Reed Solomon

Gordon Cichon

December 11, 2016

1 Finite Fields

The finite field with 4 elements F_4 was presented in the lecture. In order to develop a simple example for a systematic Reed-Solomon code we need a field with more field elements. So, we are looking at F_8.

$F_8 = F_{2^3}$ is based on $F_2 = (\{0, 1\}, \oplus, \cdot)$, with exclusive OR as addition, and AND as multiplication.

1.1 Field Elements

F_8 has the following field elements:

Short	Polynomial
0	0
1	1
2	x
3	$x + 1$
4	x^2
5	$x^2 + 1$
6	$x^2 + x$
7	$x^2 + x + 1$

1.2 Addition Table

Two polynomials are added componentwise.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Note each element is its own inverse. This is due to the base on F_2. Each row and each column contains each field element exactly once. The operation on the short representation is XOR.

1.3 Generator Polynomial

1.3.1 Irreducible Polynomials

reducible polynomials on F_4 are:

$$x \cdot x = x^2 \tag{1}$$
$$x \cdot (x+1) = x^2 + x \tag{2}$$
$$(x+1) \cdot (x+1) = x^2 + 1 \tag{3}$$
$$\tag{4}$$

irreducible polynomial in F_4: $x^2 + x + 1$
reducible polynomials in F_8:

$$x \cdot x \cdot x = x^3 = 8 \tag{5}$$
$$x \cdot x \cdot (x+1) = x^3 + x^2 = 12 \tag{6}$$
$$x \cdot (x+1) \cdot (x+1) = x^3 + x = 10 \tag{7}$$
$$(x+1) \cdot (x+1) \cdot (x+1) = x^3 + x^2 + x + 1 = 15 \tag{8}$$
$$x \cdot (x^2 + x + 1) = x^3 + x^2 + x = 14 \tag{9}$$
$$(x+1) \cdot (x^2 + x + 1) = x^3 + 1 = 9 \tag{10}$$
$$\tag{11}$$

So, there are two choices for generator polynomials:

$$x^3 + x + 1 = 11 \tag{12}$$
$$x^3 + x^2 + 1 = 13 \tag{13}$$
$$\tag{14}$$

In the following, we chose (field) generator polynomial $g(x) = x^3 + x^2 + 1$. This means, $g(x) = x^3 + x^2 + 1 = 0$.

Homework Try the same work using the other polynomial.

1.4 Multiplication Table

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	5	7	1	3
3	0	3	6	5	1	2	7	4
4	0	4	5	1	7	3	2	6
5	0	5	7	2	3	6	4	1
6	0	6	1	7	2	4	3	5
7	0	7	3	4	6	1	5	2

Note 0 is the dominant element. All elements multiplied with 0 get 0. 1 is the neutral element. All elements stay the same multiplied with 1. Each row and each column of the table contains each field element exactly once.

x	0	1	2	3	4	5	6	7
x^{-1}	-	1	6	4	3	7	2	5

```
2 = x
2*2 = x*x = x^2 = 4
2*3 = x*(x+1) = x^2+x = 6
2*4 = x*(x^2) = x^3 = -(x^2+1) = x^2+1 = 5
2*5 = x*(x^2+1) = x^3+x = x^2+1+x = 7
2*6 = x*(x^2+x) = x^3+x^2 = x^2+1+x^2 = 1
2*7 = x*(x^2+x+1) = x^3+x^2+x = x^2+1+x^2+x = x+1 = 3
3*3 = (x+1)*(x+1) = x^2+2x+1 = x^2+1 = 5
3*4 = (x+1)*x^2 = x^3+x^2 = x^2+1+x^2 = 1
3*5 = (x+1)(x^2+1) = x^3+x^2+x+1 = x^2+1+x^2+x+1 = x = 2
3*6 = (x+1)(x^2+x) = x^3+x^2+x^2+x = x^2+1+x^2+x^2+x = x^2+x+1 = 7
3*7 = (x+1)(x^2+x+1) = x^3+x^2+x+x^2+x+1 = x^3+1 = x^2+1+1 = x^2 = 4
4*4 = x^2*x^2 = x^4 = x*x^3 = x*(x^2+1) = 2*5 = 7
4*5 = x^2*(x^2+1) = x^4+x^2 = 7+4 = 3
4*6 = x^2*(x^2+x) = x^4+x^3 = 7+5 = 2
4*7 = x^2*(x^2+x+1) = x^4+x^3+x^2 = 2+4 = 6
5*5 = (x^2+1)(x^2+1) = x^4+2x^2+1 = x^4+1 = 7+1 = 6
5*6 = (x^2+1)(x^2+x) = x^4+x^3+x^2+x = 7+5+4+2 = 4
5*7 = (x^2+1)(x^2+x+1) = x^4+x^3+x^2+x^2+x+1 = 7+5+2+1 = 1
6*6 = (x^2+x)(x^2+x) = x^2(x+1)(x+1) = x^2(x^2+1) = x^4+x^2 = 7+4 = 3
6*7 = (x^2+x)(x^2+x+1) = x^4+x^3+x^2+x^3+x^2+x = x^4+x = 7+2 = 5
7*7 = (x^2+x+1)(x^2+x+1) = x^4+x^3+x^2+x^3+x^2+x+x^2+x+1 = x^4+x^2+1 = 7+4+1 = 2
```

1.5 Primitive Element

Pick a primitive element α with $\forall x : \exists j : x = \alpha^j$.

This allows for the definition of Galois logarithm. The argument of the logarithm is a field element, while the result is an ordinary integer value.

Since $n - 1$ is prime, we may chose any field element besides 0 or 1 as primitive element. A common choice is x, or as number 2.

Note This also provides for a fast method of computing Galois multiplication without polynomial division. $x \cdot y = \alpha^{\log x + \log y}$.

We pick $\alpha = x = 2$

x	0	1	2	3	4	5	6	7
$\log x$	-	0	1	5	2	3	6	4

The inverse function to log is $\exp x := \alpha^x$:

x	0	1	2	3	4	5	6	7
$\exp x$	1	2	4	5	7	3	6	1

2 Reed-Solomon Code

In this sheet, we pick an example code. The code word length $n = 7$ has to be at most one less than the number of field elements. $k = 3$ is the number of information words. Consequently, we have a minimum distance $d = n - k = 4$, and we can correct up to $t = 2$ unknown errors.

homework This looks like a lot of redundancy. What is the rate of the code? Compare that with codes based on F_{256} with $t = 8$ and $t = 32$.

2.1 Generator Polynomial

$$g(x) := \prod_{j=1}^{n-k} \left(x - \alpha^j\right) \tag{15}$$

Example:

$$\begin{aligned}
g(x) &= (x-2)(x-4)(x-5)(x-7) \\
&= x^4 \\
&+ (2+4+5+7) \cdot x^3 \\
&+ (2\cdot 4 + 2\cdot 5 + 2\cdot 7 + 4\cdot 5 + 4\cdot 7 + 5\cdot 7) \cdot x^2 \\
&+ (2\cdot 4\cdot 5 + 2\cdot 4\cdot 7 + 2\cdot 5\cdot 7 + 4\cdot 5\cdot 7) \cdot x \\
&+ (2\cdot 4\cdot 5\cdot 7) \\
&= x^4 \\
&+ (6+2) \cdot x^3 \\
&+ (5+7+3+3+6+1) \cdot x^2 \\
&+ (5\cdot 5 + 5\cdot 7 + 7\cdot 7 + 3\cdot 7) \cdot x \\
&+ (5\cdot 1) \\
&= x^4 \\
&+ 4 \cdot x^3 \\
&+ 5 \cdot x^2 \\
&+ (6+1+2+4) \cdot x \\
&+ 5 \\
&= x^4 + 4\cdot x^3 + 5 \cdot x^2 + x + 5
\end{aligned} \tag{16}$$

In our example, we pick the message of

$$m(x) = 7x^2 + 3x + 1 = \begin{pmatrix} 7 & 3 & 1 \end{pmatrix} \tag{17}$$

Note: Usually, the message vector is written with the lowest coefficient first. Here, it is an exception. The first vector element gets the highest coefficient in the polynomial. This is because so we can compute ploynomial division with feedback shift registers in message arrival order.

For encoding the message, we multiply it with x^4 and compute the reminder of dividing it by the (code) generator polynomial. This excess remainder has

to be subtracted from the sent codeword, such that the code word becomes a multiple of $g(x)$.

$$s_0(x) = 7x^6 + 3x^5 + 1x^4 \tag{18}$$

```
s_0 / g =
 (7*x^6 + 3*x^5 + 1*x^4) / (x^4 + 4*x^3 + 5*x^2 + x + 5) = 7*x^2 + 5*x + 3
-(7*x^6 + 6*x^5 + 1*x^4 + 7*x^3 + x^2)
--------------------------------------
          5*x^5          + 7*x^3 + x^2
        -(5*x^5 + 3*x^4 + 6*x^3 + 5*x^2 + 6*x)
        --------------------------------------
                  3*x^4 +   x^3 + 4*x^2 + 6*x
                -(3*x^4 + 1*x^3 + 2*x^2 + 3*x + 2)
                ----------------------------------
                                  6*x^2 + 5*x + 2
```

So, the sent word is:

$$s = s_0 + \text{rem} = 7 * x^6 + 3 * x^5 + x^4 + 6 * x^2 + 5 * x + 2 \tag{19}$$

$$= \begin{pmatrix} 7 & 3 & 1 & 0 & 6 & 5 & 2 \end{pmatrix} \tag{20}$$

During the transmission of the message, there will be an error.

$$r(x) = s(x) + e(x) \tag{21}$$

Example:

$$s = \begin{pmatrix} 7 & 3 & 1 & 0 & 6 & 5 & 2 \end{pmatrix} \tag{22}$$

$$r = \begin{pmatrix} 1 & 3 & 1 & 4 & 6 & 5 & 2 \end{pmatrix} \tag{23}$$

$$e = \begin{pmatrix} 6 & 0 & 0 & 4 & 0 & 0 & 0 \end{pmatrix} \tag{24}$$

$$e(x) = 6x^6 + 4x^3 \tag{25}$$

Let the number of errors be ν at positions $i_1 \dots i_\nu$
We defined the error position as:

$$X_k = \alpha^{i_k} \tag{26}$$

and the error values as:

$$Y_k = e_{i_k} \tag{27}$$

With that defintion, we get:

$$e(x) = \sum_{k=1}^{\nu} e_{i_k} \alpha^{i_k} = \sum_{k=1}^{\nu} Y_k X_k \tag{28}$$

Example:

$$\nu = 2 \tag{29}$$
$$i_1 = 3 \tag{30}$$
$$i_2 = 6 \tag{31}$$
$$X_1 = 5 \tag{32}$$
$$X_2 = 6 \tag{33}$$
$$Y_1 = 4 \tag{34}$$
$$Y_2 = 6 \tag{35}$$

3 Decoding the Message

3.1 Syndrome Calculation

It is now possible to regard the received message as a polynomial and evaluate that polynomial at the zeros of the generator polynomial. By design of the generator polynomial, the sent message evaluates to zero at all zeros of the generator polynomial, since it is a multiple of the generator polynumial.

$$S_j = r(\alpha^j) = s(\alpha^j) + e(\alpha^j) = e(\alpha^j) \tag{36}$$

with $j = 1, 2, \ldots, n-k$

From this, we can infer:

$$S_j = \sum_{k=1}^{\nu} e_{i_k} (\alpha^j)^{i_k} \tag{37}$$
$$= \sum_{k=1}^{\nu} Y_k X_k^j \tag{38}$$

Example:

$$S_1 = r(2) = 0 \tag{39}$$
$$S_2 = r(4) = 5 \tag{40}$$
$$S_3 = r(5) = 2 \tag{41}$$
$$S_4 = r(7) = 5 \tag{42}$$
$$\tag{43}$$

3.2 Error Locator Polynomial

Definition:

$$\Lambda(x) := \prod_{k=1}^{\nu} (1 - x \cdot X_k) \tag{44}$$
$$= 1 + \Lambda_1 x^1 + \Lambda_2 x^2 + \cdots + \Lambda_\nu x^\nu \tag{45}$$

Note: The error locator polynomial has zeros at the reciprocals of X_k.

$$\Lambda(X_k^{-1}) = 0 \tag{46}$$
$$= 1 + \Lambda_1 X_k^{-1} + \Lambda_2 X_k^{-2} + \cdots + \Lambda_\nu X_k^{-\nu} \tag{47}$$

Example:

$$X_1 = 5 \tag{48}$$
$$X_2 = 6 \tag{49}$$
$$X_1^{-1} = 7 \tag{50}$$
$$X_2^{-1} = 2 \tag{51}$$

$$\Lambda(x) = (1 - 5 \cdot x)(1 - 6 \cdot x) = 1 + 3x + 4x^2 \tag{52}$$

Does such polynomial exist? Yes, but we don't know its parameters (yet). How to determine the parameters from the Syndromes?

We know that $\Lambda(X_k^{-1}) = 0$. We can multiply that with $Y_k X_k^{j+\nu}$ for any j:

$$Y_k X_k^{j+\nu} \cdot \Lambda(X_k^{-1}) = 0 \tag{53}$$

we get:

$$Y_k X_k^{j+\nu} + Y_k X_k^{j+\nu} \Lambda_1 X_k^{-1} + Y_k X_k^{j+\nu} \Lambda_2 X_k^{-2} + ... + Y_k X_k^{j+\nu} \Lambda_\nu X_k^{-\nu} = 0 \tag{54}$$

or simplified:

$$Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu-1} + \Lambda_2 Y_k X_k^{j+\nu-2} + ... + \Lambda_\nu Y_k X_k^{j} = 0 \tag{55}$$

now, we sum this over all errors k:

$$\begin{aligned} &\sum_{k=1}^{\nu} Y_k X_k^{j+\nu} \\ &+\Lambda_1 \sum_{k=1}^{\nu} Y_k X_k^{j+\nu-1} \\ &+\Lambda_2 \sum_{k=1}^{\nu} Y_k X_k^{j+\nu-2} \\ &\quad + \dots \\ &+\Lambda_\nu \sum_{k=1}^{\nu} Y_k X_k^{j} \\ &= 0 \end{aligned} \tag{56}$$

and still the result is 0.

These sums in the formula just happen to be of the form of the syndromes:

$$S_j = \sum_{k=1}^{\nu} Y_k X_k^j \tag{57}$$

Consequently:

$$S_{j+\nu} + \Lambda_1 S_{j+\nu-1} + \cdots + \Lambda_\nu S_j = 0 \tag{58}$$

This is the convolution of S and Λ:

$$S * \Lambda = 0 \tag{59}$$

Note: This is not a coincidence. There exists a base transformation in the finite field, similar to a Fourier Transformation, in which these two vectors are orthogonal.

3.3 Peterson-Gorenstein-Zierler Algorithm

If we have enough known values of S_j, we can use them to determine the unknown values of Λ. For example, we can express $S_{j+\nu}$ in terms of a linear equation of the remaining S_j and Λ:

$$S_j\Lambda_\nu + S_{j+1}\Lambda_{\nu-1} + \cdots + S_{j+\nu-1}\Lambda_1 = -S_{j+\nu} \tag{60}$$

or:

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_\nu \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu+1} \end{bmatrix} \times \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu} \end{bmatrix} \tag{61}$$

It can be seen, that twice as many syndromes are required, as there are unknown error locations.

If the number of errors if known, this linear equation can be solved directly. This is called Peterson-Grenstein-Zierler algorithm.

Example: We have $\nu = 2$ errors, so:

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \times \begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_3 \\ -S_4 \end{bmatrix} \tag{62}$$

with numbers:

$$\begin{bmatrix} 0 & 5 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} \tag{63}$$

This can be solved:

```
0 \Lambda_2 + 5 \Lambda_1 = 2
5 \Lambda_2 + 2 \Lambda_1 = 5
---
\Lambda_1 = 2/5 = 2*7 = 3
---
5 \Lambda_2 + 2*3 = 5
5 \Lambda_2 + 6 = 5
5 \Lambda_2 = 3
\Lambda_2 = 3/5 = 3*7 = 4
```

$$\Lambda_1 = 3 \tag{64}$$
$$\Lambda_2 = 4 \tag{65}$$

$$\Lambda(x) = 1 + 3x + 4x^2 \tag{66}$$

We can verify the desired properties:

$$\Lambda(0) = 1 \tag{67}$$
$$\Lambda(1) = 6 \tag{68}$$
$$\Lambda(2) = 0 \tag{69}$$
$$\Lambda(3) = 7 \tag{70}$$
$$\Lambda(4) = 6 \tag{71}$$
$$\Lambda(5) = 1 \tag{72}$$
$$\Lambda(6) = 7 \tag{73}$$
$$\Lambda(7) = 0 \tag{74}$$

With Peterson-Gorenstein-Zierler Algorithm, you have to know the number of errors. When receiving a transmitted message, the number of errors is unknown. In order to obtain the correct result, the algorithm has to start with the assumtion of the maximal number of recoeverable errors, in order to establish the solvable equation. If the number of errors is less, then the matrix becomes singular, and the equation is not solvable. In this case, the process has to be repeated with a lower number of errors, until the right number of errors is found. This is particularly inconvenient since the solution of a linear equation requires $O(n^3)$ complexity, with n being the number of errors. So, actually decoding a code word requires working from the maximum number errors downward, reducing the number of errors until the correct decodable code word is found.

For a good channel, we are looking for a different method that assumes a low number of errors and works itself upwards, until the number of errors reaches the maximum number of permitted errors, and the code word eventually becomes non-decodable if it surpasses that number.

3.4 Berlekamp Massey Algorithm

Berlekamp Massey Algorithm has two advantages over the Peterson-Gorenstein-Zierler Algorithm. Firstly, it has a lower computational complexity. Secondly, it starts with the assumption of few errors, and works its way upwards, until the maximum recoverable number of errors is reached.

The idea of the algorithm is to construct a minimal linear shift feedback register (LSFR) that produces exactly the syndromes. The LSFR is actually evaluating the error locator polynomial, and thus directly producing the desired result.

The Berlekamp Massey Algorithm is also applicable in complexity theory, in which it is used as a measure for the complexity of a specific number sequence. This complexity is measured as the size of the minimum LSFR that is producing the sequence.

The idea of the of the algorithm is to start with the partial sequence of syndromes, one by one, and adapting the generating LSFR such that it generates the sequence of syndromes that have been processed so far.

The Berlekamp Massey Algorithm works like follows: It inspects the syndromes, one by one, and checks whether the currently available error locator polynomial is capable of producing the sequence. For this purpose, the algorithm computes the discrepancy d of the currently seen syndrome S_n, and the value predicted by the currently assumed error locator polynomial $C(x)$.

If the discrepancy is zero, everything is fine, and the algorithm can proceed.

If however, the discrepancy is non-zero, the current estimation of the error locator polynomial has to be adjusted. This adjustment is performed by keeping around the last estimated polynomial $B(x)$ that resulted in a non-zero discrepancy b. This polynomial $B(X)$ is multiplied by the ratio of the discrepancies, so the two discrepancies will cancel each other out when subtracted from each other. Hence the factor $\frac{d}{b}$ for the adjustment of $C(x)$.

Furthermore, the subtracted adjustment polynomial has to be carefully shifted backwards to that position, at which it produced the observed discrepancy b. For this purpose, a counter m is being kept around that records at how many symbols backwards the discrepancy was observed. Hence, the correction polynomial is additionally multiplied by x^m, such that it affects the syndrome at m positions in the past.

The algorithm starts with a number of estimated errors L of zero. This means an empty error locator polynomal $C(x) = 1$. At each step, the discrepancy of the found and estimated syndrome at position n is computed as:

$$d = 1 \cdot S_n + C_1 \cdot S_{n-1} + \cdots + C_L \cdot S_{n-L} \tag{75}$$

If this discrepancy is non-zero, the current estimate has to be updated. For this purpose, we use the previously used polynomal $B(x)$ with its discrepancy b and distance m:

$$C'(x) = C(x) - \frac{d}{b} x^m B(x) \tag{76}$$

If the previous polynomial $B(x)$ produced discrepancy b at position j, then $m = n - j$. If we compute the discrepancy of the new polynomial $C'(x)$, we obtain:

$$d' = \underbrace{1 \cdot S_n + C_1 \cdot S_{n-1} + \cdots + C_L \cdot S_{n-L}}_{d} \tag{77}$$

$$- \frac{d}{b} \underbrace{(1 \cdot S_j + B_1 \cdot S_{j-1} + \dots)}_{b} \tag{78}$$

$$= d - \frac{d}{b} \cdot b = 0 \tag{79}$$

TODO: prove that the algorithm actually terminates.
TODO: prove that the resulting LSFR is actually the shortest.

Algorithm

```
polynomial(field K) s(x) = ... /* coeffs are s_j; output sequence as N-1 degree polynomi
/* connection polynomial */
polynomial(field K) C(x) = 1;  /* coeffs are c_j */
polynomial(field K) B(x) = 1;
int L = 0;
int m = 1;
field K b = 1;
int n;

/* steps 2. and 6. */
for (n = 0; n < N; n++) {
    /* step 2. calculate discrepancy */
    field K d = s_n + \Sigma_{i=1}^L c_i * s_{n-i};

    if (d == 0) {
        /* step 3. discrepancy is zero; annihilation continues */
        m = m + 1;
    } else if (2 * L <= n) {
        /* step 5. */
        /* temporary copy of C(x) */
        polynomial(field K) T(x) = C(x);

        C(x) = C(x) - d b^{-1} x^m B(x);
        L = n + 1 - L;
        B(x) = T(x);
        b = d;
        m = 1;
    } else {
        /* step 4. */
        C(x) = C(x) - d b^{-1} x^m B(x);
        m = m + 1;
    }
}
return L;
```

Example:

$$s(x) = 5x^3 + 2x^2 + 5x + 0 \tag{80}$$
$$C(x) = 1 \tag{81}$$
$$B(x) = 1 \tag{82}$$
$$L = 0 \tag{83}$$
$$m = 1 \tag{84}$$
$$b = 1 \tag{85}$$

Iteration: $n = 0$

$$d = s_0 + \sum_{i=1}^{L} c_i \cdot s_{n-i} \tag{86}$$

$$= s_0 = 0 \tag{87}$$

$$m = m + 1 = 2 \tag{88}$$

Iteration: $n = 1$

$$d = s_n + \sum_{i=1}^{L} c_i \cdot s_{n-i} = \tag{89}$$

$$= s_1 = 5 \tag{90}$$

$$d \neq 0 \tag{91}$$

$$\underbrace{2L}_{0} \leq \underbrace{n}_{1} \tag{92}$$

$$T(x) = C(x) = 1 \tag{93}$$

$$C(x) = \underbrace{C(x)}_{1} - \underbrace{d}_{5} \cdot \underbrace{b^{-1}}_{1} \cdot \underbrace{x^m}_{x^2} \cdot \underbrace{B(x)}_{1} \tag{94}$$

$$= 5x^2 + 1 \tag{95}$$

$$L = \underbrace{n}_{1} + 1 - \underbrace{L}_{0} = 2 \tag{96}$$

$$B(x) = T(x) = 1 \tag{97}$$

$$b = d = 5 \tag{98}$$

$$m = 1 \tag{99}$$

Iteration: $n = 2$

$$d = s_n + \sum_{i=1}^{L} c_i \cdot s_{n-i} = \tag{100}$$

$$= \underbrace{s_2}_{2} + \underbrace{c_1}_{0} \cdot \underbrace{s_1}_{5} + \underbrace{c_2}_{5} \cdot \underbrace{s_0}_{0} = 2 \tag{101}$$

$$d \neq 0 \tag{102}$$

$$\underbrace{2L}_{4} \not\leq \underbrace{n}_{2} \tag{103}$$

$$C(x) = \underbrace{C(x)}_{5x^2+1} - \underbrace{\underbrace{d}_{2} \cdot \underbrace{b^{-1}}_{\underbrace{5^{-1}}_{7}}}_{3} \cdot \underbrace{x^m}_{x^1} \cdot \underbrace{B(x)}_{1} = \tag{104}$$

$$= 5x^2 + 3x + 1 \tag{105}$$

$$m = \underbrace{m}_{1} + 1 = 2 \tag{106}$$

Iteration: $n = 3$

$$d = s_n + \sum_{i=1}^{L} c_i \cdot s_{n-i} = \tag{107}$$

$$= \underbrace{s_3}_{5} + \underbrace{\underbrace{c_1}_{3} \cdot \underbrace{s_2}_{2}}_{6} + \underbrace{\underbrace{c_2}_{5} \cdot \underbrace{s_1}_{5}}_{6} = \tag{108}$$

$$= 5 \tag{109}$$

$$d \neq 0 \tag{110}$$

$$\underbrace{2L}_{4} \not\leq \underbrace{n}_{3} \tag{111}$$

$$C(x) = \underbrace{C(x)}_{5x^2+3x+1} - \underbrace{d}_{5} \cdot \underbrace{b^{-1}}_{5^{-1}} \cdot \underbrace{x^m}_{x^2} \cdot \underbrace{B(x)}_{1} = \tag{112}$$

$$= 5x^2 + x^2 + 3x + 1 = 4x^2 + 3x + 1 \tag{113}$$

$$m = \underbrace{m}_{2} + 1 = 3 \tag{114}$$

Result

$$\Lambda(x) = C(x) = 4x^2 + 3x + 1 \tag{115}$$

3.5 Chien Search

After having obtained the error locator polynomial Λ, the next objective is finding its zeros $\Lambda(x) = 0$.

$$\sum_{i=1}^{\nu} \Lambda_i x^i = 0 \tag{116}$$

Example:

$$\underbrace{\Lambda_2}_{4} \cdot x^2 + \underbrace{\Lambda_1}_{3} \cdot x + \underbrace{\Lambda_0}_{1} = 0 \tag{117}$$

The zeros can be determined by evaluation of each individual field element. For a field with n elements, this takes $O(n)$ multiplications and $O(n)$ additions (Note: evaluate using Horner's Algorithm!).

Example:

$$(\Lambda_2 \cdot x + \Lambda_1) \cdot x + \Lambda_0 = 0 \tag{118}$$

This algorithm can be slightly improved. The idea of this improvement is evaluation in the order of α^i. This improved algorithm is called Chien Search.

Definition:

$$\gamma_{i,j} = \Lambda_i \cdot \left(\alpha^j\right)^i \tag{119}$$

$$\Lambda(\alpha^j) = \sum_{i=1}^{\nu} \Lambda_i (\alpha^j)^i = \sum_{i=1}^{\nu} \gamma_{i,j} \tag{120}$$

$$\Lambda(\alpha^{j+1}) = \sum_{i=1}^{\nu} \gamma_{i,j+1} \tag{121}$$

$$\gamma_{i,j+1} = \Lambda_i \cdot (\alpha^{(j+1)})^i = \Lambda_i \cdot (\alpha^j)^i \cdot \alpha^i = \gamma_{i,j} \cdot \alpha^i \tag{122}$$

Observation: α^i are constants. By evaluation using this scheme, only $O(n)$ GF-additions and $O(n)$ GF-multiplications by constants per iteration are required. Note also, that these are the same constants that are also required for evaluation of the syndrome.

Example:

$$\alpha^0 = 1 \tag{123}$$

$$\alpha^1 = 2 \tag{124}$$

$$\alpha^2 = 2 \cdot 2 = 4 \tag{125}$$

Iteration: α^0

$$\gamma_{0,0} = \Lambda_0 = 1 \tag{126}$$

$$\gamma_{1,0} = \Lambda_1 = 3 \tag{127}$$

$$\gamma_{2,0} = \Lambda_2 = 4 \tag{128}$$

$$\Lambda(\alpha^0) = \Lambda(1) = \gamma_{0,0} + \gamma_{1,0} + \gamma_{2,0} = 1 + 3 + 4 = 6 \tag{129}$$

Iteration: α^1

$$\gamma_{0,1} = \gamma_{0,0} * \alpha^0 = 1 * 1 = 1 \tag{130}$$

$$\gamma_{1,1} = \gamma_{1,0} * \alpha^1 = 3 * 2 = 6 \tag{131}$$

$$\gamma_{2,1} = \gamma_{2,0} * \alpha^2 = 4 * 4 = 7 \tag{132}$$

$$\Lambda(\alpha^1) = \Lambda(2) = \gamma_{0,1} + \gamma_{1,1} + \gamma_{2,1} = 1 + 6 + 7 = 0 \tag{133}$$

We found a zero at $X^{-1} = 2$. The error location is $\log X = \log 2^{-1} = \log 6 = 6$. In our received message, the position 6 is wrong.

Iteration: α^2

$$\gamma_{0,2} = \gamma_{0,1} * \alpha^0 = 1 * 1 = 1 \tag{134}$$

$$\gamma_{1,2} = \gamma_{1,1} * \alpha^1 = 6 * 2 = 1 \tag{135}$$

$$\gamma_{2,2} = \gamma_{2,1} * \alpha^2 = 7 * 4 = 6 \tag{136}$$

$$\Lambda(\alpha^2) = \Lambda(4) = \gamma_{0,2} + \gamma_{1,2} + \gamma_{2,2} = 1 + 1 + 6 = 6 \tag{137}$$

Iteration: α^3

$$\gamma_{0,3} = \gamma_{0,2} * \alpha^0 = 1 * 1 = 1 \tag{138}$$
$$\gamma_{1,3} = \gamma_{1,2} * \alpha^1 = 1 * 2 = 2 \tag{139}$$
$$\gamma_{2,3} = \gamma_{2,2} * \alpha^2 = 6 * 4 = 2 \tag{140}$$

$$\Lambda(\alpha^3) = \Lambda(5) = \gamma_{0,3} + \gamma_{1,3} + \gamma_{2,3} = 1 + 2 + 2 = 1 \tag{141}$$

Iteration: α^4

$$\gamma_{0,4} = \gamma_{0,3} * \alpha^0 = 1 * 1 = 1 \tag{142}$$
$$\gamma_{1,4} = \gamma_{1,3} * \alpha^1 = 2 * 2 = 4 \tag{143}$$
$$\gamma_{2,4} = \gamma_{2,3} * \alpha^2 = 2 * 4 = 5 \tag{144}$$

$$\Lambda(\alpha^4) = \Lambda(7) = \gamma_{0,4} + \gamma_{1,4} + \gamma_{2,4} = 1 + 4 + 5 = 0 \tag{145}$$

We found a zero at $X^{-1} = 7$. The error location is $\log X = \log 7^{-1} = \log 5 = 3$. In our received message, the position 3 is wrong.

Iteration: α^5

$$\gamma_{0,5} = \gamma_{0,4} * \alpha^0 = 1 * 1 = 1 \tag{146}$$
$$\gamma_{1,5} = \gamma_{1,4} * \alpha^1 = 4 * 2 = 5 \tag{147}$$
$$\gamma_{2,5} = \gamma_{2,4} * \alpha^2 = 5 * 4 = 3 \tag{148}$$

$$\Lambda(\alpha^5) = \Lambda(3) = \gamma_{0,5} + \gamma_{1,5} + \gamma_{2,5} = 1 + 5 + 3 = 7 \tag{149}$$

Iteration: α^6

$$\gamma_{0,6} = \gamma_{0,5} * \alpha^0 = 1 * 1 = 1 \tag{150}$$
$$\gamma_{1,6} = \gamma_{1,5} * \alpha^1 = 5 * 2 = 7 \tag{151}$$
$$\gamma_{2,6} = \gamma_{2,5} * \alpha^2 = 3 * 4 = 1 \tag{152}$$

$$\Lambda(\alpha^6) = \Lambda(6) = \gamma_{0,6} + \gamma_{1,6} + \gamma_{2,6} = 1 + 7 + 1 = 7 \tag{153}$$

3.6 Error Correction

Once we have obtained the error locations, our final goal is to compute the correct values for the defective values.

One way to do this is to solve the syndrome equation for the error values:

$$\begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_\nu^1 \\ X_1^2 & X_2^2 & \cdots & X_\nu^2 \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{n-k} & X_2^{n-k} & \cdots & X_\nu^{n-k} \end{bmatrix} \cdot \begin{bmatrix} Y_1 \\ Y_2 \\ \cdots \\ Y_\nu \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \cdots \\ S_{n-k} \end{bmatrix} \tag{154}$$

Note This matrix is a Vandermonde matrix.

Example:

$$\begin{bmatrix} X_1^1 & X_2^1 \\ X_1^2 & X_2^2 \\ X_1^3 & X_2^3 \\ X_1^4 & X_2^4 \end{bmatrix} \cdot \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} \tag{155}$$

$$\begin{bmatrix} 5^1 & 6^1 \\ 5^2 & 6^2 \\ 5^3 & 6^3 \\ 5^4 & 6^4 \end{bmatrix} \cdot \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 2 \\ 5 \end{bmatrix} \tag{156}$$

$$\begin{bmatrix} 5 & 6 \\ 6 & 3 \\ 4 & 7 \\ 3 & 5 \end{bmatrix} \cdot \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 2 \\ 5 \end{bmatrix} \tag{157}$$

$$\begin{aligned}
(I) \quad & 5Y_1 + 6Y_2 = 0 && \cdot 5 && (158)\\
(II) \quad & 6Y_1 + 3Y_2 = 5 && && (159)\\
(I') \quad & 6Y_1 + 4Y_2 = 0 && && (160)\\
(I' + II) \quad & 7Y_2 = 5 && \cdot 7^{-1} = 5 && (161)\\
& Y_2 = 6 && \text{in}(I) && (162)\\
& 5Y_1 + 6 \cdot 6 = 0 && && (163)\\
& Y_1 = 6 \cdot 6 \cdot 5^{-1} = 3 \cdot 7 && && (164)\\
& Y_1 = 4 && && (165)
\end{aligned}$$

3.7 Forney Algorithm

Solving equation systems requires $O(\nu^3)$ complexity. For a larger number of errors, Forney's algorithm can evaluate the missing values in better complexity. It is based on Lagrange interpolation.

Definition: Error Evaluator Polynomial

$$\Omega(x) = S(x) \cdot \Lambda(x) \mod x^{2t} \tag{166}$$

When looking at Ω, we can infer the following properties:

$$\Omega(x) = S(x) \cdot \Lambda(x) \mod x^{2t} \tag{167}$$

$$= \overbrace{\left(\sum_{j=0}^{2t-1} S_{j+1} x^j \right)}^{S(x)} \cdot \overbrace{\prod_{k=1}^{\nu} (1 - x \cdot X_k)}^{\Lambda(x)} \mod x^{2t} \tag{168}$$

$$= (\sum_{j=0}^{2t-1} \overbrace{\sum_{k=1}^{\nu} Y_k X_k^{j+1}}^{S_{j+1}} x^j) \cdot \prod_{k=1}^{\nu} (1 - x \cdot X_k) \mod x^{2t} \tag{169}$$

Now, we switch the sums and factor out constants:

$$= \left(\sum_{k=1}^{\nu} Y_k X_k \cdot \sum_{j=0}^{2t-1} (X_k \cdot x)^j \right) \cdot \prod_{k=1}^{\nu} (1 - X_k \cdot x) \mod x^{2t} \tag{170}$$

now, we can take one factor from the big product:

$$= \sum_{k=1}^{\nu} \left(Y_k X_k \cdot (1 - X_k \cdot x) \sum_{j=0}^{2t-1} (X_k \cdot x)^j \cdot \prod_{l=1, l \neq k}^{\nu} (1 - X_l \cdot x) \right) \mod x^{2t} \tag{171}$$

Let's have a closer look at the factor $(1 - X_k \cdot x) \sum_{j=0}^{2t-1} (X_k \cdot x)^j$. This is actually a telescope sum:

$$(1 - X_k \cdot x) \sum_{j=0}^{2t-1} (X_k \cdot x)^j = 1 - x^{2t} \tag{172}$$

Or modulo x^{2t}, we get:

$$(1 - X_k \cdot x) \sum_{j=0}^{2t-1} (X_k \cdot x)^j \equiv 1 \mod x^{2t} \tag{173}$$

Consequently,

$$\Omega(x) = \sum_{k=1}^{\nu} \left(Y_k X_k \cdot \prod_{l=1, l \neq k}^{\nu} (1 - X_l \cdot x) \right) \mod x^{2t} \tag{174}$$

In the following, we assume that $\deg \Omega < 2t$, and drop the $\mod x^{2t}$.

If we now evaluate Ω at an error location X_j^{-1}:

$$\Omega(X_j^{-1}) = \sum_{k=1}^{\nu} \left(Y_k X_k \cdot \prod_{l=1, l \neq k}^{\nu} \left(1 - X_l \cdot X_j^{-1}\right) \right) \tag{175}$$

The big product always contains a zero. Except if $k = j$:

$$\Omega(X_j^{-1}) = Y_j \cdot \left(X_j \prod_{l=1, l \neq j}^{\nu} \left(1 - X_l \cdot X_j^{-1}\right) \right) \tag{176}$$

Here we can see, that if we evaluate the error evaluator polynomial, we get the error value, multiplied with something that depends on the error location.

Definition: Formal Derivative

$$\Lambda'(x) = \sum_{i=1}^{\nu} \left(\sum_{j=1}^{i} \Lambda_i \right) x^{i-1} \tag{177}$$

We have to show that the formal derivate Λ' corresponds to the factor that is multiplied to the error value when evalating the error evaluator polynomial Ω.

$$\Lambda(x) = \prod_{k=1}^{\nu} (1 - x \cdot X_k) \tag{178}$$

$$\Lambda'(x) = -\sum_{l=1}^{\nu} X_l \prod_{k=1,k\neq l}^{\nu} (1 - x \cdot X_k) \tag{179}$$

$$\Lambda'(X_j^{-1}) = -\sum_{l=1}^{\nu} X_l \prod_{k=1,k\neq l}^{\nu} \left(1 - X_j^{-1} \cdot X_k\right) \tag{180}$$

The product always has a zero, except if $j = l$

$$\Lambda'(X_j^{-1}) = -X_l \prod_{k=1,k\neq j}^{\nu} \left(1 - X_j^{-1} \cdot X_k\right) \tag{181}$$

$$\tag{182}$$

Note: For a field based on F_2, every second coefficient is zero. So, the derivative is obtained by shifting coeffients by one and zeroing every other coefficient.

Using the error evaluator polynomial, the defective values can be computed as follows:

$$Y_j = -\frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})} \tag{183}$$

Example:

$$S(x) = 5x^3 + 2x^2 + 5x \tag{184}$$

$$\Lambda(x) = 1 + 3x + 4x^2 \tag{185}$$

$$\Lambda'(x) = 3 \tag{186}$$

$$\frac{1}{\Lambda'(x)} = 4 \tag{187}$$

$$\Omega(x) = S(x) \cdot \Lambda(x) \mod x^{2t} \tag{188}$$

$$= (5x^3 + 2x^2 + 5x) \cdot (4x^2 + 3x + 1) \mod x^4 \tag{189}$$

$$= (5 \cdot 4)x^5 + (5 \cdot 3 + 2 \cdot 4)x^4 + \underbrace{(\underbrace{5 \cdot 1}_{5} + \underbrace{2 \cdot 3}_{6} + \underbrace{5 \cdot 4}_{3})}_{0} x^3 + \underbrace{(\underbrace{2 \cdot 1}_{2} + \underbrace{5 \cdot 3}_{2})}_{0} x^2 + 5x \tag{190}$$

$$\Omega(x) = 5x \tag{191}$$

$$Y_1 = -\frac{\Omega(X_1^{-1})}{\Lambda'(X_1^{-1})} = \Omega(7) \cdot 4 = \underbrace{5 \cdot 7}_{1} \cdot 4 = 4 \tag{192}$$

$$Y_2 = -\frac{\Omega(X_2^{-1})}{\Lambda'(X_2^{-1})} = \Omega(2) \cdot 4 = \underbrace{5 \cdot 2}_{7} \cdot 4 = 6 \tag{193}$$

3.7.1 Correction of Known Errors

In more complex encoding systems, e.g. with inner and outer codes, the presence of an error might already be known at the decoding state of Reed-Solomon (which is typically being used as outer code). These known errors are also known as erasures.

In this case, these errors can be corrected much more simply:

Definition: An erasure locator polynomial

$$\Gamma(x) = \prod_{k=1}^{\mu} (1 - x\alpha^{i_k}) \tag{194}$$

might be constructed from the known error locations.

This is much easier than computing it from the syndromes. By this method, up to $2t$ erasures (i.e. twice as many as unkown errors) can be corrected.

If the message contains both known and unknown error locations, the computed error locator polynomial Λ can also be combined with the erasure locator polynomial Γ into a combined polynomial $\Psi(x) = \Lambda(x) \cdot \Gamma(x)$, and the Forney algorithm can be performed on the combined polynomial.

4 Homework

On a space mission to Mars, an extra-terrestrial artefact, potentially an alien space probe, is found. This space probe is millions of years old and quite damaged.

So, the question is if the alien civilization is capable of doing mathematics. As a universal code, the aliens might have used the Gödel Codons presented by Hofstadter's Typographic Number Theory.

TNT Codons As a reference, here a table of its encodings:

0	666
S	123
$=$	111
$+$	112
$\cdot$	236
(	362
)	323
$<$	212
$>$	213
[	312
]	313
a	262
$\prime$	163
$\wedge$	161
$\vee$	616
$\Longrightarrow$	633
$\neg$	223
$\exists$	333
$\forall$	626
:	636
.	611

(195)

Furthermore, the scientific commission has determined that an alien civilization capable of doing mathematics and sending space probes might as well be capable of performing error correct coding on its million years long space travels. In particular, the aliens might just coincidentally use the Reed-Solomon Encoding that also human earthlings happen to learn at the coding theory lecture.

On the space probe, you find a quite weathered CD, from which you manage to read the following code words:

6	6	6	3	3	2	7
1	1	2	7	7	6	3
6	3	6	1	1	3	4
6	3	3	6	7	3	7
3	1	3	2	0	0	3
X	X	2	7	7	6	3
X	X	X	X	1	1	1
6	X	6	6	6	7	6

What can you say about the alien civilization? Please justify your answer.

5 References

- https://en.wikipedia.org/wiki/Reed_Solomon
- https://en.wikipedia.org/wiki/Berlekamp%E2%80%93Massey_algorithm
- https://en.wikipedia.org/wiki/Chien_search
- https://en.wikipedia.org/wiki/Forney_algorithm
- http://www.site.uottawa.ca/~damours/courses/ELG_5372/Lecture17.pdf
- http://www.site.uottawa.ca/~damours/courses/ELG_5372/Lecture18.pdf
- Douglas R. Hofstadter: Gödel Escher Bach, an Eternal Golden Braid, Basic Books, New York, 1979