

BAB IV

IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi dari rancangan aplikasi steganografi pada audio menggunakan metode *parity coding* meliputi lingkungan perangkat keras (*Hardware*) dan lingkungan perangkat lunak (*Software*).

4.1.1 Lingkungan Perangkat Keras (*Hardware*)

Lingkungan perangkat keras untuk mengimplementasikan dan pengembangan aplikasi steganografi pada audio menggunakan metode *parity coding* adalah sebagai berikut :

1. Prosesor Intel® Core™ i3-2330M CPU @ 2.20GHz
2. Memori 2 GB DDR3
3. Harddisk 500 GB
4. VGA Intel HD 3000
5. Monitor 14"

4.1.2 Lingkungan Perangkat Lunak (*Software*)

Perangkat lunak yang digunakan dalam mengimplementasikan dan pengembangan aplikasi steganografi pada audio menggunakan metode *parity coding* adalah sebagai berikut :

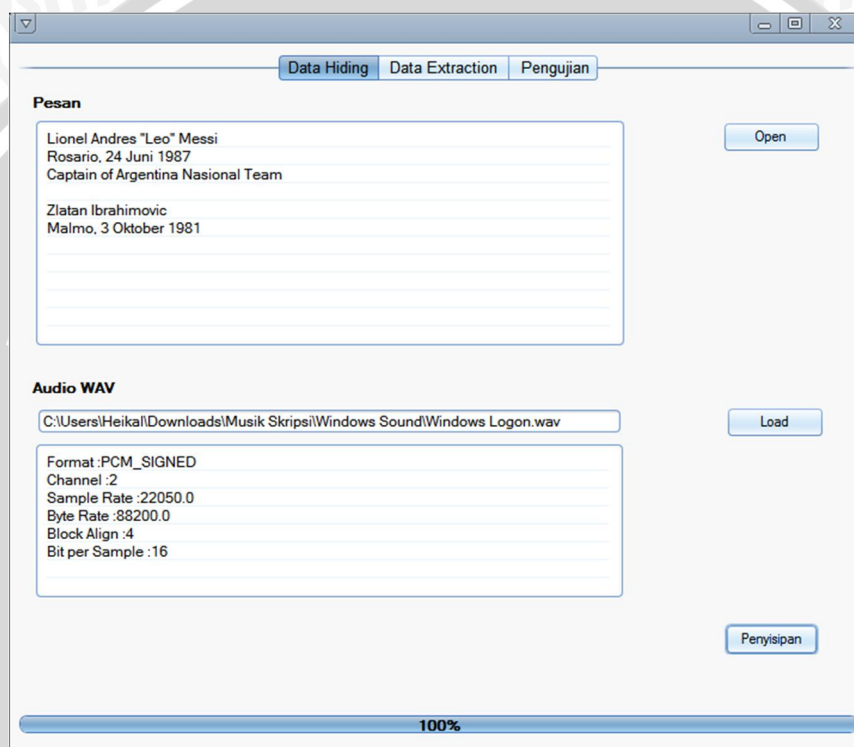
1. Sistem operasi *Windows 7 Ultimate* 32-bit sebagai lingkungan aplikasi dijalankan.
2. Netbeans IDE 7.3 sebagai salah satu *software development* untuk pemrograman.

4.2 Implementasi Program

Implementasi pembuatan aplikasi steganografi pada audio menggunakan metode *parity coding* berdasarkan perancangan sistem pada Bab III. Bahasa pemrograman yang digunakan adalah JAVA. Terdapat tiga proses utama dalam implementasi pembuatan aplikasi yaitu *data hiding*, *data extraction*, dan pengujian.

4.2.1 Proses *Data Hiding*

Pada aplikasi steganografi pada audio menggunakan metode *parity coding*, terdapat kelas *MainForm* yang menangani seluruh proses, termasuk proses *data hiding*. Proses *data hiding* merupakan proses penyisipan *payload* ke dalam berkas audio. Terdapat tiga proses utama pada proses *data hiding* yaitu, pembacaan *payload*, pembacaan berkas audio, dan proses penyisipan *payload*. Antarmuka *data hiding* dapat dilihat pada gambar 4.1



Gambar 4.1 Antarmuka *Data Hiding*

Pada proses *data hiding*, dilakukan pemanggilan kelas *Parity* oleh kelas *MainForm*. Kelas *Parity* memiliki dua *method* yang berfungsi untuk melakukan penyisipan pesan, yaitu :

1. *Method* Penyisipan

Method Penyisipan berfungsi untuk pembentukan region sebanyak jumlah bit pesan pada berkas audio yang telah diubah ke dalam bentuk biner dan menyisipkan pesan ke dalam berkas audio. Dalam prosesnya, *method* Penyisipan melakukan pemanggilan *method* *HitungParityBit*.

2. *Method* HitungParityBit

Method HitungParityBit berfungsi menghitung nilai *parity* tiap region yang terbentuk.

4.2.1.1 Implementasi Pembacaan *Payload* (Pesan)

Tahap awal untuk melakukan penyisipan *payload* ke dalam berkas audio adalah melakukan pembacaan berkas teks berformat “.txt” atau dengan memasukkan *payload* secara langsung pada tempat yang disediakan. Kemudian, dilanjutkan dengan mengubah *payload* berupa teks tersebut ke dalam biner dan ditambahkan bit penanda yang berfungsi pada saat pengungkapan pesan. Proses pembacaan berkas “.txt” yang akan disisipkan dapat dilihat pada *source code* 4.1

```
String path = open.getSelectedFile().getPath().toString();
File text = new File(path);
try{
    isiPesan.setText("");
    BufferedReader inFile = new BufferedReader(new FileReader(text));
    String line;
    while ((line = inFile.readLine()) != null){
        isiPesan.append(line+"\n");
    }
    String temp = isiPesan.getText();
    isiPesan.setText(temp.substring(0, temp.length()-1));
    inFile.close();
}
```

Source code 4.1 Proses pembacaan berkas “.txt”

Proses pembacaan *payload* untuk *payload* yang dimasukkan secara langsung pada tempat yang telah disediakan dapat dilihat pada *source code* 4.2

```
sbPesan = new StringBuilder(isiPesan.getText());
binPesan = new StringBuilder();
```

Source code 4.2 Proses pembacaan *payload* yang dimasukkan secara langsung

Proses selanjutnya mengubah *payload* yang akan disisipkan ke dalam bentuk biner yang dapat dilihat pada *source code* 4.3

```
for (int i = 0; i < sbPesanan.length(); i++){
    int pesan = (int) sbPesanan.charAt(i);
    binPesanan.append(biner(Integer.toBinaryString(pesan)));
    binPesanan.append(biner(Integer.toBinaryString((int) '0')));
    binPesanan.append(biner(Integer.toBinaryString((int) 'a')));
```

Source code 4.3 Proses mengubah *payload* ke dalam bentuk biner

4.2.1.2 Implementasi Pembacaan Berkas Audio

Tahap kedua dari penyisipan *payload* adalah pembacaan berkas audio. Proses pertama adalah membaca *header* berkas audio untuk diambil informasi dan menampilkannya. Pembacaan *header* dapat dilihat pada *source code* 4.4

```
try{
    AudioInputStream inFile = AudioSystem.getAudioInputStream(new
    File(path));
    AudioFormat af = inFile.getFormat();
    Encoding format = af.getEncoding();
    int channel = af.getChannels();
    int bitPerSample = af.getSampleSizeInBits();
    float sampleRate = af.getSampleRate();
    int blockAlign = af.getFrameSize();
    float byteRate = sampleRate * channel *(bitPerSample/8);
    inFile.close();
}
```

Source code 4.4 Pembacaan *header* berkas audio

Setelah proses pembacaan *header* selesai, selanjutnya proses dilanjutkan dengan mengubah berkas audio terpilih ke dalam bentuk biner. Pengubahan berkas audio ke dalam bentuk biner dapat dilihat pada *source code* 4.5

try{
FileInputStream audio = new FileInputStream(path);
binWAV = new StringBuilder();
int count = audio.available();
for (int i = 0; i < count; i++){
binWAV.append(biner(Integer.toBinaryString(audio.read()))).append("/");
progress.setValue(((i+1)*100)/count);}
sizeMainData = ((count - 43) * 8)/16;
audio.close();
}

Source code 4.5 Pengubahan berkas audio ke bentuk biner

4.2.1.3 Implementasi Penyisipan *Payload* (Pesan)

Setelah *payload* dan berkas audio telah dimasukkan ke dalam aplikasi dan diubah ke dalam bentuk biner, proses selanjutnya adalah menyisipkan *payload* ke dalam berkas audio. Dibentuk region sebanyak 16 bit dari berkas audio sebanyak *bitPayload*. Dari setiap region yang terbentuk, dicari nilai *parity*-nya untuk dibandingkan dengan *bitPayload*. Pembentukan region dari berkas audio dan membandingkan nilai *parity* region dengan *bitPayload* dapat dilihat pada *source code 4.6*

1	int index1 = 43 * 9;
2	for (int i = 0 ; i < bin.length(); i++){
3	String temp = bin.charAt(i)+"";
4	String region1 = musik.substring(index1, index1+8);
5	String region2 = musik.substring(index1+9, index1+17);
6	String parity = HitungParityBit(region1+region2);
7	if(!parity.equals(temp)){
8	if(musik.charAt(index1+16)=='1'){
9	musik.setCharAt(index1+16, '0');
10	else{
11	musik.setCharAt(index1+16, '1');
12	}
13	index1 = index1+18;
14	}

Source code 4.6 Pembentukan region dan membandingkan *parity bit* region dengan *bitpayload*

Baris 1 menunjukkan bahwa data audio yang akan dimodifikasi dimulai pada *byte* ke-44. Pembentukan region ditunjukkan pada baris 4 dan 5. Kemudian dicari nilai *parity* dari region tersebut ditunjukkan pada baris 6. Dilanjutkan dengan perbandingan *parity* dengan *bitpayload* yang ditunjukkan pada baris 7 sampai dengan 12. Cara untuk mendapatkan nilai *parity* tiap region dapat dilihat pada *source code* 4.7

1	private String HitungParityBit(String region){
2	int count = 0;
3	for(int i=0;i<region.length();i++){
4	if(region.charAt(i)=='1'){
5	count++;}
6	}
7	if(count%2==0){
8	return "0";}
9	else{
10	return "1";} }

Source code 4.7 Mencari nilai *parity* tiap region

Baris 3 sampai dengan 6 adalah perhitungan jumlah bit “1” dari tiap region. Apabila ditemukan bit ”1” maka variabel *count* ditambahkan 1. Perulangan berlanjut sampai dengan bit terakhir region. Setelah didapatkan jumlah total bit “1” pada region, proses dilanjutkan dengan mencari nilai *parity* yang ditunjukkan pada baris 7 sampai dengan 10.

Proses terakhir dari rangkaian proses penyisipan pesan adalah pembentukan kembali berkas audio yang telah dimodifikasi. Pembentukan kembali berkas audio ditunjukkan *source code* 4.8

Parity p = new Parity();
String save = simpan.getSelectedFile().getPath().toString();
WAVname = save+".wav";
String [] byteBaru = (p.Penyisipan(binPesan, binWAV)).toString().split("/");
try{
File trashedSong = new File(save + ".wav");
FileOutputStream trash = new FileOutputStream(trashedSong);

```

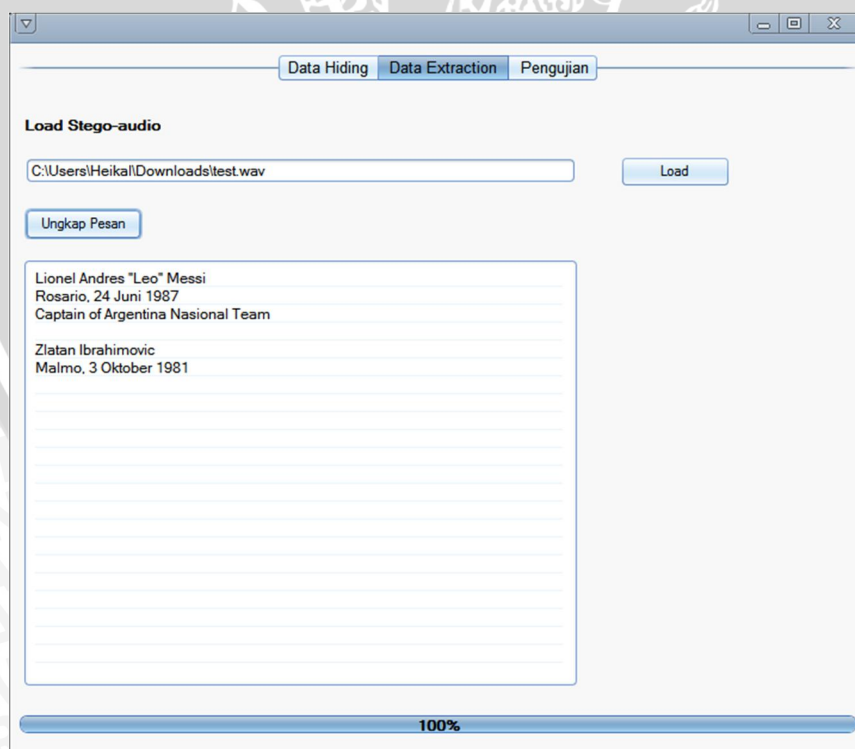
for(int i = 0; i < byteBaru.length; i++){
    trash.write(Integer.parseInt(byteBaru[i],2));
    progress.setValue(((i+1)*100)/byteBaru.length);
    trash.close();
}
catch(Exception e){
    System.out.println("Error â€" + e.toString());
}

```

Source code 4.8 Pembentukan berkas audio

4.2.2 Proses Data Extraction

Seperti halnya proses *data hiding*, proses *data extraction* merupakan bagian proses dari kelas *MainForm*. Proses *data extraction* merupakan proses pengungkapan pesan dari berkas audio (*stego-data*). Terdapat dua proses utama pada proses *data extraction* yaitu pembacaan berkas audio dan proses pengungkapan pesan. Pembacaan berkas *stego-data* yang terdapat pada proses *data extraction* sama dengan pembacaan audio pada proses *data hiding*. Antarmuka *data extraction* dapat dilihat pada gambar 4.2



Gambar 4.2 Antarmuka *Data Extraction*

Pada proses *data extraction*, dilakukan pemanggilan kelas *Parity* oleh kelas *MainForm*. Kelas *Parity* memiliki dua *method* yang berfungsi untuk melakukan pengungkapan pesan, yaitu :

1. *Method* Pengungkapan

Method Pengungkapan berfungsi untuk mengungkapkan pesan dengan cara mencari dan membentuk kembali pesan sampai bertemu dengan karakter penanda pada berkas audio *stego-data* tersebut. Dalam prosesnya *method* ini melakukan pemanggilan *method* *HitungParityBit*.

2. *Method* *HitungParityBit*

Method *HitungParityBit* berfungsi menghitung nilai *parity* tiap region yang terbentuk.

4.2.2.1 Implementasi Pengungkapan Pesan

Proses awal pada pengungkapan pesan adalah dengan memasukkan berkas audio *stego-data* ke dalam aplikasi dan diubah ke dalam bentuk biner. Dimulai pada byte ke-44, proses pengungkapan pesan dilakukan dengan cara membentuk kembali region-region. Setiap region akan dicari nilai *parity*-nya karena nilai *parity* merupakan bit pesan itu sendiri.

Nilai *parity* dikumpulkan sampai berjumlah 8 bit dan kemudian diubah ke dalam bentuk ASCII. Kemudian dilakukan pengecekan apakah nilai ASCII merupakan pesan atau karakter penanda. Apabila merupakan pesan maka nilai ASCII itu ditampung dan proses pengungkapan dilanjutkan, sedangkan apabila nilai ASCII tersebut merupakan bit penanda maka proses pengungkapan dihentikan. Proses pengungkapan pesan ditunjukkan *source code* 4.9

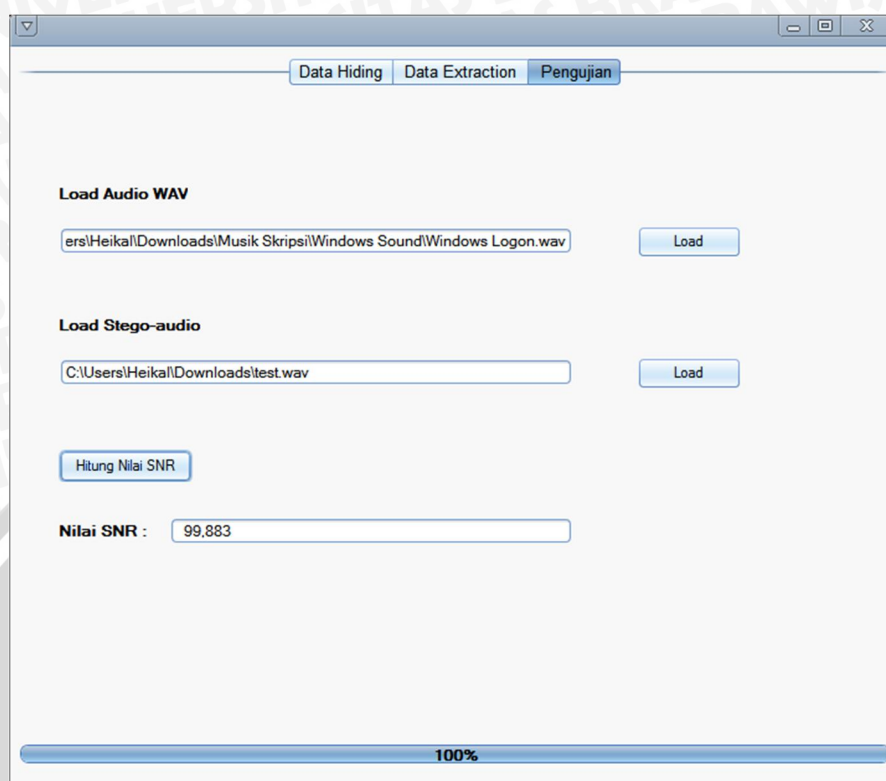
<code>stegoWAV = false;</code>
<code>bin = new StringBuilder();</code>
<code>StringBuilder musix = new StringBuilder(binWAV.toString());</code>
<code>int index1 = 43 * 9;</code>
<code>int N = 0;</code>
<code>String temp = "";</code>
<code>boolean fit = false;</code>
<code>int check = 0;</code>

do{
N++;
String region1 = musix.substring(index1, index1+8);
String region2 = musix.substring(index1+9, index1+17);
String parity = HitungParityBit(region1 + region2);
bin.append(parity);
temp = temp + parity;
if(N%8 == 0){
char cek = (char)(Integer.parseInt(temp, 2));
if(fit){
if(cek != '0'){
if(cek == 'æ'){
stegoWAV = true;
bin.replace(bin.length()-16, bin.length(), "");}
else{
fit = false;}}
else{
if (cek == '0'){
fit = true;}}
temp = "";
if(stegoWAV) break;
index1 = index1 + 18;
check = index1+18;
} while (check <= musix.length());

Source code 4.9 Proses pengungkapan pesan

4.2.3 Proses Pengujian

Proses pengujian yang terdapat pada aplikasi ini merupakan pengujian nilai *signal to noise ratio*. Proses pengujian ini dilakukan pada kelas *MainForm* seperti pada dua proses sebelumnya, yaitu proses *data hiding* dan proses *data extraction*. Pada proses pengujian ini hanya dilakukan dengan pemanggilan satu kelas, yaitu kelas SNR. Antarmuka pengujian dapat dilihat pada gambar 4.3



Gambar 4.3 Antarmuka penguian

4.2.3.1 Implementasi Penguian

Proses awal penguian untuk menghitung nilai *signal to noise ratio* adalah dengan memasukkan berkas audio yang belum dimodifikasi dan berkas audio *stego-data*. Kedua berkas audio tersebut dibandingkan, apabila jumlah bit kedua audio sama maka proses akan dilanjutkan dengan menghitung banyak bit terubah dari audio *stego-data*. Setelah jumlah total bit terubah didapatkan, maka penghitungan *signal to noise ratio* dapat dilakukan. Proses penghitungan nilai *signal to noise ratio* dapat dilihat pada *source code* 4.10

```
try{
    File audio1 = new File(pathAudio);
    File audio2 = new File(pathStego);
    FileInputStream inFile1 = new FileInputStream(audio1);
    FileInputStream inFile2 = new FileInputStream(audio2);
    jum1=inFile1.available();
    jum2=inFile2.available();
    if(jum1 == jum2){
```

```

for(int i=0; i<juml; i++){
    if(inFile1.read()!=inFile2.read()) count++;
    progress.setValue(((i+1)*100)/juml);}

    nilaiSNR = (((double)juml*8) -
    ((double)count*8))/((double)juml*8)*100;
}

else{
    JOptionPane.showMessageDialog(null, "Jumlah bit tidak sama",
    "Error", 0);}

    inFile1.close();
    inFile2.close();
}

```

Source code 4.10 Proses penghitungan nilai signal to noise ratio

