Results to Improve the Efficiency of BCH and CRC Codes

Vishal Arul, Glen Frost, Derek Jung, Donald W. Newhart

August 7, 2013

Abstract

A binary codeword is a vector $v = (a_0\ a_1\ \ldots\ a_n) \in \mathbb{F}_2^{n+1}$ that is used to transmit information. Note that v naturally corresponds to the polynomial

$$v(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n \in \mathbb{F}_2[x].$$

Thus, we are naturally interested in investigating polynomials with coefficients in $\mathbb{F}_2$, and we do so in the first two sections. In the third section, we determine how to find the roots of quartic polynomials in $\mathbb{F}_{2^m}[x]$ in order to help detect errors in BCH codes. Each section investigates a problem whose solution helps improve the efficiency and effectiveness of Bose-Chaudhuri-Hocquenghem (BCH) and Cyclic Redundancy Check (CRC) codes. As our solutions are meant to be applied, each solution is in the form of a well-described algorithm. In addition, each of our algorithms keep efficiency in mind to ensure that they can used in application to error-correcting codes.

1. Parity-Check Multiplicity in Binary Cyclic Codes

Suppose C is a primitive BCH code over $\mathbb{F}_2$ of length $n = 2^m - 1$, and let α be a primitive element of $\mathbb{F}_{2^m}$. If $c = (c_0\ c_1\ \ldots\ c_{n-1})$ is an n-tuple, note that c naturally corresponds to the polynomial $\sum_{k=0}^{n-1} c_kx^k$. If $d = 2t + 1$ is the designed distance for C, recall that the generator polynomial $g(x)$ for C is the least degree divisor of x^n-1 such that $g(\alpha^i) = 0$ for $b \leq i \leq b+2t-1$, some $b \in \mathbb{Z}^+$. With this choice, algorithms- such as the Berlekamp-Massey algorithm- will correct up to t errors per received vector.

It is common to modify binary BCH codes by including factors of $(1 + x)$ in their generator polynomials. While this does increase a BCH code's correction capacity, it can help detect decode failure.

For $j \in \mathbb{Z}^+$, suppose we have included $(1 + x)^j$ in the generator polynomial for the code C above. Note this implies that the corresponding polynomial of every codeword in C is divisible by $(1+x)^j$. Given a received vector r, let $r(x)$ be the corresponding polynomial. We then calculate the *syndromes* $S_1 = r(\alpha^b),\ S_2 = r(\alpha^{b+1}),\ \ldots,\ S_{2t} = r(\alpha^{b+2t-1})$. As described in [2], we then apply the Berlekamp-Massey algorithm to $S_1,\ S_2,\ \ldots,\ S_{2t}$ to obtain a new polynomial $\tilde{r}(x)$. If less than t errors occurred in transmission, the algorithm would have corrected each of the errors in r and the vector corresponding to $\tilde{r}(x)$ was the originally sent codeword. However, if we find that $(1 + x)^j$ does not divide $\tilde{r}(x)$, then more than t errors must have occurred since every codeword is divisible by $(1+x)^j$; hence, we can conclude decode failure. This document provides simple tests to determine if $(1 + x)^j$ divides a polynomial in $\mathbb{F}_2[x]$. Hence, in our scenario, we can apply our tests to determine if the vector corresponding to $\tilde{r}(x)$ is a valid candidate for the original codeword.

It is well-known including a factor of $(1+x)$ in the generator polynomial of a binary cyclic code C ensures that every codeword has even weight. Hence, the case $j = 1$ above is sometimes referred

to as a *parity check*. For general j, we are suitably calling our tests *parity-check multiplicity*. As demonstrated, our results can help simplify the detection of decode failure in binary BCH codes with a generator polynomial that includes factors of $(1+x)$. We consider the following problem:

Problem 1. For a given polynomial $f(x) \in \mathbb{F}_2[x]$, find an efficient algorithm for determining if

$$(1+x)^j \mid f(x), \qquad j \in \mathbb{Z}^+.$$

Solution. Let $f(x) = \sum_{k=0}^{n} c_k x^k$, $w = \sum_{k=0}^{n} c_k$, $v_0 = \sum_{k\equiv 0 \bmod 2} c_k$, $v_1 = \sum_{k\equiv 1 \bmod 2} c_k$, $w_0 = \sum_{k\equiv 0 \bmod 4} c_k$, $w_1 = \sum_{k\equiv 1 \bmod 4} c_k$, $w_2 = \sum_{k\equiv 2 \bmod 4} c_k$, and $w_3 = \sum_{k\equiv 3 \bmod 4} c_k$.

For $1 \le j \le 4$, we found that whether or not $(1+x)^j$ divides $f(x)$ depends solely on the values of w, v_0, $w_2 + w_3$, and w_0. We illustrate this explicitly in Figure 1 below.

For general j, we merely have to store the first j diagonals of the first $n+1$ rows of the $\mathbb{F}_2$-version of Pascal's triangle. An example of what a decoder would store for $n = 15$ and $j = 9$ is represented by Figure 2, with grey boxes representing 1's and white boxes representing 0's. (Note that we edited the original picture from [5] by changing the color of squares and removing squares.) Then, using this reduced triangle, we can easily calculate $\sum_{k=0}^{n} c_k \binom{k}{m} \pmod 2$ for $0 \le m \le j-1$. Moreover, $(1+x)^j | f(x)$ iff $\sum_{k=0}^{n} c_k \binom{k}{m} \equiv 0 \pmod 2$ for $0 \le m \le j-1$.

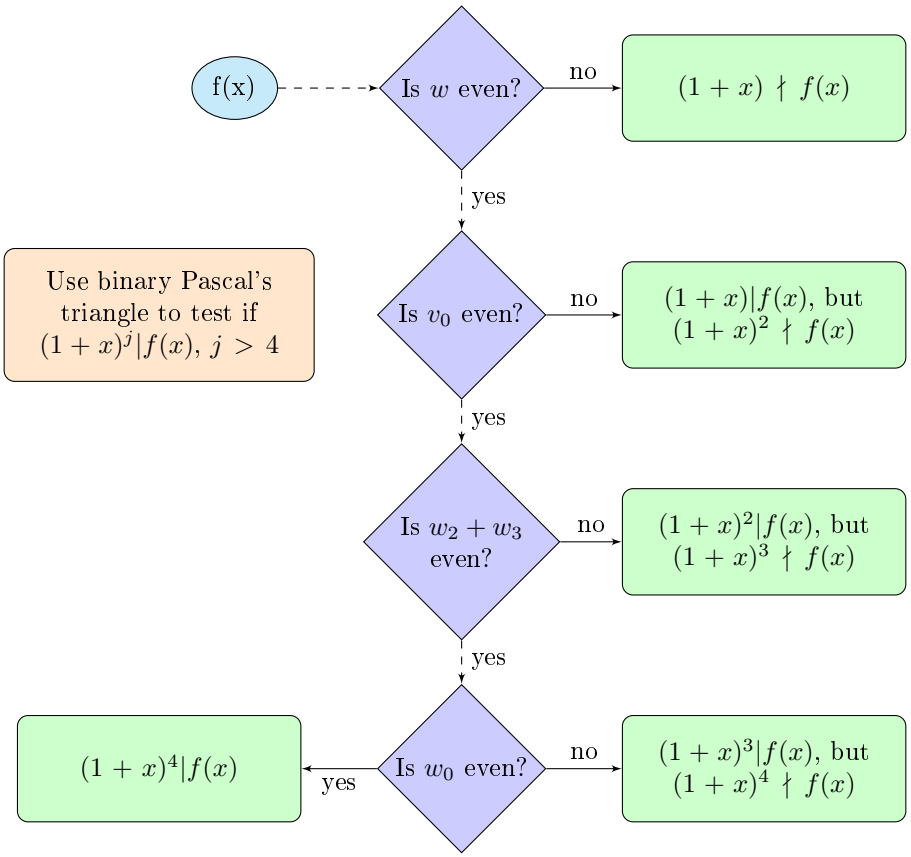

Figure 1: Flowchart for if $(1+x)^j | f(x) \in \mathbb{F}_2[x]$, $1 \le j \le 4$

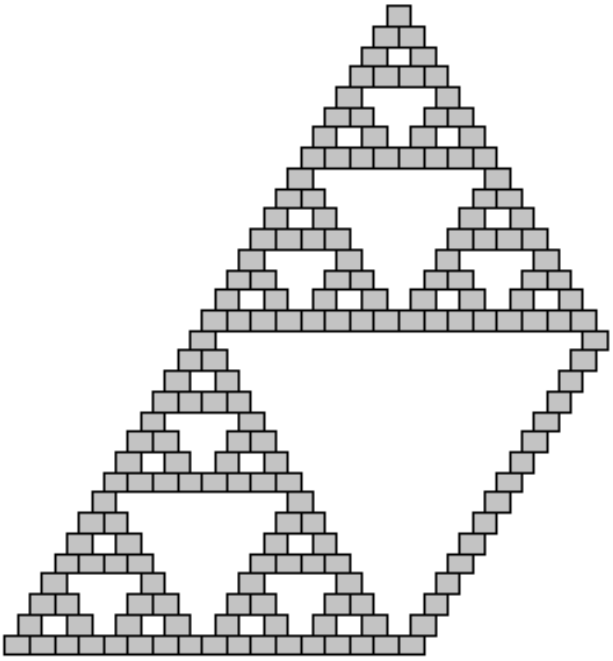

Figure 2: Example of reduced binary Pascal's triangle in [4]

Proof. Case $j = 1$: Note $(x+1)$ divides $f(x)$ if and only if $f(1) = 0$, so the check is: w *is even.*

Case $j = 2$: Since $(x+1)^2 = x^2 + 1$, it follows that $x^2 \equiv 1 \bmod (x+1)^2$. Therefore,

$$f(x) = \sum_{k \equiv 0 \bmod 2} c_k x^k + \sum_{k \equiv 1 \bmod 2} c_k x^k \equiv \sum_{k \equiv 0 \mod 2} c_k + x \sum_{k \equiv 1 \bmod 2} c_k = v_0 + v_1 x \pmod{(x+1)^2}.$$

Since $(x+1)^2$ divides $f(x)$ if and only if this remainder is zero, it follows that the check is: v_0, v_1 *are even.* Note that if $(1+x)|f(x)$, then $(1+x)^2|f(x)$ iff v_0 is even.

Case $j = 3$: First consider the substitution $y = x + 1$, reducing the problem to checking whether or not y^3 divides $f(y+1) = \sum_{k=0}^{n} c_k (y+1)^k$. However, since $(y+1)^4 = y^4 + 1 \equiv 1 \bmod y^3$, it follows that in order to compute $(y+1)^k$ in modulo y^3, it suffices to only consider k modulo 4. For these cases, it is simple to work out that $(y+1)^0 \equiv 1$, $(y+1)^1 \equiv y+1$, $(y+1)^2 \equiv y^2+1$, and $(y+1)^3 \equiv y^2 + y + 1$. Translating this back to the coefficients of f,

$$\begin{aligned} f(y+1) &\equiv w_0 + w_1(y+1) + w_2(y^2+1) + w_3(y^2+y+1) \pmod{y^3} \\ &= (w_2 + w_3)y^2 + (w_1 + w_3)y + (w_0 + w_1 + w_2 + w_3) \\ &= (w_2 + w_3)y^2 + v_1 y + w. \end{aligned}$$

Hence, the check is: $w_2 + w_3$, v_1, w *are even.* Note that if $(1+x)^2|f(x)$, then $(1+x)^3|f(x)$ iff $w_2 + w_3$ is even.

Case $j = 4$: Since $(y+1)^4 = y^4 + 1$, a similar method as in the $j = 2$ case shows that the check is: w_0, w_1, w_2, w_3 *are even.* Observe that if $(1+x)^3|f(x)$, then $(1+x)^4|f(x)$ iff w_0 is even.

For general j, consider the same substitution as in the $j = 3$ case. Build $n+1$ rows of a $\mathbb{F}_2$-version of Pascal's triangle, keeping track of only the first j diagonals: this is the same as the

regular Pascal's triangle, except all of the entries are in $\mathbb{F}_2$. Equivalently, this triangle stores the parity of the binomial coefficient $\binom{k}{m}$ for $0 \leq k \leq n$ and $0 \leq m \leq j-1$. Since

$$(y+1)^k = \sum_{m=0}^{k} \binom{k}{m} y^m \equiv \sum_{m=0}^{j-1} \binom{k}{m} y^m \pmod{y^j},$$

the triangle contains all the information to calculate these coefficients. Then it suffices to check that for every m between 0 and $j-1$ inclusive, $\sum_{k=0}^{n} c_k \binom{k}{m} \equiv 0 \pmod 2$. Indeed, note that $\sum_{k=0}^{n} c_k \binom{k}{m}$ is the coefficient of y^m in $f(y+1)$ for $0 \leq m \leq j-1$, and $y^j | f(y+1)$ iff each of these coefficients is zero in $\mathbb{F}_2$ ■

Let C be a primitive BCH code over $\mathbb{F}_2$ with generator polynomial $g(x)$. We consider the following examples to illustrate how our tests could be applied to detect decode failure and help determine where errors occurred. Note for each $f(x)$ below, we write $f(x) = \sum_{k=0}^{deg(f(x))} c_k x^k$.

1. Suppose we include $(1+x)^2$ in $g(x)$, and after the Berlekamp-Massey algorithm, we have obtained a vector with corresponding polynomial

   $$f(x) = x^{10} + x^7 + x^6 + x^4 + x^2 + 1.$$

   Observe $w = 6$ and $v_0 = 5$. By Figure 2, we can conclude $(1+x) | f(x)$, but $(1+x)^2 \nmid f(x)$, hence decode failure. Note this implies that an odd number of errors occurred in each of the sets $\{c_k \mid k \equiv 0 \pmod 2\}$ and $\{c_k | k \equiv 1 \pmod 2\}$.

2. Suppose we include $(1+x)^3$ in $g(x)$ and we have received a vector with corresponding polynomial

   $$f(x) = x^{28} + x^{24} + x^{21} + x^{19} + x^{14} + x^{10} + x^8 + x^7 + x^3 + 1.$$

   Observe $w = 10$, $v_0 = 6$, and $w_2 + w_3 = 3$. By Figure 2, we can conclude $(1+x)^2 | f(x)$, but $(1+x)^3 \nmid f(x)$. Note this implies that an odd number of errors occurred in each of the sets
   $\{c_k \mid k \equiv 0 \text{ or } k \equiv 1 \pmod 4\}$ and $\{c_k \mid k \equiv 2 \text{ or } k \equiv 3 \pmod 4\}$.

3. Suppose we include $(1+x)^4$ in $g(x)$, and after the Berlekamp-Massey algorithm, we have received a vector with corresponding polynomial

   $$f(x) = x^{55} + x^{37} + x^{20} + x^{18} + x^{15} + x^{12} + x^7 + x^6 + x^2 + 1.$$

   Note $w = 10$, $v_0 = 6$, $w_2 + w_3 = 6$, and $w_0 = 3$. By Figure 2, we can conclude $(1+x)^3 | f(x)$, but $(1+x)^4 \nmid f(x)$, hence decode failure. Note this implies that an odd number of errors occurred in each of the sets $\{c_k \mid k \equiv 0 \pmod 4\}$, $\{c_k \mid k \equiv 1 \pmod 4\}$, $\{c_k \mid k \equiv 2 \pmod 4\}$, and $\{c_k \mid k \equiv 3 \pmod 4\}$.

4. Finally, suppose we include $(1+x)^5$ in $g(x)$ and we are only interested in testing polynomials in $\mathbb{F}_2[x]$ of degree less than 16. So, before any testing, we will store the first 5 diagonals of the first 16 rows of the $\mathbb{F}_2$-version of Pascal's triangle. We illustrate this in Figure 3 (where similar changes to the original picture from [5] have been made as in Figure 1).

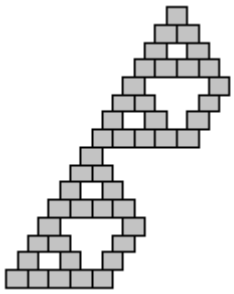

Figure 3: from [5]

Now, suppose we have received a vector with corresponding polynomial

$$f(x) = \sum_{k=0}^{9} c_k x^k = x^9 + x^8 + x^6 + x^5 + x^2 + 1.$$

We apply the first 10 rows of Figure 3 to calculate:

$$\sum_{k=0}^{9} c_k \binom{k}{0} = 6, \ \sum_{k=0}^{9} c_k \binom{k}{1} = 2, \ \sum_{k=0}^{9} c_k \binom{k}{2} = 2,$$

$$\sum_{k=0}^{9} c_k \binom{k}{3} = 0, \ \sum_{k=0}^{9} c_k \binom{k}{4} = 2.$$

As each of these values is even, we can then conclude $(1+x)^5 | f(x)$. Indeed, note

$$f(x) = (1+x)^5(x^4 + x + 1).$$

Note, we have proven that whether $(1+x)^j$ divides $f(x) \in \mathbb{F}_2[x]$ depends solely on the values of w_0, w_1, w_2, and w_3. Indeed,

- $(1+x)$ divides $f(x)$ if and only if $w_0 + w_1 + w_2 + w_3$ is even.
- $(1+x)^2$ divides $f(x)$ if and only if $w_0 + w_1 + w_2 + w_3$, $w_0 + w_2$ are even.
- $(1+x)^3$ divides $f(x)$ if and only if $w_0 + w_1 + w_2 + w_3$, $w_0 + w_2$, $w_2 + w_3$ are even.
- $(1+x)^4$ divides $f(x)$ if and only if $w_0 + w_1 + w_2 + w_3$, $w_0 + w_2$, $w_2 + w_3$, w_0 are even.

Hence, in application, suppose we obtain a vector $v = (a_0 \ a_1 \ \ldots \ a_n)$ after applying the Berlekamp-Massey algorithm. We can apply a shift register to v as in Figure 4. Observe that the shift register will determine the values of w_0, w_1, w_2, w_3 as the last value entering the shift register is a_0. We can then easily find the values of $w_0 + w_1 + w_2 + w_3$, $w_0 + w_2$, $w_2 + w_3$, w_0 and determine whether $(1+x)^j$ divides the associated polynomial $v(x)$ for $1 \leq j \leq 4$ by considering the parity of these four values. Hence, we could use this shift register to conclude that v is not a valid codeword, that more than t errors occurred, and detect decode failure. Note we write v in reverse order in Figure 4 to represent that this will be the order in which the entries will be received by the shift register.

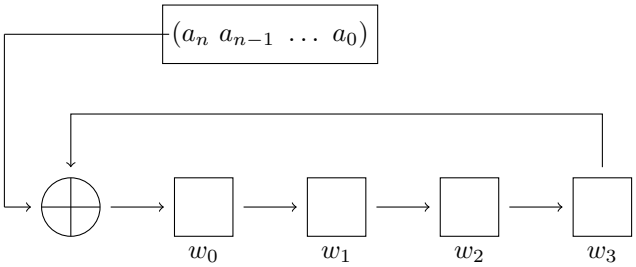

Figure 4:

In this section, we have constructed simple tests of *parity-check multiplicity* to determine if $(1+x)^j | f(x)$ for $j \in \mathbb{Z}^+$, $f(x) \in \mathbb{F}_2[x]$. In particular, when testing $1 \leq j \leq 4$, our tests only require the calculation of partial sums of the coefficients of $f(x)$. These tests can be combined to create a flowchart for mathematical richness or simple shift register to detect decode failure in applications. For $D \in \mathbb{Z}^+$, $j > 4$, we build the first j diagonals of the first D rows of the $\mathbb{F}_2$-version of Pascal's triangle prior to testing. Then, we can directly apply this edited triangle to test if $(1+x)^j | f(x)$ for $f(x) \in \mathbb{F}_2[x]$, $\deg(f(x)) < D$. These tests could be applied after the Berlekamp-Massey algorithm to help detect decode failure in binary BCH codes with a generator polynomial that includes factors of $(1+x)$. The detection of decoding failure for BCH codes is especially helpful in iterated hard-decision decoding of product codes created from them.

2. Remainder-Check in Binary Cyclic Codes

Another setting where multiple factors of $(1+x)$ are included in a generator polynomial $g(x)$ is in a CRC code. As in CRC-32C (Castagnoli) and CRC-32K (Koopman), it is common to raise the degree of $g(x)$ (see [1]) to be a multiple of eight by attaching factors of $(1+x)$. Technically, adjoining a new factor of (1+x) increases the length of the underlying cyclic code; however, to retain the detection capacity, that technicality is ignored in practice. The conditions described above will allow simple tests for higher multiplicity of $(1+x)$ as part of the error-detection step.

[6] also suggests that CRC's can be used as efficient simple hash schemes on binary data; the hash value for a received vector r is the remainder of the associated polynomial modulo $g(x)$. A problem that arises is that distinct vectors can have the same hash value; this is referred to as *collisions*. If the minimum distance of the code is d, then inputs with the same hash value must differ in at least d places, thus limiting the collisions for near repeats in the data (see U.S. patent 8,363,825 for a scheme producing multiple hash values, such as required in Kirsch and Mitzenmacher).

Another way to help avoid this problem is including factors of $(1+x)$ in $g(x)$. This increases the size of the hash value, hence increases the chance that distinct received vectors will have differing hash values. This document shows that including factors of $(1+x)$ in $g(x)$ only has a limited effect in an effort to avoid collisions. In particular, if two distinct vectors share the same hash value with respect to a polynomial $g(x)$, the probability that they will still share the same hash value with respect to $(1+x)g(x)$ is approximately 2^{-1}. More generally, under the same conditions, the probability that they will still share the same hash value with respect to $(1+x)^k g(x)$ is approximately 2^{-k}. Thus, including small factors of $(1+x)$ in $g(x)$ only has a limited effect in avoiding collisions.

Our next result suggests that adjoining factors of $(1+x)$ in the context of hashing provides very limited further value to the hash. In particular, if there was a collision between $f(x)$ and $g(x)$ by taking their hash values $\pmod{g(x)}$, there are only two possibilities for their hash values

with respect to $g(x)$: there will still be a collision or the difference between their hash values is $g(x)$.

Problem 2. Let $g(x) \in \mathbb{F}_2[x]$, $\deg(g(x)) = l$, and $k \in \mathbb{Z}_{\geq 0}$ be given. For given $f(x) \in \mathbb{F}_2[x]$, find an efficient algorithm for determining $b(x) - a(x)$, where

$$f(x) \equiv a(x) \pmod{(1+x)^k g(x)},\ \deg(a(x)) < k + l$$
$$f(x) \equiv b(x) \pmod{(1+x)^{k+1} g(x)},\ \deg(b(x)) < k + l + 1,$$

i.e. $a(x)$ is the remainder of $f(x) \pmod{(1+x)^k g(x)}$, and
$b(x)$ is the remainder of $f(x) \pmod{(1+x)^{k+1} g(x)}$.

Solution. We have the following answer:

$$(\dagger) \qquad b(x) - a(x) = \begin{cases} 0 & \text{if } \deg(b(x)) < k + l \\ (1+x)^k g(x) & \text{if } \deg(b(x)) = k + l. \end{cases}$$

Moreover, determining whether $a(x) = b(x)$ reduces to applying the methods in problem 1.

Proof. Let $g(x), k, f(x), a(x), b(x)$ be given as in the problem statement.

Observe $(1+x)^k g(x) | (b(x) - a(x))$ implies $b(x) - a(x) \in \{0, (1+x)^k g(x)\}$. Moreover, observe $b(x) = a(x)$ if and only if $\deg(b(x)) < k + l$. This implies that $(\dagger)$ holds above. In particular, to calculate $b(x) - a(x)$, it suffices to determine whether $b(x) = a(x)$.

As $f(x) \equiv a(x) \pmod{(1+x)^k g(x)}$, we can write

$$f(x) = q_1(x)(1+x)^k g(x) + a(x)$$

for unique $q_1(x) \in \mathbb{F}_2[x]$.

This implies the following:

$$\begin{aligned} & a(x) = b(x) \\ \Leftrightarrow & f(x) \equiv a(x) \pmod{(1+x)^{k+1} g(x)} \\ \Leftrightarrow & f(x) = q_2(x)(1+x)^{k+1} g(x) + a(x) \quad \text{for unique } q_2(x) \in \mathbb{F}_2[x] \\ \Leftrightarrow & (1+x) | q_1(x) \qquad (\text{as } g(1) = 1 \Rightarrow (1+x) \nmid g(x)) \\ \Leftrightarrow & (1+x)^{k+1} | q_1(x)(1+x)^k g(x) = f(x) - a(x). \end{aligned}$$

Thus, if we can find $a(x)$, then we are reduced to applying our tests of *parity-check multiplicity* from problem 1. Indeed, we could then determine if $(1+x)^{k+1} | (f(x) - a(x))$, which we showed is equivalent to if $a(x) = b(x)$.

Note that we do not need to find $q_1(x)$ as defined above. So, we can apply methods that find $a(x)$ which do not store $q_1(x)$ (cf [3]), and we are done ■

3. Roots of Quartic Polynomials in $\mathbb{F}_{2^m}[x]$ in Application to BCH Codes

Let α be a primitive element of the field $\mathbb{F}_{2^m}$. Then, in BCH codes over $\mathbb{F}_{2^m}$, a received vector v with s errors corresponds to a degree s polynomial $f(Z) \in \mathbb{F}_{2^m}[Z]$ in the following sense:

If $i_1, i_2, \ldots, i_s$ are the locations of the errors in v,
then $\alpha^{i_1}, \alpha^{i_2}, \ldots, \alpha^{i_s}$ are the roots of $f(Z)$.

Thus, we see the importance of being able to find the roots of a given polynomial over $\mathbb{F}_{2^m}$. One such method is the use of a Chien search, but we seek a more direct method for smaller degree polynomials. The case of degree 2 and degree 3 polynomials are handled by Polklinghorn in [4]. The case of degree 4 polynomials is discussed by Yan and Ko in [7]. We similarly discuss degree 4 polynomials to help illuminate this case. In particular, we offer a more detailed perspective on how the test can detect Decoding Failures in application. We solve the following problem:

Problem 3. Given a degree 4 polynomial $f(Z) \in \mathbb{F}_{2^m}[Z]$ with four distinct roots in $\mathbb{F}_{2^m}$, find an efficient algorithm to determine the roots.

Solution. We solve the roots of $f(Z)$ in 5 steps:

1. Transform $f(Z)$ into a polynomial $g(Z)$ of the form Z^4+bZ^2+cZ+d by applying invertible transformations.
2. Define $h(Z) = Z^3+b^2Z+c^2$. If r_1, r_2, r_3, r_4 are the distinct roots of $g(Z)$, then $(r_1+r_2)^2$, $(r_1+r_3)^2$, $(r_2+r_3)^2$ are the three distinct roots of $h(Z)$.
3. Solve for the roots of $h(Z)$ using the method described by Pocklinghorn in [4].
4. Using the method described by Pocklinghorn in [4], determine the roots of two quadratic equations to find the roots of $g(Z)$.
5. Apply inverse transformations to find the roots of $f(Z)$.

Proof. The goal is to find the roots of $f(Z) = Z^4+aZ^3+bZ^2+cZ+d \in \mathbb{F}_{2^m}[Z]$. The first step is to create an invertible transformation which makes the Z^3 term disappear. If $a = 0$ already, then skip this step and move on to the next one.

Lemma. Let $F := \mathbb{F}_{2^m}$. For each $x \in F$, there exists a unique element $y \in F$ such that $y^2 = x$. In particular, $\sqrt{x}$ makes sense for each $x \in F$.

Proof of Lemma. For existence, let α be a primitive element for F. Then, for given $x \in F$, existence is clear if $x = 0$ or $x = \alpha^{2k}$ for some $k \geq 0$. So, suppose $x = \alpha^{2k+1}$, some $k \geq 0$. Then, $\alpha^{2^m-1} = 1$ implies

$$x = \alpha^{2^m-1}\alpha^{2k+1} = \alpha^{2^m+2k} = \left(\alpha^{2^{m-1}+k}\right)^2,$$

and existence follows.

For uniqueness, suppose $\beta^2 = x = \gamma^2$ for $x, \beta, \gamma \in F$. Then, observe that this implies $\beta = \gamma$, hence uniqueness holds. Indeed, as $\mathrm{char}(F) = 2$, we have the following:

$$(\beta-\gamma)^2 = (\beta+\gamma)^2 = \beta^2+\gamma^2 = \beta^2-\gamma^2 = 0.$$

This implies $\beta = \gamma$ and the lemma follows ■

Now suppose $a \neq 0$. Then c/a is well-defined, and by the lemma, it has a unique square root e such that $e^2 = c/a$. Consider the transformation $Z \mapsto Z+e$, which transforms $f(Z)$ to one whose Z term is zero. Redefine a, b, c so that the quartic now looks like $Z^4+aZ^3+bZ^2+c = 0$. Note that $c \neq 0$ as $f(Z)$ has distinct roots. Now consider the reciprocal polynomial, and note that its Z^3-coefficient is now zero. Again redefine b, c, and d so that the reciprocal polynomial times c^{-1} looks like $Z^4+bZ^2+cZ+d = 0$. This quartic has the desired form now, and the

transformations involved (translating $Z \mapsto Z + e$ and moving to the reciprocal polynomial) are all easily invertible. Let $g(Z)$ be the transformed polynomial $Z^4 + bZ^2 + cZ + d$.

One can check using Viète's formulae that for any quartic $Z^4+a_3Z^3+a_2Z^2+a_1Z+a_0 \in \mathbb{F}_{2^m}[Z]$ with roots r_1, r_2, r_3, r_4, that the cubic $Z^3 + (a_1a_3 + a_2^2)Z + (a_1^2 + a_0a_3^2 + a_1a_2a_3)$ has roots $(r_1 + r_2)(r_3 + r_4)$, $(r_1 + r_3)(r_2 + r_4)$, and $(r_1 + r_4)(r_2 + r_3)$. In our particular case, the resolvent cubic ends up being $h(Z) = Z^3 + b^2Z + c^2$. Furthermore, since $r_1 + r_2 + r_3 + r_4 = 0$ (since the coefficient of Z^3 is zero), this also implies that the roots of $h(Z)$ are $(r_1 + r_2)^2$, $(r_1 + r_3)^2$, and $(r_2 + r_3)^2$. Since we only care about the case when r_1, r_2, r_3, r_4 are all in the base field, it follows that the roots of $h(Z)$ must also lie there (otherwise, it should be a decoding failure).

So the next step is to solve the resolvent cubic $h(Z)$. Recall that h has multiple roots if and only if $\gcd(h, h') \neq 1$. Since $h'(Z) = Z^2 + b^2 = (Z + b)^2$, this gcd will be nontrivial if and only if $0 = g(b) = b^3 + b^3 + c^2 = c^2$, if and only if $c = 0$. But $c = 0$ will happen if and only if one of the roots of $h(Z)$ is zero (since then $h(Z) = Z^3 + b^2Z$), but that forces one of $(r_1 + r_2)^2$, $(r_1 + r_3)^2$, $(r_2 + r_3)^2$ to be zero, implying a multiple root (decode failure). So now we may assume that $c \neq 0$ (or else decode failure) and that the roots of $h(Z)$ are distinct elements of the base field. Now we may use Pocklinghorn's method in [4] to solve this cubic.

Suppose that the roots obtained from this method are $\beta_1 = (r_1 + r_2)^2$, $\beta_2 = (r_1 + r_3)^2$, and $\beta_3 = (r_2 + r_3)^2$. Since we can take unique square roots in this field, let $\gamma_1 = r_1 + r_2$, $\gamma_2 = r_1 + r_3$, and $\gamma_3 = r_2 + r_3$. Finally, $r_2 = r_1 + \gamma_1$, $r_3 = r_1 + \gamma_2$, and $r_4 = r_1 + r_2 + r_3 = r_1 + \gamma_3$. Hence, if we know r_1, then we can get the other three roots since we know γ_1, γ_2, and γ_3.

From another application of Viète's formulae, $d = r_1r_2r_3r_4$. Therefore,

$$\begin{aligned} d &= r_1r_2r_3r_4 \\ &= r_1(r_1 + \gamma_1)(r_1 + \gamma_2)(r_1 + \gamma_3) \\ &= [r_1(r_1 + \gamma_3)][(r_1 + \gamma_1)(r_1 + \gamma_2)] \\ &= [r_1^2 + \gamma_3 r_1][r_1^2 + (\gamma_1 + \gamma_2)r_1 + \gamma_1\gamma_2] \\ &= [r_1^2 + \gamma_3 r_1][r_1^2 + \gamma_3 r_1 + \gamma_1\gamma_2] \end{aligned}$$

Now, set $u = r_1^2 + \gamma_3 r_1$. Then the above equation gives $u^2 + u\gamma_1\gamma_2 + d = 0$. Note the coefficient of u will not be zero, or else either γ_1 or γ_2 is zero (which forces $r_1 = r_2$ or $r_1 = r_3$, which is decode failure). This is now a quadratic in u, whose distinct roots must also lie in the base field (since $u = r_1^2 + \gamma_3 r_1$, and both $r_1, \gamma_3 = r_2 + r_3 \in \mathbb{F}_{2^m}$). Then the solution to this quadratic can also be obtained from a method described by Pocklinghorn in [4], which allows us to find u (Choose a value of u out of the two possible and r_1, r_2, r_3, r_4 will be the same up to order). Now that we know u, we may substitute this into the quadratic equation $r_1^2 + \gamma_3 r_1 + u = 0$. Again, use the method in [4] to solve this quadratic, giving us r_1. We can now find r_2, r_3, and r_4 by using $r_2 = \gamma_1 + r_1$, $r_3 = \gamma_2 + r_1$, and $r_4 = \gamma_3 + r_1$. Hence, we have the roots of $g(Z)$, and we can transform them to find the roots of $f(Z)$ ■

Remarks. Note that it suffices to consider degree 4 polynomials $f(Z)$ with four distinct roots. For if $f(Z)$ has multiple roots, this case could have been handled in a lower degree case. Also, note that it is simple to detect Decoding Failures. Indeed, we only need to check the following:

1. The resolvent cubic has all of its roots in $\mathbb{F}_{2^m}$.
2. The constant term of the resolvent cubic is nonzero.

If both of these conditions hold, then we can find the four distinct roots of $f(Z)$ in $\mathbb{F}_{2^m}$. Finally, note that the only step in our algorithm that could be expensive efficiency-wise is finding the

roots of the resolvent cubic and a couple quadratic polynomials. We agreed that finding the roots of cubic and quadratic polynomials are necessary steps in finding the roots of a general quartic. Hence, we believe that our algorithm is optimal as the rest of the algorithm consists of applying transformations and their inverses and making substitutions.

References

1. *Error Detection Using CRC*. Retrieved from <http://measure.feld.cvut.cz/en/system/files/files/en/education/courses/AE4B38DSP/lab9%20-%20CRC.pdf>.

2. Morelos-Zaragoza, R.H. (2006). Decoding of Binary BCH Codes. The Art of Error Correcting Coding (2nd Ed.) (pp. 55-60). Chichester: John Wiley.

3. Newhart, D. (2010). *U.S. Patent No. 7,734,991.* Washington DC: U.S.

4. Pocklinghorn, F., Jr. (1966). Decoding of Double and Triple Error Correcting Bose-Chaudhuri Codes. *IEEE Transactions on Information Theory.* 12(4), 480-481.

5. Sawicki, A. (2009, November 12). *Properties of Pascal Triangle*. Retrieved from <http://www.asawicki.info/news.php5?x=tag>.

6. Stigge, Martin; Plötz, Henryk; Müller, Wolf; Redlich, Jens-Peter (May 2006). Reversing CRC – Theory and Practice. Berlin: Humboldt University Berlin. p. 2. Retrieved 3 August 2013.

7. Yan, F.Y., & Ko, C.C. (1998). Method for Finding Roots of Quartic Equations with Application to RS Codes. *Electronic Letters.* 34(25), 2399-400.