International Journal of Engineering Sciences & Emerging Technologies, Sept. 2017.
ISSN: 22316604 Volume 9, Issue 6, pp: 243-252 ©IJESET

EXTENDED EUCLIDEAN ALGORITHM FOR GOPPA CODE

Ajay Kumar
Assistant Professor in LLRIET, Moga, Punjab, India
E mail: drajaysharma84@gmail.com

ABSTRACT

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

I. INTRODUCTION

1.1. Finite Fields

A finite field is termed as a finite set on which the 4 operations addition, subtraction, multiplication and division excluding division by zero are described, substantial the rules of mathematics identified as the field axioms. Consequently, modest samples of finite fields are determined prime fields. Each prime number is denoted as p . And also the field $GF(p)$ of order p is simply created as the integer modulo p .However; the elements of a determined prime field may be characterized by integers in the range $0,..., p-1$. Where, the sum, the difference and also the product are calculated by taking the remainder by p of the integer outcome [1].
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: $\forall a,b,c \in GF(2^m)$

1. Any addition or multiplication of two elements within the field will result in another element within the field, $a+b=c$
2. Elements are associative, i.e. $a+(b+c)=(a+b)+c$ and $a.(b.c)=(a.b).c$
3. Elements are commutative, i.e. a + b = b + a and a · b = b · a
4. There exists an additive and multiplicative identity element $a+0=a$ and $a.1=a$
5. Each element has an additive and multiplicative inverse, i.e. $a+b=0$ and $a.c=1$
6. Multiplication is distributive over addition, i.e. $a.(b+c)=(a.b)+(a.c)$

For Goppa codes, calculations done are in the field $GF(2^m)$ with an irreducible polynomial T .

Goppa Codes are a type of error correcting code, they have the following properties: Support set $L \subseteq GF(2^m)$

Irreducible polynomial $g(\deg ree\ t)=s.t,\ g(X)\neq 0\ \forall X \in L$

Parity-Check Matrix $H(t \times n)$

Generator Matrix $G(k \times n)$ - the rows of a generator matrix of a linear code are a basis of the code can correct up to t errors.

1.1.1. Construction of Parity-Check Matrix

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

$$C = \begin{bmatrix} g_t & 0 & 0 & \cdots & 0 \\ g_{t-1} & g_t & 0 & \cdots & 0 \\ g_{t-2} & g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{bmatrix} \quad (2)$$

$$H = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \frac{\alpha_1^2}{g(\alpha_1)} & \frac{\alpha_2^2}{g(\alpha_2)} & \cdots & \frac{\alpha_n^2}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{bmatrix} \quad (3)$$

1.1.2. Representation of Field Elements

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

1. Polynomial: x
   $x^3 + x + 1 = 1x^3 + 0x^2 + 1x + 1$
2. Binary: $(1011)_2$
3. Integer: $(11)_{10}$
4. Polynomials in GF(2m) [X] /g(X) can be represented by:
5. Polynomial: $f_{t-1}X^{t-1} + \ldots + f_1X + f_0$
6. Array: $[f_0, f_1, \ldots, f_{t-1}]$

1.2. Goppa Code

In computer science and mathematics, where the binary Goppa code is represent an error-correcting code that belongs to the general codes computer which is initially termed by V.D. Goppa, then the binary structure stretches it numerous mathematical benefits concluded non-binary variants, also

given that a improved fit for mutual procedure in computers and also telecommunication. Consequently, Binary Goppa codes have exciting properties appropriate for cryptography represented in cryptosystem sand comparable setups.

1.2.1. Goppa code sample

Where,

$t = 2$

$m = 3$

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

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

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

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

1.2.2. Error Correction

The sample Goppa code has the following code words (words where the syndrome is 0):
If 2 errors are added to the second codeword (positions 2 and 7)

$$01000010 \oplus 01011011 \rightarrow 00011001$$

While adding the error back we get (the original codeword):A codeword can be corrected with up to t errors.

$$00011001 \oplus 01000010 \rightarrow 01011011$$

A codeword can be corrected with up to t error.

1.3. Decoding

Decoding of the binary Goppa codes is usuallydetermined by Patterson algorithm, which stretchesdecent error-correcting ability and is similarly fairly simple to device [2]. Patterson algorithm changes a syndrome to a vector of errors. The syndrome of a word $c = (c_0, ..., c_{n-1})$ is expected to take a form of

$$s(x) \equiv \sum_{c_i=1} \frac{1}{x - L_i} \bmod g(x) \tag{4}$$

The following image illustrates how cipher texts can be corrected:

1. The black crosses represent the code words (i.e. the word the cipher text will be corrected to).
2. The plain white bubbles represent all the different cipher texts that will correct to the bubble's codeword.
3. 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.
4. d represents the minimum distance between any two code words.
5. The diagonal line (d) represents elements that cannot be corrected by a decoding algorithm.

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

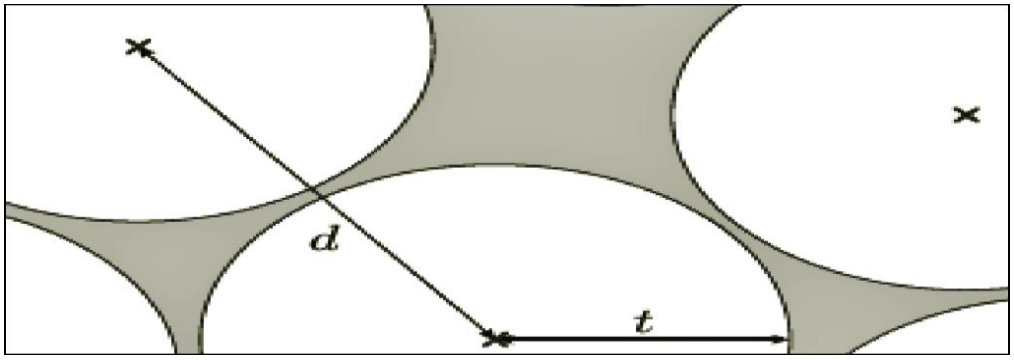

Fig 1. Decoding Pattern

II. Algorithm

2.1. Decoding Algorithms

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

1. Patterson algorithm

2. Berlekamp-Massey algorithm

3. Extended Euclidean algorithm (EEA)

2.1.1. Patterson Algorithm

Algorithm Overview

The Patterson algorithm is one of the most common algorithms used for decoding cipher texts from a binary Goppa code [3]. The Patterson algorithm can correct up to t errors using 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.

Construction of Parity-Check Matrix

$$H_{t,n} = CX = \begin{bmatrix} \frac{g_t}{g(\alpha_1)} & \cdots & \frac{g_t}{(\alpha_n)} \\ \frac{g_{t-1}+\alpha_1 g_t}{g(\alpha_1)} & \cdots & \frac{g_{t-1}+\alpha_n g_t}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{g_1+\alpha_1 g_2+\ldots+\alpha_1^{t-1} g_t}{g(\alpha_1)} & \cdots & \frac{g_1+\alpha_n g_2+\ldots+\alpha_n^{t-1} g_t}{g(\alpha_n)} \end{bmatrix} \quad (5)$$

$$C = \begin{bmatrix} g_t & 0 & 0 & \cdots & 0 \\ g_{t-1} & g_t & 0 & \cdots & 0 \\ g_{t-2} & g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{bmatrix} \quad (6)$$

$$H = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \frac{\alpha_1^2}{g(\alpha_1)} & \frac{\alpha_2^2}{g(\alpha_2)} & \cdots & \frac{\alpha_n^2}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{bmatrix} \quad (7)$$

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

Pseudo code– Patterson algorithm
Input – Goppa code, Parity check matrix, Cipher text
Output – Error Locator Polynomial

$S \leftarrow H^{'}c$
$T \leftarrow S^{-1} \bmod$
g
$\tau \leftarrow \sqrt{T + X} \bmod$
$a, b \leftarrow a \equiv b\tau \bmod$
$\sigma \leftarrow a^2 + Xb^2$

2.1.2. Berlekamp-Massey Algorithm

Algorithm Overview

The Berlekamp-Massey algorithm has not been thoroughly analyzed for side-channel attacks but it is an alternative to the Patterson algorithm [4]. The Berlekamp-Massey algorithm can correct up to t errors only when using

Construction of Parity-Check Matrix

$$H_{2t,n} = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{\alpha_1^{2t-1}}{g(\alpha_1)} & \cdots & \frac{\alpha_n^{2t-1}}{g(\alpha_n)} \end{bmatrix} \quad (8)$$

Pseudo code– Berlekamp-Massey algorithm

Input – Syndrome polynomial

Output – Error Locator Polynomial

$L \leftarrow 0$

$m \leftarrow -1$

$f(x) \leftarrow 1$

$g(x) \leftarrow 1$

$d_m \leftarrow 1$

$for\ k = 0,2t$

$d_k \leftarrow s_k + \sum_{t=1}^{L} f_t s_{k-t} \quad where\ d_k \neq 0$

$h(x) \leftarrow f(x)$

$f(x) \leftarrow f(x) + \frac{d_k}{d_m} g(x) x^{k-m}$

$L \leq \frac{k}{2}$

$L \leftarrow k + 1 - L$

$m \leftarrow k$

$g(x) \leftarrow h(x)$

$k \leftarrow k + 1$

2.1.3. Extended Euclidean Algorithm (EEA)

Algorithm Overview

The EEA, like the Berlekamp-Massey algorithm, uses g(x) denoted by extended greatest common divisor (XGCD) in order to computer t errors [5]. It is sometimes

$$H_{2t,n} = HX = \begin{bmatrix} \frac{g_{2t}}{g(\alpha_1)} & \cdots & \frac{g_{2t}}{g(\alpha_n)} \\ \frac{g_{2t-1} + \alpha_1 g_{2t}}{g(\alpha_1)} & \cdots & \frac{g_{2t-1} + \alpha_n g_{2t}}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{g_1 + \alpha_1 g_{2t-1} + \ldots + \alpha_1^{2t-1} g_{2t}}{g(\alpha_1)} & \cdots & \frac{g_1 + \alpha_n g_{2t-1} + \ldots + \alpha_n^{2t-1} g_{2t}}{g(\alpha_n)} \end{bmatrix} \quad (9)$$

$$C_{2t,2t} = \begin{bmatrix} g_{2t} & 0 & 0 & \cdots & 0 \\ g_{2t-1} & g_{2t} & 0 & \cdots & 0 \\ g_{2t-2} & g_{2t-1} & g_{2t} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_{2t} \end{bmatrix} \quad (10)$$

$$H_{2t,n} = \begin{bmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \frac{\alpha_1^2}{g(\alpha_1)} & \frac{\alpha_2^2}{g(\alpha_2)} & \cdots & \frac{\alpha_n^2}{g(\alpha_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_1^{2t-1}}{g(\alpha_1)} & \frac{\alpha_2^{2t-1}}{g(\alpha_2)} & \cdots & \frac{\alpha_n^{2t-1}}{g(\alpha_n)} \end{bmatrix} \quad (11)$$

Steps of the Extended Euclidean Algorithm

1. Calculate the syndrome polynomial

2. Perform the Extended Euclidean algorithm on g Evaluate the coefficient to find its roots C Implementation

III. Results and Discussions : Implementation of Field Operations

The following field operations have been implemented:

1. Multiplication
2. Power
3. Division
4. The following polynomial operations have been implemented:
5. Modulo g
6. Evaluate
7. Scalar Multiplication
8. Coefficient Left Shift (P(X)X)

9. Addition (XOR)
10. Multiplication
11. Power
12. Square Root
13. Division

3.1. Alternate Square Root Algorithm

An alternate way to find the square root of a polynomial

$$P(X) = p_{t-1}X^{t-1} + \ldots + p_1 X + p_0 \tag{11}$$

is by using:

$$\sqrt{P(X)} \equiv \sum_{i=0}^{\left\lceil \frac{t-1}{2} \right\rceil} p_{2i}^{2^{m-1}} X^i + \sum_{i=0}^{\left\lceil \frac{t}{2}-1 \right\rceil} p_{2i+1}^{2^{m-1}} X^i \sqrt{X} \bmod \tag{12}$$

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.

$$\gcd(a,b) = u(X)a(X) + v(X)b(X) \tag{13}$$

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 codeGoppa Code Generation

• We generate a support 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 $GF(2^m)$

• We then generate the parity-check matrix H by using the formula shown previously

Results

3.3. 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.

Table 1: Patterson algorithm

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 2: Berlekamp Massey algorithm

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 error	7284.6	136.3	7067.5	7587.0
Runtime	16793	283	16492	17480

IV. Side-Channel Attacks

4.1. Timing Attack on Evaluation of σ(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 $\sigma(x)$. With the secret message the attacker is able to find the error locator polynomial.

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

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

Step 1: deterministically add coefficients to guarantee deg (σ) = t and that all coefficients are non-zero.

Step 2: use coefficients from the non-support to guarantee deg (σ) = t and that all coefficients are non-zero.

4.2. Power attack on usage of EEA

This side-channel attack focuses on 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. This attack is done by sending selected cipher texts 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.

Step 1:

• If t is even then deg (a) must be equal to b

• If t is odd then deg (b) must be equal to b cipher text was not corrupted)

To enforce the continuation of XGCD execution you must manipulate the remainder polynomials $r_i(x)$ to have deg $(r_i(x) - 1)$. This manipulation is performed using pre defined coefficients or pseudo random coefficients which are derived deterministically from the cipher text. Do not use random data to implement this countermeasure as the random data is susceptible to further power analysis attacks and tc to ensure that we ≥ t in the last iteration. 2tc to ensure that we ≥ t in the last iteration (provided the Timing Attack on Determination of σ(x).This side-channel attack is another attack that relates to σ. The attacker tries to find the t correct bits (where the bit is 1) in the cipher text. There is a different procedure depending on whether there is an even or odd value for t.

Attack: Flip a bit of the cipher text 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 we > 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 we = t − 1.

This makes the XGCD iteration number ≤ the previous case. This procedure is repeated for all bits of the cipher text. 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.

Step 2:

If the deg (r_i) < b

If this is the case we alter $r_i(x)$ to r t−1, 2 − 1 and this causes the decryption time to be shorter than in tc any iteration i, then the corresponding cipher text is manipulated.

V. Conclusions

Problems Encountered

1. The recursive EEA did not work properly (fixed by using an iterative approach).
2. 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.
3. Irreducibility test did not work.
4. Problem with using the EEA to decode when using high values of m and t.

Reference

[1]. Chauhan, S., & Sharma, A. (2016). Fuzzy Commitment Scheme based on Reed Solomon Codes. *arXiv preprint arXiv:1603.06830*.
[2]. Tillich, J. P. (2016, February). Cryptanalysis of the McEliece Public Key Cryptosystem Based on Polar Codes. In *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings* (Vol. 9606, p. 118). Springer.
[3]. Petrvalsky, M., Richmond, T., Drutarovsky, M., Cayrel, P. L., & Fischer, V. (2016, April). Differential power analysis attack on the secure bit permutation in the McEliece cryptosystem. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)* (pp. 132-137). IEEE.
[4]. Liang, Z., & Zhang, W. (2016). Efficient Berlekamp-Massey Algorithm and Architecture for Reed-Solomon Decoder. *Journal of Signal Processing Systems*, 1-15.
[5]. Smart, N. P. (2016). Modular Arithmetic, Groups, Finite Fields and Probability. In *Cryptography Made Simple* (pp. 3-25). Springer International Publishing.

AUTHOR'S BIOGRAPHY

Ajay Kumar was born in Fazilka (Punjab), India, in 1985. He received the Bachelor in Inter Arts degree from the University of PU, city, in 2004 and the Master in Mathematics degree from the University of Bikaner, Sri Ganga Nagar, in 2006. He is currently pursuing the Ph.D. degree with the Department of Mathematics, PTU. His research interests include Information Theory, Coding Theory, Complex Analysis and Linear Algebra.