BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi dari rancangan aplikasi penyembunyian pesan rahasia terenkripsi pada berkas audio MP3 menggunakan metode *parity coding* yang telah dibahas di Bab III, terdiri dari lingkungan perangkat keras dan lingkungan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

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

1. *Processor* Intel® Core™ i3-2330M CPU @ 2.20 GHz
2. *Memory* 2048 MB DDR3
3. *Harddisk* 500 GB
4. *System type* 64-bit OS
5. *Graphics* Intel® HD Graphics Family @ 784 MB

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang diterapkan dalam pengembangan aplikasi penyembunyian pesan rahasia terenkripsi pada berkas audio MP3 menggunakan metode *parity coding* dan uji cobanya adalah :

1. Sistem operasi Windows 7 Ultimate 64-bit sebagai lingkungan aplikasi dijalankan.
2. NetBeans IDE 6.9.1 sebagai salah satu *software development* untuk pemrograman.

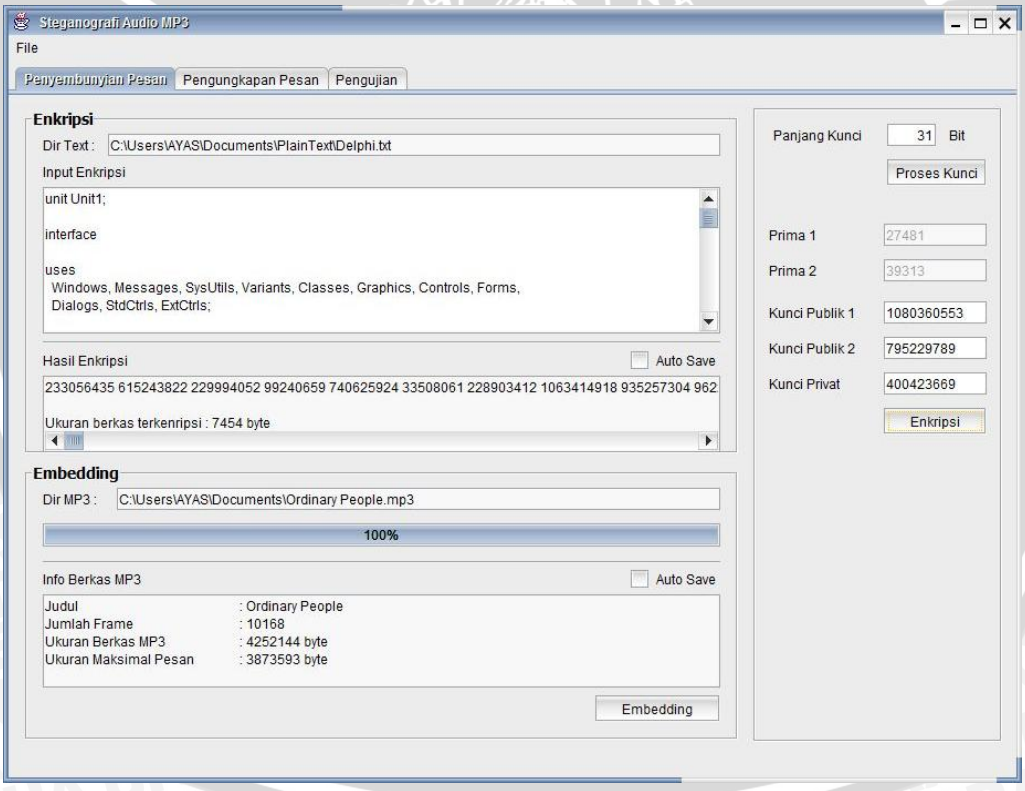
4.2 Implementasi Program

Pada subbab ini akan dijelaskan implementasi program dalam pembuatan aplikasi penyembunyian pesan rahasia terenkripsi pada berkas audio MP3 menggunakan metode *parity coding* berdasarkan perancangan sistem yang telah dijelaskan pada Bab III. Implementasi pembuatan aplikasi dibuat dengan

menggunakan bahasa pemrograman JAVA. Terdapat tiga proses utama dalam implementasi pembuatan aplikasi yaitu penyembunyian pesan, pengungkapan pesan, dan pengujian.

4.2.1 Proses Penyembunyian Pesan

Proses penyembunyian pesan merupakan proses pengenkripsian terhadap pesan dan proses penyembunyian pesan hasil enkripsi tersebut ke dalam berkas audio MP3. Proses ini terdiri dari empat langkah utama yaitu pembangkitan kunci, enkripsi, penyisipan informasi jumlah region, dan *embedding*. Selain itu terdapat dua langkah pembacaan berkas yaitu pembacaan berkas teks dan pembacaan berkas audio. Antarmuka penyembunyian pesan ditunjukkan pada gambar 4.1.



Gambar 4.1 Antarmuka Penyembunyian Pesan

4.2.1.1 Implementasi Pembacaan Pesan

Penyembunyian pesan ke dalam MP3 diawali dengan pembacaan pesan berupa berkas bertipe teks (.txt). Proses pembacaan pesan ini dilakukan perbaris

dan sebagai penanda diakhir baris akan ditambahkan karakter '\n' sebagai implementasi perintah pindah baris. Implementasi pembacaan pesan ditunjukkan pada *source code* 4.1.

```
private StringBuilder text;
public void Read(String dir) {
    File fileIn = new File(dir);
    text = new StringBuilder();
    try {
        BufferedReader fIn = new BufferedReader(new
FileReader(fileIn));
        String line;
        while ((line = fIn.readLine()) != null) {
            text.append(line).append('\n');
        }
    } catch (Exception e) {
    }
    text.replace(text.length() - 1, text.length(), "");
}
```

***Source code* 4.1** Implementasi pembacaan berkas bertipe teks (.txt)

4.2.1.2 Implementasi Pembangkitan Kunci

Langkah selanjutnya yang dilakukan adalah membangkitkan kunci yang akan digunakan untuk melakukan proses enkripsi. Untuk membangkitkan kunci, terlebih dahulu dilakukan proses pembangkitan dua buah bilangan prima. Dua buah bilangan tersebut akan digunakan untuk membentuk tiga buah kunci yang terdiri dari kunci publik 1, kunci publik 2, dan satu kunci privat.

Pembangkitan dua buah bilangan prima ini dilakukan dengan membangkitkan semua kemungkinan bilangan prima mulai dari 2 hingga akar pangkat dua dari nilai maksimal tipe data *integer*. Dari sekian banyak bilangan prima yang telah dibangkitkan, akan dipilih secara acak dua buah bilangan prima. Implementasi pembangkitan bilangan dua buah bilangan prima ditunjukkan pada *source code* 4.2.

```
private BigInteger n;
private boolean[] prima;
```

```
private BigInteger p, q;
private Random r;
private int key_length;
public void generatePrima(int key_length) {
    this.key_length = key_length;
    n = BigInteger.valueOf((int) Math.sqrt(Integer.MAX_VALUE));
    prima = new boolean[n.intValue()];
    for (int i = 1; i <= n.intValue(); i++) {
        prima[i - 1] = true;
    }
    prima[0] = false;
    double max = Math.sqrt(n.intValue());
    for (int i = 2; i <= max; i++) {
        if (prima[i - 1]) {
            for (int j = i * i; j <= n.intValue(); j++) {
                if ((j % i) == 0) {
                    prima[j - 1] = false;
                }
            }
        }
    }
    randomPrima();
}
public void randomPrima() {
    r = new Random();
    p = BigInteger.valueOf(r.nextInt(n.intValue() - 2) + 1);
    q = BigInteger.valueOf(r.nextInt(n.intValue() - 2) + 1);
    int key = p.intValue() * q.intValue();
    int length = Integer.toBinaryString(key).length();
    while (!prima[p.intValue() - 1] || !prima[q.intValue() - 1] ||
p.compareTo(q) == 0 || length != key_length) {
        p = BigInteger.valueOf(r.nextInt(n.intValue() - 2) + 1);
        q = BigInteger.valueOf(r.nextInt(n.intValue() - 2) + 1);
        key = p.intValue() * q.intValue();
        length = Integer.toBinaryString(key).length();
    }
}
```

***Source code* 4.2** Implementasi pembangkitan dua buah bilangan prima

Setelah didapatkan dua buah bilangan prima, proses akan dilanjutkan dengan pembentukan kunci. Pada proses ini dilakukan pemanggilan kelas-kelas lainnya, antara lain :

1. Kelas Euclidean

   Kelas Euclidean digunakan untuk menghitung nilai FPB dari dua buah bilangan.

2. Kelas ExtEuclidean

   Kelas ExtEuclidean digunakan untuk menghitung nilai invers modulo dari dua buah bilangan.

Implementasi pembentukan kunci ditunjukkan pada *source code* 4.3.

```
private BigInteger kunciPublik1, kunciPublik2, kunciPrivat, phiEuler;
private Euclidean e;
private ExtEuclidean ext;
private Random r;
private ArrayList<Integer> quotient;
public Key(BigInteger p, BigInteger q) {
    kunciPublik1 = p.multiply(q);
    phiEuler       =       p.subtract(BigInteger.ONE).multiply(q.subtract
(BigInteger.ONE));
    r = new Random();
    do {
        do {
            kunciPublik2          =          BigInteger.valueOf(r.nextInt
((phiEuler.intValue() - 1) - 2) + 1);
            e = new Euclidean(phiEuler, kunciPublik2);
        } while (e.getFPB().compareTo(BigInteger.ONE) == 1);
        ext = new ExtEuclidean(kunciPublik2, phiEuler);
        kunciPrivat = ext.getInverseMod();
    } while (kunciPrivat.compareTo(BigInteger.ONE) == -1);
}
```

***Source code* 4.3** Implementasi pembentukan kunci

4.2.1.3 Implementasi Enkripsi

Langkah selanjutnya yang dilakukan adalah melakukan enkripsi pada pesan teks menggunakan kunci publik 1 dan kunci publik 2. Pesan nantinya akan dipecah menjadi blok-blok karena enkripsi dilakukan satu per satu pada setiap

blok menggunakan persamaan 2.12. Implementasi enkripsi ditunjukkan pada *source code* 4.4.

```
1   private BigInteger kunciPublik1, kunciPublik2, kunciPrivat;
2   private String[] blokPlain;
3   private FastExponent fExp;
4   public   StringBuilder   enkripsi(StringBuilder   pesan,   BigInteger
5   kunciPublik1, BigInteger kunciPublik2) {
6       this.kunciPublik1 = kunciPublik1;
7       this.kunciPublik2 = kunciPublik2;
8       StringBuilder plainText = new StringBuilder();
9       StringBuilder cipherText = new StringBuilder();
10      for (int i = 0; i < pesan.length(); i++) {
11          char temp = pesan.charAt(i);
12          if (temp == '\n') {
13              plainText.append(Integer.toString
14              ((int) temp + 117));
15          } else {
16              plainText.append(Integer.toString
17              ((int) temp));
18          }
19      }
20      int N = plainText.length();
21      int m = Integer.toString   (kunciPublik1.intValue()).length();
22      Random r = new Random();
23      int sizeBlok = r.nextInt(m - 1) + 1;
24      int sisaBagi = N % sizeBlok;
25      int NBlok = N / sizeBlok;
26      boolean sisa = false;
27      if (sisaBagi != 0) {
28          NBlok++;
29          sisa = true;
30      }
31      int indeks1 = 1;
32      int M = sizeBlok;
33      for (int i = 1; i <= NBlok; i++) {
34          if (sisa) {
35              if (i == NBlok) {
36                  M = sisaBagi;
37              }
38          }
```

```
39          int indeks2 = indeks1 + M;
40          StringBuilder     prePlain     =     new     StringBuilder
41  (plainText.substring(indeks1 - 1, indeks2 - 1));
42          if (sisa) {
43              if (i == NBlok) {
44                  for (int j = 0; j < sizeBlok - sisaBagi; j++)
45                  {
46                      prePlain.append("0");
47                  }
48              }
49          }
50          BigInteger        plain        =        BigInteger.valueOf
51  (Integer.parseInt(prePlain.toString()));
52          indeks1 = indeks2;
53          fExp = new FastExponent(plain, kunciPublik2, kunciPublik1);
54          BigInteger cipher = fExp.getResult();
55          if (i != NBlok) {
56              cipherText.append(Integer.toString
57  (cipher.intValue())).append(" ");
58          } else {
59              cipherText.append(Integer.toString
60  (cipher.intValue()));
61          }
62      }
63      return cipherText;
64  }
```

***Source code* 4.4** Implementasi enkripsi

Baris ke-11 sampai dengan 18 merupakan implementasi pengubahan setiap karakter dalam pesan yang akan dienkripsi menjadi bentuk desimal atau kode ASCII. Baris ke-19 sampai dengan 29 merupakan implementasi penghitungan jumlah blok dan ukuran setiap blok. Baris ke-32 sampai dengan 59 merupakan implementasi proses enkripsi yang dilakukan pada setiap blok.

4.2.1.4 Implementasi Pembacaan Berkas Audio

Langkah ini dilakukan untuk membaca byte-byte dari berkas audio MP3 yang akan digunakan sebagai media penampung penyembunyian pesan. Setiap

byte akan diubah ke bentuk 8 bit. Implementasi pembacaan berkas audio ditunjukkan pada *source code* 4.5.

```
public void ReadAudio(final String dir) {
    Thread t = new Thread() {
        @Override
        public void run() {
            try {
                File song = new File(dir);
                FileInputStream file = new FileInputStream(song);
                int count = file.available();
                bit_audio = new StringBuilder();
                for (int i = 0; i < count; i++) {
                    bit_audio.append(fixBinary
(Integer.toBinaryString(file.read())));
                    bar1.setValue((i + 1) * 100 / count);
                }
                file.close();
                InfoAudio(dir);
            } catch (Exception e) {
                System.out.println("Error " + e.toString());
            }
        }
    };
    t.start();
}
```

***Source code* 4.5** Implementasi pembacaan berkas audio

Prosedur InfoAudio pada *source code* 4.5 berfungsi untuk mencari informasi dan karakteristik dari berkas audio MP3. Informasi tersebut antara lain meliputi judul MP3, panjang bit *header* MP3, jumlah *frame*, jarak antara *frame*, panjang bit MP3, panjang maksimal bit MP3 yang dapat disisipi pesan, dan panjang maksimal bit pesan yang dapat ditampung.

4.2.1.5 Implementasi Penyisipan Informasi Jumlah Region

Langkah ini dilakukan untuk menyisipkan informasi panjang dari bit pesan atau jumlah region ke dalam bit audio MP3. Sebelumnya dilakukan terlebih dahulu pengubahan pesan ke dalam bentuk desimal atau kode ASCII. Untuk dapat

disembunyikan ke dalam bit audio, pesan harus diubah ke dalam bentuk biner 8 bit. Pengubahan pesan ke dalam bentuk biner ditunjukkan pada *source code* 4.6.

```
bit_pesan = new StringBuilder();
for(int i=0; i<cipher_text.length(); i++){
    int code = (int)cipher_text.charAt(i);
    bit_pesan.append(fixBinary(Integer.toBinaryString (code)));
}
size_pesan = bit_pesan.length();
```

***Source code* 4.6** Pengubahan pesan ke dalam bentuk biner

Selanjutnya informasi panjang dari bit pesan diubah ke dalam tipe data *string*. Setelah diubah ke tipe data string, informasi panjang tersebut diubah ke dalam bentuk desimal atau kode ASCII. Untuk dapat disembunyikan ke dalam bit-bit audio, informasi panjang harus diubah ke dalam bentuk biner 8 bit. Pada bagian akhir dari bit informasi ditambahkan bit pembatas berupa "1000000110001101" sebagai penanda akhir dari bit informasi. Pengubahan informasi panjang ke dalam bentuk biner ditunjukkan pada *source code* 4.7.

```
bit_nregion = new StringBuilder();
nregion = size_pesan;
String region = Integer.toString(nregion);
for (int i = 0; i < region.length(); i++) {
    int code = (int) region.charAt(i);
    bit_nregion.append(fixBinary(Integer.toBinaryString   (code)));
}
String bitPenanda = "1000000110001101";
bit_nregion.append(bitPenanda);
```

***Source code* 4.7** Pengubahan informasi panjang ke dalam bentuk biner

Selanjutnya, bit informasi disembunyikan ke dalam bit audio MP3. Setiap satu bit informasi akan disembunyikan pada byte pertama *main data* dari sebuah *frame*. Penyisipan dilakukan hingga semua bit informasi telah disembunyikan. Implementasi penyisipan informasi jumlah region ditunjukkan pada *source code* 4.8.

```
1   private StringBuilder pre_stego;
2   public void HideInfo(StringBuilder bit_audio, StringBuilder info,
3   int start_frame, ArrayList<Integer> Length)
4   {
5       pre_stego = new StringBuilder(bit_audio.toString());
6       int index1, index2;
7       int K = info.length();
8       int N = 0;
9       int frame = 0;
10      index1 = start_frame;
11      while(N<K)
12      {
13          index2 = index1 + 288;
14          index1 = index1 + Length.get(frame);
15          frame++;
16          for(int i=0; i<8; i++)
17          {
18              pre_stego.setCharAt(index2 + i, info.charAt(N));
19              N++;
20          }
21       }
22   }
```

***Source code* 4.8** Implementasi penyisipan informasi jumlah region

Baris ke-11 sampai dengan 21 menunjukkan implementasi penyisipan setiap delapan bit informasi jumlah region pada byte pertama *main data* dari setiap *frame*. Baris ke-13 menunjukkan bahwa *main data* dimulai pada bit ke 288 pada sebuah *frame*. Baris ke-16 sampai dengan 20 mengimplementasikan penggantian nilai *bit* informasi pada byte pertama tersebut.

4.2.1.6 Implementasi *Embedding*

Langkah *embedding* merupakan langkah penyembunyian bit pesan ke dalam bit audio MP3. Bit audio dibagi menjadi beberapa region sebanyak bit pesan. Penyembunyian pesan dilakukan dengan mengganti nilai *parity* bit dari setiap region dengan bit pesan. Implementasi *embedding* ditunjukkan pada *source code* 4.9.

```
1   private StringBuilder stego;
2   private int size_region;
3   public   void   Embedding(StringBuilder   bit_audio,   StringBuilder
4   bit_pesan,  int  nregion,  int  available_bit,  int  start_frame,
5   ArrayList<Integer> Length) {
6       stego = new StringBuilder(bit_audio.toString());
7       size_region = available_bit / nregion;
8       int frame = 0, indexLSB = 0;
9       int index1, index2, index3;
10      index1 = start_frame;
11      index2 = index1 + 296;
12      index1 = index1 + Length.get(frame);
13      frame++;
14      for (int i = 0; i < nregion; i++) {
15          String parity = "";
16          index3 = index2 + size_region;
17          if (index3 >= index1) {
18              StringBuilder region = new StringBuilder();
19              int N = 0;
20              while (index3 >= index1) {
21                  region.append(stego.substring(index2, index1));
22                  N = index3 - index1;
23                  index2 = index1 + 296;
24                  index1 = index1 + Length.get(frame);
25                  frame++;
26                  index3 = index2 + N;
27              }
28              region.append(stego.substring(index2, index3));
29              parity = ParityBit(region);
30              if (index2 == index3) {
31                  indexLSB = index3 - 297;
32              } else {
33                  indexLSB = index3 - 1;
34              }
35              index2 = index3;
36          } else {
37              StringBuilder          region          =          new
38  StringBuilder(stego.substring(index2, index3));
39              parity = ParityBit(region);
40              indexLSB = index3 - 1;
41              index2 = index3;
```

```
42          }
43          if (!parity.equals(bit_pesan.charAt(i) + "")) {
44              if (stego.charAt(indexLSB) == '1') {
45                  stego.setCharAt(indexLSB, '0');
46              } else {
47                  stego.setCharAt(indexLSB, '1');
48              }
49          }
50      }
51  }
```

Source code 4.9 Implementasi *embedding*

Baris ke-8 menunjukkan penghitungan ukuran dari setiap region. Baris ke-11 sampai dengan 52 menunjukkan implementasi penyembunyian setiap bit pesan ke dalam setiap region bit audio MP3. Baris ke-12 menunjukkan bahwa bit pada *main data* yang dapat dimodifikasi dimulai pada bit ke 286 pada sebuah *frame*. Baris Baris ke-30 dan ke-40 menunjukkan penghitungan nilai *parity* bit pada sebuah region. Nilai *parity* bit diperoleh dari fungsi ParityBit pada *source code* 4.10. Baris ke-44 sampai dengan 50 mengimplementasikan penggantian nilai *parity* bit pada sebuah region.

Implementasi penghitungan nilai *parity* bit dari suatu bit ditunjukkan pada *source code* 4.10.

```
public String ParityBit(StringBuilder region) {
    int num = 0;
    for (int i = 0; i < region.length(); i++) {
        if (region.charAt(i) == '1') {
            num++;
        }
    }
    if (num % 2 == 0) {
        return "0";
    } else {
        return "1";
    }
}
```

***Source code* 4.10** Implementasi penghitungan *parity* bit

Langkah terakhir dari penyembunyian pesan ke dalam MP3 adalah pembetukan kembali berkas audio MP3 dari bit audio yang telah disisipi bit pesan. Implementasi pembentukan berkas audio MP3 ditunjukkan pada *source code* 4.11.

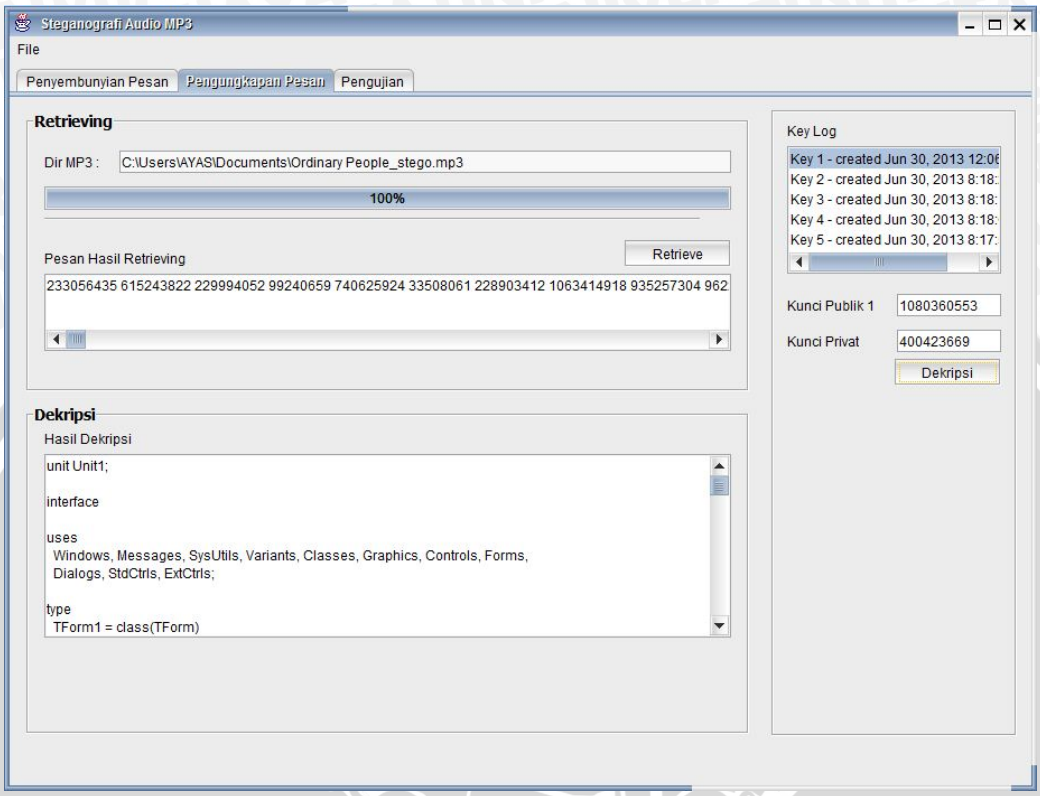
```
public void WriteAudio(final String save) {
    Thread t = new Thread() {
        @Override
        public void run() {
            int count = stego.length() / 8;
            String[] new_byte = new String[count];
            int index = 0;
            for (int i = 1; i <= count; i++) {
                new_byte[i - 1] = stego.substring(index, i * 8);
                index = i * 8;
            }
            try {
                File trashedSong = new File(save + ".mp3");
                FileOutputStream trash = new
FileOutputStream(trashedSong);
                for (int i = 0; i < count; i++) {
                    trash.write(Integer.parseInt(new_byte[i], 2));
                    bar1.setValue((i + 1) * 100 / count);
                }
                trash.close();
            } catch (Exception e) {
                System.out.println("Error â€" " + e.toString());
            }
        }};
    t.start();}
```

***Source code* 4.11** Implementasi pembentukan berkas audio MP3

4.2.2 Proses Pengungkapan Pesan

Proses pengungkapan pesan merupakan proses pengungkapan pesan dari berkas audio MP3 dan prose dekripsi terhadap pesan hasil pengungkapan tersebut. Proses pengungkapan pesan dalam MP3 terdiri dari tiga langkah utama yaitu pengambilan informasi jumlah region, *retrieving*, dan dekripsi. Selain itu terdapat

pula langkah pembacaan berkas yaitu pembacaan berkas audio. Antarmuka pengungkapan pesan ditunjukkan pada gambar 4.2.



Gambar 4.2 Antarmuka Pengungkapan Pesan

4.2.2.1 Implementasi Pengambilan Informasi Jumlah Region

Pengungkapan pesan dalam MP3 diawali dengan pengambilan informasi jumlah region. Informasi jumlah region diperlukan pada proses *retrieving*. Sebelum dilakukan pengambilan informasi jumlah region, dilakukan terlebih dahulu pembacaan berkas audio MP3. Implementasi pembacaan berkas audio ditunjukkan pada *source code* 4.5. Langkah ini dilakukan untuk membaca byte-byte dari berkas audio MP3 yang selanjutnya akan diubah ke bentuk 8 bit. Langkah ini juga dilakukan untuk mencari informasi dan karakteristik dari berkas audio MP3.

Selanjutnya, dilakukan pengambilan bit informasi pada byte pertama *main data* dari setiap *frame*. Dari bit yang diambil pada byte pertama tersebut dapat disusun bit informasi. Informasi jumlah region diperoleh dengan mengubah bit

informasi menjadi ke bentuk desimal atau kode ASCII yang kemudian akan diubah ke bentuk karakter.

Pengambilan bit informasi dilakukan hingga ditemukan bit pembatas. Apabila hingga *frame* terakhir tidak ditemukan bit pembatas, maka diasumsikan bit-bit audio tidak mengandung pesan yang disembunyikan. Implementasi pengambilan informasi jumlah region ditunjukkan pada *source code* 4.12.

```
1   private int info;
2   private boolean kosong;
3   public void ExtractInfo(StringBuilder bit_audio, int start_frame,
4   int frame_length, ArrayList<Integer> Length)
5   {
6       String Char = "", region = "";
7       boolean detect = false;
8       int frame = 0;
9       int index1, index2;
10      kosong = true;
11      index1 = start_frame;
12      while(kosong && frame < frame_length)
13      {
14          index2 = index1 + 288;
15          index1 = index1 + Length.get(frame);
16          frame++;
17          for(int i=0; i<8; i++)
18          {
19              Char = Char + bit_audio.charAt(index2 + i);
20          }
21          if(detect){
22              if(Char.equals("10000001")){
23                  region = region + (char)Integer.parseInt(Char, 2);
24              }else{
25                  if(Char.equals("10001101")){
26                      kosong = false;
27                      region = region.substring(0, region.length() -
28  1);
29                      info = Integer.parseInt(region);
30                  }else{
31                      detect = false;
32                      region = region + (char)Integer.parseInt(Char,
33  2);
```

```
34                  }
35              }
36          }else{
37              if(Char.equals("10000001")){
38                  detect = true;
39              } region = region + (char)Integer.parseInt(Char, 2);
40          } Char = "";
41      }
42  }
```

***Source code* 4.12** Implementasi penyisipan informasi jumlah region

Baris ke-12 sampai dengan 41 menunjukkan implementasi pembilan setiap bit informasi jumlah region pada byte pertama *main data* dari setiap *frame*. Baris ke-21 sampai dengan 40 mengimplementasikan pengecekan bit pembatas pada bit informasi yang telah disusun.

4.2.2.2 Implementasi *Retrieving*

Langkah *retrieving* merupakan langkah pengungkapan bit pesan dari dalam bit audio MP3. Bit audio dibagi menjadi beberapa region sebanyak jumlah region. Pengungkapan pesan dilakukan dengan menghitung nilai *parity* bit dari setiap region. Dari nilai *parity* bit tersebut dapat disusun bit pesan. Implementasi *embedding* ditunjukkan pada *source code* 4.13.

```
1   private StringBuilder hidden;
2   private int size_region;
3   public  void  Retrieving(StringBuilder  stego,  int  nregion,  int
4   available_bit, int start_frame, ArrayList<Integer> Length) {
5       hidden = new StringBuilder();
6       size_region = available_bit / nregion;
7       int frame = 0;
8       int index1, index2, index3;
9       index1 = start_frame;
10      index2 = index1 + 296;
11      index1 = index1 + Length.get(frame);
12      frame++;
13      for (int i = 0; i < nregion; i++) {
14          String parity = "";
```

```
15          index3 = index2 + size_region;
16          if (index3 >= index1) {
17              StringBuilder region = new StringBuilder();
18              int N = 0;
19              while (index3 >= index1) {
20                  region.append(stego.substring(index2, index1));
21                  N = index3 - index1;
22                  index2 = index1 + 296;
23                  index1 = index1 + Length.get(frame);
24                  frame++;
25                  index3 = index2 + N;
26              }
27              region.append(stego.substring(index2, index3));
28              parity = ParityBit(region);
29              index2 = index3;
30          } else {
31              StringBuilder region = new
32  StringBuilder(stego.substring(index2, index3));
33              parity = ParityBit(region);
34              index2 = index3;
35          }
36          hidden.append(parity);
37      }
38  }
```

Source code 4.13 Implementasi *retrieving*

Baris ke-6 menunjukkan penghitungan ukuran dari setiap region. Baris ke-13 sampai dengan 37 menunjukkan implementasi penyembunyian setiap bit pesan ke dalam setiap region bit audio MP3. Baris ke-28 dan ke-33 menunjukkan penghitungan nilai *parity* bit pada sebuah region. Nilai *parity* bit diperoleh dari fungsi ParityBit pada *source code* 4.10. Baris ke-36 mengimplementasikan penyusunan nilai *parity* bit yang menjadi bit pesan.

Pesan hasil pengungkapan diperoleh dengan mengubah bit pesan menjadi ke bentuk desimal atau kode ASCII yang kemudian akan diubah ke bentuk karakter. Pengubahan bit pesan menjadi pesan hasil pengungkapan ditunjukkan pada *source code* 4.14.

```
bit_pesan = pcd.getHidden();
```

```
String temp = "";
for (int i = 1; i <= bit_pesan.length(); i++) {
    temp = temp + bit_pesan.charAt(i - 1);
    if (i % 8 == 0) {
        cipher_text.append(String.valueOf((char)
Integer.parseInt(temp, 2)));
        temp = "";
    }
}
```

***Source code* 4.14** Pengubahan bit pesan ke pesan

4.2.2.3 Implementasi Dekripsi

Langkah selanjutnya yang dilakukan adalah melakukan dekripsi pada pesan hasil pengungkapan menggunakan kunci publik 1 dan kunci privat. Pesan nantinya akan dipecah menjadi blok-blok berdasarkan karakter spasi karena dekripsi dilakukan satu per satu pada setiap blok menggunakan persamaan 2.13. Implementasi enkripsi ditunjukkan pada *source code* 4.15.

```
1  public StringBuilder dekripsi(StringBuilder cipherText, BigInteger
2  kunciPublik1, BigInteger kunciPrivat) {
3      this.kunciPublik1 = kunciPublik1;
4      this.kunciPrivat = kunciPrivat;
5      StringBuilder plainText = new StringBuilder();
6      StringBuilder pesan = new StringBuilder();
7      int N = cipherText.length();
8      StringBuilder preCipher = new StringBuilder();
9      String[] blokCipher = cipherText.toString().split(" ");//
10     int NBlok = blokCipher.length;
11     blokPlain = new String[NBlok];//
12     int pointer = 0;//
13     for (int i = 1; i <= N; i++) {
14         String temp = cipherText.charAt(i - 1) + "";
15         if (temp.equals(" ")) {
16             BigInteger                    cipher                    =
17 BigInteger.valueOf(Integer.parseInt(preCipher.toString()));
18             preCipher = new StringBuilder();
19             fExp    =    new    FastExponent(cipher,    kunciPrivat,
20 kunciPublik1);
21             BigInteger plain = fExp.getResult();
```

```
22              blokPlain[pointer]                                  =
23  Integer.toString(plain.intValue());//
24              pointer++;//
25          } else {
26              preCipher.append(temp);
27          }
28      }
29      BigInteger                          cipher                  =
30  BigInteger.valueOf(Integer.parseInt(preCipher.toString()));
31      fExp = new FastExponent(cipher, kunciPrivat, kunciPublik1);
32      BigInteger plain = fExp.getResult();
33      blokPlain[pointer] = Integer.toString(plain.intValue());//
34      int sizeBlok = Integer.MIN_VALUE;
35      for (int i = 0; i < NBlok; i++) {
36          StringBuilder             temp              =             new
37  StringBuilder(blokPlain[i].toString());
38          int size = temp.length();
39          if (sizeBlok < size) {
40              sizeBlok = size;
41          }
42      }
43      for (int i = 0; i < NBlok; i++) {
44          StringBuilder             temp              =             new
45  StringBuilder(blokPlain[i].toString());
46          if (temp.length() == sizeBlok) {
47              plainText.append(temp);
48          } else if (temp.length() < sizeBlok) {
49              int sizeTemp = temp.length();
50              for (int j = 0; j < sizeBlok - sizeTemp; j++) {
51                  StringBuilder zero = new StringBuilder("0");
52                  zero.append(temp);
53                  temp = new StringBuilder(zero.toString());
54              }
55              blokPlain[i] = temp.toString();//
56              plainText.append(temp);
57          }
58      }
59      pesan = toChar(plainText);
60      return pesan;
61  }
```

***Source code* 4.15** Implementasi dekripsi

Baris ke-13 sampai dengan 33 merupakan implementasi proses dekripsi yang dilakukan pada setiap blok sesuai dengan persamaan 2.13. Baris ke-43 sampai dengan 58 merupakan implementasi penambahan jumlah digit pada blok yang memliki jumlah digit kurang dari ukuran blok sehingga setiap blok memiliki ukuran yang sama. Penambahan digit dilakukan dengan menambahkan bilangan 0 pada blok yang memiliki jumlah digit kurang dari ukuran blok. Fungsi toChar pada baris ke-59 berfungsi untuk mengubah pesan hasil dekripsi yang masih berupa kode ASCII menjadi bentuk karakter.

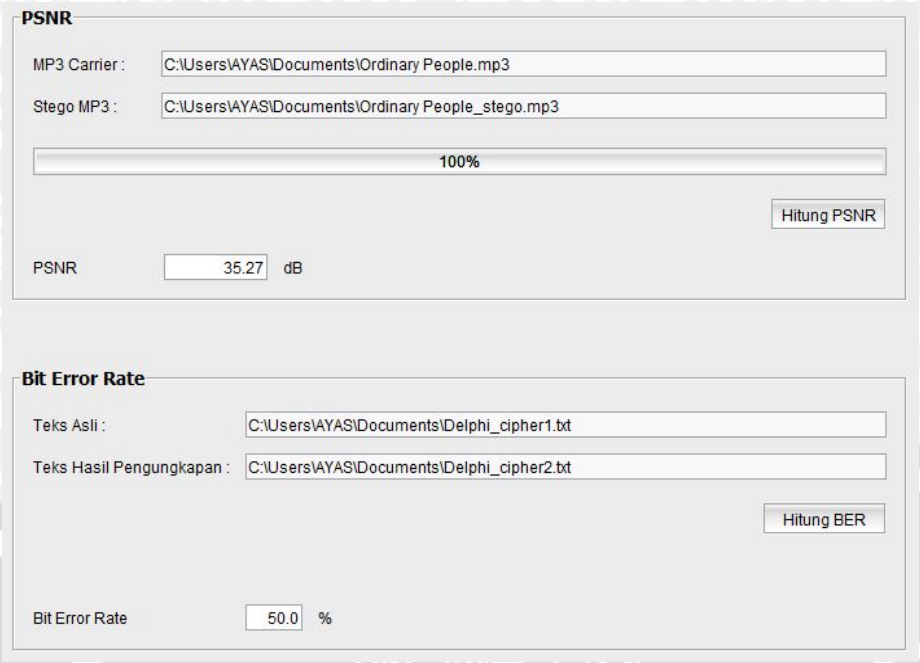
Langkah terakhir dari pegungkapan pesan dalam MP3 adalah pembentukan pesan dalam berkas bertipe teks (.txt). Implementasi pembentukan berkas bertipe teks (.txt) ditunjukkan pada *source code* 4.16.

```
public void Write(String a, StringBuilder text) {
    String[] temp = text.toString().split("\n");
    File fileOut = new File(a + ".txt");
    try {
        BufferedWriter fOut = new BufferedWriter(new
FileWriter(fileOut));
        for (int i = 0; i < temp.length; i++) {
            fOut.write(temp[i]);
            if (i != temp.length - 1) {
                fOut.newLine();
            }
        }
        fOut.close();
    } catch (Exception e) {
    }
}
```

***Source code* 4.16** Implementasi pembentukan berkas bertipe teks (.txt)

4.2.3 Proses Pengujian

Proses pengujian yang terdapat pada aplikasi ini merupakan pengujian PSNR pada dua berkas audio MP3 dan juga pengujian banyak nilai *bit error rate* dari dua berkas teks. Antarmuka pengujian ditunjukkan pada gambar 4.3.



Gambar 4.3 Antarmuka Pengujian

4.2.3.1 Implemetasi PSNR

Pada pengujian PSNR ini akan dibandingkan dua berkas MP3 yaitu MP3 asli dan MP3 yang telah disisipi pesan. Dari dua berkas MP3 tersebut, dapat dihitung nilai PSNR pada berkas MP3 yang telah disisipi pesan dalam satuan dB. Implementasi pengujian PSNR ditunjukkan pada *source code* 4.17.

```
private double MSE;
private double PSNR;
public PSNR(final String dir_MP3, final String dir_stego, final
JProgressBar bar, final JTextField outputPSNR)
{
    Thread t = new Thread()
    {
        @Override
        public void run()
        {
            try
            {
                File song1 = new File(dir_MP3);
                File song2 = new File(dir_stego);
                FileInputStream file1 = new FileInputStream(song1);
```

```
                FileInputStream file2 = new FileInputStream(song2);
                int count1 = file1.available();
                int count2 = file2.available();
                double n;
                double sum = 0;
                double max = 1;
                if (count1 == count2)
                {
                    n = count1 * 8;
                    for (int i = 0; i < count1; i++)
                    {
                        String bitstream1 =
fixBinary(Integer.toBinaryString(file1.read()));
                        String bitstream2 =
fixBinary(Integer.toBinaryString(file2.read()));
                        for(int j=0; j<bitstream1.length(); j++)
                        {
                            String x = bitstream1.charAt(j)+"";
                            String y = bitstream2.charAt(j)+"";
                            if(!x.equals(y))
                                sum++;
                        }
                        bar.setValue((i + 1) * 100 / count1);
                    }
                    sum = sum + 0.000001;
                    MSE = (double)1 / n * sum;
                    PSNR = (double)20 * Math.log10(max /
Math.sqrt(MSE));
                    BigDecimal b = new BigDecimal(PSNR);
                    b = b.setScale(2, RoundingMode.HALF_EVEN);

outputPSNR.setText(Double.toString(b.doubleValue()));
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Jumlah
byte dari kedua MP3 berbeda", "ERROR", 0);
                }
            }
            catch (Exception e)
            {
                System.out.println("Error â€" " + e.toString());
```

```
            }
        }
    };
    t.start();
}
```

***Source code* 4.17** Implementasi pengujian PSNR

4.2.3.2 Implementasi *Bit Error Rate*

Pada pengujian *bit error rate* ini akan dibandingkan dua berkas teks. Dua berkas teks yaitu berkas teks asli dan berkas teks hasil pengungkapan dari berkas MP3. Pengujian *bit error rate* ini menghitung ketepatan bit pesan dari pesan hasil pengungkapan. Implementasi pengujian *bit error rate* ditunjukkan pada *source code* 4.18.

```
bit_uji1 = new StringBuilder();
bit_uji2 = new StringBuilder();
for (int i = 0; i < text_uji1.length(); i++) {
    int code = (int) text_uji1.charAt(i);
    bit_uji1.append(fixBinary(Integer.toBinaryString(code)));
}
for (int i = 0; i < text_uji2.length(); i++) {
    int code = (int) text_uji2.charAt(i);
    bit_uji2.append(fixBinary(Integer.toBinaryString(code)));
}
if (bit_uji1.length() == bit_uji2.length()) {
    int B = bit_uji1.length();
    int n = 0;
    for (int i = 0; i < B; i++) {
    String cek1 = bit_uji1.charAt(i) + "";
    String cek2 = bit_uji2.charAt(i) + "";
    if (!cek1.equals(cek2)) {
        n++;
        }
    }
    double bit_error_rate = 100 * n / B;
```

***Source code* 4.18** Implementasi pengujian *bit error rate*