University of Lyon
Hubert Curien Laboratory

Comparison of Different Decoding Algorithms for Binary Goppa Codes

St. Etienne 2014 Rayan Safieddine and Amund Desmarais

Abstract

Comparison of the Patterson algorithm, the Berlekamp-Massey algorithm, and the Extended Euclidean algorithm as decoding algorithms for Goppa codes implemented in C. We explore the process of each algorithm and time their computation speeds.

Contents

1 Introduction

1.1 Finite Fields

- A finite field (also known as Galois field) is a field that contains a finite number of elements.
- A finite field has the same properties as a normal field as well as some additional properties outlined below.
- Properties of a finite field: $\forall\ a,\ b,\ c \in GF(2^{\mathrm{m}})$
  1. Any addition or multiplication of two elements within the field will result in another element within the field, i.e. $a + b = c$
  2. Elements are associative, i.e. $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
  3. Elements are commutative, i.e. $a + b = b + a$ and $a \cdot b = b \cdot a$
  4. There exists an additive and multiplicative identity element, i.e. $a + 0 = a$ and $a \cdot 1 = a$
  5. Each element has an additive and multiplicative inverse, i.e. $a + b = 0$ and $a \cdot c = 1$
  6. Multiplication is distributive over addition, i.e. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- For Goppa codes, calculations done are in the field $GF(2^{\mathrm{m}})$ with an irreducible polynomial T.

1.2 Goppa Codes

- Goppa Codes are a type of error correcting code, they have the following properties:
  - Support set $\mathcal{L} \subseteq GF(2^{\mathrm{m}})$
  - Irreducible polynomial g (*degree* t) s.t. $g(X) \neq 0,\ \forall X \in \mathcal{L}$
  - Parity-Check Matrix $\mathcal{H}$ $(t \times n)$
  - Generator Matrix $\mathcal{G}$ $(k \times n)$ - the rows of a generator matrix of a linear code are a basis of the code
- Can correct up to t errors

1.2.1 Construction of Parity-Check Matrix

$$\mathcal{H}_{t,n} = CX = \begin{bmatrix} \frac{g_t}{g(\alpha_1)} & \cdots & \frac{g_t}{(\alpha_n)} \\ \frac{g_{t-1}+\alpha_1 g_t}{g(\alpha_1)} & \cdots & \frac{g_{t-1}+\alpha_n g_t}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{g_1+\alpha_1 g_2+\cdots+\alpha_1^{t-1} g_t}{g(\alpha_1)} & \cdots & \frac{g_1+\alpha_n g_2+\cdots+\alpha_n^{t-1} g_t}{g(\alpha_n)} \end{bmatrix}$$

$$C = \begin{bmatrix} g_t & 0 & 0 & \cdots & 0 \\ g_{t-1} & g_t & 0 & \cdots & 0 \\ g_{t-2} & g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{bmatrix} \quad H = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \frac{\alpha_1^2}{g(\alpha_1)} & \frac{\alpha_2^2}{g(\alpha_2)} & \cdots & \frac{\alpha_n^2}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{bmatrix}$$

Note: This is the construction of the parity-check matrix for the Patterson and Extended Euclidean (g^2) algorithms.

1.2.2 Representation of Field Elements

Field elements in GF(2^m) can be represented in several ways including:

- Polynomial: $x^3 + x + 1 = \mathbf{1}x^3 + \mathbf{0}x^2 + \mathbf{1}x + \mathbf{1}$
- Binary: $(1011)_2$
- Integer: $(11)_{10}$

Polynomials in $GF(2^m)[X]/g(X)$ can be represented by:

- Polynomial: $f_{t-1}X^{t-1} + \cdots + f_1X + f_0$
- Array: $[f_0, f_1, \ldots, f_{t-1}]$

1.2.3 Sample Goppa Code

$$\text{With } t = 2 \text{ and m } = 3$$

$$T = \mathbf{1}x^3 + \mathbf{0}x^2 + \mathbf{1}x + \mathbf{1} = (1011)_2 = (11)_{10}$$

$$\mathcal{L} = \{0, 1, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6\} = \{0, 1, 2, 4, 3, 6, 7, 5\}$$

$$g(X) = X^t + g_{t-1}X^{t-1} + \cdots + g_0 = X^2 + X + \alpha_3 = X^2 + X + 3$$

$$\mathcal{H} = \begin{bmatrix} 6 & 6 & 2 & 1 & 2 & 4 & 4 & 1 \\ 6 & 0 & 6 & 5 & 4 & 1 & 5 & 4 \end{bmatrix}$$

1.2.4 Error Correction

The sample Goppa code has the following codewords (words where the syndrome is 0):

- 00000000
- 01011011
- 10110101
- 11101110

If 2 errors are added to the second codeword (positions 2 and 7):

$$01000010 \oplus 01011011 \rightarrow 00011001$$

Adding the error back we get (the original codeword):

$$00011001 \oplus 01000010 \rightarrow 01011011$$

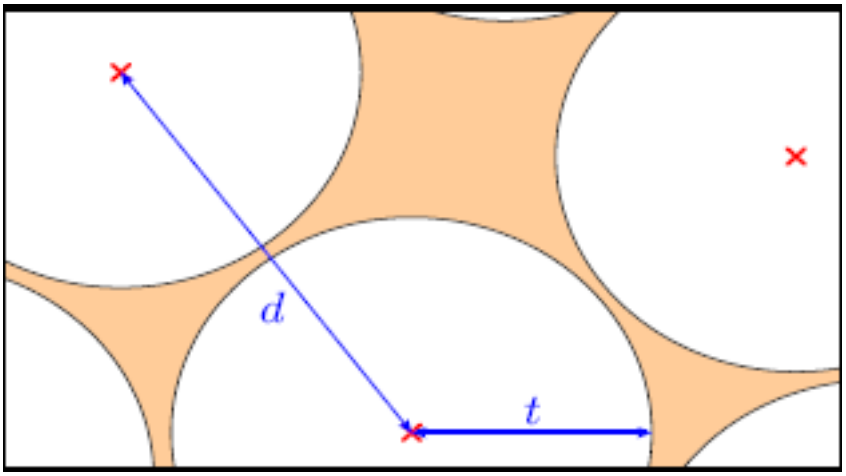
A codeword can be corrected with up to t errors.

1.2.5 Decoding

The following image illustrates how ciphertexts can be corrected:

- The red crosses represent the codewords (i.e. the word the cipher text will be corrected to).
- The white bubbles represent all the different ciphertexts that will correct to the bubble's codeword.
- t represents the correction capacity (i.e. how many errors that can be in a given codeword) - in this diagram, t is the radius of the bubble.
- d represents the minimum distance between any two codewords.
- The colored area represents elements that cannot be corrected by a decoding algorithm.

Any ciphertext in one of the white bubbles will be corrected to the nearest codeword within t errors. In other words, any ciphertext within one of the white bubbles will be corrected to the nearest red cross (x). The red cross will be within t errors because as we can see t is also the radius of the bubble.



2 Decoding Algorithms

There are several algorithms to correct words in a (binary) Goppa code. These include:

1. the Patterson algorithm
2. the Berlekamp-Massey algorithm
3. the Extended Euclidean algorithm (EEA)

2.1 Patterson Algorithm

2.1.1 Algorithm Overview

The Patterson algorithm is one of the most common algorithms used for decoding ciphertexts from a binary Goppa code. The Patterson algorithm can correct up to t errors using a Goppa code over an irreducible polynomial of degree, $g(x)/t$. The algorithm has five major steps, detailed below, that are followed in order to retrieve the error locator polynomial.

2.1.2 Construction of Parity-Check Matrix

$$\mathcal{H}_{t,n} = HX = \begin{bmatrix} \frac{g_t}{g(\alpha_1)} & \cdots & \frac{g_t}{(\alpha_n)} \\ \frac{g_{t-1}+\alpha_1 g_t}{g(\alpha_1)} & \cdots & \frac{g_{t-1}+\alpha_n g_t}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{g_1+\alpha_1 g_2+\cdots+\alpha_1^{t-1} g_t}{g(\alpha_1)} & \cdots & \frac{g_1+\alpha_n g_2+\cdots+\alpha_n^{t-1} g_t}{g(\alpha_n)} \end{bmatrix}$$

$$C_{t,t} = \begin{bmatrix} g_t & 0 & 0 & \cdots & 0 \\ g_{t-1} & g_t & 0 & \cdots & 0 \\ g_{t-2} & g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{bmatrix} \quad H_{t,n} = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \frac{\alpha_1^2}{g(\alpha_1)} & \frac{\alpha_2^2}{g(\alpha_2)} & \cdots & \frac{\alpha_n^2}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{bmatrix}$$

2.1.3 Steps of the Patterson algorithm

1. Calculate the syndrome polynomial
2. Find the inverse of the syndrome polynomial modulo g(X)
3. Compute the square root of the syndrome polynomial inverse plus X modulo g(X)
4. Determine the even and odd parts of the error locator polynomial (ELP)
5. a. Determine the ELP

   b. Evaluate the ELP in order to find its roots

2.1.4 Pseudo-code

Algorithm 1 Patterson Algorithm

Inputs: Goppa code, Parity-check matrix, Ciphertext

Outputs: Error locator polynomial

1. $\mathcal{S} \leftarrow \mathcal{H}^t c$
2. $T \leftarrow \mathcal{S}^{-1} \bmod g$ (via EEA)
3. $\tau \leftarrow \sqrt{T + X} \bmod g$
4. $a,\ b \leftarrow a \equiv b\tau \bmod g$ (via EEA)
5. $\sigma \leftarrow a^2 + Xb^2$ (make sure σ is in the support)

2.1.5 Step by Step Example

Patterson Algorithm: Step 1

Calculate the syndrome polynomial: $\mathcal{S} = c\mathcal{H}^T$
An example for $c = 00011001$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$
$$\times$$
$$c\mathcal{H}^T = \begin{bmatrix} 6 & 6 & 2 & 1 & 2 & 4 & 4 & 1 \\ 6 & 0 & 6 & 5 & 4 & 1 & 5 & 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \oplus \begin{bmatrix} 2 \\ 4 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} \mathbf{2} \\ \mathbf{5} \end{bmatrix}$$
$$\Rightarrow \mathbf{2}X + \mathbf{5} = \mathcal{S}(X)$$

Patterson Algorithm: Step 2

Find $T(X) \equiv \mathcal{S}^{-1}(X)$ mod g(X) (using EEA)

$$\mathcal{S}(X)T(X) \equiv 1 \bmod g(X)$$
$$(2X + 5)(2X + 7) \equiv 1 \bmod g(X) \Rightarrow T(X) = \mathbf{2X} + \mathbf{7}$$

The inverse of $2X + 5$ is $2X + 7$.

Patterson Algorithm: Step 3

a. Compute $\mathcal{H}(X) = T(X) + X$

$$T(X) + X = \mathbf{2X} + \mathbf{7} + \mathbf{X} = 3X + 7 = \mathcal{H}(X)$$

b. Calculate $\tau(X) \equiv \sqrt{\mathcal{H}(X)}$ mod g(X) from $\mathcal{H}(X)$ above:

$$\sqrt{\mathcal{H}(X)} = \mathcal{H}(X)^{2^{mt-1}} = (3X + 7)^5 = 7X + 6 = \tau(X)$$

$$Verification: \ \tau^2(X) = (7X + 6)^2 = (7X + 6)(7X + 6) = 3X^2 + 2$$
$$= 3X^2 + 2 + 3(X^2 + X + 3) = 3X + 5 + 2 = 3X + 7 = \mathcal{H}(X)$$

Patterson Algorithm: Step 4

Calculate $a(X) \equiv b(X)\tau(X)$ mod g(X) (using the EEA) with deg(a(X)) $\leq \lfloor \frac{t}{2} \rfloor$ and deg(b(X)) $\leq \lfloor \frac{t-1}{2} \rfloor$

$$\begin{aligned} & a(X) \equiv b(X)\tau(X) \bmod g(X) \\ \Leftrightarrow\ & a(X) \equiv b(X)(3X+7) \\ \Leftrightarrow\ & (\mathbf{3X+7}) = \mathbf{1}(3X+7) \\ \Leftrightarrow\ & a(X) = 3X+7 \text{ and } b(X) = 1 \end{aligned}$$

Note: a(X) $= \tau(X)$ is valid because deg$(\tau) \leq 1$

Patterson Algorithm: Step 5

a. Find $\sigma(X) = a^2(X) + b^2(X) \cdot X$

$$\begin{aligned} a^2(X) + b^2(X) \cdot X = (3X+7)^2 + (1^2)X &= 3X^2 + 2 + X \\ &= \mathbf{3X^2 + X + 2} = \sigma(X) \end{aligned}$$

b. Find the roots of $\sigma(X)$ using Horner's method (denoted below):

$$\sigma(X) = (((\sigma_t X + \sigma_{t-1})X + \cdots)X + \sigma_1)X + \sigma_0$$

Finding the roots

$$\begin{aligned} \sigma(\alpha) &= (3 \cdot \alpha + 1) \cdot \alpha + 2 \\ \sigma(0) &= (3 \cdot 0 + 1) \cdot 0 + 2 = (1) \cdot 0 + 2 = 2 \\ \sigma(1) &= (3 \cdot 1 + 1) \cdot 1 + 2 = (2) \cdot 1 + 2 = 0 \\ \sigma(2) &= (3 \cdot 2 + 1) \cdot 2 + 2 = (7) \cdot 2 + 2 = 7 \\ \sigma(4) &= (3 \cdot 4 + 1) \cdot 4 + 2 = (6) \cdot 4 + 2 = 7 \\ \sigma(3) &= (3 \cdot 3 + 1) \cdot 3 + 2 = (4) \cdot 3 + 2 = 5 \\ \sigma(6) &= (3 \cdot 6 + 1) \cdot 6 + 2 = (0) \cdot 6 + 2 = 2 \\ \sigma(7) &= (3 \cdot 7 + 1) \cdot 7 + 2 = (3) \cdot 7 + 2 = 0 \\ \sigma(5) &= (3 \cdot 5 + 1) \cdot 5 + 2 = (5) \cdot 5 + 2 = 5 \\ & \Rightarrow \text{there are errors in positions 2 and 7} \end{aligned}$$

2.2 Berlekamp-Massey Algorithm

2.2.1 Algorithm Overview

The Berlekamp-Massey algorithm has not been thoroughly analyzed for side-channel attacks but it is an alternative to the Patterson algorithm. The Berlekamp-Massey algorithm can correct up to t errors only when using $g(x)^2$.

2.2.2 Construction of Parity-Check Matrix

$$\mathcal{H}_{2t,n} = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{\alpha_1^{2t-1}}{g(\alpha_1)} & \cdots & \frac{\alpha_n^{2t-1}}{g(\alpha_n)} \end{bmatrix}$$

2.2.3 Pseudo-code

Algorithm 2 Berlekamp-Massey Algorithm

Inputs: Syndrome polynomial

Outputs: Error locator

1. $L \leftarrow 0$
2. $m \leftarrow -1$
3. $f(x) \leftarrow 1$
4. $g(x) \leftarrow 1$
5. $d_m \leftarrow 1$
6. **for** $k = 0$ to $k = 2t$ - 1 **do**
7. $\quad d_k \leftarrow S_k + \sum_{i=1}^{L} f_i S_{k-i}$
8. $\quad$ **if** $d_k \neq 0$ **then**
9. $\quad\quad h(x) \leftarrow f(x)$
10. $\quad\quad f(x) \leftarrow f(x) + \frac{d_k}{d_m} g(x) x^{k-m}$
11. $\quad\quad$ **if** $L \leq k/2$ **then**
12. $\quad\quad\quad L \leftarrow k + 1 - L$
13. $\quad\quad\quad m \leftarrow k$
14. $\quad\quad\quad g(x) \leftarrow h(x)$
15. $\quad\quad$ **end if**
16. $\quad$ **end if**
17. $\quad k \leftarrow k + 1$
18. **end for**
19. **return** ELP

2.2.4 Step by Step Example

Using $c = 00011001$ and

$$\mathcal{H} = \begin{bmatrix} 0 & 2 & 7 & 5 & 6 & 4 & 7 & 6 \\ 0 & 2 & 6 & 6 & 2 & 7 & 1 & 7 \\ 0 & 2 & 3 & 4 & 7 & 2 & 4 & 5 \\ 2 & 2 & 4 & 1 & 4 & 6 & 6 & 1 \end{bmatrix}$$

we get $\mathcal{S} = 5X^3 + 3X^2 + 6X + 4$
$L = 0$, m = -1, $f(x) = 1$, $g(x) = 1$, $d_{-1} = 1$

$\underline{k = 0}$

- $d_0 = S_0 + \underbrace{\sum_{i=1}^{0} f_i \cdot S_{0-i}}_{=\ 0} = S_0 = 4$

$\underline{d_0 \neq 0}$

- $h(x) = f(x) = 1$
- $f(x) = f(x) + \frac{d_0}{d_{-1}} \cdot g(x) \cdot x^{0-(-1)} = 1 + \frac{4}{1} \cdot 1 \cdot x^1 = 1 + 4x$

$\underline{L = 0 \leq \frac{0}{2}}$

- $L = 0 + 1 - 0 = 1$
- $m = 0$
- $g(x) = 1$

$\underline{k = 1}$

- $d_1 = S_1 + \sum_{i=1}^{1} f_i \cdot S_{1-i} = S_1 + f_1 \cdot S_0 = 6 + 4 \cdot 4 = 0$

$\underline{d_1 = 0}$

$\underline{k = 2}$

- $d_2 = S_2 + \sum_{i=1}^{1} f_i \cdot S_{2-i} = S_2 + f_1 \cdot S_1 = 3 + 4 \cdot 6 = 6$

$\underline{d_2 \neq 0}$

- $h(x) = 1 + 4x$
- $f(x) = 1 + 4x + \frac{6}{4} \cdot 1 \cdot x^{2-0} = 1 + 4x + \frac{6}{4} \cdot x^2 = 1 + 4x + 4 \cdot x^2 = 1 + 4x + 4x^2$

$\underline{L = 1 \leq \frac{2}{2}}$

- $L = 2 + 1 - 1 = 2$
- $m = 2$
- $g(x) = 1 + 4x$

$\underline{k = 3}$

- $d_3 = S_3 + \sum_{i=1}^{2} f_i S_{3-i} = S_3 + f_1 \cdot S_2 + f_2 \cdot S_1 = 5 + 4 \cdot 3 + 4 \cdot 6 = 5 + 7 + 5 = 7$

$\underline{d_3 \neq 0}$

- $h(x) = 1 + 4x + 4x^2$
- $f(x) = 1+4x+4x^2+\frac{7}{6}\cdot(1+4x)\cdot x^{3-2} = 1+4x+4x^2+2\cdot(1+4x)\cdot x = 1+4x+4x^2+(2+3x)\cdot x = 1 + 4x + 4x^2 + 2x + 3x^2 = 1 + 6x + 7x^2$

Finding the roots:

$$\begin{aligned}
\sigma(\alpha) &= (1 \cdot \alpha + 6) \cdot \alpha + 7 \\
\sigma(0) &= (1 \cdot 0 + 6) \cdot 0 + 7 = (6) \cdot 0 + 7 = 7 \\
\sigma(1) &= (1 \cdot 1 + 6) \cdot 1 + 7 = (7) \cdot 1 + 7 = 0 \\
\sigma(2) &= (1 \cdot 2 + 6) \cdot 2 + 7 = (4) \cdot 2 + 7 = 4 \\
\sigma(4) &= (1 \cdot 4 + 6) \cdot 4 + 7 = (2) \cdot 4 + 7 = 4 \\
\sigma(3) &= (1 \cdot 3 + 6) \cdot 3 + 7 = (5) \cdot 3 + 7 = 3 \\
\sigma(6) &= (1 \cdot 6 + 6) \cdot 6 + 7 = (0) \cdot 6 + 7 = 7 \\
\sigma(7) &= (1 \cdot 7 + 6) \cdot 7 + 7 = (1) \cdot 7 + 7 = 0 \\
\sigma(5) &= (1 \cdot 5 + 6) \cdot 5 + 7 = (3) \cdot 5 + 7 = 3 \\
&\Rightarrow \text{there are errors in positions 2 and 7}
\end{aligned}$$

2.3 Extended Euclidean Algorithm (EEA)

2.3.1 Algorithm Overview

The EEA, like the Berlekamp-Massey algorithm, uses $g(x)^2$ in order to computer t errors. It is sometimes denoted by extended greatest common divisor (XGCD).

2.3.2 Construction of Parity-Check Matrix

$$\mathcal{H}_{2t,n} = HX = \begin{bmatrix} \frac{g_{2t}}{g(\alpha_1)} & \cdots & \frac{g_{2t}}{(\alpha_n)} \\ \frac{g_{2t-1}+\alpha_1 g_{2t}}{g(\alpha_1)} & \cdots & \frac{g_{2t-1}+\alpha_n g_{2t}}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{g_1+\alpha_1 g_2+\cdots+\alpha_1^{2t-1} g_{2t}}{g(\alpha_1)} & \cdots & \frac{g_1+\alpha_n g_2+\cdots+\alpha_n^{2t-1} g_{2t}}{g(\alpha_n)} \end{bmatrix}$$

$$C_{2t,2t} = \begin{bmatrix} g_{2t} & 0 & 0 & \cdots & 0 \\ g_{2t-1} & g_{2t} & 0 & \cdots & 0 \\ g_{2t-2} & g_{2t-1} & g_{2t} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_{2t} \end{bmatrix} H_{2t,n} = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \frac{\alpha_1^2}{g(\alpha_1)} & \frac{\alpha_2^2}{g(\alpha_2)} & \cdots & \frac{\alpha_n^2}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{2t-1}}{g(\alpha_1)} & \frac{\alpha_2^{2t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{2t-1}}{g(\alpha_n)} \end{bmatrix}$$

2.3.3 Steps of the Patterson algorithm

1. Calculate the syndrome polynomial
2. Perform the Extended Euclidean algorithm on g^2 and the syndrome to find the coefficient of the syndrome
3. Evaluate the coefficient to find its roots

2.3.4 Step by Step Example

Using the ciphertext $c = 00011001$ and $g^2(X) = X^4 + X^2 + 5$ we get the parity-check matrix:

$$\mathcal{H} = \begin{bmatrix} 2 & 2 & 4 & 1 & 4 & 6 & 6 & 1 \\ 0 & 2 & 3 & 4 & 7 & 2 & 4 & 5 \\ 2 & 0 & 2 & 7 & 6 & 1 & 7 & 6 \\ 0 & 0 & 4 & 1 & 1 & 6 & 3 & 3 \end{bmatrix}$$

Calculate the syndrome polynomial:

$$S = \begin{bmatrix} 1+4+1 \\ 4+7+5 \\ 7+6+6 \\ 1+1+3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 7 \\ 3 \end{bmatrix}$$

The syndrome polynomial is: $S(X) = 4X^3 + 6X^2 + 7X + 3$ Use the extended euclidean algorithm with $S(X) = 4X^3 + 6X^2 + 7X + 3$ and $g^2(x) = X^4 + X^2 + 5$. It returns: $4 = g^2(X) \cdot (2X + 4) + S(X) \cdot (7X^2 + 4X + 3)$. This means that $\sigma(X) = 7X^2 + 4X + 3$

Finding the roots:

$$\begin{aligned} \sigma(\alpha) &= (7 \cdot \alpha + 4) \cdot \alpha + 3 \\ \sigma(0) &= (7 \cdot 0 + 4) \cdot 0 + 3 = (4) \cdot 0 + 3 = 3 \\ \sigma(1) &= (7 \cdot 1 + 4) \cdot 1 + 3 = (3) \cdot 1 + 3 = 0 \\ \sigma(2) &= (7 \cdot 2 + 4) \cdot 2 + 3 = (1) \cdot 2 + 3 = 1 \\ \sigma(4) &= (7 \cdot 4 + 4) \cdot 4 + 3 = (5) \cdot 4 + 3 = 1 \\ \sigma(3) &= (7 \cdot 3 + 4) \cdot 3 + 3 = (6) \cdot 3 + 3 = 2 \\ \sigma(6) &= (7 \cdot 6 + 4) \cdot 6 + 3 = (0) \cdot 6 + 3 = 3 \\ \sigma(7) &= (7 \cdot 7 + 4) \cdot 7 + 3 = (7) \cdot 7 + 3 = 0 \\ \sigma(5) &= (7 \cdot 5 + 4) \cdot 5 + 3 = (2) \cdot 5 + 3 = 2 \\ &\Rightarrow \text{there are errors in positions 2 and 7} \end{aligned}$$

3 C Implementation

3.1 Implementation of Field Operations

The following field operations have been implemented:

- Multiplication
- Power
- Division

The following polynomial operations have been implemented:

- Modulo g
- Evaluate
- Scalar Multiplication
- Coefficient Left Shift ($P(X)X$)
- Addition (XOR)
- Multiplication
- Power
- Square Root*
- Division

3.1.1 Alternate Square Root Algorithm

An alternate way to find the square root of a polynomial

$$P(X) = p_{t-1}X^{t-1} + \cdots + p_1X + p_0$$

is by using:

$$\sqrt{P(X)} \equiv \sum_{i=0}^{\lfloor \frac{t-1}{2} \rfloor} p_{2i}^{2^{m-1}} \; X^i + \sum_{i=0}^{\lfloor \frac{t}{2}-1 \rfloor} p_{2i+1}^{2^{m-1}} \; X^i \; \sqrt{X} \bmod g(X)$$

We implemented this algorithm in our C program as the square root operator.

3.2 Extended Euclidean Algorithm

The EEA takes two polynomials a and b and forms a linear combination from the greatest common divisor with coefficients u and v.

$$gcd(a, b) = u(X)a(X) + v(X)b(X)$$

We implemented the EEA in two ways:

- Recursively - our original implementation; we had trouble stopping it to get specific degrees for u and v
- Iteratively - influenced by the Pari-GP code

3.3 Goppa Code Generation

- We generate a support $\mathcal{L}$ with a given m and T (T is an irreducible polynomial represented as an integer)
- We generate an irreducible g of degree t, with coefficients in $GF(2^{\mathrm{m}})$
- We then generate the parity-check matrix $\mathcal{H}$ by using the formula shown previously

4 Timing Analysis

The following tables show the mean, standard deviation, minimum, and maximum running times of different operations in both the Patterson and the Berlekamp-Massey algorithms. The tests were run on 2000 random cases and the timings were taken after each operation. The timings show number of clock cycles on a Linux OS.

Variable	**Mean**	σ	Min	Max
Compute S	**3607.1**	13.5	3583.0	3642.0
Invert S	**4552.3**	57.9	4273.2	4596.6
Compute τ	**205.80**	0.853	204.01	207.63
EEA	**3239.4**	15.0	3215.3	3280.9
Create σ	**212.93**	0.786	211.50	215.23
Find errors	**3934.4**	49.3	3696.4	3978.6
Runtime	**15758**	112	15272	15917

Table 1: Patterson: $m = 11,\ t = 27$

Variable	**Mean**	σ	Min	Max
Compute S	**16416**	258	16178	17599
Invert S	**34043**	566	32649	36509
Compute τ	**829.12**	12.58	817.43	885.05
EEA	**28227**	436	27827	30240
Create σ	**955.25**	15.00	941.94	1024.90
Find errors	**16801**	256	16153	17895
Runtime	**97283**	1521	95210	104169

Table 2: Patterson: $m = 12,\ t = 56$

Variable	**Mean**	σ	Min	Max
Compute S	**7817.1**	135.7	7697.9	8162.6
BMA	**1685.8**	90.9	1257.8	1775.3
Find errors	**7284.6**	136.3	7067.5	7587.0
Runtime	**16793**	283	16492	17480

Table 3: Berlekamp-Massey: $m = 11,\ t = 27$

Variable	**Mean**	σ	Min	Max
Compute S	**33170**	430	32750	34316
BMA	**7586.9**	280.6	5662.6	7894.7
Find errors	**32374**	420	31795	33557
Runtime	**73139**	1017	70323	75751

Table 4: Berlekamp-Massey: $m = 12,\ t = 56$

5 Side-Channel Attacks

5.1 Timing Attack on Evaluation of $\sigma(x)$

This side-channel attack proposes a major threat to both the Patterson algorithm and the Berlekamp-Massey algorithm. The purpose of the attack is to recover the secret message by attacking the step where we try to find the roots of $\sigma(x)$ (Patterson step ❺). With the secret message the attacker is able to find the error locator polynomial.

Attack: Timing attack on the $\deg(\sigma)$ to find the differences between $\deg(\sigma) = t$ and $\deg(\sigma) = t - 1$.

The countermeasure aims to manipulate $\sigma(x)$ in order to ensure $\deg(\sigma) = t$. The idea behind the countermeasure is to have a consistent running time when solving $\sigma(x)$. This particular countermeasure can be implemented in two ways:

Countermeasure 1: deterministically add coefficients to guarantee $\deg(\sigma) = t$ and that all coefficients are non-zero.
Countermeasure 2: use coefficients from the non-support to guarantee $\deg(\sigma) = t$ and that all coefficients are non-zero.

5.2 Power attack on usage of EEA

This side-channel attack focuses on a vulnerability in the Patterson algorithm. The purpose of this attack is to extract the secret error vector by attacking the use of the XCGD when determining the error locator polynomial (Patterson step ❹). This attack is done by sending selected ciphertexts and measuring the power consumption of the XGCD. With the secret error vector the attacker will be able to get the plaintext.

Attack: Power consumption attack on use of the XGCD to determine the ELP.

The countermeasure aims to determine if XGCD was terminated prematurely (if the degrees of $a(x)$ and $b(x)$ are not proper). If this is the case it enforces the continuation of the XGCD until proper degrees are reached.

Countermeasure:

- **If t is even** then $\deg(a)$ must be equal to $\lfloor \frac{t}{2} \rfloor$ to ensure that $w_e \geq t$ in the last iteration
- **If t is odd** then $\deg(b)$ must be equal to $\lfloor \frac{t}{2} \rfloor$ to ensure that $w_e \geq t$ in the last iteration (provided the ciphertext was not corrupted)

To enforce the continuation of XGCD execution you must manipulate the remainder polynomials $r_i(x)$ to have $\deg(r_{i-1})$ - 1. This manipulation is performed using predefined coefficients or pseudo random coefficients which are derived deterministically from the ciphertext. Do not use random data to implement this countermeasure as the random data is susceptible to further power analysis attacks.

5.3 Timing Attack on Determination of $\sigma(x)$

This side-channel attack is another attack that relates to σ (Patterson step ❹). The attacker tries to find the t correct bits (where the bit is 1) in the ciphertext. There is a different procedure depending on whether there is an even or odd value for t.

Attack: Flip a bit of the ciphertext and measure the decryption time. If the corresponding bit of the error vector is not a correct bit then the bit flipping causes an additional error, so that $w_e > t$. If the corresponding bit of the error vector is a correct bit then the bit flipping causes the removal of this error, so that $w_e = t - 1$. This makes the XGCD iteration number $\leq \frac{t-1}{2} - 1$ and this causes the decryption time to be shorter than in the previous case. This procedure is repeated for all bits of the ciphertext.

The countermeasure ensures that not only is the highest coefficient of $r_i(x)$ set to a non-zero value, but also all other potential leading zero coefficients are non-zero. This countermeasure is identical to the one listed above.

Countermeasure: If the $\deg(r_i) < \lfloor \frac{t}{2} \rfloor$ in any iteration i, then the corresponding ciphertext is manipulated. If this is the case we alter $r_i(x)$ to $r'_i(x)$ so that $\deg(r'_i) = \deg(r_{i-1})$ - 1.

6 Conclusions

6.1 Problems Encountered

- The recursive EEA did not work properly (fixed by using an iterative approach).
- Among the decodable elements, several extra words were being decoded. These extra words do not affect the cryptosystem because no element will have more than t errors from any codeword.
- Irreducibility test did not work.
- Problem with using the EEA to decode when using high values of m and t.

References

[1] Biswas, Bhaskar. 2010. *Implementational Aspects of Code-based Cryptography.*

[2] Cayrel, Pierre-Louis. 2014. *Cours de cryptographie basée sur les codes correcteurs d'erreurs.*

[3] Dragoi, Vlad. 2013. *Side-Channel Attacks in Code-based Cryptography.*

[4] Dragoi, V., Cayrel, P., Colombier, B., & Richmond, T. *Polynomial Structures in Code-based Crytography.*

[5] Hudde, Hans Christoph. 2013. *Development and Evaluation of a Code-based Crytography Library for Constrained Devices.*

[6] Molter, H. G., Stöttinger, M., Shoufan, A., & Strenzke, F. *A Simple Power Analysis Attack on a McEliece Cryptoprocessor.* Springer-Verlag.

[7] Shoufan, A., Strenzke, F., Molter, H. G., & Stöttinger, M. *A Timing Attack against Patterson Algorithm in the McEliece PKC.*