Implementasi dan Pengukuran Kinerja Operasi Aritmatika *Finite Field* Berbasis Polinomial Biner

Wenny Franciska Senjaya [1], Budi Rahardjo [2]

[1]*Teknik Informatika, Universitas Kristen Maranatha*
[1]*Jl.Prof.drg.Suria Sumantri No.65, Bandung, Indonesia*
[2]*Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung*
[2]*Jl. Ganesha no. 10, Bandung, Indonesia*
[1] wenny.fs@itmaranatha.org
[2] br@paume.itb.ac.id

Abstract — **Cryptography is a method which is commonly used to secure crucial information on data transmission. This field of study has three kinds of protocols which are Symmetric Algorithms, Asymmetric (Public Key) Algorithms, and Cryptographic Protocols. Elliptic Curve Cryptography (ECC) is a public key encryption which is requires high computation to solve arithmetic operations. Finite field has important role in ECC implementation, finite field is required in curve operations. Finite field consists of many arithmetic operations such as add, multiply, square, inverse, and div operation which can be represented in many form such as prime field GF(p) and binary field GF(2m). There are two representation in binary field, polynomial basis and normal basis. This research presents implementation of arithmetic operation algorithms on polynomial basis with maximum key 299 bits and measured it based on its time and memory usage.**

Keywords — **cryptography, elliptic curve, finite field, polynomial basis, binary field**

I. PENDAHULUAN

Pertukaran data atau informasi secara elektronik sekarang ini semakin berkembang dan semakin mudah, hal ini didukung oleh pengembangan serta pemanfaatan teknologi jaringan dan komputer [1][2]. Perkembangan tersebut tentu diikuti juga dengan semakin besarnya peluang terhadap ancaman pencurian data dan manipulasi data [2]. Dalam hal ini keamanan informasi menjadi salah satu hal sangat penting. Kerahasiaan data, keutuhan data yang dikirim, dan identitas dari pengirim data harus dijaga keasliannya [2][3].

Keamanan komputer dan keamanan jaringan merupakan dua cabang utama dari keamanan informasi. Keamanan komputer merupakan sekumpulan peranti yang dirancang dengan tujuan untuk melindungi data dari ancaman penyerangan, sedangkan keamanan jaringan berperan dalam melindungi data selama proses transmisi [2][4].

Keamanan informasi diterapkan dengan tujuan utama yaitu otentikasi (*authentication*), otorisasi (*authorization*), kerahasiaan data (*confidentiality*), keutuhan data (*data integrity*), akuntabilitas (*accountability*), ketersediaan data (*availability*), dan *non-repudiation* [2][3][4][5][6]. Kebenaran identitas pengirim dan penerima harus dapat dipastikan. Untuk memverifikasi hal tersebut dibutuhkan Otentikasi (*authentication*) identitas pengirim dan penerima data. Otorisasi (*authorization*) bertujuan untuk melakukan pengecekan hak akses seseorang terhadap data yang akan diakses. Selama proses transmisi data, data yang dikirimkan harus dilindungi kerahasiaannya (*confidentiality*). Sehingga data tidak dapat diambil oleh pihak yang tidak berwenang. Selain melindingi kerahasiaan data, data juga harus dijaga keutuhannya (*data integrity*). Hal ini bertujuan melindungi data supaya data tersebut tidak diubah oleh pihak ketiga. Dalam hal ini harus dipastikan bahwa data yang diterima oleh penerima dan data yang dikirim oleh pengirim merupakan data yang sama. Akuntabilitas (*accountability*) bertujuan untuk merekam jejak pengaksesan dan memastikan bahwa pihak yang berwenang atas data dapat mengetahui jejak penyerang jika terjadi sesuatu yang salah dalam proses transmisi data. Ketersediaan data (*availability*) bertujuan untuk memastikan bahwa data dapat diakses ketika ada permintaan dari pihak yang berwenang atas data tersebut. *Non-repudiation* bertujuan supaya pihak pengirim data tidak dapat menyangkal bahwa pengiriman data dilakukan oleh pengirim tersebut dan sebaliknya penerima data tidak dapat menyangkal bahwa data telah diterima oleh penerima tersebut.

Terdapat banyak teknik yang dapat digunakan untuk mengamankan data, salah satu teknik untuk mengamankan proses pengiriman dokumen secara elektronik ialah dengan mengunakan metode enkripsi dan *digital signature* [1][7]. Kriptologi merupakan sebuah bidang ilmu yang mempelajari teknik dan metoda mengenai keamanan data. Bidang ilmu ini memiliki dua cabang utama, yaitu kriptografi dan *cryptanalysis*. Kriptografi merupakan sebuah ilmu yang mempelajari teknik dan metoda tulisan rahasia, dimana tulisan rahasia ini bertujuan untuk menyembunyikan makna sebenarnya

dari pesan yang akan dikirim. *Cryptanalysis* merupakan ilmu dan seni untuk memecahkan kriptografi, selain itu dapat juga digunakan untuk menguji [8]. Kriptografi terdiri dari enkripsi dan dekripsi pesan. Pada enkripsi, pesan yang ada atau dinamakan *plaintext* akan diubah menjadi *cipher text* dengan menggunakan sebuah algoritma. Sedangkan dekripsi merupakan sebuah proses pengambilan pesan hasil enkripsi sehingga pesan dapat dibaca [7].

Terdapat tiga jenis metode dalam kriptografi, yaitu *Symmetric Algorithms*, *Asymmetric* (*Public Key*) *Algorithms*, dan *Cryptographic Protocols*. *Asymmetric* (*Public-Key*) Algorithm diperkenalkan oleh Whitfield Diffie, Martin Hellman, dan Ralph Merkle. Istilah *public-key cryptography* diperkenalkan pada 1975 oleh Diffie, Hellman dan Merkle untuk menangani kekurangan pada *symmetric-key cryptography* [5]. Pada *public-key cryptography* pengguna memiliki *secret key* dan *public key*. *Public-key cryptography* menyediakan solusi yang baik dari permasalahan pada *symmetric-key cryptography* [1].

Salah satu metode enkripsi pada public-key adalah *Elliptic curve cryptography* (ECC). ECC menggunakan komputasi tinggi dalam menyelesaikan operasi aritmatika yang kompleks [9]. Metode ini pertama kali diperkenalkan tahun 1985 oleh Neal Koblitz dan Victor Miller [3][10]. *Elliptic curve* digunakan pada kriptografi karena memiliki sifat matematis yang sama dengan persyaratan untuk enkripsi dan dekripsi. *Elliptic curve* memiliki operasi aritmatika tersendiri yang sangat spesifik dan tidak dapat ditebak. Hal ini membuat hasil kriptografi kuat dan menjadi algoritma kriptografi yang paling baik untuk menggantikan RSA [9].

ECC memiliki banyak *layer*, pembatasan, dan kombinasi dimana membuat variasi dari implementasi ECC sulit untuk dibandingkan. Setiap level pada ECC memberikan banyak hal yang harus diperiksa. Panjang *key* yang digunakan menentukan *level* pada keamanan [9]. Jika dibandingkan dengan RSA, ECC membutuhkan panjang *key* yang lebih sedikit. Tabel I merupakan perbandingan antara algoritma RSA dan ECC berdasarkan tingkat keamanam, ukuran data, ukuran pesan yang dienkripsi, dan kekuatan komputasi yang sama [11].

TABEL I
PERBANDINGAN PANJANG KEY RSA DAN ECC

ECC Key Size (bits)	RSA Key Size (bits)	Key Size Ratio
160	1024	1: 6
224	2048	1: 9
256	3072	1: 12
384	7680	1: 20
512	15360	1: 30

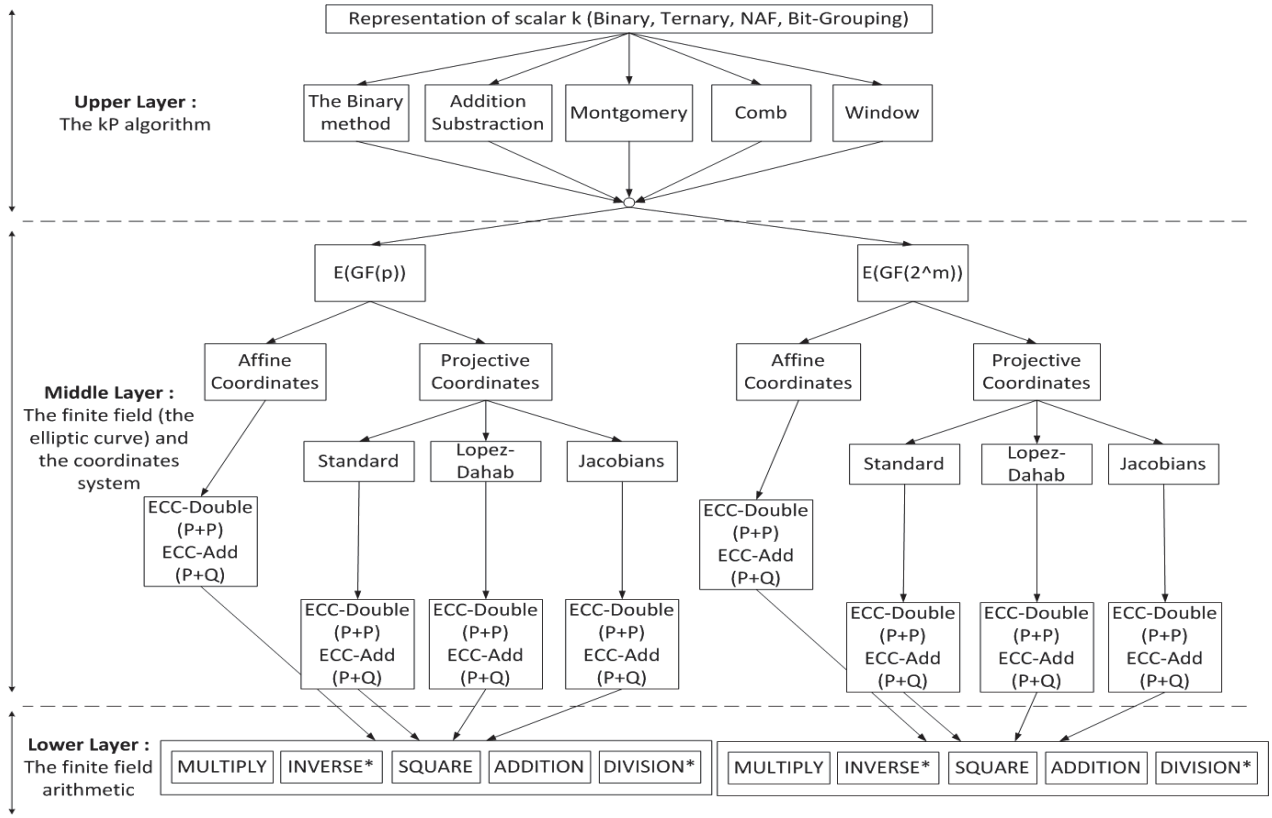
ECC terdiri dari tiga *layer*, pada *layer* atas (*upper layer*) terdapat beberapa algoritma yang dapat digunakan untuk merepresentasikan *k*. *Layer* tengah (*middle layer*) berisi kombinasi untuk merepresentasikan *finite field*, sistem koordinat, dan operasi kurva. Sedangkan *layer* bawah (*lower layer*) terdapat operasi aritmatika pada ECC [7] (Gambar 1).

Operasi yang dapat dilakukan pada *layer* bawah adalah operasi penambahan, perkalian, pembagian, *invers* dan perpangkatan dua (*squaring*). Operasi pada *layer* bawah akan digunakan pada layer atasnya. Sampai saat ini belum ada *tools* yang dapat digunakan sebagai *library*, sehingga pengguna dapat menggunakan fungsi-fungsi operasi aritmatika saja dan langsung mengeluarkan hasilnya. *Tools* yang sudah ada sekarang ini kebanyakan menyediakan perhitungan dalam bilangan prima, sedangkan untuk pendekatan ke arah *hardware* dibutuhkan dalam bilangan biner. *Tools* tersebut juga dibatasi oleh jumlah bit yang kecil, sehingga sulit untuk melakukan perhitungan dengan jumlah bit yang panjang.

Setiap operasi memiliki berbagai metode, pada penelitian ini akan dilakukan implementasi dan pengukuran kinerja dari dua buah algoritma untuk melakukan operasi aritmatika penjumlahan, perkalian, perpangkatan dua, dan *invers* pada *polynomial basis* dengan panjang *key* maksimum 299 bit.

Implementasi akan dilakukan dengan menggunakan bahasa pemrograman python. Penggunaan bahasa pemrograman python dengan pertimbangan python merupakan bahasa pemrograman tinggi, sintaks sederhana dan konsisten, terdapat banyak standar *library* yang siap digunakan, serta dapat di-*run* pada berbagai sistem operasi [12]. Pengukuran kinerja akan dilakukan dalam pengukuran kinerja berdasarkan waktu dan *memory* yang digunakan ketika melakukan operasi aritmatika. Pengukuran kinerja akan dilakukan pada polinom biner dengan *m* adalah 4, 8, 16, 32, 64, 128, 256, dan 299.

Penelitian ini akan menggunakan $GF(2^{299})$ sebagai batas maksimum. Hal ini merujuk pada [13], karena $GF(2^{299})$ dapat dipecah menjadi *base field* $GF(2^{13})$ dan *extension field* $GF(2^{23})$ sehingga dapat terbentuk *composite field* $GF((2^{13})^{23})$. Penelitian ini juga menggunakan daftar *irreducible binary polynomial* hasil penelitian G. Seroussi [14].



Gambar 1 Layer ECC [7]

II. PENELITIAN TERKAIT

Finite field memilki beberapa operasi aritmatka, dimana operasi aritmatika menjadi subjek penelitian untuk waktu yang lama. Berikut penelitian yang melakukan percobaan dengan pengimplementasian dari berbagai metode dan dengan batasan bit yang berbeda-beda.

A. *Implementing Elliptic Curve Cryptography*

Penelitian yang dilakukan oleh Michael Rosing [15] ini menggunakan bahasa pemrograman C. Jumlah bit maksimum adalah 158 bit. Pada penelitiannya, operasi aritmatika berbasis polinomial dan normal (ONB). Michael Rosing juga melakukan implementasi matematika untuk kurva eliptik. Michael Rosing memecahkan permasalahan *large integer* dengan membagi rangkaian integer berdasarkan mesin (*hardware*) yang digunakan.

B. *Kriptografi untuk Keamanan Jaringan*

Penelitian ini dilakukan oleh Rifki Sadikin [2] dengan menggunakan bahasa pemrograman Java. Pada penelitian ini dilakukan implementasi operasi aritmatika *finite field* pada bilangan prima GF(p) dan pada elemen polinomial GF(p^n). Penelitian ini juga mengimplementasikan kurva eliptik serta sistem kriptografi ElGamal.

C. *Efficient Multiplication in GF(p^k) for Elliptic Curve Cryptography*

Penelitian ini dilakukan oleh J.-C. Bajard, L. Imbert, C.Negre dan T.Plantard [16]. Pada penelitian ini, peneliti memperkenalkan algoritma perkalian baru untuk implementasi ECC pada $GF(p^k)$ dengan mendeskripsikan dalam bentuk matriks.

D. *Generic GF(2^m) Arithmetic in Software and its Application to ECC*

Penelitian ini dilakukan oleh Andre Weimerskirch, Douglas Stebila, dan Sheueling Chang Shantz [17]. Pada penelitian ini operasi aritmatika dilakukan pada *binary field* dan diimplementasikan pada *general elliptic curves* dan Koblitz *curves*.

E. *Performance of Finite Field Arithmetic in an Elliptic Curve Cryptosystem*

Penelitian ini dilakukan oleh Zhi Li, John Higgins, dan Mark Clement [10]. Penelitian ini mengukur dan membandingkan kinerja dari proses perhitungan aritmatika dengan jangkauan ukuran *field* dari 100 sampai 1279 bit untuk representasi polinomial basis, dan 100 sampai 1019 bit untuk representasi normal basis.

F. *Implementation of Finite Field Arithmetic Operations for Polynomial and Normal Basis Representations*

Penelitian ini mengimplementasikan operasi aritmatika pada *binary polynomial* dan *optimal normal basis* (ONB). Selain itu, penelitian ini juga mengimplementasikan konversi dari *binary polynomial* ke ONB dan sebaliknya [18].

III. KAJIAN TEORI

A. *Lapangan Hingga (Finite Field)*

Field merupakan pengkhususan dari struktur aljabar *ring*. *Ring* sendiri memiliki dua buah operator untuk satu

himpunan simbol dan harus memenuhi kondisi berikut ini [2][5]:

- Operasi penjumlahan memiliki identitas 0
- Operasi perkalian memiliki identitas 1 untuk bilangan tak-nol
- Berlaku sifat asosiatif, komutatif, dan distributif

Field $\mathbb{F}$ sering dinotasikan {$\mathbb{F}$, +, X}. Kondisi operator pada *field* sama dengan *ring* dengan tambahan operator perkalian memiliki invers untuk seluruh elemen yang bukan merupakan bilangan nol [2][5]. Seperti halnya *ring*, *field* $\mathbb{F}$ memiliki dua operasi biner yaitu penjumlahan dan perkalian [5].

Finite field atau *galois field* merupakan *field* yang jumlah elemennya terbatas dan dapat didefinisikan dengan F_q atau GF(q). Jumlah elemen menjadi syarat dari *finite field*, dimana *q* harus merupakan bilangan prima atau kelipatan dari bilangan prima, $q = p$ atau $q = p^m$, dimana bilangan prima *p* disebut karakteristik dari *finite fields* dan m adalah bilangan bilangan bulat positif. Jika $m = 1$, maka $\mathbb{F}$ disebut *prime field*. Jika $m \geq 2$, maka $\mathbb{F}$ disebut *extension field* [4][5][19][20].

Extension field $GF(p^m)$ dihasilkan oleh suatu fungsi polinomial derajat ke-*m* yang tidak tereduksi pada *GF(p),* dimana merupakan residu modulus yang dihasilkan dari *field* polinomial tidak tereduksi. Oleh sebab itu, dalam polinomial elemen $GF(p^m)$ direpresentasikan pada derajat ke-(m-1) dengan koefisiensi pada *GF(p)* [15].

Polinomial merupakan hasil pernjumlahan variabel-variabel yang memiliki koefisien dan nilai pangkat yang berbeda-beda. Variabel tersebut umumnya dilambangkan dengan *x* [15]. Pada aritmatika komputer, koefisien yang ada pada polinomial tidak digunakan. Koefisien yang ada hanya koefisien yang memiliki nilai 0 atau 1 sehingga dapat direpresentasikan dalam bentuk biner [19]. Misalkan terdapat polinomial $x^7+x^5+x^4+x+1$, maka polinomial tersebut akan disimpan pada komputer dalam byte seperti berikut ini [21]:

x^7		x^5	x^4			x^1	x^0	Polinomial
1	0	1	1	0	0	1	1	Koefisien
7	6	5	4	3	2	1	0	

Finite field dengan orde 2^m disebut *binary field*, untuk membangun *binary field* dapat dilakukan dengan merepresentasikan *polynomial basis*. Berikut ini element $\mathbb{F}_{2^m}$ yang merupakan *binary polynomial* di mana memiliki koefisien pada field $\mathbb{F}_2$={0, 1} dengan derajat terbesar yaitu (m – 1) [5][20]:

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 : a_i \in \{0,1\}\}$$

dengan koefisien a_i yang berada pada *GF(p)*. Nilai pangkat pada z^i menunjukkan posisi koefisien a_i [2]. *Binary polynomial* ini memiliki *irreducible binary polynomial f(z)* dengan derajat *m*. *Irreducible binary polynomial* (polinomial tidak tereduksi) merupakan polinomial yang tidak habis dibagi polinomial biner yang memiliki derajat lebih kecil dari *m* [5][2].

Terdapat empat operasi aritmatika pada polinomial, yaitu penjumlahan, pengurangan, perkalian, dan invers. Aritmatika *finite field* berperan penting dalam ECC, karena *finite field* merupakan inti pada perkalian skalar ECC [4][20].

B. Algoritma Operasi Aritmatika

Finite fields pada orde 2^m disebut dengan *binary field*. Misalkan *binary field* ($\mathbb{F}_{2^m}$) memiliki dua buah elemen A, B ϵ $\mathbb{F}_{2^m}$. Operasi penjumlahan dapat diselesaikan dengan operasi n-bit XOR. Operasi perkalian dapat diselesaikan dengan perkalian biasa (A. B) kemudian di-modulus polinomial tidak tereduksi *P(x)* pada $\mathbb{F}_{2^m}$.

Operasi perpangkatan dua diselesaikan dengan mengubah jumlah bit dengan modulus polinomial tidak tereduksi. Invers dapat dihitung dengan A^{-1} dimana (A. A^{-1}) mod *P(x)* = 1 [20]. Terdapat empat buah notasi yang digunakan dalam operasi biner, yaitu [5]:

- A $\oplus$ B, bitwise exclusive-or
- A & B, bitwise AND
- A $\gg$ i, biner A geser kanan sebanyak i
- A $\ll$ i, biner A geser kiri sebanyak i

Contoh *binary field* dengan $\mathbb{F}_{2^4}$ [5]:
$P(x) = x^4 + x + 1$ dan $A(x) = x^3 + x^2 + 1$, $B(x) = x^2 + x + 1$

1. Penjumlahan:
   $A + B = x^3 + x^2 + x^2 + x + 1 + 1 = x^3 + x$
   $A + B = 01101 \oplus 00111 = 01010$
2. Pengurangan:
   $A + B = x^3 + x^2 + x^2 + x + 1 + 1 = x^3 + x$
   $A + B = 01101 \oplus 00111 = 01010$
   Pada $\mathbb{F}_2$, $-1 = 1$, jadi $-a = a$ untuk semua $a \in \mathbb{F}_{2^m}$.
3. Perkalian:
   A. B = $x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^2 + x + 1 = x^5 + x + 1$

$$\begin{aligned} (x^5 + x + 1) \mod (x^4 + x + 1) &= \ldots \\ x^4 + x + 1 &\leftrightarrow x^4 = x + 1 \\ x^5 + x + 1 &= x^4 \cdot x + x + 1 \\ &= (x + 1)x + x + 1 \\ &= x^2 + x + x + 1 \\ &= x^2 + 1 \end{aligned}$$

$$\begin{aligned} A \cdot B &= 01101 \cdot 00111 = 100011 \\ &\quad 100011 \oplus 00111 = 000101 \end{aligned}$$

4. Perpangkatan
   Untuk $\mathbb{F}_{p^n}$, berlaku $(a + b)^{p^n} = (a)^{p^n} + (b)^{p^n}$

$$\begin{aligned} A^2 &= (x^3 + x^2 + 1)^2 \\ A^{2^1} &= (x^3 + x^2 + 1)^{2^1} \\ &= (x^3)^{2^1} + (x^2)^{2^1} + (1)^{2^1} \\ &= x^6 + x^4 + 1 \end{aligned}$$

$$\begin{array}{rcl} (x^6+x^4+1) \mod (x^4+x+1) = \ldots & & \\ x^4+x+1 & \leftrightarrow & x^4 = x+1 \\ x^6+x^4+1 & = & x^4 \cdot x^2 + x^4 + 1 \\ & = & (x+1)x^2 + (x+1) + 1 \\ & = & x^3 + x^2 + x + 1 + 1 \\ & = & x^3 + x^2 + x \end{array}$$

$01101 \cdot 01101 = 01010001$
$01010001 \oplus 10011 = 1110$

5. Invers:
   $A^{-1} = x^2$ karena $(x^3+x^2+1).\ x^2 \bmod (x^4+x+1) = 1$
   $A^{-1} = 00100$

1. Penjumlahan pada $\mathbb{F}_2^m$
   Pada *finite field* operasi penjumlahan dapat diselesaikan dengan operasi n-bit XOR. Berikut ini algoritma untuk melakukan operasi penjumlahan pada $\mathbb{F}_2^m$ [5].

Algoritma 1 Penjumlahan pada $\mathbb{F}_2^m$

```
Input: polinomial biner a(x) dan b(x) dengan
derajat terbesar m - 1
Output: c(x) = a(x) + b(x)
For i from 0 to t – 1 do
 C[i] = A[i] ⊕ B[i]
Return(c)
```

2. Perkalian pada $\mathbb{F}_2^m$
   Pada operasi perkalian ini, akan diimplementasikan dengan dua algoritma. Pada algoritma pertama setiap bit pada *b* akan dikalikan dengan *a*(*z*) dan dimulai dari bit ke nol, kemudian dilakukan *shift* kiri sebanyak *i*. Lakukan operasi XOR untuk setiap bitnya. Karena pada algoritma ini belum disertai proses reduksi, maka pada implementasi ditambahkan proses reduksi.

Algoritma 2 Perkalian pada $\mathbb{F}_2^m$

```
Input: polinomial biner a(x) dan b(x) dengan
derajat terbesar m - 1
Output: c(x) = a(x). b(x) mod f(x)
If b0 = 0 then c ← 0; else c ← a
For i from 1 to deg(b) do
 k ← 0
 if bi = 0 then k ← 0;
 else k ← a
 k ← k. x^i
 c ← c + k
Return(c)
```

Algoritma kedua pada operasi perkalian pada $\mathbb{F}_2^m$ ialah menggunakan algoritma *right-to-left shif-and-add* method. Berikut ini algoritma untuk perkalian pada $\mathbb{F}_2^m$ [5].

Algoritma 3 Perkalian Right-to-left shif-and-add

```
Input: polinomial biner a(x) dan b(x) dengan
derajat terbesar m - 1
Output: c(x) = a(x). b(x) mod f(x)
If a0 = 1 then c ← b; else c ← 0
For i from 0 to m – 1 do
 b ← b. x mod f(x)
 if ai = 1 then c ← c + b
Return(c)
```

Algoritma *right-to-left shif-and-add* didasarkan pada:

$$a(x).b(x) = a_{m-1}\, x^{m-1}\, b(x) + \ldots + a_2 x^2 b(x) + a_1 x b(x) + a_0 b(x)$$

Perulangan *i* dimaksudkan untuk melakukan perhitungan $x^i b(x) \bmod f(x)$. $x^i b(x) \bmod f(x)$ dapat dihitung dengan melakukan operasi shift kiri terhadap *b(x)*. Algoritma 3 lebih cocok untuk *hardware* karena *vector shift* dapat dilakukan dalam satu *clock cycle* [5].

3. Perpangkatan Dua (*Squaring*) pada $\mathbb{F}_2^m$
   Pada operasi perpangkatan dua, metoda pertama yang akan diimplementasi ialah proses perkalian matematika dengan menggunakan algoritma perkalian. Pada proses ini sebuah polinomial akan dikalikan dengan dirinya sendiri. Diberikan sebuah polinomial pada $\mathbb{F}_2^4$, $a(x) = x^3+x^2+1$ maka untuk mendapatkan $a(x)^2$ dilakukan proses perkalian $a(x).\ a(x)$.

Algoritma 4 Perpangkatan dua pada $\mathbb{F}_2^m$

```
Input: polinomial biner a(x) dengan derajat
terbesar m - 1
Output: c(x) = a(x)^2
   1. c = Multiplication(a, a)
   2. Return (c)
```

Pada implementasi ini, proses perhitungan dilakukan dengan menggunakan Algoritma 2. Algoritma 5 akan lebih cepat jika dibandingkan dengan Algoritma 4 [5].
Jika $a(z) = a_{m-1}z^{m-1}+\ldots+a_2z^2+a_1z+a_0$, maka:
$a(z)^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0$
Jika $a(z)^2$ direpresentasikan dalam bentuk biner maka akan dihasilkan bilangan biner, dimana pada setiap bit *a* akan diselipkan bit 0. Pada proses ini, jumlah bit yang dihasilkan adalah dua kali dari jumlah bit *a*(*z*) [5].

Algoritma 5 Perpangkatan dua pada $\mathbb{F}_2^m$

```
Input: polinomial biner a(x) dengan derajat
terbesar m - 1
Output: c(x) = a(x)^2
   1. Untuk setiap bit a = (an-1, an-2,...,a2,
```

```
    a1, a0)
2.  c = (an-1, 0, an-2, 0,...,a2, 0, a1, 0,
    a0)
3.  Return (c)
```

4. *Invers* pada $\mathbb{F}_2^m$

   Algoritma 6 bertujuan untuk menghitung nilai invers. Algoritma ini menggunakan algoritma Euclidean yang pada awalnya digunakan untuk mencari *greatest common divisor (gcd)*. *Greatest common divisor (gcd)* merupakan polinomial biner dengan *d* derajat tertinggi yang membagi *a* dan *b*. Jika *a* dan *b* merupakan polinomial biner, maka *gcd(a, b) = gcd(b – ca, a)* untuk semua polinomial biner *c* [5].

Algoritma 6 Algoritma Extended Euclidean untuk invers $\mathbb{F}_2^m$

```
Input: polinomial biner bukan nol a dengan
derajat paling banyak m - 1
Output: a-1 mod f
u ← a, v ← f.
g1 ← 1, g2 ← 0.
While u ≠ 1 do
 j ← deg(u) – deg(v).
 If j < 0 then: u ↔ v, g1 ↔ g2, j ← -j.
 u ← u + z j v.
 g1← g1 + z j g2
Return(g1).
```

Algoritma kedua yang digunakan untuk mencari nilai *invers* adalah algoritma 7 dibawah ini. Pada algoritma 6 bit u dan v mengalami pergeseran ke kiri, sedangkan pada algoritma 7 bit u dan v mengalami pergeseran ke kanan.

Algoritma 7 Algoritma Biner untuk invers $\mathbb{F}_2^m$ [5]

```
Input: polinomial biner bukan nol a dengan
derajat paling banyak m - 1
Output: a-1 mod f
u ← a, v ← f.
g1 ← 1, g2 ← 0.
While (u ≠ 1 and v ≠ 1) do
 While z divides u do
 u←u/z
 if z divides g1 then: g1 ← g1/z;
 else g1 ← (g1+f)/z
 While z divides v do
 v←v/z
 if z divides g2 then: g2 ← g2/z;
 else g2 ← (g2+f)/z
 if der(u) > der(v) then: u ← u+v, g1 ← g1+g2;
 else: v ← v+u, g2 ← g2+g1
if u=1 then return(g1); else return(g2)
```

5. Reduksi pada $\mathbb{F}_2^m$

   Derajat polinom terbesar harus selalu (*m*-1). Jika hasil perhitungan aritmatika polinom tersebut lebih dari (*m*-1), maka harus dilakukan proses reduksi. Pada Algoritma 10 proses reduksi merupakan proses penambahan antara polinom biner dengan *irreducible binary polynomial* pada derajat *m* sampai polinom biner memiliki derajat terbesar (*m*-1).

Algoritma 10 Reduksi Modular

```
Input: polinomial biner c(x) dengan derajat
terbesar 2m - 2
Output: c(x) mod f(x)
while deg(c) > (m-1)
 if (deg(c) – m) > 0
 then c(x).x(deg(c) – m)
c ← c + f
return c
```

IV. Implementasi dan Pengukuran Kinerja

Bagian ini merupakan implementasi algoritma-algoritma dibagian sebelumnya dengan menggunakan bahasa pemrograman python. Kemudian, akan dilakukan pengukuran kinerja berdasarkan waktu dan *memory* yang digunakan untuk memproses sebuah operasi. Pengujian akan dilakukan pada polinom biner dengan *m* adalah 4, 8, 16, 32, 64, 128, 256, dan 299.

Pengukuran waktu pemrosesan diukur dengan menggunakan *library time* pada python. Untuk mengukur kinerja waktu pemrosesan, setiap method dijalankan sebanyak seribu kali untuk *test case* yang sama kemudian diambil waktu rata-rata dalam satuan *milisecond*.

Pengukuran *memory* yang terpakai diukur dengan menggunakan *library* psutil pada phyton. Untuk setiap algoritma, pengujian dilakukan sebanyak seribu kali dan diambil nilai rata-rata pemakaian *memory* dalam satuan *megaByte* (MB).

Hasil implementasi dari algoritma untuk operasi penjumlahan, perkalian, perpangkatan dua (*squaring*), *invers*, dan reduksi diuji secara fungsional. Pengujian fungsional dilakukan secara manual, dimana nilai dari $a(x)$ dan $b(x)$ dimasukkan oleh peneliti. Pengujian fungsional akan dilakukan dengan menguji data benar, data salah (*abuse*), dan data ekstrim [22].

Pengujian data benar dilakukan untuk seluruh *method* hasil implementasi. Setiap jenis operasi akan diuji dengan panjang *key* 4 bit, 8 bit, 16 bit, 32 bit, 64 bit, 128 bit, 256 bit, dan 299 bit. Data yang digunakan sebagai data masukan dipilih secara acak dan merupakan polinomial biner $\mathbb{F}_2^m$ dengan derajat (*m*-1).

Pengujian data salah dilakukan untuk mengecek derajat dan polinom biner yang dimasukkan sebagai input. Pada pengujian data ekstrim, data yang diuji merupakan polinom biner pada batas minimum dan maksimum.

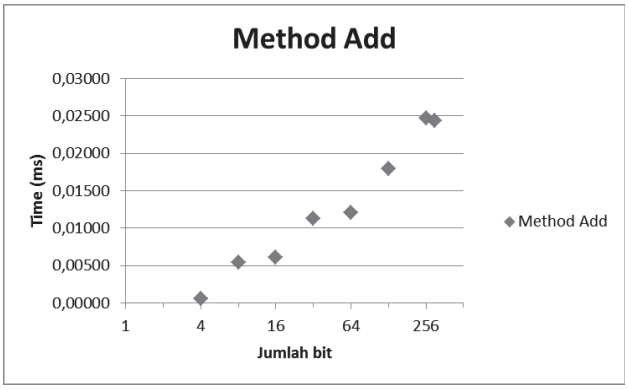
C. Penjumlahan pada $\mathbb{F}_2^m$

Gambar 2 merupakan hasil implementasi dari operasi penjumlahan (Algoritma 1). Pada proses ini dua buah polinomial biner langsung dioperasikan dengan XOR pada python.

```
def add(self, A, B):
 hasil=A^B
 return hasil
```

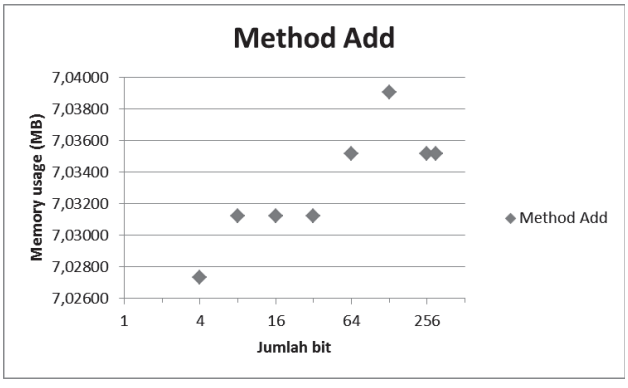
Gambar 2 Method Add

Gambar 3 merupakan grafik hasil pengukuran waktu untuk metode penjumlahan. Pada grafik ini terlihat, semakin panjang bit, maka waktu yang dibutuhkan semakin banyak.



Gambar 3 Hasil Pengukuran Waktu Operasi Penjumlahan

Gambar 4 merupakan hasil pengukuran *memory* untuk metode penjumlahan dalam skala kecil. Pada grafik terlihat, pada operasi beberapa bit yang berbeda, memory yang terpakai sama.



Gambar 4 Hasil Pengukuran *Memory* Operasi Penjumlahan

D. Perkalian pada $\mathbb{F}_2^m$

Gambar 5 merupakan hasil implementasi dari operasi perkalian yang pertama yaitu dengan mengalikan setiap bit dari polinomial biner (Algoritma 2).

```
def binMul(self, A, B):
 bitA=bin(A)[2:]
 bitB=bin(B)[2:]

 if bitB[self.countDegree(B)]== '0':
 C=0
 else:
 C=A

 for i in range(self.countDegree(B)-1, -1,-1):
 if bitB[i]=='0':
 temp=0
 else:
 temp=A
 temp=temp<<(self.countDegree(B)-i)
 C^=temp

 if self.countDegree(C)>(self.degree-1):
 return self.binRed(C)
 else:
 return C
```

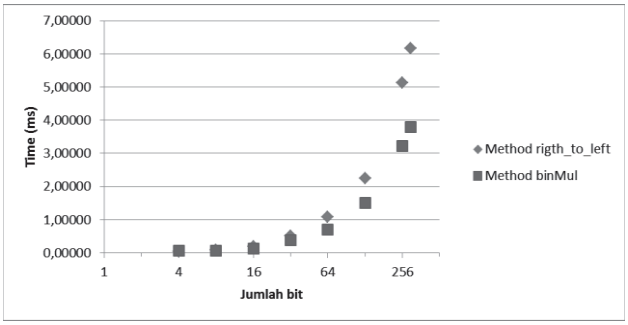
Gambar 5 Method binMul

Gambar 6 merupakan implementasi dari metode perkalian *Right-to-left shif-and-add* (Algoritma 3) [18]. Pada kedua metode yang digunakan ini, jika hasil perkalian melebihi (m-1) maka harus dilakukan reduksi.

```
def rigth_to_left(self, A, B):
 bitA=bin(A)[2:]
 if bitA[(self.countDegree(A))]=='1':
 C=B
 else:
 C=0

 i=self.countDegree(A)-1
 while i>=0:
 B=B<<1
 if self.countDegree(B)>=self.degree:
 B=self.binRed(B)

 if bitA[i]=='1':
 C^=B

 i-=1
 return C
```

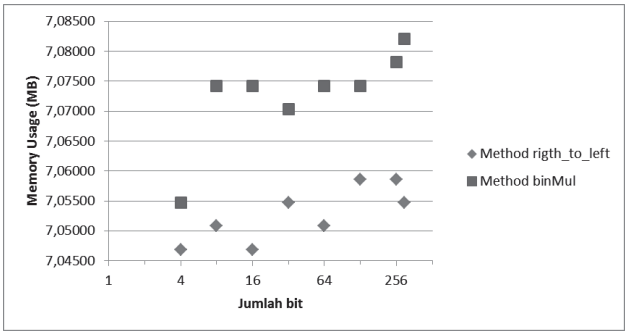
Gambar 6 Method right_to_left

Gambar 7 merupakan hasil pengukuran waktu operasi perkalian dengan menggunakan metode perkalian setiap bit dan metode *Right-to-left shif-and-add*. Dapat terlihat bahwa *method* binMul() membutuhkan waktu pemrosesan lebih cepat untuk jumlah *key* yang lebih dari 32 bit.



Gambar 7 Hasil Pengukuran Waktu Operasi Perkalian

Gambar 8 merupakan hasil pengukuran *memory* yang terpakai dalam operasi perkalian dalam skala kecil. Terlihat dari grafik, method binMul() walaupun membutuhkan waktu yang lebih sedikit, tapi memakai *memory* yang lebih besar dibandingkan dengan metode *Right-to-left shif-and-add*. Sedangkan metode *Right-to-left shif-and-add* membutuhkan waktu pemrosesan yang lebih lama, namun memakai *memory* yang lebih sedikit.



Gambar 8 Hasil Pengukuran *Memory* Operasi Perkalian

E. Perpangkatan Dua (Squaring) pada $\mathbb{F}_2^m$

Gambar 9 merupakan implementasi dari metode perpangkatan dua dengan mengalikan sebuah polinomial dengan dirinya sendiri. Pada implementasi ini, metode perkalian yang dipakai adalah Algoritma 2.

```
def binSquare(self, A):
 return self.binMul(A,A)
```

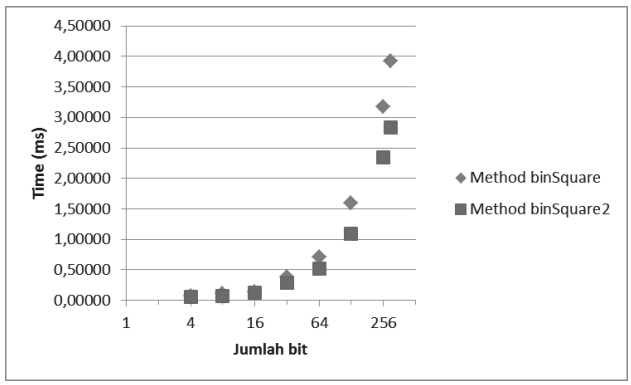
Gambar 9 Method binSquare

Gambar 10 merupakan implementasi dari Algoritma 5 namun ditambahkan dengan proses reduksi polinomial jika derajat dari hasil perkalian melebihi (*m-1*).

```
def binSquare2(self, A):
 if A==0:
 return 0
 bitA=bin(A)[2:]
 C=[""]*(((self.countDegree(A)+1)*2)-1)
 x=0;
 for i in range(0,((self.countDegree(A)+1)*2)-1,1):
 if i%2==0:
 C[i]=bitA[x]
 x+=1
 else:
 C[i]='0'
 C="".join(C)
 if self.countDegree(C)>(self.degree-1):
 return self.binRed(int(C,2))
 else:
 return int(C,2)
```

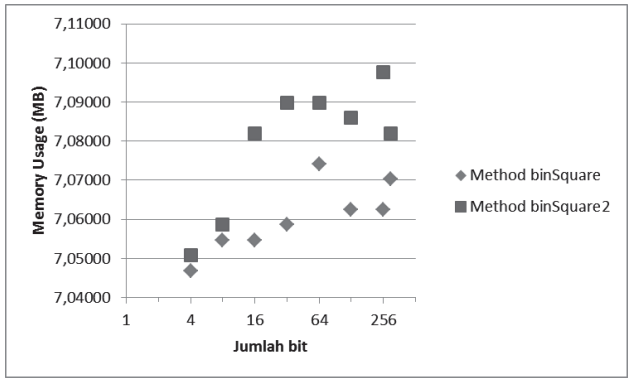
Gambar 10 Method binSquare2

Gambar 11 merupakan hasil pengukuran waktu dari operasi perpangkatan dua (*squaring*). Terlihat dari grafik, bahwa metode binSquare lebih cepat untuk pemrosesan key yang panjang.



Gambar 11 Hasil Pengukuran Waktu Operasi Perpangkatan Dua

Gambar 12 merupakan hasil pengukuran *memory* yang terpakai untuk melakukan proses perpangkatan dua dalam skala kecil. Terlihat dari grafik, bahwa *method* binSquare(), walaupun membutuhkan waktu yang lebih banyak, namun memakai *memory* yang lebih sedikit dibandingkan *method* binSquare2(). Sedangkan method binSquare2() membutuhkan waktu pemrosesan yang lebih sedikit namun memakai *memory* yang lebih banyak.



Gambar 12 Hasil Pengukuran *Memory* Operasi Perpangkatan Dua

F. Invers pada $\mathbb{F}_2^m$

Gambar 13 merupakan hasil implementasi operasi invers dengan menggunakan Algoritma Extended Euclidean (Algoritma 6) [18].

```
def extEuclidean(self, A):
 if A==0:
 print "Angka tidak boleh nol"
 return 0
 temp=0
 u=A
 v=int(self.bitIrredPol,2)
 g1=1
 g2=0

 while u!=1:
 j=self.countDegree(u)-self.countDegree(v)

 if j<0:
 temp=u
 u=v
 v=temp

 temp=g1
 g1=g2
 g2=temp
 j= j*(-1)

 u=self.add(u,(v<<j))
 g1=self.add(g1,(g2<<j))

 return g1
```

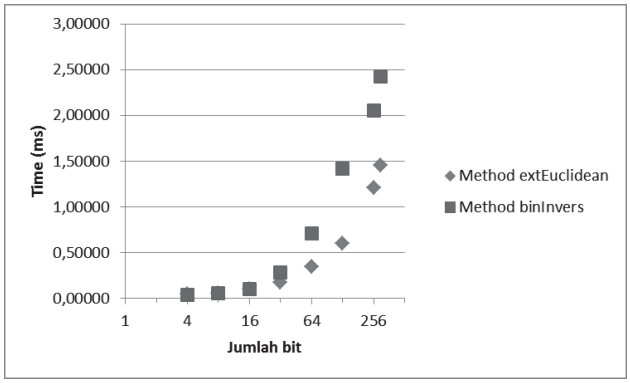
Gambar 13 Method extEuclidean

Gambar 14 merupakan hasil implementasi algoritma Biner untuk operasi *invers* (Algoritma 7).

```
def binInvers(self, A):
 if A==0:
 print "Angka tidak boleh nol"
 return 0
 u=A
 v=int(self.bitIrredPol,2)
 f=int(self.bitIrredPol,2)
 g1=1
 g2=0
 while u!=1 and v!=1:
 while u&1==0:
 u=u>>1
 if g1&1==0:
 g1=g1>>1
 else:
 g1=(g1^f)>>1
 while v&1==0:
 v=v>>1
 if g2&1==0:
 g2=g2>>1
 else:
 g2=(g2^f)>>1
 if self.countDegree(u)>self.countDegree(v):
 u=u^v
 g1=g1^g2
 else:
 v=v^u
 g2=g2^g1

 if u==1:
 return g1
 else:
 return g2
```

Gambar 14 Method binInvers

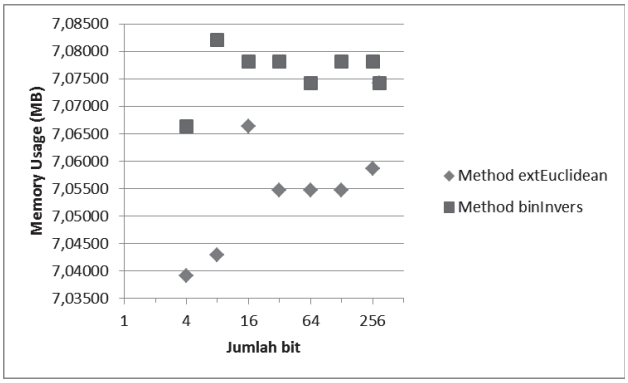
Gambar 15 merupakan hasil pengukuran waktu untuk operasi *invers*. Dapat terlihat dari grafik, metode Extended Euclidean lebih cepat untuk melakukan proses *invers*, terutama untuk *key* yang panjang.



Gambar 15 Hasil Pengukuran Waktu Operasi Invers

Gambar 16 merupakan hasil pengukuran memory pada operasi *invers* dalam skala kecil. Terlihat dari grafik, metode Extended Euclidean membutuhkan waktu pemrosesan yang lebih cepat dan *memory* yang terpakai lebih sedikit. Sedangkan metode binInvers() membutuhkan

waktu pemrosesan yang lebih lama dan memakai *memory* yang lebih banyak.



Gambar 16 Hasil Pengukuran *Memory* Operasi Invers

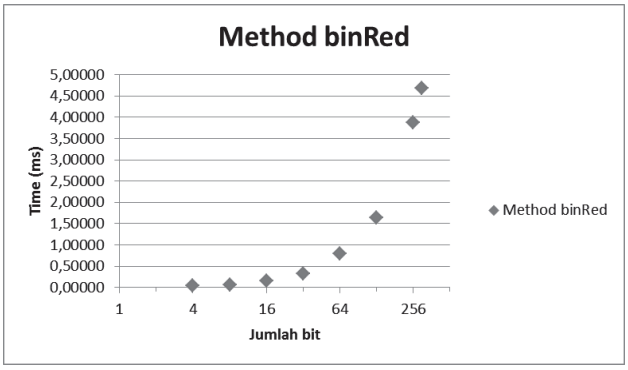
G. Reduksi pada $\mathbb{F}_2^m$

Metode reduksi digunakan ketika hasil dari operasi aritmatika melebihi (m-1). Pada proses reduksi ini polinom biner akan dijumlahkan dengan *irreducible binary polynomial* pada derajat *m* sampai polinom biner memiliki derajat terbesar (*m*-1).

```
def binRed(self,A):
 iPol=0;
 while(self.countDegree(A) > (self.degree-1)):
 iPol=int(self.bitIrredPol,2)
 if (self.countDegree(A)-self.degree)>0:
 iPol=iPol<<(self.countDegree(A)- self.degree)
 A^=iPol
 return A
```

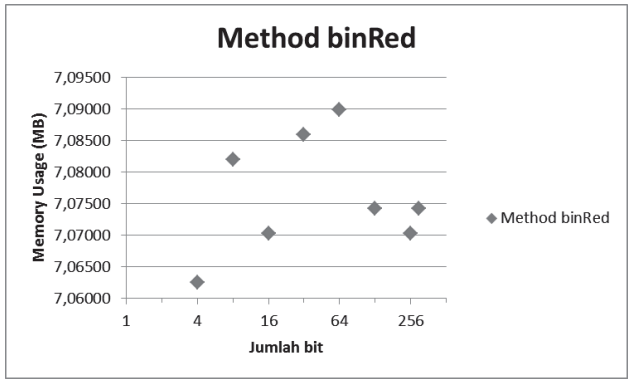
Gambar 17 Method binRed

Gambar 18 merupakan hasil pengukuran waktu untuk operasi reduksi. Semakin panjang *key*, maka waktu yang dibutuhkan semakin banyak.



Gambar 18 Hasil Pengukuran Waktu Operasi Reduksi

Gambar 19 merupakan hasil pengukuran pemakaian *memory* untuk operasi reduksi dalam skala kecil.



Gambar 19 Hasil Pengukuran *Memory* Operasi Reduksi

Berdasarkan hasil implementasi dan pengukuran kinerja, metode untuk penjumlahan hanya dapat dilakukan dengan melakukan operasi XOR pada dua buah polinomial biner. Semakin panjang bit, maka semakin banyak waktu yang dibutuhkan untuk menyelesaikan operasi. Untuk metode pada operasi perkalian dapat disesuaikan dengan kebutuhan. Jika penggunaan berfokus pada kebutuhan waktu yang lebih cepat, maka algoritma 2 dapat digunakan. Namun jika penggunaan berfokus pada keterbatasan *memory*, maka algoritma 3 lebih baik dibandingkan algoritma 2. Pada operasi perpangkatan dua (*squaring*), jika penggunaan berfokus pada kebutuhan waktu yang lebih cepat, maka dianjurkan menggunakan algoritma 5. Namun jika penggunaan berfokus pada besar *memory* yang dibutuhkan, maka algoritma 4 lebih baik dibandingkan algoritma 5. Untuk operasi *invers*, algoritma Extended Euclidean memiliki kinerja yang lebih baik dari segi waktu dan *memory* yang dibutuhkan dibandingkan algoritma 7.

V. SIMPULAN

Adapun simpulan yang didapatkan selama pelaksanaan penelitian ini adalah sebagai berikut:

1. Telah diimplementasikan operasi-operasi aritmatika pada *finite field* dalam bahasa pemrograman python dengan representasi pada polinomial biner dengan panjang *key* maksimum 299 bit. Fungsi operasi aritmatika yang disediakan adalah operasi penjumlahan, perkalian, perpangkatan dua, dan *invers*.
2. Pengujian dilakukan dengan menguji dari segi fungsionalitas dan segi kinerja dalam waktu pemrosesan serta pemakaian *memory*. Berdasarkan hasil pengujian, perangkat lunak dapat melakukan perhitungan untuk polinomial biner dengan panjang *key* maksimum 299 bit.
3. Berdasarkan hasil pengukuran kinerja, sebuah metode bisa lebih cepat dalam hal waktu pemrosesan, tapi membutuhkan ruang *memory* yang lebih banyak atau sebaliknya.

DAFTAR PUSTAKA

[1] K. Rabah, "Elliptic Curve Cryptography," Information Security Research Journal, vol. 1, no. 1, pp. 2-5, 2009.
[2] R. Sadikin, Kriptografi untuk Keamanan Jaringan, Yogyakarta: ANDI, 2012.
[3] V. Kapoor, V. S. Abraham dan R. Singh, "Elliptic Curve Cryptography," ACM Abiquity, vol. 9, no. 20, pp. 1-8, 2008.
[4] W. Stallings, Cryptography and Network Security 3rd ed, New York: McGraw-Hill, 2003.
[5] D. Hankerson, A. Menezes dan S. Vanstoe, Guide to Elliptic Curve Cryptography, New York: Springer, 2004.
[6] N. Daswani, C. Kern dan A. Kesavan, Foundations of Security: What Every Programmer Needs to Know, New York: Springer Verlag, 2007.
[7] M. W. Paryasto, K. S. Sutikno dan A. Sasongko, "Issues in Elliptic Curve Cryptography Implementation," Internetworking Indonesia Journal, vol. 1, no. 1, pp. 29-33, 2009.
[8] C. Paar dan J. Pelzl, Understanding Cryptography, New York: Springer, 2010.
[9] M. W. Paryasto, B. Rahardjo, F. Yuliawan, I. M. Alamsyah dan Kuspriyanto, "Composite Field Multiplier Based on Look-Up Table for Elliptic Curve Cryptography Implementation," ITB J.ICT, vol. 6, no. 1, pp. 63-81, 2010.
[10] Z. Li, J. Higgins dan M. Clement, "Performance of Finite Field Arithmetic in an Elliptic Curve Cryptosystem," Proc. 9th Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 249-256, 2001.
[11] R. Shanmugalakshmi dan M. Prabu, "Research Issues on Elliptic Curve Cryptography and Its Applications," IJCSNS, vol. 9, no. 6, pp. 19-22, 2009.
[12] Python Software Foundation, "General Python FAQ,"[Online]. Available: docs.python.org/2/faq/general.html#what-is-python.[Diakses 1 Juni 2014].
[13] M. W. Paryasto, "Arsitektur Unit Pengali Composite Field Kombinasi MH-KOA untuk ECC," Disertasi, STEI Institut Teknologi Bandung, Bandung, 2012.
[14] G. Seroussi, "Table of Low-Weight Binary Irreducible Polynomials," Hewlett Packard Company, 1998.
[15] M. Rosing, Implementing Elliptic Curve Cryptography, Greenwich: Manning Publications Co., 1999.
[16] J. C. Bajard, L. Imbert, C. Negre dan T. Plantard, "Efficient Multiplication in GF(pk) for Elliptic Curve Cryptography," dalam 16th IEEE Symposium on Computer Arithmetic (ARITH'03), 2003.
[17] A. Weimerskirch, D. Stebila dan S. C. Shantz, "Generic GF(2m) Arithmetic in Software and its Application to ECC," dalam The Eighth Australasian Conference on Information Security and Privacy (ACISP 2003), Wollongong, 2003.
[18] M. Maulana, W. F. Senjaya, B. Rahardjo, I. Muchtadi-Alamsyah dan M. W. Paryasto, "Implementation of Finite Field Arithmetic Operations for Polynomial and Normal Basis Representations," dalam 3rd International Conference on Computation for Science and Technology (ICCST-3), Denpasar, 2015.
[19] S. Baktir. dan B. Sunar., "Finite Field Polynomial Multiplication in the Frequency Domain with Application to Elliptic Curve Cryptography," Computer and Information Sciences-ISCIS 2006, vol. 4263, pp. 991-1001, 2006.
[20] S. M. Shohdy, A. B. El-Sisi dan N. Ismail, "Hardware Implementation of Efficient Modified Karatsuba Multiplier Used in Elliptic Curves," International Journal of Network Security, vol. 11, no. 3, pp. 155-162, Nov 2010.
[21] R. S. Pressman, Software Engineering, 7th ed., New York: McGraw-Hill, 2003.
[22] B. Rahadjo, "Keamanan Perangkat Lunak," PT Insan Infonesia, 2014