Optimasi Algoritma Kunci Publik RSA, Rabin, dan ElGamal

Harry Octavianus Purba
Teknik Informatika, STEI
Institut Teknologi Bandung
Bandung, Indonesia
13514050@std.stei.itb.ac.id

***Abstract*-- Algoritma kriptografi kunci publik melibatkan komputasi yang sangat rumit dengan bilangan bulat yang sangat besar di dalamnya. Hal ini dilakukan agar semakin menyulitkan kriptanalis untuk mendapatkan informasi rahasia yang dikirimkan. Hal tersebut berlaku untuk tiga algoritma kriptografi kunci publik yang terkenal saat ini yaitu RSA, Rabin, dan ElGamal. Akibat komputasi yang sangat besar yang dilakukan oleh algoritma ini, waktu yang diperlukan untuk melakukan proses enkripsi dan dekripsi semakin lama. Solusi untuk persoalan ini adalah dengan menggunakn metode binary pada operasinya, dan mengimplementasikannya secara paralel. Dari hasil pengujian didapatkan bahwa semakin besar ukuran data atau semakin banyak thread yang digunakan pada implementasi paralel , maka speed up jika dibandingkan terhadap implementasi serial semakin tinggi.**

Keywords—kriptografi, binary, paralel

I. PENDAHULUAN

Kriptografi adalah suatu ilmu atau teknik yang dilakukan untuk menjamin kerahasiaan suatu pesan. Salah satu kategori algoritma kriptografi adalah algoritma kunci publik. Pada algoritma kunci publik atau asimetris, komputasi yang dilakukan sangatlah kompleks dan melibatkan bilangan-bilangan yang sangat besar. Hal ini menimbulkan suatu kelebihan sekaligus kerugian. Kelebihannya adalah semakin sulitnya informasi rahasia tersebut akan didapatkan oleh pihak yang tidak berkepentingan, dan kelemahannya adalah durasi melakukan enkripsi yang relatif lama jika dibandingkan dengan algoritma simetris.

Contoh algoritma kunci publik yang terkenal saat ini adalah algoritma RSA, ElGamal, dan Rabin. Algoritma RSA dan algoritma Rabin terkenal dengan kesulitan untuk memfaktorkan suatu bilangan yang besar menjadi dua faktor prima, sedangkan algoritma ElGamal terkenal dengan kesulitannya untuk menemukan nilai logaritma diskrit dalam suatu ruang modulo.

Untuk melakukan proses enkripsi maupun dekripsi, algoritma kunci publik melakukan operasi yang sangat kompleks dengan bilangan bulat yang sangat besar. Hal ini berimbas pada waktu eksekusi menjadi kurang efisien seiring dengan ukuran data yang enkripsi/dekripsi. Oleh karena itu pada makalah ini akan diimplementasikan solusi untuk dapat mengoptimasi algoritma-algoritma kunci publik RSA, Rabin, dan ElGamal.

II. LANDASAN TEORI

A. Kriptografi

Kriptografi adalah suatu ilmu dan seni untuk menjaga keamanan pesan (Schneier, 1996). Dalam kriptografi terdapat dua konsep utama yaitu enkripsi dan dekripsi. Enkripsi merupakan proses menyandikan plainteks (pesan yang sebenarnya) menjadi cipherteks (pesan yang telah disandikan), sedangkan dekripsi adalah kebalikannya yaitu proses mengembalikan cipherteks menjadi plainteks.

Algoritma kriptografi berdasarkan kunci yang digunakan, dibedakan menjadi dua yaitu algoritma simetris dan algoritma asimetris. Algoritma simetris adalah algoritma kriptografi dimana untuk proses enkripsi dan dekripsi menggunakan kunci yang sama yaitu kunci privat. Dengan menggunakan algoritma ini, pengirim pesan dan penerima pesan terlebih dahulu harus menyepakati kunci privat yang akan digunakan sebelum nantinya mengirim pesan. Contoh algoritma simetris yang terkenal adalah DES, Rinjdael, Blowfish, IDEA, dan GOST. Algoritma asimeteris atau algoritma kunci publik adalah algoritma kriptografi dimana untuk proses enkripsi dan dekripsi menggunakan kunci yang berbeda yaitu kunci publik untuk enkripsi dan kunci privat untuk dekripsi. Contoh algoritma asimetris yang terkenal adalah algoritma RSA, Rabin, ElGamal, DSA, dan ECC.

B. Algoritma Kunci Publik

Pada jenis kriptografi kunci publik terdapat dua buah kunci yang digunakan yaitu kunci publik untuk mengenkripsi pesan dan kunci privat untuk mendekripsi pesan. Kunci publik dan kunci privat ditentukan oleh calon penerima pesan. Kunci publik akan dikirimkan kepada calon pengirim

pesan. Pengiriman kunci publik ini tidak perlu melalui saluran yang aman. Sedangkan kunci privat hanya boleh diketahui oleh si penerima pesan saja.

Pembangkitan kunci pada sebagian besar algoritma kunci publik didasarkan persoalan sebagai berikut:

1. Pemfaktoran bilangan prima
   Informasi yang didapatkan oleh kriptanalis dari saluran pengiriman pesan adalah suatu bilangan bulat n. Untuk dapat melakukan dekripsi pada cipherteks yang dikirimkan, kriptanalis harus dapat memfaktorkan n menjadi dua bilangan prima.
   Contoh :
   n = 6, maka p1 = 2 dan p2 = 3
   n = 55, maka p1 = 5 dan p2 = 11
   n = 1881647025341, maka p1 = 94771 dan p2 = 19854671

   Semakin besar nilai dari integer n, maka akan semakin sulit difaktorkan. Algoritma yang menggunakan persoalan ini adalah RSA dan Rabin.

2. Logaritma Diskrit
   Informasi yang didapatkan oleh kriptanalis dari saluran pengiriman pesan adalah suatu bilangan bulat y, g dan p. Untuk dapat melakukan dekripsi pada cipherteks yang dikirimkan, kriptanalis harus dapat mencari suatu nilai x yang memenuhi persamaan $g^x \equiv y \,(mod\ p)$. Hal yang membedakan dengan logaritma diskrit biasa adalah pada logaritma diskrit biasa $g^x = y$, solusi dapat ditemukan dengan mudah karena nilai y bertambah seiring dengan nilai x (grafik naik), sedangkan pada persoalan $g^x \equiv y \,(mod\ p)$, bertambahnya nilai x dapat menyebabkan nilai y menjadi bertambah ataupun menurun. Hal ini terjadi karena dibatasi oleh ruang modulo p. Semakin besar nilai dari integer p, maka nilai x akan semakin sulit untuk dicari. Algoritma yang menggunakan persoalan ini adalah ElGamal.

C. Algoritma RSA

Algoritma RSA adalah algoritma kriptografi kunci publik yang paling terkenal dan paling banyak aplikasinya. Sesuai dengan asal mula terbentuk namanya, algoritma ini ditemukan oleh Ron Rivest, Adi Shamir, dan Len Adleman pada 1976. Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi 2 faktor prima. Algoritma RSA memiliki tiga proses utama yaitu proses pembentukan kunci, proses enkripsi, dan proses dekripsi.

Algoritma RSA memiliki komponen antara lain :

1. p dan q bilangan prima — *rahasia*
2. n = p . q — *tidak rahasia*
3. $\phi(n) = (p-1)(q-1)$ — *rahasia*
4. kunci enkripsi e — *tidak rahasia*
5. kunci dekripsi d — *rahasia*
6. plainteks m — *rahasia*
7. cipherteks k — *tidak rahasia*

Proses pembentukan kunci pada algoritma RSA adalah :

1. Memilih pasangan bilangan prima p dan q
2. Hitung n = pq
3. Hitung $\phi(n) = (p-1)(q-1)$
4. Memilih sembarang e relatif prima terhadap $\phi(n)$
5. Menghitung kunci d, dimana $d \equiv e^{-1} \bmod \phi(n)$

Sehingga kunci publik adalah pasangan nilai e dan n dan kunci privat adalah pasangan nilai d dan n.

Proses enkripsi algoritma RSA adalah :

1. Membagi plainteks menjadi blok-blok plainteks m_0, m_1, m_2, ..., m_{n-1} dengan syarat $0 < m_i < n-1$
2. Menghitung blok cipherteks c_i untuk blok plainteks p_i dengan persamaan
   $$c_i = m_i^{\,e}\ mod\ n$$

Proses dekripsi algoritma RSA adalah :

Untuk seluruh blok-blok cipherteks c_i yang dihasilkan dari proses enkripsi, akan didekripsi menghasilkan plainteks semula p_i dengan persamaan

$$m_i = c_i^{\,d}\ mod\ n$$

D. Algoritma Rabin

Algoritma Rabin adalah algoritma kriptografi kunci publik yang dibuat oleh Michael O. Rabin pada tahun 1979. Tingkat keamanan algoritma ini sama dengan algoritma RSA yaitu sulitnya memfaktorkan bilangan yang besar menjadi dua faktor prima. Algoritma Rabin memiliki tiga proses utama yaitu proses pembentukan kunci, proses enkripsi, dan proses dekripsi.

Algoritma Rabin memiliki komponen antara lain :

1. p dan q bilangan prima — *rahasia*
2. n = pq — *tidak rahasia*
3. p invers dan q invers dimana : p * p invers + q * q invers = 1 — *rahasia*
4. plainteks m — *rahasia*
5. cipherteks k — *tidak rahasia*

Proses pembentukan kunci pada algoritma Rabin adalah :

1. Memilih pasangan bilangan prima p dan q
2. Hitung n = pq

Sehingga kunci publik adalah n dan kunci privat adalah p dan q.

Proses enkripsi algoritma Rabin adalah :

1. Membagi plainteks menjadi blok-blok plainteks m_0, m_1, m_2, ..., m_{n-1} dengan syarat $0 < m_i < n$
2. Menghitung blok cipherteks c_i untuk blok plainteks p_i dengan persamaan

$$c_i = {m_i}^2 \ mod \ n$$

Proses dekripsi algoritma Rabin adalah :

Proses dekripsi dilakukan dengan menggunakan operasi perpangkatan modular dan *Chinese Remainder Theorem.* Dari proses dekripsi algoritma Rabin akan dihasilkan empat buah blok plainteks, dimana plainteks yang sebenarnya adalah salah satu dari keempat blok plainteks tersebut.

Untuk seluruh blok-blok cipherteks c_i yang dihasilkan dari proses enkripsi, akan didekripsi menghasilkan empat buah plainteks dengan persamaan

$$m_p = \sqrt{c_i} \ mod \ p$$
$$m_q = \sqrt{c_i} \ mod \ q$$

$$m_1 = (p \cdot pinvers \cdot mq + q \cdot qinvers \cdot mp) mod \ n$$
$$m_2 = n - m1$$
$$m_3 = (p \cdot pinvers \cdot mq - q \cdot qinvers \cdot mp) mod \ n$$
$$m_4 = n - m3$$

E. Algoritma ElGamal

Algoritma ElGamal adalah algoritma kriptografi kunci publik yang dibuat oleh Taher Elgamal pada tahun 1985. Keamanan algoritma ElGamal terletak pada sulitnya menghitung logaritma diskrit dalam suatu ruang modulo. Algoritma ElGamal memiliki tiga proses utama yaitu proses pembentukan kunci, proses enkripsi, dan proses dekripsi.

Algoritma ElGamal memiliki komponen antara lain :

1.	Bilangan prima p	*tidak rahasia*
2.	Bilangan acak g (g<p)	*tidak rahasia*
3.	Bilangan acak x(x<p)	*rahasia*
4.	$y = g^x \bmod p$	*tidak rahasia*
5.	plainteks m	*rahasia*
6.	cipherteks c	*tidak rahasia*

Proses pembentukan kunci pada algoritma ElGamal adalah :

1. Memilih bilangan prima p
2. Memilih bilangan acak g dan x (g <p dan 0 <x <p-1)
3. Hitung $y = g^x \bmod p$

Sehingga kunci publik adalah pasangan nilai g, y, dan p dan kunci privat adalah pasangan nilai x dan p.

Proses enkripsi algoritma RSA adalah :

1. Membagi plainteks menjadi blok-blok plainteks m_0, m_1, m_2, ..., m_{n-1} dengan syarat $0 < m_i < p-1$
2. Memilih bilangan acak k dimana $0 < k < p-1$
3. Menghitung blok cipherteks a dan b untuk blok plainteks p_i dengan persamaan

$$a_i = g^k \ mod \ p$$
$$b = y^k \cdot m \ \ mod \ p$$

Pada proses enkripsi ElGamal, setiap blok akan menghasilkan dua buah cipherteks, sehingga cipherteks utuh berukuran dua kali ukuran plainteks utuh.

Proses dekripsi algoritma ElGamal adalah :

Untuk seluruh blok-blok cipherteks a_i dan b_i yang dihasilkan dari proses enkripsi, akan didekripsi menghasilkan plainteks semula p_i dengan persamaan

$$m_i = b_i({a_i}^{-x}) \ mod \ p$$

F. Metode Binary Perpangkatan Modular

Perpangkatan modular adalah suatu operasi matematika yang melibatkan operasi perpangkatan dan operasi modular, yaitu mencari suatu nilai x dari persamaan $x = a^b \ mod \ c$. Jika menggunakan cara yang biasa atau yang dilakukan oleh kalkulator yaitu menghitung terlebih dahulu a^b lalu mencari hasil modulo terhadap c sangatlah tidak efisien. Kekurangannya adalah iterasi yang dilakukan linear terhadap bilangan pangkat b, dan jika b adalah suatu bilangan yang sangat besar, hal ini dapat menyebabkan *overflow* pada memori.

Metode binary adalah suatu metode perpangkatan modular dengan menggunakan prinsip jika b adalah suatu bilangan genap, maka $a^b \ mod \ c = a^{b/2^2} \ mod \ c$. Sehingga kompleksitasnya bukanlah linear seperti metode naive, namun menjadi $\log_2 n$. Algoritma untuk metode binary dapat dilihat pada Algoritma 1.

Metode Binary

Input : a, b, c

Output : $x = a^b \ mod \ c$

1. if $b_{k-1} = 1$ then $x = a$ else $x = 1$
2. for i = k-2 downto 0
   2a. $x = x * x \ mod \ c$
   2b. if $b_i = 1$ then $x = x * a \ mod \ c$
3. return x

Algoritma 1. Metode Binary Perpangkatan Modular

G. Pemrograman Paralel

Pemrograman paralel merupakan pemrograman komputer yang eksekusinya dapat dilakukan secara simultan, baik dalam satu komputer (komputer multiprosesor) maupun banyak komputer. Latar belakang adanya pemrograman paralel adalah lamanya waktu atau kurang efisiennya waktu ketika suatu persoalan komputasi dijalankan secara serial, sehingga tujuan adanya pemrograman paralel adalah untuk meningkatkan performa komputasi.

Pemrograman paralel dapat dijalankan di beberapa lingkungan. Salah satu contohnya adalah pada GPU. GPU awalnya dirancang untuk mengolah grafis, namun seiring dengan perkembangan zaman, GPU saat ini dapat digunakan menjadi prosesor yang paralel dengan kemampuan *multithread.* Perbedaannya dengan CPU adalah GPU dispesialisasikan untuk masalah komputasi yang besar dan dapat diparalelkan.

III.ANALISIS ALGORITMA

A. Analisis Algoritma RSA

Algoritma RSA memiliki tiga proses utama yaitu pembentukan kunci, proses enkripsi , dan proses dekripsi. Karena proses pembetukan kunci hanya memerlukan iterasi komputasi sebanyak satu kali saja, maka proses ini lebih cocok jika diimplementasikan secara sekuensial saja. Pada proses enkripsi dan dekripsi, seluruh blok-blok data (plainteks maupun cipherteks) akan dieksekusi dengan algoritma yang sama atau biasa disebut dengan SIMD (Single Instruction Multiple Data) dan blok yang satu tidak saling bergantung dengan blok yang lain. Hal ini membuat proses enkripsi dan dekripsi RSA sangat cocok untuk diimplementasikan secara paralel, dimana masing-masing thread akan menangani satu blok data. Sehingga jumlah thread yang digunakan akan sama dengan banyaknya blok data yang akan dienkripsi maupun dekripsi.

Di dalam proses enkripsi maupun dekripsi yang dilakukan pada blok pesan melibatkan operasi perpangkatan modular yaitu mencari suatu nilai dari persamaan :

$$c = m^e \bmod n \text{ (enkripsi)}$$

$$m = c^d \bmod n \text{ (dekripsi)}$$

Kedua operasi tersebut (enkripsi dan dekripsi) melibatkan bilangan bulat yang besar, agar tingkat keamanan lebih baik. Akibatnya cara biasa ataupun cara naive menjadi kurang efisien untuk menghitung persamaan tersebut karena banyaknya iterasi yang dilakukan akan bertambah secara linear seiring meningkatnya nilai eksponen (d atau e). Oleh karena itu, operasi perpangkatan modular ini akan lebih baik jika dioptimasi dengan menggunakan metode binary. Dengan metode binary, banyaknya iterasi yang dilakukan adalah sebanyak logaritma 2 dari nilai eksponen.

B. Analisis Algoritma Rabin

Sama seperti algoritma RSA, algoritma Rabin memiliki tiga proses utama yaitu pembentukan kunci, proses enkripsi , dan proses dekripsi. Karena proses pembetukan kunci hanya memerlukan iterasi komputasi sebanyak satu kali saja, maka proses ini lebih cocok jika diimplementasikan secara sekuensial. Pada proses enkripsi dan dekripsi, seluruh blok-blok data (plainteks maupun cipherteks) akan dieksekusi dengan algoritma yang sama atau biasa disebut dengan SIMD (Single Instruction Multiple Data) dan blok yang satu tidak saling bergantung dengan blok yang lain. Hal ini membuat proses enkripsi dan dekripsi Rabin sangat cocok untuk diimplementasikan secara paralel, dimana masing-masing thread akan menangani satu blok data. Sehingga jumlah thread yang digunakan akan sama dengan banyaknya blok data yang akan dienkripsi maupun dekripsi.

Proses enkripsi yang dilakukan pada masing-masing blok pesan hanyalah melakukan satu buah operasi perkalian modular saja. Hal ini tidak menjadi masalah, karena iterasi yang akan dilakukan untuk proses ini adalah 1 kali untuk operasi perkalian dan 1 kali untuk operasi modular. Namun untuk proses dekripsi, masing-masing blok data akan terlibat dengan operasi perpangkatan modular, yaitu mencari suatu nilai dari persamaan :

$$a = c^{(p+1)/4} \bmod p \text{ (dekripsi)}$$

$$b = c^{(q+1)/4} \bmod q \text{ (dekripsi)}$$

Kedua operasi tersebut (dekripsi) melibatkan bilangan bulat yang besar, agar tingkat keamanan lebih baik. Akibatnya cara biasa ataupun cara naive menjadi kurang efisien untuk menghitung persamaan tersebut karena banyaknya iterasi yang dilakukan akan bertambah secara linear seiring meningkatnya nilai eksponen (p atau q). Oleh karena itu, operasi perpangkatan modular ini akan lebih baik jika dioptimasi dengan menggunakan metode binary. Dengan metode binary, banyaknya iterasi yang dilakukan adalah sebanyak logaritma 2 dari nilai eksponen.

C. Analisis Algoritma ElGamal

Sama seperti RSA dan Rabin, algoritma ElGamal juga memiliki tiga proses utama yaitu pembentukan kunci, proses enkripsi , dan proses dekripsi. Karena proses pembetukan kunci hanya memerlukan iterasi komputasi sebanyak satu kali saja, maka proses ini lebih cocok jika diimplementasikan secara sekuensial saja. Pada proses enkripsi dan dekripsi, seluruh blok-blok data (plainteks maupun cipherteks) akan dieksekusi dengan algoritma yang sama atau biasa disebut dengan SIMD (Single Instruction Multiple Data) dan blok

yang satu tidak saling bergantung dengan blok yang lain. Hal ini membuat proses enkripsi dan dekripsi ElGamal sangat cocok untuk diimplementasikan secara paralel, dimana masing-masing thread akan menangani satu buah data. Sehingga jumlah thread yang digunakan akan sama dengan banyaknya blok data yang akan dienkripsi maupun dekripsi.

Di dalam proses enkripsi maupun dekripsi yang dilakukan pada blok pesan melibatkan operasi perpangkatan modular yaitu mencari suatu nilai dari persamaan :

$$a_n = g^k \bmod p \text{ (enkripsi)}$$

$$b_n = y^k \cdot m_n \bmod p \text{ (enkripsi)}$$

$$m_n = b_n \bullet {a_n}^{p-1-x} \bmod p \text{ (dekripsi)}$$

Ketiga operasi tersebut (2 enkripsi dan 1 dekripsi) melibatkan bilangan bulat yang besar, agar tingkat keamanan lebih baik. Akibatnya cara biasa ataupun cara naive menjadi kurang efisien untuk menghitung persamaan tersebut karena banyaknya iterasi yang dilakukan akan bertambah secara linear seiring meningkatnya nilai eksponen (k atau p). Oleh karena itu, operasi perpangkatan modular ini akan lebih baik jika dioptimasi dengan menggunakan metode binary. Dengan metode binary, banyaknya iterasi yang dilakukan adalah sebanyak logaritma 2 dari nilai eksponen.

IV. RANCANGAN ALGORITMA

A. Interaksi antara CPU dan GPU

Untuk melakukan paralelisasi algoritma RSA, Rabin, dan ElGamal maka interaksi antara CPU dan GPU adalah penting seperti alokasi memori dan penyalinan informasi. Interaksi CPU dan GPU dalam melakukan proses enkripsi adalah :

1. CPU menerima informasi kumpulan blok plainteks dan kunci publik
2. CPU mengalokasikan memori pada GPU sesuai ukuran informasi yang akan dikirim
3. CPU menyalin informasi enkripsi dari host (CPU) ke device(GPU)
4. CPU melakukan invokasi kernel pada GPU
5. Masing--masing thread pada GPU menjalankan kode pada kernel yang berisi algoritma enkripsi. Banyaknya thread yang digunakan adalah sama dengan banyaknya blok plainteks. Masing-masing thread akan menghasilkan dan menyimpan cipherteks
6. GPU menyalin informasi cipherteks dari device(GPU) ke host (CPU)

Interaksi CPU dan GPU dalam melakuan proses dekripsi adalah :

1. CPU menerima informasi kumpulan blok cipherteks dan kunci privat
2. CPU mengalokasikan memori pada GPU sesuai ukuran informasi yang akan dikirim
3. CPU menyalin informasi dekripsi dari host (CPU) ke device(GPU)
4. CPU melakukan invokasi kernel pada GPU
5. Masing--masing thread pada GPU menjalankan kode pada kernel yang berisi algoritma dekripsi. Banyaknya thread yang digunakan adalah sama dengan banyaknya blok/pasangan blok cipherteks. Masing-masing thread akan menghasilkan dan menyimpan plainteks.
6. GPU menyalin informasi plainteks dari device(GPU) ke host (CPU)

B. Algoritma untuk Kernel GPU

Agar GPU dapat menjalankan algoritma secara paralel, maka diperlukan invokasi kernel oleh CPU. Kernel berisi kode pemrograman yang ditulis dalam bahasa CUDA C, yang akan dieksekusi oleh seluruh *thread* yang ada pada CUDA secara bersamaan sesuai dengan banyak *core* yang dimiliki oleh GPU. Kode kernel untuk proses enkripsi dan dekripsi RSA, Rabin, dan ElGamal dapat dilihat pada Algoritma 2 hingga Algoritma 7.

```
kernelenkripsiRSA
Input : e,n,* m
Output :c

1.i=blockDim.x*blockIdx.x+threadIdx.x
2.c_i = modexp(m_i, e, n)
```

Algoritma 2. Kernel Enkripsi RSA

```
kerneldekripsiRSA
Input : d,n,* c
Output :m

1.i=blockDim.x*blockIdx.x+threadIdx.x
2.m_i = modexp(c_i, d, n)
```

Algoritma 3. Kernel Dekripsi RSA

kernelenkripsiRabin

Input : $n,* m$

Output :c

1.i=blockDim.x*blockIdx.x+threadIdx.x

2.$c_i = m_i * m_i \ mod \ n$

Algoritma 4. Kernel Enkripsi Rabin

kerneldekripsiRabin

Input : $p, q, pinv, qinv, n,* c$

Output :m

1.i=blockDim.x*blockIdx.x+threadIdx.x

2.a. $mp = \ modexp(c_i, (p+1)/4, p)$

b. $mq = \ modexp(c_i, (q+1)/4, q)$

2.a. $m_{4*i} = (p * pinv * mq + q * qinv * mp) \ mod \ n$

b. $m_{4*i+1} = n - m_{4*i}$

c. $m_{4*i+2} = (p * pinv * mq - q * qinv * mp) \ mod \ n$

d. $m_{4*i+3} = n - m_{4*i+2}$

Algoritma 5. Kernel Dekripsi RSA

kernelenkripsiElGamal

Input : $g, y, p,* m ,* k$

Output :c

1.i=blockDim.x*blockIdx.x+threadIdx.x

2.a. $c_{2 \cdot i} = modexp(g, k_i, p)$

b. $c_{2 \cdot i+1} = m_i \ \cdot \ modexp(y, k_i, p) \ mod \ p$

Algoritma 6. Kernel Enkripsi ElGamal

kerneldekripsiElGamal

Input : $* c , x, p$

Output :m

1.i=blockDim.x*blockIdx.x+threadIdx.x

2.$m_i \ = \ c_{2 \cdot i+1} \ \cdot \ modexp(c_{2 \cdot i}, p-1-x, p) \ mod \ p$

Algoritma 7. Kernel Dekripsi ElGamal

Dari 6 algoritmayang diimplementasikan, 5 diantaranya memanggil fungsi modexp. Fungsi modexp adalah fungsi untuk menghitung nilai dari suatu persamaan modular eksponensial dengan menggunakan metode binary. Algoritma untuk modexp dapat dilihat pada Algoritma 8.

modexp

Input : a , b , c

Output : $x \ = \ a^b \ mod \ c$

1. $x = 1$

2. for i = k-1 downto 0

2a. $x = x \ * x \ mod \ c$

2b. if $b_i = 1$ then $x = x \ * a \ mod \ c$

3. return x

Algoritma 8. Algoritma Modexp

V. PENGUJIAN

Pengujian untuk algoritma RSA, Rabin dan ElGamal, dilakukan dengan membandingkan durasi eksekusi ketika diimplementasikan secara paralel terhadap durasi eksekusi ketika diimplementasikan secara sekuensial untuk beberapa variasi ukuran data.

A. Pengujian RSA

Hasil pengujian implementasi paralel dan sekuensial untuk enkripsi dan dekripsi algoritma RSA pada berbagai variasi ukuran data dapat dilihat pada Tabel 1 dan Tabel 2.

Tabel 1. Perbandingan Hasil Implementasi Sekuensial dan Paralel pada Enkripsi Algoritma RSA

Ukuran data (byte)	Durasi Paralel (ms)	Durasi Sekuensial (ms)	Speed Up
1024	0.039	0.287	7.35
2048	0.046	0.610	13.26
4096	0.047	1.260	26.80
8192	0.037	1.515	40.94
16384	0.023	3.732	162.26
32768	0.036	7.812	217.00

Tabel 2. Perbandingan Hasil Implementasi Sekuensial dan Paralel pada Dekripsi Algoritma RSA

Ukuran data (byte)	Durasi Paralel (ms)	Durasi Sekuensial (ms)	Speed Up
1024	0.022	0.290	13.18
2048	0.026	0.357	13.73
4096	0.020	1.278	63.90
8192	0.016	1.438	89.87
16384	0.028	2.951	105.39
32768	0.034	6.004	176.59

B. Pengujian Rabin

Hasil pengujian implementasi paralel dan sekuensial untuk enkripsi dan dekripsi algoritma Rabin dapat dilihat pada Tabel 3 dan Tabel 4.

Tabel 3. Perbandingan Hasil Implementasi Sekuensial dan Paralel pada Enkripsi Algoritma Rabin

Ukuran data (byte)	Durasi Paralel (ms)	Durasi Sekuensial (ms)	Speed Up
1024	0.037	0.051	1.38
2048	0.036	0.101	2.80
4096	0.053	0.152	2.86
8192	0.036	0.407	11.30
16384	0.013	0.808	62.15
32768	0.014	1.043	74.50

Tabel 4. Perbandingan Hasil Implementasi Sekuensial dan Paralel pada Dekripsi Algoritma Rabin

Ukuran data (byte)	Durasi Paralel (ms)	Durasi Sekuensial (ms)	Speed Up
1024	0.035	0.459	13.11
2048	0.032	0.772	24.12
4096	0.025	0.872	34.88
8192	0.021	3.084	146.85
16384	0.024	5.055	210.42
32768	0.042	9.242	280.95

C. Pengujian ElGamal

Hasil pengujian implementasi paralel dan sekuensial untuk enkripsi dan dekripsi algoritma Rabin dapat dilihat pada Tabel 5 dan Tabel 6.\

. Tabel 5. Perbandingan Hasil Implementasi Sekuensial dan Paralel pada Enkripsi Algoritma ElGamal

Ukuran data (byte)	Durasi Paralel (ms)	Durasi Sekuensial (ms)	Speed Up
1024	0.054	0.698	12.92
2048	0.057	1.378	24.18
4096	0.059	2.756	46.71
8192	0.046	4.543	98.76
16384	0.042	9.112	216.92
32768	0.070	16.556	236.42

Tabel 6. Perbandingan Hasil Implementasi Sekuensial dan Paralel pada Dekripsi Algoritma ElGamal

Ukuran data (byte)	Durasi Paralel (ms)	Durasi Sekuensial (ms)	Speed Up
1024	0.024	0.297	12.37
2048	0.029	0.577	19.89
4096	0.018	1.164	64.67
8192	0.024	1.907	79.45
16384	0.024	2.735	113.95
32768	0.032	5.216	163.00

D. Analisis Hasil Pengujian

Dari seluruh data hasil perbandingan antara implementasi paralel dan implementasi sekuensial ketiga algoritma kunci publik RSA, Rabin, dan ElGamal, dapat dilihat hubungan antara ukuran data enkripsi/dekripsi dalam byte dan durasi eksekusinya dalam milidetik. Pada implementai sekuensial jelas terlihat bahwa meningkatnya ukuran data akan menyebabkan meningkatnya durasi eksekusi secara linear. Akan tetapi hal ini berbeda pada implementasi paralel dimana durasi eksekusi relatif konstan. Hal ini disebabkan implementasi sekuensial hanya dijalankan oleh satu *core* sementara implementasi paralel dijalankan oleh banyak *core* secara bersamaan.

Untuk setiap tabel hasil pengujian terlihat bahwa semakin besar ukuran data atau semakin banyak thread yang digunakan, maka speed up yang didapatkan semakin besar, hal ini terjadi karena penambahan jumlah *thread* mengakibatkan tingkat konkurensi menjadi semakin tinggi, sehingga berdampak baik pada peningkatan kinerja.

Dengan membandingkan ketiga hasil pengujian, untuk banyak thread yang sama, terlihat perbedaan *speedup* antara algoritma RSA, Rabin, dan ElGamal. Dapat dilihat untuk ukuran thread sebesar 32768, pada enkripsi algoritma Rabin hanya menghasilkan speed up sebesar 74.5, sementara dekripsi algoritma Rabin dapat menghasilkan *speedup* sebesar 280.95. Didapatkan bahwa semakin rumit atau kompleks komputasi yang ada pada suatu algoritma, maka performa yang akan didapatkan ketika diimplementasikan secara paralel akan lebih maksimal.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Pada makalah ini dilakukan eksperimen untuk mengoptimasi algoritma kunci publik RSA, Rabin, dan ElGamal, dari hasil pengujian didapatkan untuk seluruh variasi ukuran data, waktu eksekusi secara paralel selalu lebih kecil dibandingkan dengan waktu eksekusi sekuensial. Semakin banyak thread yang digunakan, maka kinerja yang didapatkan juga akan semakin meningkat. Oleh karena itu optimasi dengan menggunakan pemrograman paralel pada CUDA sangat cocok digunakan untuk menangani kekurang efisienan enkripsi dan dekripsi kunci publik RSA, Rabin, dan ElGamal.

B. Saran

Agar speed up yang dihasilkan semakin besar, maka implementasi paralel algoritma RSA, Rabin, dan ElGamal dapat dilakukan pada GPU yang memiliki core lebih banyak atau dilakukan lebih dari satu buah GPU.

REFERENSI

1. ElGamal, T. 1985. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.*
2. Mahajan, S. & Singh, M. 2014. *Analysis of RSA Algorithm Using GPU Programming.*
3. Munir, Rinaldi. 2018. *Slide Kuliah IF4020 Kriptografi: Algoritma Kriptografi Modern*.
4. NVIDIA. 2018. *CUDA Toolkit Documentation v9.1*, https://docs.nvidia.com/cuda.
5. Rabin, M. 1979. *Digitalized Signatures and Public Key Functions as Intractable as Factorization*.
6. Rauber, T. & Runger, G. 2010. *Parallel Programming for Multicore and Cluster Systems.*

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, 18 Mei 2018

Harry Purba