JIPETIK: Jurnal Ilmiah Penelitian Teknologi Informasi & Komputer **ISSN XXXX-XXXX**
Vol. 1, No. 1, Juni 2020, Hal. 28-34
Available Online at journal.upgris.ac.id/index.php/jipetik

Analisa Improvisasi Algoritma RSA Menggunakan RNG LCG pada Instant Messaging Berbasis Socket TCP

Sofyan Arifianto[1], Aminudin[2], Muhammad Furqon Sidiq[3]
[1,2,3]*Jurusan Teknik Informatika, Universitas Muhammadiyah Malang, Malang, Indonesia*
e-mail: sofyan_arifianto@umm.ac.id[*1],aminudin2008@gmail.com[*2],
furqonsidiq10@gmail.com[3]

Abstrak*— *Socket TCP adalah abstraksi yang digunakan aplikasi untuk mengirim dan menerima data melalui koneksi antar dua host dalam jaringan komputer. Jaringan yang biasa kita gunakan bersifat publik yang sangat rentan akan penyadapan data. Masalah ini dapat teratasi dengan menggunakan algoritma kriptografi pada socket TCP, salah satunya menggunakan algoritma RSA. Tingkat keamanan algoritma RSA standar memiliki celah keamanan pada kunci public ataupun privat yang berasal dari inputan 2 bilangan prima saat pembangkitan kunci. Beberapa penelitian telah dilakukan untuk mengembangkan algoritma RSA, namun hasil dari penelitian tersebut membuat performa dari algoritma RSA menjadi lebih lambat. Peningkatkan performa dapat menggunakan RNG LCG pada pembangkitankunci RSA. RNG LCG memiliki kelebihan yang utama pada segi kecepatannya. RNG LCG dapat menghasilkan bilangan prima yang berasal dari inputan nama yang tidak ditemukan pada RNG lainnya. Hasil pengujian performa waktu pembangkitan kunci, enkripsi, dekripsi dengan panjang karakter mulai dari 40 hingga 81920 menunjukkan bahwa algoritma improvisasi RSA menggunakan RNG LCG lebih baik dibandingkan algoritma RSA. Pengujian kemanan menggunakan known plaintext attack dan fermat factorization menunjukkan bahwa algoritma improvisasi RSA menggunakan RNG LCG lebih baik dibandingkan algoritma RSA.

***Kata Kunci*— *algoritma RSA, improvisasialgoritma RSA, RNG LCG, instant messaging*.**

Abstrak*— *TCP sockets are abstractions that applications use to send and receive data through connections between two hosts in a computer network. The networks that we usually use are public and are very vulnerable to data tapping. This problem can be overcome by using a cryptographic algorithm on the TCP socket, one of which uses the RSA algorithm. The level of security of the standard RSA algorithm has security holes in public or private keys originating from the input of 2 primes during key generation. Several studies have been conducted to develop the RSA algorithm, but the results of these studies make the performance of the RSA algorithm slower. Improved performance can use LCG RNG on RSA key generation. RNG LCG has the main advantage in terms of speed. LCG RNG can generate prime numbers that come from input names that are not found in other RNGs. The results of performance tests for key generation, encryption, decryption with character lengths ranging from 40 to 81920 indicate that the RSA improvisation algorithm using the LCG RNG is better than the RSA algorithm. Security testing using known plaintext attacks and fermat factorization shows that the RSA improvisation algorithm using the LCG RNG is better than the RSA algorithm.

Keywords: RSA algorithm, improvised RSA algorithm, RNG LCG, instant messaging

PENDAHULUAN

Socket TCP adalah abstraksi yang digunakan aplikasi untuk mengirim dan menerima data melalui koneksi antar dua host dalam jaringan komputer. Programmer dapat menggunakan koneksi jaringan tersebut untuk menentukan lalu lintas data untuk ditulis dan dibaca[1]. Jaringan yang biasa kita gunakan bersifat publik yang sangatrentanakanpenyadapan data [2]. Masalahinidapatteratasidenganmenggunakanalgoritmakriptografi pada socket TCP, salah satunya menggunakan algoritma RSA. Algoritma RSA memiliki tingkat keamanan yang berfokus pada sulitnya faktorisasi bilangan besar pada nilai N menjadi 2 bilangan prima (p dan q) [3].

Tingkat keamanan algoritma RSA standar memiliki celah keamanan pada kunci public atau pun privat yang berasal dari inputan 2 bilangan prima saat pembangkitan kunci[4]. Bilangan prima bernilai kecil menjadi celah keamanan pada algoritma RSA, sehingga perlu adanya pengembangan dari bilangan prima tersebut. Bilangan prima bernilai besar akan membuat algoritma RSA lebih aman. Selain itu, bisa juga dengan menambahkan jumlah bilangan prima sebagai variable dalam pembangkitan kunci publik dan privat, sebagai mana penelitian yang telahdilakukan oleh N. Somani dan D. Mangal (2014) yang melakukan improvisasi algoritma RSA dengan menggunakan 3 bilangan prima [5]. Penambahan bilangan prima yang lebih dari 2 akan mengurangi kecepatan baik dari pembangkitan kunci, enkripsi maupun pada dekripsi sebagaimana yang telah dilakukan oleh R. Saputra (2018). Pembangkitan kunci yang dilakukan oleh R. Saputra (2018) menggunakan Miller-Rabin tetapi pembangkitan kuncinya menghasilkan bilangan prima dengan waktu yang lama, untuk itu peningkatkan performa dapat menggunakan RNG pada pembangkitan kunci RSA.

Random Number Generator (RNG) sangat membantu dalam mengacak angka yang akan digunakan dalam algoritma RSA. Linear Congruential Generator (LCG) merupakan salah satu metode pengacakan bilangan berdasarkan linier [6]. LCG memiliki kelebihan yang utama

pada segi kecepatannya. LCG dapat menghasilkan bilangan prima yang berasal dari inputan nama yang tidak ditemukan pada RNG lainnya. LCG memiliki peran yang tak kalah penting yaitu dapat menghasilkan bilangan prima yang besar. RSA yang dikombinasikan dengan LCG diharapkan dapat lebih efisien dalam segi kecepatan.

Perbedaan penelitian ini dengan RSA standar terletak pada RNG LCG untuk mendapatkan bilangan prima. Penelitian ini mengubah inputan dari parameter M pada rumus RNG LCG dimana penelitian yang dilakukan oleh M. Khairani (2017) memiliki kelemahan yaitu bilangan prima yang dihasilkan kecil dan terbatas. Penelitian ini dilakukan perubahan inputan dari segi parameter sehingga akan menghasilkan bilangan prima yang besar dan banyak[6].

Penelitian yang dilakukan oleh R. Saputra (2018) menambahkan bilangan prima menjadi 6 bilangan prima namun performa yang dihasilkan jauh lebih lambat dibandingkan dengan RSA standar yang hanya menggunakan 2 bilangan prima [7]. Pembangkitan kunci pada penelitian R.S. Dhakardkk (2012) yang menggunakan 4 bilangan prima dan hasil penelitian tersebut menunjukkan performa yang lebih lambat dari RSA standar[8]. Penelitian yang dilakukan oleh Thangavel, M dkk (2014) menggunakan 4 bilangan prima juga menghasilkan performa yang lebih lambatdari RSA standar walau menggunakan metode pembangkitan kunci yang berbeda[9].

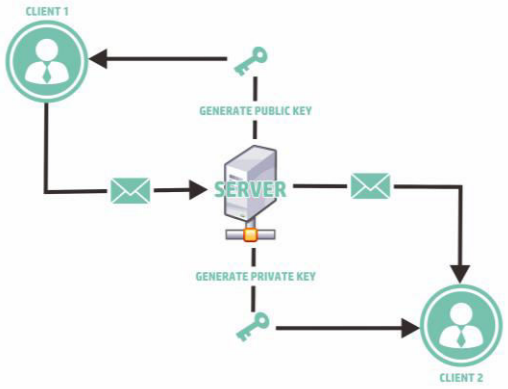
Banyaknya bilangan prima yang digunakan pada penelitian sebelumnya, membuat performa RSA menjadi menurun maka pada penelitian ini hanya menggunakan 2 bilangan prima saja. Perbedaan penelitian ini dengan beberapa penelitian sebelumnya adalah pada proses pembangkitan kunci yang akan diimprovisasi pada rumus $\Phi(N)$ di penelitian yang dilakukan oleh M. Bunder dkk (2018) mampu menghasilkan kunci publik dan kunci privat dengan nilai yang besar walau hanya menggunakan 2 bilangan prima [10]. Perubahan tersebut diharapkan mampu meningkatkan performa dan keamanan dari penelitian ini.

Penelitian ini bertujuan untuk menganalisis perbandingan performa waktu proses dari awal hingga akhir antara algoritma RSA standar dengan improvisasi algoritma RSA menggunakan RNG LCG, serta metode known plaintext attack dan fermat factorization digunakan untuk melakukan pengujian terhadap perbandingan tingkat keamanan antara algoritma RSA standar dengan improvisasi algoritma RSA menggunakan RNG LCG. Penelitian ini diharapkan mampu meningkatkan keamanan dan mempercepat proses enkripsi dan dekripsi algoritma RSA.

I. METODE PENELITIAN

A. Rancangan Aplikasi Chat

Algoritma RSA dan improvisasi algoritma RSA akan diterapkan pada aplikasi chat menggunakan jaringan local. Gambar 1 terdapat dua orang client yang akan saling berkomunikasi. Terdapat satu buah server yang berfungsi untuk menghubungkan kedua client tersebut agar dapat mengirim dan menerima pesan. Server juga berfungsi untuk memastikan pesan yang dikirim aman dengan menerapkan algoritma RSA dan improvisasi algoritma RSA.



Gambar 1 Gambaran aplikasi chat

Alur dari cara kerja aplikasi sebagai berikut :

1. Client 1 dan 2 memasukan IP Address server agar bias terhubung dengan server.
2. Client 1 dan 2 mendapat kunci dari server
3. Client 1 mengirim pesan yang terenkripsi menggunakan kunci publik yang dimiliki Client 2.
4. Client 2 menerima pesan dari Client 1 dan mendekripsinya menggunakan kunci privatnya.

B. Algoritma RSA

Algoritma RSA termasuk ke dalam algoritma kriptografi kunci asimetris yang memiliki dua buah kunci, kunci publik dan kunci privat. Algoritma ini sangat umum digunakan dalam proses data enkripsi maupun aplikasi digital signature. Bilangan prima menjadi penentu tingkat kesulitan pada algoritma RSA. Bilangan prima tersebut didapatkan dengan cara dilakukan pemfaktoran untuk mendapatkan kunci privat yang digunakan untuk memecahkan ciphertext.

Algoritma RSA ini memiliki 3 tahapan di antaranya pembangkitan kunci, proses enkripsi, proses dekripsi. Berikut ini penjelasan tiap tahapannya:

1. Membangkitkan bilangan prima p dan q bernilai acak
2. Menghitung nilai N = p ∗ q
3. Menghitung nilai $\Phi(N) = (p - 1) * (q - 1)$
4. Menentukan nilai e dengan rumus gcd $(e,\Phi(N)) = 1$
5. Mencari nilai d yang sesuai dengan $e*d \equiv 1 \bmod \Phi(N)$
6. Hasil dari pembangkitan kunci yaitu kunci publik (e,N) dan kunci privat (d,N). Proses pembangkitan kunci dapat dilihat pada gambar 2.

Input: panjang bit bilangan prima (α), panjang bit eksponen enkripsi (β)
Output: kunci publik dan kunci privat
1. Bangkitkan dua bilangan prima berukuran α bit (p dan q)
2. $N = p \times q$
3. $\Phi(N) = (p - 1) \times (q - 1)$
4. Pilih bilangan bulat e berukuran β bit dengan gcd(e, $\Phi(N)$) = 1, $1 < e < \Phi(N)$
5. $d = e^{-1} \bmod \Phi(N)$

Gambar 2 Pseucode pembangkitan kunci algoritma RSA

7. Menghitung ciphertext untuk mengenkripsi plaintext dengan rumus1.

$$C = M^e \bmod N \qquad (1)$$

Pseucode enkripsi dapat dilihat pada gambar 3.

Input: Kunci publik = (e, N), plain text (M)
Output: *ciphertext* (C)
1. for i ← 0 to panjang *plaintext* − 1 do
2. Konversi Mi ke nilai desimal
3. Ci ← (Mi)e mod N
4. end for
5. Return C

Gambar 3 Pseucode enkripsi algoritma RSA

8. Menghitung nilai Muntukmendekripsi ciphertext,

$$M = C^d \bmod N \qquad (2)$$

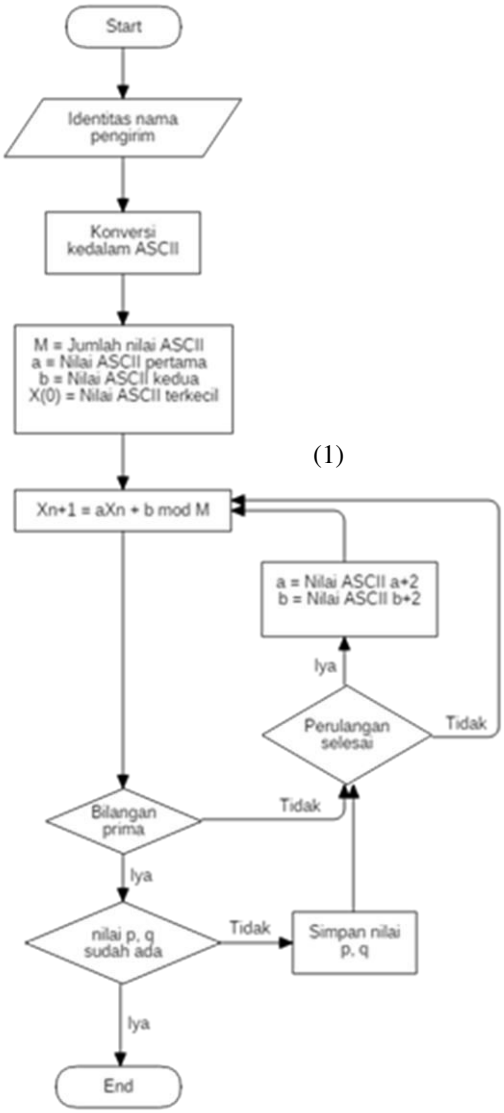
Pseucode dekripsi algoritma RSA dapat dilihat pada gambar 4.

Input: Kunci privat = (d, N), *ciphertext* (C)
Output: *plaintext* (M)
1. for i ← 0 to panjang *Ciphertext* − 1 do
2. Mi ← (Ci)d mod N
3. Konversi nilai Mi ke karakter
4. end for
5. Return M

Gambar 4 Pseucode dekripsi algoritma RSA

C. *RNG LCG*

Rancangan RNG LCG menggunakan rancangan yang terdapat pada penelitian yang dilakukan oleh Mufidah Khairani (2017). Perubahan dilakukan pada isi dari variabel M yang semula didapat dari nilai ASCII terbesar, namum pada penelitian ini variabel M diubah dengan total penjumlahan nilai ASCII dari nama pengirim. Improvisasi dilakukan karena dari analisa hasil yang didapat pada penelitian sebelumnya nilai M yang kecil menghasilkan bilangan *output* yang kecil juga. Nilai M pada penelitian ini ditingkatkan jumlahnya yang diharapkan akan meningkatkan hasil dari *output* yang dihasilkan. Rancangan RNG LCG sebagai mana gambar 5.



Gambar 5 Flowchart RNG LCG

D. *Algoritma RSA improvisasi*

Perbedaan antara algoritma RSA dengan RSA improvisasi terletak pada pembangkitan kunci. RSA improvisasi mendapatkan bilangan prima p dan q dari RNG LCG. Perbedaan lain nya terdapat improvisasi pada rumus $\Phi(N)$. *Pseudocode* dari proses pembangkitan kunci algoritma improvisasi algoritma RSA menggunakan RNG LCG sebagai berikut:

Input: p, q, panjang bit eksponen enkripsi (β)
Output: Kunci publik, Kunci privat
1. Input 2 bilangan prima p dan q dari proses RNG LCG
2. $N = p \times q$
3. $\Phi(N) = (p^2 - 1) \times (q^2 - 1)$
4. Pilih bilangan bulat e berukuran β bit dengan gcd(e, Φ(N)) = 1, 1 < e < Φ(N)
5. $d = e^{-1} \bmod \Phi(N)$
6. Kpublik = (e, N), Kprivat = (d, N)

Gambar 6 Pembangkitan kunci improvisasi algoritma RSA

E. Known Plaintext Attack

Attacker mengetahui kunci publik serta pesan yang terenkripsi. Selanjutnya *attacker* akan membentuk baris himpunan ASCII antara *plaintext* (P) dan *ciphertext* (C). *Attacker* mencocokan antara *ciphertext* dengan *plaintext* apakah terdapat *plaintext* yang berkorespondensi. Attacker akan menyimpan *plaintext* jika terdapat *plaintext* yang berkorespondensi.misal Dayat memiliki kunci publik (130931,736255235558669442766648479002934564 17) yang dibagikan ke Opang dan Yusuf. Yusuf kemudian mengenkripsi himpunan ASCII desimal ke a-z (97 - 122) dengan kunci publik Dayat maka Yusuf memiliki himpunan data sebagai berikut:

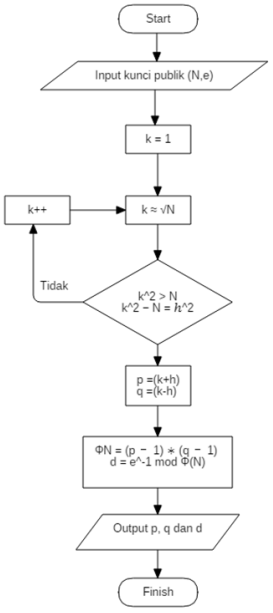
Tabel 1Contoh himpunan plaintext(P) dan ciphertext(C)

P	C	P	C	P	C
97	20428	106	21175	115	66588
98	99834	107	67021	116	51266
99	130615	108	41582	117	21074
100	60004	109	10647	118	59861
101	100743	110	54014	119	43787
102	114041	111	100756	120	18997
103	246	112	46134	121	28233
104	87120	113	73443	122	58327
105	121279	114	89167		

Kemudian Opang menyadap pesan yang dikirimkan oleh Dayat dan Yusuf dengan mencocokan nilai *ciphertext* (C) yang dikirimkan dengan Tabel 3.1. Misal pesan yang dikirimkan dalam bentuk *ciphertext* adalah 87120 20428 121279. Maka dapat dicocokan dengan Tabel 3.1 dimana 87120 = 104 (huruf "h"), 20428 = 97 (huruf "a") dan 121279 = 105 (huruf "i") menghasilkan *plaintext* "hai".

F. Fermat Factorization

Seorang *attacker*apabilamengetahui faktor dari nilai N pada kuncimaka dia akan mengetahui nilai eksponen d yang terdapat pada kunci privat (N,d) yang didapat dari nilai kunci publik (N,e). Pesan yang telah dienkripsi akan dengan mudah didekripsi oleh *attacker* tersebut. Alur kerjadarifermat factorization dapatdilihat pada gambar 7.



Gambar 7 Alur fermat factorization

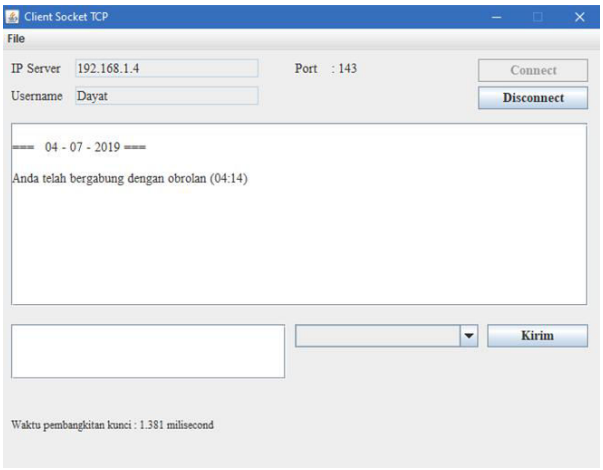
Fermat factorization memulai proses dengan menginputkan kunci publik. Mencari nilai k yang mendekati hasil dari $\sqrt{N}$ dengan syarat nilai $k^2 > N$ dan nilai $h^2 = k^2 - N$. Jika tidak memenuhi syarat tersebut maka nilai k akan dilakukan penambahan hingga kondisinya terpenuhi. Nilai p dan q akan didapatkan jika kondisi di atas terpenuhi, sedangkan untuk nilai d dapat dihitung menggunakan rumus yang terdapat pada langkahkelima pada gambar 2. Sistem akan menampilkan bilangan prima (p dan q) dan kunci privat (d). Jika sudah di dapat ketiga poin tersebut, *attacker* akan dengan mudah mengetahui *plaintext* ataupun *ciphertext* yang dikirim oleh *user*.

II. HASIL DAN PEMBAHASAN

Hasil dari penelitian ini ada beberapa diantaranya implemtasi kedua algoritma pada aplikasi chat, waktu yang dibutuhkan kedua algoritma dalam melakukan proses pembangkitan kunci, enkripsi dan dekripsi, tingkat keamanan dari kedua algoritma dengan menguji dengan serangan known plaintext attack dan fermat factorization.

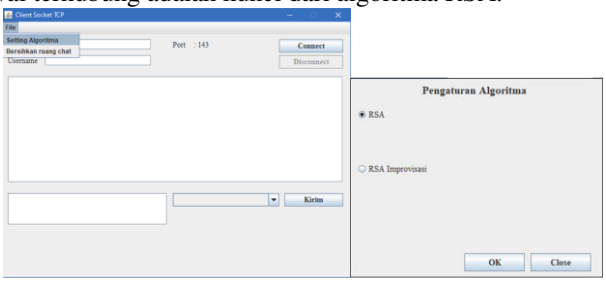
A. Implementasi Aplikasi Chat

Aplikasi yang dibangun meliputi aplikasi pada *client* dan pada server. Aplikasi pada sisi*client*, pesan yang akan dikirim akan dilakukan proses enkripsi terlebih dahulu dengan algoritma RSA atau algoritma RSA improvisasi. Aplikasi pada sisi server akan menampilkan aktivitas yang dilakukan *client* seperti pembangkitan kunci, pesan yang terenkripsi, dll.



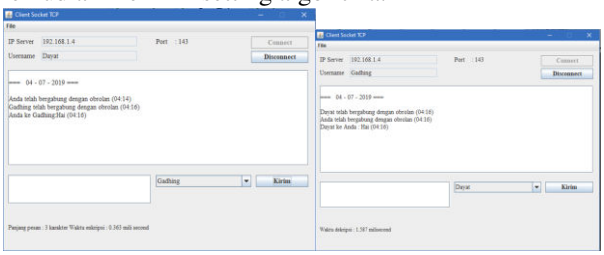
Gambar 8 Tampilan client setelah terhubung

User akan terhubung jika sudah memasukan ip server dan juga username lalu menekan tombol Connect. Aplikasi akan menampilkan pesan jika user telah terhubung dengan jaringan lokal. Aplikasi juga menampilkan waktu pembangkitan kunci. Algoritma asal yang digunakan adalah algoritma RSA sehingga waktu pembangkitan kunci saat awal terhubung adalah kunci dari algoritma RSA.



Gambar 9 Tampilan task bar file dan pengaturan algoritma

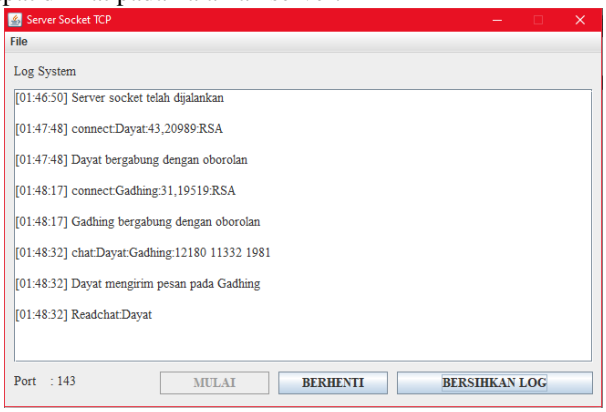
User dapat memilih algoritma yang akan digunakan untuk melakukan enkripsi pada pesan yang akan dikirim dengan menekan task bar "file" sebagaimana pada gambar 9 kemudian memilih setting algoritma.



A B

Gambar 10 Tampilan pesan yang terkirim (A) dan diterima (B)

Pesan yang dikirim akan ditampilkan pada halaman user pengirim dan penerima. Panjang pesan dan waktu enkripsi akan ditampilkan pada halaman pengirim sebagaimana pada gambar 10A. Halaman penerima akan menampilkan pesan yang sudah didekripsi dan juga waktu dekripsi sebagaimana pada gambar 10B. Semua pesan yang dikirim dapat dilihat pada halaman server.



Gambar 11Tampilan server

Kita dapat lihat pada log server terdapat aktivitas yang dilakukan *client* mulai dari pembangkitan kunci, memilih algoritma yang digunakan untuk mengenkripsi dan mendekripsi pesan, menampilkan pesan yang telah ternkripsi yang dikirimkan antar user. Pembangkitan kunci dapat dilihat pada bagian "Connect:Dayat 43,20989:RSA merupakan" maksudnya adalah Dayat merupakan username dari *client* 1, 43,20989 merupakan kunci public dari *client* 1 dan RSA merupakan algoritma yang digunakan untuk mengenkripsi dan mendekripsi pesan. Pesan yang terenkripsi dalam bentuk angka, dapat dilihat pada log pesan yang sudah terenkripsi adalah 12180 11332 1981.

B. Pengujian Waktu Pembangkitan Kunci

Pengujian waktu pembangkitan kunci dilakukan sebanyak 12 kali dengan parameter inputan panjang karakter yang berbeda. Hasil pengujian dapat dilihat pada Tabel 2.

Tabel 2 Perbandingan waktu pembangkitan kunci

No	Panjang karakter (karakter)	Algoritma RSA Standar (ms)	Improvisasialgoritma RSA menggunakan RNG LCG (ms)
1	40	7,094	0,405
2	80	4,824	0,472
3	160	1,840	0,390
4	320	1,555	0,262
5	640	2,361	0,265
6	1280	1,200	0,224
7	2560	1,475	0,204
8	5120	1,052	0,295
9	10240	5,080	0,280
10	20480	1,948	0,222
11	40960	1,458	0,205
12	81920	1,714	0,359
Rata-rata		2,633	3,580

Hasil dari pengujian waktu pembangkitan kunci pada Tabel 2 menunjukan hasil yang tidak terlalu banyak perubahan. Hal tersebut membuktikan bahwa pada pembangkitan kunci, panjang karakter tidak berpengaruh

pada kecepatan pembangkitan kunci. Panjang bit bilangan prima pada algoritma RSA dan bilangan prima yang dihasilkan oleh RNG LCG pada improvisasi algoritma RSA mempengaruhi hasil dari kecepatan waktu pembangkitan kunci. Hasil rata – rata waktu pembangkitan kunci pada Tabel 2 dapat disimpulkan bahwa improvisasi algoritma RSA menggunakan RNG LCG memiliki performa lebih baik dibandingkan algoritma RSA standar pada proses pembangkitan kunci.

C. Pengujian Waktu Enkripsi

Pengujian waktu enkripsi dilakukan sebanyak 12 kali dengan parameter inputan panjang karakter yang berbeda. Skenario pengujiannya yaitu user 1 mengirim pesan kemudian waktu enkripsi akan tampil sebagaimana pada gambar 11. Hasil pengujian dapat dilihat pada Tabel 3.

Tabel 3 Perbandingan waktu enkripsi

No	Panjang karakter (karakter)	Algoritma RSA Standar (ms)	Improvisasialgoritma RSA menggunakan RNG LCG (ms)
1	40	4,069	2,031
2	80	7,387	4,198
3	160	9,288	7,598
4	320	11,699	9,503
5	640	22,824	9,350
6	1280	39,680	25,984
7	2560	76,824	47,966
8	5120	268,520	151,595
9	10240	893,258	771,605
10	20480	2018,575	1881,650
11	40960	7086,568	7197,305
12	81920	29224,275	29478,118
Rata-rata		3305,247	3298,909

Pengujian waktu enkripsi pada Tabel 3 menunjukan hasil semakin panjang karakter maka semakin lama waktu yang dibutuhkan RSA standar dan improvisasi algoritma RSA. Hal yang memengaruhi hasil waktu enkripsi adalah panjang karakter pada pesan yang dikirim, jika semakin panjang karakternya maka akan semakin lama waktu enkripsinya. Jenis karakter seperti simbol juga mempengaruhi waktu enkripsi, misal tanda tanya, tanda seru, dan lain-lain. Hasil rata-rata waktu enkripsi pada Tabel 3 dapat disimpulkan bahwa improvisasi algoritma RSA menggunakan RNG LCG memiliki performa lebih baik dibandingkan algoritma RSA standar pada proses enkripsi.

D. Pengujian Waktu Dekripsi

Pengujian waktu dekripsi dilakukan sebanyak 12 kali dengan parameter inputan panjang karakter yang berbeda. Skenario pengujiannya yaitu user 2 menerima pesan kemudian waktu dekripsi akan tampil sebagaimana pada gambar 12. Hasil pengujian dapat dilihat pada Tabel 4.

Tabel 4 Perbandingan waktu dekripsi

No	Panjang karakter (karakter)	Algoritma RSA Standar (ms)	Improvisasialgoritma RSA menggunakan RNG LCG (ms)
1	40	10,773	9,965
2	80	15,247	10,797
3	160	23,131	15,676
4	320	28,301	20,755
5	640	27,668	18,217
6	1280	39,608	30,029
7	2560	97,039	50,596
8	5120	298,791	92,458
9	10240	598,487	312,207
10	20480	865,068	696,668
11	40960	1790,928	933,997
12	81920	2423,171	1746,716
Rata-rata		3305,247	518,184

Hasil dari pengujian waktu dekripsi pada Tabel 4 menunjukan hasil waktu yang mirip dengan enkripsi yaitu semakin panjang karakter pada pesan maka semakin lama proses dekripsinya. Hal yang mempengaruhi hasil waktu dekripsi adalah panjang karakter dan juga jenis karakter dari pesan yang dikirim. Panjang karakter sangat berpengaruh, jika semakin panjang karakternya maka akan semakin lama waktu dekripsinya. Jenis karakter seperti simbol juga mempengaruhi waktu dekripsi, misal tanda tanya, tanda seru, dan lain-lain. Hasil rata-rata waktu dekripsi pada Tabel 4 dapat disimpulkan bahwa improvisasi algoritma RSA menggunakan RNG LCG memiliki performa lebih baik dibandingkan algoritma RSA standar pada proses dekripsi.

E. Pengujian Known Plaintext Attack

Pengujian *known plaintext attack* pada kedua algoritma diawali dengan mendapatkan *ciphertext* dari server. Skenario pengujiannya adalah *attacker* mendapatkan kunci publik dan ciphertext yang ada pada server dengan sniffing. *Ciphertext* dan kunci public kemudian akan digunakan untuk memecahkan plaintext atau pesan dari *client*. Pengujian *known plaintext attack* menggunakan parameter panjang ciphertext, waktu eksekusi dan persentase keberhasilan. Panjang ciphertext yang digunakan adalah 50, 100 dan 160.Adapun hasilnya dapat dilihat pada tabel 5.

Tabel 5 Pengujian known plaintext attack

No	Panjang karakter (karakter)	Waktu eksekusi (ms)		Persentasekeberhasilan (%)	
		RSA standar	Improvisasi RSA	RSA standar	Improvisasi RSA
1	50	23,447	26,393	100	100
2	100	29,165	43,278	100	100
3	160	31,098	58,264	100	100
Rata – rata		27,903	42,645	100	100

Kesimpulan pengujian known plaintext attack pada Tabel 5 adalahalgoritma RSA dan improvisasi algoritma RSA menggunakan RNG LCG dapat diserang dengan metode ini sebagaimana pada kolom persentase keberhasilan menunjukan 100%. Berdasarkan hasil rata-rata waktu eksekusi, algoritma RSA standar lebih cepat diserang dibandingkan improvisasi algoritma RSA menggunakan RNG LCG. Kesimpulannya improvisasi algoritma RSA menggunakan RNG LCG lebih baik dari algoritma RSA standar.

F. Pengujian Fermat Factorization

Pengujian*fermat factorization* dilakukan untuk menentukan hasil faktorisasi dan mendapatkan kunci privat dari kedua algoritma. Skenario pengujian sedikit berbeda dari pengujian *known plaintext attack* yaitu pada pengujian ini hanya membutuh kunci publik. Kunci public terdiri dari nilai e dan N yang akan digunakan untuk mendapatkan nilai p, q dan d. Adapun hasilnya dapat dilihat pada tabel 6.

Tabel 6 Perbandingan waktu dekripsi

Algoritma	Waktu eksekusi (ms)	Status kunciprivat	Keterangan
RSA Standar	0.492	Ditemukan	-
Improvisasi RSA	1.506	Ditemukan	Ditemukannilai p dan q yang sesuainamunkunciprivattidaksesuai

Kesimpulan pengujian *fermat factorization* pada Tabel 6 adalah algoritma RSA dan improvisasi algoritma RSA menggunakan RNG LCG masih dapat diserang dengan metode ini. Hal ini disebabkan oleh kecilnya bilangan prima yang dihasilkan. Nilai kunci privat yang ditemukan pada improvisasi algoritma RSA menggunakan RNG LCG tidak sesuai dengan kunci privat yang sebenarnya. Perhitungan $\Phi(N)$ pada algoritma improvisasi sudah dimodifikasi sehingga nilai dari kunci privatnya tidak sesuai.

Alasan lain algoritma dapat diserang oleh metode ini adalah keduanya memiliki nilai N sebagai nilai yang digunakan untuk difaktorkan. Nilai N tersebut dihasilkan dari nilai p dan q yang merupakan nilai awal dari kedua algoritma ini. Nilai N yang relatif kecil menjadi sebab mudahnya kedua algoritma diserang dengan metode ini. kesimpulannya bahwa kedua algoritma tersebut masih dapat diserang dengan menggunakan metode *fermat factorization*.

III. KESIMPULAN

Penelitian yang sudah dilakukan dapat ditarik kesimpulan diantaranya algoritma RSA dan improvisasi algoritma RSA menggunakan RNG LCG dapat diimplementasikan pada aplikasi instant messaging socket TCP, improvisasialgoritma RSA menggunakan RNG LCG memiliki performa waktu yang lebih baik disbanding algoritma RSA standar pada proses pembangkitan kunci, enkripsi dan dekripsi, improvisasi algoritma RSA menggunakan RNG LCG lebih aman disbanding algoritma RSA dari serangan known plaintext attack serangan menggunakan metode fermat factorization

DAFTAR PUSTAKA

[1] E. R. Harold, *Java Network Programming*, 4th ed. Sebastopol, CA: O'Reilly Media, Inc, 2013.

[2] M. J. Donahoo and K. L. Calvert, *TCP/IP Sockets in C Practical Guide for Programmers*. Burlington: Morgan Kaufmann publications, 2009.

[3] A. Arief and R. Saputra, "Implementasi Kriptografi Kunci Publik dengan Algoritma RSA-CRT pada Aplikasi Instant Messaging," *Sci. J. Informatics*, vol. 3, no. 1, pp. 46–54, 2017.

[4] A. Fatima and R. R. Chaudhary, "Modified Trial Division Algorithm Using Lagrange ' s Interpolation Function to Factorize RSA Public Key Encryption," no. 3, pp. 1861–1865, 2017.

[5] N. Somani and D. Mangal, "An Improved RSA Cryptographic System," *Int. J. Comput. Appl.*, vol. 105, no. 16, pp. 975–8887, 2014.

[6] A. Khairan, M. Imrona, and I. Ummah, "Analisis Dan Implementasi Kriptografi Rsa Pada Aplikasi Chatting Client-Server Based," pp. 1–7, 2014.

[7] R. . Saputra, "Analisa Improvisasi Algoritma RSA Berdasarkan Dari Jumlah Penggunaan Bilangan Prima Pada Instant Messaging Berbasis Socket TCP," Malang, 2018.

[8] R. S. Dhakar, "Modified RSA Encryption Algorithm (MREA)," pp. 2–5, 2012.

[9] M. Thangavel, P. Varalakshmi, M. Murrali, and K. Nithya, "ScienceDirect An Enhanced and Secured RSA Key Generation Scheme (ESRKGS)," *J. Inf. Secur. Appl.*, pp. 1–8, 2014.

[10] M. Bunder, A. Nitaj, W. Susilo, and J. Tonien, "Cryptanalysis of RSA-type cryptosystems based on Lucas sequences, Gaussian integers and elliptic curves," *J. Inf. Secur. Appl.*, vol. 40, pp. 193–198, 2018.