PENYISIPAN PESAN TERENKRIPSI PADA *FILE AUDIO* MP3 MENGGUNAKAN METODE *LEAST SIGNIFICANT BIT (LSB)*

SKRIPSI

Oleh:
Nur Hadi Sulaiman
0810963058-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2012

UNIVERSITAS BRAWIJAYA

PENYISIPAN PESAN TERENKRIPSI PADA *FILE AUDIO* MP3 MENGGUNAKAN METODE *LEAST SIGNIFICANT BIT (LSB)*

SKRIPSI

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

Oleh:
Nur Hadi Sulaiman
0810963058-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2012

UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

Penyisipan Pesan Terenkripsi Pada File *Audio* MP3 Menggunakan Metode *Least Significant Bit (LSB)*

Oleh :

Nur Hadi Sulaiman
0810963058-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 6 Agustus 2012

dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	Pembimbing II,
Edy Santoso, SSi., M.Kom	Lailil Muflikhah, S.Kom., MSc
NIP. 197404142003121004	NIP. 197411132005012001

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP.196709071992031001


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	: Nur Hadi Sulaiman
NIM	: 0810963058-96
Jurusan	: Matematika
Program Studi	: Ilmu Komputer
Penulis skripsi berjudul	: Penyisipan Pesan Terenkripsi Pada File *Audio* MP3 Menggunakan Metode *Least Significant Bit (LSB)*

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, 6 Agustus 2012
Yang menyatakan,

Nur Hadi Sulaiman
NIM. 0810963058-96


UNIVERSITAS BRAWIJAYA

PENYISIPAN PESAN TERENKRIPSI PADA *FILE AUDIO* MP3 MENGGUNAKAN METODE *LEAST SIGNIFICANT BIT (LSB)*

ABSTRAK

Untuk meningkatkan faktor keamanan yang menjadi permasalahan dalam kegiatan penyampaian pesan dan komunikasi data, dapat dilakukan dengan mengenkripsi pesan atau menyisipkan pesan ke dalam sebuah media *carrier*. Dalam penelitian ini kedua cara tersebut akan dilakukan. Data berupa teks akan dienkripsi menggunakan algoritma RSA kemudian hasil enkripsi disisipkan ke dalam *file* MP3 menggunakan metode *Least Significant Bit (LSB).* Metode *Least Significant Bit (LSB)* akan mengganti bit-bit yang kurang berarti dalam MP3 dengan bit-bit pesan. Proses penggantian bit dilakukan dengan mode *standart* dan mode per-50 *frame.* Hasil penelitian menunjukkan adanya penurunan kualitas MP3 yang didasarkan pada SNR, seiring dengan bertambahnya ukuran pesan terenkripsi yang disisipkan. Semakin besar ukuran pesan terenkripsi yang disisipkan, maka SNR semakin menurun. Nilai SNR yang dihasilkan dalam penelitian ini berkisar antara 29 dB - 39 dB sedangkan prosentase SNR berkisar 99%. Nilai MOS yang dihasilkan memiliki rata-rata nilai 3,05 untuk penyisipan dengan mode *standart* yang berarti terdapat derau yang sedikit menggangu, sedangkan untuk penyisipan dengan mode per-50 *frame* memiliki rata-rata 2,783 yang berarti terdapat derau mengganggu. Dari uji iterasi diketahui bahwa algoritma RSA dapat diserang menggunakan *Brute Force Attack (BFA).* Banyaknya iterasi BFA dipengaruhi oleh besarnya kunci publik pertama dan nilai minimum bilangan prima yang digunakan pada saat enkripsi. Akan tetapi faktor yang paling berpengaruh adalah nilai minimum bilangan prima. Semakin besar nilai minimum bilangan prima yang dipilih maka semakin banyak iterasi BFA yang dibutuhkan yang berarti hasil enkripsi semakin aman.

Kata Kunci : *Least Significant Bit (LSB),* RSA, SNR, MOS , *Brute Force Attack (BFA)*

INSERTION OF ENCRYPTED MESSAGE INTO MP3 AUDIO FILE USING LEAST SIGNIFICANT BIT (LSB) METHOD

ABSTRACT

To increase the security problem in the messaging activities and data communications, can be done by encrypting a message or insert messages into a carrier medium. In this research, both of methods will be conducted. Text data will be encrypted using the RSA algorithm and the result will be inserted into the MP3 file using the Least Significant Bit (LSB). Least Significant Bit (LSB) will replace the bits that are less meaningful in MP3 with the message bits. Bit replacing process conducted using Standard mode and each 50 frame mode. The result show there are decrease MP3 quality based from SNR, along with increasing the size of the encrypted message that inserted into MP3. The larger size of the encrypted message is inserted, the SNR decreases. SNR value that resulted in this research ranged between 29 dB - 39 dB while SNR range percentage about 99%. MOS values that resulted has an average value 3.05 for standard mode insertion, that mean there is a little disturbing noise, while for the each 50 frame mode insertion has average value 2.783, that mean there is a disturbing noise. From iteration test show that the RSA algorithm can be attacked using the Brute Force Attack (BFA). The number of BFA iteration affected by the magnitude of the first public key and the minimum value of prime numbers that used during the encryption. However, the most influential factor is the minimum value of prime numbers. The larger the minimum value of the selected prime number, the more BFA iterations needed, that mean safer encryption result.

Key Word : *Least Significant Bit (LSB),* RSA, SNR, MOS , *Brute Force Attack (BFA)*

KATA PENGANTAR

Puji syukur kehadirat Allah SWT atas segala limpahan rahmat dan hidayah-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul "Penyisipan Pesan Terenkripsi Pada File MP3 Menggunakan *Metode Least Significant Bit (LSB)".*

Skripsi ini merupakan salah satu syarat untuk memenuhi persyaratan akademis untuk menyelesaikan studi di program Sarjana Ilmu Komputer Universitas Brawijaya.

Selama penulisan skripsi ini, penulis mendapat bantuan, saran dan dukungan dari banyak pihak. Untuk itu, penulis ingin mengucapkan terima kasih kepada :

1. Edy Santoso, SSi., M.Kom., dan Lailil Muflikhah, S.Kom., M.Sc. selaku dosen pembimbing yang telah dengan bijaksana dan sabar membimbing penulis selama penyusunan skripsi ini.
2. Dr. Abdul Rouf Alghofari, M.Sc., selaku Ketua Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Brawijaya.
3. Drs. Marji, MT. selaku Ketua Program Studi Ilmu Komputer, Jurusan Matematika, FMIPA Universitas Brawijaya dan penasehat akademik.
4. Edy Santoso, SSi., M.Kom., selaku dosen pembimbing akademik atas nasehat dan bimbingan akademik yang telah diberikan selama penulis menempuh ilmu di bangku perkuliahan.
5. Segenap Bapak dan Ibu Dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
6. Orang tua dan kakak atas doa restunya dan dukungan untuk penulis.
7. Yuyun Deviatun Ni'mah A.Md. atas dukungan dan motivasi yang diberikan kepada penulis.
8. Dian Cahyono, Rizky Setyo Pambudi, dan Heikal Mahendra Rukmana yang telah banyak memberikan bantuan dan fasilitas kepada penulis.

9. Teman-teman di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan bantuannya demi kelancaran pelaksanaan penyusunan skripsi ini.
10. Serta pihak-pihak lain yang tidak dapat penulis sebutkan satu persatu, tetapi turut serta membantu penulis menyelesaikan skripsi ini.

Penulis sadari bahwa dalam laporan ini kemungkinan masih ada kekurangan, oleh karena itu penulis sangat menghargai saran dan kritik yang sifatnya membangun dari pembaca. Semoga skripsi ini dapat bermanfaat.

Malang, Agustus 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SOURCE CODE

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dalam kegiatan penyampaian pesan dan komunikasi data selalu melibatkan pengiriman pesan dari pengirim ke penerima yang terkadang bersifat rahasia, yang berarti pesan tidak boleh diketahui oleh pihak yang tidak berhubungan. Oleh karena itu masalah keamanan data menjadi masalah utama dalam komunikasi data. Untuk mengangani permasalahan tersebut, dibutuhkan pengamanan data yang dapat menghindari data atau pesan diketahui oleh pihak yang tidak berhubungan.

Pengamanan data dapat dilakukan dengan cara kriptografi ataupun steganografi. Kriptografi adalah suatu teknik untuk menyamarkan arti dari suatu pesan (Kurniawan, 2008), sedangkan steganografi adalah seni dan ilmu menulis pesan tersembunyi atau menyembunyikan pesan (Rosediana, 2011). 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 akan lebih aman bila disembunyikan ke dalam suatu data lain melalui teknik steganografi.

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 dalam ukuran satu bit kemudian bit tersebut akan menggantikan *Least Significant Bit* dari media penyisipan atau yang disebut dengan metode *Least Significant Bit (LSB).*

Dalam steganografi dibutuhkan media pembawa (*carrier*) untuk menyisipkan pesan. Media pembawa pada umumnya dapat berupa teks, gambar, suara, ataupun video.

Gambar dengan format *jpg* ataupun *bmp* digunakan Basuki, Nadhori, dan Maulana (2009) sebagai media pembawa (*carrier*) pesan rahasia. Pada penelitian tersebut penyisipan pesan rahasia berupa teks ke dalam gambar dilakukan dengan metode *Least Significant Bit (LSB)*. Sebelum dilakukan penyisipan, digunakan metode enkripsi *DES (Data Encryption Standard)* sebagai proses

pengamanan pesan. Dari penelitian diketahui bahwa teknik penyisipan pesan dengan metode *LSB* hanya mampu menyimpan informasi dengan ukuran yang sangat terbatas. Untuk citra atau gambar 24-bit (R=8 bit, G=8 bit, B=8 bit) hanya mampu menyimpan pesan rahasia maksimum sebesar 1/8 dari ukuran citra penampung. Metode *Least Significant Bit* dipilih karena kesederhanaan dalam penggunaanya serta tidak terlalu banyak mengubah struktur dari media pembawa.

Pada skripsi ini akan digunakan berkas *audio* atau suara sebagai media pembawa, 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 pembawa pesan karena tidak akan menimbulkan kecurigaan adanya pesan rahasia didalamnya.

Untuk meningkatkan keamanan dan kerahasiaan pesan yang akan disisipkan ke dalam berkas MP3 maka sebelum dilakukan proses penyisipan, terlebih dahulu akan dilakukan proses enkripsi pada pesan. Proses enkripsi yang akan dilakukan menggunakan algoritma RSA.

RSA termasuk algoritma asimetri, yang berarti memiliki dua kunci, yaitu kunci publik dan kunci rahasia. Algoritma RSA dianggap sangat aman karena algoritma ini melakukan pemfaktoran bilangan yang sangat besar dari bilangan nonprima menjadi faktor primanya (Kurniawan, 2004).

Berdasarkan latar belakang diatas maka dalam skripsi ini diambil judul **"Penyisipan Pesan Terenkripsi Pada *File Audio* MP3 Menggunakan Metode *Least Significant Bit (LSB)*".**

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, dapat dirumuskan masalah-masalah sebagai berikut :

1. Bagaimana mengamankan sebuah pesan berupa teks terenkripsi dengan algoritma RSA ke dalam berkas *audio* MP3 dengan

mengimplementasikan teknik steganografi *Least Significant Bit (LSB).*

2. Berapa tingkat perubahan kualitas audio yang diukur dari nilai *Mean Opinion Score (MOS)* dan *Signal to Noise Ratio (SNR)* pada MP3 sebagai media pembawa pesan setelah dilakukan proses penyisipan pesan menggunakan LSB mode *standard* dan mode per-50 *frame.*
3. Bagaimana tingkat keamanan algoritma RSA yang diukur berdasarkan banyaknya iterasi dari *Brute Force Attack* untuk mendapatkan kunci rahasia.

1.3. Batasan Masalah

Batasan ruang lingkup masalah yang didefinisikan dalam tugas akhir ini adalah :

1. Pesan yang akan disembunyikan hanyalah berupa teks.
2. Sebelum disembunyikan ke dalam berkas pembawa (carrier), pesan terlebih dahulu akan dienkripsi menggunakan algoritma kriptografi RSA.
3. Parameter yang akan diuji adalah perubahan kualitas berkas MP3 setelah disisipi pesan dengan mode *standard* dan per-50 *frame* yang diukur berdasarkan *SNR* dan nilai *MOS* serta banyaknya iterasi *Brute Force Attack* untuk mendapatkan kunci rahasia sebagai parameter tingkat keamanan algoritma RSA

1.4. Tujuan Penelitian

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

1. Mengimplementasikan pengamanan sebuah pesan berupa teks terenkripsi dengan algoritma RSA ke dalam berkas *audio* MP3 dengan mengimplementasikan teknik steganografi *Least Significant Bit (LSB).*
2. Mengukur perubahan kualitas dengan menghitung nilai *Mean Opinion Score (MOS),* prosentase dan nilai *Signal to Noise Ratio (SNR)* MP3 setelah disisipi pesan dengan sebelum disisipi pesan.
3. Menghitung iterasi atau perulangan pada *Brute Force Attack* untuk mendapatkan kunci rahasia sebagai parameter tingkat keamanan algoritma RSA.

1.5. Manfaat

Manfaat dari penelitian ini diharapkan dapat menjadi solusi untuk menjaga aspek kerahasiaan dan integritas data pada pengamananan data ke dalam berkas *audio* MP3.

1.6. Sistematika Penulisan

Skripsi ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN
   Berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian serta sistematika penulisan skripsi.
2. BAB II TINJAUAN PUSTAKA
   Menjelaskan teori – teori yang berhubungan dengan Steganografi dan Kriptografi.
3. BAB III METODOLOGI DAN PERANCANGAN SISTEM
   Pada bab ini akan dijelaskan mengenai metode-metode yang digunakan dan tahapan-tahapan teknik steganografi dan kriptografi.
4. BAB IV IMPLEMENTASI DAN PEMBAHASAN
   Dalam bab ini akan dijelaskan mengenai implementasi program, pengujian dan analisa hasil penelitian.
5. BAB V KESIMPULAN DAN SARAN
   Bab ini berisi kesimpulan dari seluruh rangkaian penelitian serta saran kemungkinan pengembangan.

BAB II
TINJAUAN PUSTAKA

2.1. Steganografi

2.1.1. Pengertian Steganografi

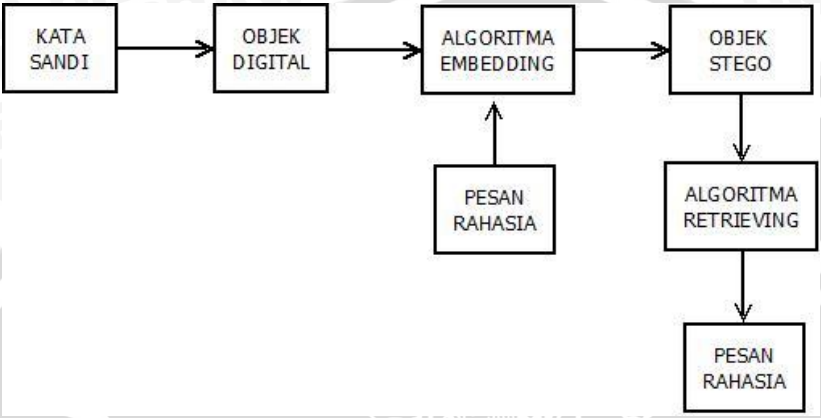
Istilah steganografi atau yang dalam dunia internasional dikenal dengan *steganography* sebenarnya berasal dari Bahasa Yunani, yaitu *steganos* dan *graphein*. *Steganos* berarti "penyamaran" atau "penyembunyian", sedangkan *graphein* memiliki arti "tulisan". Sehingga secara bahasa, *steganography* dapat diartikan sebagai seni menyamarkan atau menyembunyikan pesan tertulis ke dalam pesan lainya (Ariyus, 2007).

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 (Cox dkk, 2008).

Steganografi membutuhkan dua properti, yaitu media penampung dan data rahasia atau pesan rahasia yang akan disembunyikan. Pada teknik steganografi digital media penampung yang digunakan adalah media digital, misalnya citra (gambar), suara (*audio*), teks, dan video. Data rahasia yang disembunyikan juga dapat berupa citra, suara, teks, atau video (Kurniawan, 2008).

Menurut Hapsari (2009), steganografi digunakan untuk menyembunyikan suatu data atau informasi ke dalam sebuah media sehingga sulit dideteksi keberadaan data atau informasi tersebut karena hasil dari penyembunyian tersebut tidak berbeda dengan sumbernya. Hal pertama yang perlu dilakukan untuk melakukan steganografi adalah memilih media penampung pesan. Kemudian data atau informasi tersebut baru disisipkan atau disembunyikan ke

dalam media tersebut. Gambar 2.1 dapat memperlihatkan gambaran umum proses steganografi.



Gambar 2.1 Proses Steganografi (Hapsari, 2009)

Untuk menyisipkan informasi atau data rahasia ke dalam objek digital diperlukan suatu algoritma yang disebut dengan algoritma *embedding*. Algoritma tersebut dapat memodifikasi objek digital sehingga menghasilkan objek digital baru yang berisi informasi tersembunyi. Dalam proses modifikasi, perubahan yang terjadi antara objek digital (media asli) dengan objek digital yang mengandung pesan rahasia tidak boleh terlalu terlihat perbedaanya. Sedangkan untuk mengungkapkan kembali pesan rahasia yang terdapat pada suatu media digital diperlukan algoritma *retrieving*.

Dalam menyembunyikan pesan, ada beberapa kriteria yang harus dipenuhi (Munir, 2004) :

1. *Fidelity*. Mutu citra penampung tidak jauh berubah. Setelah penambahan data rahasia, citra hasil steganografi masih terlihat dengan baik. Pengamat tidak mengetahui kalau di dalam citra tersebut terdapat data rahasia.
2. *Robustness*. Data yang disembunyikan harus tahan (*robust*) terhadap berbagai operasi manipulasi yang dilakukan pada citra penampung seperti pengubahan kontras, penajaman, pemampatan, rotasi, perbesaran gambar, pemotongan (*cropping*), enkripsi, dan sebagainya. Bila pada citra dilakukan operasi-operasi pengolahan citra tersebut, maka pada data yang

disembunyikan seharusnya tidak rusak (tetap valid jika diekstraksi kembali).

3. *Recovery*. Data yang disembunyikan harus dapat diiungkapkan kembali (*reveal*). Karena tujuan stegnografi adalah data hiding, maka sewaktu-waktu data rahasia di dalam citra penampung harus dapat diambil kembali untuk digunakan lebih lanjut.

2.1.2. Metode *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 (Kalangi, 2010) :

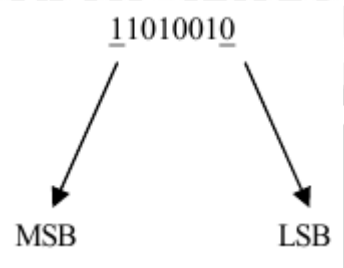
1. Penggantian bit. Cara ini lazim digunakan dalam teknik digital steganografi yaitu mengganti bagian tertentu dari bit-bit datanya dengan data rahasia yang disisipkan. Dengan metode ini keuntungan yang didapatkan adalah ukuran pesan yang disisipkan relatif besar, namun berdampak pada hasil *audio* yang berkualitas kurang dengan banyaknya derau.
2. Metode kedua yang digunakan adalah merekayasa fasa dari sinyal masukan. 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.
3. Metode yang ketiga adalah penyebaran spektrum. 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.

4. Metode terakhir yang sering digunakan adalah menyembunyikan pesan melalui teknik *echo*. 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.

2.2. Metode Least Significant Bit (LSB)

Metode Least Significant Bit merupakan teknik subtitusi pada steganografi. Biasanya arsip 24-bit atau 8-bit digunakan untuk menyimpan informasi. Metode ini lebih sering digunakan untuk menyisipkan pesan ke dalam citra digital ataupun suara (*audio*). Representasi warna piksel-piksel pada citra digital dapat diperoleh dari warna-warna primer, yaitu merah, hijau, dan biru. Pada citra berukuran 24-bit digunakan 3 bytes untuk masing-masing piksel, dimana setiap warna primer direpresentasikan dengan 1 byte. Citra 24-bit memungkinkan setiap piksel merepresentasikan nilai warna sebanyak 16.777.216 kemungkinan. Satu-dua bit saluran warna bisa digunakan untuk menyisipkan data. Proses penyisipan tersebut akan mengubah jenis warna pikselnya menjadi 64-warna. Hal tersebut juga akan mengakibatkan perbedaan yang sangat sedikit yang dapat dideteksi secara kasat mata oleh manusia. Oleh karena itu metode ini disebut Least Significant Bit atau bit yang tidak penting (Ariyus, 2007).

Pada susunan bit di dalam sebuah byte (1 byte = 8 bit), ada bit yang paling berarti dan berpengaruh terhadap informasi yang dikandungnya yaitu angka 1 yang terletak di paling depan. Bit ini sering disebut dengan istilah *Most Significant Bit (MSB)*. Semakin ke kanan, bit-bit tersebut semakin kecil pengaruhnya terhadap keutuhan data yang dikandung dan sebaliknya ke kiri bit-bit tersebut semakin besar pengaruhnya terhadap kualitas dan keutuhan data. Bit yang paling kurang berarti dinamakan *Least Significant Bit (LSB)* (Rengganis, 2009). Letak MSB dan LSB dapat dilihat pada gambar 2.2.



Gambar 2.2 Letak MSB dan LSB (Hastari, 2009).

Teknik steganografi dengan *metode LSB (Least Significant Bit)* adalah teknik yang paling sering digunakan. Informasi yang akan disembunyikan akan diambil nilai binernya kemudian disisipkan pada LSB sederetan byte. Kelemahan dari metode LSB adalah besar pesan yang dapat disembunyikan sangat tergantung dari media yang dipergunakan. Sebagai contoh adalah sebuah media berupa gambar yang akan disisipkan informasi rahasia. Media gambar yang mempunyai ke dalaman 24 bit setiap piksel-nya terdiri atas susunan tiga warna merah, hijau dan biru (RGB) yang masing-masing disusun oleh bilangan 8 byte dari 0 sampai 255 dalam format biner dari 00000000 sampai 1111111. Dengan menggunakan metode LSB, sebuah huruf dapat disisipkan dengan menggunakan tiga piksel gambar. Setiap piksel gambar dapat menampung tiga bit data.

Sebagai contoh akan disisipkan huruf "A" ke dalam tiga piksel warna degan representasi biner :

(10010101 11000101 00101010) → piksel pertama (R, G ,B)
(00011100 10000110 01100110) → piksel kedua (R, G ,B)
(10000111 10010100 00100010) → piksel ketiga (R, G ,B)

Huruf A memiliki representasi biner 01000001, sehingga bila disisipkan, data diatas akan berubah menjadi data

(10010100 11000101 00101010) → piksel pertama (R, G ,B)
(00011100 10000110 01100110) → piksel kedua (R, G ,B)
(10000110 10010101 00100010) → piksel ketiga (R, G ,B)

Dari hasil perubahan yang terjadi pada piksel diatas dapat diketahui bahwa hanya pada warna merah piksel pertama, warna merah piksel ketiga dan warna hijau piksel ketiga sajalah yang mengalami perubahan. Perubahan tersebut tidak terlalu besar sehingga untuk mata manusia perubahan warna yang terjadi tidak akan tampak. Perubahan pada LSB ini akan terlalu kecil untuk terdeteksi oleh mata manusia sehingga pesan dapat disembunyikan secara efektif (Maya, 2006).

2.3. Kriptografi

2.3.1. Pengertian Kriptografi

Kata kriptografi berasal dari Yunani yaitu, *cryptos* yang berarti rahasia dan *graphein* yang berarti tulisan. Menurut terminologinya, kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan ketika pesan dikirim dari suatu tempat ke tempat lain (Ariyus, 2008). Kriptografi juga dapat diartikan sebagai suatu ilmu yang berguna untuk mengacak atau *masking* data sedemikian rupa sehingga data tidak dapat dibaca oleh pihak yang tidak berhubungan. Data yang diacak harus bisa dikembalikan ke bentuk semula oleh pihak yang berwenang (Morgana, 2011).

Kriptografi telah menjadi salah satu cara yang sering digunakan dalam melawan atau menghalangi penyerangan terhadap keamanan dan privasi seseorang, menjamin integritas data dan kerahasiaan data, dan memberikan kepercayaan pada global *ecommerce*. Kriptografi telah menjadi hal utama dalam menyediakan kebutuhan keamanan digital pada era komunikasi digital modern saat ini. Tujuan kriptografi adalah untuk menjamin aspek kerahasiaan, integritas data, autentikasi, dan non-repudansi pada semua komunikasi dan pertukaran informasi (Kizza, 2005).

2.3.2. Komponen Kriptografi

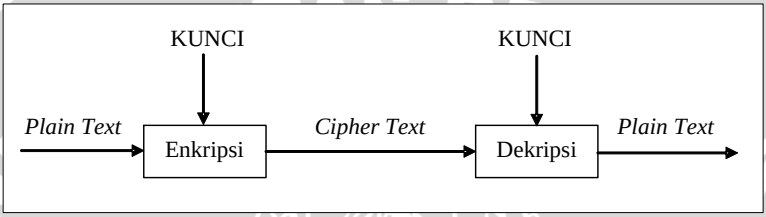
Menurut Kizza (2005) kriptografi terdiri dari empat komponen esensial :

1. *Plaintext*
   Teks asli yang akan dikirim.
2. Sistem Kriptografi (Kriptosistem) atau *Chiper*
   Terdiri dari algoritma matematis proses enkripsi dan dekripsi.
3. *Ciphertext*

Merupakan hasil dari pengaplikasian algoritma enkripsi untuk plaintext sebelum dikirim ke penerima.

4. Key atau *Kunci*
   Merupakan bit string yang digunakan untuk algoritma matematis proses enkripsi dan dekripsi.

Hubungan keempat komponen esensial dari kriptografi ditunjukkan pada gambar 2.3 :

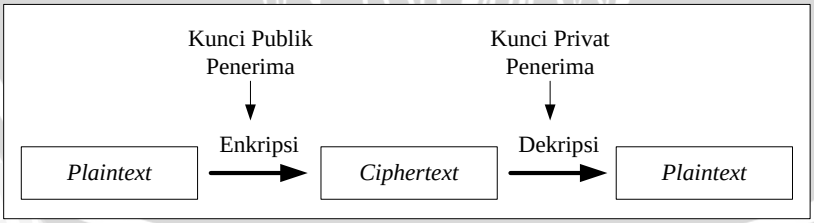


Gambar 2.3 Gambaran umum proses kriptografi

2.4. Metode Kriptografi RSA

2.4.1. Sejarah dan Pengertian RSA

RSA ditemukan oleh tiga orang yaitu Ron Rivest, Adi Shamir, dan Leonard Adleman. Nama RSA sendiri adalah singkatan dari nama belakang penemu. Tingkat keamanan RSA tidak pernah dibuktikan karena sulitnya pemfaktoran bilangan yang sangat besar. Oleh sebab itu algoritma kriptografi RSA masih dianggap aman hingga sekarang. RSA termasuk algoritma asimetri, yang berarti memiliki dua kunci, yaitu kunci publik dan kunci privat. Kunci Publik digunakan untuk melakukan enkripsi, dan dapat diketahui oleh orang lain. Sedangkan kunci privat tetap dirahasiakan dan digunakan untuk melakukan dekripsi. (Kurniawan, 2004). Penggunaan kunci public dan kunci privat digambarkan pada gambar 2.4:



Gambar 2.4 Penggunaan kunci publik dan privat (Kizza, 2005)

Algoritma RSA pertama kali dipublikasikan di tahun 1977 oleh Ron Rivest, Adi Shamir, dan Leonard Adleman dari Massachusetts Institute of Technology (MIT). Clifford Cocks, seorang matematikawan yang berasal dari Inggris sebenarnya juga telah mengembangkan algoritma yang hampir sama dengan RSA ini pada tahun 1973. Namun algoritma buatannya tidak begitu dikenal oleh publik, penemuan Clifford Cocks tidak terungkap hingga tahun 1997 karena alasan top-secret classification. Walau begitu algoritma yang dikembangkan Rivest, Shamir, dan Adleman tidak berhubungan dengan pekerjaan Cocks. Pada algoritma RSA terdapat tiga langkah utama yaitu key generation (pembangkitan kunci), enkripsi, dan dekripsi.

2.4.2. Konsep Matematis dalam RSA

RSA merupakan algoritma kriptografi yang menggunakan perhitungan matematis dengan pangkat besar, untuk itu dibahas konsep matematis (Riyanto dan Ardhian, 2008):

1. Fungsi Phi Euler $\phi(n)$

   Fungsi Phi-Euler $\phi(n)$ merupakan fungsi terhadap bilangan bulat positif n yang menyatakan banyaknya elemen Zn yang mempunyai invers terhadap operasi pergandaan. Diketahui bahwa Zn belum tentu merupakan grup terhadap operasi penggandaan. Dengan kata lain, $\phi(n)$ adalah banyaknya elemen $\{x, 0 \leq x < n \mid FPB(x,n) = 1\}$. Jika $n = pq$ dengan p dan q adalah bilangan prima, maka :

   $$\phi(n) = (p - 1)(q - 1). \quad (2.1)$$

   Jika n adalah bilangan prima, maka :

   $$\phi(n) = n - 1. \quad (2.2)$$

2. Algoritma Euclide

   Algoritma ini digunakan untuk mencari nilai pembagi persekutuan terbesar dari dua bilangan bulat. Algoritma ini didasarkan pada pernyataan berikut ini. Diberikan bilangan bulat r_0 dan r_1, dengan $r_0 \geq r_1$, kemudian dihitung menggunakan algoritma pembagian seperti pada persamaan 2.3:

   $$r_0 = q_1 r_1 + r_2, \quad 0 < r_2 < r_1$$
   $$r_1 = q_2 r_2 + r_3, \quad 0 < r_3 < r_2$$
   $$\ldots\ldots\ldots$$

$$r_{n-2} = q_{n-1} r_{n-1} + r_n , \quad 0 < r_n < r_{n-1}$$
$$r_{n-1} = q_n r_n \qquad (2.3)$$

Dari persamaan 2.3, dapat ditunjukkan persamaan 2.4

$$FPB(r_0 , r_1) = FPB(r_1 , r_2) = ...$$
$$= FPB(r_{n-1} , r_n) = FPB(r_n , 0) = r_n . \qquad (2.4)$$

Contoh Penggunaan Algoritma Euclide
Akan dihitung FPB(40,24).
Jawab: 40 = 1.24 + 16
24 = 1.16 + 8
16 = 2.8
Jadi, FPB(40,24)=8.

3. Algoritma Euclide Diperluas

Algoritma ini merupakan perluasan dari algoritma Euclide (*Extended Euclidean Algorithm*), digunakan untuk mencari invers terhadap operasi pergandaan. Algoritma ini didasarkan pada pernyataan pada persamaan 2.5 bila diiberikan bilangan bulat positif r_0 dan r_1 , $r_0 > r_1$.

jika FPB $(r_0,r_1) = 1$, maka $r_1^{-1} \bmod r_0$ ada
jika $FPB(r_0,r_1) \neq 1$, maka $r_1^{-1} \bmod r_0$ tidak ada. (2.5)

Dari persamaan tersebut digunakan persamaan rekurensi 2.6 dan 2.7.

$$t_0 = 0 \ , \ t_1 = 1$$
$$t_j = t_{j-2} - q_{j-1} t_{j-i} , \ j \geq 2 \qquad (2.6)$$

dimana,

$$q_{j=} \ FPB(r_0,r_1) \qquad (2.7)$$

Contoh Penggunaan Algoritma Euclide diperluas
Akan dihitung $28^{-1} \bmod 75$. Diketahui $r_0 = 75$ dan $r_1 = 28$. Hitung FPB(75,28) menggunakan algoritma Euclide, yaitu:

$75 = 2.28 + 19$ n = 1, $r_1 = 28$, $q_1 = 2$
$28 = 2.19 + 9$ n = 2, $r_2 = 19$, $q_2 = 2$
$19 = 2.9 + 1$ n = 3, $r_3 = 9$, $q_3 = 2$
$9 = 9.1$ n = 4, $r_4 = 1$, $q_4 = 9$

Jadi, FPB(75,28) = 1, dengan kata lain $28^{-1} \bmod 75$ ada. Dari penyelesaian algoritma Euclide diperoleh diperoleh n = 4.

Selanjutnya, menggunakan persamaan (2.6), diperoleh: (semua perhitungan dilakukan dalam mod 75)

$t_2 = t_0 - q_1 t_1 = 0 - 2.1 = -2 = 73$

$t_3 = t_1 - q_2 t_2 = 1 - 1.(-2) = 3$

$t_4 = t_2 - q_3 t_3 = -2 - 2.3 = -8 = 67$

$r_4 = t_4 r_1 = 67.28 = 1 \Leftrightarrow 67 = 28^{-1}$

Dari hasil terakhir di atas, diperoleh $28^{-1} \bmod 75 = 67$.

4. Metode *Fast Exponentiation*

Metode ini digunakan untuk menghitung operasi pemangkatan besar bilangan bulat modulo dengan cepat. Metode Fast Exponentiation memanfaatkan ekspansi biner dari eksponennya dan didasarkan pada persamaan 2.8:

$$g^{2^{j+1}} = (g^{2^j})^2 \qquad (2.8)$$

Contoh Penggunaan Metode Fast Exponentiation

Akan dihitung $6^{73} \bmod 100$.

Jawab :

$73 = 1.2^6 + 1.2^3 + 1.2^0$ atau $73 = (1001001)_2$ (semua perhitungan dilakukan dalam mod 100)

$6^{2^0} = 6,\ 6^{2^1} = 36,\ 6^{2^2} = 36^2 = 96,\ 6^{2^3} = 16,\ 6^{2^4} = 16^2 = 56,\ 6^{2^5} = 56^2 = 36,\ 6^{2^6} = 96$

Sehingga $6^{73} = 6.6^{2^3}.6^{2^6} = 6.16.96$

$6^{73}\ mod\ 100 = 16$

2.4.3. Proses Enkripsi dan Dekripsi Pada RSA

Didalam kriptografi RSA terdapat tiga proses yaitu proses pembentukan kunci, proses enkripsi, dan yang terakhir proses dekripsi (Riyanto dan Ardhian, 2008).

1. Pembentukan Kunci

Berikut ini adalah proses pembentukan kunci. Proses ini dilakukan oleh pihak penerima.

A. Dipilih bilangan prima p dan q
B. Dihitung n = pq Sebaiknya $p \neq q$, sebab jika p = q maka r = p2 sehingga p dapat diperoleh dengan menarik akar pangkat dua dari r.
C. Dihitung $\phi(n) = (p-1)(q-1)$.

D. Dipilih Kunci Publik(PK) dengan PK=b, $1 < b < \phi(n)$, dengan FPB(b,ϕ(n)) = 1.

E. Dilakukan pembangkitan Kunci Rahasia (SK) dengan persamaan 2.9

$$SK \cdot PK \equiv 1 \pmod{\phi(n)} \quad (2.9)$$

Perhatikan bahwa $SK \cdot PK \equiv 1 \pmod{\phi(n)}$ ekivalen dengan $SK \cdot PK = 1 + m\phi(n)$, sehingga *SK* dapat dihitung dengan persamaan 2.10

$$SK = \frac{1 + m\phi(n)}{PK} \quad (2.10)$$

Untuk SK=a; dan PK=b dapat juga dihitung invers dari b, yaitu dengan persamaan 2.11

$$a = b^{-1} \bmod \phi(n) . \quad (2.11)$$

F. Kunci publik: (n, b) dan kunci rahasia: a.

2. Enkripsi

Berikut ini adalah proses enkripsi RSA. Seluruh perhitungan pemangkatan bilangan modulo dilakukan menggunakan metode *fast exponentiation*.

A. Diambil kunci publik (n,b).

B. Dipilih plainteks m, dengan $0^n \leq m^n \leq n - 1$.

C. Dihitung dengan persamaan 2.12

$$c = m^b \bmod n \quad (2.12)$$

D. Diperoleh cipherteks c, dan kirimkan kepada penerima.

3. Dekripsi

Berikut ini adalah proses dekripsi RSA. Dilakukan oleh pihak penerima cipherteks.

A. Diambil kunci publik (n,b) dan kunci rahasia a.

B. Dihitung menggunakan persamaan 2.13

$$m = c^a \bmod n \quad (2.13)$$

2.5. *Audio* MP3

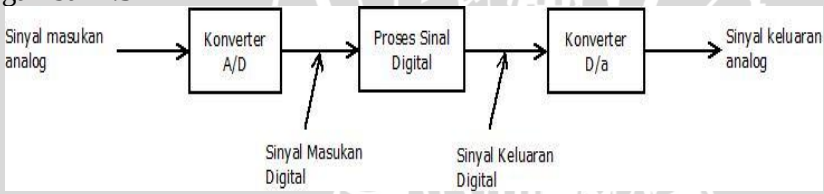
2.5.1. Pengertian *Audio*

Audio didefinisikan sebagai sesuatu yang dihasilkan oleh benda yang mengalami getaran sehingga menghasilkan gelombang

yang berada di udara. Bentuk gelombang yang berulang-ulang pada waktu tertentu disebut suatu periode. Suatu bentuk gelombang yang tidak menghasilkan suara yang periodik sama seperti sebuah derau. Frekuensi dari suatu suara adalah banyaknya periode gelombang dalam waktu satu detik (Hz). Frekuensi yang dapat didengar oleh manusia adalah 20 Hz –20 KHz. Selain frekuensi tersebut ada jenis-jenis frekuensi yang lain yaitu *Infra-sound* (0 – 20 Hz), *Ultra-sound* (20 KHz – 1 GHz) dan *Hyper-sound* (1 GHz – 10 THz)

Representasi audio dalam komputer terjadi secara berkelanjutan dikarenakan adanya gelombang analog. Untuk merubah gelombang analog ke dalam komputer dapat dilakukan dengan cara digitalisasi gelombang analog. *Analog to Digital Converter (ADC)* mengubah amplitude sebuah gelombang ke dalam waktu interval (samples) sehingga menghasilkan representasi digital dari suara. Sebaliknya untuk menampilkan suara digital ke dalam alat suara analog seperti speaker digunakan *Digital to Analog Converter (DAC)* untuk mengkonversinya (Wafda, 2011).

Pemrosesan sinyal digital menyediakan suatu metode alternatif untuk pemrosesan sinyal analog. Untuk melakukan pemrosesan secara digital, diperlukan dua *interface* yang disebut pengkonversi analog menjadi digital. Proses ditunjukkan pada gambar 2.5



Gambar 2.5 Pemrosesan Sinyal Analog (Wafda, 2011)

2.5.2. Sejarah MP3

MPEG-1 *Audio* Layer 3 atau lebih dikenal sebagai MP3 adalah salah satu format berkas pengodean suara yang memiliki kompresi yang baik sehingga ukuran berkas bisa memungkinkan menjadi lebih kecil. Berkas ini dikembangkan oleh Karlheinz Brandenburg dan Heinz Gerhauser dari Fraunhofer Institute, Erlangen, Jerman. Seitzer memimpin sejak 1985-1993, dilanjutkan Gerhauser sejak 1993 yang saat itu berusaha menyusun algoritma yang mampu membuat musik yang dipecah kodenya dan terdengar

se-realistik mungkin bagi telinga. 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 musik. Awalnya dari proyek EUREKA EU147, digagas Digital *Audio Broadcasting (DAB).* Bersama tim Fraunhofer akhirnya menemukan algoritma yang distandarisasi sebagai ISO-MPEG *Audio* Layer-3 (IS 11172-3 dan IS13818-3). Algoritma Layer 3 yang dihasilkan memampatkan musik dengan menyaring suara yang tak tertangkap telinga. Sebuah berkas suara bisa dimampatkan sebesar 1/10 dari ukuran semula. Namun untuk membuat formatnya jadi universal, dibutuhkan standar internasional. Maka pada 1988 terjadi 'koalisi' dengan *Moving Picture Experts Group (MPEG).* Menggunakan kode suara MPEG, data suara dapat direduksi dengan faktor reduksi 12, tanpa mengurangi kualitas suara, atau menambah faktor reduksi (24) dengan mengurangi sedikit kualitas suara. MP3 mengurangi jumlah bit yang diperlukan dengan menggunakan model psychoacoustic untuk menghilangkan komponen-komponen suara yang tidak terdengar oleh manusia. Meski ukurannya jadi lebih kecil, tapi berkualitas baik. Pada 1995, tim Fraunhofer merilis Winplay, pemutar musik versi Windows yang bisa memecah kode berkas MP3 di PC secara *real time*. MP3 memakai pengodean *Pulse Code Modulation (PCM)* (Rosgani, 2008). Pada awalnya MPEG *Audio* Layer-3 banyak dipakai oleh para pengguna komputer. Berkas-berkas MPEG *Audio* Layer-3 disimpan dengan ekstensi nama berkas MP3. Kemudian MPEG *Audio* Layer-3 selanjutnya banyak dikenal sebagai MP3 (Chasanah, 2009).

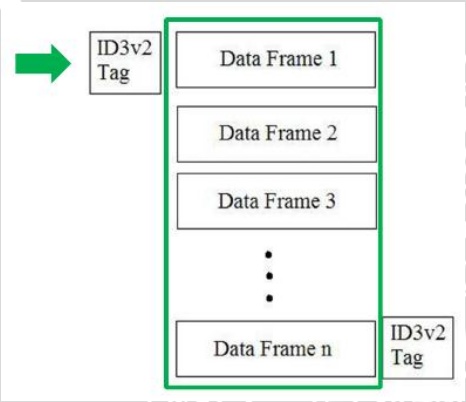
Kesuksesan MP3 dimulai pada 1998, ketika WinAmp, sebuah mesin pemutar MP3 yang dibuat oleh sepasang mahasiswa bernama Justin Frankel dan Dmitry Boldyrev, ditawarkan secara cuma-cuma di internet. Dalam waktu singkat, pengguna musik di seluruh dunia terhubung dalam satu jaringan terpusat bernama MP3, dan saling menawarkan musik-musik yang memiliki hak cipta secara gratis. Dalam rentang waktu yang tidak terlalu yang singkat, banyak programmer menciptakan berbagai perangkat lunak pendukung untuk para pengguna MP3.

2.5.3. Struktur Berkas MP3

MP3 memakai sebuah transformasi hybrid untuk mentransformasikan sinyal pada ranah waktu ke sinyal pada ranah frekuensi:

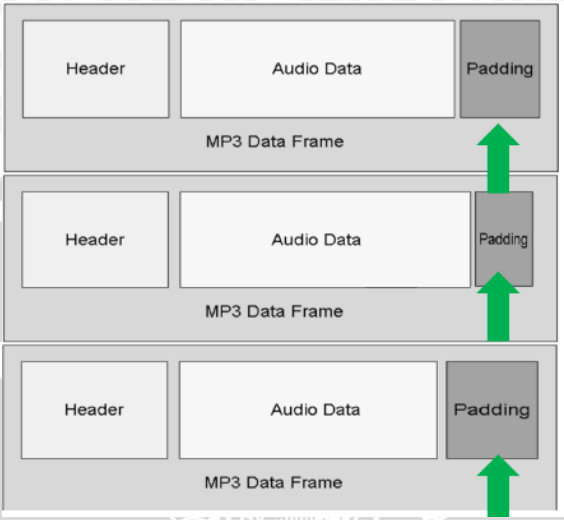
- *Filter polyphase quadrature 32-band*
- 36 atau 12 *MDCT (modified discrete cosine transform)*, dengan ukuran dapat dipilih secara independen untuk sub-band 0…1 dan 2…31
- Postproses aliasing reduction

MP3 tersusun atas banyak *frame*. Setiap *frame* memiliki waktu atau durasi yang sama yaitu 26 milidetik. Diperlihatkan pada gambar 2.6 struktur dari MP3.



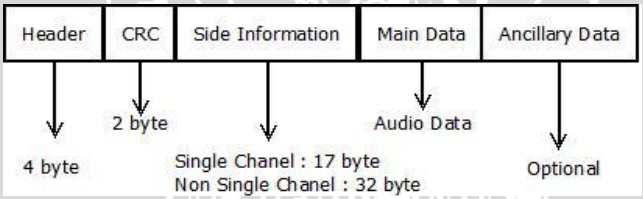
Gambar 2.6 Struktur MP3 (Peter Chan, 2008).

Walaupun memiliki durasi yang sama, *frame* satu dengan yang lainya memiliki ukuran byte yang berbeda tergantung dari bitrate-nya. Untuk menyamakan durasi dari tiap *frame*, maka biasanya akan ditambahkan padding pada tiap *frame*-nya. Ditunjukkan pada gambar 2.7 struktur umum dari *frame* pada MP3.



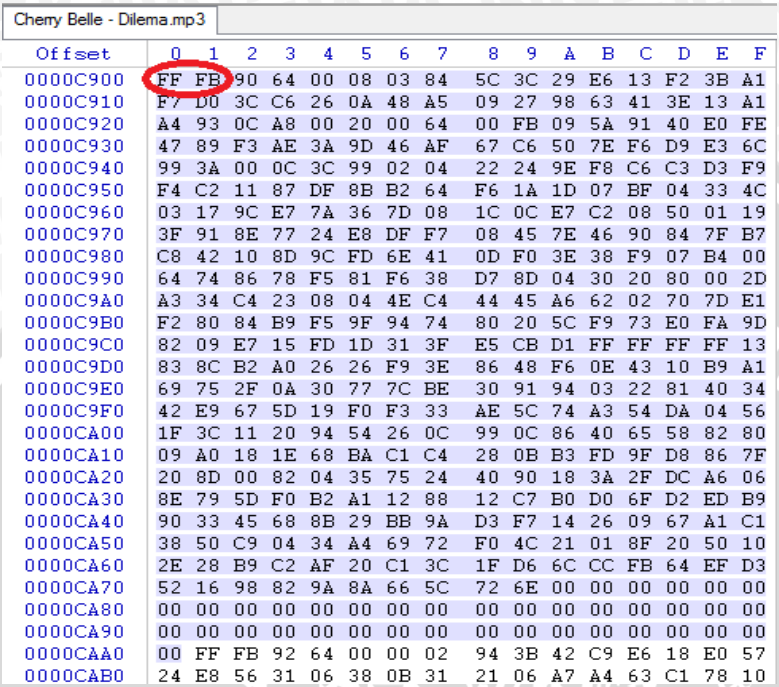
Gambar 2.7 Bentuk Umum Struktur *frame* (Peter Chan, 2008).

Menurut Rassol (2002) sebuah *frame* MP3 secara umum terdiri atas 5 bagian, yaitu *Header*, *CRC*, *Side Information*, *Main Data* dan *Ancillary Data* yang ditunjukkan pada gambar 2.8



Gambar 2.8 Struktur *frame* (Rasol, 2002)

Menurut Baskara (2008), Setiap *tag* MP3 atau awal berkas MP3 diawali dengan bilangan 3 *byte* hexadecimal 49, 44, dan 33. Sedangkan untuk setiap *frame* pada MP3 selalui diawali dengan MP3 *header* yang berukuran 36 byte. Setiap awal *header* ditandai dengan bilangan Hexadecimal FF dan FB. Pada gambar 2.9 ditunjukkan *frame* dari sebuah berkas MP3 dengan menggunakan aplikasi WinHex.


Cherry Belle - Dilema.mp3

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000C900	FF	FB	90	64	00	08	03	84	5C	3C	29	E6	13	F2	3B	A1
0000C910	F7	D0	3C	C6	26	0A	48	A5	09	27	98	63	41	3E	13	A1
0000C920	A4	93	0C	A8	00	20	00	64	00	FB	09	5A	91	40	E0	FE
0000C930	47	89	F3	AE	3A	9D	46	AF	67	C6	50	7E	F6	D9	E3	6C
0000C940	99	3A	00	0C	3C	99	02	04	22	24	9E	F8	C6	C3	D3	F9
0000C950	F4	C2	11	87	DF	8B	B2	64	F6	1A	1D	07	BF	04	33	4C
0000C960	03	17	9C	E7	7A	36	7D	08	1C	0C	E7	C2	08	50	01	19
0000C970	3F	91	8E	77	24	E8	DF	F7	08	45	7E	46	90	84	7F	B7
0000C980	C8	42	10	8D	9C	FD	6E	41	0D	F0	3E	38	F9	07	B4	00
0000C990	64	74	86	78	F5	81	F6	38	D7	8D	04	30	20	80	00	2D
0000C9A0	A3	34	C4	23	08	04	4E	C4	44	45	A6	62	02	70	7D	E1
0000C9B0	F2	80	84	B9	F5	9F	94	74	80	20	5C	F9	73	E0	FA	9D
0000C9C0	82	09	E7	15	FD	1D	31	3F	E5	CB	D1	FF	FF	FF	FF	13
0000C9D0	83	8C	B2	A0	26	26	F9	3E	86	48	F6	0E	43	10	B9	A1
0000C9E0	69	75	2F	0A	30	77	7C	BE	30	91	94	03	22	81	40	34
0000C9F0	42	E9	67	5D	19	F0	F3	33	AE	5C	74	A3	54	DA	04	56
0000CA00	1F	3C	11	20	94	54	26	0C	99	0C	86	40	65	58	82	80
0000CA10	09	A0	18	1E	68	BA	C1	C4	28	0B	B3	FD	9F	D8	86	7F
0000CA20	20	8D	00	82	04	35	75	24	40	90	18	3A	2F	DC	A6	06
0000CA30	8E	79	5D	F0	B2	A1	12	88	12	C7	B0	D0	6F	D2	ED	B9
0000CA40	90	33	45	68	8B	29	BB	9A	D3	F7	14	26	09	67	A1	C1
0000CA50	38	50	C9	04	34	A4	69	72	F0	4C	21	01	8F	20	50	10
0000CA60	2E	28	B9	C2	AF	20	C1	3C	1F	D6	6C	CC	FB	64	EF	D3
0000CA70	52	16	98	82	9A	8A	66	5C	72	6E	00	00	00	00	00	00
0000CA80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000CA90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000CAA0	00	FF	FB	92	64	00	00	02	94	3B	42	C9	E6	18	E0	57
0000CAB0	24	E8	56	31	06	38	0B	31	21	06	A7	A4	63	C1	78	10

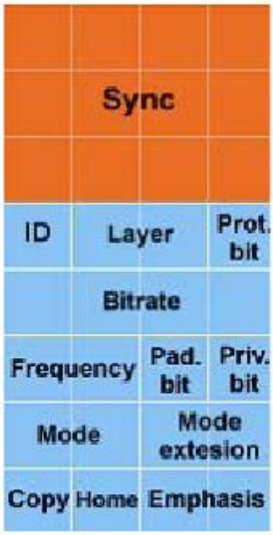
Gambar 2.9. *Frame* dalam MP3

Bagian-bagian detail dari tiap *Frame* adalah :

1. *Header*

   Header adalah bagian dari sebuah *frame* dengan panjang 4 byte (32 bit) yang berisi bit-bit sinkronisasi dan deskripsi tentang *frame* tersebut. Sebelas bit yang pertama adalah sinkronisasi *frame* dan ke-sebelas bit ini selalu bernilai 1. Bit ke-duabelas sampai bit ke-tigapuluhdua berisi informasi mengenai versi dari MPEG, layer, proteksi, *bitrate*, frekuensi cuplik, mode (stereo/mono) dan emphasis.

Anatomi dari *header* sebuah *frame* MP3 adalah sebagai berikut.



Gambar 2.10 Struktur *Header* MP3

Keterangan mengenai kebutuhan pada bit *header* ditunjukkan pada
tabel 2.1

Tabel 2.1 Fungsi dan Kebutuhan Bit *Header*.

Posisi	Keterangan Fungsi	Panjang (Bit)
Sync	*Frame* sinkronisasi	11
Id	Versi MPEG *audio* (MPEG-1, MPEG-2, dll)	2
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
Frequensy	Frekuensi sampling rate	2
Pad Bit	Bit padding	1
Priv. Bit	Private bit	1
Mode	Mode kanal (stereo, joint stereo, dll)	2
Mode	Mode ekstensi	2

Extention		
Copy	Hak cipta	1
Home	original	1
Emphasis	emphasis	2

Beberapa keterangan *bit position* antara lain :

a. *Bit Id*
   Bit ID merupakan kumpulan bit dari nama berkas *audio*
b. *Bit Layer*
   Bit Layer yang terdiri dari dua bit pada *audio* dibedakan menjadi 3, yaitu Layer 1, Layer 2, dan Layer 3 seperti pada tabel 2.2

Tabel 2.2 Bit Layer

Nilai Bit	Layer
00	Tidak terdefinisi
01	Layer 3
10	Layer 2
11	Layer 1

c. Bit Frekuensi *Sampling*
   Sampling frekuensi atau sampling rate didefinisikan sebagai suatu nilai sample per detik yang dihasilkan dari signal tak putus menjadi signal diskrit. Notasinya adalah hertz (Hz). Kebalikan dari sampling frekuensi adalah periode sampling atau waktu sampling, yaitu waktu yang dibutuhkan untuk melakukan sampling. Tidak ada aturan baku yang mengatur berapa batas sampling yang diperkenankan pada satu periode sampling. Pembagian nilai bit frekuensi ditunjukkan pada tabel 2.3

Tabel 2.3 Bit Frekuensi

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

d. Bit *Mode*
   Bit Mode pada *audio* dibedakan menjadi empat yaitu :

1. *Mono*

   Audio mono memiliki kanal *audio* hanya satu (mono). *Audio* mono hanya menghasilkan 1 suara yang didengar oleh kedua telinga kita. sehingga suara yang diterima oleh kedua telinga kita selalu sama. Signal mono adalah R+L (right and Left), dimana R dan L digabungkan, sehingga jadi satu signal R+L. ini dibuat agar bisa mendengar kedua sinyal dalam satu sumber suara.

2. *Stereo*

   Audio yang satu ini dapat menghasilkan effect suara seperti effect doppler, Musik yang berbeda antar kanal, misalnya kanal R bass, yang kiri suara Guitar.

3. *Joint Stereo*

   Joint-stereo adalah satu trik untuk mengkodekan sinyal frekuensi rendah secara mono sementara sisanya tetap stereo.

4. *Dual Chanel*

   Salah satu trik untuk menggabungkan *audio* jenis stereo dan juga *audio* mono

Pembagian bit mode ditunjukkan pada tabel 2.4

Tabel 2.4 Bit Mode

Nilai Bit	Mode
00	Stereo
01	Joint Stereo
10	Dual Chanel
11	Mono

1. 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 memeriksa kesalahan

pada suatu data yang akan ditransmisikan atau disimpan. CRC bekerja secara sederhana, yakni dengan menggunakan perhitungan matematika terhadap sebuah bilangan yang disebut sebagai *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* tersebut akan menghitung kembali apakah *frame* yang diterima benar-benar tanpa kerusakan, dengan membandingkan nilai frame yang dihitung dengan nilai *frame* yang terdapat dalam *header frame*. Jika dua nilai tersebut berbeda, maka *frame* tersebut telah berubah.

2. *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 dekoding Main Data.
3. *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 dekoding Huffman code bits ini terdapat pada bagian Side Information. Bagian *frame* yang akan digunakan dalam steganografi ini adalah dengan mengganti isi dari Main Data tersebut. Teknik kompresi MP3 juga menerapkan 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
4. *Ancilary data*
   Ancillary Data merupakan bagian opsional, tidak banyak terdapat informasi pada bagian ini dan pada umumnya sebuah *frame* tidak mempunyai Ancillary Data.

2.6. *Mean Opinion Score (MOS)*

Mean Opinion Score merupakan sebuah metode numerikal yang bertujuan untuk mengukur tingkat kualitas *video* atau *audio*.

Pengukuran ini didasarkan pada deskripsi kualitatif dari apa yang didengar oleh penilai. *Mean Opinion Score (MOS)* memberikan indikasi numerik tentang kualitas suara yang didapatkan setelah mengalami jalur transmisi atau setelah mengalami pengkodean.

Nilai MOS diapresiasikan dengan sebuah nilai antara satu hingga lima. Nilai satu menunjukan kualitas yang paling buruk, sedangkan nilai lima menunjukan kualitas yang paling baik. Penilaian MOS bersifat sangat subjektif, Hal ini didasari oleh hasil yang dirasakan oleh penilai selama proses pengujian (Unuth, 2012).

Pengujian MOS akan dilakukan untuk menguji kualitas *audio* MP3 yang telah disisipi pesan rahasia. Beberapa orang akan diminta untuk mendengarkan *audio* MP3 hasil proses penyisipan pesan dan akan diminta untuk menilai kualitas *audio* tersebut menggunakan skala satu sampai lima. Kriteria penilaian MOS ditunjukkan pada tabel 2.5

Tabel 2.5 Kriteria Penilaian MOS

Nilai	Kualitas MP3	Kriteria
5	*Excellent*	Tidak terdapat derau
4	*Good*	Derau terdengar namun tidak mengganggu
3	*Fair*	Derau sedikit mengganggu
2	*Poor*	Derau menggangu
1	*Bad*	Derau sangat mengganggu

Hasil penilaian MOS dari beberapa orang akan dujumlahkan dan kemudian akan dicari rata-ratanya. Nilai rata-rata inilah yang akan menjadi nilai akhir MOS dan menjadi nilai kualitas dari MP3 yang diuji.

2.7. Perbandingan Berkas Audio

Perbandingan berkas *audio* dibutuhkan untuk melihat seberapa besar perubahan atau error yang terjadi pada berkas *audio* yang telah mengalami perubahan. Perbandingan dilakukan kepada berkas *audio* awal dan berkas *audio* yang telah mengalami perubahan dengan analisa nilai dan prosentase *Signal to Noise Ratio (SNR)*. untuk menghitung nilai *Signal to Noise Ratio (SNR)* dalam satuan decibel (dB) digunakan persamaan 2.14:

$$SNR = 10.\log_{10}\frac{I}{N} \quad (2.14)$$

I merupakan jumlah sinyal dan *N* adalah jumlah derau, bila diterapkan dalam berkas MP3 maka *I* merupakan jumlah bit dari *audio* asal sebelum mengalami perubahan dan *N* adalah jumlah bit yang berubah setelah proses perubahan berkas.

Semakin tinggi nilai SNR akan semakin baik karena ratio sinyal terhadap derau juga tinggi sehingga kualitas berkas dapat dikatakan bagus (Basuki, 2005). Berkas audio dikatakan layak untuk didengar bila memiliki nilai SNR lebih dari 30 dB (Rasyid, 2009).

Sedangkan untuk menghitung prosentase SNR dapat dipersamaankanseperti pada persamaan 2.15:

$$SNR(\%) = \left(\frac{jumlahTotalBitAwal - jumlahBitTerubah}{jumlahTotalBitAwal}\right)100\% \quad (2.15)$$

JumlahTotalBitAwal merupakan jumlah bit dari *audio* asal sebelum mengalami perubahan, Sedangkan jumlahBitTerubah merupakan jumlah bit yang berubah pada *audio* awal setelah mengalami proses perubahan berkas. Besar prosentase yang dihasilkan menunjukan seberapa besar kesesuaian berkas MP3 yang telah mengalami perubahan dengan berkas MP3 awal (Arubusman,2007).

2.8. *Brute Force Attack*

Brute Force Attack merupakan suatu tindakan yang dibuat untuk mengungkap *plainteks* atau kunci dengan mencoba semua kemungkinan kunci (*trial and error*). Brute Force dilakukan dengan asumsi :

a. Kriptanalis mengetahui algoritma kriptografi
b. Kriptanalis memiliki sebagian *plainteks* dan *chiperteks* yang bersesuaian.

Langkah dari Brute Force Attack adalah plainteks yang diketahui dienkripsikan dengan setiap kemungkinan kunci, dan hasilnya dibandingkan dengan chiperteks yang bersesuaian. Jika hanya chiperteks yang tersedia, chiperteks tersebut didekripsi dengan dengan setiap kemungkinan kunci dan plainteks hasilnya diperiksa apakah mengandung makna. Misalkan sebuah sistem kriptografi

membutuhkan kunci yang panjangnya 8 karakter, karakter dapat berupa angka (10 buah), huruf (26 huruf besar dan 26 huruf kecil), maka jumlah kunci yang harus dicoba adalah

$$62 \times 62 \times 62 \times 62 \times 62 \times 62 \times 62 \times 62 = 628 \text{ buah.}$$

Secara teori, serangan secara exhaustive ini dipastikan berhasil mengungkap plainteks tetapi dalam waktu yang sangat lama (Ratna, 2007).

Dalam algoritma RSA, *Brute Force Attack* dilakukan untuk memfaktorkan kunci publik pertama menjadi bilangan prima p dan q dengan persamaan 2.16 :

$$q = PK_1 / p \qquad (2.16)$$

dimana PK_1 merupakan kunci publik pertama sedangkan p dan q adalah bilangan prima. Nilai p diinisialisasi sesuai hasil iterasi dengan cara *Brute Force*. bila p dan q berhasil difaktorkan maka pada persamaan 2.1 yaitu $\phi(n) = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya, karena kunci publik enkrispi atau kunci publik kedua juga tidak rahasia, maka kunci dekripsi atau kunci rahasia dapat dihitung dari persamaan 2.11 yaitu $a = b^{-1} \bmod \phi(n)$, dimana a merupakan kunci rahasia dan b merupakan kunci publik enkrispi atau kunci publik kedua. Bila kunci rahasia yang diperoleh dari proses *Brute Force Attack* sama dengan kunci rahasia asli maka *Brute Force Attack* dianggap telah berhasil. (Ariyus, 2008).

Halaman ini sengaja dikosongkan

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Bab ini berisi penjelasan mengenai metode dan langkah-langkah perancangan sistem yang dilakukan dalam penelitian. Langkah-langkah yang akan dilakukan ditunjukkan pada gambar 3.1

Gambar 3.0.1 Langkah-Langkah Penelitian

Berdasarkan gambar 3.1, langkah langkah yang dilakukan dalam penelitian ini adalah sebagai berikut :

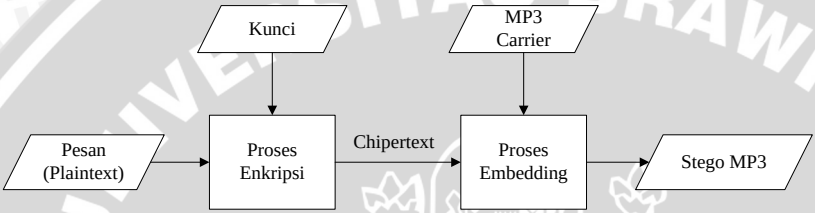
1. Melakukan studi literatur mengenai karakteristik berkas MP3, penggunaan kriptografi RSA, dan juga penyisipan pesan melalui teknik steganografi LSB.
2. Melakukan perancangan sistem.
3. Mengimplementasikan perancangan sistem yang telah dibuat kebentuk perangkat lunak yang dapat menyisipkan pesan rahasia ke dalam berkas MP3.

4. Melakukan hasil uji coba terhadap perangkat lunak yang telah dibuat.
5. Menganalisa hasil uji dan mengevaluasi hasil tersebut.

3.1. Skenario Proses

3.1.1. Penyisipan Pesan

Skenario proses penyisipan pesan atau skema penyisipan pesan ditunjukkan pada gambar 3.2.

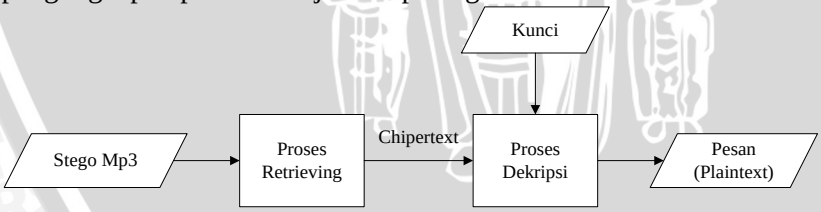


Gambar 3.2. Skenario proses penyisipan pesan

Dari gambar 3.2 diperlihatkan bahwa pesan atau *plaintext* akan dienkripsi terlebih dahulu. Hasil dari proses enkripsi atau *chipertext* kemudian akan disisipkan kedalam MP3 *Carrier* dengan melakukan proses *embedding*. Hasil dari proses *embedding* adalah MP3 yang telah disisipi pesan atau yang disebut stego MP3.

3.1.2. Pengungkapan Pesan

Skenario proses pengungkapan pesan atau skema pengungkapan pesan ditunjukkan pada gambar 3.3.



Gambar 3.3. Skenario proses pengungkapan pesan

Dari gambar 3.3 diperlihatkan bahwa file stego MP3 atau MP3 yang telah disisipi pesan terlebih dahulu dilakukan proses *retrieving* untuk mendapatkan pesan berupa chipertext. Pesan chipertext yang telah didapatkan kemudian akan didekripsi untuk mendapatkan pesan plaintext.

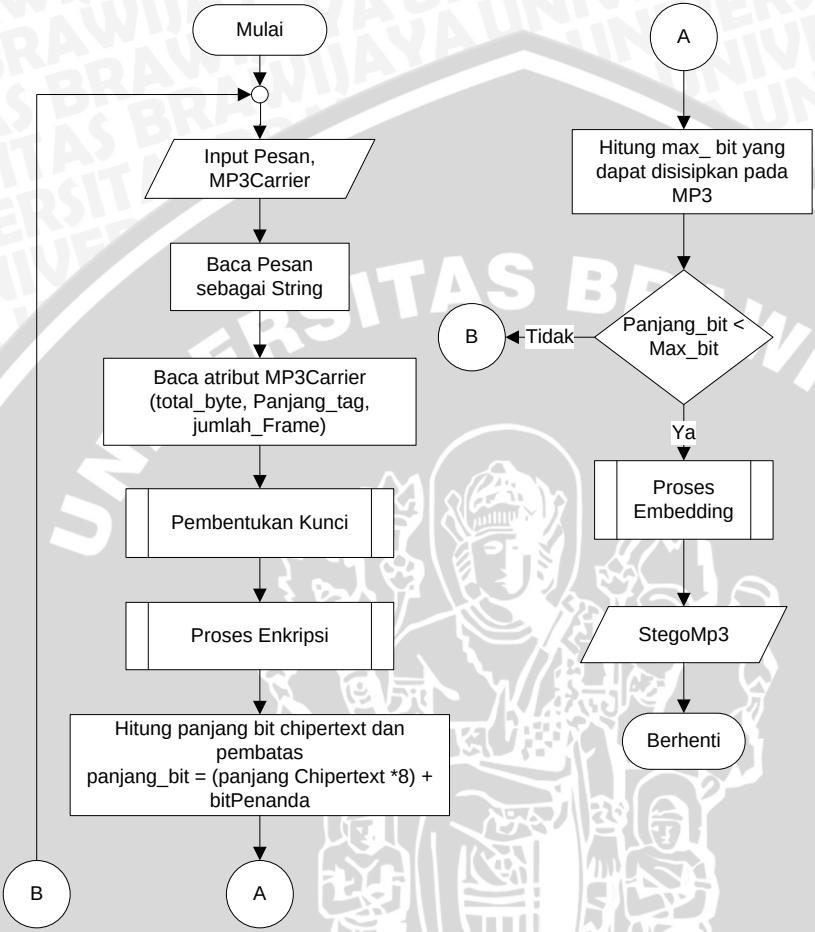
3.2. Analisa dan Perancangan Sistem

3.2.1. Deskripsi Umum Sistem

Perangkat lunak yang akan dibuat merupakan perangkat lunak yang dapat menyisipkan pesan rahasia berupa teks ke dalam sebuah berkas dengan format MP3. Perangkat lunak akan mengimplementasikan teknik kriptografi dengan algoritma RSA dan juga teknik steganografi dengan metode *Least Bit Significant (LSB)*. Teknik kriptografi RSA digunakan untuk mengenkripsi pesan agar arti dari pesan tersebut tidak mudah diketahui oleh pihak lain. Pesan yang telah dienkripsi kemudian akan disisipkan ke dalam *berkas* berformat MP3. Penyisipan akan dilakukan dengan menggunakan teknik steganografi dengan metode *Least Significan Bit t (LSB)* yang memanfaatkan bit yang kurang berarti dalam berkas MP3. Untuk dapat mengungkapkan arti pesan yang telah disisipkan ke dalam MP3 akan dilakukan dengan pengambilan bit-bit terakhir dalam tiap byte MP3 kemudaian menyusunnya menjadi bentuk karakter ASCII yang kemudian akan dilakukan proses dekripsi menggunakan algoritma RSA.

3.2.2. Perancangan Sistem

Perangkat lunak yang akan dibuat memiliki dua proses utama, yaitu proses penyisipan pesan dan proses pengungkapan pesan. Pada proses penyisipan pesan yang digambarkan pada gambar 3.2 memperlihatkan bahwa proses ini melibatkan beberapa proses lain yaitu pembentukan kunci dan enkripsi, yang merupakan bagian dari algoritma RSA, dan juga proses *embedding* yang menjadi inti dari proses penyisipan data.



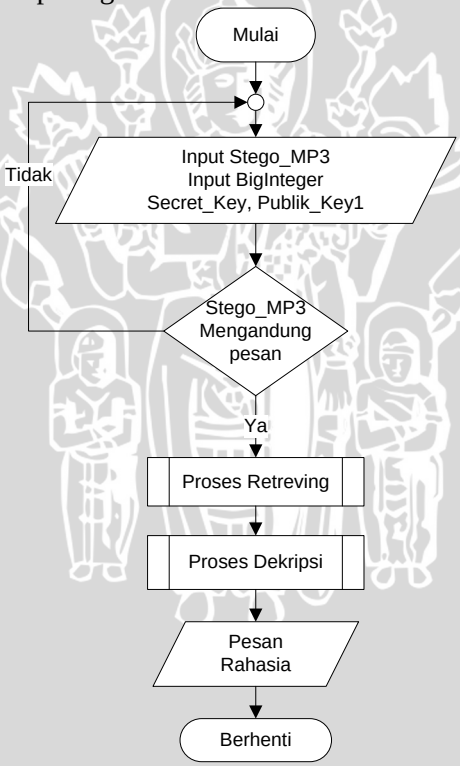
Gambar 3.4 Diagram alir penyisipan data

Berikut adalah penjelasan mengenai proses penyisipan data.

1. Pesan dan berkas MP3 yang akan digunakan sebagai media penampung (MP3Carrier) diinputkan ke dalam sistem
2. Pesan dan atribut yang ada pada MP3Carrier akan dibaca oleh sistem.

3. Proses selanjutnya adalah pembentukan kunci publik dan kunci rahasia yang akan digunakan dalam proses enkripsi dan dekripsi.
4. Proses penyisipan dilanjutkan dengan proses enkripsi yang akan menghasilkan chipertext.
5. Dihitung panjang bit dari chipertext untuk selanjutnya dibandingkan dengan maksimal bit yang dapat disisipkan ke dalam MP3..
6. Dilakukan proses *embedding* untuk menyisipkan pesan. Proses ini akan menghasilkan berkas MP3 yang telah disisipi bit pesan.

Proses utama selanjutnya adalah proses pengungkapan pesan. Proses ini berfungsi untuk mengambil data yang telah disisipkan ke dalam berkas MP3. Diagram alir proses pengungkapan pesan ditunjukkan pada gambar 3.5



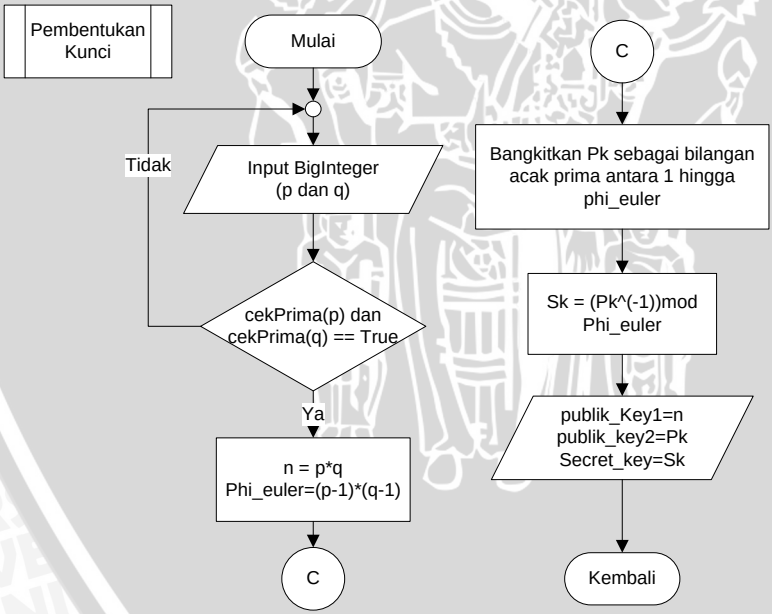
Gambar 3.5 Diagram alir pengungkapan pesan

Berikut ini adalah penjelasan mengenai proses pengungkapan data.

1. Berkas MP3 yang telah disisipi pesan (stegoMP3), kunci rahasia (Secret_Key), dan kunci publik pertama (Publik_Key1) dimasukan ke dalam sistem.
2. Dilakukan proses *retrieving* untuk mendapatkan data terenkripsi dari berkas MP3.
3. Data terenkripsi yang telah didapat kemudian didekripsi untuk mendapatkan pesan rahasia kembali.

3.2.2.1. Pembentukan Kunci

Proses pembentukan kunci merupakan bagian dari algoritma kriptografi RSA. Proses pembentukan kunci ini akan menghasilkan kunci publik dan kunci rahasia yang nantinya akan digunakan dalam proses enkripsi pada penyisipan data dan proses dekripsi pada pengungkapan data. Diagram alur pembentukan kunci ditunjukkan pada gambar 3.6

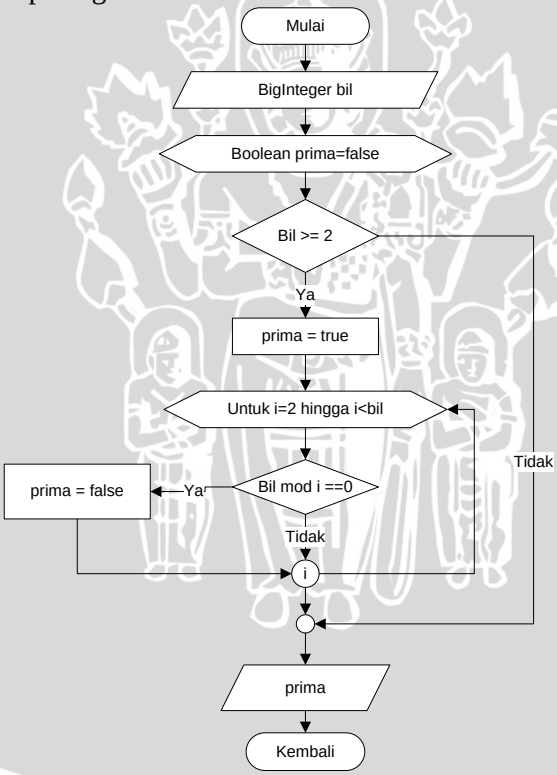


Gambar 3.6 Diagram Alir Pembentukan Kunci

Berikut ini adalah penjelasan dan alur dari proses pembentukan kunci.

1. Dimasukan dua bilangan yang keduanya merupakan bilangan prima (p dan q).
2. Kalikan kedua bilangan tersebut untuk mendapatkan nilai n.
3. Dihitung nilai phi euler ($\phi(n)$).
4. Dibangkitkan kunci publik prima antara 1 hingga phi euler $\phi(n)$.
5. Dihitung kunci rahasia dengan persamaan $Pk^{-1} \bmod \phi(n)$, dimana Pk adalah kunci publik.
6. Didapatkan dua kunci publik (publik_Key1 dan publik_Key2) dan kunci rahasia (Secret_key).

Sedangkan untuk diagram alir pengecekan bilangan prima ditunjukkan pada gambar 3.7



Gambar 3.7 Diagram Alir Pengecekan Bilangan Prima

Berikut ini adalah penjelasan dan alur pengecekan bilangan prima.

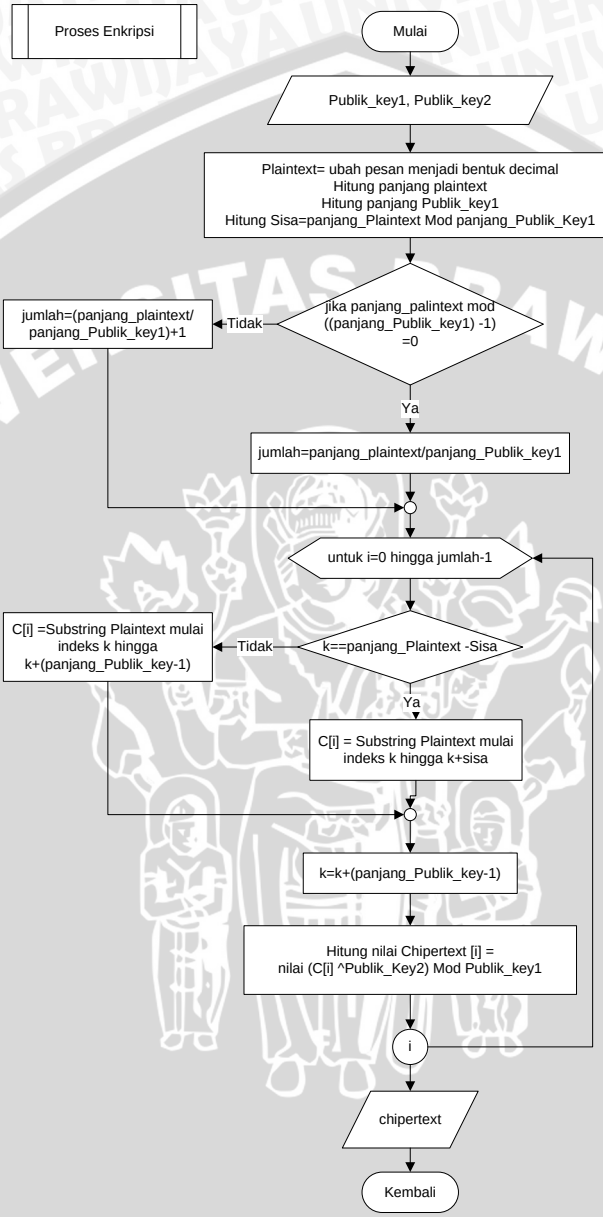
1. Dimasukan sebuah bilangan bil.
2. Dilakukan inisialisasi Boolean prima bernilai salah
3. Dilakukan pengecekan apakah bil lebih dari sama dengan dua atau bukan. Jika tidak maka hasil akhir menunjukan bahwa bil bukanlah bilangan prima.
4. Bila bil bernilai lebih dari sama dengan dua maka nilai prima adalah benar.
5. Dilakukan perulangan mulai i=2 hingga i bernilai sama dengan bil. Bila bil habis dibagi i maka prima bernilai salah dan perulangan berhenti, bila bil tidak habis dibagi i maka perulangan akan dilanjutkan hingga ditemukan bil habis dibagi i atau seluruh nilai i dalam perulangan telah diproses.

3.2.2.2. Enkripsi

Proses enkripsi merupakan proses untuk mengubah pesan menjadi chipertext agar pesan tetap terjaga kerahasiaanya. Proses ini memanfaatkan kunci publik dari proses pembentukan kunci.

Berikut ini adalah langkah-langkah proses enkripsi yang digambarkan pada gambar 3.8

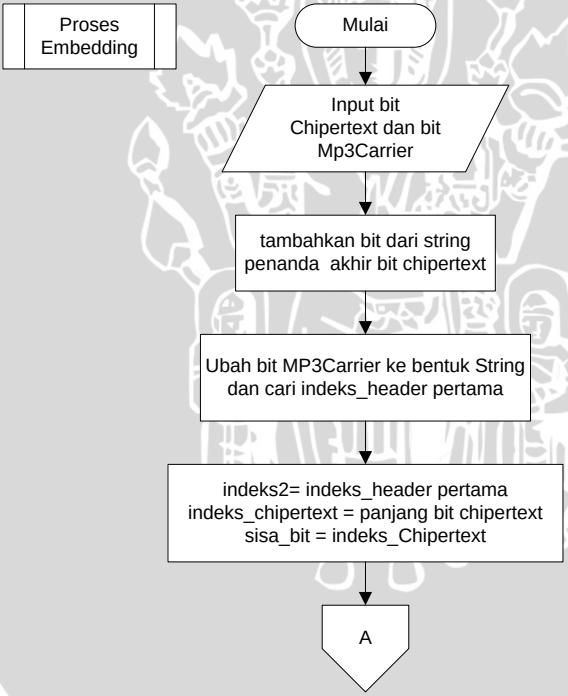
1. Dimasukan dua kunci publik(publik_Key1 dan publik_Key2) lalu ubah plaintext menjadi desimal.
2. Dihitung panjang plaintext atau pesan dan juga panjang kunci publik yang pertama.
3. Panjang plaintext akan dibagi menjadi bilangan-bilangan yang nilainya kurang dari kunci publik pertama dengan membatasi panjang digit bilangan = panjang_Publik_key1 – 1.
4. Dihitung banyak bilangan yang dapat dihasilkan dari plaintext.
5. Dilakukan perulangan sepanjang banyaknya bilangan yang dapat dihasilkan (jumlah) untuk selanjutnya dilakukan pemisahan plaintext yang akan ditampung pada array C.
6. Plaintext yang telah dipisahkan akan dijadikan chipertext dengan persamaan C^{Publik_key2} Mod Publik_key1.

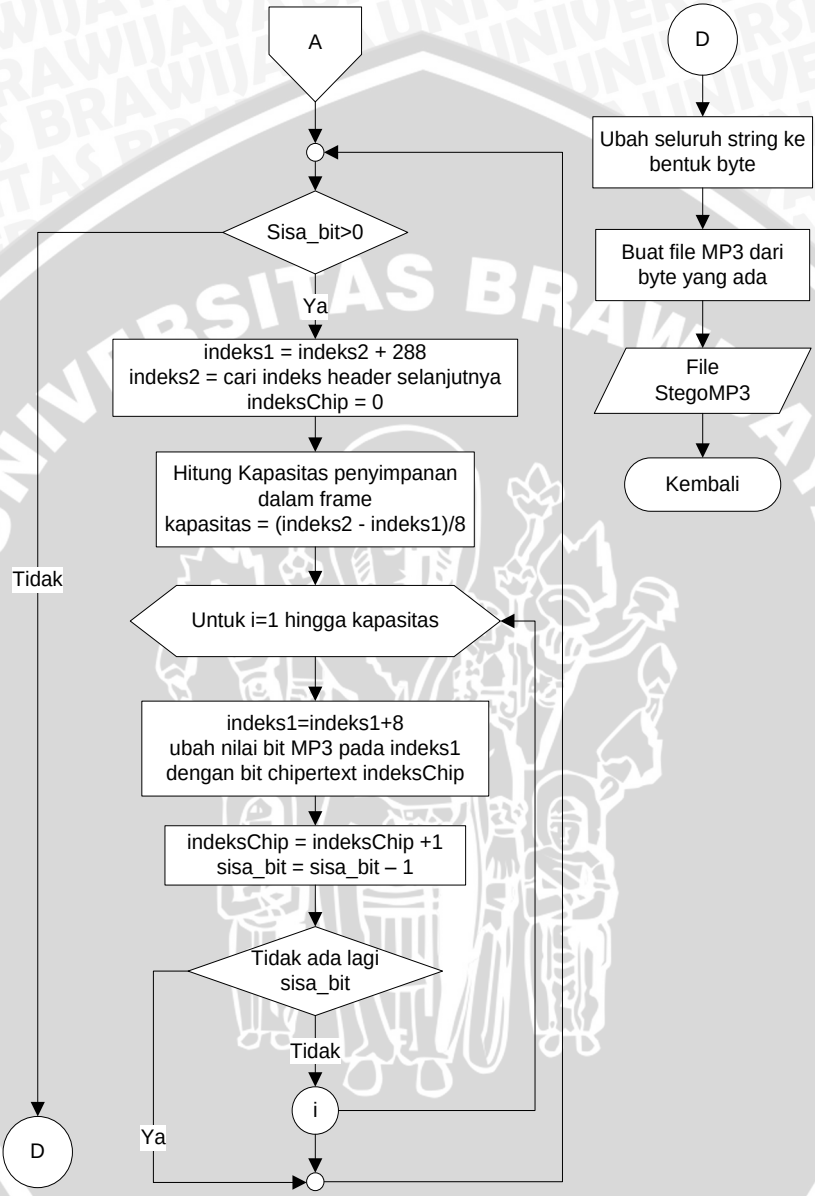


Gambar 3. 8 Diagram alir enkripsi

3.2.2.3. *Embedding*

Proses *embedding* merupakan proses utama dalam penyisipan data. Pada proses ini diterapkan metode Least Significant Bit (LSB). Proses ini membaca bit dari MP3 kemudian menggantikan bit yang kurang signifikan dengan bit pesan. Pada proses ini dibedakan menjadi dua mode yaitu mode *standard* dan mode penyisipan per-50 *frame*. Mode *standard* adalah penyisipan bit pesan kedalam main data dari tiap *frame* yang berurutan dan dimulai dari *frame* pertama MP3 sedangkan penyisipan dengan mode per 50 *frame* adalah penyisipan bit pesan kedalam main data dari *frame*, akan tetapi tidak secara berurutan melainkan menyisipkan bit pesan tiap 50 *frame* dimulai dari *frame* pertama. Diagram alir proses *embedding* pada mode *standard* ditunjukkan pada gambar 3.9



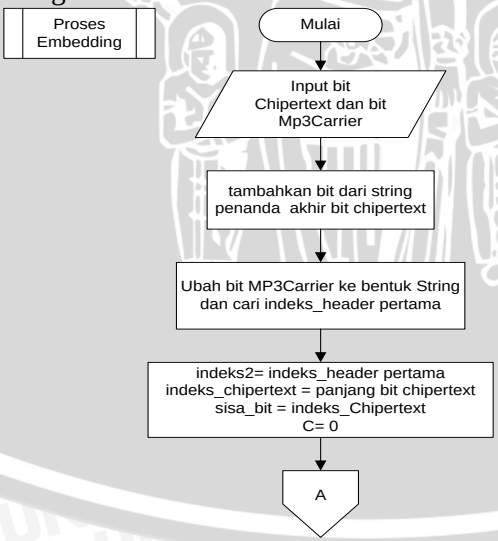


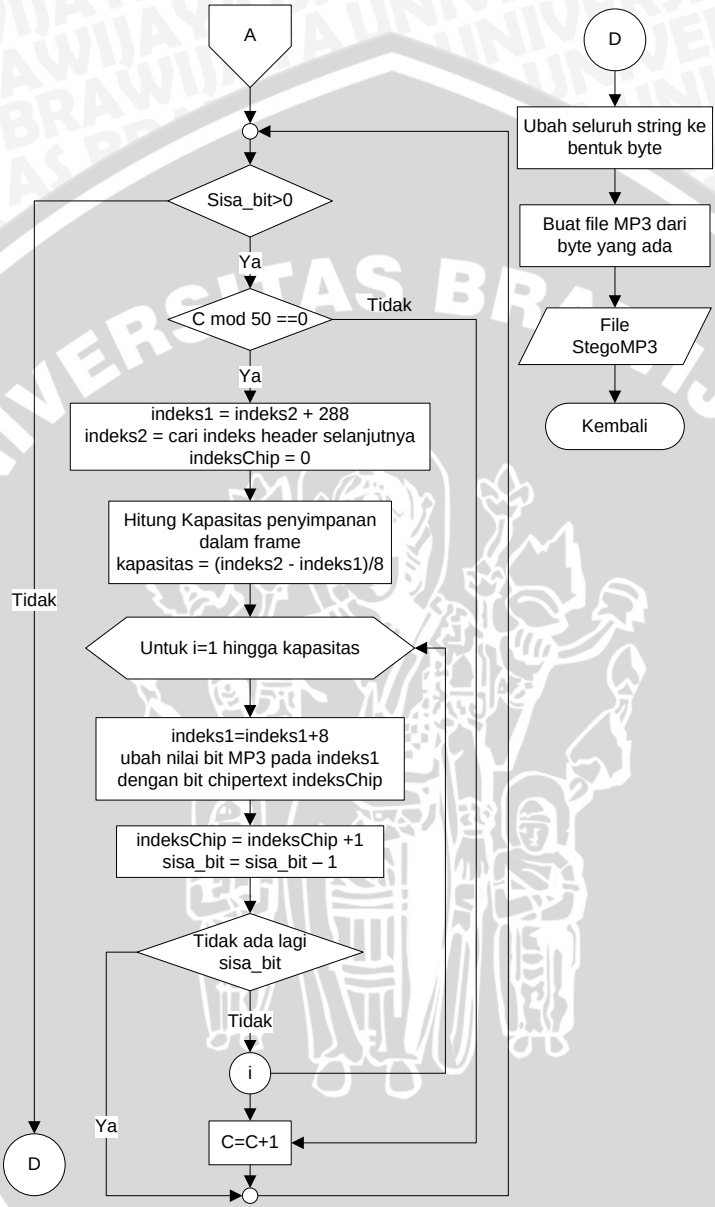
Gambar 3.9 Diagram Alir *Embedding Mode Standard*

Berikut ini adalah penjelasan dari diagram alir *embedding* untuk *mode standard*.

1. Dimasukan ke dalam sistem, bit dari chipertext dan bit MP3Carrier.
2. Tambahkan bit penanda berupa bit dari karakter # diakhir dari bit chipertext.
3. Dicari indeks *frame header* pertama.
4. Digunakan pointer indeks2 dan indeks1 untuk menandai awal dan akhir suatu *frame* pada MP3.
5. Dilakukan perulangan selama masih ada bit pesan maka indeks1 maju hingga akhir *side information* pada *frame* dan indeks2 maju hingga indeks *header* selanjutnya. Hal ini berarti pointer menunjukan awal dan akhir *main data* pada *frame*.
6. Dihitung jumlah maksimum bit yang dapat disisipkan ke dalam *frame* tersebut (kapasitas).
7. Diakukan perulangan untuk mengubah tiap LSB pada byte dalam *main data* pada *frame* tersebut dengan memajukan pointer indeks1. Nilai bit pada indeks1 diganti dengan nilai bit dari indeksChip
8. Bila tidak ditemukan lagi bit untuk disisipkan maka bit yang telah diubah akan dijadikan berkas MP3 baru.

Untuk diagram alir proses *embedding* pada mode *per-50 frame* ditunjukkan pada gambar 3.10





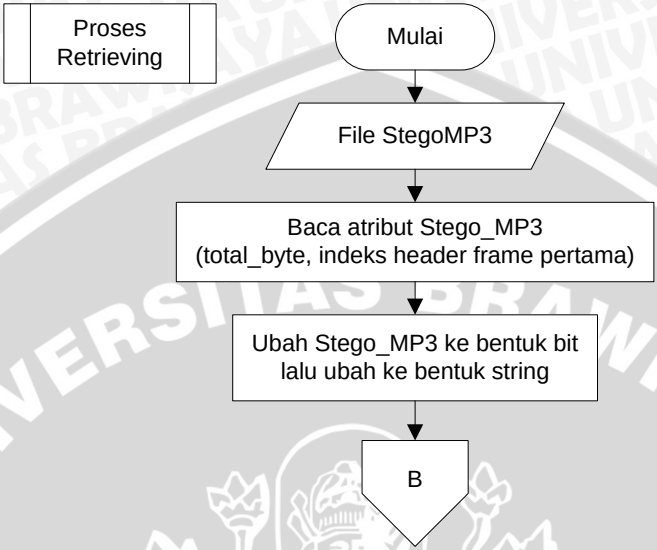
Gambar 3.10 Diagram Alir *Embedding Mode per-50 frame*

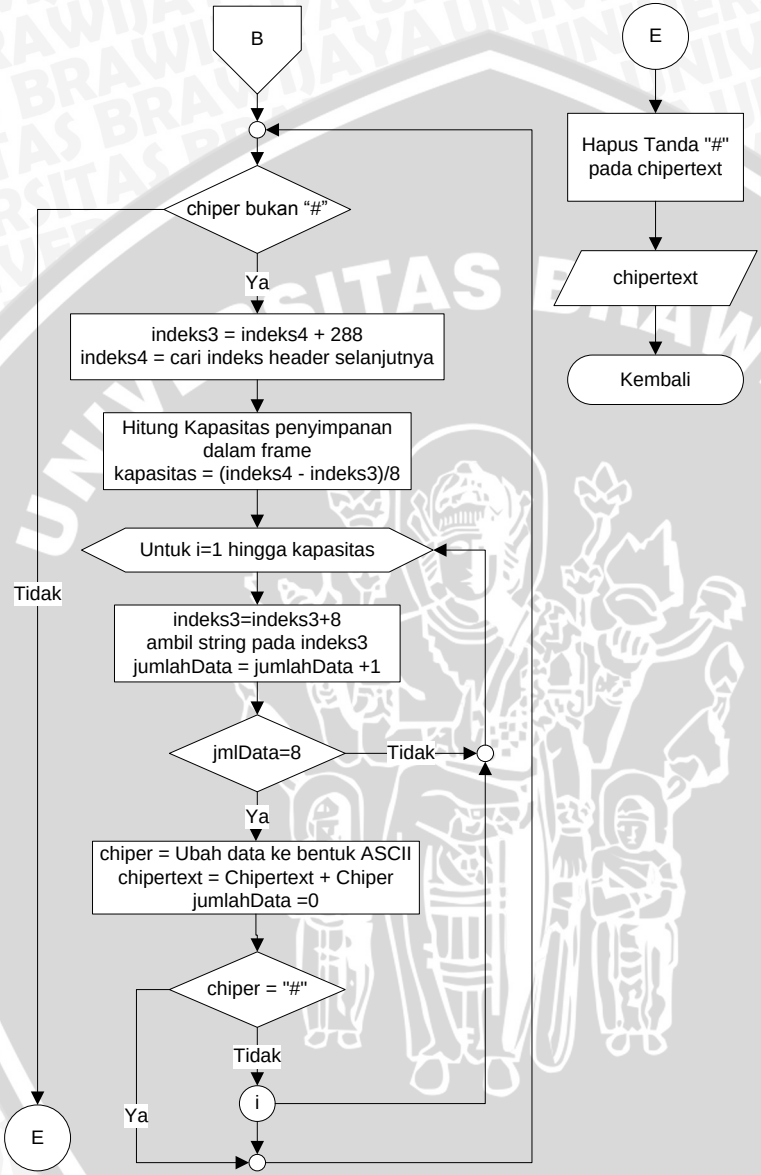
Berikut ini adalah penjelasan dari diagram alir *embedding*.

1. Dimasukan ke dalam sistem, bit dari chipertext dan bit MP3Carrier.
2. Tambahkan bit penanda berupa bit dari karakter # diakhir dari bit chipertext.
3. Dicari indeks *frame header* pertama.
4. Digunakan pointer indeks2 dan indeks1 untuk menandai awal dan akhir suatu *frame* pada MP3.
5. Dilakukan perulangan selama masih ada bit pesan maka maka dicek apakah *frame* tersebut merupakan kelipatan 50 atau bukan. Bila *frame* merupakan *frame* kelipatan 50 maka indeks1 maju hingga akhir *side information* pada *frame* dan indeks2 maju hingga indeks *header* selanjutnya. Hal ini berarti pointer menunjukan awal dan akhir *main data* pada *frame*.
6. Dihitung jumlah maksimum bit yang dapat disisipkan ke dalam *frame* tersebut (kapasitas).
7. Diakukan perulangan untuk mengubah tiap LSB pada byte dalam *main data* pada *frame* tersebut dengan memajukan pointer indeks1. Nilai bit pada indeks1 diganti dengan nilai bit dari indeksChip
8. Bila tidak ditemukan lagi bit untuk disisipkan maka bit yang telah diubah akan dijadikan berkas MP3 baru.

3.2.2.4. *Retrieving*

Retrieving merupakan proses inti dalam pengungkapan data. Proses ini akan mengumpulkan bit-bit yang telah disisipkan ke dalam berkas MP3. Pada proses ini dibedakan menjadi dua mode yaitu mode *standard* dan mode penyisipan per-50 *frame*. Mode *retrieving standard* dilakukan untuk mendapatkan chipertext yang telah disisipkan menggunakan *embedding* mode stadart sedangkan *retrieving* mode per-50 *frame* dilakukan untuk mengungkapkan chipertext yang telah disisipkan menggunakan *embedding* mode per-50 *frame*. Diagram alir dari proses *retrieving* untuk mode *standard* ditunjukkan pada gambar 3.11


Proses Retrieving
Mulai
File StegoMP3
Baca atribut Stego_MP3 (total_byte, indeks header frame pertama)
Ubah Stego_MP3 ke bentuk bit lalu ubah ke bentuk string
B

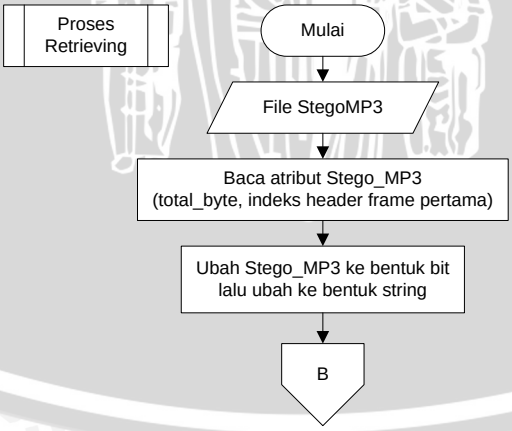


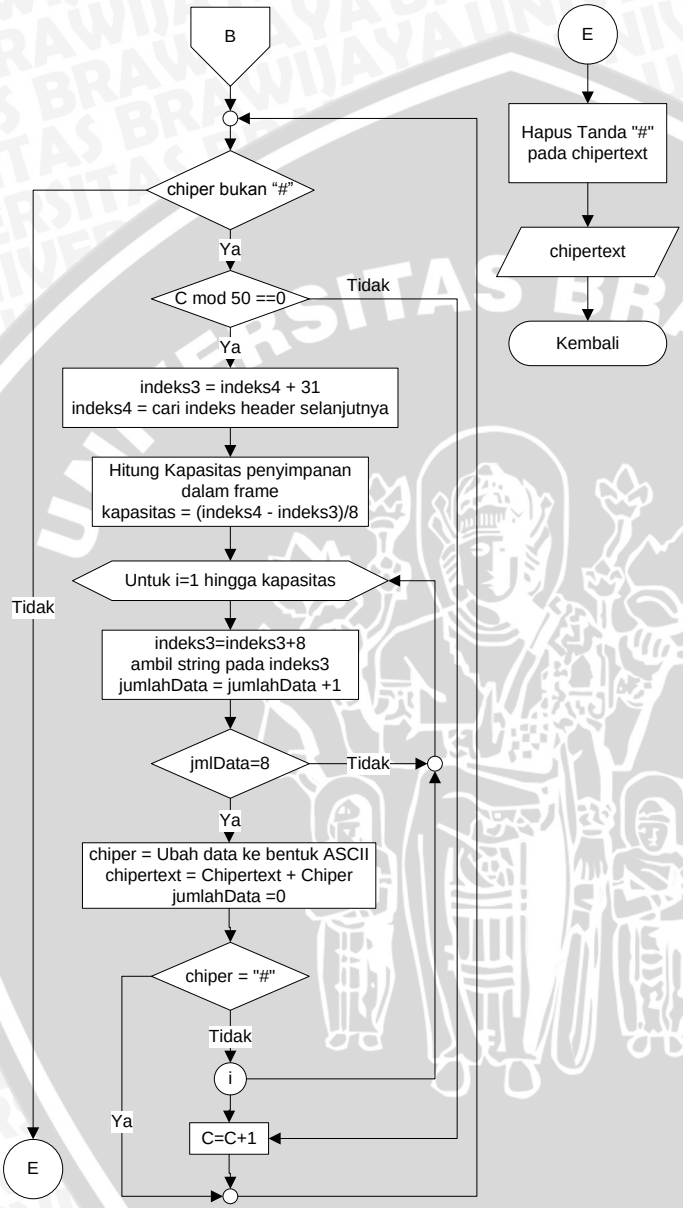
Gambar 3.11 Diagram Alir *Retrieving* mode *standard*

Berikut ini adalah penjelasan dari diagram alir proses *retrieving*.

1. Dimasukan berkas MP3 yang telah disisipi pesan ke dalam sistem.
2. Dibaca indeks *header frame* pertama dari MP3
3. Berkas MP3 diubah kebentuk bit yang selanjutnya diubah ke bentuk string agar mudah diproses.
4. Dilakukan perulangan selama pesan yang terungkap bukan karakter "#".
5. Digunakan pointer indeks3 dan indeks4 untuk menandai awal dan akhir suatu *frame*, indeks3 maju hingga akhir *side information* pada *frame* dan indeks4 maju hingga indeks *header* selanjutnya. Hal ini berarti pointer menunjukan awal dan akhir *main data* pada *frame*.
6. Dihitung kapasitas penyimpanan dalam *main data frame* tersebut.
7. Dilakukan perulangan sebanyak jumlah kapasitas penyimpanan pada *frame.*
8. Ambil nilai bit pada LSB dengan memajukan indeks3.
9. Bila bit yang terkumpul telah berjumlah delapan bit maka bit tersebut diubah ke bentuk desimal, dan jumlah bit akan dihitung mulai dari 0 lagi.
10. Proses akan berhenti ketika telah diperoleh pesan dengan karakter "#".
11. Tanda "#" akan dihapus dan diperoleh pesan.

Diagram alir dari proses *retrieving* untuk mode *per-50 frame* ditunjukkan pada gambar 3.12

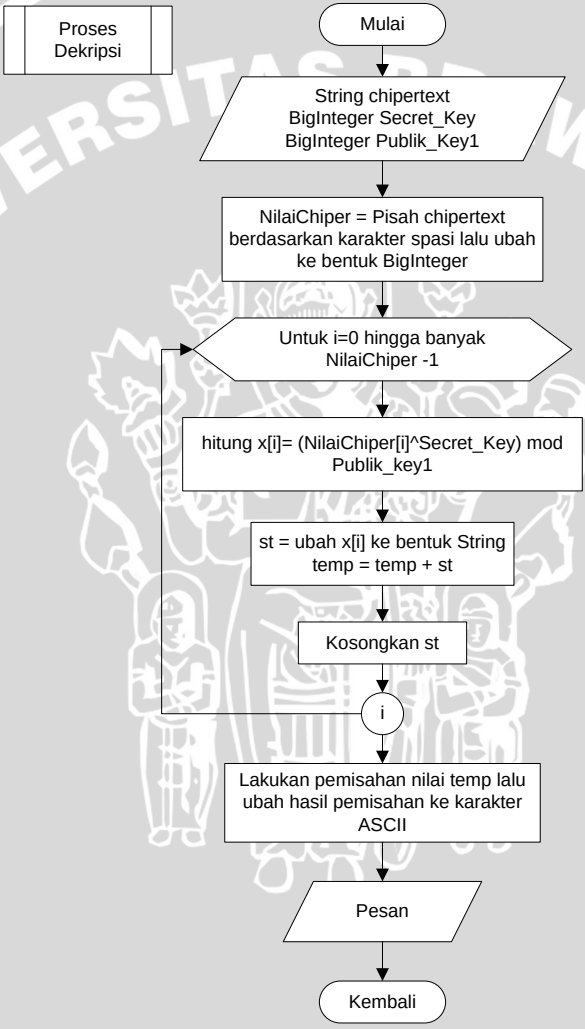




Gambar 3.12 Diagram Alir *Retrieving* mode per-50 *frame*

3.2.2.5. Dekripsi

Proses Dekripsi merupakan tahapan atau langkah akhir dalam pengungkapan pesan. Proses ini akan mendekripsi pesan yang telah didapat dari proses *retrieving* sehingga pesan dapat dibaca dan dimengerti artinya kembali. Diagram alir proses dekripsi ditunjukkan pada gambar 3.13



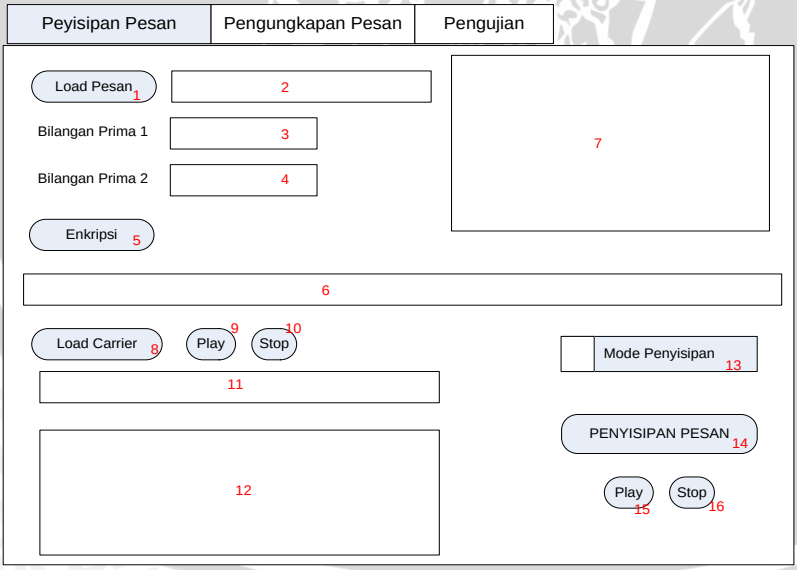
Gambar 3.13 Diagram Alir Dekripsi

Penjelasan mengenai diagram alir dekripsi adalah sebagai berikut.

1. Chipertext yang telah didapat dari proses *retrieving* dan juga kunci publik pertama (Publik_key1) dan kunci rahasia (Secret_Key) menjadi masukan dalam proses ini.
2. Chipertext akan dipisah berdasarkan karkter spasi lalu diubah ke bentuk bilangan dan akan disimpan sebagai array NilaiChiper..
3. Dilakukan perulangan sebanyak NilaiChiper untuk menghitung nilai dekripsi (x) yaitu $\text{NilaiChiper}^{\text{Secret\_Key}}$ mod Publik_key1.
4. Untuk setiap nilai dekripsi (x) yang terkumpul akan disatukan dan diubah ke bentuk string lalu disimpan ke variable temp.
5. Bentuk string dari seluruh nilai dekripsi akan dipisah dan dijadikan ke bentuk karakter ASCII .

3.3. Perancangan Antarmuka

Antarmuka atau *interface* pada sistem ini terdiri dari tiga bagian utama, yaitu bagian yang berfungsi untuk penyisipan pesan, bagian yang berfungsi untuk pengungkapan pesan, dan yang terakhir adalah bagian untuk pengujian. Untuk bagian antarmuka penyisipan pesan ditunjukkan pada gambar 3.14

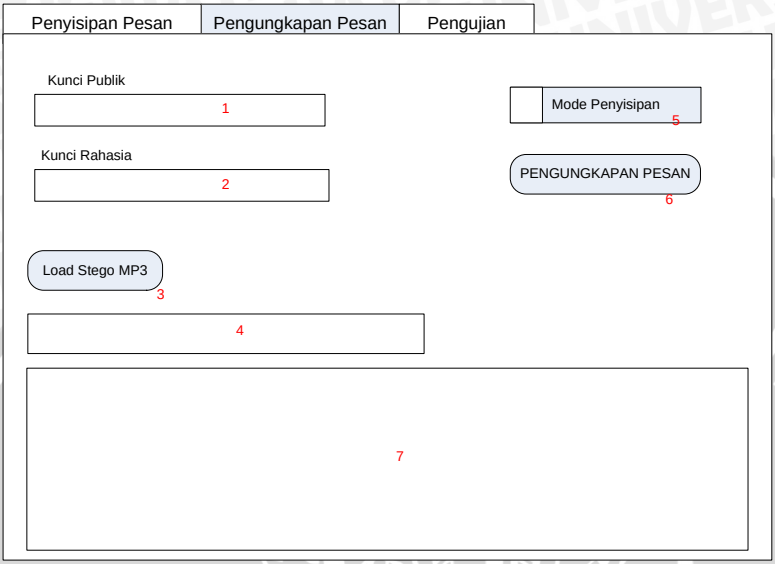


Gambar 3.14 Antarmuka penyisipan pesan

Antarmuka penyisipan pesan pada gambar 3.12 terdiri dari :

1. Tombol *load* pesan yang berfungsi untuk menginputkan berkas dengan format .txt sebagai pesan.
2. *Texfield* untuk menampilkan direktori dari file pesan.
3. *Textfield* sebagai masukan bilangan prima yang pertama.
4. *Textfield* sebagai masukan bilangan prima yang kedua.
5. Tombol enkripsi yang berguna untuk mengenkripsi pesan.
6. *Textarea* yang berfungsi untuk menampilkan hasil enkripsi.
7. *Textarea* yang berfungsi untuk menampilkan informasi hasil enkripsi beserta kunci publik dan rahasia.
8. Tombol *load carrier* yang berfungsi untuk menginputkan berkas MP3 yang akan digunakan sebagai *carrier*.
9. Tombol play yang berfungsi untuk memutar musik mp3 *carrier*.
10. Tombol stop yang berfungsi untuk menghentikan mp3 *carrier* yang sedang dimainkan.
11. *Textfield* yang akan menunjukan direktori dari berkas MP3.
12. *Textarea* yang berfungsi untuk menampilkan *properties* atau data dari berkas MP3.
13. *Combo box* yang berguna untuk pemilihan mode dari penyisipan pesan kedalam mp3.
14. Tombol penyisipan pesan yang berfungsi untuk melakukan proses penyisipan pesan ke dalam berkas MP3.
15. Tombol play yang berfungsi untuk memutar musik mp3 yang telah disisipi pesan.
16. Tombol stop yang berfungsi untuk menghentikan mp3 hasil penyisipan pesan yang sedang dimainkan

Untuk bagian antarmuka pengungkapan pesan ditunjukkan pada gambar 3.15.

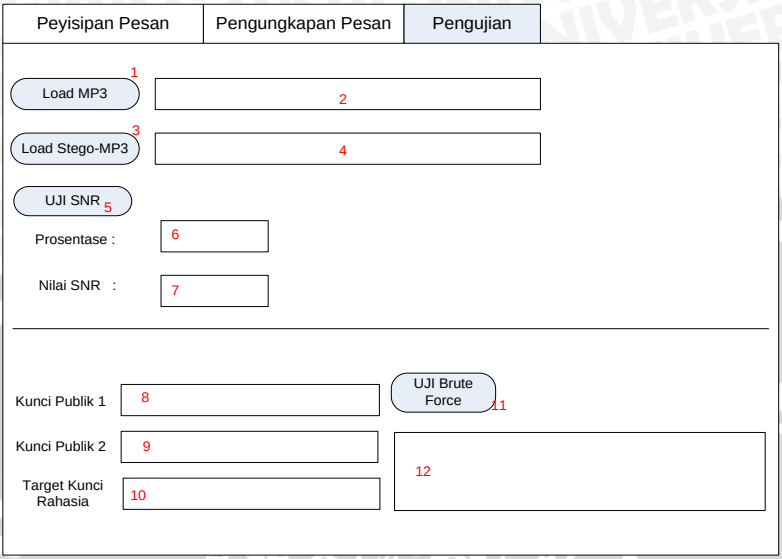


Gambar 3.15 Antarmuka Pengungkapan Pesan

Antarmuka penyisipan pesan pada gambar 3.13 terdiri dari :

1. *Textfield* sebagai masukan kunci publik.
2. *Textfield* sebagai masukan kunci rahasia.
3. Tombol *load* stego MP3 yang berfungsi untuk menginputkan berkas MP3 yang telah disisipi pesan.
4. *Textfield* yang berfungsi untuk menampilkan direktori dari berkas MP3.
5. *Combo box* yang berguna untuk pemilihan mode pengungkapan pesan di dalam mp3.
6. Tombol pengungkapan pesan yang berfungsi untuk melakukan proses pengungkapan pesan.
7. *Textarea* yang berfungsi untuk menampilkan hasil pesan.

Sedangkan untuk bagian antarmuka pengujian ditunjukkan pada gambar 3.16.



Gambar 3.16 Antarmuka pengujian

Antarmuka pengujian pada gambar 3.14 terdiri dari :

1. Tombol *load* MP3 yang berfungsi untuk menginputkan berkas MP3 awal sebelum dilakukan penyisipan pesan.
2. *Textfield* yang menunjukan direktori dari berkas MP3 awal.
3. Tombol *load* stego MP3 yang berfungsi untuk menginputkan berkas MP3 setelah dilakukan penyisipan pesan.
4. *Textfield* yang menunjukan direktori dari berkas MP3 yang telah dilakukan penyisipan pesan.
5. Tombol uji yang berfungsi untuk melakukan proses uji dan perhitungan *Signal to Noise Ratio (SNR).*
6. *Textfield* yang berfungsi untuk menampilkan hasil uji dan perhitungan nilai *Signal to Noise Ratio (SNR).*
7. *Textfield* yang berfungsi untuk menampilkan hasil uji dan perhitungan nilai prosentase *Signal to Noise Ratio (SNR).*
8. *Textfield* sebagai inputan kunci publik pertama yang akan dilakukan pemfaktoran.
9. *Textfield* sebagai inputan kunci publik kedua yang akan digunakan untuk mencari kunci rahasia.
10. *Textfield* sebagai inputan target kunci rahasia yang akan digunakan untuk pembanding proses *Brute Force Attack.*

11. Tombol uji yang akan melakukan proses *Brute Force Attack*.
12. *Textarea* yang berfungsi untuk menampilkan waktu yang dibutuhkan dalam proses *brute force attack.*

3.4. Perhitungan Manual

Pada perhitungan manual ini akan dilakukan tiga proses yaitu penyisipan pesan berupa karakter " t " ke dalam nilai bit pada *carrier,* mengungkapkan kembali pesan yang telah disispkan ke dalam bit *carrier,* dan menghitung prosentase SNR.

3.4.1. Penyisipan Pesan

Berikut adalah langkah-langkah dari penyisipan pesan.

1. Dilakukan inisialisai dua bilangan prima, misalkan nilai p = 17 dan q = 23.
2. Inisialisasi kunci publik pertama.
   $n = p * q = 17 * 23 = 391$.
3. Dicari nilai dari $\phi(n)$ dengan menggunakan persamaan 2.1.
   $\phi(n) = (p-1)*(q-1) = (17-1)*(23-1) = 352$.
4. Dipilih sebuah bilangan antara 0- $\phi(n)$ yang relatif prima terhadap $\phi(n)$ sebagai nilai kunci publik kedua.
   Dipilih b = 29, karena FPB(b, $\phi(n)$) =1
5. Dicari kunci rahasia dengan menggunkan persamaan 2.11.
   $a = b^{-1} \bmod \phi(n) = 29^{-1} \bmod 352$
   untuk mempermudah perhitungan maka akan digunakan persamaan 2.3 atau persamaan Euclidean.
   $r_0=352$, $r_1=29$

n=1	352 = 12.29 + 4	$r_1 = 29$	$q_1 = 12$
n=2	29 = 7.4 +1	$r_2 = 4$	$q_2 = 7$
n=3	4=4.1	$r_3 = 1$	$q_3 = 4$

   selanjutnya akan digunakan persamaan rekurensi pada persamaan 2.6.
   $t_2 = t_0 - q_1\ t_1 = 0 - (12.1) = -12$
   $t_3 = t_1 – q_2\ t_2 = 1 - (7(-12)) = 85$; $29^{-1} \bmod 352 = 85$
6. Didapat :
   Kunci publik pertama (PK_1) = n =391.
   Kunci publik kedua (PK_2) = b = 29.
   Kunci rahasia (SK) = a = 85.
7. Karakter *plaintext* "t" diubah ke dalam bentuk desimal, t = 116.

8. Representasi desimal *plaintext* dipecah menjadi $x_1 = 11$, $x_2 = 6$.
9. Dilakukan enkripsi dengan menggunkan persamaan 2.12

   $x_1^{PK2} \bmod PK_1 = 11^{29} \bmod 391$

   $x_2^{PK2} \bmod PK_1 = 6^{29} \bmod 391$

   Untuk mempermudah perhitungan akan digunakan persamaan 2.8 yaitu *Fast Exponentiation*

   $29 = (11101)_2$

   $29 = 1.2^0+0.2^1+1.2^2+1.2^3+1.2^4$

   $x_1^{PK2} \bmod PK_1 = 11^{29} \bmod 391$

   $= (11^{2^0} \bmod 391 \,.\, 11^{2^2} \bmod 391 \,.\, 11^{2^3} \bmod 391 \,.\, 11^{2^4} \bmod 391) \bmod 391 = (11 \,.\, 174 \,.\, 169 \,.\, 18) \bmod 391 = 5822388 \bmod 391 = 7$

   $x_2^{PK2} \bmod PK_1 = 6^{29} \bmod 391$

   $= (6^{2^0} \bmod 391 \,.\, 6^{2^2} \bmod 391 \,.\, 6^{2^3} \bmod 391 \,.\, 6^{2^4} \bmod 391) \bmod 391 = (6 \,.\, 123 \,.\, 271 \,.\, 324) \bmod 391 = 64799352 \bmod 391 = 95$
10. Dihasilkan *Chipertext* "7 95".
11. Chipertext diubah ke bentuk biner menjadi 00110111 00100000 00111001 00110101.
12. Representasi biner *chipertext* akan disisipkan ke dalam representasi bit dari hasil pencuplikan berkas MP3 *carrier* berikut :

10001101	00001000	01000111	01100101	01010000
10010011	10011010	01111011	01110010	01110111
11101010	00001010	11111010	00111101	11100110
01011110	01011001	01100101	10000001	01010110
01101101	01100001	11101101	11000001	10101111
00101000	01101100	11110100	11000100	00100001
01111101	11010011	01011001	11000100	00100001

Setelah representasi biner chipertext disisipkan maka representasi biner carrier berubah menjadi :

1000110**0**	0000100**0**	0100011**1**	0110010**1**	0101000**0**
1001001**1**	1001101**1**	0111101**1**	0111001**0**	0111011**0**
1110101**1**	0000101**0**	1111101**0**	0011110**0**	1110011**0**
0101111**0**	0101100**0**	0110010**0**	1000000**1**	0101011**1**

0110110**1**	0110000**0**	1110110**0**	1100000**1**	1010111**0**
0010100**0**	0110110**1**	1111010**1**	1100010**0**	0010000**1**
0111110**0**	1101001**1**	01011001	11000100	00100001

3.4.2. Pengungkapan Pesan

Berikut adalah langkah-langkah dari pengungkapan pesan.

1. Mengambil bit paling kanan dari setiap byte pada representasi biner *carrier* yang telah disisipi pesan.

1000110**0**	0000100**0**	0100011**1**	0110010**1**	0101000**0**
1001001**1**	1001101**1**	0111101**1**	0111001**0**	0111011**0**
1110101**1**	0000101**0**	1111101**0**	0011110**0**	1110011**0**
0101111**0**	0101100**0**	0110010**0**	1000000**1**	0101011**1**
0110110**1**	0110000**0**	1110110**0**	1100000**1**	1010111**0**
0010100**0**	0110110**1**	1111010**1**	1100010**0**	0010000**1**
0111110**0**	1101001**1**	01011001	11000100	00100001

   Didapat hasil 00110111 00100000 00111001 00110101 101.
2. Setiap delapan bit, diubah ke representasi karakter sehingga didapat "7 95". Hal ini berarti didapat nilai *chipertext* $y_1 = 7$ dan $y_2 = 95$.
3. Dilakukan dekripsi dengan menggunakan persamaan 2.13

   $y_1^{SK} \bmod PK_1 = 7^{85} \bmod 391$

   $y_2^{SK} \bmod PK_1 = 95^{85} \bmod 391$

   Untuk mempermudah perhitungan akan digunakan persamaan 2.8 yaitu *Fast Exponentiation*

   $85 = (1010101)_2$

   $85 = 1.2^0+0.2^1+1.2^2+0.2^3+1.2^4+0.2^5+0.2^6$

   $y_1^{SK} \bmod PK_1 = 7^{85} \bmod 391$

   $= (7^{2^0} \bmod 391 \; . \; 7^{2^2} \bmod 391 \; . \; 7^{2^4} \bmod 391 \; . \; 7^{2^6} \bmod 391) \bmod 391 = (7 \; . \; 55 \; . \; 52 \; . \; 307) \bmod 391 = 6146140 \bmod 391 = 11$

   $y_2^{SK} \bmod PK_1 = 95^{85} \bmod 391$

   $= (95^{2^0} \bmod 391 \; . \; 95^{2^2} \bmod 391 \; . \; 95^{2^4} \bmod 391 \; . \; 95^{2^6} \bmod 391) \bmod 391 = (95 \; . \; 242 \; . \; 358 \; . \; 18) \bmod 391 = 148147560 \bmod 391 = 6$
4. Didapat nilai y = 116 yang dalam karakter ASCII adalah karakter "t".

3.4.3. Signal to Noise Ratio (SNR)

1. Mengumpulkan *Least Significant Bit* pada berkas *carrier*.

10001101	00001000	01000111	01100101	01010000
10010011	10011010	01111011	01110010	01110111
11101010	00001010	11111010	00111101	11100110
01011110	01011001	01100101	10000001	01010110
01101101	01100001	11101101	11000001	10101111
00101000	01101100	11110100	11000100	00100001
01111101	11010011	01011001	11000100	00100001

Jumlah total bit = 35 *8 = 280 bit

LSB awal = [10110 10101 00010 01110 11111 00001 11101].

2. Mengumpulkan *Least Significant Bit* pada berkas *carrier* yang telah mengalami proses penyisipan pesan.

1000110**0**	0000100**0**	0100011**1**	0110010**1**	0101000**0**
1001001**1**	1001101**1**	0111101**1**	0111001**0**	0111011**0**
1110101**1**	0000101**0**	1111101**0**	0011110**0**	1110011**0**
0101111**0**	0101100**0**	0110010**0**	1000000**1**	0101011**1**
0110110**1**	0110000**0**	1110110**0**	1100000**1**	1010111**0**
0010100**0**	0110110**1**	1111010**1**	1100010**0**	0010000**1**
0111110**0**	1101001**1**	01011001	11000100	00100001

LSB akhir = [00110 11100 10000 00011 10010 01101 01101].

3. Membandingkan Least Significant Bit (LSB) pada berkas Carrier sebelum dan sesudah disisipi pesan.

Tabel 3.1 Perbandingan LSB sebelum dan sesudah disisipi pesan

No	LSB Berkas Awal	LSB Berkas Akhir	Berbeda
1	1	0	Ya
2	0	0	Tidak
3	1	1	Tidak
4	1	1	Tidak
5	0	0	Tidak
6	1	1	Tidak

7	0	1	Ya
8	1	1	Tidak
9	0	0	Tidak
10	1	0	Ya
11	0	1	Ya
12	0	0	Tidak
13	0	0	Tidak
14	1	0	Ya
15	0	0	Tidak
16	0	0	Tidak
17	1	0	Ya
18	1	0	Ya
19	1	1	Tidak
20	0	1	Ya
21	1	1	Tidak
22	1	0	Ya
23	1	0	Ya
24	1	1	Tidak
25	1	0	Ya
26	0	0	Tidak
27	0	1	Ya
28	0	1	Ya
29	0	0	Tidak
30	1	1	Tidak
31	1	0	Ya
32	1	1	Tidak
33	1	1	Tidak
34	0	0	Tidak
35	1	1	Tidak

Jumlah bit yang berubah = 14 bit

4. Dihitung nilai *Signal to Noise Ratio (SNR)* dengan menggunakan persamaan 2.14.

$$SNR = 10.\log_{10}\frac{I}{N} = 10.\log_{10}\frac{280}{14} = 13,01\ dB$$

5. Dihitung prosentase *Signal to Noise Ratio (SNR)* dengan menggunakan persamaan 2.15.

$$SNR(\%) = \left(\frac{jumlahTotalBit - jumlahBitTerubah}{jumlahTotalBit}\right)100\%$$

$$SNR(\%) = \frac{280 - 14}{280} * 100\% = 95\%$$

Dari hasil perhitungan, berkas MP3 yang telah disisipi pesan bersesuaian atau memiliki kemiripan 95% dengan berkas MP3 awal sebelum disisipi pesan dan memiliki SNR sebesar 13,01dB

3.4.4. Brute Force Attack

Akan dicari banyak iterasi untuk mendapatkan kunci rahasia SK = 85 dengan diketahui PK_1 = 391 dan PK_2 = 29. Pencarian banyak iterasi dapat dilihat pada tabel 3.2

Tabel 3.2 Pencarian kunci rahasia dengan *Brute Force Attack*

p = i	q = PK_1/p	$\phi(n) = (p-1)(q-1)$	SK = PK_2^{-1} mod $\phi(n)$
1	391	0	∞
2	Bukan bil. bulat	-	-
3	Bukan bil. bulat	-	-
4	Bukan bil. bulat	-	-
5	Bukan bil. bulat	-	-
6	Bukan bil. bulat	-	-
7	Bukan bil. bulat	-	-
8	Bukan bil. bulat	-	-
9	Bukan bil. bulat	-	-
10	Bukan bil. bulat	-	-
11	Bukan bil. bulat	-	-
12	Bukan bil. bulat	-	-
13	Bukan bil. bulat	-	-
14	Bukan bil. bulat	-	-
15	Bukan bil. bulat	-	-
16	Bukan bil. bulat	-	-
17	23	352	85

Untuk dapat menemukan kunci rahasia (SK) = 85 dibutuhkan 17 kali perulangan.

3.5. Perancangan Uji Coba

Setelah sistem dibuat, langkah selanjutnya adalah melakukan pengujian terhadap sistem.Uji coba dilakukan untuk mengetahui tingkat perubahan kualitas berkas MP3 berdasarkan prosentase dari hasil perhitungan *Signal to Noise Ratio* (*SNR).* Selain itu uji coba juga akan dilakukan untuk mengetahui ketepatan sistem dalam mengungkapkan pesan yang telah disisipkan ke dalam berkas MP3. Tabel Uji untuk mengetahui tingkat perubahan kualitas dan ketepatan dalam pengungkapan pesan dapat dilihat pada tabel 3.3.

Tabel 3.3 Uji SNR dan ketepatan pengungkapan pesan.

No.	Nama MP3 Carrier	Nama dan Ukuran Pesan terenkripsi	Ratio ukuran pesan	Peng ung-kapan Pesan	SNR (dB)	SNR (%)
1						
.						
.						
.						
n						

Pengujian berikutnya adalah uji coba kualitas audio MP3 berdasarkan penilaian secara subjektif oleh sepuluh orang penguji atau responden. Pemilihan responden didasarkan pada kriteria memiliki pendengaran normal dan berusian antara 18 - 25 tahun. Pemilihan rentang usia responden dilakukan dengan mempertimbangkan faktor penurunan pendengaran manusia akibat pertambahan usia dan intensitas responden dalam menikmati audio MP3. Penilaian dilakukan dengan menggunkan metode *Mean Opinion Score (MOS)*. Tabel uji untuk mengetahui nilai MOS ditunjukkan pada tabel 3.4.

Tabel 3.4 Pengujian Nilai MOS

No.	Nama Berkas MP3	Mode sisip	Nilai Dari Tiap Penguji ke-										Nilai MOS
			1	2	3	4	5	6	7	8	9	10	
1.													
…													
n													

Untuk pengujian terhadap tingkat keamanan algoritma RSA akan dilakukan dengan menghitung banyaknya iterasi atau perulangan pada *Brute Force Attack* yang digunakan untuk mendapatkan kunci rahasia pada algoritma RSA dengan menggunakan persamaan 2.1, 2.16, dan 2.11. Tabel uji untuk mengetahui banyaknya iterasi dapat dilihat pada tabel 3.4

Tabel 3.5 Uji banyaknya iterasi *Brute Force Attack*

No.	Kunci Publik Pertama	Kunci Publik Kedua	Kunci Rahasia	Faktor Prima	Banyaknya Iterasi
1.				a.	
				b.	
2.				a.	
				b.	
…					
n.				a.	
				b.	

Halaman ini sengaja dikosongkan

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1. Lingkungan Implementasi

Lingkungan implementasi dari rancangan aplikasi penyisipan pesan terenkripsi pada *file* audio MP3 menggunakan metode *Least Significant Bit (LSB)* 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(R) *Core*(TM) 2 Duo CPU E7500 @ 2.93GHz
2. *Memory* 2024 MB RAM
3. *Harddisk* 500 Gb
4. *System type* 32-bit OS
5. *Graphics* NVIDIA GeForce 9500 GT@ 1024MB

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan aplikasi penyisipan pesan terenkripsi pada *file* audio MP3 dan uji cobanya adalah:

1. Sistem operasi Windows 7 Ultimate 32-bit sebagai lingkungan aplikasi dijalankan.
2. NetBeans IDE 6.9.1 sebagai salah satu *software development* untuk pemrograman.

4.2 Implementasi Program

Pada subbab ini akan dijelaskan implementasi program dalam pembuatan aplikasi penyisipan pesan terenkripsi pada *file* audio MP3 berdasarkan perancangan sistem yang telah dijelaskan pada Bab III. Implementasi pembuatan aplikasi dibuat dengan menggunakan bahasa pemrogaman JAVA.

4.2.1. Implementasi Penyisipan Pesan Ke Dalam MP3

Proses penyisipan pesan ke dalam MP3 terdiri dari dua tahapan utama yaitu proses enkripsi yang bertujuan untuk merubah

struktur dari pesan, dan yang terakhir adalah proses *embedding* yang berfungsi menyisipkan pesan hasil enkripsi kedalam MP3.

4.2.1.1 Proses Enkripsi

Implementasi program proses enkripsi diawali dengan pembacaan pesan berformat .txt sebagai *plaintext*. Proses pembacaan dilakukan saat ditekanya tombol "*Load* Pesan" pada antar muka aplikasi. Pembacaan pesan dilakukan perbaris dan sebagai penanda diakhir baris akan ditambahkan karakter "à". Implementasi pembacaan pesan ditunjukkan pada *source code* 4.1

```
String dir = fc.getSelectedFile().getPath().toString();
File text = new File(dir);
sbPesan = new StringBuilder();
try{
   BufferedReader fIn = new BufferedReader(new FileReader
   (text));
   String line;
   while((line=fIn.readLine())!=null){
         sbPesan.append(line).append("à");
   }
}catch(Exception e){}

sbPesan.replace(sbPesan.length()-1, sbPesan.length(), "");
dirPesan.setText(dir);
```

***Source code* 4.1:** Pembacaan Pesan

Setelah proses pembacaan pesan, pengguna diharuskan memasukan dua buah bilangan prima p dan q. Kedua buah bilangan tersebut akan digunakan untuk membentuk tiga buah kunci yang terdiri dari dua kunci publik dan satu kunci rahasia. Untuk melakukan proses enkripsi terhadap pesan maka akan ditekan tombol "Enkripsi . Antar muka proses enkripsi ditunjukkan pada gambar 4.1.

Gambar 4.1 : Antar muka proses enkripsi

Saat tombol "Enkripsi" ditekan maka proses pertama yang akan dilakukan adalah pengecekan kedua bilangan masukan merupakan bilangan prima atau tidak. Implementasi pengecekan bilangan prima ditunjukkan pada *source code* 4.2.

```
boolean cekPrima(BigInteger bil){
        boolean prime = false;
        BigInteger a=BigInteger.valueOf(2);
        BigInteger zero=BigInteger.valueOf(0);
        BigInteger j;
        if (bil.compareTo(a)>=0){
            prime = true;
            for (int i=2; i<bil.intValue(); i++){
               j=BigInteger.valueOf((long)i);
               if(bil.remainder(j).compareTo(zero)==0)
                {
                    prime = false;

                    break;
                }
            }
        }return prime;
    }
```

***Source code* 4.2** : Pengecekan bilangan prima

Apabila masukan bukan bilangan prima maka akan muncul peringatan, tetapi apabila kedua masukan merupakan bilangan prima, proses akan dilanjutkan dengan pembentukan kunci. Implementasi proses pembentukan kunci ditunjukkan pada *source code* 4.3.

```
public void GenerateKey (BigInteger p, BigInteger q){
   n=p.multiply(q);

   phi_euler=(p.subtract(BigInteger.ONE)).multiply(q.subtract(B
   igInteger.ONE));

   pk1=n;
   BigInteger temp = BigInteger.ZERO;
   while (phi_euler.gcd(temp).compareTo(BigInteger.ONE)==1)
   {
         acak=rand.nextInt(phi_euler.divide(BigInteger.valueOf(
         2)).intValue());
        temp=BigInteger.valueOf(acak);
   }
    pk2=temp;
    BigInteger expo = BigInteger.valueOf(-1);
    sk=pk2.modPow(expo, phi_euler);
}
```

***Source code* 4.3** : Pembentukan Kunci

Kemudian akan dilanjutkan dengan proses perubahan tiap karakter dalam pesan yang akan disisipkan menjadi bentuk desimal ASCII-nya. Proses ini ditunjukkan pada *source code* 4.4.

```
for(int i=0; i<pesan.length(); i++){
    int des = (int) pesan.charAt(i);
    plaintext = plaintext.append(Integer.toString(des));
}
```

***Source code* 4.4:** Perubahan ke bentuk ASCII

Nilai ASCII yang telah diperoleh dari setiap karakter akan disatukan dan kemudian akan dipecah sesuai panjang kunci publik pertama dikurangi satu. Hasil dari tiap pemecahan tersebut akan dilakukan proses enkripsi sesuai dengan persamaan 2.12. Proses ini diimplementasikan pada *source code* 4.5

```
panjang_plaintext = plaintext.length();
panjang_publik_key1 = pk1.toString().length();
int sisa = panjang_plaintext%(panjang_publik_key1-1);
if (panjang_plaintext%(panjang_publik_key1 - 1)== 0)
    jumlah =panjang_plaintext/(panjang_publik_key1-1);
```

```
else jumlah =panjang_plaintext/(panjang_publik_key1 -1) +1;

String []C = new String[jumlah+1];
BigInteger [] Chipertext = new BigInteger [jumlah+1];
BigInteger [] Ch = new BigInteger [jumlah+1];
for (int i=0; i<jumlah ; i++){
     if (k==panjang_plaintext-sisa)
       C[i]=plaintext.substring(k, k+sisa);
     else
        C[i]=plaintext.substring(k, k+(panjang_publik_key1-1));
     k=k+(panjang_publik_key1-1);
     Ch[i]= new BigInteger (C[i]);
     Chipertext[i] = Ch[i].modPow(pk2, pk1);
     chipertext = chipertext.append
                  (Chipertext[i].toString()).append(" ");
}return chipertext;
```

***Source code* 4.5:** Proses Enkrisi

Untuk dapat menyisipkan pesan yang telah terenkripsi (*chipertext*) kedalam MP3, maka *chipertext* akan diubah kebentuk biner dan kemudian akan ditambahkan bentuk biner dari karakter penanda berupa biner dari karakter "!" diawal dan biner dari karakter "#" diakhir dari biner *chipertext*. Agar mendapatkan nilai biner sebanyak 8 bit maka selama proses perubahan karakter *chipertext* ke bentuk biner perlu ditambahkan padding. *Source code* untuk proses *padding* ditunjukkan pada *source code* 4.6, sedangkan untuk proses perubahan *chipertext* ke bentuk biner ditunjukkan pada *source code* 4.7

```
private String binary(String x){
        switch(x.length()){
            case 1: return "0000000"+x;
            case 2: return "000000"+x;
            case 3: return "00000"+x;
            case 4: return "0000"+x;
            case 5: return "000"+x;
            case 6: return "00"+x;
            case 7: return "0"+x;
            default: return x;
         }
}
```

***Source code* 4.6:** *Padding*

```
bin.append("00100001");
for (int i=0; i<chiper.length(); i++){
     code = (int) chiper.charAt(i);
      bin = bin.append(binary(Integer.toBinaryString(code)));
```

```
}
bin=bin.append("00100011");
```

***Source code* 4.7:** Perubahan *Chipertext* ke biner

Hasil dari enkripsi pesan akan ditampilkan pada *text area* hasil enkripsi. Selain itu informasi tentang kunci publik, kunci rahasia dan ukuran *file* terenkripsi akan ditampilkan pada *text area* info

4.2.1.2 Proses *Embedding*

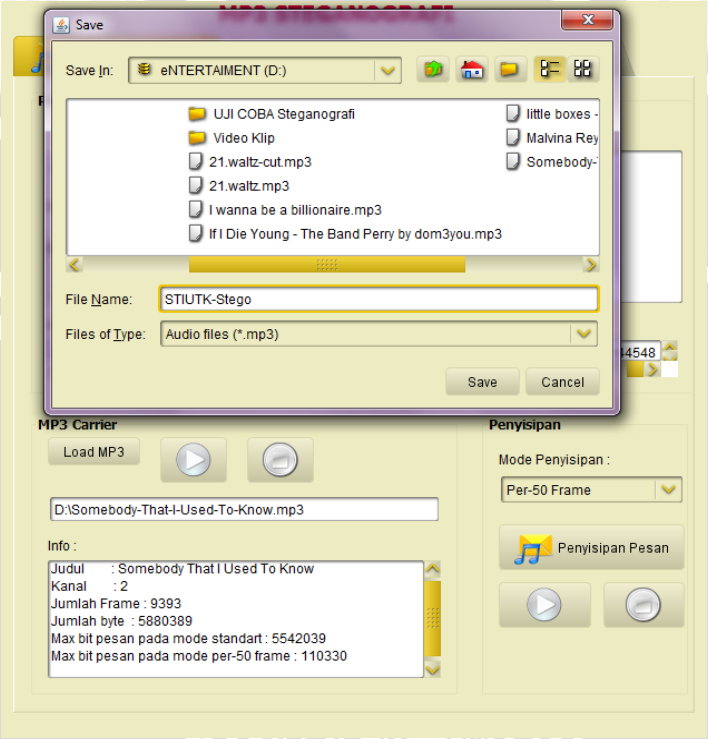
Proses ini merupakan proses penyisipan biner *chipertext* ke dalam *Least Significant Bit* MP3. Untuk memilih MP3 *carrier* pengguna dapat menekan tombol "*Load* MP3" pada antar muka penyisipan pesan dan dilanjutkan dengan memilih *file* MP3 pada *file chooser* kemudian akan terjadi pembacaan byte MP3 yang kemudian setiap byte akan diubah ke bentuk 8 bit. Implementasi pembacaan bit MP3 ditunjukkan pada *source code* 4.8

```
public void bacaMp3(String dir){
 try {
     File song = new File(dir);
     FileInputStream file = new FileInputStream(song);
     int count = file.available();
    sbMp3 = new StringBuilder();
    for (int i = 0; i < count; i++)   {
         sbMp3.append(binary(Integer.toBinaryString(file.read()
         ))).append("/");
    }
    file.close();
  } catch (Exception e) {
    System.out.println("Error â€" " + e.toString());
    }
}
```

***Source code* 4.8:** Pembacaan bit MP3

Sesudah MP3 dipilih, akan ditampilkan mengenai informasi identitas MP3 dan juga ukuran kapasitas maksimum penyimpanan pesan pada MP3 tersebut. Pengguna juga dapat memainkan MP3 yang dipilih dengan menekan tombol "Play", untuk menghentikan permainan MP3 pengguna dapat menekan tombol "Stop". Kemudian akan dipilih mode penyisipan pesan pada combo box. Terdapat dua mode penyisipan yaitu mode *standard* dan mode per-50 *frame*. Ketika mode telah dipilih maka akan ditekan tombol "Penyisipan

Pesan". Pada saat tombol ditekan maka maka akan muncul *file chooser* untuk memberi nama hasil MP3 yang telah disisipi pesan dan memilih lokasi penyimpanan. Proses ini ditunjukkan pada gambar 4.2



Gambar 4.2 : Antar muka proses penyisipan pesan

Proses penyisipan diawali dengan mencari *header frame* pertama pada MP3 kemudian akan dicari *main data* yaitu byte ke-37 pada *frame* hingga *header frame* berikutnya, Untuk menghindari hasil subtitusi LSB dengan bit pesan yang mungkin akan menghasilkan kombinasi dengan bit-bit yang sama dengan bit-bit dari penanda *frame header* maka akan dilakukan pencegahan dengan mengubah LSB dari byte yang beresiko menjadi penanda *header* saat proses subtitusi. Implementasi dari proses ini ditunjukkan pada *source code* 4.9

```
String normal = sb.toString().replace("11111110/11111010/",
```

```
                "1111111x/1111101x/")
                .replace("11111110/11111011/",
                "1111111x/1111101y/")
                .replace("11111111/11111010/",
                "1111111y/1111101x/");
sb.delete(0, sb.length());
sb = new StringBuilder(normal);
```

***Source code* 4.9:** Pengamanan bit yang beresiko menjadi penanda *header*

Setelah proses pengamanan bit yang beresiko untuk menjadi penanda *header* maka proses *embedding* dapat dilakukan. Untuk implementasi dari proses *embedding* dengan mode *standard* ditunjukkan pada *source code* 4.10 sedangkan untuk mode per-50 *frame* ditunjukkan pada source code 4.11

```
index2=sb.indexOf("11111111/11111011/");
 sisaBit= bin.length();
 while(sisaBit > 0)
 {
    index1=index2 + 323;
    index2 = sb.indexOf("11111111/11111011/",index1);
    if(index2<0)index2=sb.length()-1;
    kapasitas=((index2-index1)/9);

    for(int i=0; i<kapasitas; i++){
        index1=index1+8;
         if(sb.charAt(index1)!='x' && sb.charAt(index1)!='y'){
           sb.setCharAt(index1, bin.charAt(index3));
           index3++;
           sisaBit--;
           if(sisaBit==0) break;
         }
         index1++;
    }

 }
 bitBaru=sb.toString().replace("x", "0").replace("y", "1");
 sb.delete(0, sb.length());
 return bitBaru;
```

***Source code* 4.10:** Proses *embedding* mode *standard*

```
c=0;
index2=sb.indexOf("11111111/11111011/");
sisaBit= bin.length();
while(sisaBit > 0)
{
   index1=index2 + 323;
   index2 = sb.indexOf("11111111/11111011/",index1);
```

```
    if(index2<0)index2=sb.length()-1;

    if(c%50==0){
      kapasitas=((index2-index1)/9);
      for(int i=0; i<kapasitas; i++){
          index1=index1+8;
           if(sb.charAt(index1)!='x' && sb.charAt(index1)!='y'){
             sb.setCharAt(index1, bin.charAt(index3));
             index3++;
             sisaBit--;
             if(sisaBit==0) break;
          }
          index1++;
      }
   } c++;
}
bitBaru=sb.toString().replace("x", "0").replace("y", "1");
sb.delete(0, sb.length());
return bitBaru;
```

***Source code* 4.11:** Proses *embedding* mode per-50 *frame*

Setelah proses *embedding*, maka proses yang terakhir dari implementasi penyisipan pesan adalah pembentukan bit-bit yang telah disisipi pesan kebentuk MP3 kembali. Proses ini ditunjukkan pada *source code* 4.12

```
String save = fc.getSelectedFile().getPath().toString();
Mp3name= save+".mp3";
String [] byteBaru = bitBaru.split("/");

try{
    File trashedSong = new File(save+".mp3");
    FileOutputStream trash = new FileOutputStream(trashedSong);

   for (int i=0;i<byteBaru.length;i++)
       trash.write(Integer.parseInt(byteBaru[i],2));
   trash.close();
   byteBaru=null;
    }catch (Exception e) {
System.out.println("Error â€" " + e.toString());
```

***Source code* 4.12:** Pembentukan MP3

4.2.2. Implementasi Penngungkapan Pesan Dalam MP3

Proses pengungkapan pesan dalam MP3 terdiri dari penggabungan dua tahapan yaitu proses *retrieving* yang berfungsi untuk mendapatkan bit *chipertext* dari dalam MP3 dan proses yang kedua adalah proses dekripsi yaitu proses untuk mengubah *chipertext* menjadi *plaintext* kembali. Proses ini diawali dengan memasukan

kunci publik pertama dan kunci rahasia yang kemudian dilanjutkan dengan memasukan *file* MP3 yang mengandung pesan. Untuk melakukan pengungkapan pesan maka dipilih mode pengungkapan dan akan ditekan tombol "Pengungkapan Pesan". Antar muka proses pengungkapan pesan ditunjukkan pada gambar 4.3.

Gambar 4.3 : Antar muka proses pengungkapan pesan

Pada saat tombol ini ditekan maka dilakukan pengecekan apakah MP3 memiliki pesan atau tidak. Implementasi pengecekan adanya pesan dalam MP3 ditunjukkan pada *source code* 4.13.

```
public boolean adaPesan(StringBuilder sb){
    int pointer1 = 0;
    int pointer2 = sb.indexOf("11111111/11111011/");
    boolean ada = false;
    String temp="", pesan="";
```

```
    pointer1=pointer2+323;
    int jum =0, desimal;
    char ascii;
    while (jum<8){
         pointer1=pointer1+8;
           temp=temp+String.valueOf(sb.charAt(pointer1));
         jum++;
          pointer1++;
    }
    desimal=Integer.parseInt(temp,2);
    ascii=(char)desimal;
    pesan=String.valueOf(ascii);

    if(pesan.equals("!"))ada = true;
    else ada=false;

    return ada;
}
```

***Source code* 4.13:** Pengecekan pesan dalam MP3

Karena pada proses *embedding* dilakukan pengamanan untuk menghindari hasil subtitusi LSB dengan bit pesan yang mungkin akan menghasilkan kombinasi dengan bit-bit yang sama dengan bit-bit dari penanda *frame header*, maka dalam proses *retrieving* juga akan dilakukan pengamanan yang telah diimplementasikan pada *source code* 4.8. Setelah proses pengamanan barulah implementasi *retrieving* dilakukan. Untuk implementasi *retrieving* mode *standard* ditunjukkan pada *source code* 4.14, sedangkan untuk implementasi *retrieving* mode per-50 *frame* ditunjukkan pada *source code* 4.15

```
index4=sb.indexOf("11111111/11111011/");
while(akhir){
     index3=index4+323;
     index4=sb.indexOf("11111111/11111011/",index3);
     if(index4<0)index4=sb.length()-1;
    kapasitas=((index4-index3)/9);
    for(int i=0; i<kapasitas; i++){
        index3=index3+8;
         if(sb.charAt(index3)!='x' && sb.charAt(index3)!='y'){
           temp=temp+String.valueOf(sb.charAt(index3));
           jmlData++;
           if(jmlData==8){
              desimal=Integer.parseInt(temp,2);
              ascii=(char)desimal;
              pesan=String.valueOf(ascii);
              jmlData=0;
              msg=msg.append(pesan);
```

```
                temp="";
                if(pesan.equals("#")){
                   akhir= false;
                   break;
                }
             }
          }index3++;
    }
}
sb.delete(0, sb.length());
message=msg.toString().replaceAll("[#!]", "");
return message;
```

***Source code* 4.14:** *Retrieving* mode *standard*

```
index4=sb.indexOf("11111111/11111011/");
while(akhir){
     index3=index4+323;
     index4=sb.indexOf("11111111/11111011/",index3);
     if(index4<0)index4=sb.length()-1;

  if(c%50==0){
    kapasitas=((index4-index3)/9);
    for(int i=0; i<kapasitas; i++){
        index3=index3+8;
         if(sb.charAt(index3)!='x' && sb.charAt(index3)!='y'){
           temp=temp+String.valueOf(sb.charAt(index3));
           jmlData++;
           if(jmlData==8){
              desimal=Integer.parseInt(temp,2);
              ascii=(char)desimal;
              pesan=String.valueOf(ascii);
              jmlData=0;
              msg=msg.append(pesan);
              temp="";
              if(pesan.equals("#")){
                 akhir= false;
                 break;
              }
           }
       }index3++;
   }
  }c++;
}
sb.delete(0, sb.length());
message=msg.toString().replaceAll("[#!]", "");
return message;
```

***Source code* 4.15:** *Retrieving* mode per-50 *frame*

Setelah mendapatkan *chipertext* hasil dari proses *retrieving* maka akan dilakukan proses dekripsi. Pada proses ini akan digunakan kunci publik pertama dan kunci rahasia yang merupakan

masukan dari pengguna. kedua kunci tersebut akan digunakan untuk proses dekripsi sesuai dengan persamaan 2.13. Nilai *chipertext* dari hasil proses *retrieving* akan dipisah berdasarkan karakter spasi, tiap nilai *chipertext* tersebut kemudian akan didekripsi. Hasil dari proses dekripsi akan disatukan yang kemudian akan dipisah untuk mendapatkan nilai decimal ASCII tiap karakter dari pesan yang sebenarnya. Implementasi dari proses tersebut ditunjukkan pada *source code* 4.16

```
public String dekripsi(String message, BigInteger pk1,
BigInteger sk){
  int jumlahY;
  String digit=Integer.toString(pk1.toString().length()-1);
  String m = "", pot="";
  StringBuilder rahasia = new StringBuilder();
  int awal=0, akhir=0, des;
  jumlahY = (message.length()-message.replaceAll(" ",
   "").length());
  String [] temp = new String [jumlahY +1];
        temp = message.split(" ");
        BigInteger []Y = new BigInteger[jumlahY+1];
        BigInteger []X = new BigInteger[jumlahY+1];
        for(int i=0; i<jumlahY; i++){
            Y[i]= new BigInteger (temp[i]);
            X[i]= Y[i].modPow(sk, pk1);
            if (i<jumlahY-1) m=m+ String.format
            ("%1$0"+digit+"d", X[i]);// 3 diubah seusi panjang
           publik key1 -1
            else m = m + X[i].toString();
        }
        while (awal != m.length()){
            char x = m.charAt(akhir);
            if(x>='3'){
                 akhir = awal + 2;
                 if (akhir > m.length()){
                     akhir=awal+1;
                     awal = awal-1;
                 }
            }

            else{
                akhir = awal + 3;
                if (akhir > m.length()){
                    akhir=awal+2;
                    if (akhir > m.length()){
                        akhir=awal +1;
                        awal = awal-1;
                    }
                }
            }
```

```
            String y = m.substring(awal, akhir);

            if (y.startsWith("1")&& y.length()<3){
                y=y.charAt(0)+"0"+y.charAt(1);
            }

            awal = akhir;
            des = Integer.parseInt(y);
            rahasia=rahasia.append(String.valueOf((char)des));

        }
        return rahasia.toString().replaceAll("à", "\n");
    }
```

***Source code* 4.16:** Proses dekripsi

Hasil dari dekripsi akan ditampilkan pada text area pesan, yang juga akan disimpan dalam *file* txt yang telah dipilih pada saat awal ditekanya tombol "Pengungkapan Pesan".

4.2.3. Implementasi Pengujian

Pada implementasi pengujian akan dijelaskan mengenai pengujian SNR pada dua berkas MP3 dan juga pengujian banyaknya iterasi untuk mendapatkan kunci rahasia dengan menggunakan *Brute Force Attack*.

4.2.3.1 Pengujian SNR

Pada implementasi pengujian SNR ini akan dibandingkan dua *file* MP3 yaitu *file* MP3 asli dan *file* MP3 yang telah disisipi pesan. Pengujian akan dilakukan dengan menghitung prosentase SNR dalam satuan persen dan juga Nilai SNR dalam satuan dB.

Proses ini diawali dengan memasukan dua *file* MP3 yang kemudian akan ditekan tombol "SNR" untuk menghitung SNR. Antar muka pengujian SNR ditunjukan pada gambar 4.4.

Gambar 4.4: Antar muka proses pengujian SNR

Perhitungan prosentase SNR didasarkan pada persamaan 2.15, sedangkan perhitungan nilai SNR didasarkan pada persamaan 2.14. Implementasi pengujian SNR ditunjukkan pada *source code* 4.17.

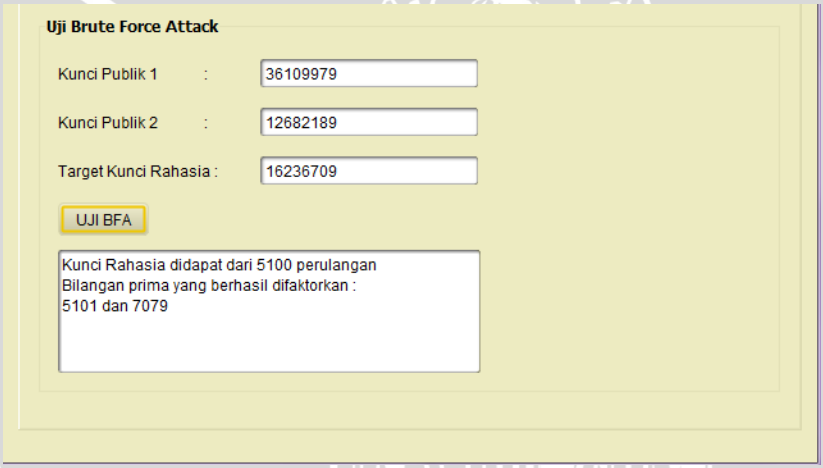
```
public void ujiSNR(String a1, String a2){
   prosenSNR=0;
   SNR=0;
   int ubah =0;
   try {
        File song1 = new File(a1);
        File song2 = new File(a2);
        FileInputStream file1 = new FileInputStream(song1);
        FileInputStream file2 = new FileInputStream(song2);
        jum1=file1.available();
        jum2=file2.available();

         if(jum1==jum2){
            for (int i = 0; i < jum1; i++) {
                 if (file1.read()!= file2.read()) ubah++;
            }
          prosenSNR = (((double)jum1*8)-(double)ubah)/
                      ((double)jum1*8)*100;
          SNR = 10*Math.log10((jum1*8)/ubah);
         }
         else{
              JOptionPane.showMessageDialog(null, "Jumlah byte
              dari kedua MP3 berbeda", "ERROR", 0);
             }

          file1.close();
          file2.close();
```

```
    } catch (Exception e) {
    System.out.println("Error â€" " + e.toString());}
}
```

***Source code* 4.17:** Uji SNR

4.2.3.2 Pengujian Iterasi secara BFA

Implementasi ini bertujuan untuk mendapatkan banyaknya iterasi yang dilakukan untuk memperoleh nilai dari kunci rahasia dengan cara *Brute Force Attack*. Pengguna diharuskan untuk memasukkan kunci publik pertama, kunci publik kedua, dan juga kunci rahasia sebagai targetnya. Setelah itu akan ditekan tombol "Uji BFA" untuk mendapatkan hasil pengujian yang akan ditampilkan pada *text area*. Gambar antar muka pengujian BFA ditunjukan pada gambar 4.5



Gambar 4.5: Antar muka pengujian BFA

Ketika tombol "Uji BFA" ditekan maka kunci publik pertama akan difaktorkan kemudian hasil pemfaktoran tersebut akan digunakan untuk membangkitkan kunci rahasia, kemudian akan dihitung banyaknya iterasi yang dibutuhkan agar kunci rahasia yang dibangkitkan sama dengan target. Implementasi dari proses pengujian iterasi dengan BFA ditunjukkan pada *source code* 4.18

```
public void BFA(BigInteger pk1, BigInteger pk2, BigInteger sk){
    int i = 2;
```

```
BigInteger nol = BigInteger.ZERO;
BigInteger rahasia= BigInteger.valueOf(0);
BigInteger expo = BigInteger.valueOf(-1);
BigInteger phi_euler=BigInteger.valueOf(0);
while(rahasia.compareTo(sk)!=0){
      p= BigInteger.valueOf(i);
      if(pk1.remainder(p)==nol){
         q=pk1.divide(p);
          phi_euler=(p.subtract(BigInteger.ONE)
                    .multiply(q.subtract(BigInteger.ONE)));
         rahasia=pk2.modPow(expo, phi_euler);
         }
         i++;
         iterasi++;
     }
 }
```

***Source code* 4.18:** Uji Banyaknya iterasi menggunakan BFA

4.3. Implementasi Uji Coba

Implementasi uji coba terhadap aplikasi penyisipan pesan terenkripsi menggunakan metode *Least Significant Bit* yang dilakukan, mengacu pada perancangan uji coba subbab 3.5. Uji coba yang akan dilakukan adalah untuk mengetahui tingkat kualitas Stego-MP3 atau MP3 yang telah disisipi pesan dan juga keberhasilan pengungkapan pesan yang berada dalam stego-MP3. Pengujian juga akan dilakukan untuk mengetahui tingkat keamanan enkripsi RSA. MP3 yang akan digunakan sebagai bahan uji atau *carrier* terdiri dari enam buah *file* MP3 yang memiliki ukuran berbeda. Daftar MP3 yang digunakan dapat dilihat pada tabel 4.1.

Tabel 4.1 Daftar *File* MP3

No.	Nama *File* MP3	Ukuran (byte)	Rasio Kapasitas Peyisipan	
			Standard	Per-50 *Frame*
1.	*result.MP3*	897530	11.6757 %	0.2298 %
2.	*Statan Zunea-cleopatra.mp3*	2885258	11.4120 %	0.2272 %
3.	*I Wanna Be Billionaire.mp3*	3359604	11.4130 %	0.2276 %
4.	*akb48 - heavy rotation. mp3*	4567160	11.4097 %	0.2278 %
5.	*Somebody-That-I-Used-To-Know.mp3*	5880839	11.7808 %	0.2345 %

6.	*J-Rocks Kau Curi Lagi.mp3*	7016448	11.7176 %	0.2333 %

Dari tabel 4.1 terlihat bahwa rasio kapasitas penyimpanan pesan pada MP3 dengan menggunakan mode *standard* berkisar 11%, sedangkan untuk penyisipan dengan menggunakan mode Per-50 *Frame* berkisar antara 0.22%. Rasio kapasitas penyisipan dengan menggunakan mode per 50 *frame* jauh lebih kecil dibandingkan dengan menggunakan mode standard. Hal ini dikarenakan pada mode per-50 *frame* penyisipan pesan tidak dilakukan secara berurutan untuk setiap *frame* melainkan dilakukan pada main *frame* di setiap 50 *frame* dan diawali dengan frame pertama sehingga kapasiatas penyisipan pesan berukuran sekitar 1/50 dari kapasitas penyisipan dengan mode standard.

Sedangkan untuk pesan yang dicoba untuk disisipkan ke dalam *carrier* MP3 terdiri dari sepuluh *file* bertipe teks (.txt) yang memiliki ukuran yang berbeda-beda. Daftar *file* teks yang digunakan diperlihatkan pada tabel 4.2

Tabel 4.2 Daftar *File* Teks

No.	Nama *File* Pesan	Ukuran Pesan Asli (byte)
1.	Uji Steganografi.txt	63
1.	JFC.txt	495
2.	Biodata.txt	277
3.	Code Cek Prima.txt	779
4.	Steganografi.txt	1408
5.	Sejarah MP3.txt	2672
6.	Cv.txt	1098
7.	Daftar Pustaka.txt	3078
8.	Latar Belakang.txt	3795
9.	Cerita.txt	3524

Sebelum disispkan kedalam MP3, pesan teks akan dienkripsi menggunakan algoritma RSA. Ukuran pesan hasil dari proses enkripsi atau *chipertext* akan berbeda dan bertambah besar dibandingkan dengan ukuran *file* aslinya. Ukuran dari hasil enkripsi

pada *file* teks yang sama juga bervariasi tergantung dari kunci yang digunakan pada saat proses enkripsi.

Prosedur uji coba yang dilakukan adalah dengan menyisipkan tiga buah *chipertext* hasil enkripsi dari tiga *file* teks yang berbeda kedalam sebuah MP3 *carrier*. Penyisipan dilakukan dengan dua cara yaitu mode *standard* dan mode per-50 *frame.* Hasil MP3 dari penyisipan akan dinamai sama dengan nama *file* MP3 asli. Bila menggunakan mode *standard* maka ditambahkan kata "std" sedangkan hasil penyisipan dengan mode per-50 *frame* akan ditambahkan kata "50F".

4.4. Hasil Pengujian dan Pembahasan

Berdasarkan rancangan pengujian yang telah dijelaskan dalam bab 3 maka hasil pengujian akan dibedakan menjadi hasil pengujian perubahan kualitas MP3 dan ketepatan pengungkapan pesan, serta pengujian tingkat keamanan algoritma RSA berdasarkan penyerangan kunci rahasia menggunakan *Brute Force Attack*.

4.4.1. Hasil Pengujian dan Pembahasan Perubahan Kualitas MP3 dan Ketepatan Pengungkapan Pesan

Pengujian perubahan kualitas MP3 dilakukan secara matematis dan secara subjektif. Pengujian secara matematis dilakukan dengan menghitung nilai dan prosentase *Signal to Noise Ratio (SNR),* sedangkan secara subjektif dilakukan dengan menghitung nilai *Mean Opinion Score (MOS)*. Untuk menguji ketepatan pengungkapan pesan dilakukan dengan membandingkan apakah hasil pengungkapan sama dengan pesan asli atau tidak. Hasil pengujian SNR dilakukan berdasarkan mode penyisipan yang dilakukan. Ada dua mode yang digunakan yaitu mode *standard* dan mode per-50 *frame*.

Nilai SNR digunakan untu mengetahui kualitas dari audio MP3, semakin besar nilai SNR yang diukur dalam satuan dB maka semakin bagus kualitas dari MP3 tersebut. Sedangkan prosentase SNR digunakan untuk mengetahui seberapa samakah karakteristik bit pada MP3 hasil penyisipan pesan dengan MP3 asli (*carrier*). Hasil pengujian SNR dan ketepatan pengungkapan pesan untuk mode *standard* dan mode per-50 *frame* ditunjukkan pada tabel 4.3 dan 4.4

Tabel 4.3 Hasil Pengujian SNR dan ketepatan pengungkapan pesan dengan mode *standard*

No.	Nama MP3 Carrier	Nama dan Ukuran Pesan terenkripsi	Ratio ukuran pesan (%)	Peng-ung-kapan Pesan	SNR (dB)	SNR (%)
1.	*result.MP3* (897530 byte)	Uji Steganografi .txt (193 byte)	0.0215	Tepat	39.759	99.989
		JFC.txt (1592 byte)	0.1773	Tepat	30.500	99.911
		Biodata.txt (747 byte)	0.0832	Tepat	33.694	99.957
2.	*Statan Zunea-cleopatra. mp3* (2885258 byte)	code Cek Prima.txt (2068 byte)	0.0716	Tepat	35.293	99.970
		Steganografi .txt (4174 byte)	0.1446	Tepat	31.821	99.934
		Sejarah MP3.txt (8631 byte)	0.2991	Tepat	28.500	99.859
3.	*I Wanna Be Billionaire. mp3* (3359604 byte)	JFC.txt (1570 byte)	0.0467	Tepat	36.395	99.977
		Cv.txt (3130 byte)	0.0932	Tepat	33.363	99.954
		Steganografi .txt (4292 byte)	0.1277	Tepat	31.970	99.936
4.	*akb48 - heavy rotation. mp3* (4567160 byte)	Code cek prima.txt (2381 byte)	0.0521	Tepat	35.714	99.973
		Sejarah MP3.txt (8373 byte)	0.1833	Tepat	30.077	99.902
		Daftar pustaka.txt (9198 byte)	0.2014	Tepat	29.685	99.893
5.	*Somebody-That-I-*	Steganografi .txt	0.0753	Tepat	34.280	99.963

	Used-To-Know.mp3 (5880389)	(4429 byte)				
		Daftar pustaka.txt (8593 byte)	0.1461	Tepat	31.414	99.928
		Latar Belakang.txt (13030 byte)	0.2216	Tepat	29.590	99.890
6.	*J-Rock Kau Curi Lagi.mp3* (7016448 byte)	belek.txt (15149 byte)	0.2159	Tepat	29.666	99.892
		Sejarah Mp3.txt (8444 byte)	0.1203	Tepat	32.245	99.940
		Latar belakang.txt (12468 byte)	0.1777	Tepat	30.538	99.912

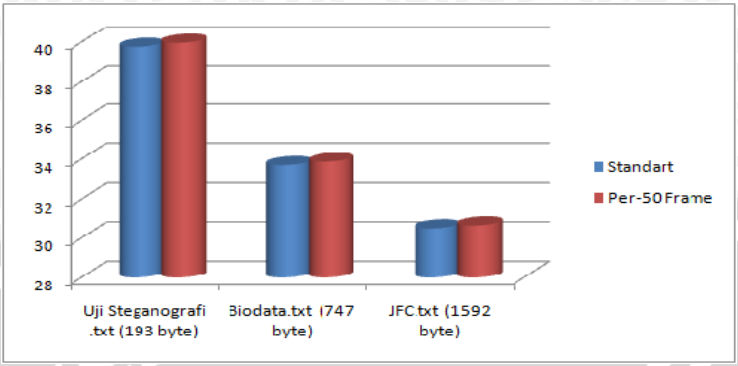
Tabel 4.4 Hasil Pengujian SNR dan ketepatan pengungkapan pesan dengan mode per-50 *frame*

No.	Nama MP3 Carrier	Nama dan Ukuran Pesan terenkripsi	Ratio ukuran pesan (%)	Pengungkapan Pesan	SNR (dB)	SNR (%)
1.	*result.MP3* (897530 byte)	Uji Steganografi .txt (193 byte)	0.0215	Tepat	39.964	99.990
		JFC.txt (1592 byte)	0.1773	Tepat	30.652	99.914
		Biodata.txt (747 byte)	0.0832	Tepat	33.872	99.959
2.	*Statan Zunea-cleopatra. mp3* (2885258 byte)	code Cek Prima.txt (2068 byte)	0.0716	Tepat	34.506	99.965
		Steganografi .txt (4174 byte)	0.1446	Tepat	31.480	99.929
		Sejarah MP3.txt (8631 byte)	0.2991	Pesan tidak disispkan	-	-
3.	*I Wanna*	JFC.txt	0.0467	Tepat	36.365	99.977

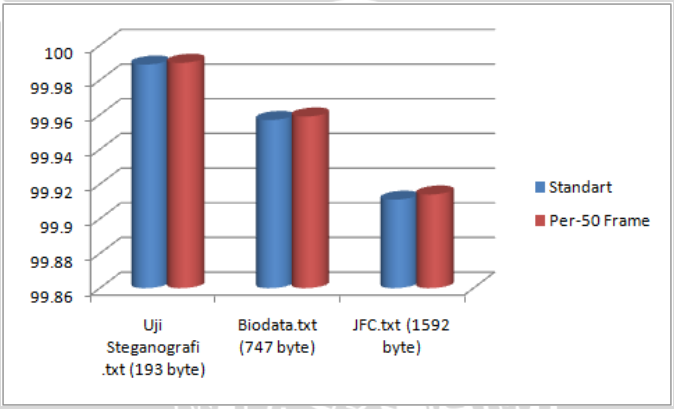
	Be Billionaire. mp3 (3359604 byte)	(1570 byte)				
		Cv.txt (3130 byte)	0.0932	Tepat	33.398	99.954
		Steganografi .txt (4292 byte)	0.1277	Tepat	31.998	99.937
4.	*akb48 - heavy rotation. mp3* (4567160 byte)	Code cek prima.txt (2381 byte)	0.0521	Tepat	35.850	99.974
		Sejarah MP3.txt (8373 byte)	0.1833	Tepat	30.402	99.909
		Daftar pustaka.txt (9198 byte)	0.2014	Tepat	29.996	99.900
5.	*Somebody-That-I-Used-To-Know.mp3* (5880389)	Steganografi .txt (4429 byte)	0.0753	Tepat	34.307	99.963
		Daftar pustaka.txt (8593 byte)	0.1461	Tepat	31.414	99.928
		Latar Belakang.txt (13030 byte)	0.2216	Tepat	29.590	99.890
6.	*J-Rock Kau Curi Lagi.mp3* (7016448 byte)	belek.txt (15149 byte)	0.2159	Tepat	29.671	99.892
		Sejarah Mp3.txt (8444 byte)	0.1203	Tepat	32.212	99.940
		Latar belakang.txt (12468 byte)	0.1777	Tepat	30.523	99.911

Dari tabel 4.3 dan 4.4 dapat diketahui bahwa aplikasi yang dibuat dapat mengungkapkan isi pesan dengan tepat dan sesuai dengan pesan asli.

Berdasarkan tabel 4.3 dan 4.4 maka dapat dibuat grafik SNR untuk MP3 carrier "*result.mp3*" yang ditunjukkan pada gambar 4.6. dan 4.7



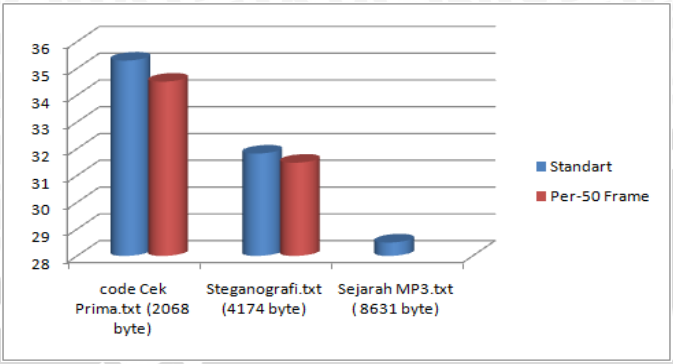
Gambar 4.6 Grafik nilai SNR untuk "*result.mp3*".



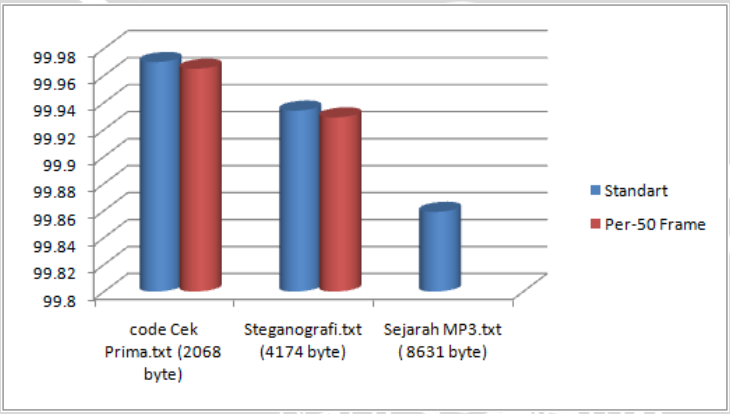
Gambar 4.7 Grafik prosentase SNR untuk "*result.mp3*".

Dari grafik SNR yang diperlihatkan pada gambar 4.6 dan 4.7 dapat diketahui bahwa terjadi penurunan nilai dan prosentase SNR pada stego MP3 hasil penyisipan pesan ke dalam *file* "*result.mp3"*. Semakin besar jumlah byte yang disispkan ke dalam *file* "*result.mp3"* maka nilai dan prosentase SNR juga semakin rendah. Pada kasus penyisipan pesan kedalam *file result.MP3*, nilai dan prosentase SNR untuk penyisipan menggunakan mode per-50 *frame* lebih besar dibandingkan dengan mode *standard*.

Untuk grafik SNR hasil penyisipan pesan pada *file* MP3 *Statan Zunea-cleopatra.mp3* ditunjukkan pada gambar 4.8 dan 4.9



Gambar 4.8 Grafik nilai SNR untuk "*Statan Zunea-cleopatra.mp3*".



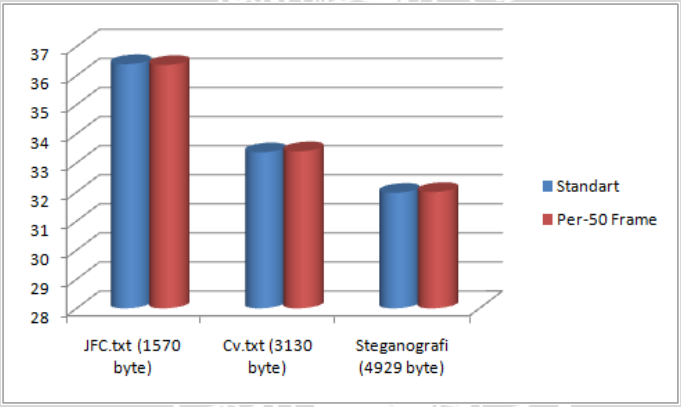
Gambar 4.9 Grafik prosentase nilai SNR untuk "*Statan Zunea-cleopatra.mp3*".

Dari grafik SNR yang diperlihatkan pada gambar 4.8 dan 4.9 menunjukkan adanya penurunan nilai dan prosentase SNR pada stego MP3 hasil penyisipan pesan pada *file* "*Statan Zunea-cleopatra.mp3*". Semakin besar jumlah byte yang disispkan ke dalam *file* "*Statan Zunea-cleopatra.mp3*" maka nilai dan prosentase SNR juga semakin rendah. Untuk kasus penyisipan pesan hasil enkripsi "Sejarah MP3.txt" yang berukuran 8631 byte tidak dapat disisipkan kedalam *file* "*Statan Zunea-cleopatra.mp3*" dengan menggunakan mode per-50 *frame*. Hal ini dikarenakan ukuran pesan hasil enkripsi yang lebih besar dibandingkan kapasitas maksimum byte yang dapat disisipkan pada *file* "*Statan Zunea-cleopatra.mp3*" dengan

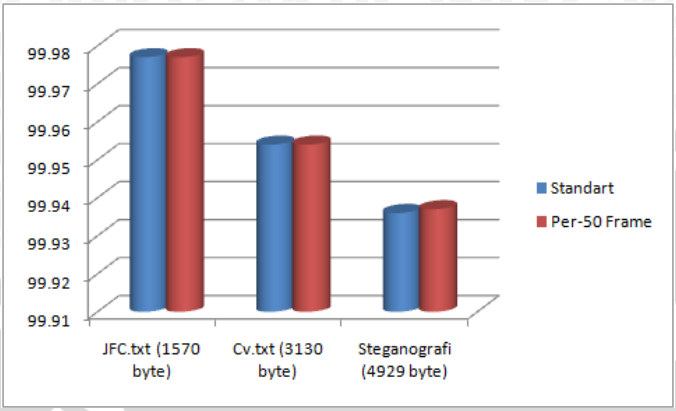
menggunakan mode per-50 *frame*. Hal tersebut ditunjukkan dengan nilai rasio ukuran pesan "Sejarah MP3.txt " terenkripsi yang bernilai 0.2991% , nilai rasio tersebut lebih besar dari nilai rasio kapasitas penyisipan pesan untuk *file* "*Statan Zunea-cleopatra.mp3"* pada mode per-50 *frame* yang telah ditunjukkan pada tabel 4.1 yang hanya bernilai 0.2272 %.

Pada kasus penyisipan pesan kedalam *file* "*Statan Zunea-cleopatra.mp3"*, nilai dan prosentase SNR untuk penyisipan menggunakan mode *standard* lebih besar dibandingkan dengan mode per-50 *frame*. Pada kasus penyisipan pesan terenkripsi Sejarah MP3.txt dengan mode *standard* nilai SNR yang didapat kurang dari 30 dB yang berarti MP3 tersebut memiliki kualitas yang kurang.

Untuk grafik SNR hasil penyisipan pesan pada *file* MP3 *I Wanna Be Billionaire.mp3* ditunjukkan pada gambar 4.10 dan 4.11



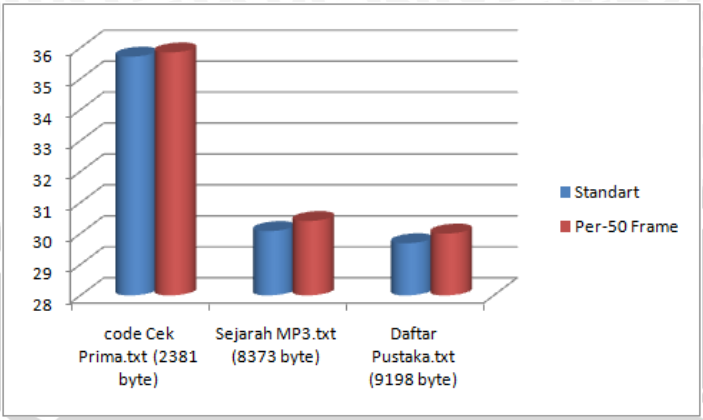
Gambar 4.10 Grafik nilai SNR untuk "*I Wanna Be Billionaire.mp3*".



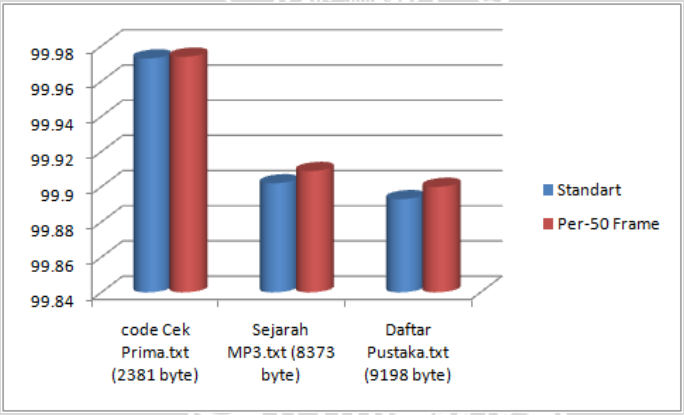
Gambar 4.11 Grafik prosentase SNR untuk "*I Wanna Be Billionaire.mp3*".

Dari grafik SNR yang diperlihatkan pada gambar 4.10 dan 4.11 dapat diketahui bahwa terjadi penurunan nilai dan prosentase SNR pada stego MP3 hasil penyisipan pesan ke dalam *file* "*I Wanna Be Billionaire.mp3*". Semakin besar jumlah byte yang disispkan ke dalam *file* "*I Wanna Be Billionaire.mp3*" maka nilai dan prosentase SNR juga semakin rendah. Pada kasus penyisipan pesan kedalam *file* "*I Wanna Be Billionaire.mp3*", nilai SNR untuk penyisipan menggunakan mode per-50 *frame* lebih besar dibandingkan dengan mode *standard*, sedangkan untuk nilai prosentase SNR pada kasus penyisipan pesan hasil enkripsi JFC.txt dan Cv.txt penyisipan dengan menggunakan mode *standard* dan mode per 50 *frame* memiliki prosentase yang sama besar.

Untuk grafik SNR hasil penyisipan pesan pada *file* MP3 *akb48 - heavy rotation*. mp3 ditunjukkan pada gambar 4.12 dan 4.13



Gambar 4.12 Grafik nilai SNR untuk "*akb48 - heavy rotation. mp3*".

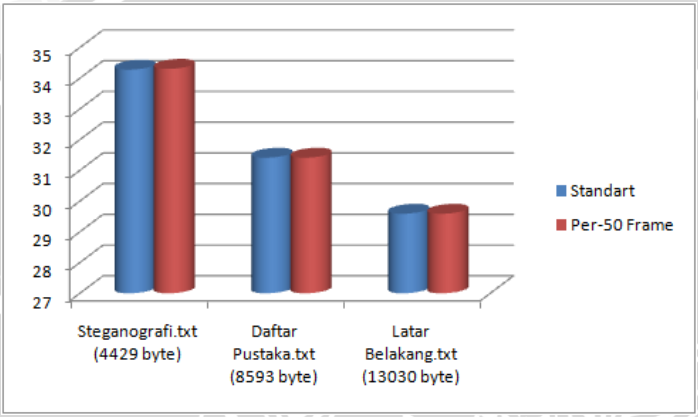


Gambar 4.13 Grafik prosentase SNR untuk "*akb48 - heavy rotation. mp3*".

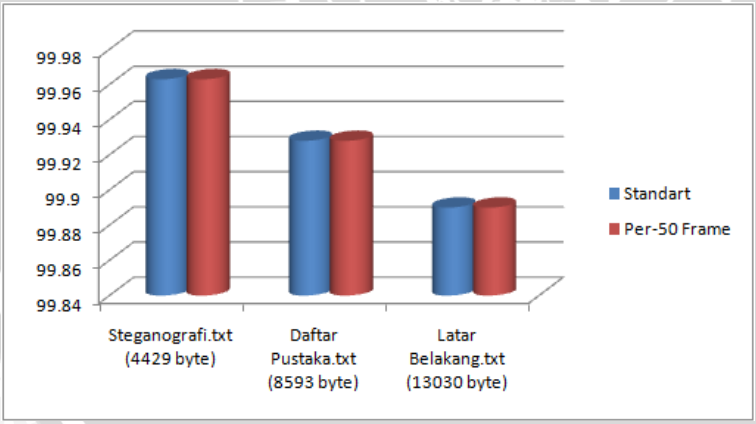
Dari grafik SNR yang diperlihatkan pada gambar 4.12 dan 4.13 dapat diketahui bahwa terjadi penurunan nilai dan prosentase SNR pada stego MP3 hasil penyisipan pesan ke dalam *file* "akb48 - heavy rotation.mp3". Semakin besar jumlah byte yang disispkan ke dalam *file* "*akb48 – heavy rotation.mp3*" maka nilai dan prosentase SNR juga akan semakin rendah. Pada kasus penyisipan pesan kedalam *file* "*akb48 - heavy rotation.mp3*", nilai dan prosentase SNR untuk penyisipan menggunakan mode per-50 *frame* lebih besar

dibandingkan dengan mode *standard*. Pada kasus penyisipan pesan hasil enkripsi Daftar Pustaka.txt baik menggunakan mode *standard* atau per 50 *frame* nilai SNR yang didapat kurang dari 30 dB.

Untuk grafik SNR hasil penyisipan pesan pada *file* MP3 *Somebody-That-I-Used-To-Know.mp3* ditunjukkan pada gambar 4.14 dan 4.15



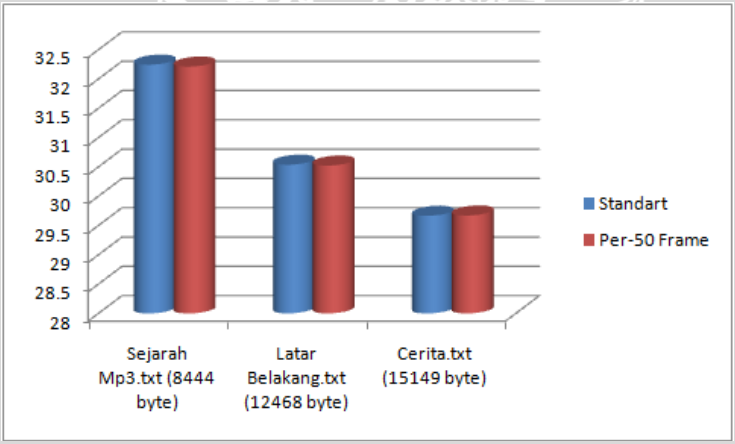
Gambar 4.14 Grafik nilai SNR untuk "*Somebody-That-I-Used-To-Know.mp3*".



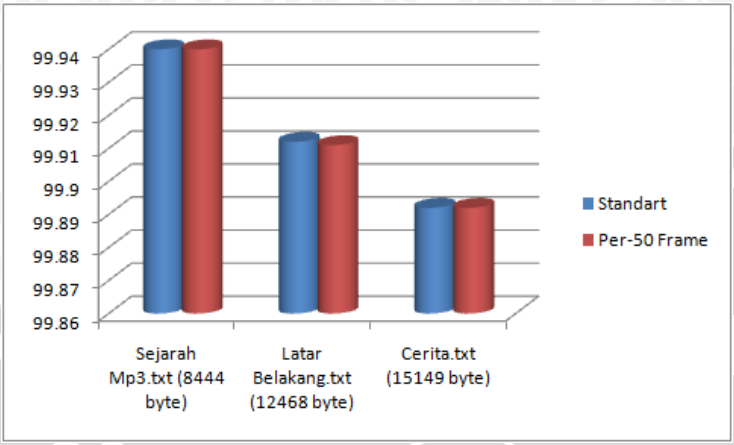
Gambar 4.15 Grafik prosentase SNR untuk "*Somebody-That-I-Used-To-Know.mp3*".

Dari grafik SNR yang diperlihatkan pada gambar 4.14 dan 4.15 dapat diketahui bahwa terjadi penurunan nilai dan prosentase SNR pada stego MP3 hasil penyisipan pesan ke dalam *file* "*Somebody-That-I-Used-To-Know.mp3*". Semakin besar jumlah byte yang disispkan ke dalam *file* "*Somebody-That-I-Used-To-Know.mp3*" maka nilai dan prosentase SNR juga akan semakin rendah. Pada kasus penyisipan pesan kedalam *file* "*Somebody-That-I-Used-To-Know.mp3*", nilai SNR untuk kasus penyisipan pesan hasil enkripsi Steganografi.txt dengan menggunakan mode *standard* lebih besar daripada menggunakan mode per-50 *frame*. Sedangkan untuk dua kasus lainya nilai SNR dengan menggunakan mode *standard* sama dengan nilai SNR dengan menggunakan mode per-50 *frame*. Pada kasus penyisipan pesan hasil enkripsi Latar Belakang.txt baik menggunakan mode *standard* atau per 50 *frame* nilai SNR yang didapat kurang dari 30 dB. Untuk prosentase SNR, hasil penyisipan pesan ke dalam *file* "*Somebody-That-I-Used-To-Know.mp3*" dengan menggunakan mode *standard* memiliki prosentase SNR yang sama dengan menggunakan mode per-50 *frame*.

Untuk grafik SNR hasil penyisipan pesan pada *file* MP3 *J-Rocks Kau Curi Lagi.mp3* ditunjukkan pada gambar 4.16 dan 4.17.



Gambar 4.16 Grafik nilai SNR untuk "*J-Rocks Kau Curi Lagi.mp3*".



Gambar 4.17 Grafik prosentase SNR untuk "*J-Rocks Kau Curi Lagi.mp3*".

Dari grafik SNR yang diperlihatkan pada gambar 4.16 dan 4.17 dapat diketahui bahwa terjadi penurunan nilai dan prosentase SNR pada stego MP3 hasil penyisipan pesan ke dalam *file* "*J-Rocks Kau Curi Lagi.mp3*". Semakin besar jumlah byte yang disispkan ke dalam *file* "*J-Rocks Kau Curi Lagi.mp3*" maka nilai dan prosentase SNR juga akan semakin rendah. Pada kasus penyisipan pesan hasil enkripsi dari pesan Cerita.txt nilai SNR pada hasil penyisipan menggunakan mode per-50 *frame* lebih besar daripada menggunakan mode *standard*, akan tetapi untuk kasus penyisipan hasil enkripsi Sejarah MP3.txt dan Latar Belakang.txt nilai SNR akan lebih besar bila menggunakan mode *standard*. Pada kasus penyisipan pesan hasil enkripsi Cerita.txt baik menggunakan mode *standard* atau per 50 *frame* nilai SNR yang didapat kurang dari 30 dB. Untuk prosentase SNR hasil penyisipan pesan terenkripsi Cerita.txt dan Sejarah MP3.txt menggunakan mode *standard* memiliki prosentase yang sama dengan bila menggunakan mode per-50 *frame*. Pada kasus penyisipan hasil enkripsi Latar belakang.txt prosentase SNR lebih besar bila menggunakan mode *standard*.

Pengujian kualitas audio juga dilakukan dengan menghitung nilai *Mean Opinion Score (MOS)*. Hasil pengujian nilai MOS didapat dari hasil penilaian subjektif sepuluh orang responden terhadap dua belas kualitas suara *audio* MP3 yang telah disisipi pesan. Pengujian

nilai MOS dibedakan menjadi dua, yaitu pengujian terhadap MP3 hasil penyisipan dengan mode *standard* dan MP3 hasil penyisipan dengan mode per-50 *frame*. Tabel pengujian MOS untuk mode *standard* dan per-50 *frame* ditunjukkan pada tabel 4.5 dan 4.6

Tabel 4.5 Pengujian MOS mode *standard*

No.	Nama Berkas MP3	Nilai Dari Tiap Penguji ke-										Nilai MOS
		1	2	3	4	5	6	7	8	9	10	
1.	result_std2	3	3	4	4	4	4	4	2	4	4	3,6
2	Statan Zunea cleopatra_std2	4	5	4	4	3	4	4	2	4	4	3,8
3.	i wanna be a billionaire-std3	4	2	2	2	1	3	2	1	4	3	2,4
4.	akb48-heavy rotation-std3	4	4	4	4	4	4	4	5	5	4	4,8
5.	Somebody-That-I-UsedTo-Know-std2	2	1	2	2	2	3	2	1	3	2	2
6.	J-Rocks Kau Curi Lagi-std3	2	1	1	2	2	2	2	1	2	2	1,7
											Rata-Rata	3.05

Tabel 4.6 Pengujian MOS mode per-50 *frame*

No.	Nama Berkas MP3	Nilai Dari Tiap Penguji ke-										Nilai MOS
		1	2	3	4	5	6	7	8	9	10	
1.	result_50F2	2	3	3	2	2	3	3	3	3	3	2,7
2	Statan Zunea cleopatra_50F2	2	3	3	1	3	3	2	4	2	2	2,5
3.	i wanna be a billionaire-50F3	3	4	3	3	2	2	1	3	3	4	2,8
4.	akb48-heavy rotation-50F3	3	3	3	3	4	4	3	4	4	4	3,5
5.	Somebody-That-I-Used-To-Know-50F2	3	4	2	2	2	2	2	3	3	3	2,6

6.	J-Rock Kau Curi Lagi-50F3	3	4	3	3	3	1	2	3	2	2	2,6
											Rata-Rata	2.783

Dari nilai MOS untuk setiap MP3 carrier yang sama yang terlihat pada tabel 4.5 dan 4.6 menunjukkan bahwa kualitas MP3 hasil penyisipan pesan untuk setiap mode berbeda-beda. Pada kasus MP3 dengan judul *result, Statan Zunea-cleopatra,* dan *akb48-heavy rotation* nilai MOS dari penyisipan mode *standard* lebih besar dibandingkan dengan mode per-50 *frame*, akan tetapi untuk MP3 dengan judul *i wanna be a billionaire, Somebody-That-I-Used-To-Know*, dan *J-Rock Kau Curi Lagi* nilai MOS dari penyisipan dengan mode per-50 *frame*-lah yang memiliki nilai MOS lebih besar.

Rata-rata nilai MOS untuk setiap mode penyisipan adalah 3.05 untuk mode *standard* dan 2.789 untuk mode per-50 *frame*. Rata-rata tersebut menunjukkan bahwa kualitas MP3 berkisar antara derau sedikit menggangu dan derau menggangu. Dari rata-rata nilai MOS tersebut juga dapat diketahui bahwa responden lebih menyukai penyisipan dengan mode *standard* dibanding penyisipan dengan mode per-50 *frame*.

Secara umum, dari pengamatan seluruh pengujian SNR dan ketepatan pengungkapan pesan yang dilakukan yang ditunjukkan pada tabel 4.3 dan 4.4 dapat dikatakan bahwa aplikasi yang telah dibuat dapat mengungkapkan pesan secara tepat. Kualitas MP3 yang telah disisipi pesan sangat ditentukan oleh besar ukuran pesan hasil enkripsi yang disisipkan. Semakin besar ukuran pesan hasil enkripsi yang disispkan ke dalam MP3, maka kualitas audio semakin menurun. Penurunan juga terjadi pada kemiripan karakteristik bit MP3 hasil penyisipan dengan bit MP3 asli. Kedua hal ini ditunjukkan dengan adanya penurunan nilai SNR dalam satuan dB dan penurunan prosentase SNR seiring dengan bertambahnya ukuran pesan hasil enkripsi yang disisipkan.

Penurunan kualitas paling besar ditunjukkan pada penyisipan pesan hasil enkripsi Sejarah MP3.txt yang berukuran 8631 byte ke dalam *file* MP3 *Statan Zunea-cleopatra.mp3* dengan menggunakan mode *standard*. Hal ini ditunjukkan dengan nilai SNR sebesar 28.5 dB yang berarti MP3 tersebut tidak layak untuk didengar karena nilai SNR yang kurang dari 30 dB. Selain MP3 tersebut ada juga beberapa

stego MP3 hasil pengujian yang tidak layak untuk didengar karena memiliki nilai SNR kurang dari 30 dB. Sedangkan stego MP3 dengan kualitas terbaik ditunjukkan pada pada penyisipan hasil enkripsi dari pesan Uji Steganografi.txt yang berukuran 193 byte ke dalam *file* MP3 *result.MP3* dengan menggunakan mode per-50 *frame* dengan nilai SNR sebesar 39.964 dB.

Dari nilai dan prosentase SNR yang didapat melalui hasil pengujian SNR dan ketepatan pengungkapan pesan pada kedua mode penyisipan dapat diketahui bahwa sebuah MP3 yang disisipi pesan terenkripsi yang sama memiliki kualitas yang berbeda tergantung dari mode penyisipanya. Hal ini ditunjukkan dengan perbedaan SNR dari setiap mode penyisipan. Dari hasil uji SNR ada beberapa MP3 yang memiliki kualitas lebih baik bila disisipi pesan dengan menggunakan mode *standard*, dan beberapa lagi memiliki kualitas lebih baik bila disisipi pesan dengan menggunakan mode per-50 *frame*. Hal ini didukung dengan nilai MOS yang diberikan oleh responden. Hasil nilai MOS dari beberapa MP3 lebih tinggi pada mode *standard*, dan beberapa lagi lebih tinggi pada mode per-50 *frame*.

Dari rata-rata nilai dan prosentase SNR yang dihasilkan memiliki nilai yang cukup tinggi, akan tetapi nilai MOS yang dihasilkan dari rata-rata penilaian subjektif responden tergolong rendah karena pada kualitas audio MP3 hasil penyisipan pesan dinyatakan terdengar derau yang sedikit menggangu dan derau yang menggangu. Perbedaan ini dikarenakan SNR diukur berdasarkan banyaknya perubahan bit pada MP3 setelah disisipi pesan, sedangkan file MP3 merupakan format file audio yang terkompresi dimana tiap bit dalam MP3 sebenarnya merupakan hasil pemampatan dari banyak bit, sehingga adanya perubahan satu bit dalam MP3 akan merubah banyak informasi yang terkandung dalam file MP3 tersebut. Hal inilah yang menjadikan kualitas MP3 hasil penyisipan pesan akan terdengar derau yang menggangu meskipun hanya sedikit jumlah bit yang terubah.

4.4.2. Hasil Pengujian dan Pembahasan Tingkat Keamanan Algoritma RSA

Pengujian tingkat keamanan algoritma RSA dilakukan dengan cara Brute Force Attack (BFA). BFA dilakukan untuk

mendapatkan kunci rahasia yang digunakan untuk proses dekripsi hasil pengungkapan pesan. Parameter yang akan digunakan untuk pengujian adalah banyaknya iterasi yang dilakukan untuk mendapatkan kunci rahasia. Semakin banyak iterasi yang perlu dilakukan untuk mendapatkan kunci rahasia maka pesan terenkripsi akan semakin sulit untuk dipecahkan oleh pihak yang tidak bersangkutan.

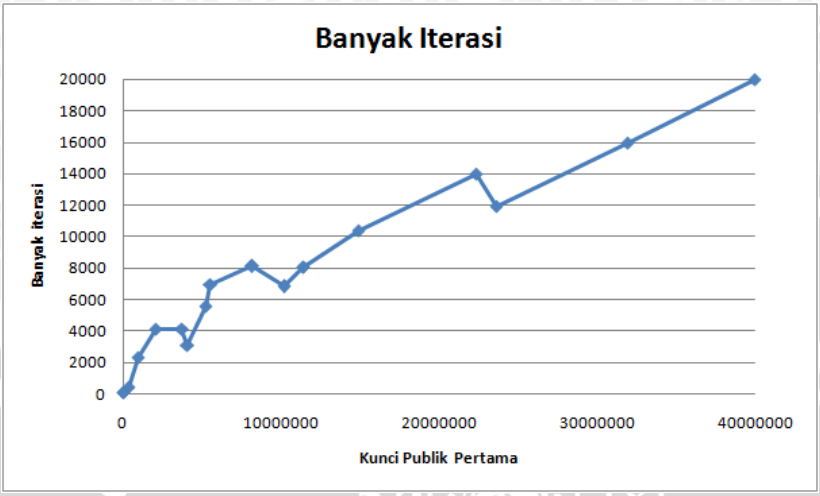
Brute Force Attack yang akan dilakukan adalah mencoba memfaktorkan kunci publik pertama sesuai dengan persamaan 2.16, hasil dari pemfaktoran tersebut akan digunakan untuk menghitung nilai dari kunci rahasia yang didasarkan pada persamaan 2.1 dan persamaan 2.11. Proses pemfaktoran akan terus dilakukan hingga kunci rahasia hasil Brute Force Attack sama dengan kunci rahasia yang menjadi nilai masukan. Kunci publik pertama yang akan difaktorkan merupakan kunci-kunci yang didapatkan dari hasil enkripsi pada saat penyisipan pesan rahasia ke dalam file MP3. Hasil dari pengujian banyaknya iterasi ditunjukkan pada tabel 4.7.

Tabel 4.7 Hasil pengujian banyaknya iterasi Brute Force Attack

No	Kunci Publik Pertama	Kunci Publik Kedua	Kunci Rahasia	Faktor Prima	Banyak Iterasi
1.	10057	4101	9421	a.89 b.113	88
2.	55249457	27462421	6773821	a. 7937 b.6961	6960
3.	20851613	9066413	17461205	a.5087 b.4099	4098
4.	113949743	17067701	49152797	a.14087 b.8089	8088
5.	113949743	1897	94228969	a.14087 b.8089	8088
6.	37274437	7898459	36250679	a. 9067 b. 4111	4110
7.	40707203	2181799	4371799	a. 3121 b. 13043	3120
8.	81973583	29442571	48210979	a. 8117 b. 10099	8116
9.	148764043	58167799	61107655	a. 14347	10368

				b. 10369	
10.	3708233	1547201	2728901	a. 439	438
				b. 8447	
11.	318844861	111748935	237464019	a. 15959	15958
				b. 19979	
12.	223346297	80957941	161730841	a. 13967	13966
				b. 15991	
13.	236301937	107125261	74898085	a. 11923	11922
				b. 19819	
14.	102073063	39966635	96474299	a. 14947	6828
				b. 6829	
15.	9464981	3609061	33229	a. 4057	2322
				b. 2333	
16.	9379	3231	415	a. 113	82
				b. 83	
17.	399680063	150416909	242352389	a. 19993	19990
				b. 19991	
18.	52452523	12768643	39079531	a. 5557	5556
				b. 9439	

Berdasarkan tabel 4.7 dapat diketahui bahwa kriptografi dengan algoritma RSA dapat diserang menggunakan Brute Force Attack dengan cara memfaktorkan kunci publik yang pertama. Hal ini terlihat dari keberhasilan seluruh percobaan Brute Force Attack dalam mendapatkan faktor prima dari kunci publik pertama dan nilai dari kunci rahasia. Selain itu dari tabel 4.7 dapat dibuat grafik hubungan antara kunci publik pertama dengan banyaknya iterasi yang dibutuhkan untuk mendapatkan kunci rahasia yang ditunjukkan pada gambar 4.18.



Gambar 4.18 Grafik hubungan kunci publik pertama dengan banyaknya iterasi

Dari gambar 4.18 terlihat bahwa terjadi kecenderungan meningkatnya jumlah iterasi seiring dengan meningkatnya nilai kunci publik yang pertama. Akan tetapi terlihat adanya penurunan jumlah iterasi di beberapa titik ketika kunci publik pertama semakin besar, untuk itu perlu dilakukan pengujian regresi untuk mengetahui hubungan antara kunci publik pertama dengan banyaknya iterasi. Hasil dari pengujian regresi yang dilakukan ditunjukkan pada tabel 4.8 dan 4.9.

Tabel 4.8 Statistik Regresi

Nama Variabel Statistik	Nilai
Multiple R	0.96913
R Square	0.93922
Adjusted R Square	0.93542

Dari tabel 4.8 dapat dilihat bahwa nilai *Multiple R* atau R majemuk yang menunjukkan tingkat keeratan hubungan antara kunci publik pertama dan banyaknya iterasi menunjukkan nilai positif yang mendekati nilai satu sebesar 0.96913. Hal ini menunjukkan adanya hubungan yang kuat antara kunci publik pertama dan banyaknya iterasi. Nilai *R Square* dan *Adjusted R Square* yang bernilai hampir

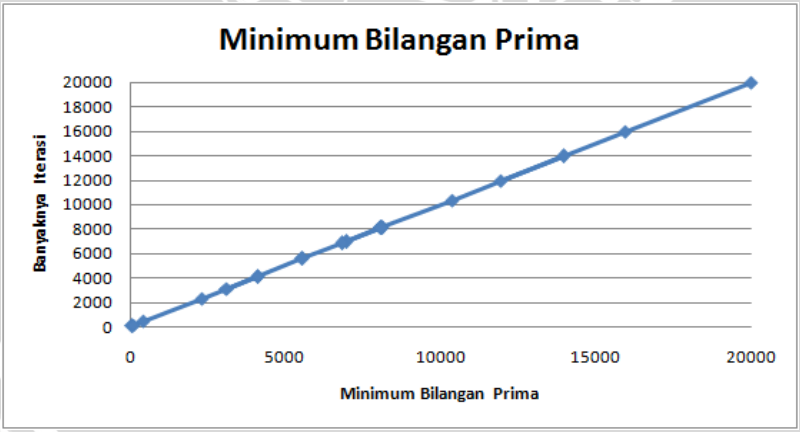
mendekati nilai satu yaitu 0.93922 dan 0.93542 menunjukkan kecocokan model yang sangat baik untuk persamaan regresi.

Tabel 4.9 Koefisien Persamaan Regresi

Nama Koefisien	Nilai Koefisien
Intercept	2149.099652
Kunci Publik Pertama	0.000046673

Dari tabel 4.9 dapat dibentuk persamaan regresi untuk banyaknya iterasi yaitu
Y = 2149.0996 + 0.000046673 * X, dimana Y merupakan banyaknya iterasi dan X merupakan kunci publik pertama.
Dari persamaan regresi untuk banyaknya iterasi tersebut diketahui bahwa koefisien bernilai positif sehingga pengaruh kunci publik pertama searah atau berbanding lurus dengan banyaknya iterasi. Setiap kenaikan satu nilai kunci publik pertama maka banyaknya iterasi akan meningkat sebesar 0.000046673.

Dari tabel 4.7 terlihat bahwa hal yang paling mempengaruhi banyaknya iterasi adalah nilai minimum dari dua bilangan prima yang dipilih untuk melakukan enkripsi. Grafik hubungan nilai minimum bilangan prima dan banyaknya iterasi ditunjukkan pada gambar 4.19



Gambar 4.19 Grafik hubungan nilai minimum bilangan prima dan banyaknya iterasi

Dari grafik 4.19 diketahui adanya pengaruh searah dari nilai minimum bilangan prima yang dipilih terhadap banyaknya iterasi *Brute Force Attack* untuk mendapatkan kunci rahasia. Semakin besar nilai minimum bilangan prima yang dipilih maka semakin banyak iterasi *Brute Force Attack* yang dibutuhkan untuk mendapatkan kunci rahasia. Hal ini dikarenakan proses *Brute Force Attack* dilakukan dengan cara memfaktorkan kunci publik pertama yang merupakan hasil perkalian dua bilangan prima. Sehingga untuk mendapatkan tingkat keamanan yang lebih tinggi perlu dilakukan pemilihan bilangan prima yang bernilai besar.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Kesimpulan dari penelitian ini adalah sebagai berikut :

1. *Metode Least Significant Bit (LSB)* dapat diimplementasikan untuk menyisipkan pesan terenkripsi ke dalam *file* MP3. Langkah awal proses penyisipan adalah dengan mengenkripsi pesan dengan algoritma RSA kemudian mengubah hasil enkripsi atau chipertext menjadi bentuk biner. Selanjutnya LSB dari *file* MP3 akan disubtitusikan dengan hasil biner dari chipertext yang kemudian akan dibentuk *file* MP3 baru.
2. Kualitas MP3 hasil penyisipan pesan yang didasarkan pada SNR, mengalami penurunan, prosentase dan nilai SNR akan turun seiring bertambahnya ukuran pesan terenkripsi yang disisipkan. Nilai SNR yang dihasilkan dari penyisipan mode *standart* berkisar antara 28,5 dB hingga 39,759 dB dan untuk prosentase SNR-nya berkisar antara 99,859% hingga 99,989%. Sedangkan nilai SNR yang dihasilkan dari penyisipan dengan mode per-50 *frame* berkisar antara 29,590 dB hingga 39,964 dB untuk prosentase SNR-nya berkisar antara 99,89% hingga 99,99%. Untuk kualitas MP3 hasil penyisipan pesan yang didasarkan pada nilai MOS memiliki rata-rata nilai 3,05 untuk penyisipan dengan mode *standart* yang berarti kualitas suara terdapat derau yang sedikit menggangu. Sedangkan nilai MOS untuk penyisipan dengan mode per-50 *frame* memiliki rata-rata 2,783 yang berarti kualitas suara terdapat derau yang mengganggu. Perbedaan kualitas audio berdasarkan nilai MOS yang tergolong rendah dan kualitas audio berdasarkan SNR yang tinggi dikarenakan file MP3 merupakan file audio terkompresi sehingga perubahan sedikit bit dalam MP3 akan mengubah banyak informasi audio.
3. Algoritma kriptografi RSA yang digunakan, dapat diserang menggunakan *Brute Force Attack* dengan cara memfaktorkan kunci publik yang pertama yang kemudian dapat dihitung nilai dari kunci rahasia. Banyaknya iterasi atau perulangan yang dihasilkan dalam *Brute Force Attack* untuk mendapatkan kunci rahasia berkisar antara 82 hingga 19990 perulangan. Banyaknya

iterasi atau perulangan yang digunakan *Brute Force Attack* bergantung pada besarnya kunci publik pertama dan nilai minimum bilangan prima yang digunakan pada saat enkripsi. Semakin besar nilai minimum bilangan prima yang dipilih maka semakin banyak iterasi *Brute Force Attack* yang dibutuhkan yang berarti hasil enkripsi semakin aman.

5.2. Saran

Kekurangan dari hasil penelitian penyisipan pesan terenkripsi ke dalam MP3 menggunakan Least Significant Bit yang telah dilakukan adalah kualitas suara MP3 hasil penyisipan pesan yang didasarkan pada nilai MOS terdengar banyak derau yang diakibatkan perubahan bit MP3 yang merupakan file audio terkompresi, sehingga untuk penelitian selanjutnya diharapkan metode Least Significant Bit dapat diterapkan untuk format audio tak terkompresi seperti WAV, AIFF, atau AU

DAFTAR PUSTAKA

Ariyus, Dony. 2007. Keamanan Multimedia. Yogyakarta: Andi Offset.

Ariyus, Dony. 2008. Pengantar Ilmu Kriptografi : Teori, Analisis, Implementasi. Yogyakarta: Andi Offset.

Arubusman, Yusrian Roman. 2007. Audio Steganografi. Skripsi Teknik Informatika Universitas Gunadarma. Jakarta.

Baskara, Tara. 2008. Studi dan Implementasi Steganografi Pada MP3 Dengan Teknik *Spread Spectrum*. Tugas Akhir Program Studi Sarjana Teknik Informatika ITB. Bandung

Basuki , A. 2005. Pengolahan Citra Digital Menggunakan Visual Basic. Yogyakarta : Graha Ilmu.

Basuki, D. Kurnia, Nadhori, I. Uzzin dan Maulana, A. Mansur. 2009. *Data Hiding Steganograph Pada File Image Menggunakan Metode Least Significant Bit*. Jurnal Tugas Akhir Jurusan Teknik Informatika ITS, Surabaya.

Cox, Ingemar J., Miller, Matthew L., Bloom, Jeffrey A., Fridrich, Jesica And Kalker,Ton. 2008. *Digital Watermarking And Steganography Second Edition*. Burlington: Morgan Kaufman Publisher,

Hapsari, Dian Dwi. 2009. Aplikasi Video Steganography Dengan Metode Least Significant Bit (LSB). Jurnal Konferensi Nasional Sistem dan Informatika, Bali.

Kizza, Joseph Mingga. 2005. *Computer Network Security*. New York : Springer Science and Business Media, Inc.

Kalangi, Jeff Andre. 2010. Pembuatan Aplikasi *Steganography* Pada *File Audio* MP3 Dengan Metode *Parity Coding*. Tugas Akhir Jurusan Teknik Informatika Unirversitas Komputer Indonesia. Bandung.

Kurniawan, Indra. 2008. Amankan Data Dengan Steganografi. http://user36.wordpress.com. Diakses tanggal 19 Maret 2012.

Kurniawan, Yusuf. 2004. Keamanan Internet dan Jaringan Komunikasi. Bandung: Informatika.

Maya. 2006. *Steganografi LSB*. http://maya9luthu.blogsome.com/2006/12/11/steganografi-lsb/. Diakses tanggal 19 Maret 2012.

Morgana. 2011. Dasar Kriptografi. http://koboyit.com/2011/03/dasar-kriptografi.html. Diakses tanggal 18 Maret 2012.

Morgana. 2011. Steganografi. http://koboyit.com/2011/03/steganografi.html. Diakses tanggal 18 Maret 2012.

Munir, Renaldi. 2004. Pengolahan Citra Digital Dengan Pendekatan Algoritmik. Bandung : Informatika.

Pramudianti, Shinta. 2012. Kriptografi. http://shinta-pramudianti.ugm.ac.id/2012/02/23/ pengenalan-kriptografi/. Diakses tanggal 19 Maret 2012.

Raissi, Rassol. 2002. *The Theory Behind MP3*. http://www.mp3-tech.org/programmer/docs/ mp3_theory.pdf. Diakses tanggal 20 Maret 2012.

Rasyid, M. Fajrin. 2009. Kriptografi *Audio* Dengan Teknik Interferensi Data No Biner. Jurnal Tugas Akhir Tehnik Informatika ITB, Bandung.

Rengganis, Suhati Novalia. 2009. Steganografi Citra Digital Dengan Metode Modifikasi *LSB (Least Significant Bit).* http://suhatinovalia.upi.edu/seminar-ilmu-komputer/. Diakses tanggal 19 Maret 2012.

Riyanto, M. Zaki, Ardhian, Ardi. 2008. Kriptografi Kunci Publik : Sandi RSA. Jurnal Kelompok Studi Sandi, Yogyakarta.

Rosediana, Diah. 2011. Steganografi Definisi. http://user36.wordpress.com/2008/06/09/. Diakses tanggal 19 Maret 2012.

Rosgani. 2008. Sejarah Singkat MP3. http://www.angelfire.com/id/rosgani/MP3STORY.HTML. Diakses tanggal 19 Maret 2012.

Unuth, Nadeem. 2012. *Mean Opinion Score (MOS) – A Measure Of Voice Quality*. Diakses tanggal 22 Juni 2012.

Wafda, Naufal Fakhzani. 2011. Definisi Suara, *Digitizing, Nyquist Theorema, Coding Audio Using PSM.* http://naufalfakhzani20.com/2011/10/. Diakses tanggal 19 Maret 2012.