PENGAMANAN DATA DENGAN ALGORITMA RSA *(Rivest Shamir Adleman),* LZW *(Lempel Ziv Welch)* DAN EOF (*End Of File*) PADA DIVISI MARKETING RS KPJ MEDIKA BSD

Silvester Rian Ananta[1)], **Imelda**[2)]
[1,2] Program Studi Teknik Informatika, Fakultas Teknologi Informasi, Universitas Budi Luhur
Jl. Raya Ciledug, Petukangan Utara, Kebayoran Lama, Jakarta Selatan 12260
Telp. (021) 5853753, Fax. (021) 5866369
E-mail : silvester.rian3@gmail.com[1)], imelda@budiluhur.ac.id[2)]

Abstract

KPJ Medika Hospital BSD is a private general hospital in Bumi Serpong Damai South Tangerang. At KPJ Medika Hospital BSD each computer are connected in a local network, and local networks that exist in the hospital connected to the internet, which means is connected to the WAN (Wide Area Network). The problem that arises is when there are parties who are able to break into the security of the local network and steal important data that exist on each computer. With this condition, the hospital needed a solution related to data security issues. Therefore, this paper aims to provide a solution to secure data on hospital KPJ Medika BSD. Encryption and steganography is a solution of data security. Encryption is a data security with encryption methods to the data itself. The result of this encryption will be compressed before stegangrafi process. Steganography is the insertion of data into other media such as pictures, videos, and music. But in this study, the data is only inserted into the picture. Initially the data will be encrypted using RSA. Then data result of encryption will be compressed using LZW. Then inserted into the image using the EOF. The result is data on marketing division will be stored and safely obscured by the media images, only those who are entitled only to read the data on marketing division. Thus the important data on the marketing division at the hospital KPJ Medika BSD is more secure because the data has been encrypted by RSA cryptography, compressed by LZW and obscured by steganography EOF.

Keywords: *RSA, LZW, EOF, cryptography, compression, steganography, encryption*

Abstrak

Rumah Sakit KPJ Medika BSD merupakan Rumah Sakit umum swasta di kawasan Bumi Serpong Damai Tangerang Selatan. Pada Rumah Sakit KPJ Medika BSD tiap komputernya saling terhubung dalam jaringan lokal, dan jaringan lokal yang ada di Rumah Sakit terhubung dengan internet yang artinya terhubung dengan WAN (Wide Area Network). Permasalahan yang muncul adalah ketika ada pihak-pihak yang mampu membobol keamanan dari jaringan lokal dan mencuri data-data penting yang ada pada tiap komputer. Dengan kondisi demikian, rumah sakit membutuhkan solusi terkait masalah keamanan datanya. Oleh karena itu penulisan ini bertujuan untuk memberikan solusi pengamanan data pada rumah sakit KPJ Medika BSD. Enkripsi dan steganografi merupakan solusi dari pengamanan data. Enkripsi merupakan pengamanan data dengan metode penyandian terhadap data itu sendiri. Hasil enkripsi ini akan dikompresi sebelum proses stegangrafi. Steganografi merupakan penyisipan data ke dalam media lain seperti gambar, video, dan musik. Namun pada penelitian ini, data hanya disisipkan ke dalam gambar. Awalnya data akan dienkripsi menggunakan RSA. Kemudian data hasil enkripsi akan dikompresi menggunakan LZW. Lalu disisipkan ke gambar menggunakan EOF. Hasilnya data-data pada devisi marketing akan tersimpan dan tersamarkan dengan aman pada media gambar, hanya orang yang berhak saja yang dapat membaca data-data pada devisi marketing. Dengan demikian data-data penting pada devisi marketing di rumah sakit KPJ Medika BSD lebih aman karena data telah tersandi oleh kriptografi RSA, terkompresi oleh LZW dan tersamarkan oleh steganografi EOF.

Kata Kunci : *RSA, LZW, EOF, kriptografi, kompresi, steganografi, enkripsi*

1.PENDAHULUAN

Rumah Sakit KPJ Medika BSD merupakan Rumah Sakit umum swasta di kawasan Bumi Serpong Damai Tangerang Selatan. Hingga saat ini Rumah Sakit KPJ Medika BSD terus berkembang dan semakin meningkatkan pelayanannya dalam bidang kesehatan. Rumah Sakit KPJ Medika BSD juga merupakan Rumah Sakit rujukan BPJS Kesehatan kelas B di daerah BSD. Dari hasil riset dan pengamatan yang dilakukan terlihat kepadatan pasien di Rumah Sakit ini terlebih untuk pasien peserta BPJS Kesehatan. Hari minggu dan hari libur sekalipun tetap terlihat kepadatan di Rumah Sakit ini.

Pada Rumah Sakit KPJ Medika BSD tiap komputernya saling terhubung dalam jaringan lokal, dan jaringan lokal yang ada di Rumah Sakit terhubung dengan internet yang artinya

terhubung dengan WAN *(Wide Area Network)*. Permasalahan yang muncul adalah ketika ada pihak-pihak yang mampu membobol keamanan dari jaringan lokal dan mencuri data-data penting yang ada pada tiap komputer. Hal ini bisa dilakukan oleh pihak dari dalam Rumah Sakit sendiri ataupun oleh pihak luar karena jaringan lokal yang ada di Rumah Sakit terhubung dengan WAN.

Dari permasalahan yang ada, rumah sakit KPJ Medika BSD terutama pada divisi marketing membutuhkan solusi pengamanan data yang dapat membantu memberikan perlindungan dari serangan para *hacker* ataupun orang-orang yang tidak berhak mengetahui data-data tersebut. Di sini untuk mencoba menjawab dan memberikan solusi dari masalah keamanan data yang dialami oleh Rumah Sakit KPJ Medika BSD maka dibuatlah aplikasi pengamanan data Kriptografi, Kompresi dan Steganografi, namun hanya difokuskan pada divisi marketing, karena pada divisi ini terdapat data-data penting seperti data rincian perhitungan *margin*, data metode reveral pasien, data kebijakan menajemen, data *business development planning* juga data *budgeting and action plan*.

Tujuan penulisan ini adalah membantu memberikan solusi dari masalah yang dihadapi Rumah Sakit KPJ Medika BSD dalam keamanan data yang berbasis java *desktop* dengan mengimplementasikan algoritma kriptografi RSA, kompresi LZW dan steganografi EOF untuk mengamankan data. Manfaat dari dibuatnya aplikasi ini tentu saja untuk memberikan perlindungan terhadap data-data rumah sakit dari orang-orang yang tidak berhak.

Pada penelitian sebelumnya telah dibahas algoritma RSA yang dilakukan oleh Muhammad Arief, Fitriyani dan Nurul Ikhsan [1]. Algoritma RSA ini diimplementasikan pada aplikasi *file* transfer *client server based*. Pada penelitian ini RSA digunakan untuk mengamankan *file* yang di*upload* pada jaringan publik, jadi sebelum di*upload file-file* tersebut dienkripsi. Hanya *client* yang mempunyai kunci privat saja yang dapat membaca isi dari file yang di*upload* tersebut. Penelitian ini menghasilkan aplikasi pengamanan untuk transfer *file* di jaringan publik dengan mengorbankan kecepatan upload *file* tetapi mendapatkan keamanan yang berlipat ganda. Adapun kelebihan dari penelitian yang dilakukan adalah algoritma RSA dapat diterapkan pada FTP *client* dan dapat diimplementasikan ke semua *file*. Kekurangannya adalah waktu upload menjadi lebih lama karena harus melewati proses enkripsi dan juga *file* yang dapat dienkripsi hanya terbatas pada 9 Mb.

Selain itu telah dibahas pula penelitian tentang algoritma LZW oleh Fuja Suhastra [2] yang membahas tentang implementasi algoritma LZW pada berkas digital. Berlatar belakang semakin banyaknya kebutuhan informasi dari internet pada penelitian ini, LZW digunakan untuk mengkompres data-data digital, karena semakin kecil ukuran data maka akan semakin hemat ruang penyimpanan dan *bandwidth*. Hasil dari penelitian ini adalah semakin sering dan tidak bervariasinya suatu substring yang muncul atau digunakan di dalam penyusunan teks tersebut maka rasio pemampatannya semakin besar. Jadi tidak ada karakter atau simbol yang hilang ketika didekompresi. Kekurangan dari algoritma ini adalah jika karakter dan simbol yang dikompresi bervariasi maka semakin kecil rasio pengkompresannya.

Pada penelitian lain juga telah dibahas mengenai algoritma EOF oleh Sandro Sembiring [3], yang membahas tentang diterapkannya algoritma EOF untuk menyisipkan pesan teks pada gambar. Untuk alasan privasi pada pengiriman pesan teks yang dikirim melalui jalur komunikasi publik, aplikasi ini ditujukan untuk menyamarkan pesan teks yang akan dikirim oleh pengguna pertama dan akan diterima oleh pengguna ke dua dengan aman karena pesan disisipkan pada gambar. Dari hasil penelitian diperoleh aplikasi pengamanan pesan teks yang dapat menyamarkan pesan teks yang dikirim dalam gambar. Namun aplikasi ini memiliki kekurangan pada jumlah pesannya, karena jika pesan terlalu banyak maka akan merusak citra pada gambar yang disisipi dan citra gambar yang dihasilkan oleh aplikasi ini tidak tahan terhadap serangan yang bersifat merusak atau merubah gambar, seperti aplikasi pengolah gambar.

Namun pada penelitian sebelumnya belum ada yang membahas mengenai penggunaan algoritma RSA, LZW dan EOF secara bersamaan untuk pengamanan data. Oleh karena itu kontribusi penelitian ini mengamankan data menggunakan kriptografi RSA, kompresi LZW dan steganografi EOF khusus untuk divisi marketing. Dengan demikian manfaat penelitian ini adalah memberikan solusi pengamanan data secara berlapis yang ditujukan untuk divisi marketing rumah sakit KPJ Medika BSD.

2. Algoritma RSA, LZW dan EOF

Pada penelitian ini, ada beberapa langkah dalam melakukan *encode* dan *decode*. Pada saat proses *encode* atau pengamanan data, langkah pertama adalah membuat kunci publik dan kunci privat. Kunci publik digunakan untuk enkripsi data, dan kunci privat digunakakn untuk dekripsi data. Setelah membuat kunci, kemudian dilakukan proses enkripsi menggunakan algoritma RSA menggunakan kunci publik yang

telah dibuat. Langkah selanjutnya adalah mengkompresi data yang telah dienkripsi menggunakan algoritma LZW. Langkah terakhir adalah menyisipkan data yang telah dienkripsi dan dikompresi ke dalam gambar menggunakan algoritma EOF. Untuk mengembalikan data ke bentuk semula dimulai dengan mengeluarkan data dari dalam gambar menggunakan algoritma EOF. Lalu data ini di dekompres menggunakan algoritma LZW. Data yang telah di dekompres akan di dekripsi menggunakan algoritma RSA. Dekripsi yang dilakukan algoritma RSA membutuhkan kunci privat yang telah dibuat.

Kriptografi adalah sebuah ilmu yang mempelajari tentang penyembunyian huruf atau tulisan sehingga membuat tulisan tersebut tidak dapat dimengerti atau dibaca [1]. Kriptografi sudah dikenal sejak dulu, sejarah mencatat kriptografi sudah digunakan oleh bangsa mesir sejak 4000 tahun SM, mereka menggunakan *hierogylphcs* untuk menyembunyikan tulisan, *Hierogylphics* diturunkan dari bahsa Yunani hierogylphica yang berarti ukiran rahasia [4].

Algoritma Rivest Shamir Adleman (RSA)

Algoritma RSA dibuat oleh tiga orang peneliti dari MIT (*Massachussets Institute of Technology*) pada tahun 1976, yaitu Ron Rivest, Adi Shamir dan Leonard Adleman. RSA adalah salah satu kriptografi asimetris dimana kunci untuk proses enkripsi berbeda dengan kunci yang digunakan untuk dekripsi. Kunci untuk proses enkripsi disebut kunci publik, sedangkan kunci untuk proses dekripsi disebut kunci privat. Orang yang memiliki kunci publik dapat melakukan proses enkripsi tetapi yang dapat melakukan proses dekripsi hanyalah orang yang memiliki kunci privat. Kunci publik boleh dimiliki oleh sembarang orang, tetapi kunci privat hanya boleh dimiliki oleh orang yang berhak menerima data [5].

Untuk pembangkitan pasangan kunci RSA, digunakan algoritma sebagai berikut :

Dipilih dua buah bilangan prima sembarang yang besar, p dan q. Nilai p dan q harus dirahasiakan.
Hitung **n = p x q**. Besaran n tidak perlu dirahasiakan.
Hitung **m = (p – 1)(q – 1).**
Dipilih sebuah bilangan bulat sebagai kunci publik, disebut namanya **e**, yang relatif prima terhadap **m**.
e relatif prima terhadap **m** artinya faktor pembagi terbesar keduanya adalah 1, secara matematis disebut **gcd (e,m) = 1**. Untuk mencarinya dapat digunakan algoritma *Euclid*.
Dihitung kunci privat, disebut namanya **d** sedemikian agar **(d x e)** mod **m = 1**. Untuk mencari nilai **d** yang sesuai dapat juga digunakan algoritma *Extended Euclid*. Hasil dari algoritma tersebut diperoleh kunci publik adalah pasangan **(e,n)**, kunci privat adalah pasangan **(d,n)**, **n** tidak bersifat rahasia, namun ia diperlukan pada perhitungan enkripsi atau dekripsi.

Untuk algoritma enkripsi menggunakan RSA adalah sebagai berikut :
Plaintext *m* dinyatakan menjadi blok-blok $m_1, m_2, \ldots$ sedemikian sehingga setiap blok mempresentasikan nilai di dalam selang [0, n–1] dan setiap blok diubah ke dalam bentuk desimal.
Setiap blok *m* dienkripsi menjadi blok c dengan rumus $\mathbf{c = m^e \bmod n}$
Sedangkan untuk dekripsi, maka setiap blok *ciphertext c* didekripsikan kembali menjadi blok *m* dengan rumus $\mathbf{m = c^d \bmod n}$ setelah didapatkan m dalam bentuk decimal, perlu diubah ke dalam bentuk karakter untuk mengembalikan ke bentuk asli.

Algoritma Kompresi LZW

Algoritma Lempel-Ziv-Welch (LZW), dikembangkan oleh Abraham Lempel, Jacob Ziv, dan Terry Welch [2]. Dan dipublikasikan pada tahun 1984 oleh Terry Welch. Algoritma ini mereduksi jumlah *token* yang dibutuhkan menjadi 1 simbol saja. Simbol ini merujuk kepada *index* dalam kamus. LZW mengisi kamus ini dengan seluruh simbol alphabet yang dibutuhkan. 256 *index* pertama dari kamus akan diisi dengan karakter ASCII dari 0-255. Karena kamus telah diisi dengan semua kemungkinan karakter terlebih dahulu, maka karakter *input*an pertama akan selalu dapat ditemukan dalam kamus. Inilah yang menyebabkan *token* pada LZW hanya memerlukan 1 simbol saja yang merupakan *pointer* pada kamus.

Prinsip kerja LZW, dimulai dengan membaca karakter *input* satu persatu dan diakumulasi pada sebuah *srting* I. Lalu dilakukan pencarian dalam kamus, apakah terdapat *string* I. Selama *string* I ditemukan di dalam kamus, *string* ini ditambahkan dengan satu karakter berikutnya, lalu dicari lagi dalam kamus. Pada saat tertentu, menambahkan satu karakter x pada *string* I akan menyebabkan tidak ditemukan dalam kamus. *String* I ditemukan, tetapi *string* Ix tidak. Dalam tahap ini, algoritma akan menulis *index* dari *string* I sebagai *output*, menambahkan *string* Ix kedalam kamus, dan menginisialisasi *string* I dengan x, lalu proses dimulai lagi dari awal.

Kumpulan output yang berupa angka-angka inilah sebagai hasil kompresi. Pengorganisasian kamus juga diperlukan, misalnya seperti seberapa besar kamus yang disediakan, dan apa yang akan dilakukan jika kamus sudah penuh. Kelebihan algoritma ini yaitu cepat dalam implementasi dan kekurangannya kurang optimal karena hanya melakukan analisis terbatas pada data. Algoritma ini melakukan kompresi dengan menggunakan kamus, dimana fragmen-fragmen

teks digantikan dengan indeks yang diperoleh dari sebuah "kamus". Pendekatan ini bersifat efektif karena banyak karakter dapat dikodekan dengan mengacu pada *string* yang telah muncul sebelumnya dalam teks. Prinsip kompresi tercapai jika referensi dalam bentuk pointer dapat disimpan dalam jumlah bit yang lebih sedikit dibandingkan *string* aslinya [2].

Proses kompresi algoritma LZW dapat dijelaskan dengan rumus sebagai berikut:
Kamus diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'},P adalah karakter pertama dalam stream karakter dan C karakter berikutnya dalam stream karakter. Kemudian cek apakah *string* (P + C) terdapat dalam kamus? Jika ya, maka P = P + C, gabungkan P dan C menjadi *string* baru. Jika tidak, maka output sebuah kode untuk menggantikan *string* P dan tambahkan *string* (P + C) ke dalam kamus dan berikan nomor/kode berikutnya yang belum digunakan dalam kamus untuk *string* tersebut. Apakah masih ada karakter berikutnya dalam stream karakter ?
Jika ya, maka kembali ke langkah 2. Jika tidak, maka output kode yang menggantikan *string* P, lalu terminasi proses (stop).

Sebagai contoh, *string* "ANANTA" akan dikompresi dengan LZW. Isi kamus pada awal proses diset dengan tiga karakter dasar yang ada: "A", "N", "T", dapat dilihat pada Tabel 1.

Tabel 1 : Tabel Kompresi LZW

Encoder		**Kamus**	
Input	***Output***	***Index***	***Entry***
		1	A
		2	N
		3	T
AN	1	4	AN
NA	2	5	NA
ANT	4	6	ANT
TA	3	7	TA

Kolom *entry* pada tabel di atas diisi dengan semua karakter yang ada yaitu A, N, T dan pada kolom *index* diberi nomor secara berurutan mulai dari satu hingga akhir proses. Kemudian mulai lakukan pengecekan, apakah AN terdapat pada kamus, jika tidak maka gabungkan AN lalu masukan pada kamus sebagai *string* baru, masukan juga pada kolom *input* dan beri kode 1 pada output. Lakukan pengecekan lagi, apakah NA terdapat pada kamus, jika tidak maka *input* NA ke dalam kamus dan juga kolom *input*, lalu beri kode output 2. Cek lagi apakah AN terdapat pada kamus, jika yam aka gabungkan dengan karakter berikutnya menjadi ANT, masukan pada kamus dan kolom *input* lalu berikan output 4 karena *string* AN terdapat pada *index* ke 4. Cek lagi apakah TA ada didalam kamus, jika tidak maka masukan TA ke dalam kamus dan kolom *index* lalu beri output 3 karena *string* T berada pada *index* ke 3. Didapati output 1, 2, 4 dan 3 sebagai hasil dari proses kompresi. Yang tadinya 6 karakter setelah dikompresi menjadi 4 karakter.

Proses dekompresi pada LZW dilakukan dengan prinsip yang sama seperti proses kompresi.
Kamus diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'}. CW = kode pertama dari stream kode (menunjuk ke salah satu karakter dasar) kemudian lihat kamus dan output *string* dari kode tersebut (*string* CW) ke stream karakter. PW = CW ; CW = kode berikutnya dari stream kode. Lalu cek apakah *string* CW terdapat dalam kamus ? Jika ada, maka output *string* CW ke stream karakter, P = *string* PW, C = karakter pertama dari *string* CW dan tambahkan *string* (P+C) ke dalam kamus. Jika tidak, maka P = *string* PW, C = karakter pertama dari *string* PW dan Output *string* (P+C) ke stream karakter dan tambahkan *string* tersebut ke dalam kamus. Lakukan pengecekan lagi apakah terdapat kode lagi di stream kode ? Jika ya, maka kembali ke langkah 4 dan jika tidak, maka terminasi proses (stop). Tabel dekompresi dapat dilihat pada Tabel 2.

Tabel 2 : Tabel Dekompresi LZW

Kamus		***Encoder***	
Index	***Entry***	***Input***	***Output***
1	A		
2	N		
3	T		
4	AN	1	A
5	NA	2	N
6	ANT	4	AN
7	TA	3	T

Proses dekompresi dilakukan dengan mencocokan angka pada kolom input yang merupakan string hasil proses kompresi dengan angka pada kolom index, angka pada kolom index adalah kode untuk setiap karakter pada kamus. Seperti pada tabel di atas, pada kolom input pertama terdapat angka 1, dan angka 1 merupakan kode untuk karakter A, maka outputnya adalah A, 2 merupakan kode untuk karakter N maka outputnya adalah N, 4 merupakan kode untuk karakter AN maka outputnya adalah AN dan 3 adalah kode untuk karakter T maka outputnya adalah T. Hasil dekompresinya menjadi ANANTA.

Algoritma Steganografi EOF

Secara umum teknik steganografi menggunakan *redundant bits* sebagai tempat menyembunyikan pesan pada saat dilakukan kompresi data, dan kemudian menggunakan kelemahan indera manusia yang tidak sensitive sehingga pesan tersebut tidak ada perbedaan yang terlihat. Teknik EOF atau *End Of File* merupakan salah satu teknik yang digunakan dalam steganografi. Teknik ini digunakan dengan cara menambahkan data atau pesan rahasia pada akhir *file*. Perhitungan ukuran *file* yang telah

disisipkan data sama dengan ukuran *file* sebelum disisipkan data ditambah ukuran data rahasia yang telah diubah menjadi *encoding file* [6].

Proses metode *end of file*
Langkah-langkah encoding menggunakan metode End Of *File* adalah sebagai berikut [3] :

a. Proses encoding dimulai dengan pesan yang akan disisipkan. Pesan diubah kedalam bentuk biner dengan representasi 1 atau 0.
b. Kemudian disisipkan angka 1 didepan rangkaian biner tersebut.langkah selanjutnya rangkaian biner tersebut dikonversikan menjadi bilangan decimal dan menghasilkan sebuah bilangan yang dinamakan dengan m
c. Menghitung jumlah warna yang terdapat pada berkas RGB yang menjadi objek steganografi.dan akan menghasilkan sebuah bilangan.Bilangan tesebut dinamakan dengan n, maka apabila m>n!-1 maka pesan yang akan disisipkan berukuran terlalu besar sehingga proses penyisipan tidak dapat dilakukan.
d. Warna dalam palet warna diurutkan sesuai dengan urutan yang "natural".Setiap warna dengan format RGB dikonversikan kedalam bilangan integer dengan aturan (Merah * 65536 + Hijau * 256 + Biru). Kemudian diurutkan berdasarkan besar bilangan integer yang mewakili warna tersebut.
e. Setelah itu lakukan proses iterasi terhadap variable i dengan nilai i adalah dari 1 sampai n. Setiap warna dengan urutan n – i dipindahkan ke posisi baru yaitu m mod i , kemudian m dibagi dengan i.
f. Kemudian palet warna yang baru hasil iterasi pada langkah ke – 4 dimasukkan kedalm palet warna berkas RGB. Apabila ada tempat yang diisi oleh dua buah warna, maka warna sebelumnya yang menempati tempat tesebut akan digeser satu tempat ke samping.
g. Apabila ternyata besar dari palet warna yang baru lebih kecil dari 256 maka palet warna akan diisi dengan warna terakhir dari palet warna sebelumnya.
h. Kemudian berkas RGB akan dikompresi ulang dengan palet warna yang baru, untuk menghasilkan berkas yang baru dengan ukuran dan gambar yang sama, namun telah disisipi pesan.

Langkah-langkah proses decoding atau mengekstrak pesan dari citra RGB yang telah disisip pesan dengan metode End Of *File* adalah sebagai berikut:

a. Masukkan nomor sesuai dengan posisi setiap warna pada palet warna citra RGB yang telah disisipkan pesan.
b. Warna diurutkan berdasarkan konversi RGB ke nilai integer dengan rumus: (Merah * 65536 + Hijau * 256 + Biru).
c. m diberi nilai 0
d. Iterasi variabel i dari i+1 sampai n-1. m=m*(n-1) + posisi warna ke i iterasi variabel j dari i +1 sampai n-1 jika posisi warna ke j > nilai posisi warna ke-i, maka posisi warna ke i dikurangkan 1
e. Setelah nilai m diperoleh, maka nilai m dikonversikan kebilangan binari utuk memperoleh pesan asli kembali.

3. HASIL DAN PEMBAHASAN

Aplikasi ini dibuat dengan tujuh *form* menu, diantaranya adalah menu *key*, menu *encode*, menu *decode*, menu *log*, menu *about*, menu *help*, dan menu *login*. Pertama *user* harus *login* dahulu, pada saat *user* berhasil *login* maka akan masuk ke menu utama. Lalu *user* wajib membuat kunci pada menu *key* pada Gambar 1 sebelum melakukan *encode* data. Setelah berhasil membuat kunci, baru *user* bisa menggunakan menu *encode* seperti pada Gambar 2. Jika *user* ingin mengembalikan data ke bentuk aslinya maka dapat menggunakan menu *decode* seperti pada Gambar 3. *Encode* sendiri merupakan proses mengamankan suatu informasi dengan membuat informasi tersebut tidak dapat dibaca tanpa bantuan pengetahuan khusus, sedangkan *decode* adalah proses agar informasi yang sudah di*encode* dapat dibaca kembali [7].

Gambar 1: Tampilan Layar Form Generate Key

Gambar 2 : Tampilan Layar Form Encode

Gambar 3: Tampilan Layar Form Decode

Algoritma *Form Key*
Untuk membuat kunci seperti pada Gambar 1, maka langkah-langkahnya dapat dilihat pada Gambar 4. Dari Gambar 4 dapat dijabarkan cara menggunakan *form key* dan tahapan-tahapannya.

1) Masuk ke halaman K
2) *Input* Kunci & Lokasi Penyimpanan
3) *Input* pilih
4) *If* pilih = *Generate then*
5) *If* Kunci == null || Lokasi Penyimpanan == *null then*
6) Tampilkan pesan *isi semua form*
7) Kembali ke baris 2
8) *Else*
9) Proses *generate keys*
10) Tampilkan pesan Pembuatan Kunci Berhasil
11) Kembali ke baris 2
12) *End if*
13) *Else if* pilih = *Clear then*
14) Nama kunci == null || Lokasi Penyimpanan == null
15) *Else if* pilih = *Close then*
16) Kembali ke halaman MU
17) *Else*
18) Kembali ke baris 2
19) *End if*

Gambar 4: Algoritma Form Key

Algoritma *Form Encode*
Untuk melakukan proses *encode* seperti pada Gambar 2, maka langkah-langkahnya dapat dilihat pada Gambar 5. Dari Gambar 5 dapat dijabarkan cara menggunakan *form encode* dan tahapan-tahapannya.

1) Masuk ke halaman E
2) *Input* Pilih *image*, data, kunci publik, lokasi simpan
3) *Input* Pilih
4) *If* Pilih = *Encode Then*
5) *If image*, data, kunci publik, lokasi simpan == *null Then*
6) Tampilkan Pesan isi semua *form*
7) *Else*
8) Proses Enkripsi, Kompresi dan *Embed file*
9) Simpan *File*
10) Tampilkan pesan *Encode* data berhasil
11) Kembali ke baris 2
12) *Else If = Clear Then*
13) *image*, data, kunci publik, lokasi simpan == *null*
14) Kembali ke baris 2
15) *Else If = Close Then*
16) Kembali ke halaman MU
17) *Else*
18) Kembali ke baris 2
19) *End if*
20) *End if*
21) *End if*
22) *End if*

Gambar 5: Algoritma Form Encode

Algoritma Form *Decode*
Untuk melakukan proses *decode* seperti pada Gambar 3, maka langkah-langkahnya dapat dilihat pada Gambar 6. Dari Gambar 6 dapat dijabarkan cara menggunakan *form decode* dan tahapan-tahapannya.

1) Masuk ke halaman D
2) *Input* Pilih *image*, kunci privat, lokasi simpan
3) *Input* Pilih
4) *If* Pilih = *Decode Then*
5) *If image*, kunci privat, lokasi simpan == *null Then*
6) Tampilkan Pesan isi semua *form*
7) *Else*
8) Proses Dekripsi, Dekompresi dan Retreive *file*
9) Simpan *File*
10) Tampilkan pesan *Decode* data berhasil
11) Kembali ke baris 2
12) *Else If = Clear Then*
13) *image*, kunci privat, lokasi simpan == *null*
14) Kembali ke baris 2
15) *Else If = Close Then*
16) Kembali ke halaman MU
17) *Else*
18) Kembali ke baris 2
19) *End if*
20) *End if*
21) *End if*
22) *End if*

Gambar 6: Algoritma Form Decode

Uji Coba Program

Pengujian dilakukan pada *file* word (.doc, .docx), *file* excel (.xls, .xlsx), *file* pdf (.pdf), *file* text (.txt), pengujian ini bertujuan untuk melihat kinerja dari aplikasi yang dibuat. Hasil pengujian dapat di lihat pada Tabel 3-6. Pada Tabel 3 adalah daftar tipe data dan Gambar yang akan diuji lengkap dengan ukuran dan resolusi Gambarnya.

Tabel 3 : Data Uji Coba

No	Nama File	Ukuran File	Nama Gambar	Jenis Gambar	Ukuran Gambar	Resolusi Gambar
1	BUDGET 2013.xls	109 kb	img1	jpg	145 kb	72 ppi
2	BUDGET 2013.xlsx	69 kb	img2	jpg	47 kb	350 ppi
3	Kebijakan.doc	124 kb	img3	jpg	112 kb	96 ppi
4	Kebijakan.docx	71 kb	img4	jpg	92 kb	96 ppi
5	Steganography.pdf	108 kb	img5	jpg	393 kb	96 ppi
6	tutorial.txt	23 kb	img6	jpg	71 kb	96 ppi

Pada Tabel 4 adalah hasil uji coba aplikasi, pada baris pertama data yang diuji adalah *file* .xls yang akan disisipkan ke *image* / gambar .jpg, *file* .xls memiliki ukuran 109 kb dan gambarnya berukuran 145 kb. Dibutuhkan waktu 2,491 milidetik untuk melakukan proses *encode* pada data tersebut, dan data menjadi berukuran 466 kb dengan resolusi gambar tetap 72 ppi. Pengujian selanjutnya dapat dilihat pada Tabel 4.

Tabel 4 : Data Hasil Uji Coba *Encode*

No	Nama File	Nama Gambar	Ukuran *File*	Ukuran Gambar	Waktu Proses (Milidetik)	Ukuran Setelah *Encode*	Resolusi Setelah *Encode*	Status
1	BUDGET 2013.xls	img1.jpg	109 kb	145 kb	2,491	466 kb	72 ppi	Berhasil
2	BUDGET 2013.xlsx	img2.jpg	69 kb	47 kb	2,505	333 kb	350 ppi	Berhasil
3	Kebijakan.doc	img3.jpg	124 kb	112 kb	2,905	520 kb	96 ppi	Berhasil
4	Kebijakan.docx	img4.jpg	71 kb	92 kb	2,831	388 kb	96 ppi	Berhasil
5	Steganography.pdf	img5.jpg	108 kb	393 kb	6,092	859 kb	96 ppi	Berhasil
6	tutorial.txt	img6.jpg	23 kb	71 kb	0,348	165 kb	96 ppi	Berhasil
Rata-rata			84 kb	143.33 kb	3,365	455 kb		

Tabel 5 adalah tabel hasil uji coba *decode*, pada kolom nama pambar adalah data gambar-gambar yang sudah di*encode* pada Tabel 4. Pada baris pertama ada img1.jpg dengan ukuran 466 kb yang berisikan *file* .xls, untuk mengeluarkan *file* .xls ke bentuk semula dibutuhkan waktu 2,461 milidetik. Untuk pengujian selanjutnya dapat dilihat pada Tabel 5.

Tabel 5 : Data Hasil Uji Coba *Decode*

No	Nama Gambar	Ukuran Hasil *Encode*	Waktu Proses (Milidetik)	Ukuran Setelah *Decode*	Status
1	img1.jpg	466 kb	2,461	109 kb	Berhasil
2	img2.jpg	333 kb	2,762	69 kb	Berhasil
3	img3.jpg	520 kb	2,94	124 kb	Berhasil
4	img4.jpg	388 kb	3,0	71 kb	Berhasil
5	img5.jpg	859 kb	5,79	108 kb	Berhasil
6	img6.jpg	165 kb	1,5	23 kb	Berhasil
Rata-rata		**455 kb**	**2,612**	**84 kb**	

Uji coba juga dilakukan dengan menggunakan kunci privat yang salah seperti pada Tabel 6, pengujian ini bertujuan untuk mengetahui apakah gambar bisa di *decode* dengan kunci privat yang salah. Pada baris pertama ada *file* img1.jpg yang berisi data .xls, ketika di *decode* menggunakan kunci privat yang salah *file* .xls tetap bias dikeluarkan dari dalam gambar bahkan membutuhkan waktu lebih cepat daripada menggunakan kunci privat yang benar, namun *file* .xls ini tidak dapat dibuka, begitu pula dengan gambar-gambar lainnya pada Tabel 6.

Tabel 6: Data Hasil Uji Coba *Decode* Dengan Kunci Privat Salah

No	Nama Gambar	Ukuran Hasil *Encode*	Waktu Proses (Milidetik)	Ukuran Setelah *Decode*	Status
1	img1.jpg	466 kb	1,727	109 kb	Berhasil tetapi tidak bisa dibaca
2	img2.jpg	333 kb	1,975	69 kb	Berhasil tetapi tidak bisa dibaca
3	img3.jpg	520 kb	2,243	124 kb	Berhasil tetapi tidak bisa dibaca
4	img4.jpg	388 kb	2,324	71 kb	Berhasil tetapi tidak bisa dibaca
5	img5.jpg	859 kb	11,271	108 kb	Berhasil tetapi tidak bisa dibaca
6	img6.jpg	165 kb	0,966	23 kb	Berhasil tetapi tidak bisa dibaca
Rata-rata		**455 kb**	**3,908**	**84 kb**	

Evaluasi

Setelah selesai merancang, membuat, dan menguji program tiba saatnya untuk melakukan evaluasi dimana evaluasi program berguna untuk melakukan penilaian tentang program yang telah dibuat. Dalam melakukan evaluasi program didapati kelebihan dan kekurangan dari program tersebut. Adapun kelebihan program ini adalah dapat memberikan sistem pengamanan kunci yang terbaik dan memberikan pengamanan ganda, enkripsi dan steganografi. Tampilan aplikasi ini juga sederhana dan *user friendly* sehingga mudah dipahami, proses *encode* dan *decode*nya pun juga cepat dan data rahasia tersamarkan dengan aman pada media gambar. Sedangkan kekurangannya adalah ukuran *file* hasil *encode*nya masih tergolong besar. Ukuran hasil *encode* yang besar menimbulkan kecurigaan bahwa media gambar sudah disisipi data rahasia.

5. KESIMPULAN

Berdasarkan uji coba dari aplikasi ini, maka dapat diambil suatu kesimpulan bahwa keamanan data menjadi lebih terjamin dengan mengimplementasikan kriptografi RSA, kompresi LZW dan Steganografi EOF karena data akan tersandikan, terkompres dan tersamarkan pada media gambar, sekalipun pencuri menemukan gambar yang berisi data, pencuri sulit untuk membuka datanya karena harus melakukan *retrieve*, dekompresi dan dekripsi. Penggunaan kunci enkripsi dan dekripsi yang berbeda akan sangat menyulitkan pencuri data dalam melakukan pembobolan. Waktu yang dibutuhkan dalam melakukan proses *encode* dan *decode* sangat tergantung pada ukuran data yang akan diproses. Semakin besar data, semakin lama prosesnya dan semakin kecil data semakin cepat prosesnya. Untuk melakukan proses *encode* dan *decode* pada beberapa data berukuran rata-rata 84 kb membutuhkan waktu rata-rata 3,365 milidetik untuk *encode* dan mebutuhkan waktu rata-rata 2,216 milidetik untuk *decode*.

Selain itu ada juga saran-saran untuk pertimbangan dalam pengembangan aplikasi ini agar lebih baik, antara lain proses *encode file* pada aplikasi ini diharapkan dapat ditingkatkan kinerjanya agar tidak hanya dapat meng*encode file* seperti *word* (.doc, .docx), *file excel* (.xls, .xlsx), *file pdf* (.pdf), dan *file text* (.txt) namun *file* dokumen lainnya, proses *encode* dan *decode* tidak terbatas pada ukuran *file* 1 Mb dan waktu proses *encode* dan *decode* tidak terlampau lama dengan *hardware* yang lebih mumpuni dan ukuran *file* gambar hasil *encode* tetap sama seperti sebelum disisipi atau bahkan lebih kecil dengan algoritma lain yang lebih baik agar tidak mudah diketahui bahwa gambar tersebut disisipi data.

DAFTAR PUSTAKA

[1] Arief, Muhammad, Fitriyani & Nurul Ikhsan. (2015). Kriptografi Rsa Pada Aplikasi *File* Transfer *Client- Server Based*. Jurnal Ilmiah Teknolog Informasi Terapan, *I* (3), 45–51.

[2] Suhastra, Fuja, (2014), Implementasi Algoritma Kompresi Lampel Ziv Welch (Lzw) Pada Berkas Digital, Pelita Manajemen Budi Darma, 54–57.

[3] Sembiring, Sandro, (2013), Perancangan Aplikasi Steganografi Untuk Menyisipkan Pesan Teks Pada Gambar Dengan Metode End of *File*, Pelita Manajemen Budi Darma, 45–51.

[4] Ariyus, Dony. 2008. Pengantar Ilmu KRIPTOGRAFI Teori, Analisis dan Implementasi. Yogyakarta: ANDI.

[5] Rahajoeningroem, Tri & Muhammad Aria, (2011), Studi dan Implementasi Algoritma RSA untuk pengamanan Data transkip Mahasiswa, Majalah Ilmiah Unikom, *8* (1), 77–90.

[6] Martono & Irawan, (2013), Penggunaan Steganografi dengan Metode *End Of File* (Eof) pada Digital *Watermarking*, *Jurnal TICOM*, 229–235.

[7] Wijaya, Anderias Eko, (2012), Pembuatan Aplikasi Encode dan Decode Berbasis Web Menggunakan Algortima Base64 Untuk Konfirmasi Pengiriman Pin, STMIK Subang, 1–15.