Peningkatan Kinerja Algoritma ElGamal dengan Pemrograman Paralel pada Platform CUDA

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

Dr. Ir. Rinaldi Munir, M.T.
Teknik Informatika, STEI
Institut Teknologi Bandung
Bandung, Indonesia
rinaldi-m@stei.itb.ac.id

Dr. Judhi Santoso, M.Sc,
Teknik Informatika, STEI
Institut Teknologi Bandung
Bandung, Indonesia
judhi@informatika.com

***Abstract*—Algoritma ElGamal merupakan salah satu algoritma kriptografi kunci publik yang memiliki keamanan sangat tinggi. Namun, algoritma ElGamal masih memiliki kelemahan yaitu komputasi untuk melakukan enkripsi maupun dekripsi kurang efisien. Terdapat beberapa metode untuk mengatasi kelemahan tersebut, salah satunya adalah dengan mengimplementasikannya secara paralel. CUDA sebagai platform komputasi paralel *general purpose* pada GPU digunakan untuk implementasi ini. Dari hasil analisis algoritma ElGamal didapatkan bahwa proses enkripsi dan dekripsi pada ElGamal sangat memiliki potensi untuk untuk diimplementasikan secara paralel karena memiliki sifat SIMD dan saling independen. Peningkatan kinerja juga dapat dilakukan dengan menggunakan metode binary. Solusi optimasi kemudian diimplementasikan pada CUDA dan diuji. Dari hasil pengujian secara keseluruhan didapatkan bahwa implementasi paralel dapat meningkatkan kinerja algoritma ElGamal.**

***Keywords*— ElGamal, SIMD, paralel, kompleksitas**

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 komputasi yang dilakukan sangatlah kompleks. 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 ElGamal. Perbedaannya dengan algoritma kunci publik lain adalah keamanan algoritma ElGamal terletak pada sulitnya menghitung logaritma diskrit dalam ruang modulo. Untuk melakukan proses enkripsi maupun dekripsi, algoritma ElGamal 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 mengurangi waktu eksekusi Algoritma 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, Rijndael, Blowfish, IDEA, dan GOST. Algoritma asimetris 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 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 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 ElGamal 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

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

D. 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

E. 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

Proses kriptografi pada algoritma ElGamal terdiri dari tiga bagian besar yaitu proses pembentukan kunci, proses enkripsi, dan proses dekripsi. Proses pembentukan kunci hanya melakukan satu buah iterasi perpangkatan modular saja, oleh karena itu proses pembentukan kunci lebih baik jika diimplementasikan secara sekuensial pada CPU.

Pada proses enkripsi dan dekripsi, masing-masing blok plainteks / pasangan blok cipherteks akan dienkripsi / dekripsi menggunakan algoritma yang sama. Hal ini sangat sesuai dengan sifat pada arsitektur komputer SIMD (*Single Instruction Multiple Data*). Karena sifatnya SIMD dan tiap blok saling independen, proses enkripsi dan dekripsi sangat cocok jika diimplementasikan secara paralel. Setiap blok plainteks / pasangan blok cipherteks akan ditangani oleh satu buah *thread* yang kemudian secara bersama-sama akan dienkripsi/dekripsi.

Proses enkripsi maupun dekripsi yang dilakukan masing-masing blok pesan melibatkan operasi perpangkatan modular dengan formula sebagai berikut:

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

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

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

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. Metode binary dapat digunakan untuk menyederhanakan perpangkatan modular ini. Dengan metode binary, banyaknya iterasi yang dilakukan adalah sebanyak logaritma 2 dari nilai eksponen.

IV. RANCANGAN ALGORITMA

A. Interaksi antara CPU dan GPU

Proses paralelisasi algoritma ElGamal memerlukan interaksi antara CPU dan GPU. 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 melakukan 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 Algoritma ElGamal dapat dilihat pada Algoritma 2 dan Algoritma 3.

```
kernelenkripsiElGamal
Input : g,y,p,*m,*k
Output :c

1.i=blockDim.x*blockIdx.x+threadIdx.x
2.a. c_{2·i} = modexp(g,k_i,p)
  b. c_{2·i+1} = m_i · modexp(y,k_i,p) mod p
```

Algoritma 2. Kernel Enkripsi ElGamal

```
kerneldekripsiElGamal
Input : *c,x,p
Output :m

1.i=blockDim.x*blockIdx.x+threadIdx.x
2.m_i = c_{2·i+1} · modexp(c_{2·i},p−1−x,p) mod p
```

Algoritma 3. Kernel Dekripsi ElGamal

Terlihat kedua algoritma tersebut mempunyai operasi dasar yang sama yaitu operasi perpangkatan modular. Untuk operasi perpangkatan modular, kedua kernel kemudian akan memanggil fungsi modexp yaitu operasi perpangkatan modular dengan menggunakan metode binary (Algoritma 1).

V. IMPLEMENTASI DAN PENGUJIAN

Implementasi dan pengujian dilakukan pada perangkat keras dan perangkat lunak dengan spesifikasi dapat dilihat pada Tabel 1.

Tabel 1. Spesifikasi Perangkat Keras dan Lunak GPU

GPU	GeForce GTX 1080
CUDA Capability	6.1
Total Core	20 x 128
Total Global Memory	8118 MB
Total Constant Memory	65536 bytes
Shared Memory per Block	49152 bytes
Total Register per Block	65536
Maximum Thread Per Block	1024
Maximum Block Dimension	1024 x 1024 x 64
Maximum Grid Dimension	2147483647 x 65535 x 65535

Performa implementasi paralel Algoritma ElGamal diukur dengan menghitung nilai *throughput*. Nilai *throughput* didapat dengan menghitung hasil pembagian ukuran data yang dienkripsi/dekripsi terhadap durasi enkripsi/dekripsi.

Pengujian kinerja implementasi paralel Algoritma ElGamal dilakukan dengan menghitung nilai *speedup*. Nilai *speedup* didapatkan dengan menghitung rasio/perbandingan *throughput* implementasi paralel terhadap *throughput* implementasi serial. Nilai *throughput* implementasi serial untuk ukuran kunci kecil (32 bit) adalah 2645.98 KB/s untuk enkripsi dan 5074.218 KB/s untuk dekripsi. Nilai *throughput* implementasi serial untuk ukuran kunci besar (1024 bit) adalah 0.038 KB/s untuk enkripsi dan 0.075 KB/s untuk dekripsi.

A. Hasil Pengujian untuk Ukuran Kunci Kecil (32 Bit)

Hasil pengujian implementasi paralel untuk ukuran kunci kecil (32 bit) dapat dilihat pada Tabel 2 hingga Tabel 5.

Tabel 2. Hasil Pengujian Enkripsi Ukuran Kunci Kecil

Ukuran Data (Byte)	Throughput Paralel (KB/s)	Speedup
20480	81.41299927	0.03076858
40960	163.7830325	0.06189885
81920	329.4534866	0.12451101
163840	657.7099622	0.24856963
327680	1307.094181	0.49399269
655360	2600.697714	0.98288683
1310720	5226.214077	1.97515341
2621440	10173.05313	3.84472208
5242880	19363.01054	7.31790084
20971520	58900.84819	22.260514
83886080	140214.1473	52.9914098
335544320	201775.1373	76.2572764

Tabel 3. Hasil Pengujian Dekripsi Ukuran Kunci Kecil

Ukuran Data (Byte)	Throughput Paralel (KB/s)	Speedup
20480	82.55640761	0.01626978
40960	164.237753	0.03236711
81920	330.7188512	0.06517632
163840	664.2287748	0.13090269
327680	1301.999035	0.25659107
655360	2644.906578	0.52124417
1310720	5284.75655	1.04149181
2621440	10286.01665	2.02711364
5242880	19785.77903	3.89927646
20971520	68601.84701	13.5196884
83886080	186557.1869	36.7657015
335544320	304125.3635	59.9354146

Tabel 4. Hasil Pengujian Enkripsi Ukuran Kunci Besar

Ukuran Data (Byte)	Throughput Paralel (KB/s)	Speedup
20480	0.543880202	14.1677805
40960	1.095112947	28.5270906
81920	2.188492267	57.0090213
163840	4.152435295	108.168658
327680	4.174870172	108.753074
655360	4.142303285	107.904725
1310720	4.126115939	107.483053
2621440	4.129748381	107.577677
5242880	4.032178018	105.036022

Tabel 5. Hasil Pengujian Dekripsi Ukuran Kunci Besar

Ukuran Data (Byte)	Throughput Paralel (KB/s)	Speedup
20480	1.346992882	17.9915307
40960	2.723943616	36.3832028
81920	5.429634693	72.5226099
163840	10.14060252	135.446122
327680	10.83052836	144.661331
655360	13.72690482	183.347687
1310720	14.20211422	189.69497
2621440	15.07824442	201.397276
5242880	15.31077668	204.503166

B. Analisis Hasil Pengujian

Dapat dilihat pada seluruh data hasil pengujian bahwa implementasi paralel mengakibatkan peningkatan kinerja yang besar, terlihat dari nilai *speedup* yang tinggi untuk ukuran kunci kecil maupun besar. Peningkatan kinerja yang ini diakibatkan oleh beberapa hal berikut :

1. Transfer data yang dilakukan baik dari memori CPU ke GPU maupun dari memori GPU ke CPU relatif sedikit dan dilakukan hanya dua kali saja (sebelum dan sesudah pemanggilan kernel), sehingga waktu untuk melakukan transfer jauh lebih kecil jika dibandingkan dengan waktu melakukan enkripsi/dekripsi.

2. Masing-masing *thread* akan menjalankan komputasi yang sangat besar, dimana setiap *thread* akan melakukan dua kali operasi perpangkatan modular pada enkripsi dan satu kali pada dekripsi. Hal ini mengakibatkan usaha-usaha lain seperti pembangkitan *thread* menjadi sangat kecil dan tidak sebanding. Dengan demikian waktu pemrosesan lebih banyak dialokasikan untuk melakukan komputasi inti, hal ini membuat penggunaan *thread* menjadi efektif.

3. Penggunaan jumlah *thread* yang sangat besar mengakibatkan tingkat paralelisme menjadi sangat tinggi, sehingga berdampak baik pada peningkatan kinerja. Dibandingkan dengan eksekusi yang dilakukan oleh CPU yang menggunakan satu prosesor saja, GPU melakukan eksekusi dengan lebih dari satu core, sehingga sangat banyak blok data yang dapat diproses secara bersamaan.

Pada seluruh tabel hasil pengujian terlihat bahwa semakin banyak *thread* yang digunakan atau semakin besar data yang dienkripsi atau dekripsi, maka semakin tinggi kinerja yang didapatkan dari implementasi paralel. Hal ini disebabkan semakin banyak *thread* yang digunakan, maka seluruh *core* yang dimiliki oleh GPU akan semakin produktif (pembagian pekerjaan merata), sehingga semakin banyak blok data yang diproses dalam suatu kurun waktu. Namun penggunaan *thread* ini memiliki suatu titik optimum dimana penambahan *thread* berikutnya tidak lagi akan menambah *speedup* melainkan tetap atau bahkan menurun. Hal ini disebabkan seluruh *core* telah melakukan performa maksimumnya, dan adanya biaya *overhead* ketika masing-masing kernel dijalankan. Biaya *overhead* yang sangat mempengaruhi *throughput* implementasi paralel pada kasus ini adalah banyaknya akses memori yang dilakukan oleh masing-masing *thread* terutama akses ke memori global. Semakin besar ukuran kunci, maka akses memori terutama akses global memory yang dilakukan oleh masing-masing kernel semakin banyak yang membuat *throughput* menjadi semakin kecil.

Pada Tabel 2 dan Tabel 3, hasil pengujian untuk ukuran kunci kecil (32 bit), terlihat bahwa untuk data yang berukuran lebih kecil dari 1,3 MB, *throughput* implementasi paralel enkripsi maupun dekripsi yang dihasilkan lebih kecil dari *throughput* implementasi serial. Hal ini disebabkan alokasi memori dan penyalinan memori pada GPU masih memakan waktu yang jauh lebih besar dibandingkan dengan komputasi inti yang dilakukan (enkripsi/dekripsi) (99 % waktu eksekusi dilakukan oleh alokasi, sementara seluruh API lainnya termasuk komputasi inti hanya menggunakan 1 % sisanya), sehingga penggunaan implementasi paralel pada ukuran data yang lebih kecil dari 1,3 MB menjadi tidak efektif.

VI. KESIMPULAN DAN SARAN

A. Kesimpulan

Pada makalah ini dilakukan eksperimen untuk meningkatkan kinerja algoritma ElGamal khususnya waktu eksekusi. Peningkatan kinerja algoritma ElGamal dapat dilakukan dengan menyederhanakan operasi matematika dalam algoritmanya menggunakan metode binary dan dengan mengimplementasikan proses enkripsi/dekripsi secara paralel. Semakin banyak *thread* yang digunakan, maka kinerja yang didapatkan juga akan semakin meningkat. Oleh karena itu peningkatan kinerja dengan menggunakan pemrograman paralel pada CUDA sangat cocok digunakan untuk menangani kekurang efisienan waktu enkripsi dan dekripsi Algoritma ElGamal.

B. Saran

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

REFERENSI

1. Dick, L. W. (2017). *Large Integer Arithmetic in GPU for Cryptography*.
2. ElGamal, T. 1985. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*.
3. Fadhil, H. M. (2014). *Parallelizing RSA Algorithm on Multicore CPU and GPU*.
4. Furqan, M. (2016). *Pemrograman Paralel pada GPU: Implementasi dan Optimasi Algoritma Kriptografi Serpent*. Tugas Akhir, ITB, Teknik Informatika.
5. Koc, C. K.(1994). *High Speed RSA Implementation*.
6. Mahajan, S. & Singh, M. 2014. *Analysis of RSA Algorithm Using GPU Programming.*
7. Munir, Rinaldi. 2018. *Slide Kuliah IF4020 Kriptografi: Algoritma Kriptografi Modern*.
8. NVIDIA. 2018. *CUDA Toolkit Documentation v9.1*, https://docs.nvidia.com/cuda.
9. Rauber, T. & Runger, G. 2010. *Parallel Programming for Multicore and Cluster Systems.*