A new approach to the key equation and to the Berlekamp-Massey algorithm

M. Bras-Amorós[1], M. E. O'Sullivan[2], M. Pujol[1]

[1] *Universitat Rovira i Virgili, Tarragona, Catalonia, Spain, {maria.bras,marta.pujol}@urv.cat*
[2] *San Diego State University, California, USA, mosulliv@sciences.sdsu.edu*

The two primary decoding algorithms for Reed-Solomon codes are the Berlekamp-Massey algorithm [5] and the Sugiyama et al. adaptation of the Euclidean algorithm [7], both designed to solve Berlekamp's key equation [1]. Their connections are analyzed in [2, 4, 6]. We present a new version of the key equation for errors and erasures, more natural somehow, and a way to use the Euclidean algorithm to solve it. A straightforward reorganization of the algorithm yields the Berlekamp-Massey algorithm.

Settings on Reed-Solomon codes Let $\mathbb{F}$ be a finite field of size q and let α be a primitive element in $\mathbb{F}$. Let $n=q-1$. We identify the vector $u=(u_0,\dots,u_{n-1})$ 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 defined as the cyclic code with generator polynomial $(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k})$, The dual Reed-Solomon code $C(k)$ of dimension k is the cyclic code with generator polynomial $(x-\alpha^{n-(k+1)})(x-\alpha^{n-(k+2)})\cdots(x-\alpha)(x-1)$.

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 which we denote by $c\mapsto c^*$. It takes $C(k)$ to $C^*(k)$. The codeword c^* can be defined either as $iG^*(k)\in C^*(k)$ where i is the information vector of dimension k such that $c=iG(k)\in C(k)$ or componentwise as $c^*=(c_0,\alpha^{-1}c_1,\alpha^{-2}c_2,\dots,\alpha c_{n-1})$ where $c=(c_0,c_1,\dots,c_{n-1})$. Then, $(c_0^*,\alpha c_1^*,\alpha^2c_2^*,\dots,\alpha^{n-1}c_{n-1}^*)$. In particular, $c(\alpha^i)=c^*(\alpha^{i+1})$.

A decoding algorithm for a primal Reed-Solomon code may be used to decode a dual Reed-Solomon code by first applying the bijection $*$ to the received vector u. If u differs from a codeword $c\in C(k)$ by an error vector e of weight t, then u^* differs from the codeword $c^*\in C^*(k)$ by the error vector e^* of weight t. If the primal Reed-Solomon decoding algorithm can decode u^* to obtain c^* and e^* then, transforming by the inverse of $*$ we may obtain c and e. Conversely, a decoding algorithm for a dual Reed-Solomon code may be used to decode a primal Reed-Solomon code by applying the inverse of $*$, decoding, and then applying $*$.

Decoding for errors and erasures Suppose that $c\in C(k)$ is transmitted and that errors occurred at t different positions and that other s positions were erased, with

$2t+s<d$. Suppose that u is the received word once the erased positions are put to 0 and that $e=u-c$. Define the *erasure locator polynomial* as $\Lambda_r=\prod_{i:c_i \text{was erased}}(x-\alpha^i)$ and the *error locator polynomial* as $\Lambda_e=\prod_{i:e_i\neq 0, c_i \text{not erased}}(x-\alpha^i)$. We will use Λ for the product $\Lambda_r\Lambda_e$. Notice that Λ_r is known from the received word, while Λ_e is not. Define the error evaluator as $\Omega=\sum_{\substack{i:e_i\neq 0\\ \text{or } c_i \text{ erased}}} e_i\prod_{\substack{j:e_j\neq 0 \text{ or } c_j \text{ erased,}\\ \text{and } j\neq i}}(x-\alpha^i)$. The error positions can be identified by $\Lambda_e(\alpha^i)=0$ and the error values, as well as the erased values, can be derived from an analogue of the Forney formula [3], $e_i=\frac{\Omega(\alpha^i)}{\Lambda'(\alpha^i)}$.

The *syndrome polynomial* is defined as $S=e(\alpha^{n-1})+e(\alpha^{n-2})x+\cdots+e(\alpha)x^{n-2}+e(1)x^{n-1}$. It can be proved that $\Omega(x^n-1)=\Lambda S$.The general term of S is $e(\alpha^{n-1-i})x^i$, but from a received word we only know $e(1)=u(1),\ldots,e(\alpha^{n-k-1})=u(\alpha^{n-k-1})$. Define $\bar{S}=e(\alpha^{n-k-1})x^k+e(\alpha^{n-k-2})x^{k+1}+\cdots+e(1)x^{n-1}$. The polynomial $\Omega(x^n-1)-\Lambda\bar{S}=\Lambda(S-\bar{S})$ has degree at most $t+s+k-1<\frac{d-s}{2}+s+n-d=n-\frac{d-s}{2}$. Next theorem provides an alternative key equation for dual Reed-Solomon codes.

Theorem 1. *If s erasures and at most $\lfloor\frac{d-s-1}{2}\rfloor$ errors occurred, then Λ_e and Ω are the unique polynomials f and φ satisfying the following properties. 1. $deg(f\Lambda_r\bar{S}-\varphi(x^n-1))<n-\frac{d-s}{2}$; 2. $deg(f)\leq\frac{d-s}{2}$; 3. f,φ are coprime; 4. f is monic*

Suppose first that only erasures occurred. Then $\Lambda=\Lambda_r$, $\Lambda_e=1$, and Ω can be directly derived from this inequality. Indeed, Ω is the sum of monomials in $\Lambda_r\bar{S}$ with degrees at least $n-\frac{d-s}{2}$, divided by $x^{n-\frac{d-s}{2}}$.

Suppose that a combination of errors and erasures occured. The extended Euclidean algorithm applied to $\Lambda_r\bar{S}$ and $-(x^n-1)$ computes not only $\gcd(\Lambda_r\bar{S},x^n-1)$ but also two polynomials $\lambda(x)$ and $\eta(x)$ such that $\lambda\Lambda_r\bar{S}-\eta(x^n-1)=\gcd(\Lambda_r\bar{S},x^n-1)$. At each intermediate step a new remainder r_i is computed, with decreased degree, together with two intermediate polynomials $\lambda_i(x)$ and $\eta_i(x)$ such that $\lambda_i\Lambda_r\bar{S}-\eta_i(x^n-1)=r_i$. Truncating this algorithm at a proper point we can get a pair of polynomials λ_i and η_i such that $\lambda_i\Lambda_r\bar{S}-\eta_i(x^n-1)$ has degree as small as desired (in particular, smaller than $n-\frac{d-s}{2}$). Algorithm 1 is the truncated 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

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 $\mathbf{deg}(r_i)\geq n-\frac{d-s}{2}$**:**

$q_i=\mathbf{Quotient}(r_{i-2},r_{i-1})$

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

end while

Return $f_i/\mathbf{LC}(f_i)$, $\varphi_i/\mathbf{LC}(f_i)$

Theorem 2. *If a codeword $c \in C(k)$ is transmitted and s erasures and t errors occur with $2t+s<d$ then the algorithm outputs Λ_e and Ω.*

For all $i \geq -1$ consider the matrices $\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \tilde{\mathring{R}}_i & \tilde{\mathring{F}}_i & \tilde{\mathring{\Phi}}_i \end{pmatrix} = \begin{pmatrix} 1/\mathrm{LC}(r_i) & 0 \\ 0 & -\mathrm{LC}(r_i) \end{pmatrix} \begin{pmatrix} r_i & f_i & \varphi_i \\ r_{i-1} & f_{i-1} & \varphi_{i-1} \end{pmatrix}$
Notice that $\mathring{R}_i$ is monic. The update step in the algorithm can be replaced by
$\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \tilde{\mathring{R}}_i & \tilde{\mathring{F}}_i & \tilde{\mathring{\Phi}}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathrm{LC}(\tilde{\mathring{R}}_{i-1} - Q_i \mathring{R}_{i-1})} & 0 \\ 0 & -\mathrm{LC}(\tilde{\mathring{R}}_{i-1} - Q_i \mathring{R}_{i-1}) \end{pmatrix} \begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-1} & \mathring{F}_{i-1} & \mathring{\Phi}_{i-1} \\ \tilde{\mathring{R}}_{i-1} & \tilde{\mathring{F}}_{i-1} & \tilde{\mathring{\Phi}}_{i-1} \end{pmatrix}$,
where Q_i is the quotient of $\tilde{\mathring{R}}_{i-1}$ by $\mathring{R}_{i-1}$. Moreover, 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

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

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

It can be easily shown that $\mathrm{LC}(\tilde{\mathring{R}}_{i-1} - Q_i \mathring{R}_{i-1})$ as well as all the $Q_i^{(j)}$'s, are the LC of the left-most, top-most element in the previous product of all the previous matrices. This is because $\mathring{R}_i$ is monic. If we define μ to be the (changing) LC of the left-most, top-most element in the product of all the previous matrices, then $\begin{pmatrix} \mathring{R}_i & \mathring{F}_i & \mathring{\Phi}_i \\ \tilde{\mathring{R}}_i & \tilde{\mathring{F}}_i & \tilde{\mathring{\Phi}}_i \end{pmatrix}$ equals

$$\begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathring{R}_{i-1} & \mathring{F}_{i-1} & \mathring{\Phi}_{i-1} \\ \tilde{\mathring{R}}_{i-1} & \tilde{\mathring{F}}_{i-1} & \tilde{\mathring{\Phi}}_{i-1} \end{pmatrix} =$$

$$\begin{pmatrix} \frac{1}{\mu} & 0 \\ 0 & -\mu \end{pmatrix} \overbrace{\begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix}}^{M_m} \overbrace{\begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix}}^{M_{m-1}} \cdots \begin{pmatrix} 1 & -\mu x^{l_i-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\mu x^{l_i} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix}$$

$$\vdots$$

$$\overbrace{\begin{pmatrix} 1 & -\mu \\ 0 & 1 \end{pmatrix}}^{M_{l_0}} \overbrace{\begin{pmatrix} 1 & -\mu x \\ 0 & 1 \end{pmatrix}}^{M_{l_0-1}} \cdots \overbrace{\begin{pmatrix} 1 & -\mu x^{l_0-1} \\ 0 & 1 \end{pmatrix}}^{M_1} \overbrace{\begin{pmatrix} 1 & -\mu x^{l_0} \\ 0 & 1 \end{pmatrix}}^{M_0} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathring{R}_{-1} & \mathring{F}_{-1} & \mathring{\Phi}_{-1} \\ \tilde{\mathring{R}}_{-1} & \tilde{\mathring{F}}_{-1} & \tilde{\mathring{\Phi}}_{-1} \end{pmatrix}$$

Let us define now,

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

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

One can prove that now $\tilde{R}_i$ and F_i are monic for all $i \leq m$. 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

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 $\mathbf{deg}(R_i) \geq n - \frac{d-s}{2}$**:**

$\mu = \mathbf{LC}(R_i)$
$p = \mathbf{deg}(R_i) - \mathbf{deg}(\tilde{R}_i)$
if $p \geq 0$ **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**,** Φ_i

After each step corresponding to $p < 0$ the new p is exactly the previous one with opposite sign and so is μ. This is because the polynomials $\tilde{R}_i$ are monic. So, we can join each step corresponding to $p < 0$ with the next one and get that, in this case, $\begin{pmatrix} R_{i+1} & F_{i+1} & \Phi_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{\Phi}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & \mu x^{-p} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & \Phi_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{\Phi}_i \end{pmatrix}$

This modification does not alter the output F_i, Φ_i. Furthermore, the only reason to keep the polynomials R_i (and $\tilde{R}_i$) is that we need to compute their leading coefficients (the μ_i's). One can show that $\mathrm{LC}(R_i) = \mathrm{LC}(F_i \Lambda_r \bar{S})$, and so these leading coefficients may be obtained without reference to the polynomials R_i. This allows us to compute the F_i, Φ_i iteratively and dispense with the polynomials R_i.

Algorithm 2 can be transformed in a way such that the remainders are not kept but their degrees. We use $d_i, \tilde{d}_i$ which satisfy at each step $d_i \geq \deg(R_i)$, $\tilde{d}_i = \deg(\tilde{R}_i)$.

Algorithm 3

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, Φ_i

Algorithm 3 is exactly 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 and thus having no terms of negative order in its expression as a Laurent series in $1/x$, and from the equality $\frac{S}{x^n-1} = \frac{1}{x}\left(e(1) + \frac{e(\alpha)}{x} + \frac{e(\alpha^2)}{x^2} + \cdots\right)$.

References

[1] Elwyn R. Berlekamp. *Algebraic coding theory*. McGraw-Hill Book Co., New York, 1968.

[2] Jean-Louis Dornstetter. On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Trans. Inform. Theory*, 33(3):428–431, 1987.

[3] G. D. Forney, Jr. On decoding BCH codes. *IEEE Trans. Inform. Theory*, IT-11:549–557, 1965.

[4] Agnes E. Heydtmann and Jørn M. Jensen. On the equivalence of the Berlekamp-Massey and the Euclidean algorithms for decoding. *IEEE Trans. Inform. Theory*, 46(7):2614–2624, 2000.

[5] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Information Theory*, IT-15:122–127, 1969.

[6] T. D. Mateer. On the equivalence of the Berlekamp-Massey and the Euclidean algorithms for algebraic decoding. In *12th Canadian Workshop on Inf. Theory (CWIT)* pp. 139–142, 2011.

[7] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding Goppa codes. *Information and Control*, 27:87–99, 1975.