PENERAPAN KRIPTOGRAFI MENGGUNAKAN ALGORITMA AES UNTUK DATA TEKS

Ahmad Reza.
Fakultas Teknik Informatika,Universitas Muhammadiyah Jember
Jalan Karimata No.49 Jember
E-mail : habsyreza3@gmail.com

Abstrak

Kompresi data dilakukan sehingga dapat menurunkan jumlah data yang akan dikirim hingga 50%, sehingga biaya transmisi dan pemakaian *bandwidth* internet juga akan mengecil. Pertukaran informasi melalui internet dapat menimbulkan dampak yang tidak diharapkan, misalnya data tersebut tanpa disengaja dimiliki oleh pihak atau seseorang yang tidak berhak mengakses data tersebut. Sehingga aspek keamanan dalam hal *privacy* merupakan hal yang penting. Kriptografi merupakan salah satu solusi atau metode pengamanan data yang tepat untuk menjaga kerahasiaan informasi dari orang yang tidak berhak mengaksesnya.

Kompresi Huffman merupakan algoritma kompresi yang cukup populer untuk kompresi data. Untuk memberikan pengamanan pada kompresi *Huffman*, digunakan metode kriptografi *Advanced Encryption Standard* (AES) 128. Proses kriptografi pada kompresi *Huffman* dilakukan pada Header *file* terkompresi saja. Hal ini dilakukan agar proses Kriptografi dapat lebih efisien dalam memberikan pengamanan pada hasil Kompresi.

Hasil penelitian menunjukkan bahwa algoritma *Advanced Encryption Standard* 128 bit dapat menyandikan isi *header* dari file terkompresi sehingga dapat mengamankan *file* tersebut. Sementara rasio hasil kompresi pada Kompresi yang mengimplementasikan metode Kriptografi AES menjadi sistem terpadu adalah sebesar 41,80% untuk file uji **.txt* dan 25,09% untuk file uji **.htm*

Kata Kunci : Kriptografi, enkripsi, dekripsi, *Advanced Encryption Standard*, kompresi Huffman

Abstract

Data compression can reduce the amount of data to be sent up to 50%, so the cost of transmission and bandwidth usage will also shrink. Transmission of information through the Internet pose a risk such as wrong transmission to person or party who did not have right to access the file. So the security aspects in the case of confidentiality or privacy is important. Cryptography is one of the solutions right to maintain the confidentiality of data,

Huffman compression is a compression algorithm that is quite popular for data compression. To provide security at Huffman compression, AES 128 cryptography method is used. Cryptographic process on Huffman compression performed on compressed file header only. This is done so that the process of cryptography can be more efficient provides security in data compression.

The results showed that 128 bit AES algorithm could encrypt the contents of the header of the compressed file, so the file can be secured. Compression ratio results in the implementation of the AES cryptographic methods and huffman is 41,80% *for *. Txt test files, and 25,09 % for *. htm test files.*

Keywords*: Cryptography, encryption, decryption, Advanced Encryption Standard, Huffman compression*

1. PENDAHULUAN

1.1 Latar Belakang

Teori informasi erat hubungannya dengan kompresi data. Kompresi data dilakukan sehingga dapat menurunkan jumlah data hingga dapat memberikan ukuran 50% dari total data awal. Kompresi data merupakan hal yang penting, sebab dengan ini seseorang yang mengkompresi data dapat menurunkan biaya; biaya terhadap penyimpanan data dan biaya untuk transmisi data. Dengan kompresi seseorang juga akan dapat menghemat ukuran memori penyimpanan, dimana dengan ini seseorang tersebut dapat menyimpan informasi atau data yang lain. Selain itu, proses kompresi juga akan dapat mempersingkat waktu dalam tujuan untuk mengirimkan data melalui internet[KAT-06].

Kompresi data secara umum dapat dibagi dalam dua macam, *lossy* dan *lossless.* Kompresi *lossy* akan menyebabkan data yang sudah dikompresi tidak dapat dikembalikan seperti data semula. Kompresi seperti ini digunakan untuk gambar, video atau suara dimana kehilangan (*loss*) data dapat diijinkan dalam kasus tertentu. Contoh dari kompresi *lossy* ini adalah JPEG dan GIF untuk gambar, MPEG untuk video dan MP3 untuk suara. Kompresi *lossless* adalah teknik kompresi dimana sama sekali tidak diijinkan perbedaan antara data awal (sebelum kompresi) dan data setelah dilakukan kompresi. Contoh program kompresi *lossless* seperti winzip, winrar, dan pkzip[KAT-06].

Kompresi *lossless* biasanya diimplementasikan dengan menggunakan dua jenis pendekatan yang berbeda: *statistical* dan *dictionary based*. Untuk pendekatan secara *statistical* ini, data akan dikompresi dengan menggunakan probabilitas kemunculan karakater. Salah satu algoritma yang paling terkenal untuk pendekatan secara *statistical* ini adalah algoritma *Huffman*[KAT-06].

Dengan berkembangnya jumlah data yang disimpan dalam komputer, kebutuhan akan keamanan transmisi dan reduksi terhadap penyimpanan data telah meningkat setiap hari. Salah satu solusi yang dapat digunakan dalam permasalahan tentang keamanan ini adalah dengan menerapkan Kriptografi [YUN-09] Kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan.

Jika kita hendak mengirimkan sebuah pesan kepada seseorang tanpa pesan tersebut dapat dibaca oleh pihak yang tidak berkepentingan, maka teknik Kriptografi dapat memberikan solusi[SCH-96].

Saat ini, AES (*Advanced Encryption Standard*) merupakan algoritma kriptografi yang cukup aman untuk melindungi data atau informasi yang bersifat rahasia. Pada tahun 2001, AES digunakan sebagai standar algoritma kriptografi terbaru yang dipublikasikan oleh NIST (*National Institute of Standard and Technology*) sebagai pengganti algoritma DES (*Data Encryption Standard*) yang sudah berakhir masa penggunaannya. Algoritma AES adalah algoritma kriptografi yang dapat mengenkripsi dan mendekripsi data dengan panjang kunci yang bervariasi, yaitu 128 bit, 192 bit, dan 256 bit.

Metode kompresi dan enkripsi yang ada umumnya melakukan langkah-langkah prosesnya secara berurutan. Maksudnya adalah, proses enkripsi baru akan dilakukan hanya ketika proses kompresi telah selesai dilakukan.

Pada jurnal berjudul "*Simultaneous Data Compression and Encryption*" yang ditulis oleh M. Soujanya dan T. Revanthi, dipublikasikan pada bulan Mei tahun 2011, diajukan sebuah pendekatan yang akan melakukan proses kompresi dan enkripsi sekaligus. Dasar utama dari pendekatan ini adalah dengan mengenkripsi isi dari *header* yang terbentuk dari proses algoritma Huffman. *Header* ini berupa informasi yang dibutuhkan dalam proses dekompresi nantinya. Dengan *header* yang telah teracak oleh proses enkripsi, maka otomatis proses dekompresi tidak akan dapat menghasilkan *file* seperti aslinya. Melalui pendekatan ini, waktu proses yang dibutuhkan komputer dalam melakukan proses enkripsi dapat dikurangi.

Berdasarkan hal-hal di atas, maka dalam penulisan skripsi ini akan dilakukan implementasi terhadap metode enkripsi AES 128 bit dan metode Huffman. Kedua metode ini akan digabungkan untuk menghasilkan

sistem terpadu yang dapat memberikan kemampuan dalam mereduksi ukuran data sekaligus menjamin keamanan data tersebut.

1.2 Rumusan Masalah

1. Bagaimana mengimplementasikan algoritma kriptografi AES 128 bit untuk pengamanan pada algoritma kompresi Huffman sehingga menjadi sebuah sistem terpadu.
2. Berapa tingkat rasio hasil kompresi sebuah *file* dengan mengimplementasikan algoritma AES dan algoritma Huffman.

1.3 Tujuan

- Mengimplementasikan algoritma kriptografi AES untuk pengamanan proses kompresi huffman sehingga menjadi sebuah sistem terpadu.
- Menghitung rasio kompresi *file* hasil kompresi dengan *file* sebelum kompresi.

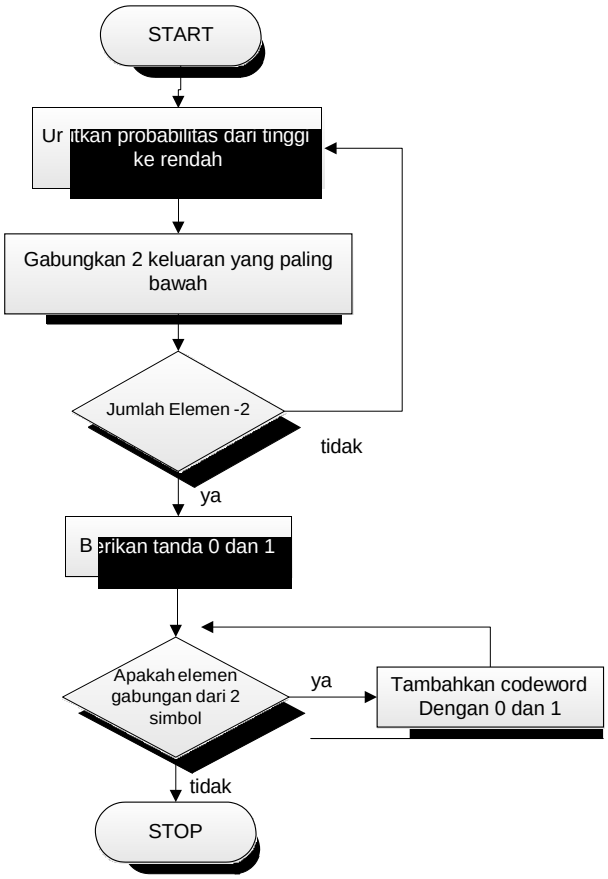
1.4 Batasan Masalah

a. Metode kompresi yang digunakan, yaitu metode kompresi Huffman.
b. Metode kriptografi yang digunakan, yakni algoritma kriptografi AES (*Advanced Encryption Standard*) 128 bit
c. Data yang digunakan dalam penelitian ini adalah *file* teks dengan format encoding UTF-8, seperti *.txt, dan *.html.
d. Bahasa pemrograman yang akan digunakan adalah bahasa pemrograman Java.

2. TINJAUAN PUSTAKA

2.1 Algoritma Huffman

Kode Huffman pada dasarnya adalah himpunan yang berisi sekumpulan kode biner yang direpresentasikan dari pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon biner diberikan label 0, sedangkan pada cabang kanan diberikan label 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan pengkodean untuk karakter yang berpadanan. Pohon biner ini biasa disebut pohon Huffman (*Huffman tree*). Langkah-langkah pembentukan pohon Huffman ditunjukkan gambar 1:



Gambar 1 Diagram alir pengkodean Huffman
Sumber :[ARI-06]

2.2 Proses Encoding

Proses untuk melakukan pembentukan kode dari suatu data tertentu disebut *encoding*. Dalam hal ini, kode Huffman akan terbentuk sebagai suatu kode biner. Kode Huffman didapatkan dengan membaca setiap kode dari daun simbol tersebut hingga ke akarnya. Ketika suatu kode Huffman telah dibentuk, suatu data dapat dengan mudah di-*encode* dengan cara mengganti setiap simbol menggunakan kode yang telah dibentuk [ARI-06].

2.3 Proses Decoding

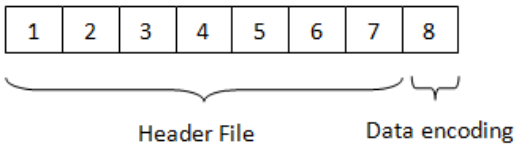
Decoding merupakan kebalikan dari *encoding*. *Decoding* berarti menyusun kembali data dari string biner menjadi sebuah karakter kembali. *Decoding* dapat dilakukan dengan dua cara, yaitu yang

pertama dengan menggunakan pohon Huffman dan yang kedua dengan menggunakan tabel kode Huffman.

2.4 Format Penyimpanan *File* Hasil Kompresi Huffman

File hasil pemampatan harus ditandai pada awal datanya dengan *header*, sehingga sewaktu pengembalian ke *file* asli dapat dikenali apakah *file* tersebut benar merupakan hasil pemampatan dengan algoritma Huffman.

Header file merupakan bagian dari data yang akan disimpan ke dalam *file* terkompresi. *Header file* berisi bagian-bagian yang berfungsi sebagai pedoman atau kamus dalam proses dekompresi nantinya. Adapun *header file* ditunjukkan pada gambar 2 :



Gambar 2 Mekanisme penyimpanan *file* terkompresi
Sumber :[CAL-07]

Keterangan :

1. Bagian satu berisi ekstensi *.HUF yang merupakan ekstensi dari *file* terkompresi.
2. Bagian dua berisi ekstensi dari *file* yang akan dikompresi.
3. Bagian tiga berisi jumlah karakter penyusun *file*.
4. Bagian empat berisi jumlah tambahan bilangan biner nol yang diperlukan untuk pengkonversian string biner ke dalam karakter ASCII.
5. Bagian lima berisi karakter yang terdapat pada tabel kompresi.
6. Bagian enam berisi kode Huffman dari karakter pada bagian empat.
7. Bagian tujuh berisi jumlah bit kode huffman karakter.
8. Bagian delapan merupakan data hasil *encoding.*

2.5 Advaced Encryption Standard

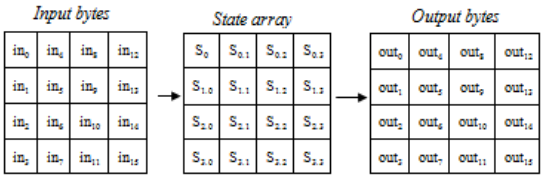
Input dan *output* dari algoritma AES terdiri dari urutan data sebesar 128 bit. Urutan data yang sudah terbentuk dalam satu kelompok 128 bit tersebut disebut juga sebagai blok data atau *plaintext* yang nantinya akan dienkripsi menjadi *ciphertext*. *Cipher key* dari AES terdiri dari *key* dengan panjang 128 bit, 192 bit, atau 256 bit. Perbedaan panjang kunci akan mempengaruhi jumlah round yang akan diimplementasikan pada algoritma AES ini. Berikut ini adalah Tabel 1 yang memperlihatkan jumlah round / putaran (Nr) yang harus diimplementasikan pada masing- masing panjang kunci.

Tabel 1 Perbandingan Jumlah Round dan Key [YUN-09]

	Jumlah Key (Nk)	Ukuran Block (Nb)	Jumlah Putara (Nr)
AES-128	4	4	10
AES-196	6	4	12
AES-256	8	4	14

Skrisi ini mengimplementasikan AES 12b bit.

Pada dasarnya, operasi AES dilakukan terhadap *array of byte* dua dimensi yang disebut dengan state. State mempunyai ukuran NROWS X NCOLS. Pada awal enkripsi, data masukan yang berupa in0, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14, in15 disalin ke dalam array state. State inilah yang nantinya dilakukan operasi enkripsi / dekripsi. Kemudian keluarannya akan ditampung ke dalam array out. Gambar 3 mengilustrasikan proses penyalinan dari input bytes, state array, dan output bytes

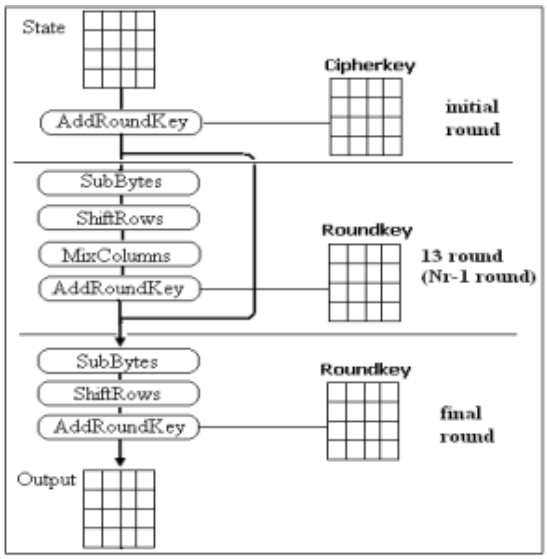


Gambar 3 State Array, input dan Output
Sumber : [FED-01]

2.5.1 Proses Enkripsi AES

Proses enkripsi algoritma AES terdiri dari 4 jenis transformasi bytes, yaitu

SubBytes, ShiftRows, Mixcolumns, dan *AddRoundKey*. Pada awal proses enkripsi, input yang telah dicopykan ke dalam *state* akan mengalami transformasi *byte AddRoundKey*. Setelah itu, state akan mengalami transformasi *SubBytes, ShiftRows, MixColumns,* dan *AddRoundKey* secara berulang-ulang sebanyak Nr. Proses ini dalam algoritma AES disebut sebagai round function. Round yang terakhir agak berbeda dengan round-round sebelumnya dimana pada round terakhir, state tidak mengalami transformasi *MixColumns*. Ilustrasi proses enkripsi AES dapat digambarkan seperti pada Gambar 4:



Gambar 4 Skema proses enkripsi AES
Sumber : [KUR-07]

Dalam *initial round*, transformasi *AddRoundKey*() dilakukan terhadap kunci utama. Sedangkan dalam 10 *round* yang lain, proses *AddRoundKey* dilakukan terhadap kunci putaran (*round key*). Proses *AddRoundKey* didefinisikan sebagai operasi XOR antara *array state* dengan *round key*. Operasi XOR dilakukan pada masing-masing *byte* dalam *array* sehingga menghasilkan nilai baru pada *array* hasil dengan ukuran *array* hasil sama dengan ukuran *array state* awal dan *array key*, yaitu sebesar 4x4. Hasil untuk masing-masing baris dan kolom pada *array state* hasil diperoleh dari hasil operasi XOR antara *array state* awal dengan *array key* untuk baris dan kolom yang sama.

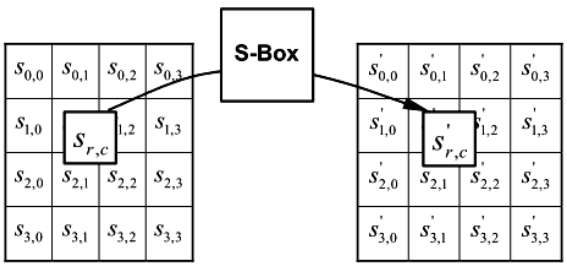
Transformasi *SubBytes*() memetakan setiap *byte* dari *array state* dengan menggunakan tabel substitusi *S-Box*. Tabel *S-Box* dapat dilihat pada Tabel 2.

Tabel 2: Tabel S-Box AES (FIPS 197, 2001)

HEX	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
x 5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Sumber : [FED-01]

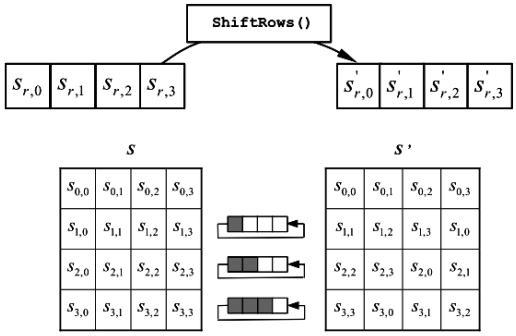
Cara pensubstitusian adalah sebagai berikut: untuk setiap byte pada *array*, misalkan S[r,c] = xy, yang dalam hal ini xy adalah digit heksadesimal dari nilai S[r,c], maka nilai substitusinya, yang dinyatakan dengan S'[r,c], adalah elemen di dalam *S-Box* yang merupakan perpotongan baris x dengan kolom y. Gambar 5 menunjukkan transformasi *SubBytes* (Munir, 2006).



Gambar 5 Transformasi *SubBytes*
Sumber : [FED-01]

Transformasi *ShiftRows*() melakukan pergeseran secara *wrapping* (siklik) pada 3 baris terakhir dari *array state*. Jumlah pergeseran bergantung pada nilai baris (r). Baris r = 1 digeser sejauh 1 byte, baris r = 2

digeser sejauh 2 *byte*, dan baris r = 3 digeser sejauh 3 *byte*. Baris r = 0 tidak digeser. Gambar 6 memperlihatkan transformasi *ShiftRows*.



Gambar 6 Transformasi *ShiftRows*
Sumber : [FED-01]

Transformasi *MixColumns*() dilakukan setelah transformasi *ShiftRows*, merupakan sumber utama dari difusi pada algoritma AES (www.wikipedia.org, 2012). Difusi merupakan prinsip yang menyebarkan pengaruh satu bit *plaintext* atau kunci ke sebanyak mungkin *ciphertext*. Sebagai contoh, pengubahan kecil pada *plaintext* sebanyak satu atau dua bit menghasilkan perubahan pada *ciphertext* yang tidak dapat diprediksi. Prinsip difusi juga menyembunyikan hubungan statistik antara *plaintext*, *ciphertext* dan kunci sehingga membuat kriptanalisis menjadi sulit (Munir, 2006).

Transformasi *MixColumns*() mengalikan setiap kolom dari *array state* dengan polinom $a(x)$ mod $(x^4 + 1)$. Setiap kolom diperlakukan sebagai polinom 4 suku pada *GF* (2^8). Polinom $a(x)$ yang ditetapkan pada persamaan 1

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (1)$$

Transformasi ini dinyatakan sebagai perkalian matriks seperti pada persamaan 2

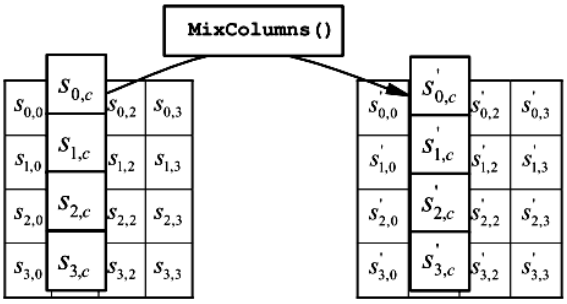
$$s'(x) = a(x) \otimes s(x) \quad (2)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Hasil dari perkalian matriks tersebut, setiap *byte* dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap *byte* tersebut pada persamaan 3

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (*02\} \bullet s_{3,c}) \quad (3)$$

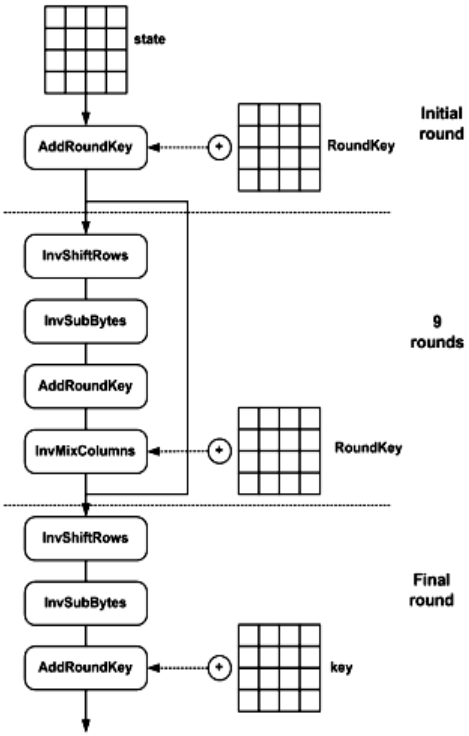
Gambar 7 memperlihatkan transformasi *MixColumns*



Gambar 7 Transformasi *MixColumns*
Sumber : [FED-01]

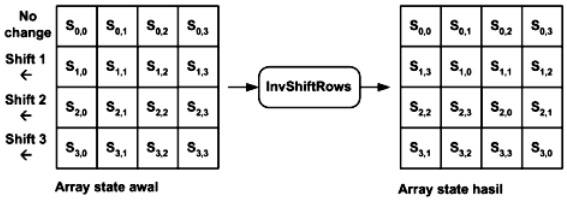
2.5.2 Proses Dekripsi AES 128

Transformasi cipher dapat dibalikkan dan diimplementasikan dalam arah yang berlawanan untuk menghasilkan *inverse cipher* yang mudah dipahami untuk algoritma AES. Transformasi *byte* yang digunakan pada invers cipher adalah *InvShiftRows*, *InvSubBytes*, *InvMixColumns*, dan *AddRoundKey*. Algoritma dekripsi dapat dilihat pada gambar 8:



Gambar 8 Skema global proses dekripsi AES 128
Sumber : [FED-01]

InvShiftRows adalah transformasi *byte* yang berkebalikan dengan transformasi *ShiftRows*. Pada transformasi *InvShiftRows*, dilakukan pergeseran bit ke kanan sedangkan pada *ShiftRows* dilakukan pergeseran bit ke kiri. Ilustrasi transformasi *InvShiftRows* terdapat pada gambar 9:



Gambar 9 Transformasi *InvShiftRows*
Sumber : [KUR-07]

InvSubBytes juga merupakan transformasi bytes yang berkebalikan dengan transformasi SubBytes. Pada InvSubBytes, tiap elemen pada state dipetakan dengan menggunakan tabelInverse S-Box. Tabel Inverse S-Box akan ditunjukkan dalam tabel 3:

Tabel 3 Tabel *Inverse S-Box*

HEX		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	D5	30	36	A5	38	Bf	40	A3	9e	81	F3	D7	Fb
	1	7c	E3	39	82	9b	2f	Ff	87	34	8e	43	44	C4	De	E9	Cb
	2	54	7b	94	32	A6	C2	23	3d	Ee	4c	95	0b	42	Fa	C3	4e
	3	08	2e	A1	66	28	D9	24	B2	76	5b	A2	49	6d	8b	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5c	Cc	5d	65	B6	92
	5	6c	70	48	50	Fd	Ed	B9	Da	5e	15	46	57	A7	8d	9d	84
	6	90	D8	Ab	00	8c	Bc	D3	0a	F7	E4	58	05	B8	B3	45	06
	7	d0	2c	1e	8f	Ca	3f	0f	02	C1	Af	Bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	Dc	Ea	97	F2	Cf	Ce	F0	B4	E6	73
	9	96	Ac	74	22	E7	Ad	35	85	E2	F9	37	E8	1c	75	Df	6e
	a	47	F1	1a	71	1d	29	C5	89	6f	B7	62	0e	Aa	18	Be	1b
	b	fc	56	3e	4b	C6	D2	79	20	9a	Db	C0	Fe	78	Cd	5a	F4
	c	1f	Dd	A8	33	88	07	C7	31	B1	12	10	59	27	80	Ec	5f
	d	60	51	7f	A9	19	B5	4a	0d	2d	E5	7a	9f	93	C9	9c	Ef
	e	A0	E0	3b	4d	Ae	2a	F5	B0	C8	Eb	Bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	D6	26	E1	69	14	63	55	21	0c	7d

Sumber : [FED-01]

Setiap kolom dalam state dikalikan dengan matrik perkalian dalam AES. Perkalian dalam matrik dapat dituliskan :

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0B & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Hasil dari perkalian matriks tersebut, setiap *byte* dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap *byte* tersebut pada persamaan 2.9

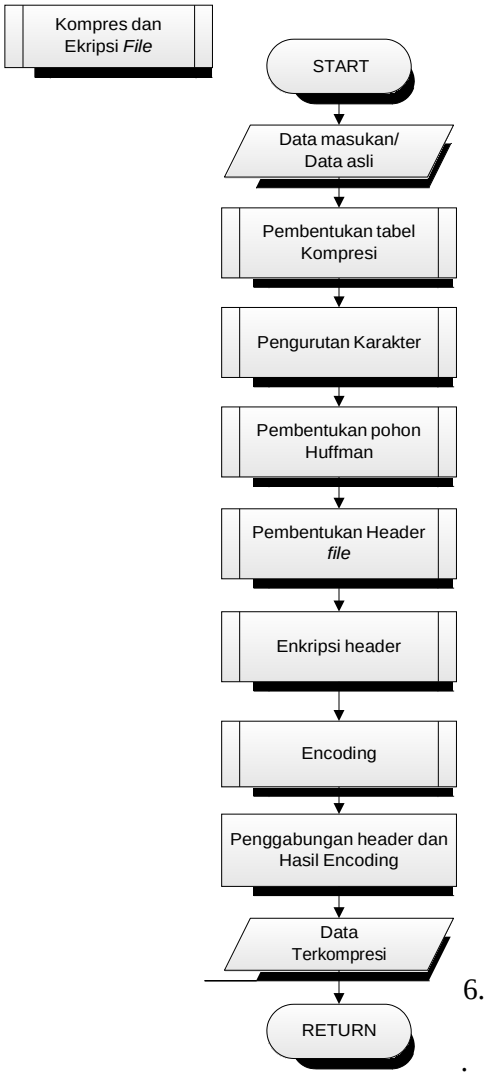
$$\begin{aligned} s'_{0,c} &= (\{0E\} \bullet s_{0,c}) \oplus (\{0B\} \bullet s_{1,c}) \oplus (\{0D\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0E\} \bullet s_{1,c}) \oplus (\{0B\} \bullet s_{2,c}) \oplus (\{0D\} \bullet s_{3,c}) \\ s'_{2,c} &= (\{0D\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0E\} \bullet s_{2,c}) \oplus (\{0B\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{0B\} \bullet s_{0,c}) \oplus (\{0D\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (*0E\} \bullet s_{3,c}) \end{aligned} \quad (2.9)$$

[FED-01]

3. PERANCANGAN SISTEM

3.1 Rancangan Proses Kompresi Data

Langkah-langkah yang dilakukan pada proses kompresi *file* ditunjukkan gambar 10



Gambar 10. Diagram Alir Proses kompresi dan Enkripsi *File*

Header file merupakan bagian dari data yang akan disimpan ke dalam data terkompresi. Header file berisi bagian-bagian yang berfungsi sebagai pedoman atau kamus dalam proses dekompresi nantinya. Adapun header file ditunjukkan pada gambar 11.

Gambar 11. Mekanisme susunan *file* Terkompresi

Keterangan:

1. Bagian satu berisi ekstensi Huf yang merupakan ekstensi dari file terkompresi.
2. Bagian dua berisi ekstensi dari file yang akan dikompresi.

Disediakan 3 *byte* ruang untuk menyimpan ekstensi dari file yang dikompresi.

3. Bagian tiga berisi jumlah karakter penyusun pohon Huffman.
4. Bagian empat berisi jumlah tambahan bilangan biner nol yang diperlukan untuk pengkonversian string biner hasil encoding ke dalam karakter ASCII.
5. Bagian lima berisi urutan karakter penyusun pohon Huffman, diikuti kode Huffman tiap-tiap karakter kemudian jumlah bit kode karakter Huffman tersebut.

Disediakan 2 *byte* ruang untuk menyimpan kode Huffman tiap-tiap karakter. Proses enkripsi akan memberikan hasil keluaran berkelipatan 16 *byte*, namun karena proses enkripsi tidak mesti dilakukan, maka header yang tidak di enkripsi tidak selalu menghasilkan panjang header yang berkelipatan 16 *byte*.

6. Data adalah merupakan string biner hasil *encoding.*

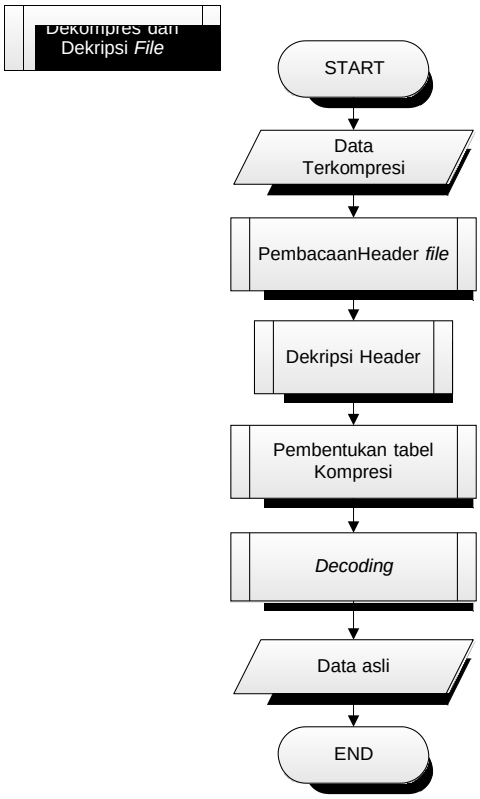
Setelah header file terbentuk, langkah selanjutnya adalah mengenkripsi header dari file yang dikompresi jika proses Kriptografi hendak dijalankan. Enkripsi header tidak dilakukan pada keseluruhan bagian header file. Enkripsi hanya akan dilakukan pada bagian 5 (gambar 11). Bagian 1,2,3 dan 4 pada gambar 3.2 jika dienkripsi akan menimbulkan ambiguitas dalam proses dekompresi nantinya.

Misalnya, jika jumlah karakter penyusun pohon huffman diacak, maka acuan

dalam inputan proses dekripsi akan menjadi berubah. Dimana hal ini secara otomatis dapat menyebabkan perubahan pada hasil dekompresi kedepannya.

3.2 Rancangan Proses Dekompresi Data

Proses dekompresi adalah pengembalian sebuah file yang terkompres menjadi seperti file aslinya. Proses dekompresi ini dilakukan oleh penerima pesan. Proses ini diawali dengan pembacaan file input yaitu berupa file terkompresi. Diagram alir proses Dekompresi dapat dilihat pada Gambar 12.



Gambar 12. Diagram alir proses Dekompresi *file*

Berikut ini merupakan penjelasan dari pembacaan header file :

1. 3 byte pertama dari header adalah ekstensi dari file tersimpan.
2. 3 byte berikutnya dibaca sebagai ekstensi dari file asli sebelum dikompresi
3. Byte selanjutnya adalah jumlah karakter pada file yang dikompresi.
4. Byte berikutnya adalah jumlah tambahan 0 yang diperlukan untuk pengkonversian string biner hasil encoding ke dalam karakter ASCII .
5. Kemudian Nbyte selanjutnya adalah merupakan header yang telah dienkripsi.

Jumlah N didapatkan dari,

N = (jumlah karakter x 4) + *temp*.

dimana,

temp = 16 - ((4 x jumlah karakter) % 16);

Misalnya, jumlah karakter adalah 9.

temp = 16 - ((4 x 9) % 16)
= 12.

Maka, jumlah panjang header yang harus diambil untuk proses dekripsi adalah sebanyak (9x4+12) = 48 byte. Nilai temp tidak akan lebih dari 16, dan temp = 16 akan dianggap sama dengan temp = 0.

Faktor pengali 4 didapatkan karena karakter, kode huffman, dan jumlah panjang kode huffman untuk masing-masing karakter memerlukan 4 byte ruang penyimpanan.

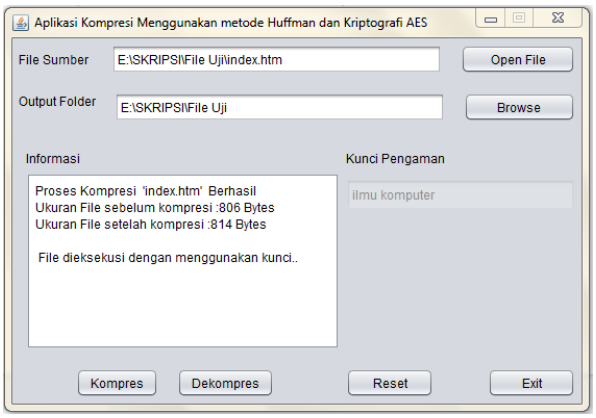
6. Bagian selanjutnya merupakan data hasil encoding.

4. IMPLEMENTASI DAN PEMBAHASAN

Berikut ini adalah implementasi dari rancangan proses-prosess yang telah dijelaskan sebelumnya.

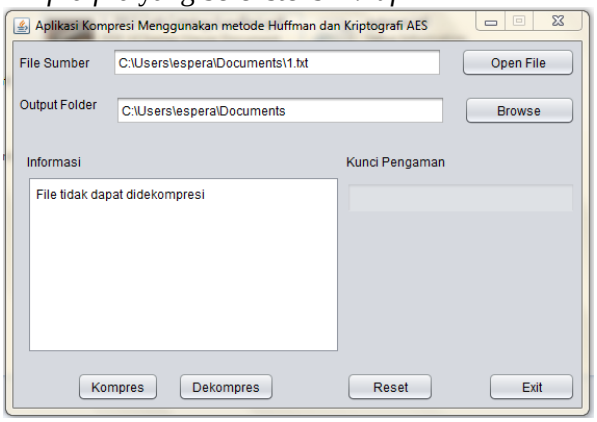
4.1 Implementasi Antarmuka

Gambar 14 menunjukkan proses kompresi telah selesai dilakukan dengan menggunakan kunci "**ilmu komputer**". Pada gambar dapat dilihat bahwa terdapat informasi mengenai berhasilnya *file* dikompresi, ukuran *file* sebelum dan setelah kompresi, serta keterangan mengenai penggunaan kunci.



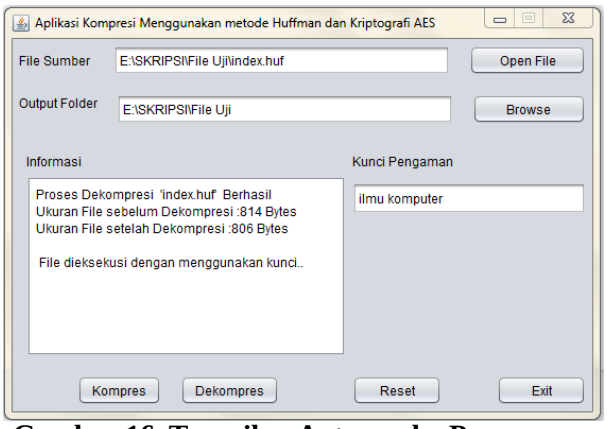
Gambar 14. Tampilan Antarmuka Proses Kompresi Selesai Dengan Kunci "ilmu komputer"

Jika yang menjadi *file* sumber adalah file dengan ekstensi selain **.huf*, maka proses dekompresi akan gagal dilakukan. Hal ini ditunjukkan pada gambar 15. Sistem hanya akan menjalankan proses dekompresi pada *file-file* yang berekstensi **.huf*



Gambar 15. Tampilan Antarmuka Proses Dekompresi Tidak Berjalan

Gambar 16 menunjukkan proses dekompresi dilakukan dengan menggunakan kunci "**ilmu komputer**". Ini berarti bahwa proses dekompresi akan menjalankan fungsi Kriptografi sebagai salah satu subprosenya.



Gambar 16. Tampilan Antarmuka Proses Dekompresi Selesai Dengan Kunci

4.3 Implementasi Uji Coba dan Evaluasi Hasil

4.3.1 Pengujian Algoritma Enkripsi AES

Penyerangan terhadap metode AES dengan mencoba-coba setiap peluang kunci yang ada pada saat dekompresi, sampai diketemukan padanan kata yang sesuai dengan *chipertext*. Hasil Uji coba ditunjukkan tabel 4.

Nama File (Ukuran *file*	**Kunci AES Untuk Kompresi**	**Lama waktu proses kompresi dan Enkripsi (ms)**	**Kunci AES Untuk Dekripsi dan Dekompresi**	**Lama waktu proses Dekripsi dan Dekompresi (ms)**	**Hasi Dekompresi**
Tutorial.txt (36.557 *byte*)	Ujian	11.559	Coba-coba saja	44.446	Tutorial.txt (0 *byte*). [File tidak dapat dibaca]
			Ujian	8.072	Tutorial.txt (36.557 *byte*). [File kembali ke bentuk asli]
	Ilmu komputer	12.641	Tes	44.532	Tutorial.txt (0 *byte*) File tidak dapat dibaca
			Ilmu komputer	7.968	Tutorial.txt (36.557 *byte*). [File kembali ke bentuk asli]
	abcdefghijklmnop	11.834	kuda	45.682	Tutorial.txt (0 *byte*) [File tidak dapat dibaca]
			Mata-mata	44.894	Tutorial.txt (0 *byte*) [File tidak dapat dibaca]
			abcdefghijklmnop	7.376	Tutorial.txt (36.557 *byte*) [File kembali ke bentuk asli]
Thinking Java.txt (50.541 *byte*)	Tupai	23.313	Tupai	14.722	Thinking Java(50.541 *byte*).txt [File kembali ke bentuk asli]
			Universitas 101	92.088	Thinking Java (0 *byte*).txt [File tidak dapat dibaca]
	9876543210654321	22.405	Kertas	91.842	Thinking Java(0 *byte*).txt [File tidak dapat dibaca]
			9876543210654321	14.476	Thinking Java(50.541 *byte*).txt [File kembali ke bentuk asli]
			Ilmu komputer UB	90.793	Thinking Java(0 *byte*).txt [File tidak dapat dibaca]
	Presiden 07	22.992	Demo	91.409	Thinking Java(0 *byte*).txt [File tidak dapat dibaca]
			Konspirasi tikus	92.359	Thinking Java(0 *byte*).txt [File tidak dapat dibaca]
			Presiden 07	14.143	Thinking Java(50.541 *byte*).txt [File kembali ke bentuk asli]

Tabel 4.Hasil Uji Coba percobaan Dekompresi pada *file* terenkripsi

Tabel 4.Peluang serangan *brute force* terhadap kunci

4.3.2 Analisa Hasil Percobaan Dekompresi pada file terenkripsi

Dari tabel 4, dapat ilihat bahwa faktor panjang kunci yang digunakan untuk mengenkripsi sebuah *file* yang sama, tidak berpengaruh secara signifikan terhadap lama waktu dari proses eksekusi program hingga selesai. Namun tentunya, semakin panjang kunci yang digunakan untuk kompresi, maka peluang *file* tersebut untuk di serang dengan metode Bruteforce akan menjadi lebih tinggi.

Perbedaan waktu kompresi yang signifikan terjadi jika file sumber memiliki perbedaan ukuran yang signifikan pula. Dari Tabel 4.1, dapat dilihat bahwa waktu eksekusi kompresi sebuah file akan berbanding lurus dengan ukuran file tersebut. Semakin besar file, waktu eksekusi pun bertambah.

Jika sebuah file yang telah terkompresi dan terproteksi dengan kunci tertentu hendak didekompresi dengan kunci yang salah, maka waktu yang dibutuhkan selama proses dekompresi lebih besar jika dibanding dengan proses dekompresi dengan menggunakan kunci yang benar.

4.3.3 Analisa Hasil Pengujian Kompresi

Proses kompresi terhadap sebuah data yang berukuran sangat kecil akan menghasilkan sebuah file dengan ukuran yang lebih besar dari aslinya. Hal ini disebabkan karena adanya tambahan header untuk sebuah file terkompresi.

Dari hasil pengujian kompresi pada file *.txt, proses kompresi akan efektif jika file yang hendak di kompresi memiliki ukuran minimal ±700 *Byte*. Namun jika ukuran file kurang dari 200 *Byte* proses kompresi tidak akan memberikan hasil yang efektif.

Sedangkan untuk file uji berupa *.*htm* proses kompresi akan efektif jika file uji memiliki ukuran lebih dari 806 *Byte* dan kurang dari 1.198 *Byte*.

Oleh karena algoritma kompresi yang baik adalah yang dapat menghasilkan rasio kompresi rata-rata sebesar 1,5, maka dapat disimpulkan bahwa :

- Untuk kompresi *file* *.txt, efisiensi kompresi yang baik didapatkan jika file berukuran 1.800 *byte* atau lebih.
- Untuk kompresi *file* *.htm, efisiensi kompresi yang baik didapatkan jika file berukuran 11.000 *byte* atau lebih.

4.3.4 Analisa Hasil Pengujian Dekompresi

Hasil uji coba untuk proses dekompresi dengan menggunakan kunci menunjukkan bahwa sistem dapat mengembalikan file hasil kompresi menjadi file sumber dengan akurat. Ukuran dan komposisi file hasil dekompresi sama dengan file sumber sebelum dikompresi**.**

File yang dijadikan sumber adalah file bertipe *.*huf* yang telah terbentuk dari proses kompresi dengan menggunakan kunci.

Jika kunci yang digunakan untuk dekompresi tidak sama dengan kunci yang digunakan pada saat kompresi, maka *file* yang dihasilkan pada proses dekompresi akan berisi kosong.

5. PENUTUP

5.1 Kesimpulan

Dari implementasi dan pembahasan yang telah dilakukan pada pengerjaan Skripsi ini, maka dapat diambil kesimpulan sebagai berikut :

1. Metode kompresi Huffman dengan menerapakan keamanan kriptografi AES (*Advanced Encryption Standard*) telah mampu menyusutkan ukuran data serta menjaga kerahasiaan data tersebut dari pihak yang tidak berkepentingan. Implementasi algoritma kriptografi AES untuk pengamanan proses kompresi Huffman dilakukan dengan cara mengenkripsi sebagian header *file* terkompresi.

2. Rata-rata rasio hasil kompresi Huffman dengan mengimplementasikan metode Kriptografi AES adalah sebesar 41,80 % untuk file uji **.txt* dan 25, 09 % untuk file uji **.htm*. Hal ini menunjukkan bahwa sistem dapat menyusutkan ukuran file dengan cukup baik terutama pada file **.txt*.

5.2 Saran

Sistem ini menggunakan *file* sumber dengan format ASCII UTF-8, dimana hal ini hanya mampu mengkompresi karakter latin saja. Untuk pengembangan lebih lanjut, disarankan untuk dikembangkan dengan menggunakan file sumber dengan format UNICODE, sehingga karakter yang bisa dikompresi tidak berupa huruf latin saja, namun juga huruf-huruf Arab, Cina, Jepang dan lainnya.

DAFTAR PUSTAKA

[ADL-01] Adler, M.a.M., M. *Towards Compressing Web Graphs*. in *IEEE Data Compression Conference*. 2001. Utah, USA.

[ANT-05] Anton. 2005. Kompresi dan Teks. Fakultas Teknik Informatika. Universitas Kristen Duta Wacana. http://lecturer.ukdw.ac.id/anton/download/multimedia6.pdf. Diakses pada tanggal 15 April 2012.

[ARI-06] Arinie, Farida S. 2006. Implementasi *Binary Tree* Pada Kompresi *File Text* Data Subsistem *Encoder*. Jurnal ELTEK, Volume 04 Nomor 02.

[ASC-05] ASCII - Code. *The extended ASCII Table*. 2005. http://www.injosoft.se. Diakses pada tanggal 12 April 2012.

[AYU-07] Ayuningtyas, Nadhira. 2007. Implementasi Algoritma Huffman dalam Aplikasi Kompresi Teks pada Layanan SMS. Jurusan Teknik Informatika Institut Teknologi Bandung : Bandung.

[CAL-07] Callista, Nessya. 2007. Aplikasi Greedy Pada Algoritma Huffman Untuk Kompresi Teks. Sekolah Teknik Elektro Dan Informatika Institut Teknologi Bandung : Bandung.

[FED-01] Federal Information Processing Standards Publication 197. 2001. *Advanced Encryption Standard* (*AES*). U.S. Department Of Commerce/National Institute of Standards and Technology.

[GLA-01] Gladman, B. Dr. 2001. *A Specification for Rijndael, the AES Algorithm* v3.1.

[HAN-01] Handayani, Dewi. 2001. Sistem Berkas. Yogyakarta: J&J Learning.

[HAU-95] Hauter, A ., R. Ramanathan. 1995. *Compression and Encryption*. CSI 801 *Project Fall*. http://www.science.gmu.edu/~mchacko/csi801/proj-ckv.html. diakses pada tanggal 26 Maret 2012

[JAN-05] Jando, Emanuel. 2005. Pengantar Arsitektur dan Sistim Operasi Komputer. Fakultas Ilmu Komputer. Universitas Indonesia.

[KAT-06] Kattan, Ahmed. 2006. *Universal Lossless Compression Technique With Built In Encryption*. University of Essex.

[KUR-07] Kurniawan, Y., C. 2007. Penerapan Algoritma Kriptografi DES, AES dan

RSA serta Algoritma Kompresi LZW untuk Pengamanan Berkas Digital. Universitas Kristen Petra. Surabaya.

[LIN-04] Linawati dan Henry Panggabean. 2004. Perbandingan Algoritma Kompresi pada Berbagai Tipe File. FMIPA. Universitas Katolik Parahyangan Bandung.

[MAD-96] Madenda, Sarifuddin, Hayet L. dan I. Bayu. 1996. Kompresi Citra Berwarna Menggunakan Metode Pohon Biner Huffman. Universitas Gunadarma.

[MUN-06] Munir, Rinaldi. 2006. Kriptografi. Informatika Bandung. Bandung.

[RAH-05] Rahardjo, Budi. 2005. Keamanan Sistem Informasi Berbasis Internet. PT Insan Infonesia : Bandung.

[RES-12] Restyandito. Metode Statistik Kompresi Data. http://www2.ukdw.ac.id/kuliah/info/TP4113/HO03-MetodeStatistik.pdf.Diakses pada tanggal 17 April 2012.

[RIZ-09] Riza, Toni P. 2009. ANALISIS *AVALANCHE EFFECT* TERHADAP ALGORITMA KRIPTOGRAFI *DATA ENCRYPTION STANDARD* (DES) DAN *ADVANCE ENCRYPTION STANDARD* (AES) DALAM ENKRIPSI DATA. Universitas Brawijaya.

[SCH-96] Schneier, B. 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd ed. John Wiley and Sons. New Jersey.

[STA-05] Stalling, W. 2005. *Cryptography and Network Security Principles and Practice, Fourth Edition*. Prentice Hall.

[WIN-06] Winanti, Winda. 2006. Aplikasi Pohon Biner. Teknik Informatika Institut Teknologi Bandung.

[YUN-09] Yuniati, Voni. 2009. Enkripsi dan Dekripsi Dengan Algoritma AES 256 Untuk Semua Jenis File. Univ Kristen Duta Wacana