Available at https://jtsiskom.undip.ac.id (15 February 2020)
DOI:10.14710/jtsiskom.8.2.2020.113-120

Jurnal Teknologi dan Sistem Komputer, 8(2), 2020, 113-120

Algoritme RSA menggunakan pembangkit kunci ESRKGS untuk enkripsi pesan chat dengan protokol TCP/IP

RSA algorithm using key generator ESRKGS to encrypt chat messages with TCP/IP protocol

Aminudin[*)], Gadhing Putra Aditya, Sofyan Arifianto

Program Studi Informatika, Universitas Muhammadiyah Malang
Jalan Raya Tlogomas No. 246, Kel. Tlogomas, Kec. Lowokwaru, Kota Malang, Indonesia 65144

Cara sitasi: A. Aminudin, G. P. Aditya, and S. Arifianto, "Algoritme RSA menggunakan pembangkit kunci ESRKGS untuk enkripsi pesan chat dengan protokol TCP/IP," *Jurnal Teknologi dan Sistem Komputer*, vol. 8, no. 2, pp. 113-120, 2020. doi: 10.14710/jtsiskom.8.2.2020.113-120, [Online].

Abstract – This study aims to analyze the performance and security of the RSA algorithm in combination with the key generation method of enhanced and secured RSA key generation scheme (ESRKGS). ESRKGS is an improvement of the RSA improvisation by adding four prime numbers in the property embedded in key generation. This method was applied to instant messaging using TCP sockets. The ESRKGS+RSA algorithm was designed using standard RSA development by modified the private and public key pairs. Thus, the modification was expected to make it more challenging to factorize a large number n into prime numbers. The ESRKGS+RSA method required 10.437 ms faster than the improvised RSA that uses the same four prime numbers in conducting key generation processes at 1024-bit prime number. It also applies to the encryption and decryption process. In the security testing using Fermat factorization on a 32-bit key, no prime number factor was found. The test was processed for 15 hours until the tester computer resource runs out.

Keywords *– prime numbers; RSA algorithm; RSA ESRKGS; Fermat factorization*

Abstrak – Penelitian ini bertujuan untuk menganalisis kinerja dan keamanan dari algoritme RSA yang dikombinasikan dengan metode pembangkitan kunci enhanced and secured RSA key generation scheme (ESRKGS). Metode tersebut diterapkan pada aplikasi pesan instan menggunakan socket TCP. ESRKGS merupakan pengembangan dari improvisasi RSA dengan menambahkan empat bilangan prima di dalam properti yang tertanam di dalam pembangkitan kunci. Algoritme ESRKGS+RSA dirancang dengan menggunakan pengembangan RSA standar dengan memodifikasi pasangan kunci privat dan publik. Modifikasi tersebut bertujuan agar dapat mempersulit dalam memfaktorkan bilangan besar n menjadi faktor primanya. Pengujian kinerja menunjukkan bahwa ESRKGS+RSA membutuhkan waktu 10,437 ms yang lebih cepat daripada RSA improvisasi yang sama-sama menggunakan empat bilangan prima dalam proses pembangkitan kunci pada bilangan prima 1024 bit. Hal tersebut berlaku juga pada proses enkripsi dan dekripsi. Pengujian keamanan dengan menggunakan faktorisasi Fermat untuk pada kunci 32 bit tidak ditemukan nilai faktor bilangan prima. Pengujian tersebut diproses selama 15 jam sampai sumber daya komputer penguji habis.

Kata kunci *– bilangan prima; algoritme RSA; RSA ESRKGS; faktorisasi Fermat*

I. PENDAHULUAN

RSA termasuk algoritme penyandian dengan kunci asimetrik atau algoritme penyandian kunci publik yang didasarkan pada teori pertukaran kunci Diffie-Hellman [1]. Algoritme ini terbagi dalam tiga proses, yaitu pembangkitan kunci, enkripsi dan dekripsi. RSA memiliki tingkat keamanan yang berfokus pada sulitnya faktorisasi bilangan besar pada nilai *n* menjadi 2 bilangan prima *p* dan *q* [2].

Algoritme RSA standar memiliki celah keamanan pada kunci publik maupun privat yang berasal dari masukan dua bilangan prima saat pembangkitan kunci [3]. Operasi matematika pada RSA terdiri dari perkalian dua buah bilangan prima *p* dan *q* yang dipilih secara acak. Hasil dari perkalian tersebut adalah *n*. Semakin besar ukuran *n*, semakin baik tingkat keamanannya karena akan menyulitkan penyerang untuk memecahkan faktorisasi dari nilai *n* [4]. Bilangan prima bernilai besar akan membuat RSA sulit untuk ditemukan nilai faktor bilangan prima *p* dan *q*-nya. Selain itu, untuk memperkuat algoritme RSA dapat dilakukan dengan menambahkan jumlah bilangan prima dalam pembangkitan kunci publik maupun kunci privat [5].

Namun, penambahan jumlah bilangan prima masih dapat dipecahkan dengan penyerangan faktorisasi, salah satunya dengan metode faktorisasi Fermat [6]. Di samping itu, penambahan jumlah bilangan prima membuat kinerja RSA menjadi menurun dalam hal

[*)] Penulis korespondensi (Aminudin)
Email: aminudin2008@umm.ac.id

Submitted: 24 September 2019; Revised: 3 February 2020; Accepted: 7 February 2020; Published: 30 April 2020

waktu pemrosesan, baik dalam pembangkitan kunci, proses enkripsi, dan dekripsi. Teknik agar pemfaktoran bilangan prima sulit untuk dilakukan tetapi tetap menjaga kinerja RSA berupa waktu pemrosesan tetap terjaga perlu dilakukan.

Di sisi lain, teknik penyandian diperlukan dalam aplikasi pertukaran data, salah satunya adalah aplikasi pesan instan (*instant messaging*, IM) untuk komunikasi secara *real time* yang terhubung pada protokol Internet. Pesan instan ini memerlukan media penghubung antara pengirim dan penerima, salah satunya menggunakan socket TCP yang digunakan untuk pengiriman dan penerimaan data melalui jaringan komputer [7]-[9]. Socket TCP terdiri atas server dan klien TCP dimana server TCP menyimpan kunci publik dari klien. Jaringan yang digunakan umumnya bersifat publik yang memungkinkan terjadinya penyadapan data sehingga teknik kriptografi perlu diimplementasikan.

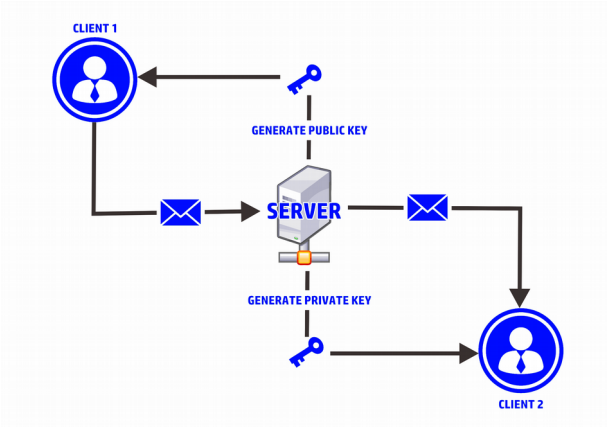
Beragam kajian telah membahas tentang kombinasi dan improvisasi untuk menambahkan tingkat keamanan RSA [5], [10]-[14]. Namun, penelitian tersebut hanya membahas pengembangan algoritme RSA dengan menggunakan satu metode saja. Penggunaan tiga bilangan prima (*p*, *q*, dan *s*) dan sebuah variabel (*n*) dilakukan dalam [10], [11]. Ivy dkk. [12] melakukan improvisasi RSA menggunakan 4 buah bilangan prima (*p*, *q*, *r*, dan *s*) dan sebuah variabel (*n*). Swami dkk. [13] menggunakan RSA dual modulus berbasis Jordan-Totient untuk mengamankan terhadap serangan *brute-force*. Lal dkk. [14] mengembangkan faktorisasi KNJ untuk mempercepat kinerja RSA dengan mengecek faktor yang bernilai ganjil dan prima.

Penelitian ini bertujuan mengembangkan algoritme RSA yang dikombinasikan melalui metode pembangkitan kunci menggunakan *enhanced and secured RSA key generation scheme* (ESRKGS). Penelitian ini menggunakan dua buah metode, yaitu menambahkan jumlah bilangan prima menjadi empat dan mengubah proses pengembangan pasangan kunci publik yang awalnya $gcd(e,\Phi(n))=1$ menjadi $gcd(E,\Phi(N) \times E_1)=1$ dan pasangan kunci privat yang awalnya $d=e^{-1} \bmod \Phi(n)$ menjadi $d = e^{-1} \bmod (\Phi(N) \times e_1)$. Algoritme ini diterapkan dalam aplikasi pesan instan menggunakan socket TCP. Perbandingan kinerja dan tingkat keamanan dilakukan untuk RSA standar, improvisasi RSA improvisasi, dan RSA+ESRKGS. Pengujian kinerja meliputi kecepatan waktu eksekusi pembangkitan kunci, proses enkripsi, dan proses dekripsi. Pengujian keamanan menggunakan metode *known plaintext attack* dan faktorisasi Fermat.

II. METODE PENELITIAN

A. Rancangan aplikasi chat

Algoritme RSA, improvisasi RSA, dan RSA+ESRKGS diterapkan pada aplikasi chat menggunakan jaringan lokal. Gambar 1 menunjukkan rancangan aplikasi chat yang dibangun dengan dua orang klien yang saling berkomunikasi. Sebuah server berfungsi untuk menghubungkan kedua klien tersebut agar dapat



Gambar 1. Rancangan aplikasi chat yang dibangun

mengirim dan menerima pesan. Server juga berfungsi untuk memastikan pesan yang dikirim aman dengan menerapkan ketiga algoritme tersebut. Komunikasi disetel oleh klien 1 dan klient 2 dengan memasukkan alamat IP server agar bisa terhubung dengan server. Klien 1 dan klien 2 mendapat kunci privat dan publik dari server. Klien 1 mengirim pesan yang terenkripsi menggunakan kunci publik yang dimiliki klien 2 dan klien 2 menerima pesan dari klien 1 dan mendekripsinya menggunakan kunci privatnya.

B. Algoritme RSA standar

Algoritme penyandian dengan kunci asimetrik terbagi dalam tiga proses, yaitu pembangkitan kunci, enkripsi dan dekripsi [1]. Dalam proses pembangkitan kunci hal pertama yang dilakukan adalah membangkitkan dua bilangan prima besar. Pada algoritme ini, terdapat pasangan nilai yang disebut dengan kunci publik yang dapat dipublikasikan untuk enkripsi dan kunci privat yang bersifat rahasia untuk dekripsi. Untuk melakukan enkripsi dan dekripsi, algoritme ini menggunakan operasi eksponen dan modular. RSA memiliki tingkat keamanan yang berfokus pada sulitnya faktorisasi bilangan besar pada nilai *n* menjadi 2 bilangan prima *p* dan *q* [10], [11].

Pembangkitan kunci dilakukan untuk menghasilkan bilangan prima *p* dan *q* yang yang diperoleh secara acak dan menghitung nilai $n=p\times q$. Penghitungan Totient Euler *n* dilakukan dengan cara $\Phi(n)=(p-1)\times(q-1)$ yang digunakan untuk perhitungan kunci publik dan kunci privat. Pasangan kunci public dihitung dengan cara menentukan nilai *e* menggunakan rumus $gcd(e,\Phi(n))=1$ dan kunci privat dihitung dengan mencari nilai eksponen dekripsi *d* sehingga $d=e^{-1} \bmod \Phi(n)$. Hasil dari proses pembangkitan kunci adalah kunci publik *(e,n)* dan kunci privat *(d,n)*. Proses pembangkitan kunci RSA ditunjukkan pada Algoritme 1.

Setelah pasangan kunci publik dan privat diperoleh, langkah selanjutnya adalah menghitung cipherteks *c* untuk mengenkripsi plainteks *m* dengan kunci publik *e* menggunakan Persamaan 1. Nilai *m* merupakan plainteks yang terbagi menjadi blok $m_0, m_1, \ldots, m_{n-1}$ sedemikian sehingga setiap blok merepresentasikan nilai

dengan jangkauan $[0, n-1]$. Fungsi enkripsi ditunjukkan pada Algoritme 2. Proses dekripsi dilakukan untuk mencari plainteks *m* dari nilai cipherteks *c* dengan kunci privat *d* menggunakan Persamaan 2. Alur fungsi dekripsi dinyatakan dalam Algoritme 3.

$$c = m^e \bmod n \quad (1)$$

$$m = c^d \bmod n \quad (2)$$

C. Algoritme improvisasi RSA

Algoritme RSA telah berhasil dipecahkan menggunakan serangan faktorisasi (*factorization attack)* yang digunakan untuk mendapatkan nilai bilangan prima *p* dan *q*. Modifikasi RSA telah dilakukan dalam [12] dengan menambah jumlah bilangan prima menjadi empat buah, yaitu *p*, *q*, *r*, dan *s*, yang 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 RSA standar.

Hampir sama seperti RSA standar, algoritme RSA dengan model improvisasi ini juga memiliki 3 proses tahapan seperti pembangkitan kunci, proses enkripsi, proses dekripsi. Proses pertama yaitu membangkitkan empat bilangan prima p, q, r, s secara acak. Nilai n dihitung dengan formula $n = p \times q \times r \times s$. Totient Euler *n* dihitung dengan cara $\Phi(n) = (p-1) \times (q-1) \times (r-1) \times (s-1)$. Setelah proses tersebut dilakukan, pasangan kunci publik dicari dengan menghitung nilai *e* yang diperoleh dari $gcd(e, \Phi(n)) = 1$. Nilai d yang digunakan untuk kunci privat dihitung dengan $d = e^{-1} mod\, \Phi(n)$. Hasil dari pembangkitan kunci adalah kunci publik (e, n) dan kunci privat (d, n). Proses pembangkitan kunci dinyatakan dalam Algoritme 4. Proses enkripsi dan dekripsi menggunakan Persamaan 1 dan Persamaan 2.

D. Kombinasi algoritme RSA dan ESRKGS

Algoritme ESRKGS+RSA adalah skema dan modifikasi pembangkitan kunci dari RSA standar dan improvisasi RSA. ESRKGS+RSA menggunakan 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 menghasilkan nilai *n* seperti dinyatakan dalam Algoritme 5.

Nilai *e* dan *d* tergantung pada nilai *n* yang merupakan hasil proses dari empat bilangan prima. Nilai e_1 dan e_2 diperlukan untuk menemukan nilai E_1 sehingga meningkatkan waktu yang dibutuhkan untuk menyerang sistem. Prosesnya adalah membangkitkan empat bilangan prima besar *p*, *q*, *r*, dan *s* secara acak dan menghitung nilai *n*, *m*, dan *N* dengan cara $n = p \times q, m = r \times s, N = n \times m$. Nilai Toient Euler dicari dari nilai *Φ(n)*, *Φ(m)*, dan *Φ(N)* dimana *Φ(n) = (p - 1) x (q - 1)*, *Φ(m) = (r - 1) x (s - 1)*, dan *Φ(N) = Φ(n) x Φ(m)*. Untuk mencari nilai kunci publik *E* tahapan yang dilakukan adalah :

- mencari bilangan yang relatif prima dimana $gcd(e_1, \Phi(n)) = 1$;
- mencari bilangan yang relatif prima dimana $gcd(e_2, \Phi(m)) = 1$;
- menghitung nilai E_1 dimana $E_1 = e_1^{e2} \bmod N$; dan
- mencari bilangan yang relatif prima dimana $gcd(E, \Phi(N) * E_1) = 1$.

Kunci privat dicari dengan menghitung nilai $D = E^{-1} \bmod (\Phi(N) * E_1)$. Hasil dari pembangkitan kunci adalah kunci publik *(E,n)* dan kunci privat *(D,n)*.

Proses enkripsi dilakukan untuk menghitung cipherteks *c* dari plainteks *m* dengan kunci publik *E* menggunakan Persamaan 3. Proses dekripsi dilakukan untuk menghitung plainteks *m* dari cipherteks *c* dengan kunci privat *D* menggunakan Persamaan 4. Proses enkripsi dan dekripsi menggunakan ESRKGS+RSA dinyatakan dalam Algoritme 6 dan Algoritme 7.

Algoritme 1. Pembangkitan kunci RSA standar

Input : Dua buah bilangan prima (p dan q)
Output : Kunci publik {e, n}, kunci privat {d, n}
Deklarasi : p, q, e, d : integer
Algoritme
1: n = p x q
2: Φ(n) = (p - 1) x (q - 1)
3: gcd (e, Φ(n)) = 1< e < Φ(n) dan gcd(e, Φ(n)) = 1
4: d = e^{-1} mod Φ(n)

Algoritme 2. Proses enkripsi RSA standar

Input : Kunci publik (e, n), plainteks (M)
Output : cipherteks (C)
Deklarasi : M : teks
Algoritme
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

Algoritme 3. Proses dekripsi RSA standar

Input : Kunci privat (d, n), cipherteks (C)
Output : plainteks (M)
Deklarasi : C : integer
Algoritme
1: **for** i ← 0 to PanjangCiphertext-1 **do**
2: M_i ← C^d mod n
3: Konversi nilai m ke alfabet
4: **end for**
5: **return** M

Algoritme 4. Proses pembangkitan RSA improvisasi

Input : Empat bilangan prima (p, q, r dan s)
Output : Kunci publik {e, n}, kunci privat {d, n}
Deklarasi : p, q, r, s, e, d : integer
Algoritme
1: n = p x q x r x s
2: Φ(n) = (p - 1) x (q – 1) x (r - 1) x (s - 1)
3: gcd (e, Φ(n)) = 1
4: d = e^{-1} mod Φ(n)

$$c = m^E \bmod n \quad (3)$$

$$m = c^D \bmod n \quad (4)$$

E. Serangan *known plaintext attack*

Known plaintext attack dapat dilakukan oleh penyerang yang mengetahui kunci publik dan pesan terenkripsi. Penyerang akan membentuk baris himpunan ASCII antara plainteks *P* dan cipherteks *C*. Penyerang mencocokkan antara cipherteks dengan plainteks apakah terdapat plainteks yang berkorespondensi. Penyerang menyimpan plainteks jika terdapat plainteks yang berkorespondensi. Contaoh serangan ini adalah misalnya 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 seperti dinyatakan dalam Tabel 1.

UserB menyadap pesan yang dikirimkan oleh UserA dan UserC dengan mencocokkan nilai cipherteks *C* yang dikirimkan dengan Tabel 1. Misalnya, jika pesan yang dikirimkan dalam bentuk cipherteks adalah 87120 20428 121279, maka dapat dicocokan dengan tabel tersebut dimana 14522 = 104 (huruf 'h'), 26589 = 97 (huruf 'a'), 6059 = 108 (huruf 'l') dan 39987 = 111 (huruf 'o'). Plainteks hasil serangan adalah "halo".

F. Serangan faktorisasi Fermat

Serangan ini dilakukan oleh seorang penyerang jika mengetahui faktor dari nilai *n* pada kunci. Penyerang akan mengetahui nilai eksponen *d* yang terdapat pada kunci privat (d, n) yang diperoleh dari nilai kunci publik (e, n). Pesan yang telah dienkripsi dengan mudah dapat didekripsi oleh penyerang tersebut. Alur kerja dari serangan faktorisasi Fermat ditunjukkan pada Algoritme 8.

Serangan faktorisasi Fermat memulai proses dengan memasukkan kunci publik. Nilai *k* yang mendekati hasil dari $\sqrt{N}$ dicari dengan syarat nilai $k^2 > N$ dan nilai $h^2 = k^2 - N$. Jika tidak memenuhi syarat tersebut, maka nilai *k* dilakukan penambahan hingga kondisinya terpenuhi. Nilai *p* dan *q* didapatkan jika kondisi di atas terpenuhi, sedangkan untuk nilai *d* dapat dihitung menggunakan rumus yang terdapat pada langkah kesembilan pada Algoritme 8. Sistem menampilkan bilangan prima *p, q* dan kunci privat *d*. Jika ketiga parameter tersebut diperoleh, penyerang dapat dengan mudah mengetahui plainteks ataupun cipherteks yang dikirim oleh pengguna.

III. HASIL DAN PEMBAHASAN

Hasil dari penelitian ini adalah berupa hasil implementasi dari ketiga algoritme pada aplikasi chat, kinerja waktu yang dibutuhkan dalam melakukan proses pembangkitan kunci, enkripsi, dan dekripsi, dan tingkat keamanan dari ketiga algoritme dengan pengujian terhadap serangan *known plaintext attack* dan faktorisasi Fermat.

Algoritme 5. Pembangkitan kunci ESRKGS+RSA

Input : Empat bilangan prima (p, q, r dan s)
Output : kunci publik {E, n}, kunci privat {D, n}
Deklarasi : p, q, r, s, E, D, N : integer
Algoritme
1: n = p x q
2: m = r x s
3: N = n x m
4: Φ(n) = (p - 1) x (q - 1)
5: Φ(m) = (r - 1) x (s - 1)
6: Φ(N) = Φ(n) x Φ(m)
7: $E_1 = e_1^{e2} \bmod N$
8: 1 < E < Φ(N) x E_1 dan gcd(E, Φ(N) x E_1) = 1
9: $D = E^{-1} \bmod (\Phi(N) \times E_1)$

Algoritme 6. Proses enkripsi ESRKGS+RSA

Input : Kunci publik {E, n}, plainteks (M)
Output : cipherteks Cc)
Deklarasi : M : teks
Algoritme
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

Algoritme 7. Proses dekripsi ESRKGS+RSA

Input : Kunci publik {D, n}, chiperteks (C)
Output : plainteks (M)
Deklarasi : D, C: integer
Algoritme
1: **for** i ← 0 to PanjangCiphertext - 1 **do**
2: M_i ← C^D mod n
3: Konversi nilai M ke alfabet
4: **end for**
5: **return** M

Tabel 1. Contoh himpunan plainteks P dan cipherteks 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		

A. Implementasi aplikasi chat

Aplikasi yang dibangun meliputi aplikasi pada klien dan pada server. Di aplikasi klien, pesan yang dikirim dilakukan proses enkripsi terlebih dahulu dengan algoritme RSA, improvisasi RSA, dan ESRKGS+RSA. Aplikasi pada sisi server menampilkan aktivitas yang dilakukan klien, di antaranya pembangkitan kunci dan menampilkan pesan yang terenkripsi seperti ditunjukkan pada Gambar 2.

Pengguna terhubung jika sudah memasukan alamat IP server dan akunnya yang berisi username dan password. Aplikasi menampilkan pesan jika pengguna telah terhubung dengan jaringan lokal. Aplikasi juga menampilkan waktu pembangkitan kunci. Algoritme awal yang digunakan adalah RSA sehingga waktu pembangkitan kunci saat awal terhubung adalah kunci dari RSA. Pesan yang dikirim ditampilkan pada halaman pengirim dan penerima. Panjang pesan dan waktu enkripsi ditampilkan pada halaman pengirim. Halaman penerima juga menampilkan pesan yang sudah didekripsi dan waktu dekripsinya. Aplikasi pesan chat di jaringan TCP/IP telah mengimplementasikan penyandian pesan terkirim untuk menambah tingkat keamaannya seperti dalam [7]-[9], namun dengan menggunakan penyandian yang berbeda.

B. Pengujian waktu pembangkitan kunci

Pengujian aplikasi dilakukan dengan menggunakan spesifikasi perangkat lunak dan perangkat keras. Perangkat keras yang digunakan adalah komputer dengan prosesor Intel(R) Core(TM) i5-4200U CPU @1.60–2.30 Ghz, RAM 8 GB (4 GB + 4 GB), serta penyimpan SSHD 500 GB dan SSD 120GB. Perangkat lunak yang digunakan untuk melakukan pengujian adalah Netbeans IDE versi 8.2 sebagai editor Java, kompiler Java JDK versi 8.0 dan JRE versi 8.0, dan sistem operasi Windows 10 Home 64 bit.

Pengujian pembangkitan kunci dilakukan sebanyak tiga kali dengan parameter inputan panjang bit 256, 512, dan 1024. Analisis ini dilakukan untuk membandingkan performa waktu pembangkitan kunci antara algoritme RSA standar, improvisasi RSA, dan ESRKGS+RSA.

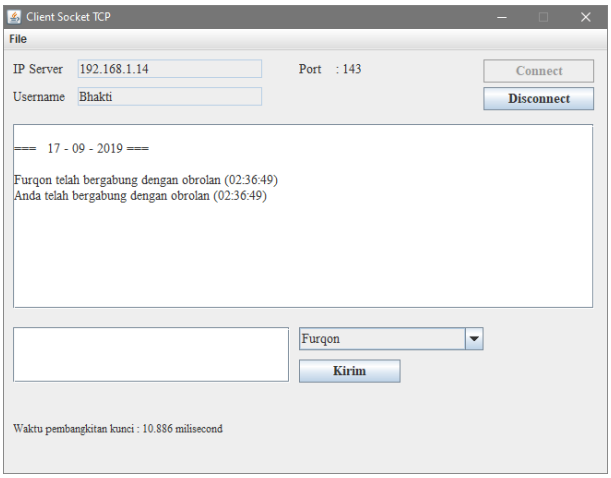
Hasil pengujian ditunjukkan pada Tabel 2. Pengujian kinerja tersebut menunjukkan bahwa waktu pembangkitan kunci algoritme RSA standar adalah yang tercepat sedangkan improvisasi RSA adalah yang paling lambat, dan ESRKGS+RSA berada di antara kedua algoritme tesebut. Hal ini disebabkan karena adanya beberapa faktor yang mempengaruhi waktu pembangkitan kunci, di antaranya adalah bilangan prima, panjang bit yang digunakan, dan kecepatan sistem dari tiap algoritme. RSA standar hanya mempunyai dua bilangan prima, sedangkan improvisasi RSA dan ESRKGS+RSA mempunyai empat bilangan prima. Namun, dengan jumlah bilangan prima yang sama, ESRKGS+RSA lebih cepat dari improvisasi RSA.

C. Pengujian waktu enkripsi

Pengujian waktu dekripsi dilakukan sebanyak sembilan kali pada tiap algoritme dengan parameter masukan panjang karakter 100, 250 dan 400 serta panjang bit bilangan prima 256, 512, dan 1024. Untuk tiap parameter panjang karakter, dilakukan pengujian pada tiga parameter panjang bit bilangan prima pada tiap algoritme. Analisis ini dilakukan untuk membandingkan kinerja waktu enkripsi antara algoritme RSA standar, improvisasi RSA, dan ESRKGS+RSA.

Algoritme 8. Serangan faktorisasi Fermat

Input : Kunci publik {N, e}
Output : p, q, dan kunci privat d
Deklarasi : N: integer
Algoritme
1: k = 1
2: **while** (k^2 <= N) **do**
3: k++
4: $h^2 = k^2 - N$
5: **end while**
6: p = (k+h)
7: q = (k-h)
8: Φ(n) = (p - 1) x (q – 1)
9: $d = e^{-1} \bmod \Phi(n)$



Gambar 2. Tampilan klien setelah terhubung

Tabel 2. perbandingan waktu pembangkitan kunci

No	Panjang bit Bilangan Prima	Waktu pembangkitan kunci (ms) RSA Standar	RSA Improvisasi	ESRKGS + RSA
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

Tabel 3. Perbandingan waktu enkripsi tiap algoritme

Panjang karakter pesan	Panjang Bilangan Prima (bit)	Waktu enkripsi (ms) RSA Standar	RSA Improvisasi	ESRKGS + RSA
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

Skenario pengujiannya adalah pengguna 1 mengirim pesan dan waktu enkripsi tampil di dalam aplikasi.

Hasil pengujian waktu enkripsi dinyatakan pada Tabel 3. Hasil tersebut menunjukkan bahwa semakin panjang karakter maka semakin lama waktu yang

dibutuhkan ketiga algoritme 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 semakin lama waktu enkripsinya. Demikian juga dengan panjang bit bilangan prima. Semakin panjang bit dari bilangan prima, maka semakin lama waktu enkripsinya. Algoritme ESRKGS+RSA memiliki performa enkripsi tercepat, sedangkan improvisasi RSA adalah yang paling lambat, dan RSA standar berada di antara kedua algoritme tersebut.

D. Pengujian waktu dekripsi

Pengujian perbandingan waktu sembilan kali pada masing-masing algoritme 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 algoritme. Analisa ini dilakukan untuk membandingkan performa waktu dekripsi antara algoritme RSA standar, improvisasi RSA, dan ESRKGS+RSA.

Hasil pengujian waktu dekripsi dinyatakan pada Tabel 4. Hasil pengujian waktu dekripsi tersebut menunjukan hasil waktu yang hampir sama 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 yang didekripsi sangat berpengaruh. Semakin panjang karakternya, maka semakin lama waktu dekripsinya. Demikian juga dengan panjang bit bilangan prima. Semakin besar bit dari bilangan prima, maka semakin lama waktu dekripsinya.

Hasil rata-rata waktu dekripsi menunjukkan bahwa RSA standar memiliki performa dekripsi tercepat, sedangkan improvisasi RSA adalah yang paling lambat, dan algoritme ESRKGS+RSA berada di antara kedua algoritme tersebut. Namun, ESRKGS+RSA lebih cepat dari improvisasi RSA yang sama-sama menggunakan empat buah bilangan prima.

Hasil kinerja waktu eksekusi algoritme RSA improvisasi dan ESRKGS+RSA tersebut di atas sejalan dengan [3], [12], [13]. RSA yang diimprovisasikan dan dikombinasikan dengan metode yang lain untuk memperkuat tingkat keamanan dalam menemukan kunci publik dan kunci privat memiliki tingkat pemrosesan waktu yang relatif lebih lama dibandingkan dengan RSA standar. RSA dengan *optimal asymmetric encryption padding* (OAEP) memiliki tingkat waktu kenaikan proses enkripsi dan dekripsi yang cukup signifikan daripada RSA yang standar dalam [3]. Hal yang sama dinyatakan dalam [12] yang memodifikasi modulus dual berdasarkan fungsi Jordan-Totient DMRJT. Waktu eksekusi DMRJT lebih lambat dari RSA tetapi DMRJT memiliki keamanan yang tinggi daripada RSA karena menggunakan dua kali lipat kunci privat dan publik.

Tabel 4. Perbandingan waktu dekripsi tiap algoritme

Panjang karakter pesan	Panjang Bilangan Prima (bit)	Waktu dekripsi (ms)		
		RSA Standar	RSA Improvisasi	ESRKGS + RSA
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	1.186,905	885,352
Rata-rata		28,811	67,518	297,957

Tabel 5. Hasil pengujian *known plaintext attack*

Waktu eksekusi (ms)			Persentase Keberhasilan (%)		
RSA Standar	RSA Improvisasi	ESRKG +RSA	RSA Standar	RSA Improvisasi	ESRKGS +RSA
128	236	184	100	100	0
139	290	261	100	100	0
173	315	256	100	100	0
147	280	234	100	100	0

E. Pengujian *known plaintext attack*

Skenario pengujian adalah penyerang mendapatkan kunci publik dan cipherteks yang ada pada server dengan teknik *sniffing*. Cipherteks dan kunci publik kemudian digunakan untuk memecahkan plainteks atau pesan dari klien. Pengujian *known plaintext attack* menggunakan parameter panjang cipherteks, waktu eksekusi, dan persentase keberhasilan. Semakin kecil persentase keberhasilan serangan, maka sistem enkripsi semagin bagus. Panjang cipherteks yang digunakan adalah 50, 100 dan 160.

Hasil pengujian *known plaintext attack* ditunjukkan pada Tabel 5. RSA standar dan improvisasi RSA dapat diserang oleh metode tersebut dengan persentase keberhasilan mencapai 100% atau secara keseluruhan cipherteks dapat dipecahkan oleh sistem. Hal ini sejalan dengan [3]. ESRKGS+RSA tidak berhasil diserang oleh metode tersebut dengan persentase keberhasilan 0% atau secara keseluruhan cipherteks tidak dapat dipecahkan oleh sistem. Faktor yang mempengaruhi hasil dari kedua algoritme tersebut adalah secara keseluruhan RSA standar dan improvisasi RSA mempunyai kemiripan mulai dari pembangkitan kunci, enkripsi, dan dekripsi. Satu-satunya perbedaan dari kedua algoritme tersebut hanya terletak pada jumlah bilangan prima yang berjumlah empat buah bilangan prima pada improvisasi RSA dan dua buah bilangan prima pada RSA standar.

Di sisi lain, ESRKGS+RSA secara keseluruhan berbeda dari kedua algoritme tersebut mulai dari pembangkitan kunci, enkripsi, dan dekripsi. Satu-satunya persamaan adalah jumlah empat buah bilangan prima seperti pada algoritme improvisasi RSA seperti [12]. ESRKGS+RSA tidak dapat dipecahkan oleh *known plaintext attack* disebabkan karena mulai dari

proses pembangkitan kunci hingga dekripsi mempunyai rumus yang berbeda sehingga sistem tidak dapat mencocokkan antara plainteks dan cipherteks yang berkorespondensi seperti pada kedua algoritme pembanding.

Berdasarkan hasil rata-rata waktu eksekusi serangan, pengujian *known plaintext attack* lebih cepat memecahkan algoritme RSA standar. Improvisasi RSA adalah yang paling lambat untuk dipecahkan sedangkan ESRKGS+RSA berada di antara kedua algoritme tersebut. Namun, meskipun algoritme improvisasi RSA mendapatkan rata-rata waktu yang paling lambat, algoritme tersebut masih dapat dipecahkan oleh *known plaintext attack* dibandingkan dengan ESRKGS yang mendapatkan rata-rata waktu lebih cepat daripada improvisasi RSA dan tidak dapat dipecahkan.

F. Pengujian serangan faktorisasi Fermat

Pengujian serangan faktorisasi Fermat dilakukan untuk menentukan hasil faktorisasi dan mendapatkan kunci privat dari kedua algoritme. Skenario pengujian sedikit berbeda dari pengujian *known plaintext attack*, yaitu pada pengujian ini hanya membutuhkan kunci publik. Kunci publik terdiri dari nilai *e* dan *n* yang digunakan untuk mendapatkan nilai *p*, *q*, dan *d* untuk algoritme RSA standar dan *p*, *q*, *r*, *s*, dan *d* untuk improvisasi RSA dan ESRGKS RSA.

Hasil pengujian terhadap serangan faktorisasi Fermat ditunjukkan pada Tabel 6. Algoritme RSA, improvisasi RSA, dan ESRKGS+RSA masih dapat diserang dengan metode ini. Hal ini disebabkan oleh kecilnya bilangan prima yang dihasilkan. Namun, pada ESRKGS+RSA nilai kunci privat yang ditemukan tidak sesuai dengan kunci privat yang sebenarnya. Perhitungan $\Phi(n)$ pada algoritme 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.

Hal lain yang menyebabkan algoritme improvisasi RSA dan ESRKGS+RSA masih dapat diserang oleh metode faktorisasi Fermat 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 algoritme ini. Nilai *n* yang relatif kecil menyebabkan ketiga algoritme mudah diserang. Nilai *n* yang lebih besar membuat faktorisasi *p* dan *q* lebih sulit dilakukan [4], [5]. Dengan kata lain, dengan *n* kecil ketiga algoritme masih dapat diserang dengan menggunakan metode faktorisasi Fermat walaupun nilai *d* pada ESRKGS+RSA tidak sesuai. Namun, penambahan jumlah bilangan prima pada improvisasi RSA dan ESRKGS mampu membuat tingkat keamanannya lebih baik seperti dinyatakan [10]-[12].

Hal ini juga bersesuaian [6] yang menyatakan bahwa nilai *n* masih dapat difaktorkan dengan metode faktorisasi Fermat walaupun nilai *n* yang ada di dalam algoritme RSA sudah dimodifikasi sedemikian rupa untuk mempersulit dalam memfaktorkan nilai *n*. Namun, walaupun nilai *n* dapat difaktorkan dalam ESRKGS+RSA, pasangan kunci privat yang diperoleh tidak sesuai dan proses dekripsi tidak dapat dijalankan dengan kunci ini. Hal ini dapat disebabkan karena hubungan antara kunci publik yang ditemukan dari $gcd(E, \Phi(N) * E_1) = 1$ dan kunci privat dari $D = E^{-1} \bmod (\Phi(N) * E_1)$ yang sudah dibangkitkan dalam metode ESRKGS mampu menyembunyikan hubungan yang terjadi di dalam proses pembangkitan kuncinya.

Tabel 6. Perbandingan hasil pengujian menggunakan faktorisasi Fermat

Algoritme	Bit	Waktu eksekusi (ms)	Status kunci privat	Keterangan
RSA Standar	16	3,250	Ditemukan	Kunci privat sesuai
	32	4.810.694,563	Ditemukan	Kunci privat sesuai
Improvisasi RSA	16	47,359	Ditemukan	Kunci privat sesuai
	32	56.100.000,190	Belum ditemukan	Resource tidak mencukupi
ESRKGS+RSA	16	89,164	Ditemukan	Nilai bilangan prima sesuai, namun kunci privat tidak sesuai
	32	56.100.000,320	Belum ditemukan	Resource tidak mencukupi

IV. KESIMPULAN

Pola persamaan kombinasi dalam pembangkitan kunci ESRKGS mampu menyembunyikan hubungan kunci publik dan kunci privat dalam algoritme ESRKGS+RSA sehingga walaupun dengan kunci 16 bit dapat diperoleh kunci privat dengan metode faktorisasi Fermat, namun kunci tersebut tidak bisa digunakan untuk mendekripsi pesan di aplikasi pesan chat. Selain itu, ESRKGS+RSA mempunyai waktu pembangkitan kunci, proses enkripsi, dan dekripsi yang lebih cepat daripada improvisasi RSA yang mempunyai empat bilangan prima.

UCAPAN TERIMA KASIH

Penelitian ini didanai oleh Direktorat Penelitian dan Pengabdian Universitas Muhammadiyah Malang (DPPM-UMM). Terima kasih juga diberikan untuk Laboratorium Data Sains dan Rekayasa Perangkat Lunak Teknik Informatika Universitas Muhammadiyah Malang sebagai penyedia sarana prasarana penelitian.

DAFTAR PUSTAKA

[1] W. Stallings, *cryptography and network security principles and practice, 7th edition.* Pearson, 2016.

[2] S. Upadhyay, "Attack on RSA cryptosystem," *International Journal of Scientific and Engineering Research*, vol. 2, no. 9, pp. 1–4, 2011.

[3] M. Preetha and M. Nithya, "A study and performance analysis of RSA algorithm," *International Journal of Computer Science and*

Mobile Computing, vol. 2, no. 6, pp. 126–139, 2013.

[4] M. Shringirishi, A. K. Solanki, M. Gupta, Y. Singh, and P. Gupta, "Novel Approach to Factorize the RSA Public Key Encryption," *International Journal of Advanced Research in Computer Science*, vol. 6, no. 6, pp. 172–176, 2015.

[5] A. Fatima and R. R. Chaudhary, "Modified trial division algorithm using lagrange's interpolation function to factorize RSA public key encryption," *International Journal of Advance Research and Innovative Ideas in Education*, vol. 3, no. 3, pp. 1861–1865, 2017.

[6] B. R. Ambedkar, A. Gupta, P. Gautam, and S. S. Bedi, "An efficient method to factorize the RSA public key encryption," in *2011 International Conference on Communication Systems and Network Technologies*, Katra, India, Jun. 2011, pp. 108-111. doi: 10.1109/CSNT.2011.29

[7] A. Arief, "Implementasi kriptografi kunci publik dengan algoritma RSA-CRT pada aplikasi instant messaging," *Scientific Journal of Informatics*, vol. 3, no. 1, pp. 46–54, 2016. doi: 10.15294/sji.v3i1.6115

[8] M. Cai, "The design method of network chat system based on socket and cloud computing," in *2012 International Conference on Computer Science and Service System*, Nanjing, China, Aug. 2012, pp. 610–613. doi: 10.1109/CSSS.2012.157

[9] A. Aminudin, A. F. Helmi, and S. Arifianto, "Analisa kombinasi algoritma Merkle-Hellman Knapscak dan logaritma diskrit pada aplikasi chat," *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 5, no. 3, pp. 325–334, 2018. doi: 10.25126/jtiik.201853844

[10] N. Somani and D. Mangal, "An improved RSA cryptographic system," *International Journal of Computer Applications*, vol. 105, no. 16, pp. 18-22, 2014. doi: 10.5120/18461-9820

[11] A. H. Al-Hamami and I. A. Aldariseh, "Enhanced method for RSA cryptosystem algorithm," in *2012 International Conference on Advanced Computer Science Applications and Technologies*, Kuala Lumpur, Malaysia, Nov. 2012, pp. 402–408. doi: 10.1109/ACSAT.2012.102

[12] B. P. U. Ivy, P. Mandiwa, and M. Kumar, "A modified RSA cryptosystem based on 'n' prime numbers," *International Journal of Engineering and Computer Science*, vol. 1, no. 2, pp. 2319–7242, 2012.

[13] B. Swami, R. Singh, and S. Choudhary, "Dual modulus RSA based on Jordan-totient function," *Procedia Technology*, vol. 24, pp. 1581–1586, 2016. doi: 10.1016/j.protcy.2016.05.143

[14] N. Lal, A. P. Singh, and S. Kumar, "Modified trial division algorithm using KNJ-factorization method to factorize RSA public key encryption," in *2014 International Conference on Contemporary Computing and Informatics*, Mysore, India, Nov. 2014, pp. 992–995. doi: 10.1109/IC3I.2014.7019588