F.
I_i	Evaluation point of a codeword symbol.
F_i	A sequence of symbols consisting of codeword symbols corresponding to the evaluation point I_i. The length of F_i is denoted as $\lvert F_i \rvert$.
a	A k-element vector $a = (a_0, a_1 \dots a_{k-1})$ which is the input of the encoding algorithm.
b	An n-element vector $b = (b_0, b_1 \dots b_{n-1})$ which is the output of the encoding algorithm.
$FNT_k(\cdot)$	k-point Fermat number transform.
$IFNT_k(\cdot)$	k-point inverse Fermat number transform.
$A_k(x)$	A k-element vector $A_k(x) = (1, x, x^2 \dots x^{k-1})$.
$\otimes$	$Y_0 \otimes Y_1$ denotes the pair-wise product of vectors Y_0 and Y_1.
$*$	$Y_0 * Y_1$ denotes convolution of vectors Y_0 and Y_1.
$\beta, \hat{\beta}$	Two sets containing the auxiliary data generated from the initialization of Algorithm 2.
$\gamma, \hat{\gamma}_j$	γ is a set and $\hat{\gamma}_j$ is a vector. γ and $\hat{\gamma}_j$ contains the auxiliary data generated from the initialization of Algorithm 3.

each a_i indicates a message symbol in the ready-to-process sub-vector. Let the $f(x) = \sum_{i=0}^{k-1} f_i x^i$ denote the coding polynomial, which is chosen to satisfy the k equalities below:

$$a_i = f(I_i), \forall i = 0 \dots k-1. \tag{1}$$

The n-symbol codeword is the evaluation of $f(x)$ at n distinct evaluation points $I = \{I_i\}_{i=0}^{n-1}$. Let the vector $b = (b_0, b_1 \dots b_{n-1})$ represent the codeword, and

$$b_i = f(I_i), \forall i = 0 \dots n-1. \tag{2}$$

The I_i is namely the evaluation point of b_i. By definition, the codeword b can be divided into two parts, where the message part is located in

$$b_i = a_i = f(I_i), \forall i = 0 \dots k-1, \tag{3}$$

and the parity part is located in

$$b_j = f(I_j), \forall j = k \dots n-1. \tag{4}$$

To decode the message symbols, the decoding side should receive arbitrary k codeword symbols at least. To facilitate the description of algorithms, this paper adopts two types of notations to identify those received symbols. In the first decoding algorithm (Section IV), the received codeword is expressed as an n-element vector $\hat{b} = (\hat{b}_0, \hat{b}_1 \dots \hat{b}_{n-1})$, where $\hat{b}_j = b_j$ expresses the received symbol, or else $\hat{b}_j = 0$ for the erasures. In the second decoding algorithm (Section V), the k pairs of the received symbols are identified as $\{(\tilde{I}_j, \tilde{b}_j)\}_{j=0}^{k-1}$, where the $\tilde{b}_j$ denotes the received codeword symbol at the evaluation point $\tilde{I}_j$.

By following the second notation, the coding polynomial can be reconstructed via

$$f(x) = \sum_{i=0}^{k-1} \tilde{b}_i \prod_{j=0, j \neq i}^{k-1} \frac{x - \tilde{I}_j}{\tilde{I}_i - \tilde{I}_j}. \tag{5}$$

Thus, the message symbols can be retrieved through $\{a_i = f(I_i)\}_{i=0}^{k-1}$.

Based on the above coding system, the encoding process individually converts each sub-vector of F into a n-symbol codeword. Let the sequence F_i denote the $f(I_i)$ in those codewords. The n generated shadows are defined as $\{(I_i, F_i)\}_{i=0}^{n-1}$. To reduce the computational complexity, the key technique, namely Fermat number transform (FNT), is introduced in Section II-A. Then the basic principles of the proposed algorithms are briefly described in Section II-B, under conditions k being the power of two, and n being multiple of k. Based on the basic principles, the details of the three proposed algorithms are separately addressed in Sections III, IV and V. By the truncation technique in Section VIII-A, the proposed algorithms can serve for all feasible values of n and k.

A. Fermat number transforms

Fermat number transform (FNT) is defined as the discrete Fourier transform over the Fermat field $GF(2^r+1)$, and $r \in \{1, 2, 4, 8, 16\}$. Given a k-element vector $y = (y_0, y_1 \dots y_{k-1})$ with $k \le 2^r$ and k belonging to a power of two, the FNT of y is expressed as the function $Y = FNT_k(y)$, where the $Y = (Y_0, Y_1 \dots Y_{k-1})$ is a k-element vector. The inverse transform (IFNT) is expressed as the function $y = IFNT_k(Y)$. The relationship between Y and y is defined as

$$Y_i = \sum_{j=0}^{k-1} y_j w_k^{ij}; y_i = \frac{1}{k}\sum_{j=0}^{k-1} Y_j w_k^{-ij}, \tag{6}$$

where the w_k denotes the primitive k^{th} root of unity over Fermat field. The w_k can be found via $w_k = \alpha^{2^r/k}$, where α denotes the primitive element of Fermat field. Let $Y(x) = \sum_{j=0}^{k-1} Y_j x^j$ and $y(x) = \sum_{j=0}^{k-1} y_j x^j$, the (6) can be treated as the polynomial evaluations given by

$$Y_i = y(w_k^i); y_i = \frac{1}{k} Y(w_k^{-i}). \tag{7}$$

By applying radix-2 fast Fourier transform (FFT) over Fermat field, the k-point FNT takes $k\log_2 k$ additions and $0.5k\log_2 k$ multiplications, so that the computational cost is significantly lower than the ordinary approach with complexity $\mathrm{O}(k^2)$.

The shift theorem is utilized in the design of the proposed algorithms. Suppose the k-element vector $\tilde{Y} = (y(\Delta w_k^0), y(\Delta w_k^1) \dots y(\Delta w_k^{k-1}))$ is the desired objective to be calculated, where the Δ denotes an element in the finite field. Each term of $\tilde{Y}$ can be formulated as

$$y(\Delta w_k^i) = \sum_{j=0}^{k-1} (y_j \Delta^j) w_k^{ij}. \tag{8}$$

Let

$$A_k(\Delta) = (1, \Delta, \Delta^2 \dots \Delta^{k-1}) \tag{9}$$

denote a vector containing the twisted coefficients. Then we have $\tilde{Y} = FNT_k(A_k(\Delta) \otimes y)$, and the computation of $\tilde{Y}$ requires a k-point FNT and $k-1$ multiplications.

As the FNT is a linear transform, the addition operation $+$ possesses the equality:

$$FNT(Y_0) + FNT(Y_1) = FNT(Y_0 + Y_1). \tag{10}$$

This equality will be utilized in the design of the second decoding algorithm (Section IV-B).

An important application of FNT is for the fast polynomial multiplication $Y_0(x) \times Y_1(x) = \sum_{i=0}^{k-1} y_{0,i}x^i \times \sum_{i=0}^{k-1} y_{1,i}x^i$, or equivalently the convolution $Y_0 * Y_1$. By FNT, the fast convolution requires three times of $2k$-point FNTs:

$$Y_0 * Y_1 = IFNT_{2k}(FNT_{2k}(Y_0) \otimes FNT_{2k}(Y_1)), \tag{11}$$

where the operation $\otimes$ denotes the pair-wise product of two vectors.

B. Basic principles of proposed algorithms

This subsection describes the basic principles of the proposed coding algorithms, under the conditions that k is a power of two and n is multiple of k. Note that the proposed algorithms employ k-point FNTs, as opposed to the n-point FNTs used in previous work [12], [13], [14]. In decoding, as each received part of codeword (taken from the shadows) has the common evaluation points, the decoding process for each codeword involves a repetitive part of computations. This common part can be completed in one round, and the generated temporal data are used in the decoding of each codeword taken from the received shadows. This part is named as the initialization of the decoding algorithm. This subsection presents the design rationale of the algorithms, and the details are explained in the corresponding sections.

1) Encoding algorithm: By the definition of coding system, the encoding process consists of two coding phases. The first phase is to compute the coefficients of coding polynomial $f(x)$, and the second phase is to compute the codeword symbols $f(I_i)$. To apply the k-point FNT in encoding process, the codeword b is treated as n/k individual sub-vectors $b = (\hat{b}_0 \hat{b}_1 \dots \hat{b}_{n/k-1})$, and the length of each $\hat{b}_j$ is k. The first sub-vector $\hat{b}_0 = a$ is the message vector, and the remaining $n/k-1$ parity sub-vectors $\{\hat{b}_i\}_{i=1}^{n/k-1}$ are the objectives to be calculated. In the first phase, we choose the evaluation points of those message symbols such that the message symbols are exactly the input of a k-point IFNT. Thus, the coefficients of $f(x)$ can be efficiently computed via a k-point IFNT. In the second phase, each sub-vector $\hat{b}_i$ is computed via the k-point FNT on the coefficients of $f(x)$. By such computational structure, we can obtain the evaluation points for those parity sub-vectors $\{\hat{b}_i\}_{i=1}^{n/k-1}$. The details are explained in Section III.

2) First decoding algorithm: Section IV-A firstly explains a tentative version of the proposed decoding algorithm. The key polynomial is defined as $g(x) = f(x)\bar{L}(x)$, where the $f(x)$ is the coding polynomial, and the $\bar{L}(x)$ is defined in (17). By the structure of $\bar{L}(x)$, the formal derivative of $g(x)$ can be used to compute the erasures. The algorithm involves a n-point IFNT to compute the coefficients of $g(x)$. Then the coefficients of $g'(x)$ can be easily obtained within $\Theta(n)$ operations. A k-point

FNT is then applied to compute the erasures. However, the n-point IFNT requires $O(n \log n)$ operations, which is the bottleneck of the tentative algorithm. Section IV-B then presents the proposed algorithm. To further reduce the complexity, the n-symbol received codeword is divided into n/k individual sub-vectors with k symbols each, and each sub-vector takes a k-point IFNT. We proved that the above modified version generates the same result as the tentative algorithm does. As a result, the computational overhead is reduced to $O(n \log k)$.

3) Second decoding algorithm: The second decoding algorithm is modified from the fast polynomial interpolation [25]. The fast polynomial interpolation follows a divide-and-conquer strategy, and the FNT is utilized in the computations to reduce the complexity. The proposed decoding algorithm is motivated from two observations. Firstly, since the evaluation points of received symbols of all codewords are identical, the calculations requiring only the evaluation points can be completed in one round without repetitive calculations in later iterations. Secondly, based on the characteristics of the decoding system, several algorithm steps are modified to reduce the computational cost.

III. Encoding algorithm

As described in Section II-B1, the encoding process consists of two coding phases. To efficiently compute the coefficients of $f(x)$ in the first phase, each evaluation point of message symbol a_i is defined as w_k^i, so that $a_i = f(w_k^i), \forall i = 0 \dots k-1$. By (7), the equality $a = FNT_k(f)$ holds, so the coefficients $f(x)$ can be computed via the inverse transform $f = IFNT_k(a)$.

The second phase utilizes the shift theorem of FNT. The codeword is divided into n/k individual sub-vectors $b = (\hat{b}_0, \hat{b}_1 \dots \hat{b}_{n/k-1})$, and each sub-vector is defined as $\hat{b}_j = (\tilde{b}_j[0], \tilde{b}_j[1] \dots \tilde{b}_j[k-1])$. For $i = 0 \dots k-1$, the evaluation point of $\tilde{b}_j[i]$ is defined as $\Delta_j w_k^i$ with the shift amount Δ_j, so $\tilde{b}_j[i] = f(\Delta_j w_k^i), \forall i = 0 \dots k-1$. By the shift theorem (8), the above equality can be rewritten as

$$\tilde{b}_j[i] = f(\Delta_j w_k^i) = \sum_{l=0}^{k-1} (f_i \Delta_j^l) w_k^{il}. \tag{12}$$

Thus, the computation $\hat{b}_j$ requires a k-point FNT and $k-1$ multiplications:

$$\hat{b}_j = FNT_k(A_k(\Delta_j) \otimes f). \tag{13}$$

It is noted that for $\Delta_j = 1$, the transform generates the message symbols:

$$\hat{b}_j = FNT_k(A_k(1) \otimes f) = FNT_k(f) = a.$$

The $\{\Delta_j\}_{i=1}^{n/k-1}$ can be chosen as $n/k-1$ distinct elements among $\{\alpha^i | 1 \le i < 2^r/k\}$. To facilitate the description of the algorithm, each element is defined as $\Delta_j = \alpha^j, \forall j = 1 \dots n/k-1$. The above definition gives a systematic definition of evaluation points of n codeword symbols. For each evaluation point I_l of codeword symbol b_l, the value l is can be factorized as $l = i + kj$, where $i = l(\text{mod } k)$ and $j = l/k$. The range of i and j are $0 \le i \le k-1$, and $0 \le j \le n/k-1$. By above definition, the evaluation point is set as

$$I_l = I_{i+kj} = \alpha^{j+i \times 2^r/k} = \alpha^j w_k^i, \forall l = 0 \dots n-1. \tag{14}$$

The encoding algorithm is provided below.

Algorithm 1: The encoder.
Input: k message symbols $a = (a_0, a_1 \dots a_{k-1})$.
Output: $n-k$ parity symbols.
Main procedure:

1) Compute the vector $f = IFNT_k(a)$.
2) For $j = 1 \dots n/k-1$, compute the vector $\hat{b}_j = FNT_k(A_k(\alpha^j) \otimes f)$. Each $\hat{b}_j$ contains k parity symbols, so there are $(n/k-1)k = n-k$ symbols in the output.

IV. First decoding algorithm

This section explains the first decoding algorithm in the following two sub-sections. The first sub-section gives the formulas of the tentative algorithm within complexity $O(n \log n)$. In the second sub-section, a modified version is presented to improve the complexity to $O(n \log k)$. In this section, the received codeword is expressed as an n-element vector $\hat{b} = (\hat{b}_0, \hat{b}_1 \dots \hat{b}_{n-1})$. Let $\hat{l}$ denote the set of locations of k received codeword symbols. Then the received codeword symbols are expressed as

$$\hat{b}_j = \begin{cases} b_j & \text{if } j \in \hat{l}; \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

By the above definition, the polynomial $L(x)$ is defined as

$$L(x) = \prod_{j \in \hat{l}} (x - I_j). \tag{16}$$

The algorithm of computing the coefficients of $L(x)$ is placed in Appendix A.

A. The tentative algorithm

Two polynomials are respectively defined as

$$\bar{L}(x) = (x^{2^r} - 1)/L(x), \tag{17}$$

$$g(x) = f(x)\bar{L}(x) = \sum_{i=0}^{2^r-1} g_i x^i, \tag{18}$$

where the $f(x)$ is the coding polynomial (1). As the roots of $(x^{2^r} - 1)$ are all the non-zero elements of $GF(2^r+1)$, the roots of $\bar{L}(x)$ are the complement of the roots of $L(x)$ over the $\{\alpha^i\}_{i=0}^{2^r-1}$, i.e., $\bar{L}(I_j) = 0, \forall j \notin \hat{l}$. Suppose the coefficients of $g(x)$ had been computed. Then the formal derivative of $g(x)$, expressed as $g'(x) = \sum_{i=1}^{2^r-1} i g_i x^{i-1}$, can be computed within 2^r multiplications. By product rule, the $g'(x)$ possesses the equality

$$g'(x) = f'(x)\bar{L}(x) + f(x)\bar{L}'(x). \tag{19}$$

For any $j \notin \tilde{l}$, the equality $\bar{L}(I_j) = 0$ gives the following formula:

$$g'(I_j) = f(I_j)\bar{L}'(I_j), \forall j \notin \tilde{l}. \tag{20}$$

Thus, the erasures can be computed via

$$f(I_j) = g'(I_j)\bar{L}'(I_j)^{-1}, \forall j \notin \tilde{l}. \tag{21}$$

The above formula can be further refined. A polynomial $h(x) = \sum_{i=0}^{k-1} h_i x^i$ is defined as

$$h(x) = xg'(x) \bmod (x^k - 1). \tag{22}$$

As the roots of $(x^k - 1)$ are $\{w_k^i\}_{i=0}^{k-1}$, the $h(x)$ possesses the equality:

$$h(x) = xg'(x),\ \forall x \in \{w_k^i\}_{i=0}^{k-1}.$$

By the definition (14), the evaluation points of message symbols are the $\{w_k^i\}_{i=0}^{k-1}$. Thus, the term $g'(I_j)$ in (21) can be replaced with $h(I_j)I_j^{-1}$, resulting in

$$f(I_j) = h(I_j)(I_j \bar{L}'(I_j))^{-1}, \forall j \notin \tilde{l} \text{ and } 0 \le j \le k-1. \tag{23}$$

As the degree of $h(x)$ is smaller than $g'(x)$, the evaluation $h(I_j)$ in (23) takes fewer arithmetic operations than the $g'(I_j)$ in (21). In the following, the algorithms of computing $g(x)$ and $\{h(w_k^j)\}_{j=0}^{k-1}$ are presented respectively.

1) Algorithm of computing the coefficients of $g(x)$: To compute the coefficients of $g(x)$, we examine the values $\{g(\alpha^j)\}_{j=0}^{2^r-1}$ given by

$$g(\alpha^j) = f(\alpha^j)\bar{L}(\alpha^j) = \begin{cases} \hat{b}_j \bar{L}(I_j), & \text{if } j \in \hat{l}; \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

Those values are formed as a 2^r-point vector

$$G = (g(\alpha^0), g(\alpha^1) \dots g(\alpha^{2^r-1})). \tag{25}$$

The computation of G requires the values $\bar{L}(I_j)$ defined as a set

$$\beta = \{\bar{L}(I_j) | j \in \hat{l}\}, \tag{26}$$

As the elements in the β require only the evaluation points, the (26) can be calculated in the algorithm initialization. Then, the coefficients of $g(x)$ are computed via

$$g = IFNT_{2^r}(G). \tag{27}$$

2) Algorithm of computing the (23): The terms $\{h(w_k^j)\}_{j=0}^{k-1}$ in (23) can be computed via

$$H = FNT_k(h), \tag{28}$$

and the required values $h(I_j)$ are in the vector H. Notably, if the computational step follows the (21), the terms $\{g'(I_j)\}_{j=0}^{k-1}$ can be computed via a 2^r-point FNT $FNT_{2^r}(g')$, which requires more operations than the (28).

The (23) also requires the terms $(I_j \bar{L}'(I_j))^{-1}$, whose evaluations require only the evaluation points, so those values can be calculated in initialization. Those terms are defined as a set

$$\hat{\beta} = \{(I_j \bar{L}'(I_j))^{-1} | j \notin \hat{l} \text{ and } 0 \le j \le k-1\}. \tag{29}$$

3) Steps of tentative algorithm: In summary, a tentative method is presented to calculate the erasures. In the algorithm initialization, the sets β and $\hat{\beta}$ respectively defined in (26) and (29) are computed. Another algorithm of computing β and $\hat{\beta}$ is introduced in Appendix B-A. In the main procedure, the first step calculates the vector G through (24). The second step calculates the vector g via (27) via a 2^r-point FNT. The third step calculates the coefficients of $h(x)$ defined in (22). The final step calculates the erased message symbols through (23).

B. The proposed algorithm

It can be shown that the complexity of the above tentative algorithm is dominated by a 2^r-point IFNT to compute the (27). To reduce the computational complexity further, a new formula of $h(x)$ is presented below.

Given the vector G defined in (25), all elements of G are classified as $2^r/k$ individual sub-vectors $[G_0, G_1 \dots G_{2^r/k-1}]$, where each k-element sub-vector $G_j = (g(\alpha^j), g(\alpha^j w_k^1) \dots g(\alpha^j w_k^{k-1}))$. For each G_j, we define the corresponding polynomial $g_j(x) = \sum_{i=0}^{k-1} g_i^{[j]} x^i$ to satisfy the equalities:

$$g_j(\alpha^j w_k^i) = g(\alpha^j w_k^i), \forall i = 0 \dots k-1. \tag{30}$$

By the following formulas, the coefficients of $g_j(x)$ can be efficiently computed from the G_j. By shift theorem of FNT, the $g_j(\alpha^j w_k^i)$ can be formulated as

$$g_j(\alpha^j w_k^i) = \sum_{t=0}^{k-1} g_t^{[j]} (\alpha^j w_k^i)^t = \sum_{t=0}^{k-1} (\alpha^{jt} g_t^{[j]}) w_k^{it}. \tag{31}$$

By (6), the inversion of above formula is

$$\alpha^{jt} g_t^{[j]} = \frac{1}{k} \sum_{t=0}^{k-1} g_j(\alpha^j w_k^i) w_k^{-it}. \tag{32}$$

As the $\{g_j(\alpha^j w_k^i)\}_{i=0}^{k-1}$ are the terms of G_j, the $IFNT_k(G_j)$ outputs the values $\{\alpha^{jt} g_t^{[j]}\}_{t=0}^{k-1}$. Then each value is divided by α^{jt} to obtain the coefficients of $g_j(x)$, expressed as

$$g^{[j]} = A_k(\alpha^{-j}) \otimes IFNT_k(G_j), \tag{33}$$

where $A_k(\alpha^{-j})$ is a vector defined in (9). In the following, we derive the formula of $h(x)$ as in those functions $\{g_j(x)\}_{j=0}^{2^r/k-1}$.

The formula of expressing the $g(x)$ as in $\{g_j(x)\}_{j=0}^{2^r/k-1}$ is given by

$$g(x) = \frac{k}{2^r} \sum_{j=0}^{2^r/k-1} T_j(x) g_j(x), \text{ and } T_j(x) = \frac{x^{2^r} - 1}{w_{2^r/k}^{-j} x^k - 1}. \tag{34}$$

The validity of (34) is proved in Appendix B-B. The formal derivative of (34) is formulated as

$$g'(x) = \frac{k}{2^r} \sum_{j=0}^{2^r/k-1} (T_j'(x) g_j(x) + T_j(x) g_j'(x)), \tag{35}$$

where the derivative of $T_j(x)$ is

$$T_j'(x) = \frac{w_{2^r/k}^j x^{k-1} [(2^r - k)x^{2^r} - w_{2^r/k}^j 2^r x^{2^r-k} + k]}{(x^k - w_{2^r/k}^j)^2}. \tag{36}$$

We plug (35) into (22) to obtain the objective formula

$$h(x) = xg_0'(x) + \frac{2^r - k}{2} g_0(x) + \sum_{j=1}^{2^r/k-1} \frac{k g_j(x)}{w_{2^r/k}^{-j} - 1}. \tag{37}$$

Thus, the $h(x)$ can be calculated from $\{g_j(x)\}_{j=0}^{2^r/k-1}$.

The (37) can be refined further. Firstly, the term $(2^r - k)/2 \times g_0(x)$ can be discarded, due to $g_0(w_k^j) = g(w_k^j) = 0$, $\forall j \notin$

$\hat{l}$ by (30)(24). Secondly, the term $\sum_{j=1}^{2^r/k-1} \frac{kg_j(x)}{w_{2^r/k}^{-j}-1}$ can be simplified as $\sum_{j=1}^{n/k-1} \frac{kg_j(x)}{w_{2^r/k}^{-j}-1}$ due to $g_j(x)=0$ for $j \geq n/k$. To explain this property, by the definition of evaluation point (14), each erasure position is expressed as $l = i + kj$ for $0 \leq j \leq n/k-1$. Thus, there do not exist erasures in $j \geq n/k$, so the set $\tilde{l}$ does not contain the erasures in $j \geq n/k$. For $j \geq n/k$, the definition (24) gives $g(\alpha^{i+kj}) = 0$, so the G_j is a zero vector. The polynomial $g_j(x)$, which is computed via $IFNT_k(G_j)$, is also zero. By the above observations, the modified polynomial is

$$\tilde{h}(x) = xg_0'(x) + \sum_{j=1}^{n/k-1} \frac{kg_j(x)}{w_{2^r/k}^{-j}-1}. \quad (38)$$

The decoding algorithm is summarized below:

Algorithm 2: The first decoder.

Input: A codeword $\hat{b}$ with $n-k$ erasures.

Output: The erased message symbols a_j, $\forall j \notin \hat{l}$ and $j = 0 \dots k-1$.

Initialization: Compute the sets β and$\hat{\beta}$ respectively defined in (26) and (29).

Main procedure:

1) Compute the vector G defined in (24), where the term $\bar{L}(I_j)$ is taken from the set β. Then divide the G into $2^r/k$ sub-vectors $G = (G_0, G_1 \dots G_{2^r/k-1})$, and each G_j contains k elements.
2) For each G_j, compute the corresponding polynomial $g_j(x)$ defined in (33).
3) Compute the $\tilde{h}(x)$ defined in (38).
4) Compute the (28), where the vector h uses the coefficients of $\tilde{h}(x)$. Specifically, compute $H = FNT_k(\tilde{h})$.
5) Compute the erasures through (23), where the values $h(w_k^j)$ are taken from the vector H, and the terms $(w_k^j \bar{L}'(w_k^j))^{-1}$ are taken from the set $\hat{\beta}$.

V. Second decoding algorithm

For the scenario $n >> k$, the complexity of first decoding algorithm $O(n \log k)$ may be higher than the ordinary approach $O(k^2)$. This section introduces another decoding algorithm within complexity $O(k \log^2 k)$. The first subsection reviews the fast polynomial interpolation [25], and the second subsection explains our modified version to reduce its leading constant. In this section, the k received codeword symbols with the corresponding evaluation points are denoted as $\{(I_j, b_j) | j \in \tilde{l}\}$.

A. Fast polynomial interpolation algorithm

The decoding formula (5) can be reformulated as

$$\begin{aligned} f(x) &= \sum_{i=0}^{k-1} \tilde{b}_i \frac{\prod_{j=0,j\neq i}^{k-1}(x-\tilde{I}_j)}{\prod_{j=0,j\neq i}^{k-1}(\tilde{I}_i-\tilde{I}_j)} \\ \Rightarrow f(x) &= \sum_{i=0}^{k-1} \tilde{b}_i \frac{\prod_{j=0}^{k-1}(x-\tilde{I}_j)}{(x-\tilde{I}_i)\prod_{j=0,j\neq i}^{k-1}(\tilde{I}_i-\tilde{I}_j)} \\ \Rightarrow f(x) &= \prod_{j=0}^{k-1}(x-\tilde{I}_j) \sum_{i=0}^{k-1} \frac{\tilde{b}_i}{(x-\tilde{I}_i)\prod_{j=0,j\neq i}^{k-1}(\tilde{I}_i-\tilde{I}_j)} \\ \Rightarrow \frac{f(x)}{L(x)} &= \sum_{i=0}^{k-1} \frac{\tilde{b}_i}{(x-\tilde{I}_i)L'(\tilde{I}_i)}, \end{aligned} \quad (39)$$

where the polynomials $L(x)$ and its derivative are respectively defined as

$$L(x) = \prod_{j=0}^{k-1}(x-\tilde{I}_j);$$

$$L'(x) = \sum_{i=0}^{k-1} \prod_{j=0,j\neq i}^{k-1} (x-\tilde{I}_j).$$

Let

$$\{f_i^{[0]}(x) = \tilde{b}_i / L'(\tilde{I}_i)\}_{i=0}^{k-1} \quad (40)$$

denote a set of k constant polynomials. Then the (39) can be rewritten as

$$\frac{f(x)}{L(x)} = \sum_{i=0}^{k-1} \frac{f_i^{[0]}(x)}{x-\tilde{I}_i}. \quad (41)$$

The right-hand side is the summation of k partial fractions. The fast polynomial interpolation is a method to sum all those partial fractions together. The result is a fraction whose denominator is the $L(x)$, and the numerator is the standard form of polynomial $f(x)$. The algorithm steps [25] are addressed below:

1) Calculate the coefficients of $L(x)$ and $L'(x)$.
2) Construct the set
$$\gamma = \{1/L'(\tilde{I}_i)\}_{i=0}^{k-1}. \quad (42)$$
3) Construct k constant polynomials defined as (40).
4) Compute the summation (41) to obtain a fraction whose numerator polynomial is the $f(x)$.

In step 1, the coefficients of $L(x)$ can be calculated within $O(k \log^2 k)$ as introduced in Appendix A. Then the coefficients of $L'(x)$ can be directly obtained within k multiplications. In Step 2, the fast polynomial evaluation algorithm requires $O(k \log^2 k)$ operations. The details can be referred to the Chapter 1-4 of [25] for more information. In Step 3, each value $1/L'(\tilde{I}_i)$ is taken from the set γ, so the construction of set (40) requires k multiplications. In the final step, the numerator $f(x)$ is computed by a divide-and-conquer strategy. It is noted that the temporary data generated in step 1 can be reused in this step. This step consists of $\log_2 k$ stages. Initially, the process constructs k individual fractions $\{f_i^{[0]}(x)/L_i^{[0]}(x)\}_{i=0}^{k-1}$, where $L_i^{[0]}(x) = (x - \tilde{I}_i)$.

In the first stage, the process pairwise sums those k partial fractions to obtain $k/2$ partial fractions:

$$\begin{aligned} & \frac{f_{2i}^{[0]}(x)}{L_{2i}^{[0]}(x)} + \frac{f_{2i+1}^{[0]}(x)}{L_{2i+1}^{[0]}(x)} \\ = & \frac{f_{2i}^{[0]}(x)L_{2i+1}^{[0]}(x) + f_{2i+1}^{[0]}(x)L_{2i}^{[0]}(x)}{L_{2i}^{[0]}(x)L_{2i+1}^{[0]}(x)} \\ = & \frac{f_i^{[1]}(x)}{L_i^{[1]}(x)}, \forall i = 0 \dots k/2-1. \end{aligned} \quad (43)$$

Note that the denominator can be taken from the temporary data (64) which have been generated in step 1. The denominator and the computed numerator are respectively labeled as $L_i^{[1]}(x)$ and $f_i^{[1]}(x)$. In the j^{th} stage $(1 \leq j \leq \log_2 k)$, the process follows such recursive strategy to compute

$$\frac{f_i^{[j]}(x)}{L_i^{[j]}(x)} = \frac{f_{2i}^{[j-1]}(x)}{L_{2i}^{[j-1]}(x)} + \frac{f_{2i+1}^{[j-1]}(x)}{L_{2i+1}^{[j-1]}(x)} \; \forall i = 0 \dots k/2^j - 1. \quad (44)$$

The numerator is formulated as

$$f_i^{[j]}(x) = f_{2i}^{[j-1]}(x)L_{2i+1}^{[j-1]}(x) + f_{2i+1}^{[j-1]}(x)L_{2i}^{[j-1]}(x). \quad (45)$$

It is noted that the denominator has been computed in step 1 (see (65)), and we do not need to compute it again. In the final stage, the process generates the fraction $f_0^{[\log_2 k]}(x)/L_0^{[\log_2 k]}(x)$, and the numerator is the coefficients of $f(x)$.

The computation (45) requires two polynomial multiplications and a polynomial addition. By applying fast polynomial multiplication, the computation (45) can be done via six times of 2^j-point FNT:

$$\begin{aligned} f_i^{[j]} = \; & IFNT_{2^j}((FNT_{2^j}(f_{2i}^{[j-1]}) \otimes FNT_{2^j}(L_{2i+1}^{[j-1]}))+ \\ & IFNT_{2^j}(FNT_{2^j}(f_{2i+1}^{[j-1]}) \otimes FNT_{2^j}(L_{2i}^{[j-1]})). \end{aligned} \quad (46)$$

This step (46) requires $\mathrm{O}(2^j \log 2^j)$ operations. Totally, the fast polynomial interpolation takes $\mathrm{O}(k \log^2 k)$ operations. When the coefficients of $f_0^{[\log_2 k]}$ have been obtained, the message is calculated via

$$a = FNT_k(f_0^{[\log_2 k]}). \quad (47)$$

B. The proposed algorithm

In the following, we list the observations which are used for the construction of the proposed algorithm. Then the decoding algorithm is presented.

1) Reduce the number of FNTs in the computation of $f_i^{[j]}$: In (46), the computation of $f_i^{[j]}$ requires six times of 2^j-point FNT. To facilitate the expression, we define

$$\hat{F}_i^{[j-1]} = FNT_{2^j}(f_i^{[j-1]}). \quad (48)$$

By the linearity of FNT (10), the (46) can be rewritten as

$$f_i^{[j]} = IFNT_{2^j}(F_i^{[j]}), \quad (49)$$

where the $F_i^{[j]}$ is a 2^j-point vector defined as

$$\begin{aligned} F_i^{[j]} = \; & \hat{F}_{2i}^{[j-1]} \otimes FNT_{2^j}(L_{2i+1}^{[j-1]})+ \\ & \hat{F}_{2i+1}^{[j-1]} \otimes FNT_{2^j}(L_{2i}^{[j-1]}). \end{aligned} \quad (50)$$

By (49)(50), the computation $f_i^{[j]}$ only requires five times of 2^j-point FNT.

2) Reduce the size of FNT: The size of FNT used in (49)(50) can be reduced. In the $(j-1)^{\text{th}}$ stage, the process computes $f_i^{[j-1]} = IFNT_{2^{j-1}}(F_i^{[j-1]})$ by (49). Then in the next stage, the (50) requires the set $\{\hat{F}_i^{[j-1]}\}_{i=0}^{k/2^{j-1}-1}$ as the input. By the above observation, the (48) is rewritten as

$$\begin{aligned} \hat{F}_i^{[j-1]} &= FNT_{2^j}(f_i^{[j-1]}) \\ &= FNT_{2^j}(IFNT_{2^{j-1}}(F_i^{[j-1]})). \end{aligned} \quad (51)$$

Notably, the $FNT_{2^j}(IFNT_{2^{j-1}}(F_i^{[j-1]}))$ represents a 2^j-point vector containing the 2^{j-1}-point vector $F_i^{[j-1]}$ at the even positions. As the $F_i^{[j-1]}$ has been obtained in the previous stage, the objective is to compute the other elements, denoted as a 2^{j-1}-point vector $\hat{F}_{i,1}^{[j-1]}$. The $\hat{F}_{i,1}^{[j-1]}$ can be calculated via

$$\hat{F}_{i,1}^{[j-1]} = FNT_{2^{j-1}}(A_{2^{j-1}}(w_{2^j}) \otimes f_i^{[j-1]}), \quad (52)$$

where $A_{2^{j-1}}(w_{2^j})$ is a vector defined in (9). Equivalently, we can plug $f_i^{[j-1]} = IFNT_{2^{j-1}}(F_i^{[j-1]})$ into (52) to obtain

$$\hat{F}_{i,1}^{[j-1]} = FNT_{2^{j-1}}(A_{2^{j-1}}(w_{2^j}) \otimes IFNT_{2^{j-1}}(F_i^{[j-1]})). \quad (53)$$

The objective $\hat{F}_i^{[j-1]}$ is expressed as

$$\hat{F}_i^{[j-1]} = Merge(F_i^{[j-1]}, \hat{F}_{i,1}^{[j-1]}), \quad (54)$$

which merges two vectors with placing the elements of $F_i^{[j-1]}$ and $\hat{F}_{i,1}^{[j-1]}$ in even and odd positions, respectively.

3) Remove the redundant computations in the final stage: In the final stage of the above algorithm, the process computes $F_0^{[\log_2 k]}$ by (50) and the $f_0^{[\log_2 k]} = IFNT_k(F_0^{[\log_2 k]})$ by (49). Then the process computes the (47), which can be rewritten as

$$a = FNT_k(f_0^{[\log_2 k]}) = FNT_k(IFNT_k(F_0^{[\log_2 k]})) = F_0^{[\log_2 k]}. \quad (55)$$

Thus, the message is $a = F_0^{[\log_2 k]}$, and the (49) in the final stage and the (47) are redundant in the decoding system.

4) The calculations requiring only the evaluation points: There exists two sets of auxiliary data computed in algorithm initialization. The first set is the γ defined in (42). The second set is defined as

$$\hat{\gamma}_j = \{FNT_{2^j}(L_i^{[j-1]}) | i = 0 \dots k/2^j - 1\}, \forall j = 1 \dots \log_2 k. \quad (56)$$

This set is used in the (50). Based on those modifications, the second decoding algorithm is presented below.

Algorithm 3: The second decoder.
Input: k received codeword symbols $\{(\tilde{I}_i, \tilde{b}_i) | i = 0 \dots k-1\}$.
Output: The message vector a.
Initialization: Compute the sets γ and $\{\hat{\gamma}_j | j = 1 \dots \log_2 k\}$ as respectively defined in (42)(56).
Main procedure:

1) For $i = 0 \dots k-1$, compute the $F_i^{[0]}$ through

$$F_i^{[0]} = FNT_1(f_i^{[0]}) = \tilde{b}_i / L'(\tilde{I}_i), \forall i = 0 \dots k-1. \quad (57)$$

It is noted that the term $L'(\tilde{I}_i)$ is taken from the γ. Let $j = 1$.

2) For $i = 0 \dots k/2^{j-1} - 1$, compute the vector $\hat{F}_i^{[j-1]}$ through (54)(53).
3) For $i = 0 \dots k/2^j - 1$, compute the vector $F_i^{[j]}$ through (50), where the terms $FNT_{2^j}(L_{2i+1}^{[j-1]})$ and $FNT_{2^j}(L_{2i}^{[j-1]})$ are taken from $\hat{\gamma}_j$.
4) If $j = \log_2 k$, output $a = F_0^{[\log_2 k]}$; else $j = j+1$ and then goto step two.

We briefly compare the computational cost of the proposed version (50) with the original version (46). In the j^{th} stage, the original version totally requires $3k/2^{j-1}$ times of 2^j-point FNT, and the proposed version requires $2k/2^{j-1}$ times of 2^{j-1}-point FNT. Since the cost of a 2^j-point FNT is roughly equivalent to two 2^{j-1}-point FNTs, the modified version (48) requires about $1/3$ amount of computational cost of the original formula (46).

TABLE I: Complexities of the proposed algorithms.

	Operation counts of main procedure		Complexity
	Additions	Multiplications	
Encoder	$n\log_2 k$	$0.5n\log_2 k$	$O(n\log k)$
First decoder	$(n+k)\log_2 k$	$0.5(n+k)\log_2 k$	Init.:$O(k\log^2 k)$ Main:$O(n\log k)$
Second decoder	$k\log_2^2 k$	$0.5k\log_2^2 k$	Init.:$O(k\log^2 k)$ Main:$O(k\log^2 k)$

VI. Complexity analysis

By the radix-2 FFT over Fermat field, the k-point FNT takes $\phi^+(k)=k\log_2 k$ additions and $\phi^\times(k)=0.5k\log_2 k$ multiplications. This section discusses the computational complexities of proposed algorithms, and the analysis results are listed in Table I.

A. The encoder (Algorithm 1)

In Algorithm 1, step 1 requires a k-point FNT, and step 2 requires $n/k-1$ times of k-point FNT. Thus, Algorithm 1 overall requires n/k times of k-point FNT, which takes $n/k\times\phi^+(k)=n\log_2 k$ additions and $n/k\times\phi^\times(k)=0.5n\log_2 k$ multiplications.

B. The first decoder(Algorithm 2)

In the main procedure of Algorithm 2, step 1 requires k multiplications, and step 2 requires n/k times of k-point FNTs. Step 3 takes $O(n)$ additions and multiplications. Step 4 requires a k-point FNT, and step 5 takes $O(k)$ multiplications. The complexity is dominated by the $n/k+1$ times of k-point FNTs, which totally takes $(n/k+1)\times\phi^+(k)=(n+k)\log_2 k$ additions and $(n/k+1)\times\phi^\times(k)=0.5(n+k)\log_2 k$ multiplications.

The initialization of Algorithm 2 constructs the two sets β and $\hat{\beta}$, which require the polynomial coefficients of $L(x)$ with complexity $O(k\log^2 k)$ via Appendix A. Since the cost of initialization is dominated by the computation of $L(x)$, the initialization requires complexity $O(k\log^2 k)$.

C. The second decoder(Algorithm 3)

In the main procedure of Algorithm 3, step 1 requires k multiplications. Steps 2-4 are formulated as a loop with $\log_2 k$ iterations. For $j=1\ldots\log_2 k$, step 2 requires $2(k/2^{j-1})$ times of 2^{j-1}-point FNTs with complexity $2(k/2^{j-1})\times O(2^{j-1}\log 2^{j-1})=O(2k\log 2^{j-1})$, and step 3 requires $O(k)$ multiplications and additions. Thus, the whole complexity is dominated by step 2, which requires $\sum_{j=1}^{\log_2 k}\frac{2k}{2^{j-1}}\phi^+(2^{j-1})\approx k\log_2^2 k$ additions, and $\sum_{j=1}^{\log_2 k}\frac{2k}{2^{j-1}}\phi^\times(2^{j-1})\approx 0.5k\log_2^2 k$ multiplications.

The initialization of Algorithm 3 constructs the sets γ and $\{\hat{\gamma}_j\}_{j=1}^{\log_2 k}$. The γ requires the formal derivative of $L(x)$, and the construction of $\hat{\gamma}_j$ requires the temporary data (65) of computing $L(x)$. Since the cost of initialization is dominated by the computation of $L(x)$, the initialization requires complexity $O(k\log^2 k)$.

VII. Comparisons

This section compares the proposed algorithms with existing work. The time complexity of the proposed decoding algorithm can be treated as $min\{\mathrm{O}(n\log k),\mathrm{O}(k\log^2 k)\}$ by selecting a better algorithm among Algorithms 2 and 3. Table II lists the computational complexities of the compared methods with the leading constants. The complexities of those methods are dominated by the cost of computing FNTs, or the cost of matrix multiplications. The leading constant represents the overhead of the dominated part in the method. The coding system of [1] can be treated as Cauchy RS code, and the codes [12], [13], [14], [15], [16] and ours are isomorphic under Vandermonde RS codes. Those codes have the property that the n is no larger than the size of used finite field. Thus, to conduct a fair comparison, we consider that the codes [12], [13], [14], [15] and ours are over the Fermat field $GF(2^r+1)$, and the [16], [1] are over $GF(2^r)$.

It is noted that the RS code possess the condition that n(or $n-1$) must be no larger than the size of utilized field size. Thus, under a specific n, the minimal field size of those RS-like codes are asymptotically equivalent. For the systematic codes, such as [13], [14], [15], [16] and ours, we consider the situation that all received symbols are in the parity part, so the decoder cannot directly retrieve the message from the received symbols. In the following, the dominant parts of the methods are briefly explained. Notably, an n-point FNT takes $n\log_2 n$ additions and $0.5n\log_2 n$ multiplications, so the n-point FNT requires $1.5n\log_2 n$ operations.

The conventional method of IDA is introduced by Rabin [1], which employs matrix multiplications in encoding and decoding. The encoding algorithm requires about $2nk$ operations to multiply the k message symbols with an $n\times k$ Cauchy matrix, and the decoder takes about $2k^2$ operations to multiply the k codeword symbols with the $k\times k$ inverse matrix. The [12], [13], [14], [15] are FNT-based algorithms. For encoding, the [12] and non-systematic cases of [13], [14] employ a n-point FNT on the message symbols, resulting in n codeword symbols. For decoding, the [12], [13], [14], [15] respectively introduce three kinds of decoding methods. The [12] firstly computes the $n-k$ un-received symbols with matrix multiplications, and then an n-point IFNT is applied on the n codeword symbols to obtain the decoded data. The [13] employs a n-point IFNT and a recursion formula within $2(n-k)k$ operations. The [14] uses fast convolutions to calculate the coefficients of the Lagrange polynomial within eight times of n-point FNT.

The [13], [14], [15] also introduce the systematic coding algorithms. The encoding of [13] is similar to conventional RS encoding by polynomial division. The information polynomial is divided by the generator polynomial, resulting in the parity polynomial containing $n-k$ parity symbols. The polynomial division requires $2(n-k)k$ operations. For decoding, the systematic case of [13] is similar to the non-systematic version [13], but then applies an n-point FNT on the generated intermediate data. The systematic approach of [14] takes eight times of n-point FNT to compute the coefficients of Lagrange polynomial, and then the FNT co-

TABLE II: Comparisons with previous work.

	Systematic(Yes/No)	Complexities of encoding algorithm	Complexities of decoding algorithm
Rabin[1]	N	$2nk = O(nk)$	$2k^2 = O(k^2)$
Preparata[12]	N	$1.5n\log_2 n = O(n\log n)$	$2n(n-k) + 1.5n\log_2 n = \mathrm{O}(n(n-k+\log n))$
Dianat and Marvasti[13]	N	$1.5n\log_2 n = O(n\log n)$	$1.5n\log_2 n + 2(n-k)k = O(n\log n + k(n-k))$
	Y	$2(n-k)k = O(k(n-k))$	$3n\log_2 n + 2(n-k)k = O(n\log n + k(n-k))$
Soro and Lacan[14]	N	$1.5n\log_2 n = O(n\log n)$	$8 \times 1.5n\log_2 n = O(n\log n)$
	Y	$9 \times 1.5n\log_2 n = O(n\log n)$	$9 \times 1.5n\log_2 n = O(n\log n)$
Lacan and Fimes[15]	Y	$1.5k\log_2 k + \epsilon(\max\{k, n-k\})$ $= \mathrm{O}(k\log^2 k + (n-k)\log^2(n-k))$	$1.5k\log_2 k + \iota(k) = \mathrm{O}(k\log^2 k)$
Didier[16]	Y	$(3\log_2 n + 1) \times n\log_2 n = \mathrm{O}(n\log^2 n)$	$(3\log_2 n + 1) \times n\log_2 n = \mathrm{O}(n\log^2 n)$

TABLE III: Comparisons with Lin and Chung [38].

	Lin and Chung [38]	The proposed algorithm	
Range of par.	$k \geq n/2$	$k \leq n/2$	
Enc. comp.	$O(n\log(n-k))$	$O(n\log k)$	
Dec. comp.	$O(n\log(n-k))$	$O(n\log k)$	$O(k\log^2 k)$
Dec. formula	Forney algorithm	New formulation	Fast Lagrange interpolation

efficients are computed through an n-point FNT. The [15] transforms the message symbols with a k-point IFNT and the fast polynomial evaluation, and the decoding algorithm uses fast polynomial interpolation and a k-point FNT. By the [32], the $\epsilon(t) = O(t\log^2 t)$ denotes the complexity of polynomial evaluation, and $\iota(t) = O(t\log^2 t)$ denotes the complexity of polynomial interpolation.

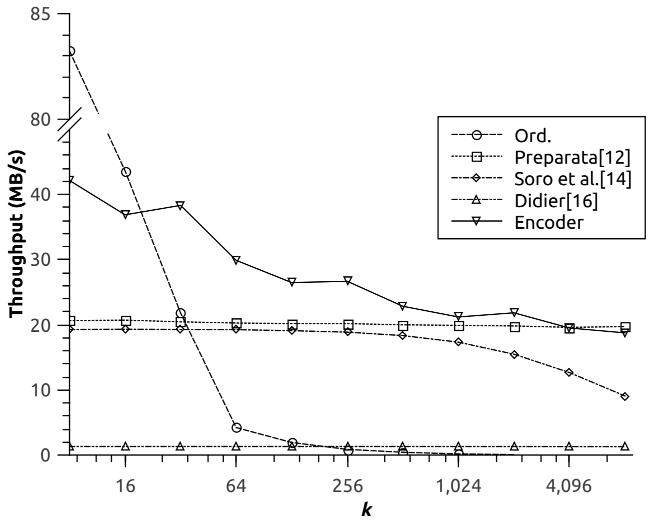
The [16] introduces the encoding and decoding systematic erasure RS codes over $GF(2^r)$. Rather than most algorithms using fast Fourier transforms, the [16] introduces a coding algorithm via fast Walsh transforms. An n-point Walsh transform takes $n\log_2 n$ additions but no multiplications. However, the [16] requires r times of n-point Walsh transform in both encoding and decoding, so the complexities are $O(n\log^2 n)$.

As shown in Table II, the best cases of the compared encoding algorithms require $\mathrm{O}(n\log n)$ operations [12], [13], [14], where the non-systematic cases [12], [13], [14] require a n-point FNT, and the systematic case [14] requires nine times of n-point FNT. The Algorithm 1 takes about n/k times of k-point FNT operations. For $k << n$, the proposed encoding algorithm requires fewer operations than the compared methods. For decoding, the best performance of the compared decoding algorithms takes $\mathrm{O}(n\log n)$ [14] or $O(k\log^2 k)$ operations [15]. The [14] requires nine times of n-point FNT for systematic case. By comparing Algorithm 2 with [14], the Algorithm 2 has smaller big-O complexity ($O(n\log k)$ vs. $O(n\log n)$) and smaller leading constant (1.5 vs.13.5). The systematic codes of [15] requires about $\mathrm{O}(k\log^2 k)$ operations by fast polynomial interpolation. As shown in Section V-B, the Algorithm 2 has smaller leading constant than the ordinary fast polynomial interpolation, so we expect that Algorithm 2 also has smaller leading constant than [15].

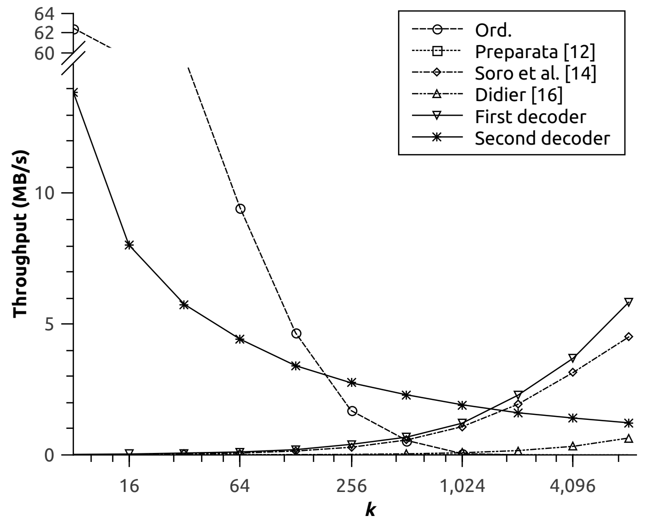
The [38] presents a (n, k) IDA for $k \geq n/2$ over Fermat field. For a reasonably large file, both encoding and decoding algorithms require $O(n\log(n-k))$ in processing k symbols. The major distinction of the [38] and the proposed algorithm is that the [38] is applicable to high code rate $k \geq n/2$ and the current proposed algorithm is applicable to low code rate $k \leq n/2$. The detailed comparisons are listed in Table III. In encoding, the [38] requires $n/(n-k)$ times of $(n-k)$-point FNT (for $n-k$ being the power of two), and Algorithm 1 requires n/k times of k-point FNT (for k being the power of two). In decoding, the [38] follows the Forney algorithm, which computes the error values of BCH codes at known error locations. In contrast, to the best of our knowledge, Algorithm 2 does not follow any existing decoding formulation, and Algorithm 3 is based on the framework of fast Lagrange interpolation. As shown in Table III, the complexities of three decoding algorithms are different. It is noted that we cannot alternate the use of the [38] and the proposed algorithm. Precisely, the codeword generated by the encoding of [38] cannot be directly decoded by Algorithms 2 and 3, and vice versa.

To verify the above complexity analysis, Figure 1 shows the simulations of [12], [14], [16] and ours for $n = 2^{14}$ and $k = 2^3, 2^4 \ldots 2^{13}$. We implement the algorithms of the ordinary method (Matrix multiplications), Preparata [12], and ours by C programs. The simulation also tests the code implemented by Didier [16]. For [14], this simulation tests the systematic FNT case implemented by Soro and Lacan in the experiment. Those codes are executed on Intel i7-950 3.06 GHz, Windows 7 with GCC compiler. Figure 1 shows the time spent of those algorithms. The main procedure of each code is repeated in $ITER$ times. The data throughput is defined as
Encoding throughput = $ITER \times$Size of n codeword symbols/Encoding time,
Decoding throughput = $ITER \times$Size of k data symbols/Decoding time,
where $ITER = 10^4$ in most cases (a few cases adopt smaller values to shorten the execution time). For decoding, the time of the initialization is ignored, because the main procedure dominates the time complexity when the file size is reasonably large. It is noted that the real simulations are controlled by several factors, such as optimization techniques of compilers, memory management techniques, task management system, and hardware properties. Therefore, the simulations may be slightly different from the analyses.

For encoding, the [12] performs about 20 (MB/s) for any k. When k is large, the throughput of [12] and proposed encoder are very close, and for the extreme case $k = 8$, the throughput of proposed encoder is about twice of [12]. It is noted that the [12] is non-systematic, and the proposed code is systematic, so the two algorithms are distinct in aspects of systematic or non-systematic codes. For [14], the throughput of [14] is about 60%



(a)



(b)

Fig. 1: The simulations for $n = 2^{14}$ and $k = 2^3, 2^4 \dots 2^{13}$. (a). Encoding case. (b). Decoding case.

of proposed encoder on average. In particular, the ordinary algorithm provides better throughput when $k \leq 16$ due to its simple architecture, but the degradation is very fast due to the quadratic complexity. As shown in Figure 1(a), the encoding algorithm has significant improvement in $k \in [32, 512]$. For decoding, the [14] provides about 68% throughput of the first decoder on average. In particular, the intersection point of two proposed decoders is between $k = 1024$ and 2048. When $k \leq 128$, the matrix multiplication method performs better due to its simple computational architecture. As shown in Figure 1(b), the encoding algorithm has significant improvement in $k \in [200, 1000]$.

VIII. Discussions

A. The relaxation of the constraints on parameters

The proposed algorithm requires that k is the power of two, and n is the multiple of k. Suppose the given parameters (n', k') do not precisely satisfy the above two conditions. We find two integers l and m such that

$$2^{l-1} < k' \leq 2^l; (m-1)2^l < n' - k' \leq m2^l. \tag{58}$$

The new parameters $(n = (m+1)2^l, k = 2^l)$ is adopted in the coding algorithm. In (n, k) encoding (Algorithm 1), the k input symbols are the k' message symbols with concatenating $k - k'$ zeros. The encoder then generates $n - k$ parity symbols. The n'-symbol codeword is the concatenation of k' message symbols with the $n' - k'$ parity symbols, which are a portion of the generated $n - k$ parity symbols. In (n, k) decoding, the decoder side receives k' symbols, and the $k - k'$ zeros are placed at the corresponding locations in the message part. Thus, the received codeword contains k known values, so Algorithm 2 and Algorithm 3 can be applied successfully. By such substitution method, the (n', k') algorithm requires the same computational cost as the (n, k) algorithm does. In the following, we prove the (n', k') encoder and the (n', k') first decoder require complexity $O(n' \log k')$, and the (n', k') second decoder requires complexity $O(k' \log^2 k')$.

By (58), the upper bounds of n and k can be termed as $n'$and k' given by

$$k < 2k'; n < n' + 3k'. \tag{59}$$

By the above inequalities, the big-O $O(n \log k)$ can be rewritten as

$$O(n \log k) = O((n' + 3k') \log(2k')) = O(n' \log k').$$

Furthermore, the big-O $O(k \log^2 k)$ can also be rewritten as

$$O(k \log^2 k) = O(2k' \log^2(2k')) = O(k' \log^2 k').$$

Thus, the substitution strategy does not increase the big-O bounds for arbitrary (n', k').

B. The criterion of selecting the two decoding algorithms

Given the (n, k), this subsection discusses the criterion of choosing Algorithms 2 or 3 as the decoding algorithm. The decision criterion is at the point that the complexities of both algorithms are asymptotically identical. The formulation is

$$\begin{aligned} &1.5(n+k)\log_2 k = 1.5k \log_2^2 k \\ &\Rightarrow n = k \log_2 k - k \\ &\Rightarrow n \approx k \log_2 k. \end{aligned} \tag{60}$$

Therefore, Algorithm 3 is chosen for $n \geq k \log_2 k$; or else Algorithm 2 is suggested. For $n = 16384$ as an example, the(60) gives the intersection point $k = 1546$. Figure (1) verifies this value, as the intersection point of Algorithms 2 and 3 is between $k = 1024$ and 2048.

C. The partial FFT on Algorithm 2

As defined in (24), the vector G contains $n - k$ erasures (zeros). Then the G is divided into $2^r/k$ sub-vectors G_j for $j = 0 \dots 2^r/k - 1$ in the first step of Algorithm 2. Thus, we can consider those sub-vectors as sparse, and the partial FFTs [26], [27], [28] can be applied on the second step of Algorithm 2 to improve the performance. The partial FFT is a technique of pruning the unnecessary computation nodes in FFT butterfly. In Algorithm 2, the unnecessary computations are caused by the erasures in the input G_j. Since the FNT in second step (33) dominates the complexity of Algorithm 2, the performance can be improved by utilizing the partial FFT. However, the performance of partial FFT depends on the positions of erasures. In general, when the positions of erasures are centralized in several sub-vectors G_j, the partial FFT performs much better than conventional FFT. In contrast, when the erasures are uniformly distributed among those vectors G_j, the improvement is smaller. This observation gives a potential direction to further improve the decoding performance.

D. The strategy of storing the shadow elements in binary data format

In (2^r+1)-ary numeral system, the length of each shadow F_i is one-k^{th} length of the file F. One difficulty in the (2^r+1)-ary numeral system is that the 2^r+1 possible values are unable to be coded in a r-bit unit. Soro and Lacan [14] introduce a solution by recording the positions of the extra symbol in the shadow header. In the following, we introduce another method based on the truncated binary encoding. Given a shadow symbol b_i whose value is between 0 and 2^r, for the value in the range $0 \leq b_i \leq 2^r-2$, the symbol b_i is encoded as a r-bit binary digit. For the overflowing cases $b_i \in \{2^r-1, 2^r\}$, the b_i is encoded as a $r+1$-bit binary digit concatenating r ones with an extra bit 0 or 1, respectively corresponding to $b_i = N-1$ or N. In the extraction procedure, the extractor sequentially takes r-bit digit from the shadow F_i as the value b_i. When the r bits are all ones, one extra bit is taken to identify $b_i = N-1$ or N. Due to the extra bits stored in the shadow, the length of each shadow F_i is slightly larger than one-k^{th} length of the file F. For b_i uniformly distributed, the probability of generating the extra bit is $2/(2^r+1)$, which is relatively small as compared to the reasonably large 2^r. For example, the probability of $r=16$ is $2/65537 \approx 3.05\times10^{-5}$.

E. The proposed (n,k)IDA over $GF(t\times 2^r+1)$

Since the known largest Fermat field is $GF(2^{16}+1)$, the proposed (n,k) IDA has the constraints $n \leq 2^{16}$. For the rare application requiring $n > 2^{16}$, a larger field is needed. The Proth prime is defined as $t2^r+1$ for odd t, positive integer r, and $2^r > t$. It is noted that the Fermat prime is a specific case of Proth prime for $t=1$. To resolve the limit $n \leq 2^{16}$ caused by the Fermat field [29], the algorithm can be applied over Proth field $GF(t2^r+1)$. There are many known Proth primes, and in practice, the Proth field $GF(3\times2^{30}+1)$ is suitable for 64-bit CPU processors. The implementation of addition over $GF(3\times2^{30}+1)$ is straightforward, and the multiplication over $GF(3\times2^{30}+1)$ can be done by an unsigned multiplication and a modulo operation on a 64-bit CPU. When the coding algorithms are applied over Proth field, there are several issues should be considered.

1) The range of (k,n) and the implementation of FFT over Proth field: The implementation of 2^r-point FFT needs the primitive 2^r-th root of unity over Proth field. Given the primitive element α of $GF(t\times2^r+1)$, the primitive 2^r-th root of unity can be calculated via $w_{2^r} = \alpha^t$. As the range of k depends on the size of FFT, we have $k \leq 2^r$. For the range of n, the range depends on the size of field, so $n \leq t\times 2^r$. Therefore, for $t=3$ and $r=30$, the range of (k,n) is $k \leq 2^{30} \approx 1.07\times10^9$ and $n \leq 3\times2^{30} \approx 3.22\times10^9$.

2) The strategy of storing the shadow elements in binary data format: The coding algorithms need a function to convert a sequence from two-ary to $(t2^r+1)$-ary numeral representation, and vice versa. The conversion can be implemented by truncated binary encoding, which can be treated as the Huffman coding for uniform probability distribution. The set of input symbols of the Huffman coding are drawn from $GF(t2^r+1)$, and the probability of each symbol is set to $1/(t2^r+1)$. Thus, the constructed Huffman tree forms a complete binary tree with $t2^r+1$ leaves. More precisely, the Huffman tree contains $t2^r+1-b$ leaves at depth $\lfloor\log_2(t2^r+1)\rfloor$, and $2b$ leaves at depth $\lceil\log_2(t2^r+1)\rceil$, where $b=(t-2^{\lfloor\log_2 t\rfloor})2^r+1$. The conversion from $(t2^r+1)$-ary to two-ary numeral representation is as the conventional Huffman encoding. The variable-length code table is constructed via traversing the Huffman tree for all leaves. Thus, the variable-length code table contains $t2^r+1$ slots, and each slot takes $\lceil\log_2(t2^r+1)\rceil$ bits of memory. By the constructed code table, we scan and encode the base-$t2^r+1$ sequence using the legal codewords. Similarly, the conversion from two-ary to $(t2^r+1)$-ary numeral representation is equivalent to the conventional Huffman decoding. We traverse the Huffman tree node by node according to each bit read from the input binary sequence. When a leaf is reached, the corresponding symbol is generated. By construction, to reach a leaf needs traversing $\lfloor\log_2(t2^r+1)\rfloor$ or $\lceil\log_2(t2^r+1)\rceil$ nodes in Huffman tree.

IX. Conclusions

This paper proposes the coding algorithms for (n,k) IDA. The code system is under (n,k) systematic Reed-Solomon codes over Fermat field. By assigning the evaluation points to specific values, the k-point FNT is applied to improve the performance. The proposed encoding algorithm takes $O(n\log k)$ operations to code k message symbols. In decoding, if $n \leq k\log_2 k$, the first decoder with $O(n\log k)$ is chosen; or else the second decoder with $O(k\log^2 k)$ is suggested. Both decoders contain two parts, the initialization and the main procedure. For the reasonably large source file, the complexity in the initialization is insignificant, so we focus on the investigation of the complexity in the main procedure. By our survey of (n,k) erasure codes, the proposed encoding algorithm takes $O(n\log k)$, as opposed to the $O(n\log n)$ in existing work; the proposed decoding algorithm takes $O(n\log k)$ or $O(k\log^2 k)$, as opposed to the $O(n\log n)$ or $O(k\log^2 k)$ in existing work.

Appendix A
Algorithm of computing the coefficients of $L(x)$

Given the evaluation points of k received symbols $\{\tilde{I}_i|\tilde{I}_i \in I, i=0\ldots k-1\}$, a polynomial can be expressed as

$$L(x) = \prod_{i=0}^{k-1}(x-\tilde{I}_i) = \sum_{i=0}^{k} L_i x^i, \tag{61}$$

where $L_k = 1$. The formal derivative of $L(x)$ is defined as

$$L'(x) = \sum_{i=0}^{k-1} \prod_{j=0, j\neq i}^{k-1} (x - \tilde{I}_j) = \sum_{i=1}^{k} iL_i x^{i-1}. \quad (62)$$

The $L(x)$ is similar to the error locator polynomial used in BCH code decoding, except, in the algorithm design, the $L(x)$ records the evaluation points of received symbols, rather than erasures (or errors) in the error locator polynomial.

The computation of $L(x)$ follows the divide-and-conquer strategy. Initially, each term $(x - \tilde{I}_i)$ in (62) is denoted as a polynomial given by

$$L_i^{[0]}(x) = x - \tilde{I}_i, \forall i = 0 \ldots k-1. \quad (63)$$

Then those polynomials are pairwise multiplied, resulting in $k/2$ polynomials

$$L_i^{[1]}(x) = L_{2i}^{[0]}(x) L_{2i+1}^{[0]}(x), \forall i = 0 \ldots k/2 - 1. \quad (64)$$

Then the process recursively computes $L_i^{[2]}(x) = L_{2i}^{[1]}(x) L_{2i+1}^{[1]}(x), \forall i = 0 \ldots k/4 - 1$. By following such steps, the process recursively computes the results from the previous results in each stage. The recursive function is expressed as

$$L_i^{[j]}(x) = L_{2i}^{[j]}(x) L_{2i+1}^{[j]}(x) \forall j = 1 \ldots \log_2 k, i = 0 \ldots k/2^j - 1. \quad (65)$$

For $j = \log_2 k$, the resultant $L_i^{[\log_2 k]}(x)$ is the polynomial $L(x)$. The convolution expression of (65) is expressed as

$$L_i^{[j]} = L_{2i}^{[j-1]} * L_{2i+1}^{[j-1]}, \forall j = 1 \ldots log_2 k, i = 0 \ldots k/2^j - 1. \quad (66)$$

By the fast convolution, the overall complexity of the recursive structure requires $\mathrm{O}(k \log^2 k)$ operations. It is noted that the [16] introduces another efficient algorithm to calculate the coefficients of $L(x)$ with complexity $\mathrm{O}(n \log n)$.

APPENDIX B
SUPPLEMENT OF FIRST DECODING ALGORITHM

A. *Another algorithm of computing β and $\hat{\beta}$*

For the β defined in (26), another derivation by L'Hopital's rule is given by

$$\bar{L}(I_j) = \lim_{x \to I_j} \frac{x^{2^r} - 1}{L(x)} = \lim_{x \to I_j} \frac{(x^{2^r} - 1)'}{L'(x)} = \frac{2^r}{I_j L'(I_j)}. \quad (67)$$

The (26) can be reformulated as

$$\beta = \{2^r (I_j L'(I_j))^{-1} | j \in \tilde{l}\}. \quad (68)$$

For the $\hat{\beta}$ defined in (29), the derivative of $\bar{L}(x)$ is given by

$$\bar{L}'(x) = \frac{2^r x^{2^r - 1} L(x) - (x^{2^r} - 1) L'(x)}{L^2(x)}. \quad (69)$$

The (69) is substituted into (29) to obtain

$$\hat{\beta} = \{2^{-r} L(I_j) | j \notin \tilde{l} \text{ and } 0 \leq j \leq k-1\}. \quad (70)$$

Thus, the β and $\hat{\beta}$ can be constructed via the polynomial $L(x)$.

B. *The validity of* (34)

To test the validity (34), the $x = \alpha^p w_k^i$ is substituted into $T_j(x)$. For the case$j \neq p$, the evaluation is given by

$$T_j(\alpha^p w_k^i) = \frac{(\alpha^p w_k^i)^{2^r} - 1}{w_{2^r/k}^{-j} (\alpha^p w_k^i)^k - 1} = \frac{0}{\alpha^{(p-j)k} - 1} = 0. \quad (71)$$

For the case $j = p$, both numerator and denominator of $T_j(x)$ approach zeros, so the evaluation employs L'Hopital's rule:

$$\begin{aligned} T_p(\alpha^p w_k^i) &= \lim_{x \to \alpha^p w_k^i} \frac{(x^{2^r} - 1)'}{(w_{2^r/k}^{-p} x^k - 1)'} \\ &= \lim_{x \to \alpha^p w_k^i} \frac{2^r x^{2^r - 1}}{k w_{2^r/k}^{-p} x^{k-1}} = \frac{2^r}{k}. \end{aligned} \quad (72)$$

Based on the above results, the (34) is rewritten as

$$\begin{aligned} g(\alpha^p w_k^i) &= \tfrac{k}{2^r} \textstyle\sum_{j=0}^{2^r/k-1} T_j(\alpha^p w_k^i) g_j(\alpha^p w_k^i) \\ &= \tfrac{k}{2^r} T_p(\alpha^p w_k^i) g_p(\alpha^p w_k^i) \\ &= g_p(\alpha^p w_k^i). \end{aligned} \quad (73)$$

The equality $g_p(\alpha^p w_k^i) = g(\alpha^p w_k^i)$ responds to the definition (30), so the validity of the formulation (34) is verified.

REFERENCES

[1] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335-348, 1989.
[2] M. O. Rabin, "The information dispersal algorithm and its applications," *Springer-Verlag New York*, pp. 406-419, 1990.
[3] P. Béguin and A. Cresti, "General information dispersal algorithms," *Theoretical Computer Science*, vol. 209, no. 1-2, pp. 87-105, Dec. 1998.
[4] D. Ellard and J. Megquier, "DISP: Practical, efficient, secure and fault-tolerant distributed data storage," *ACM tran. Storage*, vol. 1, no. 1, pp. 71-94, Feb. 2005.
[5] G. L. Feng, R. H. Deng, F. Bao, and J. C. Shen, "New Efficient MDS Array Codes for RAID Part II: Rabin-Like Codes for Tolerating Multiple (greater than or equal to 4) Disk Failures," *IEEE tran. Computers*, vol. 54, n. 12, pp. 1473-1483, Dec. 2005.
[6] S. A. Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335-371, Dec. 2004.
[7] J. M. Park, E. K. P. Chong, and H. J. Siegel, "Efficient multicast stream authentication using erasure codes," *ACM tran. Information and System Security*, vol. 6, no. 2, pp. 258-285, May 2003.
[8] W. C. Siu and A. G. Constantinides, "On the computation of discrete Fourier transform using Fermat number transform," *IEE Proceedings F Communications, Radar and Signal Processing* , vol. 131, no. 1, 1984.
[9] Y. Wang and X. Zhu, "A fast algorithm for the Fourier transform over finite fields and its VLSI implementation," *IEEE tran. selected areas in communications*, vol. 6, no. 3, pp. 572-577, Apr. 1988.
[10] T. K. Truong, P. D. Chen, L. J. Wang, Y. Chang, and I. S. Reed, "Fast, prime factor, discrete Fourier transform algorithms over $GF(2^m)$ for $8 \leq m \leq 10$," *Information Sciences*, vol. 176, no. 1, pp. 1-26, Jan. 2006.
[11] T. K. Truong , P. D. Chen , L. J. Wang , Y. Chang, and I. S. Reed, "Erratum: Erratum to "Fast, prime factor, discrete Fourier transform algorithms over $GF(2^m)$ for $8 \leq m \leq 10$ [Informat. Sci. 176(1) (2006) 1-26]," *Information Sciences*, vol.177, no.3, pp.967-968, Feb. 2007.
[12] F.P. Preparata, "Holographic dispersal and recovery of information," *IEEE tran. Information theory*, vol. 35, no. 5, Sep. 1989.
[13] R.Dianat and F.Marvasti, "FFT-based fast Reed-Solomon codes with arbitrary block lengths and rates," *IEE Proceedings Communications*, vol. 152, no. 2, pp. 151-156, Apr. 2005.
[14] A. Soro and J. Lacan, "FNT-based Reed-Solomon erasure codes," *7th Annual IEEE Consumer Communications and Networking Conference*, 2010.
[15] J. Lacan and J. Fimes, "Systematic MDS erasure codes based on Vandermonde matrices," *IEEE Communications Letters*, vol. 8, no. 9, pp. 570-572, Sept. 2004.
[16] F. Didier, "Efficient erasure decoding of Reed-Solomon codes," *Computing Research Repository - CORR*, vol. abs/0901.1886, 2009.

[17] G. L. Feng, R. H. Deng, F. Bao, and J. C. Shen, ”New Efficient MDS Array Codes for RAID Part I: Reed-Solomon-Like Codes for Tolerating Three Disk Failures,” *IEEE tran. Computers*, vol. 54, no. 9, pp. 1071-1080, 2005.

[18] G. L. Feng, R. H. Deng, F. Bao, and J. C. Shen, ”New Efficient MDS Array Codes for RAID Part II: Rabin-Like Codes for Tolerating Multiple (greater than or equal to 4) Disk Failures,” *IEEE tran. Computers*, vol. 54, no. 12, pp. 1473-1483, 2005.

[19] T. K. Truong, J. H. Jeng, and T. C. Cheng, ”A new decoding algorithm for correcting both erasures and errors of Reed-Solomon codes,” *IEEE tran. Communications*, vol. 51, no. 3, pp. 381-388, Mar. 2003.

[20] T. K. Truong, P. D. Chen, L. J. Wang, and T. C. Cheng, ”Fast transform for decoding both errors and erasures of Reed-Solomon codes over $GF(2^m)$ for $8 \leq m \leq 10$,” *IEEE tran. Communications*, vol. 54, no. 2, pp. 181-186, Feb. 2006.

[21] T. C. Lin, T. K. Truong, and P. D. Chen, ”A Fast Algorithm for the Syndrome Calculation in Algebraic Decoding of Reed-Solomon Codes,” *IEEE tran. Communications*, vol. 55, no. 12, pp. 2240-2244, Dec. 2007.

[22] I. S. Reed and G. Solomon, ”Polynomial codes over certain finite field,” *Siam Journal on Applied Mathematics*, vol. 8, no. 2, pp. 300-304, 1960.

[23] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, ”Efficient erasure correcting codes,” *IEEE tran. Information Theory*, vol. 47, no. 2, pp. 569-584, Feb. 2001.

[24] N. Alon and M. Luby, ”A linear time erasure-resilient code with nearly optimal recovery,” *IEEE tran. Information Theory*, vol. 42, no. 6, pp. 1732, Aug. 2002.

[25] D. Bini and V. Y. Pan, ”Polynomial and matrix computations fundamental algorithms vol. 1,” Birkhäuser Boston, 1994.

[26] J. Markel, ”FFT pruning,” *IEEE tran. Audio and Electroacoustics*, vol. 19, no. 4, pp. 305-311, Dec. 1971.

[27] D. Skinner, ”Pruning the decimation in-time FFT algorithm,” *IEEE tran. Acoustics, Speech and Signal Processing*, vol. 24, no.2, pp. 193-194, Apr. 1976.

[28] H. V. Sorensen and C.S. Burrus, ”Efficient computation of the DFT with only a subset of input or output points,” *IEEE tran. Signal Processing*, vol. 41, no. 3, pp. 1184-1200, Mar. 1993.

[29] 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 Progress Report 42-35*, Jet Propulsion Laboratory, Pasadena, CA, July 1976, pp. 64-78.

[30] S. Lin, D.J. Costello, ”Binary field arithmetic,” in *Error control coding*, 2th ed. Englewood Cliffs, NJ: Pearson: Prentice-Hall, 2004, ch. 2, pp. 25-65.

[31] T.C. Lin, T.K. Truong, H.C. Chang, and H.P. Lee, ”A Future Simplification of Procedure for Decoding Nonsystematic Reed-Solomon Codes Using the Berlekamp-Massey Algorithm,” *IEEE trans. Communications*, vol. 59, no. 6, pp. 1555-1562, Jun. 2011.

[32] I. Gohberg and V. Olshevsky, ”Fast algorithms with preprocessing for matrix-vector multiplication problems,” *Journal of Complexity*, vol. 10, no. 4, pp. 411-427, Dec. 1994.

[33] Krawczyk, H., ”Secret Sharing Made Short” *Advances in Cryptology−CRYPTO 93 Proceedings, Lecture Notes in Computer Science Vol. 773, Springer-Verlag, D. R. Stinson, ed*, 1993, pp. 136-146.

[34] M. Mitzenmacher, ”Digital fountains: a survey and look forward,” *IEEE 2004 Information Theory Workshop*, Oct. 2004, pp. 271-276.

[35] A. Shamir, ”How to Share a Secret”, *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.

[36] W.W. Chan, T.F. Wong and J.M. Shea, ”Secret-Sharing LDPC Codes for the BPSK-Constrained Gaussian Wiretap Channel”, *IEEE trans. Information Forensics and Security*, vol. 6, no. 3, pp. 551-564, 2011.

[37] W. Luh and D. Kundur, ”Distributed Secret Sharing for Discrete Memoryless Networks,” *IEEE trans. Information Forensics and Security*, vol. 3, no. 3, pp. 1-7, 2008.

[38] S. J. Lin and W. H. Chung, ”An Efficient (n, k) Information Dispersal Algorithm for High Code Rate System over Fermat Fields,” *IEEE Communications Letters*, vol. 16, no. 12, pp. 2036-2039, 2012.

[39] Li Ping, W. K. Leung and K. Y. Wu, ”Low rate turbo-Hadamard codes,”*IEEE Transactions on Information Theory*, vol. 49, no. 12, pp. 3213-3224, 2003.

Sian-Jheng Lin was born in Taichung, Taiwan, in 1981. He received the B.S., M.S., and Ph.D. degrees in computer science from National Chiao Tung University, in 2004, 2006, and 2010, respectively. 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 postdoctoral fellow with the Research Center for Information Technology Innovation, Academia Sinica. His recent research interests include erasure coding algorithms, data hiding and modulations.

Wei-Ho Chung (M’ 11) received the B.Sc. and M.Sc. degrees in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in Electrical Engineering from the University of California, Los Angeles, in 2009. From 2002 to 2005, he was a system engineer at ChungHwa Telecommunications Company, where he worked on data networks. In 2008, he was a research intern working on CDMA systems at Qualcomm, Inc., San Diego, CA. His research interests include communications, signal processing, and networks. Dr. Chung received the Taiwan Merit Scholarship from 2005 to 2009 and the Best Paper Award in IEEE WCNC 2012, and has published over 30 refereed journal articles and over 40 refereed conference papers. Since January 2010, Dr. Chung has been a tenure-track assistant research fellow and leads the Wireless Communications Lab in Research Center for Information Technology Innovation, Academia Sinica, Taiwan.