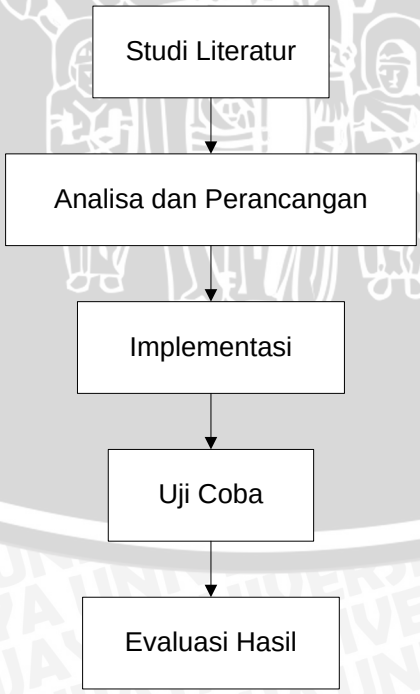
BAB III
METODOLOGI DAN PERANCANGAN

Pembahasan pada bab ini meliputi metode dan langkah-langkah perancangan yang dilakukan dalam penelitian untuk menyembunyikan pesan rahasia terenkripsi pada berkas audio MP3 menggunakan metode *Parity Coding*.

Langkah-langkah yang dilakukan dalam penelitian ini meliputi :

1. Melakukan studi literatur mengenai *Parity Coding*, algoritma kriptografi RSA, dan literatur lain yang berkatian seperti yang telah dijelaskan pada bab 2.
2. Menganalisa dan melakukan perancangan sistem.
3. Mengimplementasikan dalam bentuk perangkat lunak berdasarkan analisis dan perancangan yang telah dilakukan.
4. Melakukan uji coba terhadap perangkat lunak yang telah dibuat.
5. Mengevaluasi *output* hasil analisa dari sistem.

Langkah-langkah penelitian ini digambarkan pada Gambar 3.1



Gambar 3.1 Alur Penelitian

3.1 Deskripsi Umum Sistem

Perangkat lunak yang akan dibuat merupakan perangkat lunak yang dapat menyembunyikan pesan rahasia berupa teks ke dalam sebuah berkas audio dengan format MP3. Perangkat lunak akan mengimplementasikan teknik kriptografi dengan algortima RSA dan teknik steganografi dengan metode *parity coding*.

Teknik kriptografi RSA digunakan untuk mengenkripsi pesan agar arti dari pesan tersebut tetap terjaga kerahasiaannya. RSA melakukan enkripsi pada *plaintext* atau pesan asli menjadi sebuah *ciphertext* menggunakan kunci publik. Pesan yang sudah terenkripsi kemudian disembunyikan ke dalam media penampung berkas MP3 menggunakan teknik steganografi dengan metode *parity coding*.

Pada metode *parity coding*, bit-bit dari pesan rahasia akan disembunyikan ke setiap region dari berkas MP3 yang telah dibagi menjadi beberapa region sebanyak panjang pesan. Penyisipan bit dari pesan rahasia pada tiap region berdasarkan *parity bit* pada tiap region.

Pengungkapan pesan dilakukan untuk mendapatkan kembali pesan rahasia yang disembunyikan pada berkas MP3. Pesan diektraksi dari berkas MP3 dengan menyusun nilai parity bit dari tiap region. Selanjutnya dilakukan dekripsi menggunakan algoritma RSA dengan kunci privat untuk mendapatkan pesan asli sehingga arti dari pesan tersebut dapat diketahui.

3.2 Perancangan Sistem

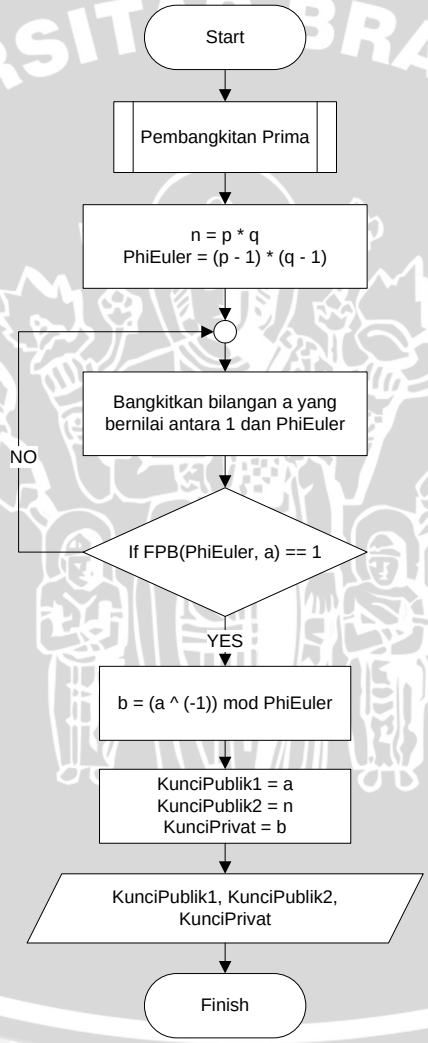
Perangkat lunak yang akan dibuat memilki dua proses utama, yaitu proses penyembunyian pesan dan proses pengungkapan pesan. Selain dua proses utama tersebut, terdapat proses pembentukan kunci. Proses pembentukan kunci bertujuan untuk mencari kunci publik dan kunci privat. Kunci publik digunakan dalam proses enkripsi pada penyembunyian pesan sedangkan kunci privat digunakan dalam proses dekripsi pada pengungkapan pesan.

Langkah-langkah untuk proses pembentukan kunci adalah sebagai berikut :

1. Dilakukan proses pembangkitan bilangan bilangan prima. Hasil keluaran dari proses tersebut adalah bilangan prima *p* dan *q*.

2. Dihitung nilai n menggunakan persamaan 2.10. Dihitung pula nilai $\varphi(n)$ menggunakan persamaan 2.4.
3. Pilih bilangan a yang bernilai antara 1 dan $\varphi(n)$. Jika faktor persekutuan terbesar dari $\varphi(n)$ dan a bernilai 1, maka dilakukan langkah 4.
4. Dihitung nilai b menggunakan persamaan 2.11.
5. Hasil akhir berupa kunci publik 1 (a), kunci publik 2 (n), dan kunci privat (b).

Diagram alir untuk proses pembentukan kunci ditunjukkan pada Gambar 3.2.



Gambar 3.2 Diagram Alir Proses Pembentukan Kunci

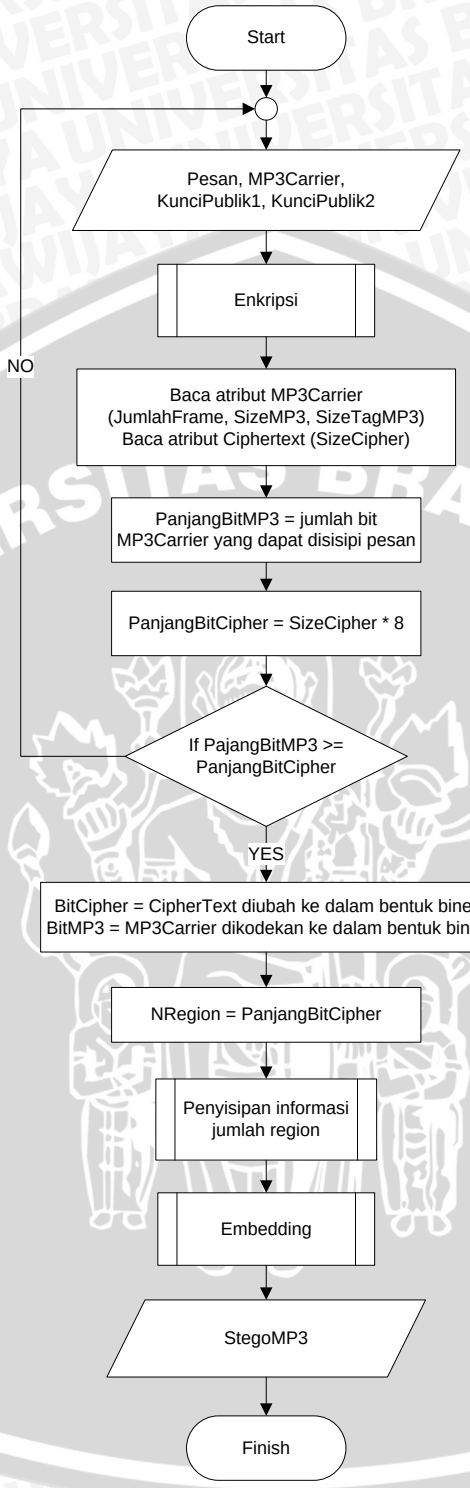
Proses penyembunyian pesan bertujuan untuk menjaga keamanan pesan rahasia dengan melakukan enkripsi dan penyembunyian pesan pada berkas MP3.

Pesan asli atau *plaintext* akan dienkripsi terlebih dahulu. Pesan yang sudah terenkripsi atau *ciphertext* kemudian disembunyikan ke dalam MP3 *carrier*, yaitu media penampung berupa berkas MP3 dengan melakukan proses *embedding*. Hasil dari proses *embedding* adalah *stego* MP3, yaitu berkas MP3 yang sudah disisipi pesan.

Langkah-langkah untuk proses penyembunyian pesan adalah sebagai berikut :

1. Masukan yang digunakan adalah pesan, berkas MP3 yang akan digunakan sebagai media penampung (MP3 *carrier*), dan pasangan kunci publik (kunci publik 1 dan kunci publik 2).
2. Dilakukan proses enkripsi untuk mengubah pesan asli atau *plaintext* menjadi *ciphertext*.
3. Dilakukan pembacaan atribut pada MP3 *carrier* seperti jumlah *frame*, ukuran berkas, dan panjang tag *id* MP3.
4. Dihitung panjang bit MP3 *carrier* yang dapat digunakan untuk menampung pesan, kemudian dibandingkan dengan panjang bit pesan yang sudah terenkripsi (*ciphertext*). Perbandingan dilakukan untuk mengetahui apakah MP3 *carrier* dapat digunakan untuk menampung pesan. Apabila MP3 *carrier* dapat digunakan untuk menampung pesan, maka dilakukan langkah 5.
5. MP3 *carrier* dikodekan ke dalam bentuk byte kemudian diubah ke dalam bentuk biner. Selanjutnya, *ciphertext* diubah ke dalam kode ASCII, kemudian diubah ke dalam bentuk biner.
6. Dilakukan proses penyisipan informasi jumlah region ke dalam bit MP3 *carrier*. Proses ini bertujuan untuk menyisipkan informasi panjang dari bit *ciphertext* yang merepresentasikan jumlah pembagian region pada bit MP3 *carrier*. Informasi jumlah region digunakan pada proses *retrieving* pada pengungkapan pesan.
7. Dilakukan proses *embedding* untuk menyembunyikan bit *ciphertext* ke dalam bit MP3 *carrier* menggunakan metode *Parity Coding*.
8. Hasil akhir adalah adalah *stego* MP3, yaitu berkas MP3 yang sudah disisipi pesan.

Diagram alir untuk proses penyembunyian pesan ditunjukkan pada Gambar 3.3.



Gambar 3.3 Diagram Alir Proses Penyembunyian

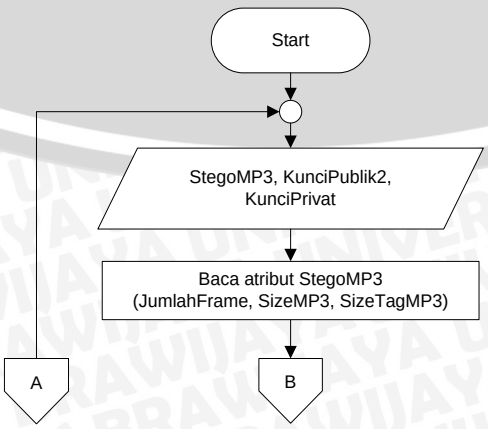
Proses pengungkapan pesan bertujuan untuk mendapatkan kembali pesan yang disembunyikan ke dalam berkas MP3. Pesan diperoleh melalui proses

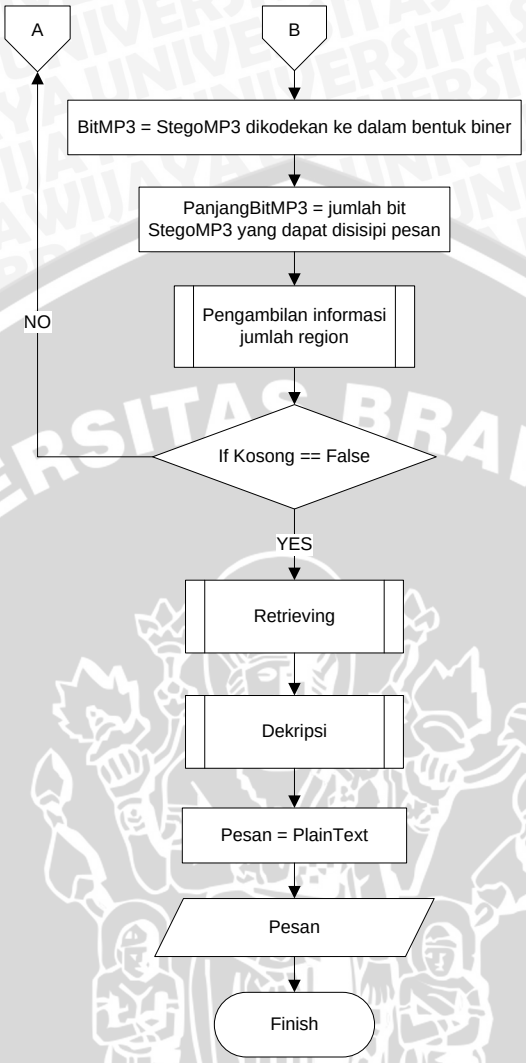
retrieving pada *stego* MP3. Pesan tersebut masih berupa pesan terenkripsi atau *ciphertext*. Selanjutnya *ciphertext* akan diubah menjadi pesan asli atau *plaintext* pada proses dekripsi.

Langkah-langkah untuk proses pengungkapan pesan adalah sebagai berikut :

1. Masukan yang digunakan adalah berkas MP3, kunci publik 2, dan kunci privat.
2. Dilakukan pembacaan atribut pada berkas MP3 seperti jumlah *frame*, ukuran berkas, dan panjang tag *id* MP3.
3. Berkas MP3 dikodekan ke dalam bentuk byte kemudian diubah ke dalam bentuk biner kemudian dihitung panjang bit MP3 yang dapat digunakan untuk menampung pesan.
4. Dilakukan proses pengambilan informasi jumlah region, proses ini bertujuan untuk mengetahui informasi jumlah pembagian region pada bit MP3 yang akan digunakan pada saat proses *retrieving*. Proses ini juga melakukan pengecekan apakah pada berkas MP3 tersebut terdapat suatu pesan rahasia atau tidak. Apabila terdapat pesan rahasia, berkas MP3 disebut sebagai *stego* MP3 dan dilakukan langkah 5.
5. Dilakukan proses *retrieving* untuk mendapatkan pesan dari *stego* MP3. Pesan tersebut masih berupa pesan terenkripsi atau *ciphertext*.
6. Dilakukan proses dekripsi untuk mengubah *ciphertext* menjadi *plaintext* atau pesan asli.
7. Hasil akhir berupa pesan yang sebelumnya disembunyikan pada *stego* MP3.

Diagram alir untuk proses pengungkapan pesan ditunjukkan pada Gambar 3.4.





Gambar 3.4 Diagram Alir Proses Pengungkapan

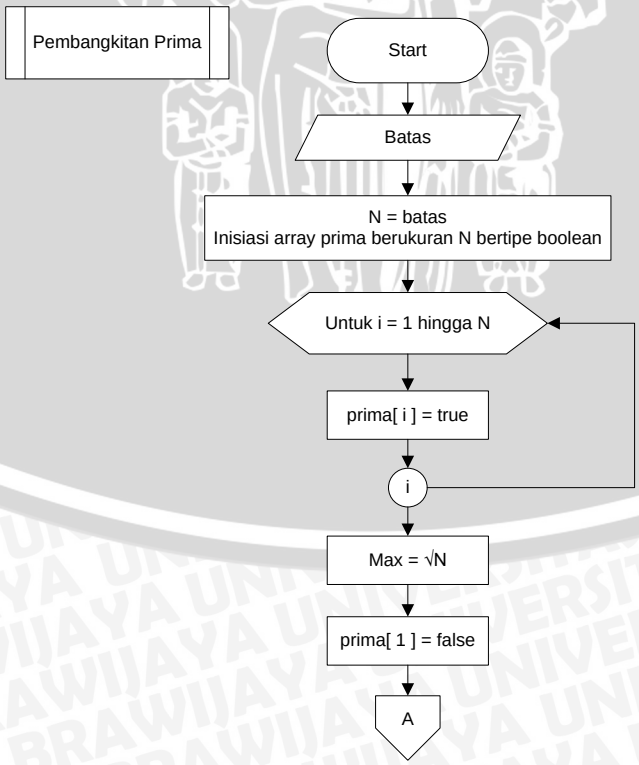
3.2.1 Pembangkitan Bilangan Prima

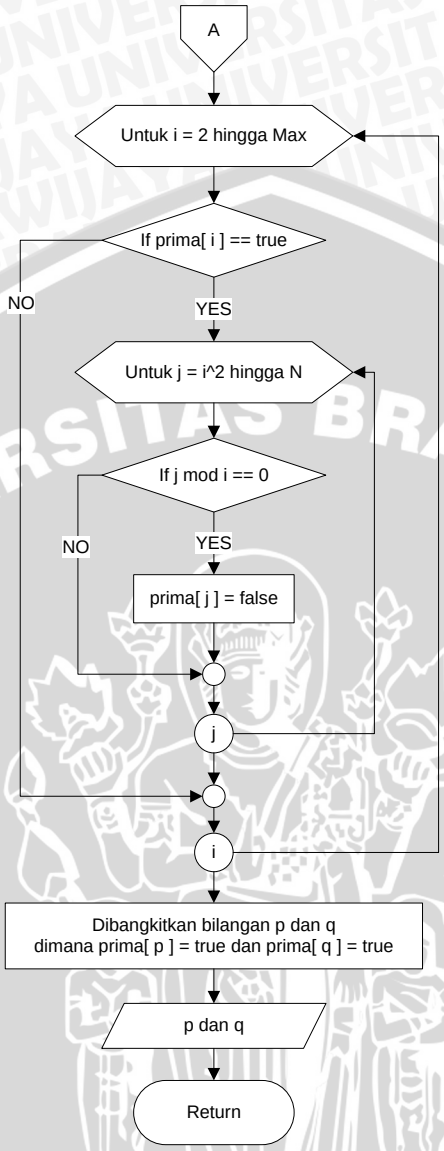
Pembangkitan bilangan prima merupakan proses yang dilakukan untuk membangkitkan bilangan prima *p* dan *q* yang akan digunakan pada proses pembangkitan kunci. Metode yang digunakan untuk pembangkitan bilangan prima adalah *sieve of eratosthenes*. Setelah sejumlah bilangan prima dalam rentang tertentu dibangkitkan, dipilih dua bilangan prima secara acak. Langkah-langkah untuk proses pembangkitan bilangan prima adalah sebagai berikut :

1. Masukan yang digunakan adalah batas atas. Pencarian bilangan prima dilakukan dalam rentang antara 2 hingga batas atas.

2. Diinisiasi sebuah *array* bertipe boolean berukuran *N* dimana *N* adalah batas atas.
3. Semua elemen pada *array* tersebut di set bernilai true, kecuali pada elemen pada indeks ke-1 di set bernilai false.
4. Inisiasi variabel i bernilai 2.
5. Apabila elemen ke-i bernilai true, maka semua elemen yang memiliki indeks kelipatan i mulai dari $i * i$ hingga kelipatan *i* yang bernilai kurang dari atau sama dengan *N* di set bernilai *false*. Apabila elemen ke-i bernilai false, tidak dilakukan apa-apa.
6. Nilai variabel *i* dinaikkan sebanyak satu.
7. Lakukan proses 5 dan 6 hingga *i* kurang dari atau sama dengan akar kuadrat *N*.
8. Bangkitkan secara acak bilangan *p* dan *q* yang memiliki nilai di antara 2 hingga *N* dimana elemen pada indeks ke-*p* dan *q* bernilai *true*.
9. Hasil akhir diperoleh adalah bilangan prima *p* dan *q*.

Diagram alir untuk proses pembangkitan bilangan prima ditunjukkan pada Gambar 3.5.





Gambar 3.5 Diagram Alir Proses Pembangkitan Bilangan Prima

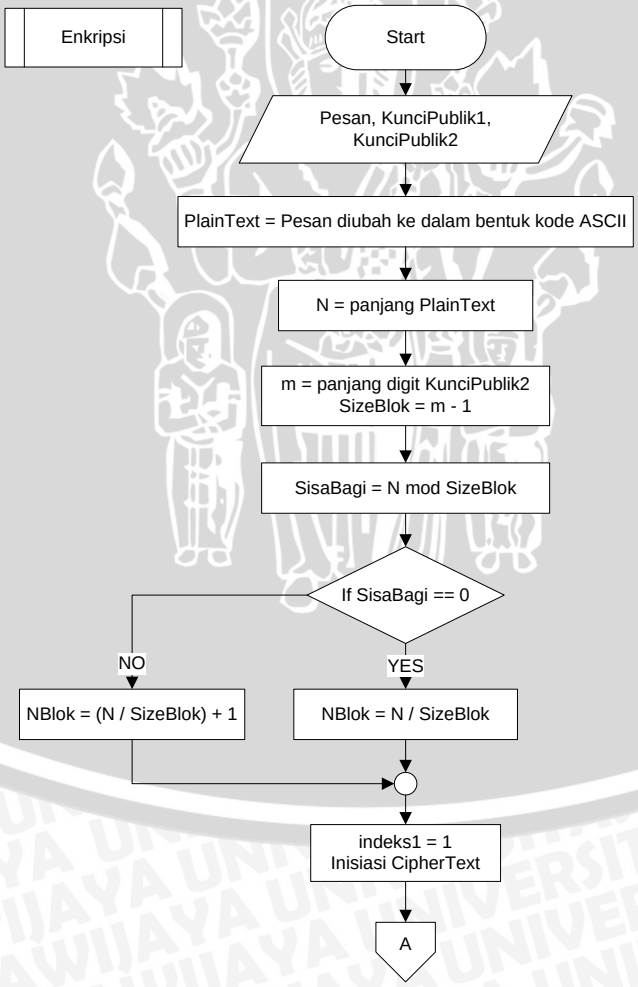
3.2.2 Enkripsi

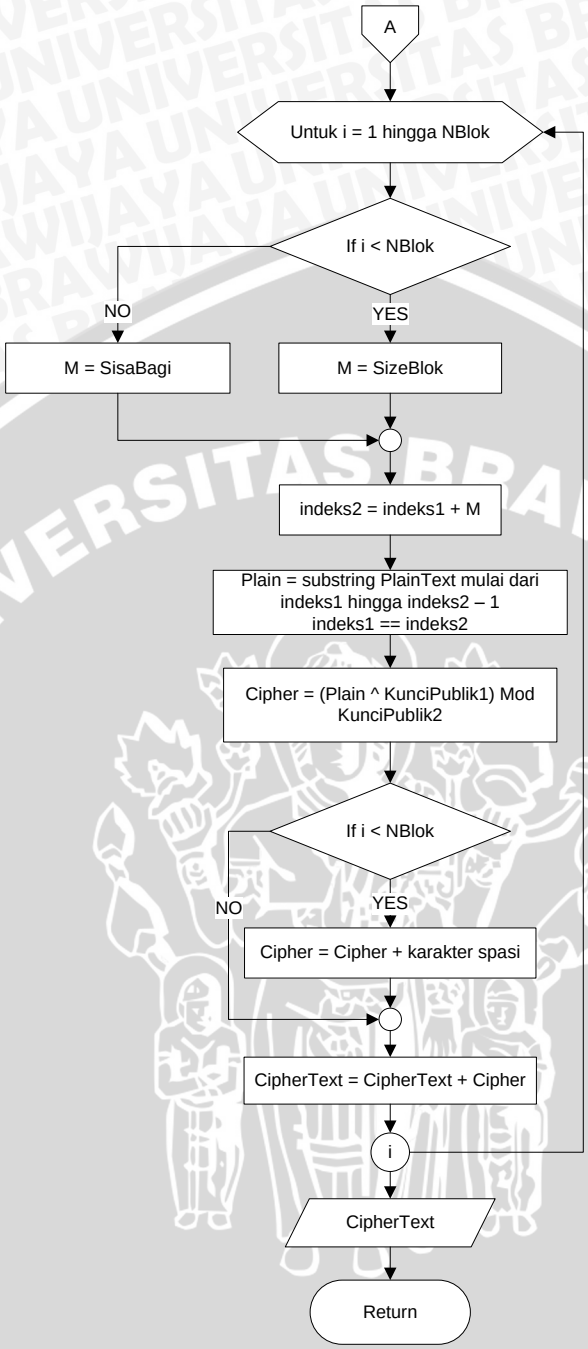
Enkripsi merupakan proses yang bertujuan untuk mengubah pesan asli atau *plaintext* menjadi pesan terenkripsi atau *ciphertext*. Langkah-langkah untuk proses enkripsi adalah sebagai berikut :

1. Masukan yang digunakan adalah pesan yang akan dienkripsi (*plaintext*), kunci publik 1, dan kunci publik 2.
2. *Plaintext* diubah ke dalam kode ASCII.

3. Dihitung ukuran blok (*SizeBlok*), yaitu panjang karakter pada tiap blok. Ukuran blok sama dengan panjang digit kunci publik 2 – 1.
4. Dihitung jumlah blok (*NBlok*) pada *plaintext* dengan membagi panjang *plaintext* dengan ukuran blok.
5. *Plaintext* dibagi menjadi menjadi beberapa blok sejumlah *NBlok*.
6. Pada setiap blok, diterapkan persamaan 2.12.
7. Setiap blok digabung menjadi sebuah *ciphertext*. Antara blok ditambahkan karakter spasi untuk pembatas tiap blok. Karakter spasi berfungsi untuk memudahkan pemisahan tiap blok pada proses dekripsi.
8. Hasil akhir diperoleh *ciphertext* atau pesan yang telah terenkripsi.

Diagram alir untuk proses enkripsi ditunjukkan pada Gambar 3.6.





Gambar 3.6 Diagram Alir Proses Enkripsi

3.2.3 Penyisipan Informasi Jumlah Region

Proses penyisipan informasi jumlah region bertujuan untuk meyisipkan informasi jumlah pembagian region pada bit MP3 *carrier*. Informasi jumlah region digunakan pada proses *retrieving* pada pengungkapan pesan. Informasi tersebut disembunyikan pada byte pertama *main data* pada setiap *frame* pada MP3

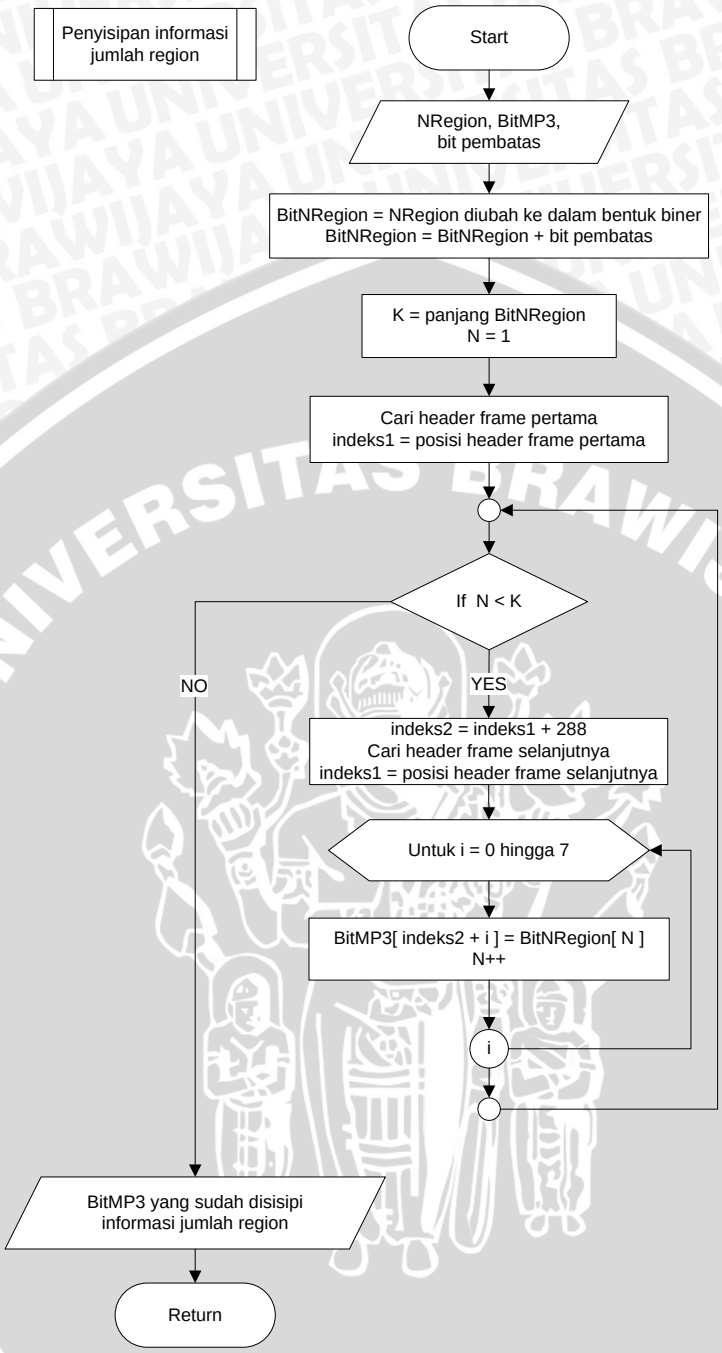
carrier. Pada proses ini, informasi jumlah region diubah ke dalam bentuk biner sehingga dapat disembunyikan ke dalam bit MP3 *carrier*.

Karakter pembatas merupakan sebuah karakter unik yang digunakan sebagai tanda bagian akhir dari set bit informasi jumlah region. Karakter pembatas diubah ke dalam bentuk biner dan ditambahkan pada bagan akhir dari bit informasi jumlah region. Karakter tersebut juga digunakan sebagai penanda bahwa dalam berkas MP3 tersebut terdapa pesan yang disembunyikan.

Langkah-langkah untuk proses penyisipan informasi jumlah region adalah sebagai berikut :

1. Masukan yang digunakan adalah informasi jumlah region, bit MP3 *carrier*, dan karakter pembatas.
2. Informasi jumlah region diubah ke dalam dalam bentuk biner. Pada bagian akhir bit informasi jumlah region, ditambahkan bit karakter pembatas yaitu bentuk biner dari karakter pembatas.
3. Inisiasi variabel $n = 1$ sebagai penanda posisi *frame*.
4. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
5. Langkah 6, 7, dan 8 dilakukan hingga semua bit informasi jumlah region berhasil disembunyikan.
6. Bit pertama pada *main data* pada *frame* ke-*n* berjarak 288 bit dari pointer indeks1 yang berada pada posisi *header frame* ke-*n*. Pointer indeks2 sebagai penanda posisi bit pertama tersebut.
7. Mulai dari posisi indeks2, sebanyak 8 bit diganti dengan bit informasi jumlah region.
8. Nilai variabel *n* dinaikkan sebanyak satu. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
9. Hasil akhir yang didapatkan adalah bit MP3 *carrier* yang telah disisipi informasi jumlah region.

Diagram alir untuk proses penyisipan informasi jumlah region ditunjukkan pada Gambar 3.7.



Gambar 3.7 Diagram Alir Proses Penyisipan Informasi Jumlah Region

3.2.4 *Embedding*

Proses *embedding* bertujuan untuk menyembunyikan pesan terenkripsi (*ciphertext*) ke dalam MP3 *carrier*. Proses *embedding* ini merupakan implementasi dari metode *Parity Coding*. Bit MP3 *carrier* dibagi menjadi sejumlah region sebanyak panjang dari bit pesan. Bit pesan disembunyikan secara

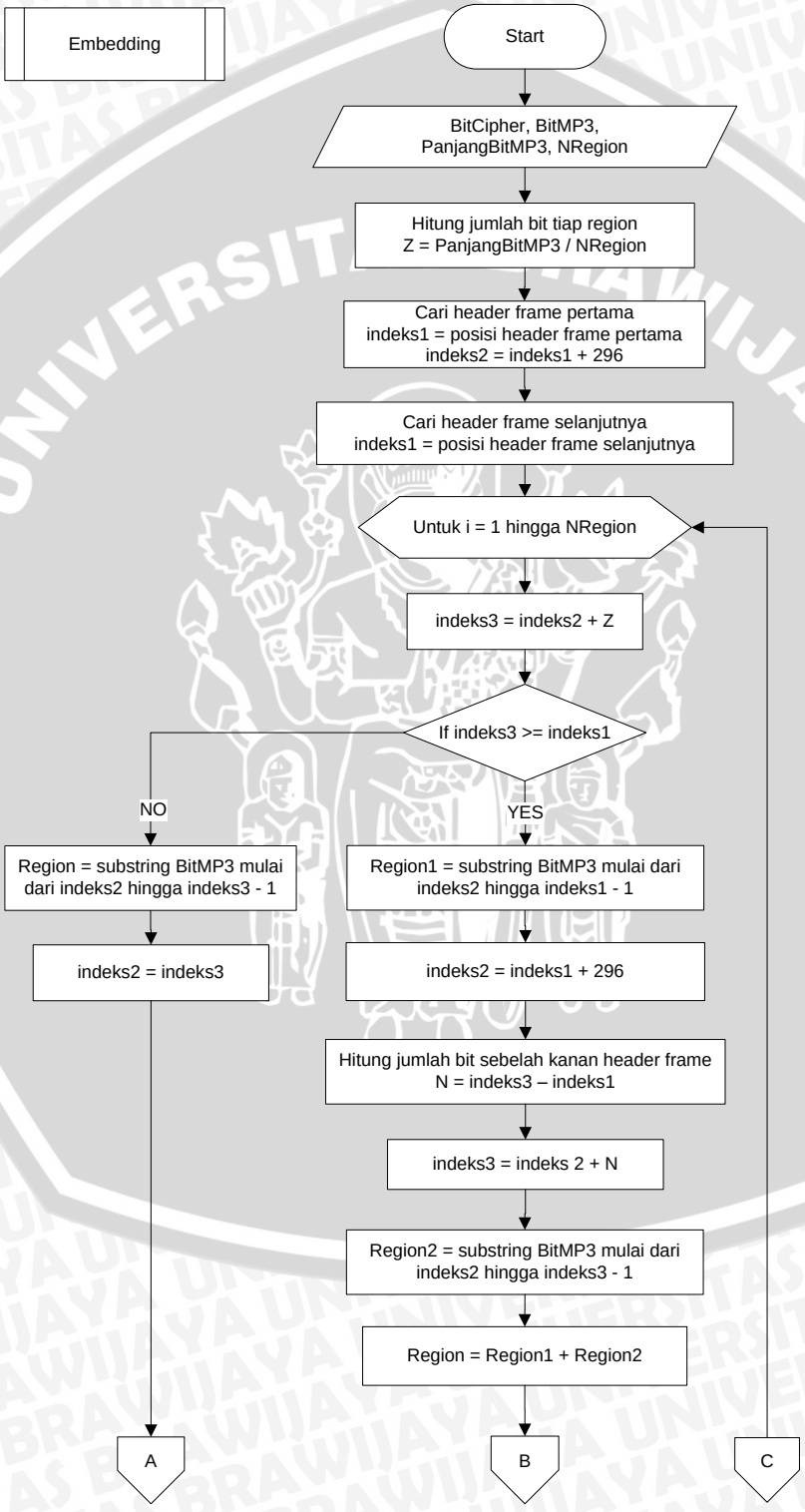
merata pada seluruh region dimana tepat satu bit disembunyikan pada satu region. Selanjutnya, nilai parity bit pada tiap region dihitung. Jika nilai *parity* bit pada region tersebut tidak sama dengan bit dari pesan yang akan disembunyikan, maka LSB pada region tersebut diganti sehingga nilai *parity* bit sama dengan bit dari pesan yang akan disembunyikan.

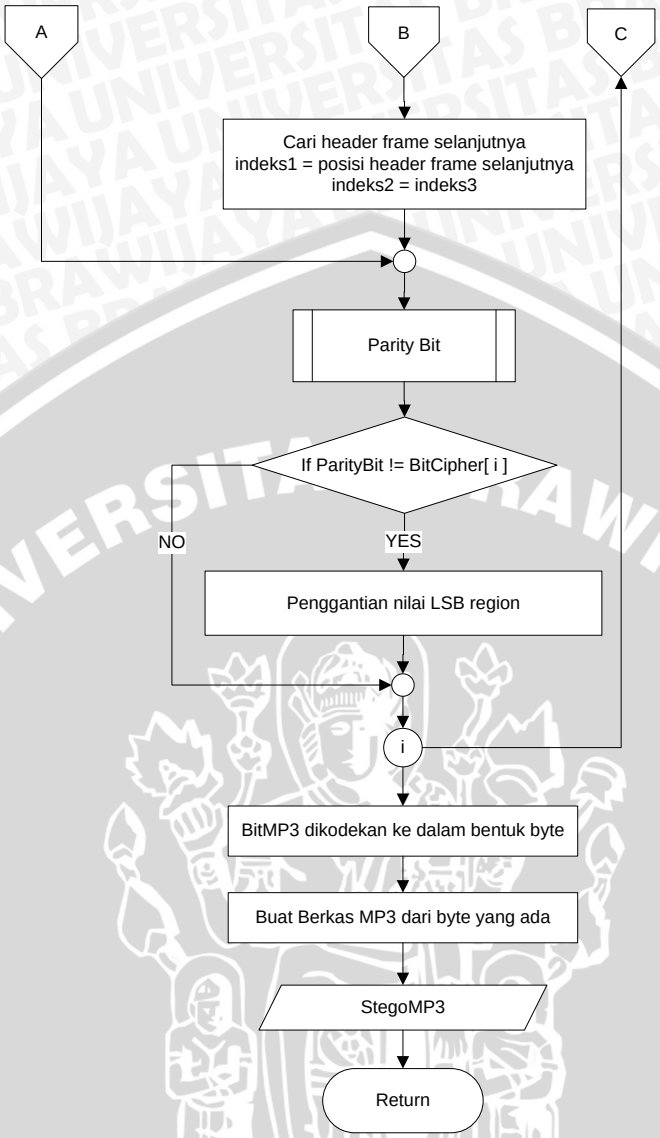
Langkah-langkah untuk proses *embedding* adalah sebagai berikut :

1. Masukan yang digunakan adalah bit MP3 *carrier*, bit *ciphertext*, panjang bit MP3 *carrier* yang dapat disisipi pesan, dan jumlah region.
2. Dihitung jumlah bit pada tiap region (Z).
3. Inisiasi variabel $n = 1$ sebagai penanda posisi *frame* dan variabel $i = 1$ sebagai penanda posisi region.
4. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
5. Bit pertama pada *main data* yang dapat disisipi pesan pada *frame* ke-*n* berjarak 296 bit dari pointer indeks1 yang berada pada posisi *header frame* ke-*n*. Pointer indeks2 sebagai penanda posisi bit pertama tersebut.
6. Nilai variabel *n* dinaikkan sebanyak satu. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
7. Langkah 8, 9, 10, 11, 12, dan 13 dilakukan sebanyak jumlah region.
8. Pointer indeks3 sebagai penanda batas region ke-*i*. Apabila indeks3 tidak melebihi posisi *header frame* ke-*n*, maka dilakukan langkah 9. Apabila sebaliknya, maka dilakukan langkah 10, 11, dan 12.
9. Hitung nilai *parity* bit antara indeks2 dan indeks3. Posisi indeks2 maju ke posisi indeks3.
10. Karena region ke-*i* berada pada *frame* yang berbeda, maka dilakukan perhitungan *parity* bit pada region sebelah kiri header *frame* ke-*n*.
11. Dilakukan perhitungan *parity* bit pada region sebelah kanan header *frame* ke-*n* . Posisi indeks2 maju ke hingga batas region ke-*i*.
12. Nilai variabel *n* dinaikkan sebanyak satu. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
13. Jika bit pesan ke-*i* tidak sama dengan nilai *parity* bit, maka LSB pada region ke-*i* diganti. Nilai variabel *i* dinaikkan sebanyak satu.

14. Bit MP3 *carrier* dijadikan berkas MP3 baru.
15. Hasil akhir diperoleh *stego* MP3, yaitu berkas MP3 yang sudah disisipi pesan.

Diagram alir untuk proses *embedding* ditunjukkan pada Gambar 3.8.





Gambar 3.8 Diagram Alir Proses *Embedding*

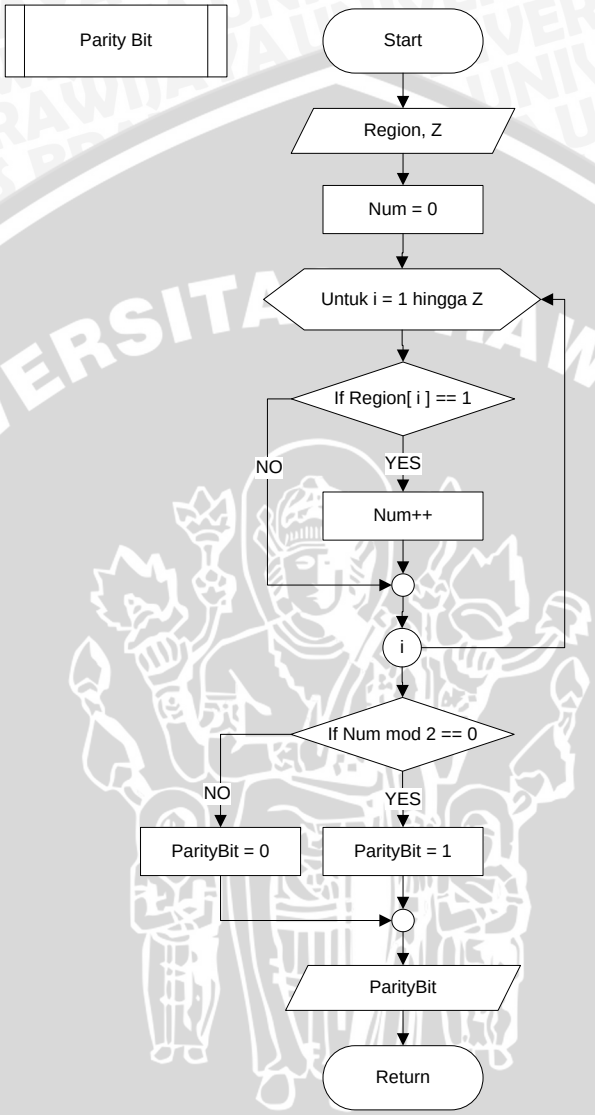
3.2.5 *Parity* Bit

Proses *parity* bit bertujuan untuk mencari nilai *parity* bit dari set bit yang merupakan bit-bit dari sebuah region dari MP3 *carrier*. Langkah-langkah untuk proses *parity* bit adalah sebagai berikut :

1. Masukan yang digunakan adalah set bit pada sebuah region dan jumlah bit dalam satu region.
2. Dihitung jumlah bit 1 dalam set bit tersebut.
3. Apabila jumlah bit 1 ganjil, maka *parity* bit bernilai 1. Apabila jumlah bit 1 genap, maka *parity* bit bernilai 0.

4. Hasil akhir yang didapat adalah nilai *parity* bit dari suatu set bit.

Diagram alir untuk proses *embedding* ditunjukkan pada Gambar 3.9.



Gambar 3.9 Diagram Alir Proses *Parity* Bit

3.2.6 Pengambilan Informasi Jumlah Region

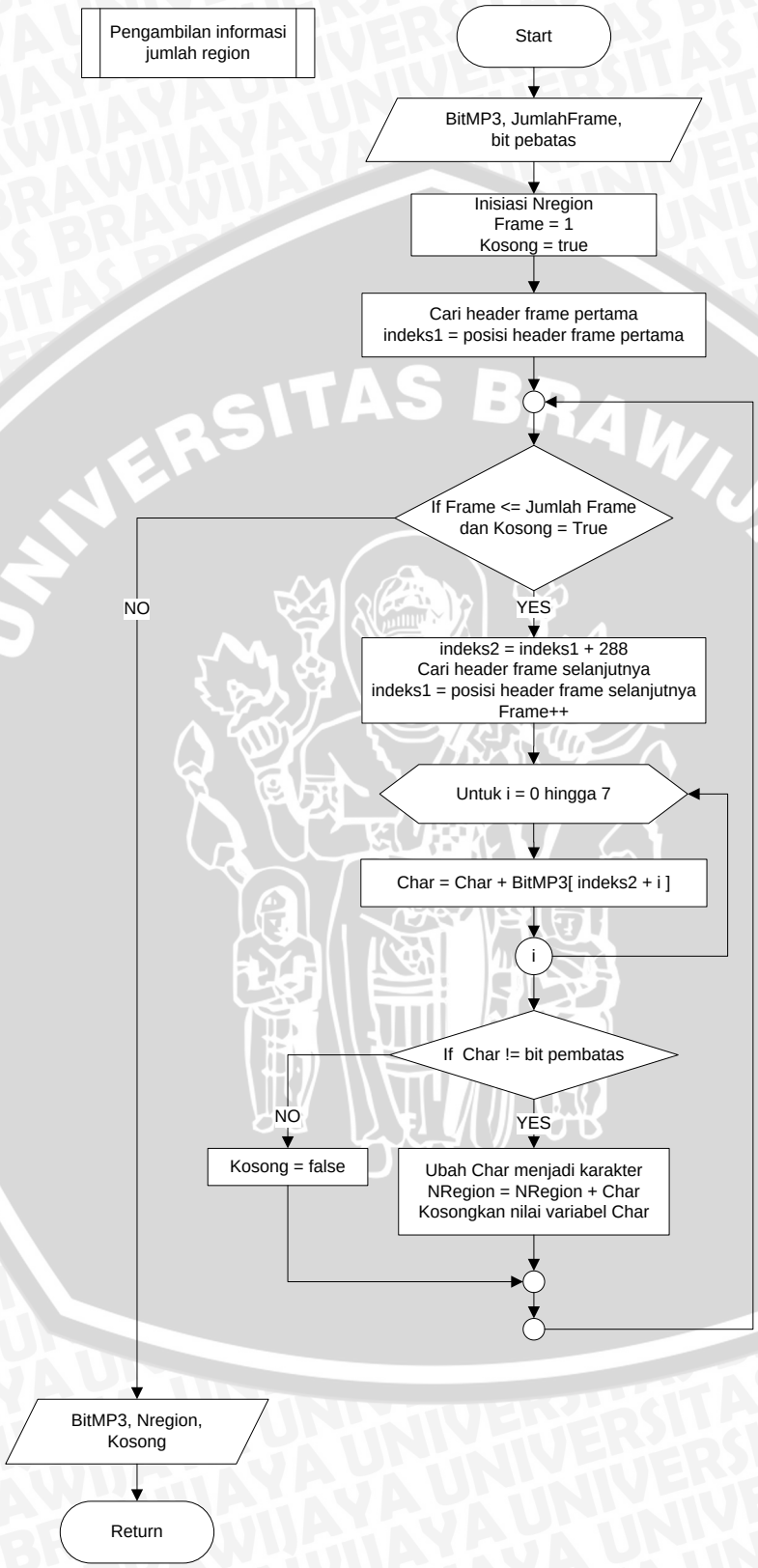
Proses pengambilan informasi jumlah region bertujuan untuk mengetahui berapa jumlah pembagian region pada bit MP3 pada proses *retrieving*. Informasi jumlah region terletak pada byte pertama *main data* pada setiap *frame* berkas MP3. Informasi yang terdapat pada bit MP3 berupa set bit sehingga harus diubah ke bentuk kode ASCII dan selanjutnya ke bentuk karakter.

Pencarian set bit tersebut dilakukan pada byte pertama *main data* pada setiap *frame*. Pencarian tersebut berhenti apabila ditemukan set bit yang merupakan bentuk biner dari bit pembatas. Bit pembatas tersebut selain mendadakan bagian akhir dari set bit informasi jumlah region juga menandakan bahwa di dalam bit MP3 tersebut terdapat pesan yang disembunyikan sehingga bit MP3 tersebut dapat disebut *stego* MP3. Pencarian dilakukan hingga byte pertama *main data* pada *frame* terakhir hingga ditemukan bit pembatas pembatas. Apabila pada *frame* terakhir tidak ditemukan bit pembatas, maka pencarian dihentikan dan bit MP3 diasumsikan tidak mengandung pesan yang disembunyikan.

Langkah-langkah untuk proses pengambilan informasi jumlah region adalah sebagai berikut :

1. Masukan yang digunakan adalah bit MP3, jumlah *frame*, dan bit pembatas.
2. Inisiasi variabel $n = 1$ sebagai penanda posisi *frame*.
3. Cari letak header *frame* ke-n. Pointer indeks1 sebagai penanda posisi header *frame* ke-n.
4. Bit pertama pada *main data* pada *frame* ke-*n* berjarak 288 bit dari pointer indeks1 yang berada pada posisi *header frame* ke-*n*. Pointer indeks2 sebagai penanda posisi bit pertama tersebut.
5. Mulai dari posisi indeks2, diambil sebanyak 8.
6. Nilai variabel *n* dinaikkan sebanyak satu. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
7. Jika 8 bit tersebut bukan bit pembatas, maka 8 bit dan diubah ke dalam kode ASCII. Kode ASCII tersebut diubah ke bentuk karakter, selanjutnya karakter tersebut ditampung.
8. Langkah 4, 5, 6, dan 7 dilakukan hingga *frame* terakhir atau hingga ditemukannya bit pembatas pada bit MP3.
9. Jika hingga *frame* terakhir tidak ditemukan bit pembatas, maka pada bit MP3 tersebut tidak terdapat pesan yang disembunyikan.
10. Hasil akhir didapatkan informasi jumlah region dan informasi mengenai ada atau tidaknya pesan yang disembunyikan pada bit MP3 tersebut.

Diagram alir untuk proses pengambilan informasi jumlah region ditunjukkan pada Gambar 3.10.



Gambar 3.10 Diagram Alir Proses Pengambilan Informasi Jumlah Region

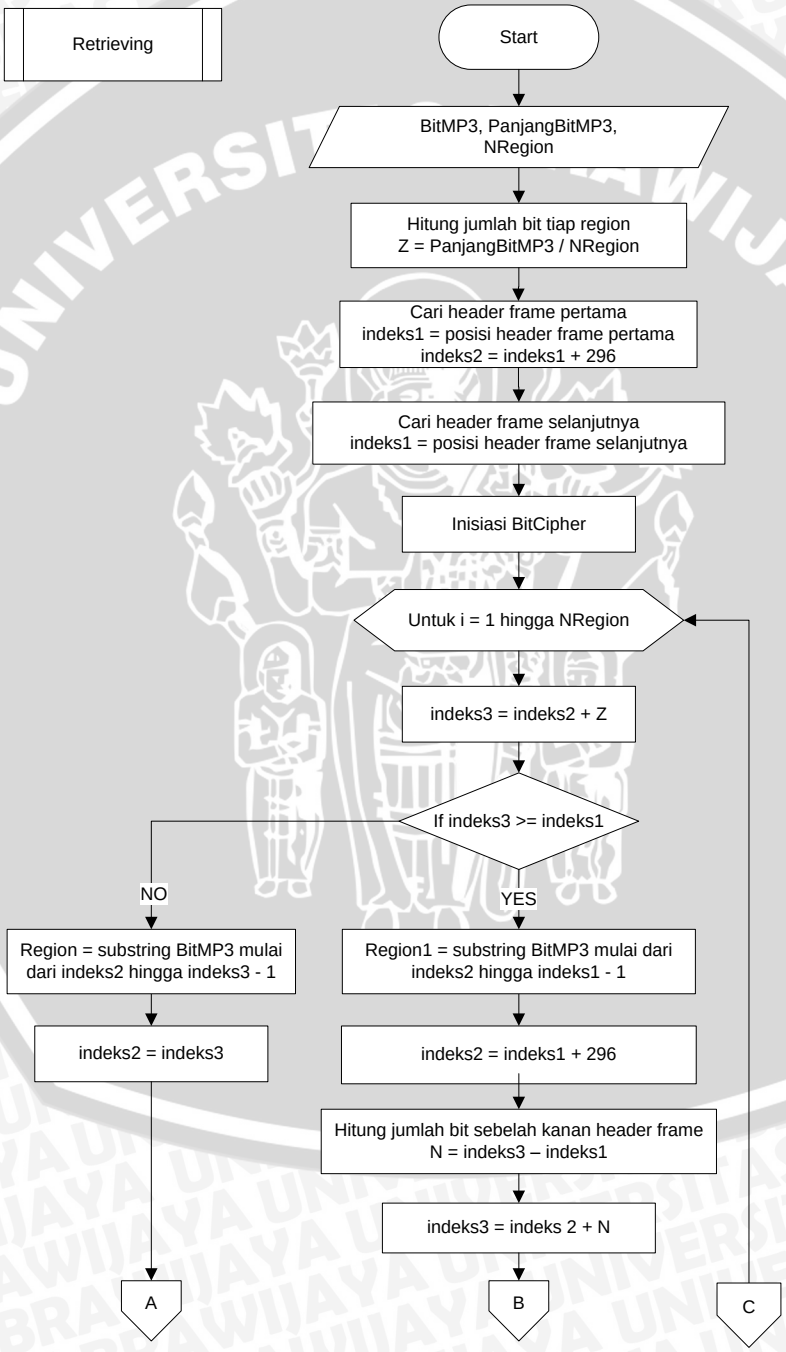
3.2.7 *Retrieving*

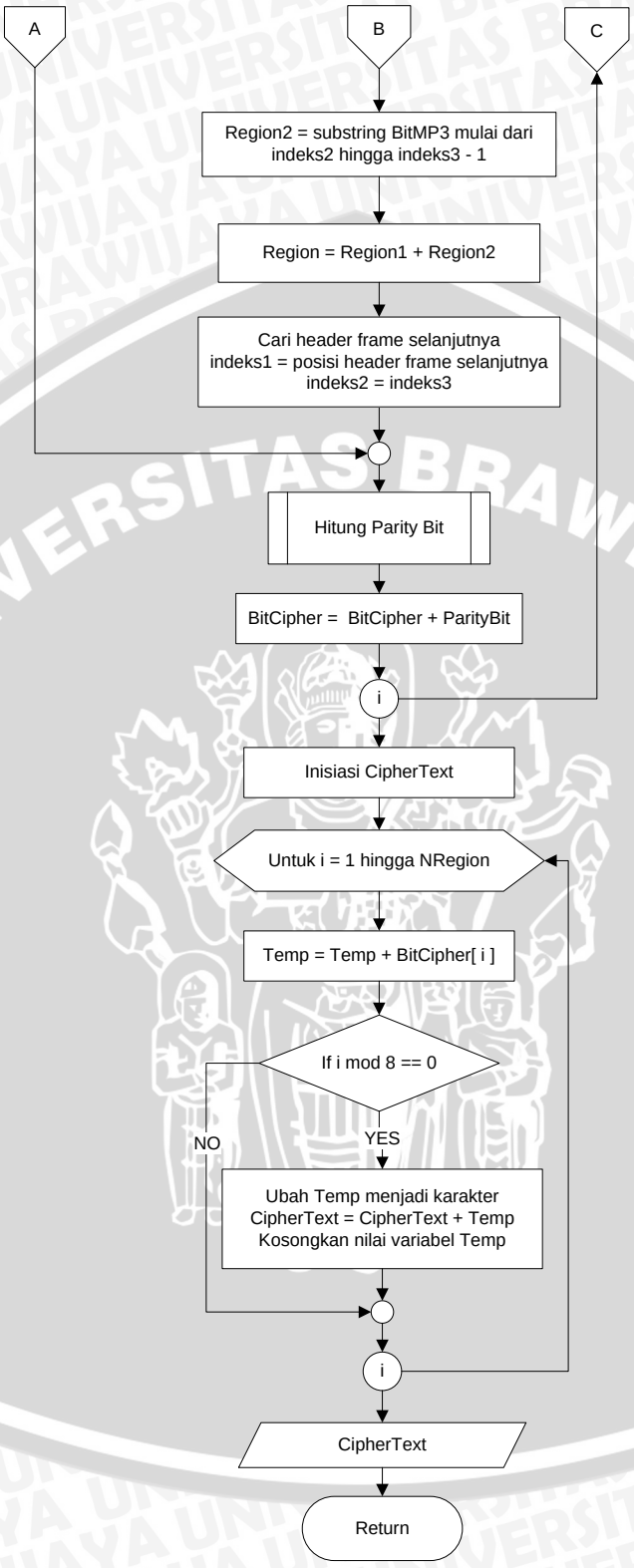
Proses *retrieving* bertujuan untuk mendapatkan kembali pesan yang disembunyikan di dalam *stego* MP3. Pada proses ini dibutuhkan informasi jumlah region yang diperoleh dari proses sebelumnya untuk membagi bit *stego* MP3 menjadi sejumlah region. Bit pesan diperoleh dari perhitungan nilai *parity* bit dari seluruh region. Langkah-langkah untuk proses *retrieving* adalah sebagai berikut :

1. Masukan yang digunakan adalah bit *stego* MP3, panjang bit *stego* MP3 yang dapat disisipi pesan, dan jumlah region.
2. Dihitung jumlah bit pada tiap region (Z).
3. Inisiasi variabel $n = 1$ sebagai penanda posisi *frame* dan variabel $i = 1$ sebagai penanda posisi region.
4. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
5. Bit pertama pada *main data* yang dapat disisipi pesan pada *frame* ke-*n* berjarak 296 bit dari pointer indeks1 yang berada pada posisi *header frame* ke-*n*. Pointer indeks2 sebagai penanda posisi bit pertama tersebut.
6. Nilai variabel *n* dinaikkan sebanyak satu. Cari letak *header frame* ke-*n*. Pointer indeks1 sebagai penanda posisi *header frame* ke-*n*.
7. Langkah 8, 9, 10, 11, 12, dan 13 dilakukan sebanyak jumlah region.
8. Pointer indeks3 sebagai penanda batas region ke-*i*. Apabila indeks3 tidak melebihi posisi *header frame* ke-*n*, maka dilakukan langkah 9. Apabila sebaliknya, maka dilakukan langkah 10, 11, dan 12.
9. Hitung nilai parity bit antara indeks2 dan indeks3. Posisi indeks2 maju ke posisi indeks3.
10. Karena region ke-*i* berada pada *frame* yang berbeda, maka dilakukan perhitungan parity bit pada region sebelah kiri header *frame* ke-*n*.
11. Dilakukan perhitungan parity bit pada region sebelah kanan header *frame* ke-*n* . Posisi indeks2 maju ke hingga batas region ke-*i*.
12. Nilai variabel *n* dinaikkan sebanyak satu. Cari letak header *frame* ke-*n*. Pointer indeks1 sebagai penanda posisi header *frame* ke-*n*.
13. Nilai *parity* bit yang sudah dihitung ditampung.

14. Kumpulan nilai *pariy* bit yang merupakan bit pesan tersebut diubah ke dalam kode ASCII dan kemudian ke bentuk karakter.
15. Hasil akhir yang diperoleh berupa pesan.

Diagram alir untuk proses pengambilan informasi jumlah region ditunjukkan pada Gambar 3.11.





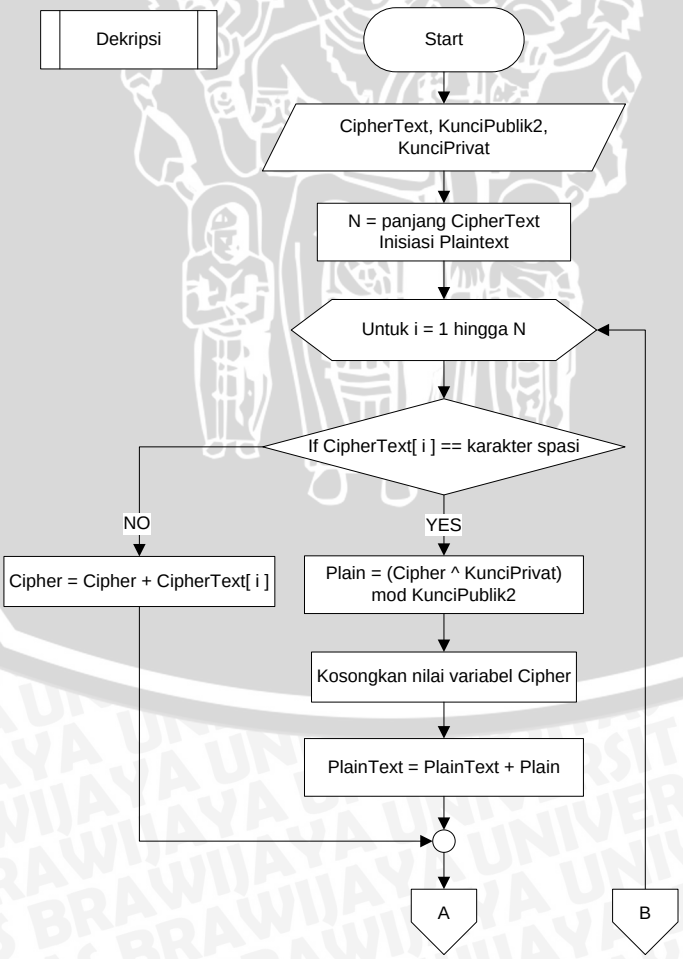
Gambar 3.11 Diagram Alir Proses *Retrieving*

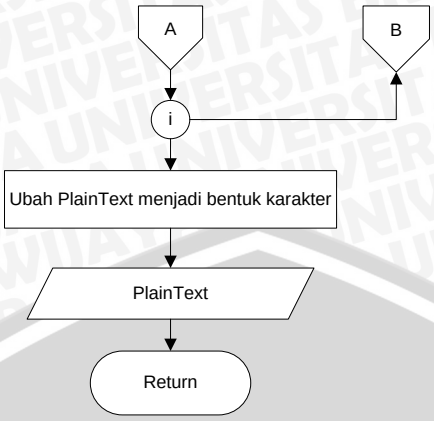
3.2.8 Dekripsi

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

1. Masukan yang digunakan adalah pesan terenkripsi (*ciphertext*), kunci privat, dan kunci publik 2.
2. *Ciphertext* dipisah menjadi beberapa blok dengan karakter spasi sebagai pemisah.
3. Pada setiap blok, diterapkan persamaan 2.13.
4. Setiap blok digabung menjadi sebuah *plaintext*.
5. *Plaintext* diubah menjadi menjadi karakter.
6. Hasil akhir diperoleh pesan asli atau *plaintext*.

Diagram alir untuk proses enkripsi ditunjukkan pada Gambar 3.12.





Gambar 3.12 Diagram Alir Proses Dekripsi

3.3 Perhitungan Manual

Pada subbab ini akan dijelaskan mengenai contoh perhitungan manual pada penyembunyian pesan terenkripsi pada berkas MP3 menggunakan metode *parity coding*. Perhitungan manual yang dibahas adalah pembangkitan kunci untuk enkripsi dan dekripsi, penyembunyian pesan ke dalam berkas MP3, dan pengungkapan kembali pesan yang telah disembunyikan. Pesan yang digunakan pada perhitungan manual ini adalah sebuah berkas teks yang berisi kata "CS".

3.3.1 Pembentukan Kunci

Proses pembangkitan kunci dilakukan untuk mencari kunci publik untuk enkripsi pada proses penyembunyian dan kunci privat untuk dekripsi pada proses pengungkapan pesan. Berikut ini adalah langkah-langkah pembentukan kunci pada algoritma RSA.

a. Dilakukan pembangkitan bilangan prima p dan q menggunakan metode *sieve of eratosthenes*, misalnya didapatkan $p = 17$ dan $q = 23$.

b. Dihitung nilai n yang merupakan kunci publik pertama menggunakan persamaan 2.10.

$$n = pq$$
$$= 17 * 23$$
$$= 391$$

c. Dihitung $\varphi(n)$ menggunakan persamaan 2.4.

$$\varphi(n) = (p-1)(q-1)$$

$\varphi(391) = (17-1) * (23-1)$

$= 352$

d. Dipilih bilangan bulat positif b yang merupakan kunci publik kedua dimana $1 < b < \varphi(n)$ dan prima relatif terhadap (n) atau dengan kata lain $FPB(b, \varphi(n)) = 1$. Maka, dipilih $b = 29$ karena $FPB(29, 352) = 1$. Berikut ini dilakukan pembuktian bahwa $FPB(29, 352) = 1$ menggunakan algoritma euclide menggunakan persamaan 2.6 dan 2.7. Jika diketahui $r_0 = 352$ dan $r_1 = 29$, maka :

$r_0 = q_1 r_{1+} r_2$

$352 = 12 * 29 + 4$

$29 = 7 * 4 + \mathbf{1}$

$4 = 4 * 1$

e. Dihitung bilangan bulat positif a yang merupakan kunci privat menggunakan persamaan 2.11.

$a = b^{-1} \, mod \, \varphi(n)$

$= 29^{-1} \, mod \, 352$

Untuk mempermudah perhitungan, maka akan digunakan algoritma euclide diperluas. Langkah pertama adalah menerapkan algoritma euclide menggunakan persamaan 2.6 dan 2.7 untuk mencari q_0 hingga q_n.

$352 = 12 * 29 + 4 \quad n = 1, \quad q_1 = 12$

$29 = 7 * 4 + 1 \quad n = 2, \quad q_2 = 7$

$4 = 4 * 1 \quad n = 3, \quad q_3 = 4$

Langkah kedua adalah menghitung $29^{-1} \, mod \, 352$ menggunakan persamaan 2.8.

$t_1 = 0$

$t_2 = 1$

$t_3 = t_1 - q_1 t_2 \, mod \, r_0$

$= 0 - 12 * 1 \, mod \, 352$

$= -12 \, mod \, 352$

$= 340$

$t_4 = t_2 - q_2 t_3 \, mod \, r_0$

$= 1 - 7 * 340 \ mod \ 352$

$= -2379 \ mod \ 352$

$= 85$

Didapatkan pasangan kunci publik $(n, b) = (391, 29)$ dan kunci rahasia $a = 85$.

3.3.2 Penyembunyian Pesan

Proses penyembunyian pesan meliputi enkripsi *plaintext* menjadi sebuah *ciphertext* dan proses *embedding* ke dalam berkas MP3 (MP3 *carrier*).

3.3.2.1 Enkripsi

Untuk melakukan enkripsi, *plaintext* diubah ke dalam bentuk kode ASCII.

Karakter	ASCII
C	67
S	83

Didapatkan representasi desimal dari *plaintext* "CS" yaitu 6783. Selanjutnya *plaintext* dipecah menjadi suatu blok berukuran $n - 1$, karena suatu *plaintext* m haruslah bernilai $0 \leq m \leq n - 1$. Dipilih ukuran blok sama dengan dua digit, sehingga *plaintext* dipecah menjadi :

m_1	67
m_2	83

Dilakukan enkripsi menggunakan persamaan 2.12.

$c_1 = {m_1}^b \ mod \ n = 67^{29} \ mod \ 391$

$c_2 = {m_2}^b \ mod \ n = 83^{29} \ mod \ 391$

Untuk mempermudah perhitungan, digunakan metode *fast exponentiation*.

- Dihitung $67^{29} \ mod \ 391$

  29 mempunyai bentuk biner 00011101 sehingga ekspansi biner dari 67^{29} adalah :

$67^{29} = 67^{2^0} * 67^{2^2} * 67^{2^3} * 67^{2^4}$

karena :

$29 = 1 * 2^0 + 1 * 2^2 + 1 * 2^3 + 1 * 2^4$

Oleh karena itu, $67^{29}\ mod\ 391$ dapat diubah menjadi :

$67^{29}\ mod\ 391 = (67^{2^0} mod\ 391 * 67^{2^2} mod\ 391 * 67^{2^3} mod\ 391$
$* 67^{2^4} mod\ 391)\ mod\ 391$

Kemudian dilakukan perhitungan untuk mencari nilai $67^{2^0} mod\ 391$, $67^{2^2} mod\ 391$, $67^{2^3} mod\ 391$, dan $67^{2^4} mod\ 391$:

$(67)^1\ mod\ 391 = 67\ mod\ 391 = 67$

$(67)^2\ mod\ 391 = 4489\ mod\ 391$
$= 188$

$(67)^4\ mod\ 391 = (67^2)^2\ mod\ 391$
$= (188)^2\ mod\ 391$
$= 35344\ mod\ 391$
$= 154$

$(67)^8\ mod\ 391 = (67^4)^2\ mod\ 391$
$= (154)^2\ mod\ 391$
$= 23716\ mod\ 391$
$= 256$

$(67)^{16}\ mod\ 391 = (67^8)^2\ mod\ 391$
$= (256)^2\ mod\ 391$
$= 65536\ mod\ 391$
$= 239$

sehingga :

$67^{29}\ mod\ 391 = (67^{2^0} mod\ 391 * 67^{2^2} mod\ 391 * 67^{2^3} mod\ 391$
$* 67^{2^4} mod\ 391)\ mod\ 391$
$= (67 * 154 * 256 * 239)\ mod\ 391$
$= 631296512\ \ mod\ 391$
$= 33$

- Dihitung $83^{29}\ mod\ 391$

29 mempunyai bentuk biner 00011101 sehingga ekspansi biner dari 83^{29} adalah :

$83^{29} = 83^{2^0} * 83^{2^2} * 83^{2^3} * 83^{2^4}$

karena :

$29 = 1 * 2^0 + 1 * 2^2 + 1 * 2^3 + 1 * 2^4$

Oleh karena itu, $83^{29}\ mod\ 391$ dapat diubah menjadi :

$$83^{29}\ mod\ 391 = (83^{2^0} mod\ 391 * 83^{2^2} mod\ 391 * 83^{2^3} mod\ 391 * 83^{2^4} mod\ 391)\ mod\ 391$$

Kemudian dilakukan perhitungan untuk mencari nilai $83^{2^0} mod\ 391$, $83^{2^2} mod\ 391$, $83^{2^3} mod\ 391$, dan $83^{2^4} mod\ 391$:

$(83)^1\ mod\ 391 = 83\ mod\ 391 = 83$

$(83)^2\ mod\ 391 = 6889\ mod\ 391$

$= 242$

$(83)^4\ mod\ 391 = (83^2)^2\ mod\ 391$

$= (242)^2\ mod\ 391$

$= 58564\ mod\ 391$

$= 305$

$(83)^8\ mod\ 391 = (83^4)^2\ mod\ 391$

$= (305)^2\ mod\ 391$

$= 93025\ mod\ 391$

$= 358$

$(83)^{16}\ mod\ 391 = (67^8)^2\ mod\ 391$

$= (358)^2\ mod\ 391$

$= 128164\ mod\ 391$

$= 307$

sehingga :

$$83^{29}\ mod\ 391 = (83^{2^0} mod\ 391 * 83^{2^2} mod\ 391 * 83^{2^3} mod\ 391 * 83^{2^4} mod\ 391)\ mod\ 391$$

$$= (83 * 305 * 358 * 307)\ mod\ 391$$

$$= 2782270390\ \ mod\ 391$$

= 19

Dari perhitungan diatas, dihasilkan *ciphertext* "33 19".

3.3.2.2 *Embedding*

Pada proses *embedding*, *ciphertext* dari proses enkripsi disembunyikan ke dalam MP3 *carrier*. *Ciphertext* diubah ke dalam bentuk biner seperti pada tabel 3.1.

Tabel 3.1 Representasi biner dari *ciphertext*

Karakter	ASCII	Biner
3	51	00110011
3	51	00110011
Spasi	32	00100000
1	49	00110001
9	57	00111001

Dari tabel 3.1 dapat diketahui ukuran pesan yang akan disembunyikan dalam kasus ini adalah *ciphertext* adalah 40 bit.

Selanjutnya, MP3 *carrier* berukuran 320 bit dikodekan ke dalam bentuk biner. Berkas MP3 *carrier* dibagi menjadi 40 region sesuai dengan ukuran pesan dimana setiap region mempunyai ukuran yang statis. Dengan ukuran MP3 *carrier* sebanyak 320 bit, maka setiap region berukuran 8 bit. Berikut ini adalah representasi biner dari MP3 *carrier* yang telah dibagi menjadi 40 region :

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	01011011	11000100	00100001

01010110 11000111 00111101 01101100 10011010

Proses *embedding* menggunakan metode *Parity Coding* dimana setiap satu bit dari pesan disembunyikan ke dalam tepat satu region MP3 *carrier*. Dalam perhitungan manual ini menggunakan kondisi *even parity*. Tabel 3.2 menunjukkan contoh perhitungan *parity* bit pada region ke-1 dan ke-2.

Tabel 3.2 Perhitungan *parity* bit pada region ke-1 dan ke-2

Region	Bit Awal	Perhitungan *Parity* Bit
1	10001101	$1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$
2	00001000	$0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 1$

Tabel 3.3 menunjukkan penyisipan semua bit pesan ke setiap region MP3 *carrier*.

Tabel 3.3 Penyisipan bit pesan ke MP3 *carrier*

Region	Bit MP3 *carrier*	*Parity* Bit	Bit Pesan	Bit *stego* MP3	Berbeda
1	10001101	0	0	10001101	Tidak
2	00001000	1	0	00001001	Ya
3	01000111	0	1	01000110	Ya
4	01100101	0	1	01100100	Ya
5	01010000	0	0	01010000	Tidak
6	10010011	0	0	10010011	Tidak
7	10011010	0	1	10011011	Ya
8	01111011	0	1	01111010	Ya
9	01110010	0	0	01110010	Tidak
10	01110111	0	0	01110111	Tidak
11	11101010	1	1	11101010	Tidak
12	00001010	0	1	00001011	Ya
13	11111010	0	0	11111010	Tidak
14	00111101	1	0	00111100	Ya
15	11100110	1	1	11100110	Tidak

16	01011110	0	1	01011111	Ya
17	01011001	0	0	01011001	Tidak
18	01100101	0	0	01100101	Tidak
19	10000001	0	1	10000000	Ya
20	01010110	0	0	01010110	Tidak
21	01101101	1	0	01101100	Ya
22	01100001	1	0	01100000	Ya
23	11101101	0	0	11101101	Tidak
24	11000001	1	0	11000000	Ya
25	10101111	0	0	10101111	Tidak
26	00101000	0	0	00101000	Tidak
27	01101100	0	1	01101101	Ya
28	11110100	1	1	11110100	Tidak
29	11000100	1	0	11000101	Ya
30	00100001	0	0	00100001	Tidak
31	01111101	0	0	01111101	Tidak
32	11010011	1	1	11010011	Tidak
33	01011011	1	0	01011010	Ya
34	11000100	1	0	11000101	Ya
35	00100001	0	1	00100000	Ya
36	01010110	0	1	01010111	Ya
37	11000111	1	1	11000111	Tidak
38	00111101	1	0	00111100	Ya
39	01101100	0	0	01101100	Tidak
40	10011010	0	1	10011011	Ya

3.3.3 Pengungkapan Pesan

3.3.3.1 *Retrieving*

Pada proses *retrieving*, dilakukan pengungkapan pesan dari MP3 sudah disisipi pesan atau *stego* MP3. *Stego* MP3 dikodekan ke dalam bentuk biner kemudian dibagi menjadi 40 region sesuai dengan ukuran pesan yang

disembunyikan. Selanjutnya dihitung nilai *parity bit* pada tiap region. Tabel 3.4 menunjukkan pengungkapan pesan dari *stego* MP3.

Tabel 3.4 Pengungkapan bit pesan dari *stego* MP3

Region	Representasi Biner	*Parity* Bit
1	10001101	0
2	00001001	0
3	01000110	1
4	01100100	1
5	01010000	0
6	10010011	0
7	10011011	1
8	01111010	1
9	01110010	0
10	01110111	0
11	11101010	1
12	00001011	1
13	11111010	0
14	00111100	0
15	11100110	1
16	01011111	1
17	01011001	0
18	01100101	0
19	10000000	1
20	01010110	0
21	01101100	0
22	01100000	0
23	11101101	0
24	11000000	0
25	10101111	0
26	00101000	0

27	01101101	1
28	11110100	1
29	11000101	0
30	00100001	0
31	01111101	0
32	11010011	1
33	01011010	0
34	11000101	0
35	00100000	1
36	01010111	1
37	11000111	1
38	00111100	0
39	01101100	0
40	10011011	1

Nilai *parity* bit dari setiap region dikumpulkan.

00110011	00110011	00100000	00110001	00111001

Kumpulan *parity bit* tersebut merupakan bentuk biner dari pesan yang disembunyikan. Selanjutnya pesan tersebut diubah kedalam bentuk karakter seperti pada tabel 3.5.

Tabel 3.5 Pengubahan pesan ke bentuk karakter

Biner	ASCII	Karakter
00110011	51	3
00110011	51	3
00100000	32	Spasi
00110001	49	1
00111001	57	9

Dari tabel 3.5 dapat diketahui bahwa pesan berupa karakter "33 19" dimana pada kasus ini pesan tersebut merupakan *ciphertext* yang akan didekripsi pada proses dekripsi.

3.3.3.2 Dekripsi

Untuk melakukan dekripsi, *ciphertext* "33 19" dipisahkan tiap blok berdasarkan karakter spasi sehingga didapatkan :

c_1	33
c_2	19

Dilakukan dekripsi menggunakan persamaan 2.13

$m_1 = {c_1}^a \ mod\ n = 33^{85}\ mod\ 391$

$m_2 = {c_2}^a \ mod\ n = 19^{85}\ mod\ 391$

Untuk mempermudah perhitungan, digunakan metode *fast exponentiation*.

- Dihitung $33^{85}\ mod\ 391$

  85 mempunyai bentuk biner 01010101 sehingga ekspansi biner dari 33^{85} adalah :

  $33^{85} = 33^{2^0} * 33^{2^2} * 33^{2^4} * 33^{2^6}$

  karena :

  $85 = 1 * 2^0 + 1 * 2^2 + 1 * 2^4 + 1 * 2^6$

  Oleh karena itu, $33^{85}\ mod\ 391$ dapat diubah menjadi :

  $33^{85}\ mod\ 391 = (33^{2^0} mod\ 391 * 33^{2^2} mod\ 391 * 33^{2^4} mod\ 391 * 33^{2^6} mod\ 391)\ mod\ 391$

  Kemudian dilakukan perhitungan untuk mencari nilai $33^{2^0} mod\ 391$, $33^{2^2} mod\ 391$, $33^{2^4} mod\ 391$, dan $33^{2^6} mod\ 391$:

  $(33)^1\ mod\ 391 = 33\ mod\ 391 = 33$

  $(33)^2\ mod\ 391 = 1089\ mod\ 391$

  $= 307$

  $(33)^4\ mod\ 391 = (33^2)^2\ mod\ 391$

  $= (307)^2\ mod\ 391$

$= 94249 \ mod \ 391$

$= 18$

$(33)^8 \ mod \ 391 = (33^4)^2 \ mod \ 391$

$= \ (18)^2 \ mod \ 391$

$= 324 \ mod \ 391$

$= 324$

$(33)^{16} \ mod \ 391 = (33^8)^2 \ mod \ 391$

$= \ (324)^2 \ mod \ 391$

$= 104976 \ mod \ 391$

$= 188$

$(33)^{32} \ mod \ 391 = (33^{16})^2 \ mod \ 391$

$= \ (188)^2 \ mod \ 391$

$= 35344 \ mod \ 391$

$= 154$

$(33)^{64} \ mod \ 391 = (33^{32})^2 \ mod \ 391$

$= \ (154)^2 \ mod \ 391$

$= 23716 \ mod \ 391$

$= 256$

sehingga :

$33^{85} \ mod \ 391 = (33^{2^0} mod \ 391 * 33^{2^2} mod \ 391 * 33^{2^4} mod \ 391$

$* 33^{2^6} mod \ 391) \ mod \ 391$

$= (33 * 18 * 188 * 256) \ mod \ 391$

$= 28588032 \ \ mod \ 391$

$= 67$

- Dihitung $19^{85} \ mod \ 391$

  85 mempunyai bentuk biner 01010101 sehingga ekspansi biner dari 19^{85} adalah :

  $19^{85} = 19^{2^0} * 19^{2^2} * 19^{2^4} \ * 19^{2^6}$

  karena :

  $85 = 1 * 2^0 + 1 * 2^2 + 1 * 2^4 + 1 * 2^6$

  Oleh karena itu, $19^{85} \ mod \ 391$ dapat diubah menjadi :

$$19^{85} \; mod \; 391 = (19^{2^0} mod \; 391 * 19^{2^2} mod \; 391 * 19^{2^4} mod \; 391 * 19^{2^6} mod \; 391) \; mod \; 391$$

Kemudian dilakukan perhitungan untuk mencari nilai $19^{2^0} mod \; 391$, $19^{2^2} mod \; 391$, $19^{2^4} mod \; 391$, dan $19^{2^6} mod \; 391$:

$(19)^1 \; mod \; 391 = 19 \; mod \; 391 = 19$

$(19)^2 \; mod \; 391 = 361 \; mod \; 391$
$= 361$

$(19)^4 \; mod \; 391 = (19^2)^2 \; mod \; 391$
$= (361)^2 \; mod \; 391$
$= 130321 \; mod \; 391$
$= 118$

$(19)^8 \; mod \; 391 = (19^4)^2 \; mod \; 391$
$= (118)^2 \; mod \; 391$
$= 13924 \; mod \; 391$
$= 239$

$(19)^{16} \; mod \; 391 = (19^8)^2 \; mod \; 391$
$= (239)^2 \; mod \; 391$
$= 57121 \; mod \; 391$
$= 35$

$(19)^{32} \; mod \; 391 = (19^{16})^2 \; mod \; 391$
$= (35)^2 \; mod \; 391$
$= 1225 \; mod \; 391$
$= 52$

$(19)^{64} \; mod \; 391 = (19^{32})^2 \; mod \; 391$
$= (52)^2 \; mod \; 391$
$= 2704 \; mod \; 391$
$= 358$

sehingga :

$$19^{85} \; mod \; 391 = (19^{2^0} mod \; 391 * 19^{2^2} mod \; 391 * 19^{2^4} mod \; 391 * 19^{2^6} mod \; 391) \; mod \; 391$$

$$= (19 * 118 * 35 * 358)\ mod\ 391$$
$$= 28092260\ \ mod\ 391$$
$$= 83$$

Dari perhitungan di atas, diperoleh *plaintext* 6783. *Plaintext* yang berupa kode ASCII tersebut diubah menjadi karakter "CS", karena 67 berarti karakter "C" dan 78 berarti karakter "S".

3.3.4 PSNR (*Peak Signal to Noise Ratio*)

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

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

dimana :

n = ukuran byte berkas MP3

I = nilai byte pada MP3 *carrier* (lihat Tabel 3.3)

K = nilai byte pada *stego* MP3 (lihat Tabel 3.3)

Perhitungan MSE ditunjukkan pada tabel 3.6.

Tabel 3.6 Tabel perhitungan MSE

Byte (i)	MP3 *carrier* (I)		*Stego* MP3 (K)		$\|I(i) - K(i)\|^2$
	Biner	Desimal	Biner	Desimal	
1	10001101	141	10001101	141	0
2	00001000	8	00001001	9	1
3	01000111	71	01000110	70	1
4	01100101	101	01100100	100	1
5	01010000	80	01010000	80	0
6	10010011	147	10010011	147	0
7	10011010	154	10011011	155	1
8	01111011	123	01111010	122	1
9	01110010	114	01110010	114	0

10	01110111	119	01110111	119	0
11	11101010	234	11101010	234	0
12	00001010	10	00001011	11	1
13	11111010	250	11111010	250	0
14	00111101	61	00111100	60	1
15	11100110	230	11100110	230	0
16	01011110	94	01011111	95	1
17	01011001	89	01011001	89	0
18	01100101	101	01100101	101	0
19	10000001	129	10000000	128	1
20	01010110	86	01010110	86	0
21	01101101	109	01101100	108	1
22	01100001	97	01100000	96	1
23	11101101	237	11101101	237	0
24	11000001	193	11000000	192	1
25	10101111	175	10101111	175	0
26	00101000	40	00101000	40	0
27	01101100	108	01101101	109	1
28	11110100	244	11110100	244	0
29	11000100	196	11000101	197	1
30	00100001	33	00100001	33	0
31	01111101	125	01111101	125	0
32	11010011	211	11010011	211	0
33	01011011	91	01011010	90	1
34	11000100	196	11000101	197	1
35	00100001	33	00100000	32	1
36	01010110	86	01010111	87	1
37	11000111	199	11000111	199	0
38	00111101	61	00111100	60	1
39	01101100	108	01101100	108	0
40	10011010	154	10011011	155	1

	$\sum_{i=1}^{n} \|I(i) - K(i)\|^2$	20

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1} \|I(i) - K(i)\|^2$$
$$= \frac{1}{40} * 20$$
$$= 0.5$$

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

Selanjutnya dilakukan perhitungan nilai PSNR menggunakan persamaan 2.14.

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

MAX_I merupakan *bits per sample* pada sinyal berkas MP3. MAX_I dihitung menggunakan persamaan 2.15. Oleh karena jumlah bit pada setiap byte berkas MP3 bernilai 8, maka :

$$MAX_I = 2^I - 1$$
$$= 2^8 - 1$$
$$= 255$$
$$PSNR = 20 \log_{10} \left(\frac{255}{\sqrt{0.5}}\right)$$
$$= 20 \log_{10} \left(\frac{255}{\sqrt{0.5}}\right)$$
$$= 20 \log_{10} (360.69)$$
$$= 20 * 2.56$$
$$= 51.14 \text{ dB}$$

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

3.4 Perancangan Uji Coba dan Analisis

Setelah sistem dibuat, langkah selanjutnya adalah melakukan pengujian terhadap sistem. Pengujian pertama dilakukan dengan menghitung nilai PSNR menggunakan persamaan 2.14. Nilai PSNR digunakan untuk mengetahui kualitas berkas MP3 yang telah disisipi pesan atau *stego* MP3.

Nilai PSNR yang baik menunjukkan bahwa kualitas media penampung atau berkas MP3 tidak banyak berubah akibat penyembunyian pesan. Kualitas audio yang baik memiliki nilai PSNR minimal 30 dB. Apabila hasil yang didapat dibawah 30 dB, maka berkas MP3 tersebut memiliki kualitas yang buruk dan tidak layak didengar.

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

Pada setiap pesan yang akan disembunyikan pada *stego* MP3, akan diuji cobakan dengan mengenkripsi pesan tersebut menggunakan beberapa kunci publik algoritma RSA yang mempunyai panjang yang berbeda-beda. Hal ini dimaksudkan untuk mengetahui pengaruh ukuran pesan dan panjang kunci publik yang mengenkripsi pesan tersebut terhadap nilai PSNR pada *stego* MP3. Dengan parameter berupa ukuran pesan dan panjang kunci tersebut, diharapkan akan diketahui ukuran pesan dan panjang kunci yang akan menghasilkan nilai PNSR terbaik pada *stego* MP3.

Tabel 3.7 menunjukkan contoh tabel pengujian nilai PSNR berdasarkan parameter ukuran pesan yang akan disembunyikan dan panjang kunci publik yang mengenkripsi pesan tersebut. Pada tabel tersebut juga ditampilkan rasio ukuran pesan, yaitu rasio ukuran pesan hasil enkripsi atau *ciphertext* terhadap ukuran berkas MP3 sebagai media penampung penyembunyian pesan tersebut.

Tabel 3.7 Tabel contoh pengujian nilai PSNR

No.	Berkas MP3	Berkas Teks	Panjang Kunci (Bit)	Rasio Ukuran Pesan (%)	PSNR (dB)

Pengujian kedua dilakukan untuk mengetahui ketahanan steganografi *parity coding* terhadap manipulasi pada berkas *stego* MP3. Manipulasi yang dilakukan berupa operasi penambahan derau *gaussian*. Operasi penambahan derau *gaussian* dilakukan terhadap *stego* MP3. Steganografi *parity coding* dapat dikatakan memilki ketahanan terhadap operasi penambahan derau *gaussian* apabila pesan dapat diekstraksi dari *stego* MP3 dan tidak mengalami kerusakan.

Pengujian ini dilakukan dengan menambahkan derau *gaussian* di sepanjang audio. Penambahan derau dilakukan dengan menambahkan derau dengan intensitas tertentu. Setelah dilakukan penambahan derau pada *stego* MP3, dilakukan pengungkapan pesan pada *stego* MP3 tersebut.

Parameter yang digunakan pada pengujian ini adalah status pengungkapan pesan dan nilai *bit error rate* (BER) pada pesan hasil pengungkapan. Status pengungkapan pesan menyatakan keberhasilan sistem dalam mengekstraksi pesan dari *stego* MP3. Apabila pesan berhasil diekstraksi, maka selanjutnya dilakukan penghitungan *bit error rate*. *Bit error rate* digunakan untuk menghitung persentase bit pesan yang berbeda dari bit pesan asli saat proses pengungkapan pesan. *Bit error rate* dihitung menggunakan persamaan 2.17.

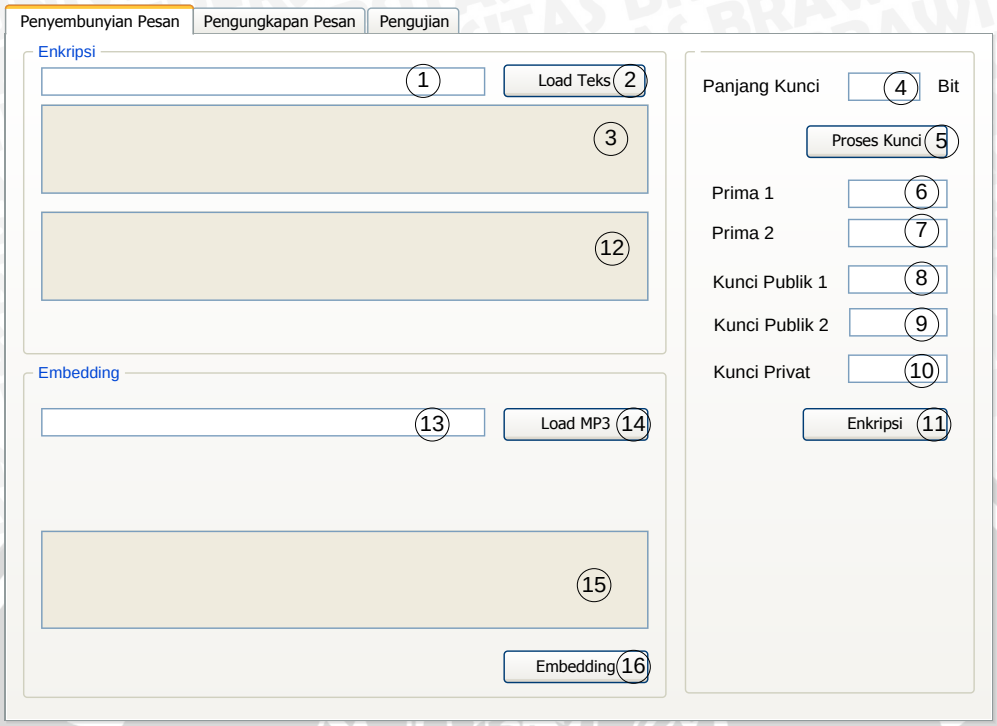
Apabila pesan hasil pengungkapan memiliki nilai *bit error rate* 0%, maka hal ini mengindikasikan bahwa steganografi *parity coding* memiliki ketahanan terhadap operasi penambahan derau sehingga pesan yang disembunyikan pada berkas audio tetap dapat diekstraksi dan tidak mengalami kerusakan. Tabel 3.8 menunjukkan contoh tabel pengujian ketahanan steganografi *parity coding* terhadap operasi penambahan derau *gaussian* pada *stego* MP3.

Tabel 3.8 Tabel contoh pengujian ketahanan steganografi *parity coding*

Berkas MP3	Perlakuan	Status Pengungkapan	*Bit error rate* (%)

3.5 Perancangan Antarmuka

Antarmuka atau *interface* pada sistem ini terdiri dari tiga bagian, yaitu bagian penyembunyian pesan, pengungkapan pesan, dan pengujian. Untuk rancangan antarmuka bagian penyembunyian pesan ditunjukkan pada Gambar 3.13.



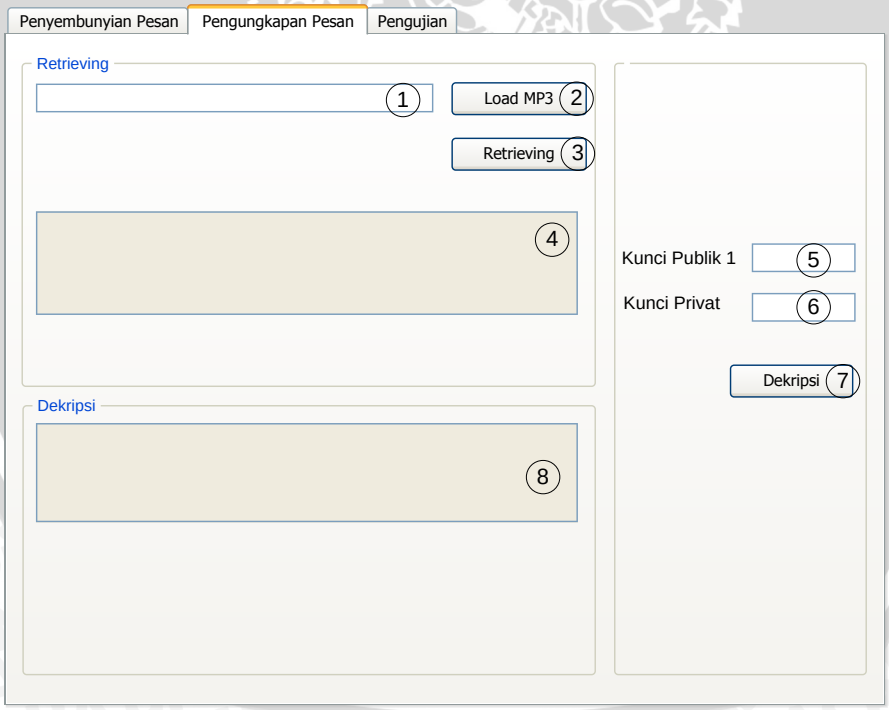
Gambar 3.13 Rancangan Antarmuka Penyembunyian Pesan

Rancangan antarmuka penyembunyian pesan pada Gambar 3.13 terdiri dari :

1. *Textfield* yang menampilkan direktori pesan yang akan disembunyikan.
2. Tombol yang berfungsi membuka jendela untuk memilih pesan.
3. *Textarea* yang berfungsi menampilkan *plaintext* atau pesan yang akan dienkripsi.
4. *Textfield* yang berfungsi sebagai masukan panjang kunci yang akan dibangkitkan.
5. Tombol yang berfungsi untuk membangkitkan bilangan pasangan kunci publik dan kunci privat.
6. *Textfield* yang berfungsi menampilkan nilai bilangan prima pertama hasil pembangkitan bilangan prima.
7. *Textfield* yang berfungsi menampilkan nilai bilangan prima kedua hasil pembangkitan bilangan prima.
8. *Textfield* yang menampilkan kunci publik 1 yang telah dibangkitkan.
9. *Textfield* yang menampilkan kunci publik 2 yang telah dibangkitkan.
10. *Textfield* yang menampilkan kunci privat yang telah dibangkitkan.

11. Tombol yang berfungsi menjalankan perintah untuk melakukan proses enkripsi.
12. *Textarea* yang menampilkan *ciphertext* atau pesan yang telah terenkripsi.
13. *Textfield* yang menampilkan direktori berkas MP3 yang akan digunakan sebagai media penampung pesan atau MP3 *carrier*.
14. Tombol yang berfungsi membuka jendela untuk memilih berkas MP3 *carrier*.
15. *Textarea* yang menampilkan informasi berkas MP3 yang dipilih, meliputi ukuran berkas, jumlah *frame*, dan ukuran maksimal pesan yang dapat ditampung.
16. Tombol yang berfungsi menjalankan perintah untuk melakukan proses *embedding*.

Untuk rancangan antarmuka bagian pengungkapan pesan ditunjukkan pada gambar 3.14.

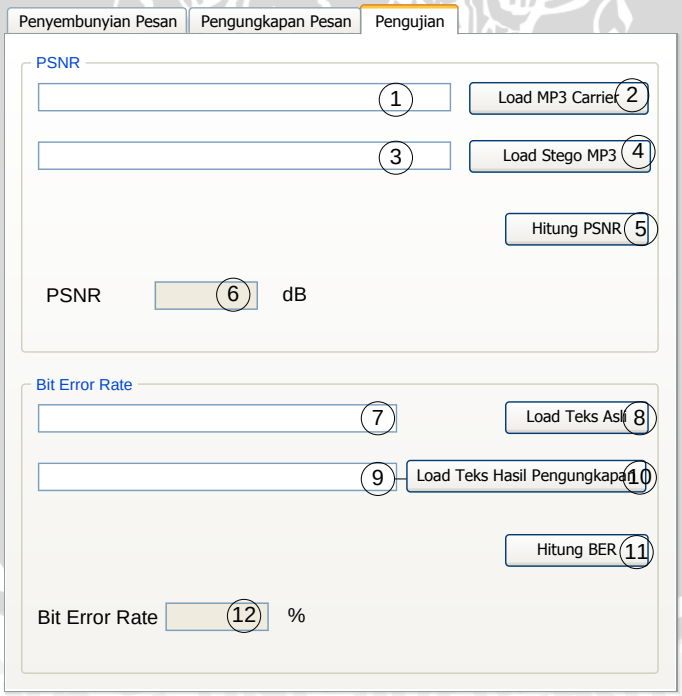


Gambar 3.14 Rancangan Antarmuka Pengungkapan Pesan

Rancangan antarmuka pengungkapan pesan pada Gambar 3.14 terdiri dari :

1. *Textfield* yang menampilkan direktori dari berkas MP3 yang telah disisipi pesan atau *stego* MP3.
2. Tombol yang berfungsi membuka jendela untuk memilih berkas MP3.
3. Tombol yang berfungsi untuk menjalankan perintah untuk melakukan proses *retrieving*.
4. *Textarea* yang menampilkan pesan yang berhasil diungkap dari *stego* MP3.
5. *Textarea* sebagai masukan kunci publik 1.
6. *Textarea* sebagai masukan kunci privat.
7. Tombol yang berfungsi untuk menjalankan perintah untuk melakukan proses dekripsi.
8. *Textarea* yang menampilkan pesan asli atau *plaintext* yang didapatkan dari proses dekripsi.

Untuk rancangan antarmuka bagian pengujian ditunjukkan pada gambar 3.15.



Gambar 3.15 Rancangan Antarmuka Pengujian

Rancangan antarmuka pengujian pada Gambar 3.15 terdiri dari :

1. *Textfield* untuk menampilkan direktori berkas MP3 asli yang belum disisipi pesan atau MP3 *carrier*.
2. Tombol yang berfungsi untuk menampilkan jendela untuk memilih berkas MP3 *carrier*.
3. *Textfield* untuk menampilkan direktori berkas MP3 yang sudah disisipi pesan atau *stego* MP3.
4. Tombol yang berfungsi untuk menampilkan jendela untuk memilih *stego* MP3.
5. Tombol untuk menjalankan perintah untuk melakukan perhitungan nilai PSNR.
6. *Textfield* yang menampilkan nilai PSNR.