Pengaruh Algoritma - Algoritma Pembangkit Bilangan Acak Semu Terhadap Performansi Algoritma Enkripsi ElGamal

Muhammad Reza Mandala Putra (13509003)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung 40132, Indonesia
13509003@std.stei.itb.ac.id

***Sari*—Algoritma ElGamal merupakan salah satu algoritma kriptografi kunci publik yang dibuat oleh Taher ElGamal. Dalam implementasinya, algoritma ini memanfaatkan penggenerasian bilangan acak sebagai salah satu kekuatan algoritma enkripsinya. Dalam makalah ini akan dibahas sejauh mana pengaruh algoritma-algoritma pembangkit bilangan acak semu terhadap performansi algoritma ElGamal.**

***Kata kunci*—algoritma kunci publik, algoritma Elgamal, *pseudo-random* number, performansi.**

I. PENDAHULUAN

Algoritma ElGamal merupakan salah satu algoritma kriptografi kunci-publik yang dibuat oleh Taher ElGamal pada tahun 1984. Algoritma ini pada umumnya digunakan untuk *digital signature*, namun kemudian dimodifikasi sehingga juga bisa digunakan untuk enkripsi dan deskripsi. Algoritma Elgamal digunakan dalam perangkat lunak sekuriti yang dikembangkan oleh GNU, program PGP, dan pada sistem sekuriti lainnya. Kekuatan algoritma ini terletak pada sulitnya menghitung logaritma diskrit.

Dalam implementasinya sering kali algoritma Elgamal menggunakan bilangan random dalam melakukan enkripsi pesan. Namun uniknya, nilai bilangan acak untuk melakukan enkripsi pesan ternyata tidak memengaruhi hasil dekripsi pesan. Pesan yang telah dienkripsi tetap dapat didekripsi dengan baik.

Hingga saat ini sudah cukup banyak terdapat algoritma yang dirancang untuk melakukan pembangkitan bilangan acak. Algoritma-algoritma tersebut biasanya dirancang untuk keperluan tertentu. Algoritma-algoritma tersebut biasanya dirancang sedemikian rupa sehingga nilai bilangan acak yang akan dibangkitkan menggunakan algoritma itu mendekati *truly random*, atau dengan kata lain bilangan-bilangan yang dibangkitkan terlihat benar-benar acak.

Algoritma-algoritma pembangkit bilangan acak tentunya memiliki kompleksitas algoritmanya masing-masing. Algoritma pembangkit bilangan acak yang satu belum tentu sama dengan yang lain. Dalam makalah yang akan dibuat ini akan dibahas perbandingan beberapa algoritma pembangkit bilangan acak serta pengaruhnya terhadap performansi algoritma Elgamal.

II. DASAR TEORI

A. Algoritma Elgamal

.Algoritma ElGamal merupakan algoritma enkripsi kunci asimetris untuk kriptografi kunci publik yang didasari kesepakatan kunci Diffie-Hellman. Pada tahun 1984, Taher ElGamal mempresentasikan suatu kriptosistem yang berdasarkan permasalahan logaritma diskrit. Kriptosistem ini didasari oleh asumsi bahwa permasalahan logaritma diskrit tidak dapat ditemukan dengan mudah, sementara operasi inverse dapat dihitung secara efisien.

Sistem kunci-publik orisinil yang diberikan oleh Diffie dan Hellman membutuhkan interaksi dari kedua belah pihak untuk menghitung kunci rahasia. Hal ini dapat menimbulkan masalah apabila kriptosistem ini diterapkan ke dalam suatu sistem komunikasi dimana kedua belah pihak tidak dapat berinteraksi karena adanya keterlambatan dalam transmisi atau karena pihak yang menerima sedang tidak ada. Algoritma ElGamal menyederhanakan algoritma pertukaran kunci Diffie-Hellman dengan memperkenalkan bilangan eksponen acak k. Bilangan eksponen ini menggantikan bilangan eksponen rahasia dari pihak penerima. Hasil dari penyederhanaan algoritma ini dapat digunakan untuk melakukan enkripsi dalam satu arah, tanpa perlu mendapatkan interaksi dari pihak kedua.

Besaran-besaran yang digunakan dalam pembangkitan kunci publik algoritma ElGamal:

1. Bilangan prima, p (tidak rahasia)
2. Bilangan acak, g (g < p) (tidak rahasia)
3. Bilangan acak, x (x < p) (rahasia)
4. Blok plainteks M (plainteks) (rahasia)
5. a dan b (cipherteks) (tidak rahasia)

Langkah-langkah dalam menggenerasi kunci publik y adalah sebagai berikut:

1. Pilih sembarang bilangan prima p,
2. Pilih dua buah bilangan acak, g dan x, dengan syarat g < p dan x < p.

3. Setelah bilangan p, g, dan x diketahui pada untuk membangkitkan kunci publik dapat dilakukan dengan rumus $y = g^x \bmod p$.
4. Didapat kunci publik y dan kunci privat x. Nilai g dan p tidak dirahasiakan dan dapat diumumkan kepada publik agar dapat mengenkripsi pesan yang akan dikirim

Untuk mengirim pesan kepada pihak yang mengumumkan kunci publik, terlebih dahulu pesan dienkripsi dengan memanfaatkan kunci publik yang telah diumumkan tersebut. Langkah-langkah melakukan enkripsi pesan adalah sebagai berikut:

1. Plainteks disusun menjadi blok-blok *m1*, *m2*, …, sedemikian sehingga setiap blok merepresentasikan nilai di dalam rentang 0 sampai *p–1*.
2. Pilih bilangan acak *k*, yang dalam hal ini $0 < k < p–1$, sedemikian sehingga *k* relatif prima dengan *p–1*.
3. Setiap blok m dienkripsi dengan rumus
$$a = g^k \bmod p\ ,$$
dan
$$b = y^k m \bmod p.$$
4. Pasangan a dan b adalah cipherteks untuk blok pesan *m*. Jadi, ukuran cipherteks dua kali ukuran plainteksnya.

Untuk mendekripsi a dan b digunakan kunci rahasia, *x*, dan plainteks *m* diperoleh kembali dengan persamaan
$$m = \frac{b}{a^x} \bmod p.$$
Karena
$$a^x \equiv g^{kx} \ (\bmod p)$$
maka
$$\frac{b}{a^x} \equiv \frac{y^k m}{a^x} \equiv \frac{g^{xk} m}{g^{xk}} \equiv m(\bmod p)$$
yang berarti bahwa plainteks dapat ditemukan kembali dari pasangan cipherteks a dan b [1].

B. Algoritma Pembangkit Bilangan Acak Semu

1. *Linear Congruential Generator*

Linear Congruential Generator (LCG) merupakan pembangkit bilangan acak yang sederhana, mudah dimengerti teorinya, dan juga mudah untuk diimplementasikan. LCG didefinisikan dalam relasi berulang berikut [2]:
$$x_{i+1} = \ (ax_i + c) \bmod m,$$
dimana
x_{i+1} = bilangan acak ke-*i*
x_i = bilangan acak sebelumnya
a = faktor pengali
c = *increment*
m = modulus
x_0 adalah kunci pembangkit atau disebut juga umpan (seed).

Periode LCG paling besar adalah *m* bahkan pada kebanyakan kasus periodenya kurang dari *m*. Maksudnya adalah deret bilangan acak yang dihasilkan tidak lebih banyak dari modulonya.

2. *Lagged Fibonacci Generator*

Lagged Fibonacci Generators (LFG) mencoba memperbaiki LCG dengan menggunakan lebih dari satu nilai sebelumnya dalam urutan, dalam rangka untuk mengurangi korelasi dan meningkatkan periode. Hal ini mirip dengan LCG gabungan, tetapi dalam kasus ini angka yang diambil dari urutan tunggal, bukan dua urutan independen.

Suatu bilangan Fibonacci dihasilkan dengan cara menambahkan dua deret bilangan yang dihasilkan sebelumnya, seperti rumus:
$$S_n = \ S_{n-1} + \ S_{n-2}$$
Namun, untuk menghasilkan bilangan acak berbasis deret Fibonacci dapat dilakukan dengan rumus:
$$S_n \equiv \ (S_{n-1} * \ S_{n-2}) \bmod p,$$
dengan * merupakan *binary operator*, seperti operator penjumlahan, pengurangan, perkalian, atau operator *bitwise* XOR [3]. Bilangan acak yang dihasilkan berada di antara 0 sampai p-1.

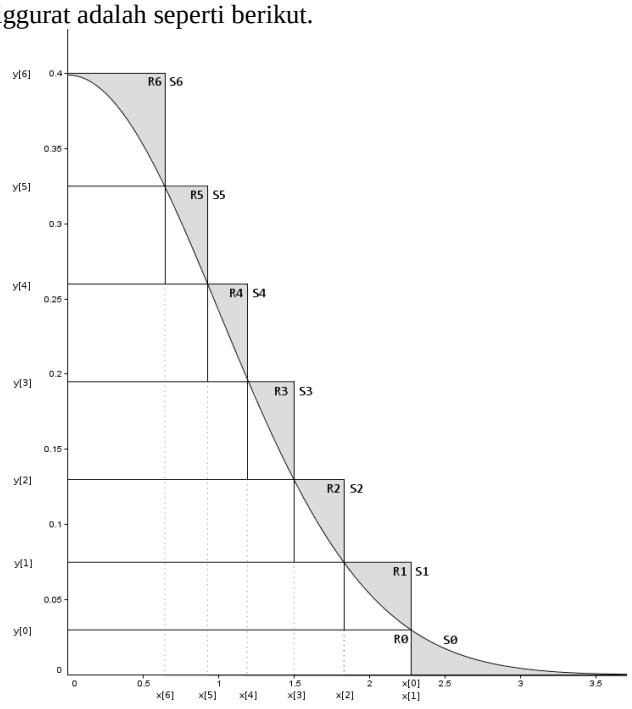
3. *Blum Blum Shub Generator*

Blum Blum Shub Generator (BBS) adalah sebuah algoritma pembangkit bilangan acak yang dibuat pada tahun 1986 oleh Lenore Blum, Manuel Blum, dan Michael Shub. BBS mengambil bentuk sebagai berikut:
$$x_i = x_i^2 \bmod m,$$
di mana $m = pq$ adalah hasil kali dari dua bilangan prima besar *p* dan *q*, serta *p* dan *q* harus kongruen dengan 3 (mod 4) [4].

4. *Ziggurat Algorithm*

Ziggurat Algorithm adalah metode untuk mengambil sampel bilangan acak secara efisien dari distribusi probabilitas seperti distribusi Gaussian. Algoritma ini merupakan salah satu algoritma penolakan pengambilan sampel bilangan acak yang dilakukan dengan cara sampel bilangan acak suatu distribusi *uniform* dipilih atau ditolak sehingga sampel yang dipilih memiliki distribusi tertentu yang diinginkan.

Grafik distribusi bilangan acak menggunakan algoritma Ziggurat adalah seperti berikut.



Gambar 1. Grafik distribusi Gaussian menggunakan algoritma Ziggurat

Langkah-langkah membangkitkan bilangan acak menggunakan algoritma Ziggurat adalah sebagai berikut:

1. Tentukan salah satu segmen dari grafik pada gambar 1 secara acak, sebut saja S_i
2. S_0 merukana kasus khusus. Jika S_0 terpilih, lalu generasikan sebuah bilangan acak yang terletak di antara 0 hingga luas area A. jika nilai tersebut kurang dari atau sama dengan area R_0 maka bilangan acak dapat diterima (dipilih). Jika tidak, ulangi langkah 1.
3. Apabila S_i bukan merupakan S_0, pilih bilangan acak *x* yang terletak pada R_i. Bilangan acak x dapat diterima apabila $x < x_{i+1}$. Namun jika tidak, pilih bilangan acak *y* yang terletak di antara R_i. Apabila $y < f(x)$, bikangan acak x dapat diterima. Jika tidak, ulangi langkah 1 hingga mendapatkan bilangan acak yang diinginkan.

Untuk selengkapnya dapat dilihat pada [5].

5. Mersenne Twister

Pada tahun 1997, algoritma *Mersenne Twister* ditemukan oleh Matsumoto and Takuji Nishimura. Algoritma ini dapat menjawab persoalan yang dihadapai oleh pembangkit-pembangkit acak semu sebelumnya. *Mersenne Twister* memiliki periode yang sangat besar yaitu 2^{19937}-1 iterasi yang tidak akan selesai dihitung secara komputasi selamanya. Keunggulan lainnya adalah memiliki keseragaman keluaran hingga dimensi 623 (untuk yang 32 bit), dan waktu eksekusinya lebih cepat daripada pembangkit acak yang baik lainnya. *Mersenne Twister* sekaran ini telah menjadi pilihan utama sebagai pembangkit bilangan acak semu untuk simulasi dan pemodelan [4].

Bilangan acak *Mersenne Twister* didapat melalui persamaan rekursif liner berikut [6]:

$$\vec{x}_{k+n} = \vec{x}_{k+m} \oplus \left(\overrightarrow{x}^{u}_{k} \,\middle|\, \overrightarrow{x}^{l}_{k+1} \right) A$$

III. ANALISIS DAN PENGUJIAN

Untuk mengukur performansi algoritma ElGamal dilakukan dengan dua cara. Cara pertama adalah dengan menghitung jumlah iterasi untuk mendapatkan bilangan acak yang relatif prima dengan *p-1*. Sedangakan cara kedua dilakukan dengan cara menghitung lama waktu untuk melakukan penggenerasian bilangan acak tersebut. Pengujian dilakukan dengan menggunakan parameter sebagai berikut:

- Pesan: ABCDEFGH
- Jumlah blok: 4
- P: 2579
- G: 2
- X: 765
- Y: 949

Hasil perbandingan dapat dilihat pada tabel berikut:

	Lama enkripsi (ms)
LCG	23
LFG	22
BBS	27
Ziggurat	672631
Mersenne Twister	54437

Tabel 1. Perbandingan lama enkripsi

Dari tabel di atas dapat dilihat perbandingan lama enkripsi algoritma ElGamal menggunakan pembangkit bilangan acak yang ada pada tabel. Pada tabel terlihat bahwa lama enkripsi ElGamal menggunakan pembangkit bilangan acak LCG, LFG, dan BBS adalah sekitar 20-30ms. Namun penggunaan algoritma pembangkit bilangan acak Ziggurat dan *Mersenne Twister* justru membuat algoritma ElGamal membutuhkan waktu yang lama untuk melakukan enkripsi. Hal ini dikarenakan banyaknya operasi biner yang terdapat di dalam kedua algoritma tersebut.

		Blok 1	Blok 2	Blok 3	Blok 4
LCG	Lama eksekusi (ms)	1.4714	0.0289	0.0326	0.0267
	Jumlah iterasi	1	1	1	1
	Bilangan acak	1633	1633	1633	1633
LFG	Lama eksekusi (ms)	2.6144	0.0294	0.0262	0.0267
	Jumlah iterasi	1	1	1	1
	Bilangan acak	945	357	2219	439
BBS	Lama eksekusi (ms)	1.9387	0.0362	0.0276	0.0267
	Jumlah iterasi	1	1	1	1
	Bilangan acak	2395	2553	625	1347
Ziggurat	Lama eksekusi (ms)	6375.9606	428248.6158	283.7214	237700.1564
	Jumlah iterasi	1	1	1	1
	Bilangan acak	2577	2577	2577	2577
Mersenne Twister	Lama eksekusi (ms)	2.7965	0.0729	0.0828	0.0371
	Jumlah iterasi	4	2	1	2
	Bilangan acak	78721	213767	230561	240883

Tabel 2. Perbandingan algoritma pseudo-random

Dari tabel 2 terlihat perbedaan lama eksekusi yang dihasilkan dari kelima algoritma yang dibandingkan. Dari tabel 2 terlihat bahwa lama eksekusi untuk hampir semua algoritma pembangkit bilangan acak semu berkisar antara 0 hingga 3 milisekon. Namun ternyata hal ini tidak berlaku untuk algoritma Ziggurat. Algoritma Ziggurat rata-rata membutuhkan waktu dalam hitungan detik untuk membangkitkan suatu bilangan acak.

Dari tabel 2 juga terlihat jumlah iterasi yang dibutuhkan oleh tiap-tiap algoritma pembangkit bilangan

acak semu untuk mencari suatu bilangan acak yang memiliki faktor pembagi terbesar dengan *p-1* bernilai 1, atau dengan kata lain bilangan acak tersebut relatif prima dengan *p-1*. Dari tabel 2 terlihat bahwa hampir semua algoritma langsung menemukan hanya dengan satu kali iterasi saja. Tetapi ternyata tidak untuk *Mersenne Twister*. Algoritma tersebut rata-rata membutuhkan lebih dari satu kali iterasi untuk mencari bilangan acak yang relatif prima dengan *p-1*.

Dari tabel 2 juga terlihat hasil bilangan acak yang dihasilkan oleh masing-masing algoritma untuk mengenkripsi tiap-tiap blok.dari tabel 2 terlihat bahwa algoritma LFG, BBS, dan *Mersenne Twister* menghasilkan bilangan acak yang berbeda-beda untuk tiap-tiap proses enkripsi. Hal ini dikaernakan pembangkitan bilangan acaknya dihasilkan dari operasi terhadap bilangan acak yang telah dihasilkan sebelumnya. Sedangkan untuk algoritma LCG dan Ziggurat, bilangan acak yang dihasilkan selalu sama. Walaupun bilangan acaknya dihasilkan dari operasi terhadap bilangan acak sebelumnya, operasi tersebut tidak memengaruhi hasil akhir dari pembangkitan bilangan acak berikutnya.

IV. KESIMPULAN

Dari hasil pengujian terhadap kelima algoritma pembangkit bilangan acak semu didapati beberapa kesimpulan sebagai berikut:

- Algoritma pembangkit bilangan acak semu cukup berpengaruh terhadap performansi algoritma enkripsi ElGamal.
- Waktu yang dibutuhkan algoritma ElGamal untuk melakukan enkripsi pesan "ABCDEFGH" memanfaatkan algoritma LCG, LFG, dan BBS berkisar antara 20-30ms, sedangkan Ziggurat dan *Mersenne Twister* lebih dari 50 detik.
- Waktu yang dibutuhkan oleh algoritma Ziggurat untuk menghasilkan bilangan acak jauh lebih lama dibandingkan dengan keempat algoritma lainnya.
- Rata-rata jumlah iterasi yang dibutuhkan oleh *Mersenne Twister* untuk menghasilkan bilangan acak yang relatif prima dengan *p-1* lebih banyak daripada keempat algoritma lainnya.
- Algoritma LCG dan Ziggurat memiliki keluaran bilangan acak yang sama pada tiap-tiap blok yang akan dienkripsi, sedangkan LFG, BBS, dan *Mersenne Twister* menghasilkan keluaran bilangan acak yang tidak sama.

.

V. APPENDIX

Daftar Gambar

Gambar 1. Grafik distribusi Gaussian menggunakan algoritma Ziggurat 3

Daftar Tabel

Tabel 1. Perbandingan lama enkripsi.....3

Tabel 2. Perbandingan algoritma pseudo-random.....3

VI. ACKNOWLEDGMENT

Pertama-tama saya panjatkan syukur kepada Tuhan atas selesainya makalah ini. Lalu, saya juga ingin mengucapkan terimakasih kepada Bapak Ir. Rinaldi Munir selaku dosen IF3058 Kriptografi, yang telah mengajar saya seluk beluk macam-macam dan teknik kriptografi hingga saat ini. Juga, saya ingin mengucapkan terimakasih kepada teman-teman yang selalu membantu saya, memberi semangat, dan petunjuk-petunjuk berharga.

REFERENSI

[1] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *Advances in Cryptology*, vol. 196, pp. 10 - 18, 1985.

[2] M. Xiannong, "Linear Congruential Method," Bucknell University, 18 10 2002. [Online]. Available: http://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node40.html. [Accessed 19 5 2013].

[3] P. Coddington, "Lagged Fibonacci Generators," [Online]. Available: http://cs.adelaide.edu.au/~paulc/teaching/montecarlo/node109.html. [Accessed 19 5 2013].

[4] "Pseudo-Random Number Generators," 2011. [Online]. Available: http://finsharplib.net/sitedoc/PseudoRandom.html. [Accessed 20 5 2013].

[5] "The Ziggurat Algorithm for Random Gaussian Sampling," 2011. [Online]. Available: http://colingreenstuff.org/zigguratalgorithm/zigguratalgorithm.html. [Accessed 19 5 2013].

[6] "Random Number Generation - Download as PowerPoint," 2011. [Online]. Available: http://www.docstoc.com/docs/122582920/Random-Number-Generation---Download-as-PowerPoint. [Accessed 19 5 2013].

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, 20 Mei 2013

Muhammad Reza Mandala Putra
13509003