PENINGKATAN KEAMANAN DATA TEKS TERENKRIPSI ALGORITMA LUCIFER MENGUNAKAN STEGANOGRAFI GIFSHUFFLE PADA CITRA

Dian Anggara¹, Abdul Sani Sembiring²

^{1,2} Program Studi Teknologi Informatika STMIK Budi Darma, Medan, Indonesia
Email: ¹Diananggara02@gmail.com, ²gurkiy@gmail.com

Abstrak

Peningkatan ketahanan terhadap algoritma yang digunakan untuk mengamankan pesan rahasia sangat perlu dilakukan. Hal ini disebabkan oleh kemampuan para penyerang yang mampu menghalalkan berbagai cara untuk memecahkan algoritma yang digunakan untuk mengamankan data. Algoritma lucifer memiliki banyak *variant* salah satunya adalah lucifer dengan 16-round *feistel network* yang beroperasi pada 128 bit *block* dan menggunakan 128-bit *key*. Komputasi 2^{36} *plaintext* dan 2^{36} kompleksitas dapat memecahkan *variant* ini. Penelitian ini menguraikan pemanfaatan algoritma gifshuffle untuk mengoptimalkan algoritma lucifer dalam mengamankan data. Algoritma gifshuffle dimanfaatkan untuk menyembunyikan pesan terenkripsi berdasarkan algoritma Lucifer ke dalam media gambar berformat gif. Berdasarkan pengujian yang dilakukan, diperoleh bahwa tidak didapatkan perbedaan yang signifikan terhadap citra asli dan citra stegano setelah dilakukan proses penyisipan pesan.

Kata Kunci: Kriptografi, Steganografi, Lucifer, Gifshuffle, pesan

Abstract

Increased resistance to the algorithm used to secure secret messages is very necessary. This is caused by the ability of the attackers who are able to justify various ways to solve the algorithm used to secure data. Lucifer algorithm has many variants, one of which is a lucifer with a 16-round feistel network that operates on 128 bit blocks and uses 128-bit keys. Computing 2^{36} plaintext and 2^{36} complexity can attack this variant. This study describes the use of the gifshuffle algorithm to optimize the Lucifer algorithm in securing data. The gifshuffle algorithm is used to hide encrypted messages based on the Lucifer algorithm into the gif format media. Based on tests conducted, it was found that there were no significant differences in the cover image and stegano image after the message insertion process was carried out.

Keywords: Cryptography, Steganography, Lucifer, Gifshuffle, message

1. PENDAHULUAN

Perkembangan teknologi dan informasi seperti sekarang ini kecepatan dalam memperoleh atau mengakses data sangatlah penting. Oleh karena itu, keamanan data dan informasi sangat diperlukan untuk menjaga kerahasiaan data sehingga orang lain tidak mudah untuk mengetahui makna aslinya. Algoritma Lucifer adalah algoritma yang memiliki banyak *variant* dalam pengembangannya. Salah satu jenis pengembangan terhadap algoritma Lucifer adalah dengan menggunakan 16-round *Feistel network* yang beroperasi pada 128 bit *block plaintext* dan menggunakan 128-bit *key*. Namun *variant* jenis ini masih dapat dipecahkan dengan menggunakan 2^{36} *plaintext* dan 2^{36} kompleksitas [1]. Sehingga data yang telah diamankan dapat diketahui oleh pihak lain yang tidak bertanggung jawab. Hal ini dapat menimbulkan terjadinya pemalsuan data, penghapusan data atau pencurian data, dan mengakibatkan kerugian bagi pihak-pihak yang data ternyata mudah diketahui oleh pihak lain.

Salah satu solusi yang dapat dilakukan untuk meningkatkan keamanan adalah dengan cara mengkombinasikan ilmu kriptografi dengan steganografi. Penelitian ini menguraikan bagaimana mengamankan sebuah teks tersandi berdasarkan algoritma Lucifer dengan menyembunyikannya pada sebuah citra berformat gif berdasarkan algoritma *gifshuffle*. Pemanfaatan teknik steganografi dapat meminimalkan kecurigaan pihak lain terhadap pesan yang telah terenkripsi karena keberadaan pesan tidak dapat diketahui secara langsung.

2. TEORITIS

2.1 Kriptografi

Kriptografi merupakan ilmu dan seni untuk menjaga kerahasiaan menggunakan teknik enkripsi dan dekripsi, yaitu dengan cara melakukan proses pengacakan teks asli (*plaintext*) dengan menggunakan kunci (*key*) menjadi teks acak atau simbol-simbol tertentu (*chiphertext*) sehingga makna dari teks asli tidak dapat diketahui oleh pihak lain [2][3][4].

2.1.1 Lucifer

Algoritma *block cipher* pertama yang menggunakan *feistel cipher* adalah algoritma lucifer. Horst Feistel adalah penemu algoritma lucifer yang kemudian dijadikan sebagai dasar algoritma DES. Proses enkripsi dapat dilakukan terhadap 128 bits dari *plainteks* setiap bloknya dan dengan jumlah kunci 128 bits. Proses enkripsi dilakukan dengan menggunakan fungsi-f di dalam jaringan *feistel* yang digunakan [5].

Langkah-langkah proses enkripsi algoritma lucifer :

1. *Plaintext* dibagi menjadi dua bagian yaitu *Left bit* (L0) dan *Right bit* (R0).
2. Lakukan perhitungan pertama dengan rumus :

$$R_{i+1} = L_i \oplus F(R_i, K_i) \quad (1)$$

untuk mencari nilai R dan hasilnya akan digunakan untuk R_1 di putaran kedua.

3. Untuk mencari nilai L menggunakan rumus :

$$L_{i+1} = R_i \quad (2)$$

4. Setelah setiap putaran kecuali yang terakhir, bagian kanan dan kiri *block* ditukar.

Proses di atas masing-masing diulang sebanyak 16 ronde. Berdasarkan uraian di atas, terlihat bahwa dalam pengimplementasiannya, algoritma lucifer telah menggunakan fungsi permutasi dan iterasi. Tujuan dari fungsi-fungsi tersebut agar dapat menyulitkan kriptanalisis dalam memecahkan *ciphertext* yang dihasilkan [6].

Key byte access schedule adalah pengaturan jadwal dalam pengaksesan bit kunci pada algoritma Lucifer, dimana panjang maksimal pada *Message Byte* adalah 8, dan C-I-D *round* memiliki kapasitas maksimal 16 *round*, serta *Key Byte* memiliki kapasitas maksimal 15 yang terhitung mulai dari 0 [6].

2.2 Steganografi

Steganografi merupakan ilmu dan seni yang digunakan untuk menyembunyikan pesan atau data rahasia kedalam media seperti citra digital atau gambar, suara, dan video sehingga pesan atau data rahasia tersebut tidak terdeteksi oleh indra manusia [7][8].

2.2.1 Gifshuffle

Gifshuffle adalah algoritma steganografi yang digunakan untuk menyembunyikan pesan atau data kedalam media citra dengan format GIF [9]. Ada dua macam aktifitas yang berbeda namun saling bersangkutan, pada algoritma *gifshuffle*, yaitu :

1. *Encoding*

Encoding merupakan proses penyisipan pesan atau data teks ke media, citra berformat GIF sebagai media untuk menyembunyikan pesan atau data teks tersebut. Hasil dari proses penyisipan pesan atau data disebut sebagai *stego image*.

2. *Decoding*

Decoding merupakan proses ekstraksi pesan atau pengambilan pesan dari objek penyembunyiannya, masukannya adalah citra berformat GIF tempat pesan atau data teks disembunyikan (*stego image*) dan keluarannya adalah pesan atau data teks. Proses ini dapat dikatakan sebagai proses pembalikan dari *encoding*.

2.2.2 Proses Algoritma Gifshuffle

Pemanfaatan *header file* GIF menggunakan palet warna sebagai media penyisipan pesan merupakan dasar dari algoritma *gifshuffle*. Aspek robustness dapat diwujudkan dengan memanfaatkan algoritma ini pada citra berformat GIF karena tidak menyebabkan terjadinya perubahan apapun. Algoritma ini akan melakukan perubahan posisi terhadap ke 256 palet warna dalam berkas citra berformat GIF.

Langkah-langkah dalam proses Algoritma *GifShuffle* :

1. Pesan yang akan disisipkan diubah kedalam bentuk biner.
2. Setelah diubah kedalam bentuk biner, maka ditambahkan angka "1" pada pesan yang telah diubah kedalam bentuk biner, lalu dikonversikan kedalam bentuk desimal. Namakan bilangan yang diperoleh ini sebagai M.
3. Lakukan perhitungan jumlah warna yang terdapat dalam citra GIF yang ingin disisipkan. N merupakan variabel nilai hasil penjumlahan warna yang terdapat pada citra GIF. Pesan tidak dapat disisipkan bila nilai $M > N!-1$.
4. Urutkan warna natural dalam palet warna. Setiap warna dengan format RGB dikonversikan ke bilangan integer dengan aturan (merah * 65536 + hijau * 256 + biru). Hasil proses aturan tersebut akan diurutkan dari nilai terkecil hingga nilai terbesar.
5. Lakukan perulangan terhadap variabel I, dengan nilai I dari 1 sampai N. Posisi baru dari nilai warna pada urutan N-i ditentukan dengan formula $M \bmod i$, kemudian M dibagi dengan i (dilakukan pemindahan posisi nilai warna).
6. Hasil langkah ke 5 akan dipelatkan menjadi palet warna berkas citra GIF. Bila terdapat tempat yang diisi dua buah warna, maka warna yang sebelumnya menempati tempat tersebut akan bergeser satu tempat kesamping.

3. ANALISA DAN PEMBAHASAN

Pada tahap analisa dilakukan pengumpulan informasi dan penguraian mengenai proses yang akan digunakan untuk membangun sistem peningkatan keamanan data teks terenkripsi algoritma Lucifer menggunakan steganografi Gifshuffle. Dalam hal ini akan diuraikan tentang proses penyisipan data teks terenkripsi algoritma Lucifer ke suatu media menggunakan steganografi dengan algoritma Gifshuffle. Sebelum dilakukannya proses penyisipan, terlebih dahulu dilakukan proses enkripsi pada pesan atau data teks menggunakan algoritma Lucifer untuk menjaga keaslian dari pesan tersebut. Sehingga pesan atau data teks yang telah dienkripsi tersebut akan disisipkan ke suatu media citra GIF menggunakan steganografi Gifshuffle.

Berkaitan dengan hal di atas, tahapan yang akan dilakukan adalah melakukan penginputan. Dalam hal penginputan, data yang akan dimasukkan yaitu *Plaintext* dengan panjang 128 bit, Citra GIF dengan ukuran sampel

10x13, dan *key* dengan panjang 128 bit. Berikutnya dilakukan proses pembentukan kunci, lalu *plaintext* dienkripsi dengan algoritma Lucifer menggunakan *key* yang telah dibentuk. Ketika proses enkripsi selesai dan menghasilkan *ciphertext* maka proses selanjutnya yaitu melakukan *encoding* atau penyisipan, sehingga setelah dilakukannya proses *encoding*, maka keluaran atau *output* yang dihasilkan berupa *StegoImage*.

Data yang akan dienkripsi berupa teks dengan panjang *plaintext* 128 bit dan dengan *key* atau kunci dengan panjang 128 bit. Proses yang dilakukan untuk menghasilkan *ciphertext* berdasarkan algoritma lucifer, diuraikan pada contoh berikut.

Plaintext = STMIKBUDIDARMA10

Key = DIAN_ANGGARA1910

Sebelum dilakukannya proses pengenkripsian, maka terlebih dahulu yang akan dilakukan adalah *Key Byte Access Schedule* atau pengaturan jadwal dalam pengaksesan kunci pada algoritma Lucifer. Adapun proses dari penjadwalan pengaksesan kunci adalah sebagai berikut.

Konversi kunci ke biner :

Tabel 1. Tabel Konversi Kunci Ke Biner

Char	Biner	Nomor Key Byte
D	01000100	0
I	01001001	1
A	01000001	2
N	01001110	3
–	01011111	4
A	01000001	5
N	01001110	6
G	01000111	7
G	01000111	8
A	01000001	9
R	01010010	10
A	01000001	11
1	00110001	12
9	00111001	13
1	00110001	14
0	00110000	15

Setelah dilakukan konversi kunci ke bilangan biner, tahap berikutnya adalah mengatur pengaksesan kunci dimana panjang maksimal *Message Byte* adalah 8, hal ini dilakukan untuk mengetahui kunci setiap *round* yang akan digunakan dalam proses enkripsi dan dekripsi. Urutan susunan *message byte* dari biner kunci harus mengikuti aturan tabel *Key Byte Access Schedule* pada algoritma Lucifer.

Tabel 2. Tabel *Key Byte Access Schedule*

index	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	7	8	9	10	11	12	13	14
3	14	15	0	1	2	3	4	5
4	5	6	7	8	9	10	11	12
5	12	13	14	15	0	1	2	3
6	3	4	5	6	7	8	9	10
7	10	11	12	13	14	15	0	1
8	1	2	3	4	5	6	7	8
9	8	9	10	11	12	13	14	15
10	15	0	1	2	3	4	5	6
11	6	7	8	9	10	11	12	13
12	13	14	15	0	1	2	3	4
13	4	5	6	7	8	9	10	11
14	11	12	13	14	15	0	1	2
15	2	3	4	5	6	7	8	9
16	9	10	11	12	13	14	15	0

Berikut ini adalah hasil penyusunan *message byte* kunci yang telah dikonversi menjadi biner:

Tabel 3. Tabel Hasil Proses *Message Byte Key*

Index	0	1	2	3	4	5	6	7
-------	---	---	---	---	---	---	---	---

0	01000100	01001001	01000001	01001110	01011111	01000001	01001110	01000111
1	01000111	01000111	01000001	01010010	01000001	00110001	00111001	00110001
2	00110001	00110000	01000100	01001001	01000001	01001110	01011111	01000001
3	01000001	01001110	01000111	01000111	01000001	01010010	01000001	00110001
4	00110001	00111001	00110001	00110000	01000100	01001001	01000001	01001110
5	01001110	01011111	01000001	01001110	01000111	01000111	01000001	01010010
6	01010010	01000001	00110001	00111001	00110001	00110000	01000100	01001001
7	01001001	01000001	01001110	01011111	01000001	01001110	01000111	01000111
8	01000111	01000001	01010010	01000001	00110001	00111001	00110001	00110000
9	00110000	01000100	01001001	01000001	01001110	01011111	01000001	01001110
10	01001110	01000111	01000111	01000001	01010010	01000001	00110001	00111001
11	00111001	00110001	00110000	01000100	01001001	01000001	01001110	01011111
12	01011111	01000001	01001110	01000111	01000111	01000001	01010010	01000001
13	01000001	00110001	00111001	00110001	00110000	01000100	01001001	01000001
14	01000001	01001110	01011111	01000001	01001110	01000111	01000111	01000001
15	01000001	01010010	01000001	00110001	00111001	00110001	00110000	01000100

Berdasarkan tabel di atas, maka diketahui *subkey* yang akan digunakan untuk melakukan proses enkripsi dan dekripsi pada setiap *round* adalah sebagai berikut :

$K_0 = 010001000100100101000001010011100101111010000010100111001000111$
 $K_1 = 0100011101000111010000010101001001000001001100010011100100110001$
 $K_2 = 001100010011000001000100010010010100000101001110010111101000001$
 $K_3 = 0100000101001110010001110100011101000001010100100100000100110001$
 $K_4 = 0011000100111001001100010011000001000100010010010100000101001110$
 $K_5 = 0100111001011111010000010100111001000111010001110100000101010010$
 $K_6 = 0101001001000001001100010011100100110001001100000100010001001001$
 $K_7 = 0100100101000001010011100101111101000001010011100100011101000111$
 $K_8 = 0100011101000001010100100100000100110001001110010011000100110000$
 $K_9 = 0011000001000100010010010100000101001110010111110100000101001110$
 $K_{10} = 0100111001000111010001110100000101010010010000010011000100111001$
 $K_{11} = 0011100100110001001100000100010001001001010000010100111001011111$
 $K_{12} = 0101111101000001010011100100011101000111010000010101001001000001$
 $K_{13} = 0100000100110001001110010011000100110000010001000100100101000001$
 $K_{14} = 0100000101001110010111110100000101001110010001110100011101000001$
 $K_{15} = 0100000101010010010000010011000100111001001100010011000001000100$

1. Proses Enkripsi Berdasarkan Kriptografi Algoritma Lucifer

Algoritma *lucifer* adalah algoritma *block cipher* pertama yang menggunakan *feistel cipher* yang terdiri dari 16 round. Kemudian awal dari *block plaintext* akan dibagi dua yaitu *left block* dan *right block*. Setelah itu *left block* dan *subkey* pada *round* itu akan dimasukkan ke *F-Function*, lalu di XOR kan dengan *right block*. Yang hasilnya menjadi setengah bagian *block* untuk *round* berikutnya. Pada setiap putaran *left block* dan *right block* akan ditukar kecuali pada putaran terakhir. *Ciphertext* didapatkan dengan mengkombinasikan nilai *left block* dan *right block* terakhir yang telah dihasilkan. Berikut ini adalah proses pengenkripsian algoritma Lucifer :

Plaintext adalah STMIKBUDIDARMA10, kemudian dikonversi ke biner seperti berikut ini.

Tabel 4. Konversi *Plaintext* Ke Biner

char	Biner	char	biner
S	01010011	I	01001001
T	01010100	D	01000100
M	01001101	A	01000001
I	01001001	R	01010010
K	01001011	M	01001101
B	01000010	A	01000001
U	01010101	1	00110001

$$\begin{array}{r} D \quad 01000100 \\ 0 \quad 00110000 \end{array}$$

Kemudian bagi dua *block* awal menjadi *left block* (L) dan *right block* (R).

$$L_0 = 01010011010101000100110101001001011010000100101010101000100$$

$$R_0 = 010010010100010001000001010100100101101010000010011000100110000$$

Setelah itu lakukan proses pengenkripsian sebanyak 16 *round* dengan menggunakan rumus berikut ini :

Rumus mencari nilai L adalah $L_1 = R_0$

Rumus mencari nilai R adalah $R_1 = L_0 \oplus F(R_0, K_0)$

Putaran 1 :

$$L_1 = 0100100101000100010000010101001001001101010000010011000100110000$$

$$R_1 = L_0 \oplus F(R_0, K_0)$$

$$\begin{array}{l} 0100100101000100010000010101001001001101010000010011000100110000 \\ 010001000100100101000001010011100101111010000010100111001000111 \oplus \\ = 0000110100001101000000000011100000100100000000011111101110111 \\ 010100110101010001001101010010010101010000100101010101000100 \\ R_1 = 01011110010110010100110101010101011001010000100010101000110011 \end{array}$$

Putaran 2 :

$$L_2 = 01011110010110010100110101010101011001010000100010101000110011$$

$$R_2 = L_1 \oplus F(R_1, K_1)$$

$$\begin{array}{l} 01011110010110010100110101010101011001010000100010101000110011 \\ 0100011101000111010000010101001001000001001100010011100100110001 \oplus \\ = 0001100100011110000011000000011100011000011100110001001100000010 \\ 0100100101000100010000010101001001001101010000010011000100110000 \\ R_2 = 01010000010110100100110101010101010101001100100010001000110010 \end{array}$$

Proses perhitungan seperti di atas dilakukan sebanyak 16 putaran sehingga diperoleh biner-biner *ciphertext* sebagai berikut :

$$\begin{array}{l} 00100011010100000100010101011110010100100111001001010100011110100111001010111000011100101 \\ 01110001011011010001000101000001001101 \end{array}$$

Biner *ciphertext* yang didapatkan merupakan biner hasil putaran L_{16} dan R_{16} , kemudian dikelompokkan menjadi 8 bit perkelompok agar dapat dikonversikan ke karakter. Berdasarkan tabel ASCII, maka diperoleh karakter *Ciphertext* adalah #PE_)9*=9\|DPM

2. Proses Encoding Berdasarkan Algoritma Gifshuffle

Berdasarkan hasil dari enkripsi algoritma Lucifer, maka karakter yang diambil untuk contoh penyisipan pesan adalah “P” ke dalam berkas GIF dengan jumlah palet warna sebanyak 6 buah, adalah sebagai berikut :

- Pesan yang akan disisipkan adalah “P” yang diubah ke dalam bentuk biner dengan pengkodean ASCII menghasilkan bilangan biner : 01010000. Agar nilai M ditemukan, maka disisipkan angka 1 di depan pada rangkaian biner sehingga: $m = 1 + 00100011 = 101010000 = 336$.
- Jumlah palet warna yang diambil untuk contoh penyisipan adalah 6 palet warna, maka apabila $291 > 6! - 1$, proses *encoding* dapat dilakukan.
- Urutan warna pada palet warna yang natural dapat dilihat pada gambar 1.

0	1	2	3	4	5
Warna A	Warna B	Warna C	Warna D	Warna E	Warna F
R G B	R G B	R G B	R G B	R G B	R G B
168 106 91	169 107 92	170 108 93	169 107 92	168 107 89	165 104 86
11037275	11103068	11168861	11103068	11037529	10840150

Gambar 1. Pengkonversian RGB ke Nilai Integer

Berdasarkan gambar 1 di atas menunjukkan bahwa nilai natural dari beberapa warna, nilai tersebut didapat dari besar nilai RGB. Berdasarkan nilai-nilai desimal tersebut maka untuk melakukan konversi kebilangan integer dapat dilakukan dengan rumus berikut :

$$(Red * 65536 + Green * 256 + Blue) = (168 * 65536 + 106 * 256 + 91)$$

Sehingga hasil nilai integernya adalah 11037275.

Susunanpalet warna setelah diurutkan berdasarkan besar nilai integer terkecil sampai terbesar dapat dilihat pada gambar berikut.

Gambar 2. Susunan Palet Warna Dari Terkecil Sampai Terbesar

- d. Iterasi variable i mulai dari 1 sampai n . Setiap warna indeks ke $n - 1$ dipindahkan keposisi baru dengan rumus $(m \bmod i)$, Kemudian m dibagi dengan i untuk menghasilkan indeks berikutnya.

1. Untuk $i = 1$

$$336 \bmod 1 = 0$$

$$m = 336, m = \frac{336}{1} = 336$$

Maka, warna indeks ke-5 dipindahkan pada posisi ke-0 pada susunan palet warna yang baru.

2. Untuk $i = 2$

$$336 \bmod 2 = 0$$

$$m = 336, m = \frac{336}{2} = 168$$

Maka, warna indeks ke-4 dipindahkan pada posisi ke-0 pada susunan palet warna yang baru.

3. Untuk $i = 3$

$$168 \bmod 3 = 0$$

$$m = 168, m = \frac{168}{3} = 56$$

Maka, warna indeks ke-3 dipindahkan pada posisi ke-0 pada susunan palet warna yang baru.

4. Untuk $i = 4$

$$56 \bmod 4 = 0$$

$$m = 56, m = \frac{56}{4} = 14$$

Maka, warna indeks ke-2 dipindahkan pada posisi ke-0 pada susunan palet warna yang baru.

5. Untuk $i = 5$

$$14 \bmod 5 = 4$$

$$m = 14, m = \frac{14}{5} = 2$$

Maka, warna indeks ke-1 dipindahkan pada posisi ke-4 pada susunan palet warna yang baru.

6. Untuk $i = 6$

$$2 \bmod 6 = 2$$

$$m = 2, m = \frac{2}{6} = 0$$

Maka, warna indeks ke-0 dipindahkan pada posisi ke-2 pada susunan palet warna yang baru.

- e. Setiap warna yang menempati indeks sebelumnya akan bergeser sekali ke kanan atau ke indeks berikutnya bila terdapat warna yang menempati index yang sama. Susunan palet warna yang baru sesuai aturan tersebut dapat dilihat pada gambar berikut ini :

Gambar 3. Susunan Palet Warna Yang Baru

4. IMPLEMETASI

Pengujian sistem merupakan tahap dalam mengidentifikasi hasil dari implementasi sistem aplikasi. Pengukuran kualitas citra hasil dari proses steganografi dapat dilakukan dengan membandingkan nilai MSE (*Mean Square Error*) dan PSNR (*Peak Signal to Noise Ratio*) antara citra asli dengan citra *stegano*. Semakin rendah nilai dari *mean square error* maka kualitas pada citra semakin baik, sedangkan kualitas citra *stegano* dikatakan baik jika nilai *peak signal to noise ratio* 40 db atau lebih. Berikut ini adalah hasil pengujian terhadap kualitas citra.

Tabel 4. Hasil Pengujian Terhadap Mutu Citra

Citra Cover	File Pesan	Citra Stego	MSE (db)	PSNR(db)
Gambar1.gif (20,4 KB)	File_enkripsi1.txt	Stegoimage1.gif (22,3 KB)	6,34	40,14
Gambar2.gif (17,9 KB)	File_enkripsi2.txt	Stegoimage2.gif (19,8 KB)	5,13	41,06
Gambar3.gif (17,3 KB)	File_enkripsi3.txt	Stegoimage3.gif (19,1 KB)	5,54	40,73

Berdasarkan tabel di atas dapat dilihat nilai rata-rata MSE adalah 5,67 db dan nilai rata-rata PSNR adalah 40,64. Hal ini menunjukkan semakin rendah nilai MSE maka semakin bagus kualitas citra yang dihasilkan dan mutu *stego-image* dikatakan baik jika nilai PSNR 40 db atau lebih.

5. KESIMPULAN

Adapun kesimpulan dari hasil penelitian ini adalah sebagai berikut:

1. Data teks dengan format txt dapat diamankan dengan teknik kriptografi algoritma Lucifer serta dapat disembunyikan ke dalam sebuah media berupa gambar berformat GIF dengan teknik steganografi.
2. Proses penyembunyian dan pengekstrakan data teks terenkripsi dapat dilakukan dengan menggunakan algoritma *Gifshuffle* pada *image* dengan citra GIF. Namun, citra penampung dalam Algoritma *Gifshuffle* tidak memiliki ketahanan yang baik terhadap gangguan karena apabila *stegoimage* mendapat gangguan (*rotating*, *cropping*, dan lain-lain) pesan tidak dapat diekstrak kembali.
3. Sistem yang dibangun dapat mempermudah proses pengamanan dan penyembunyian data teks ke dalam citra GIF.

6. REFERENCES

- [1] M. Mizan, "Studi Perbandingan Algoritma Simetri Lucifer dan Blowfish," Institut Teknologi Bandung, Bandung, 2010.
- [2] B. Prasetyo, "Kombinasi Steganografi Bit Matching dan Kriptografi DES Untuk Pengamanan Data," Universitas Diponegoro Semarang, Semarang, 2013.
- [3] S. Kromodimoeljo, Teori dan Aplikasi Kriptografi, : SPK IT Consulting, 2009.
- [4] F. Humendru and T. Zebua, "Implementation of Triple Transposition Vegenere Cipher Algorithm and Cipher Block Chaining for Encoding Text," Int. J. of Inf. And Computer Science, vol. 1, no. 2, pp. 21-31, 2017
- [5] I. B. Aroya and E. Biham, "Differential Cryptanalysis of Lucifer," Journal of Cryptology, vol. 9, no. 1, pp. 21-34, 1996.
- [6] B. Buchanan, "Feistel Ciphers," Medium Corporation, 16 September 2018. [Online]. Available: <https://medium.com/asecuritysite-when-bob-met-alice/feistel-ciphers-18b400f9e1a9>. [Accessed 28 April 2019].
- [7] D. Darwis, "Teknik Steganografi Untuk Penyembunyian Pesan Teks Menggunakan Algoritma *Gifshuffle*," Jurnal Teknoinfo, vol. 11, no. 1, pp. 1-6, 2017.
- [8] M.E. Laoli and T. Zebua, "Penerapan Metode Dynamic Cell Spreading (DCS) Untuk Menyembunyikan Teks Tersandi Pada Citra," Semin. Nas. Inov. dan Teknol., no. 3, pp.766-770, 2016
- [9] M. Khairani and S. Sembiring, "Analisis dan Implementasi Steganografi Pada Citra GIF Menggunakan Algoritma *Gifshuffle*," SNASTIKOM, pp. 9-14, 2013.