Combination of Blowfish Algorithm First of File Algorithm and End of File Algorithm on Image Security

Dian Rachmawati*, Mohammad Andri Budiman*, and M Fattahurrahman H N*
1Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Medan 20155, Indonesia

*Email: dian.rachmawati@usu.ac.id, mandrib@usu.ac.id, fathurnasution21@gmail.com

Abstract. In this age, there are many types of information. An image is a form of information that can be sent from anyone to anyone in various ways. But in the shipping process, there is often a hijacking or theft by an unwanted party. So that information must be secured by cryptography to avoid this. To add a sense of security between the sender and recipient, cryptography will be combined with steganography. In this study, the Blowfish Algorithm was combined with First Of File and End Of File steganography methods. Information that has been encrypted will be inserted into other media. Likewise for the recipient, can extract information then decrypt information.

Based on experiments conducted on the Blowfish algorithm, the time of encryption and decryption needed when the resolution of the image gets larger is linearly proportional to the value. MSE gets smaller, and the PSNR value gets higher, it shows better image quality. Cover image will increase in size after the embedding process so that it becomes a stego image.

1. Introduction

The image was a form of data or information that exist in visual form. Nowadays, information or data transmission could be done by anyone and anyway. However, not all of the information tended to be common. There were classified messages which the originality must be kept along the transmission. A way to secure the data could be done by using cryptography and steganography.

Cryptography comes from the words crypto and graphia. Crypto means secret, and graphia means writing[1]. Thus, cryptography is the science of maintaining the security of messages when messages are sent from one place to another place. The Blowfish algorithm was created by a cryptanalyst named Bruce Schneier and published in 1994[2]. Blowfish is a 64-bit cipher block with variable key lengths and consists of two parts: key expansion and data encryption [3].

Steganography is the development of cryptography. Steganography is the hiding of messages in a storage medium and is not scrambling the contents of the file [4]. Steganography protects messages from unwanted people so that others can only see stego images without knowing the secret message[5]. There are several methods in steganography such as End of File (EOF) and First of File (FOF). In the End of File (EOF) method, the message is inserted at the end of the value. While First of File (FOF), the message is inserted at the beginning of the value [7].

2. Method

In this implementation there are 4 main menus, they are the blowfish encryption, FOF, and EOF Embedding, FOF and EOF extraction and blowfish decryption.
2.1 Encryption

The passkey used in this system test is "password" where the passkey will be processed in the Expansion Key on Blowfish. Passkey "password" will be XORed with P-Array which consists of 18 32-bit subkeys as follows:

Change "password" to the binary number so that we can get a binary from "password" is: 01110000 01100001 01110011 01110011 01110111 01101111 01110010 01100100

XOR the initial 32 bits of the passkey with P [0] from the subkey and the next 32 bits with P [1] from the subkey repeat the process until all the P-arrays are XOR.

| P[0] | 01110000 01100001 01110011 01110011 |
|-----|-------------------------------------|
| ⊕   | 00100100 00111111 01101010 10001000 |
|     | 01010100 01011110 00011001 11111011 |

| P[1] | 01110111 01101111 01110010 01100100 |
|-----|-------------------------------------|
| ⊕   | 01000101 10100011 00001000 11010011 |
|     | 11100100 11001000 01111010 10110111 |

| P[2] | 01110000 01100001 01110011 01110011 |
|-----|-------------------------------------|
| ⊕   | 00010011 00011001 10001010 00101110 |
|     | 01100111 01111000 01110011 01001000 |

| P[3] | 01110111 01101111 01110010 01100100 |
|-----|-------------------------------------|
| ⊕   | 00101001 11011111 00110001 11010000 |
|     | 11101000 01010001 11110010 10010001 |

| P[4] | 01110000 01100001 01110011 01110011 |
|-----|-------------------------------------|
| ⊕   | 11010100 01101000 01001011 01010001 |
|     | 11101110 11110000 01000011 10110100 |

| P[5] | 01110111 01101111 01110010 01100100 |
|-----|-------------------------------------|
| ⊕   | 00101001 10011111 00110001 11010000 |
|     | 01110000 01100001 01110011 01110011 |

| P[6] | 01110010 00101110 11111010 10011000 |
|-----|-------------------------------------|
| ⊕   | 01111000 01001111 10001001 11101011 |
|     | 01110000 01100001 01110011 01110011 |

| P[7] | 11101100 01001100 01101100 01100001 |
|-----|-------------------------------------|
| ⊕   | 01010101 01100011 01110011 01101001 |
|     | 11010111 01101111 01110010 01100100 |

| P[8] | 01110000 01100001 01110011 01110011 |
|-----|-------------------------------------|
| ⊕   | 01000101 00101000 01000001 11100010 |
|     | 00110101 01001001 01010010 10010101 |

| P[9] | 11110111 01110111 01110010 01001000 |
|-----|-------------------------------------|
| ⊕   | 00111000 11010000 00010011 01111011 |
|     | 01001111 10111111 01100001 00010011 |
From the calculation above, the new key-key value is obtained, then the encryption string is entirely zero with the subkey that has been obtained from the process above, then replace all P-arrays with the result that the string encryption is entirely zero so that the output is continuously changing. So we can get

\[
\begin{align*}
P[0] &= 01110000 01100001 01110011 01110011 \\
P[1] &= 01110111 01101111 01110010 01100100 \\
P[2] &= 00110100 11101001 00001100 01101100 \\
P[3] &= 01100011 10000110 01111100 00000100 \\
P[4] &= 01110000 01100001 01110011 01110011 \\
P[5] &= 01110111 01101111 01110010 01100100 \\
P[6] &= 00110100 11101001 00001100 01101100 \\
P[7] &= 01110000 01100001 01110011 01110011 \\
P[8] &= 01110111 01101111 01110010 01100100 \\
P[9] &= 00110100 11101001 00001100 01101100 \\
P[10] &= 01110000 01100001 01110011 01110011 \\
P[11] &= 01110111 01101111 01110010 01100100 \\
P[12] &= 00110100 11101001 00001100 01101100 \\
P[13] &= 01110000 01100001 01110011 01110011 \\
P[14] &= 01110111 01101111 01110010 01100100 \\
P[15] &= 00110100 11101001 00001100 01101100 \\
P[16] &= 01110000 01100001 01110011 01110011 \\
P[17] &= 01110111 01101111 01110010 01100100 \\
P[18] &= 00110100 11101001 00001100 01101100
\end{align*}
\]
After getting the value in the subkey, we will encrypt the plain image, for the RGB value of the initial pixel of the plain image, this test is: 76 54 77 53 51 86 61
1. Convert the pixel value that will be encrypted to binary so that it becomes:

\[
P[16] = 001000010111111001001111101110101
\]
\[
P[17] = 001111111000001100001111100111111
\]

2. Distribute the string into two parts \( x_L \) and \( x_R \), so it becomes:

\[
x_L = 01001100010001001110011010000100
\]
\[
x_R = 001101010011101011000111010101011
\]

3. \( x_L = x_L \oplus P_0 \)

\[
01001100 00110110 00110010 01001101
\]
\[
00100010 10000010 01011111 00101010
\]

\[
01101110 10110100 01101101 01100111
\]
\[
x_R = F(x_L) \oplus x_R
\]
\[
10110010 01110111 11010101 01000000
\]
\[
00110101 00110111 01010110 00111110
\]

Where is the function \( F \): Divide \( x_L \) into 4 8-bit sections: \( a, b, c, \) and \( d \)

\[
F(x_L) = ((S1, a + S2, b \mod 2^{32}) \oplus S3, c) + S4, \mod 2^{32}
\]

This cycle is repeated up to \( P_{17} \), so the encrypted pixel results are:

\[
173 255 251 255 255 244 173
\]

2.2 Embedding

In this process, the plain image that has been encrypted into a cipher image (cipher.bmp) will be inserted into the cover image measuring 500x500 pixels. The cipher image will be divided into two parts so that one section is inserted in the top pixel of the image and one part is inserted into the bottom pixel of the image. So that it produces a stego image.

| Table 1. Table of Some Cover Pixel Values Before Embedding |
|------------------|------------------|------------------|------------------|------------------|------------------|
| \( R = 37 \) | \( R = 43 \) | \( R = 41 \) | \( R = 40 \) | \( R = 43 \) | \( R = 42 \) | \( R = 44 \) |
| \( G = 22 \) | \( G = 33 \) | \( G = 35 \) | \( G = 34 \) | \( G = 36 \) | \( G = 34 \) | \( G = 36 \) |
| \( B = 20 \) | \( B = 19 \) | \( B = 15 \) | \( B = 17 \) | \( B = 20 \) | \( B = 17 \) | \( B = 18 \) |
| R = 67 | R = 72 | R = 93 | R = 52 | R = 50 | R = 68 | R = 78 |
| G = 35 | G = 35 | G = 58 | G = 29 | G = 27 | G = 31 | G = 38 |
| B = 30 | B = 24 | B = 43 | B = 28 | B = 19 | B = 31 | B = 30 |

| Table 2. Table of Some Cover Pixel Values After Embedding |
|------------------|------------------|------------------|------------------|------------------|------------------|
| \( R = 173 \) | \( R = 255 \) | \( R = 173 \) | \( R = 255 \) | \( R = 255 \) | \( R = 207 \) | \( R = 41 \) |
| \( G = 255 \) | \( G = 255 \) | \( G = 31 \) | \( G = 255 \) | \( G = 228 \) | \( G = 255 \) | \( G = 255 \) |
| \( B = 251 \) | \( B = 244 \) | \( B = 116 \) | \( B = 204 \) | \( B = 98 \) | \( B = 159 \) | \( B = 214 \) |
| \( R = 37 \) | \( R = 43 \) | \( R = 41 \) | \( R = 40 \) | \( R = 43 \) | \( R = 42 \) | \( R = 44 \) |
| \( G = 22 \) | \( G = 33 \) | \( G = 35 \) | \( G = 34 \) | \( G = 36 \) | \( G = 34 \) | \( G = 36 \) |
| \( B = 20 \) | \( B = 19 \) | \( B = 15 \) | \( B = 17 \) | \( B = 20 \) | \( B = 17 \) | \( B = 18 \) |
2.3 Extracting
In this process, the stego image will be extracted so that the cipher image will be removed from the cover image. Cipher image will return to its original size as before Embedding.

Table 3 Table of Some Stego Image Pixel Values

| R   | G   | B   | R   | G   | B   | R   | G   | B   | R   | G   | B   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 173 | 255 | 255 | 255 | 255 | 207 | 41  |
| 255 | 255 | 31  | 228 | 255 | 255 |
| 251 | 244 | 116 | 204 | 159 | 214 |
| 37  | 43  | 41  | 43  | 42  | 44  |
| 22  | 33  | 36  | 34  | 34  | 36  |
| 20  | 19  | 15  | 17  | 17  | 18  |

In this test stego image is the result of insertion from cipher.bmp to sunlight.bmp. Then after the Extracting process, cipher.bmp will return separately from sunlight.bmp.

Table 4 Table of Some Cover Pixel Values After Extracting

| R   | G   | B   | R   | G   | B   | R   | G   | B   | R   | G   | B   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 67  | 72  | 93  | 52  | 50  | 68  | 78  |
| 35  | 35  | 58  | 29  | 27  | 31  | 38  |
| 30  | 24  | 43  | 28  | 19  | 31  | 30  |
| 255 | 2  | 219 | 255 | 20  | 51  | 239 |
| 146 | 15  | 26  | 167 | 79  | 29  | 122 |
| 84  | 62  | 67  | 60  | 60  | 46  | 61  |

2.4 Decryption
In this process, the cipher image that has been generated from the extracting process will be decrypted. So that the cipher image will return to the plain image as before. The decryption process is the same as doing blowfish encryption, only begins with P[17] and ends with P[0]. The key used is "password."

Decryption steps are the same as encryption only if encryption starts from P[0] then the decryption will start from P[17]. The results of key expansion in this decryption process are

P[17] = 00010011000100100010001001000010110000110010001110010100101010101000010010001100100010000101010100001000010001000010000100100101001000100100001000010001000010000100010000100001000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001000010000100001
After getting the value in the subkey, we will decrypt the cipher image, for the RGB value of the initial pixel of the cipher image, this test is: 173 255 251 255 255 244 173 31
1. Convert the pixel value that will be encrypted to binary so that it becomes:

$$1010110111111111111111101111111111111111111111010010110100011111$$

Distribute the string into two parts, $x_L$, and $x_R$ so that it becomes:

$$x_L = 1010110111111111111111101111111111111111111111010010110100011111$$

$$x_R = 111111111111010010101101000011111$$

2. $x_L = x_L \oplus P_{17}$

$$10101101 11111111 11111011 11111111$$

$$00010011 00101011 00001001 00011001$$

$$111111111111010010101101000011111$$

$x_R = F(x_L) \oplus x_R$

$$10011011 01101010 11011011 11001100$$

$$01101111 10010010 10101101 00011111$$

This cycle is repeated up to P0, so the decrypted pixel results are:

76 54 50 77 53 51 86 61

3. Results and Discussions

The experiments were performed on ASUS X452C with Intel-Core i3 processor and 2GB RAM. This study uses Eclipse Java Oxygen and uses the Java programming language. The results of the experiments for each set are presented in Tables 5 and 6.

| No. | Key Length | Resolution (Pixel) | Time (ms) |
|-----|------------|--------------------|-----------|
| 1   | 8          | 100 x 10           | 74.2      |
| 2   | 8          | 100 x 20           | 124.4     |
| 3   | 8          | 100 x 30           | 180.4     |
| 4   | 8          | 100 x 40           | 249       |
| 5   | 8          | 100 x 50           | 308.8     |
In table 5, the graph of the results of encryption testing with variations in plain image resolution shows the greater the image resolution, so encryption process longer, if the image resolution smaller, the encryption process time be faster.

| No. | Key Length | Resolution(Pixel) | Time(ms) |
|-----|------------|-------------------|----------|
| 1   | 8          | 100 x 10          | 1181.2   |
| 2   | 8          | 100 x 20          | 1308.8   |
| 3   | 8          | 100 x 30          | 1217.6   |
| 4   | 8          | 100 x 40          | 1570.4   |
| 5   | 8          | 100 x 50          | 1584.8   |

In table 6, the graph of the results of decryption testing with variations in cipher image resolution shows the greater the image resolution, so decryption process longer, if the image resolution smaller, the decryption process time be faster.

4. Conclusions
Based on the experiments conducted in this study, it can be concluded that the process of running time of encryption and decryption blowfish algorithm is time needed when the resolution of the image gets bigger is relatively stable.

5. Acknowledgments
The authors gratefully acknowledge that the present research is supported by Fund Dissemination IPTEKS Research Results for Lecturers / Researchers Universitas Sumatera Utara.

References
[1] Ariyus, Dony 2008 Pengantar Ilmu Kriptografi Teori, Analisis, dan Implementasi Andi Offset Yogyakarta
[2] Landge, Irfan, Contractor, Burhanuddin, Patel, Aamna, and Choudhary, Rozina 2012 Image Encryption and Decryption using blowfish Algorithm World Journal of Science and Technology Vol 2 No 3 hal 151-156
[3] Schneier, Bruce 1996 Applied Cryptography 2nd John Wiley & Sons
[4] Wandani, Henny, Budiman, Mohammad Andri, dan Sharif, Amer 2012 Implementasi Sistem Keamanan Data dengan Menggunakan Teknik Steganografi End of File (EOF) dan Rabin Public Key Cryptosystem Alkhawarizmi Vol 1 no.1
[5] D. Rachmawati and M. A. Budiman 2017 New approach toward data hiding by using affine cipher and least significant bit algorithm 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT) Kuta Bali 2017 pp 1-6. doi: 10.1109/CAIPT.2017.8320737
[6] Dian Rachmawati and Yeni Rosalin Munthe 2018 J. Phys.: Conf. Ser. 1090 012062