A New Approach to Hide Data in Color Image Using LSB Steganography Technique

Zinia Sultana[1], Fatima Jannat[2], Sadman Sakib Saumik, Niloy Roy, Nishith Kumar Datta, Muhammad Nazrul Islam[3]
Department of Computer Science and Engineering
Military Institute of Science and Technology, Dhaka, Bangladesh
Email: {[1]sultana.hiramony, [2]fj.jannat20, [3]nazrulturku}@gmail.com

***Abstract*—With the advancement of information technology, almost everything in this digital world evolves with information and data. The importance of security and secrecy of data is also increasing enormously. Steganography presents the practice of hiding data or information in any sort of cover medium, e.g image, audio, video. In most of the existing procedures, with the exposure of the keys intruders get successful in picking their required bits. The objective of this paper is to propose and implement a tool for increasing the secrecy of data transmission using steganography. In this paper, firstly we have proposed a technique which adds double layer security to hide data in image using LSB algorithm, AES-128 encryption and a new approach of choosing index of image pixel. Secondly, we have developed a steganography tool using our proposed technique. Finally, we have evaluated the performance of the proposed technique using Mean Square Error (MSE) method, Peak Signal to Noise Ratio (PSNR) and by measuring payload capacity. A comparative analysis of the developed tool with the some existing tools has showed that our tool performs better comparing with some other tools.**

***Keywords*—Steganography, LSB, AES (Advanced Encryption Standard), MSB, MSE (Mean square Error)**

I. Introduction

Information security prevents any sort of unauthorized and illegal access of data. Now-a-days, the key concern of the information technology industries (IT) is to protect information and data from being stolen. There are a number of techniques in the information technology that maintain the confidentiality, secrecy and concealment of data. Among them, one of the popular techniques is steganography which ensures the security of the hidden commutation between the sender and the receiver.

Steganography is the craft of hiding data within images, videos and texts through transmission channels. It is aimed to ensure the confidentiality of the sent information through two levels of security: avoidance of detection and decryption. Avoiding detection covers both software & visual detection i.e. the secret text needs to embedded in the image in such a manner that it is not noticeable. One must confuse the intruder that the cover image can contain any hidden data. Decryption covers the part that even by any chance it is detected that an image is carrying any confidential data the intruders cannot decrypt it i.e. get the actual plain text.

In steganography, the hiding techniques will fail if an intruder can suspect the cover medium [18]. While using a steganography tool, one should ensure whether the hiding technique used by the tool is prevailing all sort of detection by the intruder. In some cases, after hiding an information or data in an image, people can understand that there is some distortion in the image. The more the success rate of avoiding detection of a steganography tool is, the more trustworthy the tool will be.

The objective of this paper is to propose a good performing steganography tool with a new hiding technique and to evaluate the performance of the proposed tool using Mean Square Error (MSE) method, Peak Signal to Noise Ratio (PSNR) and payload capacity.

The remaining sections of this paper are organized as follows. A brief overview of the related work and literature review is presented in section 2. The development framework of proposed steganography tool with the new technique and implementation processes are discussed in section 3. In section 4, the evaluation of tool is discussed, followed by conclusion and future work in section 5.

II. Literature Review

This section discusses about the related prior works of different stegenography data hiding techniques and tools.

In [2], an analysis of different algorithms for data hiding has been presented. This paper describes some of the mostly used stegnography algorithms briefly. They are Blindhide, Pixelswap, Hide Seek, Filter First, and Battle Steg. After that 2 types of filters (i.e. Laplace & Sobel) used for finding the best zones in images to hide have been mentioned. These are followed by a comparative analysis of performance among algorithms in terms of machine learning accuracy as a function of embedding rate. It concludes with the proposal and mathematical representation of a plausible deniability scheme.

Blowfish algorithm and LSB technique have been used respectively to encrypt the secret message and to hide the cipher into the bits of the frames of videos in [3]. Authors claimed the blowfish algorithm to be better than the other ones and evaluated their work of hiding into frames of videos as a new dimension in the steganographic field.

The work described in [4] is potentially in the field of cryptography. The idea of play fair method has been implemented in encrypting messages which resulted in MSA method (named after the names of designers). They have evaluated

978-1-5386-2307-7/17/$31.00 ©2017 IEEE

their way of encrypting with experiments and have considered it best for watermarking.

In [5], modified versions of AES encryption algorithm have been proposed. The work is supported with adequate experimental data. It also compares AES+A5/1 and AES+W7 in terms of the performance parameters like correlation vertical, correlation horizontal, PSNR value and entropy.

Instead of following the common method of modifying an existing image the work in [6] has chosen the time of synthesizing of a texture image to hide the text in that. Also claims the capability of dealing with JPEG compression.

The process of maintaining confidentiality evolved around steganography has been tried in the field of electronic voting in [7]. The techniques are LSB and the Minimal Impact Decimal Digit Embedding (MIDDE). Personal identification number and fingerprint are the keys here.

The paper [8] tells about an approach to detect LSB steganography. This approach is based on statistical measures of sample pairs. Bounds on estimation errors of this simple and fast approach have also been measured.

Paper [9] just tells about another hiding scheme consisting of parity-bit pixel value difference and improved right most digit placement which promises not only good visual quality but also security.

OpenStego [16] is an open source steganography software which is distributed under GNU general public license v2.0. In this software the authors have provided two main functionalities: data hiding and watermarking (beta). This Java based tool supports password-based encryption of data for additional layer of security. DES algorithm is used for data encryption, along with MD5 hashing to derive the DES key from the password provided. It uses a plugin based architecture, where various plugins can be created for different kind of steganography algorithms.

QuickCrypto [17] is another paid steganography tool for windows operating system. It uses DES algorithm to encrypt and decrypt.

In sum, the existing literature shows that the mostly used and popular hiding technique is the LSB and the best hiding places can be found using filters and laplace equations. The review study also showed that different algorithms have different fields of application where they fit most (i.e. some are best for watermarking and some are best for steganography). The earlier researches also showed that the performance of any steganographic tool can be measured based on Embedding rate, Robustness, Imperceptibility, Mean Square Error and Peak Signal to Noise Ratio. This work thus will focus on a new hiding technique where we will use the LSB hiding technique and AES encryption technique and measure the performance of the tool using some metrics that has been used in the existing literature.

III. Tool Development

This section discusses briefly about the total procedure of the new technique and how the tool has been implemented step by step.

A. Developing the Technique

The complete technique is divided into two major parts: Firstly, embedding secret data in a cover image avoiding detection and decryption of data from the image. To avoid the detection of data we have used LSB algorithm and a new approach of choosing pixel index in the cover image. On the other hand, to avoid decryption we have used AES-128 encryption technique to encrypt the secret text.

LSB (Least Significant Bit) algorithm insert bits of the secret text in the least significant bit of a pixel value so that the distortion is not detectable. In this paper, we have used two number series and MSB of the pixel value to get a random distance from one hidden bit to another. So that detection become more difficult.

To avoid decryption we have used cryptography technique. Among different types of cryptography techniques we have chosen symmetric key algorithm, AES-128 to encrypt the secret text before embedding it in the image using LSB. The strength of AES-128 bit is very high. It takes billions of years to break the encrypted text. [1]

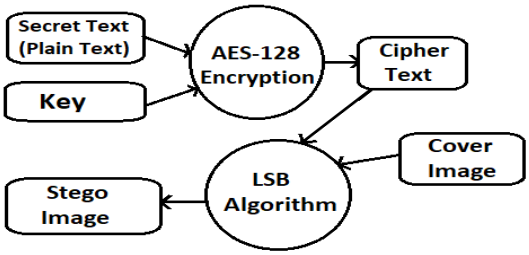

Fig. 1. Overview of Steganograohy Technique

The next part of the paper presents the algorithm 1 used to embed data in the image. While implementing algorithm 1 we have integrated our new technique (Algorithm 3) for choosing pixel index of cover image.

Algorithm 1: Algorithm for embedding Data in cover image

Step-1: Take input of secret text, cover image and a key.
Step-2: Check image quality according to the text. If not good enough then give warning to increase.
Step-3: Get byte array from the image and form pixel array using only blue color value of each pixel of the image.
Step-4: Encrypt the given secret text using AES algorithm with the given key known as **cipher text**.
Step-5: Apply **Algorithm 3** on the pixel byte array.
Step-6: Save the Steg Object.

Secondly, the next major part is to extract the secret data from the image. Counter part of the previous algorithm extracts the hidden data. In this case we give input of the key with the stego image known as stego-object and use AES-128 bit decryption mode as shown in algorithm 2.

In case of LSB algorithm one of the most important parts is finding the exact pixels of the cover image where the data

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

Step 1: Take inputs: stego-object & the key.
Step 2: Apply counter part of the **Algorithm 3** to get hidden bit stream.
Step 3: Get char value of the bit steam.
Step 4: Decrypt using AES decryption using the key.
Step 5: Outputs the secret Text.

would be hidden. For algorithm 1 and algorithm 2 we need to find the index of cover image pixel. Our proposal of a new approach for finding the desired index value of pixels in the cover image while inserting the secret data is described in Algorithm 3. This approach makes the detection more difficult for the intruders. For this purpose we consider two series randomly. They are,

1) Byte jumping series
2) Zero case series

First, we consider a **byte jumping series** which is used to traverse through the pixel indices of the image. It is used to get the next position or index of the byte array (pixel) where next bit of the cipher text is inserted. Byte jumping series is hard coded. Using its value we get the decimal value of the number of bits of the MSB of that byte. By adding the decimal value with the current positions column, we get the next byte where insertion takes place.

Second, we consider a **zero series** for a special case. As we use a byte series to get the next byte, there may be a case that the decimal value of the MSB is zero. In that case we get the next byte using zero case series.

An example case to describe the approach of finding pixel indices and then inserting data into that pixels are presented below: At first we consider the following two series as byte jumping series and zero case series.

1) Byte jumping series: [2 3 2 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.

	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

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

Algorithm 3: Pseudo algorithm 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 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

At first, (row, column) = (i, j) = (1, 1) = first binary value of the byte array (10110111) of Figure 2. Next, cipher_text_indexth bit of the cipher bit stream is 1, so insert 1 in the LSB of the value of (1, 1). As the LSB of 10110111 is 1, so no change takes place. If the LSB would be 0, then we

would change it into 1. Next, jumping_series_indexth value of the byte jumping series = 2. So consider 2 bits from the most significant bit (MSB) that is '1 0'. Decimal value of $(10)_2$ = $(2)_{10}$. So, n = 2. As n $\neq$ 0, so nothing needs to do. Calculate m = j + n = 1 + 2 = 3 Next, check whether m > **col** or not. As the byte array is 8-by-8, col = 8. So m < col and j = m = 3. Now (i, j) = (1, 3). Then, increment value of cipher_text_index and jumping_series_index. The above steps are continued until we reach the last bit of the cipher bit stream. Doing this the byte jumping series or the zero case series may finish. In that case we start from the beginning. After 4th iteration m > **col**.

Then new (i, j) = (i++, n).

Now consider the value of index(i, j) = (2, 4). It is 00010101. Here we see that decimal value of 3 MSB ('000') is 0. In this case position does not change, that means value of i and j is remained same. But we need a different index position to insert the next bit of the cipher bit stream. For this case the concept of zero case series comes. Upgrade the value of n with the 1st value of zero case series and increment zero_series_index. Table I shows the iterations to embed the cipher bit stream

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 the 8-by-8 byte array. As the length of cipher bit stream is 9, so the number of total required iteration to insert the bits in 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 it plays a vital role when the MSB value will result in 000 or 00 to get the next position to insert the next cipher bit.

B. Developing the Tool

This section gives the broad view of our implemented steganography tool using the methodology that is proposed in this paper. The tool is implemented in MATLAB (Matrix Laboratory)[15] (The source code can be found by e-mail request from the corresponding authors). There are two main parts: embedding data and extracting data

i) AES Encryption / Decryption

To avoid detection first we encrypt the plaintext i.e. the given secret text as input with AES-128 encryption using a key. This key is an input given by user which acts like a password. To use AES-128 bit the key needs to contain 16 characters. To make users unaware of AES-128 and its working process, we have written a section of code so that if the key is less than 16 character, then we have filled it with additional stars (*) to make the key length 16.

In this paper, Electronic Codebook Mode (ecb) [14] has been used to encrypt and decrypt data using AES. It is a mode of block cipher. In this mode the plaintext is divided in blocks and then each block is encrypted separately. Using the following segment of code we encrypt plaintext i.e. secret text.

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

First we make both plaintext and key in double data type. Then we call aseinit() function. This function mainly generate the substitute box also named as S-box using the key and return the S-box. Then we call the aes() function which does the encryption of the plaintext and return the cipher text. In the aes() function we also mention the mode of block cipher. Thus we get the desired cipher text. Again the counterpart does the decryption as shown below which is needed after extracting 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. First we take input of the secret text and a key. Then we encrypt it using AES-128. Next we convert the encrypted text in a bit stream and take the cover image of BMP file format. After that we get the byte array of the image. The byte array (pixels) of the image contain three values of red, green and blue component of each pixel as the image is a color image. We only consider the blue color matrix to hide cipher bit stream. Then according to the LSB algorithm and our pixel index choosing technique (algorithm 3), we insert the bits. In case of insertion if we have cipher bit 1 need to insert, and the byte value is even, we increment it by one, as a number even means in binary it contains zero (0) in the LSB. If byte value is odd, then nothing needs to do. And if we have cipher bit 0 and the byte value is odd, then we decrement it by one. Otherwise do nothing. We also insert an end of file (EOF) 00000 after inserting the cipher bits. To insert this EOF, we ensure that, the main cipher bits do not contain 5 consecutive zeros. To ensure this, one 1 is inserted after every consecutive 4 zeros. This procedure is called bit stuffing.

After inserting all the bits in the blue color matrix, we reconstruct the full image and saved it as a BMP file which is the desired stego object.

iii) Extracting the Secret Data

The extracting part follows the steps of algorithm 2. At first we take input of the stego image and the key. Then we collect the LSB value using the counter part of algorithm 3 until we get the EOF which is encrypted using AES. From

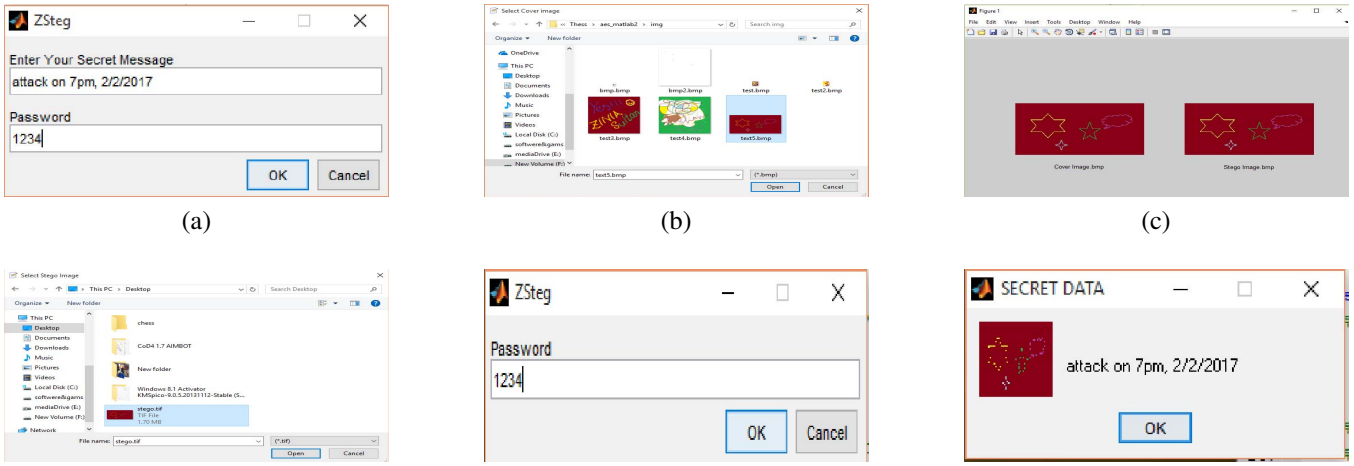

Fig. 3. Snippets of embedding and extracting secret data. (a) Giving input of secret text and the password (key). (b) Selecting a bmp image to hide the secret text. (c) Showing the cover image and stego image together. (d) Selecting the stego image to extract the secret data. (e) Giving password (key)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

this cipher text, the 1's are removed after every consecutive 4 zero's which have been inserted during the embedding stage. Then using the AES decryption we get the plaintext.

Figure 3 shows some screenshots of the implemented tool. In fig 3. (a) an user enters the secret text and the password (key). Then in fig 3. (b) a bmp image needs to be selected where the text will be hidden. Thus the stego image embedded with secrect text will be found which is shown in fig 3. (c). In case of extracting secret data part, at first the user needs to select the stego image from its destination folder (fig 3. (d)). Then like 3. (e) user will have to enter the password(key). Finally if the password is correct, the secret data will be extracted like 3. (f)

IV. Evaluation of Implemented Tool

There are many ways to measure the performance of an implemented tool. There are many performance metrics like Mean Absolute Error (MAE), Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), Time Complexity, Qualitative Analysis, Normalized Color Deviation (NCD), Normalized Histogram Intersection Coefficient, Bhattacharyya Coefficient and Universal Image Quality Index (UIQI)[12]. Among these, we have analyzed our implemented tool in three criteria. They are Mean square error (MSE), Peak Signal to Noise Ratio and payload capacity.

A. Mean Square Error (MSE)

Mean square error is calculated by dividing sum of the square of all pixel values difference of the cover image and stego image by the total number of pixel. The following equation will give a better understanding. A steganography tool will be better if the MSE value becomes less.

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

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

x is color value of i^{th} pixel in the cover image.

y is color value of i^{th} pixel in the stego image.

N is the total number of pixel. We calculate the MSE value

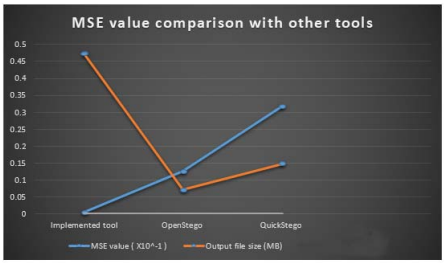

Fig. 4. Graph showing the comparison of MSE values and output file size among different tools.

TABLE II
MSE value comparison with other tools

Tools name	MSE value	Output file type	Output file size
Implemented tool	0.00041875	.bmp/.tiff	473 KB
OpenStego	0.012436427209154482	.bmp	70.3 FB [13]
QuickStego	0.03176652892561983	.bmp	147 KB [13]

of our implemented tool using the above formula. The MSE value (0.00041875) of the tool is relatively low comparing with the other two tools (Table II).

B. Peak Signal to Noise Ratio (PSNR)

PSNR is calculated in decibel. The more will be the PSNR value, the tool will be better than other. PSNR is calculated by the following formula,

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

Here MAX_I is the maximum intensity value of each color component of a pixel. For color depth 24 bit, each component

of a pixel holds 8 bits. So MAX_I will be 255 for 8 bit. The following Table III shows the PSNR value of our tool and other available tool. It is based on Table II.

TABLE III
SHOWING PSNR VALUE OF DIFFERENT TOOLS

Tools name	PSNR value
Implemented tool	81.91125541
OpenStego	67.18384729
QuickStego	63.11110598

C. Payload Capacity

Payload capacity measures how much data can be hidden in an image using the steganography tool. A 'text to image' ratio in percentage is needed to measure this payload capacity (equation 3). Let, Secret text size (in byte) = x, Total number of pixels in Image = y

$$x <= \frac{y}{8*8} = \frac{y}{64} \tag{3}$$

As byte jumping series or zero case series have been used to jump from one pixel position to another, according to the values of series's at most 7 bytes can be jumped over at once. Considering 8, the number of available pixels $y/8$. Again one byte of the secret data will be hidden in 8 pixels of the cover image. So again divide it by 8. Thus the ratio becomes that secret data bits size should be less or equal to the quotient of the total number of pixel in the image divided by 64. So during implementation, a warning is given requesting to increase the image size or decrease the secret text size, if the ratio is not maintained. The payload capacity can be measured from equation 3,

$$x : y = 1 : 64 \tag{4}$$

In percentage,

$$x : y = 1.5625 : 100 \tag{5}$$

So the calculation indicates that the payload capacity of the proposed technique is 1.6% with respect to image size. This plays a vital role. Whatever the image size is, only 1.6% will be affected after applying steganography. The more payload capacity is, the more secret data can be hidden in an image. But there should be a threshold or it may be perceptible that a particular image is having some sort of secret data. MSE value will also be greater for greater payload capacity which is supposed to keep low for less distortion of stego-image with original image.

V. CONCLUSION AND FUTURE WORK

This paper proposed a technique to choose the pixel index in the cover image of steganography for doubling the data security at the detection level. LSB algorithm and AES-128 encryption method have been integrated with this new technique. We have developed a steganography tool incorporating this proposed technique. The performance of our tool has been tested using MSE, PSNR method and measuring payload capacity. Since the proposed technique is not solely based on secrecy of keys, so even if the keys (series) are detected, it will be still difficult for intruder to pick up the desired bits. An intruder will face extream difficulty to reveal the message due to the procedure of dynamically choosing the bits to hide as a function of the values of pixel bits. Our future plan includes: a) measuring the performance of our tool in some other metrics (like NDC) to compare with the performance of exsisting tools, b) comparing the Payload capacity with other methods, c) addition of an interface for the users to add keys so that after a certain time, keys can be updated and d) update the tool for the compressed images e.g. jpeg, png etc.

REFERENCES

[1] "Advanced Encryption Standard". [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. Accessed: Jan 14, 2017.

[2] 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.

[3] 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.

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

[5] 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.

[6] Owada, M., Nagashima, Y. and Kimura, T., "Image processing apparatus and method for synthesizing first and second image data," in U.S. Patent No. 5,857,038. 5 Jan. 1999.

[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] Dumitrescu, S., Wu, X., and Wang, Z., "Detection of LSB steganography via sample pair analysis," in IEEE transactions on Signal Processing 51.7 (2003): 1995-2007.

[9] Wu, H-C., Wu, N-I., Tsai, C-S. and Hwang, M-S. , "Image steganographic scheme based on pixel-value differencing and LSB replacement methods," in IEEE Proceedings-Vision, Image and Signal Processing 152.5 (2005): 611-615.

[10] "Root: En: Projects [RFMLabWiki]," 2014. [Online]. Available: http://radio.feld.cvut.cz/personal/matejka/wiki/doku.php?id=root:en:projects. Accessed: Jan. 14, 2017.

[11] "Why does the human eye see more shades of green than any other colour?," [Online]. Available: https://www.quora.com/Why-does-the-human-eye-see-more-shades-of-green-than-any-other-colour. Accessed: Jan. 14, 2017.

[12] Dhall, S., Bhushan, B. and Gupta, S., "An In-depth Analysis of Various Steganography Techniques." International Journal of Security and Its Applications 9.8 (2015): 67-94.

[13] 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.

[14] "Electronic Codebook (ECB)". [Online]. Available: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Electronic_Codebook_.28ECB.29. Accessed: Sep 25, 2017.

[15] "MATLAB- MathWorks". [Online]. Available: https://www.mathworks.com/products/matlab.html. Accessed: Sep 25, 2017.

[16] "OpenStego," [Online]. Available: http://www.openstego.com. Accessed: Sep. 27, 2017.

[17] "Free Steganography software - QuickStego," 2015. [Online]. Available: http://quickcrypto.com/free-steganography-software.html. Accessed: Sep. 27, 2017.

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