CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This project used java programming and Netbeans for the GUI. The program divides to 4 views. The first view is Home. This is a blank view when the program start running. The second view is Embedding. This is a view to embed message to an image. The third view is Extraction. This is a view to extract message from an image. The last view is About. This view shows the title of this project and another information of author.

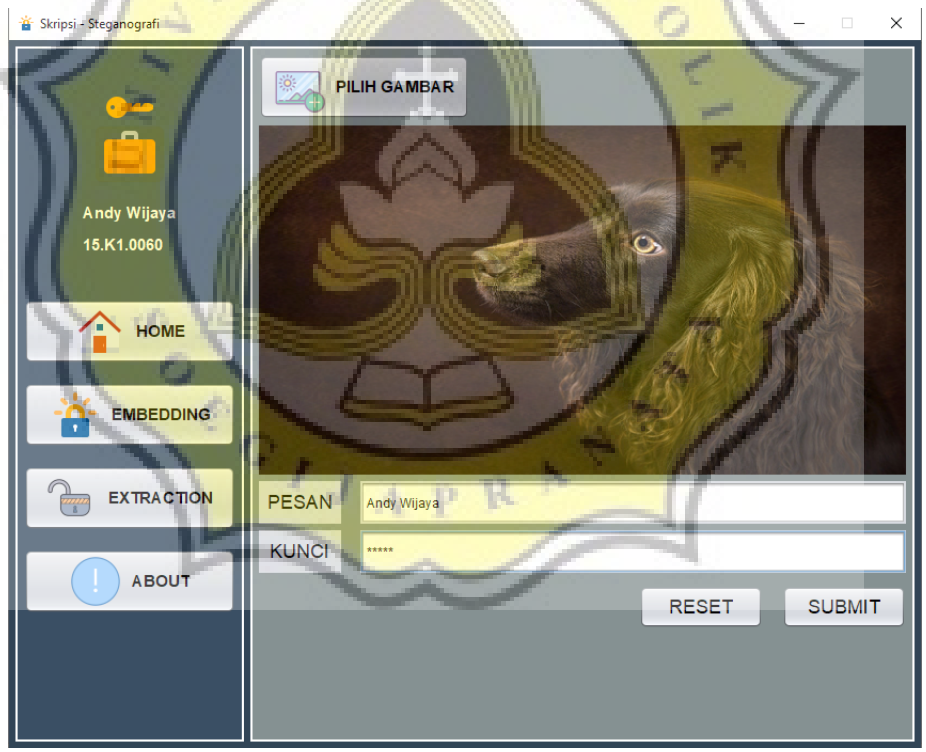

Illustration 5.1: Example Embedding

This is the view of Embedding section. User can choose image that will be the cover image with click on the Pilih Gambar button. Then fill the Pesan field to

input the message. After that input the key in Kunci field. The illustration above is the example that already input by user. Click on Submit button to embed the message into the image.

Below is the code will process:

```
private void BtnEmbeddingSubmitActionPerformed(java.awt.event.ActionEvent evt) {
    if(message.getText().equals("") || keyEmbedding.getText().equals(""))
    {
        JOptionPane.showMessageDialog(this,"Isi yang lengkap ya!","Peringatan",JOptionPane.ERROR_MESSAGE);
    }
    else
    {
        String kunci = keyEmbedding.getText();
        String ci = selectedpathEmbedding.getText(); //menangkap alamat file
        String pesan = message.getText()+"*888#"; //*888# sebagai penanda nanti untuk decrypt pesan
        int [] bit_pesan = bit(pesan);
        int [] bit_spread = Spread(bit_pesan); //spread pesan
        int hasilkunci = modKunci(kunci);
        int [] Pseudo = LCG(hasilkunci);
        int [] bit_lcg = biner_lcg(Pseudo);
        int [] hasil_spread = ModPesanLCG(bit_lcg,bit_spread);
        embed(hasil_spread,ci);
        valEmbedding.setVisible(false);
    }
}
```

Illustration 5.2: Embedding Code

The code above starts with checking the fields. If there a blank field in the program, a pop ups window will appear and will tell you to fill the blank field. If there is no blank field, then the embedding process will be start. First the program will read the key, the path of image, and the message. *888# will be added to the end of message as a marker. Then the program will convert message to binary. Then the converted message will be spread with 4 scales. After that, the program will XOR the characters of key. The result of XOR will be seed value for LCG Formula. Line 650 will be process seed value to LCG Formula to generate 4 random numbers. Then the program will convert this 4 random numbers to binary. After that, the program will XOR it with the result of spread message. Then the program will embed it to the cover image. The output of this process is image that has message inside it or can be called as stego image.

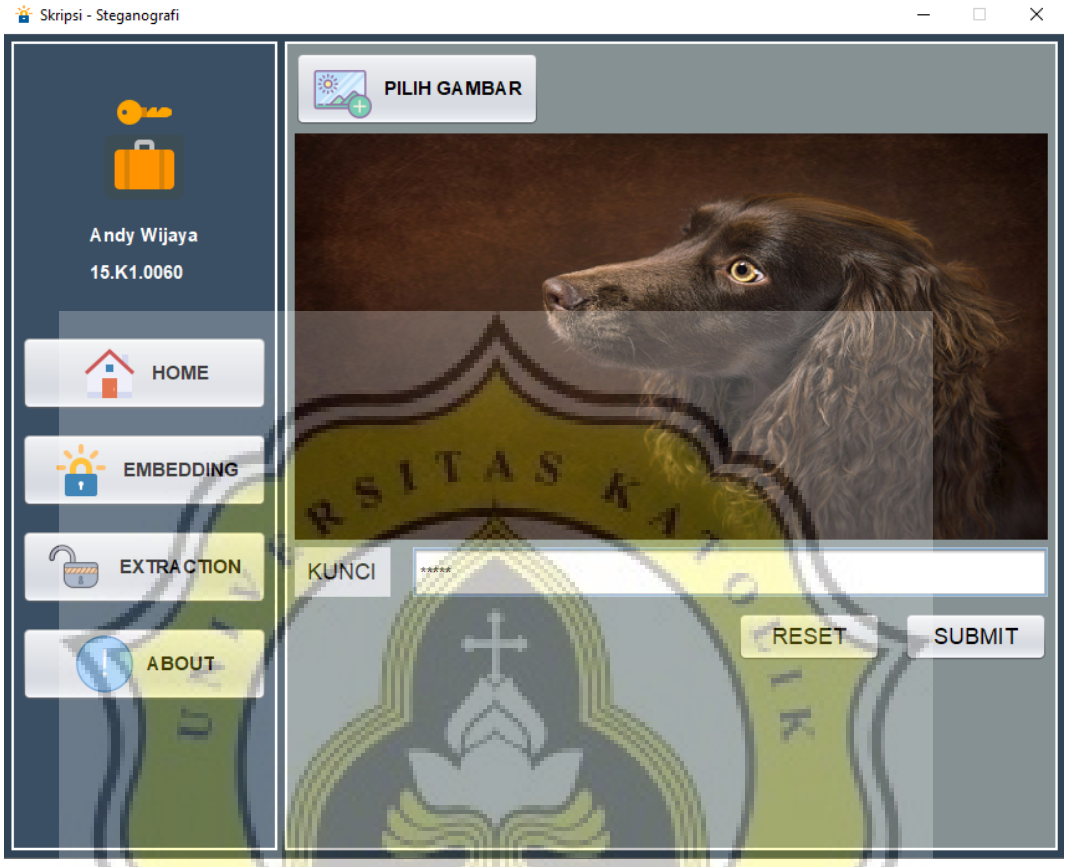

Illustration 5.3: Example Extraction

This is the view of Extraction section. User can choose image that has message inside it with click on the Pilih Gambar button. Then input the key on Kunci field. The illustration above is the example that already input by user. Click on Submit button to extract the message from the image.

Here is the code of extraction:

```
private void BtnExtractionSubmitActionPerformed(java.awt.event.ActionEvent evt) {
    if(keyExtraction.getText().equals(""))
    {
        valExtraction.setVisible(true);
    }
    else
    {
        valExtraction.setVisible(false);
        String si = selectedpathExtraction.getText();
        String pesan = Extract(si);
        String kunci = keyExtraction.getText();
        int hasilkunci = modKunci(kunci);
        int [] Pseudo = LCG(hasilkunci);
        int [] bit_lcg = biner_lcg(Pseudo);
        int [] hasilXOR = XOR(pesan,bit_lcg);
        Decrypt(hasilXOR);
    }
}
```

Illustration 5.4: Extraction Code

The code above starts with read the key and the path of image. Then program will extract all of LSB value from image to String pesan. After that, the program will XOR the characters of key. The result of XOR will be seed value for LCG Formula. LCG Formula will generate 4 random numbers. Then the program will convert this 4 random numbers to binary. After that, the program will XOR it with String pesan. The result will be despread with 4 scales. Convert the result to characters. When the program found *888# in the converted characters, the program will stop and shows the result of this process in a pop ups window.

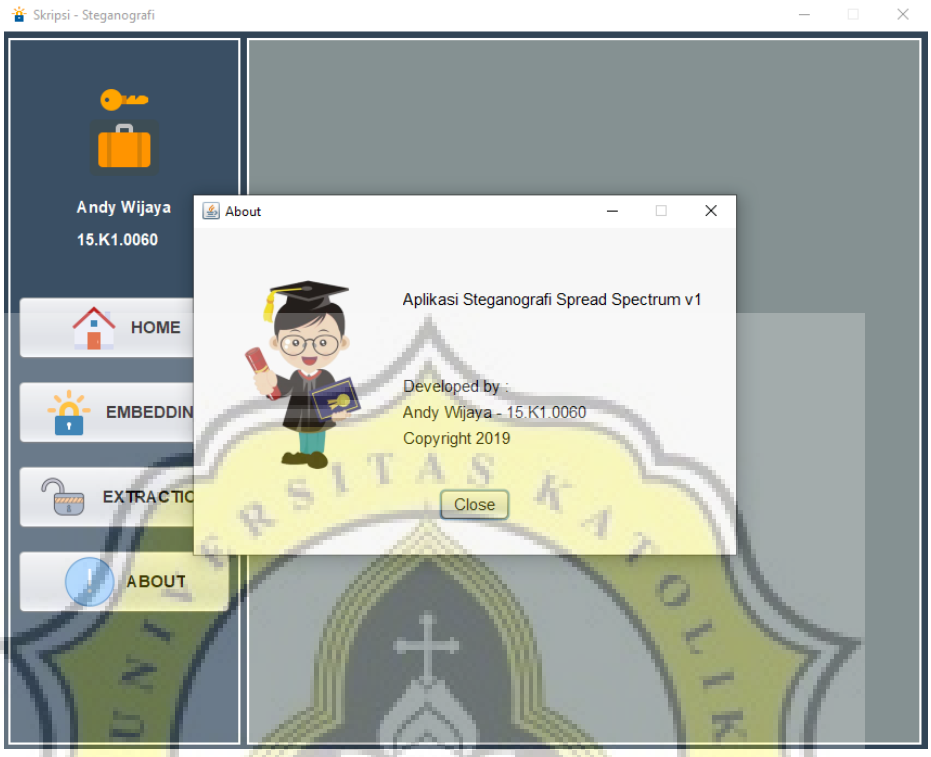

Illustration 5.5: About View

When user clicks on About Button, this pop ups window will appear. The view shows the title of this program and authors name. The title is *Aplikasi Steganografi Spread Spectrum v1*. And the authors name is Andy Wijaya.

5.2 Testing

This sub chapter will explain about the result of this research. There are 2 test that will be examine to the program. The first test is examine the program with different messages length and check the quality of output with parameter Peak Signal to Noise Ratio (PSNR). Then the second test is examine the program with different keys to decrypt the message.

1. Message Length

The program will be examine using several messages with 1 key: **unika**. This key will be assembled in 4 messages with different character length. Those are 11 characters, 191 characters, 684 characters and 960 characters and it will use 100 sample images with 320x277 pixel. This sample images will be embed by those different character length. Then the quality of the result images will be examine using Peak Signal to Noise Ratio (PSNR). The PSNR value measured in decibels (dB).

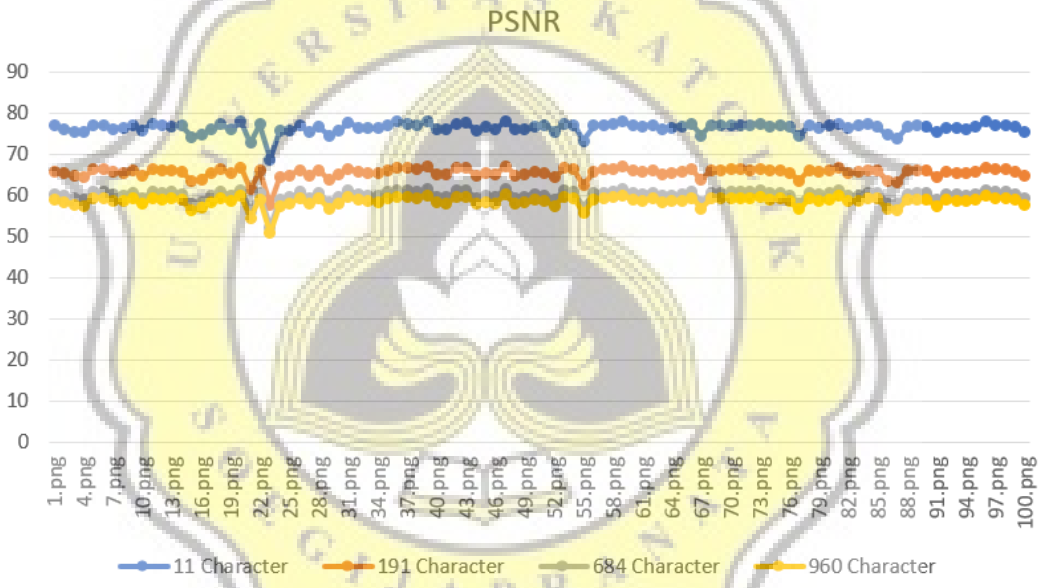

Illustration 5.6: PSNR result of testing different character length (higher is better)

The chart above shown the quality of 4 different messages length that examine above. It shown that the different of character length have impact on PSNR value. The message which have 11 characters have average of PSNR value around 68 dB to 78 dB. And the message which have 960 characters only have average of PSNR value around 50 dB to 60 dB. The longer character embed on the image will have lower PSNR value.

2. Different key

The program will be examine using several keys. Those are **andy, akinu, and aaaa**. First embed 10 sample images with message **uniqlo** and the key that used is **unika**. Then the result of test will be entered into the table with 2 columns. The first column shows the name of sample images. The second column shows the result of the examine.

1. Test 1 is examine the stego images with **andy** as the key:

Table 5.1: Test 1

Filename	**Result**
1_stegano.png	ÿäãûæå ²²²©Æ> iáꜿ¬Cꜿ^ ❿Ä Ïxï_5O9z❷⑨Uú²6❽#⑨#2ÁÊÔmT
2_stegano.png	ÿäãûæå ²²²©9999999999999y99999999999999999999999999
3_stegano.png	ÿäãûæå ²²²©#«p#«p#«p#«p#«p#T❹âT❹âT❹âT❹âT❹â T❹âT❹
4_stegano.png	ÿäãûæå ²²²©ÆÆKp#TFø•ý❸Á2④x⑥ÆÆÆ9#vÆÆ ÆÆÂcq.À##❾Ç9#v
5_stegano.png	ÿäãûæå ²²²©»låäꜿÍꜿ❿ã[ûTkzÈxwL:Ü#ZN❷B»❸JO⑧#a" Òj❶;
6_stegano.png	ÿäãûæå ²²²©⑦?Ù9*òÕ②=1
7_stegano.png	ÿäãûæå ²²²©#H#Ë$ðÉ«⑧ã# â# 3øoÉ#P#þ#1ìQwèÉS❼d©ꜿõå
8_stegano.png	ÿäãûæå ²²²©ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ
9_stegano.png	ÿäãûæå ²²²©#Uc
10_stegano.png	e#ꜿ¿<Û❶❷a6ùá❸m*Ò¿

The table 5.1 shown the result of decrypted stego images using **andy** as key. It shown the result of 10 examined images. All the result shown the false message.

2. Test 2 is examine stego images with **akinu** as the key:

Table 5.2: Test 2

Filename	Result
1_stegano.png	uniqlo
2_stegano.png	uniqlo
3_stegano.png	uniqlo
4_stegano.png	uniqlo
5_stegano.png	uniqlo
6_stegano.png	uniqlo
7_stegano.png	uniqlo
8_stegano.png	uniqlo
9_stegano.png	uniqlo
10_stegano.png	uniqlo

The table above shown the result of decrypted stego images using **akinu** as key. The key that used to decrypt in this test has same characters with the right key. So in this test, the key that used is **unika** with different characters position. All the result shown the true message.

3. Test 3 is examine stego images with **aaaa** as the key:

Table 5.3: Test 3

Filename	Result
1_stegano.png	uniqlo
2_stegano.png	uniqlo

3_stegano.png	uniqlo
4_stegano.png	uniqlo
5_stegano.png	uniqlo
6_stegano.png	uniqlo
7_stegano.png	uniqlo
8_stegano.png	uniqlo
9_stegano.png	uniqlo
10_stegano.png	uniqlo

The table 5.3 shown the result of decrypted stego images using **aaaa** as key. The key that used is very different with the right key. But all the result shown the true message.