Fast Decoding of Gabidulin Codes

Antonia Wachter, Valentin Afanassiev, Vladimir Sidorenko

Fast Decoding of Gabidulin Codes

Antonia Wachter[1], Valentin Afanassiev[2], and Vladimir Sidorenko[1]

[1] Institute of Telecommunications and Applied Information Theory,
University of Ulm, Ulm, Germany *
[2] Institute for Information Transmission Problems,
Russian Academy of Sciences, Moscow, Russia
antonia.wachter@uni-ulm.de, afanv@iitp.ru, vladimir.sidorenko@uni-ulm.de

Abstract. A decoding algorithm for Gabidulin codes (defined over $\mathbb{F}_{q^m}$) is shown that directly provides the evaluation polynomial using an equivalent of the Euclidean Algorithm. To obtain low complexity, a fast symbolic product and a fast symbolic division are presented. The achieved complexity of the whole decoding algorithm for Gabidulin codes is $\mathcal{O}(m^3 \log m)$ operations over the *ground* field $\mathbb{F}_q$.

1 Introduction

Gabidulin codes [1] are the rank metric analogues of *Reed–Solomon* (RS) codes. In [1], Gabidulin presented decoding based on a *linearized* equivalent of the *Extended Euclidean Algorithm* (LEEA). In [8], [9], [10], a generalization of the *Berlekamp-Massey Algorithm* (BMA) was given. The BMA and LEEA are used in the decoding process for solving a *key equation*. As a second step the so–called *Gabidulin Algorithm* (GA) has to be carried out, which can be seen as the rank metric analogue to calculating the error values. Loidreau formulated a *Welch–Berlekamp–like Algorithm* (WBA) that uses interpolation techniques [6] and directly gives the linearized evaluation polynomial of the code. Hence, for the WBA no GA is necessary. The complexity of all these decoding methods is quadratic with the length of the code.

In [11], a method for fast encoding and decoding of Gabidulin codes was presented, where the complexity of the two most demanding steps was reduced. Considering their complexity reduction, solving the key equation and the GA are the most complex steps.

In this contribution, we present a method to solve an alternative (transformed) key equation that directly outputs the evaluation polynomial using the LEEA. We use the accelerated LEEA from [12], which has complexity $\mathcal{O}(\mathcal{M}(m) \log m)$, where the Gabidulin code is defined over $\mathbb{F}_{q^m}$ and $\mathcal{M}(m)$ denotes the complexity of the *symbolic product*. To achieve a low overall complexity, we present a fast symbolic product and a fast symbolic division with complexity

* This work was supported by the German Research Council "Deutsche Forschungsgemeinschaft" (DFG) under Grant No. Bo867/21-1. V. Sidorenko is on leave from IITP, Russian Academy of Sciences, Moscow, Russia.

$\mathcal{O}(m^3)$ operations over the *ground* field $\mathbb{F}_q$ using low–complexity normal bases. Since our algorithm does not require the GA afterwards, the complexity of the *whole* decoding process is accelerated to $\mathcal{O}(m^3 \log m)$ operations over $\mathbb{F}_q$.

This paper is organized as follows: In Section 2, we give some preliminaries. Section 3 states the problem and in Section 4, we present the fast symbolic product and the fast symbolic division. Section 5 provides the decoding algorithm and Section 6 concludes this paper.

2 Preliminaries

2.1 Linearized Polynomials

Gabidulin codes are defined by means of *linearized polynomials* which were introduced by Ore [7]. Let q be a power of a prime and let us denote the Frobenius q-power by $x^{[i]} = x^{q^i}$ where i is an integer. A linearized polynomial over the field $\mathbb{F}_{q^m}$ is a polynomial of the form $f(x) = \sum_{i=0}^{d_f} f_i x^{[i]}$, with $f_i \in \mathbb{F}_{q^m}$. If $f_{d_f} \neq 0$, we call $\deg_q f(x) \stackrel{\text{def}}{=} d_f$ the *q-degree* of $f(x)$. An important property of linearized polynomials $\forall\ \alpha_1, \alpha_2 \in \mathbb{F}_q$ and $\forall\ a, b \in \mathbb{F}_{q^m}$ is $f(\alpha_1 a + \alpha_2 b) = \alpha_1 f(a) + \alpha_2 f(b)$.

The *symbolic product* of two linearized polynomials $f(x)$ and $g(x)$ is:

$$f(x) \otimes g(x) = f(g(x)). \tag{1}$$

Denote $\deg_q f(x) = d_f$ and $\deg_q g(x) = d_g$, then $\deg_q(f(x) \otimes g(x)) = d_f + d_g$. The symbolic product is associative and distributive, but in general non-commutative. The (usual) addition and the symbolic product convert the set of linearized polynomials into a non-commutative ring with identity element $x^{[0]} = x$. Throughout this paper, all polynomials are linearized polynomials.

We call $b(x)$ a *right symbolic divisor* of $a(x)$, if $a(x) = q(x) \otimes b(x)$ for some $q(x)$. We denote the algorithmic calculation of the *right symbolic division* by $q(x), rem(x) \leftarrow \texttt{RDiv}\ (a(x), b(x))$, where $rem(x)$ with $\deg_q rem(x) < \deg_q a(x)$ denotes a possible remainder. Equivalently, $b(x)$ is a *left symbolic divisor* of $a(x)$, if $a(x) = b(x) \otimes q(x)$ and we denote the *left symbolic division* by $q(x), rem(x) \leftarrow \texttt{LDiv}\ (a(x), b(x))$.

2.2 The Linearized Euclidean Algorithm

Let $r_{-1}(x) = a(x)$ and $r_0(x) = b(x)$ be two linearized polynomials with $\deg_q a(x) \geq \deg_q b(x)$. The LEEA with a *stopping condition* $d_S > 0$ is given in Algorithm 1, where for the remainder $\deg_q r_i(x) < \deg_q r_{i-1}(x)$ holds. If $d_S = 1$, the last non–zero remainder $r_j(x)$ is the so–called *right symbolic greatest common divisor* $\text{rsgcd}(a(x), b(x))$. With the additional polynomials $u_i(x), v_i(x)$, we can rewrite each remainder:

$$r_i(x) = v_i(x) \otimes a(x) + u_i(x) \otimes b(x), \qquad \forall i. \tag{2}$$

Algorithm 1: LEEA (Linearized Extended Euclidean Algorithm)

Input: $a(x)$, $b(x)$ with $\deg_q a(x) \geq \deg_q b(x)$, Stopping Degree d_S

Initialize: $r_{-1}(x) \leftarrow a(x)$, $r_0(x) \leftarrow b(x)$, $i \leftarrow 1$, $u_{-1}(x) = 0$, $u_0(x) = x^{[0]}$, $v_{-1}(x) = x^{[0]}, v_0(x) = 0$

while $\deg_q r_{i-1}(x) \geq d_S$ **do**
- $q_i(x), r_i(x) \leftarrow \texttt{RDiv}\ (r_{i-1}(x), r_{i-2}(x))$
- $u_i(x) \leftarrow u_{i-2}(x) - q_i(x) \otimes u_{i-1}(x)$
- $v_i(x) \leftarrow v_{i-2}(x) - q_i(x) \otimes v_{i-1}(x)$
- $i \leftarrow i + 1$

Output: $r_{i-1}(x)$, $u_{i-1}(x)$, $v_{i-1}(x)$

2.3 Normal Bases

A basis $\mathcal{B} = \{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ is a *normal basis* if $\beta_i = \beta^{[i]}$ for all i and $\beta \in \mathbb{F}_{q^m}$ is called a *normal element*. There is a normal basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ for any prime power q and any positive integer m [4]. A basis $\widetilde{\mathcal{B}} = \{\widetilde{\beta}_0, \widetilde{\beta}_1, \ldots, \widetilde{\beta}_{m-1}\}$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ is called a *dual basis* to $\mathcal{B}$ if $\mathrm{Tr}(\widetilde{\beta}_i \beta_j)$ is equal to 1 for $i = j$ and 0 in all other cases. The dual basis of a normal basis is again a normal basis [4]. If $\mathcal{B} = \widetilde{\mathcal{B}}$, it is called a *self–dual normal basis*.

In a normal basis $\mathcal{B}$, the product of $a, b \in \mathbb{F}_{q^m}$ is usually done as follows. We represent $a = \sum_{i=0}^{m-1} a^{(i)} \beta_i$, $b = \sum_{i=0}^{m-1} b^{(i)} \beta_i$ with all $a^{(i)}, b^{(i)} \in \mathbb{F}_q$ and $\underline{a} = (a^{(0)} \ \ldots \ a^{(m-1)})$, $\underline{b} = (b^{(0)} \ \ldots \ b^{(m-1)})$ denote the vector representations. A *multiplication table* $\mathbf{T} \in \mathbb{F}_q^{m \times m}$ is defined such that [4]:

$$\beta_0 \cdot (\, \beta_0 \ \beta_1 \ \ldots \ \beta_{m-1} \,)^T = \mathbf{T} \cdot (\, \beta_0 \ \beta_1 \ \ldots \ \beta_{m-1} \,)^T.$$

This is used to calculate the product $\underline{a} \cdot \underline{b} = \sum_{i=0}^{m-1} b^{(i)} (\underline{a}^{\leftarrow i} \mathbf{T})^{\rightarrow i}$, where the arrows denote a cyclic shift of the vector by i positions to the right/left. The number of non–zero entries of $\mathbf{T}$ is called complexity $C(\mathbf{T})$ and is lower bounded by $C(\mathbf{T}) \geq 2m - 1$. Low–complexity normal bases (i.e., $C(\mathbf{T}) \approx 2m$) exist in many cases, e.g. for $q = 2^s$ if $\gcd(m, s) = 1$ and $8 \nmid m$. For $q = 2^s$ and odd m, all these low–complexity normal bases are self–dual (see also [4]).

For a normal basis $\mathcal{B}$, the q-transform of a linearized polynomial $f(x)$ is:

Definition 1 (q-Transform [11]). *The q-transform of a vector $\mathbf{f} \in \mathbb{F}_{q^m}^m$ (or a linearized polynomial $f(x)$ with $\deg_q f(x) < m$) with respect to a normal element β is the vector $(F_0 \ F_1 \ \ldots \ F_{m-1}) \in \mathbb{F}_{q^m}^m$ (or $F(x) = \sum_{j=0}^{m-1} F_j x^{[j]}$), given by*

$$F_j = f(\beta^{[j]}) = \sum_{i=0}^{m-1} f_i \beta^{[i+j]}, \qquad j = 0, \ldots, m-1. \tag{3}$$

Using the multiplication table, $F_j = \sum_{k=0}^{m-1} \sum_{i=0}^{m-1} f_k^{(i)} (\mathbf{T}_{j+k-i})^{\rightarrow i}$, where $\underline{f}_k = (f_k^{(0)} \ \ldots \ f_k^{(m-1)})$ is the vector representation of f_k over $\mathbb{F}_q$ and $\mathbf{T}_{j+k-i}$ is the $(j+k-i)$-th row of $\mathbf{T}$. Here, the index ℓ of $\mathbf{T}_\ell$ is calculated mod m.

Theorem 1 (Inverse q-Transform [11]). *The inverse q-transform of a vector $(F_0\ F_1\ \ldots\ F_{m-1}) \in \mathbb{F}_{q^m}^m$ (or a linearized polynomial $F(x)$) with resprect to β is given by $f_i = F(\widetilde{\beta}^{[j]}) = \sum_{j=0}^{m-1} F_j \widetilde{\beta}^{[j+i]}, j = 0, \ldots, m-1$, where $\widetilde{\mathcal{B}} = \{\widetilde{\beta}_0, \ldots, \widetilde{\beta}_{m-1}\}$, is dual to $\mathcal{B}$.*

2.4 Complexity of Elementary Operations

For simplicity, throughout this paper, let us consider only self–dual normal bases, i.e., $\widetilde{\beta}_i = \beta_i = \beta^{[i]}$. In a normal basis representation, q-exponentiations are only cyclic shifts and hence the complexity of the Frobenius powers is negligible. The product $a \cdot b$ of any two elements $a, b \in \mathbb{F}_{q^m}$ can be calculated by the multiplication table $\mathbf{T}$ with complexity $\mathcal{O}(m^2)$ operations over $\mathbb{F}_q$. The calculation of the (inverse) q-transform (3) requires at most $(m-1)mC(\mathbf{T})$ additions over $\mathbb{F}_q$, i.e., in the order of $\mathcal{O}(m^3)$ operations over $\mathbb{F}_q$ [11].

Let $\mathcal{M}(n)$ denote the complexity of calculating the symbolic product (1) where $n = \max\{d_f, d_g\}$. The complexity of the (right/left) symbolic division will be denoted by $\mathcal{D}(n)$. With standard implementation, the complexity of both is in the order of $\mathcal{O}(n^2)$ operations over $\mathbb{F}_{q^m}$ [3]. In Section 4, we give fast algorithms for calculating the symbolic product and division.

In [12], a *fast* LEEA was presented which achieves complexity $\mathcal{O}(\mathcal{M}(n) \log n)$ if $n \geq \deg_q a(x) \geq \deg_q b(x)$.

2.5 Gabidulin Codes

Definition 2 (Gabidulin Code [1]). *A linear $\mathcal{G}(n,k)$ Gabidulin code over $\mathbb{F}_{q^m}$ for $n \leq m$ is the set of all codewords, which are the evaluation of a q-degree restricted linearized polynomial:*

$$\mathcal{G}(n,k) \stackrel{\text{def}}{=} \{\mathbf{c} = (f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_{n-1}) | \deg_q f(x) < k)\},$$

where the fixed elements $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{F}_{q^m}$ are linearly independent over $\mathbb{F}_q$.

If we use the basis elements $\alpha_i = \beta^{[i]}$ as evaluation points (where β is a normal element), the codeword $c(x)$ is the inverse q-transform of the corresponding $f(x)$.

Given a basis $\mathcal{B}$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, there exists a one–to–one mapping for each vector $\mathbf{x} \in \mathbb{F}_{q^m}^n$ on a matrix $\mathbf{X} \in \mathbb{F}_q^{m \times n}$. Let the *rank norm* $\text{rank}_q(\mathbf{x})$ be the rank of $\mathbf{X}$ over $\mathbb{F}_q$. The *minimum rank distance* d of a code $\mathcal{G}$ is defined by $d = \min\{\text{rank}_q(\mathbf{c}) \mid \mathbf{c} \in \mathcal{G}, \mathbf{c} \neq \mathbf{0}\}$. Gabidulin codes are *Maximum Rank Distance* (MRD) codes, i.e., $d = n - k + 1$.

Let $\mathbf{c}$ be the transmitted codeword that is corrupted by an additive error vector $\mathbf{e} \in \mathbb{F}_{q^m}^n$ of $\text{rank}_q(\mathbf{e}) = t$. The received vector $\mathbf{r} \in \mathbb{F}_{q^m}^n$ is $\mathbf{r} = \mathbf{c} + \mathbf{e}$. We can use a $t \times n$ matrix $\mathbf{Y} \in \mathbb{F}_q^{t \times n}$ of rank t to decompose the error:

$$\mathbf{e} = \mathbf{x} \cdot \mathbf{Y} = (x_1\ x_2\ \ldots\ x_t) \cdot \mathbf{Y}, \tag{4}$$

with $x_1, x_2, \ldots, x_t \in \mathbb{F}_{q^m}$ are linearly independent over $\mathbb{F}_q$. Note, that this decomposition into $\mathbf{x}$ and $\mathbf{Y}$ is not unique.

Let $e(x)$, $r(x)$ denote the error and received word as linearized polynomials. Let $E(x)$, $R(x)$ denote the q-transform of $e(x)$, $r(x)$, where $R(x) = f(x) + E(x)$.

Classical decoding of Gabidulin codes mainly consists of two steps: First, a key equation has to be solved to determine $\mathbf{x}$ (4). Second, the matrix $\mathbf{Y}$ (4) has to be calculated to reconstruct the error. However, using Loidreau's WBA or our new approach, we directly obtain the evaluation polynomial $f(x)$.

Solving the key equation is equivalent to finding an *error span polynomial* $\Lambda(x)$ that contains all linear combinations of $x_1, x_2, \ldots, x_t$ (4) as roots. We use an alternative key equation [11] for our algorithm. Note that there should be a similar relation as for Reed–Solomon codes between (5) and Loidreau's WBA.

Theorem 2 (Transformed Key Equation [11]). *Let $E(x)$ be the q-transform (3) of the error word $e(x)$, then the linearized error span polynomial $\Lambda(x)$ satisfies*

$$\Lambda(x) \otimes E(x) \equiv 0 \mod (x^{[m]} - x), \tag{5}$$

where $\deg_q \Lambda(x) = t \leq \lfloor (n-k)/2 \rfloor$.

3 Problem Formulation

To obtain a low–complexity decoding algorithm, we need a fast symbolic product (1) and a fast symbolic division.

Problem 1 (Fast Symbolic Product). Let two linearized polynomials $f(x)$, $g(x)$ over $\mathbb{F}_{q^m}$ with q-degrees d_f, d_g be given. Find with subquadratic complexity a linearized polynomial $h(x)$, such that

$$h(x) = \sum_{i=0}^{d_f+d_g} h_i x^{[i]} = f(x) \otimes g(x) \mod (x^{[m]} - x) = f(g(x)) \mod (x^{[m]} - x).$$

Problem 2 (Fast Symbolic Division). Let two linearized polynomials $a(x)$, $b(x)$ over $\mathbb{F}_{q^m}$ be given. Find with subquadratic complexity $r(x)$ and $q(x)$, such that for the *right* symbolic division

$$a(x) = q(x) \otimes b(x) + r(x), \quad \text{where } \deg_q r(x) < \deg_q b(x),$$

or for the *left* symbolic division

$$a(x) = b(x) \otimes q(x) + r(x), \quad \text{where } \deg_q r(x) < \deg_q b(x).$$

The problem of solving the transformed key equation is as follows.

Problem 3 (Solving Transformed Key Equation). Given $R(x)$ as the q-transform (3) of the received word $r(x)$ with rank distance $\leq \lfloor (n-k)/2 \rfloor$ from a codeword from $\mathcal{G}(n,k)$, find the linearized error span polynomial $\Lambda(x)$ and the linearized evaluation polynomial $f(x)$ such that

$$\Lambda(x) \otimes E(x) = \Lambda(x) \otimes (R(x) - f(x)) \equiv 0 \mod (x^{[m]} - x), \tag{6}$$

where $\deg_q \Lambda(x) = t \leq \lfloor (n-k)/2 \rfloor$.

4 Fast Symbolic Product and Division

In order to calculate the symbolic product (1) and the symbolic division efficiently, we give two approaches. The first uses similar transform–based ideas as in [2] and the second uses directly the normal basis representation for the multiplication of two elements. By means of low–complexity normal bases, we achieve the smallest known complexity for solving Problems 1 and 2.

4.1 Transform–Based Fast Symbolic Product

Algorithm 2 calculates the symbolic product $h(x) = f(g(x)) \mod (x^{[m]} - x)$ in three main steps. It is based on the fact that the coefficients of $H(x)$ can be calculated by $H_i = h(\beta^{[i]}) = f(g(\beta^{[i]}))$, where $H(x)$ is q-transform of $h(x)$ (3).

Algorithm 2: Fast Symbolic Product

Input: Linearized polynomials $f(x)$, $g(x)$

1 Calculate the q-transform of $g(x)$: $G_i = g(\beta^{[i]})$ for $i = 0, \ldots, m-1$ (3)
2 Calculate $H_i = f(G_i)$ for $i = 0, \ldots, m-1$, $H(x) = \sum_{i=0}^{m-1} H_i x^{[i]}$
3 Calculate inverse q-transform of $H(x)$: $h_i = H(\beta^{[i]})$ for $i = 0, \ldots, m-1$

Output: Symbolic product $h(x) = \sum_{i=0}^{m-1} h_i x^{[i]}$

Steps 1 and 3 are (inverse) q-transforms, which require complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$ using low–complexity normal bases (see Section 2.3). Step 2 corresponds to the following general problem of multipoint–evaluation of a linearized polynomial.

Problem 4 (Multipoint Evaluation of a Linearized Polynomial). Given a linearized polynomial $f(x)$ over $\mathbb{F}_{q^m}$, find the evaluation values $f(G_0), f(G_1), \ldots, f(G_{m-1})$, where $G_0, \ldots, G_{m-1}$ are arbitrary elements from $\mathbb{F}_{q^m}$.

This problem can be solved as follows. We represent $G_i = \sum_{j=0}^{m-1} \gamma_j^{(i)} \cdot \beta^{[j]}$, with $\gamma_j^{(i)} \in \mathbb{F}_q$ for all $0 \le i, j \le m-1$. Since $\alpha^{[k]} = \alpha$ for any $\alpha \in \mathbb{F}_q$ and all k:

$$f(G_i) = f\Big(\sum_{j=0}^{m-1} \gamma_j^{(i)} \cdot \beta^{[j]} \Big) = \sum_{j=0}^{m-1} \gamma_j^{(i)} \cdot f\left(\beta^{[j]}\right).$$

All m evaluation values can be calculated by

$$\begin{aligned} &\big(f(G_0) \;\; f(G_1) \;\; \ldots \;\; f(G_{m-1}) \big) = \\ &\big(f(\beta^{[0]}) \;\; f(\beta^{[1]}) \;\; \ldots \;\; f(\beta^{[m-1]}) \big) \cdot \begin{pmatrix} \gamma_0^{(0)} & \gamma_0^{(1)} & \cdots & \gamma_0^{(m-1)} \\ \gamma_1^{(0)} & \gamma_1^{(1)} & \cdots & \gamma_1^{(m-1)} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma_{m-1}^{(0)} & \gamma_{m-1}^{(1)} & \cdots & \gamma_{m-1}^{(m-1)} \end{pmatrix}. \end{aligned} \tag{7}$$

Also, we represent $f(\beta^{[i]}) = \sum_{j=0}^{m-1} f_j^{(i)} \cdot \beta^{[j]}$, with $f_j^{(i)} \in \mathbb{F}_q$, for all $0 \le i, j \le m-1$.

Hence, we obtain:

$$\left(f(G_0)\ f(G_1)\ \ldots\ f(G_{m-1}) \right) \stackrel{\text{def}}{=} \boldsymbol{\beta} \cdot \mathbf{F} \cdot \mathbf{G}$$

$$= \left(\beta^{[0]}\ \beta^{[1]}\ \ldots\ \beta^{[m-1]} \right) \cdot \begin{pmatrix} f_0^{(0)} & f_0^{(1)} & \cdots & f_0^{(m-1)} \\ f_1^{(0)} & f_1^{(1)} & \cdots & f_1^{(m-1)} \\ \cdots & \cdots & \cdots & \cdots \\ f_{m-1}^{(0)} & f_{m-1}^{(1)} & \cdots & f_{m-1}^{(m-1)} \end{pmatrix} \cdot \begin{pmatrix} \gamma_0^{(0)} & \cdots & \gamma_0^{(m-1)} \\ \gamma_1^{(0)} & \cdots & \gamma_1^{(m-1)} \\ \cdots & \cdots & \cdots \\ \gamma_{m-1}^{(0)} & \cdots & \gamma_{m-1}^{(m-1)} \end{pmatrix}.$$

Thus, Problem 4 can be solved by Algorithm 3.

Algorithm 3: Multipoint Evaluation of a Linearized Polynomial

Input: Linearized polynomial $f(x)$, Evaluation points $G_0, \ldots, G_{m-1}$

1 Calculate the q-transform of $f(x)$: $F_i = f(\beta^{[i]})$ for $i = 0, \ldots, m-1$ (3)

2 Calculate representation of $F(x)$ over $\mathbb{F}_q$: $f_j^{(i)}$, $0 \le i, j \le m-1$

3 Calculate $\mathbf{F} \cdot \mathbf{G}$.

Output: $f(G_0), \ldots, f(G_{m-1})$

Step 1 is a q-transform and requires complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$. Step 2 requires no complexity since the representation over $\mathbb{F}_q$ is given. Finally, Step 3 consists of multiplying two $m \times m$ matrices over $\mathbb{F}_q$ which is done with at most $\mathcal{O}(m^3)$ operations over $\mathbb{F}_q$ and Algorithms 2 and 3 have complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$.

4.2 Direct Fast Symbolic Product

Recall that in a normal basis representation, the multiplication $c = a \cdot b \in \mathbb{F}_{q^m}$ is done in the representation over the ground field by $\underline{c} = \sum_{i=0}^{m-1} b^{(i)} (\underline{a}^{\leftarrow i} \mathbf{T})^{\rightarrow i}$. The multiplication of an element $a \in \mathbb{F}_{q^m}$ and an element of the normal basis $\beta_i = \beta^{[i]}$ is done by $(\underline{a}^{\leftarrow i} \mathbf{T})^{\rightarrow i}$ with complexity $\mathcal{O}(m)$ over $\mathbb{F}_q$.

To calculate the symbolic product $h(x) = f(g(x)) \mod (x^{[m]} - x)$ (1), let us denote $n = \min\{\deg_q h(x), m\}$ and the representation of elements over $\mathbb{F}_q$ by $\underline{f_j} = (f_j^{(0)}\ \ldots\ f_j^{(m-1)})$ and analogously, we denote $\underline{g_j}$. Note that a cyclic shift to the right by i positions is denoted by $\underline{g_j}^{\rightarrow i}$ and corresponds to the representation of $g_j^{[i]} \in \mathbb{F}_{q^m}$ over the ground field $\mathbb{F}_q$. The coefficients of the symbolic product can then be written by

$$h_\ell = \sum_{j=0}^{n-1} f_j g_{\ell-j}^{[j]} = \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} g_{\ell-j}^{(i)\rightarrow j} \cdot (\underline{f_j}^{\leftarrow i} \mathbf{T})^{\rightarrow i}, \quad \forall \ell = 0, \ldots, n-1.$$

Note that the calculation of $(\underline{f_j}^{\leftarrow i} \mathbf{T})^{\rightarrow i}$ is the same for all ℓ. Hence, we calculate it for all $i = 0, \ldots, m-1$ and $j = 0, \ldots, n-1$ with complexity $\mathcal{O}(nm^2)$ over $\mathbb{F}_q$. Then, we sum up for each $\ell = 0, \ldots, n-1$, which consists each time of nm multiplications over $\mathbb{F}_q$. Hence, the sum is calculated with complexity $\mathcal{O}(n^2 m)$ operations over $\mathbb{F}_q$ and all coefficients of the symbolic product are calculated with complexity $\mathcal{O}(nm^2)$ over $\mathbb{F}_q$. This approach is favorable to the transform–based approach if $n < m$.

4.3 Fast Symbolic Division

The fast symbolic division uses similar ideas as the fast (tranformed) symbolic product and will be described only shortly. We can rewrite (7) by

$$\left(\beta^{[0]} \ \ldots \ \beta^{[m-1]}\right) \cdot \mathbf{H} = \left(\beta^{[0]} \ \ldots \ \beta^{[m-1]}\right) \cdot \mathbf{F} \cdot \mathbf{G},$$

where $\mathbf{H} \in \mathbb{F}_q^{m \times m}$ denotes the representation of $(H_0 \ H_1 \ \ldots \ H_{m-1}) = (f(\beta^{[0]}) \ \ldots \ f(\beta^{[m-1]}))$ over the small field.

For the right symbolic division, $\mathbf{H}$ and $\mathbf{G}$ are known and for the left symbolic division, $\mathbf{H}$ and $\mathbf{F}$ are known. We have to solve $\mathbf{H} = \mathbf{F} \cdot \mathbf{G}$ for $\mathbf{G}$ or $\mathbf{F}$. This requires at most complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$ and the (right/left) symbolic division can also be accomplished with complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$.

5 Solving the Transformed Key Equation

5.1 Idea and Algorithm

Equation (6) can be rewritten with some polynomial $\Omega(x)$:

$$\Lambda(x) \otimes E(x) = \Omega(x) \cdot (x^{[m]} - x),$$

and with $R(x) = f(x) + E(x)$, we obtain $\Lambda(x) \otimes f(x) = \Lambda(x) \otimes R(x) - \Omega(x) \cdot (x^{[m]} - x)$. Comparing this equation to (2), we obtain Algorithm 4. We can run the LEEA with the input polynomials $(x^{[m]} - x)$ and $R(x)$ and the stopping condition $d_S = \lfloor (n+k)/2 \rfloor$. If $\mathrm{rank}_q(\mathbf{r} - \mathbf{c}) \leq \lfloor (n-k)/2 \rfloor$ for some codeword $\mathbf{c}$, the remainder $r_i(x) = \Lambda(x) \otimes f(x)$ and $u_i(x) = \Lambda(x)$ and we can find $f(x)$ by a (fast) left symbolic division. Otherwise, the remainder of the symbolic division will be unequal to zero, and we return *decoding failure*.

This algorithm can also be seen as a generalization of the *Gao Algorithm* for Reed–Solomon codes [5] for linearized polynomials.

Algorithm 4: Solving the Transformed Key Equation with the LEEA

Input: Received word $r(x)$

1 Calculate $R(x)$ as the q-transform of $r(x)$ (3)
2 $r(x), u(x), v(x) \leftarrow$ `LEEA` $(x^{[m]} - x, R(x))$ with $d_S = \lfloor (n+k)/2 \rfloor$
3 $\widehat{f}(x), rem(x) \leftarrow$ `LDiv` $(r(x), u(x))$
4 **if** $rem(x) = 0$ **then**
  Output: Evaluation Polynomial $\widehat{f}(x)$, $\Lambda(x) = u(x)$
5 **else**
  Output: Decoding Failure

The following theorem proves the correctness of our algorithm.

Theorem 3 (Correctness of Algorithm 4). *If the received word* $\mathbf{r}$ *has rank distance* $t \leq \lfloor (n-k)/2 \rfloor$ *from a codeword* $\mathbf{c} \in \mathcal{G}(n,k)$*, then Algorithm 4 solves Problem 3, i.e., it returns* $\Lambda(x)$ *and the evaluation polynomial* $f(x)$ *of* $\mathcal{G}(n,k)$*, otherwise decoding failure.*

Proof (Sketch). If $t = \mathrm{rank}_q(\mathbf{r} - \mathbf{c}) \leq \lfloor (n-k)/2 \rfloor$, there is a unique solution of (6) (except for a constant factor), where $\deg_q f(x) < k$ and $\deg_q \Lambda(x) = t$. We show that Algorithm 4 finds exactly this minimal–degree solution.

Due to (2), for each step of $\texttt{LEEA}(x^{[m]} - x, R(x))$

$$r_i(x) \equiv u_i(x) \otimes R(x) \mod (x^{[m]} - x)$$

holds and because of our stopping condition, for the output of the LEEA $\deg_q r_i(x) < \lfloor (n+k)/2 \rfloor$ and $\deg_q u_i(x) \leq \lfloor (n-k)/2 \rfloor$.

Assume the LEEA does not find the solution of *minimal* degree, i.e., we obtain a second solution such that:

$$\widetilde{\Lambda}(x) \otimes \widetilde{f}(x) \equiv \widetilde{\Lambda}(x) \otimes R(x) \mod (x^{[m]} - x),$$

where $\deg_q \widetilde{\Lambda}(x) > t$. Since

$$R(x) \equiv \texttt{LDiv}\ (\Lambda(x) \otimes f(x), \Lambda(x)) \equiv \texttt{LDiv}\ (\widetilde{\Lambda}(x) \otimes \widetilde{f}(x), \widetilde{\Lambda}(x)) \mod (x^{[m]} - x),$$

the following has to hold with some polynomial $b(x)$:

$$\begin{aligned} \widetilde{\Lambda}(x) &= b(x) \otimes \Lambda(x) \mod (x^{[m]} - x), \\ \widetilde{\Lambda}(x) \otimes \widetilde{f}(x) &= b(x) \otimes \Lambda(x) \otimes f(x) \mod (x^{[m]} - x). \end{aligned}$$

And hence, for the second solution we have:

$$b(x) \otimes \Lambda(x) \otimes f(x) \equiv b(x) \otimes \Lambda(x) \otimes R(x) - b(x) \otimes \Omega(x) \otimes (x^{[m]} - x).$$

If this was a result of $\texttt{LEEA}(x^{[m]} - x, R(x))$, then in this step $\widetilde{u}_i(x) = b(x) \otimes \Lambda(x)$ and $\widetilde{v}_i(x) = b(x) \otimes \Omega(x)$. Since $\widetilde{u}_i(x)$ and $\widetilde{v}_i(x)$ have to be symbolically relatively prime, $b(x)$ is a scalar and Algorithm 4 finds the unique solution except for a constant factor.

On the other hand assume $t > \lfloor (n-k)/2 \rfloor$ and Algorithm 4 returns $\widetilde{f}(x)$ with $\deg_q \widetilde{f}(x) < k$. In last step of the LEEA, the following holds:

$$u_i(x) \otimes R(x) \equiv u_i(x) \otimes \widetilde{f}(x) \mod (x^{[m]} - x),$$

and hence also

$$u_i(R(\beta_j) - \widetilde{f}(\beta_j)) = u_i(r_j - c_j) = 0, \qquad \forall j = 0, \ldots, m-1.$$

Due to the stopping condition, $\deg_q u_i(x) \leq \lfloor (n-k)/2 \rfloor$ and $u_i(x)$ can have at most $\lfloor (n-k)/2 \rfloor$ linearly independent roots. Hence, $r_j = c_j$ for at least $n - \lfloor (n-k)/2 \rfloor$ values of j. Therefore, there exists a $\mathbf{c}$ such that $\mathrm{rank}_q(\mathbf{r} - \mathbf{c}) \leq \lfloor (n-k)/2 \rfloor$. This is a contradiction and Algorithm 4 fails if the rank distance of $\mathbf{r}$ is greater than $\lfloor (n-k)/2 \rfloor$ to any codeword. □

5.2 Complexity Analysis

Algorithm 4 consists of three steps. As explained in Section 2.4, the q-transform can be calculated with complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$ if low–complexity normal bases are used. For the second step, we can use the fast LEEA from [12] which has complexity $\mathcal{O}(\mathcal{M}(m)\log m)$. This fast LEEA is based on the Divide and Conquer principle and in its recursions, the polynomials have very small q-degree. Hence, we use the direct fast symbolic product from Section 4.2, since its complexity depends on the q-degree of the input polynomials. Thus, the second step has complexity $\mathcal{O}(m^3 \log m)$ over $\mathbb{F}_q$. The third step of Algorithm 4 is a linearized division, which can be done with complexity $\mathcal{O}(m^3)$ over $\mathbb{F}_q$ using the ideas from Section 4. Hence, the overall decoding complexity is $\mathcal{O}(m^3 \log m)$ over $\mathbb{F}_q$.

6 Conclusion

We showed a method for fast decoding of Gabidulin codes. The decoding algorithm uses an equivalent of the Euclidean Algorithm and directly yields the evaluation polynomial of the code. By means of a fast symbolic product and division, the complexity of the *whole* decoding process is accelerated to $\mathcal{O}(m^3 \log m)$ operations over the *ground* field $\mathbb{F}_q$.

References

1. Gabidulin, E.M.: Theory of Codes with Maximum Rank Distance. Probl. Peredachi Inf. 21(1), 3–16 (1985)
2. Gabidulin, E.M., Afanasyev, V.B.: Kodirovanie v radioelektronike (Coding in Radio Electronics), in Russian (1986)
3. Gadouleau, M., Yan, Z.: Complexity of Decoding Gabidulin Codes. In: Information Sciences and Systems, 2008. CISS 2008. pp. 1081–1085 (2008)
4. Gao, S.: Normal Bases over Finite Fields. Ph.D. thesis, University of Waterloo, Department of Combinatorics and Optimization (1993)
5. Gao, S.: A New Algorithm for Decoding Reed-Solomon Codes. In: in Communications, Information and Network Security. vol. 2003, pp. 55–68 (2002)
6. Loidreau, P.: A Welch–Berlekamp Like Algorithm for Decoding Gabidulin Codes. Coding and Cryptography pp. 36–45 (2006)
7. Ore, O.: On a Special Class of Polynomials. Transactions of the American Mathematical Society 35, 559–584 (1933)
8. Paramonov, A.V., Tretjakov, O.V.: An Analogue of Berlekamp-Massey Algorithm for Decoding Codes in Rank Metric. In: Proc. of MIPT (1991)
9. Richter, G., Plass, S.: Error and Erasure Decoding of Rank-Codes with a Modified Berlekamp-Massey Algorithm. In: 5th International ITG Conference on Source and Channel Coding (SCC), Erlangen, January 14-16, 2004. pp. 203–211 (2004)
10. Sidorenko, V.R., Richter, G., Bossert, M.: Linearized Shift-Register Synthesis. (accepted for) IEEE Transactions on Information Theory (2011)
11. Silva, D., Kschischang, F.R.: Fast Encoding and Decoding of Gabidulin Codes. In: IEEE International Symposium on Information Theory 2009 (2009)
12. Wachter, A., Sidorenko, V., Bossert, M.: A Fast Linearized Euclidean Algorithm for Decoding Gabidulin Codes. In: Twelfth International Workshop on Algebraic and Combinatorial Coding Theory (ACCT 2010). pp. 298–303 (September 2010)