Perbandingan Kriptanalisis RSA dan Schmidt Samoa menggunakan metode faktorisasi elliptic curve dan quadratic sieve

Adhitya Ramadhanus / 13511032
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511032@std.stei.itb.ac.id

***Abstract*—Keamanan kriptografi kunci public seperti Schmidt-Samoa dan RSA diperoleh dari besarnya sumber daya yang dibutuhkan untuk melakukan faktoriasi bilangan bulat yang besar. Algoritma faktorisasi elliptic curve memiliki kompleksitas yang bergantung pada faktor dari suatu bilangan sedangkan quadratic sieve termasuk salah satu general purpose factoring algorithm sehingga kompleksitas quadratic sieve lebih bergantung pada ukuran bilangan tersebut. Kunci public Schmidt-Samoa dan RSA sama-sama dibentuk dari dua buah bilangan prima, akan tetapi untuk menghasilkan ukuran kunci yang sama maka ukuran bilangan prima yang dibutuhkan untuk Schmidt-Samoa dan RSA berbeda. Pada makalah ini akan dibandingkan performa algoritma faktorisasi elliptic curve dan quadratic sieve pada kunci public RSA dan Schmidt-Samoa.**

Keywords—Quadratic Sieve, Elliptic Curve Factorization, RSA, Schmidt-Samoa CryptoSystem, Integer Factorization, Cryptanalysis.

I. PENDAHULUAN

Pada saat ini jumlah informasi dan pihak-pihak yang terlibat dalam suatu komunikasi melalui internet semakin meningkat. Hal ini menyebabkan meningkatnya kebutuhkan akan security property seperti confidentiality, authenticity dan integrity. Kriptografi kunci public (asymmetric) dan kriptografi kunci privat (symmetric) dapat digunakan secara bersamaan untuk menghasilkan confidentiality,authenticity, dan integrity.

Berbeda dengan kriptografi kunci privat, kunci yang digunakan untuk enkripsi berbeda dengan kunci yang digunakan untuk dekripsi pada kriptografi kunci publik. Kunci publik dan kunci privat pada kriptografi kunci public Schmidt-Samoa dan RSA memiliki keterkaitan satu sama lain. Meskipun begitu, untuk menghasilkan kunci privat dari kunci public membutuhkan sumber daya komputasi yang sangat besar untuk ukuran kunci public yang besar.

Kriptanalisis kriptografi kunci publik dan kriptografi kunci privat sangatlah berbeda. Pada kriptografi kunci privat, terdapat beberapa teknik yang secara umum dapat diterapkan pada semua kriptografi kunci privat seperti Linear cryptanalysis dan differential cryptanalysis. Hal ini disebabkan kriptografi kunci privat memiliki kesamaan pada komponen-komponennya seperti terdapatnya P-Box dan S-Box pada setiap block cipher. Pada kriptografi kunci publik, kriptanalisis dilakukan dengan teknik yang menyesuaikan dengan karakteristik kriptografi kunci publik tersebut.

Pada kriptografi kunci publik RSA dan Schmidt-Samoa, teknik kriptanalisis paling intuitif yang dapat dilakukan adalah dengan melakukan faktorisasi kunci publik kedua kriptografi tersebut. Dengan mengetahui faktor dari kunci publik RSA dan Schmidt-Samoa maka kunci privat RSA dan Schmidt-Samoa dapat ditemukan dengan cepat. Faktorisasi ini tentunya membutuhkan algoritma dengan kompleksitas polynomial sehingga serangan ini dapat diimplementasikan untuk menyerang RSA dan Schmidt-Samoa namun hingga saat ini belum terdapat algoritma faktorisasi dengan kompleksitas polynomial.

Secara umum, terdapat 2 kategori algoritma faktorisasi yaitu Special-purpose dan General-purpose. Algoritma faktorisasi Special-purpose memiliki kompleksitas yang bergantung pada karakteristik suatu angka atau faktor yang membentuk angka tersebut sedangkan algoritma faktorisasi General-purpose lebih bergantung pada ukuran angka yang difaktorkan. Contoh algoritma faktorisasi special-purpose adalah pollar p-1, pollar rho, elliptic curve factorization. Contoh algoritma faktorisasi general-purpose adalah Dixon, quadratic sieve dan general number field sieve.

Pada tahun 1981, Carl Pomerance menciptakan sebuah algoritma faktorisasi bernama quadratic sieve. Algoritma ini merupakan pengembangan dari algoritma faktorisasi Dixon dan Kraitchik. Quadratic sieve adalah algoritma faktorisasi tercepat untuk ukuran angka mencapai 110 digit [7].

Contini pada tesisnya tahun 1997 [1] mengimplementasikan sebuah variant quadratic sieve bernama self-initializing quadratic sieve. Pada tesis ini dibuktikan bahwa siqs atau self-initializing quadratic sieve memiliki kecepatan diatas varian quadratic sieve lainnya seperti basic quadratic sieve maupun multiple polynomial quadratic sieve.

Metode faktorisasi elliptic curve (Elliptic Curve Method atau ECM) diciptakan pada tahun 1985 oleh H.W. LEnstra, Jr.

Meskipun jarang digunakan dalam usaha pemecahan rekor faktorisasi bilangan tapi algoritma ini memiliki performa yang cukup baik dan mudah diimplementasikan.

Arjen Lenstra dan Web Bosma [2] mengimplementasikan ECM pada tahun 1991 dengan beberapa optimasi. Optimasi-optimasi tersebut antara lain penggunaan Montgomery curve pada operasi elliptic curve dan penggunaan dua tahap pada ECM jika tahap pertama dalam ECM gagal mendapatkan faktor.

Pada makalah ini akan dilakukan analisis terhadap performansi algoritma quadratic sieve dan metode faktorisasi elliptic curve pada kriptanalisis kriptografi RSA dan Schmidt-Samoa. Bagian II berisi dasar teori dan penjelasan mengenai tahapan algoritma quadratic sieve, metode faktorisasi elliptic curve dan kriptografi RSA dan Schmidt-Samoa. Bagian III berisi implementasi algoritma quadratic sieve dan metode faktorisasi elliptic curve. Bagian IV berisi analisis perfoma algoritma quadratic sieve dan metode faktorisasi elliptic curve. Bagian V berisi kesimpulan dari hasil yang didapat pada makalah ini.

II. DASAR TEORI

A. *Quadratic Sieve*

Quadratic Sieve atau QS berusaha mencari dua buah bilangan x dan y dengan $x \neq y \pmod n$ dan $x^2 \equiv y^2 \pmod n$ sehingga $(x-y)(x+y) \equiv 0 \pmod n$. Setelah itu dengan menghitung gcd (x-y,n) (greatest common divisor) maka terdapat kemungkinan 50% bahwa hasil gcd(x-y,n) adalah faktor non-trivial dari n (faktor non-trivial artinya bukan 1 dan n). Langkah awal paling intuitif dalam pencarian x dan y adalah dengan mengawali pencarian dari bilangan $b = \sqrt{n}$ menggunakan persamaan 1.

$$G(x) = (x + \lceil\sqrt{n}\rceil)^2 - n \quad (1)$$

Dengan x=0,1,2….2m untuk suatu bilangan bulat m. Nilai m adalah interval sieving yang akan digunakan pada tahap sieving nanti.

Setelah itu yang perlu dilakukan adalah mencari bilangan G(x) yang dapat difaktorkan dengan suatu basis faktor prima tertentu. G(x) ini disebut sebagai bilangan smooth. Basis faktor prima adalah kumpulan bilangan prima yang kurang dari suatu batas F dengan syarat n adalah quadratic residue dari setiap bilangan prima tersebut atau dengan kata lain terdapat solusi terhadap persamaan 2.

$$t^2 = n \pmod p \quad (2)$$

dengan p adalah bilangan prima pada Basis faktor prima.

Jika G(x) dapat difaktorkan dengan basis faktor prima tertentu maka akan terbentuk persamaan $u^2 = v \bmod N$ dengan v = G(x) dan u = x + b. Bilangan G(x) yang dapat difaktorkan dengan bilangan prima p dapat dicari dengan menggunakan persamaan 3.

$$x = \pm t - b \bmod p. \quad (3)$$

Perlu diperhatikan bahwa persamaan 3 memiliki 2 solusi sehingga terdapat dua x dimana G(x) dapat dibagi dengan p. Selain itu, jika G(x) habis dibagi p maka G(x+p) juga habis dibagi p. Hal ini dapat dibuktikan sebagai berikut :

1. Jika $p \mid G(x)$ maka $p \mid (x+\sqrt{n})^2 - n$
2. $G(x+p) = ((x + \sqrt{n}) + p)^2 - n$
3. $((x + \sqrt{n}) + p)^2 - n = ((x+\sqrt{n})^2 - n) + p\,(2(x + \sqrt{n}) + p) = G(x) + p\,(2(x+\sqrt{n}) + p)$ sehingga $p \mid G(x + p)$

Hal ini juga berlaku untuk kelipatan p.

Dengan mengetahui fakta ini maka dapat diketahui bilangan G(x) yang habis dibagi dengan p sehingga bilangan smooth dapat dicari dengan cepat. Misalkan n=90283,F=44, b=301 dan m=30, maka basis faktor prima adalah himpunan {2,3,7,17,23,29,37,41}. Bilangan G(x) untuk x=0,1,2…60 adalah{318,921,1526,2133,2742,3353,3966,4581,5198,5817,6438,7061,7686,8313,8942,9573,10206,10841,11478,12117,12758,13401,14046,14693,15342,15993,16646,17301,17958,18617,19278,19941,20606,21273,21942,22613,23286,23961,24638,25317,25998,26681,27366,28053,28742,29433,30126,30821,31518,32217,32918,33621,34326,35033,35742,36453,37166,37881,38598,39317}. Untuk p=2 maka solusi persamaan 3 adalah $x_1=0$ dan $x_2=0$ sehingga setiap bilangan G(x) dengan x genap dapat dibagi 2. Untuk p = 3 maka solusi persamaan 3 adalah $x_1=0$ dan $x_2=1$ sehingga untuk x=0,1,3,4,6,7… dst dapat dibagi dengan 3. Hal ini dilakukan hingga semua bilanga prima pada basis faktor telah digunakan. Langkah ini mempercepat proses penemuan bilangan smooth. Langkah ini dinamakan proses sieving.

Setelah sieving, proses selanjutnya adalah mencari solusi $(G(x1)\ G(x2)\ G(x3)\ G(xn))2 = (xp_1,xp_2,xp_3,xp_k)^2 \pmod n$ dengan xp adalah faktor prima dari G(x1) G(x2) G(x3) dst dan $x_1,x_2,x_3 \ldots x_n$ adalah indeks bilangan smooth. Perkalian suatu bilangan dapat dipandang sebagai penambahan pangkat setiap faktor primanya dan suatu bilangan kuadrat selalu memiliki pangkat faktor prima yang genap. Hal ini dimanfaatkan dalam algoritma ini. Misalkan dengan contoh kasus sebelumnya, bilangan smooth yang dihasilkan dari proses sebelumnya dan vector pangkat faktorisasi primanya dapat dilihat pada Tabel 1.

Tabel 1. Bilangan Smooth dan faktorisasinya

x	Vektor pangkat faktorisasi Prima
311	1 1 0 0 0 1 1 0
317	1 0 1 0 0 0 0 0
327	1 0 1 0 0 1 0 1
331	1 0 1 1 0 0 0 0
332	0 1 0 0 1 0 0 0
348	0 0 0 1 0 0 1 0
355	1 1 1 0 1 0 1 0

Langkah selanjutnya adalah menggunakan aljabar linear untuk mengetahui solusi x yang menghasilkan vektor 0 (dengan kata lain menghasilkan bilangan kuadrat). Solusi untuk kasus diatas adalah $(331 * 332 * 348 * 355)^2 = (G(331)$

* G(332) * G(348) * G(355))^2 (mod n) dan menghasilkan faktor 137 dan 659.

B. *Elliptic Curve Method*

Pada Elliptic Curve, operasi penambahan dan penggandaan titik melibatkan operasi modular inverse. Misalkan untuk sebuah kurva pada medan Fp yang didefinisikan pada persamaan 4.

$$Y^2 \bmod p = x^3 + ax + b \pmod p \quad (4)$$

dengan $4a^3 + 27b^2 \equiv 0 \pmod p$ maka operasi penambahan dan penggandaan suatu titik pada kurva didefinisikan seperti dibawah.

1) Penambahan titik

Misal dua buah titik P (x_p,y_p) dan Q(x_q,y_q) pada suatu kurva C maka R = (x_r,y_r) = P + Q = $(s^2 - x_p-x_q, -y_p+s(x_p-x_r))$, semua operasi dilakukan dalam modulus p. S didefinisikan pada persamaan 5.

$$S = (y_p-y_q)/(x_p-x_q) \bmod p \quad (5)$$

2) Penggandaan titik

Misal P(x_p,y_p) adalah suatu titik pada kurva dengan $y_p \neq 0$ maka R = 2P = (xr,yr) = $(s^2-2x_p,-y_p+s(x_p-x_r))$. S didefinisikan pada persamaan 6.

$$S = (3x_p^2+a)/(2y_p) \bmod p \quad (6)$$

Dapat dilihat bahwa operasi penambahan dan penggandaan titik (dan juga perkalian skalar) melibatkan operasi modular inverse. Modular inverse untuk a modulus p ada jika dan hanya jika a dan p relatif prima (gcd(a,m)=1). Dengan kata lain jika operasi dalam elliptic curve gagal untuk suatu a maka gcd(a,p) $\neq$ 1. Hal ini dapat digunakan untuk faktorisasi dengan mengganti p menjadi bilangan yang akan difaktorkan dan mencoba mengalikan suatu titik sembarang pada kurva dengan suatu bilangan skalar.

Elliptic Curve Method dapat diringkas sebagai berikut:

1. Pilih kurva sembarang modulus n (n adalah bilangan yang akan difaktorkan) lalu pilih titik P sembarang pada kurva.
2. Pilih bilangan bulat k yang sesuai lalu coba lalukan perkalian scalar Q = kP

Untuk mempercepat komputasi, dapat digunakan elliptic curve Montgomery form seperti pada persamaan 7. Pada model ini, operasi modular inverse tidak diperlukan lagi dalam operasi penambahan maupun penggandaan titik namun faktorisasi tetap dapat dilakukan.

$$by^2 = x^3 + ax^2 + x \quad (7)$$

dengan a,b $\ni$ Zp dan $a^2 \neq 4$ dan b $\neq$ 0. Dengan menggunakan koordinat projective (homogenous) maka persamaan 7 menjadi

$$by^2z = x^3 + ax^2z + xz2 \quad (8)$$

Sebuah titik pada kurva direpresentasikan sebagai pasangan (x,z). Operasi penambahan dan penggandaan titk pada kurva Montgomery dapat dilihat dibawah.

3) Penambahan titik (Montgomery Curve)

Misal P1(x_1,z_1) dan P2(x_2,z_2) adalah titik-titik pada kurva dan titik P0(x_0,z_0) = P1-P2. R(xr,zr) = P1 + P2 didefinisikan pada persamaan 9 dan 10.

$$x_r = ((x_1-z_1) * (x_2+z_2) + (x_1+z_1) * (x_2-z_2))^2 * z_0 \pmod n \quad (9)$$

$$z_r = ((x_1-z_1) * (x_2+z_2) - (x_1+z_1) * (x_2-z_2))^2 * x_0 \pmod n \quad (10)$$

4) Penggandaan titik (Montgomery Curve)

Misal P(x_p,z_p) adalah titik pada kurva. Titik R (x_r,z_r) = 2P didefinisikan pada persamaan 11 dan 12.

$$x_r = (x_p+z_p)^2 *(x_p-z_p)^2 \pmod n \quad (11)$$

$$z_r = T((x_p,z_p)^2 + ST) \pmod n \quad (12)$$

Dengan S = (a + 2)/4 dan T = $(x_p+z_p)^2 - (x_p-z_p)^2$

Algoritma ECM yang diimplementasikan pada makalah ini dapat dilihat pada Gambar 1.

for $p \in \pi(2\ldots B1)$ //Tahap 1
    $a = \lfloor \log_p B_1 \rfloor$
    $Q = p^a \otimes_C Q$
g = gcd(Q_z,N)
If (1<g<n) return g
for $p \in \pi(B1\ldots B2)$ //Tahap 2
    $Q = p \otimes_C Q$
    $g = g \times Q_z \pmod n$
g= gcd(g,N)
If (1<g<n) return g
return failure

Gambar 1. Pseudo-code ECM

π Adalah himpunan bilangan prima dari 2 hingga batas B1, $\otimes_C$ adalah perkalian skalar titik pada suatu kurva C.

C. *Kriptografi RSA*

RSA adalah kriptografi kunci publik yang diciptakan Ron Rivest, Adi Shamir dan Leonard Adleman pada tahun 1978. RSA menggunakan 2 jenis kunci yaitu kunci publik dan kunci privat. Kunci public RSA terdiri atas pasangan bilangan n dan e sedangkan kunci privat RSA terdiri atas pasangan bilangan n dan d. Kunci publik dan kunci privat RSA dihasilkan melalui prosedur berikut :

1. Pilih secara acak dua buah bilangan prima p dan q
2. Hitung n = pq
3. Hitung $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$ lalu pilih $1< e < \varphi(n)$ dengan e dan $\varphi(n)$ relatif prima
4. Hitung $d = e^{-1} \pmod{\varphi(n)}$

Proses enkripsi dan dekripsi dalam RSA didefinisikan sebagai berikut :

1. Misal m (bilangan bulat) adalah pesan yang akan dikirim dan c adalah cipherteks.
2. Enkripsi : Hitung $c = m^e \pmod n$
3. Dekripsi : Hitung $m = c^d \pmod n$

D. Kriptografi Schmidt-Samoa

Schmidt-Samoa adalah kriptografi kunci publik yang diciptakan oleh Katja Schmidt-Samoa. Kriptografi ini mengandalkan trapdoor function yang mirip dengan kriptografi Rabin. Kunci publik Schmidt-Samoa adalah sebuah bilangan bulat N dan kunci privat Schmidt-Samoa adalah bilangan bulat d,p, dan q. Generasi kunci publik dan kunci privat Schmidt-Samoa adalah sebagai berikut :

1. Pilih dua buah bilangan prima p dan q lalu hitung N = p^2q
2. Hitung d = N^{-1} (mod lcm (p-1,q-1))

Proses enkripsi dan dekripsi pada kriptografi Schmidt-Samoa didefinisikan sebagai berikut :

1. Misal m adalah pesan yang akan dikirm dan c adalah cipherteks
2. Enkripsi : Hitung c = m^N (mod N)
3. Dekripsi : Hitung m = c^d (mod pq)

III. IMPLEMENTASI QUADRATIC SIEVE DAN ELLIPTIC CURVE FACTORIZATION

A. Quadratic Sieve

Secara umum, terdapat 3 proses pada algoritma quadratic sieve yaitu Inisialisasi data, sieving dan gaussian elimination.

1) Inisialisasi data

Pada proses ini, variabel n, F, m, b dan basis faktor prima ditetapkan terlebih dahulu. N adalah bilangan yang difaktorkan, b adalah akar kuadrat n , F adalah batas basis faktor prima dan m adalah interval proses sieving. F pada implementasi ini didefinisikan pada persamaan 13.

$$F = e^{((0.5 + o(1) * (\ln(n) * \ln \ln(n))1/2)} \quad (13)$$

Sedangkan m didefinisikan dengan perkiraan saja yaitu sekitar ≈ 2F. Bilangan prima p < F yang akan dimasukkan kedalam basis faktor prima dihitung menggunakan Jacobi Symbol (n/p). Jika Jacobi Symbol (n/p) bernilai 1 maka p dimasukkan kedalam basis faktor prima. Jacobi Symbol didefinisikan dalam persamaan 14

$$(n/m) = (n/p_1)^{a1} (n/p_2)^{a2} \dots (n/p_k)^{ak} \quad (14)$$

Dengan m adalah bilangan bulat positif ganjil dan $p_1^{a1}, p_2^{a2} \dots p_k^{ak}$ adalah faktorisasi prima m.

Algoritma shanks-tonelli digunakan untuk mencari solusi t^2 = n (mod p) yang nantinya akan digunakan secara tidak langsung dalam proses sieving. Hitung himpunan bilangan G(x) untuk x=0,1,2…2m sesuai dengan persamaan 1. Definisikan untuk setiap bilangan prima p dalam basis faktor prima x_{1p} = t – b (mod p) dan x_{2p} = (-t - b) (mod p).

Implementasi tahap inisialisasi quadratic sieve pada makalah ini dapat dilihat dibawah.

```
BigInteger b = Sqrt(n);
 if (b*b < n) b++;
//Determine Factor Base
 BigInteger X =
NextPrime(BigInteger(2));
 vector<BigInteger> FactorBase;
 FactorBase.push_back(BigInteger(2));
//2 Selalu ada
 while(X<F){
        if (Jacobi(n,X)==1){
            FactorBase.push_back(X);
        }
        X = NextPrime(X);
    }
int sz = FactorBase.size();
vector<BigInteger> SmoothNumbers;
//Smooth Numbers Exponent Vector Array
vector<unsigned int> SmoothX; //Keep X
of Smooth Numbers
BigInteger Accu[m*2]; //Himpunan G(x)
for(int i=0;i<m*2;i++){
            Accu[i] = (b+i)*(b+i)- n;
            Accu[i] = Accu[i] % n;}
```

Meskipun perhitungan t untuk setiap bilangan prima termasuk dalam tahap inisialisasi, namun dalam implementasi ini perhitungan t dilakukan pada tahap sieving.

2) Sieving

Dalam proses ini, Bagi $G(x_{1p}+ip)$ dan $G(x_{2p}+ip)$ dengan bilangan prima p dalam basis faktor prima dengan i=0,1,2..n sedemikan sehingga np <= 2m. Simpan semua bilangan bernilai 1 dalam himpunan bilangan G(x). Bilangan ini adalah bilangan smooth karena dapat difaktorkan dengan basis faktor prima. Hitung vektor pangkat (mod 2) untuk setiap bilangan smooth yang dihasilkan. Misal G(x) = 6438 dan basis faktor prima {2,3,7,17,23,29,37,41} maka vektor pangkat G(x) adalah <1,1,0,0,0,1,1,0> dengan entry ke-i dalam vektor berkorespondensi dengan pangkat prima ke-i dalam basis.

Implementasi tahap sieving quadratic sieve pada makalah ini dapat dilihat dibawah.

```
for(int i=0;i<FactorBase.size();i++){
//Sieveing
BigInteger t =
Ressol(n,FactorBase[i]);//Shanks-tonelli
int soln1p = (t-b) % FactorBase[i];
if (soln1p < 0) soln1p= soln1p+FactorBase[i];

int soln2p = (-t-b) % FactorBase[i];
if (soln2p < 0) soln2p= soln2p+FactorBase[i];

//SIEVING WITH SOLN1P
for(intj=soln1p;j<m*2;j+=(int)FactorBase[i]){
        while(Accu[j] % FactorBase[i] == 0 )
        Accu[j] = Accu[j] / FactorBase[i];
        if (Accu[j]==1){
                BigInteger Gx = (b+j)*(b+j)-n;
                Gx = Gx % n;
                BigInteger ExponentVector =
                BigInteger(true,sz);
        If(QSTrial(Gx,&ExponentVector,FactorB
        ase)==true){
        //Vektor pangkat dihitung dengan
        trial division
        SmoothNumbers.push_back(ExponentVecto
        r);
        SmoothX.push_back(j);
        }
        }
}
```

```
if (i!=0){ //Jika bilangan prima != 2
maka sieving juga dilakukan untuk soln2p
//SIEVING WITH SOLN2p
for(intj=soln2p;j<m*2;j+=(int)FactorBase
[i]){
while(Accu[j] % FactorBase[i] ==0 )
Accu[j] = Accu[j] / FactorBase[i];
if (Accu[j]==1){
        BigInteger Gx = (b+j)*(b+j)-n;
        Gx = Gx % n;
        BigInteger ExponentVector =
        BigInteger(true,sz);
                                if
(QSTrial(Gx,&ExponentVector,FactorBase)=
=true){

        //Vektor pangkat dihitung dengan
trial division

        SmoothNumbers.push_back(Exponent
Vector);

        SmoothX.push_back(j);
                                }
                        }
```

3) Gaussian Elimination

Dalam proses ini, dicari sebuah solusi untuk A.v = 0 menggunakan Gaussian elimination dengan A adalah matriks yang setiap kolomnya ada vektor pangkat dari setiap bilangan smooth yang ditemukan pada proses sebelumnya. Misal untuk kasus seperti pada Tabel 1, Matriks A untuk kasus ini dapat dilihat pada Gambar

$$\begin{pmatrix} 1111001 \\ 1000101 \\ 0111001 \\ 0001010 \\ 0000101 \\ 1010000 \\ 1000011 \\ 0010000 \end{pmatrix} .v = 0$$

Gambar 2. Matriks vektor pangkat bilangan smooth

Solusi v adalah matriks kolom (0,0,0,1,1,1,1). Dengan menggunakan Gaussian elimination pada GF2 maka setiap baris pada matriks dapat direpresentasikan sebagai sebuah bilangan yang representasi bitnya sama dengan baris pada matriks.

B. *Elliptic Curve Method*

Pada model montgomery yang telah didefinisikan sebelumnya, operasi penambahan titik menggunakan sebuah titik P0 = P1-P2, P1 dan P2 adalah titik yang akan dijumlahkan. ECM menggunakan perkalian scalar titik dalam algoritmanya, perkalan scalar dalam elliptic curve melibatkan operasi penambahan titik untuk suatu k bukan merupakan bilangan pangkat 2 (untuk bilangan pangkat 2 seperti 2,4,8 maka cukup menggunakan penggandaan titik). Oleh karena itu digunakan algoritma Montgomery ladder untuk menghitung perkalian scalar suatu titik sehingga P0 = P1-P2=P1 untuk setiap k.

Implementasi Montgomery ladder pada makalah ini dapat dilihat dibawah.

```
int bits=k.Log(2); //Jumlah bit k
ECPoint Q = P1;
ECPoint P = DoublingPoint(P1);
if (k==1) return Q;
else if (k==2) return P;
else{
        for(int i=bits-2;i>=0;i--){
                if (k.Testbit(i)){
                        Q = AddPoint(P,Q,P1);
                        P = DoublingPoint(P);
                }
                else{
                        P = AddPoint(P,Q,P1);
                        Q = DoublingPoint(Q);
                }
        }
        return Q;
}
```

Untuk mempercepat komputasi tahap 2 dari ECM dapat dilakukan modifikasi sebagai berikut.

1. Misal Q adalah titik yang dihasilkan pada tahap 1 ECM dan i adalah variabel yang digunakan dalam iterasi.
2. Iterasi dilakukan dari B1+1 hingga B2 dengan interval 2. Pada setiap tahap tambahkan 2Q ke sebuah variabel sementara misal T.
3. JIka i adalah bilangan prima, hitung g = g * T_z (mod n) seperti pada algoritma yang telah didefinisikan sebelumnya hanya saja kali ini tidak menggunakan Q melainkan T.

Terkait dengan parameter kurva dan titik pada kurva yang digunakan pada ECM, pada makalah ini digunakan parameter sesuai dengan persamaan 15,16,17.

$$u = \sigma^2 - 5 \quad (15)$$

$$v = 4\sigma \quad (16)$$

$$P = (u^3, v^3) \quad (17)$$

$$A = (((u-v)^3 (3u+v)) / 4u^3 v) - 2 \quad (18)$$

σ pada persamaan diatas adalah suatu seed yang diambil secara random. P adalah titik pada kurva sedangkan A adalah parameter kurva. Penentuan parameter kurva ini diusulkan oleh Brent-Suyama[5].

Untuk pemilihan nilai B1 dan B2, digunakan beberapa nilai yang ada pada Tabel 2 dan juga nilai-nilai yang ditentukan khusus untuk suatu kunci tersebut.

Tabel 2. Parameter B1 dan B2

Digit	B1	B2
15	2000	20000
20	11000	110000
25	50000	500000
30	250000	25000000

IV. ANALISIS PERFORMANSI

Implementasi Quadratic Sieve dan Ellptic Curve Method dilakukan menggunakan bahasa c++ dengan compiler g++ (tdm-1) 4.7.1, IDE Code Blocks dan Library GNU MP. Pengujian kriptanalisis dilakukan pada laptop ASUS A46CB dengan spesifikasi :

1. Operating System : Windows 7
2. Processor : Intel® Core ™ i3-3217 CPU @ 1.80 GHz
3. Memory : 4096 MB RAM

Pada pengujian ini digunakan 8 buah kunci untuk kriptografi RSA dan Schmidt-Samoa sehingga terdapat total 16 kunci yang akan difaktorkan. Kunci-kunci tersebut memiliki ukuran yang berbeda, dua buah kunci berukuran 32 bit, dua kunci 56 bit, dua 64 bit dan dua 96 bit. Batasan 96 bit dipilih karena dengan algoritma quadratic sieve yang diimplementasikan pada makalah ini, nilai G(x) yang didefinisikan pada persamaan x akan menjadi sangat besar seiring dengan bertambahnya nilai x sehingga kemungkinan G(x) adalah bilangan smooth semakin kecil. Hal ini menyebabkan untuk bilangan yang semakin besar (semakin besar pula parameter quadratic sieve seperti F dan M) maka perbandingan jumlah bilangan smooth yang ditemukan dengan parameter F dan M semakin kecil sehingga pada tahap aljabar linear kecil kemungkinan ditemukannya solusi menggunakan Gaussian elimination. Kunci-kunci yang digunakan pada pengujian ini terdapat pada Tabel 3 dan Tabel 4.

Tabel 3. Daftar kunci RSA

Jenis Kunci	RSA (n,e)
32 bit	1736511797, 1126364179
32 bit	1589432539, 1126364179
56 bit	54841863455181931, 1126364179
56 bit	32058306134731889, 1126364179
64 bit	18446743979220271189, 1126364179
64 bit	12604875800316699377, 1126364179
96 bit	61624799841276931477064691589, 1126364179
96 bit	45283888972431539732935106129, 1126364179

Tabel 4. Daftar kunci Schmidt-Samoa

Jenis Kunci	Schmidt-Samoa
32 bit	761836847
32 bit	2442822749
56 bit	44272917859136359
56 bit	39736168065635149
64 bit	5665361462213350319
64 bit	9660560302271523311
96 bit	427908534435882567165795326 47
96 bit	468050807443147519076987283 67

Pengujian dilakukan dengan melakukan 5 kali eksekusi program dan dilihat rata-ratanya. Waktu eksekusi dihitung sebagai waktu yang dibutuhkan untuk memfaktorkan kunci publik kriptografi RSA dan Schmidt-Samoa. Pengujian yang dilakukan beberapa kali ini untuk karena waktu eksekusi komputer tidak tetap dan algoritma ECM juga menggunakan seed sehingga terdapat kemungkinan dengan sedikit keberuntungan maka ECM mendapatkan seed yang tepat. Sehingga algoritma tersebut dapat menyelesaikan faktorisasi dengan sangat cepat.

A. Pengujian Elliptic Curve Method (ECM)

Hasil pengujian kriptanalisis kriptografi RSA dan Schmidt-Samoa menggunakan ECM dapat dilihat pada Tabel 5 dan Tabel 6.

Tabel 5. Hasil pengujian kriptanalisis RSA dengan ECM

Jenis Kunci	RSA		
	Waktu Eksekusi (ms)	B1	Curves
32 bit	171.6	2000	2
32 bit	136.6	2000	1.4
56 bit	5069.4	11000	3.4
56 bit	2558.8	11000	1.8
64 bit	3953.4	11000	2.4
64 bit	5229	11000	3.6
96 bit	130990.4	200000	33
96 bit	122809.6	200000	28

Tabel 6. Hasil pengujian kriptanalisis Schmidt-Samoa dengan ECM

Jenis Kunci	Schmidt-Samoa		
	Waktu Eksekusi (ms)	B1	Curves
32 bit	1652.6	2000	3
32 bit	1327.6	2000	2
56 bit	5870.8	11000	1
56 bit	6147	11000	1.4
64 bit	9028.4	11000	1.2
64 bit	8514.2	11000	1.4
96 bit	30720	200000	2.6
96 bit	29158	200000	2.6

Dari Hasil pengujian pada Tabel 5 dan Tabel 6, dapat dilihat jika kriptanalisis RSA 32 bit,56 bit dan 64 bit memiliki waktu eksekusi yang lebih singkat dibanding Schmidt-Samoa. Namun yang menarik adalah pada kriptanalisis kuci 96 bit, Eksekusi ECM untuk kunci public Schmidt-Samoa jauh lebih cepat dibanding RSA. Perbandingan kecepatannya hampir mencapai 4 kali lipat. Hal ini bisa jadi dipengaruhi bahwa dalam menghasilkan kunci 96 bit untuk Schmidt-Samoa pada pengujian ini, digunakan sebuah bilangan 24 bit sesuai dengan persamaan x yaitu $N = p^2q$. Sedangkan untuk kunci 32 bit, 56 bit dan 64 bit, meskipun kriptanalisi kunci RSA lebih cepat namun tidak terlalu signifikan perbedaannya kecuali untuk kunci 32 bit.

Selain itu, jumlah kurva yang perlu diproses pada kriptanalisis Schmidt-Samoa secara umum lebih sedikit dibanding kriptanalisis RSA. Tentunya hal ini yang berpengaruh besar pada waktu eksekusi kedua proses kriptanalisis tersebut.

B. Pengujian Quadratic Sieve

Hasil pengujian kriptanalisis kriptografi RSA dan Schmidt-Samoa menggunakan quadratic sieve dapat dilihat pada Tabel 7 dan Tabel 8.

Tabel 7. Hasil pengujian kriptanalisis RSA dengan quadratic sieve

Jenis Kunci	RSA		
	Waktu Eksekusi (ms)	F	Bilangan Smooth
32 bit	11.4	333	43
32 bit	15.6	333	45
56 bit	461.2	5200	500
56 bit	491	5200	513
64 bit	685.6	11370	368
64 bit	1208.2	11370	871
96 bit	77687.6	20000	3933
96 bit	70520.6	20000	3869

Tabel 8. Hasil pengujian kriptanalisis Schmidt-Samoa dengan quadratic sieve

Jenis Kunci	Schmidt-Samoa		
	Waktu Eksekusi (ms)	F	Bilangan Smooth
32 bit	11.4	333	34
32 bit	10.2	333	22
56 bit	641	5200	742
56 bit	655	5200	670
64 bit	1240	11370	956
64 bit	985	11370	701
96 bit	62998	20000	3077
96 bit	70940	20000	4051

Dari Hasil pengujian pada Tabel x dan Tabel x, dapat dilihat jika kriptanalisis RSA dan Schmidt-Samoa secara umum menggunakan quadratic sieve memiliki waktu eksekusi yang hampir sama. Hal ini disebabkan karena algoritma quadratic sieve termasuk dalam kategori algoritma faktorisasi general-purpose sehingga struktur bilangan atau karakteristik bilangan yang difaktorkan tidak mempengaruhi performa algoritma kecuali ukuran bilangan tersebut. Selain itu, Selisih waktu kriptanalisis RSA dan Schmidt-Samoa menggunakan quadratic sieve bisa dibilang sangat kecil sehingga tidak bisa disimpulkan jika struktur kunci public RSA dan Schmidt-Samoa yang berbeda ini memiliki pengaruh terhadap algoritma quadratic sieve.

Khusus untuk kunci 96 bit tidak digunakan persamaan 13 untuk menghitung F karena jika digunakan maka jumlah bilangan pada basis faktor prima akan menjadi sangat besar dan tidak efisien. Hal ini juga berpengaruh pada interval sieving yang makin besar dan bisa jadi bilangan smooth yang ditemukan tidak jauh lebih banyak dibandingkan dengan F yang lebih kecil. Sebagai perbandingan, untuk bilangan 60 digit yang difaktorisasi oleh contini menggunakan quadratic sieve varian multiple polynomial dan self-initializing digunakan parameter F bernilai 60000 [1].

C. Analisis perbandingan metode faktorisasi quadratic sieve dan ECM

Secara umum, waktu eksekusi quadratic sieve untuk kriptanalisis RSA dan Schmidt-Samoa jauh lebih cepat dibanding ECM. Pengecualian untuk kriptanalisis Schmidt-Samoa 96 bit, untuk kriptanalisis Schmidt-Samoa 96 bit ECM justru memiliki kecepatan hingga 2 kali lebih cepat dibanding quadratic sieve. Hal ini dipengaruhi fakta bahwa ECM adalah algoritma faktorisasi special-purpose yang kompleksitasnya bergantung pada faktor yang ditemukan, Kompleksitas ECM dalam notasi Big-O adalah O(e(log p log log p) (1 + O(1))1/2) dengan p adalah faktor terkecil dari suatu bilangan yang akan difaktorkan. Hal ini dapat dibuktikan dengan melihat bahwa faktor pertama yang untuk dua kunci Schmidt-Samoa 96 bit adalah 230196593517361 dan 153855172322569 yang

Tabel 9. Hasil faktorisasi semua kunci dan kunci privatnya

Jenis Kunci	Kunci Publik	Kunci Privat
RSA 32 bit	1736511797 = 37273 * 46589	327994459
RSA 32 bit	1589432539 = 37273 * 42643	140367211
RSA 56 bit	54841863455181931 = 226836761 * 241767971	52466219047221823
RSA 56 bit	32058306134731889 = 158738683 * 201956483	27927076237643169
RSA 64 bit	18446743979220271189 = 4294967291 * 4294967279	18310774304199697479
RSA 64 bit	12604875800316699377 = 3469926827 * 3632605651	4486739440845936319
RSA 96 bit	616247998412769314770646915 89 = 247990749309389 * 248496365339801	565324345573651198459051570 19
RSA 96 bit	452838889724315397329351061 29 = 204137719932131 * 221830090918459	254409804036249948148393663 99
Schmidt-Samoa 32 bit	$761836847 = 113^2 * 59663$	3201951
Schmidt-Samoa 32 bit	$2442822749 = 211^2 * 54869$	2823329
Schmidt-Samoa 56 bit	$44272917859136359 = 14389^2 * 213834079$	227762552167
Schmidt-Samoa 56 bit	$39736168065635149 = 14867^2 * 179779141$	1257619291669
Schmidt-Samoa 64 bit	$5665361462213350319 = 36319^2 * 4294967279$	38432513030531
Schmidt-Samoa 64 bit	$9660560302271523311 = 51659^2 * 3620014631$	44179790921801
Schmidt-Samoa 96 bit	$427908534435882567165795326 47 = 12403837^2 * 278124243713263$	14003866372816868 59
Schmidt-Samoa 96 bit	$468050807443147519076987283 67 = 15172231^2 * 203326556788447$	347789447976251017993

keduanya adalah bilangan kuadrat dengan akar kuadrat 15172231 dan 12403837. Faktorisasi semua kunci dan hasil perhitungan kunci privat dapat dilihat pada tabel 9.

Quadratic sieve adalah algoritma faktorisasi general-purpose yang kompleksitasnya hanya dipengaruhi oleh ukuran bilangan tersebut sehingga performa quadratic sieve untuk kriptanalisis RSA dan Schmidt-Samoa hampir sama.

V. KESIMPULAN DAN SARAN

Pada makalah ini telah diimplementasikan dan dibandingakan performa algoritma quadratic sieve dan metode faktorisasi elliptic curve dalam faktorisasi kunci public untuk kriptanalisis RSA dan Schmidt-Samoa. Hasil percobaan menunjukkan bahwa secara umum quadratic sieve memiliki waktu eksekusi yang lebih cepat dibandingkan dengan ECM kecuali untuk kunci Schmidt-Samoa berukuran 96 bit. Untuk kunci Schmidt-Samoa berukuran 96 bit, ECM memiliki waktu eksekusi 2 kali lipat lebih cepat dibandingkan Quadratic Sieve dan 4 kali lebih cepat dibandingkan ECM untuk RSA.

Penggunaan seed yang dibatasi hanya integer 32 bit bisa jadi berpengaruh dalam eksekusi ECM karena dengan menggunakan seed yang jauh lebih besar maka kemungkinan ditemukannya kurva yang tepat lebih besar.

Referensi

[1] S. Contini, 'Factoring Integers with the Self-Initializing Quadratic Sieve', Master of Arts, University of Georgia, 1997.

[2] A. Lenstra and W. Bosma, 'an implementation of elliptic curve integer factorization method', in *Computational Algebra and Number Theory*, 1995, pp. 119-136.

[3] C. Seibert, 'Integer Factorization using the Quadratic Sieve', in *Midwest Instruction and Computing Symposium*, 2015.

[4] W. Rundell, 'Quadratic Sieve Example', calclab.math.tamu.edu/~rundell/m470/notes/sieve_example.pdf

[5] http://www.mersennewiki.org/index.php/Elliptic_Curve_Method

[6] http://programmingpraxis.com/2010/04/23/modern-elliptic-curve-factorization-part-1/

[7] E. Landquist, 'The Quadratic Sieve Factoring Algorithm', http://www.cs.virginia.edu/crab/QFS_Simple.pdf

[8] J. Jonsson and B. S. Kaliski, 'Public key cryptography standards (PKCS) #1 : RSA cryptography specifications version 2.1' RFC 344, Internet Engineering Task Force, 2003.

[9] K. Schmidt-Samoa, "A New Rabin-type Trapdoor Permutation Equivalent to Factoring and Its Applications," Electronic Notes in Theoretical Computer Science, vol. 157, no. 3, pp. 79-94, 2006