From the Euclidean Algorithm for Solving a Key Equation for Dual Reed–Solomon Codes to the Berlekamp–Massey Algorithm*

Maria Bras-Amorós[1,**] and Michael E. O'Sullivan[2]

[1] Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
maria.bras@urv.cat
[2] Department of Mathematics and Statistics
San Diego State University
mosulliv@sciences.sdsu.edu

Abstract. The two primary decoding algorithms for Reed-Solomon codes are the Berlekamp-Massey algorithm and the Sugiyama et al. adaptation of the Euclidean algorithm, both designed to solve a key equation. This article presents a new version of the key equation and a way to use the Euclidean algorithm to solve it. A straightforward reorganization of the algorithm yields the Berlekamp-Massey algorithm.

1 Introduction

For correcting primal Reed-Solomon codes a useful tool are the so-called locator and evaluator polynomials. Once we know them, the error positions are determined by the inverses of the roots of the locator polynomial and the error values can be computed by a formula due to Forney [4] which uses the evaluator and the derivative of the locator evaluated at the inverses of the error positions.

Berlekamp presented in [1] a key equation determining the decoding polynomials for primal Reed-Solomon codes. Sugiyama et al. introduced in [9] their celebrated algoithm for solving this key equation based on the Euclidean algorithm for computing the greatest common divisor of two polynomials and the coefficients of the Bézout's identity. Another celebrated algorithm for solving the key equation is the algorithm by Berlekamp and Massey [1, 6] which is widely accepted to have better performance than the Sugiyama et al. algorithm, although its formulation is more difficult to understand. The connections between both algorithms were analyzed in [3, 5].

In this work we take the perspective of dual Reed Solomon codes. In Section 3 we present a key equation for dual Reed-Solomon codes and in Section 4 we

* An extended version is in preparation for being submitted for publication.

** This work was partly supported by the Catalan Government through a grant 2005 SGR 00446 and by the Spanish Ministry of Education through projects TSI2007-65406-C03-01 E-AEGIS and CONSOLIDER CSD2007-00004 ARES.

M. Bras-Amorós and T. Høholdt (Eds.): AAECC 2009, LNCS 5527, pp. 32–42, 2009.
© Springer-Verlag Berlin Heidelberg 2009

introduce an Euclidean-based algorithm for solving it, following Sugiyama et al.'s idea. While the key equation for primal Reed-Solomon codes states that a linear combination of the decoding polynomials is a multiple of a certain power of x, in the key equation presented here the linear combination has bounded degree. Then, while in Sugiyama et al.'s algorithm the locator and evaluator polynomials play the role of one of the Bézout's coefficients and the residue respectively, in the Euclidean algorithm presented here the locator and evaluator polynomials play the role of the two coefficients of the Bézout's identity.

The decoding polynomials are now slightly different and in a sense more natural, since the error positions are given by the roots themselves of the locator polynomial rather than their inverses and that the error values are obtained by evaluating the evaluator polynomial and the derivative of the locator polynomial at the error positions rather than evaluating them at the inverses of the error positions. In addition the equivalent of the Forney formula does not have the minus sign.

In Section 5 we prove that the new Euclidean-based algorithm is exactly the Berlekamp-Massey algorithm. The connections between the Euclidean and the Berlekamp-Massey algorithms seem to be much more transparent for the dual Reed-Solomon codes than for the primal codes [3, 5]. In Section 6 we move back to primal Reed Solomon codes and see how all the developments of the previous sections can be applied to primal codes with minor modifications.

2 Settings and Notations

In this section we establish the notions and notations that we will use in the present work. A general reference is [8]. Let $\mathbb{F}$ be a finite field of size $q = p^m$ and let α be a primitive element in $\mathbb{F}$. Let $n = q - 1$. We identify the vector $u = (u_0, \ldots, u_{n-1})$ with the polynomial $u(x) = u_0 + \cdots + u_{n-1}x^{n-1}$ and denote $u(a)$ the evaluation of $u(x)$ at a.

Classically the (primal) Reed-Solomon code $C^*(k)$ of dimension k is the cyclic code generated by the polynomial $(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k})$, and has generator matrix $G^*(k) = (\alpha^{ij})_j^i$, for $i = 0, \ldots, k-1$, and $j = 0, \ldots, n-1$, while the dual Reed-Solomon code $C(k)$ of dimension k is generated by the polynomial $(x - \alpha^{-(k+1)})\cdots(x-\alpha^{-(n-1)})(x-1)$ and has generator matrix $G^*(k) = (\alpha^{(i+1)j})_j^i$, for $i = 0, \ldots, k-1$, and $j = 0, \ldots, n-1$.

Both codes have minimum distance $d = n - k + 1$. Furthermore, $C(k)^\perp = C^*(n-k)$. There is a natural bijection from $\mathbb{F}^n$ to itself, $c \mapsto c^*$, that takes $C(k)$ to $C^*(k)$. Indeed, let i be a vector of dimension k and let $c = (c_0, c_1, \ldots, c_{n-1}) = iG(k) \in C(k)$, $c^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*) = iG^*(k) \in C^*(k)$. Then,

$$c = (c_0^*, \alpha c_1^*, \alpha^2 c_2^*, \ldots, \alpha^{n-1} c_{n-1}^*). \tag{1}$$

In particular, $c(\alpha^i) = c_0^* + \alpha c_1^* \alpha^i + \alpha^2 c_2^* \alpha^{2i} + \cdots + \alpha^{n-1} c_{n-1}^* \alpha^{(n-1)i} = c^*(\alpha^{i+1})$.

3 Key Equations for Primal and Dual Reed-Solomon Codes

3.1 Polynomials for Correction of RS Codes

Suppose that a word $c^* \in C^*(k)$ is transmitted and that an error e^* occurred with $t \leq \frac{d-1}{2}$ non-zero positions, so that $u^* = c^* + e^*$ is received. Define the *error locator polynomial* Λ^* and the *error evaluator polynomial* Ω^* as

$$\Lambda^* = \prod_{i:e_i^* \neq 0} (1 - \alpha^i x), \qquad \Omega^* = \sum_{i:e_i^* \neq 0} e_i^* \alpha^i \prod_{j:e_j^* \neq 0, j \neq i} (1 - \alpha^i x),$$

and the *syndrome polynomial*

$$S^* = e^*(\alpha) + e^*(\alpha^2)x + e^*(\alpha^3)x^2 + \cdots + e^*(\alpha^n)x^{n-1}.$$

Notice that from the received word we only know $e^*(\alpha) = u^*(\alpha), \ldots, e^*(\alpha^{n-k}) = u^*(\alpha^{n-k})$. This is why we use the *truncated syndrome polynomial*

$$\bar{S}^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^{n-k})x^{n-k-1}.$$

If Λ^* and Ω^* are known, the error positions can be identified as the indices i such that

$$\Lambda^*(\alpha^{-i}) = 0$$

and the error values can be computed by the Forney formula [4]

$$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}.$$

3.2 Polynomials for Correction of Dual RS Codes

Suppose that a word $c \in C(k)$ is transmitted and that an error e occurred with $t \leq \frac{d-1}{2}$ non-zero positions, so that $u = c + e$ is received. In this case define the *error locator polynomial* Λ and the *error evaluator polynomial* Ω as

$$\Lambda = \prod_{i:e_i \neq 0} (x - \alpha^i), \qquad \Omega = \sum_{i:e_i \neq 0} e_i \prod_{j:e_j \neq 0, j \neq i} (x - \alpha^i), \tag{2}$$

and the *syndrome polynomial*

$$S = e(\alpha^{n-1}) + e(\alpha^{n-2})x + \cdots + e(\alpha)x^{n-2} + e(1)x^{n-1}. \tag{3}$$

Notice that now, from the received word we only know $e(1) = u(1), \ldots, e(\alpha^{n-k-1}) = u(\alpha^{n-k-1})$. This is why we use the *truncated syndrome polynomial*

$$\bar{S} = e(\alpha^{n-k-1})x^k + e(\alpha^{n-k-2})x^{k+1} + \cdots + e(1)x^{n-1}. \tag{4}$$

In this case, if Λ and Ω are known, the error positions can be identified as the indices i such that

$$\Lambda(\alpha^i) = 0 \tag{5}$$

and the error values can be computed by the formula

$$e_i = \frac{\Omega(\alpha^i)}{\Lambda'(\alpha^i)}. \tag{6}$$

It is easy to check that when the error vectors e and e^* satisfy the relationship (1), then the polynomials Λ, Ω, S, S associated to the error e are related to the polynomials Λ^*, Ω^*, S^*, $\bar{S}^*$ associated to the error e^* as follows:

$$\begin{array}{ll} \Lambda = x^t \Lambda^*(1/x), & \Omega = x^{t-1}\Omega^*(1/x), \\ S = x^{n-1}S^*(1/x), & \bar{S} = x^{n-1}\bar{S}^*(1/x). \end{array}$$

3.3 Key Equations

A straightforward computation shows that

$$\Lambda S = (x^n - 1)\Omega \tag{7}$$

or, equivalently, $\Lambda^* S^* = (1 - x^n)\Omega^*$. See [7] for a proof. In particular, the polynomial $\Lambda^*\bar{S}^* + (x^n - 1)\Omega^* = \Lambda^*(\bar{S}^* - S^*)$ has only terms of order $n - k$ or larger and this implies

$$\boxed{\Lambda^*\bar{S}^* = \Omega^* \mod x^{n-k}}$$

This is the key equation introduced by Berlekamp [1]. Massey [6] gave an algorithm for solving linear feedback shift registers and its application to decoding BCH and thus RS codes. Sugiyama et al. [9] recognized that the Euclidean algorithm for finding the greatest common divisor of two polynomials and for solving Bézout's identity, could also be adapted to solve this kind of equation for Λ^* and Ω^*, starting with $r_{-2} = x^{n-k}$ and $r_{-1} = \bar{S}^*$. Their method is often referred to as the Euclidean decoding algorithm for Reed Solomon codes.

On the other hand, the polynomial $\Lambda\bar{S} - (x^n - 1)\Omega = \Lambda(\bar{S} - S)$ has degree at most $\lfloor \frac{d-1}{2} \rfloor + k - 1 = \lfloor \frac{d-1}{2} \rfloor + n - d = n - \lceil \frac{d+1}{2} \rceil$. That is

$$\boxed{\deg(\Lambda\bar{S} - (x^n - 1)\Omega) \leq n - \lceil \tfrac{d+1}{2} \rceil}$$

The aim of the present work is to deal with this alternative key equation on Λ and Ω and solving it by the Euclidean algorithm starting with $r_{-2} = x^n - 1$ and $r_{-1} = \bar{S}$.

4 Euclidean Algorithm for Dual Reed-Solomon Codes

The next theorem characterizes the decoding polynomials by means of the alternative key equation, the polynomial degrees, and their coprimality. It is the analogue of [2, Theorem 4.8] for the standard key equation, and the proof is similar.

Theorem 1. *Suppose that at most* $\lfloor \frac{d-1}{2} \rfloor$ *errors occurred. Then* Λ *and* Ω *are the unique polynomials* f, φ *satisfying the following properties.*

1. $deg(f\bar{S} - (x^n - 1)\varphi) \leq n - \lceil \frac{d+1}{2} \rceil$
2. $deg(\varphi) < deg(f) \leq \lceil \frac{d-1}{2} \rceil$
3. f, φ *are coprime.*
4. f *is monic*

Consider the following algorithm:

Initialize:
$r_{-2} = x^n - 1$, $f_{-2} = 0$, $\varphi_{-2} = -1$,
$r_{-1} = \bar{S}$, $f_{-1} = 1$, $\varphi_{-1} = 0$,
$i = -1$.
while deg$(r_i) > n - \lceil \frac{d+1}{2} \rceil$**:**
$q_i =$ **Quotient**(r_{i-2}, r_{i-1})
$r_i =$ **Remainder**(r_{i-2}, r_{i-1})
$\varphi_i = \varphi_{i-2} - q_i \varphi_{i-1}$
$f_i = f_{i-2} - q_i f_{i-1}$
end while
Return f_i/**LeadingCoefficient**$(f_i), \varphi_i$/**LeadingCoefficient**(f_i)

The polynomial φ has been initialized to negative 1 because we want that f_i, φ_i satisfy at each step $f_i\bar{S} - \varphi_i(x^n - 1) = r_i$.

The next theorem verifies the algorithm. Its proof has also been ommitted and will appear in the final form of this article.

Theorem 2. *If* $t \leq \frac{d-1}{2}$ *then the algorithm outputs* Λ *and* Ω.

5 From the Euclidean to the Berlekamp-Massey Algorithm

In this section we will perform a series of minor modifications to the previous algorithm that will lead to the Berlekamp-Massey algorithm. This will show that they are essentially the same.

First of all, notice that the updating step in the previous algorithm can be expressed in matrix form as

$$\begin{pmatrix} r_i & f_i & \varphi_i \\ r_{i-1} & f_{i-1} & \varphi_{i-1} \end{pmatrix} = \begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{i-1} & f_{i-1} & \varphi_{i-1} \\ r_{i-2} & f_{i-2} & \varphi_{i-2} \end{pmatrix}.$$

Furthermore, if $q_i = q_i^{(0)} + q_i^{(1)}x + \cdots + q_i^{(l)}x^l$ then

$$\begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -q_i^{(2)}x^2 \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

If we split all matrix multiplications we will need to add intermediate variables. Thus, it does not make sense anymore to consider matrices

$$\begin{pmatrix} r_i & f_i & \varphi_i \\ r_{i-1} & f_{i-1} & \varphi_{i-1} \end{pmatrix}.$$

We will use auxiliary polynomials $R_i, F_i, \Psi_i, \tilde{R}_i, \tilde{F}_i, \tilde{\Psi}_i$ and matrices

$$\begin{pmatrix} R_i & F_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}.$$

These matrices will satisfy at given stages of the algorithm (exactly when $\deg(R_i) < \deg(\tilde{R}_i)$) that R_i, F_i, Ψ_i are the remainder and the intermediate Bézout coefficients in one of the steps of the original Euclidean algorithm and that $\tilde{R}_i, \tilde{F}_i, \tilde{\Psi}_i$ are respectively the remainder and the intermediate Bézout coefficients previous to R_i, F_i, Ψ_i.

The previous algorithm can be expressed now:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & \Psi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Psi}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n & -1 & 0 & -1 \end{pmatrix}$$

while deg$(R_i) > n - \lceil \frac{d+1}{2} \rceil$**:**

$$\begin{pmatrix} R_{i+1} & F_{i+1} & \Psi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Psi}_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}$$

(quotient=0)

while deg$(R_i) \geq$ **deg**$(\tilde{R}_i)$:

$\mu =$ **LeadingTerm**(R_i)/**LeadingTerm**$(\tilde{R}_i)$

(quotient=quotient+μ**)**

$$\begin{pmatrix} R_{i+1} & F_{i+1} & \Psi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Psi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}$$

end while

end while

Return F_i/**LeadingCoefficient**$(F_i), \Psi_i$/**LeadingCoefficient**(F_i)

5.1 Making Remainders Monic

A useful modification is to keep $\tilde{R}_i$ monic in all steps. This makes it easier to compute the μ's. It is enough to force this every time we get $\deg(R_i) < \deg(\tilde{R}_i)$. When $\deg(R_i) \geq \deg(\tilde{R}_i)$, then $\tilde{R}_i$ stays the same and so it remains monic. To this end we divide R_i by its leading coefficient $\tilde{\mu}$, and also $\tilde{F}_i$ and $\tilde{\Psi}_i$ in order to keep the properties. Analogously, we multiply R_i, F_i and Ψ_i by $-\tilde{\mu}$ in order to make the F_i's monic. Another modification is the use of an integer p keeping track of the degree difference between R_i and $\tilde{R}_i$ and that we take leading coefficients of R_i instead of leading terms.

With these modifications, the algorithm is now:

Initialize:

$$\begin{pmatrix} R_{-1} & R_{-1} & \Psi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Psi}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\mathbf{deg}(R_i) > n - \lceil \frac{d+1}{2} \rceil$**:**

$\mu = \mathbf{LeadingCoefficient}(R_i)$

$p = \mathbf{deg}(R_i) - \mathbf{deg}(\tilde{R}_i)$

if $p < 0$ **then**

$$\begin{pmatrix} R_{i+1} & F_{i+1} & \Psi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Psi}_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & R_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & \Psi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Psi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}$$

end if

end while

Return F_i, Ψ_i

5.2 Joining Steps

It is easy to check that after each step corresponding to $p < 0$ the new p is exactly the previous one with opposite sign and so is μ. With this observation we can join each step corresponding to $p < 0$ with the following one:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & \Psi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Psi}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\mathbf{deg}(R_i) > n - \lceil \frac{d+1}{2} \rceil$**:**

$\mu = \mathbf{LeadingCoefficient}(R_i)$

$p = \mathbf{deg}(R_i) - \mathbf{deg}(\tilde{R}_i)$

if $p < 0$ **then**

$$\begin{pmatrix} R_{i+1} & F_{i+1} & \Psi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Psi}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & \Psi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Psi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Psi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Psi}_i \end{pmatrix}$$

end if

end while

Return F_i, Ψ_i

5.3 Skipping Remainders

Finally, the only reason to keep the polynomials R_i (and $\tilde{R}_i$) is that we need to compute their leading coefficients (the μ's). We now show that these leading coefficients may be obtained without reference to the polynomials R_i. This allows to compute the F_i, Ψ_i iteratively and dispense with the polynomials R_i.

On one hand, the remainder $R_i = F_i\bar{S} - \Psi_i(x^n - 1) = F_i\bar{S} - x^n\Psi_i + \Psi_i$ has degree at most $n-1$ for all $i \geq -1$. This means that all terms of $x^n\Psi_i$ cancel with terms of $F_i\bar{S}$ and that the leading term of R_i must be either a term of $F_i\bar{S}$ or a term of Ψ_i or a sum of a term of $F_i\bar{S}$ and a term of Ψ_i.

On the other hand, the algorithm only computes LeadingCoefficient(R_i) while $\deg(R_i) \geq n - \lceil \frac{d-1}{2} \rceil$. We want to see that in this case the leading term of R_i has degree strictly larger than that of Ψ_i. Indeed, one can check that for $i \geq -1$, $\deg(\Psi_i) < \deg(F_i)$ and that all F_i's in the algorithm have degree at most $\lceil \frac{d-1}{2} \rceil$. So $\deg(\Psi_i) < \deg(F_i) \leq \lceil \frac{d-1}{2} \rceil \leq d - \lceil \frac{d-1}{2} \rceil \leq n - \lceil \frac{d-1}{2} \rceil \leq \deg(R_i)$.

The previous algorithm can be transformed in a way such that the remainders are not kept but their degrees:

Initialize:
$d_{-1} = \mathbf{deg}(\bar{S})$ $d_{-2} = n$
$\begin{pmatrix} f_{-1} & \varphi_{-1} \\ F_{-1} & \Psi_{-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
while $d_i > n - \lceil \frac{d+1}{2} \rceil$**:**
$\mu = \mathbf{Coefficient}(f_i\bar{S}, d_i)$
$p = d_i - d_{i-1}$
if $\mu = 0$ **then**
$d_i = d_i - 1$
else if $p < 0$ **then**
$\begin{pmatrix} f_{i+1} & \varphi_{i+1} \\ F_{i+1} & \Psi_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} f_i & \varphi_i \\ F_i & \Psi_i \end{pmatrix}$
$d_i = d_{i-2} - 1$
else
$\begin{pmatrix} f_{i+1} & \varphi_{i+1} \\ F_{i+1} & \Psi_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} f_i & \varphi_i \\ F_i & \Psi_i \end{pmatrix}$
$d_i = d_i - 1$
end if
end while
Return f_i, φ_i

This last algorithm is the Berlekamp-Massey algorithm that solves the linear recurrence $\sum_{j=0}^{t} \Lambda_j e(\alpha^{i+j-1}) = 0$ for all $i > 0$. This recurrence is derived from $\Lambda\frac{S}{x^n-1}$ being a polynomial (see (7)) and thus having no terms of negative order in its expression as a Laurent series in $1/x$, and from the equality

$$\frac{S}{x^n - 1} = \frac{1}{x}\left(e(1) + \frac{e(\alpha)}{x} + \frac{e(\alpha^2)}{x^2} + \cdots\right).$$

A proof of this last equality can be found in [7].

6 Moving Back to Primal Reed-Solomon Codes

It is well known that the words of a dual Reed-Solomon code are in bijection with those of the primal Reed-Solomon code of the same dimension. The

natural bijection was previously mentioned in (1). Suppose that a code word c^* from the primal Reed-Solomon code $C^*(k)$ is transmitted, and that an error e^* is added to it giving the received word $u^* = c^* + e^*$. Notice that if $c = (c_0^*, \alpha c_1^*, \alpha^2 c_2^*, \ldots, \alpha^{n-1} c_{n-1}^*)$ and $e = (e_0^*, \alpha e_1^*, \alpha^2 e_2^*, \ldots, \alpha^{n-1} e_{n-1}^*)$, then $c \in C(k)$ and $u := c + e = (u_0^*, \alpha u_1^*, \alpha^2 u_2^*, \ldots, \alpha^{n-1} u_{n-1}^*)$. So, e has the same non-zero positions as e^*, and the error values e_i^* added to u_i^* can be computed from the error values e_i added to u_i by $e_i^* = e_i/\alpha^i$.

Table 1. Steps of the classical Euclidean algorithm

$q(x) = 0$
$r(x) = x^9$
$f(x) = 2$
$\varphi(x) = 0$

$q(x) = 0$
$r(x) = \alpha^{14}x^8 + \alpha x^7 + \alpha^{10}x^6 + \alpha^{10}x^5 + \alpha^{17}x^4 + \alpha^5 x^3 + x^2 + \alpha^{11}x + 2$
$f(x) = 0$
$\varphi(x) = 1$

$q(x) = \alpha^{12}x + \alpha^{12}$
$r(x) = \alpha^3 x^7 + \alpha^{22}x^6 + 2x^5 + \alpha^6 x^4 + \alpha^{16}x^3 + \alpha^9 x^2 + \alpha^5 x + \alpha^{12}$
$f(x) = 2$
$\varphi(x) = \alpha^{25}x + \alpha^{25}$

$q(x) = \alpha^{11}x + \alpha^{21}$
$r(x) = \alpha^3 x^6 + x^5 + \alpha^{23}x^4 + \alpha^{19}x^3 + \alpha^{23}x^2 + \alpha^{24}x + \alpha^{17}$
$f(x) = \alpha^{11}x + \alpha^{21}$
$\varphi(x) = \alpha^{23}x^2 + \alpha^3 x + \alpha^4$

$q(x) = x + 2$
$r(x) = \alpha^{10}x^5 + \alpha^{11}x^4 + \alpha^{14}x^3 + \alpha x^2 + \alpha^{23}x + \alpha^3$
$f(x) = \alpha^{24}x^2 + \alpha^9 x + \alpha^2$
$\varphi(x) = \alpha^{10}x^3 + \alpha^{20}x^2 + \alpha^4 x + \alpha^{16}$

$q(x) = \alpha^{19}x + \alpha^{10}$
$r(x) = \alpha^4 x^3 + \alpha^4 x^2 + \alpha^{10}x + \alpha^{20}$
$f(x) = \alpha^4 x^3 + x^2 + \alpha^7 x + 2$
$\varphi(x) = \alpha^{16}x^4 + \alpha^4 x^3 + \alpha^{17}x^2 + \alpha^9 x + \alpha^7$

Now we can perform the previous versions of the algorithm using

$$\begin{aligned}\bar{S} &= e(\alpha^{n-k-1})x^k + e(\alpha^{n-k-2})x^{k+1} + \cdots + e(1)x^{n-1} \\ &= e^*(\alpha^{n-k})x^k + e^*(\alpha^{n-k-1})x^{k+1} + \cdots + e^*(\alpha)x^{n-1},\end{aligned}$$

which is known because it is equal to $u^*(\alpha^{n-k})x^k + u^*(\alpha^{n-k-1})x^{k+1} + \cdots + u^*(\alpha)x^{n-1}$.

Then, once we have the error positions, we can compute the error values as

$$e_i^* = \frac{\Omega(\alpha_i)}{\alpha^i \Lambda'(\alpha^i)}.$$

As an example, consider $\mathbb{F}_{27} = \mathbb{F}_2[x]/(x^3 + 2x + 1)$ and suppose that α is the class of x. Suppose $d = 10$ and $e^* = \alpha^4 x + \alpha^2 x^2 + \alpha^6 x^8 + \alpha^3 x^{24}$.

For the classical Euclidean algorithm, we need the polynomial

Table 2. Steps of the new Euclidean algorithm

$q(x) = 0$
$r(x) = x^{26} + 2$
$f(x) = 0$
$\varphi(x) = 2$

$q(x) = 0$
$r(x) = 2x^{25} + \alpha^{11}x^{24} + x^{23} + \alpha^5 x^{22} + \alpha^{17}x^{21} + \alpha^{10}x^{20} + \alpha^{10}x^{19} + \alpha x^{18} + \alpha^{14}x^{17}$
$f(x) = 1$
$\varphi(x) = 0$

$q(x) = 2x + \alpha^{24}$
$r(x) = \alpha^{14}x^{24} + \alpha^{16}x^{23} + \alpha^{25}x^{22} + \alpha^{17}x^{21} + \alpha^{20}x^{20} + \alpha^6 x^{19} + \alpha^7 x^{18} + \alpha^{25}x^{17} + 2$
$f(x) = x + \alpha^{11}$
$\varphi(x) = 2$

$q(x) = \alpha^{25}x + \alpha^{17}$
$r(x) = \alpha^{16}x^{23} + \alpha^{15}x^{22} + \alpha^7 x^{21} + \alpha^7 x^{20} + \alpha^{11}x^{19} + \alpha^{25}x^{18} + 2x^{17} + \alpha^{25}x + \alpha^{17}$
$f(x) = \alpha^{12}x^2 + \alpha x + \alpha^{25}$
$\varphi(x) = \alpha^{25}x + \alpha^{17}$

$q(x) = \alpha^{24}x + \alpha^6$
$r(x) = \alpha^{24}x^{22} + 2x^{21} + \alpha^{25}x^{20} + \alpha^9 x^{19} + \alpha^{21}x^{18} + \alpha^3 x^{17} + \alpha^{10}x^2 + \alpha^{24}x + \alpha^{11}$
$f(x) = \alpha^{23}x^3 + \alpha^9 x^2 + \alpha^{22}x + \alpha^{15}$
$\varphi(x) = \alpha^{10}x^2 + \alpha^{24}x + \alpha^{11}$

$q(x) = \alpha^{18}x + \alpha^{18}$
$r(x) = \alpha^8 x^{20} + \alpha^2 x^{19} + \alpha^{21}x^{18} + \alpha^{25}x^{17} + \alpha^{15}x^3 + \alpha^5 x^2 + \alpha^{25}x + \alpha^{25}$
$f(x) = \alpha^2 x^4 + \alpha^4 x^3 + \alpha^{12}x^2 + \alpha^{25}x + \alpha^{11}$
$\varphi(x) = \alpha^{15}x^3 + \alpha^5 x^2 + \alpha^{25}x + \alpha^{25}$

$\bar{S}^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^{n-k})x^{n-k-1} = \alpha^{14}x^8 + \alpha x^7 + \alpha^{10}x^6 + \alpha^{10}x^5 + a^{17}x^4 + \alpha^5 x^3 + x^2 + \alpha^{11}x + 2.$

For the new Euclidean algorithm we will use $\bar{S} = e^*(\alpha^{n-k})x^k + e^*(\alpha^{n-k-1})x^{k+1} + \cdots + e^*(\alpha)x^{n-1} = 2x^{25} + \alpha^{11}x^{24} + x^{23} + \alpha^5 x^{22} + \alpha^{17}x^{21} + \alpha^{10}x^{20} + \alpha^{10}x^{19} + \alpha x^{18} + \alpha^{14}x^17.$

The steps of both algorithms are written in Table 1 and Table 2.

References

1. Berlekamp, E.R.: Algebraic coding theory. McGraw-Hill Book Co., New York (1968)
2. Cox, D.A., Little, J., O'Shea, D. (eds.): Using algebraic geometry, 2nd edn. Graduate Texts in Mathematics, vol. 185. Springer, New York (2005)
3. Dornstetter, J.-L.: On the equivalence between Berlekamp's and Euclid's algorithms. IEEE Trans. Inform. Theory 33(3), 428–431 (1987)
4. David Forney Jr., G.: On decoding BCH codes. IEEE Trans. Information Theory IT-11, 549–557 (1965)
5. Heydtmann, A.E., Jensen, J.M.: On the equivalence of the Berlekamp-Massey and the Euclidean algorithms for decoding. IEEE Trans. Inform. Theory 46(7), 2614–2624 (2000)
6. Massey, J.L.: Shift-register synthesis and BCH decoding. IEEE Trans. Information Theory IT-15, 122–127 (1969)

7. O'Sullivan, M.E., Bras-Amorós, M.: The Key Equation for One-Point Codes, ch. 3, pp. 99–152 (2008)
8. Roth, R.: Introduction to coding theory. Cambridge University Press, Cambridge (2006)
9. Sugiyama, Y., Kasahara, M., Hirasawa, S., Namekawa, T.: A method for solving key equation for decoding Goppa codes. Information and Control 27, 87–99 (1975)