BAB 3

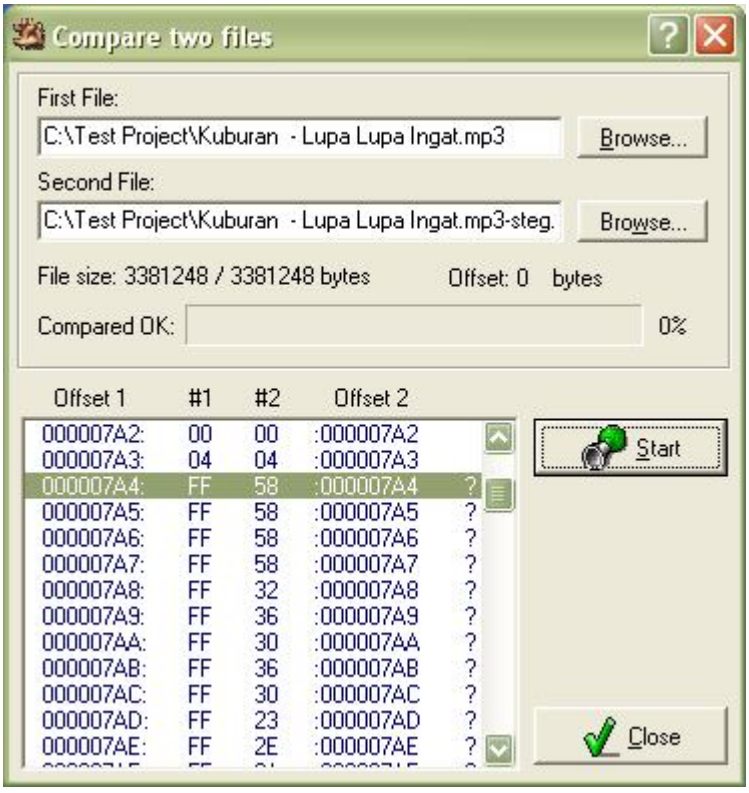
ANALISIS DAN PERANCANGAN SISTEM

Analisis ditujukan untuk memberikan gambaran secara umum terhadap perangkat lunak. Hasil analisis akan menjadi dasar untuk melakukan perancangan atau desain perangkat lunak sesuai kebutuhan sistem.

3.1 Analisis Keamanan Data pada Audio Steganografi

Semakin berkembangnya teknologi informasi dewasa ini, diikuti pula dengan semakin meningkatnya ancaman serangan pada jalur pertukaran informasi melalui internet. Saat ini dibutuhkan upaya keamanan yang lebih baik dalam mengamankan informasi. Salah satunya adalah dengan menggunakan steganografi. Pesan rahasia disisipkan ke dalam media digital lainnya tanpa menimbulkan perubahan yang dapat dipersepsi oleh manusia akibat proses penyisipan tersebut. Dalam hal audio steganografi, metode ini memanfaatkan keterbatasan pada indera pendengaran manusia (*human auditory system*). Dalam steganografi, format audio memiliki kelebihan dibandingkan format citra maupun video. Berkas audio biasanya berukuran relatif lebih besar dibandingkan dengan format citra, sehingga dapat menampung pesan rahasia dalam jumlah yang lebih besar pula. Adapun format video memang berukuran relatif sangat besar, tetapi ukurannya yang sangat besar tersebut mengurangi kepraktisannya dan juga kurangnya algoritma yang mendukung format ini (Ariyus, 2007).

Akan tetapi saat ini telah banyak diciptakan metode-metode dan perangkat steganalisis (*steganalysis)* yang digunakan untuk mendeteksi penggunaan steganografi pada berkas digital. Proses steganalisis dapat dilakukan misalnya dengan membandingkan *stegged-file* dengan media asli, dengan memperhatikan perubahan bentuk, degradasi atau karakteristik yang tidak biasa dari media yang telah disisipi. Sebagai contoh proses steganalisis dapat dilakukan dengan cara sederhana berikut ini, yaitu membandingkan nilai heksadesimal dari dua buah berkas MP3.



Gambar 3.1 Salah Satu Metode Steganalisis : Membandingkan Dua Berkas MP3 (dilihat dengan menggunakan *freeware* Audiograbber)

Dari contoh di atas dapat dilihat bahwa ukuran kedua MP3 adalah sama yaitu 3.381.248 bytes, namun byte-byte di urutan tertentu yang terdapat di dalamnya berbeda. Bagi para steganalis hal ini akan menimbulkan kecurigaan dan mengundang untuk diperiksa lebih lanjut. Dengan analisis statistik, maka pesan yang terkandung dalam berkas tersebut dapat dengan mudah ditemukan.

Dari permasalahan di atas, muncul gagasan untuk mengupayakan peningkatan keamanan data pada audio steganografi, yaitu dengan mengenkripsi pesan sebelum disisipkan ke dalam media steganografi. Sehingga walaupun pesan dapat dideteksi, namun steganalis tetap tidak dapat membacanya karena masih dalam keadaan terenkripsi. Ilustrasi peningkatan keamanan data pada audio steganografi menggunakan enkripsi adalah sebagai berikut:



Gambar 3.2 Ilustrasi Audio Steganografi Murni

Ilustrasi di atas menggambarkan sistem audio steganografi murni. Metode ini rentan terhadap steganalisis. Jika steganalis mampu mendeteksi keberadaan pesan, maka pesan akan segera didapatkan oleh steganalis tersebut. Namun tidak demikian halnya jika pesan terlebih dahulu di enkripsi dengan algoritma kriptografi yang kokoh sebelum dilakukan penyisipan. Maka keamanan data akan semakin meningkat, karena walaupun steganalis berhasil mendeteksi keberadaan pesan, pesan tersebut masih dalam keadaaan terenkripsi dan hanya dapat dibuka dengan kunci yang sesuai.



Gambar 3.3 Ilustrasi Penambahan Enkripsi Rijndael pada Audio Steganografi

Selain itu, yang harus diperhatikan dalam membangun sebuah perangkat steganografi dengan tingkat kemanan yang baik adalah sistem harus dapat memenuhi aspek-aspek keamanan data sebagai berikut:

1. ***Fidelity***

   Media yang dijadikan sebagai penampung pesan tidak mengalami degradasi yang signifikan. Artinya MP3 masih dapat didengarkan dengan baik, tidak terdapat derau ataupun perbedaan yang mencolok antara MP3 asli dengan MP3 yang telah disisipi pesan rahasia. Hal ini penting sebagai antisipasi awal untuk mengelabui para steganalis dan meminimalisir kecurigaan agar MP3 tidak diperiksa lebih lanjut oleh para steganalis.

2. ***Recovery***

   Data yang disisipkan ke dalam MP3 harus dapat diungkapkan kembali dan menghasilkan pesan yang sesuai dengan pesan asli dan tidak rusak.

3. ***Robustness***

   Pesan rahasia yang disisipkan harus tahan terhadap pengolahan sinyal yang mungkin dilakukan, misalnya kompresi, perubahan ukuran, perubahan *bitrate* dan lainnya.

4. ***Security***

   Perangkat lunak dapat menjamin keamanan pesan rahasia. Ketika kerahasiaan pesan itu ingin dibuka, hanya dapat dilakukan dengan kunci untuk digunakan dalam proses penyisipan dan pengungkapan pesan.

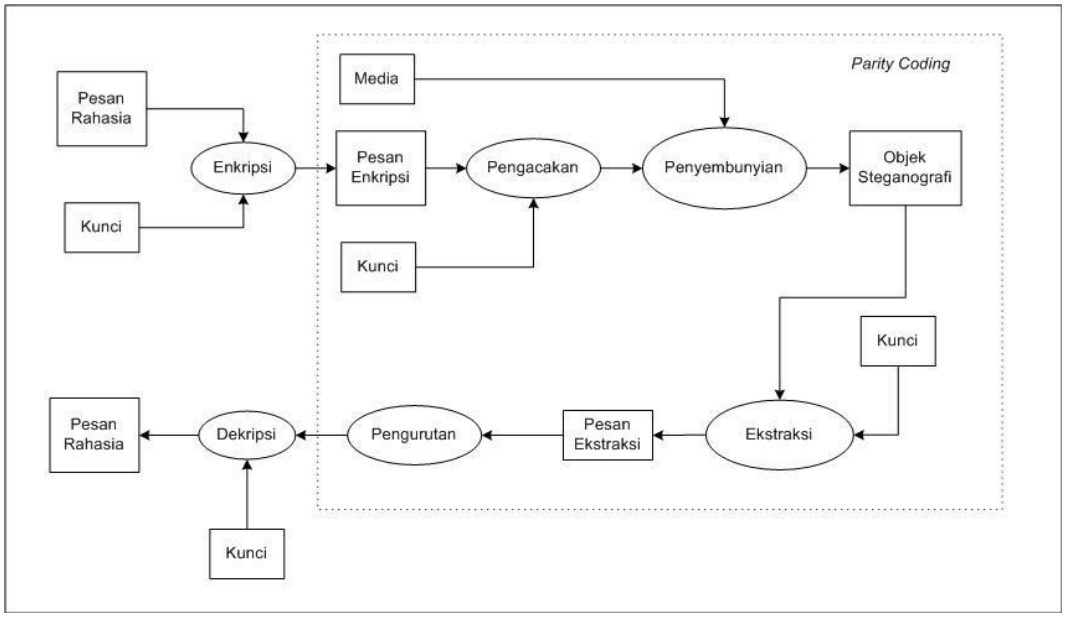
3.2 Perbandingan Metode Parity Coding dan Metode Spread Spectrum

Pada Steganografi audio, ada beberapa metode yang sering digunakan selain metode *parity coding*, yaitu metode *spread spectrum*. Penulis disni akan

membandingkan cara kerja masing-masing metode berdasarkan informasi-informasi yang didapat.

3.2.1 Metode Parity Coding

Steganografi audio mengenal sebuah teknik yang dinamakan teknik parity coding. Pada teknik parity coding sinyal dari berkas audio dipecah menjadi beberapa región berbeda dan mengenkripsi setiap bit dari pesan rahasia yang ingin disisipkan pada sebuah sampel región yang berisi parity bit. Jika parity bit dalam región yang ditentukan tidak sesuai dengan bit pesan rahasia yang akan dienkripsi, maka proses diganti dengan menukarkan LSB pada salah satu sampel región, sehingga si pengirim mempunyai pilihan lebih dalam mengenkripsi bit pesan rahasia tersebut dan sinyal audio dapat diubah dengan cara yang lebih halus.

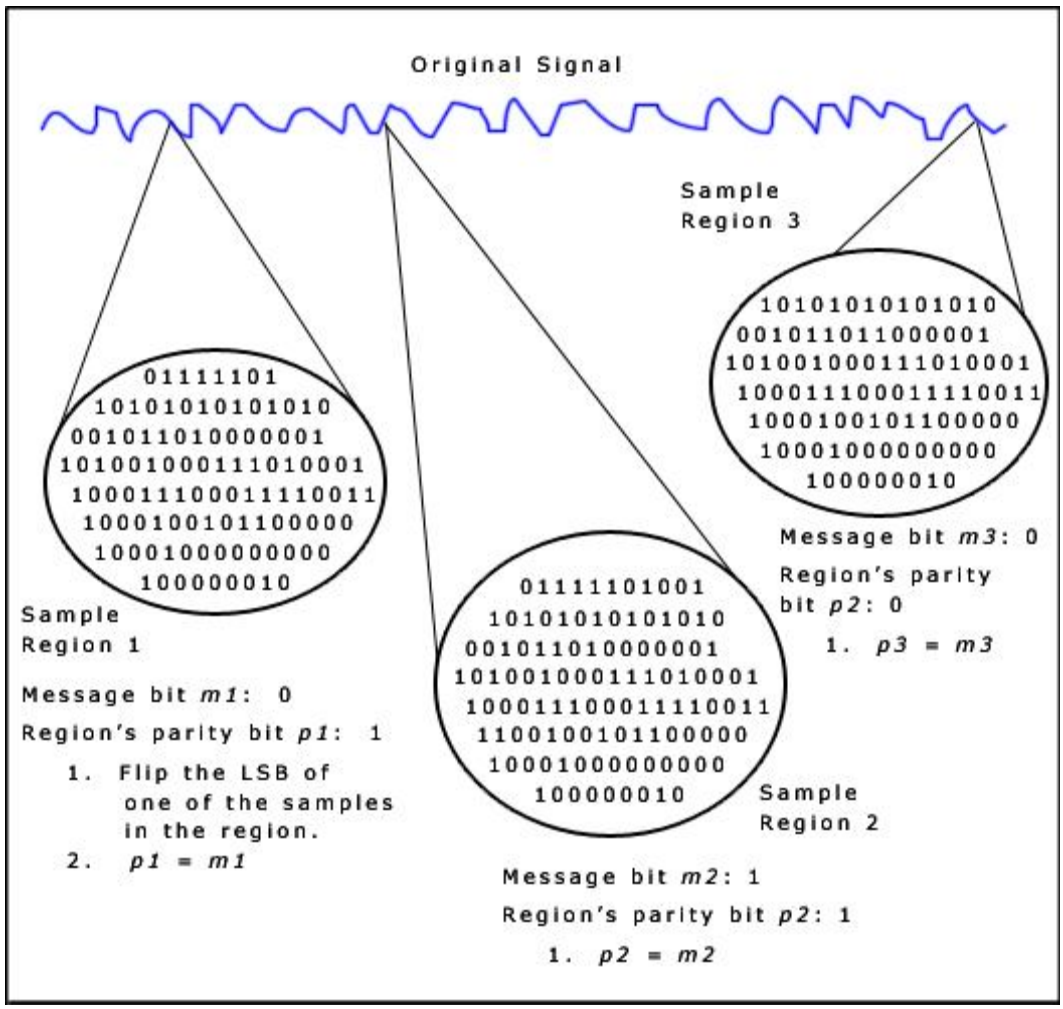


Gambar 3.4 Sistem Penyembunyian dan Ekstraksi Pesan

3.2.1.1 Penyembunyian pesan dengan Parity Coding

Sinyal media yang sudah diencode akan dibagi menjadi beberapa region terpisah dengan ukuran statis. *Parity* bit dari setiap region akan dihitung terlebih dahulu untuk disimpan nilainya. Bit dari pesan rahasia akan disisipkan secara merata kedalam region yang ada. Jika bit yang akan dimasukkan ke dalam region nilainya berbeda, maka susunan dari bit-bit LSB harus diubah sedimikian rupa sehingga *parity* bit region nilainya sama dengan bit pesan rahasia yang akan disisipkan namun jika nilainya sama region tidak diperlu diubah. Kondisi dalam penghitungan *parity* bit adalah *even parity*. **Gambar 3.5** adalah contoh penyembunyian suatu pesan rahasia “010” kedalam suat media stegano.

Proses pertama yang dilakukan dalam penyembunyian pesan adalah membagi media stegano yang dalam hal ini adala berkas MP3 kedalam region-region. Banyaknya region ini ditentukan oleh panjang isi berkas yang dijadikan objek stegano.



Gambar 3.5 Konsep Parity Coding

Berikut adalah spesifikasi faktor-faktor penyembunyian :

Nama objek : pesan.txt

Pesan *cipher* : 10001001110001011001011 0 (dalam biner)

Nama media : musik.mp3

Hasil encode media :

1010001100-0100100100-1010001100

0100100101-1010001100-0100100101

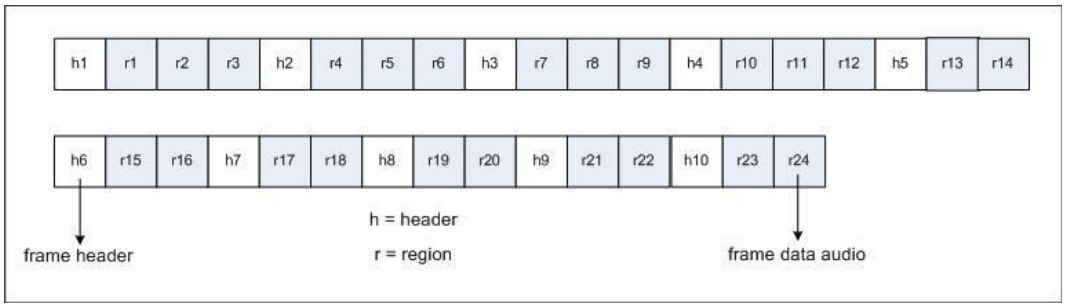
1010001100 0100100101 1010001100

0100100101-1010001100 0100100101

Dari spesifikasi diatas diketahui panjang dari isi pesan adalah 24 bit, sedangkan panjang bit dari media stegano adalah 120 bit yang terdiri dari 10 *frame*. Sebelum proses dilanjutkan ke pembagian region, informasi mengenai spesifikasi akan disimpan di header *frame* awal media steganografi. Hal ini berguna nantinya untuk melakukan ekstraksi agar pembagian region yang serupa bisa dilakukan saat ekstraksi.

Sebelum masuk ke proses pembagian region, pesan terenkripsi tersebut diacak terlebih dahulu dengan memanfaatkan angka-angka *pseudorandom* yang dibangkitkan oleh *pseudorandom generator* memakai kunci masukan.

Dalam proses pembagian region, media stegano akan dibagi menjadi region-region dengan banyak yang sesuai panjang isi pesan. **Gambar 3.6** menunjukkan contoh pembagian region pada media MP3 dengan panjang pesan 24 bit.



Gambar 3.6 Pembagian Region

Sebelum bit pesan *cipher* diproses secara parity, bit-bit pesan *cipher* tersebut terlebih dahulu diacak urutannya dengan menggunakan *random number* yang dibangkitkan dari kunci. Karena bit pesan *cipher* berjumlah 24 maka *random number* yang terbentuk berkisar dari 0 sampai 23. Sebagai contoh, *random number* yang terbentuk diasumsikan sebagai berikut :

16,3,8,6,4,10,14,9,12,23,1,20,13,18,11,2,5,22,15,17,7,19,0,21

maka bit pesan *cipher* akan diacak sesuai angka pengacakan yang berarti bit pertama pesan *cipher* akan dipindahkan ke posisi 16 (urutan *array*) atau menjadi bit ke-17 dalam susunan bit *cipher* yang baru dan seterusnya sehingga menjadi :

101011000100100011110001

Setelah media dibagi menjadi region-region yang dibutuhkan, bit-bit hasil pengacakan tersebut diproses secara *parity* dalam region masing-masing. Berikut adalah proses penyembunyian bit-bit pesan ke dalam region yang telah diasumsikan:

bit 1 : 1

region 1 : 10100011

parity bit region 1 → $1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0$

karena nilai bit pertama pesan tidak sama dengan *parity* bit region 1, maka bit-bit pada region 1 haruslah dimanipulasi sedemikian rupa agar nilai *parity* bit nya sama dengan bit pesan. Manipulasi yang dilakukan dalam tugas akhir ini adalah menukar LSB dari region.

bit 2 : 0

region 2 : 00010010

parity bit region 2 → 0 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0 = 0

karena nilai bit kedua pesan sama dengan *parity* bit region 2, maka region tidak perlu dimodifikasi.

Dari pertukaran region 1 yang dilakukan maka diperoleh region 1 baru dengan bit-bit sebagai berikut:

region 1 baru : 1010001-**1**

parity bit region 1 baru → 1⊕ 0 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ **1** = 0

Proses ini dilakukan keseluruh bit-bit yang pada pesan dalam region-region yang telah disediakan.

3.2.1.2 Ekstraksi Pesan dengan Parity Coding

Ekstraksi yang dilakukan adalah kebalikan proses dari proses penyembunyian data seperti pada **Gambar 3.4**. Pesan rahasia akan diekstrak dari media stegano dengan kunci yang sama dengan yang dimiliki oleh pengirim. Kunci ini dipakai untuk membangkitkan kembali angka-angka pseudorandom yang dipakai untuk mengacak pesan terenkripsi sebelumnya sehingga bisa diurutkan kembali ke posisi semula. Setelah proses ini, pesan harus didekripsi lagi menggunakan bit kunci *hash* sesuai ukuran blok enkripsi agar pihak penerima bisa mengetahui orisinalitas pesan tersebut.

Proses yang dilakukan untuk mengekstraksi adalah dengan menyusun kembali nilai-nilai *parity* bit yang ada pada setiap region. Untuk menghasilkan *parity* bit

region yang benar, tentu dibutuhkan pembagian region yang sama juga dengan proses penyembunyian. Oleh karena itu, dibutuhkan informasi spesifikasi penyembunyian yang disimpan pada byte pertama setiap *frame* media steganografi. Setelah dibaca barulah proses dilanjutkan ke pembagian region-region yang kemudian dilanjutkan dengan penghitungan *parity* bit region dimana nilai *parity* bit inilah yang disusun menjadi isi pesan setelah dienkripsi.

3.2.1.3 Pengujian Perangkat Lunak

Metode proses pengukuran kualitas audio akan dilakukan secara subjektif dan objektif. Cara subjektif yaitu dengan melakukan pengamatan langsung terhadap audio hasil penyembunyian dan audio yang asli. Sedangkan cara objektif akan memakai perhitungan nilai PSNR (*Peak Signal to Noise Ratio*) dengan nilai minimal 30 DB. Perhitungan PSNR ini dilakukan dengan memakai rumus persamaan :

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{P_1^2}{P_1^2 + P_0^2 - 2P_1P_0}\right)$$

Dimana P1 adalah kekuatan sinyal berkas audio setelah proses penyembunyian pesan dan P0 adalah kekuatan sinyal awal.

Pengujian kinerja perangkat lunak menunjukkan hasil yang memuaskan. Kecepatan proses penyembunyian dan ekstraksi tergantung pada besarnya media steganografi yang digunakan dan besarnya pesan yang disembunyikan. Akan tetapi, ketergantungan ini lebih dipengaruhi oleh besar media. Hal ini dikarenakan proses penyembunyian membutuhkan pembacaan media secara menyeluruh

terlebih dahulu untuk menganalisis struktur berkas audio tersebut sedangkan pada proses utamanya dibutuhkan pembacaan media sekali lagi untuk pembagian region dan penulisan bit *parity* region.

3.2.2 Metode Spread Spectrum

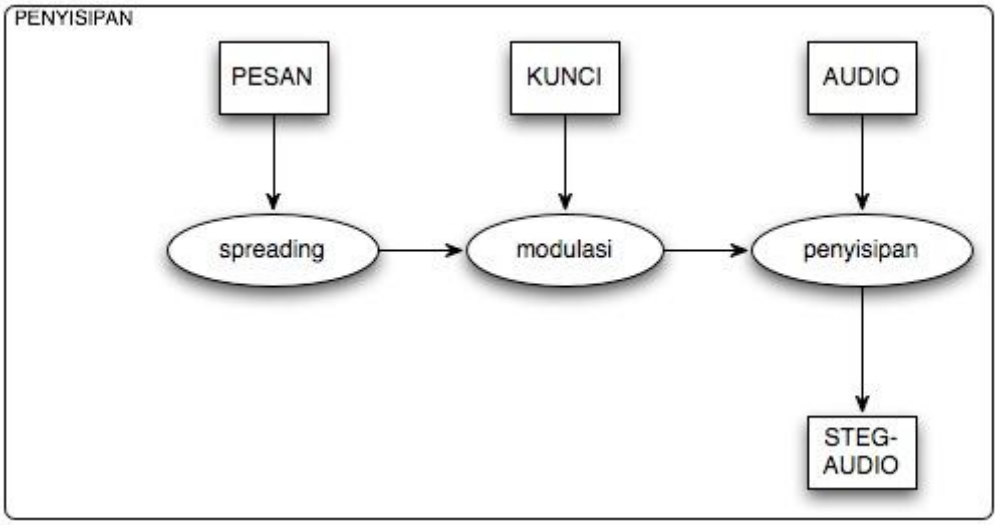
Metode *Spread Spectrum* mentransmisikan sebuah sinyal pita informasi yang sempit ke dalam sebuah kanal pita lebar dengan penyebaran frekuensi. Penyebaran ini berguna untuk menambah tingkat redundansi. Besaran redundansi ditentukan oleh faktor pengali *cr* yang bernilai skalar. Panjang bit- bit hasil penyebaran ini menjadi *cr* kali panjang bit–bit awal.

3.2.2.1 Penyisipan Pesan Pada Metode Spread Spectrum

Sistem untuk menyisipkan pesan pada audio membutuhkan masukan berupa audio dengan format MP3 (*cover-file*), pesan yang akan disisipkan (*message*), kunci (*key*), dan factor pengali *cr*.

Untuk proses penyisipan pesan, pertama dilakukan proses penyebaran (*spreading*) bit–bit informasi dari pesan yang akan disisipkan (*message*) dengan mengalikan bit-bit pesan dengan faktor pengali *cr*. Setelah itu, bit–bit informasi hasil penyebaran itu akan dimodulasi dengan *pseudo-noise signal* yang dibangkitkan menggunakan algoritma LCG secara acak berdasarkan kunci penyembunyiannya (*key*). Hasil dari proses modulasi ini akan disisipkan sebagai *noise* ke dalam sebuah berkas media audio MP3 (*cover-file*). Media yang telah disisipi inilah yang disebut *stego-audio*. *Stego-audio* ini akan mengalami sedikit penurunan dibanding

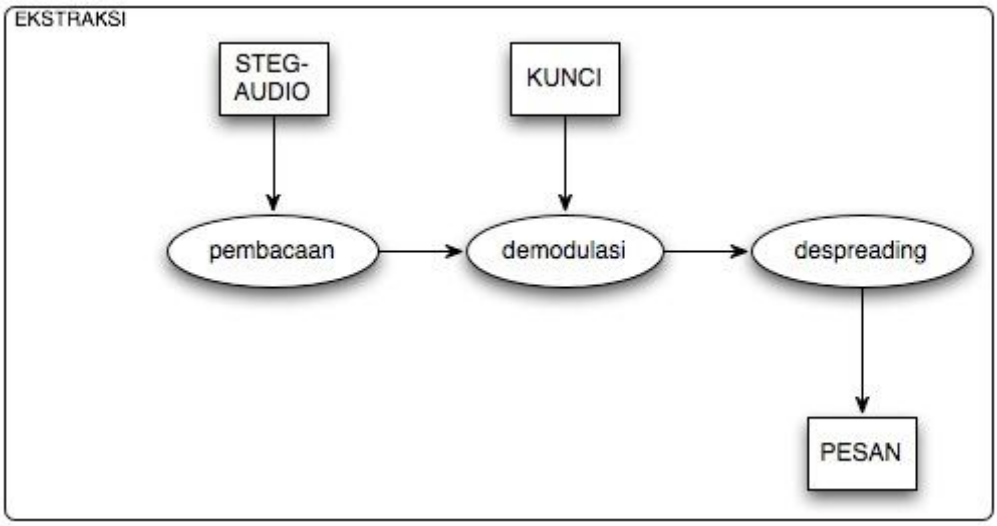
audio masukan. Namun penurunan kualitas diharapkan tidak dapat dideteksi oleh indra manusia



Gambar 3.7 Sistem Penyisipan Pesan Pada Metode Spread Spectrum

3.2.2.2 Ekstraksi Pesan Pada Metode Spread Spectrum

Sistem untuk ekstraksi pesan dari audio membutuhkan masukan berupa stego-audio, kunci (*key*), dan faktor pengali *cr*. Untuk proses ekstraksi pesan, pertama dilakukan proses penyaringan terhadap media yang telah disisipi (*stego-file*) untuk mendapatkan *noise. Noise* ini yang kemudian akan di-demodulasi dengan menggunakan *pseudo-noise signal* yang sama dengan *pseudo-noise signal* hasil pembangkitan berdasarkan kunci (*key*) pada proses penyisipan, untuk mendapatkan bit–bit yang berkolerasi. Proses selanjutnya adalah melakukan proses *de-spreading* menggunakan faktor pengali *cr* untuk menghasilkan bit–bit informasi sesungguhnya.



Gambar 3.8 Sistem Ekstraksi Pesan Pada Metode Spread Spectrum

3.2.2.3 Hasil Pengujian Perangkat Lunak

Dari hasil pengujian perangkat lunak, terbukti bahwa Steganografi dengan Metode *Spread Spectrum* dapat melakukan aspek pengamanan dengan menggunakan faktor pengali *cr*. Proses ekstraksi dengan nilai *cr* yang salah menghasilkan pesan yang berbeda dengan pesan yang asli. Pengujian juga dilakukan dengan membandingkan berkas audio hasil penyisipan , baik secara subjektif maupun objektif. Untuk cara subjektif, audio hasil penyisipan tidak dapat dibedakan dengan audio asli. Untuk cara objektif, dicari nilai PSNR (*Peak Signal to Noise Ratio*) dari masing-masing audio dalam satuan decibel (dB), dengan rumus:

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{P_I^{\,2}}{P_I^{\,2} + P_0^{\,2} - 2P_IP_0}\right)$$

dengan P0 menyatakan kekuatan sinyal awal dan P1 menyatakan kekuatan sinyal setelah disisipi data. P0 dan P1 diukur dalam satuan desibel (dB). Nilai PSNR yang wajar pada pembandingan dua berkas audio berkisar pada 30-50 dB.

Dari hasil pengujian, nilai PSNR berada dalam kisaran nilai PSNR yang wajar.

Dari hasil pengujian perangkat lunak Steganografi dengan kedua Metode diatas, bisa diambil kesimpulan, bahwa:

1. Steganografi audio dengan metode *parity coding* dan metode *spread spectrum* bisa diterapkan pada berkas audio MP3.
2. Pada Metode *Spread Spectrum*:
   - Kualitas berkas audio yang dihasilkan bergantung dari besarnya ukuran pesan. Pada perangkat lunak yang dibangun, faktor pengali *cr* untuk penyisipan dapat diatur oleh pengguna untuk menggandakan ukuran pesan. Tujuan dari penggandaan ini adalah membuat pesan terlihat seperti *noise* pada berkas audio MP3, sehingga pihak yang tidak berwenang tidak menyadari keberadaan pesan.
   - Pengujian nilai PSNR menunjukkan bahwa nilai PSNR menurun seiring dengan bertambahnya ukuran pesan yang disisipkan. Jika ukuran pesan yang disisipkan semakin besar maka nilai PSNR semakin kecil yang berarti kualitas berkas audio yang disisipkan semakin buruk.

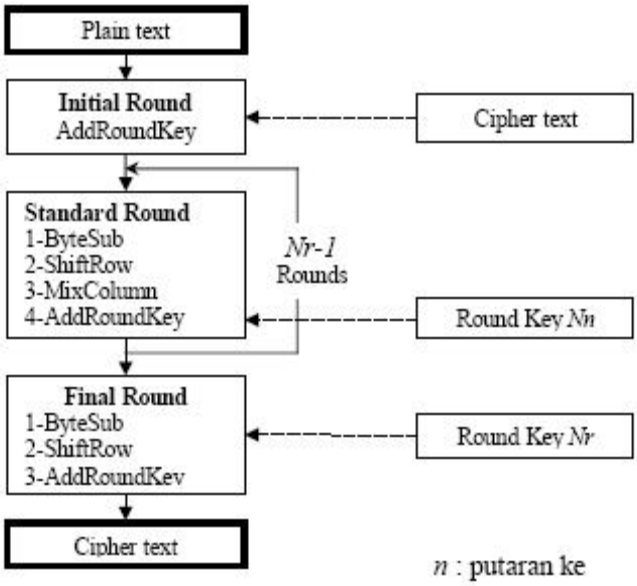
3. Pada Metode *Parity Coding*:
    - Kualitas berkas audio yang dihasilkan tergantung dari besarnya ukuran pesan. Semakin besar pesan yang dimasukkan, maka semakin besar pula noise yang terbentuk. Noise tersebut bisa dikurangi dengan cara membagi region secara merata ke seluruh data audio sehingga kualitas bisa tetap terjaga.
    - Pengujian PSNR menunjukkan bahwa nilai PSNR cenderung menurun seiring dengan bertambahnya ukuran pesan yang disembunyikan. Jika ukuran pesan yang disembunyikan semakin besar, maka nilai PSNR semakin kecil yang berarti kualitas berkas audio yang disisipkan semakin buruk. Namun kecenderungan tersebut bisa berubah tergantung kepada kesesuaian jumlah bit pesan dengan jumlah frame media (MP3 *cover*).

Pada prinsipnya metode-metode dalam Steganografi audio khususnya MP3, memiliki kesamaan yaitu menggunakan kelemahan dari system pendengaran manusia. Maka dari itu teknik/metode Steganografi dalam MP3 juga akan menggunakan kelemahan untuk menyembunyikan pesan. Yang membedakan adalah cara kerja dari masing-masing teknik/metode Steganografi.

3.3 Analisis Algoritma Rijndael

Algoritma Rijndael menggunakan tahapan transformasi SubBytes, ShiftRows, MixColumns dan AddRoundKey, menghasilkan difusi dan konfusi

yang kuat. Substitusi yang dihasilkan dari transformasi SubBytes mengubah nilai byte berdasarkan nilai asli dan nilai yang ada di kotak-S. Proses tersebut dapat dikatakan sebagai transformasi intrabyte. Permutasi yang dihasilkan dari proses ShiftRows mempertukarkan byte tanpa mengubah nilai bit di dalam byte. Transformasi ini dapat dikatakan sebagai transformasi *byte-exchange* (pertukaran byte). Pada MixColumn akan dilakukan transformasi interbyte untuk mengubah nilai bit dalam byte. Tujuannya adalah untuk menghasilkan difusi pada tingkat bit.



Gambar 3.9 Skema Enkripsi Rijndael

- **Transformasi Subtitusi *Byte***

Dalam operasi ini, setiap *byte* yang akan dienkripsi disubtitusikan dengan nilai *byte* lain dengan menggunakan *S-box*. *S-box* dibuat dari *multiplicative inverse* dari angka yang diberikan dalam *Rijndael's finite field* yang kemudian ditransformasikan dengan *affine transformation* :

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Hasilnya kemudian di-*xor* dengan 99_{10} atau $0x63_{16}$ atau 1100011_2. Operasi matriks dengan *xor* ini ekuivalen dengan persamaan:

$b'i = bi \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus C_i$

dengan *b'*, *b*, dan *c* adalah array 8 bit dan nilai *c* adalah 01100011.

Proses tersebut menghasilkan masing-masing nilai dari elemen tabel *S-box* yang hasilnya sebagai berikut :

	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

Seperti yang telah diketahui sebelumnya, AES merupakan algoritma simetri, yang berarti table subtitusi yang dibutuhkan untuk mengenkripsi berbeda dengan untuk mendekripsi. Untuk acuan tersebut, digunakanlah tabel *S-box* inverse sebagai berikut:

	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

Sebagai contoh, input yang akan dienkripsikan adalah

95	95	08	19
4f	6b	5c	6e
c8	89	80	26
fc	75	4e	6c

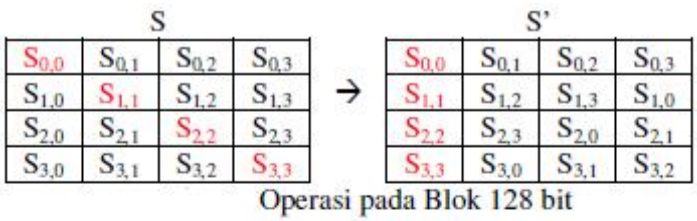
Dengan menggunakan *S-box*, hasil dari operasi ini adalah

2a	2a	30	d4
84	7f	4a	9f
e8	a7	cd	f7
b0	9d	2f	50

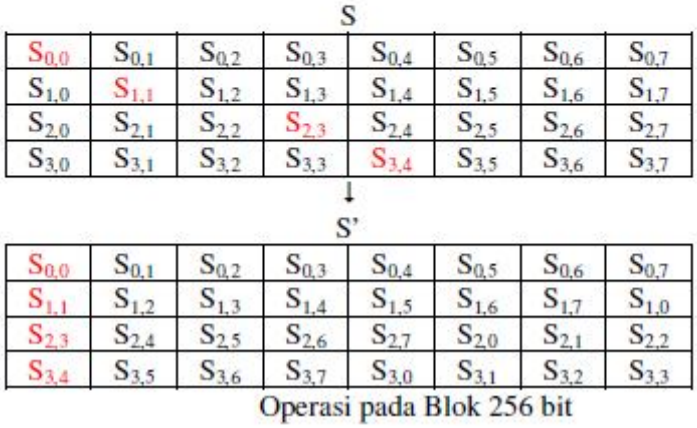
Jika hasil tersebut ingin dikembalikan ke nilai semula sebelum operasi, nilai-nilainya dapat disubtitusikan dengan menggunakan tabel *S-box* inversi. Operasi transformasi subtitusi *byte* pada proses enkripsi dan dekripsi tidak dilakukan pada putaran pertama. Operasi ini hanya dilakukan pada putaran kedua hingga terakhir.

- **Transformasi Pergeseran Baris**

Pada operasi ini, *byte-byte* pada setiap baris digeser secara memutar dengan pergeseran yang berbeda dari tiap-tiap baris. Setiap baris digeser dengan aturan tertentu untuk jenis panjang blok yang berbeda. Baris pertama blok untuk semua jenis panjang blok (128, 196, dan 256 bit) tidak digeser. Baris kedua untuk semua jenis panjang blok digeser 1 ke kiri. Pergeseran baris ketiga dan keempat untuk panjang blok 128 dan 196 bit berbeda dengan 256 bit. Pada panjang blok 128 dan 196 bit, baris ketiga digeser ke kiri sebanyak dua kali dan baris keempat digeser ke kiri sebanyak tiga kali. Pada panjang blok 256 bit, baris ketiga digeser ke kiri sebanyak tiga kali dan baris keempat digeser ke kiri sebanyak empat kali. Untuk lebih jelasnya, proses tersebut dapat dilihat sebagai berikut :



Operasi pada Blok 128 bit



Operasi pada Blok 256 bit

Sebagai contoh, hasil operasi ini terhadap input yang nilainya adalah output dari hasil operasi subtitusi *byte* sebelumnya adalah sebagai berikut :

2a	2a	30	d4
7f	4a	9f	84
cd	f7	e8	a7
50	b0	9d	2f

- **Transformasi Percampuran Kolom**

Transformasi ini mengoperasikan blok pada masing masing kolomnya. Setiap kolom diperlakukan sebagai *four-term polynomial* dengan cara Galois Field (GF) (2^8) dan dimodulokan dengan x^4+1 dengan polinom tetap $a(x)$, yaitu

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

Hal ini dapat dituliskan sebagai perkalian matriks sebagai berikut.:

$s'(x) = a(x) \oplus s(x)$

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

dengan c adalah letak kolom, sehingga hasilnya :

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

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

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

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

Jika hasil perkalian memiliki lebih dari 8 bit, bit yang lebih tidak begitu saja dibuang. Hasil tersebut di-*xor* dengan 100011011_2. Sebagai contoh, perkalian 11001010 dengan 11 dengan GF (2^8) akan berlangsung sebagai berikut :

11001010

11

---------- *

11001010

11001010

---------- xor

101011110

100011011

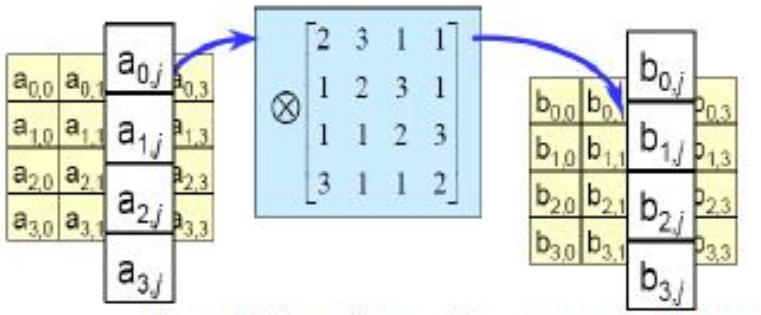
---------- xor

1000101

Nilai 1000101 merupakan hasil dari perkalian tersebut. Misalnya, jika dalam transfomasi ini input yang dipakai adalah hasil dari operasi pergeseran baris sebelumnya, hasil yang diperoleh adalah sebagai berikut :

48	cd	af	ac
c8	0c	ab	1a
24	5e	d8	74
6c	b8	06	fa

Transformasi ini dapat diilustrasikan sebagai berikut :



Ilustrasi Transformasi Percampuran Kolom

Operasi transformasi ini tidak digunakan dalam putaran terakhir, baik untuk enkripsi maupun dekripsi.

- **Transformasi Penambahan Kunci**

Dalam operasi transformasi ini, digunakanlah upakunci untuk masing-masing putaran yang berasal dari kunci utama dengan menggunakan jadwal kunci Rijndael (*Rijndael's key schedule*) yang ukuran upakunci tersebut sama dengan ukuran blok yang akan diproses. Upakunci tersebut kemudian di-*xor* dengan blok input sehingga diperoleh hasilnya.

Sebagai contoh, jika inputnya adalah :

a3	c5	08	08
78	a4	ff	d3
00	ff	36	36
28	5f	01	02

dan diperoleh upa kunci

36	8a	c0	f4
ed	cf	76	a6
08	a3	b6	78
31	31	27	6e

Maka, hasilnya adalah

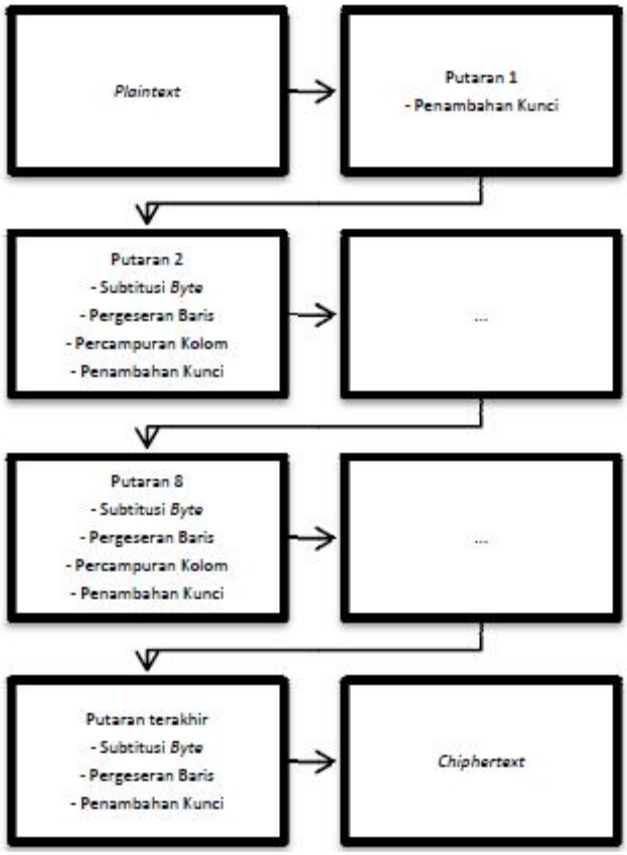
a6	34	1a	00
24	dd	f1	0e
62	a8	73	cf
48	b9	5d	61

- **Putaran**

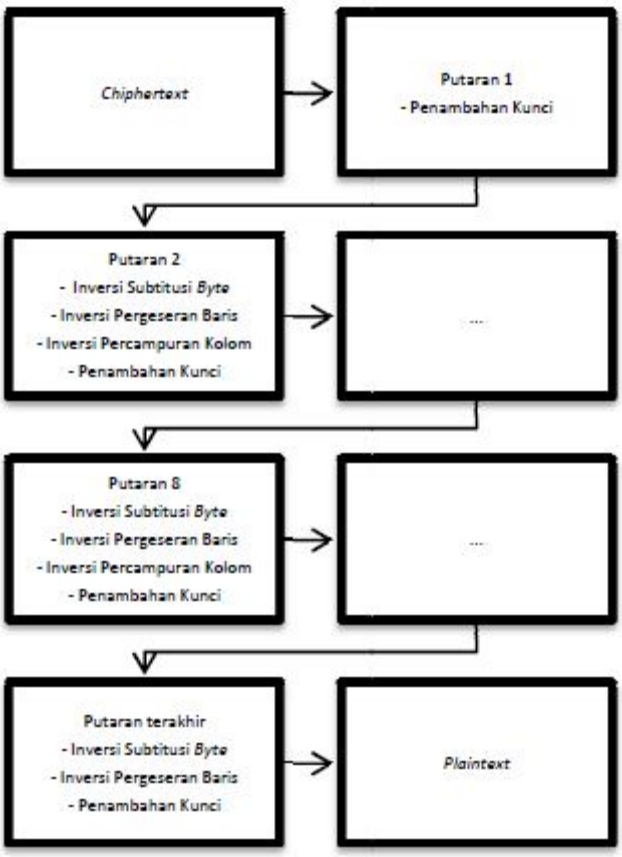
Pada mekanisme AddRoundKey, kunci Rijndael bersifat non linier yang disebabkan oleh transformasi SubWord, dan penggunaan Rcon pada AddRoundKey menghilangkan kesimetrian yang mungkin dihasilkan dari transformasi-transformasi sebelumnya. Bahkan jika semua bit pada kunci cipher utama adalah sama seperti terlihat dalam contoh berikut.

Pre-Round	:	00000000	00000000	00000000	00000000
Putaran ke-1	:	62636363	62636363	62636363	62636363
Putaran ke-2	:	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
Putaran ke-3	:	90973450	696CCFA	F2F45733	0B0FAC99
…		…	…	…	…
Putaran ke-10	:	B4EF5BCB	3E92E211	23E951CF	6F8F188E

Dari contoh di atas dapat kita lihat, nilai word pada pre-round adalah identik. Pada putaran pertama nilai word masih sama, namun setelah putaran ke-2 pola kesamaan tersebut hilang dan nilai setiap word berbeda satu sama lain.



Gambar 3.10 Diagram Proses Enkripsi



Gambar 3.11 Diagram Proses Dekripsi

Forouzan (2008, hal: 219) memberikan analisis terhadap algoritma Rijndael dengan parameter berupa keamanan, implementasi, kesederhanaan dan biaya.

1. Keamanan, sebagian besar serangan yang dikenal pada DES telah dicoba pada Rijndael, namun sejauh ini tidak ada satupun serangan yang dapat menembus keamanan Rijndael.
   a) *Brute force attack*, yaitu serangan yang mencoba semua kemungkinan kunci. Untuk serangan ini, Rijndael lebih aman dari DES karena ukuran kunci yang lebih besar (128, 192 dan 256 bit) jika dibandingkan dengan

DES dengan kunci 56 bit. Untuk DES dibutuhkan 2^{56} uji coba kunci, sedangkan Rijndael dengan kunci 128 bit membutuhkan 2^{128} uji coba.

b) Serangan analisis statistik, penggunaan difusi dan konfusi yang kuat yang dihasilkan dari kombinasi transformasi SubBytes, ShiftRows dan MixColumns menghilangkan pola pesan asli. Banyak uji coba yang gagal dalam melakukan serangan analisis statistik pada Rijndael.

c) Serangan diferensial dan linear, sejauh ini belum ada serangan diferensial maupun linear yang dapat menembus Rijndael.

2. Dalam hal implementasi, enkripsi dan dekripsi Rijndael sangat baik untuk semua platform yang ada, yang meliputi 8 bit dan 64 bit. Transformasi dapat dilakukan dengan berbasis byte atau word. Pada versi yang berbasis byte, algoritma tersebut dapat digunakan pada prosesor 8 bit. Pada versi yang berbasis word, dapat diterapkan pada prosesor 32 atau 64 bit. Rijndael tidak memakan banyak sumber daya prosesor karena mempunyai performa yang baik. Waktu *set-up* Rijndael sangat singkat. Enkripsi dan dekripsi menggunakan sedikit sumber daya RAM dan ROM. Rijndael membutuhkan sumber daya hardware yang minim.
3. Untuk kesederhanaan dan biaya, algoritma Rijndael sangat sederhana sehingga dapat dengan mudah diimplementasikan menggunakan prosesor yang murah dengan memori yang minimum.

Dari analisis sistem di atas, maka akan dibangun sebuah perangkat audio steganografi yang dikombinasikan dengan enkripsi menggunakan algoritma Rijndael dalam rangka mendapatkan sistem keamanan data yang lebih kuat. Dalam penelitian ini, proses pengenkripsian pesan memanfaatkan komponen DCPCrypt pada Borland Delphi 7. Perangkat lunak yang dihasilkan akan diuji apakah memenuhi aspek-aspek keamanan data pada steganografi, yaitu *fidelity*, *recovery*, *robustness* dan *security*.

3.4 Kebutuhan Fungsional Sistem

Dari analisis yang telah dilakukan di atas, perangkat lunak diharapkan mampu memenuhi kebutuhan-kebutuhan fungsional sebagai berikut:

1. Sistem dapat melakukan enkripsi terhadap pesan rahasia berformat teks dengan algoritma Rijndael. Kemudian pesan terenkripsi tersebut disisipkan ke dalam berkas audio berformat MP3. Dengan catatan, berkas MP3 masih dapat didengarkan dengan baik sesudah dilakukan penyisipan terhadapnya.
2. Sistem dapat mengungkapkan pesan rahasia yang telah disisipkan dalam sebuah berkas MP3 dan mengembalikannya ke dalam bentuk teks ataupun gambar seperti semula, serta isi pesan rahasia tersebut adalah sama dengan isi pesan sebelum disisipkan ke dalam MP3.

3.5 Kebutuhan Data Masukan dan Keluaran

Untuk memenuhi fungsi yang pertama, yaitu kemampuan dalam menyisipkan pesan, maka sistem memerlukan 3 bentuk masukan berupa kunci

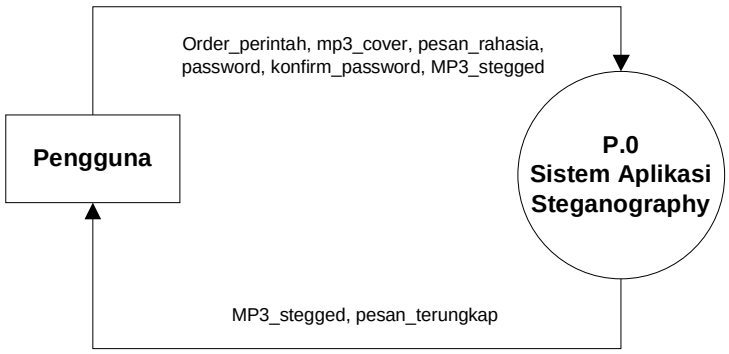
enkripsi, pesan rahasia dalam format teks (txt) atau gambar (jpg) serta berkas penampung berupa audio berformat MP3. Sedangkan untuk fungsi yang kedua yaitu kemampuan dalam mengekstraksi pesan dari dalam MP3, dibutuhkan masukan berupa kunci dekripsi (yang sama dengan kunci enkripsi) serta berkas MP3 yang sebelumnya telah mengalami proses penyisipan pesan oleh sistem. Keluaran dari fungsi penyisipan pesan berupa berkas MP3 yang didalamnya telah terkandung pesan rahasia. Sedangkan keluaran dari fungsi ekstraksi pesan adalah pesan rahasia dalam bentuk teks (txt) ataupun gambar (jpg).

3.6 Pemodelan Fungsional

Pemodelan fungsional adalah proses untuk mendeskripsikan seluruh fungsi yang terlibat di dalam perangkat lunak. Piranti yang digunakan untuk menjelaskan pemodelan fungsional ini adalah Diagram Konteks (*Context Diagram*) dan Diagram Aliran Data (*Data Flow Diagram)*.

3.6.1 Diagram Konteks

Diagram konteks adalah diagram yang menunjukkan keterhubungan antara perangkat lunak dengan konteks eksternal di luar program. Pada diagram ini, input berupa perintah diterima dari pengguna melalui *keyboard* dan *mouse*, sedangkan output akan ditampilkan kepada pengguna melalui monitor.



Gambar 3.12 Diagram Konteks

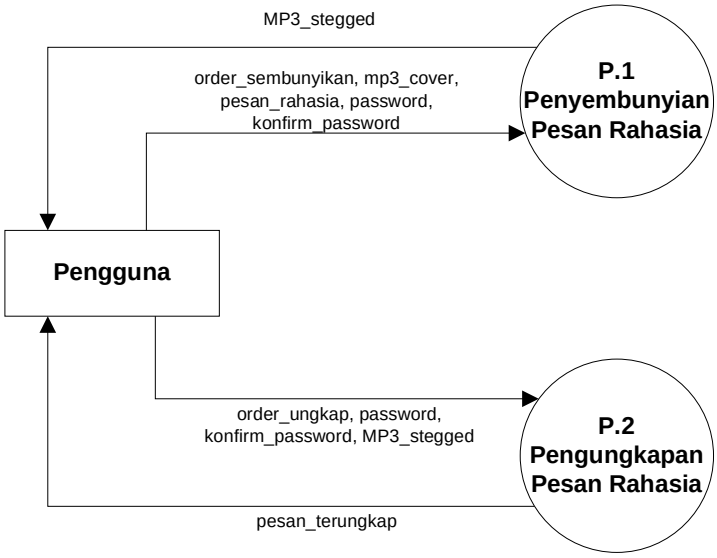
Tabel berikut adalah spesifikasi proses yang menjelaskan proses yang berlangsung dalam diagram konteks di atas.

Tabel 3.1 Spesifikasi Proses Diagram Konteks

No / Nama Proses	Input	Keterangan Proses	Output
P.0 / Sistem Aplikasi Steganography	order_perintah, pesan_rahasia, password, konfirm_password, MP3_cover, MP3_stegged	Sistem keamanan data yang dapat menyembunyikan pesan rahasia ke dalam MP3 serta dapat mengungkapkan kembali pesan rahasia tersebut.	MP3_stegged, pesan_terungkap

3.6.2 *Data Flow Diagram* Level 1

Dari diagram konteks sebelumnya, maka selanjutnya proses diuraikan menjadi bagian-bagian yang lebih kecil. Dua diantaranya merupakan proses inti dalam sistem keamanan ini, yaitu proses penyembunyian pesan dan proses pengungkapan pesan. Proses tersebut dapat diuraikan dengan DFD level 1 berikut.



Gambar 3.13 *Data Flow Diagram Level* 1

Tabel berikut adalah spesifikasi proses yang menjelaskan proses yang berlangsung dalam DFD level 1 di atas.

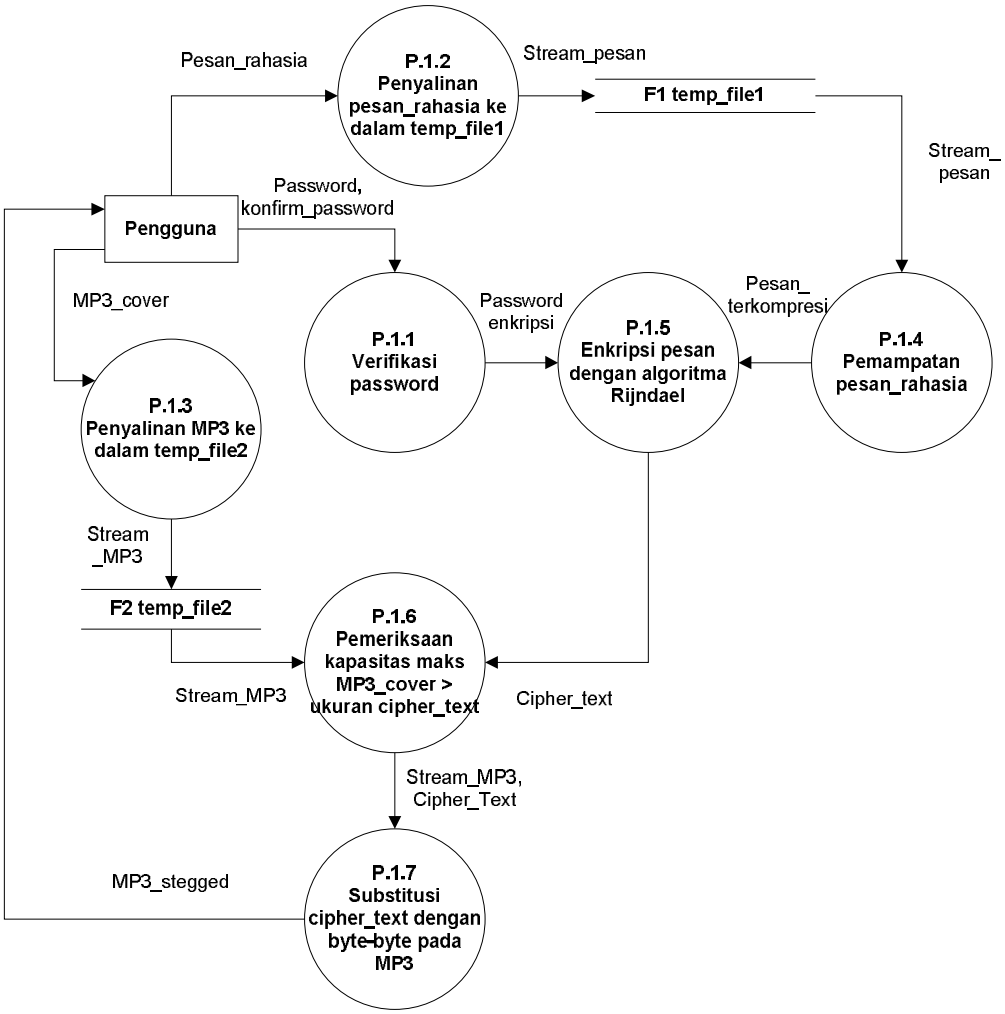
Tabel 3.2 Spesifikasi Proses DFD Level 1

No / Nama Proses	Input	Keterangan Proses	Output
P.1 / Proses penyembunyian/ penyisipan pesan	order_sembunyikan, mp3_cover, pesan_rahasia,	Menyisipkan pesan rahasia ke dalam berkas MP3	MP3_stegged

	password, konfirm_ password		
P.2 / Proses pengungkapan pesan	order_ungkap, password, konfirm_ password, MP3_stegged	Mengungkapkan pesan rahasia dari dalam berkas MP3 hasil steganografi	pesan_terungkap

3.6.3 *Data Flow Diagram* Level 2 : Penyembunyian Pesan

Pada diagram level selanjutnya menggambarkan secara khusus tentang proses penyembunyian pesan rahasia ke dalam berkas MP3. Input pada proses ini adalah kunci yang akan digunakan sebagai kunci enkripsi, pesan rahasia berupa teks berekstensi *txt* / *jpg* dan berkas MP3 yang akan digunakan sebagai media pembawa pesan. Keluaran yang dihasilkan pada proses ini berupa berkas MP3 yang telah disisipkan pesan rahasia di dalamnya.



Gambar 3.14 ***Data Flow Diagram Level*** **2, Proses P.1 : Penyembunyian Pesan**

Tabel berikut adalah spesifikasi proses yang menjelaskan proses yang berlangsung dalam DFD level 2 di atas.

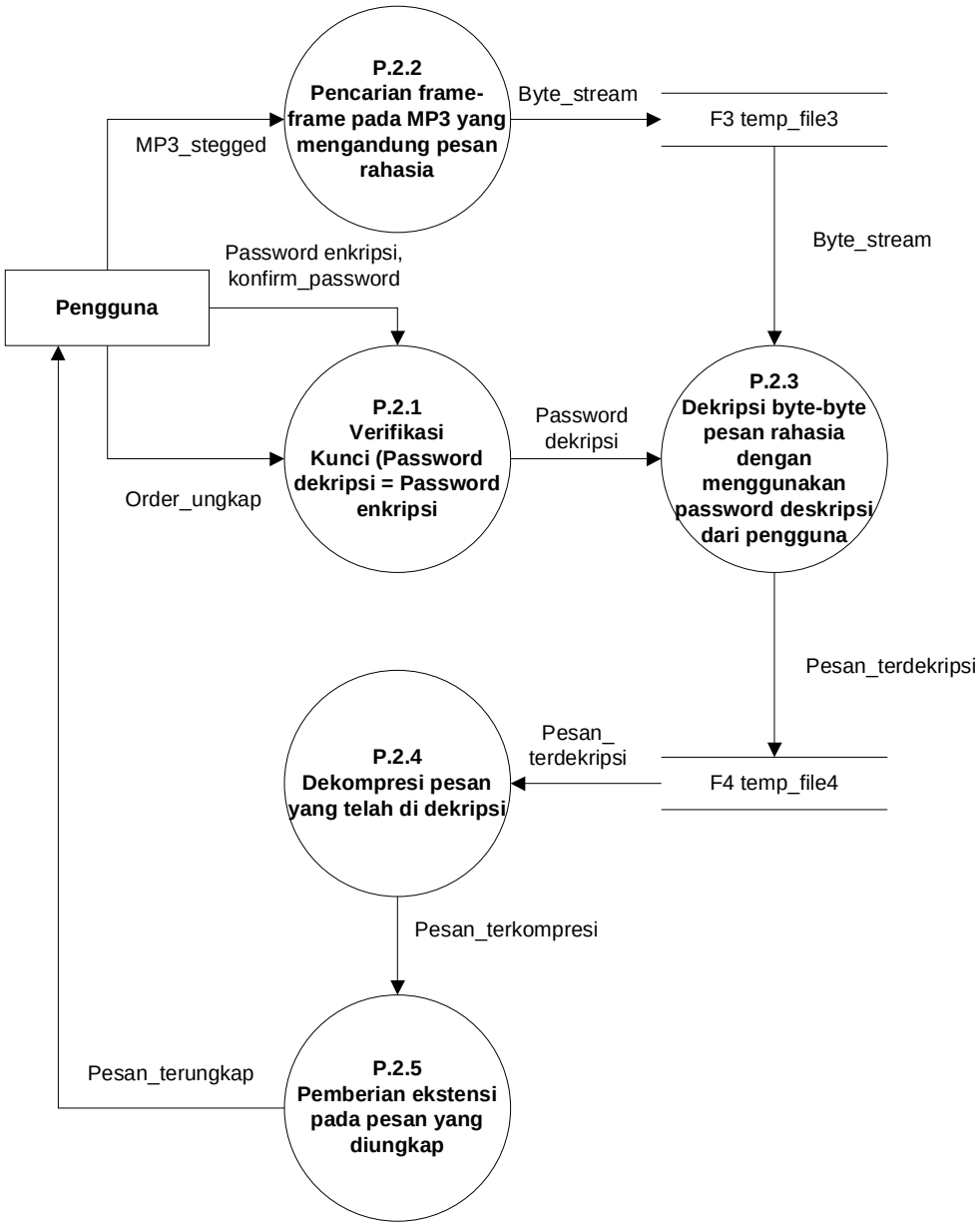
Tabel 3.3 Spesifikasi Proses DFD Level 2, Proses P.1 : Penyembunyian Pesan

No / Nama Proses	Input	Keterangan Proses	Output
P.1.1 / Verifikasi password	order_sembunyikan, password, konfirm_ password	Verifikasi password yang diberikan oleh pengguna	password
P.1.2 / Penyalinan pesan_rahasia ke dalam temporary_file1	pesan_rahasia	Menyalin pesan rahasia ke dalam file sementara	stream_pesan
P.1.3 / Penyalinan MP3 ke dalam temporary_file2	MP3_cover	Menyalin MP3 ke dalam file sementara	stream_MP3
P.1.4 / Kompresi pesan rahasia	stream_pesan	Memampatkan pesan rahasia dengan pustaka Zlib pada Borland Delphi 7	pesan_ terkompresi
P.1.5 / Enkripsi pesan dengan algoritma Rijndael	kunci, pesan_terkompresi	Enkripsi pesan rahasia terkompresi dengan Algoritma Rijndael menggunakan	cipher_text

		komponen DCPCrypt pada Borland Delphi 7	
P.1.6 / Pemeriksaan kapasitas maksimal MP3_cover > ciphertext	stream_MP3, cipher_text	Membandingkan kapasitas maksimal yang dapat disisipkan pada MP3 cover dengan ukuran cipher text	stream_MP3 cipher_text
P.1.7 / Penggantian cipher_text dengan byte-byte pada MP3	stream_ MP3, cipher_text	Menyisipkan cipher text ke dalam MP3 dengan mengganti byte-byte pada MP3 dengan byte-byte cipher text. (Parity Coding)	MP3_ stegged

3.6.4 *Data Flow Diagram* Level 2 : Pengungkapan Pesan

Proses lainnya dalam DFD level 2 ini adalah proses pengungkapan pesan rahasia. Pada proses ini membutuhkan input dari pengguna berupa kunci yang akan dijadikan sebagai kunci dekripsi pesan, serta MP3 hasil steganografi yang didalamnya terdapat pesan rahasia.



Gambar 3.15 ***Data Flow Diagram Level*** **2, Proses P.2 : Pengungkapan Pesan**

Tabel berikut adalah spesifikasi proses yang menjelaskan proses yang berlangsung dalam DFD level 2 di atas.

Tabel 3.4 Spesifikasi Proses DFD Level 2, Proses P.2 : Pengungkapan Pesan

No / Nama Proses	Input	Keterangan Proses	Output
P.2.1 / Verifikasi Password	order_ungkap, password enkripsi, konfirm_password	Verifikasi password yang diberikan oleh pengguna (Password enkripsi = password dekripsi	password
P.2.2 / Pencarian frame-frame pada MP3 yang mengandung pesan rahasia	MP3_stegged	Pencarian byte-byte pesan rahasia pada setiap frame MP3_stegged, dengan menemukan frame-frame yang telah diberi tanda pada proses penyembunyian pesan	byte_stream
P.2.3 / Dekripsi byte-byte pesan rahasia dengan menggunakan kunci dari pengguna	password, byte_stream	Dekripsi byte byte pesan yang ditemukan dengan menggunakan kunci dekripsi.	pesan_terdekripsi

P.2.4 / Dekompresi pesan yang telah di dekripsi	pesan_terdekripsi	Dekompresi pesan yang telah didekripsi sehingga didapatkan pesan rahasia semula	pesan_ terdekompresi
P.2.5 / Pemberian ekstensi pada pesan yang diungkap	pesan_ terdekompresi	Memberikan ekstensi pada pesan yang telah berhasil diungkap.	pesan_terungkap

3.7 Perancangan Perangkat Lunak

Perancangan adalah tahap mengimplementasikan keseluruhan hasil analisis yang dilakukan sebelumnya untuk menentukan kondisi akhir yang diharapkan serta merumuskan langkah-langkah dalam mencapai kondisi akhir tersebut. Perancangan dijadikan sebagai panduan yang dapat dipahami dengan mudah, memberikan gambaran yang jelas dan lengkap mengenai perangkat lunak yang akan dibangun.

3.7.1 Perancangan Prosedural

Ada beberapa proses yang akan dilakukan dalam sistem keamanan data dengan mengkombinasikan enkripsi dan steganografi ini. Secara umum, inti dari sistem keamanan ini adalah dapat berfungsi untuk menyisipkan pesan rahasia dan

dapat mengungkapkan kembali pesan rahasia tersebut. Perancangan prosedural akan dibantu dengan menggunakan bagan alir (*flowchart*).

3.7.1.1 Proses Penyisipan Pesan.

Sistem ini memerlukan masukan dari pengguna berupa kunci yang akan digunakan sebagai kunci enkripsi, pesan rahasia dalam bentuk teks atau gambar serta berkas MP3 sebagai media pembawa pesan. Pengguna akan diminta mengkonfirmasi kunci untuk memastikan bahwa kunci yang diberikan pengguna sesuai. Untuk mendapatkan kapasitas yang maksimal, maka pesan rahasia berupa teks atau gambar akan dimampatkan terlebih dahulu dengan menggunakan pustaka Zlib pada Delphi 7. Setelah itu, maka pesan rahasia yang telah dimampatkan tersebut akan di enkripsi dengan algoritma Rijndael memanfaatkan komponen DCPCrypt pada Delphi 7.

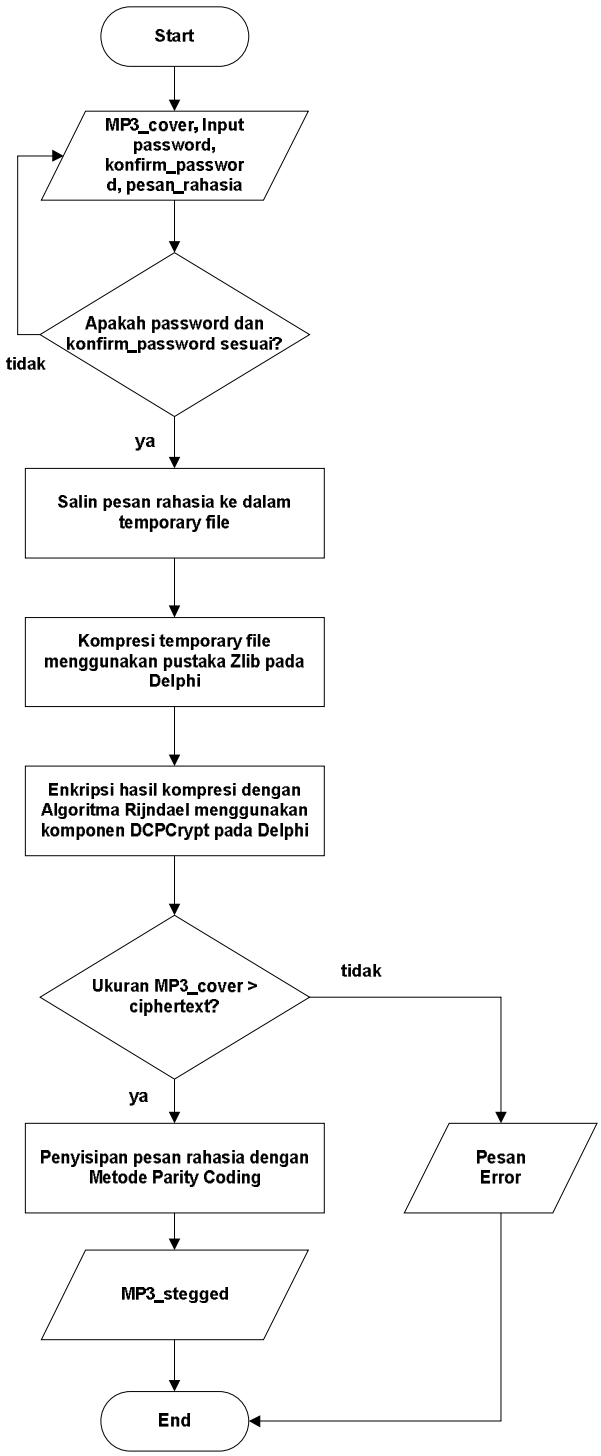
Pada MP3 yang akan dijadikan sebagai media pembawa pesan, akan dihitung kapasitas maksimal yang mampu ditampungnya. Berkas MP3 terdiri dari banyak *frame.* Jika byte-byte dalam sebuah *frame* MP3 adalah sama (homogen), maka *frame* tersebut dapat dijadikan sebagai tempat disisipkannya pesan rahasia, dengan mengganti *byte-byte* dalam *frame* MP3 tersebut dengan byte-byte *ciphertext* tanpa mempengaruhi kualitas dan besar MP3. Pada penelitian ini, berkas MP3 yang digunakan adalah berkas yang tidak menggunakan CRC, sehingga *byte-byte* homogen yang digunakan adalah setelah *byte* yang ke-36 dari posisi *frame header* ditemukan.

Setelah diperoleh kapasitas maksimal yang mampu ditampung oleh MP3, maka akan dibandingkan antara besar *ciphertext* (pesan rahasia yang telah dienkripsi) dengan kapasitas maksimal yang dapat ditampung MP3. Jika besar *ciphertext* melampaui kapasitas maksimal yang dapat ditampung MP3, maka akan ditampilkan pesan *error* dan proses dihentikan. Sebaliknya, jika besar *ciphertext* kurang dari atau sama dengan kapasitas maksimal yang dapat ditampung MP3, maka akan dilakukan proses penyisipan *cipher text* ke dalam *frame-frame* MP3, dengan cara mengganti *byte-byte* homogen pada *frame* MP3 dengan *ciphertext* tersebut.

```
Proses Penyisipan Pesan
Algoritma :
   begin
      input (kunci, konfirm_kunci, pesan_rahasia, MP3_cover)
         if (kunci = konfirm_kunci)
         begin
             temp_file1 ← pesan_rahasia
             temp_file2 ← MP3_cover
             temp_file1 ← kompresi(temp_file1)
             temp_file1 ← enkripsi(temp_file1)
             if (temp_file2 > temp_file1)
             begin
               ciphertext ← hide_file(temp_file2, temp_file1)


           end
         end
    end.
```

Berikut ini akan digambarkan diagram alir (*flowchart*) proses penyisipan pesan rahasia ke dalam berkas MP3.



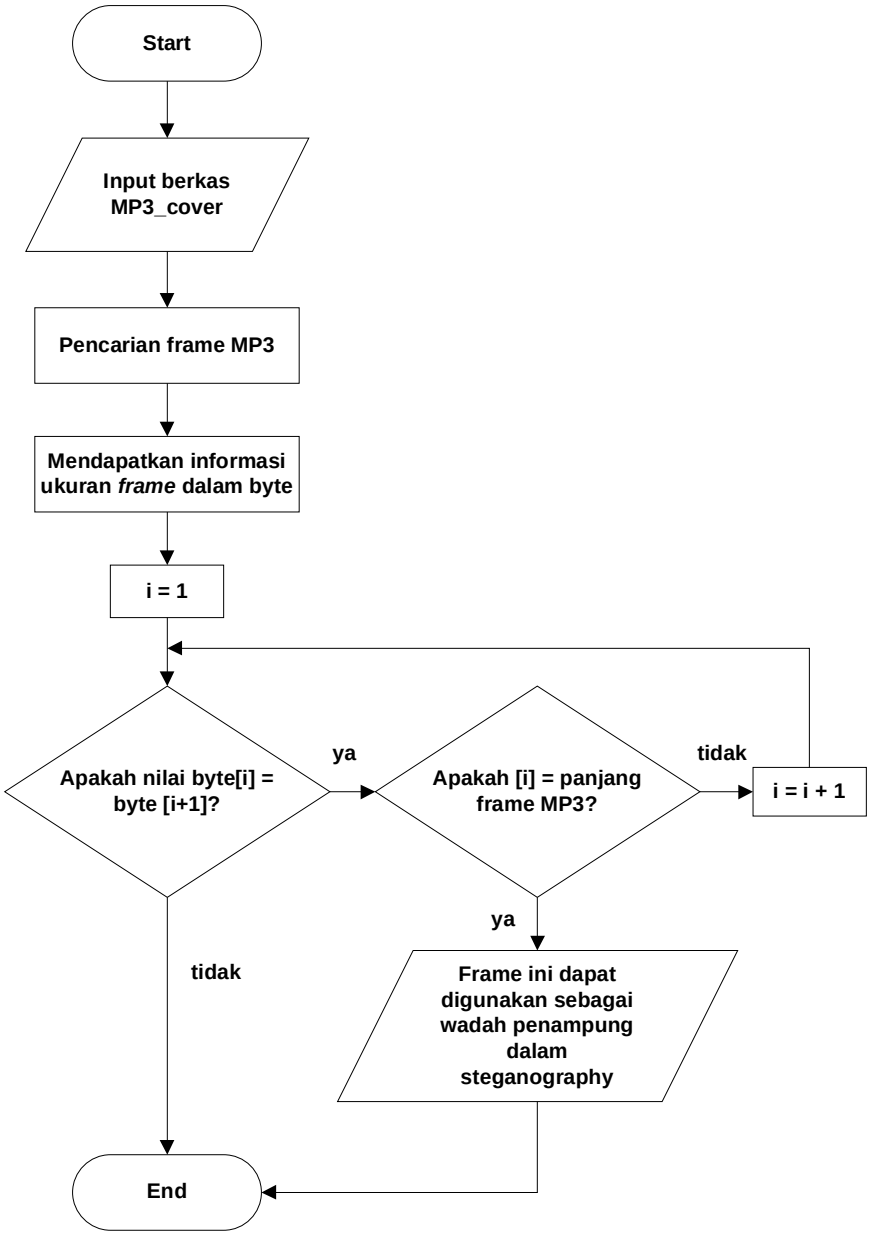
Gambar 3.16 *Flowchart* Proses Penymbunyian Pesan Secara Umum

1. **Proses Pencarian Byte-Byte Homogen pada MP3.**

Pada proses ini program akan membaca *stream file* MP3 dimulai dari awal sampai akhir *file*. Pembacaan ini bertujuan untuk mendapatkan byte-byte homogen, dengan memeriksa apakah byte yang sekarang dibaca adalah sama dengan 1 byte berikutnya. Pembacaan dilakukan dengan membaca nilai *stream* tersebut setiap 2 byte dan pemeriksaan ini dilakukan hingga mencapai akhir *file*. Jika hasil pemeriksaan pada satu *frame* tersebut adalah sama maka *frame* tersebut dapat digunakan untuk steganografi.

```
Proses Pencarian Byte-byte Homogen MP3
Algoritma :
   begin
      baca(MP3_cover)
      frame_size ← hitung ukuran frame
      i ← 1
      while (i=1 to frame_size)
            if (byte[i] = byte[i+1])
                  if (i = frame_size)
                  begin
                        output('Frame dapat digunakan');
                  end
                  else begin
                        i = i + 1;
                  end;
      end.
```

Berikut adalah *flowchart* yang menggambarkan proses pencarian byte-byte homogen pada MP3.



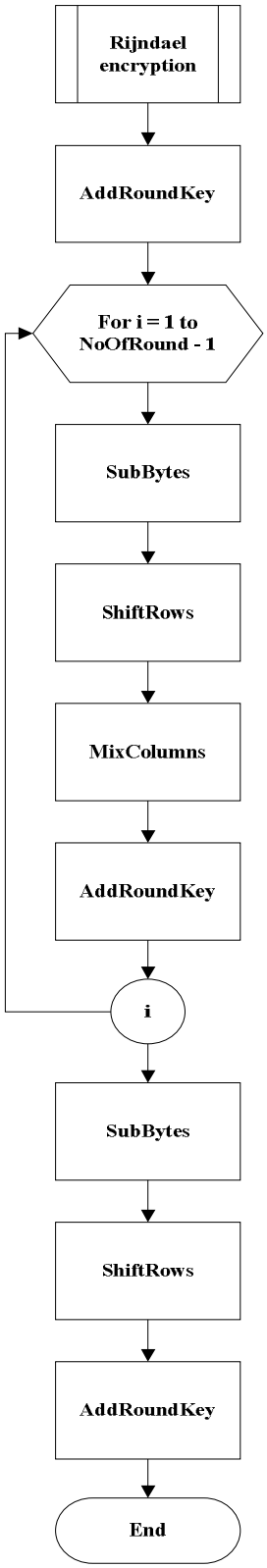
Gambar 3.17 Proses Pencarian Byte-byte Homogen MP3

2. Analisa Proses *Rijndael*

Algoritma *Rijndael* yang dipakai menggunakan kunci sepanjang 256 *bit* dan besar blok 4 (128 *bit*), sehingga Nr atau jumlah *round* yang dipakai adalah 14.

Panjang dari kunci dipilih sebesar 256 *bit* untuk menunjang tingkat keamanan yang tinggi. Program pertama kali akan menerima *inputan* berupa *file* dan kunci. Kunci akan digunakan untuk melakukan ekspansi sehingga dihasilkan nilai w sebanyak 4 * (14 + 1) = 60 *word*. Setelah itu *file* akan dibaca tiap 128 *byte* yang akan dimasukkan ke dalam variabel *state*. Jika data kurang dari 128 *byte* maka data akan ditambahkan dengan *bit* 0 hingga mencapai 128 *byte*.

Perancangan untuk sistem enkripsi *rijndael* adalah sebagai berikut:

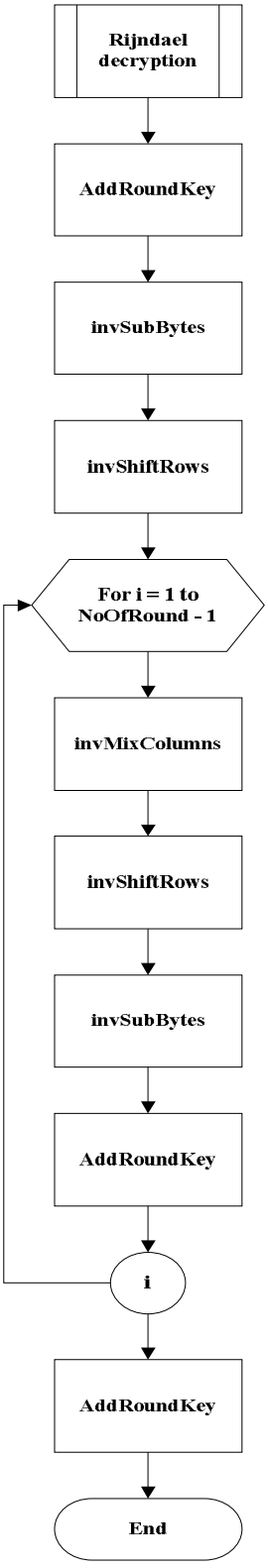


Gambar 3.18 ***Flowchart Rijndael Encryption***

Keterangan:

- *System* melakukan proses AddRoundKey, yaitu dengan cara melakukan EXOR nilai yang ada pada *state* dengan nilai dari *key schedule* (w).
- *System* melakukan proses SubBytes dengan cara mengganti isi dari tiap *byte* dari tiap *byte* pada state dengan nilai dari Sbox.
- *System* melakukan proses ShiftRows dengan cara melakukan operasi perkalian antara nilai c(x) dengan nilai dari state, untuk operasi perkalian disusun dari operasi perkalian yang lebih kecil.

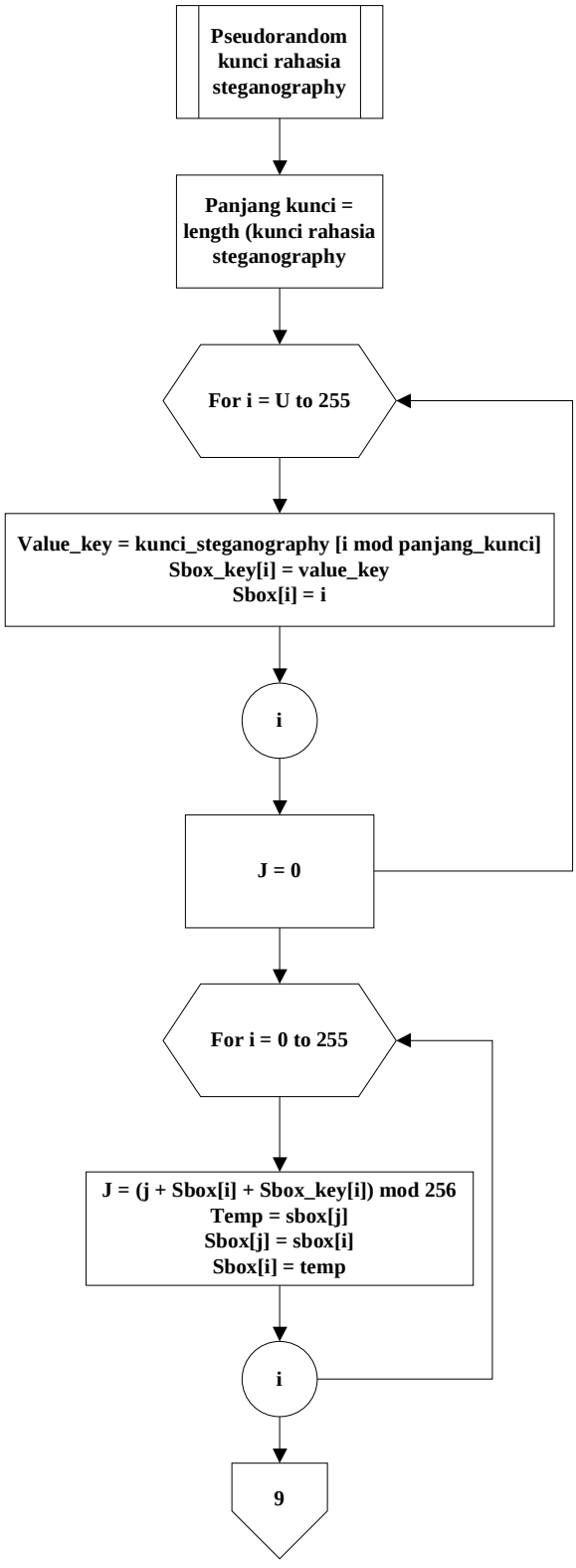
Pada proses deskripsi *rijndael* dilakukan proses yang terbaik dari proses enkripsinya, berikut adalah *flowchart* untuk sistem deskripsi *rijndael*:



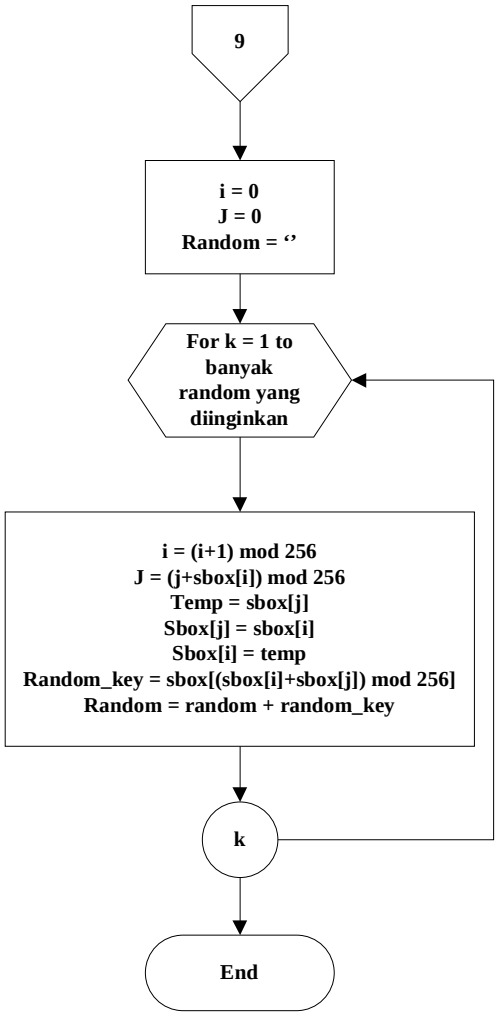
Gambar 3.19 ***Flowchart Rijndael Decryption***

3. Sistem *Pseudorandom Number Generator*

Pseudorandom number generator merupakan suatu algoritma yang digunakan untuk menghasilkan nilai *random*. Algoritma tersebut sebenarnya menghasilkan nilai *random* dengan perhitungan khusus, sehingga nilai yang dihasilkan bukan semata-mata nilai *random* yang muncul. Perancangan sistem untuk membuat *pseudorandom number generator*, dapat dilihat dari *flowchart* berikut:



Gambar 3.20 Diagram alir proses *pseudorandom number generator*



Gambar 3.21 Diagram alir proses *pseudorandom number generator* (lanjutan)

Keterangan:

- Sistem membuat S-Box berdasarkan kunci rahasia steganography, yaitu dengan cara:
    1. Sistem melakukan proses looping dari 0 sampai 255, yang melaksanakan proses:

       value_key = kunci_steganography[i mod length(kunci_steganography)].

S-BoxKey[i] = value_key

S-Box[i] = i

2. Sistem deklarasi nilai j = 0.
3. Sistem melakukan proses looping dari 0 sampai 255, yang melaksanakan proses:

   j = (j + S-Box[i] + S-BoxKey[i]) mod 256.

   swap(S-Box[i], S-Box[j]).

- Setelah sistem berhasil membuat S-Box, maka langkah selanjutnya yaitu mengambil random number dari S-Box yang telah dibuat, yaitu dengan cara:

4. Sistem deklarasi nilai i = 0, j = 0, dan mengosongkan isi variable random.
5. Sistem kemudian melakukan proses looping dari 1 sampai banyak nilai random yang ingin dipakai, yang meliputi proses berikut:

   i = (i +1) mod 256.

   j = (j + S-Box[i]) mod 256.

   swap(S-Box[i], S-Box[j]).

   random_key = S-Box[(S-Box[i]+S-Box[j]) mod 256].

   random = random + random_key.

- Sistem mengembalikan nilai random.

4. Analisa Proses Kompresi – Dekompresi (Zlib)

Library yang menangani kompresi data *lossless*. Tanpa library ini, pengguna tidak dapat mengkompresi data menyembunyikan data rahasia ataupun membuka stegodata yang terkompresi.

Dua class utama pada unit ZLib adalah TCompressionStream dan TDecompressionStream. TCompressionStream digunakan untuk kompres data. Data yang dikompres ditulis ke stream yang nantinya akan dilewatkan ke constructor TCompressionStream. Kita dapat menentukan level kompresi pada saat membuat obyek stream. Level kompresi yang disediakan adalah clNone, clFastest, clDefault, dan clMax. Untuk dekompres data kita menggunakan class TDecompressionStream.

Skrip dibawah ini untuk kompres dan dekompres file. Tidak hanya satu file saja, tapi bisa lebih dari satu. Variabel yang menyimpan nama-nama file bertipe TStrings. File kompresi ini berisi jumlah file, nama file asal, panjang nama file asal, ukuran file dan isi dari file asal.

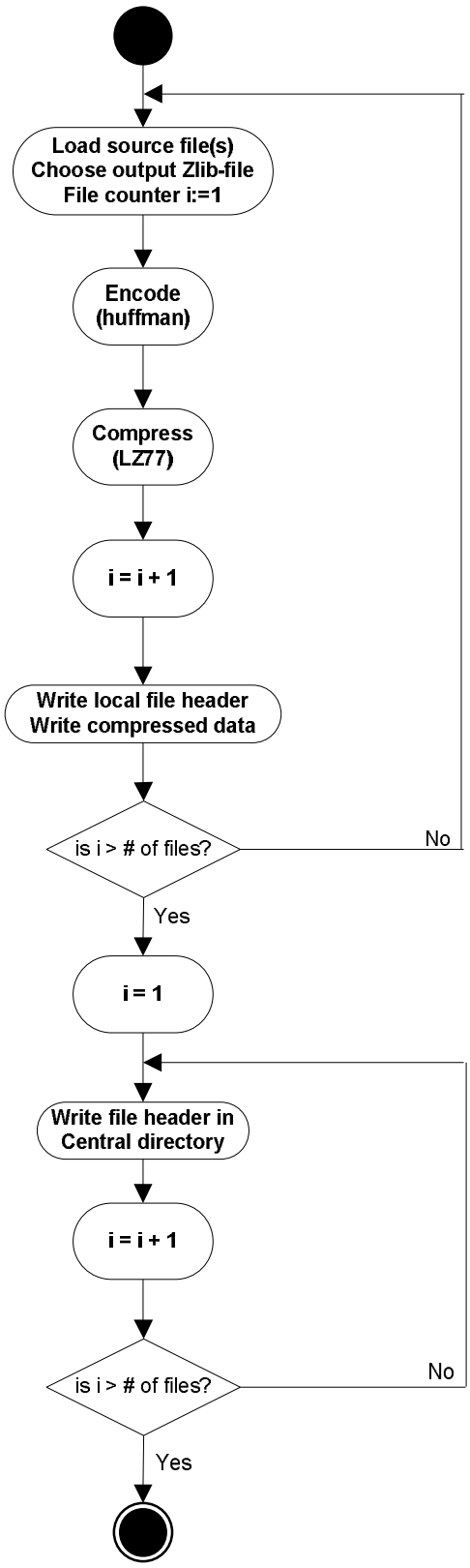
```
//Prosedur Proses Kompresi

   uses Classes, SysUtils, ZLib;

procedure CompressFiles(aFiles: TStrings; const aOutFileName:
string; aCompressionLevel: TCompressionLevel);
var
  infile, outfile, tmpfile: TFileStream;
  compr: TCompressionStream;
  i, j: Integer;
```

```
  s: string;
begin
  outfile := TFileStream.Create(aOutFileName, fmCreate);
  try
    j := aFiles.Count; // jumlah file
    outfile.Write(j, SizeOf(j));
    for i := 0 to aFiles.Count - 1 do
    begin
      infile := TFileStream.Create(aFiles[i], fmOpenRead);
      try
        s := ExtractFilename(aFiles[i]); // nama file asal
        j := Length(s); // panjang nama file asal
        outfile.Write(j, SizeOf(j));
        outfile.Write(s[1], j);
        j := infile.Size; // ukuran file
        outfile.Write(j, SizeOf(j));
        if j = 0 then Continue;
        tmpfile := TFileStream.Create('tmp', fmCreate);
        compr := TCompressionStream.Create(aCompressionLevel,
tmpfile);
        try
          compr.CopyFrom(infile, j);
        finally
          compr.Free;
          tmpfile.Free;
        end;
        tmpfile := TFileStream.Create('tmp', fmOpenRead); //
tambahkan file terkopmpresi ke file tujuan
```

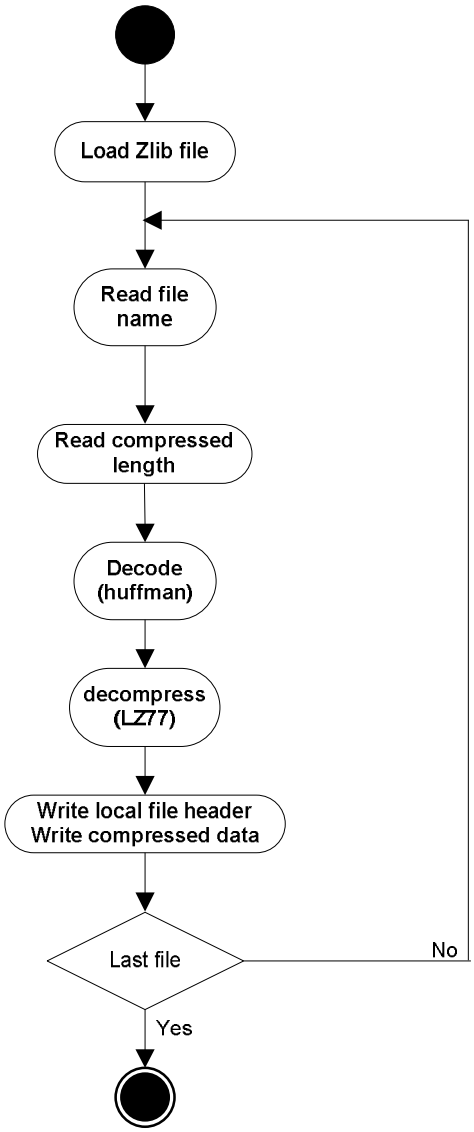
```
        try
          outfile.CopyFrom(tmpFile, 0);
        finally
          tmpfile.Free;
        end;
      finally
        infile.Free;
      end;
    end;
  finally
    outfile.Free;
    DeleteFile('tmp');
  end;
end;
```



Gambar 3.22 *Flowchart* Kompresi Zlib

```
//Prosedur Proses Dekompresi
procedure DecompressFiles(const aFileName, aDestDirectory:
string);
var
  s: string;
  decompr: TDecompressionStream;
  infile, outfile: TFileStream;
  i, j, count: Integer;
begin
  infile := TFileStream.Create(aFileName, fmOpenRead);
  try
    infile.Read(count, SizeOf(count)); // jumlah file
    for i := 1 to count do
    begin
      infile.Read(j, SizeOf(j)); // panjang nama file asal
      SetLength(s, j);
      infile.Read(s[1], j); // nama file asal
      infile.Read(j, SizeOf(j)); // ukuran file
      s := aDestDirectory + '\' + s;
      outfile := TFileStream.Create(s, fmCreate);
      decompr := TDecompressionStream.Create(infile);
      try
        if j = 0 then Continue; // antisipasi jika ukuran file = 0
        outfile.CopyFrom(decompr, j);
      finally
        outfile.Free;
        decompr.Free;
      end;
```

```
    end;
  finally
    infile.Free;
  end;
end;
```



Gambar 3.23 *Flowchart* Dekompresi Zlib

3.7.1.2 Proses Pengungkapan Pesan.

Pada proses pengungkapan pesan, dibutuhkan input berupa kunci dan berkas MP3 keluaran dari sistem steganografi. Pada berkas MP3 tersebut akan dilakukan pencarian untuk menemukan *frame* pertama yang memuat string penanda berupa “STEG”. Kemudian akan didapatkan ukuran pesan rahasia yang disisipkan dalam MP3 tersebut.

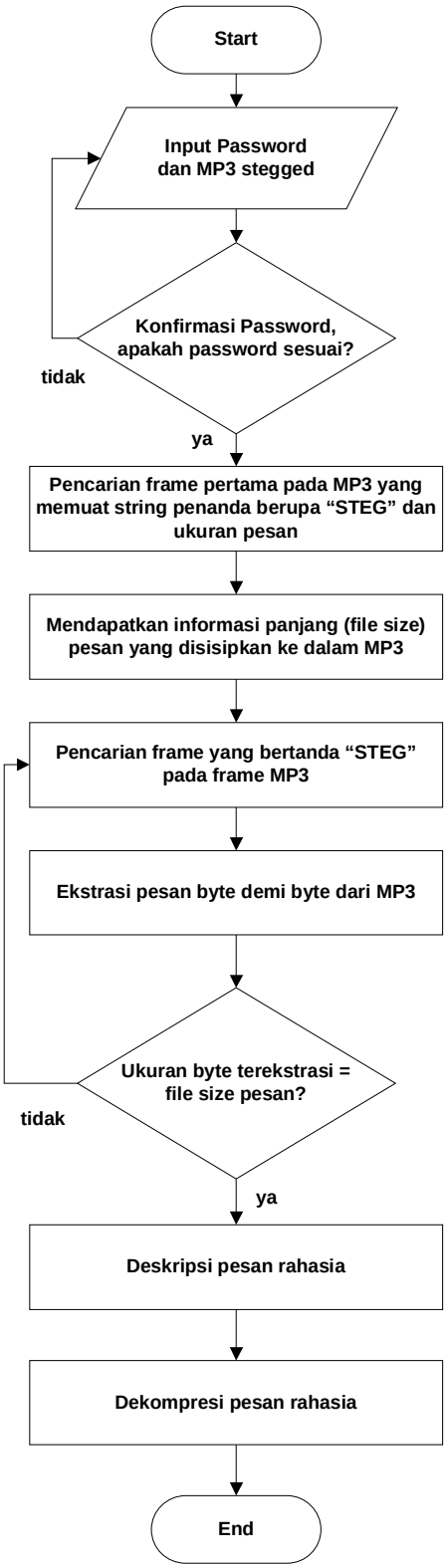
Setelah itu sistem akan memeriksa setiap *frame* dalam MP3 untuk menemukan *frame-frame* yang bertanda, yang berarti dalam *frame* tersebut tersimpan *byte-byte* pesan rahasia, Sistem akan terus mencari sampai ukuran *byte-byte* pesan yang berhasil ditemukan sama dengan panjang pesan (*file size)*.

Setelah semua *byte* pesan didapatkan, maka proses selanjutnya adalah dengan mendekripsi pesan dengan kunci enkripsi berupa kunci yang dimasukkan oleh pengguna. Langkah terakhir adalah dengan melakukan dekompresi terhadap pesan terdekripsi, sehingga didapatkan pesan rahasia dalam bentuk *file text.* Secara umum, proses pengungkapan pesan rahasia pada sistem keamanan ini dapat dilihat pada algoritma berikut.

```
Proses Penyisipan Pesan
Algoritma :
     begin
          input (kunci, konfirm_kunci, MP3_stegged)
          if (kunci = konfirm_kunci)
          begin
               cari(string_penanda);
```

```
                baca(ukuran_pesan);
                if  (byte[n]+byte[n  +  1]+byte[n  +  2]  +
                byte[n + 3] = 'STEG')
                begin
                      temp_file = 0;
                      for temp_file = 0 to ukuran_pesan
                      begin
                       temp_file ← ekstraksi_byte;
                      end;
                end;
                dekripsi(temp_file);
                dekompresi (temp_file);
          end
     end.
```

Flowchart proses pengungkapan pesan pada aplikasi ini dapat dilihat sebagai berikut.



Gambar 3.24 *Flowchart* proses pengungkapan pesan

3.7.2 Perancangan Antarmuka

Beberapa prinsip dari perancangan antarmuka yang baik telah dikembangkan dan banyak di antaranya menekankan pada antarmuka yang *user friendly* sehingga mudah digunakan. Penelitian pada bidang ini sering disebut sebagai *human factors engineering* atau *ergonomics.* Berikut adalah elemen-elemen yang harus dipertimbangkan dalam perancangan antarmuka.

1. Desain layar

Beberapa hal yang perlu dipertimbangkan dalam desain layar yaitu mengenai warna serta desain grafis yang digunakan pada antarmuka sistem dengan mengedepankan aspek *ergonomic* dan kenyamanan pengguna.

2. Umpan balik

Aspek yang paling penting dalam umpan balik (*feed back)* adalah waktu respon, yaitu waktu antara saat *user* memasukkan data dengan respon yang diberikan oleh sistem.

3. Bantuan

Pada saat *user* sedang mengoperasikan sistem, seringkali mengalami kesulitan dan tidak mengetahi apa yang harus dikerjakan berikutnya. Perancangan antarmuka Bantuan menyediakan langkah-langkah dalam menggunakan aplikasi.

4. Pengendalian kesalahan

Perancangan antarmuka harus memperhatikan pengendalian kesalahan yang dapat berupa sebagai berikut.

a. Pencegahan kesalahan

Sistem harus menyediakan instruksi yang jelas kepada pengguna tentang apa yang harus dilakukan sehingga pengguna tidak melakukan kesalahan yang seharusnya tidak perlu terjadi.

b. Pendeteksian kesalahan.

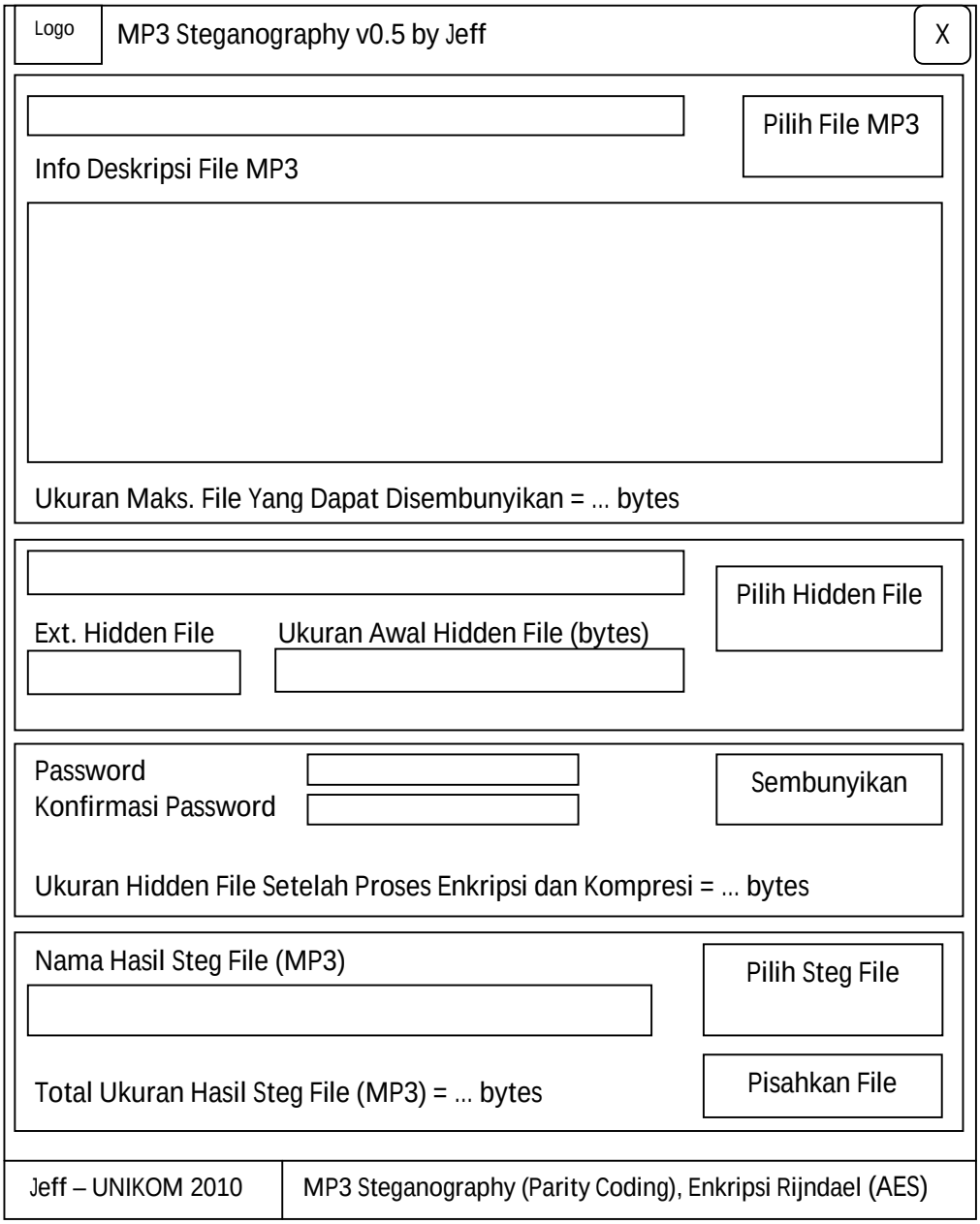
Jika suatu kesalahan terjadi, system harus dapat mengidentifikasi kesalahannya dengan jelas dan dapat menampilkan berita kesalahan ini, misalnya “Password tidak sesuai!”, atau “Fatal Error! Sistem dihentikan”.

c. Pembetulan kesalahan.

Jika suatu data yang dimasukkan salah, maka sistem harus dapat memberikan kesempatan kepada *user* untuk dapat mengoreksinya.

3.7.2.1 Tampilan Program

Tampilan program yang diberi nama “mp3STEG” ini hanya terdiri dari satu tampilan saja, dengan tampilan yang sederhana, sehingga memberi kesan mudah digunakan oleh para pengguna. Semua tombol-tombol perintah berikut dengan keterangan-keterangan hasil dari pengeksekusian tersaji dengan rapih dalam tampilan program ini. Untuk lebih jelasnya, dapat dilihat gambar perancangannya di bawah ini:



Gambar 3.25 Tampilan Program "mp3STEG"