IMPLEMENTASI ALGORITMA KRIPTOGRAFI *ADVANCED ENCRYPTION STANDAR* (AES) PADA KOMPRESI DATA TEKS

SKRIPSI

Sebagai salah satu syarat untuk memperoleh
Gelar Sarjana dalam bidang Ilmu Komputer

Oleh **:**
SONI HARZA PUTRA
0710960003

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
JURUSAN ILMU KOMPUTER
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG

2013

LEMBAR PERSETUJUAN

IMPLEMENTASI ALGORITMA KRIPTOGRAFI *ADVANCED ENCRYPTION STANDAR* (AES) PADA KOMPRESI DATA TEKS

SKRIPSI

Disusun oleh :

SONI HARZA PUTRA

NIM. 0710960003

Telah diperiksa dan disetujui oleh :

Pembimbing I,	Pembimbing II,
Edy Santoso. S.Si, M.Kom	**Lailil Muflikhah S.Kom, M.Sc**
NIP.19740414 200312 1 004	**NIP. 19741113 200501 2 001**

LEMBAR PENGESAHAN

IMPLEMENTASI ALGORITMA KRIPTOGRAFI *ADVANCED ENCRYPTION STANDARD* (AES) PADA KOMPRESI DATA TEKS

SKRIPSI

Diajukan untuk memenuhi persyaratan memperolah gelar Sarjana Komputer

Disusun oleh :

SONI HARZA PUTRA
NIM. 0710960003

Skripsi ini telah diuji dan dinyatakan lulus tanggal 23 Januari 2013

Penguji I	Penguji II
Nurul Hidayat, S.Pd., M.Sc	**Ir. Heru Nurwasito, M.Kom**
NIP. 19680430 200212 1 001	**NIP. 19650402 199002 1 001**

Penguji III

Eko Sakti Pramukantoro, S.Kom., M.Kom
NIP. 860805 0611 0252

Mengetahui
Ketua Program Studi Teknik Informatika

Drs. Marji., M.T.
NIP. 19670801 199203 1 001

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 08 Januari 2013

Mahasiswa,

Soni Harza Putra

NIM 0710960003

KATA PENGANTAR

Puji Syukur *Alhamdulillahi rabbil 'alamin* penulis panjatkan ke Hadirat Allah SWT yang telah memberikan kekuatan pada penulis untuk dapat menyelesaikan penulisan Skripsi yang berjudul "Implementasi Kriptografi AES Pada Kompresi Data Teks".

Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar Sarjana Strata Satu (S-1) Jurusan Ilmu Komputer, Program Teknologi Informasi Dan Ilmu Komputer, Universitas Brawijaya, Malang.

Sangat disadari bahwa tidak mungkin bagi penulis untuk mampu menyelesaikan penulisan Skripsi ini tanpa bantuan dari orang lain. Oleh karena itu, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang tak terhingga kepada:

1. Edy Santoso, S.Si, M.Kom., selaku pembimbing I dalam penulisan Skripsi ini. Kemudian Lailil Muflikhah, S.Kom., M.Sc., selaku pembimbing II. Terima kasih atas saran-saran, seluruh bantuan, serta waktunya dalam membimbing penulis untuk menyelesaikan Skripsi ini.
2. Drs. Mardji, MT., selaku Ketua Program Studi Ilmu Komputer
3. Dr. Abdul Rouf A, M.Sc., selaku Ketua Jurusan Matematika.
4. Segenap bapak dan ibu dosen lingkungan Fakultas MIPA yang telah memberikan ilmunya kepada penulis.
5. Bapak, Ibu dan adik-adikku, terima kasih atas doa serta dukungan yang telah diberikannya. Kalianlah yang menjadi motivasi utama penulis dalam menyelesaikan skripsi ini.
6. Pak Eko dan mas Wibi yang selalu menjaga parkiran tetap aman. Terima kasih telah selalu setia menemani penulis di kampus.
7. Para Staff dan karyawan di lingkungan Fakultas MIPA dan PTIIK.
8. Anas, Ivan, Ade, Jimi, Arief. Terima kasih atas dukungannya.
9. Rekan-rekan mahasiswa di lingkungan MIPA dan PTIIK, khususnya rekan-rekan Ilmu Komputer angkatan 2007.
10. Pihak lain yang tak dapat penulis sebutkan satu-persatu, yang telah membantu terselesaikannya penulisan skripsi ini.

Tak ada manusia yang sempurna, yang sempurna hanyalah keinginan untuk menjadi sempurna. Begitu juga dengan penulis sendiri yang menyadari bahwa tentu masih banyak kekurangan dalam laporan Skripsi ini. Oleh karenanya, kritik dan saran dari pembaca diharapkan dapat memberikan masukan bagi penulis untuk dapat memberikan karya yang lebih baik lagi.

Semoga skripsi ini merupakan salah satu pelita yang dapat memberikan cahaya bagi dunia pengetahuan Indonesia, khususnya di lingkungan Universitas Brawijaya.

Malang, Januari 2013

Penulis

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*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE PROGRAM

BAB I
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 reduksi terhadap penyimpanan data dan faktor kerahasiaan data melalui trasmisi internet 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 seseorang hendak mengirimkan sebuah *file* kepada seseorang melalui internet, dimana isi dari *file* tersebut tidak dapat dibaca oleh pihak yang tidak berkepentingan, maka teknik Kriptografi dapat memberikan solusi[SCH-96].

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[KUR-07].

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, dilakukan 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

Dari latar belakang yang telah dipaparkan di atas, maka rumusan masalah untuk penulisan skripsi ini adalah sebagai berikut:

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

Tujuan dari penulisan skripsi ini adalah :

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

1.4 Batasan Masalah

Batasan masalah dalam penulisan skripsi ini adalah:

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

1.5 Manfaat

Manfaat yang dapat diperoleh dari penulisan skripsi ini adalah mendapatkan sistem yang mampu mengkompresi sebuah data serta mampu mengamankan isi dari data tersebut dari pihak yang tidak berkepentingan.

1.6 Metode Penyelesaian Masalah

Langkah – langkah yang dilakukan untuk menyelesaikan masalah dalam penulisan skripsi ini adalah:

1. Studi Literatur

   Membaca dan mempelajari beberapa literatur (jurnal, buku dan artikel dari *website*) mengenai algoritma kompresi Huffman dan juga mengenai kriptografi khususnya Algoritma kriptografi AES.

2. Perancangan dan implementasi perangkat lunak

   Merancang dan membangum sebuah perangkat lunak yang mengimplementasikan proses kompresi dengan algoritma Huffman dan enkripsi dengan algoritma AES.

3. Uji coba dan analisis hasil implementasi

   Menganalisis hasil implementasi, yaitu rasio hasil kompresi dari beberapa jenis tipe data teks yang berbeda. Selain itu akan juga dilakukan pengujian dan perbandingan terhadap informasi data yang telah mengalami proses kompresi dan enkripsi dengan data orisinil.

1.7 Sistematika Penulisan

Bab I. PENDAHULUAN

Berisi latar belakang dan permasalahan, perumusan masalah, tujuan, ruang lingkup, metodologi penelitian dan sistematika pemulisan.

Bab II. TINJAUAN PUSTAKA

Berisi teori dan prinsip yang melandasi pembuatan skripsi.

Bab III. METODOLOGI DAN PERANCANGAN

Berisi desain program yang disajikan dalam diagram alir dan juga desain struktur data yang akan digunakan dalam program.

Bab IV. IMPLEMENTASI DAN PEMBAHASAN

Berisi penuangan hasil desain ke dalam bentuk *source code* program antara lain *procedure* dan *function* yang dipakai, serta analisa hasil uji coba.

Bb V. KESIMPULAN DAN SARAN

Berisi tentang kesimpulan dari hasil penyusunan skripsi dan saran untuk pengembangan di masa yang akan datang.

BAB II
TINJAUAN PUSTAKA

2.1 Data

Kata “data” diadopsi dari bahasa Inggris dan berasal dari kata Yunani “*datum*” yang berarti “fakta”. Data di komputer memiliki ukuran dalam penyebutannya. Data terkecil di komputer disebut dengan bit, yaitu sinyal elektronik yang melewati suatu rangkaian digital (prosesor) komputer. Bit-bit tersebut selanjutnya dirangkai, dan rangkaian tersebut diberi kode lagi yang disebut dengan *character*[HAN-01].

2.1.1 Representasi Data

Elemen-elemen data secara umum adalah terdiri dari[HAN-01] :

1. Karakter

Karakter merupakan elemen data yang paling kecil. Karakter merupakan lambang-lambang yang terdiri dari huruf, angka, serta lambang-lambang lainnya, yang dibentuk dari susunan bit. Kode yang dihasilkan dari ASCII dapat diolah oleh komputer menjadi informasi yang disebut dengan karaker tadi, sehingga manusia pun dapat membacanya. Setiap karakter yang dibentuk melalui kode komputer disebut juga dengan satu *byte*. Satu *byte* adalah merupakan sebuah karakter yang dibangun dari tujuh atau delapan bit. Satuan yang digunakan untuk menunjukkan kapasitas dalam dunia digital, termasuk komputer, besar *file*, serta ukuran lain, digunakan dalam satuan *byte* ini. Informasi yang akan dan yang diolah disimpan di dalam suatu memori.

2. *Field*

Level data lebih tinggi adalah *field* yang berisi sekumpulan karakter. *Field* kadang-kadang juga disebut sebagai suatu item. *Field* dari sebuah data menggambarkan atribut dari beberapa entitas.

3. *Record*

Hubungan *field-field* data yang digabungkan menjadi bentuk satu *record*. Suatu *record* menggambarkan kumpulan atribut yang menggambarkan entitas.

4. *File*

Sekumpulan *record* yang saling berhubungan dikenal sebagai *file* data. *File* adalah arsip yang disimpan dalam suatu media, yang terdiri dari kumpulan karakter, dan didokumentasikan dalam bentuk data digital oleh komputer. Atau dengan kata lain, *file* adalah entitas dari data yang disimpan di dalam sistem berkas yang dapat diakses dan diatur oleh pengguna. Sebuah *file* memiliki nama yang unik dalam direktori di mana file berada.

4.a. Ekstensi *File*

Setiap *file* dalam media penyimpanan memiliki tanda pengenal atau ciri-ciri yang menyatakan jenis *file* tersebut. Umumnya pengenal tipe *file* tertera pada nama *file* tersebut, yaitu tiga huruf paling kanan setelah titik. Fungsinya adalah untuk mengetahui atau membedakan jenis *file*.

2.2 Kode ASCII (UTF-8)

Kode Standar Amerika untuk Pertukaran Informasi atau ASCII (*American Standard Code for Information Interchange*) merupakan suatu standar internasional dalam kode huruf dan simbol yang bersifat universal yang selalu digunakan oleh komputer dan alat komunikasi lain untuk menunjukkan teks. Kode ASCII (UTF-8) memiliki komposisi bilangan biner sebanyak 8 bit. Dimulai dari 00000000 hingga 11111111. Total kombinasi yang dihasilkan sebanyak 256, dimulai dari kode 0 hingga 255 dalam sistem bilangan Desimal[ASC-05].

2.3 Pohon Biner (*Binary Tree*)

2.3.1 Definisi Pohon

Definisi formal dari pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit[WIN-06]. Struktur pohon adalah struktur data yang penting di bidang informatika dan pemrograman. Struktur pohon memungkinkan untuk mengorganisasi informasi berdasarkan suatu struktur lojik dan memungkinkan bebagai cara akses untuk suatu elemen.

2.3.2 Sifat Pohon

Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. *G* adalah pohon
2. Setiap pasang simpul di dalam *G* terhubung dengan lintasan tunggal.
3. *G* terhubung dan memiliki $m = n - 1$ buah sisi.
4. *G* tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. *G* tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. *G* terhubung dan semua sisinya adalah jembatan.

Beberapa istilah penting dalam pohon adalah :

- Hutan : kumpulan pohon yang saling lepas
- Anak (*child*) : subpohon dari sebuah akar
- Orangtua (*parent*) : akar dari sebuah subpohon
- Upapohon (*subtree*) : upagraf sebuah pohon sedemikian hingga upagraf tersebut mengandung x dan semua keturunannya.
- Derajat (*degree*) : jumlah upapohon (jumlah anak) pada suatu simpul.
- Ketinggian (*level*) : panjangnya jalan dari akar sampai dengan simpul yang bersangkutan.
- Daun (*leaf*) : simpul terminal dari pohon.
- Simpul (*node*) : elemen dari pohon yang memungkinkan akses pada subpohon dimana simpul tersebut berfungsi sebagai akar.
- Kedalaman (*depth*) : panjang maksimum jalan dari akar menuju ke sebuah daun.

2.3.3 Pohon biner

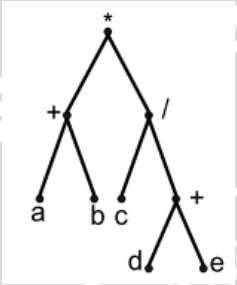
Pohon biner adalah pohon yang setiap simpulnya memiliki paling banyak dua buah anak yaitu kiri (*left*) dan kanan (*right*). Alih- alih menyebutnya anak pertama dan anak kedua dari suatu simpul dalam, dapat disebut anak kiri (*left child*) dan anak kanan (*right child*). Pohon yang akarnya adalah

anak kiri disebut upapohon kiri (*left subtree*), sedangkan pohon yang akarnya adalah anak kanan disebut upapohon kanan (*right subtree*). Karena adanya perbedaan anak/upapohon kiri dan anak/upapohon kanan, maka pohon biner adalah pohon terurut.

2.3.4 Terapan Pohon Biner

1. Pohon Ekspresi

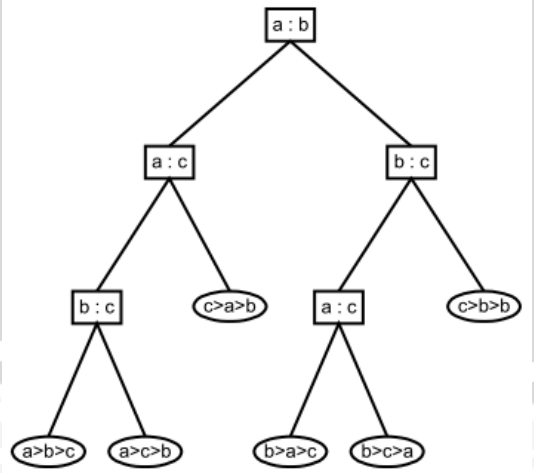
Pohon ekspresi ialah pohon biner dengan daun berupa *operand* dan simpul dalam (termasuk akar) berupa operator, seperti nampak pada gambar 2.1



Gambar 2.1 Contoh pohon ekspresi dari (*a* + *b*)*(*c*/(*d* + *e*))
Sumber :[WIN-06]

2. Pohon Keputusan

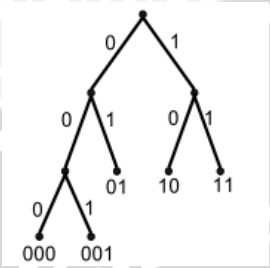
Pohon keputusan digunakan untuk memodelkan persoalan yang terdiri dari serangkaian keputusan yang mengarah ke solusi. Tiap simpul dalam menyatakan keputusan, sedangkan daun menyatakan solusi. Contoh pohon keputusan ditunjukkan gambar 2.2



Gambar 2.2 Pohon keputusan untuk mengurutkan 3 buah elemen
Sumber :[WIN-06]

3. Kode Awalan

Kode awalan (*prefix code*) adalah himpunan kode, misalnya kode biner, sedemikian sehingga tidak ada anggota
kumpulan yang merupakan awalan dari anggota yang lain. Contoh pohon biner seperti ditunjukkan gambar 2.3



Gambar 2.3 Pohon biner dari kode prefiks { 000, 001, 01, 10, 11}
Sumber :[WIN-06]

4. Kode Huffman

Kode Huffman merupakan suatu teknik yang dapat digunakan untuk mengkonstruksi kode prefiks. Contoh rangkaian bit untuk string '*ABACCDA*' berdasarkan kode ASCII adalah:

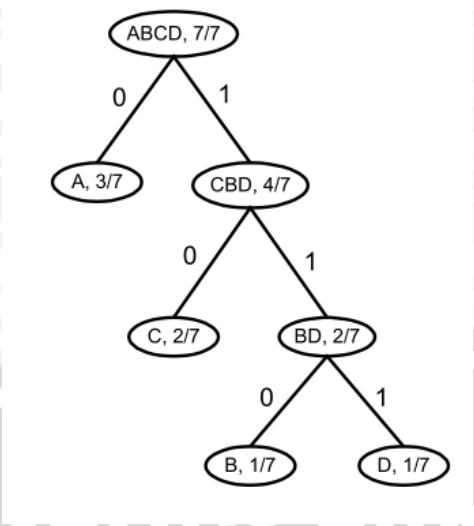
01000001010000010010000010100000110100000110100010001000001

Tabel 2.1 Tabel kekerapan dan kode Huffman untuk string 'ABACCDA'

Simbol	Frekuensi	peluang	Kode Huffman
A	3	3/7	0
B	1	1/7	011
C	2	2/7	10
D	1	1/7	111

Sumber :[WIN-06]

Informasi kode Huffman dari string 'ABACCDA' ditunjukkan tabel 2.1. Jadi, rangkaian bit kode Huffman untuk 'ABACCDA' adalah 0110010101110. Gambar 2.4 menunjukkan pohon Huffman dari string 'ABACCDA'.



Gambar 2.4 Pohon Huffman untuk pesan 'ABACCDA'
Sumber :[WIN-06]

2.4 Kompresi Data

2.4.1 Definisi

Kompresi data berarti sebuah proses mengkodekan informasi menggunakan bit atau *information-bearing unit* yang lain yang lebih rendah dari-pada representasi data yang tidak terkodekan dengan suatu sistem *encoding* tertentu[ANT-05].

Kompresi data juga diartikan sebagai suatu teknik untuk memampatkan data agar diperoleh data dengan ukuran yang lebih kecil daripada ukuran aslinya sehingga lebih efisien dalam menyimpannya atau mempersingkat waktu pertukaran data tersebut[ANT-05]. Seperti diketahui bahwa sebuah data terkadang memiliki informasi yang berulang-ulang yang membuat ukuran sebuah data menjadi besar. Dengan menggunakan algoritma dan teknik teknik tertentu, informasi yang sama dan berulang-ulang tersebut dikodekan sedemikian rupa sehingga data tersebut menjadi berukuran lebih kecil.

File atau data yang sudah dikompres agar bisa digunakan kembali harus dikembalikan lagi seperi semula. Proses pengembalian sebuah *file* terkompresi menjadi seperti *file* aslinya disebut proses dekompresi[ANT-05].

Pengiriman data hasil kompresi dapat dilakukan jika pihak pengirim/yang melakukan kompresi dan pihak penerima memiliki aturan yang sama dalam hal kompresi data. Pihak pengirim harus menggunakan algoritma kompresi data yang

sudah baku dan pihak penerima juga menggunakan teknik dekompresi data yang sama dengan pengirim sehingga data yang diterima dapat dibaca/di-dekode kembali dengan benar[ANT-05].

Misalnya terdapat kata "*Hari ini adalah hari Jum'at. Hari Jum'at adalah hari yang menyenangkan*". Jika ditelaah lebih lanjut lagi, kalimat tersebut memiliki pengulangan karakter seperti karakter pembentuk kata hari, hari Jum'at, dan adalah. Dalam teknik sederhana kompresi pada perangkat lunak, kalimat di atas dapat diubah menjadi pola sebagai berikut "**# ini $ %. % $ # ya@ menyena@kan**". Dimana dalam kalimat diatas, karakter pembentuk hari diubah menjadi karakter **#**, hari Jum'at menjadi **%**, adalah menjadi **$**, ng menjadi **@**. Saat berkas ini akan dibaca kembali, maka perangkat lunak akan mengembalikan karakter tersebut menjadi karakter awal dalam kalimat[JAN-05].

2.4.2 Jenis Teknik Kompresi

Teknik kompresi secara umum dapat diklasifikasikan menjadi tiga[RES-12], yaitu:

1. *Entropy coding* adalah teknik kompresi yang menggunakan proses lossless. Tekniknya tidak berdasarkan pada media dengan spesifikasi dan karakteristik tertentu namun berdasarkan urutan data serta tidak memperhatikan semantik data.
2. *Source coding* adalah teknik kompresi dengan menggunakan proses lossy. Teknik ini berkaitan dengan data semantik (arti data) dan media.
3. *Hybrid coding* adalah teknik kompresi dengan menggunakan kombinasi atau gabungan dari entropy coding dan source coding.

Berdasarkan outputnya, teknik kompresi dapat dibedakan menjadi dua:

1. Teknik kompresi *Lossy*

Kompresi menggunakan *lossy*, beberapa bagian data asli hilang ketika berkas di *decoded*. Keuntungan dari algortima ini adalah bahwa rasio

kompresi cukup tinggi. Hal ini dikarenakan cara algoritma *lossy* yang mengeliminasikan beberapa data dari suatu berkas. Namun data yang dieliminasikan biasanya adalah data yang kurang diperhatikan atau diluar jangkauan manusia, sehingga pengeliminasian data tersebut kemungkinan besar tidak akan mempengaruhi manusia yang berinteraksi dengan berkas tersebut.

2. Teknik kompresi *Lossless*

Kompresi menggunakan *lossless* menjamin bahwa berkas yang dikompresi dapat selalu dikembalikan ke bentuk aslinya. Algoritma *lossless* digunakan untuk kompresi berkas *text*, seperti program komputer (berkas zip, rar, dll), karena seseorang ingin mengembalikan berkas yang telah dikompres ke status aslinya. Kadangkala ada data- data yang setelah dikompresi dengan teknik ini ukurannya menjadi lebih besar atau sama.

Berdasarkan teknik pengkodean/pengubahan simbol yang digunakan, metode kompresi dapat dibagi ke dalam tiga kategori[LIN-04], yaitu :

1. Metode *symbolwise* : menghitung peluang kemunculan dari tiap simbol dalam file input, lalu mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Contoh: algoritma *Huffman*.
2. Metode *dictionary* : menggantikan karakter/fragmen dalam file input dengan indeks lokasi dari karakter/fragmen tersebut dalam sebuah kamus (*dictionary*). Contoh: algoritma LZW.
3. Metode *predictive* : menggunakan model *finite-context* atau *finite-state* untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya. Contoh: algoritma DMC.

Terdapat beberapa faktor yang sering menjadi pertimbangan dalam memilih suatu metode kompresi yang tepat karena tidak ada suatu metode kompresi yang paling efektif untuk semua jenis *file*. Faktor-faktor tersebut adalah:

1. Kualitas data hasil *encoding*: ukuran lebih kecil, data tidak rusak untuk kompresi *lossy*.
2. Ketepatan proses dekompresi data: data hasil dekompresi tetap sama dengan data sebelum dikompres (kompresi *loseless*).

3. Kecepatan, rasio, dan efisiensi proses kompresi dan dekompresi.

Proses kompresi adalah proses *encoding* yang menghasilkan data yang sudah dikompresi yang disebut aliran data *encoded*. Sebaliknya aliran data yang telah dikompresi harus dilakukan proses dekompresi untuk menghasilkan kembali aliran data yang asli. Karena proses dekompresi menghasilkan *decoding* dari aliran data yang sudah dikompresi maka hasilnya adalah aliran data *decoded*.

Tingkat pengurangan data yang dicapai sebagai hasil dari proses kompresi disebut rasio kompresi. Rasio ini merupakan perbandingan antara panjang data string asli dengan panjang data string yang sudah dikompresi, seperti dituliskan dalam persamaan 2.1:

$$\text{Rasio} = \frac{\text{ukuran file asli}}{\text{ukuran file terkompresi}} \qquad (2.1)$$

Jika dinyatakan dalam presentase, maka dituliskan dalam persamaan 2.2:

$$P = \left(1 - \frac{\text{ukuran file terkompresi}}{\text{ukuran file asli}}\right) * 100\% \qquad (2.2)$$

Ini berarti ukuran *file* berkurang sebesar *P* (dalam persentase) dari ukuran semula. Semakin tinggi rasio tingkat suatu teknik kompresi data maka semakin efektif teknik kompresi tersebut. Pada saat dikompresi, rasio kompresi akan berselang-seling berdasarkan pengaruh data terhadap algortima yang digunakan. Dengan demikian maka yang perlu diperhatikan adalah rasio kompresi rata-rata. Bukan pada rasio yang dicapai pada suatu waktu tertentu. Pada umumnya algoritma yang baik dapat mencapai presentase rasio kompresi rata - rata 1,5 atau jika dalam presentase sebesar 33%[MAD-96].

2.5 Algoritma Huffman

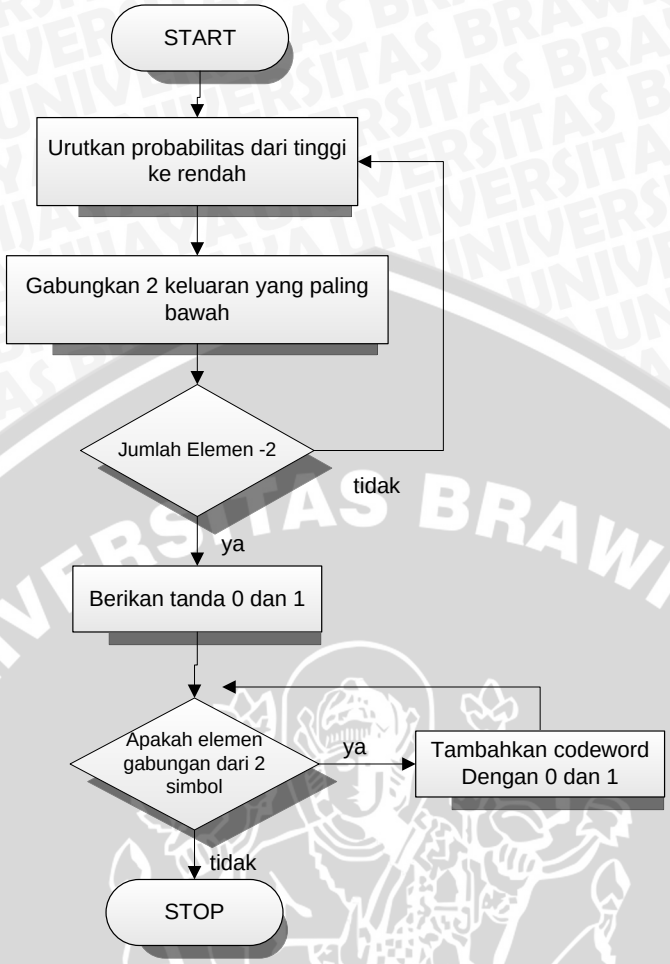
Kode Huffman menggunakan tabel dengan variasi kode panjang untuk melakukan *encoding* dari sebuah simbol. Tabel variabel kode panjang tersebut telah dibuat terlebih dahulu secara terpisah berdasarkan nilai

kekerapan munculnya suatu simbol. Metode ini ditemukan oleh David A. Huffman ketika ia melakukan studi Ph.D di MIT. Kode ini dipublikasikan tahun 1952 pada tulisannya yang berjudul "*A Method for the Constuction of Minimum-Redudancy Codes*"[AYU-07].

Prinsip kode Huffman adalah mengganti karakter yang paling sering muncul di dalam data dengan kode yang lebih pendek, sedangkan karakter yang lebih jarang muncul dikodekan dengan kode yang lebih panjang. Algoritma Huffman menerapkan metode statik yaitu menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase yaitu fase pertama untuk menghitung probabilitas kemunculan tiap karakter dan menentukan peta kodenya dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan[CAL-07].

2.5.1 Pembentukan Pohon 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 2.5.



Gambar 2.5 Diagram alir pengkodean Huffman

Sumber :[ARI-06]

Sebagai contoh, sebuah file yang akan dimampatkan berisi karakter-karakter "PERKARA". Dalam kode ASCII masing-masing karakter dikodekan sebagai :

P = 50H = 01010000B

E = 45H = 01000101B

R = 52H = 01010010B

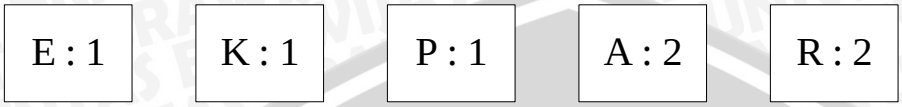
K = 4BH = 01001011B

A = 41H = 01000001B

Maka jika diubah dalam rangkaian bit, "PERKARA" menjadi :01010000010001010101001001001011010000010101001 001000001 yang berukuran 56 bit.

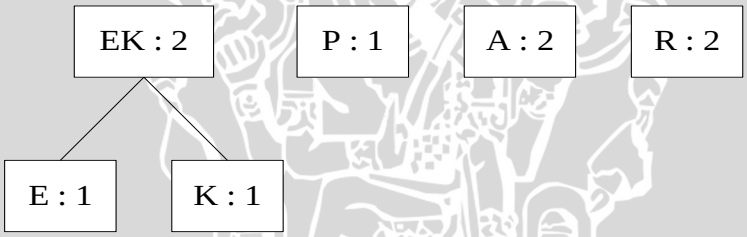
Langkah-langkah pengkodean Huffman sesuai diagram alir pada gambar 2. 5 untuk memperkecil ukuran bit adalah sebagai berikut:

1. Menyusun urutan kode dari abjad sesuai kode ASCII berdasarkan frekuensi kemunculan selanjutnya dibuat simpul-simpulnya. Susunan abjad yang terurut ditunjukkan gambar 2.6.



Gambar 2.6 Frekuensi kemunculan karakter
Sumber :[ARI-06]

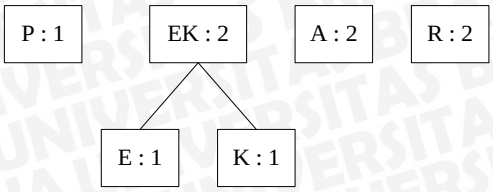
2. Ambil 2 simpul yang paling kiri (P dan E) dengan frekuensi kemunculan terkecil, lalu buat simpul baru yang merupakan gabungan dua simpul tersebut, simpul gabungan ini akan memiliki cabang masing-masing 2 simpul dari penggabungan tersebut. Frekuensi gabungan ini adalah jumlah frekuensi cabang-cabangnya. Hal ini tampak pada gambar 2.7.



Gambar 2.7 Pembentukan Huffman *Tree* langkah 1
Sumber :[ARI-06]

3. Frekuensi tiap simpul pada level paling atas, EK=2, P=1, A=2, dan R=2. Selanjutnya simpul-simpul ini diurutkan lagi dari yang paling kecil, jadi simpul EK harus digeser ke sebelah kanan *node* P dan ingat jika menggeser suatu node yang memiliki cabang, maka seluruh cabangnya harus diikutkan juga.

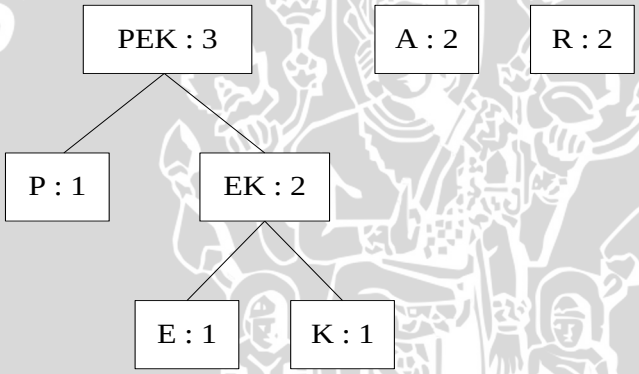
4. Setelah diurutkan hasilnya ditunjukkan pada gambar 2.8.



Gambar 2.8 Pembentukan Huffman *Tree* langkah 2
Sumber :[ARI-06]

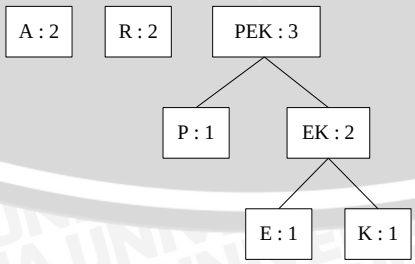
Setelah simpul pada level paling atas diurutkan (level berikutnya tidak perlu diurutkan), berikutnya digabungkan kembali 2 simpul paling kiri seperti yang pernah dikerjakan sebelumnya.

5. Simpul P digabung dengan simpul EK menjadi simpul PEK dengan frekuensi 3 dan gambarnya akan menjadi seperti gambar 2.9.



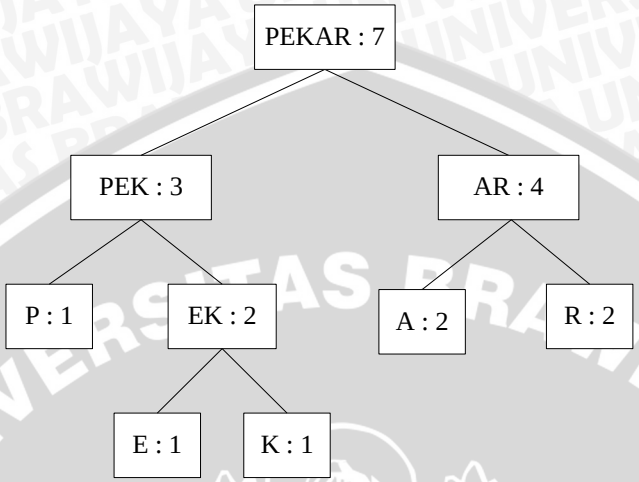
Gambar 2.9 Pembentukan Huffman *Tree* langkah 3
Sumber :[ARI-06]

6. Kemudian diurutkan lagi menjadi seperti tampak pada gambar 2.10.



Gambar 2.10 Pembentukan Huffman *Tree* langkah 4
Sumber :[ARI-06]

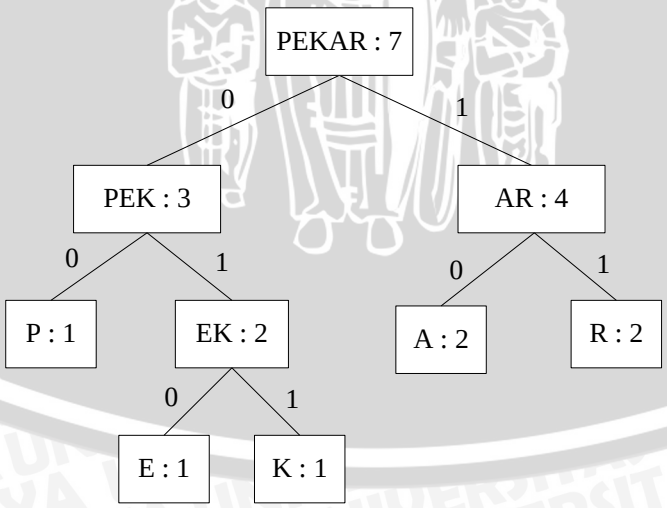
Demikian seterusnya sampai diperoleh pohon Huffman seperti ditunjukkan gambar 2.11.



Gambar 2.11 Huffman *Tree* String "PERKARA"
Sumber :[ARI-06]

7. Setelah pohon Huffman terbentuk, berikan tanda bit 0 untuk setiap cabang ke kiri dan bit 1 untuk setiap cabang ke kanan seperti gambar 2.12.

8.



Gambar 2.12 Huffman *Tree* String "PERKARA" setelah penandaan bit 0 dan 1
Sumber :[ARI-06]

9. Selanjutnya menentukan telusur dari bit-bit kode dari cabang-cabang daun (leaf) yang dilakukan sebagai berikut,untuk mendapatkan kode Huffman masing-masing karakter, telusuri karakter tersebut dari simpul yang paling atas (PEKAR) sampai ke simpul karakter tersebut dan susunlah bit-bit yang dilaluinya.

2.5.2 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.

Untuk mendapatkan kode Karakter E, dari simpul PEKAR kemudian menuju ke simpul PEK melalui bit 0 dan selanjutnya menuju ke simpul EK melalui bit 1, dilanjutkan ke simpul E melalui bit 0, jadi kode dari karakter E adalah 010. Untuk mendapatkan kode karakter K, dari node PEKAR maka harus menuju ke simpul PEK melalui bit 0 dan selanjutnya menuju ke simpul EK melalui bit 1, dilanjutkan ke simpul K melalui bit 1, jadi kode dari karakter K adalah 011.

Untuk mendapatkan kode Karakter P, dari simpul PEKAR menuju ke simpul PEK melalui bit 0 dan selanjutnya menuju ke simpul P melalui bit 0, jadi kode dari karakter P adalah 00. Untuk mendapatkan kode Karakter A, dari simpul PEKAR harus menuju ke simpul AR melalui bit 1 dan selanjutnya menuju ke simpul A melalui bit 0, jadi kode dari karakter A adalah 10. Terakhir, untuk mendapatkan kode Karakter R, dari simpul PEKAR maka harus menuju ke simpul AR melalui bit 1 dan selanjutnya menuju ke simpul R melalui bit 1, jadi kode dari karakter R adalah 11.

Hasil akhir kode Huffman ditunjukkan pada tabel 2.2.

Tabel 2.2 Tabel Huffman hasil *encoding*

Simbol	*Frekuensi*	*peluang*	*kode Huffman*
E	1	1/7	010
K	1	1/7	011
P	1	1/7	00
A	2	2/7	10
R	2	2/7	11

Sumber :[ARI-06]

Dengan kode ini, file yang berisi karakter-karakter "PERKARA" akan menjadi lebih kecil, yaitu : 00 010 11 011 10 11 10 = 16 bit.

Dengan Algoritma Huffman berarti file ini dapat dihemat sebanyak 56-16 = 40 bit.

2.5.3 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.

Untuk men-*decode* dengan menggunakan tabel kode Huffman, sebagai contoh, akan digunakan kode Huffman pada Tabel 2.2 yang merepresentasikan string "PERKARA". Dengan menggunakan Tabel 2.2 string tersebut akan direpresentasikan menjadi rangkaian bit : 00 010 11 011 10 11 10. Total jumlah bit yang dibutuhkan sebanyak 16 bit. Dari Tabel 2.2 tampak bahwa kode untuk sebuah simbol atau karakter tidak boleh menjadi awalan dari kode simbol yang lain untuk menghindari ambiguitas dalam proses dekompresi atau *decoding*. Karena tiap kode Huffman yang dihasilkan unik, maka proses *decoding* dapat dilakukan dengan mudah. Contoh: saat membaca kode bit pertama dalam rangkaian bit "00 010 11 011 10 11 10 ", didapatkan bit "0", cek kode "0" pada tabel 2.2, karena tidak terdapat kode yang dimaksud pada Tabel

2.2, maka dilakukan pembacaan bit berikutnya. Didapatkan kode ”00”, yang kemudian dicocokkan kembali pada tabel dan dapat langsung disimpulkan merupakan pemetaan dari karakter “P”. Kemudian baca kode bit selanjutnya, yaitu bit “0”. Tidak terdapat kode Huffman “0”, lalu baca kode bit selanjutnya, sehingga menjadi “01”. Tidak terdapat juga kode Huffman “01”, lalu baca lagi kode bit berikutnya, sehingga menjadi “010”. Didapatkan rangkaian kode bit “010” adalah pemetaan dari simbol “E”.

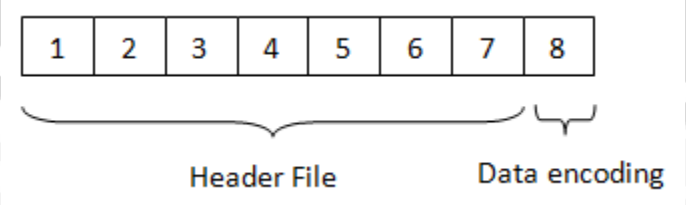
Sedangkan langkah-langkah untuk men-*decoding* suatu string biner dengan menggunakan pohon Huffman adalah sebagai berikut[AYU-07]:

1. Baca sebuah bit dari string biner.
2. Untuk setiap bit pada langkah 1, lakukan pembacaan pada cabang yang bersesuaian.
3. Ulangi langkah 1 dan 2 sampai bertemu *node*. Kodekan rangkaian yang telah dibaca dengan karakter di daun.
4. Ulangi dari langkah 1 sampai semua bit di dalam string habis.

2.5.4 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 2.13.



Gambar 2.13 Mekanisme penyimpanan *file* terkompresi
Sumber :[CAL-07]

Keterangan pada gambar 2.13 adalah sebagai berikut:

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.*

2.6 Kriptografi

Pesan (*message*) adalah data yang dapat dibaca dan dimengerti maknanya. Kata lain dari pesan adalah *plaintext*. Enkripsi merupakan proses untuk menyembunyikan pesan dalam bentuk lain yang tidak mudah dimengerti untuk menyembunyikan substansinya[SCH-96]. Pesan yang telah dienkripsi disebut *ciphertext* dan proses untuk mengembalikan *ciphertext* menjadi *plaintext* disebut dekripsi. Keterkaitan tersebut digambarkan pada Gambar 2.14.

Gambar 2.14 Enkripsi dan dekripsi pada *plaintext*
Sumber :[SCH-96]

Kriptografi (*cryptography*) adalah ilmu dan seni untuk menjaga keamanan pesan. Para praktisi kriptografi disebut kriptografer (*cryptographers*) [SCH-96]. Sebuah algoritma kriptografi disebut *cipher*, merupakan persamaan matematika yang digunakan untuk proses enkripsi dan dekripsi. Biasanya kedua persamaan matematika (untuk enkripsi dan dekripsi) memiliki hubungan matematis yang cukup erat[RAH-05].

Konsep matematis yang mendasari algoritma kriptografi adalah relasi antara dua buah himpunan, yaitu himpunan yang berisi elemen-elemen *plaintext* dan himpunan yang berisi *ciphertext*. Misalkan *P* menyatakan *plaintext* dan *C* menyatakan *ciphertext*, maka fungsi enkripsi *E* memetakan *P* ke *C*,

$$E(P) = C \tag{2.3}$$

dan fungsi dekripsi D memetakan C ke P,

$$D(C) = P \tag{2.4}$$

Karena proses enkripsi kemudian dekripsi mengembalikan pesan ke pesan awal, maka:

$$D(E(P)) = P \tag{2.5}$$

[MUN-06]

2.7 Algoritma Kriptografi

Algoritma kriptografi yang biasanya disebut dengan *cipher* adalah fungsi matematika yang digunakan untuk enkripsi dan dekripsi[SCH-96]. Berdasarkan kunci yang dipakai, algoritma kriptografi dapat dibedakan atas 2 jenis, yakni:

1. Algoritma kunci simetris (*symmetric-key cryptography*)

Dalam algoritma kunci simetris, kunci yang digunakan untuk melakukan enkripsi sama dengan kunci yang digunakan untuk dekripsi. Istilah lain yang digunakan untuk algoritma ini adalah algoritma kriptografi kunci privat. Keamanan kriptografi ini terletak pada kerahasiaan kuncinya. Contoh algoritma kunci simetris adalah DES (*Data Encryption Standard*), *Blowfish*, *Twofish*, dan *AES*.

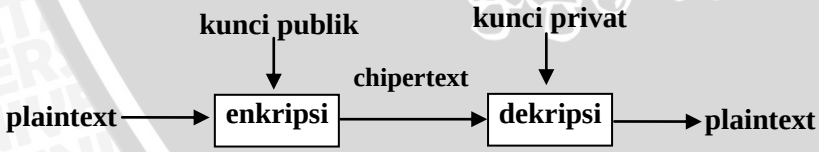
Proses enkripsi dan dekripsi algoritma kunci simetris dapat dilihat pada gambar 2.15.

Gambar 2.15 Enkripsi dan dekripsi algoritma kunci simetris
Sumber : [SCH-96]

Algoritma kriptografi simetris dibagi menjadi dua yaitu algoritma aliran (*stream ciphers*) dan algoritma blok (*block ciphers*). Pada algoritma aliran, proses enkripsi berorientasi pada satu bit atau satu byte data. Sedangkan untuk algoritma blok, proses enkripsi berorientasi pada sekumpulan bit atau *byte* data (per blok).

2. Algoritma kunci asimetris

Jika kunci yang digunakan untuk enkripsi tidak sama dengan kunci untuk dekripsi, maka algoritma kriptografinya disebut algoritma kunci asimetris. Algoritma ini disebut juga algoritma kriptografi kunci publik. Semua orang yang mendapatkan kunci publik dapat menggunakannya untuk mengenkripsikan suatu pesan, data ataupun informasi, namun hanya satu orang saja yang memiliki kunci privat untuk mendekripsikan *ciphertext* yang diterimanya menjadi *plaintext*. Contoh dari algoritma ini adalah RSA, *ElGamal* dan *Knapsack*. Proses enkripsi dan dekripsi algoritma kunci asimetris dapat dilihat pada gambar 2.16.



Gambar 2.16 Enkripsi dan dekripsi algoritma kunci asimetris
Sumber : [SCH-96]

2.8 AES 128

2.8.1 *Input* dan *Output*

Input dan output untuk AES masing-masing terdiri dari deret 128 bit (digit-digit dengan nilai 0 atau 1). Deret ini kadang disebut sebagai blok dan jumlah bit di dalamnya disebut sebagai panjangnya. *Cipher key* (*key* yang digunakan untuk enkripsi atau dekripsi) untuk AES adalah deret 128, 192 atau 256 bit. Panjang input, output atau *key* yang berbeda tidak diizinkan dalam standar ini[FED-01].

Bit-bit di dalam deret ini dihitung mulai 0 dan berahir pada panjang deret dikurangi 1. Angka i yang diberikan pada sebuah bit disebut juga sebagai indeksnya, dan akan berada dalam *range* $0 \leq i < 128$, $0 \leq i < 192$ atau $0 \leq i < 256$ bergantung pada panjang blok atau kunci[FED-01].

2.8.2 *Byte*

Unit dasar untuk pemrosesan dalam AES adalah *byte*, sebuah deret yang terdiri dari 8 bit yang dianggap sebagai satu entitas. Deret input, output dan *key* diproses sebagai *array bytes* yang dibentuk dengan cara membagi deret tadi menjadi grup-grup 8 bit berurutan untuk membentuk *array bytes*. Untuk sebuah *input*, *output* atau *key* yang disimbolkan dengan a, *bytes* dalam *array* yang dihasilkan ditunjuk menggunakan dua jenis bentuk, a_n atau a[n], dimana n ada di antara *range* berikut:

panjang *key* = 128 bit, $0 \leq n < 16$;

panjang *key* = 192 bit, $0 \leq n < 24$;

panjang *key* = 256 bit, $0 \leq n < 32$;

panjang blok = 128 bit, $0 \leq n < 16$.

Semua nilai *byte* dalam AES ditampilkan sebagai gabungan tiap-tiap bit individual (0 atau 1) di dalam tanda kurung dengan urutan $\{b_7, b_6, b_5, b4, b_3, b_2, b_1, b_0\}$. *Byte-byte* ini diterjemahkan sebagai elemen *finite field* menggunakan representasi polinomial:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_ix^i$$

Misalnya {01100011} mengidentifikasi elemen *finite field* yang spesifik yaitu $x^6 + x^5 + x + 1$ [FED-01].

Cukup mudah juga untuk menampilkan nilai *byte* menggunakan notasi heksadesimal dengan tiap-tiap dua grup 4 bit direpresentasikan oleh satu karakter, dapat dilihat pada tabel 2.3.

Tabel 2.3 Representasi heksadesimal dari susunan bit

Susunan bit	*Heksadesimal*	Susunan bit	*Heksadesimal*
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Sumber : [SCH-96]

Maka berdasarkan Tabel 2.3, elemen {01100011} dapat direpresentasikan sebagai {63}, dimana karakter yang menunjukkan grup 4 bit dengan bit bernilai tinggi berada di kiri. Beberapa operasi *finite field* melibatkan satu bit tambahan (b_8) di ujung kiri dari sebuah *byte* yang tersusun dari 8 bit. Saat bit tambahan ini ada, ia akan tampil sebagai {01} sebelum *byte*; contoh, sebuah deret 9 bit akan ditampilkan sebagai {01}{1b}[FED-01].

2.8.3 *Array bytes*

Array bytes (*array* yang berisi sejumlah *byte*) direpresentasikan pada bentuk berikut :

$a_0\, a_1\, a_2 \ldots\ldots a_{15}$

Urutan *byte-byte* dan bit dalam *bytes* didapatkan dari sebuah rangkaian input berukuran 128 bit

$input_0\ input_1\ input_2 \ldots\ldots\ldots input_{126}\ input_{127}$ sebagai berikut :

$a_0 = input_0\ input_1 \ldots\ldots\ldots input_7$;

$a_0 = input_8\ input_9 \ldots\ldots\ldots input_{15}$;

.

.

.

$a_{15} = input_{120}\ input_{121} \ldots\ldots\ldots input_{127}$;

Pola tersebut dapat diperluas menjadi rangkaian yang lebih panjang (contoh: untuk *key* dengan ukuran 192 dan 256 bit), sehingga bentuk umum dari pola rangkaian akan menjadi:

$a_n = input_{8n}\ input_{8n+1} \ldots\ldots\ldots input_{8n+7}$; [FED-01].

2.8.4 *State*

Pada dasarnya, operasi-operasi algoritma AES dilakukan pada sebuah *array bytes* dua dimensi yang disebut *state*. *State* terdiri dari 4 baris *byte*, setiap baris terdiri dari Nb buah *byte*, dimana Nb adalah panjang blok dibagi dengan 32. Pada *array state* yang dinotasikan dengan simbol S, setiap *byte* tunggal memiliki dua buah parameter, dimana nomor baris r berada pada *range* $0 \leq r < 4$ dan nomor kolom c barada pada *range* $0 \leq c < Nb$. Hal ini memungkinkan sebuah *byte* tunggal dari *state* untuk dapat dinyatakan sebagai $S_{r,c}$ atau $S[r,c]$. Sebagai standar yang digunakan di sini, Nb = 4, yang berarti jumlah kolom c adalah pada rentang $0 \leq c < 4$ [MUN-06].

Pada awal proses enkripsi (*cipher*) dan dekripsi (*inverse cipher*), input – yaitu *array byte* $in_0, in_1, \ldots in_{15}$ - disalin ke dalam *array state* sebagaimana yang diilustrasikan pada Gambar 2.17. Operasi-operasi *cipher* dan *invers cipher* kemudian dilakukan pada *array state* tersebut, sehingga didapat nilai akhir yang kemudian disalin ke dalam *output – array bytes* $out_0, out_1, \ldots out_{15}$ [FED-01].

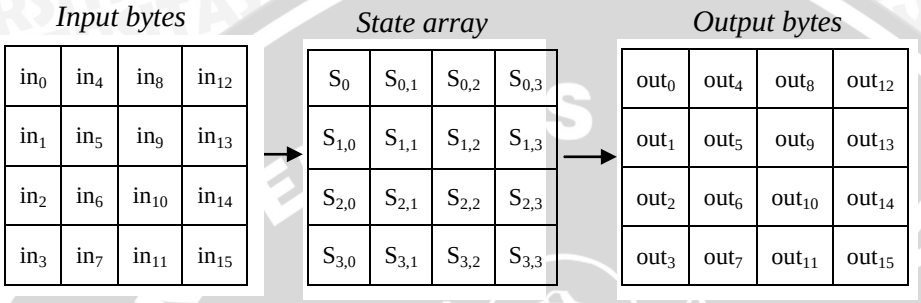
Dengan demikian, pada proses awal dari enkripsi (*cipher*) dan dekripsi (*inverse cipher*), *array input*, *in*, disalin ke dalam *array state* dengan skema sebagai berikut:

S[r,c] ← *in*[r + 4c] untuk $0 \le r < 4$ dan $0 \le c \le 4$ *Nb*

dan pada proses akhir enkripsi dan dekripsi, *state* disalin ke dalam *array* output *out* sebagai berikut :

out[r + 4c] ← S[r,c] untuk $0 \le r < 4$ dan $0 \le c \le 4$ *Nb*

[MUN-06]. Keterkaitan antara *Input bytes*, *State array* dan *Output bytes*ditunjukkan gambar 2.17.



Gambar 2.17 State Array, input dan Output
Sumber : [FED-01]

2.8.5 Perhitungan matematika

Semua *byte* dalam AES diinterpretasikan sebagai elemen *finite field* menggunakan notasi pada 2.8.2. Elemen-elemen ini dapat dijumlah dan dikalikan namun operasinya berbeda dengan penjumlahan atau perkalian pada angka-angka biasa.

2.8.5.1 Penjumlahan

Penjumlahan dua elemen dalam *finite field* diperoleh dengan "menambah" koefisien-koefisien dari pangkat-pangkat yang sesuai dari dua elemen tersebut. Penjumlahan dilakukan dengan operasi XOR. Oleh karena itu, pengurangan dari polinomial identik dengan penjumlahannya.

Secara elternatif, penjumlahan elemen *finite field* dapat dijelaskan sebagai penjumlahan modulo 2 dari bit-bit yang terkait di dalam *byte*. Untuk 2 *byte* $\{a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0\}$ dan $\{b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\}$ jumlahnya adalah $\{\ c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0\ \}$ dimana tiap-tiap $c_i = a_i \oplus b_i$.

Contoh, ekspresi berikut ekuivalen satu sama lain:

$(x^6+x^4+x^2+x+1) + (x^7+x+1) = x^7 + x^6 + x^4 + x^2$ (notasi polinomial)

$\{01010111\} \oplus \{10000011\} = \{11010100\}$ (notasi biner)

{57} ⊕ {83} = {d4} (notasi heksadesimal)

[FED-01]

2.8.5.2 Perkalian

Dalam representasi polinomial, perkalian dalam GF(2^8) (disimbolkan dengan •) adalah perkalian dari polinomial modulo sebuah polinomial yang *irreducible* dengan derajat 8. Sebuah polinomial dikatakan *irreducible* jika satu-satunya pembaginya adalah 1 dan dirinya sendiri. Untuk AES, polinomial *irreducible* ini adalah:

$m(x) = x^8+x^4+x^3+x+1$

atau {01} {1b} dalam notasi heksadesimal. Sebagai contoh : {57} • {83} = {*c*1} karena

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 x^5 + x^4 + x^3 + 1$$

dan

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod (x^8 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + 1.$$

Reduksi modular oleh m(x) memastikan bahwa hasilnya akan selalu polinomial biner dengan derajat kurang dari 8, sehingga dapat direpresentasikan dalam sebuah byte [GLA-01].

2.8.5.3 Perkalian menggunakan tabel

Jika elemen *finite field* tertentu (disebut generator) berulang-ulang dikalikan untuk menghasilkan sebuah deret dari pangkatnya, g^n, mereka secara progresif menghasilan semua 255 elemen tidak nol di dalam *field*. Ketika n mencapai 256 elemen *field* awal kembali muncul dengan g^{255} sehingga setara dengan {01}. Nilai n untuk tiap elemen *field* dapat dianggap sebagai logaritma dan ini memberikan jalan untuk mengubah perkalian menjadi penjumlahan.

Maka, dua elemen $a = g^{\alpha}$ dan $b = g^{\beta}$ menghasilkan $a \bullet b = g^{\alpha+\beta}$. Dengan

sebuah tabel logaritma yang menunjukkan pangkat dari generator untuk tiap elemen *finite field*, akan diperoleh pangkat α dan β yang sesuai untuk elemen *a* dan *b*, dan menjumlahkan nilai ini untuk menemukan pangkat dari g untuk memperoleh hasil. Tabel kebalikan dapat dipakai untuk mencari elemen produk.

Karena dua nilai pangkat awal dapat mencapai 255, jumlahnya bisa jadi lebih besar dari 255, namun jika ini terjadi bisa dikurangkan nilai 255 dari nilainya sehingga membawanya kembali ke dalam *range* dari tabel karena g^{255} = {01}. Semua eksponen ditulis dalam heksadesimal seperti pada tabel 2.4.

Tabel 2.4 Nilai-nilai N sehingga {xy}={03}^{n} untuk sebuah elemen *finite field* {xy}

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

Sumber : [GLA-01]

Untuk *field* yang dipakai di dalam Rijndael, {03} adalah generator yang menghasilkan tabel 2.4 dan tabel 2.5. Dengan menggunakan contoh sebelumnya, Tabel 2.4 menunjukkan bahwa {57}={03}$^{(62)}$ dan {83}={03}$^{(50)}$, dimana tanda kurung dalam nilai eksponen menunjukkan bahwa itu adalah nilai heksadesimal. Ini memberi produk {57} • {83}={03}$^{(62+50)}$, dan karena (62)+(50)=(b2) dalam heksadesimal

Hasil produk {c1} diberikan oleh tabel 2.5, seperti contoh sebelumnya. Kedua tabel tersebut dapat juga digunakan untuk menemukan invers dari sebuah elemen *field* karena $g^{(x)}$ memiliki sebuah invers yang berupa $g^{(ff)-(x)}$. Maka elemen {af}={03}$^{(b7)}$ memiliki invers $g^{(ff)-(b7)}$ = $g^{(48)}$ = {62}. Semua elemen kecuali {00} memiliki invers[GLA-01].

Tabel 2.5 Elemen *field* {E} sehingga {E}={03}^{xy} jika diketahui pangkat (xy)

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

Sumber : [GLA-01]

2.8.6 Algoritma kriptografi AES

Algoritma kriptografi AES (*Advance Encryption Standard*), yang sering juga disebut dengan algoritma kriptografi Rijndael, sama seperti algoritma kriptografi DES yang menggunakan substitusi, permutasi dan sejumlah putaran (*cipher* berulang). Setiap putaran menggunakan kunci internal yang berbeda, dimana kunci pada setiap putaran disebut dengan *round key*[MUN-06].

Algoritma kriptografi AES termasuk dalam klasifikasi algoritma kriptografi kunci simetri, kunci yang digunakan untuk enkripsi sama dengan kunci yang digunakan untuk dekripsi. Berbeda dengan DES, yang berorientasi bit, Rijndael beroperasi dalam *byte*. Algoritma AES dapat bekerja dalam tiga macam ukuran yakni 128 bit (16 *byte*), 192 bit (24 *byte*) dan 256 bit (32 *byte*)[KUR-07]. Skripsi ini akan mengimplementasikan AES 128 bit.

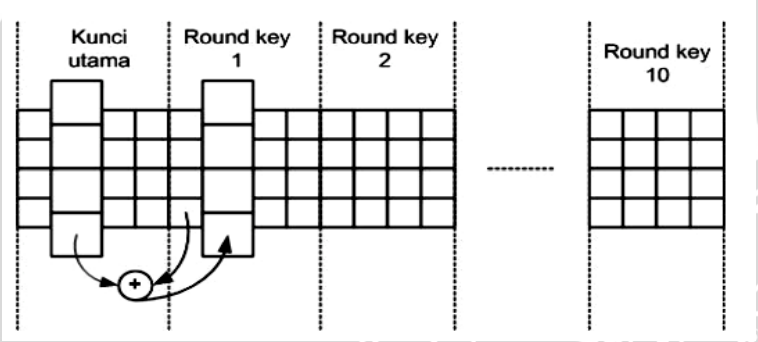
2.8.7 Proses Inisialisasi Kunci Internal

Dalam algoritma kriptografi AES terdapat 11 kali iterasi, dimana masing-masing iterasi akan menggunakan kunci yang berbeda. Untuk itu diperlukan 10 kunci internal yang berbeda (K_1, K_2,...,K_{10}). Kunci internal ini diturunkan dari kunci eksternal yang berukuran 128 bit (*array* 4 x 4 dengan

masing-masing elemen sebesar 1 byte). Kunci ini terdiri atas kunci utama (kunci eksternal yang diinputkan oleh user) dan 10 buah kunci putaran (*round key*) [MUN-06].

Proses inisialisasi kunci internal pada algoritma kriptografi AES terdiri dari dua bagian, yaitu proses untuk memperoleh nilai elemen-elemen pada kolom kedua hingga kolom keempat untuk masing-masing *round key* dan proses untuk memperoleh nilai elemen-elemen pada kolom pertama[KUR-07].

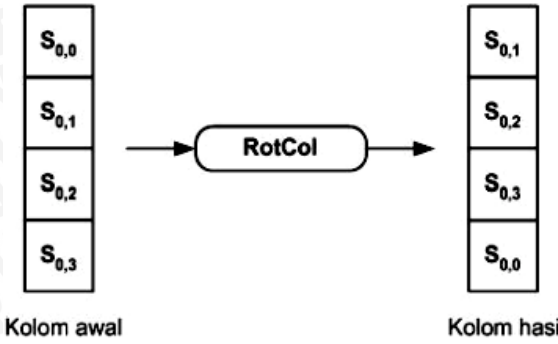
Proses untuk memeperoleh nilai elemen-elemen pada kolom kedua hingga keempat ditunjukkan gambar 2.18. Untuk kolom kedua hingga keempat dari masing-masing *round key*, elemen-elemennya diperoleh dari hasil XOR antara satu kolom sebelumnya dengan kolom yang sama pada *array* kunci sebelumnya.



Gambar 2.18 Skema proses inisialisasi kunci pada AES

Sumber : [KUR-07]

Untuk kolom pertama pada masing-masing *round key*, elemen-elemennya diperoleh melalui tiga transformasi yang dilaksanakan secara berurutan, yaitu RotCol, SubBytes dan XOR. Transformasi RotCol adalah operasi perputaran elemen berdasarkan barisnya, baris pertama menjadi baris keempat, baris kedua menjadi baris pertama, baris ketiga menjadi baris kedua, dan baris keempat menjadi baris ketiga [MUN-06]. Dapat dilihat pada gambar 2.19.



Gambar 2.19 Skema proses *RotCol* pada AES
Sumber : [MUN-06]

Hasil dari transformasi *RotCol* tersebut menjadi masukan untuk transformasi *SubBytes*, yakni operasi substitusi yang dilakukan terhadap masing-masing *byte* dengan nilai yang ditunjukkan pada matriks substitusi *S-Box*. Transformasi *SubBytes* dalam proses inisialisasi kunci internal pada AES ini sama dengan transformasi *SubBytes* pada proses enkripsi[STA-05].

Hasil dari transformasi *SubBytes* akan di-XOR-kan dengan kolom pertama pada kunci sebelumnya dan *Rcon* untuk *round key* tersebut. *Rcon* adalah *array* 4 x 1 yang merupakan deretan konstanta yang digunakan dalam proses menghasilkan *round key*, dapat dilihat pada tabel 2.7. Dari proses inisialisasi kunci internal akan diperoleh 11 kunci internal yang diperoleh dari kunci eksternal yang diinputkan oleh user dan 10 *round key*. Ukuran untuk setiap kunci internal sebesar 128 bit dengan format *array* 4 x 4 dengan masing-masing elemen sebesar 1 *byte*[KUR-07].

Tabel 2.7 Tabel *Rcon* untuk 10 *round key* AES

Rcon 1	Rcon 2	Rcon 3	Rcon 4	Rcon 5	Rcon 6	Rcon 7	Rcon 8	Rcon 9	Rcon 10
01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Sumber : [KUR-07]

2.8.8 Contoh proses inisialisasi kunci internal

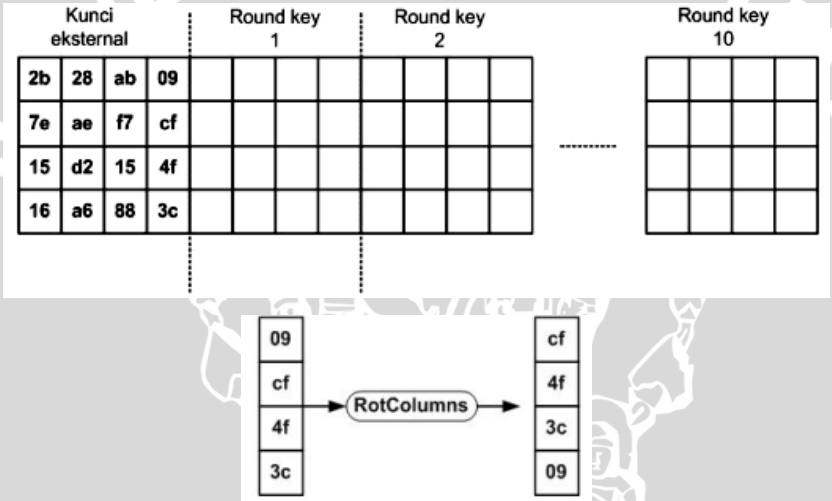
Berikut ini adalah contoh tahapan dalam proses inisialisasi kunci internal :

1) Misalkan *array* kunci eksternal diberikan oleh gambar 2.20.

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

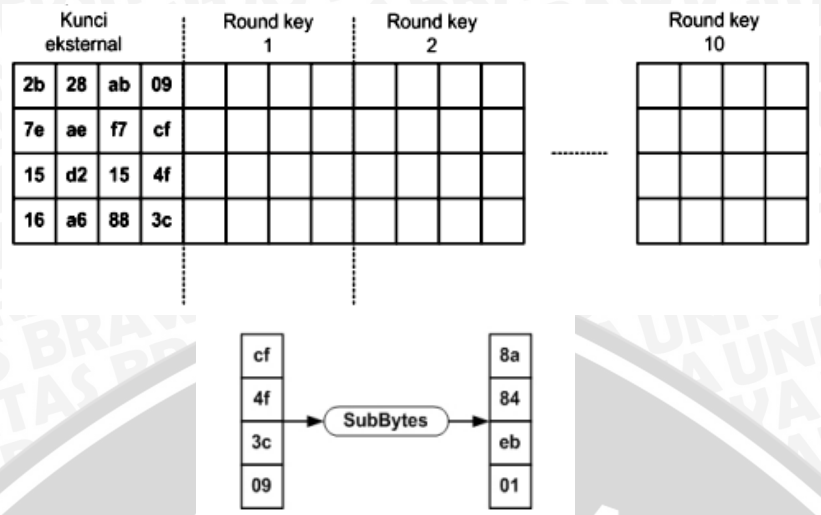
Gambar 2.20 Contoh *array* kunci eksternal
Sumber : [RIZ-09]

2) Untuk memperoleh kolom pertama pada *round key* 1, dilakukan transformasi RotCol pada kolom terakhir dari *array* kunci eksternal (Gambar 2.21).



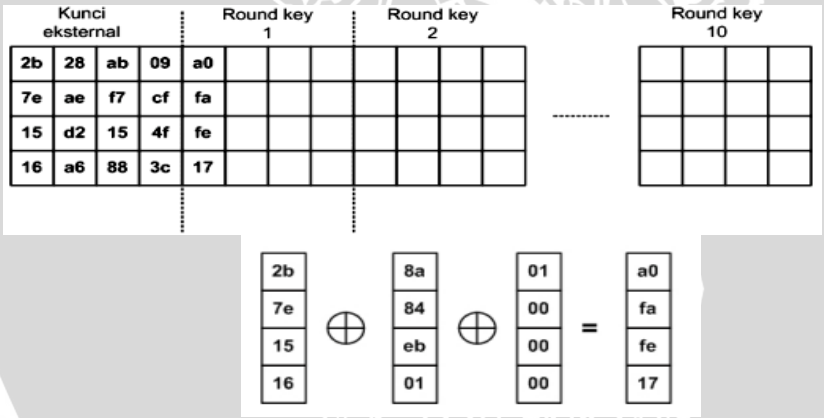
Gambar 2.21 Contoh transformasi *RotCol*
Sumber : [RIZ-09]

3) Transformasi *SubBytes* pada kolom yang terakhir *array* kunci eksternal yang sudah dilakukan transformasi RotCol (Gambar 2.22).



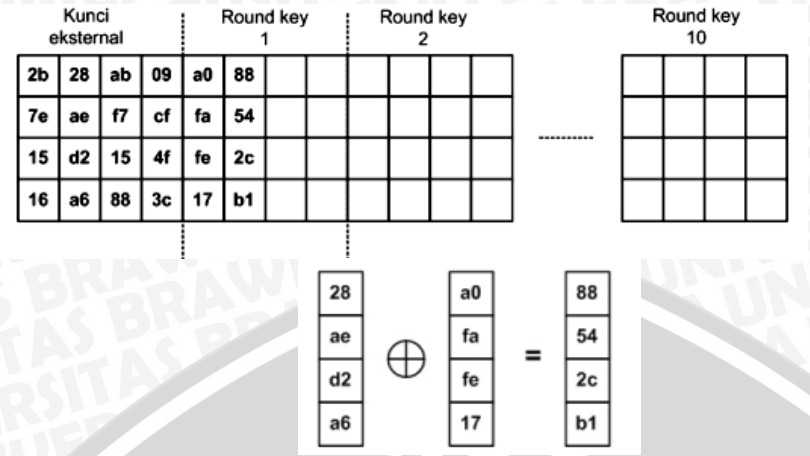
Gambar 2.22 Contoh transformasi *SubBytes*
Sumber : [RIZ-09]

4) Kolom ke-1 *array* kunci eksternal di-XOR-kan dengan kolom hasil SubBytes tersebut dan Rcon1, dan kemudian hasilnya merupakan kolom pertama dari *round key* 1 (Gambar 2.23).



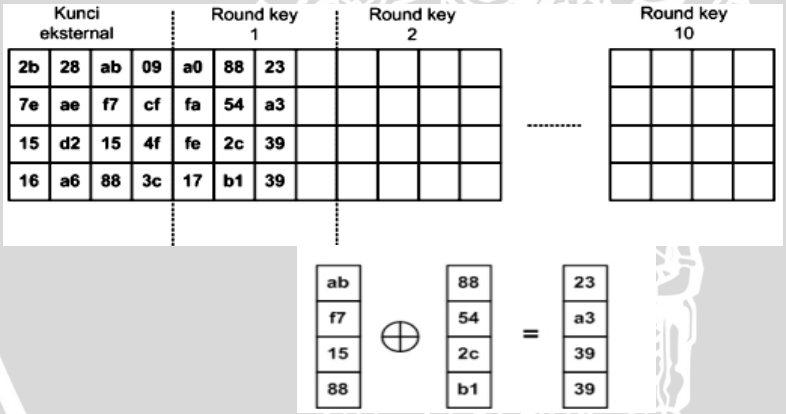
Gambar 2.23 Cara memperoleh kolom pertama *round key* 1
Sumber : [RIZ-09]

5) Untuk memperoleh kolom kedua dari *round key* 1, dilakukan peng-XOR-an kolom pertama *round key* 1 dengan kolom kedua array kunci eksternal (Gambar 2.24).



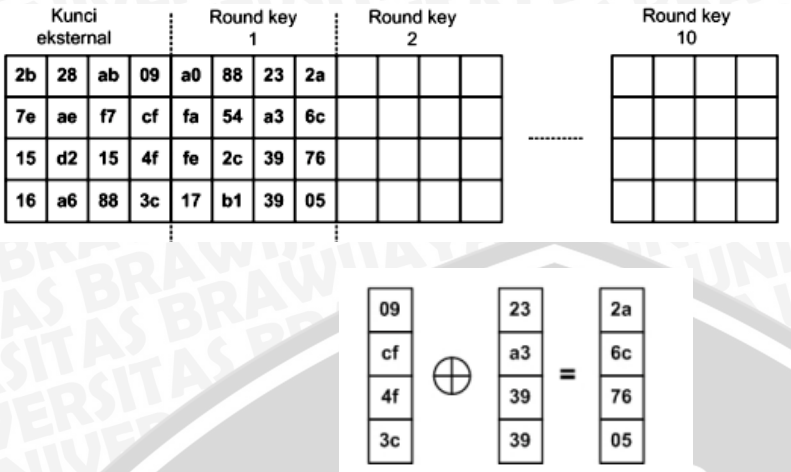
Gambar 2.24 Cara memperoleh kolom kedua *round key* 1
Sumber : [RIZ-09]

6) Untuk memperoleh kolom ketiga dari *round key* 1, dilakukan peng-XOR-an kolom kedua *round key* 1 dengan kolom ketiga *array* kunci eksternal (Gambar 2.25).



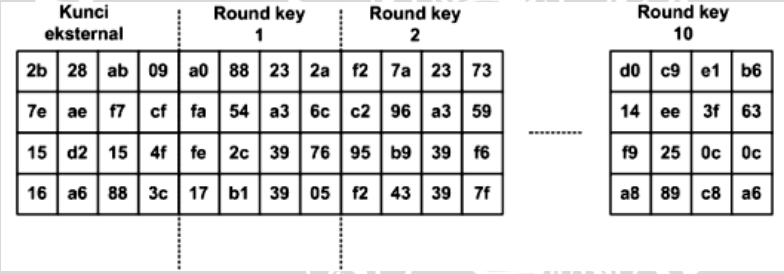
Gambar 2.25 Cara memperoleh kolom ketiga *round key* 1
Sumber : [RIZ-09]

7) Untuk memperoleh kolom keempat dari *round key* 1, dilakukan peng-XOR-an kolom ketiga *round key* 1 dengan kolom keempat *array* kunci eksternal (Gambar 2.26).



Gambar 2.26 Cara memperoleh kolom keempat *round key* 1
Sumber : [RIZ-09]

8) Langkah 2 hingga 7 diulang 9 kali lagi untuk memperoleh *round key* 2 hingga *round key* 10 (Gambar 2.27).



Gambar 2.27 Kunci eksternal dan *round key*
Sumber : [RIZ-09]

2.8.9 Proses Enkripsi

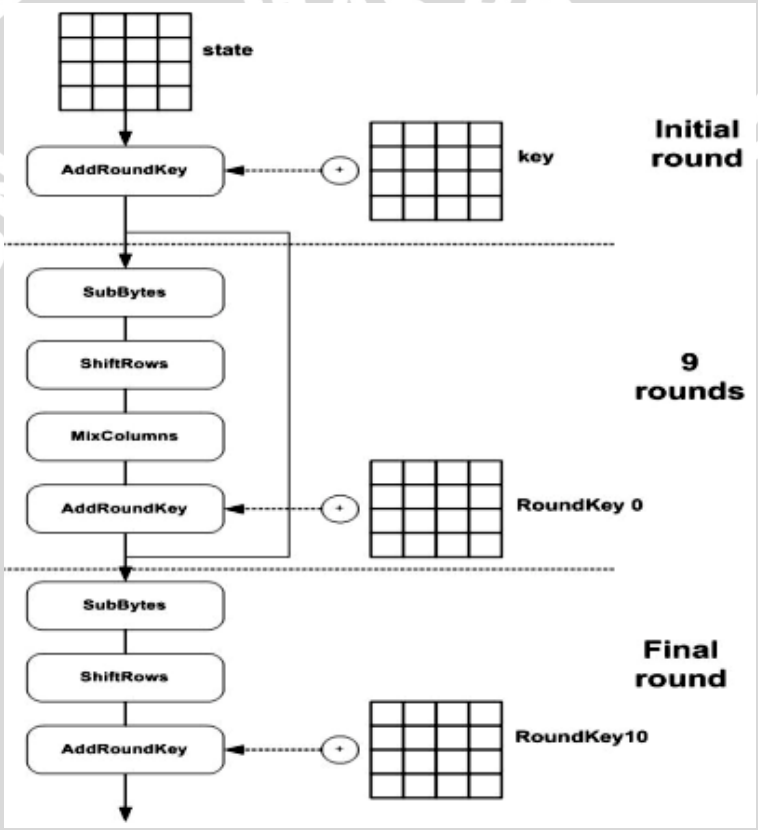
Garis besar algoritma kriptografi AES yang beroperasi pada blok 128 bit dengan kunci 128 bit adalah sebagai berikut:

1. *Initial Round*. Dalam proses ini terdapat proses *AddRoundKey*, yakni melakukan XOR antara *state* awal (*plaintext*) dengan kunci (*cipher key*).
2. Putaran sebanyak Nr-1 kali. Proses yang dilakukan pada setiap putaran adalah:
   a. *SubBytes*: melakukan substitusi *byte-byte* dalam *state* dengan menggunakan tabel substitusi kotak-S (S-Box).
   b. *ShiftRows*: melakukan pergeseran baris-baris *array state* secara *wrapping*.

c. *MixColumns*: mengacak byte-byte di masing-masing kolom *array state*.
d. *AddRoundKey*: melakukan XOR antara *state* sekarang dengan kunci *round key*.

3. *Final round*: proses untuk putaran terakhir. Dalam proses ini terdapat 3 proses yang dilakukan secara berurutan, yakni:
   a. *SubBytes*
   b. *ShiftRows*
   c. *AddRoundKey* [MUN-06].

Skema proses enkripsi AES dapat dilihat pada gambar 2.28



Gambar 2.28 Skema proses enkripsi AES
Sumber : [KUR-07]

Proses enkripsi dalam algoritma kriptografi AES dilakukan sebanyak 11 kali, satu kali dalam *initial round*, 9 kali dalam *round*(s) dan 1 kali *final round*.

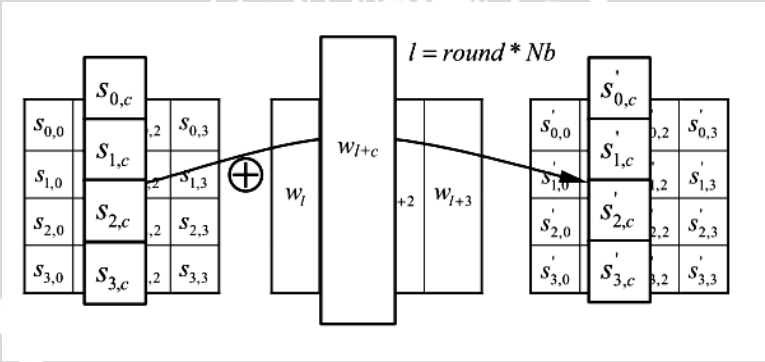
Algoritma AES memiliki 3 parameter:

1. *plaintext*: *array* yang berukuran 16 *byte*, yang berisi data masukan.
2. *ciphertext*: *array* yang berukuran 16 *byte*, yang berisi hasil enkripsi.

3. *key*: *array* yang berukuran 16 byte, yang berisi kunci *ciphering* (disebut juga *cipher key*) [MUN-06].

Dengan 16 *byte*, maka baik blok data dan kunci yang berukuran 128 bit dapat disimpan dalam ketiga *array* tersebut (128 = 16 x 8). Selama kalkulasi *plaintext* menjadi *ciphertext*, status dari data sekarang disimpan di dalam *array of bytes* dua dimensi, *state*, yang berukuran *nrows* x *ncols*. Untuk blok data 128 bit, ukuran *state* adalah 4x4. Elemen *array state* diacu sebagai S[r,c], dengan $0 \leq c \leq Nb$ (Nb adalah panjang blok dibagi 32. pada AES-128, Nb = 128/32 = 4)[FED-01].

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. Ilustrasi dari proses *AddRoundKey* dapat dilihat pada Gambar 2.29 [KUR-07].



Gambar 2.29 Transformasi *AddRoundKey*
Sumber : [FED-01]

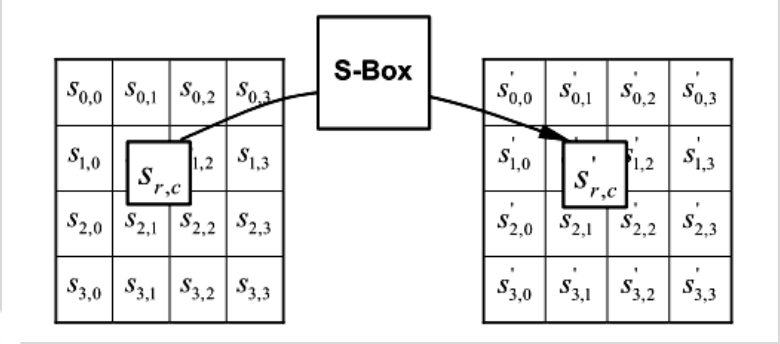
Transformasi *SubBytes*() memetakan setiap *byte* dari *array state* dengan menggunakan tabel substitusi *S-Box*. Tabel *S-Box* dapat dilihat pada Tabel 2.7.

Tabel 2.7 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
x	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
	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

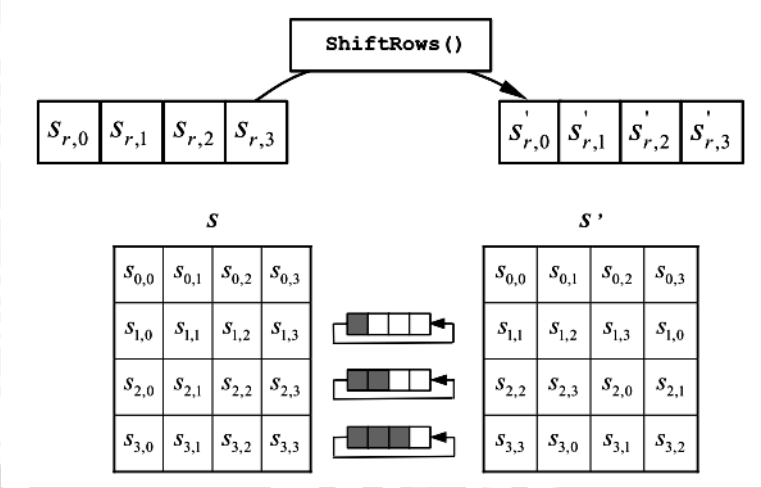
Sumber : [FED-01]

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. Transformasi *SubBytes* [MUN-06] ditunjukkan padagambar 2.30.



Gambar 2.30 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. Transformasi *ShiftRows* diperlihatkan pada gambar 2.31.



Gambar 2.31 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 [MUN-06].

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 2.6

$$a\,(x) = \{03\}x^3 + \{01\}\,x^2 + \{01\}\,x + \{02\} \qquad (2.6)$$

Transformasi ini dinyatakan sebagai perkalian matriks seperti pada persamaan 2.7

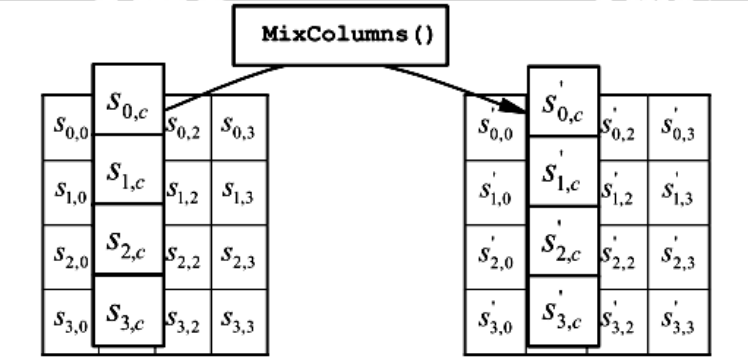
$$s'(x) = a(x) \otimes s\,(x) \qquad (2.7)$$

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

Hasil dari perkalian matriks tersebut, setiap *byte* dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap *byte* tersebut pada persamaan 2.8

$$
\begin{aligned}
s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\
s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\
s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\
s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \qquad (2.8)
\end{aligned}
$$

Transformasi *MixColumns* diperlihatkan pada Gambar 2.32.



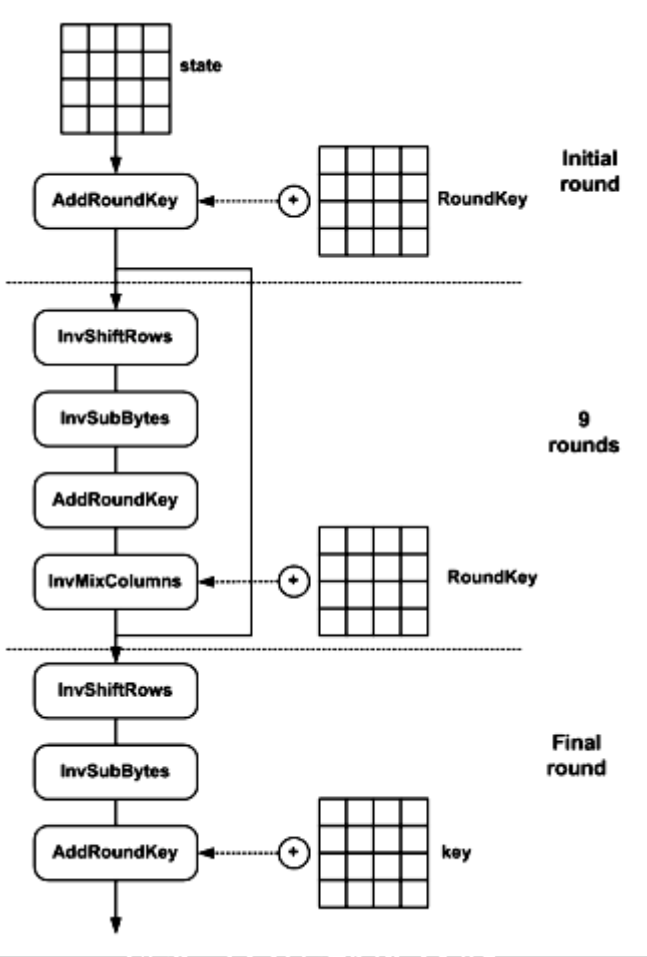
Gambar 2.32 Transformasi *MixColumns*
Sumber : [FED-01]

2.8.10 Proses Dekripsi

Proses dekripsi dalam algoritma AES (Rijndael) merupakan rangkaian pembalikan dari proses-proses yang dilakukan dalam proses enkripsinya. Gambar 2.33 menunjukkan skema global proses dekripsi AES.

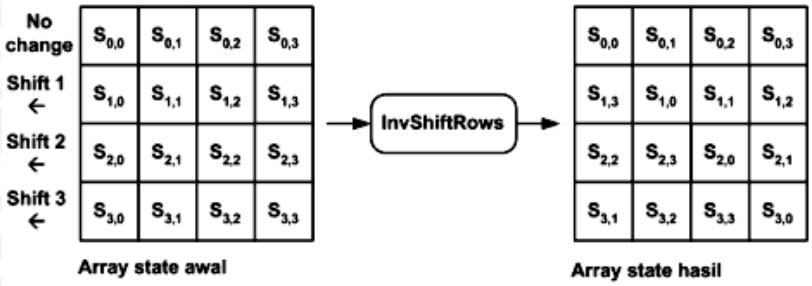
Transformasi *AddRoundKey* tidak mengalami pembalikan (invers) karena proses *AddRoundKey* merupakan operasi XOR. Pembalikan dalam proses ini terletak pada urutan kunci yang digunakan. Jika pada proses enkripsi, urutan kunci internal yang digunakan adalah kunci eksternal (kunci utama), *round key* 1, *round key* 2, *round key* 3, dan seterusnya hingga *round key* 10 (K_0, K_1, K_2, K_3,..., K_{10}); maka pada proses dekripsi urutan kunci yang digunakan adalah kebalikannya yakni menjadi *round key* 10, *round key* 9, *round key* 8, dan seterusnya sampai *round key* 1, kemudian diakhiri dengan kunci eksternal (K_{10}, K_9, K_8, ..., K_0) [KUR-07].

Transformasi *InvShiftRows* didefinisikan sebagai operasi pergeseran *byte-byte* yang terletak pada masing-masing baris. Pembalikan dalam transformasi ini terletak pada bentuk pergeserannya. Jika pada proses enkripsi, transformasi *ShiftRows* melakukan pergeseran ke kiri, maka pada proses dekripsi transformasi *InvShiftRows* melakukan pergeseran ke kanan [FED-01].



Gambar 2.33 Skema global proses dekripsi AES
Sumber : [FED-01]

Banyaknya pergeseran yang dilakukan sama dengan banyaknya pergeseran dalam transformasi *ShiftRows*, yakni baris ke-0 digeser sejauh 0 kolom, baris ke-1 digeser sejauh 1 kolom, baris ke-2 digeser sejauh 2 kolom dan pada baris ke-3 digeser sejauh 3 kolom. Transformasi *InvShiftRows* ditunjukkan gambar 2.34[KUR-07].



Gambar 2.34 Transformasi *InvShiftRows*
Sumber : [KUR-07]

Transformasi *InvSubBytes* didefinisikan sebagai operasi substitusi yang dilakukan pada masing-masing byte dalam *array state*. Masing-masing *byte* dalam *array state* disubstitusi dengan nilai baru sesuai dengan nilai yang ditunjukkan dalam matriks substitusi *S-Box* balikan (*InvS-Box*). Tabel *InvS-Box* ditunjukkan dalam Tabel 2.8.

Tabel 2.8 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]

Transformasi *InvMixColumns* (Gambar 2.35) didefinisikan sebagai operasi yang dilakukan terhadap *byte-byte* yang terletak pada masing-masing kolom. Operasi yang dilakukan terhadap masing-masing kolom adalah operasi perkalian matriks, sama seperti *MixColumns*. Hal yang membedakan antara transformasi *MixColumns* dan *InvMixColumns* adalah matriks 4x4 yang

menjadi matriks pengalinya.

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

Gambar 2.35 Transformasi *InvMixColumns*
Sumber : [FED-01]

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

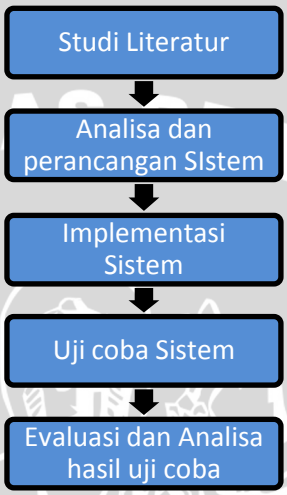
$$\begin{aligned} s'_{0,c} &= (\{0E\} \bullet s_{0,c}) \oplus (\{0B\} \bullet s_{1,c}) \oplus (\{0D\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0E\} \bullet s_{1,c}) \oplus (\{0B\} \bullet s_{2,c}) \oplus (\{0D\} \bullet s_{3,c}) \\ s'_{2,c} &=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

[FED-01]

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai metode, rancangan yang digunakan dan tahap-tahap yang dilakukan dalam penelitian Implementasi algoritma Kriptografi AES untuk keamanan sistem kompresi berbasis Huffman.

Adapun tahapan pembuatan sistem ditunjukkan gambar 3.1 :



Gambar 3.1. Langkah-langkah pembuatan Sistem

1. Melakukan studi literatur terhadap algoritma Huffman dan algoritma kripstografi AES.
2. Menganalisa dan merancang perangkat lunak untuk menerapkan algoritma Huffman dan AES.
3. Mengimplementasikan perangkat lunak berdasarkan analisa dan perancangan yang dilakukan.
4. Melakukan uji coba terhadap perangkat lunak dan mengumpulkan data hasil pengujian yang dilakukan.
5. Melakukan evaluasi terhadap perangkat lunak dan data hasil uji coba yang diperoleh.

Sistem ini dirancang untuk menghasilkan aplikasi yang mampu menerapkan kompresi data dengan metode Huffman yang sekaligus dapat memberikan keamanan pada data hasil kompresi dengan menggunakan metode

enkripsi AES. Sistem ini akan diterapkan pada mesin komputer personal (PC) dengan aplikasi berbasis Java. Sistem ini diharapkan mampu memberikan faktor keamanan pada *file* yang akan dikirimkan melalui internet. Pada sistem ini akan diujikan beberapa bentuk data masukan dengan paramaeter tipe jenis data yang berbeda.

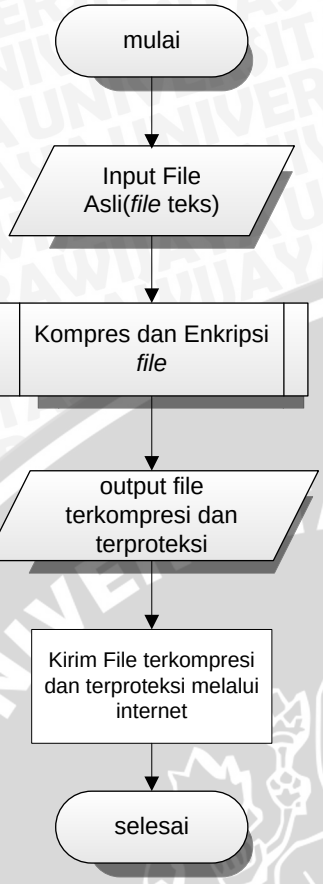
3.1 Perancangan Sistem Secara Keseluruhan

Keseluruhan sistem aplikasi dalam pengerjaan Skripsi ini dibedakan menjadi dua sistem utama, yaitu sistem Kompresi dan sistem Dekompresi.

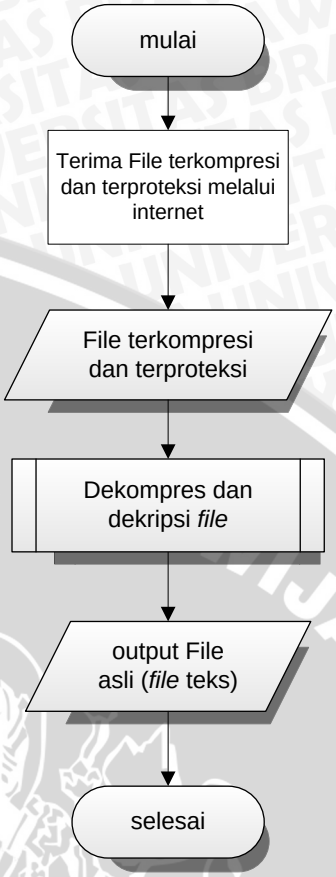
Sistem Kompresi merupakan sistem untuk mengolah data awal menjadi sebuah data baru. Di dalam sistem Kompresi, terdapat sejumlah proses yang diatur sedemikian rupa sehingga proses tersebut dapat disusun sistem pembalikannya. Dalam rancangan sistem yang penulis buat, proses enkripsi AES yang bertujuan untuk mengamankan data, merupakan subproses dari proses Kompresi itu sendiri.

Sedangkan sistem Dekompresi merupakan sistem pembalikan (*invers*) dari sistem Kompresi. Sistem Dekompresi merupakan sistem untuk mengolah data baru yang sudah dihasilkan melalui sistem Kompresi menjadi data awal kembali. Sistem pembalikan ini diperlukan untuk menerjemahkan data baru tersebut menjadi data awal kembali sehingga dapat dibaca secara langsung. Di dalam sistem Dekompresi ini juga terdapat subproses dekripsi.

Skema global dari sistem Kompresi dapat dilihat dalam gambar 3.2. Sedangkan skema global dari sistem Dekompresi dapat dilihat dalam gambar 3.3



Gambar 3.2 Rancangan sistem Kompresi



Gambar 3.3 Rancangan sistem Dekompresi

3.1.1. Proses Kompres dan Enkripsi *File*

Seperti dijelaskan pada sebelumnya bahwa kompresi data berarti sebuah proses mengkodekan informasi menggunakan bit yang lain yang lebih rendah daripada representasi data yang tidak terkodekan dengan suatu sistem encoding tertentu. Kode bit yang akan digunakan dalam penelitian ini adalah berupa kode Huffman yaitu sebuah kode yang terdiri dari kumpulan bilangan biner yang didapatkan dari pembentukan sebuah pohon Huffman. Langkah-langkah yang dilakukan pada proses kompresi *file* ditunjukkan gambar 3.4.

Gambar 3.4 Diagram Alir Proses Kompres dan Enkripsi *File*

Dari diagram alir pada gambar 3.4, dapat dilihat bahwa proses-proses yang terdapat dalam proses Kompres dan Enkripsi *File* adalah:

1. Pembentukan tabel kompresi
2. Pengurutan karakter (*sorting*)
3. Pembentukan pohon Huffman
4. Pembentukan *header file*
5. Enkripsi sub *header file*
6. *Encoding*

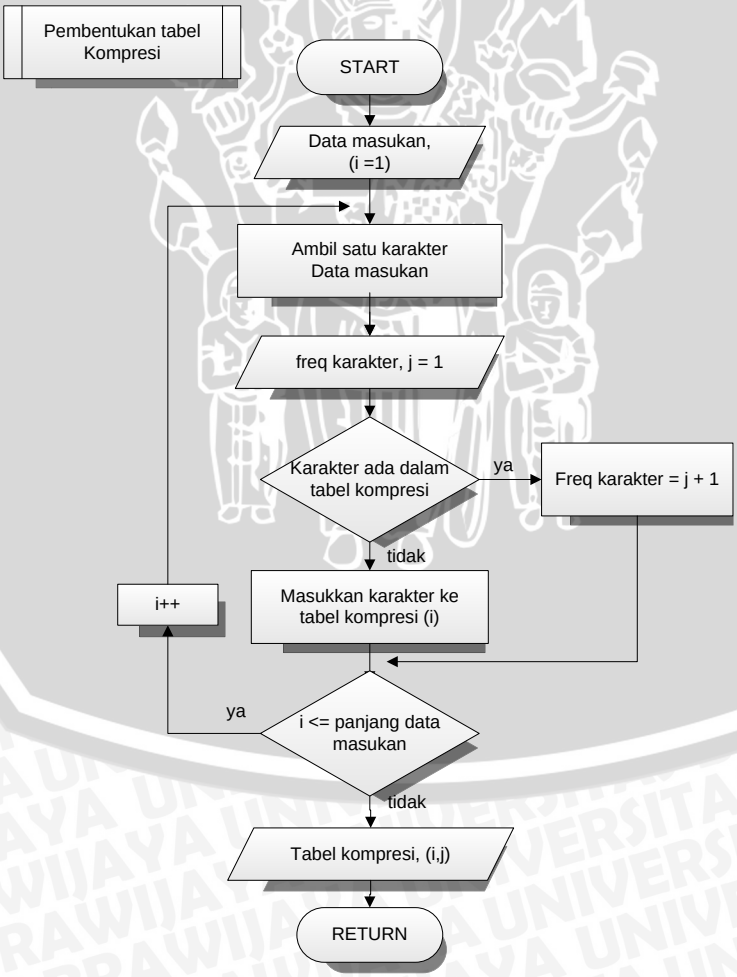
7. Penggabungan header dan hasil *encoding*.

Dari gambar 3.4 juga dapat dilihat bahwa proses Enkripsi dengan AES dilakukan pada *header file* yang telah terbentuk dari proses pembentukan *header*. Hasil dari proses enkripsi *header* ini kemudian digabung dengan hasil *encoding* dari pohon Huffman untuk didapatkan data yang telah terkompresi secara keseluruhan.

Untuk masing-masing proses yang terdapat pada proses kompresi Data dijelaskan berikut.

3.1.1.1 Pembentukan Tabel Kompresi

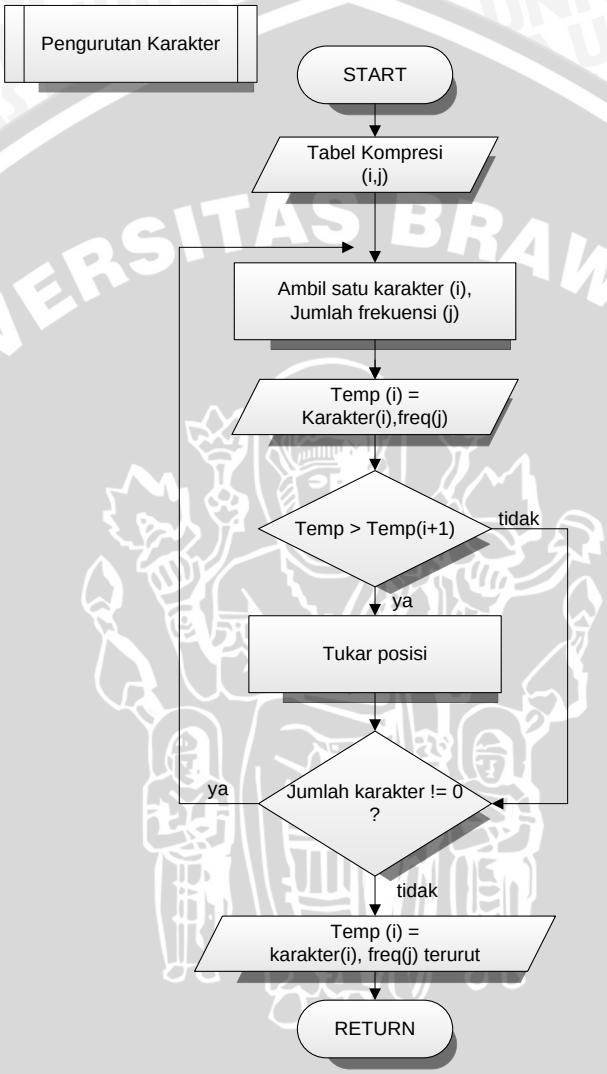
Pembentukan tabel kompresi digunakan untuk menghitung peluang kemunculan setiap karakter pada data masukan. Tabel kompresi ini selanjutnya akan digunakan dalam proses pembuatan pohon Huffman. Diagram alir pembuatan tabel kompresi dapat dilihat pada Gambar 3.5



Gambar 3.5 Diagram alir Pembentukan Tabel Kompresi.

3.1.1.2 Pengurutan Karakter

Semua karakter yang ada dalam *file* akan dibaca untuk dihitung frekuensi kemunculannya. Proses pengurutan yang dilakukan adalah pengurutan dari frekuensi terkecil ke frekuensi terbesar. Diagram alir pengurutan karakter dapat dilihat pada Gambar 3.6

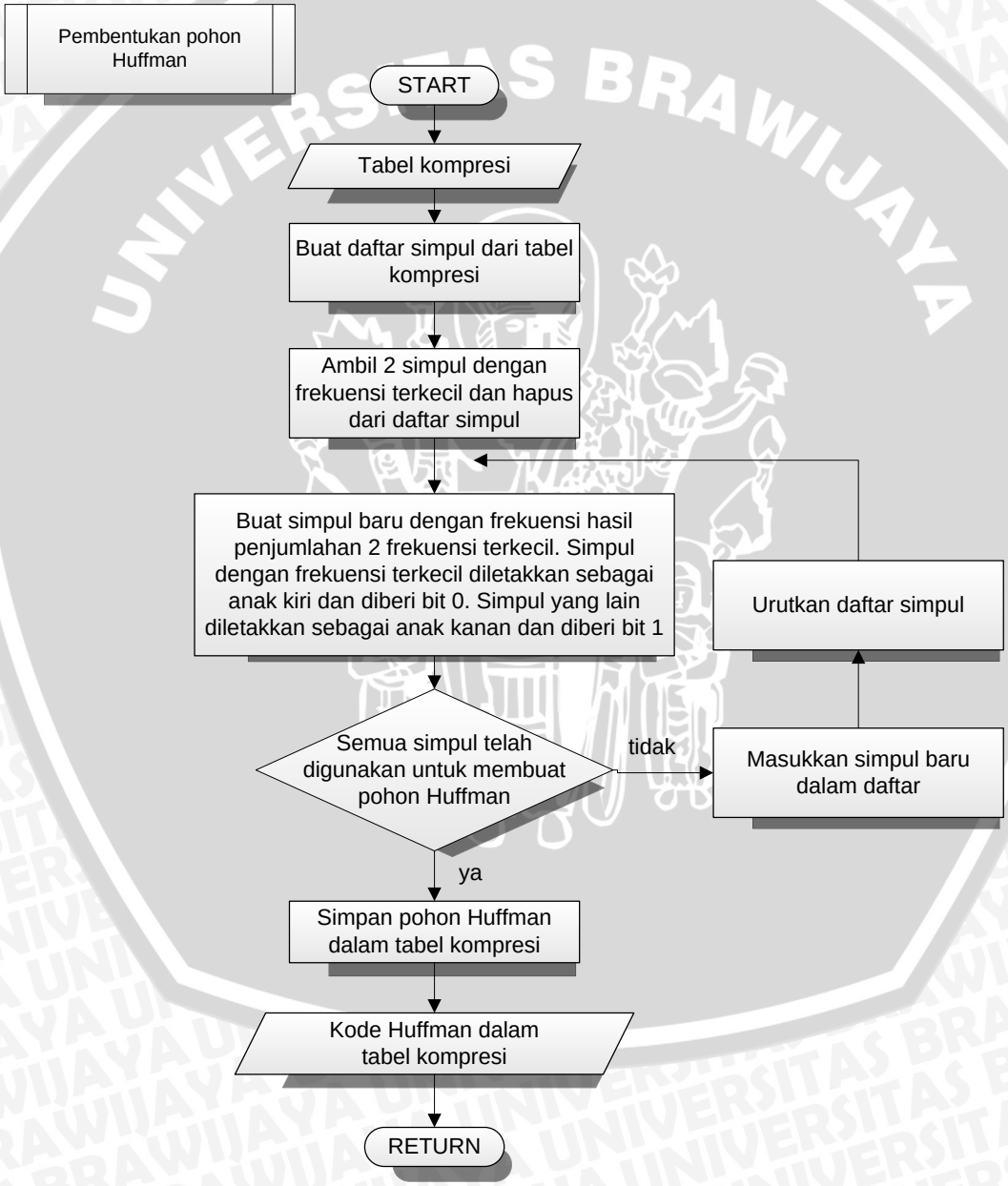


Gambar 3.6 Diagram alir Pengurutan karakter

3.1.1.3 Pembentukan Pohon Huffman

Setelah karakter-karakter dalam tabel kompresi diurutkan, maka langkah berikutnya adalah membentuk pohon Huffman. Pembentukan pohon Huffman dilakukan dengan memanfaatkan struktur data berupa *record*, dimana didalamnya terdapat *record* nama (karakter yang merupakan sebuah simpul tunggal), simpul kiri (berisi karakter yang terdapat pada simpul kiri) dan simpul kanan (berisi karakter yang terdapat pada simpul kiri). Aturan yang diterapkan

adalah jika sebuah karakter terletak pada simpul kiri maka diinisialisasi dengan nilai 0, sedangkan jika sebuah karakter terdapat pada simpul kanan maka akan diinisialisasi dengan nilai 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar (*root* awal) ke daun (sebuah karakter) merupakan kode Huffman untuk pasangan karakter yang berpadanan. Kode Huffman ini kemudian akan disimpan dalam tabel kompresi. Diagram alir pembentukan pohon Huffman ditunjukkan gambar 3.7.



Gambar 3.7 Diagram alir Pembentukan Pohon Huffman

3.1.1.4 Pembentukan Header

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 3.8.

Gambar 3.8 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*.

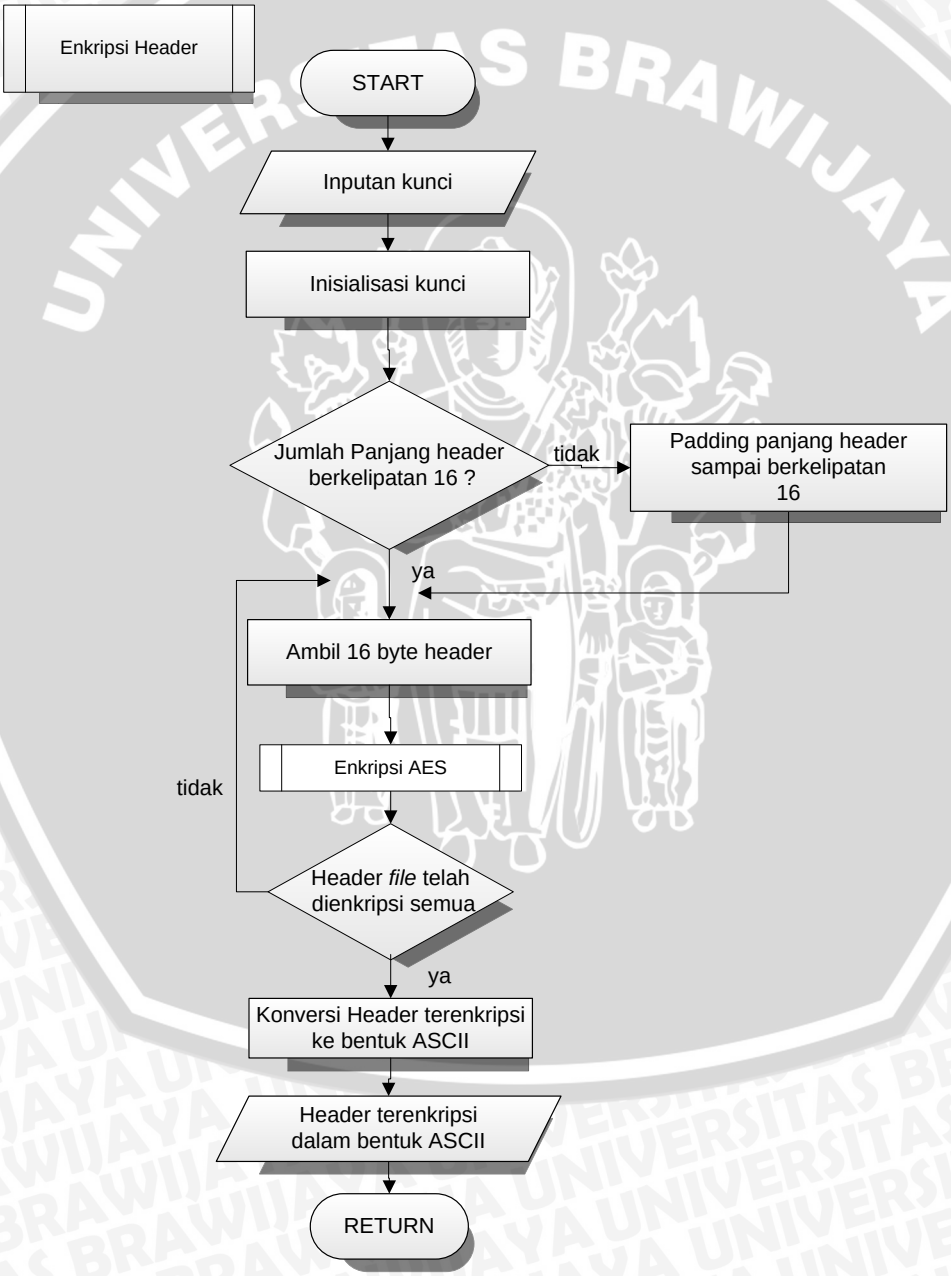
3.1.1.5 Enkripsi Header

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*.

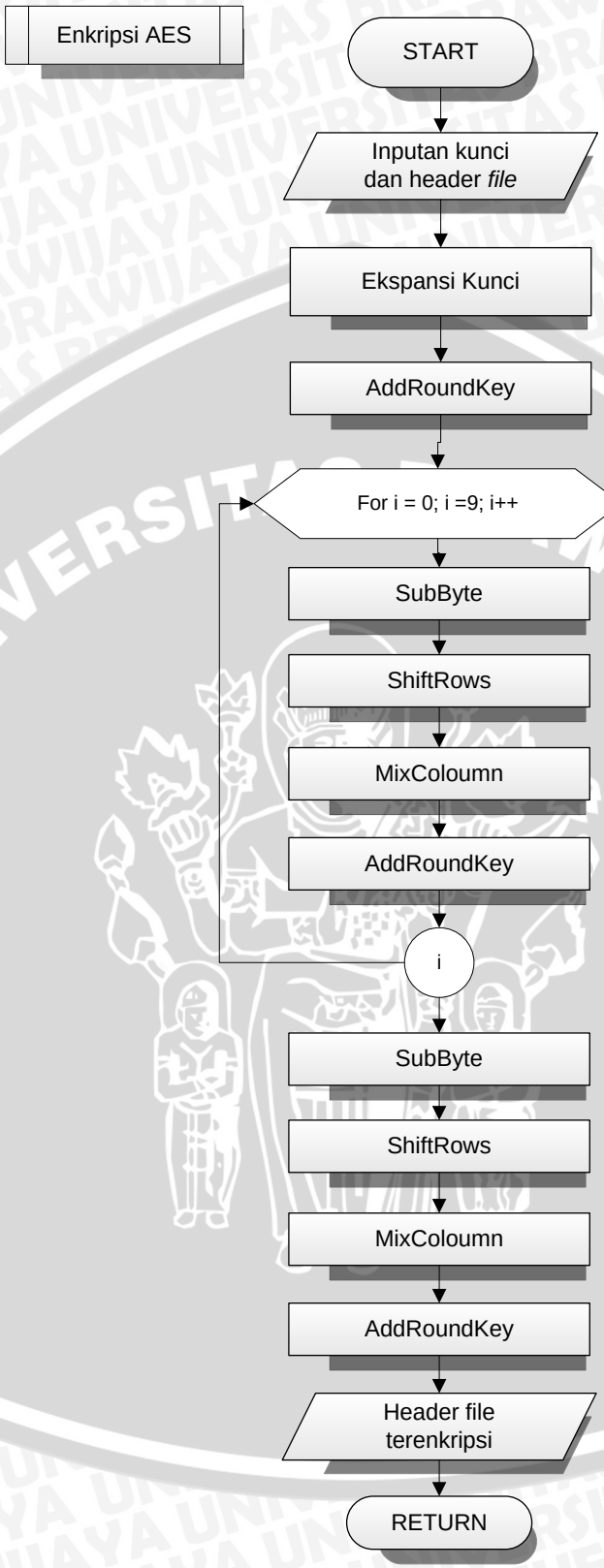
Enkripsi hanya akan dilakukan pada bagian 5 (gambar 3.8). Bagian 1,2,3 dan 4 pada gambar 3.8 jika dienkripsi akan menimbulkan ambiguitas dalam proses dekompresi nantinya.

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.

Langkah-langkah dalam proses enkripsi *header file* ditunjukkan oleh gambar 3.9.



Gambar 3.9 Diagram alir proses enkripsi *header file*



Gambar 3.9.a Diagram alir proses enkripsi AES

Diagram alir proses Enkripsi AES pada gambar 3.9.a terdiri dari delapan proses yaitu: Ekspansi Kunci, *AddRoundKey*, 9 putaran enkripsi, *SubBytes*, *ShiftRows*, dan *AddRoundKey*. Penjabaran proses-proses tersebut adalah :

1. Ekspansi Kunci

Ekspansi kunci adalah proses perluasan kunci untuk membentuk *key schedule*. Hasil *key schedule* terdiri dari *array* 4 *byte word* linear yang dinotasikan dengan [w_i]. Setiap putaran Nr membutuhkan data kunci sebanyak Nb *word* atau 4 *word*. Jadi tiap putarannya membutuhkan 4 [w_i].

Proses ini mempunyai masukan :

- Data Inputan Kunci

Proses ini mempunyai keluaran :

- 10 Round-Key (*Key* atau kunci yang dihasilkan setiap round dari proses perluasan kunci).

2. *AddRoundKey*

Proses ini merupakan suatu operasi penambahan kunci dengan operasi XOR dan setiap kunci *round* terdiri dari w[i], dimana w[i] adalah *subkey* yang diturunkan dari kunci primer. Proses ini mempunyai masukan :

- *Data-State*
- *Round Key* ke i

Proses ini mempunyai keluaran :

- Data-initial-round. (Merupakan hasil dari proses enkripsi pada permulaan putaran)

3. 9 Putaran Enkripsi

Di dalam 9 puataran terdapat 4 proses yang mengalami pengulangan sebanyak 9 kali Proses-proses dalam 1 putaran yang diulang 9 kali berturut-turut adalah *SubBytes*, *ShiftRows*, *MixColumns*, dan *AddRoundKey*. Proses ini mempunyai masukan :

- Data-*initial-round.*
- *Round Key* pertama hingga *round key* kesembilan.

Proses ini mempunyai keluaran :

- *Data-round*-9 (Merupakan hasil dari proses enkripsi pada round ke-9).

4. *SubBytes*

Proses ini merupakan suatu operasi subtitusi pada setiap *byte* dengan menggunakan tabel *S-Box*.

Proses ini mempunyai masukan :

- Data-initial round

Proses ini mempunyai keluaran :

- Data-SubBytes (merupakan hasil operasi *SubBytes*).

5. *ShiftRows*

Proses ini adalah operasi pergeseran byte-byte dimana byte paling kiri akan dipindahkan menjadi bit paling kanan.

Proses ini mempunyai masukan :

- Data-*SubBytes*.

Proses ini mempunyai keluaran :

- Data-*ShiftRows*.(Merupakan hasil operasi *ShiftRows*).

6. *MixColumns*

Proses ini mengoperasikan perkalian matriks antara satu kolom pada *state* dengan matriks tertentu dengan rumus

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

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

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

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

Proses ini mempunyai masukan :

- Data *ShiftRows*

Proses ini mempunyai keluaran :

- Data-*MixColumns* (merupakan hasil operasi *MixColumns*).

7. AddRoundKey

Pada proses *AddRoundKey* yang terakhir ini, masukannya adalah :

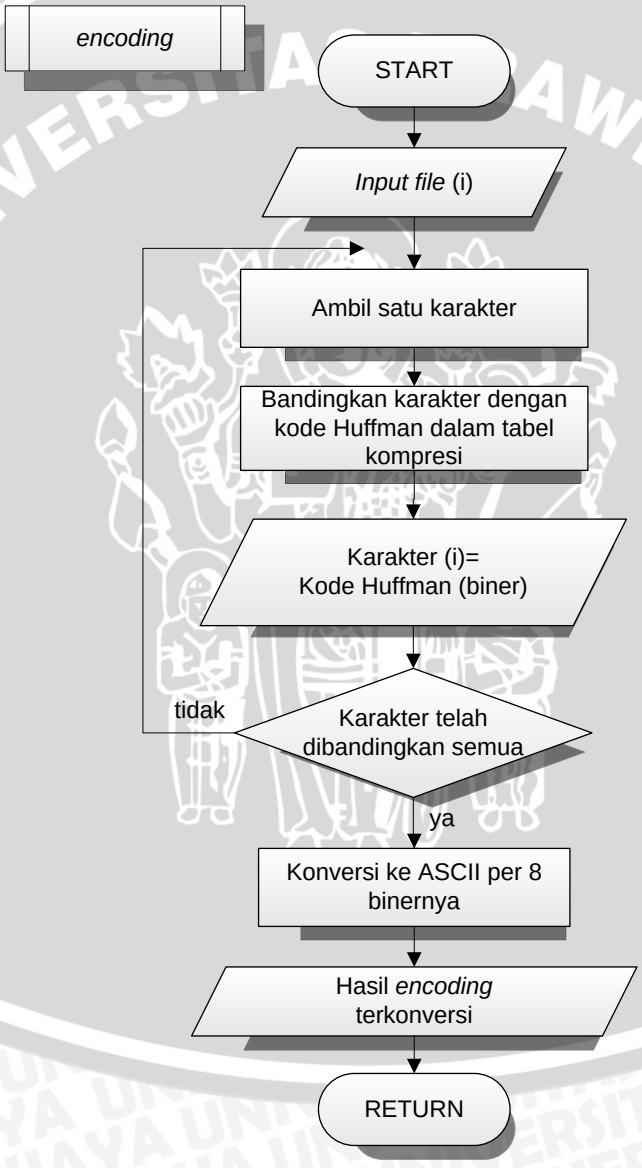
- Data- *MixColumns*
- *RoundKey* ke 10

Keluaran dari proses ini :

- *Header file* terenkripsi.

3.1.1.6 Proses *Encoding*

Proses *encoding* (gambar 3.10) bertujuan untuk menyusun string biner dari kumpulan karakter yang ada pada *file*. Kode untuk string biner diambil dari tabel kompresi yang telah dibentuk sebelumnya. Setiap karakter dalam string input akan dibandingkan dengan kode prefiks dalam tabel. Bila terjadi kecocokan maka kode Huffman diambil dari tabel kompresi tersebut dan digunakan untuk menggantikan karakter pada string data yang asli.



Gambar 3.10 Diagram alir proses *Encoding*

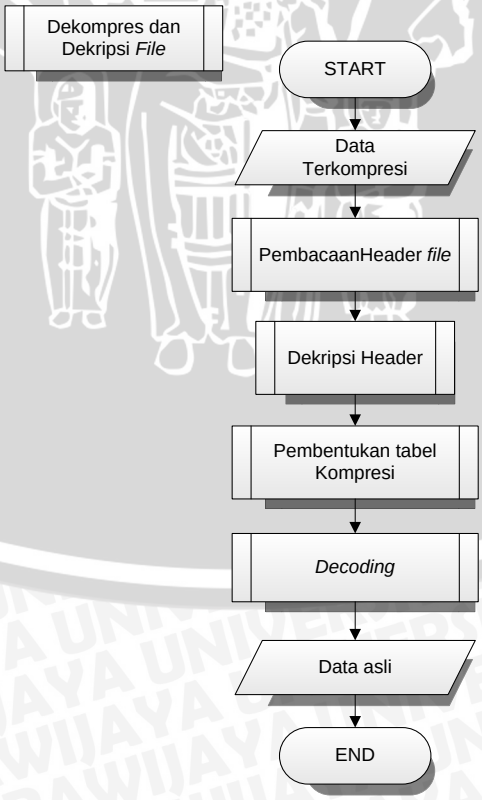
3.1.1.7 Penggabungan Header dan Hasil Encoding

Dalam proses kompresi, hasil dari proses *encoding* disimpan ke dalam file terkompresi perdelapan bit ke dalam karakter ASCII, karena apabila string biner hasil *encoding* tidak dikonversi ke dalam bentuk ASCII, maka ukuran file-nya akan menjadi besar.

File hasil kompresi terdiri atas *header* dan data hasil *encoding* yang telah terkonversi dalam bentuk ASCII. Seperti telah dijelaskan sebelumnya bahwa *header* ini digunakan sebagai kamus untuk menerjemahkan string biner menjadi karakter kembali dalam proses dekompresi. File hasil kompresi dari aplikasi ini akan disimpan dengan ekstensi seperti yang tersimpan pada *header file.*

3.1.2 Proses Dekompres dan Dekripsi *File*

Proses dekompres dan dekripsi *File* 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 3.11.



Gambar 3.11 Diagram alir proses Dekompres dan Dekripsi *File*

Dari diagram alir pada gambar di atas, dapat dilihat bahwa proses-proses yang terdapat di dalamnya adalah:

1. Pembacaan *header file*
2. Dekripsi *header file*
3. Pembentukan tabel dekompresi
4. *Decoding*

3.1.2.1 Pembacaan *Header File*

Berikut ini merupakan penjelasan dari pembacaan *header*:

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.

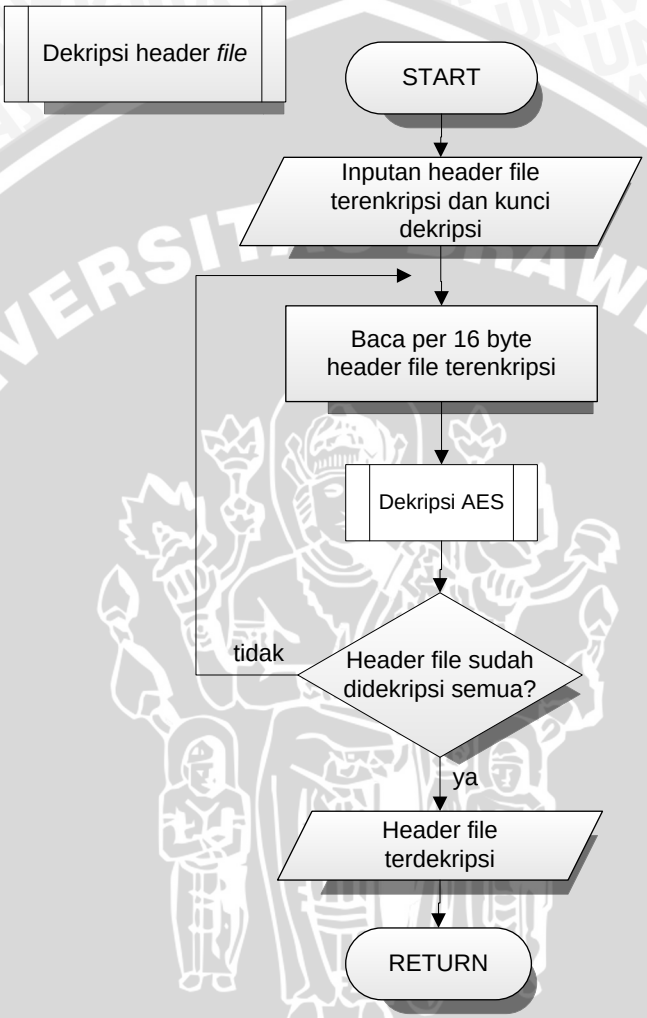
Maka, jumlah panjang header yang harus diambil untuk proses dekripsi adalah sebanyak (9x4+12) = 48 *byte*. Nilai *temp* tidak akan lebih dari 16, dan temp = 16 akan dianggap sama dengan temp = 0.

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

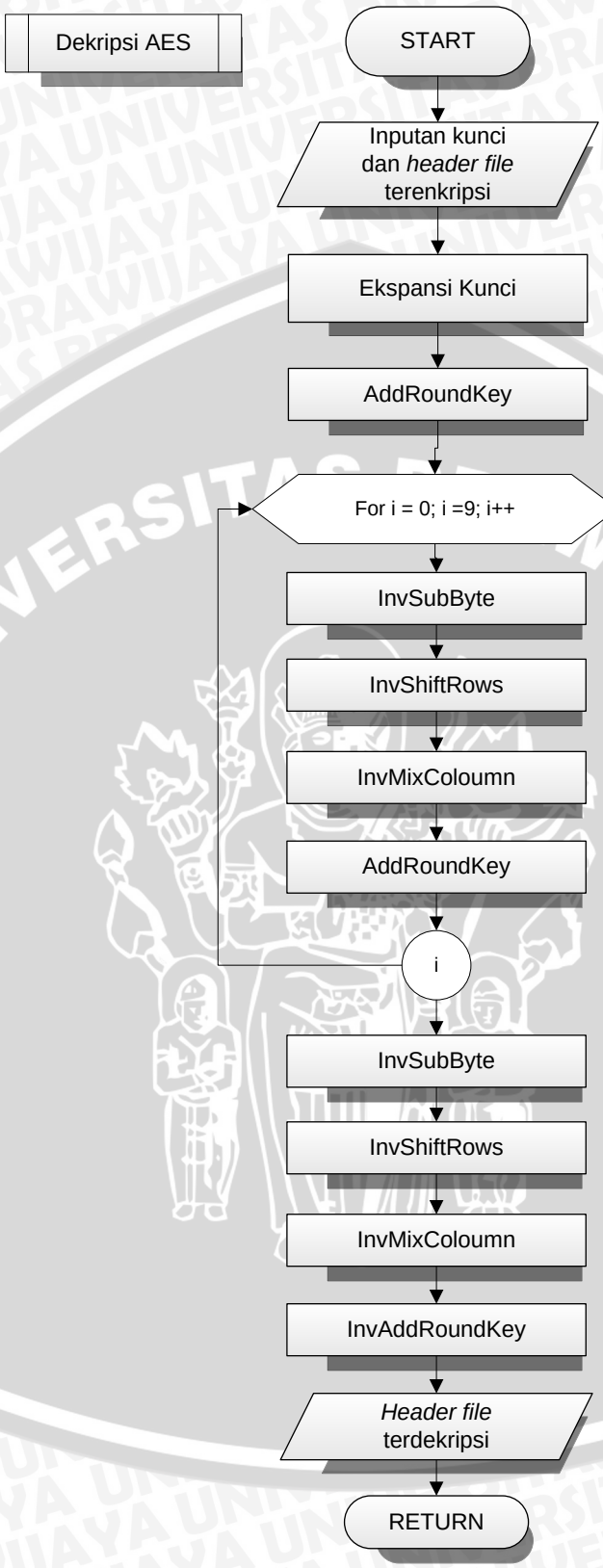
6. Bagian selanjutnya merupakan data hasil *encoding*.

3.1.2.2 Dekripsi *Header File*

Dari pembacaan *header* yang telah dilakukan, didapatkan bagian *header* yang terenkripsi. Bagian header ini kemudian didekripsi untuk dijadikan acuan pada proses selanjutnya. Langkah-langkah proses dekripsi header ditunjukkan dengan gambar 3.12.



Gambar 3.12 Diagram alir proses Dekripsi *header file*



Gambar 3.12.a Diagram alir proses dekripsi AES

Diagram alir proses Dekripsi AES pada gambar 3.12.a terdiri dari delapan proses yaitu: Ekspansi Kunci, *AddRoundKey*, 9 putaran dekripsi, *InvSubBytes*, *InvShiftRows*, dan *AddRoundKey*. Penjabaran proses-proses tersebut adalah :

1. Ekspansi Kunci

Ekspansi kunci adalah proses perluasan kunci untuk membentuk *key schedule*. Proses ini mempunyai masukan :

- Data Inputan Kunci

Proses ini mempunyai keluaran :

- 10 Round-Key (*Key* atau kunci yang dihasilkan setiap round dari proses perluasan kunci).

2. *AddRoundKey*

Proses ini merupakan suatu operasi penambahan kunci dengan operasi XOR dan setiap kunci *round* terdiri dari w[i], dimana w[i] adalah *subkey* yang diturunkan dari kunci primer. Proses ini mempunyai masukan :

- *Data-State*
- *Round Key* ke i

Proses ini mempunyai keluaran :

- Data-initial-round. (Merupakan hasil dari proses enkripsi pada permulaan putaran)

3. 9 Putaran Dekripsi

Di dalam 9 puataran terdapat 4 proses yang mengalami pengulangan sebanyak 9 kali Proses-proses dalam 1 putaran yang diulang 9 kali berturut-turut adalah *InvSubBytes, InvShiftRows, InvMixColumns,* dan *AddRoundKey*. Proses ini mempunyai masukan :

- Data-*initial-round.*
- *Round Key* pertama hingga *round key* kesembilan.

Proses ini mempunyai keluaran :

- *Data-round*-9 (Merupakan hasil dari proses enkripsi pada round ke-9).

4. *InvSubBytes*

Proses ini merupakan suatu operasi subtitusi pada setiap *byte* dengan menggunakan tabel *InvS-Box*.

Proses ini mempunyai masukan :

- Data-*initial round*

Proses ini mempunyai keluaran :

- Data-*InvSubBytes* (merupakan hasil operasi *InvSubBytes*).

5. *ShiftRows*

Proses ini adalah operasi pergeseran *byte-byte* dimana *byte* paling kiri akan dipindahkan menjadi bit paling kanan.

Proses ini mempunyai masukan :

- Data-*InvSubBytes*.

Proses ini mempunyai keluaran :

- Data-*InvShiftRows*.(Merupakan hasil operasi *InvShiftRows*).

6. *InvMixColumns*

Proses ini mengoperasikan perkalian matriks antara satu kolom pada *state* dengan matriks tertentu dengan rumus

$s'_{0,c} = (\{0E\} \bullet s_{0,c}) \oplus (\{0B\} \bullet s_{1,c}) \oplus (\{0D\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$

$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0E\} \bullet s_{1,c}) \oplus (\{0B\} \bullet s_{2,c}) \oplus (\{0D\} \bullet s_{3,c})$

$s'_{2,c} = (\{0D\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0E\} \bullet s_{2,c}) \oplus (\{0B\} \bullet s_{3,c})$

$s'_{3,c} = (\{0B\} \bullet s_{0,c}) \oplus (\{0D\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0E\} \bullet s_{3,c})$

Proses ini mempunyai masukan :

- Data *InvShiftRows*

Proses ini mempunyai keluaran :

- Data-*InvMixColumns* (merupakan hasil operasi *InvMixColumns*).

7. *AddRoundKey*

Pada proses *AddRoundKey* yang terakhir ini, masukannya adalah :

- Data *InvMixColumns*
- *RoundKey* ke 10

Keluaran dari proses ini :

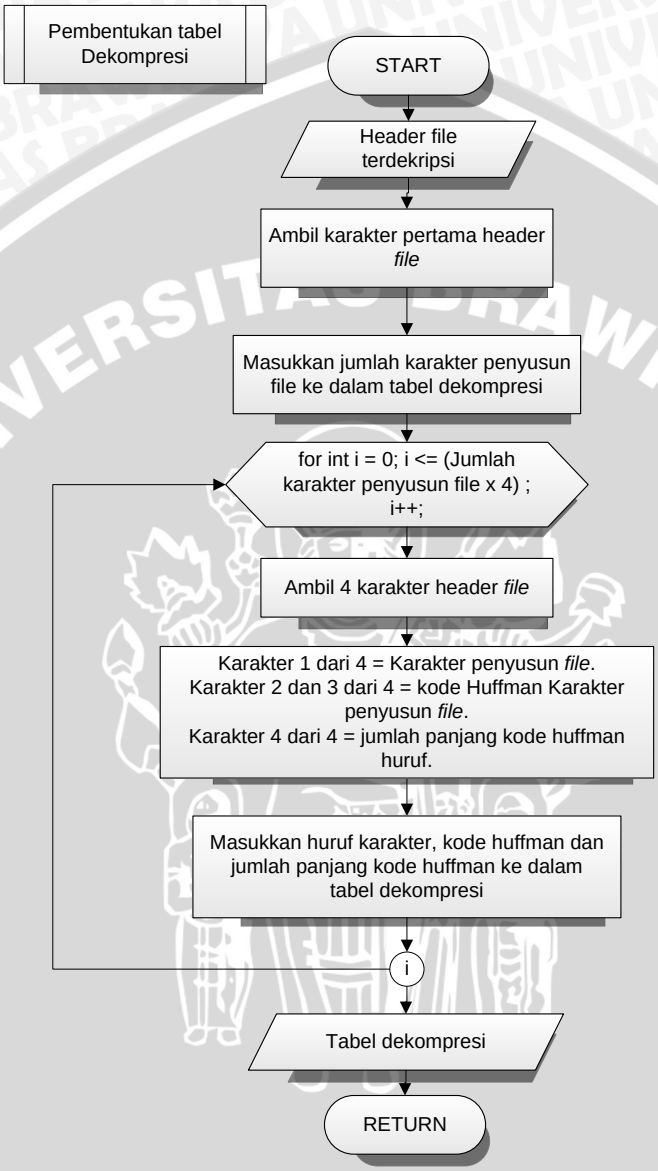
- *Header file* terdekripsi.

3.1.2.3 Pembentukan Tabel Dekompresi

Tabel dekompresi digunakan untuk proses *decoding.* Tabel dekompresi ini berisi informasi mengenai karakter-karakter penyusun *file* asli beserta kode huffman-

nya. Selain itu dalam tabel dekompresi juga berisi informasi panjang kode huffman masing-masing karakter penyusun *file* asli.

Langkah-langkah pembentukan tabel dekompresi ditunjukkan pada gambar 3.13.



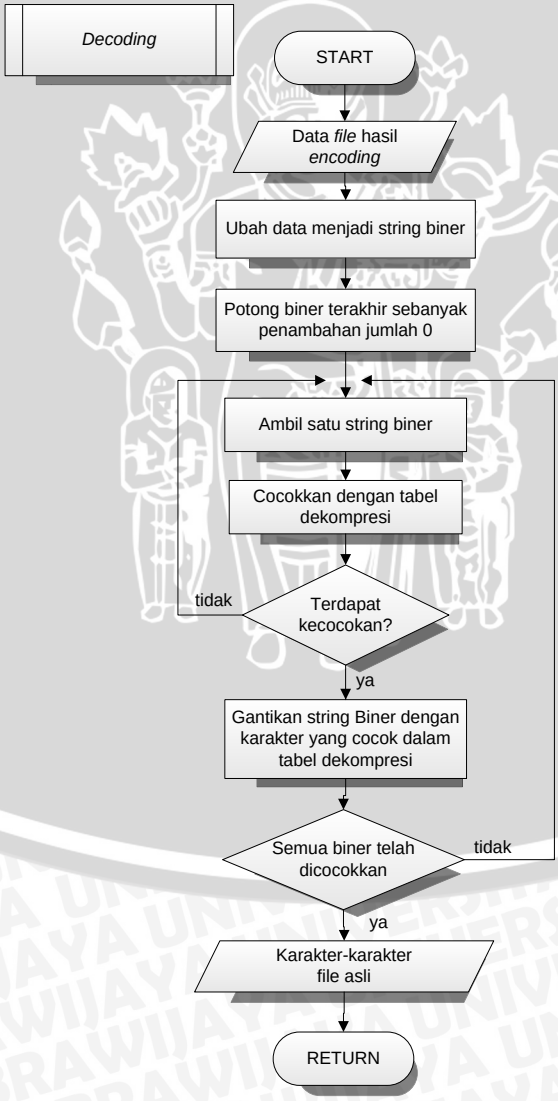
Gambar 3.13 Diagram alir proses pembentukan Tabel Dekompresi

3.1.2.4 Decoding

Decoding merupakan kebalikan dari *encoding*. Proses *decoding* bertujuan untuk mengembalikan data string biner menjadi sebuah karakter kembali. Proses *decoding* dilakukan dengan membaca data pada bagian Data (gambar 3.8), yang berupa kumpulan karakter ASCII. Kemudian kumpulan karakter ASCII ini akan dikonversi kembali ke dalam string biner. Setelah itu string biner hasil konversi tersebut akan

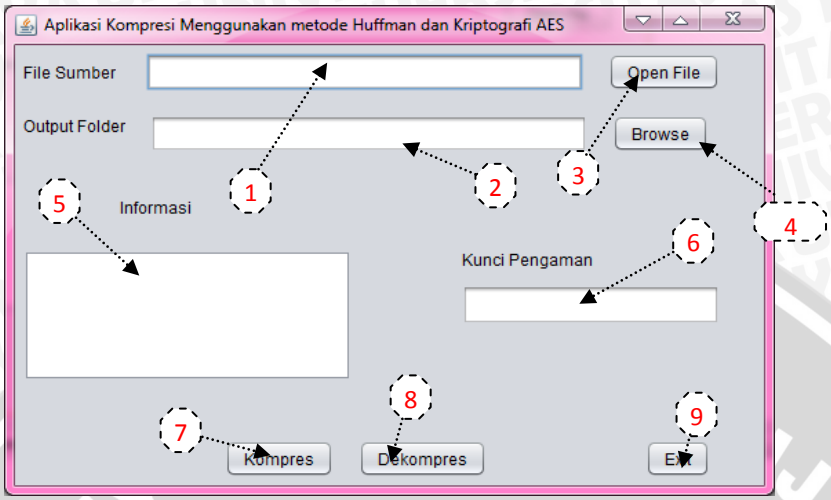
dipotong dari belakang sebanyak angka yang dibaca pada bagian empat (gambar 3.8). Bagian yang dipotong ini merupakan bilangan biner ”0” yang ditambahkan dalam proses pengkonversian ke dalam karakter ASCII. Sehingga bilangan biner yang dipotong tidak menjadi bagian dari isi file.

Proses *decoding* selanjutnya adalah membandingkan string biner hasil pemotongan di atas dengan tabel dekompresi. Pembacaan dilakukan per-bilangan biner. Jika tidak terdapat kecocokan dalam tabel dekompresi maka dibaca satu bilangan biner berikutnya. Proses berlanjut hingga ditemukan kecocokan string biner yang dibaca dengan yang terdapat pada tabel dekompresi. Jika terdapat kecocokan pada kode Huffman, maka karakter yang bersesuaian akan diambil dari tabel dekompresi untuk menggantikan kode Huffman tersebut. Diagram alir proses *decoding* dapat dilihat pada Gambar 3.14.



Gambar 3.14 Diagram alir proses *decoding*

3.2 Rancangan Antar Muka Program



Gambar 3.15 Desain Antar Muka Program

Keterangan gambar 3.15

1. Lokasi Inputan *file* yang akan dikompresi.
2. Lokasi *file* hasil kompresi akan disimpan.
3. Tombol Open File untuk mencari *file* yang akan dikompresi.
4. Tombol Browse untuk menentukan lokasi penyimpanan hasil kompresi.
5. Informasi tentang proses yang berisi konfirmasi keberhasilan proses, waktu proses serta ukuran file sebelum dan setelah proses.
6. Kunci untuk mengenkripsi dengan AES.
7. Tombol Kompresi untuk melakukan proses kompresi.
8. Tombol Dekompresi untuk melakukan proses dekompresi.
9. Tombol Exit untuk keluar dari program.

3.3 Perhitungan Manual

Sebagai contoh, terdapat *file* berjenis ***.txt*** yang akan dikompresi. Di dalam file ini terdapat informasi "***21122012***", dengan panjang 8 karakter. Dalam kode ASCII kalimat tersebut membutuhkan representasi 8 x 8 = 64 bit (8 *byte*).

Akan dilakukan kompresi terhadap *file* tersebut. Adapun langkah-langkah dalam proses kompresi adalah sebagai berikut.

1. Membuat tabel kompresi

Langkah awal yang dilakukan dalam pembuatan tabel kompresi ini adalah menghitung frekuensi kemunculan tiap karakter. Frekuensi kemunculan tiap karakter dimasukkan pada tabel kompresi awal seperti ditunjukan pada tabel 3.1

Tabel 3.1 Tabel kompresi awal

No.	Karakter	Frekuensi
1	2	4
2	1	3
3	0	1

2. Mengurutkan karakter berdasarkan frekuensi

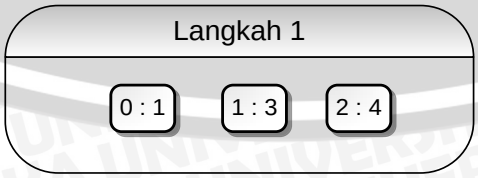
Langkah berikutnya adalah dengan mengurutkan karakter-karakter dalam tabel kompresi berdasarkan frekuensi kemunculannya. Pengurutan ini dilakukan dari karakter yang memiliki frekuensi terkecil ke karakter yang memiliki frekuensi lebih besar dari sebelumnya. Hasil pengurutan karakter dalam tabel kompresi dapat dilihta pada tabel 3.2

Tabel 3.2 Tabel kompresi setelah pengurutan

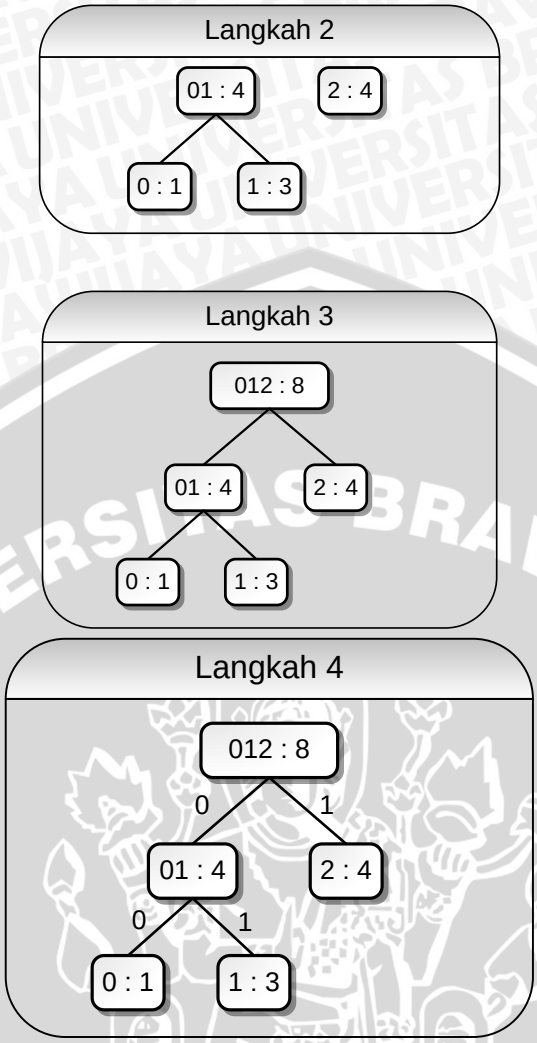
No.	Karakter	Frekuensi
1	0	1
2	1	3
3	2	4

3. Pembentukan pohon Huffman

Langkah berikutnya adalah membentuk pohon Huffman seperti ditunjukkan pada gambar 3.16.



Gambar 3.16. Langkah-langkah pembentukan pohon Huffman



Gambar 3.16.b Langkah-langkah pembentukan pohon Huffman (lanjutan)

4. Encoding

Dari pohon Huffman yang terdapat pada gambar 3.16 diperoleh kode Huffman hasil *encoding* untuk ditambahkan pada tabel kompresi seperti yang ditunjukkan pada tabel 3.3.

Tabel 3.3 Tabel Kompresi yang telah dilengkapi kode Huffman

No.	Karakter	Frekuensi	Kode Huffman
1	0	1	00
2	1	3	01
3	2	4	1

Berdasarkan tabel kompresi baru yang telah terbentuk, proses selanjutnya dalam *encoding* adalah

4.a. Mengubah karakter plainteks ke dalam bentuk kode Huffman.

Pengubahan karakter plainteks ke dalam bentuk kode Huffman dilakukan dengan mengambil kode Huffman yang bersesuaian dengan karakter plainteks. Sehingga, plainteks yang berisi string **21122012** akan menjadi: **1 01 01 1 1 00 01 1**

4.b. Menambahkan angka 0 pada akhir kode Huffman.

Untuk melakukan penambahan angka 0, yang harus dilakukan terlebih dahulu adalah menghitung jumlah karakter biner dari susunan biner yang dihasilkan.

1 01 01 1 1 00 01 1, jumlah biner = 12.

Penambahan angka 0 ditentukan oleh pengurangan 8 dari sisa bagi (mod 8) jumlah biner. Kemudian angka 0 ditambahkan di belakang kode biner sebanyak hasil pengurangan yang dihasilkan.

Sisa bagi = Jumlah biner mod 8

= 12 mod 8

= 4

Jumlah Penambahan 0 = 8 – Sisa bagi

Jumlah Penambahan 0 = 8 – 4 = **4**

Keterangan : Jumlah biner mod 8 dilakukan untuk mengetahui seberapa banyak penambahan angka 0 untuk menyempurnakan biner menjadi 8 bit karakter ASCII.

Setelah dilakukan penambahan angka 0 sebanyak 4, maka susunan biner akan menjadi:

10101110 0011 0000

4.c Translasi ke bentuk ASCII

Dalam bentuk ASCII, bentuk

1010 1110 0011 0000

menjadi

®0

5. Pembentukan Header

Berdasar mekanisme penyimpanan data terkompresi seperti yang ditunjukkan gambar 3.8, maka header dari *file* berekstensi .txt
yang berisi karakter "*21122012*" adalah:
HUF txt03 04 **30 00 00 02 31 00 01 02 32 00 01 02**

Keterangan:

- Karakter 1-3 : HUF (string)
- Karena file tersebut berekstensi .txt, maka karakter selanjutnya = txt (string)
- Karena jumlah karakter yang memiliki kode Huffman ada 3 buah, maka karakter berikutnya adalah = 03 (heksadesimal)
- Jumlah penambahan angka 0 adalah 4, maka karakter berikutnya adalah 04 (heksadesimal)
- Karakter berikutnya : Karakter "0" = 30 (heksadesimal).
- Karena kode Huffman dari karakter "0" adalah 00, sedangkan tempat yang disediakan sebanyak 2 byte maka karakter-karakter berikutnya menjadi 00000000 00000000 B = 00 00 (heksadesimal)
- Panjang kode Huffman dari karakter "0" adalah 2 bit (00), maka karakter berikutnya = 02 (heksadesimal).

Cara tersebut digunakan untuk karakter-karakter selanjutnya.

Tabel 3.4 merupakan kelanjutan dari proses konversi *header* dalam Heksa tersebut .

Tabel 3.4 Bentuk Heksa Karakter penyusun *Header file*

Karakter	Bentuk Heksa	Kode Huffman karakter	Bentuk Heksa kode Huffman karakter	Panjang kode Huffman karakter	Bentuk Heksa panjang kode hufman karakter
0	30	00	00 00	2	02
1	31	01	00 01	2	02
2	32	1	00 01	1	01

5.1 Enkripsi Header

Header yang telah terbentuk pada bagian 5 kemudian akan dilakukan enkripsi sebelum nantinya akan disimpan dan digabung dengan data hasil *encoding.*

Bagian header yang yang akan dienkripsi adalah

30 00 00 02 31 00 01 02 32 00 01 02

Sementara kunci yang digunakan untuk enkripsi adalah **"ilmu komputer"**, yang dalam bentuk Heksadesimal =

69 6C 6D 75 20 6B 6F 6D 70 75 74 65 72

5.2 Proses Inisialisasi Kunci

Untuk memperoleh *round key* 1 dilakukan tahap-tahap berikut (gambar 3.17):

a. Transformasi *RotCol* terhadap kolom ke-4 (paling kanan) dari kunci eksternal.
b. Transformasi *SubBytes* terhadap kolom hasil dari *RotCol* tahap a.
c. Kolom hasil transformasi *SubBytes* (tahap b) di-XOR-kan dengan kolom

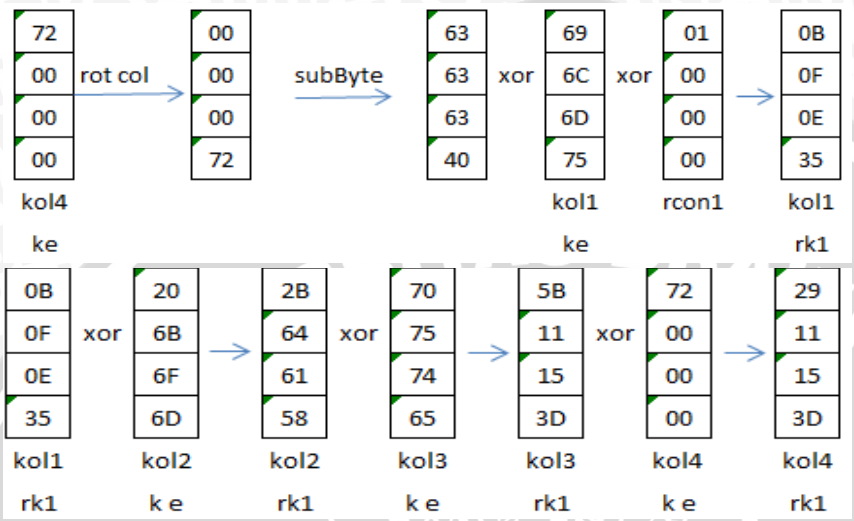
   1 kunci eksternal (kolom paling kiri) dan RCON1 sehingga menghasilkan kolom 1 dari *round key* 1.
d. Kolom 1 *round key* 1 di-XOR-kan dengan kolom 2 kunci eksternal sehingga menghasilkan kolom 2 *round key* 1.
e. Kolom 2 *round key* 1 di-XOR-kan dengan kolom 3 kunci eksternal sehingga menghasilkan kolom 3 round key 1.
f. Kolom 3 *round key* 1 di-XOR-kan dengan kolom 4 kunci eksternal sehingga menghasilkan kolom 4 *round key* 1.
g. Kolom 1 *round key* 1, kolom 2 *round key* 1, kolom 3 *round key* 1 dan kolom 4 *round key* 1 digabungkan menjadi satu dalam *state array* dan menghasilkan *round key* 1

- ***Tahap-tahap untuk mendapatkan Round Key 1***

Tahap-tahap untuk mendapatkan *Round Key* 1 ditunjukkan gambar 3.17.

69	20	70	72
6C	6B	75	00
6D	6F	74	00
75	6D	65	00

Kunci eksternal



0B	2B	5B	29
0F	64	11	11
0E	61	15	15
35	58	3D	3D

Round Key 1

Gambar 3.17 Tahap-tahap memperoleh *round key* 1

Keterangan:

ke = kunci eksternal

rk 1 = *round key* 1

Untuk memperoleh round key ke-N, dilakukan tahap-tahap berikut:

a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari round key (N – 1). Untuk round key ke 1, Transformasi RotCol terhadap kolom ke-4 dilakukan dari hasil kunci eksternal.
b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 Round Key (N-1) (kolom paling kiri) dan RCON1 sehingga menghasilkan kolom 1 dari *round key* N.

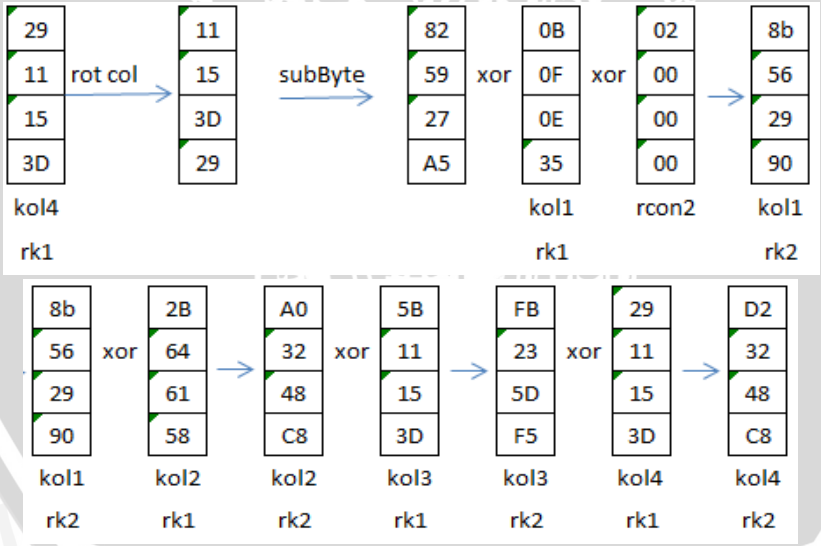
d. Kolom 1 *round key* N di-XOR-kan dengan kolom 2 Round Key (N-1) sehingga menghasilkan kolom 2 *round key* N.
e. Kolom 2 *round key* N di-XOR-kan dengan kolom 3 Round Key (N-1) sehingga menghasilkan kolom 3 round key N.
f. Kolom 3 *round key* N di-XOR-kan dengan kolom 4 Round Key (N-1) sehingga menghasilkan kolom 4 *round key* N
g. Kolom 1 *round key* N, kolom 2 *round key* N, kolom 3 *round key* N dan kolom 4 *round key* N digabungkan menjadi satu dalam *state array* dan menghasilkan *round key* N.

- ***Tahap-tahap untuk mendapatkan Round Key 2***

Tahap-tahap untuk mendapatkan *Round Key* 2 ditunjukkan gambar 3.18.

0B	2B	5B	29
0F	64	11	11
0E	61	15	15
35	58	3D	3D

Round Key 1



8b	A0	FB	D2
56	32	23	32
29	48	5D	48
90	C8	F5	C8

Round Key 2

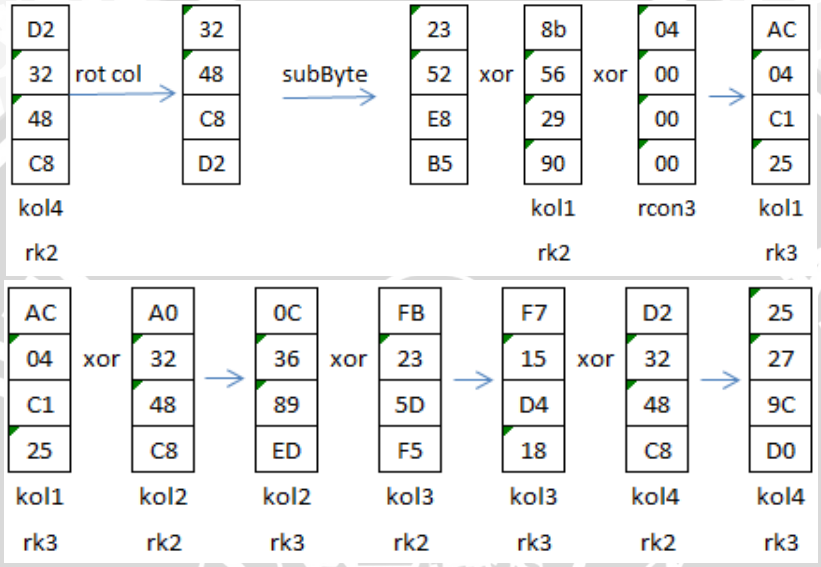
Gambar 3.18 Tahap-tahap memperoleh *round key* 2

- ***Tahap-tahap untuk mendapatkan Round Key 3***

Tahap-tahap untuk mendapatkan *Round Key* 3 ditunjukkan gambar 3.19.

8b	A0	FB	D2
56	32	23	32
29	48	5D	48
90	C8	F5	C8

Round Key 2



AC	0C	F7	25
04	36	15	27
C1	89	D4	9C
25	ED	18	D0

Round Key 3

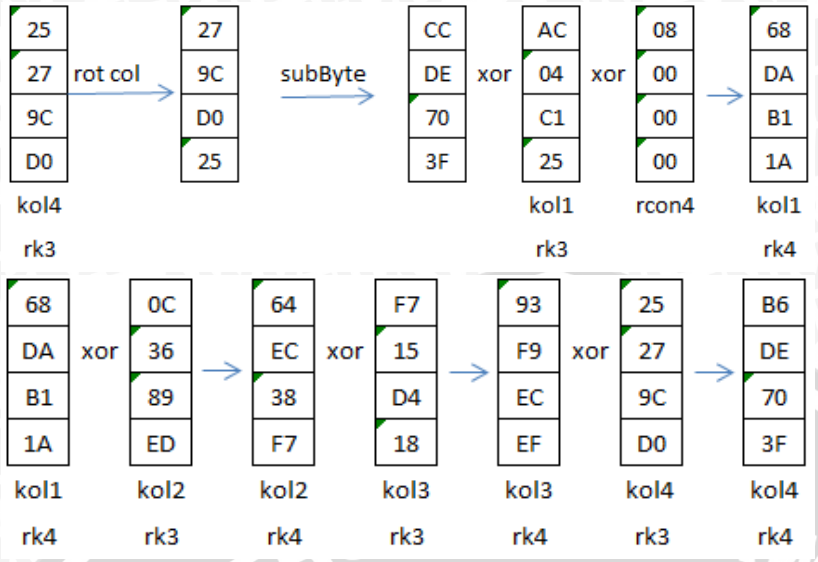
Gambar 3.19 Tahap-tahap memperoleh *round key* 3

- ***Tahap-tahap untuk mendapatkan Round Key 4***

Tahap-tahap untuk mendapatkan *Round Key* 4 ditunjukkan gambar 3.20.

AC	0C	F7	25
04	36	15	27
C1	89	D4	9C
25	ED	18	D0

Round Key 3



68	64	93	B6
DA	EC	F9	DE
B1	38	EC	70
1A	F7	EF	3F

Round Key 4

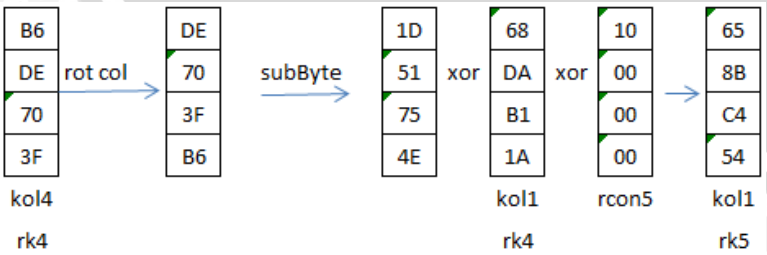
Gambar 3.20 Tahap-tahap memperoleh *round key* 4

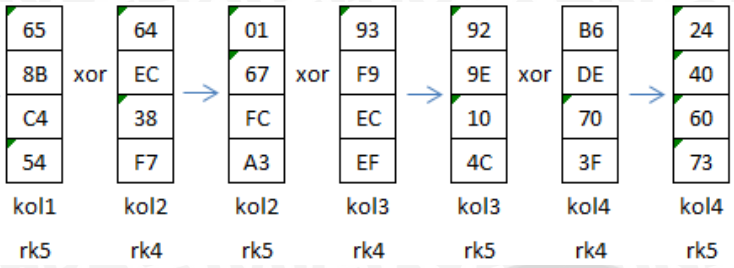
- ***Tahap-tahap untuk mendapatkan Round Key 5***

Tahap-tahap untuk mendapatkan *Round Key* 5 ditunjukkan gambar 3.21.

68	64	93	B6
DA	EC	F9	DE
B1	38	EC	70
1A	F7	EF	3F

Round Key 4





65	01	92	24
8B	67	9E	40
C4	FC	10	60
54	A3	4C	73

Round Key 5

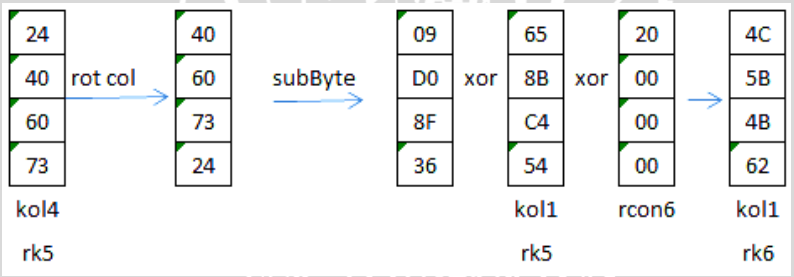
Gambar 3.21 Tahap-tahap memperoleh *round key* 5

- ***Tahap-tahap untuk mendapatkan Round Key 6***

Tahap-tahap untuk mendapatkan *Round Key* 6 ditunjukkan gambar 3.22.

65	01	92	24
8B	67	9E	40
C4	FC	10	60
54	A3	4C	73

Round Key 5



4C	4D	DF	FB
5B	3C	A2	E2
4B	B7	A7	C7
62	C1	8D	FE

Round Key 6

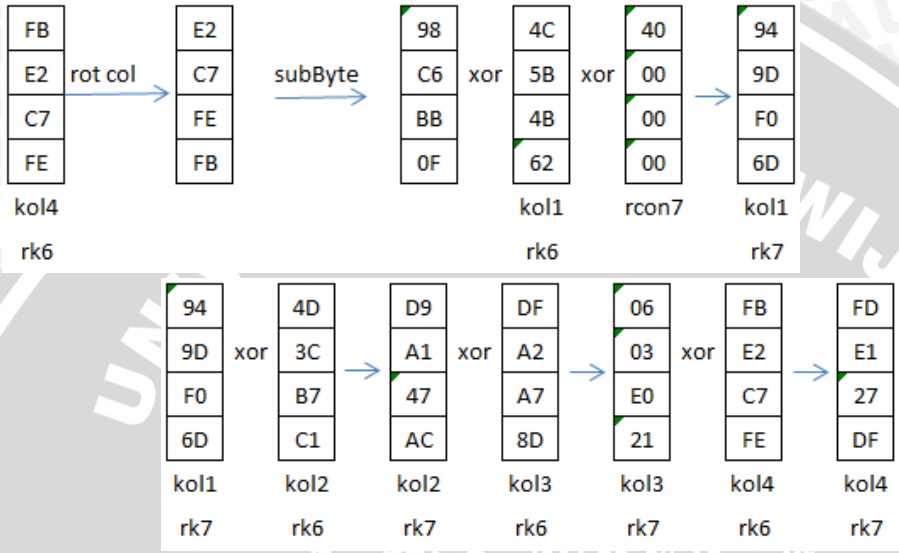
Gambar 3.22 Tahap-tahap memperoleh *round key* 6

- ***Tahap-tahap untuk mendapatkan Round Key 7***

Tahap-tahap untuk mendapatkan *Round Key* 7 ditunjukkan gambar 3.23.

4C	4D	DF	FB
5B	3C	A2	E2
4B	B7	A7	C7
62	C1	8D	FE

Round Key 6



94	D9	06	FD
9D	A1	03	E1
F0	47	E0	27
6D	AC	21	DF

Round Key 7

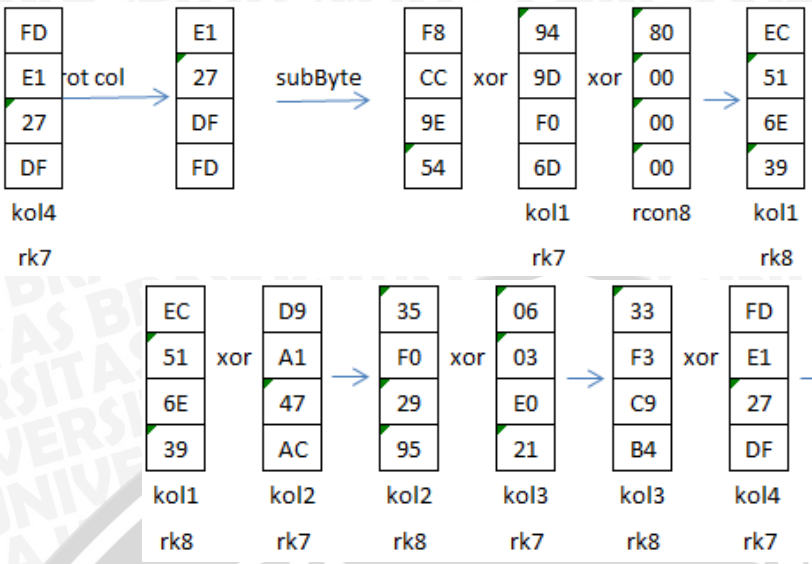
Gambar 3.23 Tahap-tahap memperoleh *round key* 7

- ***Tahap-tahap untuk mendapatkan Round Key 8***

Tahap-tahap untuk mendapatkan *Round Key* 8 ditunjukkan gambar 3.24.

94	D9	06	FD
9D	A1	03	E1
F0	47	E0	27
6D	AC	21	DF

Round Key 7



EC	35	33	CE
51	F0	F3	12
6E	29	C9	EE
39	95	B4	6B

Round Key 8

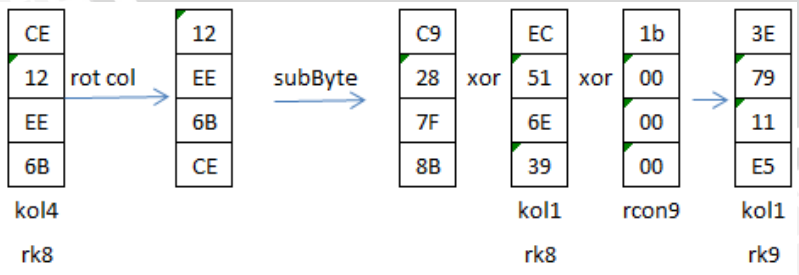
Gambar 3.24 Tahap-tahap memperoleh *round key* 8

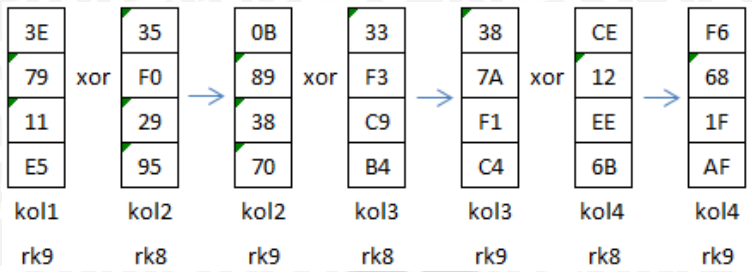
- ***Tahap-tahap untuk mendapatkan Round Key 9***

Tahap-tahap untuk mendapatkan *Round Key* 9 ditunjukkan gambar 3.25.

EC	35	33	CE
51	F0	F3	12
6E	29	C9	EE
39	95	B4	6B

Round Key 8





3E	0B	38	F6
79	89	7A	68
11	38	F1	1F
E5	70	C4	AF

Round Key 9

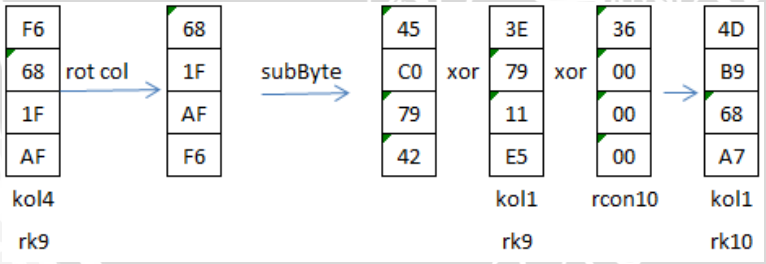
Gambar 3.25 Tahap-tahap memperoleh *round key* 9

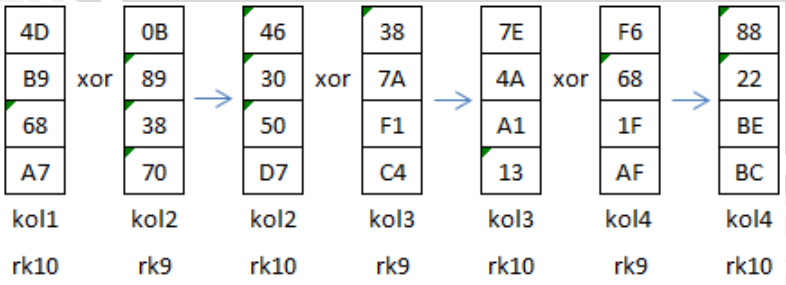
- ***Tahap-tahap untuk mendapatkan Round Key 10***

Tahap-tahap untuk mendapatkan *Round Key* 10 ditunjukkan gambar 3.25.

3E	0B	38	F6
79	89	7A	68
11	38	F1	1F
E5	70	C4	AF

Round Key 9





4D	46	7E	88
B9	30	4A	22
68	50	A1	BE
A7	D7	13	BC

Round Key 10

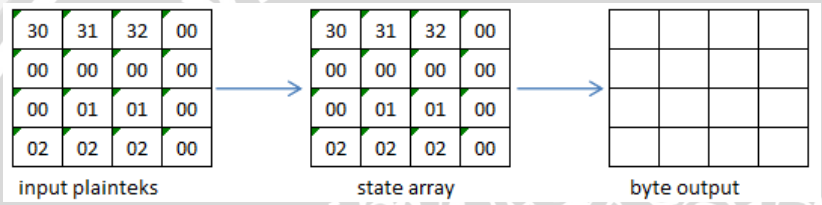
Gambar 3.26 Tahap-tahap memperoleh *round key* 10

5.3 Proses Enkripsi

Pada awal proses enkripsi, 16 *byte* data masukan, in 0 , in 1 , ..., in 15 , disalin ke dalam *array state* (Gambar 3.27)

Byte input merupakan bagian header file kompresi =

30 00 00 02 31 00 01 02 32 00 01 02

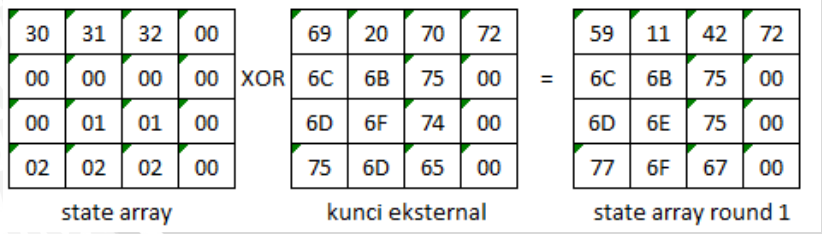


Gambar 3.27 *Byte input, state array* dan *byte output*

Byte input berisi plainteks sebesar 16 karakter (32 heksadesimal) yang kemudian isinya disalin ke *state array* untuk dilakukan proses enkripsi.

a. *Initial Round*

Initial round = state array XOR *external key*



Gambar 3.28 *Initial round*

Initial round adalah proses yang meng-XOR-kan *state array* dengan kunci eksternal dan menghasilkan *state array round* 1 (gambar 3.28).

Proses enkripsi round ke-*N*, dimana (1≤*N*≤9), dilakukan dalam beberapa tahap:

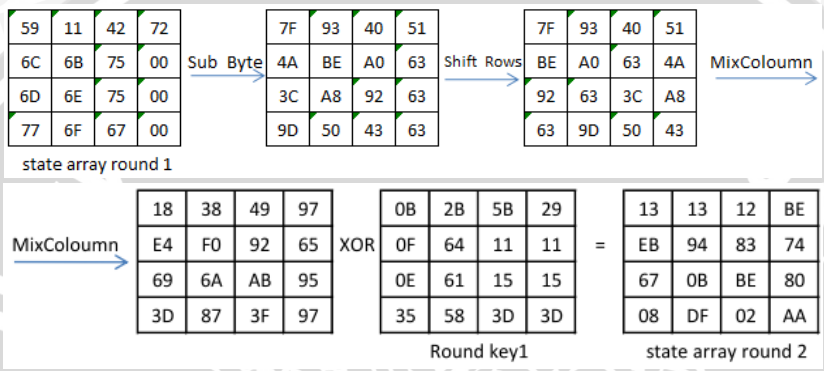
- Transformasi *SubBytes* terhadap *state array* round *N*.
- Transformasi *ShiftRows* terhadap *state array* round *N* hasil

proses *SubBytes*.

- Transformasi *MixColumns* terhadap *state array* round *N* hasil proses *ShiftRows*.
- Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round N* hasil proses *MixColumns* dengan *round key* 1, yang menghasilkan *state array round* (*N*+1).

b. *Round* 1

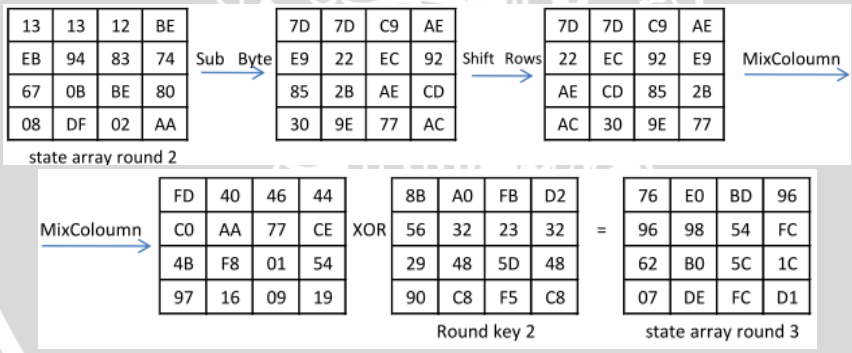
Tahap-tahap proses enkripsi *round* 1 (Gambar 3.29)



Gambar 3.29 Proses enkripsi *round*

c. *Round* 2

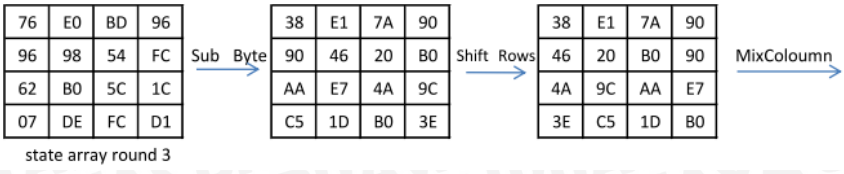
Tahap-tahap proses enkripsi *round* 2 (Gambar 3.30)

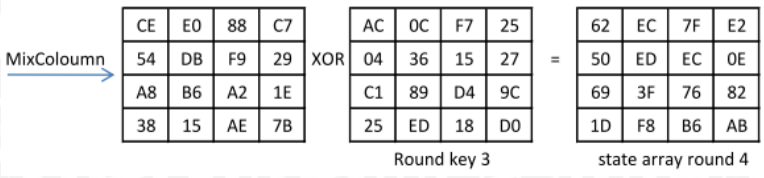


Gambar 3.30 Proses enkripsi *round* 2.

d. *Round* 3

Tahap-tahap proses enkripsi *round* 3 (Gambar 3.31)

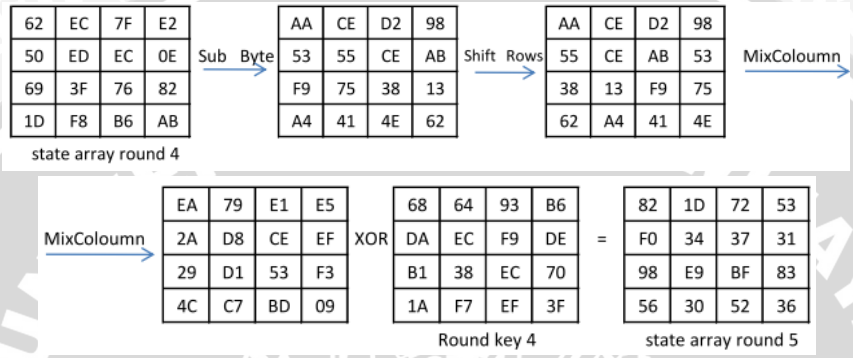




Gambar 3.31 Proses enkripsi *round* 3

e. *Round* 4

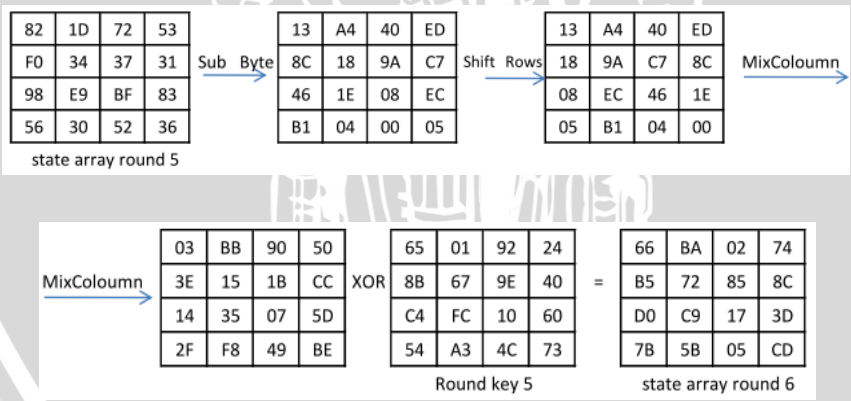
Tahap-tahap proses enkripsi *round* 4 (Gambar 3.32)



Gambar 3.32 Proses enkripsi *round* 4

f. *Round* 5

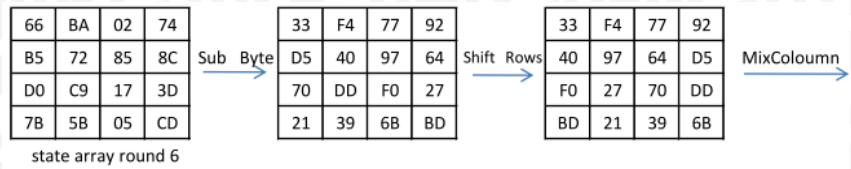
Tahap-tahap proses enkripsi *round* 5 (Gambar 3.33)

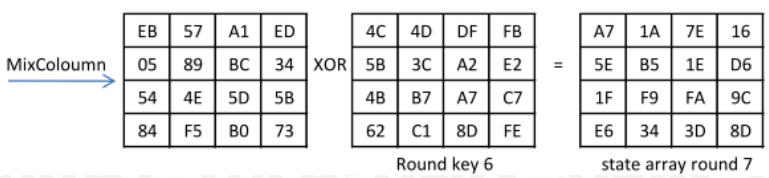


Gambar 3.33 Proses enkripsi *round* 5

g. *Round* 6

Tahap-tahap proses enkripsi *round* 6 (Gambar 3.34)

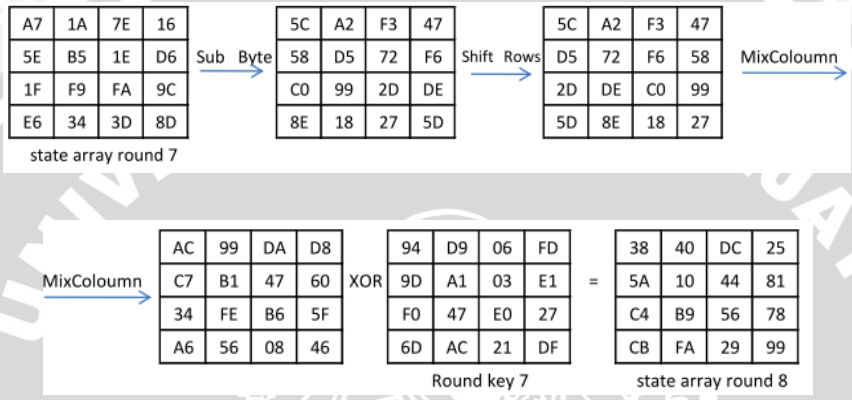




Gambar 3.34 Proses enkripsi *round* 6

h. *Round* 7

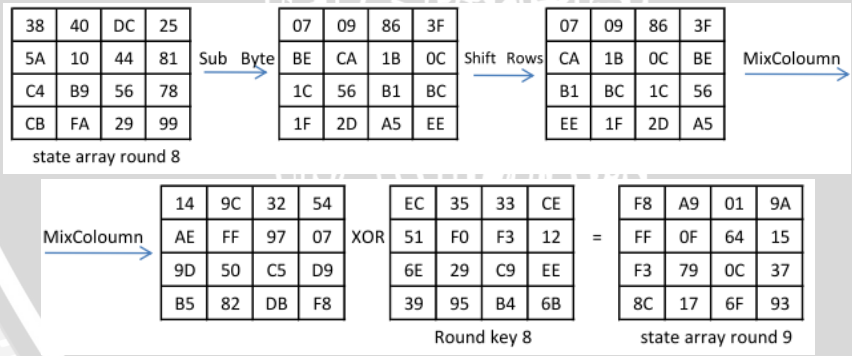
Tahap-tahap proses enkripsi *round* 7 ditunjukkan gambar 3.35



Gambar 3.35 Proses enkripsi *round* 7

i. *Round* 8

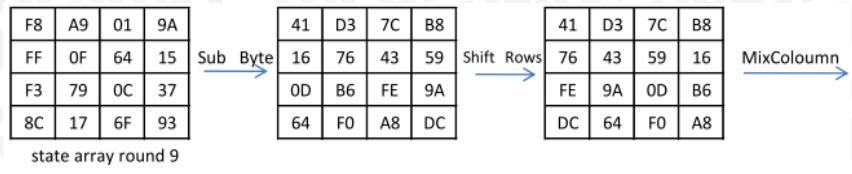
Tahap-tahap proses enkripsi *round* 8 (Gambar 3.36)

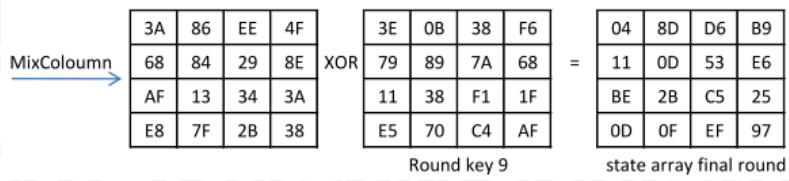


Gambar 3.36 Proses enkripsi *round* 8

j. *Round* 9

Tahap-tahap proses enkripsi *round* 9 (Gambar 3.37)



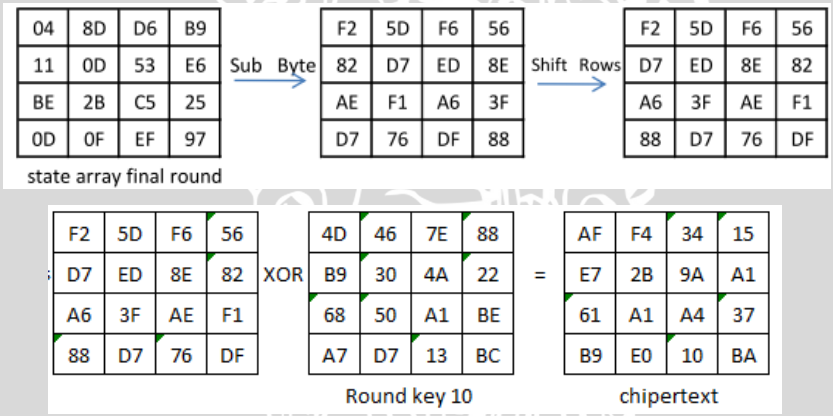


Gambar 3.37 Proses enkripsi *round* 9

k. *Final Round*

Proses enkripsi *final round* (Gambar 3.38) dilakukan dalam beberapa tahap:

- Transformasi *SubBytes* terhadap *state array final round*.
- Transformasi *ShiftRows* terhadap *state array final round* hasil proses *SubBytes*.
- Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array final round* hasil proses *ShiftRows* dengan *round key* 10, yang menghasilkan *ciphertext*.



Gambar 3.38 Proses enkripsi *final round*

5.4 Output

Output dari proses enkripsi dengan menggunakan algoritma kriptografi AES tersebut adalah *ciphertext* seperti pada gambar 3.39:

AF	F4	34	15
E7	2B	9A	A1
61	A1	A4	37
B9	E0	10	BA

Gambar 3.39 *Ciphertext* AES

Input plainteks :

30 00 00 02 31 00 01 02 32 00 01 02 (heksadesimal)

External key :

69 6C 6D 75 20 6B 6F 6D 70 75 74 65 72 (heksadesimal)

Ciphertext output :

AF E761 B9 F4 2B A1 E0 34 9A A4 10 15 A1 37 BA (heksadesimal)

Dalam bentuk ASCII, *chipertext* di atas menjadi

¯ça¹ô+¡_4š¤__¡7º

Maka, jika *file* sebelum dienkripsi adalah:

HUFtxt__ 30 00 00 02 31 00 01 02 32 00 01 02

setelah dienkripsi *file* tersebut menjadi:

HUFtxt__ ¯ça¹ô+¡_4š¤__¡7º

6. Penggabungan Header dan Hasil *Encoding*

Berdasarkan hasil *encoding* yang telah dilakukan sebelumnya, maka *header file* dan hasil *encoding* setelah digabung akan menjadi :

HUFtxt__ ¯ça¹ô+¡_4š¤__¡7º®0

Jika dibandingkan dengan *file* aslinya yang berukuran 8 *byte*, hasil ini bukan menjadi lebih kecil, tapi malah menambah *byte* sebesar 26-8=18 *byte*. Hal ini wajar dalam *file* yang berukuran sangat kecil seperti ini, tapi dalam *file-file* yang berukuran besar, algoritma Huffman akan menjadi efektif (Bab 4.3.3).

3.4 Rancangan Uji Coba dan Evaluasi Hasil

Uji coba akan dilakukan dengan mengimplementasikan algoritma Huffman dan Algoritma enkripsi AES untuk mengevaluasi tingkat keefektifan metode kompresi Huffman dan tingkat keamanan dengan metode enkripsi AES.

3.4.1 Rancangan Analisa Terhadap Serangan *Bruteforce*

Pengujian ini dilakukan untuk mengetahui kekuatan metode enkripsi AES dalam menjaga kerahasiaan isi sebuah *file* . Pengujian akan dilakukan

dengan mencoba mendekompresi file yang telah diamankan dengan kunci tertentu dengan menggunakan nama kunci yang berbeda-beda.

Rancangan tabel untuk pengujian serangan dapat dilihat pada tabel 3.5.

Tabel 3.5 Rancangan Pengujian keamanan Kriptografi AES

Isi file	Kunci AES Untuk Kompresi	Hasil Kompresi	Kunci AES Untuk Dekompresi	Hasi Dekompresi

3.4.2 Rancangan Analisa Hasil Kompresi

Pengujian terhadap kompresi dilakukan dengan membandingkan file sebelum dikompresi dengan file yang telah dikompresi dan dihitung rasio kompresinya. Rancangan tabel untuk pengujian kompresi dapat dilihat pada Tabel 3.6

Tabel 3.6 Rancangan Pengujian Rasio Kompresi

Nama *file* uji	Ukuran *file* (Kb)	Ukuran *file* setelah kompresi (Kb)	Rasio Kompresi

Tabel rancangan 3.6 tersebut akan digunakan untuk pengujian kompresi dengan menggunakan kunci “ilmu komputer“ untuk mengetahui tingkat rasio yang dihasilkan oleh proses kompresi tersebut.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Cakupan pembahasan pada bab ini meliputi implementasi sistem dan pembahasannya serta membahas uji coba dan evaluasi hasil. Implementasi sistem merupakan implementasi rancangan yang sudah dilakukan pada bab III. Sedangkan pembahasan adalah penjelasan dari masing-masing implementasi tersebut. Hasil dari uji coba ini akan digunakan untuk mengevaluasi tingkat keefektifan yang dilakukan oleh sistem yang telah dibuat.

Perangkat lunak yang digunakan dalam pengembangan sistem adalah:

1. Sistem operasi yang digunakan adalah Microsoft Windows 7 Ultimate.
2. *Software* untuk mengembangkan sistem adalah Netbeans 7.0.1.

Sedangkan perangkat keras yang digunakan untuk mengembangkan sistem adalah:

1. Prosesor Intel Core i3-2330M 2.20 GHz.
2. Memori (RAM) 2048 MB.
3. Harddisk dengan kapasitas 120 GB.

4.1 Implementasi Program

Berdasarkan analisa dan rancangan sistem pada Subbab 3.1, maka akan dilakukan implementasi proses-proses tersebut.

4.1.1 Implementasi proses kompresi

Proses kompresi bertujuan untuk memampatkan data dengan sistem *encoding* tertentu agar diperoleh data yang memiliki ukuran lebih kecil daripada ukuran *file* aslinya. Dalam proses kompresi akan dilakukan beberapa proses guna mendapatkan kode Huffman untuk menggantikan pasangan karakter penyusun file.

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

1. Pembentukan Tabel Kompresi

```
void buildTable() {

      char kar = getNextChar();
      while (kar != 65535) { // jumlah maksimal karakter# java
          totalChars++;
          file += kar;
          int i = lookUp(kar);
          if (i == -1) { // new entry
             tableArray[karakterPenyusun] = new Entry();
             tableArray[karakterPenyusun].symb = kar;
             tableArray[karakterPenyusun].weight = 1;
              tableArray[karakterPenyusun].representation     =
   "";
              karakterPenyusun++;
          } else { // existing entry
              tableArray[i].weight += 1;
          }

           kar = getNextChar();

  }
           }
int lookUp(char ch) {
        for (int j = 0; j < karakterPenyusun; j++) {
            if (tableArray[j].symb == ch) {
                return j;
            }
        }
        return -1;
    }
```

Kode Program 4.1 Implementasi Proses Pembentukan Tabel Kompresi

Dalam kode program 4.1 tersebut dipanggil fungsi **getNexChar ()** sebagai fungsi yang digunakan untuk mendapatkan karakter-karakter yang terdapat pada *file* sumber. Karakteristik dari karakter jumlah dan frekuensi akan disimpan dalam *array* yang bernama **tableArray[]**.

Dari *array* yang terbentuk kemudian akan dibentuk sebuah ***list*** yang yang akan digunakan untuk proses pengurutan.

2. Pengurutan Karakter

```
TreeNode urut() {
       ListNode l, oldl, minl = null, oldminl = null; // = null:
for compiler
       double minw = 1000000;
       oldl = list;
       l = list;
       while (l != null) {
           if (l.hufftree.weight < minw) {
               minw = l.hufftree.weight;
               oldminl = oldl;
```

```
                minl = l;
            }
            oldl = l;
            l = l.next;
        }
        if (minl == oldminl) {
            list = list.next;
            return minl.hufftree;
        }
        oldminl.next = minl.next;
        return minl.hufftree;
    }
```

Kode Program 4.2 Implementasi Proses Pengurutan Karakter

Kode program 4.2 merupakan implementasi dari proses pengurutan karakter dalam *list*. Pengurutan yang dilakukan bersifat ascending berdasarkan pada frekuensi kemunculan karakter. Sehingga untuk proses selanjutnya yaitu proses pembentukan pohon Huffman akan dapat menghasilkan kode Huffman yang efektif, dimana karakter dengan frekuensi besar akan memiliki kode Huffman yang lebih pendek dibandingkan dengan pasangan karakter dengan frekuensi yang kecil.

3. Pembentukan Pohon Huffman

```
TreeNode kompresi(int n) {
        int index;
        TreeNode tree = null; // = null for compiler
        for (index = 0; index < n - 1; index++) {
            tree = new TreeNode();
            tree.left = antrian.urut();
            tree.left.step = index + 1;
            tree.right = antrian.urut();
            tree.right.step = index + 1;
            tree.weight = tree.left.weight
                    + tree.right.weight;
            tree.symb = markerChar;
            tree.rep = "";
            antrian.insert(tree);
        }
        return tree;
}
```

Kode Program 4.3 Implementasi Proses Pembentukan Pohon Huffman

Implementasi pembentukan pohon Huffman pada kode program 4.3 dilakukan dengan membuat sebuah objek dari *TreeNode* yang bernama *tree*. Objek ini akan memiliki karakterisitik berupa posisi kanan atau kiri-nya dalam simpul, bobot atau frekuensinya.

Pembentukan pohon Huffman akan selesai jika proses pengulangan sebanyak jumlah karakter penyusun file telah dihitung seluruhnya.

Untuk mendapatkan representasi kode Huffman tiap-tiap karakter maka dijalankan fungsi **insertRep().** Fungsi ini dijalankan dengan memasukkan parameter-paramter yang telah ada, seperti *tree* dan *tableArray* yang merupakan tabel kompresi.

```
void insertRep(TreeNode tree, Entry tab[], int
jumlahRepresentasi, String representasi) {
        String string1, string2;
        tree.rep = representasi;
        if ((tree.left) == null && (tree.right) == null) {
            for (int i = 0; i < jumlahRepresentasi; i++) {
                if (tree.symb == tab[i].simbol) {
                    tab[i].representasi = tree.rep;
                    numberChar += 1;
                }
            }
            return;
        }
        string1 = representasi;
        string1 += "0";
        insertRep(tree.left, tab, jumlahRepresentasi, string1);

        string2 = representasi;
        string2 += "1";
        insertRep(tree.right,tab, jumlahRepresentasi, string2);
    }
```

Kode Program 4.4 Implementasi Fungsi **InsertRep**

4. Pembentukan Header file

```
void doThat() {

        MyData = new byte[4 * table.karakterPenyusun +
table.jumlahDataEncoding];
        Extensi=fileName.substring(fileName.length()-3,
fileName.length());
        AesCript = new byte[4 * table.karakterPenyusun];
        AesAdd = new byte[table.jumlahDataEncoding];
        int countMyData = 0;
        int countAes = 0;
        int countAdd = 0;
        int counterJumlahDataEncoding = 0;

        if (kondisiKripto == true) {
            byte[] temp = new byte[2];
            int rangeKripto = (4 * table.karakterPenyusun);
            System.out.println("rangeKripto " + rangeKripto);
            if (rangeKripto % 16 != 0) {
                rangeKripto += 16 - (rangeKripto % 16);
            }

             byte[] AesField = new byte[rangeKripto];
```

```
            for (int i = 0; i < AesCript.length; i++) {
                String s = "" + (char) AesCript[i];
                System.out.print(s);
            }
            System.out.print("\n");

        }
        else System.out.println("AesCript " + AesCript.length);
    }
```

Kode Program 4.5 Implementasi Proses Pembentukan *Header File*

Salah satu bagian yang menjadi header *file* adalah ekstensi dari file sumber untuk kompresi. Ekstensi ini di dapat dari proses substring pada objek yang bernama *FileName.* Implementasi Pembentukan Header file ditunjukkan Kode program 4.5.

Bagian *header file* selanjutnya adalah jumlah karakter penyusun *file* asli. Jumlah ini telah diinisialisasi ketika proses fungsi **insertRep()** selesai dilakukan. Inisialisai ini dapat dilihat pada objek *numberChar* pada fungsi **insertRep()**.

Byte yang menjadi bagian enkripsi disimpan dalam array yang bernama **AeSCript []**. *Array* ini berukuran sebanyak 4 dikali jumlah karakter penyusun file.

5. Enkripsi Header

```
if (kondisiKripto == true) {

            AES aes = new AES();
             AesField = aes.encrypt(AesCript, key.getBytes());
             System.out.println("AesField " + AesField.length);

            System.out.print("\n");
             MyData = new byte[AesField.length + AesAdd.length];
             System.arraycopy(AesField,     0,     MyData,     0,
AesField.length);
             System.arraycopy(AesAdd, 0, MyData, AesField.length,
AesAdd.length);
         }
```

Kode Program 4.6 Implementasi Proses Enkripsi Header File

Pada proses enkripsi Header dibuat objek dari kelas AES untuk menjalankan proses enkripsi. *Array* **AesCript** yang sebelumnya telah dibentuk kemudian dijadikan inputan pada proses enkripsi. Hasilnya disimpan dalam *array* **AesField.**

Untuk implementasi proses enkripsi per *byte* dijalankan oleh fungsi **encryptBloc** ditunkukkan pada Kode program 4.7..

```
public byte[] encryptBloc(byte[] in) {
        byte[] tmp = new byte[in.length];
        byte[][] state = new byte[4][Nb];

        for (int i = 0; i < in.length; i++) {
            state[i / 4][i % 4] = in[i % 4 * 4 + i / 4];
        }

        state = AddRoundKey(state, w, 0);
        for (int round = 1; round < Nr; round++) {
            state = SubBytes(state);
            state = ShiftRows(state);
            state = MixColumns(state);
            state = AddRoundKey(state, w, round);
        }
        state = SubBytes(state);
        state = ShiftRows(state);
        state = AddRoundKey(state, w, Nr);

        for (int i = 0; i < tmp.length; i++) {
            tmp[i % 4 * 4 + i / 4] = state[i / 4][i % 4];
        }
        return tmp;
    }
```

Kode Program 4.7 Implementasi Fungsi EncyptBloc

Pada proses Enkripsi blok, dijalankan proses seperti **SubBytes**, **Shiftrows**, **MixColumn** dan **AddRoundKey**. Karena enkripsi AES 128 bit menggunakan putaran sebanyak 11 kali, maka pada implementasi pada kode program 4.7 di atas, nilai **Nr** adalah 10.

6. Encoding

```
String encoding(String file) {
        String returnFile = "";
        for (int index = 0; index < file.length(); index++) {
            int indexLookUp = table.lookUp(file.charAt(index));
            if (indexLookUp == -1) {
              System.out.println("Error dalam encoding: tak ketemu:
"+ file.charAt(index));
                  System.exit(0);
            }

returnFile+=table.tableArray[indexLookUp].representasi;
        }
        return returnFile;
    }
```

Kode Program 4.8 Implementasi Proses *Encoding*

Langkah awal yang dilakukan dari proses *encoding* adalah dengan pembacaan per karakter dari isi *file*. Kemudian karakter tersebut dicocokkan dengan kode Huffman pada tabel kompresi. Jika karakter ditemukan di dalam

tabel kompresi maka kode Huffman yang terdapat pada array **tableArray []**.**representasi** pada kelas Tabel akan diambil untuk menggantikan karakter tersebut. Demikian seterusnya hingga semua isi file telah ter-*encoding* oleh sistem.

Setelah data ter-encoding semua, selanjutnya adalah menghitung jumlah penambahan "0" agar hasil encoding dapat terkonversi ke bentuk ASCII. Proses perhitungan jumlah tambahan "0" tersebut ditunjukkan pada kode program 4.9.

```
String Padding(String encodedFile) {
        String kumpulanDataEncoding = "";
        int dataEncodingEncodedFile = encodedFile.length() % 8;

        int i = 0;
        if(dataEncodingEncodedFile!=0){
        while (i < 8 - dataEncodingEncodedFile) {//proses padding
        encodedFile += "0";
            offset += 1;
            i += 1;
        }}
        jumlahDataEncoding = encodedFile.length() / 8;
        EncodedFileBaru = encodedFile; //hasil encode setelah
penambahan offset
        Offset = offset;
```

Kode Program 4.9 Implementasi Fungsi perhitungan Padding

Offset adalah objek yang dibentuk untuk menyimpan jumlah tambahan "0" tersebut.

7. Penggabungan Header dan Hasil Encoding

```
void saveHuff() {
        try {
            DataOutputStream outFile = new DataOutputStream(new
FileOutputStream(Path + "/" + namaFile + ".huf"));
            outFile.writeBytes("HUF");
            outFile.writeBytes(extensi);
            outFile.write(numberChar);
            outFile.write(table.Offset);

            outFile.write(MyData, 0, MyData.length);
            outFile.close();


        } catch (Exception e) {
```

```
  Logger.getLogger(Kompresi.class.getName()).log(Level.SEVERE,
null, e);
        }
    }
```

Kode Program 4.10 Implementasi Proses penggabungan Header dan Hasil *Encoding*

Dari kode program 4.10 dapat dilihat bahwa *header file* yang berupa ekstensi HUF, ekstensi file asli, jumlah karakter penyusun file , dan jumlah tambahan 0 untuk konversi ASCII dituliskan oleh *stream* untuk dijadikan file baru. MyData yang berisi karakter penyusun *file* asli, kode huffman,panjang kode huffman dan data *encoding* juga dimasukkan dalam *stream.*

4.1.2 Implementasi Proses Dekompresi

Proses dekompresi bertujuan untuk mengembalikan sebuah file terkompresi menjadi seperti *file* aslinya, agar dapat digunakan sesuai kebutuhan. Implementasi dari proses dekompresi pada gambar 3.11, adalah sebagai berikut:

1. Pembacaan Hader File

```
byte[] readHuff() {

        byte[] jumlahKarakter = new byte[2];

        try {
            DataInputStream inFile = new DataInputStream(new
FileInputStream(fileName));

            System.out.println("Reading huff file...\n");

            huffHeader = "" + (char) inFile.readByte()
                    + (char) inFile.readByte()
                    + (char) inFile.readByte();

           System.out.println("Reading extensi...\n");
           extensi = "" + (char) inFile.readByte()
                    + (char) inFile.readByte()
                    + (char) inFile.readByte();

            numberChar = (char)inFile.readByte();
            offset = (char)inFile.readByte();

            jumlahKarakterTersedia = inFile.available();
            MyData = new byte[(int) jumlahKarakterTersedia];
            inFile.read(MyData);

            System.out.println("Tabel Huffman");
            inFile.close();
```

```
        } catch (Exception e) {
Logger.getLogger(Dekompresi.class.getName()).log(Level.SEVERE,
null, e);
        }
        return MyData;
    }
```

Kode Program 4.11 Implementasi Proses Pembacaan Header

Dari kode program 4.11 dapat dilihat bahwa proses pembacaan header *file* terkompresi dilakukan dengan cara

- Mengambil 3 byte pertama sebagai ekstensi dari file terkompresi yakni *.huf*
- Mengambil 3 *byte* berikutnya sebagai ekstensi dari file asli yang dikompresi.
- Mengambil 1 *byte* berikutnya sebagai informasi jumlah tambahan "0" hasil encoding.
- Mengambil 1 *byte* berikutnya sebagai informasi jumlah karakter penyusun file asli.
- Sisa *byte* dari file terkompresi tersebut akan dijadikan sebagai data untuk proses dekripsi dan *decoding.*
-

2. Dekripsi Header

Proses dekripsi Header dilakukan untuk mendapatkan komposisi karakteristik file yang benar. Proses ini mengambil inputan berupa array yang panjangnya kelipatan 16.

```
if (kondisiKripto == true) {
        int temp = 0;
        temp = 16 - ((4 * numberChar) % 16);
        if (temp == 16) {
            temp = 0;
        }
        length = 4 * numberChar + temp;

        byte[] Decryp = new byte[length];
        System.arraycopy(MyData, 0, Decryp, 0, length);

        AES aes = new AES();
        AesField = new byte[4 * numberChar];
        AesField = aes.decrypt(Decryp,
```

```
          key.getBytes(), 4 * numberChar);
                    }
```

Kode Program 4.12 Implementasi Proses Dekripsi Header

Pada implementasi dekripsi Header (kode program 4.12), yang menjadi inputan untuk proses Dekripsi adalah *array* yang bernama Decryp dan kunci yang akan digunakan untuk proses dekripsi. Hasil dari proses dekripsi kemudian disimpan dalam *array* AesField[].

2.1 Dekripsi Blok

```
public byte[] decryptBloc(byte[] in) {
        byte[] tmp = new byte[in.length];
        byte[][] state = new byte[4][Nb];

        for (int i = 0; i < in.length; i++) {
            state[i / 4][i % 4] = in[i % 4 * 4 + i / 4];
        }
        state = AddRoundKey(state, w, Nr);
        for (int round = Nr - 1; round >= 1; round--) {
            state = InvSubBytes(state);
            state = InvShiftRows(state);
            state = AddRoundKey(state, w, round);
            state = InvMixColumns(state);
        }
        state = InvSubBytes(state);
        state = InvShiftRows(state);
        state = AddRoundKey(state, w, 0);

        for (int i = 0; i < tmp.length; i++) {
            tmp[i % 4 * 4 + i / 4] = state[i / 4][i % 4];
        }
        return tmp;
    }
```

Kode Program 4.13 Implementasi Dekripsi Blok

Dekripsi blok (kode program4.13) adalah proses untuk menjalankan **InvSubBytes, InvShiftRows, AddRoundKey dan InvMixColumns**.

3. Pembentukan Tabel Dekompresi

```
void doThat() {
while (id < MyData.length) {
       if (countMyData < 4 * numberChar) {
                if (kondisiKripto == true) {
        karakter[counterKodeHufman] = "" + (char) AesField[id];
                } else {
        karakter[counterKodeHufman] = "" + (char) MyData[id];
                }
                System.out.print(karakter[counterKodeHufman]);
                countMyData += 1;
                id += 1;
```

```
                if (kondisiKripto == true) {
                    kodeHufman[0] = AesField[id];
                } else {
                    kodeHufman[0] = MyData[id];
                }
              countMyData += 1;
                id += 1;
               if (kondisiKripto == true) {
                    kodeHufman[1] = AesField[id];
                } else {
                    kodeHufman[1] = MyData[id];
                }
                countMyData += 1;
                id += 1;
                   kodeHufmanString[counterKodeHufman]=
Integer.toBinaryString(byteArrayToInt(kodeHufman));

        if (kondisiKripto == true) {
                    panjangKodeHufman[counterKodeHufman] = (int)
AesField[id];
                } else {
                    panjangKodeHufman[counterKodeHufman] = (int)
MyData[id];
                }
             Int selisihPanjangKode =
panjangKodeHufman[counterKodeHufman] -
kodeHufmanString[counterKodeHufman].length();

                for (int i = 0; i < selisihPanjangKode; i++) {
                    kodeHufmanString[counterKodeHufman] = "0" +
kodeHufmanString[counterKodeHufman];
                }
                System.out.println(" \t" + kodeHufmanString
[counterKodeHufman]);
                countMyData += 1;
                id += 1;
                counterKodeHufman += 1;

            } else if((id >= length)&&(id<MyData.length)){
                byte[] tempArray = new byte[2];
                tempArray[0] = MyData[id];
                String string =
Integer.toBinaryString(byteArrayToInt(tempArray));

            else{id += 1;}

        }
```

Kode Program 4.14 Implementasi Proses Pembentukan Tabel Dekompresi

Dalam proses pembentukan tabel dekompresi (kode program 4.14), *array* **karakter []** digunakan untuk menampung karakter penyusun *file* asli. Kemudian array **kodeHufman[]** digunakan untuk menampung data kode huffman tiap-tiap karakter, dan *array* **panjangKodeHufman[]** digunakan untuk menampung panjang kode Huffman tiap-tiap karakter.

4. Decoding

```
    int tempLength = string.length() % 8;
                int i = 0;
                if (tempLength != 0) {
                    while (i < 8 - tempLength) {
                        string = "0" + string;
                        i += 1;
                    }
                }
                dataEncoding += string;
                id += 1;
            }
dataEncoding = dataEncoding.substring(0, dataEncoding.length() -
offset);

        String kandidat = "";
        for (int i = 0; i < dataEncoding.length(); i++) {
        kandidat = kandidat + dataEncoding.substring(i, i + 1);

     for (int j = 0; j < kodeHufmanString.length; j++) {
              if (kandidat.equals(kodeHufmanString[j])) {
                  hasilDekompresi += karakter[j];
                  kandidat = "";
              }
          }
        }
```

Kode Program 4.15 Implementasi proses *decoding*

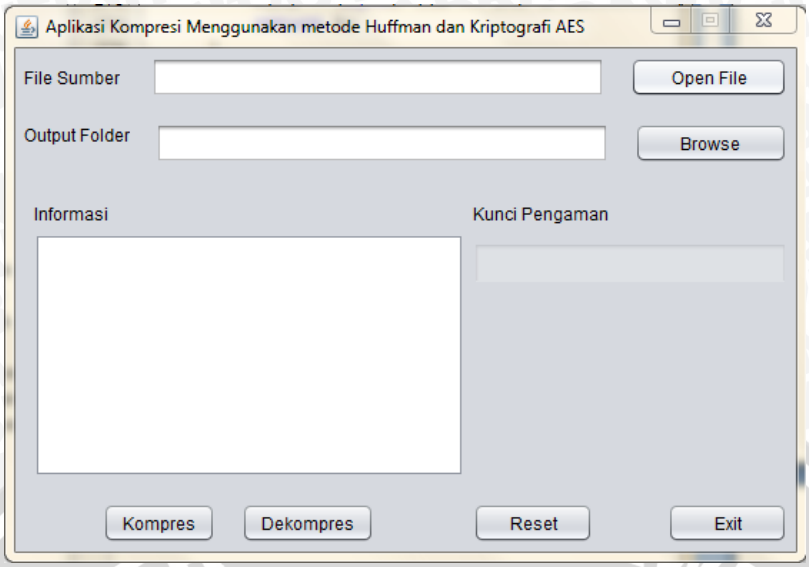
Decoding dilakukan untuk mengembalikan sebuah file terkompresi menjadi seperti file aslinya. Implementasi proses decoding ditunjukkan kode program 4.15.

Langkah pertama yang dilakukan dalam proses *decoding* adalah mengkonversi data *encoding* menjadi string biner. Langkah berikutnya adalah memotong string biner tersebut sebanyak jumlah *offset* atau jumlah tambahan bilangan "0"..

Setelah proses pemotongan, langkah berikutnya adalah membandingkan string-string biner dengan karakter yang ada pada tabel dekompresi. Array **karakter []** merupakan *array* yang menyimpan karakter-karakter penyusun *file* asli. Hasil proses dekompresi diberi nama **hasilDekompresi.**

4.2 Implementasi Antarmuka

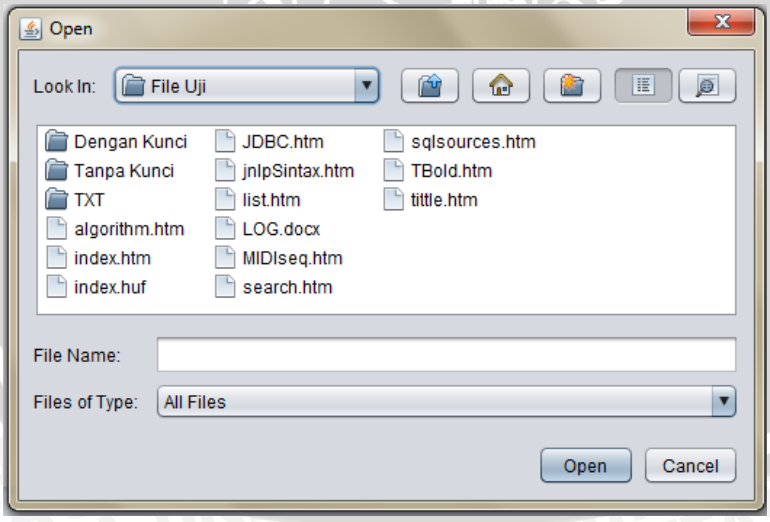
Berdasarkan pada rancangan antarmuka maka dihasilkan tampilan awal dari sistem yang ditunjukkan pada gambar 4.1.



Gambar 4.1 Tampilan Awal Antarmuka Aplikasi

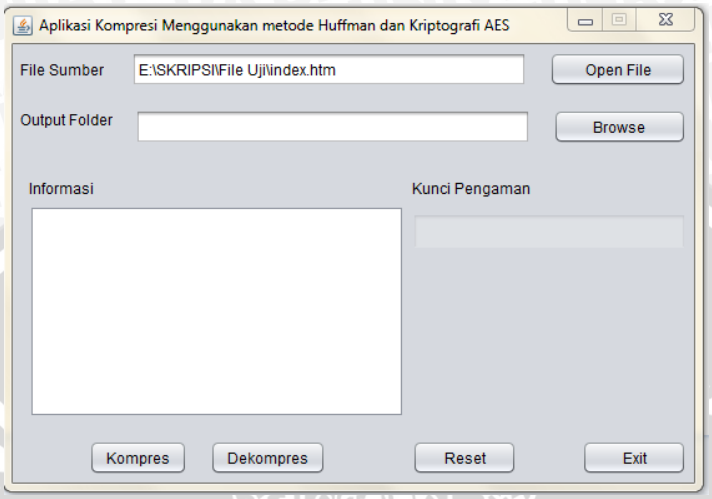
Dari tampilan pada gambar 4.1 Dapat dilihat bahwa terdapat beberapa tombol yang dapat digunakan untuk melakukan kompresi suatu *file.*

Untuk dapat memilih sebuah *file* yang akan dikompresi, pengguna sistem menekan tombol '***Open File***". Setelah tombol tersebut dipilih, kemudian akan muncul tampilan seperti pada gambar 4.2



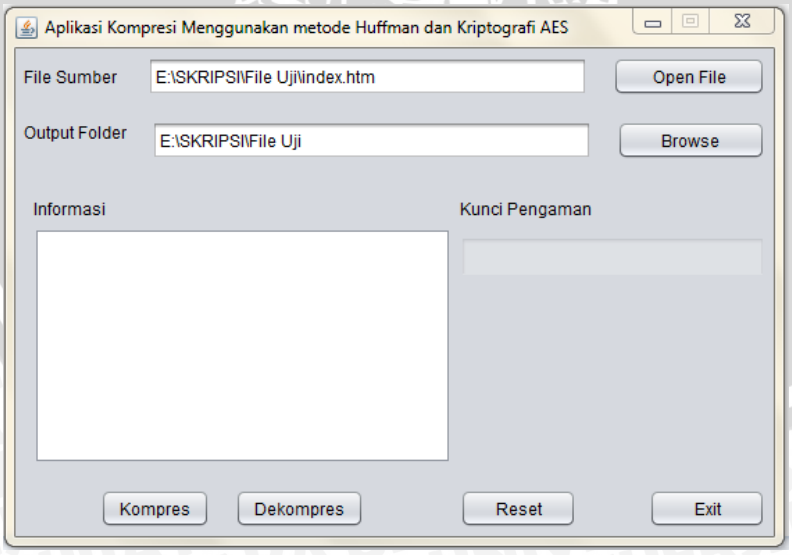
Gambar 4.2 Tampilan Antarmuka Menekan Tombol "Open File"

Disini pengguna dapat memilih sebuah *file* dengan menentukan lokasi dimana file tersebut berada. Misalnya telah dipilih file bernama '**index.htm**' yang akan dikompresi, maka tampilan selanjutnya adalah seperti gambar 4.3.



Gambar 4.3 Tampilan Antarmuka Memilih File

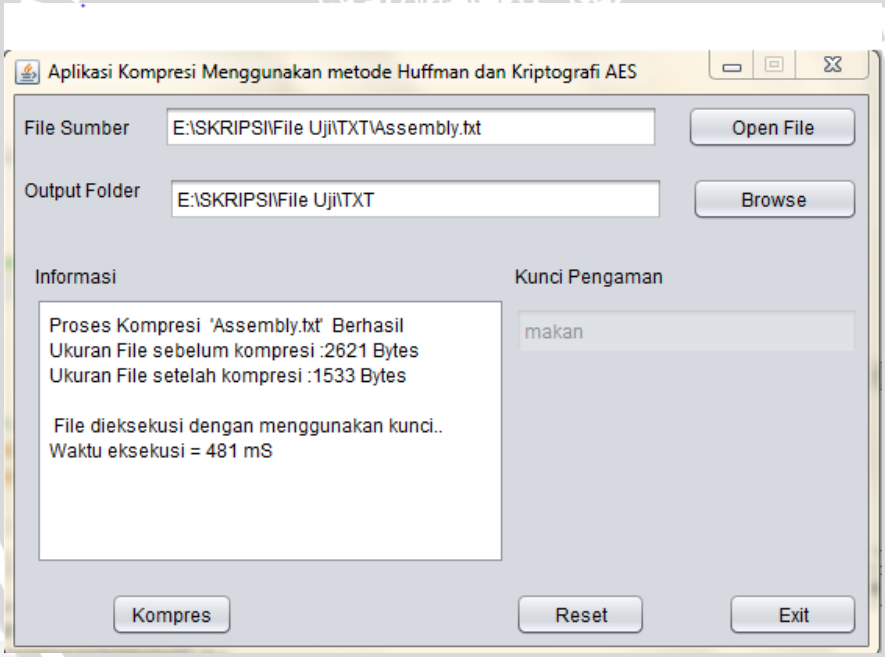
Hasil dari proses kompresi dapat ditentukan dimana akan diletakkan dalam penyimpanan dengan menekan tombol "**Browse".** Misalnya hasil proses kompresi diletekkan di *folder* yang bernama File Uji, maka sistem akan memberikan tampilan seperti pada gambar 4.4



Gambar 4.4 Tampilan Antarmuka Lokasi Penyimpanan File Terkompresi

Setelah lokasi penyimpan *file* hasil kompresi ditentukan, pengguna dapat langsung menekan tombol "**Kompres",** atau terlebih dahulu memasukkan kunci sebagai pengamanan**.** Jika field "**Kunci Pengaman**" diisi, ini menandakan bahwa proses kompresi akan diamankan dengan metode kriptografi AES. Dengan menekan tombol "**Kompres"**, maka proses untuk mengkompresi sebuah file akan dijalankan. Waktu proses kompresi hingga selesai akan bergantung pada ukuran dari *file* yang dikompresi. Semakin besar *file* yang dikompresi, waktu proses kompresi akan semakin bertambah.

Gambar 4.5 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.



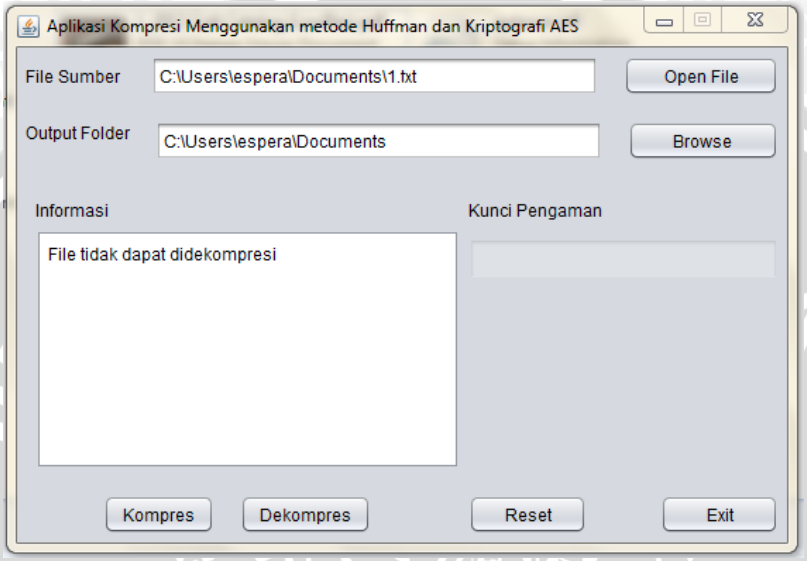
Gambar 4.5 Tampilan Antarmuka Proses Kompresi Selesai Dengan Kunci "ilmu komputer"

Gambar 4.5 menunjukkan proses kompresi telah selesai dilakukan dengan menggunakan kunci pengamanan, yang berarti bahwa proses kompresi telah diamankan dengan Kriptografi AES.

Ketika proses kompresi selesai, maka dalam *folder* File Uji akan terbentuk sebuah file baru dengan nama "index.huf".

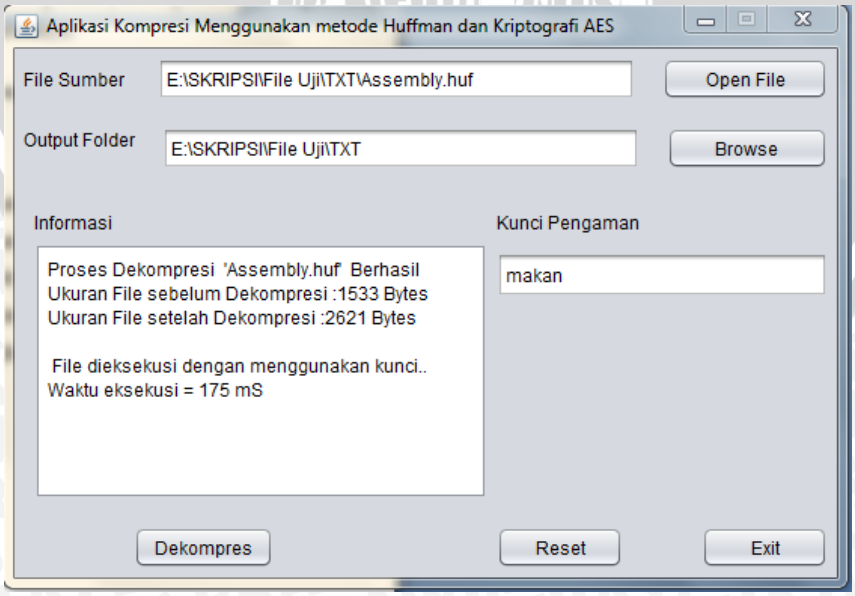
Untuk mengembalikan *file* hasil kompresi ke bentuk aslinya, akan membutuhkan proses dekompresi.

Jika yang menjadi *file* sumber adalah file dengan ekstensi selain **.huf*, maka proses dekompresi akan gagal dilakukan. Hal ini ditunjukkan pada gambar 4.6. Sistem hanya akan menjalankan proses dekompresi pada *file-file* yang berekstensi **.huf*



Gambar 4.6 Tampilan Antarmuka Proses Dekompresi Tidak Berjalan

Gambar 4.7 menunjukkan proses dekompresi dilakukan dengan menggunakan kunci "**ilmu komputer**". Ini berarti bahwa proses dekompresi akan menjalankan fungsi Kriptografi sebagai salah satu subprosenya.



Gambar 4.7 Tampilan Antarmuka Proses Dekompresi Selesai Dengan Kunci

4.3 Implementasi Uji Coba dan Evaluasi Hasil

4.3.1 Pengujian Algoritma Enkripsi AES

Pada bagian ini, akan dibahas mengenai kekuatan keamanan metode AES dengan metode serangan yang umum digunakan. Dikarenakan tidak adanya suatu teknik yang digunakan untuk menguji metode enkripsi yang bisa dianggap aman secara mutlak, maka akan dilakukan pendekatan untuk melihat apakah orang lain dapat menemukan suatu jalan untuk menghancurkan metode enkripsi ini.

Dalam sub bab ini, akan diujikan penyerangan terhadap metode AES dengan mencoba-coba setiap peluang kunci yang ada pada saat dekompresi, sampai diketemukan padanan kata yang sesuai dengan *chipertext*.

Tabel 4.1 Hasil Uji coba percobaan Dekompresi pada file Terenkripsi

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]
	9876543210	22.405	Kertas	91.842	Thinking Java(0 *byte*).txt [File tidak dapat dibaca]

	654321		9876543210 654321	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]

Dari hasil pada table 4.1 dapat dilihat bahwa ketika kunci yang digunakan pada saat dekompresi tidak sama dengan kunci yang digunakan pada saat kompresi, maka tidak akan didapatkan hasil yang sama dengan file aslinya. Namun jika kunci yang digunakan pada saat dekompresi sama dengan kunci pada saat kompresi, maka akan dihasilkan file asli yang sebenarnya.

4.3.2 Pengujian Kompresi Huffman

Pengujian dilakukan terhadap 10 *file* dengan ekstensi *.txt, dan 10 file dengan ekstensi *.htm. Uji coba dilakukan terhadap berbagai ukuran file. Data input yang digunakan dalam uji coba ditunjukkan pada tabel 4.2.

Tabel 4.2 Tabel data *input* untuk uji coba

No.	Nama *file*	Ekstensi	Ukuran file (byte)
1	Info	*.*txt*	247
2	James Gosling	*.*txt*	734
3	Java	*.*txt*	1.126
4	Kompilator	*.*txt*	1.857
5	Assembly	*.*txt*	2.621
6	Extended ASCII	*.*txt*	5.218
7	Huffman	*.*txt*	12.236
8	Data compression	*.*txt*	22.064
9	Tutorial	*.*txt*	36.557
10	Thinking Java	*.*txt*	50.541
11	Tittle	*.*htm*	293
12	Index	*.*htm*	806
13	Tbold	*.*htm*	1.198
14	Search	*.*htm*	2.551
15	JDBC	*.*htm*	11.835
16	Algorithm	*.*htm*	22.877
17	MIDIseq	*.*htm*	28.493

18	List	**.htm*	38.790
19	sqlsources	**.htm*	50.930
20	jnlpSintax	**.htm*	65.552

4.3.3 Analisa dan Evaluasi Hasil

4.3.3.1 Hasil Percobaan Dekompresi Pada File Terenkripsi

Dari tabel 4.1 dapat dilihat 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.

4.3.3.2 Hasil Pengujian Kompresi

Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi *.txt dengan menggunakan kunci “ilmu komputer” ditunjukkan pada tabel 4.3. Uji coba proses kompresi untuk sebuah file telah menghasilkan sebuah *file* terkompresi dengan ekstensi *.huf.

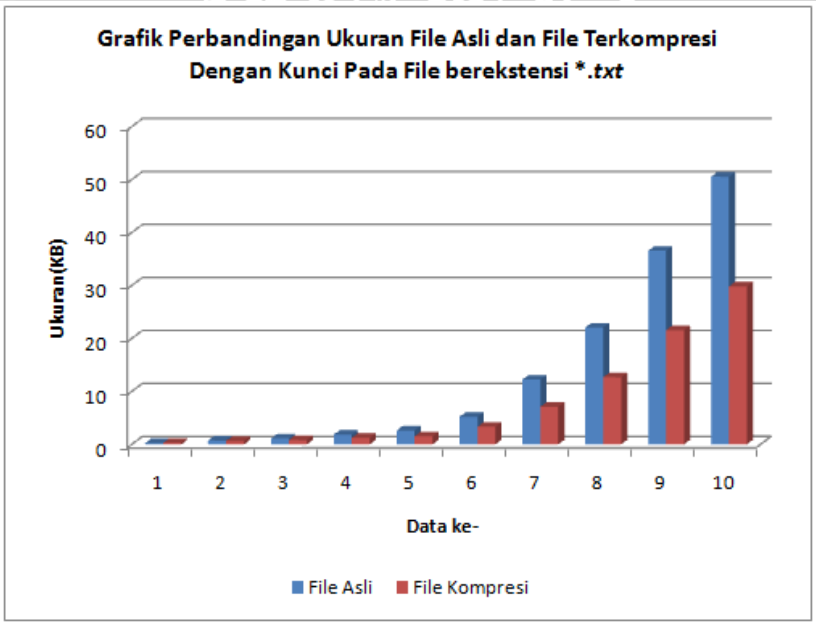
Tabel.4.3 Tabel Hasil Uji coba Kompresi pada file **.txt*

No.	Nama *file*	Ukuran file (byte)		Rasio Kompresi (%)
		Asli	Setelah kompresi	
1	Info.*txt*	247	285	-15.38
2	James Gosling.*txt*	734	653	11.04
3	Java.*txt*	1.126	811	27.98
4	Kompilator.*txt*	1.857	1.238	33.33
5	Assembly.*txt*	2.621	1.533	41.51
6	Extended	5.218	3.339	36.01

	ASCII.*txt*			
7	Huffman.*txt*	12.236	7.103	41.95
8	Data compression.*txt*	22.064	12.674	42.56
9	Tutorial.*txt*	36.557	21.556	41.03
10	Thinking Java.*txt*	50.541	29.830	94.10
Rata-rata Rasio Kompresi				**41,80**

Dari tabel 4.3 dapat dilihat bahwa rasio kompresi terbaik dihasilkan oleh file "Thinkung Java.txt" yang sebesar 94,10 %. Sedangkan rasio kompresi terburuk dihasilkan oleh file "Info.txt" yang sebesar -15, 38 %.

Dari Tabel 4.3 juga dapat disusun sebuah grafik yang menggambarkan perbandingan ukuran *file* awal dan ukuran *file* kompresi untuk setiap *file* yang diujikan. Grafik perbandingan ukuran *file* dapat dilihat pada Gambar 4.8.



Gambar 4.8 Grafik Perbandingan Ukuran File Asli dan File Terkompresi Dengan Kunci Pada File berekstensi *.*txt*

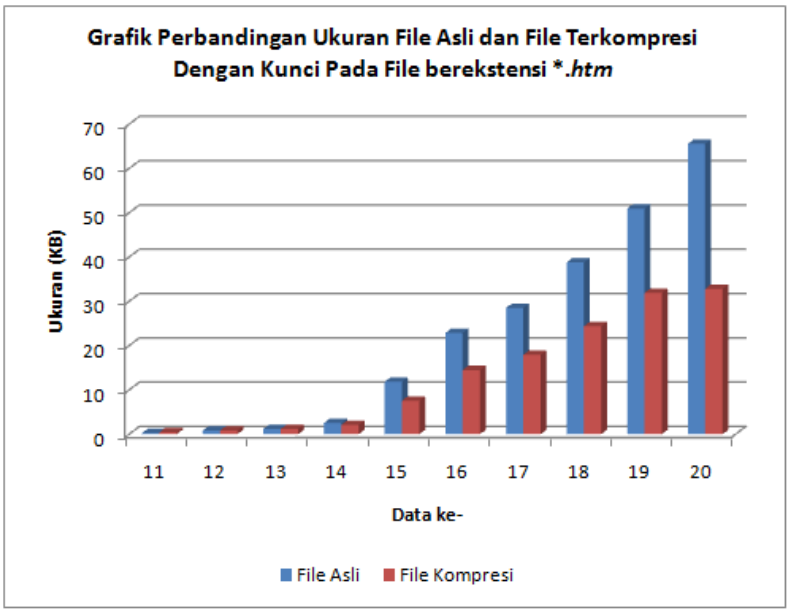
Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi *.*htm* dengan menggunakan kunci "ilmu komputer" ditunjukkan pada tabel 4.4. Uji coba proses kompresi untuk sebuah file telah menghasilkan sebuah *file* terkompresi dengan ekstensi *.huf.

Tabel.4.4 Tabel Hasil Uji coba Kompresi pada file *.*htm*

No.	Nama *file*	Ukuran file (byte)		Rasio Kompresi (%)
		Asli	Setelah kompresi	
1	Tittle.*htm*	293	436	-48.81
2	Index.*htm*	806	814	-0.99
3	Tbold.*htm*	1.198	1.162	3.01
4	Search.*htm*	2.551	2.088	18.15
5	JDBC.*htm*	11.835	7.549	36.21
6	Algorithm.*htm*	22.877	14.478	36.71
7	MIDIseq.*htm*	28.493	17.943	37.03
8	List.*htm*	38.790	24.360	37.20
9	Sqlsources.*htm*	50.930	31.899	37.37
10	jnlpSintax.*htm*	65.552	32.820	94.99
Rata-rata Rasio Kompresi				**1,38**

Dari tabel 4.4 dapat dilihat bahwa rasio kompresi terbaik dihasilkan oleh file "jnlpSintax.*htm*" yang sebesar 94,99 %. Sedangkan rasio kompresi terburuk dihasilkan oleh file "Tittle.*htm*" yang sebesar -48,81 %.

Dari Tabel 4.4 juga dapat disusun sebuah grafik yang menggambarkan perbandingan ukuran *file* awal dan ukuran *file* kompresi untuk setiap *file* yang diujikan. Grafik perbandingan ukuran *file* dapat dilihat pada Gambar 4.9.



Gambar 4.9 Grafik Perbandingan Ukuran File Asli dan File Terkompresi Dengan Kunci Pada File berekstensi *.*htm*

4.3.3.3 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.

4.3.3.4 Analisa Hasil Pengujian Dekompresi

Hasil uji coba untuk proses dekompresi dengan menggunakan kunci (tabel 4.5) 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. Hasil proses dekompresi dengan menggunakan kunci ditampilkan tabel 4.5

Tabel 4.5 Hasil uji coba proses dekompresi dengan menggunakan kunci "*ilmu komputer*"

Nama *file*	Ukuran file (byte)		Kesesuaian (%)
	Terkompresi	Setelah Dekompresi	
Info.*huf*	285	247	100
James Gosling.*huf*	653	734	100
Java.*huf*	811	1.126	100
Kompilator.*huf*	1.238	1.857	100
Assembly.*huf*	1.533	2.621	100
Extended ASCII.*huf*	3.339	5.218	100
Huffman.*huf*	7.103	12.236	100
Data compression.*huf*	12.674	22.064	100
Tutorial.*huf*	21.556	36.557	100
Thinking Java.*huf*	29.830	50.541	100
Tittle.*huf*	436	293	100
Index.*huf*	814	806	100
Tbold.*huf*	1.162	1.198	100
Search.*huf*	2.088	2.551	100
JDBC.*huf*	7.549	11.835	100
Algorithm.*huf*	14.478	22.877	100
MIDIseq.*huf*	17.943	28.493	100
List.*huf*	24.360	38.790	100
Sqlsources.*huf*	31.899	50.930	100
jnlpSintax.*huf*	32.820	65.552	100

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.

BAB V
KESIMPULAN DAN SARAN

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.

[PRA-08] Prayogo. 2008. Daftar ekstensi file.

http://prayogo.wordpress.com//daftar-ekstensi-file/
Diakses pada tanggal: 1 Mei 2012

[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. Fakultas Teknik Informatika Universitas Kristen Duta Wacana.

LAMPIRAN

Tabel Ascii

Decimal	Octal	Hex	Binary	Value	Remarks
000	000	000	00000000	**NUL**	Null character
001	001	001	00000001	**SOH**	Start of Header
002	002	002	00000010	**STX**	Start of Text
003	003	003	00000011	**ETX**	End of Text
004	004	004	00000100	**EOT**	End of Transmission
005	005	005	00000101	**ENQ**	Enquiry
006	006	006	00000110	**ACK**	Acknowledgment
007	007	007	00000111	**BEL**	Bell
008	010	008	00001000	**BS**	Backspace
009	011	009	00001001	**HT**	Horizontal Tab
010	012	00A	00001010	**LF**	Line Feed
011	013	00B	00001011	**VT**	Vertical Tab
012	014	00C	00001100	**FF**	Form Feed
013	015	00D	00001101	**CR**	Carriage Return
014	016	00E	00001110	**SO**	Shift Out
015	017	00F	00001111	**SI**	Shift In
016	020	010	00010000	**DLE**	Data Link Escape
017	021	011	00010001	**DC1**	XON Device Control 1
018	022	012	00010010	**DC2**	Device Control 2
019	023	013	00010011	**DC3**	XOFF Device Control 3
020	024	014	00010100	**DC4**	Device Control 4
021	025	015	00010101	**NAK**	Neg. Acknowledgment
022	026	016	00010110	**SYN**	Synchronous Idle
023	027	017	00010111	**ETB**	End of Transmission Block
024	030	018	00011000	**CAN**	Cancel
025	031	019	00011001	**EM**	End of Medium
026	032	01A	00011010	**SUB**	Substitute
027	033	01B	00011011	**ESC**	Escape
028	034	01C	00011100	**FS**	File Separator
029	035	01D	00011101	**GS**	Group Separator
030	036	01E	00011110	**RS**	Record Separator
031	037	01F	00011111	**US**	Unit Separator
032	040	020	00100000	**SP**	Space
033	041	021	00100001	**!**	Exclamation mark
034	042	022	00100010	**"**	Double quote
035	043	023	00100011	**#**	Number sign
036	044	024	00100100	**$**	Dollar sign
037	045	025	00100101	**%**	Percent
038	046	026	00100110	**&**	Ampersand
039	047	027	00100111	**'**	Single quote
040	050	028	00101000	**(**	Left/open parentheses
041	051	029	00101001	**)**	Right/close parentheses
042	052	02A	00101010	*****	Asterisk
043	053	02B	00101011	**+**	Plus
044	054	02C	00101100	**,**	Comma
045	055	02D	00101101	**-**	Minus or dash
046	056	02E	00101110	**.**	Dot
047	057	02F	00101111	**/**	Forward slash
048	060	030	00110000	**0**	
049	061	031	00110001	**1**	
050	062	032	00110010	**2**	
051	063	033	00110011	**3**	

Tabel Ascii (Lanjutan)

Decimal	Octal	Hex	Binary	Value	Remarks
052	064	034	00110100	4	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	Colon
059	073	03B	00111011	;	Semi-colon
060	074	03C	00111100	<	Less than
061	075	03D	00111101	=	Equal sign
062	076	03E	00111110	>	Greater than
063	077	03F	00111111	?	Question mark
064	100	040	01000000	@	AT symbol
065	101	041	01000001	A	
066	102	042	01000010	B	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	
074	112	04A	01001010	J	
075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	
086	126	056	01010110	V	
087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	Left / opening bracket
092	134	05C	01011100	\	Back slash
093	135	05D	01011101	]	Right / closing bracket
094	136	05E	01011110	^	Caret / circumflex
095	137	05F	01011111	_	Underscore
096	140	060	01100000	`	Back tick
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
102	146	066	01100110	f	
103	147	067	01100111	g	

Tabel Ascii (Lanjutan)

Decimal	Octal	Hex	Binary	Value	Remarks
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	06B	01101011	k	
108	154	06C	01101100	l	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	s	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	x	
121	171	079	01111001	Y	
122	172	07A	01111010	Z	
123	173	07B	01111011	{	Left / opening curly brace
124	174	07C	01111100	|	Vertical bar
125	175	07D	01111101	}	Right / closing curly brace
126	176	07E	01111110	~	Tilde
127	177	07F	01111111	DEL	Delete
128	200	080	10000000	Ç	Latin capital letter c with cedilla
129	201	081	10000001	ü	Latin small letter u with dieresis
130	202	082	10000010	é	Latin small letter e with acute
131	203	083	10000011	â	Latin small letter a with circumflex
132	204	084	10000100	ä	Latin small letter a with dieresis
133	205	085	10000101	à	Latin small letter a with grave
134	206	086	10000110	å	Latin small letter a with ring above
135	207	087	10000111	ç	Latin small letter c with cedilla
136	210	088	10001000	ê	Latin small letter e with circumflex
137	211	089	10001001	ë	Latin small letter e with dieresis
138	212	08A	10001010	è	Latin small letter e with grave
139	213	08B	10001011	ï	Latin small letter i with dieresis
140	214	08C	10001100	î	Latin small letter i with circumflex
141	215	08D	10001101	ì	Latin small letter i with grave
142	216	08E	10001110	Ä	Latin capital letter a with dieresis
143	217	08F	10001111	Å	Latin capital letter a with ring above
144	220	090	10010000	É	Latin capital letter e with acute
145	221	091	10010001	æ	Latin small ligature ae
146	222	092	10010010	Æ	Latin capital ligature ae
147	223	093	10010011	ô	Latin small letter o with circumflex
148	224	094	10010100	ö	Latin small letter o with dieresis
149	225	095	10010101	ò	Latin small letter o with grave
150	226	096	10010110	û	Latin small letter u with circumflex
151	227	097	10010111	ù	Latin small letter u with grave
152	230	098	10011000	ÿ	Latin small letter y with dieresis
153	231	099	10011001	Ö	Latin capital letter o with dieresis
154	232	09A	10011010	Ü	Latin capital letter u with dieresis
155	233	09B	10011011	¢	Cent sign

Tabel Ascii (Lanjutan)

Decimal	Octal	Hex	Binary	Value	Remarks
156	234	09C	10011100	£	Pound sign
157	235	09D	10011101	¥	Yen sign
158	236	09E	10011110	₧	Peseta sign
159	237	09F	10011111	ƒ	Latin small letter f with hook
160	240	0A0	10100000	á	Latin small letter a with acute
161	241	0A1	10100001	í	Latin small letter i with acute
162	242	0A2	10100010	ó	Latin small letter o with acute
163	243	0A3	10100011	ú	Latin small letter u with acute
164	244	0A4	10100100	ñ	Latin small letter n with tilde
165	245	0A5	10100101	Ñ	Latin capital letter n with tilde
166	246	0A6	10100110	ª	Feminine ordinal indicator
167	247	0A7	10100111	º	Masculine ordinal indicator
168	250	0A8	10101000	¿	Inverted question mark
169	251	0A9	10101001	⌐	Reversed not sign
170	252	0AA	10101010	¬	Not sign
171	253	0AB	10101011	½	Vulgar fraction one half
172	254	0AC	10101100	¼	Vulgar fraction one quarter
173	255	0AD	10101101	¡	Inverted exclamation mark
174	256	0AE	10101110	«	Left-pointing double angle quotation mark
175	257	0AF	10101111	»	Right-pointing double angle quotation mark
176	260	0B0	10110000	▒	Light shade
177	261	0B1	10110001	▒	Medium shade
178	262	0B2	10110010	▓	Dark shade
179	263	0B3	10110011	│	Box drawings light vertical
180	264	0B4	10110100	┤	Box drawings light vertical and left
181	265	0B5	10110101	╡	Box drawings vertical single and left double
182	266	0B6	10110110	╢	Box drawings vertical double and left single
183	267	0B7	10110111	╖	Box drawings down double and left single
184	270	0B8	10111000	╕	Box drawings down single and left double
185	271	0B9	10111001	╣	Box drawings double vertical and left
186	272	0BA	10111010	║	Box drawings double vertical
187	273	0BB	10111011	╗	Box drawings double down and left
188	274	0BC	10111100	╝	Box drawings double up and left
189	275	0BD	10111101	╜	Box drawings up double and left single
190	276	0BE	10111110	╛	Box drawings up single and left double
191	277	0BF	10111111	┐	Box drawings light down and left
192	300	0C0	11000000	└	Box drawings light up and right
193	301	0C1	11000001	┴	Box drawings light up and horizontal
194	302	0C2	11000010	┬	Box drawings light down and horizontal
195	303	0C3	11000011	├	Box drawings light vertical and right
196	304	0C4	11000100	─	Box drawings light horizontal
197	305	0C5	11000101	┼	Box drawings light vertical and horizontal
198	306	0C6	11000110	╞	Box drawings vertical single and right double
199	307	0C7	11000111	╟	Box drawings vertical double and right single
200	310	0C8	11001000	╚	Box drawings double up and right
201	311	0C9	11001001	╔	Box drawings double down and right
202	312	0CA	11001010	╩	Box drawings double up and horizontal
203	313	0CB	11001011	╦	Box drawings double down and horizontal
204	314	0CC	11001100	╠	Box drawings double vertical and right
205	315	0CD	11001101	═	Box drawings double horizontal
206	316	0CE	11001110	╬	Box drawings double vertical and horizontal
207	317	0CF	11001111	╧	Box drawings up single and horizontal double

Tabel Ascii (Lanjutan)

Decimal	Octal	Hex	Binary	Value	Remarks
208	320	0D0	11010000	╨	Box drawings up double and horizontal single
209	321	0D1	11010001	╤	Box drawings down single and horizontal double
210	322	0D2	11010010	╥	Box drawings down double and horizontal single
211	323	0D3	11010011	╙	Box drawings up double and right single
212	324	0D4	11010100	╘	Box drawings up single and right double
213	325	0D5	11010101	╒	Box drawings down single and right double
214	326	0D6	11010110	╓	Box drawings down double and right single 2
215	327	0D7	11010111	╫	Box drawings vertical double and horizontal single
216	330	0D8	11011000	╪	Box drawings vertical single and horizontal double
217	331	0D9	11011001	┘	Box drawings light up and left
218	332	0DA	11011010	┌	Box drawings light down and right
219	333	0DB	11011011	▉	Full block
220	334	0DC	11011100	▄	Lower half block
221	335	0DD	11011101	▌	Left half block
222	336	0DE	11011110	▐	Right half block
223	337	0DF	11011111	▀	Upper half block
224	340	0E0	11100000	α	Greek small letter alpha
225	341	0E1	11100001	ß	Latin small letter sharp s
226	342	0E2	11100010	Γ	Greek capital letter gamma
227	343	0E3	11100011	π	Greek small letter pi
228	344	0E4	11100100	Σ	Greek capital letter sigma
229	345	0E5	11100101	σ	Greek small letter sigma
230	346	0E6	11100110	µ	Micro sign
231	347	0E7	11100111	τ	Greek small letter tau
232	350	0E8	11101000	Φ	Greek capital letter phi
233	351	0E9	11101001	Θ	Greek capital letter theta
234	352	0EA	11101010	Ω	Greek capital letter omega
235	353	0EB	11101011	δ	Greek small letter delta
236	354	0EC	11101100	∞	Infinity
237	355	0ED	11101101	φ	Greek small letter phi
238	356	0EE	11101110	ε	Greek small letter epsilon
239	357	0EF	11101111	∩	Intersection
240	360	0F0	11110000	≡	Identical to
241	361	0F1	11110001	±	Plus-minus sign
242	362	0F2	11110010	≥	Greater-than or equal to
243	363	0F3	11110011	≤	Less-than or equal to
244	364	0F4	11110100	⌠	Top half integral
245	365	0F5	11110101	⌡	Bottom half integral
246	366	0F6	11110110	÷	Division sign
247	367	0F7	11110111	≈	Almost equal to
248	370	0F8	11111000	°	Degree sign
249	371	0F9	11111001	∙	Bullet operator
250	372	0FA	11111010	·	Middle dot
251	373	0FB	11111011	√	Square root
252	374	0FC	11111100	ⁿ	Superscript Latin small letter n
253	375	0FD	11111101	²	Superscript two
254	376	0FE	11111110	■	Black square
255	377	0FF	11111111		Non-breaking space (NBSP)