A Bit-Level Alteration Approach in Image Steganography

Niloy Roy
Research in Computer and Systems Engineering
Technische Universität Ilmenau
Ilmenau, Germany

***Abstract*—In this work, steganography refers to the method of hiding secret messages in seemingly ordinary digital cover mediums, for example, images, video, or audio files. In the current era of evolving cyber threats, this craft can be instrumental in preserving the integrity and confidentiality of sensitive data transmitted through different channels. This paper introduces a distinctive technique for choosing specific bits within the cover medium file to embed the cipher (i.e., encrypted secret message). The objective is to prevent intruders from identifying these crucial bits, even in the event of a compromised encryption key. Following this, one steganography tool was developed, incorporating the proposed algorithm. In the evaluation of performance, the tool showcased better results in specific metrics than other existing tools.**

Keywords*—*Steganography, LSB, AES, MSB, Payload capacity

I. Introduction

Steganography implements a two-layered strategy for secured data transmission by avoiding detection and resisting decryption. Avoiding detection means keeping intruders clueless about any hidden data within the cover image. The inability to bypass detection implies that intruders or detection software may have suspected or revealed the presence of confidential data [1]. In such cases, encryption makes sure intruders remain unable to decrypt and unveil the true plaintext.

In encryption, symmetric and asymmetric algorithms represent two fundamental paradigms. Symmetric encryption uses a single shared secret key for both encryption and decryption, making the process more efficient. However, the challenge lies in securely distributing the key. On the other hand, asymmetric encryption employs a pair of keys: a public key for encryption and a private key for decryption. Although this approach removes the need for secure key distribution, it tends to be computationally more intensive.

While certain publications [2], [3] explored and assessed the development of encryption algorithms, numerous studies focused on hiding techniques in the field of steganography. For example, Laplace and Sobel filters were suggested to find the best hiding points in [4]. However, advanced steganalysis techniques can exploit the patterns introduced by these filters, making it easier to detect the presence of concealed information. On top of this, filtering introduces a higher computational overhead and reduces the embedding capacity [5].

The solution to these shortcomings lies in achieving randomness with the most popular hiding technique in the existing literature, which is the Least Significant Bit (LSB) substitution method. For instance, the LSB technique has been used for video steganography in [6] and for the security of an electronic voting machine in [7]. This method puts bits from the secret text within the least significant bit of pixel values. This helps to hide the information with the least distortion.

This paper entails a retrospective summary of a research endeavor in which the author actively participated [8].To prioritize computational efficiency and ensure the highest security, this study decided to use the AES-128 symmetric key algorithm for encryption and a new variant of the LSB algorithm for hiding cipher in the RGB digital image. This tool makes detection harder for the intruders. In essence, it achieves this by using two sets of pseudo-random natural numbers and the most significant bits of the current pixel. This information goes into a hash function, and the result helps determine the spacing between the cipher bits in the cover image.

II. Tool Development

The methodology in this section is a retrospective summary of one of the author's previous work. [8]

A. Developing the Algorithm

The comprehensive method is separated into two primary components: Initially, concealing confidential information within a cover image to prevent detection and deciphering of data from the image. To thwart data detection, the LSB algorithm and an innovative method for selecting the pixel index in the cover image have been employed. Later, to prevent decryption, the AES-128 encryption technique is utilized for encrypting confidential text.

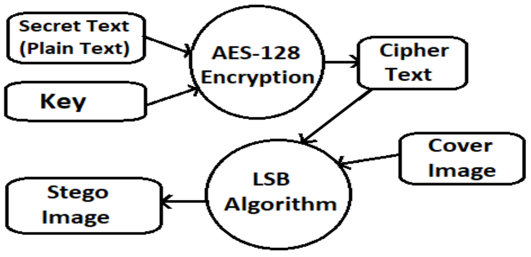


Fig. 1. Overview of Steganograohy Technique

The next part of the paper presents the algorithm 1 used to embed data in the image.

Algorithm 1: Algorithm for embedding Data in cover image

```
Step-1: Receive inputs for secret text, cover image,
  and a key.
Step-2: Assess the image quality based on the text; if
  below the standard, issue a warning to enhance it.
Step-3: Derive a byte array from the image and
  construct a pixel array using only the blue color value
  of each pixel in the image.
Step-4: Encrypt the provided secret text using the AES
  algorithm with the specified key, referred to as cipher
  text.
Step-5: Implement Algorithm 3 on the pixel byte
  array.
Step-6: Save the Steg Object.
```

Algorithm 3 illustrates the suggested method for selecting the pixel index to embed within the cover image. The subsequent significant segment involves retrieving confidential data from the image. The corresponding algorithm to the one mentioned earlier extracts the concealed data. Here, the key is provided as input along with the stego image, referred to as the stego-object, and the AES-128 bit decryption mode is applied, as depicted in algorithm 2.

Algorithm 2: Algorithm for extracting hidden data from the stego-object

```
Step 1: Input stego-object and the key.
Step 2: Utilize the counterpart of Algorithm 3 to
  obtain the hidden bit stream.
Step 3: Retrieve the character value of the bit stream.
Step 4: Decrypt using AES decryption with the
  specified key.
Step 5: Generate the secret text as the output.
```

Initially, we examine a **byte jumping series**, which guides the traversal of pixel indices within the image. This series facilitates determining the subsequent position or index in the byte array (pixel) where the next bit of the cipher text is incorporated. Utilizing its value allows us to obtain the decimal value of the most significant bit (MSB) in that byte. By adding this decimal value to the current position's column, we identify the next byte for insertion. Next, we consider a **zero series** for a specific scenario. Since a byte series is employed to obtain the next byte, there might be a situation where the decimal value of the MSB is zero. In such instances, the next byte is acquired using the zero case series.

1) Byte jumping series: [2 3 2 3 3 3 3 2 2]
2) Zero case series: [7 3 5 4 2 6 1]

Let the cipher text which will be converted in cipher bit stream is, 101110101 Let the byte array of an 8-by-8 image is the following figure 2.

Initialize all the necessary variables i.e i=1, j=1,m = 1, jumping_series_index = 1 and zero_series_index = 1 and

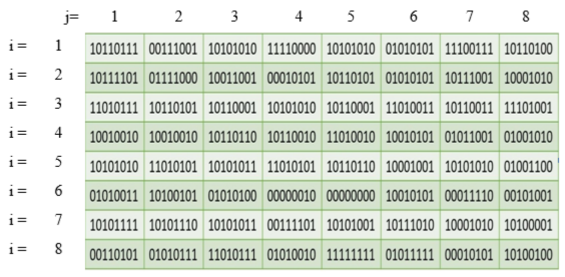
Algorithm 3: Pseudo algorithm from [8] for finding index of pixel from image and inserting cipher bits

```
Input: image, secret_text
Initialization: set: i=1 , j=1, jumping_series_index =
  1, zero_series_index = 1, cipher_bits_index = 1,
  jumping_series = [ 2 3 2 3 3 3 3 2 2 ], zero_series = [
  7 3 5 4 2 6 1 ] ;
Find: cipher_text = aes(secret_text); cipher_len =
  length (cipher_text); [ row, col ] = size (image);
while $i \leq row$ do
    while $j \leq col$ do
        insert cipher_text[cipher_bits_index] in the
          LSB of image[i, j] ;
        cipher_bits_index ++ ;
        if $cipher\_bits\_index > cipher\_len$ then
            break ;
        end
        n1 = jumping_series[jumping_series_index] ;
        if $n1 == 2$ then
            Take decimal value of 2 MSB bits by
              setting: n = image[i, j] / 64 ;
        else
            Take decimal value of 3 MSB bits by
              setting: n = image[i, j] / 32 ;
        end
        if $n == 0$ then
            n = zero_series[zero_series_index] ;
            if $zero\_series\_index < length(zero\_series)$
              then
                zero_series_index ++ ;
            else
                zero_series_index = 1 ;
            end
        end
        if $jumping\_series\_index < length$
          $(jumping\_series)$ then
            jumping_series_index ++ ;
        else
            jumping_series_index = 1 ;
        end
        update: j = j + n ;
    end
    update: i= i + 1 ;
    update: j = n ;
    if $cipher\_bits\_index > cipher\_len$ then
        break ;
    end
end
```

cipher_text_index = 1.

At first, (row, column) = (i, j) = (1, 1) = first binary value of the byte array (10110111) of Figure 2.

Next, the cipher_text_indexth position in the cipher bit stream is 1, Therefore, insert 1 into the least significant bit



	j= 1	2	3	4	5	6	7	8
i = 1	10110111	00111001	10101010	11110000	10101010	01010101	11100111	10110100
i = 2	10111101	01111000	10011001	00010101	10110101	01010101	10111001	10001010
i = 3	11010111	10110101	10110001	10101010	10110001	11010011	10110011	11101001
i = 4	10010010	10010010	10110110	10110010	11010010	10010101	01011001	01001010
i = 5	10101010	11010101	10101011	11010101	10110110	10001001	10101010	01001100
i = 6	01010011	10100101	01010100	00000010	00000000	10010101	00011110	00101001
i = 7	10101111	10101110	10101011	00111101	10101001	10111010	10001010	10100001
i = 8	00110101	01010111	11010111	01010010	11111111	01011111	00010101	10100100

Fig. 2. Byte array of an 8-by-8 image

(LSB) of the value at (1, 1). Since the LSB of 10110111 is already 1, no change occurs. Had the LSB been 0, we would switch it to 1. The byte jumping series has an index value of 2. Extracting 2 bits from the most significant bit (MSB) yields '10'. Decimal value of $(10)_2 = (2)_{10}$. resulting in n = 2. As n $\neq$ 0, no action is needed. Calculate m = j + n = 1 + 2 = 3 Next, check if m $>$ **col**. Given that the byte array is 8-by-8 with col = 8, we find m $<$ col and j = m = 3. Thus (i, j) becomes (1, 3). Increment cipher_text_index and jumping_series_index values. Repeat these steps until the last bit of the cipher bit stream. During this process, the byte jumping series or zero case series might conclude, requiring a restart. After 4^{th} iteration, when m $>$ **col**,the new (i, j) is (i++, n).

Consider the index value of (i, j) = (2, 4), which is represented as 00010101. In this instance, observe that the decimal value of the 3 most significant bits ('000') is 0. Consequently, the position remains unchanged, indicating that the values of i and j remain the same. However, a different index position is needed to insert the next bit of the cipher bit stream. In such scenarios, the concept of the zero case series is applied. Upgrade the value of n using the 1st value of the zero case series and increment the zero_series_index.

TABLE I
ITERATION TO HIDE CIPHER BIT STREAM

Iteration	**i**	**j**	**Byte jumping series value**	**MSB**	**Zero case series value**	**n**	**m = j + n**
1	1	1	2	10	7(No need)	2	$3 \leq 8$
2	1	3	3	101	-	5	$8 = 8$
3	1	8	2	10	-	2	$10 \geq 8$
4	2	2	3	011	-	3	$5 \leq 8$
5	2	5	3	101	-	5	$10 \geq 8$
6	3	5	3	101	-	5	$10 \geq 8$
7	4	5	2	11	-	3	$8 = 8$
8	4	8	2	01	-	1	$9 \geq 8$
9	5	1	2	10	-	2	$3 \leq 8$

In Table I, the iterations for embedding the cipher bit stream in the 8-by-8 byte array are illustrated. Since the length of the cipher bit stream is 9, the total number of required iterations to insert the bits into the LSB is 9.

In Table I we see that zero case series is not used, as there is no MSB value with 0. But with MSB values like '000' or '00', it can show the next cipher bit position.

B. Developing the Tool

This section constitutes a retrospective synthesis, summarizing the author's prior work as referenced [8]. It encompasses two core functions: data embedding and data extraction.

i) AES Encryption / Decryption

To prevent detection, we initially encrypt the provided secret text, which serves as the plaintext input, using AES-128 encryption and a key.

Plaintext = double(secret_text);
key = double(key);
s = aesinit(key);
cipher_text = aes(s,'encrypt','ecb', Plaintext);

Similarly, the counterpart handles the decryption process, as illustrated below, which is essential following the extraction of LSB bits from the image byte array.

plaintext = aes(s,'decrypt','ecb', cipher_text);

ii) Embedding Secret Data

Embedding secret data follows the steps of algorithm 1.

Upon inserting all the bits into the blue color matrix, we reconstructed the complete image and saved it as a BMP file, constituting the desired stego object.

iii) Extracting the Secret Data

The extracting part follows the steps of algorithm 2.

III. Evaluation of Implemented Tool

The result in this section is summarized from one of the author's previous works. [8]

A. Mean Square Error (MSE)

Mean square error is computed by dividing the sum of the squares of the differences in pixel values between the cover image and stego image by the total number of pixels.

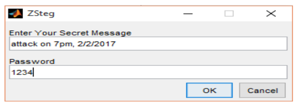
$$MSE = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{N} \tag{1}$$

Where, i = 1, 2, 3 . . . . n, up to the last pixel, e.g. for all pixels.

TABLE II
MSE VALUE COMPARISON WITH OTHER TOOLS

Tool's name	**MSE value**	**Output file type**	**Output file size**
Implemented tool	0.00041875	.bmp/.tiff	473 KB
OpenStego	0.012436427209154482	.bmp	70.3 FB [9]
QuickStego	0.03176652892561983	.bmp	147 KB [9]

We determine the MSE value of our implemented tool utilizing the aforementioned formula. The MSE value (0.00041875) obtained from the tool is comparatively low in contrast to the values from the other two tools (Table II).



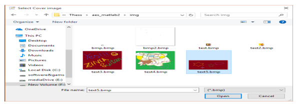
Enter Your Secret Message

attack on 7pm, 2/2/2017

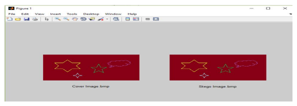
Password

1234

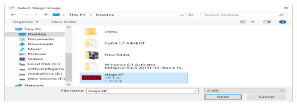
(a)



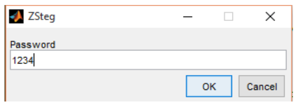
(b)



(c)



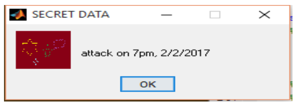
(d)



Password

1234

(e)



attack on 7pm, 2/2/2017

(f)

Fig. 3. Snippets of embedding and extracting secret data. (a) Giving input of secret text and the password (key). (b) Selecting a bmp image to hide the secret text. (c) Showing the cover image and stego image together. (d) Selecting the stego image to extract the secret data. (e) Giving password (key) at the rime of extracting. (f) Extracted secret text.

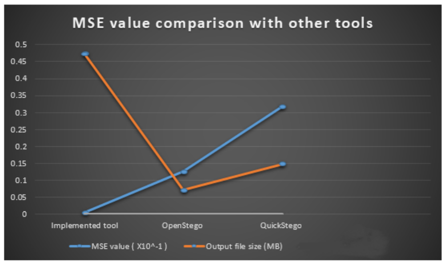


Fig. 4. Graph showing the comparison of MSE values and output file size among different tools.

B. Peak Signal to Noise Ratio (PSNR)

PSNR is measured in decibels, and a higher PSNR value indicates superior performance compared to others. It is computed using the following formula:

$$PSNR = 10log_{10}\frac{MAX_I^2}{MSE} \qquad (2)$$

Here, $\mathrm{MAX_I}$ represents the maximum intensity value for each color component of a pixel. In the case of a 24-bit color depth, where each component of a pixel contains 8 bits, $\mathrm{MAX_I}$ is set to 255 for 8 bits. The following Table III shows the PSNR value of our tool and other available tools. It is based on Table II.

TABLE III
SHOWING PSNR VALUE OF DIFFERENT TOOLS

Tool's name	PSNR value
Implemented tool	81.91125541
OpenStego	67.18384729
QuickStego	63.11110598

C. Payload Capacity

Payload capacity gauges the data that can be concealed in an image using the steganography tool. To measure this payload capacity, a 'text to image' ratio in percentage is required, as denoted by equation 3. Let x represent the secret text size in bytes, and y denote the total number of pixels in the image.

$$x <= \frac{y}{8*8} = \frac{y}{64} \qquad (3)$$

The payload capacity can be measured from equation 3,

$$x : y = 1 : 64 \qquad (4)$$

In percentage,

$$x : y = 1.5625 : 100 \qquad (5)$$

The computation reveals that the payload capacity of the presented technique is 1.6% concerning the image size.

IV. Conclusion And Future Work

The main contribution of this work is to minimize the detectability of the digital cover image in the field of steganography. The evaluation showed that achieving a lower mean square error and signal-to-noise ratio helps enhance this goal. Yet, this improvement came with a trade-off: a significant reduction in the amount of data that can be concealed. Therefore, future efforts should aim to improve payload capacity without making it easier to detect.

References

[1] Provos, N. and Honeyman, P., "Detecting Steganography Content on the Internet". CITI Technical Report 01 -11, 2001.

[2] Nath, J. and Nath, A., "Advanced Steganography Algorithm using encrypted secret message," in International journal of advanced computer science and applications 2.3 (2011).

[3] Zeghid, M., Machhout, M.,Khriji, L.,Baganne, A. and Tourki, R. "A modified AES based algorithm for image encryption," in International Journal of Computer Science and Engineering 1.1 (2007): 70-75.

[4] Umamaheswari, M.,Sivasubramanian, S. and Pandiarajan, S., "Analysis of different steganographic algorithms for secured data hiding," in IJCSNS International Journal of Computer Science and Network Security 10.8 (2010): 154-160.

[5] Sarkar, Soumyajit and Arijit Basu. "Comparison of Various Edge Detection Techniques for Maximum Data Hiding Using Lsb Algorithm."

[6] Sharma, Ms. H.,MithleshArya, Ms. and Goyal, Mr. D., "Secure Image Hiding Algorithm using Cryptography and Steganography," in IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN (2013): 2278-0661.

[7] Dutta, S., Das, X., Ganguly, R. and Mukherjee, I., "A Novel Approach to E-Voting Using Multi-bit Steganography," in Proceedings of the First International Conference on Intelligent Computing and Communication. Springer Singapore, 2017.

[8] Z. Sultana, F. Jannat, S. S. Saumik, N. Roy, N. K. Datta and M. N. Islam,"A new approach to hide data in color image using LSB steganography technique, " Proceedings of the 3rd International Conference on Electrical Information and Communication Technology (EICT), Khulna, pp.1–6, 2017.

[9] V, V. and Sebastian, S., "Comparative Study Of Steganography Tools." in International Journal of Innovations & Advancement in Computer Science IJIACS, ISSN 2347 - 8616, Volume 2, Issue2, February 2015.