BAB IV

HASIL DAN PEMBAHASAN

A. Hasil enkripsi dan deskripsi

Proses enkripsi dimulai dengan mengubah file input gambar RGB menjadi susunan bit, Kemudian masukkan kunci Stegano dengan metode *vigenere cipher*, setelah itu masukkan pesan dan kunci *viginere chipper* lalu di enkripsi. Gambar,pesan dan kunci akan di tranformasi dengan metode *LSB (least Significant Bit)* menjadi gambar *stego object* dengan pesan tersembunyi di dalamnya.

1. *Proses Embeding dengan metode LSB (Least Significant Bit)*

   a. Konsep Dasar

   - Suatu citra dapat terdiri atas pixel, contoh citra 4 x 3 pixel.

PX1	PX2	PX3	PX4
PX5	PX6	PX7	PX8
PX9	PX10	PX11	PX12

   - Setiap pixel dipecah menjadi 3 komponen RGB (Red Green Blue). Di dalam coding, Array pixel direpresentasikan dalam bentuk array dengan tipe integer.

R	G	B	
1111 0000	1010 1111	1100	0

- Pada LSB, tiap bit terakhir dari R G B disisipi satu bit dari pesan, misal bit pesan= **1111 0110** Maka, PX1 akan menjadi=

R	G	B
1111 000**1**	1010 111**1**	1100 001**1**

Maka, PX2 akan mejadi=

R	G	B
0000 000**1**	1111 010**0**	101 **1**

b. Penjelasan Algoritma Embedding LSB

Input parameter dari methode di atas adalah kumpulan pixel suatu citra yang direpresentasikan dalam bentuk array integer dan pesan rahasia dalam bentuk String. Yang dilakukan program di atas pada intinya adalah melakukan looping pada tiap pixel, dan pada tiap pixel tersebut, disisipkan 1 bit pesan pada tiap channel R G B nya.

Proses inti penyisipan LSB itu sendiri ada pada coding di bawah :

```
tmp = (byte) ((((oneDPix[element] >> binary[channelIndex]) &
0xFF) & 0xFE) | ((msg[msgIndex] >> toShift[(shiftIndex++)%
toShift.length]) & 0x1));
```

Untuk memahaminya, kita pecah kode diatas menjadi 3 bagian :

- **(((oneDPix[element] >>*binary*[channelIndex]) & 0xFF) & 0xFE)**

Setiap pixel pada gambar di looping, misal loop pertama pada pixel PX1.

Kemudian PX1 dipecah dan dilooping menjadi 3 channel.

Karena PX1 berisi 11110000 101011111 11000010

R	G	B
1111 0001	1010 1111	1100 0010

Maka untuk menyisipkan bit pada channel R saja, bit Pada PX1 perlu digeser sebanyak 16(16 adalah isi dari array *binary*index ke 0) bit ke kanan, hal ini ditunjukkan dengan tanda operasi bitwise >> pada kode di atas. sehingga menjadi 00000000 0000000 11110001 . selanjutnya 1111 0001 perlu di netralkan bit nya dengan cara dilakukan operasi AND dengan 0xFF yang mana 0xFF itu adalah hexadecimal dari bit1111 1111. Sehingga : (1111 0001) & (1111 111) =1111 0001 (mengembalikan dirinya sendiri).

Kemudian hasilnya ini perlu dinetralkan bit terakhirnya,yaitu dijadikan 0dengan cara operasi AND lagi dengan 0xFE yang mana 0xFE itu adalah hexadecimal dari bit1111 1110.

Sehingga : (1111 0001)& (1111 1110)= 1111 000**0**

Maka dari point a kita mendapatkan **1111 0000**

- **((msg[msgIndex] >>*toShift*[(shiftIndex++)% *toShift*.length]) & 0x1))**

Bit pesan adalah **<u>1</u>111 0110,** maka untuk mengambil bit **1**yang pertama, maka perlu digeser sebanyak 7(7 adalah isi dari array *toShift*index ke 0) bit kekanan.

Sehingga menjadi **0000 0001**

lalu di netralkan bit paling kanan dengan 0x01. Yang mana 0x01 adalah hexadecimal dari bit 0000 0001.

Sehingga :(0000 0001)& (0000 0001)= **0000 0001**

maka dari point b kita mendapatkan **0000 0001**

- tmp = (**byte**) ((((oneDPix[element] >>*binary*[channelIndex]) & 0xFF) & 0xFE)
  | ((msg[msgIndex] >>*toShift*[(shiftIndex++)% *toShift*.length]) & 0x1));

Lalu pesan di atas di lakukan operasi OR (|),

Sehingga : 1111 000**0**(hasil point a) OR 0000 0001 (hasil point b) = 1111 000**1**

Satu *bit* pesan berhasil disisipkan pada channel R PX1, kemudian bit 1111 0001 itu dikonversi menjadi byte, nilainya 241 dan nilai ini dimasukkan ke var tmp.Program melanjutkan hal yang samapada channel kedua PX1 yaitu G, channel ketiga yaitu B, kemudian lanjut ke pixel kedua PX2 dan seterusnya. Hal ini terus terjadi hingga semua bit pesan disisipkan.

2. Pembentukan kunci Viginere Chiper

   a. Karakter ASCII yang dapat di print hanya mulai dari Decimal 32 sampai Decimal 127. startASCIIchar=32. Jumlah printable character (numberOfChar)=(127-32) = 95.

ASCII printable characters

Dec	Hex	Binary	Char-acter	Description
32	20	00100000	**Space**	**space**
33	21	00100001	**!**	exclamation mark
34	22	00100010	**"**	double quote
35	23	00100011	**#**	number
36	24	00100100	**$**	dollar
37	25	00100101	**%**	percent
38	26	00100110	**&**	ampersand
39	27	00100111	**'**	single quote
40	28	00101000	**(**	left parenthesis
41	29	00101001	**)**	right parenthesis
42	2A	00101010	*****	asterisk
43	2B	00101011	**+**	plus
44	2C	00101100	**,**	comma
45	2D	00101101	**-**	minus
46	2E	00101110	**.**	period
47	2F	00101111	**/**	slash
48	30	00110000	**0**	zero

49	31	00110001	**1**	one
50	32	00110010	**2**	two
51	33	00110011	**3**	three
52	34	00110100	**4**	four
53	35	00110101	**5**	five
54	36	00110110	**6**	six
55	37	00110111	**7**	seven
56	38	00111000	**8**	eight
57	39	00111001	**9**	nine
58	3A	00111010	**:**	colon
59	3B	00111011	**;**	semicolon
60	3C	00111100	<	less than
61	3D	00111101	=	equality sign
62	3E	00111110	>	greater than
63	3F	00111111	**?**	question mark
64	40	01000000	**@**	at sign
65	41	01000001	**A**	
66	42	01000010	**B**	
67	43	01000011	**C**	
68	44	01000100	**D**	
69	45	01000101	**E**	
70	46	01000110	**F**	
71	47	01000111	**G**	
72	48	01001000	**H**	
73	49	01001001	**I**	

74	4A	01001010	**J**	
75	4B	01001011	**K**	
76	4C	01001100	**L**	
77	4D	01001101	**M**	
78	4E	01001110	**N**	
79	4F	01001111	**O**	
80	50	01010000	**P**	
81	51	01010001	**Q**	
82	52	01010010	**R**	
83	53	01010011	**S**	
84	54	01010100	**T**	
85	55	01010101	**U**	
86	56	01010110	**V**	
87	57	01010111	**W**	
88	58	01011000	**X**	
89	59	01011001	**Y**	
90	5A	01011010	**Z**	
91	5B	01011011	**[**	left square bracket
92	5C	01011100	\	backslash
93	5D	01011101	**]**	right square bracket
94	5E	01011110	^	caret / circumflex
95	5F	01011111	**_**	underscore
96	60	01100000	`	grave / accent
97	61	01100001	**a**	
98	62	01100010	**b**	

99	63	01100011	**c**	
100	64	01100100	**d**	
101	65	01100101	**e**	
102	66	01100110	**f**	
103	67	01100111	**g**	
104	68	01101000	**h**	
105	69	01101001	**i**	
106	6A	01101010	**j**	
107	6B	01101011	**k**	
108	6C	01101100	**l**	
109	6D	01101101	**m**	
110	6E	01101110	**n**	
111	6F	01101111	**o**	
112	70	01110000	**p**	
113	71	01110001	**q**	
114	72	01110010	**r**	
115	73	01110011	**s**	
116	74	01110100	**t**	
117	75	01110101	**u**	
118	76	01110110	**v**	
119	77	01110111	**w**	
120	78	01111000	**x**	
121	79	01111001	**y**	
122	7A	01111010	**z**	
123	7B	01111011	**{**	left curly bracket

124	7C	01111100	|	vertical bar
125	7D	01111101	}	right curly bracket
126	7E	01111110	~	tilde
127	7F	01111111	DEL	delete

b. Konversi katakunci dan pesan ke decimal ASCII dan lakukan proses enkripsi

Konversi katakunci ke decimal ASCII															
Char katakunci	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**
Dec ASCII	49	50	51	52	49	50	51	52	49	50	51	52	49	50	51
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
startASCIIchar	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
idxkatakunci	17	18	19	20	17	18	19	20	17	18	19	20	17	18	19

Konversi pesan ke decimal ASCII															
Char pesan	h	a	l	o	,	a	p	a		k	a	b	a	r	?
Dec ASCII	104	97	108	111	44	97	112	97	32	107	97	98	97	114	63
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
startASCIIchar	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
idxpesan	72	65	76	79	12	65	80	65	0	75	65	66	65	82	31

Proses Enkripsi															
(idxpesan +	89	83	0	4	29	83	4	85	17	93	84	86	82	5	50

idxkatakunci) mod 95															
+startASCIIchar	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32	+ 32
idxchiper	121	115	32	36	61	115	36	117	49	125	116	118	114	37	82
Konversi ke ASCII	y	s		$	=	s	$	u	1	}	t	v	r	%	R

Chipertext = ys $=s$u1}tvr%R

c. Konversi kata kunci dan pesan ke decimal ASCII dan lakukan proses dekripsi

Konversi chipertext ke decimal ASCII															
Char chipertext	y	s		$	=	s	$	u	1	}	t	v	r	%	R
Dec ASCII	121	115	32	36	61	115	36	117	49	125	116	118	114	37	82
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
startASCII char	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
Idxchiper	89	83	0	4	29	83	4	85	17	93	84	86	82	5	50

Konversi katakunci ke decimal ASCII															
Char katakunci	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3
Dec ASCII	49	50	51	52	49	50	51	52	49	50	51	52	49	50	51

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
startASCIIchar	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
idxkatakunci	17	18	19	20	17	18	19	20	17	18	19	20	17	18	19	

Proses dekripsi																
idxpesan= (idxchiper - idxkey)	72	65	-19	-16	12	65	-15	65	0	75	65	66	65	-13	31	
Jika idxpesan < 0, idxpesan+95			76	79			80							82		
+startASCIIchar	72 + 32	65 + 32	76 + 32	79 + 32	12 + 32	65 + 32	80 + 32	65 + 32	0 + 32	75 + 32	65 + 32	66 + 32	65 + 32	82 + 32	31 + 32	
idxpesan	104	97	108	111	44	97	112	97	32	107	97	98	97	114	63	
Konversi ke ASCII	h	a	l	o	,	a	p	a		k	a	b	a	r	?	

pesan= halo,apa kabar?

3. *Proses Extracting dengan metode LSB (Least Significant Bit)*

Input parameter dari program LSB adalah kumpulan pixel suatu citra yang direpresentasikan dalam bentuk array byte. Yang dilakukan program di atas pada intinya adalah melakukan looping pada tiap array

byte, dan pada tiap array byte tersebut, diambil 1 bit pesan. Ketika sudah terkumpul 8 bit, kemudian dibentuk menjadi 1 character yang akhir nya digabung menjadi suatu kata/kalimat.

Proses inti extract LSB itu sendiri ada pada coding di bawah :

```
tmp = (byte) (tmp | ((oneDPix[i] <<toShift[shiftIndex% toShift.length])
&andByte[shiftIndex++ % toShift.length]));
```

Untuk memahaminya, kita pecah kode diatas menjadi 3 bagian.

a. (oneDPix[i] <<*toShift*[shiftIndex% *toShift*.length])

Setiap pixel pada gambar di looping, missal loop pertama pada pixel PX1.

PX1 yang sudah di embeddberisi 11110000**1** 10101111**1** 11000011**1**

Array OneDPix langsung menunjuk pada nilai RGB, bukan nilai perpixel PX1.

Jadi OneDPix[0]=1111 0001OneDPix[1]=1010 1111OneDPix[2]= 1100 0011

R	G	B
1111 0001	1010 1111	1100 0011

Maka untuk mengextract bit pada R saja, bit pada R perlu digeser sebanyak 7 (7 adalah isi dari array *toShift* index ke 0) bit ke kiri, (ditunjukkan dengan tanda << pada kode di atas). sehingga menjadi : **1**000 000 Maka dari point a kita mendapatkan **1**000 0000

b. andByte[shiftIndex++ % toShift.length])

isi array andByte adalah kumpulan bilangan hexadecimal.

{ (byte) 0x80, 0x40, 0x20, 0x10, 0x08,0x04,0x02,0x01};

Jika dikonversi ke bit, susunan bit nya adalah :

0x80= **1**000 0000

0x40=0**1**00 0000

0x20=00**1**0 0000

0x10=000**1** 0000

0x08=0000 **1**000

0x04=0000 0**1**00

0x02=0000 00**1**0

0x01=0000 000**1**

Sehingga untuk index pertama=0 dari array andByte, dari point b kita mendapatkan 1000 0000.

jika digabungkan point a dan point b adalah

((oneDPix[i] <<*toShift*[shiftIndex% *toShift*.length])

&*andByte*[shiftIndex++ % *toShift*.length]))

Operasi point a AND point b adalah :

1000 0000 AND 1000 0000 = 1000 0000 → kita anggap hasilny sebagai **c**

c. (tmp | ((oneDPix[i] <<*toShift*[shiftIndex% *toShift*.length]) &*andByte*[shiftIndex++ % *toShift*.length]))

Nilai tmp pada awalnya adalah 0x00=0000 0000.

lalu di operasi OR kan dengan **c**, menghasilkan (0000 0000) OR (10000000) = 1000 0000. Jadi tmp=1000 0000. Pada loop ke-2 dilanjutkan dengan G atau OneDPix[1], tmp menghasilkan 1100 0000. Pada loop ke-3 dilanjutkan dengan B atau OneDpix[2], tmp menghasilkan 1110 0000. Dan seterusnya, hingga lengkap 8 bit, yaitu 1111 0110**.** Dari bit ini di

konversikan ke byte. Dan dari byte tersebut di konversi ke character yang akan membentuk string pesan. Program akan terus membentuk string setiap 8 bit. Jika string yang terbentuk tidak mengandung penanda khusus awal pesan, maka program berhenti dan memberitahukan tidak ada pesan rahasia. Jika ditemukan tanda khusus awal pesan, maka pembentukan string diteruskan sampai ditemukan penanda khusus akhir pesan dan kemudian program berhenti, dan menampilkan pesan rahasia.

4. Simulasi enkripsi dan deskripsi viginere chipper

   a. Simulasi enkripsi

Pesan = halo,apa kabar?

Katakunci = 1234

Pengecekan panjang katakunci

Length(pesan)=15 character

Length(katakunci)=4 character

Karena panjang katakunci kecil dari panjang pesan, maka ulangi katakunci sebanyak

length(pesan)-length(katakunci) = 11

Sehingga katakunci=123412341234123

   b. simulasi deskripsi

katakunci=1234

chipertext=ys $=s$u1}tvr%R

Pengecekan panjang katakunci

chipertext=ys $=s$u1}tvr%R

Length(chipertext)=15 character

Length(katakunci)=4 character

Karena panjang kunci kecil dari panjang chipertext, maka ulangi kunci sebanyak

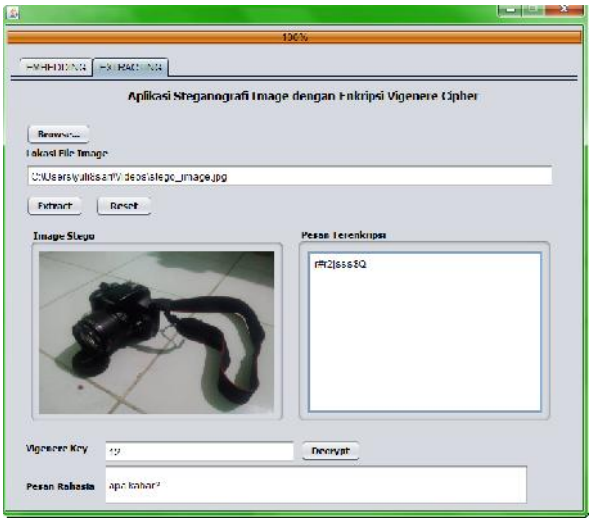
length(chipertext)-length(katakunci) = 11

katakunci=123412341234123

B. Pengujian Program

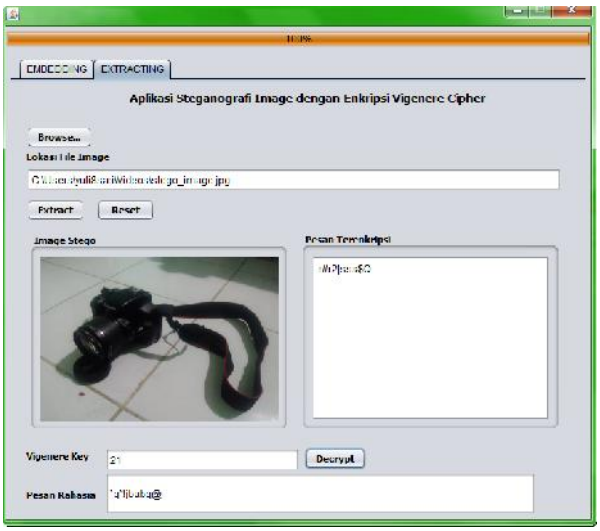
Pengujian program terhadap tingkat keberhasilan program dalam mengkonversi gambar, menyisipkan pesan dan mengunci gambar yang berisi pesan.

1. Pengujian diukur dari tingkat keamanan jika kode kunci sesuai dan tidak sesuai
2. Pengujian diukur dari jumlah karakter pesan dan jumlah karakter kode kunci.
   a. Pengujian jika kode kunci sesuai dengan gambar yang di extrak. Hasil pengujian : Pesan rahasia akan muncul jika kode kunci pada kolom *viginere key* benar.



Gambar 3.5 Pengujian benar

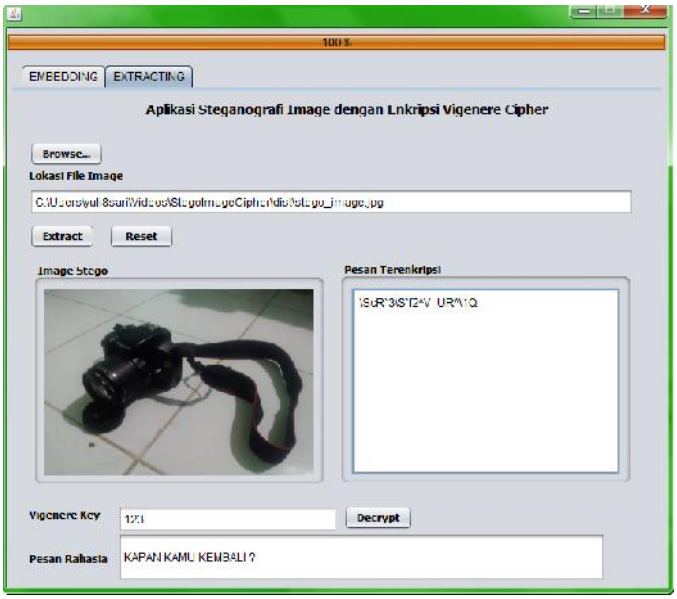
Pengujian jika kode kunci tidak sesuai dengan gambar yang di ekstrak. Hasil pengujian : Pesan rahasia akan muncul dengan kode *ascii* kebalikan dari kode kunci *viginere key*.



Gambar 3.6 Pengujian salah

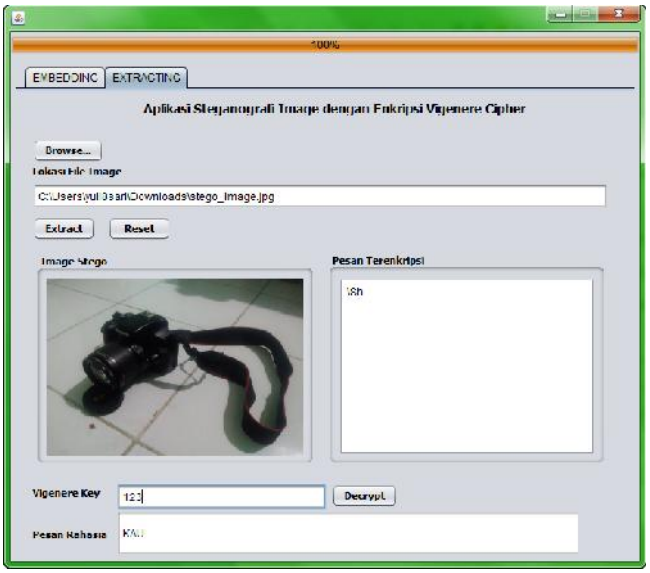
b. Pengujian jika jumlah pesan lebih banyak dari pada jumlah *viginere key*. Contoh => KAPAN KAMU KEMBALI ? => jumlah pesan = 20 karakter => *\ScR`3\S`f2^V_UR^\1Q* dan *viginere key* = 123, maka

panjang kode *ascii* pesan terenkripsi = 20 karakter. Hasil pengujian : Jumlah karakter Kode *ascii* pada jumlah pesan terenkripsi akan mengikut jumlah karakter pesan yang dimasukkan.



Gambar 3.7 Pengujian jumlah karakter

Pengujian jika jumlah *viginere key* lebih banyak dari pada jumlah pesan. Contoh => KAU=> jumlah pesan = 3 karakter, dan *viginere key* = 1234 maka panjang kode *ascii* pesan terenkripsi = 3 karakter.



Gambar 3.8 Pengujian jumlah key

pesan	key	*Encrypt*
KAU	12	\sf
KAU	123	\sh
KAU	1234	\sh

Pesan enkripsi mengikuti jumlah karakter pesan. Jadi walaupun key yg dimasukkan = 123 maka pesan akan tetap terbuka.

C. Hasil pengujian

Setelah melakukan pengujian baik pada sistem keamanan dan metode yang di pakai program tersebut dapat disimpulkan bahwa tingkat keamanan program sangat aman dan terintergrasi dengan baik. Tingkat kerahasiaan akan sangat baik jika tidak ada pihak ketiga yang mengetahui metode algoritma yang digunakan.

Setelah melakukan simulasi proses enkripsi dan deskripsi dapat disimpulkan bahwa tingkat keamanan program sangat aman dan terintergrasi dengan baik. Tingkat kerahasiaan akan sangat baik jika tidak ada pihak ketiga yang mengetahui metode algoritma yang digunakan.