

## CHAPTER 5

### IMPLEMENTATION AND TESTING

#### 5.1 Implementation

Results from the implementation of the Base64 and LSB (Least Significant Bit) algorithm programs:

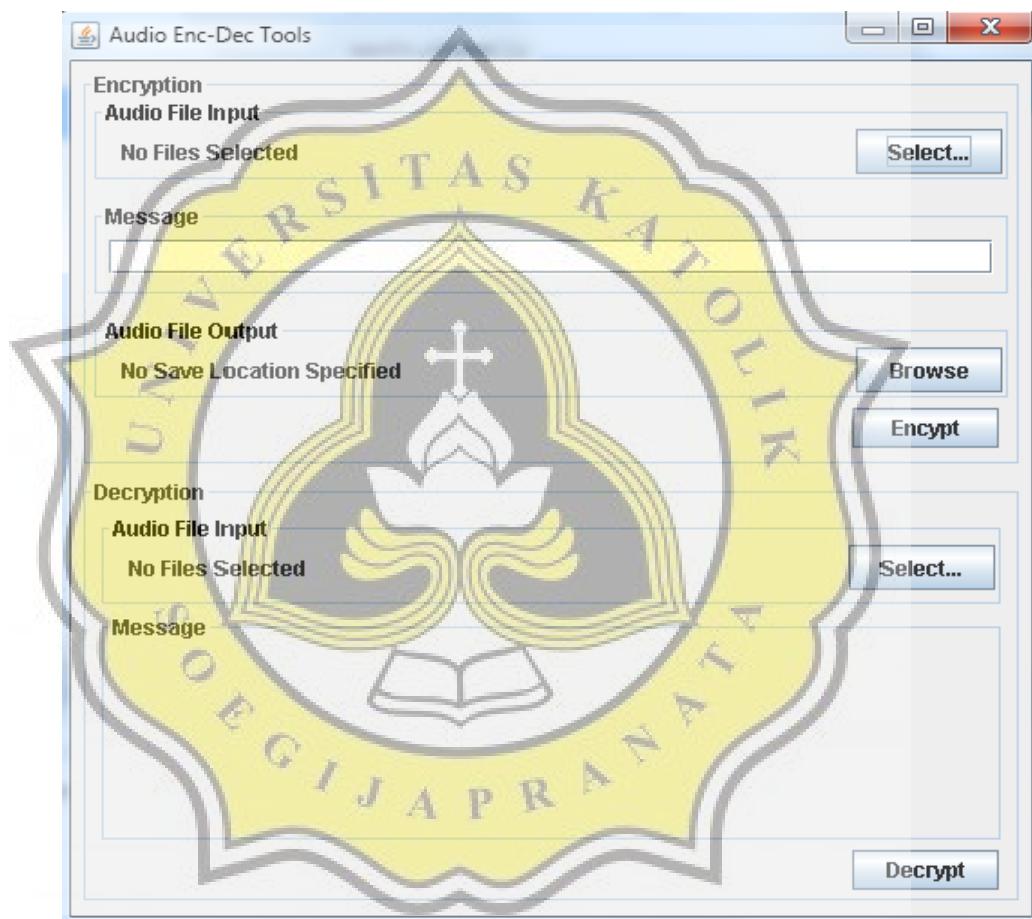


Illustration 5.1: Program Display

##### 5.1.1 Audio WAV Input

The following is the source code for entering WAV audio files :

1. `JLabel lbl_enc_audio_in = new JLabel("No Files Selected");`
2. `JButton button = new JButton("Select File");`
3. `button.addActionListener(new ActionListener() {`

```

4. public void actionPerformed(ActionEvent arg0) {
5. int result = fileChooser.showOpenDialog(contentPane);
6. if (result == JFileChooser.APPROVE_OPTION) {
7. File selectedFile = fileChooser.getSelectedFile();
8. audioInPathEnc = selectedFile.getAbsolutePath();
9. if(audioInPathEnc.contains(".wav")) {
10. lbl_enc_audio_in.setText(audioInPathEnc);
11. }else {
12. JOptionPane.showMessageDialog(mainFrame,
13. "Please pick a '.wav' file!",
14. "File Format Not Supported",
15. JOptionPane.WARNING_MESSAGE);
16. }
17. }
18. }
19. });

```

Line 1 is a command to display the selected WAV audio file. Line 2 is the command for the select WAV audio file button. Lines 3-10 are commands for the function of selecting the location of WAV audio files. Lines 11-15 are for validating WAV audio files.

### 5.1.2 Message Input

The following is the source code for entering messages to be encrypted :

```

1. JPanel panel_1 = new JPanel();
2. panel_1.setBounds(6, 76, 506, 55);
3. panel_2.add(panel_1);
4. panel_1.setBorder(new TitledBorder(null, "Message", TitledBorder.LEADING, TitledBorder.TOP, null, null));
5. panel_1.setLayout(null);
6. tf_message_in = new JTextField();
7. tf_message_in.setBounds(10, 22, 486, 20);
8. panel_1.add(tf_message_in);
9. tf_message_in.setColumns(10);

```

Lines 1-4 are commands for creating and locating the position of the "Message" form title. Lines 5-9 are commands for the input form and the location of the form.

### 5.1.3 Audio WAV Output

The following is the source code for storing WAV audio file output :

```

1. JPanel panel_3 = new JPanel();
2. panel_3.setToolTipText("");
3. panel_3.setBorder(new TitledBorder(UIManager.getBorder("TitledBorder.border"), "Audio File Output", TitledBorder.LEADING, TitledBorder.TOP, null, new Color(0, 0, 0)));
4. panel_3.setBounds(6, 142, 506, 49);
5. panel_2.add(panel_3);
6. JLabel lbl_save_loc = new JLabel("No Save Location Specified");
7. JButton btnBrowse = new JButton("Browse");
8. btnBrowse.addActionListener(new ActionListener() {
9. public void actionPerformed(ActionEvent arg0) {
10. JFrame parentFrame = new JFrame();
11. JFileChooser fileChooser = new JFileChooser();

```

```

12. fileChooser.setDialogTitle("Specify a file to save");
13. int userSelection = fileChooser.showSaveDialog(parentFrame);
14. if (userSelection == JFileChooser.APPROVE_OPTION) {
15. File fileToSave = fileChooser.getSelectedFile();
16. audioOutPath = fileToSave.getAbsolutePath();
17. lbl_save_loc.setText("Save as file: " + fileToSave.getAbsolutePath());
18. }
19. }
20. });

```

Lines 1-5 are commands for creating and positioning the "Audio File Output" form title position. Lines 6-20 are the command to make the "Browse" button and where to save the WAV audio file after encryption, if it does not save the WAV audio file after encrypting, there will be a message "No Save Location Specified".

### 5.1.2 Encryption

The following is the source code for encryption :

```

1. JButton btnNewButton = new JButton("Encrypt");
2. btnNewButton.addActionListener(new ActionListener() {
3. public void actionPerformed(ActionEvent arg0) {
4. if(audioInPathEnc == null || audioInPathEnc.isEmpty()) {
5. JOptionPane.showMessageDialog(mainFrame,
6. "Audio Input not Specified",
7. "Error",
8. JOptionPane.ERROR_MESSAGE);
9. return;
10. }
11. if(tf_message_in.getText().isEmpty()) {
12. JOptionPane.showMessageDialog(mainFrame,
13. "Message Could not Be Empty",
14. "Error",
15. JOptionPane.ERROR_MESSAGE);
16. return;
17. }
18. if(audioOutPath == null || audioInPathEnc.isEmpty()) {
19. JOptionPane.showMessageDialog(mainFrame,
20. "Audio Output not Specified",
21. "Error",
22. JOptionPane.ERROR_MESSAGE);
23. return;
24. }
25. try {
26. String message = tf_message_in.getText();
27. String encodedMessage = Base64.getEncoder().encodeToString
28. (message.getBytes());
29. System.out.println("Message: " + message);
30. System.out.println("Encoded: " + encodedMessage);
31. byte[] encodedMsgBytes = encodedMessage.getBytes();
32. byte[] formattedMsgBytes = new byte[8 + encodedMsgBytes.length];
33. byte[] encodedMsgLen = longToBytes(encodedMsgBytes.length);
34. for(int i = 0; i < 8; i++) {
35. formattedMsgBytes[i] = encodedMsgLen[i];
36. }
37. for(int i = 0; i < encodedMsgBytes.length; i++) {

```

```

38. formattedMsgBytes[8 + i] = encodedMsgBytes[i];
39. }
40. byte[] messageBits = new byte[(formattedMsgBytes.length * 8)];
41. for(int i = 0, bit_count = 0; i < formattedMsgBytes.length; i++)
{
42. for(int j = 0; j < 8; j++, bit_count++) {
43. messageBits[bit_count] = (byte) ((formattedMsgBytes[i] & (byte)
        Math.pow(2, j)) >> j);
44. }
45. }
46. WavFile wavIn = WavFile.openWavFile(new File(audioInPathEnc));
47. wavIn.display();
48. int numChannels = wavIn.getNumChannels();
49. long numFrames = wavIn.getNumFrames();
50. WavFile wavOut = WavFile.newWavFile(new File(audioOutPath),
51. numChannels,
52. numFrames,
53. wavIn.getValidBits(),
54. wavIn.getSampleRate());
55. long[] bufferIn = new long[100 * numChannels];
56. long[][] bufferOut = new long[numChannels][100];
57. int messageBitsRemaining = messageBits.length;
58. int framesRead;
59. do {
60. framesRead = wavIn.readFrames(bufferIn, 100);
61. long remaining = wavOut.getFramesRemaining();
62. int toWrite = (remaining > 100) ? 100 : (int) remaining;
63. for (int i = 0, k = 0; i < toWrite ; i++)
64. {
65. for(int j = 0; j < numChannels; j++, k++, messageBitsRemaining--)
{
66. if(messageBitsRemaining > 0) {
67. if((bufferIn[k] & 1) == 0) {
68. bufferIn[k] >= 1; bufferIn[k] <= 1; bufferOut[j][i] = bufferIn[k] +
        (long) messageBits[messageBits.length - messageBitsRe-
        maining];}else {
69. bufferOut[j][i] = bufferIn[k];
70. }
71. }
72. }
73. wavOut.writeFrames(bufferOut, toWrite);
74. } while (framesRead != 0);
75. wavIn.close();
76. wavOut.close();
77. String time_str = "Elapsed Time:\n" + (t_end-t_start) + " ns\n";
78. time_str += (t_end-t_start)/1000000 + " ms\n";
79. JOptionPane.showMessageDialog(mainFrame,
80. "Successfully Encrpyt the Message\n" + time_str,
81. "Done",
82. JOptionPane.INFORMATION_MESSAGE);
83. }catch (Exception e) {
84. JOptionPane.showMessageDialog(mainFrame,
85. "An Error Occured: " + e.getMessage(),
86. "Error",
87. JOptionPane.ERROR_MESSAGE);
88. }
89. }
90. });

```

Lines 1-24 are commands for creating and positioning the "Encrypt" button and validation before encryption. Lines 25-30 are commands for encrypting messages into the Base64 algorithm. Lines 31-39 are the process of converting messages into byte arrays. Lines 40-45 are the process of changing a byte array broken into bits. Lines 46-79 are the process bits in the WAV audio file replaced by the message bits in the LSB (Least Significant Bit) algorithm, as well as the process of reconstructing the WAV audio file after encrypting the message. Lines 80-93 are the process of calculating the encryption duration, as well as the display of the encryption time output.

### 5.1.2 Decryption

The following is the source code for decrypting WAV audio files :

```

1. JButton btn_decrypt = new JButton("Decrypt");
2. btn_decrypt.addActionListener(new ActionListener() {
3.     public void actionPerformed(ActionEvent arg0) {
4.         if(audioInPathDec == null || audioInPathDec.isEmpty()) {
5.             JOptionPane.showMessageDialog(mainFrame, "Error",
6.             "Audio Input not Specified",
7.             JOptionPane.ERROR_MESSAGE);
8.             return;
9.         }
10.        try {
11.            long[] bufferIn = new long[64];
12.            WavFile wavIn = WavFile.openWavFile(new File(audioInPathDec));
13.            int numChannels = wavIn.getNumChannels();
14.            wavIn.readFrames(bufferIn, 64 / numChannels);
15.            byte[] bytes = new byte[8];
16.            for(int i = 0, j = -1; i < 64; i++) {
17.                if( i % 8 == 0) j++;
18.                bytes[j] += (byte) (bufferIn[i] & 1) * Math.pow(2, i % 8);
19.            }
20.            long msgLength = bytesToLong(bytes);
21.            long[] bufferIn2 = new long[(int)msgLength * 8];
22.            wavIn.readFrames(bufferIn2, (int)msgLength * 8 / numChannels);
23.            byte[] msgBytes = new byte[(int)msgLength];
24.            for(int i = 0, j = -1; i < (int)msgLength * 8; i++) {
25.                if( i % 8 == 0) j++;
26.                msgBytes[j] += (byte) (bufferIn2[i] & 1) * Math.pow(2, i % 8);
27.            }
28.            String decMsg = new String(msgBytes);
29.            String realMsg = new String(Base64.getDecoder().decode(decMsg));
30.            lbl_message_out.setText("<html>" + realMsg.replace(" ", "<wbr>") +
31.                "</html>");
32.            wavIn.close();
33.            String time_str = "Elapsed Time:\n" + (t_end-t_start) + " ns\n";
34.            time_str += (t_end-t_start)/1000000 + " ms\n";
35.            JOptionPane.showMessageDialog(mainFrame,
36.             "Successfully Decrpyt the Message\n" + time_str,
37.             "Done",
38.             JOptionPane.INFORMATION_MESSAGE);
39.        }catch(Exception e) {
40.            JOptionPane.showMessageDialog(mainFrame,
41.             "An Error Occurred while Decrpyting the Message\n" + e.getMessage(),
42.             "Error", JOptionPane.ERROR_MESSAGE);
43.        }
44.    }
45. }

```

```

40. "An Error Occured: " + e.getMessage(),
41. "Error",
42. JOptionPane.ERROR_MESSAGE);
43. }
44. }
45. });

```

Lines 1-9 are commands for creating and positioning the "Decrypt" button's position and validation at decryption. Lines 10-31 are commands for decrypting WAV audio files, in which there is a process of converting LSB (Least Significant Bit) bits into byte arrays, byte arrays into Base64 messages and from Base64 into original messages. Lines 32-45 are the process of calculating the encryption duration, as well as the display of the encryption time output.

## 5.2 Testing

### 5.2.1 Encryption

These 10 WAV audio files will be tested, here are tables of the results of encryption with the same message in ms time :

Table 5.1: Table Encryption Time (in millisecond)

	Size	Duration	Quality	Message	Time
Jungle.wav	1.33 mb	15 detik	705 kbps	UNIKA Semarang	75 ms
Example_WAV_1.wav	1.02 mb	33 detik	256 kbps	UNIKA Semarang	27 ms
Example_WAV_5.wav	4.98 mb	29 detik	1411 kbps	UNIKA Semarang	128 ms
Example_WAV_10.wav	9.92 mb	58 detik	1411 kbps	UNIKA Semarang	251 ms
Gettysburg.wav	757 kb	17 detik	352 kbps	UNIKA Semarang	22 ms
BabyElephantWalk.wav	2.52 mb	60 detik	352 kbps	UNIKA Semarang	74 ms
PinkPanther30.wav	1.26 mb	30 detik	352 kbps	UNIKA Semarang	37 ms
Taunt.wav	89.1 kb	4 detik	178 kbps	UNIKA Semarang	5 ms
E02-ulaw.wav	4.56 mb	54 detik	705 kbps	UNIKA Semarang	139 ms
Afsp.wav	91.9 kb	2 detik	256 kbps	UNIKA Semarang	20 ms

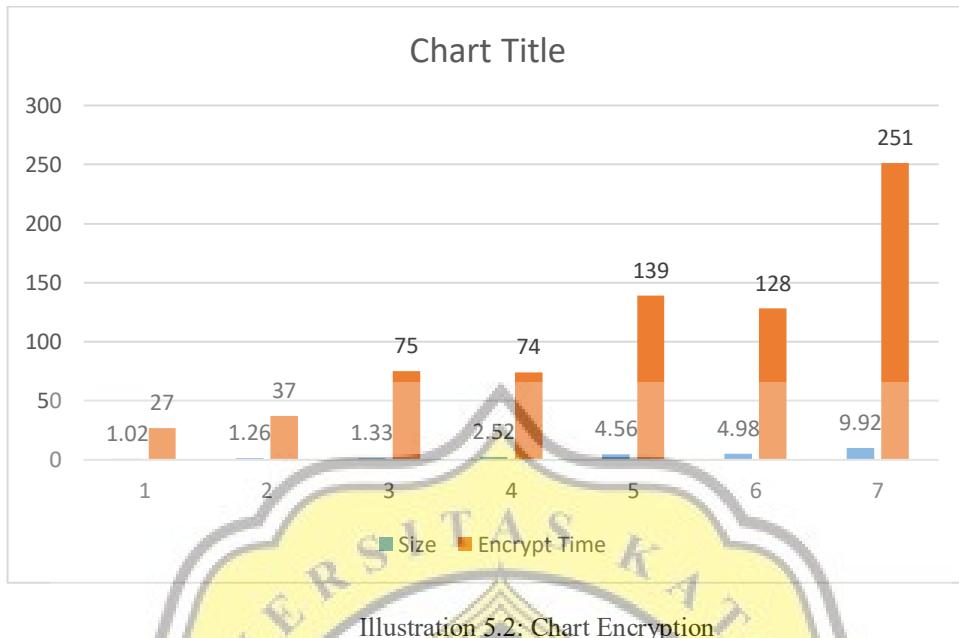


Illustration 5.2: Chart Encryption

Analysis of the results of testing the file encryption process on different WAV audio files with the same message. Shows a different time for each different WAV audio file size. Smaller WAV audio files have a longer time difference than large WAV audio files. This shows that the encryption time does not affect the length of the message character to be encrypted.

Table 5.2: Table Encryption Time (in millisecond)

	Size	Duration	Quality	Message	Time
Jungle.wav	1.33 mb	15 detik	705 kbps	a1	70
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2	76
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3	77
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4	76
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5	78
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6	92
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7	121
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7h8	77
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7h8i9	89

Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7h8i9j0	93
------------	---------	----------	----------	----------------------	----

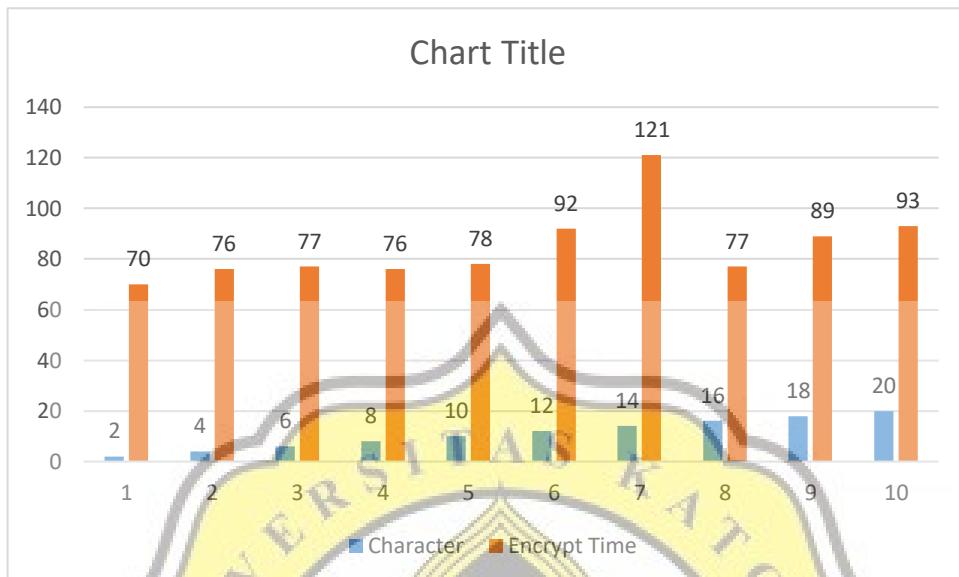


Illustration 5.3: Chart Encryption

Analysis of the results of the test file decryption process on the same WAV audio file with different messages. Indicates a different time at each message character length to be encrypted. Shorter characters have longer time than longer characters. This shows that the encryption time does not affect the length of the message character to be encrypted.

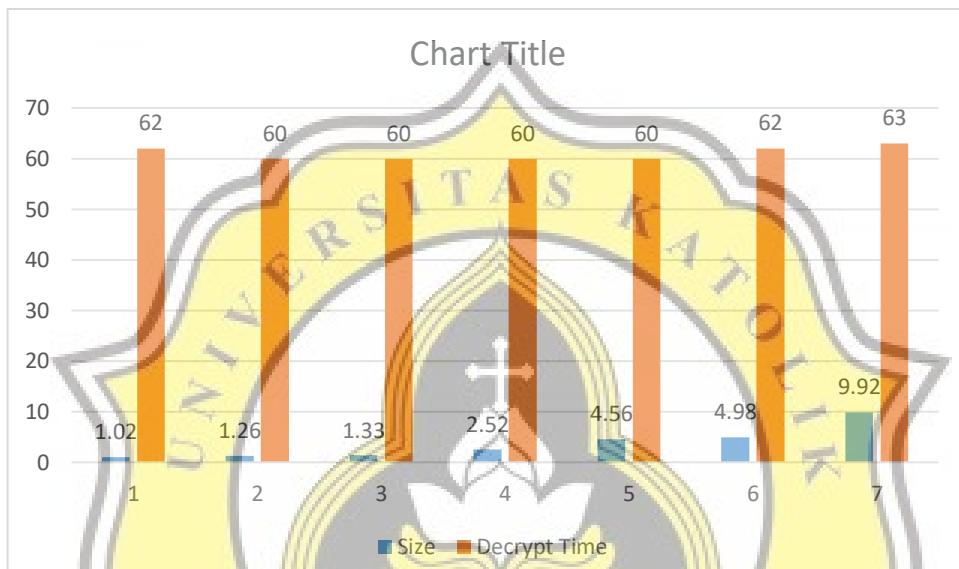
### 5.2.2 Decryption

Following is the decryption time table in ms :

Table 5.3: Table Decryption Time (in millisecond)

	Size	Duration	Quality	Message	Time
Jungle.wav	1.33 mb	15 detik	705 kbps	UNIKA Semarang	60 ms
Example_WAV_1.wav	1.02 mb	33 detik	256 kbps	UNIKA Semarang	62 ms
Example_WAV_5.wav	4.98 mb	29 detik	1411 kbps	UNIKA Semarang	62 ms
Example_WAV_10.wav	9.92 mb	58 detik	1411 kbps	UNIKA Semarang	63 ms
Gettysburg.wav	757 kb	17 detik	352 kbps	UNIKA Semarang	63 ms
BabyElephantWalk.wav	2.52 mb	1 menit	352 kbps	UNIKA Semarang	60 ms

PinkPanther30.wav	1.26 mb	30 detik	352 kbps	UNIKA Semarang	60 ms
Taunt.wav	89.1 kb	4 detik	178 kbps	UNIKA Semarang	62 ms
E02-ulaw.wav	4.56 mb	54 detik	705 kbps	UNIKA Semarang	60 ms
Afsp.wav	91.9 kb	2 detik	256 kbps	UNIKA Semarang	61 ms

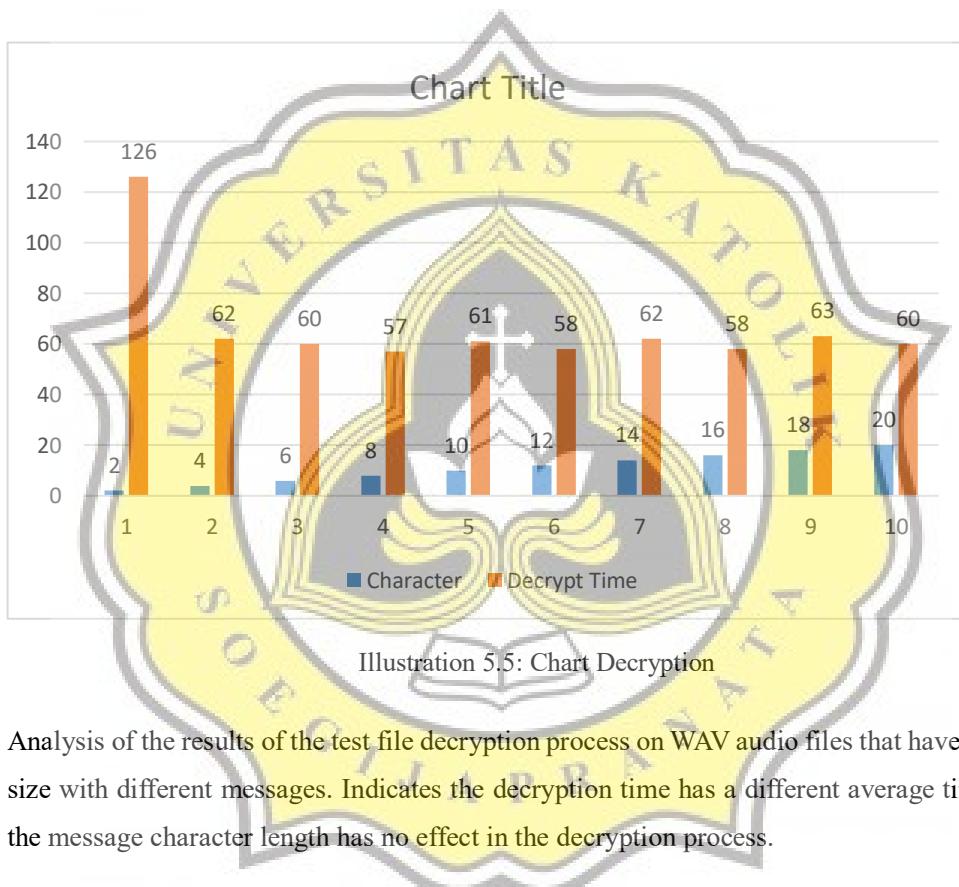


Analysis of the results of the test file decryption process on different sized WAV audio files with the same message. Show the decryption time has the same average time as a small time difference. Then the size of the WAV audio file has no effect in the decryption process.

Table 5.4: Table Decryption Time (in millisecond)

	Size	Duration	Quality	Message	Time
Jungle.wav	1.33 mb	15 detik	705 kbps	a1	126
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2	62
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3	60
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4	57
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5	61

Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6	58
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7	62
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7h8	58
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7h8i9	63
Jungle.wav	1.33 mb	15 detik	705 kbps	a1b2c3d4e5f6g7h8i9j0	60



Analysis of the results of the test file decryption process on WAV audio files that have the same size with different messages. Indicates the decryption time has a different average time. Then the message character length has no effect in the decryption process.