Security of Image File with Tiny Encryption Algorithm And Modified Significant Bit Pseudo Random Number Generator

S M Hardi, R Syah Ramadhani, E. Muisa Zamzami, JT Tarigan, I Jaya

Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan 20155, Indonesia

Email: vani.hardi@usu.ac.id

Abstract. One of the negative impacts of the development of the era of digital technology on communication media today is the damage and loss of information data. Access to communication media is a means of delivering information that is vulnerable to manipulation. Therefore, cryptography is used to keep the information confidential. Cryptography is an art in encoding messages into other forms so that the message cannot be read by unauthorized parties. Tiny Encryption Algorithm (TEA) algorithm is a symmetric cryptographic algorithm in which the key is divided into four subsections and uses 128 key bits and operates 64 bits of data blocks and then divided into two blocks into 32 bits during the encryption process. In order to make the encryption stronger, it is combined with steganography where the steganography method used is Least Significant Bit (LSB) which has the function of inserting data into the last bit of other data. Linear Congruential Generator (LCG) is a random number generator in the LSB method where its function is as a flow of inputting data that has been encrypted randomly into another data as a cover so that the combination of these methods becomes Modified Least Significant Bit Pseudo Random Number Generator. The combination of cryptography and steganography is able to create a security in sending a confidential message to be safer.

1. Introduction

Image is something that is embedded in a two-dimensional plane. Image is also a form of digital information that is very often used today. However so many part of people or organization misappropriates irresponsibility information. Therefore, information in the form of digital images that are private and confidential must be properly secured. In the activity of exchanging information through cyberspace can pose security risks in confidential information. For this reason, file security is needed by using one of the cryptographic methods that can secure image messages, namely the Tiny Encryption Algorithm (TEA) where the encoding algorithm is used to encode the message to be sent (plaintext) to a message that has been encrypted (ciphertext). Tiny Encryption Algorithm (TEA) is a cryptographic algorithm that can be used to secure image data with a symmetrical key that can minimize memory usage and maximize speed [1]. Tiny Encryption Algorithm is used to encrypt data that is to be kept confidential. So that the file does not appear to have been encrypted, then the file will then be inserted into the cover file using the steganography process.). Based on the above background, how to make the process of securing the image by implementation a Tiny Encryption Algorithm combined with the Modified Least Significant Bit Pseudo Random Number Generator that has a function for inserting a message that has been encrypted into the image of the cover randomly so that the message is secure. There are currently some publications about image encoding without changing pixel values [5].
2. Method
This research uses Tiny Encryption algorithm and the Modified Least Significant Bit Pseudo Random Number Generator for securing image.

2.1 Tiny Encryption Algorithm
Tiny Encryption Algorithm (TEA) is a block cipher designed with advantages in speed and simplicity. The TEA uses 128-bit keys and operates 64-bit data blocks which are then divided into two 32-bit p blocks during the encryption process[2].

2.2 Modified Least Significant
For improving the effectiveness and safety of the LSB method, it is necessary to modify the method. Pseudorandom number generator (PRNG) is a mathematical function that generates random numbers [3][4]. PRNG is used to randomize the channel at each pixel of the picture to be inserted a message. In this study, the PRNG used was a Linear Congruential Generator. Linear Congruential Generator is a random number generation method expressed by the formula below:

\[ Z_i = (aZ_{i-1} + c) \mod m \]

Where :
- \( Z_i \) = i random number from the sequence
- \( aZ_{i-1} \) = previous random number
- \( a \) = multiplier factor
- \( c \) = increment
- \( m \) = modulus

3. Results and Discussions
The experiment from these researches using image file with 100 x 100 pixels, 200 x 200 pixels and 300 x 300 pixels size. Cover image used is 1000 x 1000 pixels, 1500 x 1500 pixels, 2000 x 2000 pixels size. The Display of the results of the encryption-Embedding and extraction-decryption process can be seen in these below process.

Figure 1. Illustration of RGB 1 x 1 pixel image and encryption process

As an example of the above pixel to be encrypted which has an RGB value of 50,70,90,200,50,50, the encryption process of the 8 bytes of data with the TEA method begins by dividing the 8 bytes of data into 2 equal blocks, namely the left block (L) and right block (R).

| 0th byte | 1st byte | 2nd byte | 3rd byte |
|----------|----------|----------|----------|
| 50       | 70       | 90       | 200      |

\[ L = 00110010010110100000000 = 843471560 \]

\[ R = 00110010001100100000000 = 842137600 \]

Then the encryption process from round-1 to round 32 is carried out as follows:

**TEA Round 1**

\[ L = 001100100101101000000000000000000 = 843471560 \]

\[ R = 001100100011001000000000000000000 = 842137600 \]

Delta = 2654435769

1. R on the left s hift is 4 bits and added with K (0)

\[ R = 842137600 + 589299712 = 1431437312 \]

Leftshift (R,4) = 01100100100000000000000000000000

\[ H1 = (Leftshift (R,4) + K(0)) \mod 2^{32} = 2508814955 \]

2. R added with Delta

\[ R = 842137600 + 2508814955 = 3350952655 \]

3. R is shifted by 5 bits and added with K [1]
RightShift(R,5)=000000110010001100100000000000=26316800
H3=(RightShift(R,5) + K(1)) mod 2^{32}=(26316800 + 1769175393) mod 2^{32}=1795492193
4. The results in steps 1, 2 and 3 are entered into the function
H4 = H1 XOR H2 XOR H3
H4=786762419
5. Add step 4 with L
H5=(H4+L) mod 2^{32}=163023979
6. R(new)= the result of step 5 =163023979
7. L(new)= R(old)= 842137600
8. R is left shift by 4 bits and added with
R=163023979=11000010101101100101101110
H8 = Left Shift(R,4) + K(2)) mod 2^{32}=(313939888 + 1752326509) mod 2^{32}=2066266397
9. R added with Delta
R=163023979
H9=(R+Delta) mod 2^{32}= 4284669784
10. H10=(Right Shift(R,5)+K(3)) mod 11.
11. The results in steps 8, 9 and 10 are entered into the XOR function or
H11=H8 XOR H9 XOR H10
H11=2066266397 XOR 4284669784 XOR 1684915084 =3760694729
12. Add step 11 with L
H12=(H11 + L) mod 2^{32}=(3760694729 + 842137600) mod 2^{32}=1218207329
13. R(new) = the result of step 12=1218207329
14. L(new)= R(old)= 163023979=11000010101101100101101110
L(After processing)=194 86 202 123
R(After processing)=145 56 188 97
TEA Round 32
L=243 199 211 119=11110011110001111101011101110111=4089959287
R=173 229 181 198=10101101110010111011010111100010=2917512646
Delta=(Delta +2654435769) mod 2^{32}=(683130215+2654435769)mod 2^{32}=3337565984
1. R on the leftshift is 4 bits and added with K[0]
R=2917512646=10101101110010111011010111100010=2917512646
H1= LeftShift (R,4)+K(0)) mod 2^{32}=(3760694729 + 842137600) mod 2^{32}=1355077323
2. R added with Delta R=2917512646
H2=(R+Delta)mod 2^{32}=(2917512646+3337565984)mod 2^{32}=1960111334
3. R on the right shift by 5 bits and added with K[1]
R=2917512646=10101101110010111011010111100010=3730529376
H3= Right Shift(R,5)+K(1)) mod 2^{32}=(91172270 + 1769175393)mod 2^{32}=1860347663
4. The results from steps 1, 2 and 3 are entered into the XOR
H4=H1 XOR H2 XOR H3
H4=1355077323 XOR 1960111334 XOR 1860347663=1257413922
5. Add step 4 with L ,H5=(H4+L) mod 2^{32}=(1257413922408959287)mod 2^{32}=1052405913
6. R(new)=the result from step 5=1052405913
7. L(new)=R(old)= 2917512646
8. R on left shift is 4 bits and added with K[2]
R=1052405913=11111010111010111100001011001
Leftshift(R,4)= 10101101110010111011010111100010=732367248
H8= Leftshift(R,4) + K(2)) mod2^{32}=(732367248 + 1752326509) mod 2^{32}=2484693757
9. R added with Delta
R=1052405913
H9=(R+Delta) mod 2^{32}=(1052405913+3337565984)mod 2^{32}=95004601
10. R on right shift as many as 5 bit and added with K[3]
R=1052405913=11111010111010111100001011001
Right Shift(R,5)=0000011111011011011011100001000=2887684
Right Shift(R,5)+K(3)) mod 2^{32}=(32887684+1633970273)mod 2^{32}=1666857957
11. Result on step 8, 9 and 10 add to function XOR
3
H11=H8 XOR H9 XOR H10
H11=2484693757 XOR 95004601 XOR 1666857957=4075484833
12. Add step-11 with L
   H12=(H11+L)mod 2^32=(4075484833+2917512646)mod 2^32=2698030183
13. R(new)=Result from step
   12=2698030183=10100000 11010000 11010001 01100111
   L(after processing)=250 233 194 25
   R(after processing)=160 208 172 103
The result of encryption from tiny encryption algorithm is like the figure 2.

![Figure 2. Encrypted Image](image)

**Decryption Process**

**TEA Round 1**

Swap L and R values so :

L=2698030183=10100000 11010000 11010001 01100111
R=1052405913=11111010 11101001 11000010 011001
Delta=3337565984

1. R on left shift as many as 4 bit and added with K[2]
   R=1052405913=11111010 11101001 11000010 011001
   Left Shift(R,4)=10101110100111000100110010000=732367248
   H1= Left Shift(R,4)+K(2)) mod 2^32=(732367248+175232659)mod 2^32=2484693757
2. R added with Delta 2^32=4.294.967.296
   R=1052405913
   H2=(R+Delta)mod 2^32=(1052405913+3337565984)mod 2^32=95004601
3. R on right Shift as many as 5 bit and added with K[3]
   R=1052405913=11111010 11101001 11000010 011001
   Right Shift(R,5)=000001111101011110011100010=32887684
   H3=(Right shift(R,5)+K(3))mod2^32=(32887684+1633970273)mod2^32=1666857957
4. Result on step-1,2and3 add function XORH4=732367248 XOR 95004601 XOR 1666857957=1297389004
5. Subtract step-4 with L
   H5=(H4-L)mod 2^32=(1297389004 – 2698030183)mod 2^32=1406441179
6. R(new) = Result from step-5 =1406441179
7. L(new)=R(old)= 1052405913
8. R on left shift as many as 4 bit and added with K[0]
   R=1406441179=101001101111110001010110011
   Left shift(R,4)=011011111100010110101110110000=935422384
   H8=(Left shift(R,4) + K(0)) mod 2^32=(935422384+1919515243)mod 2^32=2854937627
9. R added with Delta R=1406441179
   H9=(R+Delta)mod2^32=(1406441179+3337565984)mod2^32=443239867
10. R on right shift as many as 5 bit and added with K[1]
    R=1406441179=101001101111110001010110011
    Right shift(R,5)=000001010110111110000010110100=43770036
    H10=Right shift(R,5)+K(1)) mod2^32=43770036+1769175393)mod2^32=1812945429
11. Result from step 8.9 and 10 add to function-XOR
    H11=2854937627 XOR 443239867 XOR 1812945429=3696161717
12. Substruct step-11 with L
    H12=(H11 – L)mod 2^32=(3696161717 –1052405913)mod 2^32=2643755804
13. R(new)=Result from step-12
    2643755804=1001110100101000001100011100
14.L(new)=R(old)=

---

4
1400641179=10100110111100000101101011
L(after processing)=166 248 45 27
R(after processing)=157 148 131 28

**TEA Round 32**

L=2385234236=1000111001011110100100111011000101111011011

Delta=(Delta – 2654435769) mod 2^{32}=1013904242 – 2654435769 mod 2^{32}=2654435769

1.R on left shift as many as 4 bit and added with K[2]

R=332211194=1001111001101001001111111111010

Left shift(R,4)=11100111001100100100111111110100000=483540896

H1= Left shift(R,4)+K(2)) mod 2^{32}=(483540896 + 1752326509) mod 2^{32}=2235867405

2. R add with delta

R =332211194

H2=2986646963

3. R on right shift as many as 5 bit and added K[3]

R=332211194=1001111001101001001111111111010

Right shift(R,5)=00000100111101010100110011110111=10381599

H3= Right shift(R,5)+K(3)) mod 2^{32}=(10381599+1633970273)mod 2^{32}=1644351872

4. Result on step 1,2 and 3 add to function-XOR

H4= H1 XOR H2 XOR H3

H4=2235867405 XOR 2986646963 XOR 1644351872=1430449470

5. Subtract left step-4 with L

H5=(H4-L)mod 2^{32}=(1430449470 – 2385234236) mod 2^{32}=843471560

6.(new)=Result from step-5=843471560

7.(new)=R (old)= 332211194

8.R on left shift as many as 4 bit and added with K[0]

R=332211194=1001111001101001001111111111010

Left shift(R,4)= 1001001100100110101001100110100000=38165192

Left shift(R,4)+K(0)) mod 2^{32}=(38165192 + 1919515273)mod 2^{32}=2163685963

9.R add with delta

R=843471560

H9=(R+Delta)mod 2^{32}=(843471560+2654435769)mod 2^{32}=3609220535

10.R on right shift as many as 5 bit and added with K[1]

R=843471560=110010010001101010110110110100

Right shift(R,5)= 000001001111010101001100111101110=26358486

H10=Right shift(R,5)+K(1)) mod 2^{32}=(26358486 +1769175393) mod 2^{32}=1799012416

11.Result on step 8,9 and 10 add to function-XOR

H11=H8 XOR H9 XOR H10

H11=2163685963 XOR 3609220535 XOR 1799012416=1174348794

12/Subtract left step-11 with L

H12=(H11-L)mod 2^{32}=(1174348794 – 332211194) mod 2^{32}=842137600

13.R(new)=Result from step to12=842137600=00110010011001000000000000000000

Thus, the decryption process uses the key " rizkisyahramadha " back to the beginning, namely "50 70 90 200 50 50 0 0".

Based on this experiment, the processing time for encryption and embedding by using different encryption image and using similar cover image with different file size of image, there are 100x100pixels, 200 x200pixels, 300x300pixels than can be seen in Table 1.

### Table 1. Processing Time for Encryption and Embedding with Different Encryption Image and Similar Cover Image

| Size of Cover | Encryption process time | Embedding process time |
|--------------|------------------------|-----------------------|
| 100 x 100pixels | 18 ms                  | 329 ms                |
| 200 x 200pixels | 76 ms                  | 1720 ms               |
| 300 x 300pixels | 131 ms                 | 2997 ms               |
The following are the results for the time in the encryption process by using the same encryption image and different cover image with a size of 1000x1000 pixels, 1500x1500 pixels and 2000x2000 pixels.

**Table 2. Encryption and Embedding Processing Time with the Same Encryption Image and Different Cover Image**

| Size of Cover          | Encryption process time | Embedding process time |
|------------------------|-------------------------|------------------------|
| 1000 x 1000 pixels     | 14 ms                   | 308 ms                 |
| 1500 x 1500 pixels     | 16 ms                   | 317 ms                 |
| 2000 x 2000 pixels     | 16 ms                   | 323 ms                 |

The following are the results for the time Time Process Extraction and Decryption with the Image Decryption Different and Similar Image Stego with a size of 100x100 pixels, 200x200 pixels and 300x300 pixels that can be seen in Table 3.

**Table 3. Time Process Extraction and Decryption with the Image Decryption Different and Similar Image Stego**

| Size of Cover       | Extraction process time | Decryption process time |
|---------------------|-------------------------|-------------------------|
| 100 x 100 pixels    | 315 ms                  | 51 ms                   |
| 200 x 200 pixels    | 1625 ms                 | 97 ms                   |
| 300 x 300 pixels    | 2753 ms                 | 141 ms                  |

The following are the results for Extraction and Decryption Process Time with Similar Decryption Image and Different Stego Image with a size of 1000x1000 pixels, 1500x1500 pixels and 2000x2000 pixels that can be seen in Table 4.

**Table 4. Extraction and Decryption Process Time with Similar Decryption Image and Different Stego Image**

| Size of Cover       | Extraction process time | Decryption process time |
|---------------------|-------------------------|-------------------------|
| 1000 x 1000 pixels  | 276 ms                  | 34 ms                   |
| 1500 x 1500 pixels  | 289 ms                  | 37 ms                   |
| 2000 x 2000 pixels  | 282 ms                  | 43 ms                   |

Based on the tables 1, 2, 3 and 4 relationship between the time of the encryption and embedding process and image extraction and decryption with the pixel size of the image indicates that the pixel size of the image is directly proportional to the time. The bigger the pixel size of the image to be secured, the longer the time needed to process the image security.

4. **Conclusions**

The conclusion of this research, Based on the results of MSE and PSNR tests, it can be concluded that the changes that occur in the cover image are relatively invisible, so that the use of Tiny Encryption Algorithm and Modified Least Significant Bit Pseudo Random Number Generator is safe and simple. And based on the graph of the relationship between the encryption process time and the insertion and extraction and decryption of the image by measuring the pixel image, it shows that the pixel size of the image is directly proportional to time. The larger the pixel size of the image to be secured, the longer it will take to secure the image.

5. **Acknowledgments**

The authors gratefully thank all who have supported in the completion of this research. And to the Universitas Sumatera Utara that has given me the opportunity to publish this paper.
References

[1]. Andrew B. Chapman. S. and Dearstyn. S. 2001. Tiny Encryption Algorithm Cryptography 4005. 705. 01 (TEA). Graduate Team ACD – Final Report

[2]. Ahmet Cagri Bagbaba, Berna Ors, Osman Semih Kayhan, Ahmet Turan, Erozan. 2015. JPEG image Encryption via TEA algorithm. ISBN: 978-1-4673-7386-9

[3]. Suman, Sukhjeet K.R. 2017. A Secure Steganographic Method Using Modified LSB (Least Significant Bit) Substitution. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 6, Issue 8, August, ISSN: 2278 – 1323.

[4]. Rahmat R F, Ramadhana S, Faza S, Fawwaz I, Nababan E B 2019. Implementation of Vector Algebra and the Hybrid Pseudo Random Number Generator in Android Game of Kuaci Journal of Physics: Conference Series 1235(1), 012089

[5]. K. Agung, Fatmawati, and H. Suprajitno, “Image encryption based on pixel bit modification,” J. Phys. Conf. Ser., vol. 1008, no. 1, 2018, doi: 10.1088/1742-6596/1008/1/012016.