REPOSITOR, Vol. 2, No. 11, November 2020, Pp. 1444-1455
ISSN : 2714-7975
E-ISSN : 2716-1382

Improvisasi Algoritma RSA Menggunakan Generate Key ESRKGS pada Instant Messaging Berbasis Socket TCP

Aminudin[1], Gadhing Putra Aditya[2], Sofyan Arifianto[3]

[1,2,3] Jurusan Teknik Informatika, Universitas Muhammadiyah Malang, Malang, Indonesia
e-mail: aminudin2008@umm.ac.id[*1], gadhingputra@gmail.com[2]

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 publik ataupun privat yang berasal dari inputan 2 bilangan prima saat pembangkitan kunci, begitupun dengan algoritma improvisasi RSA meskipun menggunakan 4 bilangan prima akan tetapi mulai dari pembangkitan kunci hingga dekripsi memiliki proses yang sama persis dengan RSA standar sehingga tingkat keamanan dari kedua algoritma tersebut sama – sama kurang aman meskipun jumlah bilangan prima dari algoritma improvisasi RSA lebih banyak dari RSA standar. Peningkatan keamanan dapat dilakukan dengan memodifikasi algoritma RSA dengan menggunakan ESRKGS (Enhanced and Secured RSA Key Generation Scheme). ESRKGS RSA memiliki kelebihan yang utama pada segi keamanannya. ESRKGS RSA secara total memodifikasi algoritma RSA terutama pada bagian pembangkitan kunci dan diklaim mempunyai performa lebih cepat dari algoritma improvisasi RSA yang sama – sama menggunakan 4 bilangan prima dan tentunya lebih aman dari serangan known plaintext attack dan fermat factorization attack yang akan penulis gunakan untuk pengujian keamanan pada penelitian ini. Hasil pengujian performa waktu pembangkitan kunci dengan panjang bit 256 bit, 512 bit, dan 1024 bit serta untuk proses enkripsi dan dekripsi panjang karakter yang digunakan adalah 100, 250, dan 400 menunjukkan bahwa algoritma ESRKGS RSA lebih baik dibandingkan algoritma improvisasi RSA. Pengujian kemanan menggunakan known plaintext attack dan fermat factorization attack menunjukkan bahwa algoritma ESRKGS RSA lebih baik dibandingkan algoritma RSA standar dan improvisasi RSA.

Kata kunci: *RSA, improvisasi RSA, ESRKGS RSA, instant messaging*

Abstract

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 security level of the standard RSA algorithm has security gaps on public or private keys originating from the input of 2 primes during key generation, as well as the RSA improvisation algorithm even though using 4 prime numbers but starting from generating key to decryption has the exact same process as the standard RSA so the security level of the two algorithms is equally less safe even though the number of prime numbers of the RSA improvisation algorithm is more than the standard RSA. Improved security can be done by modifying the RSA algorithm by using ESRKGS (Enhanced and Secured RSA Key Generation Scheme). RSA ESRKGS has the main advantages in terms of safety. ESRKGS RSA totally modified the RSA algorithm, especially in the key generation section and claimed to have faster performance than the RSA improvisation algorithm that both use 4 prime numbers and is certainly safer from known plaintext attacks and fermat factorization attacks that the authors will use for security testing. in this research. The results of the key generation time performance test with 256 bit length, 512 bit, and 1024 bit and for the encryption and decryption process the length of characters used is 100, 250, and 400 shows that the RSA ESRKGS algorithm is better than the RSA improvisation algorithm. Security testing using known plaintext attacks and fermat factorization attacks shows that the RSA ESRKGS algorithm is better than the standard RSA algorithm and RSA improvisation.

Keywords*: RSA, RSA improvisation, RSA ESRKGS, instant messaging*

1. Pendahuluan

Algoritma RSA termasuk algoritma penyandian dengan kunci asimetrik atau algoritma penyandian kunci publik yang didasarkan pada teori pertukaran kunci Diffie-Hellman [1]. Algoritma RSA dibagi menjadi tiga proses penyandian yaitu proses pembangkitan kunci, proses enkripsi dan proses dekripsi. Algoritma RSA memiliki tingkat keamanan yang berfokus pada sulitnya faktorisasi bilangan bernilai besar pada nilai n menjadi dua bilangan prima (p dan q) [2].

Tingkat keamanan algoritma RSA standar memiliki celah keamanan pada kunci publik ataupun privat yang berasal dari inputan dua bilangan prima saat pembangkitan kunci [3]. Operasi matematika didalam algoritma RSA terdiri dari perkalian dua buah bilangan prima (p dan q) yang dipilih secara acak, hasil dari perkalian tersebut menghasilkan nilai n. Semakin besar ukuran nilai n akan semakin bagus tingkat keamanannya karena dapat menyulitkan penyerang untuk memecahkan nilai n dengan faktorisasi [4]. Bilangan prima bernilai besar akan membuat algoritma RSA lebih aman. Selain itu bisa juga dengan menambahkan jumlah bilangan prima sebagai variabel dalam pembangkitan kunci publik dan kunci privat dengan menggunakan 3 buah bilangan prima [5], akan tetapi penambahan bilangan prima masih terbilang kurang aman karena tetap dapat dipecahkan dengan penyerangan faktorisasi. Oleh karena itu perlu adanya peningkatan keamanan dan performa algoritma RSA dengan menggunakan metode pembangkitan kunci ESRKGS.

Metode ESRKGS yang akan penulis terapkan mengacu pada penelitian M. Thangavel, P. Varalakshmi, Mukund Murrali, K. Nithya (2015) dimana metode ESRKGS RSA memiliki performa yang lebih baik dalam proses pebangkitan kunci, enkripsi dan dekripsi dibanding dengan metode improvisasi RSA yang digunakan dalam penelitian R.D. Saputra (2018). Selain itu dari segi keamanan ESRKGS RSA diklaim lebih aman dibanding RSA standar dan improvisasi RSA sehingga membutuhkan waktu penyerangan yang lama untuk memecahkan sistem [6]. Penerapan ESRKGS RSA salah satunya dapat diterapkan pada *instant messaging*. *Instant messaging* adalah sebuah aplikasi komunikasi secara *real time* yang memungkinkan dua orang atau lebih untuk saling berkomunikasi dan bertukar pesan. Pesan yang telah ditulis oleh pengirim akan dikirim kepada penerima setelah pengirim menekan tombol untuk mengirim pesan [7]. Dalam *instant messaging* perlu adanya media penghubung antara pengirim dan penerima salah satunya menggunakan socket TCP.

Socket TCP adalah protokol yang digunakan untuk penerimaan dan pengiriman data melalui koneksi antar dua host atau lebih yang memungkinkan seorang programmer menggunakan koneksi jaringan komputer untuk menentukan aliran data untuk dibaca maupun ditulis [8]. Socket TCP terdiri dari server TCP dan klien TCP, dimana server TCP akan menyimpan kunci publik dari klien TCP yang terhubung dengan server TCP. Jaringan yang umumnya digunakan bersifat publik yang memungkinkan terjadinya sebuah penyadapan data. Implementasi kriptografi terhadap socket TCP dapat mengatasi masalah tersebut [9]. Penulis mencoba mengatasi masalah – masalah tersebut dengan meningkatkan keamanan *instant messaging* pada socket TCP dengan menggunakan algoritma RSA yang di tingkatkan menggunakan metode ESRKGS *(Enhanced and Secured RSA Key Generation Scheme).*

Perbedaan dengan penelitian sebelumnya yaitu terletak pada metode pembangkitan kunci dimana penelitian sebelumnya menggunakan metode RSA Standar dan Improvisasi RSA sedangkan pada penelitian kali ini penulis menggunakan metode ESRKGS RSA dengan menggunakan 4 buah bilangan prima (p, q, r, dan s) dan 2 buah variabel (n dan m) yang akan menghasilkan nilai N untuk pembangkitan kunci sehingga membutuhkan waktu yang lama untuk melakukan pemecahan sistem dan secara signifikan dianggap mempunyai performa dan keamanan lebih baik dari metode sebelumnya. Penelitian sebelumnya dilakukan oleh D. Dermawan (2014) dan M. Ihwani (2016) yang menerapkan metode RSA Standar dengan pembangkitan kunci menggunakan dua buah bilangan prima (p dan q) dan satu buah variabel (n). Sedangkan pembangkitan kunci pada metode Improvisasi RSA sebagaima penelitian yang telah di lakukan oleh Nikita Somani dan Dharmendra Mangal (2014) yaitu dengan menggunakan tiga buah bilangan prima (p, q, dan s) dan satu buah variabel (n), serta penelitian dari R.D. Saputra (2018) yang melakukan improvisasi algoritma menggunakan empat buah bilangan prima (p, q, r, dan s) dan satu buah variabel (n).

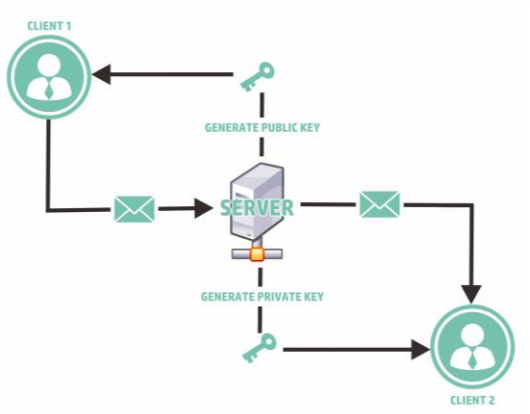
Penelitian ini bertujuan untuk menganalisis perbandingan performa waktu proses dari awal hingga akhir antara algoritma RSA standar, improvisasi RSA, dan ESRKGS RSA, dengan metode penyerangan menggunakan *Known Plaintext Attack* [10] dan *Fermat Factorization Attack* [11] yang bertujuan untuk menguji perbandingan tingkat keamanan antara algoritma RSA

standar, improvisasi RSA, dan ESRKGS RSA. Diharapkan penelitian dengan metode ESRKGS ini mampu meningkatkan keamanan dan mempercepat proses pembangkitan kunci, enkripsi, serta dekripsi dibandingkan algoritma RSA standar dan Improvisasi RSA.

2. Metode Penelitian

2.1 Rancangan Aplikasi Chat

Algoritma RSA dan improvisasi algoritma RSA akan diterpakan 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 algoritma improvisasi RSA, dan ESRKGS RSA.



Gambar 1 Gambaran aplikasi chat

Alur dari cara kerja aplikasi sebagai berikut :

1. *Client* 1 dan 2 memasukan IP Address server agar bisa 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.

2.2 Algoritma RSA Standar

Algoritma RSA termasuk algoritma penyandian dengan kunci asimetrik atau algoritma penyandian kunci publik yang didasarkan pada teori pertukaran kunci Diffie-Hellman. Algoritma RSA dibagi menjadi tiga proses penyandian yaitu proses pembangkitan kunci, proses enkripsi dan proses dekripsi. Dalam proses pembangkitan kunci hal pertama yang dilakukan adalah membangkitkan dua bilangan prima besar. Algoritma RSA mempunyai pasangan nilai yang disebut dengan kunci publik yang dapat dipublikasikan untuk enkripsi dan pasangan nilai yang lain disebut dengan kunci privat yang bersifa trahasia untuk dekripsi. Untuk melakukan enkripsi dan dekripsi, algoritma ini menggunakan operasi eksponen dan modular. Algoritma RSA memiliki tingkat keamanan yang berfokus pada sulitnya faktorisasi bilangan besar pada nilai N menjadi 2 bilangan prima (p dan q).

Algoritma RSA ini memiliki 3 tahapan diantaranya pembangkitan kunci, proses enkripsi, proses dekripsi. Penjelasan tiap tahapannya terdapat pada rincian dibawah:

1. Membangkitkan bilangan prima p dan q bernilai acak
2. Menghitung nilai n = p $*$ q
3. Menghitung nilai Φ(n) = $(p - 1) * (q - 1)$
4. Menentukan nilai e dengan rumus gcd $(e,\Phi(n)) = 1$
5. Hitung eksponen dekripsi d sehingga $d = e^{-1} \bmod \Phi(n)$

6. Hasil dari pembangkitan kunci yaitu kunci publik (e,n) dan kunci privat (d,n). Proses pembangkitan kunci dapat dilihat pada gambar 2.

Input: Dua buah bilangan prima (p dan q)

Output: Kunci publik {e, n}, Kunci privat {d, n}

1. $n = p * q$
2. $\Phi(n) = (p - 1) * (q - 1)$
3. $gcd(e, \Phi(n)) = 1$
4. Pilih bilangan bulat e secara random, $1 < e < \Phi(n)$ dan $gcd(e, \Phi(n)) = 1$
5. $d = e^{-1} \bmod \Phi(n)$

Gambar 2 Pseucode pembangkitan kunci algoritma RSA

7. Menghitung *ciphertext* untuk mengenkripsi *plaintext* dengan rumus 1.

$$C = M^e \bmod n \quad (1)$$

Pseucode enkripsi RSA standar dapat dilihat pada gambar 3.

Input: Kunci publik = (e, n), plain text (M)

Output: cipher text (C)

1. for i ← 0 to PanjangPlaintext − 1 do
2. Konversi M ke nilai ASCII
3. $C \leftarrow M^e \bmod n$
4. end for
5. Return C

Gambar 3 Pseucode enkripsi algoritma RSA

8. Menghitung nilai M untuk mendekripsi ciphertext dengan rumus 2.

$$M = C^d \bmod n \quad (2)$$

Pseucode dekripsi algoritma RSA dapat dilihat pada gambar 4.

Input: Kunci privat = (d, n), cipher text (C)

Output: plain text (M)

1. for i ← 0 to PanjangCiphertext − 1 do
2. $Mi \leftarrow C^d \bmod n$
3. Konversi nilai M ke alfabet
4. end for
5. Return M

Gambar 4 Pseucode dekripsi algoritma RSA

2.3 Algoritma Improvisasi RSA

Algoritma RSA dinilai sangat aman selama kurang lebih 20 tahun, akan tetapi berhasil dipecahkan menggunakan serangan faktorisasi (*factorization attack)* yang digunakan untuk mendapatkan nilai bilangan prima p dan q. Pada penelitian yang dilakukan oleh B.Persis Urbana Ivy, Purshotam Mandiwa dan Mukesh Kumar (2012) yaitu memodifikasi algoritma RSA dengan menambah jumlah bilangan prima menjadi 4 buah (p, q, r, dan s) yang akan digunakan untuk pembangkitan kunci privat, kunci publik, serta komponen enkripsi dan dekripsi. Penambahan bilangan prima tersebut dinilai tidak mudah untuk dipecahkan melalui serangan faktorisasi seperti pada umumnya dan juga dinilai lebih aman daripada algoritma RSA Standar. Usulan ini ditulis dalam jurnal yang diterbitkan pada tahun 2012.

Algoritma improvisasi RSA ini memiliki 3 tahapan diantaranya pembangkitan kunci, proses enkripsi, proses dekripsi. Proses enkripsi dan dekripsi menggunakan metode dan rumus yang sama dengan RSA standar, sedangkan penjelasan proses pembangkitan kunci terdapat pada rincian dibawah:

1. Membangkitkan bilangan prima p dan q bernilai acak
2. Menghitung nilai $n = p * q * r * s$
3. Menghitung nilai $\Phi(n) = (p - 1) * (q - 1) * (r - 1) * (s - 1)$
4. Menentukan nilai e dengan rumus $gcd(e, \Phi(n)) = 1$
5. Hitung eksponen dekripsi d sehingga $d = e^{-1} \bmod \Phi(n)$

6. Hasil dari pembangkitan kunci yaitu kunci publik (e,n) dan kunci privat (d,n). Proses pembangkitan kunci dapat dilihat pada gambar 5.

Input: Empat bilangan prima (p, q, r dan s)
Output: Kunci public {e, n}, Kunci privat {d, n}
1. $n = p * q * r * s$
2. $\Phi(n) = (p - 1) * (q - 1) * (r - 1) * (s - 1)$
3. $\gcd(e, \Phi(n)) = 1$
4. Pilih bilangan bulat e secara random, $1 < e < \Phi(n)$ dan $\gcd(e, \Phi(n)) = 1$
5. $d = e^{-1} \bmod \Phi(n)$

Gambar 5 Pseucode pembangkitan kunci algoritma RSA

2.4 Algoritma ESRKGS RSA

Metode ESRKGS yang akan penulis terapkan mengacu pada penelitian M. Thangavel, P. Varalakshmi, Mukund Murrali, K. Nithya (2015) dimana metode ESRKGS RSA memiliki kecepatan yang lebih cepat dibanding improvisasi RSA. Algoritma ESRKGS RSA jauh lebih aman dibanding RSA standar dan improvisasi RSA sehingga membutuhkan waktu penyerangan yang lama untuk memecahkan sistem. ESRKGS RSA adalah skema dan modifikasi pembangkitan kunci dari algoritma RSA standar dan improvisasi RSA. Penelitian diatas mengusulkan empat buah bilangan prima besar untuk meningkatkan kompleksitas sistem dibandingkan dengan RSA standar yang didasarkan pada dua bilangan prima besar. Pembangkitan kunci ESRKGS menggunakan empat bilangan prima yaitu p, q, r, dan s yang akan menghasilkan nilai N. Nilai E dan D tergantung pada nilai N yang merupakan hasil proses dari 4 bilangan prima, sedangkan nilai e_1 dan e_2 diperlukan untuk menemukan nilai E1 sehingga meningkatkan waktu yang dibutuhkan untuk menyerang sistem. Didalam usulan tersebut, komponen nilai n adalah hasil dari dua bilangan prima besar, akan tetapi nilai – nilai enkripsi (E) dan dekripsi (D) didasarkan pada hasil dari empat bilangan prima besar (N) yang membuat sistem menjadi sangat aman. Dengan teknik faktorisasi yang ada sekarang kemungkinan hanya dapat menemukan bilangan prima p, q, r, dan s untuk menemukan nilai N, akan tetapi mengetahui nilai N saja tidaklah mencukupi karena untuk menemukan nilai D maka harus menemukan nilai E_1 terlebih dahulu. Algoritma ESRKGS RSA ini memiliki 3 tahapan diantaranya pembangkitan kunci, proses enkripsi, proses dekripsi. Penjelasan tiap tahapannya terdapat pada rincian dibawah:

1. Bangkitkan empat bilangan prima besar (p, q, r, dan s) secara acak
2. Hitung nilai n, m, dan N dengan rumus $n = p * q$, $m = r * s$, dan $N = n * m$
3. Hitung nilai $\Phi(n)$, $\Phi(m)$, dan $\Phi(N)$ dengan rumus $\Phi(n) = (p - 1) * (q - 1)$, $\Phi(m) = (r - 1) * (s - 1)$, dan $\Phi(N) = \Phi(n) * \Phi(m)$
4. Cari bilangan yang relatif prima dengan rumus $\gcd(e_1, \Phi(n)) = 1$
5. Cari bilangan yang relatif prima dengan rumus $\gcd(e_2, \Phi(m)) = 1$
6. Hitung nilai E_1 dengan rumus $E_1 = e_1^{e2} \bmod N$
7. Cari bilangan yang relatif prima dengan rumus $\gcd(E, \Phi(N) * E_1) = 1$
8. Hitung nilai D dengan rumus $D = E^{-1} \bmod (\Phi(N) * E_1)$
9. Hasil dari pembangkitan kunci yaitu kunci publik (E,n) dan kunci privat (D,n). Proses pembangkitan kunci dapat dilihat pada gambar 6.

Input: Empat bilangan prima (p, q, r dan s)
Output: Kunci public {E, n}, Kunci privat {D, n}
1. $n = p * q$
2. $m = r * s$
3. $N = n * m$
4. $\Phi(n) = (p - 1) * (q - 1)$
5. $\Phi(m) = (r - 1) * (s - 1)$
6. $\Phi(N) = \Phi(n) * \Phi(m)$
7. Pilih bilangan bulat e_1 secara random, $1 < e_1 < \Phi(n)$ dan $\gcd(e_1, \Phi(n)) = 1$
8. Pilih bilangan bulat e_2 secara random, $1 < e_2 < \Phi(m)$ dan $\gcd(e_2, \Phi(m)) = 1$
9. $E_1 = e_1^{e2} \bmod N$
10. Pilih bilangan bulat E secara random, $1 < E < \Phi(N) * E_1$ dan $\gcd(E, \Phi(N) * E_1) = 1$
11. $D = E^{-1} \bmod (\Phi(N) * E_1)$

Gambar 6 Pseudocode pembangkitan kunci algoritma ESRKGS RSA

10. Menghitung *ciphertext* untuk mengenkripsi *plaintext* dengan rumus 3.

$$C = M^E \bmod n \quad (3)$$

Pseucode enkripsi dapat dilihat pada gambar 7.

```
Input: Kunci publik = (E, n), plain text (M)
Output: cipher text (C)
1. for i ← 0 to PanjangPlaintext − 1 do
2. Konversi M ke nilai ASCII
3. C ← M^E mod n
4. end for
5. Return C
```

Gambar 7 Pseudocode enkripsi algoritma ESRKGS RSA

11. Menghitung nilai M untuk mendekripsi ciphertext dengan rumus 4.

$$M = C^D \bmod n \quad (4)$$

Pseucode enkripsi dapat dilihat pada gambar 8.

```
Input: Kunci publik = (E, n), plain text (M)
Output: cipher text (C)
1. for i ← 0 to PanjangPlaintext − 1 do
2. Konversi M ke nilai ASCII
3. C ← M^E mod n
4. end for
5. Return C
```

Gambar 8 Pseudocode dekripsi algoritma ESRKGS RSA

2.5 *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. *Known Plaintext Attack* dapat dicontohkan UserA memiliki kunci publik (21, 41917) yang dibagikan ke UserB dan UserC. UserB kemudian mengenkripsi himpunan ASCII desimal ke a-z (97 - 122) dengan kunci publik UserA maka UserB memiliki himpunan data sebagai berikut:

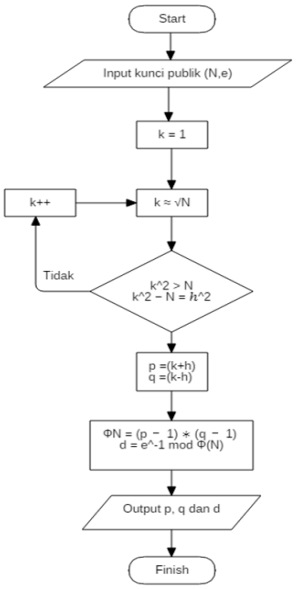
Tabel 1 Contoh himpunan plaintext(P) dan ciphertext(C)

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 UserB menyadap pesan yang dikirimkan oleh UserA dan UserC dengan mencocokan nilai *ciphertext* (C) yang dikirimkan dengan tabel 1. Misal pesan yang dikirimkan dalam bentuk *ciphertext* adalah 87120 20428 121279. Maka dapat dicocokan dengan tabel 1 dimana 14522 = 104 (huruf “h”), 26589 = 97 (huruf “a”), 6059 = 108 (huruf “l”) dan 39987 = 111 (huruf “o”) sehingga menghasilkan plaintext “halo”.

2.6 *Fermat Factorization Attack*

Seorang *attacker* apabila mengetahui faktor dari nilai N pada kunci maka 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 kerja dari fermat factorization dapat dilihat pada gambar 9.



Gambar 9 Alur fermat factorization

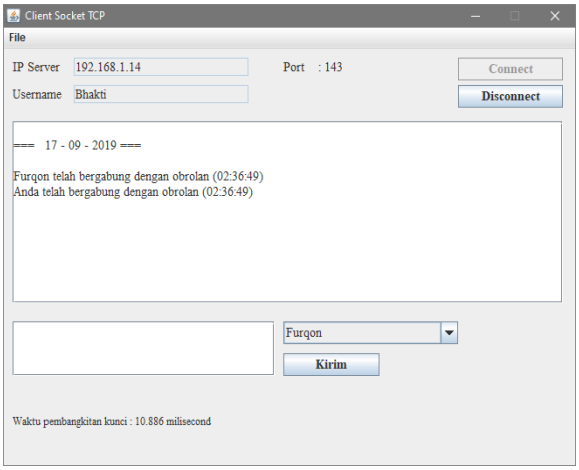
Fermat factorization attack memulai proses dengan menginputkan kunci publik. Mencari nilai k yang mendekati hasil dari $\sqrt{N}$ dengan syarat nilai $k^2 > N$ dan nilai $h^2 = k^2 - N$. Jika tidak memenuhi syarat tersebut maka nilai k akan dilakukan penambahan hingga kondisinya terpenuhi. Nilai p dan q akan didapatkan jika kondisi di atas terpenuhi, sedangkan untuk nilai d dapat dihitung menggunakan rumus yang terdapat pada langkah kelima 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*.

3. Hasil Penelitian dan Pembahasan

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

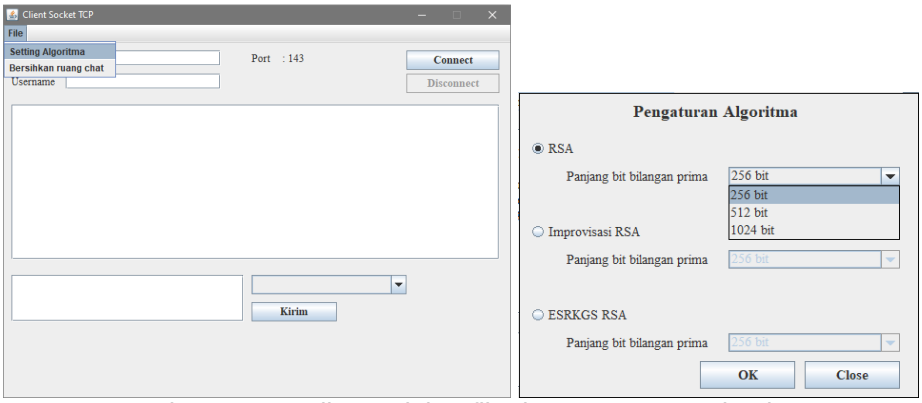
3.1 Implementasi Aplikasi Chat

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 improvisasi RSA dan ESRKGS RSA. Aplikasi pada sisi server akan menampilkan aktivitas yang dilakukan *client* seperti pembangkitan kunci, pesan yang terenkripsi, dan sebagainya.



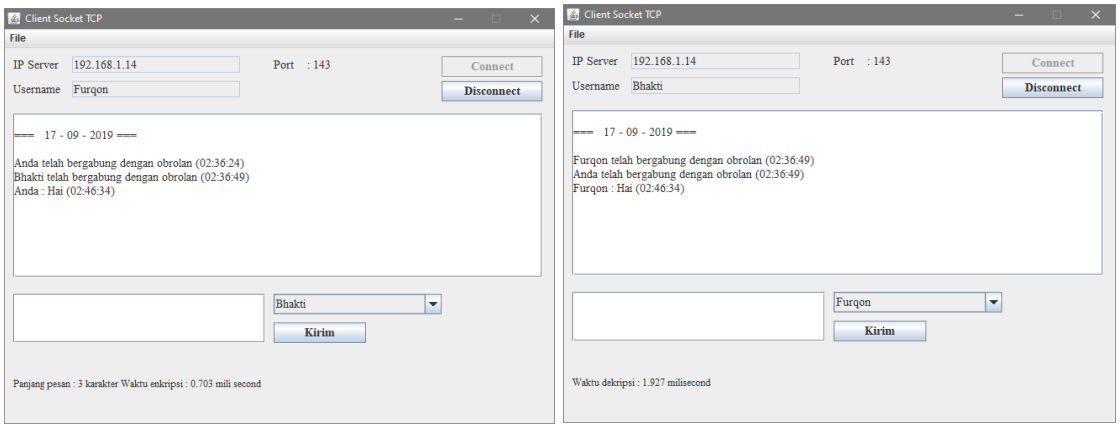
Gambar 10 Tampilan client setelah terhubung

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 11 Tampilan task bar file dan pengaturan algoritma

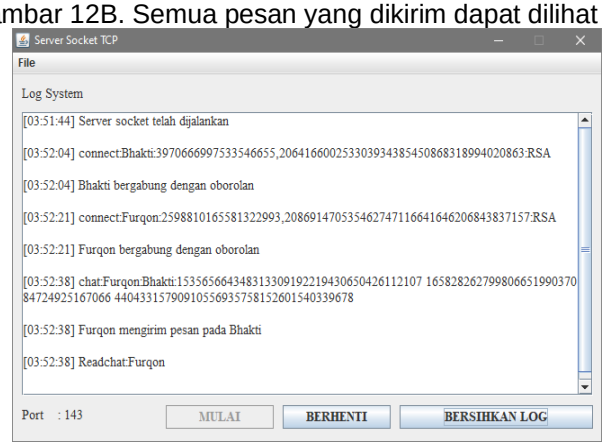
User dapat memilih algoritma yang akan digunakan untuk melakukan enkripsi pada pesan yang akan dikirim dengan menekan task bar “file” sebagaimana pada gambar 11 kemudian memilih setting algoritma.



A B

Gambar 12 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 12A. Halaman penerima akan menampilkan pesan yang sudah didekripsi dan juga waktu dekripsi sebagaimana pada gambar 12B. Semua pesan yang dikirim dapat dilihat pada halaman server.



Gambar 13 Tampilan server

Dapat dilihat 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:Bhakti:39700666997533546655,20641660025330393438545086831899402086 3:RSA merupakan" maksudnya adalah Bhakti merupakan username dari *client* 1, sedangkan 39700666997533546655,2064166002533039343854508683189940208 63 merupakan kunci publik 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 153565664348313309192219430650426112107 165828262799806651990370847249251677066 440433157909105569357581526015403396 78.

3.2 Pengujian Waktu Pembangkitan Kunci

Pengujian perbandingan waktu pembangkitan kunci dilakukan sebanyak tiga kali dengan parameter inputan panjang bit 256, 512, dan 1024. Analisa ini dilakukan untuk membandingkan performa waktu pembangkitan kunci antara algoritma RSA standar, improvisasi RSA, dan ESRKGS RSA. Hasil pengujian dapat dilihat pada Tabel 2.

Tabel 2 Perbandingan waktu pembangkitan kunci

No	Panjang bit Bilangan Prima	RSA Standar (ms)	Improvisasi RSA (ms)	ESRKGS RSA (ms)
1	256 bit	0.788	2.096	1.676
2	512 bit	3.369	8.258	5.970
3	1024 bit	9.901	17.703	10.437
Rata-rata		4.6862.	9.352	6.027

Hasil pengujian performa pada tabel 4.2 menunjukkan bahwa waktu pembangkitan kunci algoritma RSA standar adalah yang tercepat sedangkan improvisasi RSA adalah yang paling lambat, dan ESRKGS RSA berada diantara kedua algoritma tesebut. Hal ini dikarenakan adanya beberapa faktor yang mempengaruhi waktu pembangkitan kunci diantaranya adalah bilangan prima, panjang bit yang digunakan, dan kecepatan sistem dari masing – masing algoritma. Algoritma RSA standar hanya mempunyai dua bilangan prima, sedangkan improvisasi RSA dan ESRKGS RSA mempunyai empat bilangan prima, akan tetapi ESRKGS RSA lebih cepat dari improvisasi RSA yang sama – sama menggunakan empat buah bilangan prima.

3.3 Pengujian Waktu Enkripsi

Pengujian waktu dekripsi dilakukan sebanyak sembilan kali pada masing – masing algoritma dengan parameter inputan panjang karakter 100, 250 dan 400 serta panjang bit 256, 512, dan 1024. Untuk satu parameter panjang karakter dilakukan pengujian pada tiga parameter panjang bit bilangan prima pada masing – masing algoritma. Analisa ini dilakukan untuk membandingkan performa waktu enkripsi antara algoritma RSA standar, improvisasi RSA, dan ESRKGS RSA. Skenario pengujiannya yaitu user 1 mengirim pesan kemudian waktu enkripsi akan tampil sebagaimana pada gambar 14. Hasil pengujian dapat dilihat pada Tabel 3.

Tabel 3 Perbandingan waktu enkripsi

Panjang karakter pesan	Panjang bit Bilangan Prima (bit)	RSA Standar (ms)	Improvisasi RSA (ms)	ESRKGS RSA (ms)
100	256	1.381	2.436	1.294
	512	4.296	10.047	4.200
	1024	19.697	50.950	18.984
250	256	4.358	10.108	4.534
	512	13.415	31.425	11.800
	1024	54.788	141.975	54.686
400	256	8.919	17.496	7.494
	512	39.949	53.710	21.521
	1024	112.500	277.455	95.682
Rata-rata		28.811	66.178	24.466

Pengujian waktu enkripsi pada Tabel 3 menunjukan hasil bahwa semakin panjang karakter maka semakin lama waktu yang dibutuhkan ketiga algoritma tersebut untuk mengenkripsi pesan. Beberapa hal yang mepengaruhi hasil waktu enkripsi adalah panjang karakter pada pesan yang dikirim dan panjang bit bilangan prima, jika semakin panjang karakternya maka akan semakin lama waktu enkripsinya, begitu juga dengan panjang bit bilangan prima, semakin panjang bit dari bilangan prima maka semakin lama waktu dekripsinya. Hasil rata-rata waktu enkripsi pada tabel 3 dapat disimpulkan bahwa ESRKGS RSA memiliki performa enkripsi tercepat, sedangkan improvisasi RSA adalah yang paling lambat, dan algoritma RSA standar berada diantara kedua algoritma tersebut.

3.4 Pengujian Waktu Dekripsi

Pengujian perbandingan waktu sembilan kali pada masing – masing algoritma dengan parameter inputan panjang karakter 100, 250 dan 400 serta panjang bit 256, 512, dan 1024. Untuk satu parameter panjang karakter dilakukan pengujian pada tiga parameter panjang bit bilangan prima pada masing – masing algoritma. Analisa ini dilakukan untuk membandingkan performa waktu dekripsi antara algoritma RSA standar, improvisasi RSA, dan ESRKGS RSA. Skenario pengujiannya yaitu user 2 menerima pesan kemudian waktu dekripsi akan tampil sebagaimana pada gambar 15. Hasil pengujian dapat dilihat pada Tabel 4.

Tabel 4 Perbandingan waktu dekripsi

Panjang karakter pesan	Panjang bit Bilangan Prima (bit)	RSA Standar (ms)	Improvisasi RSA (ms)	ESRKGS RSA (ms)
100	256	6.754	15.096	13.740
	512	17.328	62.802	39.654
	1024	47.936	210.011	163.087
250	256	17.678	44.345	38.179
	512	43.709	130.650	95.393
	1024	135.289	687.971	447.115
400	256	20.640	52.661	46.806
	512	56.998	291.178	168.966
	1024	261.335	1186.905	885.352
Rata-rata		28.811	67.518	297.957

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. Beberapa faktor yang mempengaruhi hasil waktu dekripsi adalah panjang karakter dari pesan yang dikirim dan panjang bit bilangan prima. Panjang karakter sangat berpengaruh, jika semakin panjang karakternya maka akan semakin lama waktu dekripsinya, begitu juga dengan panjang bit bilangan prima, semakin besar bit dari bilangan prima maka semakin lama waktu dekripsinya. Hasil rata-rata waktu dekripsi pada tabel 4.5 dapat disimpulkan bahwa RSA standar memiliki performa dekripsi tercepat, sedangkan improvisasi RSA adalah yang paling lambat, dan algoritma ESRKGS RSA berada diantara kedua algoritma tersebut, akan tetapi ESRKGS RSA lebih cepat dari improvisasi RSA yang sama – sama menggunakan empat buah bilangan prima.

3.5 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 publik 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	Waktu eksekusi (ms)			Persentase Keberhasilan (%)		
		RSA Standar	Improvisasi RSA	ESRKGS RSA	RSA Standar	Improvisasi RSA	ESRKGS RSA
1	50	128.1915	236.839	184.3925	100	100	0
2	100	139.1935	290.853	261.836	100	100	0
3	160	173.64	315.1255	256.443	100	100	0
Rata-rata		147.0083	280.9391	234.2238	100	100	0

Kesimpulan pengujian known plaintext attack pada Tabel 5 adalah algoritma algoritma RSA standar dan improvisasi RSA dapat diserang oleh metode tersebut dengan persentase keberhasilan mencapai 100% atau secara keseluruhan ciphertext dapat dipecahkan oleh sistem. Algoritma ESRKGS RSA tidak berhasil diserang oleh metode tersebut dengan persentase keberhasilan 0% atau secara keseluruhan ciphertext tidak dapat dipecahkan oleh sistem. Faktor – faktor yang mempengaruhi hasil dari ketiga algoritma tersebut antara lain :

1. Secara keseluruhan algoritma RSA standar dan improvisasi RSA mempunyai kemiripan mulai dari pembangkitan kunci, enkripsi, dan dekripsi. Satu – satunya perbedaan dari kedua algoritma tersebut hanya terletak pada jumlah bilangan prima yang berjumlah empat buah bilangan prima pada improvisasi RSA dan dua buah bilangan prima pada RSA standar.
2. Algoritma ESRKGS RSA secara keseluruhan berbeda dari kedua algoritma tersebut mulai dari pembangkitan kunci, enkripsi serta dekripsi seperti yang telah penulis bahas pada bab Metode Penelitian poin D. Satu – satunya persamaan adalah jumlah empat buah bilangan prima seperti pada algoritma improvisasi RSA. ESRKGS RSA tidak dapat dipecahkan oleh known plaintext attack dikarenakan mulai dari proses pembangkitan kunci hingga dekripsi mempunyai rumus yang berbeda sehingga sistem tidak dapat mencocokkan antara plaintext dan ciphertext yang berkorespondensi seperti pada kedua algoritma pembanding.

Berdasarkan hasil – rata waktu eksekusi, sistem lebih cepat memecahkan algoritma RSA standar, dan improvisasi RSA adalah yang paling lambat untuk dipecahkan sedangkan ESRKGS RSA berada diantara kedua algoritma tersebut. Akan tetapi, meskipun algoritma improvisasi RSA mendapatkan rata – rata waktu yang paling lambat, algoritma tersebut masih dapat dipecahkan oleh known plaintext attack dibandingkan dengan ESRKGS yang mendapatkan rata – rata waktu lebih cepat daripada improvisasi RSA.

3.6 Pengujian *Fermat Factorization*

Pengujian *fermat factorization attack* 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 publik terdiri dari nilai e dan n yang akan digunakan untuk mendapatkan nalai p, q dan d untuk algorima RSA standar dan p, q, r, s, dan d untuk algoritma improvisasi RSA dan ESRGKS RSA. Adapun hasilnya dapat dilihat pada tabel 6.

Tabel 6 Pengujian Fermat Factorization Attack

Algoritma	Panjang bit bilangan prima	Waktu eksekusi (ms)	Status kunci privat	Keterangan
RSA Standar	16	3.250	Ditemukan	Kunci privat sesuai
	32	4810694.563	Ditemukan	Kunci privat sesuai
Improvisasi RSA	16	47.359	Ditemukan	Kunci privat sesuai
	32	56100000.190	Belum ditemukan	Resource tidak mencukupi
ESRKGS RSA	16	89.164	Ditemukan	Nilai bilangan prima sesuai, namun kunci privat tidak sesuai
	32	56100000. 320	Belum ditemukan	Resource tidak mencukupi

Kesimpulan pengujian *fermat factorization attack* pada tabel 4.9 adalah algoritma RSA, improvisasi RSA, dan ESRKGS RSA masih dapat diserang dengan metode ini. Hal ini disebabkan oleh kecilnya bilangan prima yang dihasilkan dan kapasitas resource yang mencukupi. Nilai kunci privat yang ditemukan pada ESRKGS RSA tidak sesuai dengan kunci privat yang sebenarnya.

Perhitungan Φ(n) pada algoritma tersebut sudah dimodifikasi yang semula menggunakan rumus $M = C^d \bmod n$ menjadi rumus $\gcd(E, \Phi(N) * E_1) = 1$ 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 ketiga algoritma diserang dengan metode ini. kesimpulannya bahwa ketiga algoritma tersebut masih dapat diserang dengan menggunakan metode *fermat factorization* meskipun nilai d pada ESRKGS RSA tidak sesuai.

4. Kesimpulan

Penelitian yang sudah dilakukan dapat ditarik beberapa kesimpulan sebagai berikut :

1. Algoritma RSA standar, improvisasi RSA, dan ESRKGS RSA dapat diimplementasikan pada aplikasi *instant messaging socket* TCP.
2. Algoritma ESRKGS RSA memiliki performa yang lebih baik dari improvisasi RSA akan tetapi masih dibawah dari RSA standar dikarenakan beberapa factor yang mempengaruhi pada saat proses pembangkitan kunci, enkripsi, dan dekripsi.
3. Algoritma ESRKGS RSA memiliki tingkat keamanan yang lebih baik dibandingkan RSA standar dan improvisasi RSA, dibuktikan dengan tidak ditemukannya kunci privat pada dua metode pengujian serangan yaitu *known plaintext attack* dan *fermat factorization attack*.

Referensi

[1] *W. Stallings, Cryptography and Network Security Principles and Practice Sixth edition, T. Johnson, Ed., boston: PEARSON, 2014, p. 758*. .

[2] A. Arief, “Implementasi Kriptografi Kunci Publik dengan Algoritma RSA-CRT pada Aplikasi Instant Messaging,” vol. 3, no. 1, pp. 46–54, 2016.

[3] 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.

[4] Z. Arifin, “Studi Kasus Penggunaan Algoritma RSA Sebagai Algoritma Kriptografi yang Aman Zainal,” vol. 4, no. 3, pp. 7–14, 2009.

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

[6] M. Thangavel, P. Varalakshmi, M. Murrali, and K. Nithya, “An Enhanced and Secured RSA Key Generation Scheme (ESRKGS),” *J. Inf. Secur. Appl.*, vol. 20, pp. 3–10, Feb. 2015.

[7] A. Khairan, M. Imrona, and I. Ummah, “Analisis dan Implementasi Kriptografi RSA Pada Aplikasi Chatting Client-Server Based.” Prodi Ilmu Komputasi Telkom University, Bandung, 2014.

[8] E. Halord, *Java Network Programming Fourth Edition*. Sebastopol: O’Reilly Media, Inc, 2013.

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

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

[11] “H. R. Sandityas, ‘Analisa Hybrid Kriptosistem RSA dan EL-GAMAL pada Instant Messaging berbasis socket TCP,’ Malang, 2018.”