symmetry

Article

The Symmetric Key Equation for Reed–Solomon Codes and a New Perspective on 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, Av. Països Catalans 26, 43007 Tarragona, Catalonia

[2] Department of Mathematics and Statistics, San Diego State University, 5500 Campanile Drive, San Diego, CA 92182-7720, USA; mosullivan@sdsu.edu

* Correspondence: maria.bras@urv.cat

Received: 19 September 2019; Accepted: 25 October 2019; Published: 2 November 2019

Abstract: This paper presents a new way to view the key equation for decoding Reed–Solomon codes that unites the two algorithms used in solving it—the Berlekamp–Massey algorithm and the Euclidean algorithm. A new key equation for Reed–Solomon codes is derived for simultaneous errors and erasures decoding using the symmetry between polynomials and their reciprocals as well as the symmetries between dual and primal codes. The new key equation is simpler since it involves only degree bounds rather than modular computations. We show how to solve it using the Euclidean algorithm. We then show that by reorganizing the Euclidean algorithm applied to the new key equation we obtain the Berlekamp–Massey algorithm.

Keywords: Reed–Solomon codes; key equation; Berlekamp–Massey algorithm; Sugiyama et al. algorithm; euclidean algorithm

1. Introduction

Reed–Solomon codes are the basis of many applications such as secret sharing [1], distributed storage [2,3], private information retrieval [4] and the analysis of cryptographic hardness [5]. The most used tool for decoding Reed–Solomon codes is the key equation by Berlekamp [6] and the milestone algorithms that solve it are the Berlekamp–Massey algorithm [7] and the Sugiyama et al. adaptation of the Euclidean algorithm [8]. Their connections are analyzed in [9–12]. This paper is meant to bring a new unified presentation of the key equation, the Sugiyama-Euclidean algorithm and the Berlekamp–Massey algorithm for correcting errors and erasures for Reed–Solomon codes.

Section 2 presents a revisited key equation for both erasures and errors using the symmetry between polynomials and their reciprocals as well as the symmetries between dual and primal codes. In the new key equation, as opposed to the classical equation, there is no need to reference computations modulo a power of the indeterminate, and the correction polynomials reveal error locations rather than their inverses. Section 3 gives a way to solve the new key equation using the Euclidean algorithm. We show how the Berlekamp–Massey algorithm can be obtained by reorganizing the Euclidean algorithm. Hence, the whole paper is, in fact, a simple presentation of the Berlekamp–Massey algorithm as a restructured Euclidean algorithm.

2. Symmetric Key Equation

2.1. Reed–Solomon Codes

Suppose that $\mathbb{F}$ is a finite field of q elements and suppose that α is a primitive element of $\mathbb{F}$. Let $n = q - 1$. Each vector $u = (u_0, \dots, u_{n-1}) \in \mathbb{F}^n$ is identified with the polynomial $u(x) = u_0 + u_1 x + \cdots + u_{n-1}x^{n-1}$. The evaluation of $u(x)$ at a is then denoted $u(a)$. The cyclic code $C^*(k)$ of length n generated by the polynomial $(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k})$ is classically referred to as a (primal) Reed–Solomon code. Its dimension is k. On the other hand, the cyclic code $C(k)$ of lenth n generated by the polynomial $(x-\alpha^{n-(k+1)})(x-\alpha^{n-(k+2)})\cdots(x-\alpha)(x-1)$ is referred to as a dual Reed–Solomon code. Its dimension is k as well. The minimum distance of both codes is $d = n-k+1$. The codes are related by the equality $C(k)^\perp = C^*(n-k)$.

The vector space $\mathbb{F}^n$ is naturally bijected to itself through a map $c \mapsto c^*$ taking $C(k)$ to $C^*(k)$. For a vector $c = (c_0, c_1, \dots, c_{n-1}) \in \mathbb{F}^n$ the vector c^* is defined componentwise as $c^* = (c_0, \alpha^{-1}c_1, \alpha^{-2}c_2, \dots, \alpha c_{n-1})$. Symmetrically, if $c^* = (c_0^*, c_1^*, \dots, c_{n-1}^*)$, then $c = (c_0^*, \alpha c_1^*, \alpha^2 c_2^*, \dots, \alpha^{n-1}c_{n-1}^*)$. In particular, $c(\alpha^i) = c_0^* + \alpha c_1^* \alpha^i + \alpha^2 c_2^* \alpha^{2i} + \cdots + \alpha^{n-1}c_{n-1}^*\alpha^{(n-1)i} = c^*(\alpha^{i+1})$.

Due to this bijective map, algorithms for correcting errors and erasures for primal Reed–Solomon code are also applicable for dual Reed–Solomon codes and vice versa. Indeed, if the codeword $c \in C(k)$ at minimum distance of a received vector u differs from u by a vector of errors e, then the codeword $c^* \in C^*(k)$ at minimum distance of a received vector u^* differs from u^* by a vector of errors e^*.

2.2. Decoding for Errors and Erasures

Suppose that a noisy channel adds t errors and erases s other components of a transmitted codeword $c \in C(k)$ with $2t + s < d$. Let u be the received word after replacing the erased positions by 0 and let $e = u - c$. The *erasure locator polynomial* is defined as $\Lambda_r = \prod_{i: c_i \text{was erased}} (x - \alpha^i)$ while the *error locator polynomial* is defined as $\Lambda_e = \prod_{i: e_i \neq 0, c_i \text{not erased}} (x - \alpha^i)$. The product $\Lambda_r \Lambda_e$ is called Λ. We remark that while Λ_r is known driectly from the received word, the Λ_e is not *a priori* known. The *error evaluator polynomial* is defined as $\Omega = \sum_{\substack{i: e_i \neq 0 \\ \text{or } c_i \text{ erased}}} e_i \prod_{\substack{j: e_j \neq 0 \text{ or } c_j \text{ erased,} \\ \text{and } j \neq i}} (x - \alpha^i) = \sum_{i=0}^{n-1} e_i \dfrac{\Lambda}{x - \alpha^i}$ The error positions can be identified by $\Lambda_e(\alpha^i) = 0$ while the error values can be derived, as well as the erased values, from the analogue of the Forney formula [13]

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

Notice that in the traditional setting, the roots of the locator polynomial are not related to the error positions but to their inverses. Hence, in the new setting we take the reciprocals of the polynomials of the traditional setting thus establishing a symmetry between the different versions. Also, the classical Forney formula involves the evaluator polynomial and the derivative of the locator polynomial evaluated at the inverses of the error positions, while with the new settings we use directly the error positions.

Finally, the polynomial $S = e(\alpha^{n-1}) + e(\alpha^{n-2})x + \cdots + e(\alpha)x^{n-2} + e(1)x^{n-1}$ is called the *syndrome polynomial* of e.

Lemma 1. *$\Omega(x^n - 1) = \Lambda S$.*

Proof. We can compute directly,

$$\begin{aligned}\Omega(x^n-1) &= (x^n-1)\sum_{i=0}^{n-1} e_i \frac{\Lambda}{x-\alpha^i} \\ &= \Lambda \sum_{i=0}^{n-1} e_i \frac{x^n-1}{x-\alpha^i} \\ &= \Lambda \sum_{i=0}^{n-1} e_i \sum_{j=0}^{n-1} x^{n-1-j}(\alpha^i)^j \\ &= \Lambda \sum_{j=0}^{n-1} x^{n-1-j} \sum_{i=0}^{n-1} e_i(\alpha^j)^i \\ &= \Lambda \sum_{j=0}^{n-1} x^{n-1-j} e(\alpha^j) \\ &= \Lambda S\end{aligned}$$

□

The general term of S is $e(\alpha^{n-1-i})x^i$, but we only know from a received word the values $e(1) = u(1), \dots, e(\alpha^{n-k-1}) = u(\alpha^{n-k-1})$. For this reason we use the *truncated syndrome polynomial* defined as $\bar{S} = e(\alpha^{n-k-1})x^k + e(\alpha^{n-k-2})x^{k+1} + \cdots + e(1)x^{n-1}$. The degree of the polynomial $\Omega(x^n-1) - \Lambda\bar{S} = \Lambda(S-\bar{S})$ is at most $t+s+k-1 < \frac{d-s}{2} + s + n - d = n - \frac{d-s}{2}$. One consequence of this bound is that the reciprocal polynomials $\Omega^* = x^{t+s-1}\Omega(1/x)$, $\Lambda^* = x^{t+s}\Lambda(1/x)$ and the polynomial $\bar{S}^* = x^{n-1}\bar{S}(1/x)$ satisfy the well known Berlekamp key equation $\Lambda^*\bar{S}^* = \Omega^* \bmod x^{n-s-k}$. Theorem 1 uses the bound on the degree of $\Omega(x^n-1) - \Lambda\bar{S}$ to derive a symmetric key equation for dual Reed–Solomon codes. To prove it, we first need the next two lemmas.

Lemma 2. *Suppose that f is a polynomial of $\mathbb{F}[x]$ with $\deg(f) < n$. Suppose that for a given $\alpha \in \mathbb{F}^*$ the polynomial $f(x)\frac{x^n-1}{x-\alpha}$ has no term of degree $n-1$. Then α is a root of f.*

Proof. The Euclidean division of f by $x-\alpha$ gives a polynomial $g \in \mathbb{F}[x]$ of degree smaller than $n-1$ that satisfies $f(x) = f(\alpha) + g(x)(x-\alpha)$. Then $f(x)\frac{x^n-1}{x-\alpha} = f(\alpha)\frac{x^n-1}{x-\alpha} + g(x)(x^n-1)$. On one hand, the product $g(x)(x^n-1)$ has no term of degree $n-1$. On the other hand, the coefficient of $f(\alpha)\frac{x^n-1}{x-\alpha}$ of degree $n-1$ is exactly $f(\alpha)$. Hence, if $f(x)\frac{x^n-1}{x-\alpha}$ has no term of degree $n-1$, then necessarily $f(\alpha) = 0$. □

Lemma 3. *Suppose that f is a polynomial of $\mathbb{F}[x]$ with $\deg(f) \le n-s-t$ such that the terms of degree $n-t, \dots, n-1$ of $f\Lambda_r S$ are all zero. Then Λ_e is a divisor of f.*

Proof. Suppose that the terms of degree $n-t,\dots,n-1$ of $f\Lambda_r S$ are all zero. Suppose c_j was not erased and $e_j \neq 0$. Consider $g(x) = \Lambda_e/(x-\alpha_j)$. We have $\deg(g) = t-1$ and consequently the term of degree $n-1$ of $fg\Lambda_r S$ is 0. Then,

$$\begin{aligned} fg\Lambda_r S &= f(x)g(x)\Lambda_r(x)\frac{\Omega(x)(x^n-1)}{\Lambda(x)} \\ &= \sum_{k:e_k\neq 0} e_k f(x)g(x)\Lambda_r(x)\frac{x^n-1}{x-\alpha_k} \\ &= e_j f(x)g(x)\Lambda_r(x)\frac{x^n-1}{x-\alpha_j} \\ &\quad + \sum_{\substack{k:e_k\neq 0,\\ c_k \text{not erased}\\ k\neq j}} e_k f(x)\frac{g(x)}{x-\alpha_k}\Lambda_r(x)(x^n-1) \\ &\quad + \sum_{k:c_k \text{erased}} e_k f(x)g(x)\frac{\Lambda_r(x)}{x-\alpha_k}(x^n-1). \end{aligned}$$

Because of the restriction on the degree of f, none of the last two summands has term of degree $n-1$. Since the term of degree $n-1$ of $fg\Lambda_r S$ is 0, so is the term of degree $n-1$ of $f(x)g(x)\Lambda_r(x)\frac{x^n-1}{x-\alpha_j}$. By Lemma 2, $x-\alpha_j$ must be a divisor of f. Since j was chosen arbitrarily such that $e_j \neq 0$ and c_j was not erased, we conclude that Λ_e must divide f. □

Theorem 1 (Symmetric key equation). *Suppose that a number s of erasures occurred together with a number of at most $\lfloor \frac{d-s-1}{2} \rfloor$ errors. Then the polynomials Λ_e and Ω are uniquely determined by the conditions*

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

Proof. It is easy to see that Λ_e and Ω satisfy conditions 1, 2, 3. It follows from the previous lemmas that Λ_e and Ω satisfy condition 4. Conversely, suppose that f, φ satisfy the conditions 3 and 4. We will prove that the terms of degrees $n-t,\dots,n-1$ of $f\Lambda_r S$ are all zero. Then, by Lemma 3, and because $\deg(f) \leq \frac{d-s}{2} \leq n - \frac{d+s}{2} = n - s - \frac{d-s}{2} < n-s-t$, it can be deduced that Λ_e is a divisor of f. Indeed, write

$$f\Lambda_r S = (f\Lambda_r\bar{S} - \varphi(x^n-1)) + f\Lambda_r(S-\bar{S}) + \varphi(x^n-1).$$

By consition 4, the degree of the first term in this sum is less than $n - \frac{d-s}{2} < n-t$. By condition 3, $\deg(f\Lambda_r(S-\bar{S})) \leq \frac{d-s}{2} + s + k - 1 = n - \frac{d-s}{2} < n-t$. By condition 4, $\deg(\varphi) + n \leq \deg(f) + s + n - 1$. Consequently $\deg(\varphi) < \deg(f) + s$ and by condition 3, $\deg(\varphi) < \frac{d-s}{2} + s = \frac{d+s}{2} \leq n - \frac{d-s}{2} < n-t$. So, the terms of degrees $n-t,\dots,n-1$ of $\varphi(x^n-1)$ are all zero. Suppose now that there exists $g \in \mathbb{F}[x]$ such that $f = g\Lambda_e$. Then

$$\begin{aligned} f\Lambda_r\bar{S} - \varphi(x^n-1) &= f\Lambda_r(\bar{S}-S) + f\Lambda_r S - \varphi(x^n-1) \\ &= f\Lambda_r(\bar{S}-S) + g\Lambda S - \varphi(x^n-1) \\ &= f\Lambda_r(\bar{S}-S) + g\Omega(x^n-1) - \varphi(x^n-1) \\ &= f\Lambda_r(\bar{S}-S) + (g\Omega - \varphi)(x^n-1). \end{aligned}$$

By condition 4, $\deg(f\Lambda_r\bar{S} - \varphi(x^n-1)) < n - \frac{d-s}{2}$ and as just seen, $\deg(f\Lambda_r(\bar{S}-S)) < n-t$. Consequently, $\varphi = g\Omega$. Now condition 1 and condition 2 imply $g = 1$ and so $\varphi = \Omega$ and $f = \Lambda_e$. □

3. Solving the Symmetric Key Equation

We first approach the case in which only erasures occurred. In this case $\Lambda = \Lambda_r$, $\Lambda_e = 1$, and Ω can be directly derived from the key equation of Theorem 1. Indeed, the polynomial Ω is exactly the sum of those monomials of $\Lambda_r \bar{S}$ of degree at least $n - \frac{d-s}{2}$, divided by the monomial $x^{n-\frac{d-s}{2}}$.

Suppose now the case in which errors and erasures occured simultaneously. The extended Euclidean algorithm applied to the quotient polynomial $\Lambda_r \bar{S}$ and the divisor polynomial $-(x^n - 1)$ gives $\gcd(\Lambda_r \bar{S}, x^n - 1)$ and two polynomials $\lambda(x)$ and $\eta(x)$ satisfying that $\lambda \Lambda_r \bar{S} - \eta(x^n - 1) = \gcd(\Lambda_r \bar{S}, x^n - 1)$. A new remainder r_i and two polynomials $\lambda_i(x)$ and $\eta_i(x)$ such that $\lambda_i \Lambda_r \bar{S} - \eta_i(x^n - 1) = r_i$ are computed at each intermediate step of the Euclidean algorithm, in a way such that the degree of r_i decreases at each step. Truncating at a proper point the Euclidean algorithm we can obtain two polynomials λ_i and η_i satisfying that the degree of $\lambda_i \Lambda_r \bar{S} - \eta_i(x^n - 1)$ is smaller than $n - \frac{d-s}{2}$. The next algorithm is a truncated version of the Euclidean algorithm. It satisfies that, for all $i \geq 0$, $\deg(r_i) \leq \deg(r_{i-1})$ and $\deg(f_i) \geq \deg(f_{i-1})$.

Algorithm 1: Euclidean Algorithm

```
**Initialize**:
        $r_{-2} = \Lambda_r \bar{S}, \quad f_{-2} = 1, \quad \varphi_{-2} = 0,$
        $r_{-1} = -(x^n - 1), \quad f_{-1} = 0, \quad \varphi_{-1} = 1,$
**while deg**$(r_i) \geq n - \frac{d-s}{2}$:
        $q_i = \text{Quotient}(r_{i-2}, r_{i-1})$
        $r_i = \text{Remainder}(r_{i-2}, r_{i-1})$
        $f_i = f_{i-2} - q_i f_{i-1}$
        $\varphi_i = \varphi_{i-2} - q_i \varphi_{i-1}$
**end while**
**Return** $f_i/\mathbf{LC}(f_i), \varphi_i/\mathbf{LC}(f_i)$
or, equivalently, in matrix form,
**Initialize:**
        $\begin{pmatrix} r_{-1} & f_{-1} & \varphi_{-1} \\ r_{-2} & f_{-2} & \varphi_{-2} \end{pmatrix} = \begin{pmatrix} -(x^n - 1) & 0 & 1 \\ \Lambda_r \bar{S} & 1 & 0 \end{pmatrix}$
**while deg**$(r_i) \geq n - \frac{d-s}{2}$:
      $q_i = \mathbf{Quotient}(r_{i-2}, r_{i-1})$
      $\begin{pmatrix} r_i & f_i & \varphi_i \\ r_{i-1} & f_{i-1} & \varphi_{i-1} \end{pmatrix} = \begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{i-1} & f_{i-1} & \varphi_{i-1} \\ r_{i-2} & f_{i-2} & \varphi_{i-2} \end{pmatrix}$
**end while**
**Return** $f_i/\mathbf{LC}(f_i), \varphi_i/\mathbf{LC}(f_i)$
```

For every integer i larger than or equal to -1 consider the matrix $\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \mathring{\tilde{R}}_i & \mathring{\tilde{F}}_i & \mathring{\tilde{\Phi}}_i \end{pmatrix} = \begin{pmatrix} 1/\mathrm{LC}(r_i) & 0 \\ 0 & -\mathrm{LC}(r_i) \end{pmatrix} \begin{pmatrix} r_i & f_i & \varphi_i \\ r_{i-1} & f_{i-1} & \varphi_{i-1} \end{pmatrix}$ It is easy to check that the polynomial $\mathring{R}_i$ is monic. In the algorithm one can replace the update step by the next multiplication.

$$\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \mathring{\tilde{R}}_i & \mathring{\tilde{F}}_i & \mathring{\tilde{\Phi}}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathrm{LC}(\mathring{\tilde{R}}_{i-1} - Q_i \mathring{R}_{i-1})} & 0 \\ 0 & -\mathrm{LC}(\mathring{\tilde{R}}_{i-1} - Q_i \mathring{R}_{i-1}) \end{pmatrix} \begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-1} & \mathring{F}_{i-1} & \mathring{\Phi}_{i-1} \\ \mathring{\tilde{R}}_{i-1} & \mathring{\tilde{F}}_{i-1} & \mathring{\tilde{\Phi}}_{i-1} \end{pmatrix},$$

where the polynomial Q_i is the quotient of the division of $\mathring{\tilde{R}}_{i-1}$ by $\mathring{R}_{i-1}$. Furthermore, if $Q_i = Q_i^{(0)} + Q_i^{(1)} x + \cdots + Q_i^{(l_i)} x^{l_i}$, then

$\begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -Q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -Q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and the update step becomes

$$\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \tilde{\mathring{R}}_i & \tilde{\mathring{F}}_i & \tilde{\mathring{\Phi}}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathrm{LC}(\tilde{\mathring{R}}_{i-1} - Q_i \mathring{R}_{i-1})} & 0 \\ 0 & -\mathrm{LC}(\tilde{\mathring{R}}_{i-1} - Q_i \mathring{R}_{i-1}) \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \cdots$$
$$\cdots \begin{pmatrix} 1 & -Q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-1} & \mathring{F}_{i-1} & \mathring{\Phi}_{i-1} \\ \tilde{\mathring{R}}_{i-1} & \tilde{\mathring{F}}_{i-1} & \tilde{\mathring{\Phi}}_{i-1} \end{pmatrix},$$

One can see that $\mathrm{LC}(\tilde{\mathring{R}}_{i-1} - Q_i \mathring{R}_{i-1})$ and the $Q_i^{(j)}$'s are the leading coefficients of the left-most, top-most polynomials in the previous product of all the previous matrices. This follows from the fact that $\mathring{R}_i$ is monic. Define μ as the (changing) leading coefficients of the left-most, top-most element in the product of all the previous matrices. It follows that

$$\begin{aligned}
\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \tilde{\mathring{R}}_i & \tilde{\mathring{F}}_i & \tilde{\mathring{\Phi}}_i \end{pmatrix} &= \begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-1} & \mathring{F}_{i-1} & \mathring{\Phi}_{i-1} \\ \tilde{\mathring{R}}_{i-1} & \tilde{\mathring{F}}_{i-1} & \tilde{\mathring{\Phi}}_{i-1} \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\
&\quad \begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_{i-1}} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-2} & \mathring{F}_{i-2} & \mathring{\Phi}_{i-2} \\ \tilde{\mathring{R}}_{i-2} & \tilde{\mathring{F}}_{i-2} & \tilde{\mathring{\Phi}}_{i-2} \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix} \\
&\qquad \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_{i-1}} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-2} & \mathring{F}_{i-2} & \mathring{\Phi}_{i-2} \\ \tilde{\mathring{R}}_{i-2} & \tilde{\mathring{F}}_{i-2} & \tilde{\mathring{\Phi}}_{i-2} \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix} \\
&\qquad \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_{i-1}} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix} \\
&\qquad \vdots \\
&\qquad \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_0} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{-1} & \mathring{F}_{-1} & \mathring{\Phi}_{-1} \\ \tilde{\mathring{R}}_{-1} & \tilde{\mathring{F}}_{-1} & \tilde{\mathring{\Phi}}_{-1} \end{pmatrix},
\end{aligned}$$

Let us label the matrices in the previous product:

$$\begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \overbrace{\begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix}}^{M_m} \overbrace{\begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix}}^{M_{m-1}} \cdots \begin{pmatrix} 1 & -\mu x^{l_i - 1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix}$$
$$\vdots$$
$$\begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \overbrace{\begin{pmatrix} 1 & -\mu x^{l_1 - 1} \\ 0 & 1 \end{pmatrix}}^{M_{l_0+3}} \overbrace{\begin{pmatrix} 1 & -\mu x^{l_1} \\ 0 & 1 \end{pmatrix}}^{M_{l_0+2}} \overbrace{\begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix}}^{M_{l_0+1}}$$
$$\overbrace{\begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix}}^{M_{l_0}} \overbrace{\begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix}}^{M_{l_0-1}} \cdots \overbrace{\begin{pmatrix} 1 & -\mu x^{l_0 - 1} \\ 0 & 1 \end{pmatrix}}^{M_1} \overbrace{\begin{pmatrix} 1 & -\mu x^{l_0} \\ 0 & 1 \end{pmatrix}}^{M_0} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathring{R}_{-1} & \mathring{F}_{-1} & \mathring{\Phi}_{-1} \\ \tilde{\mathring{R}}_{-1} & \tilde{\mathring{F}}_{-1} & \tilde{\mathring{\Phi}}_{-1} \end{pmatrix}$$

Now, we define

$$\begin{pmatrix} R_{-1} & F_{-1} & \Phi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Phi}_{-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \overset{\circ}{R}_{-1} & \overset{\circ}{F}_{-1} & \overset{\circ}{\Phi}_{-1} \\ \overset{\circ}{\tilde{R}}_{-1} & \overset{\circ}{\tilde{F}}_{-1} & \overset{\circ}{\tilde{\Phi}}_{-1} \end{pmatrix} = \begin{pmatrix} \Lambda_r \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

$$\begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix} = M_i \cdot M_{i-1} \cdot \cdots \cdot M_0 \cdot \begin{pmatrix} R_{-1} & F_{-1} & \Phi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Phi}_{-1} \end{pmatrix}$$

Lets us see now that, for all $i \leq m$, the polynomials $\tilde{R}_i$ and F_i are monic. Indeed, $\tilde{R}_{-1} = x^n - 1$ is monic, and it follows by induction and by the definition of the matrices M_i, that $\tilde{R}_i$ is monic for all i. Now, all the matrices M_i have determinant equal to 1. This implies that $R_i\tilde{F}_i - F_i\tilde{R}_i$ is constant for all i and it equals $-(x^n - 1)$. In particular, since $\mathrm{LC}(R_i\tilde{F}_i - F_i\tilde{R}_i) = -\mathrm{LC}(F_i)\mathrm{LC}(\tilde{R}_i) = -\mathrm{LC}(F_i)$, we deduce that for every i, the polynomial F_i is monic.

Algorithm 2 computes the matrices $\begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$ until $\deg(R_i) < n - \frac{d-s}{2}$.

Algorithm 2: Single Coefficient Euclidean Algorithm.

```
Initialize:
    $\begin{pmatrix} R_{-1} & F_{-1} & \Phi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Phi}_{-1} \end{pmatrix} = \begin{pmatrix} \Lambda_r \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$
while deg$(R_i) \geq n - \frac{d-s}{2}$:
    $\mu =$ LC$(R_i)$
    $p =$ deg$(R_i) -$ deg$(\tilde{R}_i)$
    if $p \geq 0$ then
        $\begin{pmatrix} R_{i+1} & F_{i+1} & \Phi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$
    else
        $\begin{pmatrix} R_{i+1} & F_{i+1} & \Phi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$
    end if
end while
Return $F_i, \Phi_i$
```

Due to the fact that the polynomials $\tilde{R}_i$ are monic, after each step with a negative value of p the new updated value p coincides with the previous one but with opposite sign and so happens for μ. Taking this into account we join each step with a negative value of p with the next step. We obtain

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

This adjustment keeps F_i, Φ_i unaltered. It can be stated as follows

At this point we observe that we only need to keep the polynomials R_i (and $\tilde{R}_i$) because we need their leading coefficients (the μ_i's). The next lemma proves that these leading coefficients can be obtained independently of the polynomials R_i. This allows the computation of the polynomials F_i, Φ_i iteratively while dispensing with the polynomials R_i.

Lemma 4. *$LC(R_i) = LC(F_i\Lambda_r\bar{S})$*

Proof. The result is obvious for $i = -1$. Since we joined two steps, before Algorithm 3, the degree of the remainder $R_i = F_i\Lambda_r\bar{S} - \Phi_i(x^n - 1) = F_i\Lambda_r\bar{S} - x^n\Phi_i + \Phi_i$ is at most $n-1$ for every $i \geq 1$. Consequently all terms of $x^n\Phi_i$ cancel with terms of $F_i\Lambda_r\bar{S}$ and R_i must have leading term equal to either a term of Φ_i or a term of $F_i\Lambda_r\bar{S}$ or a sum of a term of Φ_i and a term of $F_i\Lambda_r\bar{S}$.

On the other hand, the algorithm computes $\mathrm{LC}(R_i)$ only while $\deg(R_i) \geq n - \frac{d-s}{2}$. In particular, $2\deg(R_i) = 2n - d + s \geq n + s$. Leu us show that in this case the degree of the leading term of R_i is strictly larger than the degree of Φ_i. Indeed, since all the matrices M_i in the algorithm have determinant equal to 1, this implies that $\deg(\Phi_i) = \deg(\Lambda_r\bar{S}) - \deg(\tilde{R}_i) \leq n + s - \deg(\tilde{R}_i) < 2\deg(R_i) - \deg(R_i) = \deg(R_i)$. $\square$

Algorithm 3: Refactored Single Coefficient Euclidean Algorithm

```
Initialize:
    $\begin{pmatrix} R_{-1} & F_{-1} & \Phi_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{\Phi}_{-1} \end{pmatrix} = \begin{pmatrix} \Lambda_r\bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$
while deg$(R_i) \geq n - \frac{d-s}{2}$:
    $\mu = \mathbf{LC}(R_i)$
    $p = \mathbf{deg}(R_i) - \mathbf{deg}(\tilde{R}_i)$
    if $p \geq 0$ or $\mu = 0$ then
        $\begin{pmatrix} R_{i+1} & F_{i+1} & \Phi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$
    else
        $\begin{pmatrix} R_{i+1} & F_{i+1} & \Phi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$
    end if
end while
Return $F_i$, $\Phi_i$
```

We transform now Algorithm 3 in a way such that isntead of keeping the remainders we keep their degrees. For this we use the values $d_i, \tilde{d}_i$ satisfying, at each step, that $d_i \geq \deg(R_i)$, $\tilde{d}_i = \deg(\tilde{R}_i)$.

Algorithm 4 is exactly the Berlekamp–Massey algorithm applied to the recurrence $\sum_{j=0}^{t} \Lambda_j e(\alpha^{i+j-1}) = 0$ for all $i > 0$. This linear recurrence is a consequence of the equality $\frac{S}{x^n-1} = \frac{1}{x}\left(e(1) + \frac{e(\alpha)}{x} + \frac{e(\alpha^2)}{x^2} + \cdots\right)$ and the fact that $\Lambda\frac{S}{x^n-1}$ is a polynomial and, hence, its terms of negative order in its expression as a Laurent series in $1/x$ are all zero.

Algorithm 4: Berlekamp-Massey Algorithm

```
Initialize:
    $d_{-1} = s + \mathbf{deg}(\bar{S})$
    $\tilde{d}_{-1} = n$
    $\begin{pmatrix} F_{-1} & \Phi_{-1} \\ \tilde{F}_{-1} & \tilde{\Phi}_{-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
while $d_i \geq n - \frac{d-s}{2}$:
    $\mu = \mathbf{Coefficient}(F_i \Lambda_r \bar{S}, d_i)$
    $p = d_i - \tilde{d}_i$
    if $p \geq 0$ or $\mu = 0$ then
        $\begin{pmatrix} F_{i+1} & \Phi_{i+1} \\ \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} F_i & \Phi_i \\ \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$
        $d_{i+1} = d_i - 1$
        $\tilde{d}_{i+1} = \tilde{d}_i$
    else
        $\begin{pmatrix} F_{i+1} & \Phi_{i+1} \\ \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} F_i & \Phi_i \\ \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$
        $d_{i+1} = \tilde{d}_i - 1$
        $\tilde{d}_{i+1} = d_i$
    end if
end while
Return $F_i, \Phi_i$
```

4. Conclusions

By working with error/erasure locator polynomials whose roots correspond to the error positions rather than to their inverses and with an evaluator polynomial that gives the error values when we evaluate it at the error positions instead of evaluating it at the inverses of the error positions we get to a symmetric key equation for Reed–Solomon codes. We showed that the symmetric key equation can be solved by an adapted Euclidean algorithm whose steps can be refined leading naturally to the Berlekamp–Massey algorithm.

Author Contributions: The authors contributed equally to the theoretical framing and algorithms and the corresponding author took principle responsibility for writing the article.

Funding: The first author was partly supported by the Catalan Government under grant 2017 SGR 00705, by the Spanish Ministry of Economy and Competitivity under grant TIN2016-80250-R, and by Universitat Rovira i Virgili under grant OPEN2019.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. McEliece, R.J.; Sarwate, D.V. On sharing secrets and Reed-Solomon codes. *Commun. ACM* **1981**, *24*, 583–584. [CrossRef]
2. Dimakis, A.G.; Ramchandran, K.; Wu, Y.; Suh, C. A Survey on Network Codes for Distributed Storage. *Proc. IEEE* **2011**, *99*, 476–489. [CrossRef]

3. Tamo, I.; Ye, M.; Barg, A. The repair problem for Reed-Solomon codes: Optimal repair of single and multiple erasures with almost optimal node size. *IEEE Trans. Inf. Theory* **2019**, *65*, 2673–2695. [CrossRef]
4. Tajeddine, R.; Gnilke, O.W.; Karpuk, D.; Freij-Hollanti, R.; Hollanti, C. Private information retrieval from coded storage systems with colluding, Byzantine, and unresponsive servers. *IEEE Trans. Inf. Theory* **2019**, *65*, 3898–3906. [CrossRef]
5. Kiayias, A.; Yung, M. Cryptographic hardness based on the decoding of Reed-Solomon codes. *IEEE Trans. Inf. Theory* **2008**, *54*, 2752–2769. [CrossRef]
6. Berlekamp, E.R. *Algebraic Coding Theory*; McGraw-Hill Book Co.: New York, NY, USA, 1968; pp. xiv+466.
7. Massey, J.L. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* **1969**, *15*, 122–127. [CrossRef]
8. Sugiyama, Y.; Kasahara, M.; Hirasawa, S.; Namekawa, T. A method for solving key equation for decoding Goppa codes. *Inf. Control* **1975**, *27*, 87–99. [CrossRef]
9. Dornstetter, J.L. On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Trans. Inf. Theory* **1987**, *33*, 428–431. [CrossRef]
10. Heydtmann, A.E.; Jensen, J.M. On the equivalence of the Berlekamp-Massey and the Euclidean algorithms for decoding. *IEEE Trans. Inf. Theory* **2000**, *46*, 2614–2624.
11. Mateer, T.D. On the equivalence of the Berlekamp-Massey and the Euclidean algorithms for algebraic decoding. In Proceedings of the 12th Canadian Workshop on Information Theory (CWIT), Kelowna, BC, Canada, 17–20 May 2011; pp. 139–142.
12. Ilani, I. Berlekamp–Massey Algorithm: Euclid in Disguise. In Proceedings of the 2018 IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE), Eilat, Israel, 12–14 December 2018; pp. 1–5. [CrossRef]
13. Forney, G.D., Jr. On decoding BCH codes. *IEEE Trans. Inf. Theory* **1965**, *11*, 549–557. [CrossRef]