IMPLEMENTASI STEGANOGRAFI PADA FILE MP3 DENGAN METODE LEAST SIGNIFICANT BIT DAN ENKRIPSI RIJNDAEL

SKRIPSI

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun Oleh :

Arif Nur Yahya

NIM. 0810960034

KEMENTERIAN RISET TEKNOLOGI DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

MALANG

2015

LEMBAR PERSETUJUAN

IMPLEMENTASI STEGANOGRAFI PADA FILE MP3 DENGAN METODE LEAST SIGNIFICANT BIT DAN ENKRIPSI RIJNDAEL

SKRIPSI

Sebagai salah satu syarat untuk memperoleh
gelar sarjana dalam bidang komputer

Disusun oleh :

ARIF NUR YAHYA

NIM. 0810960034

Telah diperiksa dan disetujui oleh :

Pembimbing I,	Pembimbing II,
Ir. Sutrisno, MT.	**Imam Cholissodin, S.Si, M.Kom.**
NIP. 195703251987011001	**NIK. 85071916110422**

LEMBAR PENGESAHAN

IMPLEMENTASI STEGANOGRAFI PADA FILE MP3 DENGAN METODE LEAST SIGNIFICANT BIT DAN ENKRIPSI RIJNDAEL

SKRIPSI

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

Disusun Oleh:

ARIF NUR YAHYA

NIM. 0810960034

Penguji I

Budi Darma Setiawan, S.Kom, M.Cs.
NIP. 19841015 201404 1 002

Penguji II

Candra Dewi, S.Kom, M.Sc.
NIP. 19771114 200312 2 001

Penguji III

Randy Cahya W, S.ST.,M.kom.

NIK. 201405 880206 1 001

Mengetahui

Ketua Program Studi Informatika / Ilmu Komputer

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

PERNYATAAN ORISINALITAS

Saya yang bertanda tangan di bawah ini :

Nama : Arif Nur Yahya

NIM : 0810960034

Program Studi : Informatika / Ilmu Komputer

Penulis skripsi berjudul : Implementasi Steganografi Pada File Mp3 Dengan Metode *Least Significant Bit* Dan Enkripsi *Rijndael*

Dengan ini menyatakan bahwa :

1. Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.
2. Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 30 Juli 2015

Arif Nur Yahya

NIM. 0810960034

KATA PENGANTAR

Alhamdulillahi rabbil ‘alamin. Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat serta hidayah-Nya, sehingga penulis dapat menyelesaikan laporan tugas akhir dengan judul **“IMPLEMENTASI STEGANOGRAFI PADA FILE MP3 DENGAN METODE *LEAST SIGNIFICANT BIT* DAN ENKRIPSI *RIJNDAEL*”.**

Tugas akhir ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer di Program Studi Teknik Informatika / Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya Malang.

Dalam pelaksanaan dan penyusunan tugas akhir ini sudah cukup banyak bantuan yang diberikan berbagai pihak, baik berupa bimbingan dan saran hingga penulisan tugas akhir ini dapat terselaikan. Oleh karena itu, penulis dalam kesmpatan ini mengucapkan terima kasih kepada :

1. Bapak Ir. Sutrisno, MT. selaku Dosen Pembimbing I yang telah bersedia memberikan bimbingan, arahan, motivasi, serta meluangkan waktunya sehingga proposal skripsi ini dapat terselesaikan.
2. Bapak Imam Cholissodin, S.Si., M.Kom., selaku Dosen Pembimbing II yang telah memberikan pengetahuan, bimbingan, dan nasihat untuk kesempurnaan penulisan proposal skripsi ini.
3. Keluarga penulis, Bapak Drs. Solikan M.Pd., Ibu Suliyani dan adik-adik penulis yang tidak pernah berhenti memberikan dukungan dan doa kepada penulis.
4. Bapak Drs. Mardji, M.T., dan Bapak Issa Arwani, S.Kom, M.Sc selaku Ketua dan Sekretaris Prodi Teknik Informatika / Ilmu Komputer Universitas Brawijaya.
5. Ibu Dany Primanita Kartikasari, ST Selaku Dosen Penasehat Akademik yang telah memberikan banyak masukan dan nasehat dalam penyusunan tugas akhir ini.
6. Aditya Fredy Irawan, Andjar Marviano, Amirul Mukminin, Bayu Sutawijaya, Hernawan Adi Saputro, Fakhris Khusnu R M, Rochmad Hidayat, Dafid Eko

Firdaus, Dwi Prasetyo, Razaq Aulia Majid, Rizky Setyo Pambudi, Nur Hadi Sulaiman dan Tri Wahono Sadewo yang telah banyak memberikan bantuan dan berbagi pengalaman kepada penulis.

7. Dan semua pihak yang telah membantu terselesaikannya skripsi ini yang tidak dapat penulis sebutkan satu per satu.

Semoga skripsi ini bermanfaat bagi pembaca sekalian, Akhirnya, penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan dan mengandung banyak kekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran membangun dari pembaca.

Malang, Juli 2015

Penulis

IMPLEMENTASI STEGANOGRAFI PADA FILE MP3 DENGAN METODE LEAST SIGNIFICANT BIT DAN ENKRIPSI RIJNDAEL

ABSTRAK

Dalam kegiatan penyampaian atau pertukaran pesan melalui media yang umum digunakan seperti internet riskan dilakukan terutama apabila pesan yang di sampaikan atau ditukarkan mengandung informasi yang bersifat rahasia. Sehingga diperlukan suatu cara untuk mengamankan informasi dalam pesan tersebut. Solusi untuk menangani permasalahan tersebut adalah menggunakan kriptografi dan steganografi untuk menyembunyikan pesan tersebut. Penelitian ini menerapkan metode *Least Significant Bit* (LSB) untuk menyembunyikan pesan ke dalam berkas audio dengan format MP3. Pesan yang disembunyikan terlebih dahulu dienkripsi menggunakan algoritma *rijndael*. Selanjutnya dilakukan pengujian kualitas *stego* MP3 dengan menghitung *Mean Square Error* (MSE) dan *Peak Signal to Noise Ratio* (PSNR). Hasil pengujian kualitas s*tego* MP3 menunjukkan bahwa semakin besar ukuran pesan maka semakin tinggi nilai MSE dan semakin rendah PSNR-nya, sedangkan semakin besar ukuran berkas MP3 maka nilai MSE semakin rendah dan nilai PSNR semakin tinggi. Adapun nilai PSNR yang didapat dari hasil ujicoba yang dilakukan memiliki rentang antara 63.3 dB dan 85.9 dB

kata kunci : Steganografi *Least Significant Bit* (LSB), audio MP3, *Rijndael*, *Mean Square Error* (MSE), *Peak Signal to Noise Ratio* (PSNR).

STEGANOGRAPHY IMPLEMENTATION ON MP3 FIlE USING LEAST SIGNIFICANT BIT AND ENCRYPTION RIJNDAEL

ABSTRACT

In the activities of submission or exchange of messages through commonly used media such as the internet is risky to do especially if the message conveyed or exchanged for contain confidential information. So it requires a way of securing information in the message. The solution to address these problems is to use cryptography and steganography to hide the message. This study applies Least Significant Bit (LSB) methode to hide the message into audio files in MP3 format. Messages are first encrypted using Rijndael algorithm. The stego quality is measured by calculating the Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR). MP3 stego quality test results show that the greater the size of the message, the higher value of MSE and the lower its PSNR, while the larger the size of MP3 files, the lower of MSE value and the higher its PSNR. The PSNR values obtained from the results of tests carried out has a range between 63.3 dB and 85.9 dB.

keywords : *Least Significant Bit* Steganography (LSB), MP3 File, *Rijndael* Encryption, *Mean Square Error* (MSE), *Peak Signal to Noise Ratio* (PSNR).

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SOURCE CODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam kehidupan manusia sebagai makhluk sosial tidak pernah lepas dari komunikasi. Komunikasi adalah proses pengiriman pesan atau informasi antara dua orang atau lebih sehingga pesan yang dimaksud dapat dipahami oleh penerima pesan. Dengan semakin berkembangnya teknologi informasi, proses komunikasi dapat dilakukan dengan mudah dan sangat cepat. Tentunya, keamanan dan kerahasiaan suatu pesan menjadi bagian penting dari proses tersebut. Oleh karena itu, dibutuhkan suatu mekanisme pengamanan untuk menjaga kerahasiaan pesan.

Salah satu cara untuk mengamankan pesan dapat dilakukan dengan menggabungkan metode kriptografi dan steganografi. Kriptografi, yaitu ilmu yang mempelajari tentang metode pengiriman pesan dalam rahasia (*encipher*, menyamarkan pesan berupa kode) sehingga hanya penerima yang dituju yang dapat menguraikan dan membaca pesan tersebut (*decipher*, memecahkan kode) [MOL-07], sedangkan steganografi digunakan untuk menyembunyikan pesan rahasia ke dalam pesan lain sehingga kerahasiaannya tidak tampak oleh indera manusia [SCH-96].

Algoritma kriptografi *rijndael* termasuk dalam algoritma kriptografi yang sifatnya simetris dan *block cipher*. Panjang kunci dan ukuran blok algoritma *rijndael* dapat dipilih secara independen mulai dari 128 bit hingga 256 bit dengan kelipatan 32 bit. Untuk melakukan enkripsi atau dekripsi, algoritma *rijndael* menggunakan iterasi proses transformasi secara bertahap (*round function*). Algoritma *rijndael* juga mendefinisikan sebuah metode untuk menghasilkan serangkaian subkunci dari kunci asli. Subkunci yang dihasilkan digunakan dalam proses enkripsi dan dekripsi [DAE-02].

Steganografi merupakan metode untuk menyisipkan pesan pada media digital seperti citra, audio, atau video. Dengan teknik steganografi, pesan yang dipertukarkan tidak mencurigakan. Ada banyak cara yang dapat dilakukan untuk melakukan teknik steganografi atau menyembunyikan suatu pesan ke dalam data

lain. Salah satunya adalah dengan cara mengubah pesan ke bentuk biner dan membaginya dengan panjang satu bit, kemudian bit pesan tersebut digunakan untuk mengganti bit paling kanan (*least significant*) dari media penyisipan. Teknik steganografi ini disebut dengan metode *Least Significant Bit* (LSB). Kelebihan dari LSB ini adalah media yang disisipkan pesan sangat kecil untuk terdeteksi, namun memiliki kelemahan dalam kapasitas pesan yang tergantung pada ukuran media penyisipan [RED-10].

Pengamanan pesan menggunakan salah satu teknik di atas memungkinkan untuk menjaga keamanan suatu pesan. Kriptografi merupakan cara yang paling sering digunakan dalam pengamanan data. Walaupun cara ini dapat menyamarkan arti dari suatu pesan, tetapi cara ini tidak menyembunyikan adanya suatu pesan. Untuk itu pesan yang telah disamarkan (*chipertext*) akan lebih aman bila disembunyikan ke dalam suatu data lain melalui teknik steganografi.

Pada penelitian sebelumnya, pengamanan data yang dilakukan oleh algoritma kriptografi telah dilakukan oleh beberapa peneliti. Penelitian yang dilakukan oleh Febriansyah [FEB-12] yang berjudul "*Implementasi Least Significant Bit (Lsb) Insertion Untuk Menyembunyikan Ciphertext Blowfish Pada Citra Digital*" menggunakan metode Enkripsi Blowfish untuk merubah berkas *plaintext* menjadi *chipertext* untuk selanjutnya disisipkan kedalam citra digital menggunakan metode *Least Significant Bit*. Pada penelitian ini, peneliti menggunakan media penyisipan berupa citra digital. Hasil penelitian ini menunjukkan perubahan yang tidak signifikan terhadap media penyisipannya yakni citra digital yang ditunjukkan dengan nilai kualitas PSNR (*Peak Signal to Noise Ratio*) yang relatif sama dan nilai MSE (*Mean Square Error*) yang rendah. Hal ini menunjukkan bahwa citra digital yang telah disisipi pesan tidak akan mudah dibedakan dengan kasat mata. Penelitian lain yang dilakukan oleh Kumar [KUM12] yang berjudul "*Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms*" menyimpulkan bahwa secara umum performa *Rijndael* sebagai algoritma enkripsi dan dekripsi menunjukkan nilai diatas performa *blowfish* dalam lima kategori pengujian. Hal ini menunjukkan bahwa *Rijndael* memiliki tingkat performa lebih bagus daripada *Blowfish*, sehingga peneliti tersebut

merekomendasikan *Rijndael*. Sehingga dalam penelitian selanjutnya akan digunakan *Rijndael* sebagai algoritma kriptografi.

Pada penelitian ini akan digunakan berkas audio atau suara sebagai media penyisipan, hal ini dikarenakan media suara atau audio memiliki kapasitas yang lebih besar dibandingkan media gambar. Salah satu format audio yang sangat populer adalah MP3. MP3 merupakan salah satu format audio terkompresi yang memanfaatkan kelemahan pendengaran manusia. Walaupun ada banyak format audio terkompresi yang lebih unggul dari segi kapasitas dan kualitas suara dibanding MP3, format MP3 lebih banyak digunakan oleh sebagian besar orang. Hal ini menjadikan format audio MP3 sangat cocok untuk menjadi media penyisipan pesan karena tidak akan menimbulkan kecurigaan adanya pesan rahasia didalamnya.

Pada penelitian ini metode steganografi yang digunakan adalah *least significant bit* dan algoritma kriptografi rijndael untuk enkripsi pesan sebelum disisipkan pada berkas MP3. Diharapkan melalui penggunaan metode *least significant bit* dan enkripsi rijndael dapat meningkatkan keamanan informasi dalam suatu pesan. Berdasarkan uraian diatas maka judul skripsi ini adalah ”IMPLEMENTASI STEGANOGRAFI PADA FILE MP3 DENGAN METODE LEAST SIGNIFICANT BIT DAN ENKRIPSI RIJNDAEL”.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang diatas, dapat dirumuskan permasalahan yang akan diselesaikan yaitu:

1. Bagaimana menerapkan algoritma kriptografi Rijndael pada sebuah pesan (*plaintext*) dan metode steganografi LSB untuk menyisipkannya pada berkas MP3.
2. Bagaimana nilai evaluasi tingkat perubahan kualitas berkas audio MP3 yang diukur menggunakan *Mean Square Error* (MSE) dan *Peak Signal to Noise Ratio* (PSNR) setelah dilakukan proses penyembunyian pesan.

1.3 Batasan Masalah

Pada penelitian ini diperlukan batasan-batasan agar sesuai dengan apa yang sudah direncanakan sebelumnya sehingga tujuan penelitian dapat tercapai. Adapun batasan masalah skripsi ini adalah :

1. Pesan yang akan disembunyikan berformat teks (*.TXT*).
2. Sebelum disisipkan ke dalam berkas MP3, pesan dienkripsi terlebih dahulu.
3. Parameter yang akan diuji adalah perubahan kualitas berkas MP3 setelah disisipi pesan yang diukur berdasarkan *Mean Square Error* (MSE) dan *Peak Signal to Noise Ratio* (PSNR).

1.4 Tujuan Penelitian

Tujuan yang akan dicapai dalam pelaksanaan tugas akhir ini adalah sebagai berikut :

1. Mengimplementasikan pengamanan berkas dengan menggunakan algoritma kriptografi Rijndael dan metode steganografi *Least Significant Bit* (LSB) untuk menyisipkannya pada berkas audio MP3.
2. Mendapatkan nilai perubahan kualitas dengan menghitung nilai prosentase *Mean Square Error* (MSE) dan nilai *Peak Signal to Noise Ratio* (PSNR) MP3 setelah disisipi pesan dan sebelum disisipi pesan.

1.5 Manfaat Penelitian

Diharapkan penelitian ini dapat menjadi solusi dalam menjaga keamanan data dengan pengamanan data menggunakan algoritma kriptografi *Rijndael* dan disisipkan pada file berkas MP3 dengan mengimplementasikan metode *Least Significant Bit* (LSB).

1.6 Sistematika Penulisan

Penulisan tugas akhir ini terdiri dari beberapa bab, yang dijelaskan sebagai berikut :

BAB I PENDAHULUAN

Pada bab ini berisi tentang penjelasan latarbelakang, rumusan masalah, batasan masalah, sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Pada bab ini berisi landasan teori yang mendukung dalam penulisan skripsi.

BAB III METODOLOGI DAN PERANCANGAN

Pada bab ini membahas tentang teori-teori yang digunakan dalam perancangan dan mengimplementasikanya kedalam perangkat lunak.

BAB IV IMPLEMENTASI

Pada bab ini dibahas mengenai implementasi perangkat lunak yang dibuat, pengujiannya dan analisis sehingga dapat diketahui kekurangan dan kelebihan dari perangkat lunak yang dibuat.

BAB V UJI COBA DAN ANALISIS

Pada bab ini berisi tentang kesimpulan dari seluruh rangkaian penelitian yang dilakukan serta saran kemungkinan pengembangan.

BAB V KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dari seluruh rangkaian penelitian dan saran untuk pengembangan penelitian.

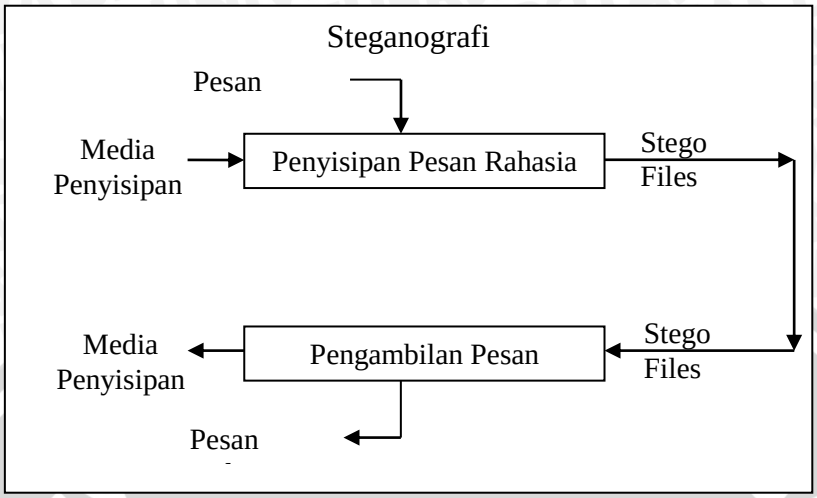
BAB II
TINJAUAN PUSTAKA

2.1 Steganografi

Istilah steganografi atau yang dalam dunia internasional dikenal dengan *steganography* sebenarnya berasal dari Bahasa Yunani, yaitu *steganos* dan *graphein*. *Steganos* berarti "penyamaran" atau "penyembunyian", sedangkan *graphein* memiliki arti "tulisan". Sehingga secara bahasa, steganografi dapat diartikan sebagai seni menyamarkan atau menyembunyikan pesan tertulis ke dalam pesan lainya [ARI-07].

Steganografi adalah sebuah komunikasi tersembunyi. Dalam hal ini hanya pengirim dan penerimalah yang mengetahui arti dan adanya komunikasi rahasia. Untuk melakukan komunikasi rahasia, pesan rahasia harus disembunyikan ke dalam sebuah media komunikasi yang kelihatan tidak berbahaya atau biasa disebut *cover* atau penutup. Untuk menggabungkan antara *cover* dengan pesan rahasia dibutuhkan sebuah metode steganografi. Syarat utama dalam steganografi adalah pesan rahasia memiliki kemampuan untuk menghindari deteksi. Hal ini berarti bahwa tidak ada algoritma yang dapat menentukan adanya pesan rahasia didalam suatu media [ARI-08].

Ada dua proses utama dalam steganografi yaitu penyisipan (*embedding*) dan penguraian (*extraction*) pesan atau informasi dalam media *cover*. *Embedding* merupakan proses menyisipkan pesan atau informasi ke dalam media *cover*, sedangkan *extraction* adalah proses menguraikan pesan yang tersembunyi dalam gambar *stego*. Pesan yang akan disembunyikan dalam sebuah gambar membutuhkan dua *file*. Pertama adalah gambar asli yang belum dimodifikasi yang akan menangani pesan tersembunyi, yang disebut gambar *cover* (*cover image*). File kedua adalah informasi pesan yang disembunyikan. Suatu pesan dapat berupa *plaintext*, *chipertext*, gambar lain, atau apapun yang dapat ditempelkan ke dalam *bit stream*. Ketika dikombinasikan, *cover image* dan pesan yang ditempelkan membuat gambar stego (*stego image*). Proses steganografi secara umum ditunjukkan oleh Gambar 2.1 [MOH-99].



Gambar 2.1 Proses Penyisipan dan Pengambilan Pesan Rahasia

2.1.1 Audio Steganografi

Audio steganografi adalah teknik penyisipan pesan rahasia dalam media suara (audio). Proses penyisipan pesan rahasia dalam steganografi pada dasarnya dilakukan dengan mengidentifikasi media audio pembawa pesan, yaitu *redundant bit* yang mana dapat dimodifikasi tanpa merusak integritas dari media audio itu sendiri. Dalam mengaplikasikan steganografi pada berkas audio dapat dilakukan dengan berbagai teknik. Berikut adalah beberapa metode yang dapat digunakan [KAL-10]:

a. Penggantian bit. Cara ini lazim digunakan dalam teknik digital steganografi yaitu mengganti bagian tertentu dari bit bit datanya dengan data rahasia yang disisipkan. Dengan metode ini keuntungan yang didapatkan adalah ukuran pesan yang disisipkan relatif besar, namun berdampak pada hasil audio yang berkualitas kurang dengan banyaknya derau.
b. Metode kedua yang digunakan adalah merekayasa fasa dari sinyal masukan (*phase encoding*). Teori yang digunakan adalah dengan mensubstitusi awal fasa dari tiap awal segmen dengan fasa yang telah dibuat dan merepresentasikan pesan yang disembunyikan. Fasa dari tiap awal segmen ini dibuat sedemikian rupa sehingga setiap segmen masih

memiliki hubungan yang berujung pada kualitas suara yang tetap terjaga. Teknik ini menghasilkan keluaran yang jauh lebih baik daripada metode pertama namun dikompensasikan dengan kerumitan dalam realisasinya.

c. Metode yang ketiga adalah penyebaran spektrum (*spread spectrum*). Dengan metode ini pesan dikodekan dan disebar ke setiap spektrum frekuensi yang memungkinkan. Maka dari itu akan sangat sulit bagi yang akan mencoba memecahkannya kecuali ia memiliki akses terhadap data tersebut atau dapat merekonstruksi sinyal acak yang digunakan untuk menyebarkan pesan pada range frekuensi.

d. Metode terakhir yang sering digunakan adalah menyembunyikan pesan melalui teknik *echo* (*echo hiding*). Teknik menyamarkan pesan ke dalam sinyal yang membentuk *echo*. Kemudian pesan disembunyikan dengan memvariasikan tiga parameter dalam *echo* yaitu besar amplitudo awal, tingkat penurunan atenuasi dan *offset*. Dengan adanya *offset* dari *echo* dan sinyal asli maka *echo* akan tercampur dengan sinyal aslinya, karena sistem pendengaran manusia yang tidak memisahkan antara *echo* dan sinyal asli.

2.1.2 Metode Least Significant Bit

Pada susunan bit di sebuah byte dimana 1 byte bernilai 8 bit, terdapat bit yang paling berarti (*Most Significant Bit* atau MSB) dan bit yang paling kurang berarti (*Least Significant Bit* atau LSB). Misalnya pada byte **1**1010010, bit 1 yang pertama (digarisbawahi) adalah MSB dan bit 0 yang terakhir (digarisbawahi) adalah LSB. LSB dikatakan paling kurang berarti karena apabila terjadi modifikasi pada bit LSB, maka hanya mengubah nilai byte tersebut satu lebih tinggi atau satu lebih rendah dari nilai sebelumnya [MUN-04]. Tabel 2.1 mengilustrasikan bagaimana pesan ‘OK’ dikodekan dalam media dengan panjang 8-bit dengan menggunakan metode LSB.

Tabel 2.1 Representasi OK Dalam Biner

Karakter	Nilai ASCII	
	Hexadecimal	Biner
O	4F	01001111
K	4B	01001011

Dan dibawah ini merupakan contoh ilustrasi bentuk biner dari media 8-bit.

10010101 00101010 10000000 01111111 00000010 01110101 01111001 10000101
11110101 01110011 10101011 01111101 01111010 01010000 00110000 11101110

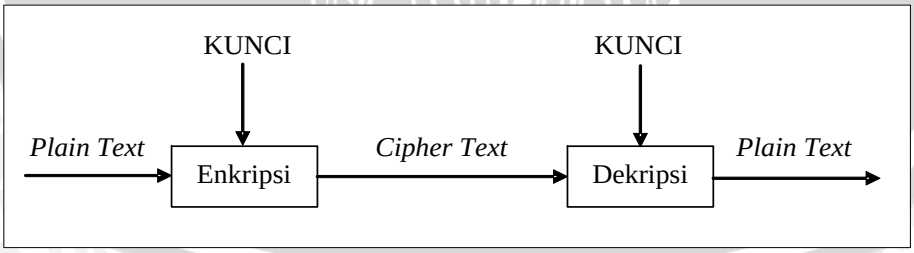
Berikut adalah hasil penyisipan pesan pada media 8-bit :

1001010**0** 0010101**1** 1000000**0** 0111111**0** 0000001**1** 0111010**1** 0111100**1** 1000010**1**
1111010**0** 0111001**1** 1010101**0** 0111110**0** 0111101**1** 0101000**0** 0011000**1** 1110111**1**

2.2 Kriptografi

Kata kriptografi berasal dari bahasa Yunani. Dalam bahasa Yunani kriptografi terdiri dari dua buah kata yaitu *cryptos* dan *graphia*. Kata *cryptos* berarti rahasia sedangkan *graphia* berarti tulisan. Berarti secara umum makna dari kata kriptografi adalah tulisan rahasia. Arti sebenarnya dari kriptografi adalah ilmu yang mempelajari tentang bagaimana menjaga kerahasiaan suatu pesan, agar isi pesan yang disampaikan aman sampai ke penerima pesan [ARI-08].

Secara umum, kriptografi merupakan teknik pengamanan informasi yang dilakukan dengan cara mengubah informasi awal (*plaintext*) dengan suatu kunci tertentu menggunakan suatu metode enkripsi tertentu sehingga menghasilkan suatu informasi baru (*ciphertext*) yang tidak dapat dibaca secara langsung. *Chipertext* tersebut dapat dikembalikan menjadi informasi awal melalui proses dekripsi. Gambaran umum kriptografi dapat dilihat pada Gambar 2.2.



Gambar 2.2 Gambaran Umum Proses Kriptografi

Menurut Wibowo [WIB-09], ada empat tujuan mendasar dari ilmu kriptografi ini yang juga merupakan aspek keamanan informasi, yaitu :

1. Kerahasiaan, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka atau mengupas informasi yang telah disandi.
2. Integritas data, adalah berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubstitusian data lain ke dalam data yang sebenarnya.
3. Autentikasi, adalah berhubungan dengan identifikasi atau pengenalan, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.
4. Non-repudiasi, adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman atau terciptanya suatu informasi oleh yang mengirimkan.

2.2.1 Komponen Kriptografi

Menurut Ariyus [ARI-07], komponen kriptografi terdiri dari beberapa komponen, seperti :

1. Algoritma, merupakan himpunan aturan matematis yang digunakan dalam enkripsi dan dekripsi.
2. Enkripsi, adalah transformasi data ke dalam bentuk yang tidak dapat terbaca tanpa sebuah kunci tertentu.
3. Dekripsi, merupakan kebalikan dari enkripsi, yaitu transformasi data terenkripsi ke bentuknya semula.
4. Kunci, digunakan pada saat melakukan enkripsi dan dekripsi. Pada kriptografi modern, keamanan enkripsi tergantung pada kunci, dan tidak tergantung kepada algoritmanya dilihat orang lain atau tidak.
5. Pesan asli (*plaintext*), merupakan teks asli yang akan diproses menggunakan algoritma kriptografi tertentu untuk menjadi *chipertext*.

6. *Ciphertext*, merupakan pesan yang telah melalui proses enkripsi yang merupakan himpunan karakter acak.
7. Kriptologi, merupakan studi tentang kriptografi dan kriptanalis.
8. Kriptanalis (*cryptanalis*), merupakan aksi memecahkan mekanisme kriptografi dengan cara menganalisanya untuk menemukan kelemahan dari suatu algoritma kriptografi sehingga akhirnya dapa ditemukan kunci atau teks asli.

2.2.2 Enkripsi dan Dekripsi

Enkripsi adalah sebuah proses dalam algoritma kriptografi. Untuk melakukan proses enkripsi dibutuhkan dua buah masukan berupa pesan asli atau *plaintext* dan kunci enkripsi. Pesan asli tersebut diubah menjadi *cipher* atau kode dari pesan tersebut. Proses mengubah *plaintext* tersebut menggunakan kunci enkripsi yang bersifat rahasia. Dekripsi merupakan kebalikan dari enkripsi. Pesan yang telah dienkripsi dikembalikan ke bentuk asalnya (pesan asli), disebut dengan dekripsi pesan.

Keamanan dari kriptografi modern didapat dengan merahasiakan kunci yang dimiliki dari orang lain, tanpa harus merahasiakan algoritma itu sendiri. Jika keseluruhan keamanan algoritma tergantung pada kunci yang dipakai, maka algoritma ini bisa dipublikasikan dan dianalisa orang lain.

2.3 Algoritma *Rijndael*

Rijndael merupakan algoritma enkripsi yang diciptakan untuk kompetisi keamanan yang diselenggarakan oleh *NIST (National Institute of Standard and Tecnology)* pada tahun 2001. Rijndael merupakan algoritma *blockcipher* simetris generasi baru yang mendukung beragam kunci dan blok data.

Input dan *output* dari algoritma rijndael terdiri dari urutan data sebesar 128 bit (16 byte). Urutan data yang sudah terbentuk dalam satu kelompok 128 bit tersebut disebut juga sebagai blok data. Ukuran besar blok data yang umum digunakan adalah 128 bit dan ukuran kunci terdiri dari 128 bit, 192 bit dan 256 bit [DAE-02].

Urutan bit data diberi nomor urut dari 0 sampai dengan n-1 dimana n adalah nomor urutan. Urutan data 8 bit secara berurutan disebut sebagai *byte* dimana *byte*

ini adalah unit dasar dari operasi yang akan dilakukan pada blok data. Dalam algoritma AES, blok data sepanjang 128 bit akan dibagi-bagi menjadi *array byte* dimana setiap *array byte* ini terdiri dari 8 bit data input yang saling berurutan. *Array byte* ini direpresentasikan dalam bentuk :

b0, b1, b2, b3, b4, b5, b6, b7 … b15

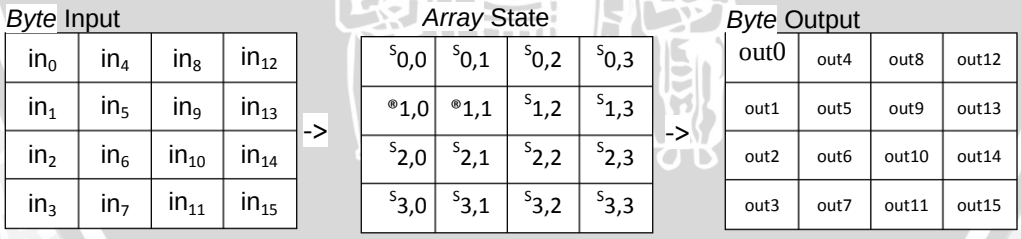
Operasi algoritma *Rijndael* dilakukan pada sebuah *State* dimana *State* sendiri adalah sebuah *array byte* dua dimensi. Setiap *State* pasti mempunyai jumlah baris yang tetap, yaitu 4 baris, sedangkan jumlah kolom tergantung dari besarnya blok data. Baris pada *State* mempunyai indeks nomor *row (r)* dimana $0 < r < 4$, sedangkan kolom mempunyai indeks nomor *column (c)* dimana $0 < c < Nb$. *Nb* sendiri adalah besarnya blok data dibagi 32. Pada saat permulaan, input bit pertama kali akan disusun menjadi suatu *array byte* dimana panjang dari *array byte* yang digunakan adalah sepanjang 8 bit data. *Array byte* inilah yang nantinya akan dimasukkan atau disalin ke dalam *State* dengan urutan :

$state[r,c] = in[r+4c]$ *untuk* $0 < r < 4$ *dan* $0 < c < Nb$

sedangkan dari *State* akan disalin ke output dengan urutan :

$out[r+4c] = s[r,c]$ *untuk* $0 < r < 4$ *dan* $0 < c < Nb$

Proses *Byte Input* yang diproses dengan *Array State* menghasilkan *Byte Output* ditunjukkan oleh Gambar 2.3



Gambar 2.3 Proses *Byte Input* yang Diproses dengan *Array State* [DAE-02].

2.3.1 Proses Enkripsi

Proses enkripsi pada algoritma rijndael terdiri dari 4 jenis transformasi byte, yaitu *SubBytes*, *ShiftRows*, *Mixcolumns*, dan *AddRoundKey*. Pada awal proses enkripsi, input yang telah disalin ke dalam akan mengalami transformasi byte *AddRoundKey*. Setelah itu, *State* akan mengalami transformasi *SubBytes*, *ShiftRows*, *MixColumns*, dan *AddRoundKey* secara berulang-ulang sebanyak *Nr*.

Proses ini dalam algoritma rijndael disebut sebagai *round function. Round* yang terakhir sedikit berbeda dengan *round-round* sebelumnya dimana pada *round* terakhir, *state* tidak mengalami transformasi *MixColumns* [DAE-02].

2.3.1.1 SubBytes

SubBytes merupakan transformasi byte dimana setiap elemen pada *State* dipetakan dengan menggunakan sebuah tabel substitusi *(S-Box).* Substitusi dilakukan dengan menggunakan 4 bit paling kiri sebagai baris dan 4 bit paling kanan sebagai kolom. Bentuk tabel S-Box dapat dilihat pada Gambar 2.4.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
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

Gambar 2.4 Tabel S-Box yang Digunakan dalam Proses Substitusi *byte.*

Hasil yang didapat dari pemetaan dengan menggunakan tabel *S-Box* ini sebenarnya adalah hasil dari dua proses transformasi byte, yaitu :

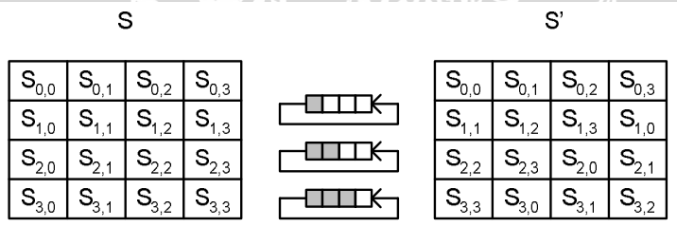
1. *Invers* perkalian dalam GF(2^8) adalah fungsi yang memetakan 8 bit ke 8 bit yang merupakan *invers* dari elemen *finite field* tersebut. Suatu byte a merupakan *invers* perkalian dari byte b bila a*b = 1, kecuali {00} dipetakan ke dirinya sendiri. Setiap elemen pada *State* akan dipetakan pada tabel *invers.* Sebagai contoh, elemen “01010011” atau {53} akan dipetakan ke {CA} atau “11001010”.
2. Transformasi *affine* pada *State* yang telah dipetakan. Transformasi *affine* ini apabila dipetakan dalam bentuk matriks adalah sebagai berikut :

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} x \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$b_7b_6b_5b_4b_3b_2b_1b_0$ adalah urutan bit dalam elemen State atau array byte dimana b_7 adalah *most significant bit* atau bit dengan posisi paling kiri [DAE-02].

2.3.1.2 Shiftrows

Transformasi *Shiftrows* pada dasarnya adalah proses pergeseran bit dimana bit paling kiri akan dipindahkan menjadi bit paling kanan (rotasi bit). Transformasi ini diterapkan pada baris 2, baris 3, dan baris 4. Baris 2 akan mengalami pergeseran bit sebanyak satu kali, sedangkan baris 3 dan baris 4 masing-masing mengalami pergeseran bit sebanyak dua kali dan tiga kali ditunjukkan pada Gambar 2.5



Gambar 2.5 Proses *Shiftrows* dengan Ilustrasi Bit yang Digeser [DAE-02].

2.3.1.3 MixColumns

Mixolumns mengoperasikan setiap elemen yang berada dalam satu kolom pada *State*. Elemen pada kolom dikalikan dengan suatu polinomial tetap

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

Secara lebih jelas, transformasi mixcolumns dapat dilihat pada perkalian matriks berikut ini :

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

Melakukan proses penjumlahan pada operasi ini berarti melakukan operasi *bitwise* XOR. Maka hasil dari perkalian matriks diatas dapat dianggap seperti perkalian yang ada di bawah ini :

$S'_{0.c} = (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$

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

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

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

2.3.1.4 AddRoundKey

Pada proses *AddRoundKey,* sebuah *round key* ditambahkan pada *State* dengan operasi bitwise XOR. Setiap *round key* terdiri dari *Nb word* dimana tiap *word* tersebut akan dijumlahkan dengan *word* atau kolom yang bersesuaian dari *State* sehingga :

$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [W_{round*Nb+c}]$ untuk $0 \leq c \leq Nb$

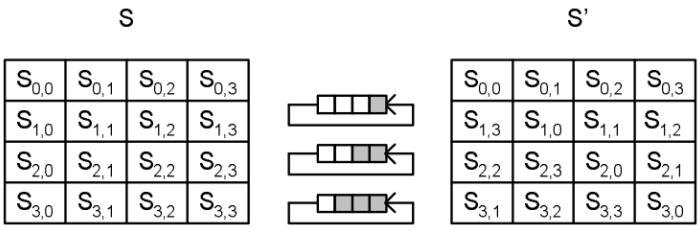
[*Wi*] adalah *word* dari key yang bersesuaian dimana *i = round*Nb+c.* Transformasi AddRoundKey diimplementasikan pertama kali pada *round* = 0, dimana *key* yang digunakan adalah *initial key* (*key* yang dimasukkan oleh kriptografer dan belum mengalami proses *key expansion*) [DAE-02].

2.3.2 Proses Dekripsi

Transformasi *cipher* dapat dibalikkan dan diimplementasikan dalam arah yang berlawanan untuk menghasilkan *inverse cipher* yang mudah dipahami untuk algoritma AES. Transformasi byte yang digunakan pada invers *cipher* adalah *InvShiftRows, InvSubBytes, InvMixColumns,* dan *AddRoundKey* [DAE-02].

2.3.2.1 InvShiftRows

InvShiftRows adalah transformasi *byte* yang berkebalikan dengan transformasi *ShiftRows*. Pada transformasi *InvShiftRows,* dilakukan pergeseran bit ke kanan sedangkan pada *ShiftRows* dilakukan pergeseran bit ke kiri. Pada baris kedua, pergeseran bit dilakukan sebanyak 3 kali, sedangkan pada baris ketiga dan baris keempat, dilakukan pergeseran bit sebanyak dua dan satu kali. Proses Tansformasi *InvShiftRows* dapat dilihat pada Gambar 2.6.



Gambar 2.6 Proses Tansformasi *InvShiftRows* [DAE-02].

2.3.2.2 InvSubBytes

InvSubBytes juga merupakan transformasi bytes yang berkebalikan dengan transformasi *SubBytes*. Pada *InvSubBytes*, tiap elemen pada *State* dipetakan dengan menggunakan tabel *inverse S-Box*. Tabel ini berbeda dengan tabel *S-Box*. Tabel yang digunakan untuk proses substitusi InvSubBytes dapat dilihat pada Gambar 2.7.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

Gambar 2.7 Tabel Inverse S-Box yang Digunakan dalam Proses Dekripsi.

2.3.2.3 InvMixColumns

Pada *InvMixColumns*, kolom-kolom pada tiap *State* (*word*) akan dipandang sebagai polinom atas GF(2^8) dan mengalikan modulo $x^4 + 1$ dengan polinom tetap $a^{-1}(x)$ yang diperoleh dari :

$$a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

Atau dalam matriks :

$$S'(x) = a(x) \otimes s(x)$$

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

Hasil perkalian diatas adalah :

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

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

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

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

2.3.2.4 Pembentukan Subkunci

Dalam proses pembentukan subkunci, algoritma rijndael menghasilkan subkunci sebanyak *Nb*(Nr+1)* word. Dengan masukan berupa kunci (K) sepanjang *Nb* word dan setiap *roundNr* membutuhkan data kunci sebanyak *Nb* word. Maka apabila kunci yang digunakan sepanjang 128 bit (4 word) maka panjang subkunci yang dihasilkan sebanyak 44 word. Subkunci yang dihasilkan berupa array 4 *byte* linear yang dinotasikan dengan (Wi) [DAE-02].

Operasi pembentukan subkunci dilakukan dengan aturan, jika nilai modulo i sama dengan 0, maka lakukan persamaan 2.1 dan jika nilai i lebih dari 4 maka dilakukan persamaan 2.2. Untuk nilai i kurang dari 4 (i_0, i_1, i_2, i_3) dilakukan proses salin nilai kunci ke-i seperti persamaan 2.3.

$$W[i] = W[i-4] \oplus SubWord(RotWord(W[i-1])) \oplus RCon[i] \quad (2.1)$$

$$W[i] = W[i-4] \oplus W[i-1] \quad (2.2)$$

$$W[i] = K[i] \quad (2.3)$$

SubWord adalah fungsi yang mengambil 4 byte *array input* dan mengaplikasikan S-Box ke setiap *data input* dan menghasilkan *output* dengan panjang yang sama. Fungsi *RotWord* mengambil *array* [a0, a1, a2, a3] sebagai input, melakukan permutasi siklik, dan mengembalikan *word* [a1, a2, a3, a0]. *Rcon*[i] terdiri dari nilai-nilai yang diberikan oleh [x^{i-1}, {00}, {00}, {00}], dengan x^{i-1} sebagai pangkat dari x (x dinotasikan sebagai {02} dalam *field GF*(2^8) [DAE-02].

2.4 Berkas Audio MP3

Asal mula MP3 dimulai dari penelitian IIS-FHG (*Institut Intergriette Schaltungen Fraunhofer Gessellschaft*), sebuah lembaga penelitian terapan di Munich, Jerman dalam penelitian terapan *coding audio perceptual*. Penelitian mengenai pemampatan berkas audio ini dipimpin oleh Karl Heinz Brandenburg, dan menghasilkan sebuah algoritma MPEG-1 Layer 3 yang kemudian dikenal sebagai MP3. Penelitian tersebut menghasilkan suatu algoritma yang menjadi standar sebaga ISO-MPEG Audio Layer-3 (MP3), yang merupakan berkas dengan teknik *lossy compression*.

Dalam upaya menghasilkan MP3, Brandenburg menganalisis bagaimana otak dan telinga manusia menangkap suara. Teknik yang digunakan berhasil memanipulasi telinga dengan membuang bagian yang kurang penting pada suatu berkas audio. Sebagai contoh, apabila terdapat dua nada yang mirip, atau apabila nada tinggi dan rendah muncul secara bersamaan, otak hanya akan memproses salah satunya. Berdasarkan keadaan tersebut, maka algoritma pada MP3 hanya akan memilih sinyal yang lebih penting dan membuang sisanya sehingga berkas audio MP3 memiliki ukuran berkas audio orisinal hingga 10 kali lebih kecil. Teknologi ini kemudian distandardisasi pada tahun 1991.

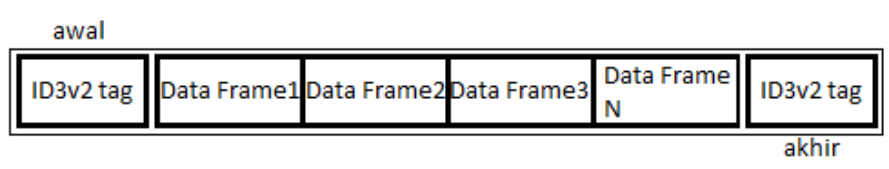
Teknik kompresi MP3 dapat menghasilkan suatu berkas audio terkompresi dengan kualitas suara yang tidak berbeda jauh dari aslinya karena memanfaatkan kelemahan atau keterbatasan dari sistem pendengaran manusia. Berikut adalah beberapa kelemahan dari sistem pendengaran yang digunakan dalam pemodelan kompresi MP3 [CHA-09] :

1. Terdapat beberapa suara yang tidak dapat didengar manusia, yaitu suara yang memiliki frekuensi di luar rentang 20 Hz sampai dengan 20000 Hz.
2. Terdapat suara yang dapat terdengar lebih baik bagi pendengaran manusia dibandingkan suara lainnya.
3. Bila terdapat dua suara yang dikeluarkan secara simultan, maka pendengaran manusia akan mendengar yang lebih keras, sedangkan yang lebih pelan akan tidak terdengar.

Kepopuleran dari MP3 yang sampai saat ini belum tersaingi disebabkan oleh beberapa hal. Salah satunya adalah MP3 dapat didistribusikan dengan mudah hampir tanpa biaya, walaupun sebenarnya hak paten dari MP3 telah dimiliki dan penyebaran MP3 seharusnya dikenai biaya. Walaupun begitu, pemilik hak paten dari MP3 telah memberikan pernyataan bahwa penggunaan MP3 untuk keperluan perorangan tidak dikenai biaya. Keuntungan lainnya adalah kemudahan akses MP3, dimana banyak perangkat lunak yang dapat menghasilkan berkas MP3 dari CD dan bersifat kosmopolit.

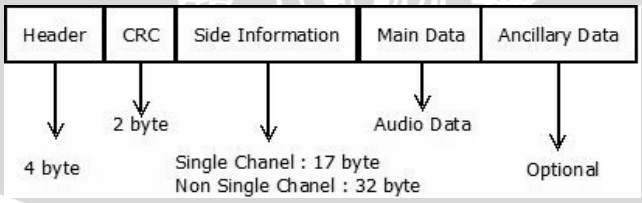
2.4.1 Struktur Berkas MP3

Menurut [RAS-02], berkas MP3 terdiri dari bagian-bagian kecil yang disebut *frame*. Setiap *frame* memiliki waktu yang konstan yaitu 0.026 detik (26 milidetik). Ukuran dari sebuah *frame* (dalam *byte*) memiliki nilai yang bervariasi tergantung dari bitrate. Gambar 2.8 menunjukkan struktur berkas MP3 yang terdiri dari beberapa *frame*.



Gambar 2.8 Struktur Berkas MP3

Sebuah *frame* data MP3 secara umum terdiri atas 5 bagian, yaitu *header*, CRC, *side information*, *main data*, dan *ancilarry data*. Gambar 2.9 menunjukkan struktur dari sebuah *frame* MP3.



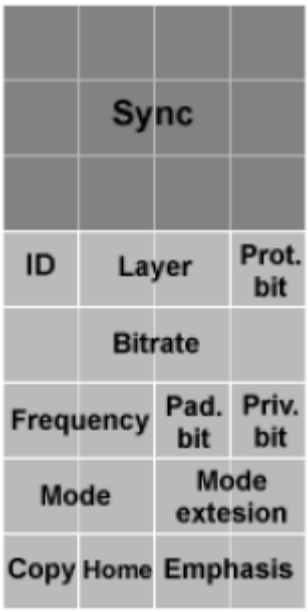
Gambar 2.9 Struktur Frame MP3 [RAS-02]

Menurut Baskara (2008), setiap *tag* MP3 atau awal berkas MP3 diawali dengan 3 byte bilangan heksadesimal 49, 44, dan 33. Untuk setiap *frame* pada MP3 selalu diawal *header* yang ditandai dengan bilangan heksadesimal FF dan FB.

Berikut ini merupakan bagian-bagian dari *frame* MP3 [RAS-02] :

1. *Header*

Header adalah bagian dari sebuah *frame* dengan panjang 4 byte (32 bit) yang berisi bit-bit sinkronisasi dan deskripsi tentang *frame* tersebut. Struktur dari *header* MP3 ditunjukkan pada Gambar 2.10 dibawah ini :



Gambar 2.10 Struktur *Header* MP3 [RAS-02]

Keterangan mengenai fungsi dan kebutuhan bit *header* ditunjukkan pada Tabel 2.3.

Tabel 2.3 Fungsi dan Kebutuhan Bit *Header*

Posisi	Keterangan Fungsi	Panjang (Bit)
Sync	*Frame* sinkronisasi	12
Id	Versi MPEG audio (MPEG-1, MPEG-2, dll)	1
Layer	MPEG *layer* (*Layer* I, II, III dll)	2
Prot. Bit	Bit proteksi, jika bit ini diset maka CRC akan digunakan	1
Bitrate	Indeks *bitrate*	4
Frequency	Frekuensi *sampling rate*	2
Pad. Bit	Bit *padding*	1
Priv. Bit	*Private* bit	1
Mode	Mode *channel* (*stereo*, *joint stereo*, dll)	2

Posisi	Keterangan Fungsi	Panjang (Bit)
Mode Extension	Mode *extension*	2
Copy	Hak cipta	1
Home	Orisinalitas	1
Emphasis	*Emphasis*	2

Beberapa keterangan *bit position* antara lain :

a. Bit sinkronisasi

Sinkronisasi terdiri dari 12 bit yang harus bernilai 1111 1111 1111.

b. Bit *id*

Menentukan versi dari MPEG. Ketika bit *id* bernilai 1, maka *frame* akan dikodekan menggunakan standar MPEG-1. *Frame* akan dikodekan menggunakan standar MPEG-2 ketika bit *id* bernilai 0. Terdapat beberapa standar lain yang hanya menggunakan 11 bit sinkronisasi sehingga sisa satu bit dapat digunakan untuk *id*. Tabel 2.4 menunjukkan penggunaan bit *id* yang terdiri dari 2 bit.

Tabel 2.4 Nilai Bit Pada Bagian *Header* Bit ID

Bit ID	Keterangan
00	MPEG-2.5 (Pengembangan dari MPEG-2)
01	*Reserved*
10	MPEG-2
11	MPEG-1

c. *Layer*

Bit *layer* terdiri dari dua bit. Tabel 2.5 menunjukkan beberapa *layer* yang ditentukan oleh bit *layer*.

Tabel 2.5 Nilai Bit Pada Bagian *Header* Bit *layer*

Nilai Bit	*Layer*
00	Tidak terdefinisi
01	Layer III

Nilai Bit	*Layer*
10	Layer II
11	Layer I

d. *Bitrate*

Bitrate merupakan informasi jumlah data yang akan disimpan dalam satuan detik pada audio yang belum terkompresi. *Bitrate* ditentukan sebelum dilakukan proses *encoding* sehingga menentukan kualitas dari hasil *encoding*. Pada *layer* III ditetapkan nilai *bitrate* antara 8kbit/s hingga 320 kbit/s dimana *default*-nya bernilai 128 kbit/s.

e. Bit *frequency*

Bit *frequency* atau *sampling frequency* didefinisikan sebagai suatu nilai sampel per detik yang dihasilkan dari sinyal *continuous* menjadi sinyal diskrit. Notasinya adalah hertz (Hz). Kebalikan dari *sampling frequency* adalah *period sampling*, yaitu waktu yang dibutuhkan untuk melakukan *sampling*. Tidak ada aturan baku yang mengatur berapa batas *sampling* yang diperkenankan pada satu *period sampling*. Pembagian nilai bit frekuensi ditunjukkan pada Tabel 2.6.

Tabel 2.6 Nilai Bit Pada Bagian *Header* Bit frekuensi

Nilai Bit	MPEG 1	MPEG 2
00	44100 Hz	22050 Hz
01	48000 Hz	24000 Hz
10	32000 Hz	16000 Hz

f. Bit *mode*

Bit *mode* dibedakan menjadi empat, yaitu :

1. *Mono*

   Audio *Mono* memiliki kanal audio hanya satu. Audio *mono* hanya menghasilkan satu suara yang didengar oleh kedua telinga kita, sehingga suara yang diterima oleh kedua telinga kita selalu sama. Sinyal *mono* adalah R+L (*right and left*), dimana R dan L digabungkan sehingga menjadi satu

sinyal R+L. Penggabungan tersebut dibuat agar bisa mendengan kedua sinyal dalam satu sumber suara.

2. *Stereo*

   Audio *stereo* dapat menghasilkan efek suara seperti efek *doppler*. Musik yang berbeda antar kanal, misalnya kanal R suara bass, kanal L suara gitar.

3. *Joint stereo*

   Joint stereo adalah suatu trik untuk mengkodekan sinyal frekuensi rendah secara *mono* sementara sisanya tetap *stereo*.

4. *Dual channel*

   Salah satu cara untuk menggabungkan audio jenis *stereo* dan juga audio *mono*.

Pembagian bit mode ditunjukkan pada Tabel 2.7

Tabel 2.7 Nilai Bit Pada Bagian *Header* Bit *Mode*

Nilai Bit	*Mode*
00	*Stereo*
01	*Joint stereo*
10	*Dual channel*
11	*Mono*

2. CRC (*Cyclic Redundancy Check*)

CRC adalah bagian dari sebuah *frame* dengan panjang 2 byte. Bagian ini hanya akan ada jika bit proteksi pada *header* diset dan memungkinkan untuk memeriksa data-data sensitif. Bit CRC ini oleh sebuah *frame* digunakan untuk memeriksa data-data sensitif pada *header* dan *side information*. Jika nilai-nilai pada CRC tersebut mempunyai kesalahan maka *frame* tersebut oleh MP3 *player* akan dinyatakan *corrupt* dan akan digantikan dengan *frame* sebelumnya. CRC adalah algoritma untuk memastikan integritas data dan memriksa kesalahan pada suatu data yang akan ditransmisikan atau disimpan. CRC bekerja secara sederhana, yakni dengan menggunakan perhitungan matematika terhadap sebuah bilangan yang disebut *checksum*, yang dibuat berdasarkan total bit yang hendak ditransmisikan atau yang hendak disimpan. *Checksum* akan dihitung terhadap setiap *frame* yang hendak ditransmisikan dan ditambahkan ke dalam *frame*

tersebut sebagai informasi dalam *header*. Penerima *frame* akan menghitung kembali apakah *frame* yang diterima benar-benar tanpa kerusakan, dengan membandingkan nilai *frame* yang dihitung dengan nilai *frame* yang terdapat pada *header frame*. Jika dua nilai tersebut berbeda, maka *frame* tersebut telah berubah.

3. *Side information*

Side information adalah bagian dari sebuah *frame* dengan panjang 17 byte untuk *mode single channel* dan 32 byte untuk mode yang lain. *Side information* mengandung informasi yang dibutuhkan untuk melakukan *decoding* bagian *main data*. Untuk mendapatkan kualitas audio yang normal, maka *side information* ini tidak boleh rusak dalam proses steganografi.

4. *Main data*

Main data adalah bagian dimana data audio dari sebuah berkas MP3 berada dan mempunyai panjang yang bervariasi. Disini terdapat *huffman code bits*, informasi untuk *decoding huffman code bits* ini terdapat pada bagian *side information*. Bagian *frame* akan digunakan dalam steganografi ini adalah dengan mengganti isi dari *main data*. Teknik kompresi MP3 juga menetapkan metode *huffman coding* untuk mendapatkan hasil kompresi yang lebih baik. Pada berkas MP3 yang tidak menggunakan CRC, umumnya letak *main data* dimulai setelah byte ke-36 hingga *frame* selanjutnya.

5. *Ancillary data*

Ancillary data merupakan bagian opsional, tidak banyak terdapat informasi pada bagian ini dan pada umumnya sebuah *frame* tidak mempunyai *ancillary data*.

2.5 MSE (*Mean Square Error*)

Mean square error adalah salah satu cara untuk menghitung nilai kedekatan antara estimator dengan nilai yang diestimasi. Jika *n* adalah jumlah sampel, *I* dan *K* masing-masing adalah estimator dan nilai yang ingin diestimasi untuk tiap sampel, maka MSE dapat dirumuskan pada persamaan 2.4.

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}\|I(i) - K(i)\|^2 \qquad (2.4)$$

dimana :

n = jumlah panjang byte dalam berkas.

i = indeks byte ke-*i*

I = nilai *byte* pada berkas awal.

K = nilai *byte* pada berkas *stego*.

Sehingga I(*i)* adalah nilai byte estimator ke-*i* yang dibandingkan dengan nilai dari yang diestimasi yaitu K(*i)* dengan nilai byte dari K ke-*i*.

2.6 PSNR (*Peak Signal to Noise Ratio*)

Dasar yang paling penting pada audio steganografi adalah penyembunyian data tidak boleh mengubah kualitas suara dari sinyal audio yang digunakan sebagai media penampung. Jadi pesan rahasia yang disembunyikan tidak boleh terdeteksi oleh pendengar [SAR-06]. Oleh karena itu, perlu dilakukan perbandingan antara dua buah berkas audio untuk mengetahui tingkat perbedaan kualitas antara berkas audio asli dengan berkas audio setelah proses steganografi. Ada beberapa metode untuk menghitung metrik perbandingan antara dua buah sinyal, diantaranya adalah PSNR.

Peak Signal to Noise Ratio (PSNR) adalah perbandingan antara nilai maksimum dari sinyal yang diukur dengan besarnya derau yang berpengaruh pada sinyal tersebut. PSNR biasanya diukur dalam satuan desibel. (ALA-09). PSNR digunakan untuk mengukur rasio antara berkas audio asli dengan berkas audio yang telah disisipi pesan. PSNR yang rendah menunjukkan bahwa berkas audio telah mengalami distorsi yang cukup besar. Kualitas audio yang baik memiliki nilai PSNR minimal 30 dB [NEY-11].

Persamaan PSNR dapat dilihat pada persamaan 2.5.

$$PSNR = 10\log_{10}\left(\frac{{MAX_I}^2}{MSE}\right) = 20\log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \quad (2.5)$$

Jika sinyal suara yang ingin dihitung mempunyai *bits per sample*, maka MAX_I dapat dirumuskan pada persamaan 2.6.

$$MAX_I = 2^I - 1 \quad (2.6)$$

dimana :

I = jumlah *bits per sample*.

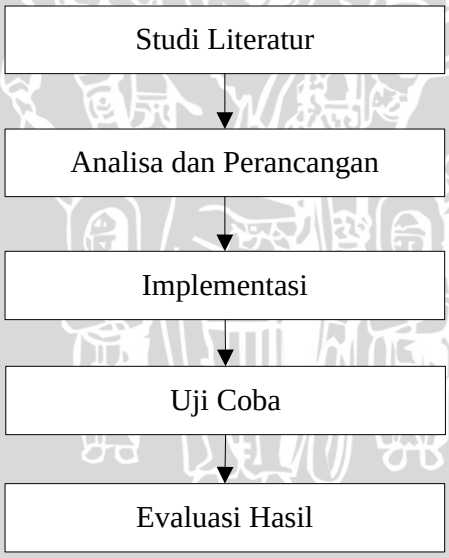
BAB III
METODOLOGI DAN PERANCANGAN

Pada bab ini dibahas mengenai metode dan langkah-langkah perancangan yang dilakukan untuk menyembunyikan pesan rahasia terenkripsi pada berkas audio MP3 menggunakan metode *least significant bit*.

Langkah-langkah yang dilakukan dalam penelitian ini meliputi :

1. Melakukan studi literatur mengenai *least significant bit*, algoritma kriptografi rijndael, dan literatur lain yang berkaitan..
2. Menganalisis dan melakukan perancangan sistem.
3. Mengimplementasikan dalam bentuk perangkat lunak berdasarkan analisis dan perancangan yang telah dilakukan.
4. Melakukan uji coba terhadap perangkat lunak yang telah dibuat.
5. Mengevaluasi *output* hasil analisis dari sistem.

Langkah-langkah penelitian ini digambarkan pada Gambar 3.1



Gambar 3.1 Alur Penelitian

3.1 Deskripsi Umum Sistem

Perangkat lunak yang akan dibuat merupakan perangkat lunak yang dapat menyembunyikan pesan rahasia berupa teks ke dalam sebuah berkas audio dengan format MP3. Perangkat lunak akan menerapkan teknik kriptografi dengan algoritma rijndael dan teknik steganografi dengan metode *least significant bit* (LSB).

Teknik kriptografi rijndael digunakan untuk mengenkripsi pesan supaya kerahasiaan pesan tetap terjaga. *Rijndael* melakukan enkripsi pada *plaintext* atau pesan asli menjadi sebuah *ciphertext* menggunakan sebuah kunci yg dimasukkan oleh pengguna perangkat lunak. Pesan yang sudah terenkripsi kemudian disembunyikan ke dalam media penampung berkas MP3 menggunakan teknik steganografi dengan metode LSB.

Pengungkapan pesan dilakukan untuk mendapatkan kembali pesan rahasia yang disembunyikan pada berkas MP3. Pesan diektraksi dari berkas MP3 dilanjutkan dengan dekripsi menggunakan algoritma AES dan menggunakan kunci melalui masukan pengguna, untuk mendapatkan pesan asli sehingga arti dari pesan tersebut dapat diketahui.

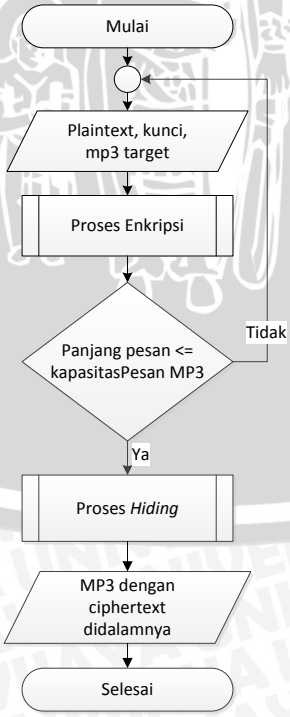
3.2 Perancangan Sistem

Perangkat lunak yang dibuat memiliki dua proses utama, yaitu proses penyembunyian pesan dan proses pengungkapan pesan. Proses penyembunyian pesan bertujuan untuk menjaga keamanan pesan dengan melakukan proses enkripsi dan proses penyisipan. Proses enkripsi menerima dua masukan berupa pesan asli untuk dienkripsi dan kunci untuk melakukan enkripsi. Setelah masukan pesan dienkripsi, kemudian disembunyikan ke dalam MP3 target, yaitu media penampung berupa berkas MP3 dengan melakukan proses penyisipan. Hasil dari proses penyisipan adalah *stego* MP3, yaitu berkas MP3 yang sudah disisipi pesan.
Langkah-langkah untuk proses penyembunyian pesan adalah sebagai berikut :

1. Masukan yang digunakan adalah pesan, kunci untuk proses enkripsi dan berkas MP3 yang akan digunakan sebagai media penampung (target MP3).
2. Setelah didapat masukan, pada pesan dan kunci dilakukan proses pengubahan pesan menjadi bentuk *byte*.

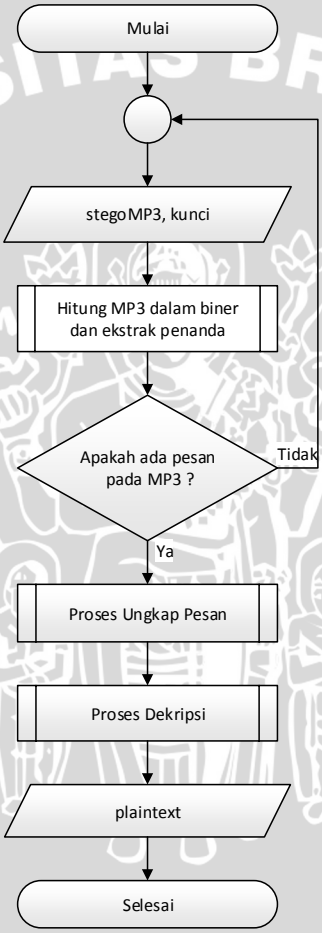
3. Dilakukan proses enkripsi dengan mengubah pesan asli atau *plaintext* menjadi *ciphertext* menggunakan kunci masukan.
4. Dilakukan pembacaan atribut pada berkas MP3 seperti ukuran berkas, jumlah kanal, dan jumlah *frame* MP3.
5. Dihitung panjang bit MP3 target yang dapat digunakan untuk menampung pesan, kemudian dibandingkan dengan panjang bit pesan yang sudah terenkripsi (*ciphertext*). Perbandingan dilakukan untuk mengetahui apakah target MP3 memiliki jumlah kapasitas yang dapat digunakan untuk menampung pesan. Apabila MP3 target dapat digunakan untuk menampung pesan, maka dilakukan langkah 6.
6. Dilakukan proses penyisipan pesan terenkripsi (*ciphertext*) pada target MP3 dengan mengganti bit-bit yang *least significant* pada *main data* tiap frame MP3.
7. Dilakukan proses penyimpanan hasil penyisipan sebagai hasil akhir proses penyembunyian pesan.
8. Hasil akhir adalah adalah *stego* MP3.

Diagram alir untuk proses penyembunyian pesan ditunjukkan pada Gambar 3.2.



Gambar 3.2 Diagram Alir Proses Penyembunyian

Setelah proses penyembunyian, dilakukan proses pengungkapan pesan. Proses pengungkapan pesan bertujuan untuk mendapatkan kembali pesan yang disembunyikan ke dalam berkas MP3. Pesan diperoleh melalui proses ungkap pesan pada *stego* MP3. Pesan tersebut masih berupa pesan terenkripsi atau *ciphertext*. Selanjutnya *ciphertext* akan diubah menjadi pesan asli atau *plaintext* pada proses dekripsi. Diagram alir untuk proses pengungkapan pesan ditunjukkan pada Gambar 3.3



Gambar 3.3 Diagram Alir Proses Pengungkapan

Langkah-langkah untuk proses pengungkapan pesan adalah sebagai berikut :

1. Masukan yang digunakan adalah berkas *stego* MP3 dan kunci.
2. Dilakukan pembacaan atribut pada berkas MP3 seperti ukuran berkas, jumlah kanal dan jumlah *frame* MP3.
3. Dilakukan pengecekan, apakah dalam berkas MP3 tersebut terdapat pesan yang disisipkan. Pengecekan dilakukan dengan mengungkap 8 bit

awal dari berkas MP3 masukan dan membandingkannya dengan *byte* penanda.

4. Dilakukan proses pengungkapan pesan dari berkas MP3 dengan melakukan pengambilan bit-bit yang *least significant* hingga dihasilkan pesan terenkripsi.
5. Dengan masukan kunci yang ada dilanjutkan dengan proses dekripsi dari pesan yang telah diungkap.
6. Dilakukan proses penyimpanan hasil dekripsi sebagai hasil akhir proses pengungkapan pesan.
7. Hasil akhir adalah sebuah berkas teks (*plaintext*).

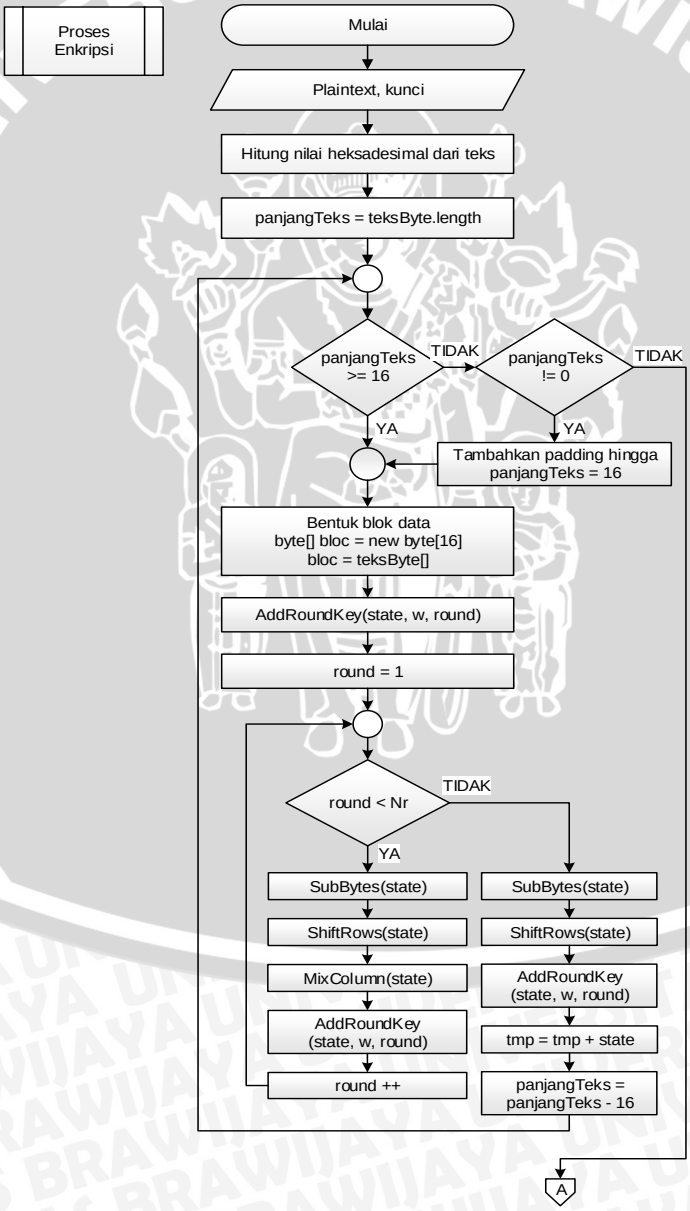
3.2.1 Enkripsi

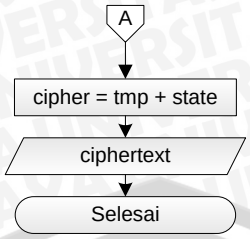
Enkripsi merupakan proses yang bertujuan untuk mengubah pesan asli atau *plaintext* menjadi pesan terenkripsi atau *ciphertext*. Langkah-langkah untuk proses enkripsi adalah sebagai berikut :

1. Masukan yang digunakan adalah pesan teks atau *plaintext* yang akan dienkripsi dan kunci untuk enkripsi.
2. Pesan teks diubah ke dalam bentuk *array byte*.
3. Dibentuk matrik blok data berukuran 4x4 yang berisikan 16 byte dari pesan teks. Untuk pesan teks dengan panjang kurang dari 16 bytes dilakukan penambahan *padding* hingga panjang 16 byte terpenuhi. Jika pesan teks memiliki panjang 16 byte atau lebih, maka akan dibentuk matriks blok data setiap 16 byte dan dilakukan proses enkripsi yang sama setiap blok data yang dibentuk. Setelah dibentuk blok data pertama, maka dilakukan langkah ke 4
4. Dilakukan proses enkripsi terhadap blok data pesan teks diawali dengan *AddRoundKey*, dilanjutkan dengan *SubBytes*, *ShiftRows*, *MixColumns* dan *AddRoundKey* untuk *round* sama dengan 1 hingga *Nr*-1. Untuk langkah terakhir (*Nr*) dilakukan *SubBytes*, *ShiftRows*, dan *AddRoundKey*. Hasil dari proses ini disimpan dalam variabel tmp dan dilakukan penggabungan dengan hasil tmp sebelumnya.

5. Dilanjutkan dengan pengecekan apakah panjang pesan teks yang tersisa lebih dari atau sama dengan 16 byte. Jika iya dilakukan langkah 4, jika tidak dilakukan penambahan padding hingga panjang pesan sama dengan 16 byte dan dilakukan langkah 4.
6. Dilakukan proses penggabungan *byte* pesan *ciphertext* hingga dihasilkan hasil akhir proses enkripsi.
7. Hasil akhir proses enkripsi diperoleh *ciphertext* atau pesan yang telah terenkripsi.

Diagram alir untuk proses enkripsi ditunjukkan pada Gambar 3.4.





Gambar 3.4 Diagram Alir Proses Enkripsi.

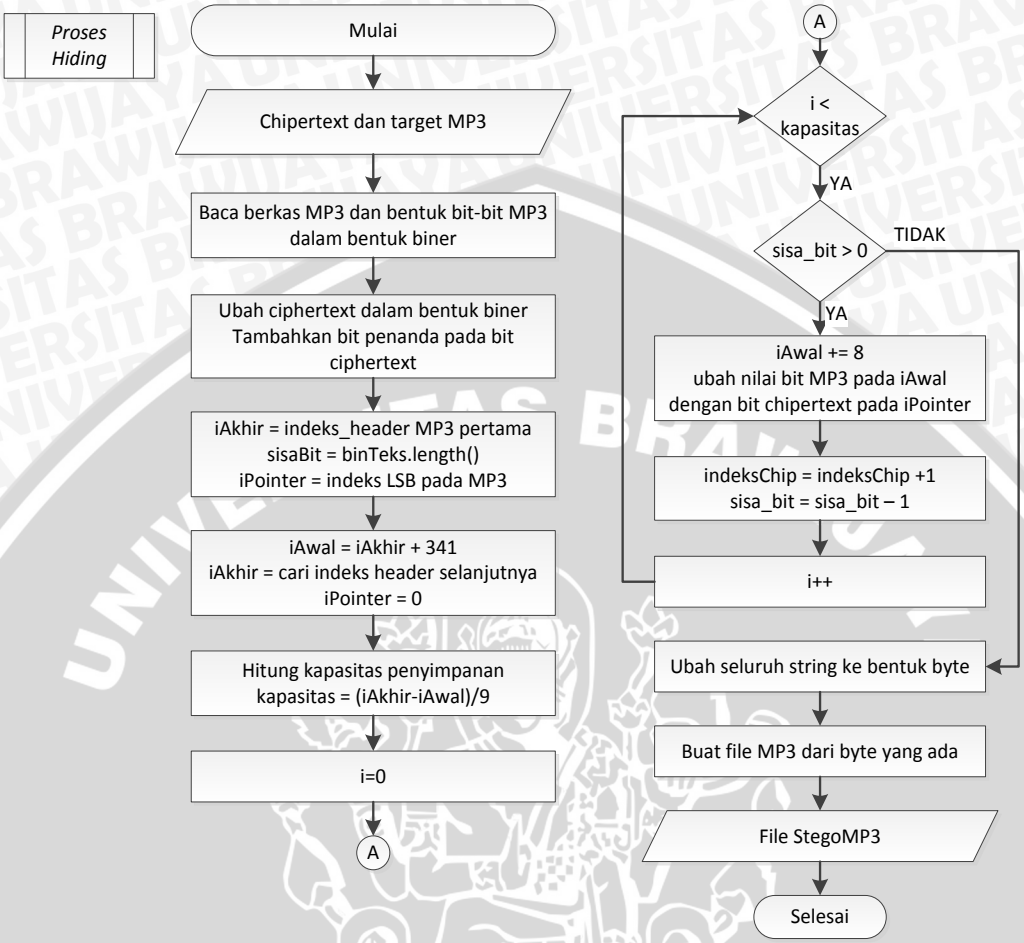
3.2.2 Penyisipan

Proses penyisipan bertujuan untuk menyembunyikan pesan terenkripsi (*ciphertext*) pada berkas MP3. Proses penyisipan ini merupakan implementasi dari metode *least significant bit* yang mengubah bit-bit *least significant* pada berkas MP3. Bit yang diubah adalah bit dari *main data* pada setiap frame dalam berkas MP3.

Langkah-langkah untuk proses penyisipan adalah sebagai berikut :

1. Masukan yang digunakan adalah target MP3 dan *ciphertext*.
2. Dilakukan pembacaan berkas MP3 dan pengubahan menjadi bentuk biner.
3. Dilakukan pengubahan *ciphertext* ke dalam bentuk biner dan dilakukan penambahan *byte* penanda.
4. Digunakan pointer iAwal dan iAkhir untuk menandai posisi awal dan akhir dari sebuah *main data* dalam *frame* MP3.
5. Bit pertama pada *main data* yang dapat disisipi pesan berjarak 341 bit dari pointer iAwal yang berada pada posisi *header frame* ke-*n*. jarak 341 bit merupakan nilai dari panjang *bit header* yang dilewati untuk mendapatkan bit *main data*. Pointer iAkhir sebagai penanda posisi bit pertama tersebut.
6. Dilakukan perulangan sesuai kapasitas *main data* yang dapat disisipkan pesan dalam *frame* tersebut dengan menggunakan iPointer sebagai pointer untuk mengubah bit *least significant* pada MP3.
7. Jika sudah tidak ada bit *chiper* yang disisipkan, Proses penyisipan selesai.
8. Didapatkan *stego* MP3 sebagai hasil akhir dari proses penyisipan.

Diagram alir untuk proses penyisipan ditunjukkan pada Gambar 3.5.



Gambar 3.5 Diagram Alir Proses *Embedding.*

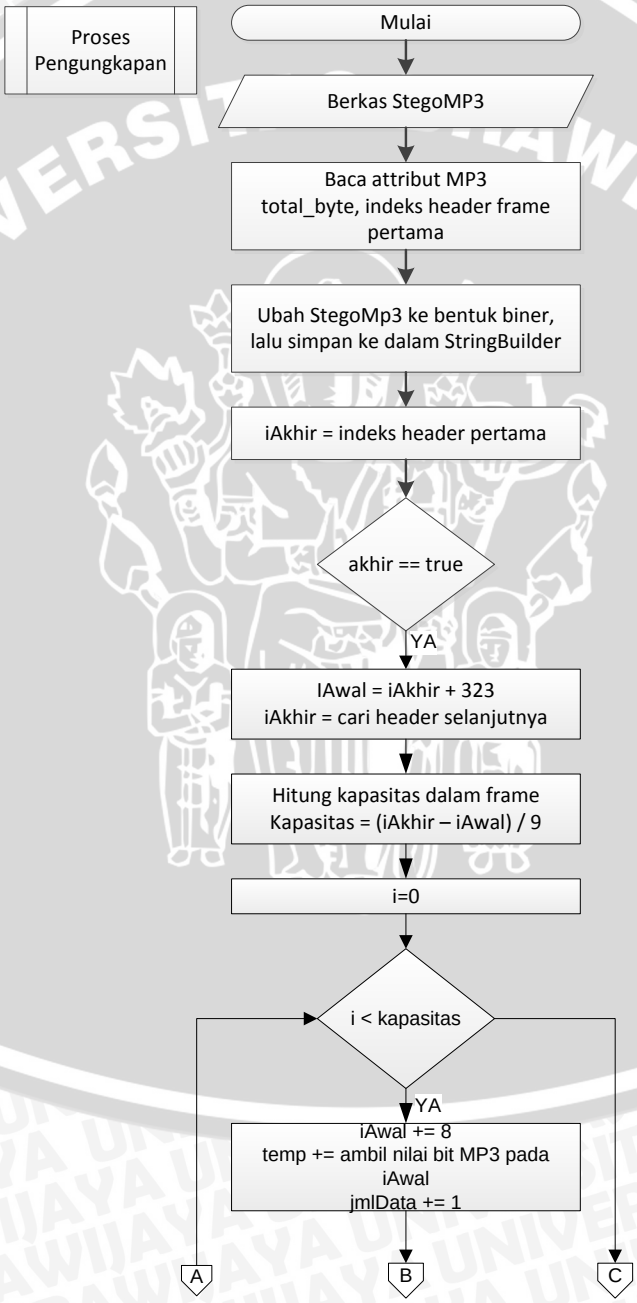
3.2.3 Pengungkapan.

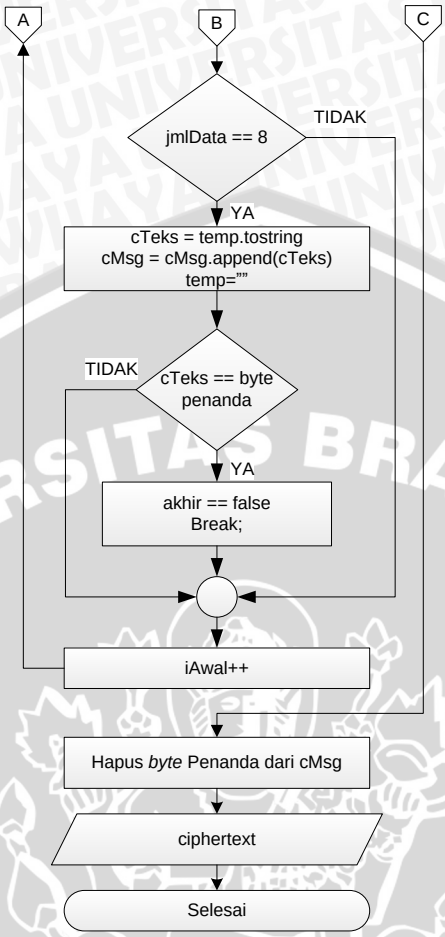
Proses pengungkapan pesan bertujuan untuk mendapatkan kembali pesan yang disembunyikan di dalam *stego* MP3. Langkah-langkah untuk proses *retrieving* adalah sebagai berikut :

1. Masukan yang digunakan adalah *stego* MP3.
2. Berkas *Stego* MP3 diubah kedalam bentuk biner, kemudian diproses dalam bentuk StringBuilder.
3. Digunakan iAwal dan iAkhir sebagai penanda awal dan akhir dari *main data* dalam suatu *frame.*
4. Dihitung kapasitas *frame* dalam menampung bit *ciphertext.*
5. Dengan iAwal sebagai penanda awal dari *main data*, dilakukan perulangan untuk mengekstrak bit *least significant*.

6. Setiap 8 bit yang didapat dari proses ekstraksi bit *least significant*, dilakukan proses pembentukan string.
7. Proses ektraksi bit *least significant* berhenti ketika ditemukan karakter penanda.
8. Dari hasil ekstraksi didapatkan hasil akhir berupa pesan *ciphertext*.

Diagram alir untuk proses *retrieving* ditunjukkan pada Gambar 3.6.





Gambar 3.6 Diagram Alir Proses *Retrieving*

3.2.4 Dekripsi

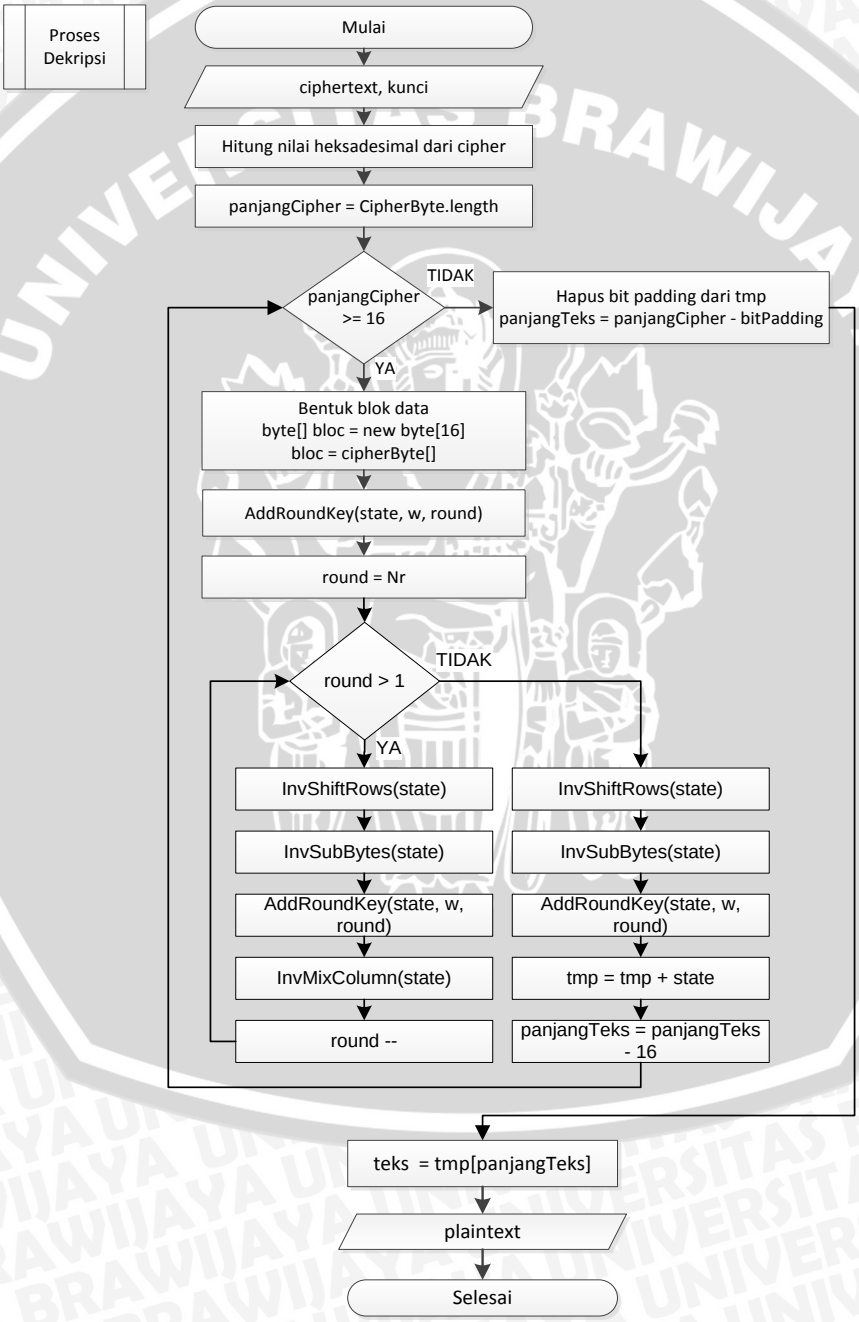
Dekripsi merupakan proses yang bertujuan untuk mengubah pesan terenkripsi atau *ciphertext* menjadi pesan asli atau *plaintext*. Langkah-langkah untuk proses enkripsi adalah sebagai berikut :

1. Masukan yang digunakan adalah pesan terenkripsi (*ciphertext*) dan kunci.
2. *Ciphertext* dalam bentuk String diubah kedalam bentuk heksadesimal.
3. Dihitung panjang cipher secara keseluruhan. Apabila panjang cipher lebih dari atau sama dengan 16 byte dilakukan langkah 4.
4. Dilakukan proses secara bertahap pada blok data *ciphertext* diawali dengan *AddRoundKey*, dilanjutkan dengan *InvShiftRows*, *InvSubBytes*, *InvMixColumns* dan *AddroundKey* untuk *round* sama dengan 1 hingga *Nr*-1.

Untuk langkah terakhir (*Nr*) dilakukan *InvShiftRows, InvSubBytes* dan *AddroundKey*.

5. Jika panjang cipher kurang dari 16, maka dilakukan proses penghapusan bit *padding*
6. Hasil akhir diperoleh pesan asli atau *plaintext*.

Diagram alir untuk proses enkripsi ditunjukkan pada Gambar 3.7.



Gambar 3.7 Diagram Alir Proses Dekripsi

3.3 Perhitungan Manual

Pada subbab ini akan dijelaskan mengenai contoh perhitungan manual pada penyembunyian pesan dengan algoritma kriptografi Rijndael dan disisipkan pada berkas MP3 menggunakan metode Least Significant Bit (LSB). Perhitungan manual yang dibahas adalah proses penyembunyian pesan ke dalam berkas MP3 dan pengungkapan kembali pesan yang telah disembunyikan. Pesan yang digunakan pada perhitungan manual ini adalah sebuah teks "arif nur yahya" dengan kunci "1234567890abcdef" dengan panjang kunci 128 bit dan besar blok data 128 bit.

3.3.1 Penyembunyian Pesan

Proses penyembunyian pesan meliputi proses pembentukan subkunci, proses enkripsi *plaintext* menjadi sebuah *ciphertext* dan proses penyisipan ke dalam berkas MP3.

3.3.1.1 Pembentukan subkunci

Proses pembentukan subkunci diawali dengan mengubah kunci masukan kedalam bentuk heksadesimal, untuk kunci masukan berupa "1234567890abcdef" didapatkan nilai heksadesimal sebagai berikut :

31 32 33 34 35 36 37 38 39 30 61 62 63 64 65 66

Dilanjutkan dengan membentuk blok data berbentuk matriks dua dimensi berukuran 4x4 seperti pada Gambar 3.8. Dilakukan perhitungan manual untuk w4 hingga w43. Pembentukan subkunci dilakukan dengan menghitung kunci selanjutnya tiap 4 byte atau per kolom yang disebut *word*. Berikut ini adalah langkah-langkah pembentukan subkunci pada algoritma Rijndael dengan kunci masukan.

Kunci masukan			
1	5	9	c
2	6	0	d
3	7	a	e
4	8	b	f

w0	w1	w2	w3	w4	w5	w6	w7
31	35	39	63	?	?	?	?
32	36	30	64	?	?	?	?
33	37	61	65	?	?	?	?
34	38	62	66	?	?	?	?
subkunci(0)				subkunci(1)			

Gambar 3.8 Bentuk Kunci Awal Dari w0 Hingga w3.

Untuk mendapatkan nilai W4, dilakukan perhitungan sesuai persamaan 2.1 :

- *RotWord* : pergeseran byte dalam satu kolom. Kolom yang dilakukan pergeseran (Wi-1) memiliki nilai heksadesimal 63 64 65 66.
  - 63 64 65 66 dilakukan operasi *RotWord* dihasilkan 64 65 66 63
- *SubWord* : merupakan proses substitusi setiap *byte* pada hasil *RotWord* dengan aturan 4 bit paling kiri sebagai baris dan 4 bit paling kanan sebagai kolom terhadap tabel S-Box.
  - 64 = 01100100 = 0110 & 0100 = 6 & 4
    disubtitusikan dengan tabel S-Box didapatkan hasil 43
  - 65 = 01100101 = 0101 & 0101 = 6 & 5
    disubtitusikan dengan tabel S-Box didapatkan hasil 4D
  - 66 = 01100110 = 0110 & 0110 = 6 & 6
    disubtitusikan dengan tabel S-Box didapatkan hasil 33
  - 63 = 01100011 = 0110 & 0011 = 6 & 3
    disubtitusikan dengan tabel S-Box didapatkan hasil FB
    dari hasil substitusi didapatkan hasil *SubWord* 43 4D 33 FB
- Fungsi XOR, berdasarkan persamaan 2.1 :
  - b1 = 31 xor 43 xor 01
    = 00110001 xor 01000011 xor 00000001
    = 01110011 = 73
  - b2 = 32 xor 4D xor 00
    = 00110010 xor 01001101 xor 00000000
    = 01111111 = 7F
  - b3 = 33 xor 33 xor 00
    = 00110011 xor 00110011 xor 00000000
    = 00000000 = 00
  - b4 = 34 xor FB xor 00
    = 00110100 xor 11111011 xor 00000000
    = 11001111 = CF

  Dari perhitungan diatas didapatkan nilai w4 = 73 7F 00 CF

Setelah didapatkan nilai k4 dilakukan perhitungan nilai W5 ;

- Fungsi XOR berdasarkan persamaan 2.2, dimana W[i-4] = 35 36 37 38 dan W[i-1] = 73 7F 00 CF. berikut proses perhitungan yang dilakukan :
  - b1 = 35 xor 73
    = 00110101 xor 01110011 = 01000100 = 46
  - b2 = 36 xor 7F
    = 00110110 xor 01111111 = 01001001 = 49
  - b3 = 37 xor 00
    = 00110111 xor 00000000 = 00110111 = 37
  - b4 = 38 xor CF
    = 00111000 xor 11001111 = 11110111 = F7

  Dari perhitungan diatas didapatkan nilai W5 = 46 49 37 F7

Dilanjutkan dengan perhitungan nilai W6

- Fungsi xor sesuai persamaan 2.3, dimana W[i-4] = 39 30 61 62 dan W[i-1] = 46 49 37 F7. berikut proses perhitungan yang dilakukan :
  - b1 = 39 xor 46
    = 00111001 xor 01000110 = 01111111 = 7F
  - b2 = 30 xor 49
    = 00110000 xor 01001001 = 01111001 = 79
  - b3 = 61 xor 37
    = 01100001 xor 00110111 = 01010110 = 56
  - b4 = 62 xor F7
    = 01100010 xor 11110111 = 10010101 = 95

  Dari perhitungan diatas didapatkan nilai k6 = 7F 79 56 95

Untuk nilai W[i] selanjutnya, jika nilai i mod 4 sama dengan 0 maka dilakukan operasi sesuai persamaan 2.2 dan jika i mod 4 tidak sama dengan 0, maka akan dilakukan fungsi xor tanpa *RotWord, SubWord,* dan *RCon* seperti persamaan 2.3. Hasil akhir dari seluruh proses pembentukan kunci dapat dilihat pada Gambar 3.9.

	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15
b1	31	35	39	63	73	46	7F	1C	D5	93	EC	F0	50	C3	2F	DF
b2	32	36	30	64	7F	49	79	1D	BC	F5	8C	91	73	86	0A	9B
b3	33	37	61	65	00	37	56	33	0D	3A	6C	5F	28	12	7E	21
b4	34	38	62	66	CF	F7	95	F3	53	A4	31	C2	DF	7B	4A	88

	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26	W27	W28	W29	W30	W31
b1	4C	8F	A0	7F	B2	3D	9D	E2	87	BA	27	C5	F9	43	64	A1
b2	8E	08	02	99	BC	B4	B6	2F	FC	48	FE	D1	C6	8E	70	A1
b3	EC	FE	80	A1	AD	53	D3	72	50	03	D0	A2	EE	ED	3D	9F
b4	41	3A	70	F8	93	A9	D9	21	0B	A2	7B	5A	AD	0F	74	2E

	W32	W33	W34	W35	W36	W37	W38	W39	W40	W41	W42	W43
b1	4B	08	6C	CD	7C	74	18	D5	CE	BA	A2	77
b2	1D	93	E3	42	7D	EE	0D	4F	12	FC	F1	BE
b3	DF	32	0F	90	AB	99	96	06	75	EC	7A	7C
b4	9F	90	E4	CA	22	B2	56	9C	21	93	C5	59

Gambar 3.9 Blok Data Hasil Pembentukan Sub Kunci.

3.3.1.2 Enkripsi

Perhitungan proses enkripsi diawali dengan mengubah pesan teks ke dalam bentuk heksadesimal. Sehingga *plaintext* yang berisikan teks "arif nur yahya" didapatkan nilai heksadesimal "41 72 69 66 20 4E 75 72 20 59 61 68 79 61". Karena teks "arif nur yahya" memiliki panjang kurang dari 16 byte, maka perlu ditambahkan *padding* 2 karakter menjadi "arif nur yahya " dan dilanjutkan dengan mengubahnya menjadi sebuah blok data. Gambar 3.10 merupakan bentuk blok data dari *plaintext*.

61	20	20	79
72	6E	79	61
69	75	61	..
66	72	68	..

61	20	20	79	
72	6E	79	61	
69	75	61	20	<= padding
66	72	68	20	<= padding

Gambar 3.10 Blok Data Teks "arif nur yahya" Dengan 2 Karakter *Padding*.

Setelah blok data dan subkunci dibentuk, dilanjutkan dengan proses enkripsi. Proses enkripsi diawali dengan *AddRoundKey*, dilanjutkan dengan *SubBytes, ShiftRows, MixColumns* dan *AddroundKey* untuk *round* sama dengan 1 hingga *Nr*-1. Untuk langkah terakhir (*Nr*) dilakukan *SubBytes, ShiftRows*, dan *AddroundKey*.

Berikut proses enkripsi tahap awal (*round* 0).

- *AddRoundKey* : dengan melakukan xor pada blok data(0) dan subkunci(0) dihasilkan *state*(0). Berikut perhitungan proses *AddRoundKey* untuk setiap *byte* pada *plaintext* :
  a) 61 xor 31 = 01100001 xor 00110001 = 01010000 = 50
  b) 72 xor 32 = 01110010 xor 00110010 = 01000000 = 40
  c) 69 xor 33 = 01101001 xor 00110011 = 01011010 = 5A
  d) 66 xor 34 = 01100110 xor 00110100 = 01010010 = 52
  e) 20 xor 35 = 00100000 xor 00110101 = 00010101 = 15
  f) 6E xor 36 = 01101110 xor 00110110= 01001000 = 58
  g) 75 xor 37 = 01110101 xor 00110111 = 00000010 = 42
  h) 72 xor 38 = 01110010 xor 00111000 = 01001010 = 4A
  i) 20 xor 39 = 00100000 xor 00111001 = 00011001 = 19
  j) 79 xor 30 = 01111001 xor 00110000 = 01001001 = 49
  k) 61 xor 61 = 01100001 xor 01100001 = 00000000 = 00
  l) 68 xor 62 = 01101000 xor 01100010 = 00001010 = 0A
  m) 79 xor 63 = 01111001 xor 01100011 = 00011010 = 1A
  n) 61 xor 64 = 01100001 xor 01100100 = 00000101 = 05
  o) 20 xor 65 = 00100000 xor 01100101 = 01000101 = 45
  p) 20 xor 66 = 00100000 xor 01100110 = 01000110 = 46

Dari perhitungan diatas didapatkan hasil *state*(0) sama dengan 50 40 5A 52 15 58 42 4A 19 49 00 0A 1A 05 45 46. Hasil *state*(0) dalam bentuk blok data dapat dilihat pada Gambar 3.11.

61	20	20	79
72	6E	79	61
69	75	61	20
66	72	68	20

blokdata(0)

31	35	39	63
32	36	30	64
33	37	61	65
34	38	62	66

subkunci(0)

50	15	19	1A
40	58	49	05
5A	42	00	45
52	4A	0A	46

hasil *state* (0)

Gambar 3.11 Operasi *AddRoundKey* (*round* 0) Menghasilkan *state*(0).

Dilanjutkan enkripsi tahap kesatu (*round* 1).

- *SubBytes* : merupakan proses substitusi setiap *byte* pada *state* dengan aturan 4 bit paling kiri sebagai baris dan 4 bit paling kanan sebagai kolom terhadap tabel S-Box. Tabel S-Box dengan hasil substitusi dapat dilihat pada Gambar

3.12. Proses ini menghasilkan blok data *SubBytes*(1). Berikut proses substitusi *state*(0) terhadap tabel S-Box :

- 50 = 01010000 = 0101 & 0000 = 5 & 0
  disubtitusikan dengan tabel S-Box didapatkan hasil 53
- 40 = 01000000 = 0100 & 0000 = 4 & 0
  disubtitusikan dengan tabel S-Box didapatkan hasil 09
- 5A = 01011010 = 0101 & 1010 = 5 & A
  disubtitusikan dengan tabel S-Box didapatkan hasil BE
- 52 = 01010010 = 0101 & 0010 = 5 & 2
  disubtitusikan dengan tabel S-Box didapatkan hasil 00
- 15 = 00010101 = 0001 & 1010 = 1 & 5
  disubtitusikan dengan tabel S-Box didapatkan hasil 59
- 58 = 01011000 = 0101 & 1000 = 5 & 8
  disubtitusikan dengan tabel S-Box didapatkan hasil 6A
- 42 = 01000010 = 0100 & 0010 = 4 & 2
  disubtitusikan dengan tabel S-Box didapatkan hasil 2C
- 4A = 01001010 = 0100 & 1010 = 4 & A
  disubtitusikan dengan tabel S-Box didapatkan hasil D6
- 19 = 00011001 = 0001 & 1001 = 1 & 9
  disubtitusikan dengan tabel S-Box didapatkan hasil D4
- 49 = 01001001 = 0100 & 1001 = 4 & 9
  disubtitusikan dengan tabel S-Box didapatkan hasil 3B
- 00 = 00000000 = 0000 & 0000 = 0 & 0
  disubtitusikan dengan tabel S-Box didapatkan hasil 63
- 0A = 00001010 = 0000 & 1010 = 0 & A
  disubtitusikan dengan tabel S-Box didapatkan hasil 67
- 1A = 00011010 = 0001 & 1010 = 1 & A
  disubtitusikan dengan tabel S-Box didapatkan hasil A2
- 05 = 00000101 = 0000 & 0101 = 0 & 5
  disubtitusikan dengan tabel S-Box didapatkan hasil 6B
- 45 = 010000101 = 0100 & 0101 = 4 & 5

disubtitusikan dengan tabel S-Box didapatkan hasil 6E

- 46 = 010000110 = 0100 & 0110 = 4 & 6

  disubtitusikan dengan tabel S-Box didapatkan hasil 5A

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
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

Gambar 3.12 Tabel S-Box yang Menunjukkan Hasil *SubBytes*(1).

Setelah dilakukan proses substitusi didapatkan hasil *SubBytes*(1) sama dengan 53 09 BE 00 59 6A 2C D6 D4 3B 63 67 A2 6B 6E 5A. Hasil *SubBytes*(1) dalam bentuk blok data dapat dilihat pada Gambar 3.13

50	15	19	1A
40	58	49	05
5A	42	00	45
52	4A	0A	46

state (0)

53	59	D4	A2
09	6A	3B	6B
BE	2C	63	6E
00	D6	67	5A

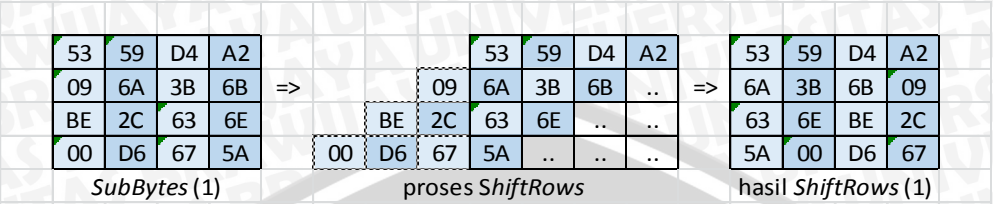
hasil *SubBytes* (1)

Gambar 3.13 Bentuk Blok Data *SubBytes*(1) Setelah Proses Substitusi.

- *ShiftRows* : proses pergesaran *byte* dilakukan dengan aturan kolom *byte* digeser ke kiri setiap baris sebanyak baris n-1. proses pergeseran *byte* diilustrasikan pada Gambar 3.14. berikut proses pergeseran yang dilakukan :

- Baris ke-1 53 59 D4 A2 dilakukan pergeseran baris sebanyak 0 kali menghasilkan 53 59 D4 A2
- Baris ke-2 09 6A 3B 6B dilakukan pergeseran baris sebanyak 1 kali menghasilkan 6A 3B 6B 09
- Baris ke-3 BE 2C 63 6F dilakukan pergeseran baris sebanyak 2 kali menghasilkan 63 6F BE 2C

- Baris ke-4 00 D6 67 5A dilakukan pergeseran baris sebanyak 3 kali menghasilkan 5A 00 D6 67



Gambar 3.14 Proses geser kolom tiap baris sebanyak baris n-1.

- *MixColumn* : proses *MixColumn* dilakukan pada blok data *ShiftRows*(1) dan dihasilkan *MixColumn*(1). Berikut proses *MixColumn* untuk setiap kolom pada blok data *ShiftRows*(1).

- b1 = 53*2 xor 6A*3 xor 63*1 xor 5A*1
  = E(L53+L02) xor E(L6A+L03) xor 63 xor 5A
  = A6 xor BE xor 63 xor 5A
  = 21
- b2 = 53*1 xor 6A*2 xor 63*3 xor 5A*1
  = 53 xor E(L6A+L02) xor E(L63+L03) xor 5A
  = 53 xor D4 xor A5 xor 5A
  = 78
- b3 = 53*1 xor 6A*1 xor 63*2 xor 5A*3
  = 53 xor 6A xor E(L63+L02) xor E(L5A+L03)
  = 53 xor 6A xor C6 xor EE
  = 11
- b4 = 53*3 xor 6A*1 xor 63*1 xor 5A*2
  = E(L53+L03) xor 6A xor 63 xor E(L5A+L02)
  = F5 xor 6A xor 63 xor B4
  = 48
- b5 = 59*2 xor 3B*3 xor 6E*1 xor 00*1
  = E(L59+L02) xor E(L3B+L03) xor 6E xor 00
  = B2 xor 4D xor 6E xor 00
  = 91
- b6 = 59*1 xor 3B*2 xor 6E*3 xor 00*1

= 59 xor E(L3B+L02) xor E(L6E+L03) xor 00

= 59 xor 76 xor B2 xor 00

= 9D

- b7 = 59*1 xor 3B*1 xor 6E*2 xor 00*3

  = 59 xor 3B xor E(L6E+L02) xor 00

  = 59 xor 3B xor DC xor 00

  = BE

- b8 = 59*3 xor 3B*1 xor 6E*1 xor 00*2

  = E((L59)+(L03)) xor 3B xor 6E xor 00

  = EB xor 3B xor 6E xor 00

  = BE

- b9 = D4*2 xor 6B*3 xor BE*1 xor D6*1

  = E(LD4+L02) xor E(L6B+L03) xor BE xor D6

  = B3 xor BD xor BE xor D6

  = 66

- b10 = D4*1 xor 6B*2 xor BE*3 xor D6*1

  = D4 xor E(L6B+L02) xor E(LBE+L03) xor D6

  = D4 xor D6 xor D9 xor D6

  = 0D

- b11 = D4*1 xor 6B*1 xor BE*2 xor D6*3

  = D4 xor 6B xor E(LBE+L02) xor E(LD6+L03)

  = D4 xor 6B xor 67 xor 61

  = B9

- b12 = D4*3 xor 6B*1 xor BE*1 xor D6*2

  = E(LD4+L03) xor 6B xor BE xor E(LD6+L02)

  = 67 xor 6B xor BE xor B7

  = 05

- b13 = A2*2 xor 09*3 xor 2C*1 xor 67*1

  = E(LA2+L02) xor E(L09+L03) xor 2C xor 67

  = 5F xor 1B xor 2C xor 67

  = 0F

- b14 = A2*1 xor 09*2 xor 2C*3 xor 67*1
  = A2 xor E(L09+L02) xor E(L2C+L03) xor 67
  = A2 xor 12 xor 74 xor 67
  = A3
- b15 = A2*1 xor 09*1 xor 2C*2 xor 67*3
  = A2 xor 09 xor E(L2C+L02) xor E(L67+L03)
  = A2 xor 09 xor 58 xor A9
  = 5A
- b16 = A2*3 xor 09*1 xor 2C*1 xor 67*2
  = E(LA2+L03) xor 09 xor 2C xor E(L67+L02)
  = FD xor 09 xor 2C xor CE
  = 16

Dari perhitungan diatas didapatkan hasil *MixColumn*(1) sama dengan 21 78 11 48 91 9D BE BE BE 66 0D B9 05 0F A3 5A 16. Hasil *MixColumn*(1) dalam bentuk blok data dapat dilihat pada Gambar 3.15.

53	59	D4	A2
6A	3B	6B	09
63	6E	BE	2C
5A	00	D6	67

ShiftRows (1)

=>

b1	b5	b9	b13
b2	b6	b10	b14
b3	b7	b11	b15
b4	b8	b12	b16

=>

21	91	66	0F
78	9D	0D	A3
11	BE	B9	5A
48	BE	05	16

hasil *MixColumn* (1)

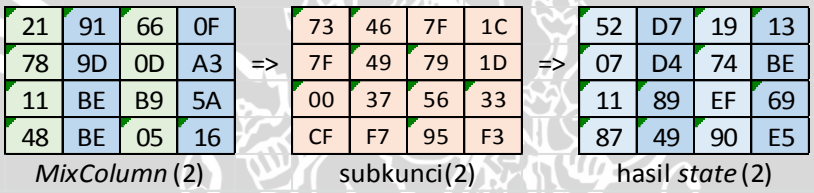
Gambar 3.15 Hasil operasi *MixColumn*(1) pada *ShiftRows*(1).

- *AddRoundKey*(1) : dengan melakukan xor pada hasil *MixColumn*(1) dengan subkunci(1) dihasilkan *state*(1). Berikut perhitungan proses *AddRoundKey*(1) hingga dihasilkan blok data *state*(1) :

- 21 xor 73 = 00100001 xor 01110011 = 01010010 = 52
- 78 xor 7F = 01111000 xor 01111111 = 00000111 = 07
- 11 xor 00 = 00010001 xor 00000000 = 00010001 = 11
- 48 xor CF = 01001000 xor 11001111 = 10000111 = 87
- 91 xor 46 = 1010001 xor 01000110 = 11010111 = D7
- 9D xor 79 = 10011101 xor 01001001 = 11010100 = D4
- BE xor 37 = 10111110 xor 00110111 = 10001001 = 89
- BE xor F7 = 10111110 xor 11110111 = 01001001 = 49

- 66 xor 7F = 01100110 xor 01111111 = 00011001 = 19
- 0D xor 79 = 00001101 xor 01111001 = 01110100 = 74
- B9 xor 56 = 10111001 xor 01010110 = 11101111 = EF
- 05 xor 95 = 00000101 xor 10010101 = 10010000 = 90
- 0F xor 1C = 00001111 xor 00011100 = 00010011 = 13
- A3 xor 1D = 10100011 xor 00011101 = 10111110 = BE
- 5A xor 33 = 01011010 xor 00110011 = 01101001 = 69
- 16 xor F3 = 00010110 xor 11110011 = 11100101 = E5

Dari perhitungan diatas, dihasilkan *state*(1) sama dengan 52 07 11 87 D7 D4 89 49 19 74 EF 90 13 BE 69 E5. Hasil *state*(1) dalam bentuk blok data dapat dilihat pada Gambar 3.16.



Gambar 3.16 Hasil *state*(2) setelah dilakukan operasi *AddRoundKey*.

Dengan urutan proses yang sama (*SubBytes*, *ShiftRows*, *MixColumn* dan *AddRoundKey*) dilakukan proses enkripsi tahap kedua hingga ketiga (Nr-1). Dari proses tersebut dihasilkan blok data pada Tabel 3.1.

Tabel 3.1 Hasil Dari Setiap Proses Enkripsi Setiap Tahap Hingga Nr-1.

	SubBytes (r)	*ShiftRow* (r)	*MixColumn* (r)	*State* (r)
r0				50 15 19 1A 40 58 49 05 5A 42 00 45 52 4A 0A 46
r1	53 59 D4 A2 09 6A 3B 6B BE 2C 63 6E 00 D6 67 5A	53 59 D4 A2 6A 3B 6B 09 63 6E BE 2C 5A 00 D6 67	21 91 66 0F 78 9D 0D A3 11 BE B9 5A 48 BE 05 16	52 D7 19 13 07 D4 74 BE 11 89 EF 69 87 49 90 E5
r2	00 0E D4 7D C5 48 92 AE 82 A7 DF F9 17 3B 60 D9	00 0E D4 7D 48 92 AE C5 DF F9 82 A7 D9 17 3B 60	DE 5F E3 69 33 36 35 7E 9D 4C 28 4D 3E 57 3D 25	0B CC 0F 99 8F C3 B9 EF 90 76 44 12 6D F3 0C E7

	SubBytes (r)				*ShiftRow* (r)				*MixColumn* (r)				*State* (r)			
r3	2B	4B	76	EE	2B	4B	76	EE	AB	99	FB	94	FB	5A	D4	4B
	73	2E	56	DF	2E	56	DF	73	CE	9B	7E	BE	BD	1D	74	25
	60	38	1B	C9	1B	C9	60	38	94	D0	7E	F4	BC	C2	00	D5
	3C	0D	FE	94	94	3C	0D	FE	7B	3A	3F	85	A4	41	75	0D
r4	0F	BE	48	B3	0F	BE	48	B3	5D	80	37	4B	11	0F	97	34
	7A	A4	92	3F	A4	92	3F	7A	2E	CD	1A	B5	A0	C5	18	2C
	65	25	63	03	63	03	65	25	0F	F1	23	3F	E3	0F	A3	9E
	49	83	9D	D7	D7	49	83	9D	63	DA	9F	B0	22	E0	EF	48
r5	82	76	88	18	82	76	88	18	B6	98	68	A2	04	A5	F5	40
	E0	A6	AD	71	A6	AD	71	E0	99	B9	B8	86	25	0D	0E	A9
	11	76	0A	0B	0A	0B	11	76	C6	63	E3	6E	6B	30	30	1C
	93	E1	DF	52	52	93	E1	DF	95	01	3A	1B	06	A8	E3	3A
r6	F2	06	E6	09	F2	06	E6	09	19	19	04	46	9E	A3	23	83
	3F	D7	AB	D3	D7	AB	D3	3F	CB	9B	18	6A	37	D3	E6	BB
	7F	04	04	9C	04	9C	7F	04	B6	3F	96	0D	E6	3C	46	AF
	6F	C2	11	80	80	6F	C2	11	C5	E3	02	02	CE	41	79	58
r7	0B	0A	26	EC	0B	0A	26	EC	8C	6F	64	2B	75	2C	00	8A
	9A	66	8E	EA	66	8E	EA	9A	43	0D	E3	53	85	83	93	F2
	8E	EB	5A	79	5A	79	8E	EB	67	F0	55	7A	89	1D	68	E5
	8B	83	B6	6A	6A	8B	83	B6	F5	E4	13	29	58	EB	67	07
r8	9D	71	63	7E	9D	71	63	7E	8E	2E	08	7F	C5	26	64	B2
	97	EC	DC	89	EC	DC	89	97	54	C8	71	39	49	5B	92	7B
	A7	A4	45	D9	45	D9	A7	A4	AF	BA	9F	2E	70	88	90	BE
	6A	E9	85	C5	C5	6A	E9	85	84	42	42	A0	1B	D2	A6	6A
r9	A6	F7	43	37	A6	F7	43	37	7E	25	01	C3	02	51	19	16
	3B	39	4F	21	39	4F	21	3B	76	2F	47	32	0B	C1	4A	7D
	51	C4	60	AE	60	AE	51	C4	59	15	04	F3	F2	8C	92	F5
	AF	B5	24	02	02	AF	B5	24	AC	A6	C4	EE	8E	14	92	72

Setelah dilakukan proses enkripsi tahap kesembilan dan dihasilkan *state*(9) maka dilanjutkan proses enkripsi tahap terakhir (*round* 10) :

- *SubBytes* : merupakan proses substitusi setiap *byte* pada *state* dengan aturan 4 bit paling kiri sebagai baris dan 4 bit paling kanan sebagai kolom terhadap tabel S-Box. Tabel S-Box dengan hasil substitusi dapat dilihat pada Gambar 3.17 Hasil proses ini menghasilkan blok data *SubBytes*(10). Berikut proses substitusi *state*(9) terhadap tabel S-Box :

  - 02 = 00000010 = 0000 & 0010 = 0 & 1
    disubtitusikan dengan tabel S-Box didapatkan hasil 77
  - 0B = 00001011 = 0000 & 1011 = 0 & B
    disubtitusikan dengan tabel S-Box didapatkan hasil 2B
  - F2 = 11110010 = 1111 & 0010 = F & 2
    disubtitusikan dengan tabel S-Box didapatkan hasil 89

- 8E = 10001110 = 1000 & 1110 = 8 & E
  disubtitusikan dengan tabel S-Box didapatkan hasil 19
- 51 = 01010001 = 0101 & 0001 = 5 & 1
  disubtitusikan dengan tabel S-Box didapatkan hasil D1
- C1 = 11000001 = 1100 & 0001 = C & 1
  disubtitusikan dengan tabel S-Box didapatkan hasil 78
- 8C = 10001100 = 1000 & 1100 = 8 & C
  disubtitusikan dengan tabel S-Box didapatkan hasil 64
- 14 = 00010100 = 0001 & 0100 = 1 & 4
  disubtitusikan dengan tabel S-Box didapatkan hasil FA
- 19 = 00011001 = 0001 & 1001 = 1 & 9
  disubtitusikan dengan tabel S-Box didapatkan hasil D4
- 4A = 01001010 = 0100 & 1010 = 4 & A
  disubtitusikan dengan tabel S-Box didapatkan hasil D6
- 92 = 10010010 = 1001 & 0010 = 9 & 2
  disubtitusikan dengan tabel S-Box didapatkan hasil 4F
- 92 = 10010010 = 1001 & 0010 = 9 & 2
  disubtitusikan dengan tabel S-Box didapatkan hasil 4F
- 16 = 00010110 = 0001 & 0110 = 1 & 6
  disubtitusikan dengan tabel S-Box didapatkan hasil 47
- 7D = 01111101 = 0111 & 1101 = 7 & D
  disubtitusikan dengan tabel S-Box didapatkan hasil FF
- F5 = 11110101 = 1111 & 0101 = F & 5
  disubtitusikan dengan tabel S-Box didapatkan hasil E6
- 72 = 010000110 = 0100 & 0110 = 4 & 6
  disubtitusikan dengan tabel S-Box didapatkan hasil 40

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7C	**77**	7B	F2	6B	6F	C5	30	01	67	**2B**	FE	D7	AB	76
1	CA	82	C9	7D	**FA**	59	**47**	F0	AD	**D4**	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	**D6**	B3	29	E3	2F	84
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

Gambar 3.17 Tabel S-Box yang Menunjukkan Hasil *SubBytes*(11).

Setelah dilakukan proses substitusi didapatkan hasil *SubBytes*(10) sama dengan 77 2B 89 19 D1 78 64 FA D4 D6 4F 4F 47 FF E6 40. Hasil *SubBytes*(10) dalam bentuk blok data dapat dilihat pada Gambar 3.18.

02	51	19	16
0B	C1	4A	7D
F2	8C	92	F5
8E	14	92	72

state (10)

77	D1	D4	47
2B	78	D6	FF
89	64	4F	E6
19	FA	4F	40

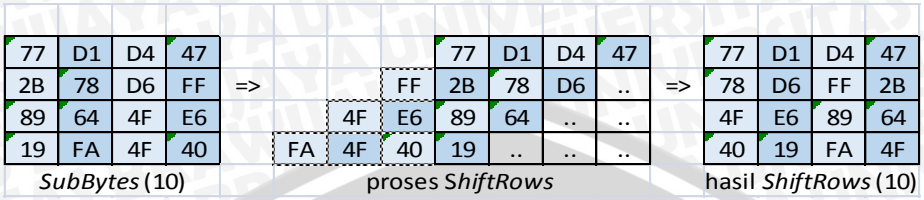
hasil *SubBytes* (11)

Gambar 3.18 Bentuk blok data *SubBytes*(10) setelah proses substitusi.

- *ShiftRows* : proses pergesaran *byte* dilakukan dengan aturan kolom *byte* digeser ke kiri setiap baris sebanyak baris n-1. Proses pergeseran *byte* diilustrasikan pada Gambar 3.19. Berikut proses pergeseran yang dilakukan terhadap hasil *SubBytes*:

  - Baris ke-1 77 D1 D4 47 dilakukan pergeseran baris sebanyak 0 kali menghasilkan 77 D1 D4 47
  - Baris ke-2 2B 78 D6 FF dilakukan pergeseran baris sebanyak 1 kali menghasilkan 78 D6 FF 2B
  - Baris ke-3 89 64 4F E6 dilakukan pergeseran baris sebanyak 2 kali menghasilkan 4F E6 89 64

- Baris ke-4 19 FA 4F 40 dilakukan pergeseran baris sebanyak 3 kali menghasilkan 40 19 FA 4F 40



Gambar 3.19 Proses Geser Kolom Tiap Baris Sebanyak Baris n-1

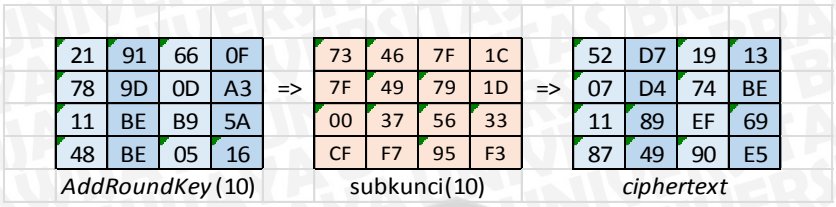
- *AddRoundKey*(10) : dengan melakukan fungsi xor pada hasil *ShiftRows*(10) dengan subkunci(10) dihasilkan *state*(10). Berikut perhitungan proses *AddRoundKey*(10) hingga dihasilkan blok data *state*(2) :

- 77 xor CE = 00100001 xor 01110011 = 01010010 = B9
- 78 xor 12 = 01111000 xor 01111111 = 00000111 = 6A
- 4F xor 75 = 00010001 xor 00000000 = 00010001 = 3A
- 40 xor 21 = 01001000 xor 11001111 = 10000111 = 61
- D1 xor BA = 1010001 xor 01000110 = 11010111 = 6B
- D6 xor FC = 10011101 xor 01001001 = 11010100 = 2A
- E6 xor EC = 10111110 xor 00110111 = 10001001 = 0A
- 19 xor 93 = 10111110 xor 11110111 = 01001001 = 8A
- D4 xor A2 = 01100110 xor 01111111 = 00011001 = 76
- FF xor F1 = 00001101 xor 01111001 = 01110100 = 0E
- 89 xor 7A = 10111001 xor 01010110 = 11101111 = F3
- FA xor C5 = 00000101 xor 10010101 = 10010000 = 3F
- 47 xor 77 = 00001111 xor 00011100 = 00010011 = 30
- 2B xor BE = 10100011 xor 00011101 = 10111110 = 95
- 64 xor 7C = 01011010 xor 00110011 = 01101001 = 18
- 4F xor 59 = 00010110 xor 11110011 = 11100101 = 16

Dari perhitungan diatas, dihasilkan *state*(10) :

B9 6A 3A 61 6B 2A 0A 8A 76 0E F3 3F 30 95 18 16.

state(10) merupakan *ciphertext* yaitu hasil akhir dari perhitungan manual proses enkripsi. *Ciphertext* dalam bentuk blok data dapat dilihat pada Gambar 3.20



Gambar 3.20 Hasil Akhir Dari Tahap Enkripsi Kesepuluh Berupa *Ciphertext.*

3.3.1.3 Penyisipan

Proses *Penyisipan* bertujuan untuk menyisipkan setiap bit *ciphertext* ke dalam MP3. Sebelum disisipkan, *ciphertext* diubah ke dalam bentuk biner seperti pada Tabel 3.2.

Tabel 3.2 Representasi Biner dari *Ciphertext*

Ciphertext	Nilai Biner
B9	10111001
6A	01101010
3A	00111010
61	01100001
6B	01101011
2A	00101010
0A	00001010
8A	10001010
76	01110110
0E	00001110
F3	11110011
3F	00111111
30	00110000
95	10010101
15	00010101
16	00010110

Setelah diubah ke dalam bentuk biner, jumlah bit *ciphertext* yang dihasilkan adalah sebanyak 128 bit. Selanjutnya, representasi biner *ciphertext* disisipkan kedalam berkas MP3 dengan kapasitas 132 byte. Berikut ini adalah representasi biner dari berkas MP3 yang akan disisipkan *ciphertext* :

10001101 00001000 01000111 01100101 01010000 10110110
10010011 10011010 01111011 01110010 01110111 00110111
11101010 00001010 11111010 00111101 11100110 11100001
01011110 01011001 01100101 10000001 01010110 11100010
01101101 01100001 11101101 11000001 10101111 01010001
00101000 01101100 11110100 11000100 00100001 01011000
01111101 11010011 01011011 11000100 00100001 10110101
01010110 11000111 00111101 01101100 10011010 01010011
01010110 11000111 00111101 01101100 10011010 01010011
10001101 00001000 01000111 01100101 01010000 10110110
10010011 10011010 01111011 01110010 01110111 00110111
11101010 00001010 11111010 00111101 11100110 11100001
01011110 01011001 01100101 10000001 01010110 11100010
01101101 01100001 11101101 11000001 10101111 01010001
00101000 01101100 11110100 11000100 00100001 01011000
01111101 11010011 01011011 11000100 00100001 10110101
01010110 11000111 00111101 01101100 10011010 01010011
01011110 01011001 01100101 10000001 01010110 11100010
01101101 01100001 11101101 11000001 10101111 01010001
00101000 01101100 11110100 11000100 00100001 01011000
01111101 11010011 01011011 11000100 00100001 10110101
01010110 11000111 00111101 01101100 10011010 01010011

Proses menyisipkan *ciphertext* menggunakan metode LSB dilakukan dengan mengganti setiap bit paling kanan dari berkas MP3 dengan bit *ciphertext* secara keseluruhan.

1000110**1** 0000100**1** 0100011**1** 0110010**1** 0101000**0** 1011011**0**
1001001**0** 1001101**0** 0111101**0** 0111001**1** 0111011**0** 0011011**1**
1110101**1** 0000101**0** 1111101**1** 0011110**0** 1110011**1** 1110000**0**
0101111**1** 0101100**1** 0110010**0** 1000000**1** 0101011**1** 1110001**0**
0110110**1** 0110000**1** 1110110**0** 1100000**1** 1010111**1** 0101000**0**
0010100**0** 0110110**0** 1111010**0** 1100010**0** 0010000**1** 0101100**1**

01111110	11010010	01011010	11000100	00100001	10110100
01010111	11000110	00111100	01101100	10011011	01010011
01010110	11000110	00111101	01101100	10011010	01010010
10001101	00001001	01000110	01100100	01010000	10110111
10010011	10011010	01111011	01110011	01110111	00110110
11101010	00001011	11111010	00111101	11100111	11100000
01011111	01011001	01100101	10000001	01010110	11100010
01101100	01100000	11101100	11000000	10101110	01010001
00101001	01101101	11110100	11000101	00100000	01011000
01111100	11010010	01011010	11000101	00100000	10110101
01010110	11000111	00111101	01101101	10011011	01010011
01011110	01011001	01100100	10000001	01010110	11100010
01101101	01100000	11101101	11000000	10101110	01010001
00101001	01101100	11110100	11000100	00100001	01011000
01111101	11010011	01011010	11000101	00100000	10110101
01010110	11000110	00111101	01101101	10011011	01010011

3.3.2 Pengungkapan Pesan

Proses pengungkapan pesan pesan meliputi proses pembangkitan subkunci, proses pengungkapan *ciphertext* dari berkas MP3 dan proses dekripsi *ciphertext* menjadi sebuah *plaintext*.

3.3.2.1 Retrieving

Pada proses *retrieving*, dilakukan pengungkapan pesan dari MP3 yang sudah disisipi pesan atau *stego* MP3. *Stego* MP3 dikodekan ke dalam bentuk biner kemudian dilakukan ekstraksi bit paling kanan dari nilai stego MP3. Berikut representasi biner dari stego MP3 :

10001101	00001001	01000111	01100101	01010000	10110110
10010010	10011010	01111010	01110011	01110110	00110111
11101011	00001010	11111011	00111100	11100111	11100000
01011111	01011001	01100100	10000001	01010111	11100010
01101101	01100001	11101100	11000001	10101111	01010000
00101000	01101100	11110100	11000100	00100001	01011001

01111100 11010010 01011010 11000100 00100001 10110101
01010111 11000110 00111100 01101100 10011011 01010010
01010110 11000110 00111101 01101100 10011010 01010010
10001101 00001001 01000110 01100100 01010000 10110110
10010011 10011010 01111011 01110011 01110111 00110110
11101010 00001011 11111010 00111101 11100111 11100000
01011111 01011001 01100101 10000001 01010110 11100010
01101100 01100000 11101100 11000000 10101110 01010001
00101001 01101101 11110100 11000101 00100000 01011000
01111100 11010010 01011010 11000101 00100000 10110101
01010110 11000111 00111101 01101101 10011011 01010011
01011110 01011001 01100100 10000001 01010110 11100010
01101101 01100000 11101101 11000000 10101110 01010001
00101001 01101100 11110100 11000100 00100001 01011000
01111101 11010011 01011010 11000101 00100000 10110101
01010110 11000110 00111101 01101101 10011011 01010011

Setiap 8 karakter bit yang diperoleh dari proses ekstrasi Stego MP3, diubah menjadi nilai heksadesimal untuk dilakukan proses dekripsi. Hasil ekstraksi dari stego MP3 diatas :

10111001 01101010 00111010 01100001 01101011 00101010 00001010 10001010 01110110 00001110 11110011 00111111 00110000 10010101 00011000 00010110

Dan bentuk heksadesimal yang dihasilkan :

B9 6A 3A 61 6B 2A 0A 8A 76 0E F3 3F 30 95 18 16.

3.3.2.2 Dekripsi

Karena *ciphertext* yang diperoleh dari hasil dekripsi sudah berupa heksadesimal, selanjutnya dilakukan pembentukan blok data *ciphertext* dengan ukuran 16 byte. Bentuk blok data *ciphertext* ditunjukan pada Gambar 3.21. Dilanjutkan dengan proses dekripsi diawali dengan *AddRoundKey*, dilanjutkan dengan *InviShiftRows*, *InvSubBytes*, *InvMixColumns* dan *AddRoundKey* untuk

round 1 hingga *Nr*-1. Untuk langkah terakhir (*Nr*) dilakukan *InvShiftRows, InvSubBytes* dan *AddRoundKey*.

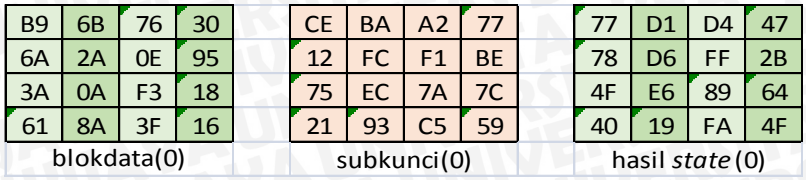
B9	6B	76	30
6A	2A	OE	95
3A	0A	F3	18
61	8A	3F	16

Gambar 3.21 Bentuk Blok Data *Ciphertext* Untuk Proses Dekripsi.

Berikut proses dekripsi tahap awal (*round* 0).

- *AddRoundKey* : dengan melakukan xor pada blok data(0) dan subkunci(0) dihasilkan *state*(0). Berikut perhitungan proses *AddRoundKey* untuk setiap byte pada *plaintext* :
  a) B9 xor CE = 10111001 xor 11001110 = 01110111 = 77
  b) 6A xor 12 = 01101010 xor 00010010 = 01111000 = 78
  c) 3A xor 75 = 00111010 xor 01110101 = 01001111 = 4F
  d) 61 xor 21 = 01100001 xor 00100001 = 01000000 = 40
  e) 6B xor BA = 01101011 xor 10111010 = 00010101 = D1
  f) 2A xor FC = 00101010 xor 11111100 = 01001000 = D6
  g) 0A xor EC = 00001010 xor 11101100 = 00000010 = E6
  h) 8A xor 93 = 10001010 xor 10010011 = 01001010 = 19
  i) 76 xor A2 = 01110110 xor 10100010 = 00011001 = D4
  j) OE xor F1 = 00001110 xor 11110001 = 01001001 = FF
  k) F3 xor 7A = 11110011 xor 01111010 = 00000000 = 89
  l) 3F xor C5 = 00111111 xor 11000101 = 00001010 = FA
  m) 30 xor 77 = 00110000 xor 01110111 = 00011010 = 47
  n) 95 xor BE = 10010101 xor 10111110 = 00000101 = 2B
  o) 18 xor 7C = 00011000 xor 01111100 = 01000101 = 64
  p) 16 xor 59 = 00010110 xor 01011001 = 01000110 = 4F

Dari perhitungan diatas didapatkan hasil *state*(0) sama dengan 77 78 4F 40 D1 D6 E6 19 D4 FF 89 FA 47 2B 64 4F. Hasil *state*(0) dalam bentuk blok data dapat dilihat pada Gambar 3.22.



Gambar 3.22 Operasi *AddRoundKey* (*round* 0) Menghasilkan *state*(0).

Dilanjutkan dekripsi tahap kesatu (*round* 1).

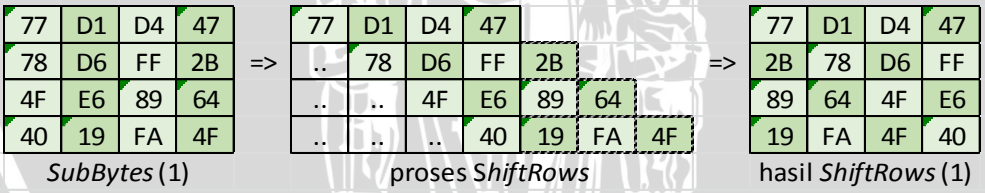
- *InvShiftRows* : dalam dekripsi, proses pergesaran *byte* dilakukan dengan aturan kolom *byte* digeser ke kanan setiap baris sebanyak baris n-1. proses pergeseran *byte* diilustrasikan pada Gambar 3.23. berikut proses pergeseran yang dilakukan :

– Baris ke-1 77 D1 D4 47 dilakukan pergeseran baris sebanyak 0 kali menghasilkan 77 D1 D4 47

– Baris ke-2 78 D6 FF 2B dilakukan pergeseran baris sebanyak 1 kali menghasilkan 2B 78 D6 FF

– Baris ke-3 4F E6 89 64 dilakukan pergeseran baris sebanyak 2 kali menghasilkan 89 64 4F E6

– Baris ke-4 40 19 FA 4F dilakukan pergeseran baris sebanyak 3 kali menghasilkan 19 FA 4F 40



Gambar 3.23 Proses geser kolom tiap baris sebanyak baris n-1.

- *InvSubBytes :* merupakan proses substitusi setiap *byte* pada *state* dengan aturan 4 bit paling kiri sebagai baris dan 4 bit paling kanan sebagai kolom terhadap tabel InvS-Box. Tabel InvS-Box dengan hasil substitusi dapat dilihat pada Gambar 3.24. Proses ini menghasilkan blok data *InvSubBytes*(1). Berikut proses substitusi *state*(0) terhadap tabel InvS-Box :

– 77 = 01110111 = 0111 & 0111 = 7 & 7
disubtitusikan dengan tabel InvS-Box didapatkan hasil 02

– 2B = 00100001 = 0010 & 1011 = 2 & B

disubtitusikan dengan tabel InvS-Box didapatkan hasil 0B

- 89 = 01011010 = 1000 & 1001 = 8 & 9
  disubtitusikan dengan tabel InvS-Box didapatkan hasil F2
- 19 = 01010010 = 0001 & 1001 = 1 & 9
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 8E
- D1 = 11010001 = 1101 & 0001 = D & 1
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 51
- 78 = 01111000 = 0111 & 1000 = 7 & 8
  disubtitusikan dengan tabel InvS-Box didapatkan hasil C1
- 64 = 01100100 = 0110 & 0100 = 6 & 4
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 8C
- FA = 11111010 = 1111 & 1010 = F & A
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 14
- D4 = 11010100 = 1101 & 0100 = D & 4
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 19
- D6 = 11010110 = 1101 & 0110 = D & 6
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 4A
- 4F = 01001111 = 0100 & 1111 = 4 & F
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 92
- 4F = 01001111 = 0100 & 1111 = 4 &F
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 92
- 47 = 01000111 = 0100 & 0111 = 4 & 7
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 16
- FF = 00000101 = 1111 & 1111 = F & F
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 7D
- E6 = 11100110 = 1110 & 0110 = E & 6
  disubtitusikan dengan tabel InvS-Box didapatkan hasil F5
- 40 = 01000000 = 0100 & 0000 = 4 & 0
  disubtitusikan dengan tabel InvS-Box didapatkan hasil 72

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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**

Gambar 3.24 Tabel *InvSubBytes* dengan hasil substitusi state(1).

Dengan dilakukan proses *InvShiftRows*, *InvSubBytes*, *InvMixColumn* dan *AddRoundKey* pada proses dekripsi tahap kedua hingga kesembilan (*Nr-1*) dan tahap terakhir dengan tanpa *InvMixColumn* didapatkan hasil berupa *plaintext*. Tabel 3.3 menunjukkan proses iterasi yang dilakukan dan hasil akhir dari proses dekripsi.

Tabel 3.3 Tabel Proses Dekripsi dan hasil akhirnya.

	***SubBytes* (r)**				***ShiftRow* (r)**				***MixColumn* (r)**				***State* (r)**			
r0													77	D1	D4	47
													78	D6	FF	2B
													4F	E6	89	64
													40	19	FA	4F
r1	77	D1	D4	47	02	51	19	16	7E	25	01	C3	A6	F7	43	37
	2B	78	D6	FF	0B	C1	4A	7D	76	2F	47	32	39	4F	21	3B
	89	64	4F	E6	F2	8C	92	F5	59	15	04	F3	60	AE	51	C4
	19	FA	4F	40	8E	14	92	72	AC	A6	C4	EE	02	AF	B5	24
r2	A6	F7	43	37	C5	26	64	B2	8E	2E	08	7F	9D	71	63	7E
	3B	39	4F	21	49	5B	92	7B	54	C8	71	39	EC	DC	89	97
	51	C4	60	AE	70	88	90	BE	AF	BA	9F	2E	45	D9	A7	A4
	AF	B5	24	02	1B	D2	A6	6A	84	42	42	A0	C5	6A	E9	85
r3	9D	71	63	7E	75	2C	00	8A	8C	6F	64	2B	0B	0A	26	EC
	97	EC	DC	89	85	83	93	F2	43	0D	E3	53	66	8E	EA	9A
	A7	A4	45	D9	89	1D	68	E5	67	F0	55	7A	5A	79	8E	EB
	6A	E9	85	C5	58	EB	67	07	F5	E4	13	29	6A	8B	83	B6
r4	0B	0A	26	EC	9E	A3	23	83	19	19	04	46	F2	06	E6	09
	9A	66	8E	EA	37	D3	E6	BB	CB	9B	18	6A	D7	AB	D3	3F
	8E	EB	5A	79	E6	3C	46	AF	B6	3F	96	0D	04	9C	7F	04
	8B	83	B6	6A	CE	41	79	58	C5	E3	02	02	80	6F	C2	11

	SubBytes (r)				*ShiftRow* (r)				*MixColumn* (r)				*State* (r)			
r5	F2	06	E6	09	04	A5	F5	40	B6	98	68	A2	82	76	88	18
	3F	D7	AB	D3	25	0D	0E	A9	99	B9	B8	86	A6	AD	71	E0
	7F	04	04	9C	6B	30	30	1C	C6	63	E3	6E	0A	0B	11	76
	6F	C2	11	80	06	A8	E3	3A	95	01	3A	1B	52	93	E1	DF
r6	82	76	88	18	11	0F	97	34	5D	80	37	4B	0F	BE	48	B3
	E0	A6	AD	71	A0	C5	18	2C	2E	CD	1A	B5	A4	92	3F	7A
	11	76	0A	0B	E3	0F	A3	9E	0F	F1	23	3F	63	03	65	25
	93	E1	DF	52	22	E0	EF	48	63	DA	9F	B0	D7	49	83	9D
r7	0F	BE	48	B3	FB	5A	D4	4B	AB	99	FB	94	2B	4B	76	EE
	7A	A4	92	3F	BD	1D	74	25	CE	9B	7E	BE	2E	56	DF	73
	65	25	63	03	BC	C2	00	D5	94	D0	7E	F4	1B	C9	60	38
	49	83	9D	D7	A4	41	75	0D	7B	3A	3F	85	94	3C	0D	FE
r8	2B	4B	76	EE	0B	CC	0F	99	DE	5F	E3	69	00	0E	D4	7D
	73	2E	56	DF	8F	C3	B9	EF	33	36	35	7E	48	92	AE	C5
	60	38	1B	C9	90	76	44	12	9D	4C	28	4D	DF	F9	82	A7
	3C	0D	FE	94	6D	F3	0C	E7	3E	57	3D	25	D9	17	3B	60
r9	00	0E	D4	7D	52	D7	19	13	21	91	66	0F	53	59	D4	A2
	C5	48	92	AE	07	D4	74	BE	78	9D	0D	A3	6A	3B	6B	09
	82	A7	DF	F9	11	89	EF	69	11	BE	B9	5A	63	6E	BE	2C
	17	3B	60	D9	87	49	90	E5	48	BE	05	16	5A	00	D6	67
r10	53	59	D4	A2	50	15	19	1A					**61**	**20**	**20**	**79**
	09	6A	3B	6B	40	58	49	05					**72**	**6E**	**79**	**61**
	BE	2C	63	6E	5A	42	00	45					**69**	**75**	**61**	**20**
	00	D6	67	5A	52	4A	0A	46					**66**	**72**	**68**	**20**

Dari Tabel 3.3 didapatkan nilai *plaintext* : **61 72 69 66 20 6E 75 72 20 79 61 68 79 61 20 20** atau “arif nur yahya”.

3.3.3 MSE dan PSNR

Untuk menghitung PSNR pada *stego* MP3, terlebih dahulu dilakukan perhitungan MSE (*Mean Square Error*). MSE dihitung menggunakan persamaan 2.16 yaitu :

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1} \|I(i) - K(i)\|^2$$

dimana :

n = ukuran byte berkas MP3

I = nilai biner pada byte pada MP3 target (lihat hal. 52)

K = nilai biner pada byte pada *stego* MP3 (lihat hal. 53)

Perhitungan MSE ditunjukkan pada Tabel 3.3.

Tabel 3.4 Tabel perhitungan MSE

Byte (*i*)	MP3 target (*I*)	*Stego* MP3 (*K*)	$\|I(i) - K(i)\|^2$
1	10001101	1000110**1**	0
2	10010011	1001001**0**	1
3	11101010	1110101**1**	1
4	01011110	0101111**1**	1
5	01101101	0110110**1**	0
6	00101000	0010100**0**	0
7	01111101	0111110**0**	1
8	01010110	0101011**1**	1
9	01010110	0101011**0**	0
10	10001101	1000110**1**	0
11	10010011	1001001**1**	0
12	11101010	1110101**0**	0
13	01011110	0101111**1**	1
14	01101101	0110110**0**	1
15	00101000	0010100**1**	1
16	01111101	0111110**0**	1
17	01010110	0101011**0**	0
18	01011110	0101111**0**	0
19	01101101	0110110**1**	0
20	00101000	0010100**1**	1
21	01111101	0111110**1**	0
22	01010110	0101011**0**	0
23	00001000	0000100**1**	1
24	10011010	1001101**0**	0
25	00001010	0000101**0**	0
26	01011001	0101100**1**	0
27	01101100	0110000**1**	1
28	11010011	0110110**0**	1

Byte (i)	MP3 target (I)	*Stego* MP3 (K)	$\|I(i) - K(i)\|^2$
29	11000111	1101001**0**	1
30	11000111	1100011**0**	1
$\sum_{i=1}^{n} \|I(i) - K(i)\|^2$			15

Pada tabel diatas hanya digunakan 30 byte pertama sebagai contoh perhitungan MSE dan PSNR. Berikut proses perhitungan yang dilakukan :

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1} \|I(i) - K(i)\|^2$$

$$= \frac{1}{30} \times 15$$

$$= 0.5$$

Dari perhitungan di atas, didapatkan nilai MSE dari MP3 target dan *stego* MP3 yaitu 0.5.

Selanjutnya dilakukan perhitungan nilai PSNR menggunakan persamaan 2.6.

$$PSNR = 20 \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

MAX_I merupakan *bits per sample* pada sinyal berkas MP3. MAX_I dihitung menggunakan persamaan 2.5. Oleh karena jumlah bit pada setiap byte berkas MP3 bernilai 8, maka :

$$MAX_I = 2^I - 1$$

$$= 2^8 - 1$$

$$= 255$$

$$PSNR = 20 \log_{10}\left(\frac{255}{\sqrt{0.5}}\right)$$

$$= 20 \log_{10}\left(\frac{255}{\sqrt{0.5}}\right)$$

$$= 20 \log_{10}(360.69)$$

$$= 20 * 2.56$$

$$= 51.14 \text{ dB}$$

Dari perhitungan di atas, didapatkan nilai PSNR pada *stego* MP3 yaitu 51.14 dB dengan nilai MSE sebesar 0,5. *Stego* MP3 tersebut dikatakan mempunyai kualitas yang baik karena memilki PSNR di atas 30 dB.

3.4 Perancangan Uji Coba dan Analisis

Setelah sistem dibuat, langkah selanjutnya adalah melakukan pengujian terhadap sistem. Pengujian pertama dilakukan dengan menghitung nilai MSE menggunakan persamaan 2.4 dan dilanjutkan dengan PSNR menggunakan persamaan 2.5. Nilai MSE digunakan untuk mengetahui estimasi bit pesan yang berubah pada *stego* MP3 dengan membandingkannya dengan MP3 asli. Sedangkan nilai PSNR digunakan untuk mengetahui kualitas berkas MP3 yang telah disisipi pesan atau *stego* MP3.

Nilai MSE dikatakan baik apabila nilai yang dihasilkan mendekati nol. Dengan kata lain perubahan nilai bit-bit pada *stego* MP3 tidak terlalu signifikan. Sedangkan untuk nilai PSNR yang baik ditunjukkan dengan nilai PSNR terbesar. Hal ini terkait dengan kualitas audio, audio yang baik memiliki nilai PSNR minimal 30 dB. Apabila hasil yang didapat dibawah 30 dB, maka berkas MP3 tersebut memiliki kualitas yang buruk dan tidak layak didengar.

Pada pengujian ini, PNSR akan dihitung pada setiap *stego* MP3. Pada setiap *stego* MP3, akan diujicobakan dengan menyembunyikan beberapa pesan yang memiliki ukuran yang berbeda-beda. Hal ini dimaksudkan untuk mengetahui pengaruh ukuran pesan terhadap nilai PSNR pada *stego* MP3.

Dengan parameter berupa besar ukuran pesan yang disembunyikan dan besar kapasitas daya tampung MP3, diharapkan akan diketahui ukuran pesan dan MP3 yang akan menghasilkan nilai PNSR terbaik pada *stego* MP3. Tabel 3.5 menunjukkan contoh tabel pengujian nilai MSE dan PSNR berdasarkan parameter yang sudah ditentukan.

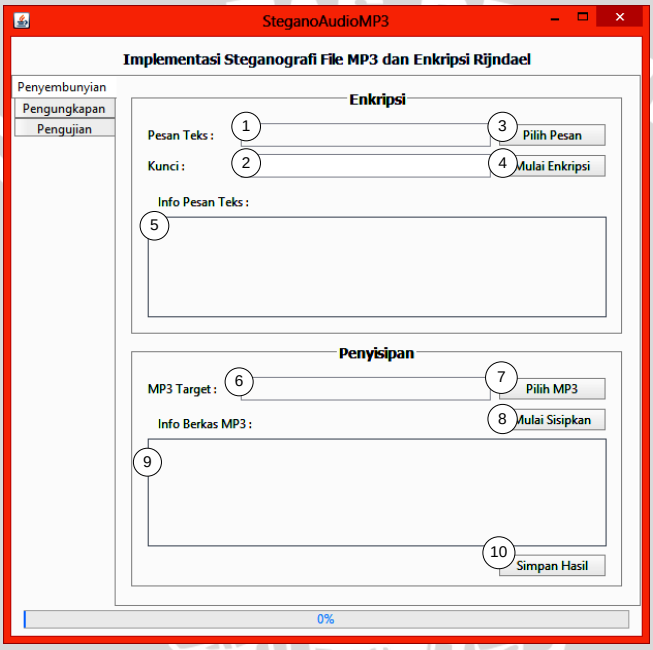
Tabel 3.5 Tabel contoh pengujian nilai MSE dan PSNR

MP3 Asli	Pesan	Rasio Pesan	MSE	PSNR

Nilai rata-rata pengujian *stego* 1				

3.5 Perancangan Antarmuka

Antarmuka pada sistem ini terdiri dari tiga bagian, yaitu bagian penyembunyian pesan, pengungkapan pesan, dan proses pengujian. antarmuka bagian penyembunyian pesan ditunjukkan pada Gambar 3.25.



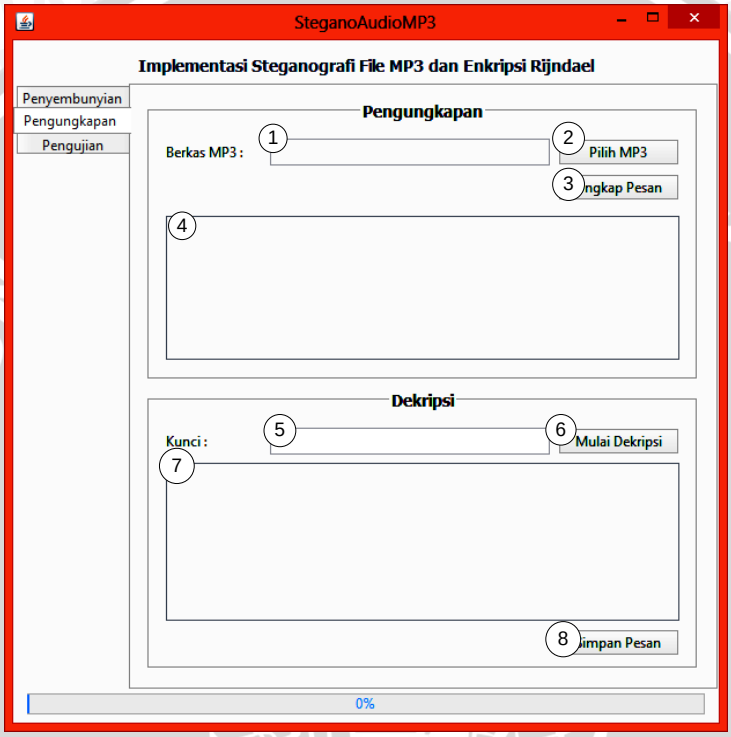
Gambar 3.25 Rancangan Antarmuka Penyembunyian Pesan

Rancangan antarmuka penyembunyian pesan pada Gambar 3.26 terdiri dari :

1. *Textfield* yang menampilkan direktori pesan yang akan disembunyikan.
2. *Textfield* yang berfungsi sebagai masukan kunci yang akan digunakan dalam proses enkripsi pesan.
3. Tombol yang berfungsi membuka jendela untuk memilih pesan.
4. Tombol yang berfungsi untuk melakukan proses enkripsi.
5. *Textarea* yang berfungsi menampilkan informasi isi pesan sebelum dienkripsi dan hasil enkripsi atau *ciphertext* beserta ukurannya.
6. *Textfield* yang menampilkan direktori MP3 target yang akan menampung pesan terenkripsi.
7. Tombol yang berfungsi membuka jendela untuk memilih berkas MP3 target.

8. Tombol yang berfungsi untuk melakukan proses penyisipan.
9. *Textarea* yang berfungsi menampilkan informasi berkas MP3 yang dipilih, proses penyisipan dan direktori hasil Stego MP3 disimpan.
10. Tombol yang berfungsi untuk melakukan proses simpan Stego *MP3*.

Untuk rancangan antarmuka bagian pengungkapan pesan ditunjukkan pada Gambar 3.26.



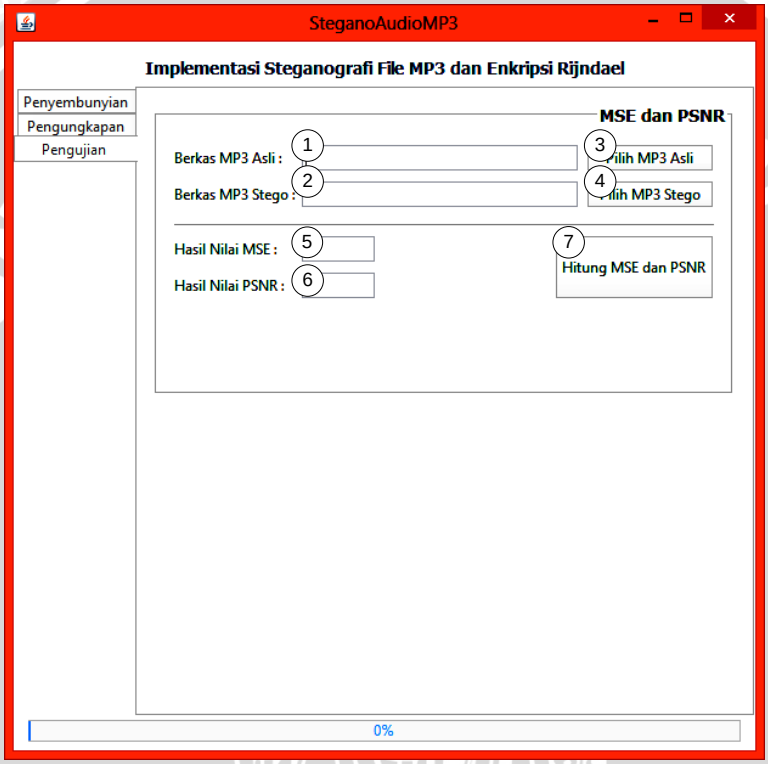
Gambar 3.26 Rancangan Antarmuka Pengungkapan Pesan

Rancangan antarmuka pengungkapan pesan pada Gambar 3.27 terdiri dari :

1. *Textfield* yang menampilkan direktori *Stego* MP3.
2. Tombol yang berfungsi membuka jendela untuk memilih *Stego* MP3.
3. Tombol yang berfungsi menjalankan perintah untuk melakukan proses mengungkap pesan.
4. *Textarea* yang berfungsi menampilkan informasi isi informasi berkas MP3 yang dipilih.
5. *Textfield* yang berfungsi sebagai masukan kunci yang akan digunakan dalam proses enkripsi pesan.

6. Tombol yang berfungsi menjalankan perintah untuk melakukan proses dekripsi *ciphertext* hasil pengungkapan.
7. *Textarea* yang berfungsi menampilkan informasi hasil dekripsi dan direktori hasil dekripsi disimpan.
8. Tombol yang berfungsi menjalankan perintah untuk melakukan proses simpan pesan teks hasil dekripsi.

Untuk rancangan antarmuka bagian pengujian ditunjukkan pada Gambar 3.27



Gambar 3.27 Rancangan Antarmuka Pengujian

Rancangan antarmuka pengujian pada Gambar 3.28 terdiri dari :

1. *Textfield* untuk menampilkan direktori berkas MP3 asli yang belum disisipi pesan atau MP3 target.
2. *Textfield* untuk menampilkan direktori berkas MP3 yang sudah disisipi pesan atau *stego* MP3.
3. Tombol yang berfungsi untuk menampilkan jendela untuk memilih berkas MP3 target .
4. Tombol yang berfungsi untuk menampilkan jendela untuk memilih *stego* MP3.

5. *Textfield* yang menampilkan nilai MSE.
6. *Textfield* yang menampilkan nilai PSNR dalam satuan desibel.
7. Tombol untuk menjalankan perintah perhitungan nilai MSE dan PSNR.

BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi dari rancangan aplikasi implementasi steganografi pada file MP3 dengan metode *least significant bit* dan enkripsi *rijndael* yang telah dibahas di Bab III, terdiri dari lingkungan perangkat keras dan lingkungan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan saat proses pengembangan dan pengujian aplikasi ini adalah sebagai berikut :

1. *Processor* Intel® Core™ i3-4130 CPU @ 3.40 GHz
2. *Memory* 4 GB DDR3
3. *System type* 64-bit OS

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang diterapkan dalam pengembangan aplikasi implementasi steganografi pada file MP3 dengan metode *least significant bit* dan enkripsi *rijndael* adalah :

1. Sistem operasi Windows 8 64-bit.
2. NetBeans IDE 8.0.2.

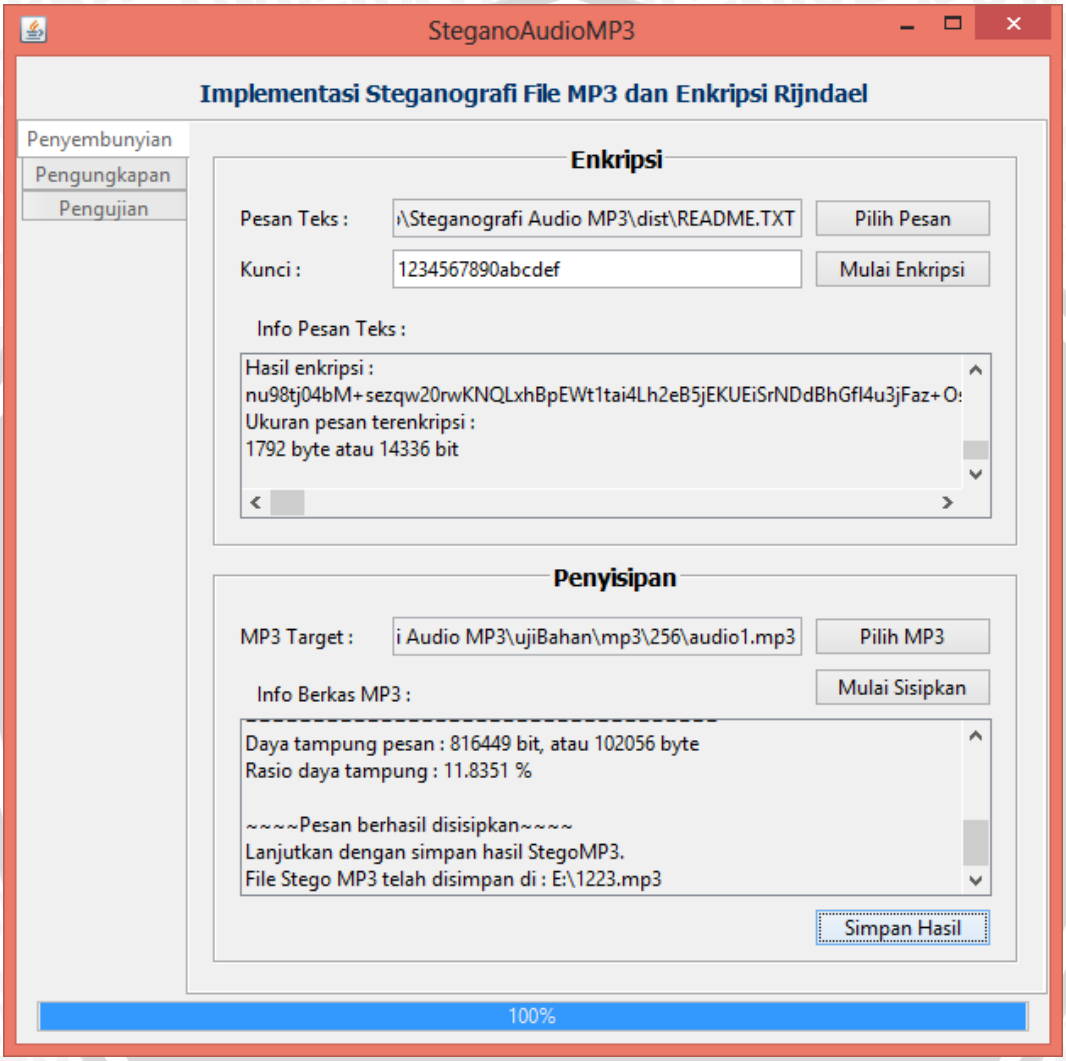
4.2 Implementasi Program

Pada subbab ini dijelaskan implementasi program dalam pembuatan aplikasi implementasi steganografi pada file MP3 dengan metode *least significant bit* dan enkripsi *rijndael* berdasarkan perancangan sistem yang telah dijelaskan pada Bab III. Implementasi pembuatan aplikasi dibuat dengan menggunakan bahasa pemrograman JAVA. Terdapat tiga proses utama dalam implementasi pembuatan aplikasi yaitu penyembunyian pesan, pengungkapan pesan, dan proses pengujian.

4.2.1 Proses Penyembunyian Pesan

Proses penyembunyian pesan merupakan serangkaian proses bertujuan agar pesan berhasil disembunyikan ke dalam berkas MP3. Proses penyembunyian pesan terdiri dari dua langkah utama yaitu proses enkripsi pesan hingga

menghasilkan *ciphertext* dan proses *hiding* dari hasil enkripsi tersebut ke dalam berkas MP3. Selain itu terdapat dua proses pembacaan berkas dan satu proses penyimpanan berkas Antarmuka penyembunyian pesan ditunjukkan pada Gambar 4.1.



Gambar 4.1 Antarmuka Penyembunyian Pesan

4.2.1.1 Implementasi Pembacaan Pesan

Penyembunyian pesan ke dalam MP3 diawali dengan pembacaan pesan berupa berkas bertipe teks (*.txt*). Proses pembacaan pesan ini dilakukan perbaris dan sebagai penanda diakhir baris akan ditambahkan karakter '\n' sebagai implementasi perintah pindah baris. Implementasi pembacaan pesan ditunjukkan pada *source code* 4.1.

```
01  private void loadPlaintext() throws IOException {
02      JFileChooser fc = new JFileChooser();
03      fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
04      fc.setMultiSelectionEnabled(false);
05      int x = fc.showOpenDialog(null);
06      if (x == JFileChooser.APPROVE_OPTION) {
07          String dir = fc.getSelectedFile().getPath();
08          File file = new File(dir);
09          Path path = Paths.get(dir);
10          dirPath = path;
11          fldPesanPath.setText(dir);
12          fldPesanPath.setEditable(false);
13       try{
14           FileReader fileReader = new FileReader(file);
15           try (BufferedReader bufferedReader = new
16  BufferedReader(fileReader)) {
17                  String text;
18                  txPesanTeks.setText("");
19                  while (
20                  null != (text = bufferedReader.readLine()))
21                  txPesanTeks.append (text + "\n") ;
22                  }
23          }
24          catch(FileNotFoundException ex){
25          }
26        }
27      }
```

***Source code* 4.1** Implementasi Pembacaan Berkas Bertipe Teks (.txt)

4.2.1.2 Implementasi Pembentukan SubKunci

Proses pembentukan Subkunci diimplementasikan seperti persamaan 2.2 dan 2.3 menggunakan masukan sebuah kunci dengan panjang 16, 24 atau 32 karakter.

```
01  private static byte[][] generateSubkeys(byte[] key) {
02     byte[][] tmp = new byte[Nb * (Nr + 1)][4];
03     int i = 0;
04     while (i < Nk) {
05        tmp[i][0] = key[i * 4];
06        tmp[i][1] = key[i * 4 + 1];
```

```
07          tmp[i][2] = key[i * 4 + 2];
08          tmp[i][3] = key[i * 4 + 3];
09       i++;
10       }
11       i = Nk;
12       while (i < Nb * (Nr + 1)) {
13          byte[] temp = new byte[4];
14          for(int k = 0;k<4;k++)
15             temp[k] = tmp[i-1][k];
16          if (i % Nk == 0) {
17             temp = SubWord(rotateWord(temp));
18             temp[0] = (byte) (temp[0] ^ (Rcon[i / Nk] & 0xff));
19           } else if (Nk > 6 && i % Nk == 4) {
20              temp = SubWord(temp);
21           }
22           tmp[i] = fungsiXOR(tmp[i - Nk], temp);
23           i++;
24       }
25       return tmp;
26    }
27    private static byte[] SubWord(byte[] in) {
28       byte[] tmp = new byte[in.length];
29       for (int i = 0; i < tmp.length; i++)
30          tmp[i] = (byte) (sbox[in[i] & 0x000000ff] & 0xff);
31    return tmp;
32    }
33    private static byte[] rotateWord(byte[] input) {
34       byte[] tmp = new byte[input.length];
35       tmp[0] = input[1];
36       tmp[1] = input[2];
37       tmp[2] = input[3];
38       tmp[3] = input[0];
39    return tmp;
40    }
```

***Source code* 4.2** Implementasi Pembentukan Subkunci.

4.2.1.3 Implementasi Enkripsi

Langkah selanjutnya yang dilakukan adalah melakukan enkripsi pada pesan teks menggunakan kunci rahasia. Pesan nantinya akan dipecah menjadi blok-blok karena enkripsi dilakukan satu per satu pada setiap blok. Implementasi enkripsi ditunjukkan pada *source code* 4.3.

```
01  public static byte[] encrypt(byte[] in,byte[] key){
02     Nb = 4;
03     Nk = key.length/4;
04     Nr = Nk + 6;
05     int lenght=0;
06     byte[] padding = new byte[1];
07     int i;
08     lenght = 16 - in.length % 16;
09     padding = new byte[lenght];
10     padding[0] = (byte) 0x80;
11     for (i = 1; i < lenght; i++)
12         padding[i] = 0;
13     byte[] tmp = new byte[in.length + lenght];
14     byte[] bloc = new byte[16];
15     w = generateSubkeys(key);
16    int count = 0;
17    for (i = 0; i < in.length + lenght; i++) {
18       if (i > 0 && i % 16 == 0) {
19           bloc = encryptBloc(bloc);
20           System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
21       }
22       if (i < in.length)
23           bloc[i % 16] = in[i];
24       else{
25           bloc[i % 16] = padding[count % 16];
26           count++;
27       }
28    }
29    if(bloc.length == 16){
30       bloc = encryptBloc(bloc);
31       System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
32    }
33    return tmp;
34  }
35  public static byte[] encryptBloc(byte[] in) {
36     byte[] tmp = new byte[in.length];
37     byte[][] state = new byte[4][Nb];
38     for (int i = 0; i < in.length; i++)
39         state[i / 4][i % 4] = in[i%4*4+i/4];
40     state = AddRoundKey(state, w, 0);
41     for (int round = 1; round < Nr; round++) {
42         state = SubBytes(state);
```

```
43          state = ShiftRows(state);
44          state = MixColumns(state);
45          state = AddRoundKey(state, w, round);
46      }
47      state = SubBytes(state);
48      state = ShiftRows(state);
49      state = AddRoundKey(state, w, Nr);
50      for (int i = 0; i < tmp.length; i++)
51          tmp[i%4*4+i/4] = state[i / 4][i%4];
52      return tmp;
53  }
```

***Source code* 4.3** Implementasi Enkripsi

Fungsi *encrypt* digunakan untuk inisiasi fungsi selanjutnya yaitu *encryptBloc*. Dalam fungsi *encrypt*, dilakukan inisiasi ukuran blok data, panjang kunci dan jumlah *round* dalam proses enkripsi. Proses pembentukan subkunci juga dipanggil dalam fungsi ini pada baris ke-17. Implementasi pembentukan subkunci dapat dilihat pada *source code* 4.6.

Baris ke-19 sampai dengan 36 dalam fungsi e*ncrypt* merupakan implementasi pembentukan blok data untuk enkripsi dan penambahan padding apabila teks kurang dari jumlah blok yang dibutuhkan. Baris ke-44 dan 45 pada fungsi *encryptBloc* membentuk *byte array* 2 dimensi untuk dijadikan state dalam proses enkripsi. Baris ke-47 hingga 56 merupakan implementasi proses enkripsi yang dilakukan pada setiap blok.

Dalam proses enkripsi yang dilakukan pada setiap blok, dilakukan pemanggilan empat fungsi yaitu *AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns*. Implementasi ke empat proses tersebut ditunjukkan pada *source code* 4.4.

```
01  private static byte[][] AddRoundKey(byte[][] state, byte[][] w,
02  int round) {
03     byte[][] tmp = new byte[state.length][state[0].length];
04       for (int c = 0; c < Nb; c++) {
05           for (int l = 0; l < 4; l++)
06               tmp[l][c] = (byte) (state[l][c] ^ w[round * Nb +
07  c][l]);
08       }
```

```
09      return tmp;
10  }
11  private static byte[][] SubBytes(byte[][] state) {
12      byte[][] tmp = new byte[state.length][state[0].length];
13          for (int row = 0; row < 4; row++)
14              for (int col = 0; col < Nb; col++)
15                  tmp[row][col] = (byte) (sbox[(state[row][col] &
16  0x000000ff)] & 0xff);
17      return tmp;
18  }
19  private static byte[][] ShiftRows(byte[][] state) {
20     byte[] t = new byte[4];
21     for (int r = 1; r < 4; r++) {
22         for (int c = 0; c < Nb; c++)
23            t[c] = state[r][(c + r) % Nb];
24         for (int c = 0; c < Nb; c++)
25             state[r][c] = t[c];
26       }
27      return state;
28  }
29  private static byte[][] MixColumns(byte[][] s){
30          int[] sp = new int[4];
31          byte b02 = (byte)0x02, b03 = (byte)0x03;
32          for (int c = 0; c < 4; c++) {
33              sp[0] = FFMulFastFast(b02, s[0][c]) ^
34  FFMulFastFast(b03, s[1][c]) ^ s[2][c]  ^ s[3][c];
35              sp[1] = s[0][c]  ^ FFMulFast(b02, s[1][c]) ^
36  FFMulFast(b03, s[2][c]) ^ s[3][c];
37              sp[2] = s[0][c]  ^ s[1][c]  ^ FFMulFast(b02, s[2][c]) ^
38  FFMulFast(b03, s[3][c]);
39              sp[3] = FFMulFast(b03, s[0][c]) ^ s[1][c]  ^ s[2][c]  ^
40  FFMulFast(b02, s[3][c]);
41              for (int i = 0; i < 4; i++) s[i][c] = (byte)(sp[i]);
42          }
43          return s;
44      }
```

Source code 4.4 Implementasi *AddRoundKey*, *SubBytes*, *ShiftRows*, dan *MixColumns*

4.2.1.4 Implementasi *Hiding*

Hiding merupakan proses penyembunyian bit pesan ke dalam bit audio MP3. Bit audio dibagi menjadi beberapa region sebanyak bit pesan. Penyembunyian pesan dilakukan dengan mengganti nilai *parity* bit dari setiap region dengan bit pesan. Implementasi *hiding* ditunjukkan pada *source code* 4.5.

```
01      public String penyisipan(StringBuilder bin, StringBuilder
02  sb){
03          String bitBaru;
04          int sisaBit, kapasitas;
05          int index1, index2, index3 = 0;
06          String                          normal                  =
07  sb.toString().replace("11111110/11111010/", "1111111x/1111101x/")
08                  .replace("11111110/11111011/",
09  "1111111x/1111101y/")
10                  .replace("11111111/11111010/",
11  "1111111y/1111101x/");
12          sb.delete(0, sb.length());
13          sb = new StringBuilder(normal);
14          index2 = sb.indexOf("11111111/11111011/");
15          sisaBit = bin.length();
16          while(sisaBit > 0)
17          {
18             index1=index2 + 341;
19             index2 = sb.indexOf("11111111/11111011/",index1);
20             if(index2<0)index2=sb.length()-1;
21             kapasitas=((index2-index1)/9);
22             for(int i=0; i<kapasitas; i++){
23                 index1=index1+8;
24                 if(sb.charAt(index1)!='x'                        &&
25  sb.charAt(index1)!='y'){
26                     sb.setCharAt(index1, bin.charAt(index3));
27                     index3++;
28                     sisaBit--;
29                     if(sisaBit==0) break;
30                 }
31                 index1++;
32             }
33          }
34           bitBaru=sb.toString().replace("x",      "0").replace("y",
35  "1");
```

```
36              sb.delete(0, sb.length());
37              System.out.println("done");
38              return bitBaru;
39          }
```

***Source code* 4.5** Implementasi *hiding*

Baris ke-17 hingga 34 merupakan proses inti dari penyembunyian pesan dengan mengganti bit pada *main data* MP3. Perulangan pada baris ke-17 menunjukkan bahwa bit pada *main data* akan di lakukan pengubahan bit *least significant bit* sepanjang pesan yang sudah dienkripsi. Pada baris ke-14 menunjukan bahwa bit yang dapat dimodifikasi dimulai pada bit ke 286 pada sebuah *frame*. Baris Baris ke-35 dan ke-36 mengimplementasikan penggantian nilai bit yang baru.

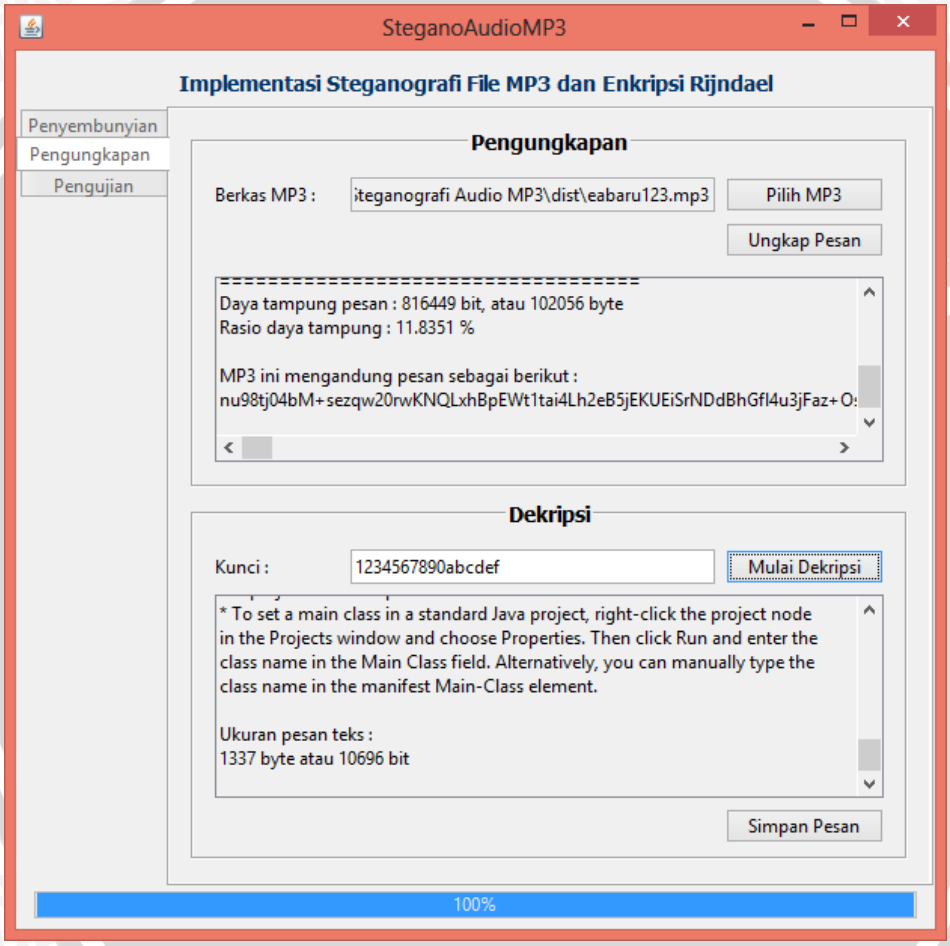
Langkah terakhir dari penyembunyian pesan ke dalam MP3 adalah pembetukan kembali berkas audio MP3 dari bit audio yang telah disisipi bit pesan. Implementasi pembentukan berkas audio MP3 ditunjukkan pada *source code* 4.6.

```
01  public void WriteAudio(final String save) {
02      Thread t = new Thread() {
03          @Override
04          public void run() {
05              int count = stego.length() / 8;
06              String[] new_byte = new String[count];
07              int index = 0;
08              for (int i = 1; i <= count; i++) {
09                  new_byte[i - 1] = stego.substring(index, i * 8);
10                  index = i * 8;}
11              try {
12                  File trashedSong = new File(save + ".mp3");
13                  FileOutputStream        trash        =        new
14  FileOutputStream(trashedSong);
15                  for (int i = 0; i < count; i++) {
16                      trash.write(Integer.parseInt(new_byte[i],
17  2));
18                      bar1.setValue((i + 1) * 100 / count);}
19                  trash.close();} catch (Exception e){
20                  System.out.println("Error â€" " + e.toString());}
21          }};t.start();}
```

***Source code* 4.6** Implementasi Pembentukan Berkas Audio MP3

4.2.2 Proses Pengungkapan Pesan

Proses pengungkapan pesan merupakan proses pengungkapan pesan dari berkas audio MP3 dan proses dekripsi terhadap pesan hasil pengungkapan tersebut. Proses pengungkapan pesan dalam MP3 terdiri dari dua langkah utama yaitu pengungkapan pesan dan dekripsi. Selain itu terdapat pula langkah pembacaan berkas yaitu pembacaan berkas audio dan langkah penyimpanan pesan hasil dekripsi. Antarmuka pengungkapan pesan ditunjukkan pada Gambar 4.2.



Gambar 4.2 Antarmuka Pengungkapan Pesan

4.2.2.1 Implementasi Pengungkapan Pesan

Langkah pengungkapan pesan merupakan langkah pengungkapan bit pesan dari dalam bit audio MP3. Pengungkapan pesan dilakukan dengan melakukan ekstraksi nilai bit *least significant* dari *main data* pada MP3. Proses ekstraksi terus dilakukan hingga berakhir dengan adanya karakter bit penanda. Dari hasil

ekstraksi nilai bit tersebut dapat disusun bit pesan. Implementasi proses pengungkapan pesan ditunjukkan pada *source code* 4.7.

```
01 public String pengungkapan(StringBuilder sb){
02    boolean akhir=true;
03    String temp = "", message;
04    int index3, index4,kapasitas, jmlData=0, desimal;
05    String pesan;
06    StringBuilder msg=new StringBuilder();
07    char ascii;
08
09    String  normal  =  sb.toString().replace("11111110/11111010/",
10 "1111111x/1111101x/")
11                .replace("11111110/11111011/",
12 "1111111x/1111101y/")
13                .replace("11111111/11111010/",
14 "1111111y/1111101x/");
15    sb.delete(0, sb.length());
16    sb = new StringBuilder(normal);
17
18    index4=sb.indexOf("11111111/11111011/");
19    while(akhir){
20    index3=index4+341;
21       index4=sb.indexOf("11111111/11111011/",index3);
22       if(index4<0)index4=sb.length()-1;
23       kapasitas=((index4-index3)/9);
24       for(int i=0; i<kapasitas; i++){
25           index3=index3+8;
26           if(sb.charAt(index3)!='x' && sb.charAt(index3)!='y'){
27              temp=temp+String.valueOf(sb.charAt(index3));
28              jmlData++;
29              if(jmlData==8){
30                  desimal=Integer.parseInt(temp,2);
31                  ascii=(char)desimal;
32                  pesan=String.valueOf(ascii);
33                  jmlData=0;
34                  msg=msg.append(pesan);
35              temp="";
36
37              if(pesan.equals("#")){
38              akhir= false;
39              break;
```

```
40                  }
41              }
42          }
43          index3++;
44          }
45      }
46      sb.delete(0, sb.length());
47      message=msg.toString().replaceAll("[#!]", "");
48      return message;
49  }
```

***Source code* 4.7** Implementasi Pengungkapan Pesan

Pesan hasil pengungkapan diperoleh dengan mengubah bit pesan menjadi ke bentuk desimal atau kode ASCII yang kemudian akan diubah ke bentuk karakter.

4.2.2.2 Implementasi Dekripsi

Langkah selanjutnya yang dilakukan adalah melakukan dekripsi pada pesan hasil pengungkapan menggunakan kunci rahasia yang harus sama antara enkripsi dan dekripsi. Pesan nantinya akan dilakukan proses dekripsi dengan menggunakan proses inverse dari enkripsi yaitu *invSubBytes, invShiftRows, invMixColumns.* Implementasi proses dekripsi ditunjukkan pada *source code* 4.8.

```
01      public static byte[] decrypt(byte[] in,byte[] key){
02          int i;
03          byte[] tmp = new byte[in.length];
04          byte[] bloc = new byte[16];
05
06          Nb = 4;
07          Nk = key.length/4;
08          Nr = Nk + 6;
09          w = generateSubkeys(key);
10
11          for (i = 0; i < in.length; i++) {
12              if (i > 0 && i % 16 == 0) {
13                  bloc = decryptBloc(bloc);
14                  System.arraycopy(bloc,   0,   tmp,   i   -   16,
15  bloc.length);
16              }
17              if (i < in.length)
18                  bloc[i % 16] = in[i];
19          }
```

```
20          bloc = decryptBloc(bloc);
21          System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
22
23          tmp = deletePadding(tmp);
24
25          return tmp;
26      }
27  public static byte[] decryptBloc(byte[] in) {
28          byte[] tmp = new byte[in.length];
29          //ubah  masukan  dalam  bentuk  array  2  dimensi  untuk
30  perhitungan
31          byte[][] state = new byte[4][Nb];
32          //for dibawah ini, proses ubah array menjadi 2 dimensi
33          for (int i = 0; i < in.length; i++)
34              state[i / 4][i % 4] = in[i%4*4+i/4];
35          // mulai dekripsi, round 1.
36          state = AddRoundKey(state, w, Nr);
37          //round 2 hingga round N-1
38          for (int round = Nr-1; round >=1; round--) {
39              state = InvSubBytes(state);
40              state = InvShiftRows(state);
41              state = AddRoundKey(state, w, round);
42              state = InvMixColumns(state);
43          }
44          //round N. round terakhir
45          state = InvSubBytes(state);
46          state = InvShiftRows(state);
47          state  =  AddRoundKey(state,  w,  0);  //hasil  perhitungan,
48  final state.
49          //ubah dari array 2 dimensi menjadi array-byte 1 dimensi
50          for (int i = 0; i < tmp.length; i++)
51              tmp[i%4*4+i/4] = state[i / 4][i%4];
52          return tmp;
53      }
```

***Source code* 4.8** Implementasi Dekripsi

Baris ke-31 sampai dengan 47 merupakan implementasi proses dekripsi yang dilakukan pada setiap blok. Langkah terakhir dari pegungkapan pesan dalam MP3 adalah pembentukan pesan dalam berkas bertipe teks (.txt). Implementasi pembentukan berkas bertipe teks (.txt) ditunjukkan pada *source code* 4.9.

```
01    public void Write(String a, StringBuilder text) {
02      String[] temp = text.toString().split("\n");
03      File fileOut = new File(a + ".txt");
04      try {
05          BufferedWriter    fOut    =    new    BufferedWriter(new
06  FileWriter(fileOut));
07          for (int i = 0; i < temp.length; i++) {
08              fOut.write(temp[i]);
09              if (i != temp.length - 1) {
10                  fOut.newLine();
11              }
12          }
13          fOut.close();
14      } catch (Exception e) {
15      }
16  }
```

***Source code* 4.9** Implementasi Pembentukan Berkas Bertipe Teks (.txt)

4.2.3 Proses Pengujian

Proses pengujian yang terdapat pada aplikasi ini merupakan pengujian PSNR pada dua berkas audio MP3. Antarmuka pengujian ditunjukkan pada Gambar 4.3.

Gambar 4.3 Antarmuka Pengujian

4.2.3.1 Implementasi PSNR

Pada pengujian PSNR ini akan dibandingkan dua berkas MP3 yaitu MP3 asli dan MP3 yang telah disisipi pesan. Dari dua berkas MP3 tersebut, dapat

dihitung nilai PSNR pada berkas MP3 yang telah disisipi pesan dalam satuan dB. Implementasi pengujian PSNR ditunjukkan pada *source code* 4.10.

```
1   private double MSE;
2   private double PSNR;
3   public PSNR(final String dir_MP3, final String dir_stego, final
4   JProgressBar bar, final JTextField outputPSNR)
5   {
6       Thread t = new Thread()
7       {
8           @Override
9           public void run()
10          {
11              try
12              {
13                  File song1 = new File(dir_MP3);
14                  File song2 = new File(dir_stego);
15                  FileInputStream file1 = new
16  FileInputStream(song1);
17                  FileInputStream file2 = new
18  FileInputStream(song2);
19                  int count1 = file1.available();
20                  int count2 = file2.available();
21                  double n;
22                  double sum = 0;
23                  double max = 1;
24                  if (count1 == count2)
25                  {
26                      n = count1 * 8;
27                      for (int i = 0; i < count1; i++)
28                      {
29                          String bitstream1 =
30  fixBinary(Integer.toBinaryString(file1.read()));
31                          String bitstream2 =
32  fixBinary(Integer.toBinaryString(file2.read()));
33                          for(int j=0; j<bitstream1.length(); j++)
34                          {
35                              String x = bitstream1.charAt(j)+"";
36                              String y = bitstream2.charAt(j)+"";
37                              if(!x.equals(y))
38                                  sum++;
39                          }
```

```
40                          bar.setValue((i + 1) * 100 / count1);
41                      }
42                      sum = sum + 0.000001;
43                      MSE = (double)1 / n * sum;
44                      PSNR = (double)20 * Math.log10(max /
45  Math.sqrt(MSE));
46                      BigDecimal b = new BigDecimal(PSNR);
47                      b = b.setScale(2, RoundingMode.HALF_EVEN);
48
49  outputPSNR.setText(Double.toString(b.doubleValue()));
50                  }
51                   else
52                  {
53                      JOptionPane.showMessageDialog(null, "Jumlah
54  byte dari kedua MP3 berbeda", "ERROR", 0);
55                  }
56              }
57              catch (Exception e)
58              {
59                  System.out.println("Error â€" " + e.toString());
60              }
61          }
62      };
63      t.start();
64  }
```

***Source code* 4.10** Implementasi Pengujian PSNR

BAB V
UJI COBA DAN ANALISIS

5.1 Implementasi Uji Coba

Implementasi uji coba terhadap aplikasi implementasi steganografi pada file mp3 dengan metode *least significant bit* dan enkripsi *rijndael*. Berdasarkan perancangan uji coba pada subbab 3.4, pengujian dilakukan untuk mengatahui kualitas berkas MP3 yang telah disisipkan pesan.

Berkas MP3 yang akan digunakan sebagai bahan uji atau *Stego* MP3 terdiri dari tiga buah berkas MP3 yang memiliki daya tampung yang berbeda. Besarnya daya tampung dihitung dari total *main data* pada setiap *frame* MP3 yang dapat disisipkan pesan terenkripsi. Daftar berkas MP3 yang digunakan ditunjukkan Tabel 5.1.

Tabel 5.1 Daftar berkas MP3

No.	Nama Berkas MP3	Ukuran Berkas	Daya Tampung
1.	*audio1.mp3*	862,316 byte	102056 byte
2.	*audio2.mp3*	1,519,965 byte	180683 byte
3.	*audio3.mp3*	2,944,588 byte	351117 byte

Sedangkan pesan yang akan disembunyikan ke dalam MP3 target terdiri dari lima berkas bertipe teks (*.txt*) yang memiliki ukuran berbeda. Perbedaan besar ukuran pesan berpengaruh secara langsung terhadap besarnya rasio pesan dalam media penyisipan. Daftar berkas teks yang akan digunakan ditunjukkan pada Tabel 5.2.

Tabel 5.2 Daftar berkas teks

No.	Nama Berkas Teks	Ukuran Pesan Asli	Ukuran Pesan Terenkripsi
1.	*Hello.txt*	792 byte	984 byte
2.	*Merdeka.txt*	4,710 byte	6,296 byte
3.	*random-10k.txt*	9,353 byte	12,480 byte
4.	*random-20k.txt*	20,191 byte	26,924 byte
5.	*permendiknas.txt*	39,392 byte	52,544 byte

5.1.1 Implementasi Pengujian Kualitas Berkas MP3

Prosedur uji coba pada pengujian kualitas berkas MP3 dilakukan dengan melakukan enkripsi pada pesan dan selanjutnya menyembunyikan *ciphertext* hasil enkripsi tersebut kedalam MP3 target sehingga didapatkan *stego* MP3. Kualitas *stego* MP3 dapat diketahui dengan menguji nilai MSE dan PSNR *stego* MP3 tersebut terhadap berkas MP3 asli.

Pesan yang akan disembunyikan adalah berkas teks pada Tabel 5.2. Pada setiap berkas teks yang berisi pesan akan dilakukan enkripsi hingga didapatkan pesan terenkripsi. Setelah dilakukan enkripsi pada semua berkas teks, maka pesan terenkripsi nantinya disembunyikan ke dalam target MP3.

Pengujian kualitas berkas MP3 terdiri dari lima skenario pengujian. Pada setiap skenario akan menggunakan satu berkas MP3 pada Tabel 5.1 sebagai media penyisipan. Pada setiap skenario dilakukan penyembunyian sebanyak lima kali sebanyak jumlah berkas teks yang ada sehingga dihasilkan lima *stego* MP3 yang masing-masing *stego* MP3 menampung tepat sebuah pesan terenkripsi dari kelima pesan tersebut. Selanjutnya dilakukan pengujian kualitas pada masing-masing *stego* MP3. Dari nilai kualitas pada masing-masing *stego* MP3, dapat dihitung rata-rata nilai kualitas pada skenario tesebut.

5.2 Hasil Pengujian Dan Pembahasan

Berdasarkan rancangan pengujian yang telah dirancang pada bab 3, maka akan dilakukan pengujian stego MP3 untuk mendapatkan hasil perbandingan kualitas MP3 setelah disisipkan pesan terenkripsi dengan ukuran berbeda.

5.2.1 Hasil Pengujian kualitas berkas MP3

Untuk pengujian kualitas berkas MP3 dilakukan dengan menghitung nilai *mean square error* (MSE) dan *peak signal to noise ratio* (PSNR). Nilai PSNR yang diukur dalam satuan desibel (dB) ini dihitung untuk mengetahui kualitas berkas MP3 yang telah disisipkan pesan atau *stego* MP3. Semakin besar nilai PSNR, maka semakin baik kualitas *stego* MP3 tersebut. Nilai MSE dan PSNR yang baik menunjukkan bahwa kualitas berkas MP3 sebagai media penampung tidak banyak berubah akibat penyembunyian pesan. Kualitas berkas MP3 yang baik memiliki nilai PSNR minimal 30 dB.

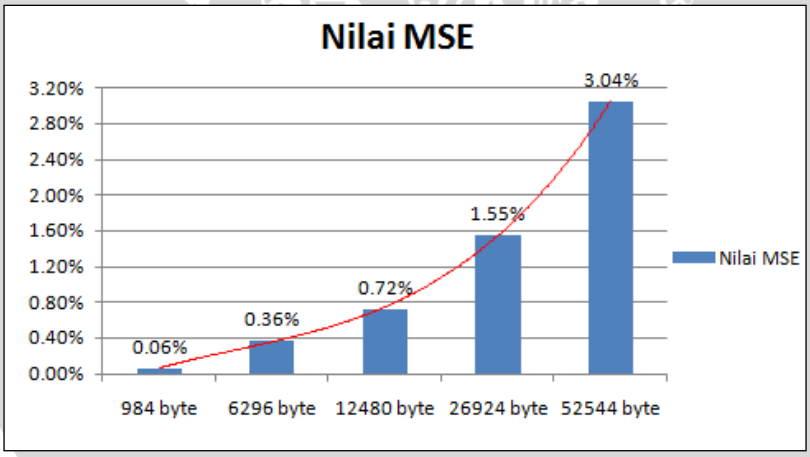
Tabel hasil pengujian dengan menggunakan berkas MP3 "*audio1.mp3*" sebagai MP3 *carrier* ditunjukkan pada Tabel 5.3.

Tabel 5.3 Hasil nilai pengujian tiap pesan pada berkas "*audio1.mp3*".

MP3 Asli	Pesan	Rasio Pesan	MSE	PSNR
audio1.mp3 (862,316 byte)	*hello.txt*	0.96 %	0.06%	80.62 dB
	merdeka.txt	6.17 %	0.36%	72.53 dB
	random-10k.txt	12.23 %	0.72%	69.55 dB
	random-20k.txt	26.38 %	1.55%	66.22 dB
	permendiknas.txt	51.49 %	3.04%	63.30 dB
Nilai rata-rata pengujian *stego* 1			**1.15%**	**70.44 dB**

Dari Tabel 5.3 dapat dilihat bahwa hasil pengujian terhadap media audio 1 dengan disisipkan beberapa pesan berkapasitas berbeda menghasilkan nilai MSE dan PSNR yang berbeda pula. Berdasarkan tabel yang sama, juga diketahui rata-rata nilai persentase MSE yaitu 1,15 % dan untuk nilai PSNR yaitu 70.44 dB.

Berdasarkan Tabel 5.3 maka dibuat grafik persentase MSE pada setiap *stego* MP3 dengan pesan yang berbeda. Persentase MSE ditunjukkan pada Gambar 5.1.

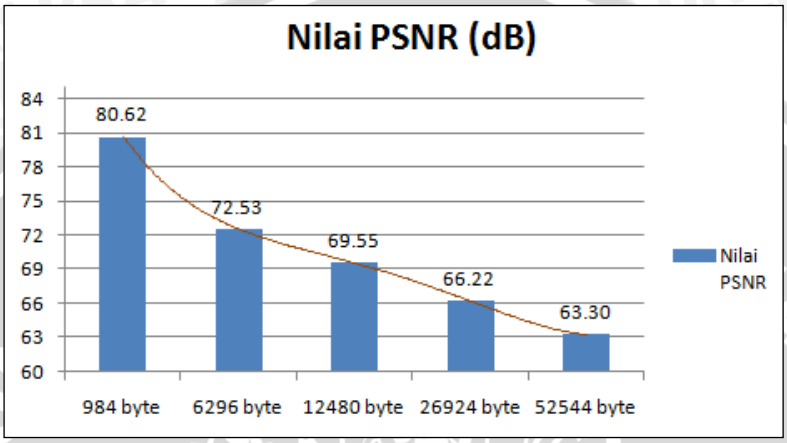


Gambar 5.1 Grafik Persentase Nilai MSE pada *audio1.mp3*.

Dari grafik nilai MSE yang ditunjukkan pada Gambar 5.1, terlihat bahwa pengujian pada berkas *audio1.mp3* memiliki nilai *MSE* terendah yaitu 0,06% ketika disisipkan pesan berukuran 984 byte. Sedangkan untuk nilai MSE tertinggi yaitu 3,04% ketika disisipkan pesan berukuran 52,544 byte. Dari grafik yang sama, dapat diketahui bahwa besarnya nilai MSE semakin bertambah seiring bertambahnya ukuran pesan yang disisipkan. Hal ini dikarenakan semakin besar

ukuran berkas teks, maka kemungkinan bit-bit yang berbeda antara MP3 asli dan *stego* MP3 akan semakin besar.

Berdasarkan Tabel 5.3, dibuat grafik nilai PSNR pada berkas *audio1.mp3* yang disisipkan pesan berbeda. Grafik nilai PSNR pada *audio1.mp3* ditunjukkan pada Gambar 5.2.



Gambar 5.2 Grafik Nilai PSNR pada *audio1.mp3* Berdasarkan Tabel 5.3.

Dari grafik pada Gambar 5.2 diketahui bahwa nilai PSNR terendah yaitu 63.30 dB dihasilkan oleh *stego* MP3 yang disisipkan pesan berukuran 52,544 byte. Sedangkan untuk nilai PSNR tertinggi yaitu 80,62 dB ketika disisipkan pesan berukuran 984 byte. Dari grafik yang sama, dapat diketahui bahwa besarnya nilai PSNR semakin berkurang seiring bertambahnya ukuran pesan yang disisipkan.

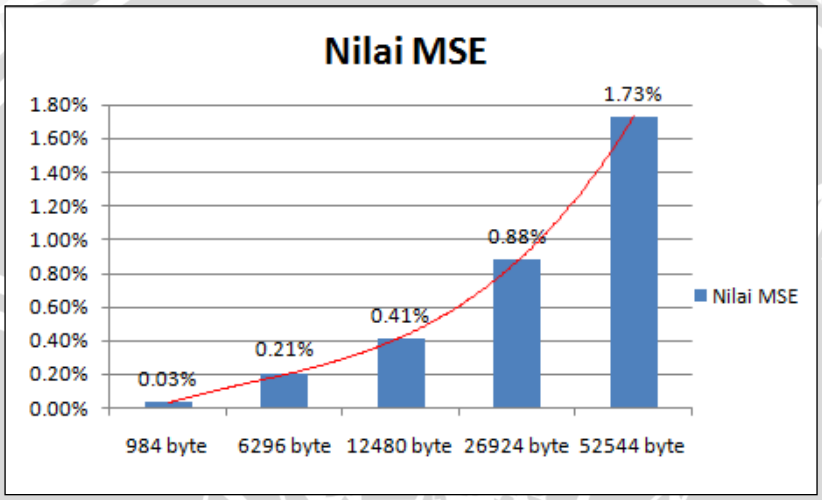
Pengujian selanjutnya dilakukan dengan menggunakan berkas MP3 "*audio2.mp3*" sebagai media penyisipan pesan yang telah dienkripsi. Hasil pengujian ditunjukkan pada Tabel 5.4.

Tabel 5.4 Hasil nilai pengujian tiap pesan pada berkas "*audio2.mp3*".

MP3 Asli	Pesan	Rasio Pesan	MSE	PSNR
audio2.mp3 (*1,519,965* byte)	*hello.txt*	0.54 %	0.03%	83.08
	merdeka.txt	3.48 %	0.21%	74.96
	random-10k.txt	6.91 %	0.41%	72.01
	random-20k.txt	14.90 %	0.88%	68.68
	permendiknas.txt	29.08 %	1.73%	65.76
Nilai rata-rata pengujian *stego* 2			**0.65%**	**72.90**

Dari Tabel 5.4 dapat dilihat bahwa hasil pengujian terhadap media *audio2.mp3* dengan disisipkan beberapa pesan berkapasitas berbeda menghasilkan nilai MSE dan PSNR yang berbeda pula, namun menghasilkan nilai yang lebih baik dibandingkan pengujian sebelumnya. Hal ini ditunjukkan dengan rata-rata nilai persentase MSE yaitu 0,65 % dan untuk nilai PSNR yaitu 72,99 dB.

Berdasarkan Tabel 5.4 maka dibuat grafik persentase MSE pada setiap s*tego* MP3 dengan pesan yang berbeda. Persentase MSE ditunjukkan pada Gambar 5.3.

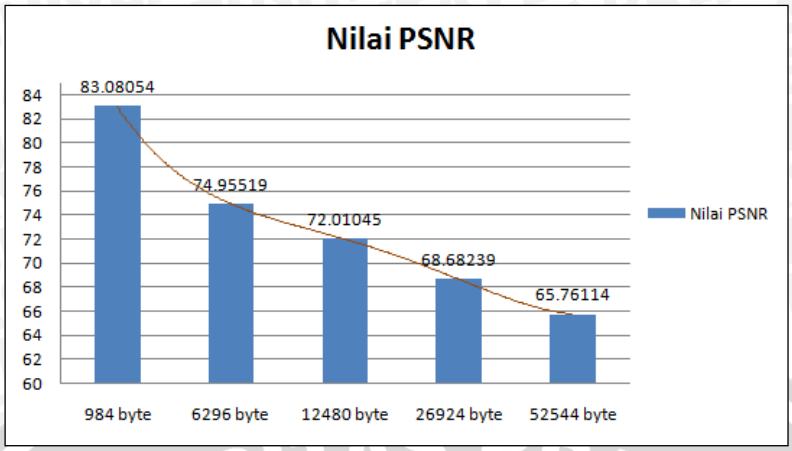


Gambar 5.3 Grafik Persentase Nilai MSE pada *audio1.mp3.*

Dari grafik nilai MSE yang ditunjukkan pada Gambar 5.3, terlihat bahwa pengujian pada berkas *audio1.mp3* memiliki nilai *MSE* terendah yaitu 0,03% ketika disisipkan pesan berukuran 984 byte. Sedangkan untuk nilai MSE tertinggi yaitu 1,73% ketika disisipkan pesan berukuran 52,544 byte.

Sama dengan pengujian yang dilakukan pada *audio1.mp3*, dapat diketahui bahwa besarnya nilai MSE semakin bertambah seiring bertambahnya ukuran pesan yang disisipkan. Hal ini dikarenakan semakin besar ukuran berkas teks, maka kemungkinan bit-bit yang berbeda antara MP3 asli dan *stego* MP3 akan semakin besar.

Berdasarkan Tabel 5.4, dibuat grafik nilai PSNR pada berkas *audio2.mp3* yang disisipkan pesan berbeda. Grafik nilai PSNR pada *audio2.mp3* ditunjukkan pada Gambar 5.4.



Gambar 5.4 Grafik Nilai PSNR pada *audio2.mp3* Berdasarkan Tabel 5.4.

Dari grafik pada Gambar 5.4 ini diketahui bahwa nilai PSNR terendah yaitu 65,76 dB dihasilkan oleh *stego* MP3 yang disisipkan pesan berukuran 52,544 byte. Sedangkan untuk nilai PSNR tertinggi yaitu 83,08 dB ketika disisipkan pesan berukuran 984 byte. Dari grafik yang sama, dapat diketahui bahwa besarnya nilai PSNR semakin berkurang seiring bertambahnya ukuran pesan yang disisipkan.

Sama halnya dengan pengujian yang dilakukan pada *audio1.mp3*, Dari grafik diatas dapat diketahui bahwa nilai PSNR yang didapatkan, semakin besar ukuran pesan teks yang disisipkan maka semakin kecil nilai PSNR yang didapatkan. Namun sebaliknya semakin kecil ukuran pesan yang disisipkan semakin besar nilai PSNR yang didapatkan.

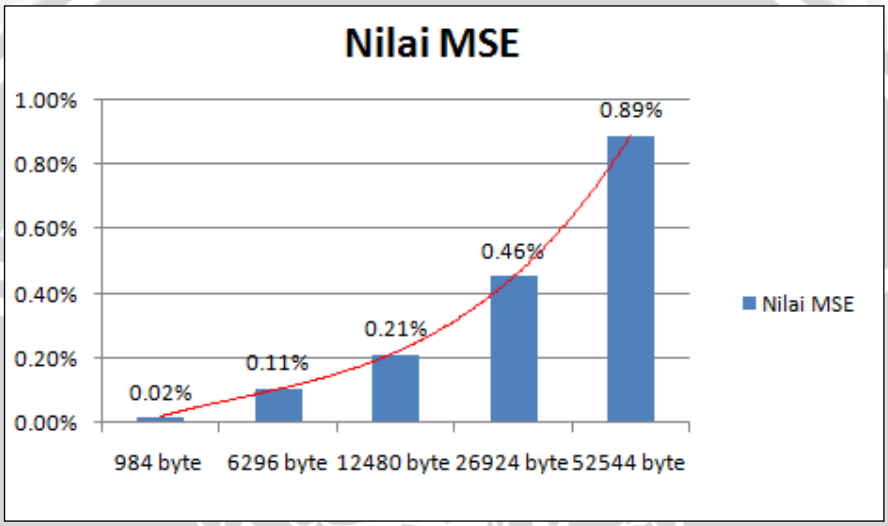
Pengujian selanjutnya dilakukan dengan menggunakan berkas MP3 "*audio3.mp3*" sebagai media penyisipan pesan yang telah dienkripsi. Hasil pengujian ditunjukkan pada Tabel 5.5.

Tabel 5.5 Hasil nilai pengujian tiap pesan pada berkas "*audio3.mp3*"

MP3 Asli	Pesan	Rasio Pesan	MSE	PSNR
audio2.mp3 (*1,519,965* byte)	*hello.txt*	0.28%	0.02%	85.95
	merdeka.txt	1.79%	0.11%	77.87
	random-10k.txt	3.55%	0.21%	74.88
	random-20k.txt	7.67%	0.46%	71.55
	permendiknas.txt	14.96%	0.89%	68.64
Nilai rata-rata pengujian *stego 3*			**0.34%**	**75.78**

Dari Tabel 5.5 dapat dilihat bahwa hasil pengujian terhadap media *audio3.mp3* dengan disisipkan beberapa pesan berkapasitas berbeda menghasilkan nilai MSE dan PSNR yang berbeda namun menghasilkan nilai yang lebih baik dibandingkan pengujian sebelumnya. Hal ini ditunjukkan dengan rata-rata nilai persentase MSE yaitu 0,34 % dan untuk nilai PSNR yaitu 75.78 dB.

Berdasarkan Tabel 5.5 maka dibuat grafik persentase MSE pada setiap s*tego* MP3 dengan pesan yang berbeda. Persentase MSE ditunjukkan pada Gambar 5.5.

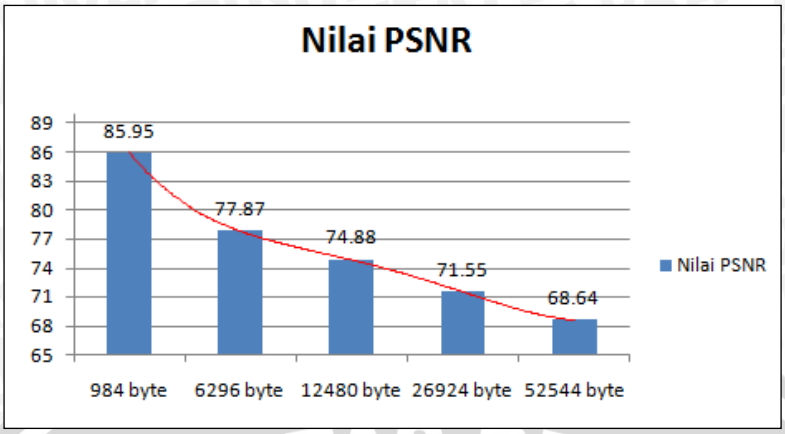


Gambar 5.5 Grafik Persentase Nilai MSE pada *audio1.mp3*.

Dari grafik nilai MSE yang ditunjukkan pada Gambar 5.5, terlihat bahwa pengujian pada berkas *audio3.mp3* memiliki nilai *MSE* terendah yaitu 0,02% ketika disisipkan pesan berukuran 984 byte. Sedangkan untuk nilai MSE tertinggi yaitu 0,89% ketika disisipkan pesan berukuran 52,544 byte.

Sama dengan pengujian yang dilakukan pada *audio1.mp3* dan *audio2.mp3*, dapat diketahui bahwa besarnya nilai MSE semakin bertambah seiring bertambahnya ukuran pesan yang disisipkan. Hal ini dikarenakan semakin besar ukuran berkas teks, maka kemungkinan bit-bit yang berbeda antara MP3 asli dan *stego* MP3 akan semakin besar.

Berdasarkan Tabel 5.5, dibuat grafik nilai PSNR pada berkas *audio2.mp3* yang disisipkan pesan berbeda. Grafik nilai PSNR pada *audio2.mp3* ditunjukkan pada Gambar 5.6.



Gambar 5.6 Grafik Nilai PSNR pada *audio3.mp3* Berdasarkan Tabel 5.4.

Dari grafik pada Gambar 5.4 ini diketahui bahwa nilai PSNR terendah yaitu 68,64 dB dihasilkan oleh *stego* MP3 yang disisipkan pesan berukuran 52,544 byte. Sedangkan untuk nilai PSNR tertinggi yaitu 85,95 dB ketika disisipkan pesan berukuran 984 byte. Dari grafik yang sama, dapat diketahui bahwa besarnya nilai PSNR semakin berkurang seiring bertambahnya ukuran pesan yang disisipkan.

Sama halnya dengan pengujian yang dilakukan pada *audio1.mp3* dan *audio2.mp3*, Dari Gambar 5,6 dapat diketahui bahwa nilai PSNR yang didapatkan, semakin besar ukuran pesan teks yang disisipkan maka semakin kecil nilai PSNR yang didapatkan. Namun sebaliknya semakin kecil ukuran pesan yang disisipkan semakin besar nilai PSNR yang didapatkan.

Dari semua grafik pada skenario pengujian, menunjukkan bahwa semakin besar ukuran berkas yang disisipkan, maka nilai MSE semakin bertambah. Hal ini disebabkan karena semakin besar ukuran berkas yang disembunyikan, semakin banyak pula bit pada berkas MP3 yang diubah.

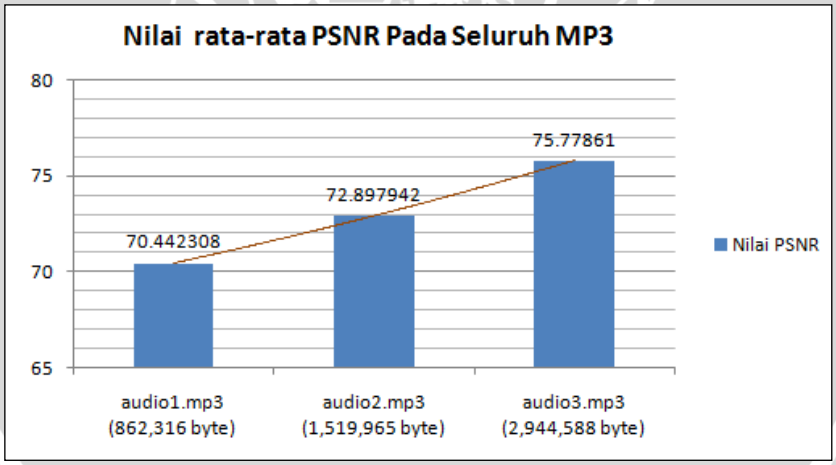
Sedangkan untuk nilai PSNR, semakin besar berkas yang disisipkan semakin turun nilai PSNR yang dihasilkan. Hal ini dikarenakan perubahan bit-bit MP3 setelah disisipkan pesan mempengaruhi kualitas MP3 yang dihasilkan. Dengan kata lain semakin besar pesan yang disisipkan semakin berkurang kualitas MP3 yang dihasilkan.

Pada pesan yang disisipkan didapatkan dari proses enkripsi menggunakan algoritma rijndael. Proses enkripsi dilakukan pada pesan dengan memproses blok-

blok data hingga menghasilkan pesan terenkripsi. Besar ukuran pesan terenkripsi yang dihasilkan tidak berbeda dengan pesan sebelum dienkripsi. Hal ini tentunya tidak memengaruhi besar ukuran pesan yang disisipkan.

Pada media penyisipan, semakin besar ukuran berkas MP3 maka semakin besar pula nilai PSNR. Berkas MP3 yang memiliki ukuran besar apabila dilakukan penyembunyian berkas ke dalamnya maka jumlah bit yang terubah pada berkas tersebut akan lebih sedikit bila dibandingkan dengan penyembunyian pada berkas MP3 yang ukurannya lebih kecil. Sehingga berkas MP3 yang memiliki ukuran lebih besar cenderung mengalami lebih sedikit penurunan kualitas dibandingkan dengan berkas MP3 yang memiliki ukuran yang kecil.

Hali ini sesuai dengan apa diperlihatkan pada grafik pada Gambar 5.7. Grafik pada Gambar 5.7 menunjukkan nilai rata-rata PSNR pada masing-masing pengujian. Nilai rata-rata PSNR paling tinggi didapatkan pada pengujian ke-3 yang menggunakan media MP3 berkapasitas paling besar. Nilai rata-rata PSNR yang paling rendah didapatkan pada pengujian ke-1 yang menggunakan media MP3 berkapasitas paling kecil.



Gambar 5.7 Grafik Nilai Rata-Rata PSNR pada Seluruh MP3.

Dari seluruh skenario pengujian, diperoleh nilai PSNR tertinggi yaitu 85,95 dB yang terdapat pada *audio3.mp3* berkapasitas 351,117 byte yang disisipkan berkas "*hello.txt*" yang telah dienkripsi dengan rasio pesan sebesar 0.28 %. Sedangkan nilai PSNR terendah yaitu 63,30 dB yang terdapat pada *audio1.mp3* berkapasitas 102,056 byte yang disisipkan berkas "*permendiknas.txt*" yang telah dienkripsi dengan rasio pesan sebesar 51.49 %.

Dari fakta yang diuraikan pada paragraf sebelumnya, diketahui bahwa PSNR tertinggi terdapat pada *stego* MP3 yang memiliki ukuran paling besar yaitu *audio3.mp3*. Selain itu PSNR tertinggi juga terdapat pada *stego* MP3 yang disisipi berkas dengan ukuran paling kecil yaitu "*hello.txt*". Hal ini sesuai dengan analisa bahwa penyembunyian berkas teks berukuran kecil dan disembunyikan pada berkas media MP3 yang berukuran besar akan menghasilkan nilai PSNR yang tinggi.

Sedangkan PSNR terendah terdapat pada *stego* MP3 yang memiliki ukuran paling kecil yaitu *audio1.mp3*. Selain itu PSNR terendah juga terdapat pada *stego* MP3 yang disisipi berkas dengan ukuran paling besar yaitu "*permendiknas.txt*". Hal ini sesuai dengan analisa bahwa penyembunyian berkas teks berukuran besar dan disembunyikan pada berkas MP3 *carrier* yang berukuran kecil akan menghasilkan nilai PSNR yang rendah.

BAB VI
PENUTUP

6.1 Kesimpulan

Setelah melakukan penelitian maka dapat disimpulkan bahwa :

1. Metode *least significant bit* dapat diimplementasikan untuk menyembunyikan pesan berupa berkas teks terenkripsi ke dalam berkas audio MP3. Langkah awal proses penyembunyian adalah dengan mengenkripsi pesan dengan algoritma Rijndael kemudian mengubah hasil enkripsi atau *ciphertext* menjadi bentuk biner. Penyembunyian pesan dilakukan dengan mengganti nilai bit *least significant* dari main data setiap frame MP3 dengan bit *ciphertext*. Sedangkan untuk pengungkapan pesan dilakukan dengan cara mengambil bit-bit *least significant* dari stego MP3.
2. Kualitas berkas MP3 yang telah disisipi pesan atau stego MP3 dapat diketahui melalui nilai PNSR. Nilai PSNR yang didapatkan berkisar antara 63.30 dB hingga 85.95 dB. Semakin besar ukuran berkas yang disisipkan, maka nilai PSNR semakin menurun. Berbeda dengan ukuran media MP3, dengan pesan yang sama, semakin besar ukuran media MP3 yang digunakan untuk menyisipkan pesan maka semakin besar pula nilai PSNR.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan penelitian selanjutnya :

1. Pada penelitian selanjutnya dapat digunakan media penampung berupa berkas audio tidak terkompresi seperti WAV atau AIFF daripada berkas audio dengan kompresi *lossy* seperti MP3, sehingga harapan peneliti nantinya hasil audio yang disisipkan pesan menunjukkan kualitas lebih baik.
2. Pada penelitian selanjutnya dapat digunakan berkas dengan format selain teks sebagai pesan yg disembunyikan seperti gambar, suara atau video.

DAFTAR PUSTAKA

[ALA-09] Alatas, Putri. 2009. Implementasi Teknik Staganografi Dengan Metode LSB Pada Citra Digital. Tugas Akhir Jurusan Sistem Informasi Fakultas Ilmu Komputer & Teknologi Informasi Universitas Gunadarma, Depok.

[ARI-07] Ariyus, Dony. 2007. Keamanan Multimedia. Yogyakarta: Andi Offset.

[ARI-08] Ariyus, Dony. 2008. Pengantar Ilmu Kriptografi : Teori, Analisis, Implementasi. Yogyakarta: Andi Offset.

[CHA-09] Chasanah, Zulfah. 2009. Steganografi Pada File Audio MP3 Untuk Pengamanan Data Menggunakan Metode *Least Significant Bit* (LSB). Tugas Akhir Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri, Malang.

[DAE-02] Daemen, Joan dan Vincent Rijmen. *The Design of Rijndael : AES – The Advanced Encryption Standard*. Springer. Berlin.

[FEB-12] Febriansyah, Ilham N. 2012. *Implementasi Least Significant Bit (LSB) Insertion Untuk Menyembunyikan Ciphertext Blowfish pada Citra Digital*. Tugas Akhir Jurusan Matematika. Universitas Brawijaya.

[KAL-10] Kalangi, Jeff Andre. 2010. Pembuatan Aplikasi *Steganography* Pada *File* Audio MP3 Dengan Metode *Parity Coding*. Tugas Akhir Jurusan Teknik Informatika Universitas Komputer Indonesia,

[KUM-12] Kumar, M Anand dan Dr. S. Karthikeyan. *Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms*. Dipublikasikan secara *online* pada Maret 2012 oleh MECS (http://www.mecs-press.org/).

[MUN-04] Munir, Rinaldi. 2004. Pengolahan Citra Digital Dengan Pendekatan Algoritmik. Bandung: Informatika.

[NEY-11] Neyman, Shelvie Nidya dan Ayi Dianitasari. 2011. Evaluasi Performansi Metode *Phase Coding* Pada Teknik Audio Watermarking. SNATIKA 2011/edisi 01/Tahun 2011, Bogor.

[RAS-02] Rassol, Raissi. 2002. *Theory Behind MP3*. http://mp3-tech.org/programmer/docs/mp3_theory.pdf. Diakses tanggal 28 Oktober 2012.

[SAR-06] Saragih, Arlando Riko. 2006. Metode *Parity Coding Versus* Metode *Spread Spectrum* Pada Audio *Steganography*. Seminar Nasionanl Aplikasi Tteknologi Informasi 2006 , 17 Juni 2006, Yogyakarta.

[WIB-09] Wibowo, Ivan, Budi Susanto, dan Junius Karel T. 2009. Penerapan Algoritma Kriptografi Asimetris Untuk Keamanan Data Di Oracle 8. Jurnal Informatika, Volume 5 Nomor 1, April 2009, Yogyakarta.