Analisis dan Implementasi Serangan Kunci secara Konkuren pada Algoritma RSA

Rezan Achmad / 13508104[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*if18104@students.if.itb.ac.id*, *rezanachmad@gmail.com*

***Abstrak* — RSA adalah salah satu kriptografi kunci publik yang saat ini masih digunakan. Kekuatan pada algoritma ini yaitu karena sulitnya untuk mefaktorkan suatu bilangan yang berukuran sangat besar. Pada makalah ini penulis mencoba untuk memecahkan bilangan RSA secara konkuren. Terdapat dua pendekatan yang digunakan yaitu percobaan dengan menggunakan *sieve of erastothenes* serta *Square of congruens*. *Square of congruens* lebih cepat dalam memecahkan kunci RSA. Pemecahan secara konkuren dalam satu komputer ternyata tidak berpengaruh untuk mempercepat komputasi, akan lebih baik jika dilakukan diberbeda komputer.**

***Kata kunci* — RSA; *Square of congruens*; *sieve of erastothenes*; Konkuren;**

I. PENGANTAR

RSA adalah salah satu algoritma enkripsi kunci publik. Algoritma ini ditemukan pada tahun 1977 oleh tiga ilmuan MIT (*Massachusetts Institute of Technology*) yaitu Ron Rivest , Adi Shamir dan Len Adleman. RSA tidak lain berasal dari nama mereka (Rivest – Shamir – Adleman).

Konsep kriptografi kunci publik atau asimetrik membutuhkan dua kunci yang berbeda untuk melakukan enkripsi maupun dekripsi. Kunci yang digunakan untuk untuk enkripsi biasa disebut kunci publik sedangakn yang digunakan untuk dekripsi disebut kunci privat. Sesuai sebutannya, kunci public boleh diketahui oleh semua orang sehingga siapa pun bisa melakukan enkripsi suatu pesan. Pesan yang enkripsi hanya bisa didekripsi oleh seseorang yang memiliki kunci privat.

Alur untuk melakukan enkripsi serta dekripsi pesan pada kriptografi kunci publik adalah sebagai berikut :

1. Misalkan A ingin mengirim pesan rahasia ke B
2. Pertama-tama A akan menghubungi B bahwa akan mengirim sebuah pesan rahasia
3. B akan membangkitkan dua buah kunci yaitu kunci public dan kunci privat
4. B mengirimkan kunci publiknya ke A
5. A melakukan enkripsi terhadap pesan rahasia tersebut kemudian mengirim pesan terenkripsi tersebut ke B
6. B menerima pesan dan membukanya / dekripsi dengan kunci privat.
7. Hal yang sama juga akan dilakukan jika B ingin mengirim pesan rahasia ke A.

A. Properti RSA

RSA memerlukan dua bilangan prima untuk menghasilkan kunci publik dan kunci privat. Secara lengkap berikut properti yang harus ada pada algoritma RSA

No.	Properti	Rahasia
1.	p dan q yang keduanya merupakan bilangan prima	Ya
2.	$n = p \cdot q$	Tidak
3.	$\phi(n) = (p-1)(q-1)$	Ya
4.	e (Kunci enkripsi) dengan syarat $GCD(e, \phi(n)) = 1$	Tidak
5.	d (Kunci dekripsi) $d \equiv e^{-1} \bmod (\phi(n))$	Ya
6.	m (plainteks)	Ya
7.	c (cipherteks)	Tidak

Penjelasan :

$\phi(n)$ merupakan Toitent Euler yang menyatakan berapa banyak bilangan lebih kecil dari n yang relatif prima terdahap n. Jika n = pq adalah komposit dengan p dan q prima maka nilai $\phi(n) = \phi(p) \cdot \phi(q) = (p-1)(q-1)$.

GCD merupakan singkatan dari *Great Common Divisor* yaitu Pembagi Bersama Terbesar (PBB). GCD(a, b) akan menghasilkan bilangan terbesar x dimana x habis membagi a serta b.

B. Pembangkitan Kunci, Enkripsi dan Dekripsi.

Proses pembangkitan kunci pada algoritma RSA adalah sebagai berikut.

1. Pilih bilangan prima p dan q, misal $p = 79$ dan $q = 97$.
2. Hitung $n = pq \rightarrow n = 79 * 97 = 7663$.
3. Pilih sebuah bilangan bulat e untuk kunci public dan e harus relatif prima terdahap $\phi(n)$. Misalkan e yang pilih yaitu 101.
4. Hitung kunci dekripsi. $ed \equiv 1 \pmod{\phi(n)}$ atau $d \equiv e^{-1} \bmod (\phi(n))$. Nilai i yang diperoleh yaitu 2669.
5. Kunci publik (e, I) = (101, 7663)
6. Kunci privat (d, n) = (2669, 7663)

Enkripsi

Sebelum melakukan enkripsi nyatakan pesan menjadi blok-blok plainteks $m_1, m_2, m_3, \ldots m_n$ dengan syarat $0 < m_i < n - 1$ atau dengan kata lain ukuran blok lebih besar dari 0 dan harus lebih kecil dari nilai *n* dikurangi satu.

Setiap blok dienkripsi dengan rumus berikut :

$$c_i = m_i^e \bmod n$$

Hasil dari rumus ini adalah blok ciphertext.

Dekripsi

Untuk melakukan dekripsi sama halnya dengan enkripsi. Rumus yang digunakan yaitu

$$m_i = c_i^d \bmod n$$

Keamanan algoritma kriptografi RSA terletak pada bilangan *n* (*p* . *q*). Walaupun bilangan ini bersifat publik akan sulit (membutuhkan waktu yang lama) juga untuk mendapatkan *p* dan *q* yang sesuai jika *p* dan *q* bernilai sangat besar.

Serangan yang dilakukan untuk memecahkan RSA ini tidak lain adalah memfaktorkan bilangan *n* tersebut. Pada implementasinya, bilangan *n* tergolong besar oleh karena itu dibutuhkan komputasi yang semangkus mungkin dan dilakukan secara konkuren di tiap proses bahkan di banyak komputer.

II. FAKTORISASI INTEGER

Sekrang ini, ada banyak cara untuk melakukan faktorisasi integer. *Brute force* dari angka prima terkecil, dengan bantuan pembangkitan bilang prima *Sieve of Erastosthenes*, faktorisasi *Fermat*, *Congruence of Squares*, *Quadratic Sieve* serta *General Number Field Sieve*.

Hal yang paling sederhana yaitu dengan *brute force*, berikut algoritmanya.

```
function isPrime(n : integer)
  div : integer
  bound : integer
  div := 2
  bound := celi(sqrt(n))

  if (n = 1)
    ➔ false

  while (div < bound)
    if (n mod div = 0)
      ➔ false
    div := div + 1

  ➔ true

function factorOf(n : integer)
  d : integer
  d := 2

  while (n <> 1)
    if(isPrime(n))
       print (n)
       n := n div n

    if(isPrime(d))
      while (n mod d = 0)
        print (d)
        n := n div d

    d := d + 1
```

Algoritma *brute force* terkadang selalu tidak mangkus. Algoritma diatas selalu memeriksa apakah bilangan d termasuk prima atau bukan sebelum dibagi dengan bilangan *n*. Akan lebih efektif jika beberapa bilangan prima disimpan didalam suatu array.

Terdapat beberapa cara untuk membangkitkan beberapa bilangan prima. Melakukan brute force dari angka 1 hingga n kemudian memeriksa setiap bilangan apakah prima atau bukan bisa dilakukan tetapi cara ini terbukti kurang ampuh. Terdapat cara lebih sederhana tetapi lebih cerdas yaitu pembangkitan dengan algoritma *sieve of eratosthenes*.

Misalkan akan dicari semua bilangan prima diantara 1 dan *n*.

1. Buat sebuah *array* dari *boolean* yang berukuran *n*.
2. Inisiasi semua array dengan *true* yang berarti semua bilangan masih dianggap prima.
3. Indeks ke 1 diberi nilai *false* karena angka 1 bukan angka prima.
4. Diiterasi dimulai dari indeks pertama.
5. Jika indeks yang sedang diperiksa bernilai *true*, buatlah kelipatan dari indeks tersebut bernilai *false*.
6. Jika indeks yang sedang diperiksa bernilai false, lewati dan periksa indeks berikutnya.
7. Langkah 5 dan 6 terus diulangi hingga indeks mencapai akar dari *n*.

Contoh ilustrasinya adalah sebagai berikut :

Algoritma diatas masih bisa dioptimasi lagi, berikut pseudo codenya :

```
function Eratosthenes(n : integer)
  a : array of integer
  p, i : integer

  a[1] := false
  p := 2

  for i := 2 to n
    a[i] := true

  while p^2 ≤ n
    i := p^2

    while (i ≤ n)
      a[i] := false
      i := i + p

    repeat
      p := p + 1
    until a[p] = true

  return a
```

Sekarang fungsi `isPrime` diubah dengan hanya mereturn sebuah nilai dari sebuah indeks array.

```
a : array of integer
input : integer

readln (input)
a := Eratosthenes(input)

function isPrime(n : integer)
  return a[n]
```

Fungsi `factorOf` semestinya tidak seribet itu untuk mencari factor dari bilangan RSA. Bilangan RSA hanya memiliki dua factor prima yaitu *p* dan *q*. Fungsi `factorOf` bisa disederhanakan lagi.

```
function generateArrayPrim(a : array of integer)
  lInt : list of integer
  i : integer

  for i = 2 to sizeof(a) do
    if a[i] = true
      lInt.push(i)


function factorOf(n : integer)
  i : integer
  lInt : list of integer
  finish : bool

  i := 1
  lInt := generateArrayPrim(a) /* a berasal
dihasilkan di program utama, lihat pseuodo
sebelumnya */
  finish := false

  while finish = false do
    if n mod lInt[i] = 0 do
      finish = true
    else
      i := i + 1

  print (“p : “ + lInt[i] + “, q : “ + n div
lInt[i])
```

Terdapat fungsi baru pada pseudo code diatas yaitu generateArrayPrime. Fungsi ini akan menampung semua bilangan prima hasil dari Sieve of Eratosthenes

Ada cara yang lebih efektif lagi untuk mencari faktor prima dari bilangan RSA yaitu *Congruence of Squares*. Algoritma ini menggunakan dasar dari teorema fermat yang menyatakan bahwa :

Terdapat bilangan *x* dan *y* yang memenuhi :
$x^2 - y^2 = n$; *n* adalah bilangan ganjil

Berikut penurusan rumus *Congruence of Squares* dari faktorisasi fermat.

$x^2 - y^2 = n$
$x^2 \equiv y^2 \pmod n$, $x \neq \pm y \pmod n$

Dari sini dapat disimpulkan

$x^2 - y^2 \equiv 0 \pmod n$, $(x + y)(x - y) \equiv 0 \pmod n$

Ini berarti *n* membagi $(x + y)(x - y)$. *n* tidak dapat dibagi langsung oleh $(x + y)$ maupun $(x - y)$ tetapi faktor-faktor dari bilangan itu yang bisa membagi. Faktor-faktor tersebut dapat diperoleh dari $GCD(x + y, n)$ dan $GCD(x - y, n)$.

Langkah – langkah untuk menemukan faktor dari sebuah bilangan RSA n yaitu

1. Nilai awal *x* adalah akar dari bilangan n dengan pembulatan keatas
2. Cari nilai y^2 yaitu akar dari $(x^2 - n)$
3. Jika y^2 bukan kuadrat dari bilangan integer, naikkan nilai *x* kemudian ulangi langkah 2.
4. Jika y^2 kuadrat dari bilangan integer, pencarian selesai.
5. Lakukan pergitungan $p = GCD(x + y, n)$ atau $GCD(x - y, n)$ dan $q = n / p$.

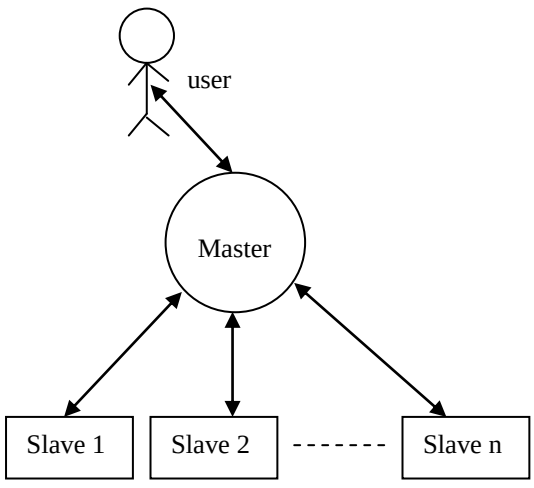
Kendala dari algoritma ini yaitu pencarian nilai *y* dan *x*. Semakin besar nilai *y* semakin besar juga nilai *x*. Ada kemungkinan terdapat bilangan *n* yang menyebabkan pencarian nilai *y* dan *x* membutuhkan waktu yang cukup lama.

III. IMPLEMENTASI

Penulis akan mengimplementasikan dua buah algoritma untuk memecahkan kunci RSA. Pertama yaitu dengan bantuan algoritma *Sieve of Eratosthenes* dan yang kedua yaitu *Congruence of Squares*.

Hasil implementasi kedua algoritma ini akan dijalankan secara *multithreading* atau konkuren. Hal ini dilakukan agar mercepat proses pencarian faktor.

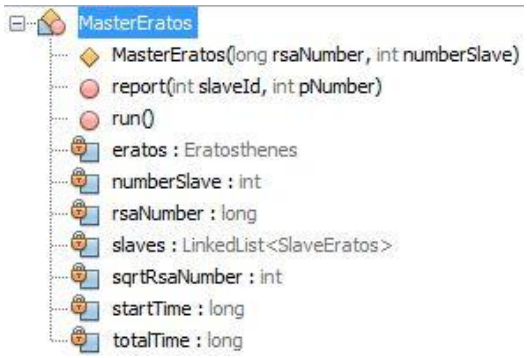
Karen besifat konkuren tentu harus ada yang melakukan koordinasi. Disini akan digunakan skema *master-slave* yaitu terdapat satu *thread* yang bertindak sebagai “bos” dan *thread* lainnya bertindak sebagai pekerja. Di awal program, “bos” akan minta masukan dari pengguna kemudian akan membuat beberapa *thread slave* baru untuk melakukan pembagian kerja. Setiap *thread slave* akan mengerjakan bagiannya masing-masing. Ketika salah satu *slave* menemukan faktor dari RSA ia harus memberi informasi ke *master* agar *master* memberhentikan semua komputasi yang dilakukan oleh *slave* lain.



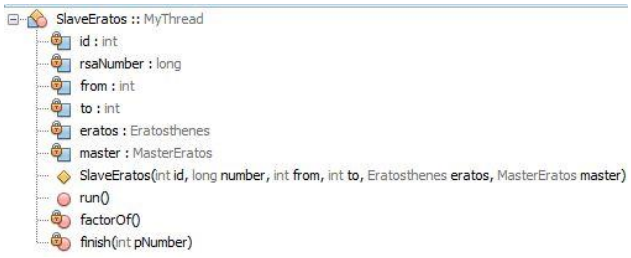
Terdapat tiga jenis program yang diimplementasikan yaitu *Eratos*, *Squares* serta *PrimeEratos*.

A. Eratos

Program ini berfungsi untuk memecahkan kunsi RSA dengan mengiterasi bilang prima. Struktur kelas pada program *Eratos* yaitu :



Gambar 1 Struktur Kelas Master Eratos

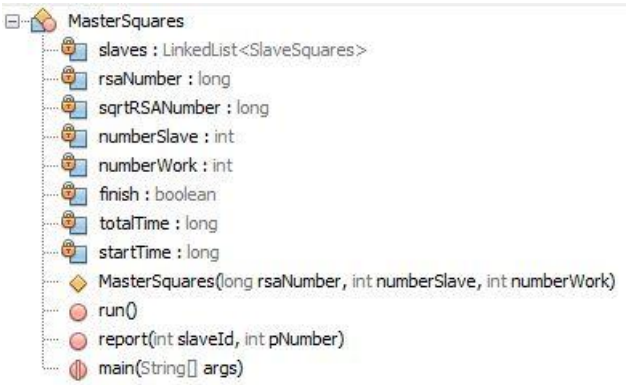


Gambar 2 Struktur Kelas Slave Eratos

Pada program Eratos ini proses pembagian bilangan RSA dengan bilangan prima dilakukan dari bilangan prima dengan nilai terbesar. Menurut pengalaman, untuk membentuk bilangan *n* RSA akan dipilih bilangan *p* dan *q* yang bernilai cukup besar. Kalaupun ada yang menggunakan pasangan angka kecil dan besar itupun sangat jarang terjadi.

B. Square

Program ini berfungsi untuk memecahkan bilangan RSA dengan menggunakan metode *Congruence of Squares* serta faktorisasi *Fermat*. Struktur kelas pada program *Square* yaitu :



Gambar 3 Struktur Kelas Master Squares

Gambar 4 Struktur Kelas Slave Squares

Pada program ini ada optimasi yang dilakukan yaitu penentuan apakah suatu bilangan termasuk bilangan kuadrat atau bukan. Secara umum algoritma untuk menentukan bilangan adalah suatu kuadrat yaitu :

```
function isQuadrat(n : integer)
  sqrt : real
  sqrt := Math.sqrt(n)

  if sqrt - floor(sqrt) = 0 do
     → true
  else
     → false
```

Optimasi yang dilakukan pada algoritma diatas yaitu mencari pola bilangan kuadrat. Bilangan kuadrat diakhiri dengan angka-angka sebagai berikut :

Quadrat	Mod 10
0 * 0	0
1 * 1	1
2 * 2	4
3 * 3	9
4 * 4	6
5 * 5	5
6 * 6	6
7 * 7	9
8 * 8	4
9 * 9	1

Bilangan-bilangan yang berakhiran dengan angka 0, 1, 4, 5, 6, 9 kemungkinan adalah bilangan kuadrat sedangkan

bilangan yang berakhiran 2, 3, 7, 8 sudah pasti bukan bilangan kuadrat. Hasil modifikasi algoritmanya yaitu :

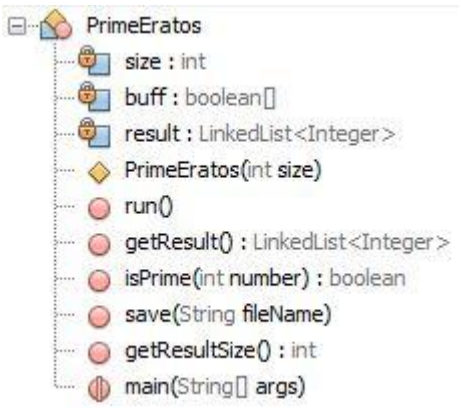
```
function isQuadrat(n : integer)
  sqrt : real
  sqrt := Math.sqrt(n)

  if (n mod 10 = 2 or n mod 10 = 3 or n mod 10 =
     7 or n mod 10 = 8)
     ➔ false

  if sqrt – floor(sqrt) = 0 do
     ➔ true
  else
     ➔ false
```

C. PrimeEratos

Program ini berfungsi untuk membangkitkan bilangan prima. Progam akan menerima masukan berupa bilangan. Bilangan ini berfungsi sebagai batas pembangkitan bilangan prima. Struktur kelas pada program *PrimeEratos* yaitu :



Gambar 5 Struktur Kelas PrimeEratos

V. PENGUJIAN

Pada bab ini akan dilakukan dua pengujian untuk kedua program yaitu *Eratos* dan *Squares*. Ada dua skenario uji yang akan dilakukan.

1. Pengujian individual. Pengujian ini hanya melibat satu *slave*.
2. Pengujian multi-*slave*. Pengujian ini melibat banyak *slave* yang dimasukkan oleh pengguna / penguji.

A. Pengujian I

Pada pengujian ini bilangan RSA yang digunakan yaitu 387983160407 (12 digit).

Untuk program Eratos hasil pengujiannya adalah sebagai berikut :

5 Slaves

```
Input
RSA Number : 387983160407
Slave Number : 5

Proses
Create slave number 0, work from 0, to 10165
Create slave number 1, work from 10165, to 20330
Create slave number 2, work from 20330, to 30495
Create slave number 3, work from 30495, to 40660
Create slave number 4, work from 40660, to 50825

Result
Crack finished. Found by slave number 4 in 1051
milis.
Total time to generate primes number : 16 milis.
RSA Number : 387983160407, p : 493027, q : 786941
```

Maksud dari *create slave number 0, work from 0 to 10165* yaitu *Master* membuat *slave* dengan *id* 0. Slave ini akan memakai bilangan prima ke 0 hingga ke 10165.

1 Slaves

```
Input
RSA Number : 387983160407
Slave Number : 1

Proses
Create slave number 0, work from 0, to 50825

Result
Crack finished. Found by slave number 0 in 331
milis.
Total time to generate primes number : 24 milis.
RSA Number : 387983160407, p : 493027, q : 786941
```

Untuk program *Squares* hasil pengujiannya adalah sebagai berikut :

5 Slaves

```
Input
RSA Number : 387983160407
Slave Number : 5
Number work : 100000

Proses
Create slave number 0, work from 622883, to
722883
Create slave number 1, work from 722883, to
822883
Create slave number 2, work from 822883, to
922883
Create slave number 3, work from 922883, to
1022883
Create slave number 4, work from 1022883, to
1122883

Output
Crack finished. Found by slave number 0 in 27
milis.
X : 639984, Y : 146957
RSA Number : 387983160407, p : 493027, q : 786941
```

Hasil program ini sedikit berbeda dari hasil program *Eratos*.

1. *Number Work* adalah maksimal jumlah bilangan yang harus komputasi oleh masing-masing *slave*.
2. *x* dan *y* adalah angka yang ditemukan sehingga $x^2 - y^2 = RSA\ Number$.

1 Slave

```
Input
RSA Number : 387983160407
Slave Number : 1
Number work : 100000

Proses
```

```
Create slave number 0, work from 622883, to
722883

Output
Crack finished. Found by slave number 0 in 24
milis.
X : 639984, Y : 146957
RSA Number : 387983160407, p : 493027, q :
786941
```

B. Pengujian II

Pada pengujian ini bilangan RSA yang digunakan yaitu 4996450155913 (13 digit),

Untuk program Eratos hasil pengujiannya adalah sebagai berikut :

7 Slaves

```
Input
RSA Number : 4996450155913
Slave Number : 7

Proses
Create slave number 0, work from 0, to 23582
Create slave number 1, work from 23582, to 47164
Create slave number 2, work from 47164, to 70746
Create slave number 3, work from 70746, to 94328
Create slave number 4, work from 94328, to
117910
Create slave number 5, work from 117910, to
141492
Create slave number 6, work from 141492, to
165079

Output
Crack finished. Found by slave number 4 in
103686 milis.
Total time to generate primes number : 132
milis.
RSA Number : 4996450155913, p : 1309999, q :
3814087
```

1 Slaves

```
Input
RSA Number : 4996450155913
Slave Number : 1

Proses
Create slave number 0, work from 0, to 165079

Output
Crack finished. Found by slave number 0 in 47867
milis.
Total time to generate primes number : 126 milis.
RSA Number : 4996450155913, p : 1309999, q :
3814087
```

Untuk program *Squares* hasil pengujiannya adalah sebagai berikut :

7 Slaves

```
Input
RSA Number : 4996450155913
Slave Number : 7
Number work : 1000000

Proses
Create slave number 0, work from 2235275, to
3235275
Create slave number 1, work from 3235275, to
4235275
Create slave number 2, work from 4235275, to
5235275
Create slave number 3, work from 5235275, to
6235275
Create slave number 4, work from 6235275, to
7235275
Create slave number 5, work from 7235275, to
8235275
Create slave number 6, work from 8235275, to
9235275

Output
Crack finished. Found by slave number 0 in 265
milis.
X : 2562043, Y : 1252044
RSA Number : 4996450155913, p : 1309999, q :
3814087
```

1 Slaves

```
Input
RSA Number : 4996450155913
Slave Number : 1
Number work : 1000000

Proses
Create slave number 0, work from 2235275, to
3235275

Output
Crack finished. Found by slave number 0 in 82
milis.
X : 2562043, Y : 1252044
RSA Number : 4996450155913, p : 1309999, q :
3814087
```

VI. ANALISIS

Dari dua pengujian sebelumnya terlihat jelas bahwa program *Squares* lebih cepat untuk menemukan faktor bilangan RSA dibanding dengan program *Eratos*. Pada pengujian kedua *Eratos* bahkan menorehkan waktu 47867 milis (1 *Slave*) untuk melakukan *crack*. Hasil ini sangat berbeda jauh yang dilakukan oleh *Squares*. Squares hanya memerlukan waktu 82 milis (1 *Slave*) untuk memecahkan bilangan RSA.

Faktorisasi fermat memang mangkus untuk melakukan pemfaktoran bilangan integer, apalagi hanya mencari satu faktor saja. Teorema fermat merupakan teorema yang fenomenal karena cukup ekstrim dalam mengkompresi waktu untuk mencari faktor suatu bilangan.

Program *Eratos* terlihat sangat lambat karena memang basisnya adalah *brute force* dan tidak memakai manipulasi atau teorema matematik. *Eratos* kehabisan waktu mengenumurasi bilangan sekaligus melakukan pembagian. Akan lebih lebih lambat lagi jika bilangannya cukup besar.

Jika dillihat dari pengujian, program 1 *slave* justru rata-rata lebih unggul dibandingkan *multi-slave*. Beberapa penyebabnya antara lain :

1. Membutuhkan waktu untuk membuat sebuah thread. Dalam sistem operasi perlu dibuat tabel untuk thread sebagai pengenal dan penyimpan state.
2. *MultiThread* tidak identik dengan *multi paralel* karena dalam sistem operasi yang terjadi sebenarnya adalah *pseudo paralelism*.

3. Kemungkinan distribusi p dan q dibagian atas (bilangan besar) sehingga cukup 1 slave saja yang melakukan penelusuran.

VII. KESIMPULAN

Beberapa kesimpulan yang dapat ditarik dari pembahasan sebelumnya yaitu :

1. Multithread tidak selalu lebih cepat daripada singlethread. Konkurensi akan lebih baik jika dilakukan diberbeda komputer karena menggukan sumber daya yang berbeda.
2. *Square of Congruens* terbukti lebih cepat memecahkan bilangan RSA dibandingkan dengan cara *Eratosthenes*.

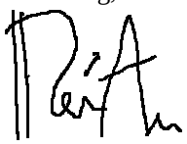
REFERENCES

[1] http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/Algoritma%20RSA.ppt
[2] http://www.rsa.com/
[3] http://en.wikipedia.org/wiki/Congruence_of_squares
[4] http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Mei 2011



Rezan Achmad / 13508104