Multi-LSB and Modified Vernam Cipher to Enhance Document File Security

Erna Budhiarti Nababan*, *Member, IAENG*, Goklas Tomu Simbolon, Opim Salim Sitompul, *Member, IAENG*

***Abstract*—Numerous methods can be used to secure important information from potential misuses. Generally, there are two methods of data security, namely steganography and cryptography. Least significant bit (LSB) is a method that is often used in steganography, whereas the Vernam cipher is a popular cryptographic algorithm used for encryption. However, the simplicity of the LSB method and Vernam cipher is inappropriate for long term use. Thus, the Vernam cipher is modified to be more complex, wherein the bit arrangement of each character is modified, and the plaintext content is scrambled by rotating the bits prior to the XOR operation. The LSB method is also modified using multi-bit LSB which can be used to insert as much as 1-, 2-, 3-, or 4-bit information. In this study, the combination of steganography and cryptography methods are tested to evaluate the influence of character length and image resolution on the mean-square error (MSE) and peak signal-to-noise ratio (PSNR) values and then to compare the percentage of pixel usage at each bit insertion level. The result shows that the resulting stego images are good in concealing the secret information, which give small MSE values and PSNR values > 40. In addition, the use of 4-bit LSB is still feasible and use smaller percentage of storage. The application of multi-bit LSB method in steganographic activities is thus advantageous, in term of each pixel can hold more message bits compared with the conventional LSB method.**

***Index Terms*—Cryptography, LSB, Steganography, Vernam encryption.**

I. Introduction

According to the Cisco Visual Networking Index, the global IP traffic that occurred in 2022 was estimated to reach 4.8 zettabytes per year [1]. Given this amount of traffic, data security is becoming increasingly important, particularly for sensitive data, such as company data and state security information, authentications are more threatened than ever due to the unlimited copying [2]. Digital information is indirectly transmitted through a data network via a small electric current that is used as a link to analog signals [3]. Data often comprise important information that must be properly secured to prevent theft by certain parties [4], [5], [6].

Steganography and cryptography are two methods of securing data. Steganography conceals data, whereas cryptography secures data by encoding the plaintext. Acts of crime are becoming more advanced, and consequently, these methods often fail. Various types of data theft are performed to solve well-encrypted ciphertexts; hence, the power of steganography and cryptography must be increased through various methods.

The least significant bit (LSB) steganography technique operates by replacing the rightmost bit of each pixel corresponding to the message bit. Herein, three color elements of a pixel in a color image: red, green, and blue are alternately used. The vulnerability of LSB lies in the placement of plaintext bits in one row, allowing intruders to easily extract bits of confidential information. Thereby, the aim of this study is to improve data security by combining steganography and cryptography techniques.

The Vernam algorithm encrypts data by performing exclusive-OR (XOR) operations on each plaintext character in a given set of data. This algorithm is modified by rotating bits. After obtaining the cryptographic results, the ciphertext is concealed via the multi-bit LSB steganography technique on 24-bit image using several storage bit models. Steganography suppresses each bit of information into RGB color components [7], and multi-bits can be used to increase security by creating variations of the LSB technique [7], [8], [9]. Each sample is tested for the mean square error (MSE) and peak signal-to-noise ratio (PSNR) values obtained from stego-images, which are used to determine the best quality image with smallest error rate; this finest image could then be used as the storage image. In this manner, the hybrid technique can be used to improve data security.

II. Cryptography and Steganography Algorithms

A. Vernam Algorithm

The Vernam cipher is an algorithm based on the principle that each character in the plaintext is encrypted using the XOR process against a certain generated key [10], [11]. The provided key is generated repeatedly or extended to have as many plaintext lengths. In the Vernam cipher algorithm, a symmetric key type signifies that the same key is used for both encryption and decryption. The Vernam cipher's vulnerability could be identified by the use of simple XOR operation for both plaintext encryption and decryption. The following illustration of message encryption and decryption are performed using the Vernam algorithm.

The encryption process is initiated by representing each character of the plaintext in ASCII character set binary format and then XORing each binary character with a provided key which also reprented in the same binary format. For example, a plaintext: “binary” and key: “modify” will generate a cipher text: ›°˙˛»`.

$$\begin{array}{r} 01100010\ 01101001\ 01101110\ 01100001\ 01110010\ 01111001 \\ 01101101\ 01101111\ 01100100\ 01101001\ 01100110\ 01111001\ \oplus \\ \hline 00001111\ 00000110\ 00001010\ 00001100\ 00010100\ 00000000 \end{array}$$

Manuscript received September 26, 2019; revised August 12, 2020.

*Corresponding author: E. B. Nababan is with the Department of Information Technology, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia, phone/fax: +6261822139; +6261822129; (e-mail: ernabrn@usu.ac.id)

G. T. Simbolon formerly was a graduate student at Department of Computer Science, Universitas Sumatera Utara; Medan, Indonesia; (e-mail: gks.simbolon@gmail.com)

O. S. Sitompul is with the Department of Information Technology, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia (e-mail: opim@usu.ac.id)

The same process is performed in decryption process in order to recreate the plaintext "binary" from the ciphertext.

$$\begin{array}{r} 00001111\ 00000110\ 00001010\ 00001100\ 00010100\ 00000000 \\ 01101101\ 01101111\ 01100100\ 01101001\ 01100110\ 01111001 \oplus \\ \hline 01100010\ 01101001\ 01101110\ 01100001\ 01110010\ 01111001 \end{array}$$

B. Least Significant Bit (LSB)

An eight-bit binary string could be viewed as an integer number, in which bits are ordered from left to right direction showing the most significant bit (MSB) located at the left-most bit (bit poition 0) to least significant bit (LSB) located at the right most bit (bit position 7). In this ordering scheme, bit sequencing from 0 to 7 could represent the intensity value of a color. For example,

value:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
number:	0	1	1	1	0	0	0	0
position:	0							7(LSB)

In this illustration, the MSB is located at bit position 1 (2^6), representing number with a value of 112 in decimal. The value will change quickly if the LSB's change. For example, 1-bit LSB changes from (01110000) to (01110001) resulting a difference value of 1 (113), 2-bit LSB changes from (01110000) to (01110011) resulting a difference value of 3 (115), and a 3-bit LSB changes from (01110000) to (01110111) will result a difference value of 7 (119).

C. Steganography

Steganography, which is a Greek term that means "closed writing," is the technique of communicating by hiding the existence of messages. Steganography plays an important role in data security, whereby a steganography system comprises three elements: the cover image, secret message, and stego-image. Digital images are represented using X and Y coordinates that contain three color elements in each pixel. Typically, a gray image uses 8 bits, whereas a color image uses 24 bits to describe the RGB color model.

Several techniques exist for hiding information in the cover image. Spatial domain techniques manipulate pixel bit values to embed confidential information, and message bits are directly encoded into the pixel bits of the cover image. Thus, spatial domain techniques can be easily implemented [12].

In digital steganography, messages can be hidden by manipulating and storing information in pixels. If a user manipulates a 2-bit LSB in each color component in pixels, the value of the color component changes, at maximum, from -3 to +3, which may not be distinguishable to the human eye [13]. An illustration of how these manipulation of the 2-bit LSB values of a 24-bit RGB black color could be seen in Table I.

D. Fidelity Measurement

Fidelity is a security element in the practice of hiding secret messages. The measurement of fidelity in steganography can be performed by calculating values of the mean squared error (MSE) and Peak Signal-to-Noise Ratio (PSNR) [14] as shown in (1) and (2), respectively. The PSNR is usually measured in decibels (dB). To determine the PSNR, first, the average value of the square of the MSE error must be determined. If the MSE value is small, a large PSNR value is produced, and vice versa. Images with a PSNR value ≥ 40 dB are considered high-quality images [15].

TABLE I
COLOR VARIATION OF 2-BIT MODIFICATION OF A BLACK IMAGE

	R	G	B	Image
Dec	0	0	0	
Hex	0h	0h	0h	
Bin	00000000	00000000	00000000	
	R	G	B	Image
Dec	3	3	3	
Hex	3h	3h	3h	
Bin	00000011	00000011	00000011	

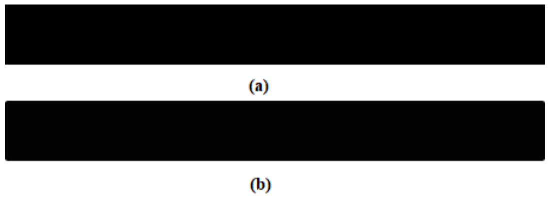

Fig. 1. Black color image: (a) RGB(0,0,0) (b) RGB(3,3,3)

The MSE value can be calculated using the following formula:

$$\textbf{MSE} = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}[f(i,j)-g(i,j)]^2}{MN} \tag{1}$$

where MSE is the MSE value of the image; M is the length of the image (in pixels); N is the width of the image (in pixels); while f(i, j) and g(i,j) are the value of the pixel coordinates of the stego image and cover image. After calculating the MSE value, the PSNR value can be calculated as in (2).

$$\textbf{PSNR} = 10 \times log_{10}\left(\frac{C_{max}^2}{MSE}\right) \tag{2}$$

In (2) the PSNR value of the image is calculated in terms of logarithm of the maximum pixel value C_{max} divided by the value of MSE. For RGB color images with values in each pixel, the MSE value is calculated for each color component, and then all values are summed-up and divided by 3 as shown in (3).

$$MSE = \frac{\left(\sum_{i=1}^{n}\left[(R_i^{'}-R_i)^2+(G_i^{'}-G_i)^2+(B_i^{'}-B_i)^2\right]\right)/3}{MN} \tag{3}$$

In (3), $R_i^{'}, G_i^{'}, B_i^{'}$ are the RGB components of the stego image i^{th}-pixel, R_i, G_i, B_i are the RGB components of the cover image i^{th}-pixel, whereas M and N denote the dimensions of the image.

Fig.1 shows an example of changing two bits of LSB values of a black image. The color difference between the two images is almost not detected with bare human eyes. In fact, this difference could be calculated in terms of MSE and PSNR using (3) and (2), respectively.

III. METHODOLOGY

The document file security model used in this research could be viewed as consists of three stages: input preparation stage, cryptography stage using modified Vernam cipher, and the steganography stage using multi-bit LSB. The general architecture of the system design is shown in Fig.2.

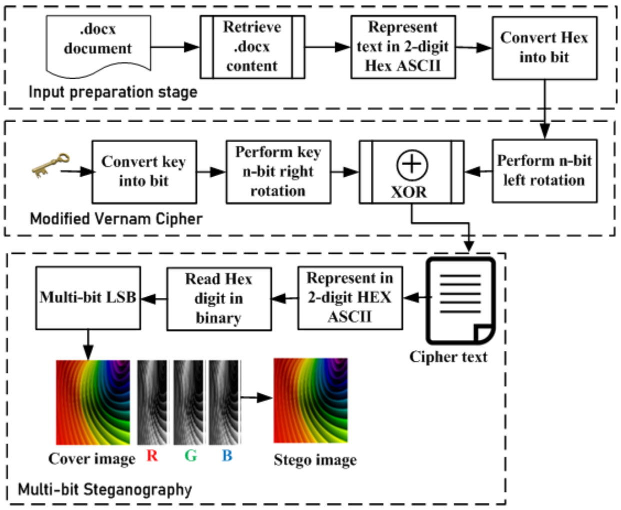

Fig. 2. General Architecture

TABLE II
CONFIDENTIAL LETTERS (CF) IN .DOCX FILE

No	.docx	Number of charachters	Size (bytes)
1.	CF1	1450	13722
2.	CF2	2149	13824
3.	CF3	1925	14029
4.	CF4	4746	21095

A. *Input Preparation Stage*

In this study we used samples of confidential letters of a person or an agency which need to be kept securely. Number of characters and size of *.docx* file used are shown in Table II. Input in the form of a *.docx* file is limited to characters that can be changed in ASCII code, and the system cannot encrypt tables, images, or symbols.

The steps performed in this preparation stage are initiated by retrieving content of the *.docx* file. Each ASCII character extracted is then represented in its corresponding 2-digit hexadecimal in order to simplified the conversion of the character into binary digit.

B. *Modified Vernam Cipher*

Modification of Vernam cipher algorithm is performed through bits rotation, whereby all character bits are rotated to the left. The cryptography key is also converted into binary digit from its 2-digit Hex representation before the key bit positions are rotated to the right. The two rotated bits are XORed to produce an encrypted character. One character contains 8 bits, and bit rotation is performed from the 1st bit to the 8th bit (this might be adjusted as needed). Because one character contains 8 bits, if the rotation is performed more than eight times, the bit position will return to its original position; i.e., 9-bit rotation is the same as 1-bit rotation, whereas 8-bit rotation is the same as 0-bit rotation.

The process of modifying the Vernam algorithm for encryption is illustrated as in the following.

1) Modifying the Vernam algorithm for encryption:

Step 1. Read plaintext:
The first step is to read the plain text characters, represented by P0, P1, ..., P4.

P0	P1	P2	P3	P4

Step 2. Convert into bit:
Each of the plain text character is converted into 8 binary digits, represented by B00, B01, ..., B07 for P0, etc.

B07	B06	B05	B04	B03	B02	B01	B00	→	P0
B17	B16	B15	B14	B13	B12	B11	B10	→	P1
B27	B26	B25	B24	B23	B22	B21	B20	→	P2
B37	B36	B35	B34	B33	B32	B31	B30	→	P3
B47	B46	B45	B44	B43	B42	B41	B40	→	P4

Step 3. Perform plaintext 1-bit left rotation:
Each binary digit of the plain text character is rotated to the left one bit, displacing bits positon left circularly.

B06	B05	B04	B03	B02	B01	B00	B07	→	P0’
B16	B15	B14	B13	B12	B11	B10	B17	→	P1’
B26	B25	B24	B23	B22	B21	B20	B27	→	P2’
B36	B35	B34	B33	B32	B31	B30	B37	→	P3’
B46	B45	B44	B43	B42	B41	B40	B47	→	P4’

Step 4. Read key:
Cryptography key is read in the form of K0, K1, ..., K4.

K0	K1	K2	K3	K4

Step 5. Convert key into bits:
Each key is converted into 8-bit sequence, represented as K00, K01, ..., K07 for K0, etc.

K07	K06	K05	K04	K03	K02	K01	K00	→	K0
K17	K16	K15	K14	K13	K12	K11	K10	→	K1
K27	K26	K25	K24	K23	K22	K21	K20	→	K2
K37	K36	K35	K34	K33	K32	K31	K30	→	K3
K47	K46	K45	K44	K43	K42	K41	K40	→	K4

Step 6. Perform key 1-bit right rotation:
Each binary key character is rotated right one bit, displacing bits position to the right circularly.

K00	K07	K06	K05	K04	K03	K02	K01	→	K0’
K10	K17	K16	K15	K14	K13	K12	K11	→	K1’
K20	K27	K26	K25	K24	K23	K22	K21	→	K2’
K30	K37	K36	K35	K34	K33	K32	K31	→	K3’
K40	K47	K46	K45	K44	K43	K42	K41	→	K4’

Step 7. Create ciphertext using XOR:
Each binary representation of character text is XORed with binary key.

B06⊕K00	B16⊕K10	B26⊕K20	B36⊕K30	B46⊕K40
B05⊕K07	B15⊕K17	B25⊕K27	B35⊕K37	B45⊕K47
B04⊕K06	B14⊕K16	B24⊕K26	B34⊕K36	B44⊕K46
B03⊕K05	B13⊕K15	B23⊕K25	B33⊕K35	B43⊕K45
B02⊕K04	B12⊕K14	B22⊕K24	B32⊕K34	B42⊕K44
B01⊕K03	B11⊕K13	B21⊕K23	B31⊕K33	B41⊕K43
B00⊕K02	B10⊕K12	B20⊕K22	B30⊕K32	B40⊕K42
B07⊕K01	B11⊕K11	B27⊕K21	B37⊕K31	B47⊕K41
↓	↓	↓	↓	↓
C0	C1	C2	C3	C4

2) Modifying the Vernam algorithm for decryption: As for the decryption process the steps are reversed, in which the first step is the XOR operation between the ciphertext and the key. The XOR results are then rotated to the right for the message to return to its original plaintext sequence.

Step 1. Read ciphertext:
Each chipertext character is read, represented by C0, C1, ..., C4.

C0	C1	C2	C3	C4

Step 2. Convert ciphertext into bits:
Each chipertext character is converted into 8 binary digits, represented by C00, C01, ..., C07 for C0, etc.

C07	C06	C05	C04	C03	C02	C01	C00	→	C0
C17	C16	C15	C14	C13	C12	C11	C10	→	C1
C27	C26	C25	C24	C23	C22	C21	C20	→	C2
C37	C36	C35	C34	C33	C32	C31	C30	→	C3
C47	C46	C45	C44	C43	C42	C41	C40	→	C4

Step 3. Read key:
The cryptography key is read, represented by K0, K1, ..., K4.

K0	K1	K2	K3	K4

Step 4. Convert key into bit:
Each cryptography key is converted into 8 binary digits, represented by K00, K01, ..., K07 for K0, etc.

K07	K06	K05	K04	K03	K02	K01	K00	→	K0
K17	K16	K15	K14	K13	K12	K11	K10	→	K1
K27	K26	K25	K24	K23	K22	K21	K20	→	K2
K37	K36	K35	K34	K33	K32	K31	K30	→	K3
K47	K46	K45	K44	K43	K42	K41	K40	→	K4

Step 5. Perform key 1-bit right rotation:
Each cryptography key is rotated one bit to the right, displacing bits position right circularly.

K00	K07	K06	K05	K04	K03	K02	K01	→	K0’
K10	K17	K16	K15	K14	K13	K12	K11	→	K1’
K20	K27	K26	K25	K24	K23	K22	K21	→	K2’
K30	K37	K36	K35	K34	K33	K32	K31	→	K3’
K40	K47	K46	K45	K44	K43	K42	K41	→	K4’

Step 6. Recreate plaintext using XOR:
The ciphertext characters are XORed with the cryptography key to recreate the plaintext characters.

C07⊕K00	C17⊕K10	C27⊕K20	C376⊕K30	C47⊕K40
C06⊕K07	C16⊕K17	C26⊕K27	C36⊕K37	C46⊕K47
C05⊕K06	C15⊕K16	C25⊕K26	C35⊕K36	C45⊕K46
C04⊕K05	C14⊕K15	C24⊕K25	C34⊕K35	C44⊕K45
C03⊕K04	C13⊕K14	C23⊕K24	C33⊕K34	C43⊕K44
C02⊕K03	C12⊕K13	C22⊕K23	C32⊕K33	C42⊕K43
C01⊕K02	C11⊕K12	C21⊕K22	C31⊕K32	C41⊕K42
C00⊕K01	C10⊕K11	C20⊕K21	C30⊕K31	C40⊕K41
↓	↓	↓	↓	↓
P0’	P1’	P2’	P3’	P4’

Step 7. Convert into bit:
The resulting characters are then converted into 8 binary bits.

B06	B05	B04	B03	B02	B01	B00	B07	→	P0’
B16	B15	B14	B13	B12	B11	B10	B17	→	P1’
B26	B25	B24	B23	B22	B21	B20	B27	→	P2’
B36	B35	B34	B33	B32	B31	B30	B37	→	P3’
B46	B45	B44	B43	B42	B41	B40	B47	→	P4’

Step 8. Perform plaintext 1-bit right rotation:
The resulting characters in binary form are rotated one bit to the right, and the original plaintext is obtained.

B07	B06	B05	B04	B03	B02	B01	B00	→	P0
B17	B16	B15	B14	B13	B12	B11	B10	→	P1
B27	B26	B25	B24	B23	B22	B21	B20	→	P2
B37	B36	B35	B34	B33	B32	B31	B30	→	P3
B47	B46	B45	B44	B43	B42	B41	B40	→	P4

C. Steganography with Multi-bit LSB

Multi-bit LSB steganography proposed in this research is performed using several bit insertion models, namely 1-bit, 2-bit, 3-bit, and 4-bit LSB storage, respectively. Each character, which has been experiencing encryption process is inserted into a cover image, bit-per-bit into the RGB components of the image.

The LSB insertion schemes are used to store each character (i.e., 8 bits) of the information into the RGB components of a pixel. In this case, the more are the number of inserted bits, the fewer are the pixels required to store the information: 1-bit insertion uses 3 pixels; 2-bit insertion uses less than 3 pixels; 3-bit insertion uses exactly 1 pixel; and 4-bit insertion uses less than 1 pixel. In addition, unused RGB color components can be used to store the next character.

To illustrate this process, let the RGB components of three pixels are (234, 89, 128) for the first pixel, (128, 251, 60) for the second pixel, and (206, 36, 60) for the third pixel. Let the character to be inserted is character A, which is 65 (01000001) in ASCII code.

Inserting character A in 1-bit LSB storage:

R	1	1	1	0	1	0	1	**0**
G	0	1	0	1	1	0	0	**1**
B	1	0	0	0	0	0	0	**0**
R	1	0	0	0	0	0	0	**0**
G	1	1	1	1	1	0	1	**0**
B	0	0	1	1	1	1	0	**0**
R	1	1	0	0	1	1	1	**0**
G	0	0	1	1	1	0	1	**1**

From this 1-bit insertion scheme, the character A needs three pixels, with one bit of the B component of the third pixel will be spared for next insertion.

With 2-bit insertion scheme, the character A will need two pixels, but two G and B components of the second pixel are reserved for next insertion.

Inserting character A in 2-bit LSB storage:

R	1	1	1	0	1	0	**0**	**1**
G	0	1	0	1	1	0	**0**	**0**
B	1	0	0	0	0	0	**0**	**0**
R	1	0	0	0	0	0	**0**	**1**

If we continue with 3-bit insertion scheme, the character only need to use exactly one pixel, but still has one bit reserved for next insertion.

Inserting character A in 3-bit LSB storage:

R	1	1	1	0	1	**0**	**1**	**0**
G	0	1	0	1	1	**0**	**0**	**0**
B	1	0	0	0	0	0	**0**	**1**

Lastly, using 4-bit insertion scheme, the character will only use two R and G components of the first pixel.

Inserting character A in 4-bit LSB storage:

R	1	1	1	0	**0**	**1**	**0**	**0**
G	0	1	0	1	**0**	**0**	**0**	**1**

In this scheme, the blue component of the pixel could be reserved for further insertions.

IV. Implementation of Systems and Methods

An example image used in this research is one containing 4×4 pixels, which can accommodate up to 48 bits. If converted to characters, the image can accommodate up to 6 characters using 1-bit LSB storage. The following are the steps for 1-bit plaintext rotation and the 1-bit key rotation.

Step 1. Read Plaintext:

S	e	c	r	e	t
83	101	99	114	101	116
01010011	01100101	01100011	01110010	01100101	01110100

Step 2. Rotate left one bit:

1010011**0**	1100101**0**	1100011**0**	1110010**0**	1100101**0**	1110100**0**
166	202	198	228	202	232

Step 3. Read Key:

W	o	r	l	d
87	111	114	108	100
01010111	01101111	01110010	01101100	01100100

Step 4. Rotate Key right one bit:

10101011	**1**0110111	**0**0111001	**0**0110110	**0**0110010
171	183	57	54	50

Step 5. Generate ciphertext:

10100110	11001010	11000110	11100100	11001010	11101000	
10101011	10110111	00111001	00110110	00110010	00100000	$\oplus$
00001101	01111101	11111111	11010010	11111000	11001000	
13	125	255	210	248	200	
,	Ż	ß	Ò	ø	È	

The resulted ciphertext is then concealed into an image using multi-bit LSB storage models, such as that illustrated in the following. In the test conducted, a 4×4-pixel image was used as a cover image; this could accommodate $4 \times 4 \times 3$ bits = 48 bits (i.e., six characters) using 1-bit LSB.

The RGB decimal representation for the 4×4 pixel cover image is:

Byte-1	R	65	58	187	190
	G	87	176	149	241
	B	116	249	167	229
Byte-2	R	222	171	203	63
	G	223	74	221	68
	B	150	134	191	182
Byte-3	R	246	101	32	89
	G	205	204	143	53
	B	128	84	243	163
Byte-4	R	192	151	36	11
	G	125	130	167	60
	B	226	101	153	23

In order to use the 4×4 pixel cover image as a container for the inserted ciphertext, the RGB components are converted into binary forms:

Byte-1	R	01000001	00111010	10111011	10111110
	G	01010111	11011111	11001101	01111101
	B	01110100	11111001	10100111	11100101
Byte-2	R	11011110	10101011	11001011	00111111
	G	11011111	01001010	11011101	01000100
	B	10010110	10000110	10111111	10110110
Byte-3	R	11110110	01100101	00100000	01011001
	G	11001101	11001100	10001111	00110101
	B	10000000	01010100	11110011	10100011
Byte-4	R	11000000	10010111	00100100	00001011
	G	01111101	10000010	10100111	00111100
	B	11100010	01100101	10011001	00010111

In this illustration, the ciphertext to be inserted: ,ŻßÒøC contains 48 bits.

Stego-image obtained using 1-bit LSB:

Byte-1	R	**64**	58	**186**	**191**
	G	**86**	**177**	149	241
	B	116	249	**166**	229
Byte-2	R	**223**	171	203	63
	G	223	**75**	221	**69**
	B	150	**135**	191	**183**
Byte-3	R	**247**	101	**33**	89
	G	205	204	**142**	53
	B	128	84	243	163
Byte-4	R	**193**	**150**	36	**10**
	G	**124**	130	**166**	**61**
	B	226	101	**152**	23

The boldface numbers in the illustration show the bits that are changed in values as a consequence of the insertions. As can be seen, there are 21 bits and therefore, using Eq. 3

and 2, we can calculate MSE and PSNR of the stego-image, respectively.

$MSE = \frac{1}{16}(7) = 0.4375$

$PSNR = 10 \times log_{10}(\frac{249^2}{0.4375}) = 51.5142$

By the same manner, we can obtained stego-images for 2-, 3- and 4-bit LSB and calculate the corresponding MSE and PSNR as illustrated in the following.

Stego-image obtained using 2-bit LSB:

Byte-1	R	**64**	**57**	187	**191**
	G	**84**	**177**	149	**243**
	B	**119**	**251**	167	**231**
Byte-2	R	**223**	**170**	**202**	**60**
	G	**221**	**75**	**220**	68
	B	**148**	**135**	**189**	**183**
Byte-3	R	247	101	33	89
	G	205	204	**142**	53
	B	128	84	243	163
Byte-4	R	193	150	36	10
	G	125	130	167	60
	B	226	101	152	23

$MSE = \frac{1}{16}(10.67) = 0.6667$

$PSNR = 10 \times log_{10}(\frac{249^2}{0.6667}) = 49.6846$

Stego-image obtained using 3-bit LSB:

Byte-1	R	**64**	**63**	**191**	**188**
	G	**83**	**182**	**151**	**246**
	B	**113**	**255**	**166**	**231**
Byte-2	R	**220**	170	202	60
	G	**218**	**75**	**220**	68
	B	**144**	134	191	182
Byte-3	R	247	101	33	89
	G	128	84	243	163
	B	205	204	**142**	53
Byte-4	R	193	150	36	10
	G	125	130	167	60
	B	226	101	152	23

$MSE = \frac{1}{16}(18) = 1.125$

$PSNR = 10 \times log_{10}(\frac{249^2}{1.125}) = 47.4142$

Stego-image obtained using 4-bit LSB:

Byte-1	R	**64**	**61**	**189**	**184**
	G	**93**	**191**	**146**	**244**
	B	**119**	**255**	**175**	**227**
Byte-2	R	222	171	203	63
	G	218	75	220	68
	B	150	134	191	182
Byte-3	R	246	101	32	89
	G	205	204	142	53
	B	128	84	243	163
Byte-4	R	192	151	36	11
	G	125	130	167	60
	B	226	101	152	23

$MSE = \frac{1}{16}(19.333) = 1.2083$

$PSNR = 10 \times log_{10}(\frac{249^2}{1.2083}) = 47.1022$

V. Results and Discussion

In this section, the process of combining steganography and cryptography is evaluated. The test uses four *.docx* files and three JPEG image files. The four *.docx* files have the following sizes and character lengths: 13.4 KB and 1450 characters; 13.8 KB and 2149 characters; 14.0 KB and 1925 characters; and 21 KB and 4746 characters. The three images have the following resolutions and sizes: 250×250 and 14.7 KB; 141×250 and 30.3 KB; and 400×250 and 56.2 KB. Meanwhile, the content of the *.docx* files were extracted into plaintext using C# modules, followed by encryption using the modified Vernam cipher algorithm. The ciphertext results were inserted in the message using the multi-bit LSB method.

Table III, Table IV, and Table V show the calculation results of MSE and PSNR as well as percentage of pixel values usage for each document file inserted into the three images using 1-, 2-, 3-, and 4-bit LSB schemes.

Both MSE and PSNR values obtained from the three stego images using all four bit LSB schemes show that the resulting stego images are good in concealing the secret images. In this case small MSE values and PSNR values ≥ 40 indicate promising results.

Important results are also obtained in terms of pixel usage which show considerable low percentages for each document inserted. The average of pixel usage percentage for the four documents are summarized in Table VI. As could be seen, the average pixel usages are reciprocal to the image resolution, whereby lower resolution images use higher percentage of pixel. But this usage is decreasing as the number of bits inserted are increased. Highest percentage of average pixel usage is 19.42% with 1-bit LSB insertion in low image resolution, and lowest percentage of 1.71% with 4-bit LSB found in high image resolution.

VI. Conclusion

The results of this study indicate that the level of data security can be enhanced by combining steganography methods with cryptography algorithms. The resolution of the

TABLE III
INSERTING ENCRYPTED *.docx1–.docx4* FILES IN IMAGE 1

Image Resolution (WxH): 250x250

	Filename (bytes)			
	.doc1 (1450)	.doc2 (2149)	.doc3 (1925)	.doc4 (4746)
1-bit LSB				
MSE	0.031088	0.04464	0.041258	0.101914
PSNR	63.20487	61.63356	61.97565	58.04843
Pixel Usage(%)	6.187	8.213	9.169	20.249
2-bit LSB				
MSE	0.081968	0.119098	0.105104	0.267328
PSNR	58.99436	57.37173	57.91461	53.86035
Pixel Usage(%)	3.093	4.584	4.106	10.124
3-bit LSB				
MSE	0.24997	0.35535	0.33952	0.791514
PSNR	54.15186	52.62418	52.82214	49.14621
Pixel Usage(%)	2.062	3.056	2.737	6.749
4-bit LSB				
MSE	0.82294	1.134906	0.942256	2.638117
PSNR	48.97707	47.5812	48.38911	43.91786
Pixel Usage(%)	1.547	2.292	2.053	5.062

TABLE IV
INSERTING ENCRYPTED *.docx1–.docx4* FILES IN IMAGE 2

Image Resolution (WxH): 141x250

	Filename (bytes)			
	.doc1 (1450)	.doc2 (2149)	.doc3 (1925)	.doc4 (4746)
1-bit LSB				
MSE	0.056302	0.080406	0.072047	0.179858
PSNR	60.55712	59.00949	59.48623	55.5131
Pixel Usage(%)	10.969	16.257	14.562	35.903
2-bit LSB				
MSE	0.13583	0.198458	0.175338	0.455347
PSNR	56.73215	55.08571	55.62364	51.47898
Pixel Usage(%)	5.484	8.128	7.281	17.951
3-bit LSB				
MSE	0.40066	0.56712	0.520104	1.278477
PSNR	52.03462	50.52565	50.9015	46.99547
Pixel Usage(%)	3.656	5.419	4.854	11.967
4-bit LSB				
MSE	1.116605	1.684614	1.400983	3.914222
PSNR	47.58431	45.7974	46.59808	42.13595
Pixel Usage(%)	2.742	4.064	3.64	8.975

cover image and character length of the encrypted message considerably affect the MSE and PSNR values. The results obtained from calculatiing MSE and PSNR values indicate that the use of 1-bit LSB is superior to that of 2-, 3-, or 4-bit LSB. In addition, according to the results, the use of 4-bit LSB is still feasible because the PSNR value for 4-bit LSB is above 40 db. Furthermore, the smaller percentage of pixel usage in 4-bit LSB storage also indicates satisfactory results even in a relatively low image resolution, that is $< 9\%$ in an image with 141×250 relosution. The application of the multi-bit LSB method in steganographic activities is thus advantageous, in term of each pixel can hold more message bits compared with the conventional LSB method.

TABLE V
INSERTING ENCRYPTED *.docx1–.docx4* FILES IN IMAGE 3

Image Resolution (WxH): 400x250

	Filename (bytes)			
	.doc1 (1450)	.doc2 (2149)	.doc3 (1925)	.doc4 (4746)
1-bit LSB				
MSE	0.01951	0.028296	0.02598	0.063483
PSNR	65.22823	63.61345	63.98441	60.1042
Pixel Usage(%)	3.867	5.73	5.133	12.656
2-bit LSB				
MSE	0.051026	0.074667	0.065076	0.168236
PSNR	61.05283	59.39953	59.99655	55.87159
Pixel Usage(%)	1.933	2.865	2.567	6.328
3-bit LSB				
MSE	0.151606	0.215456	0.198946	0.47618
PSNR	56.32362	54.7972	55.14343	51.35309
Pixel Usage(%)	1.289	1.91	1.711	4.218
4-bit LSB				
MSE	0.446203	0.65474	0.549803	1.50513
PSNR	51.63547	49.97011	50.72872	46.35506
Pixel Usage(%)	0.967	1.432	1.283	3.164

TABLE VI
AVERAGE PERCENTAGE OF PIXEL USAGE FOR *.docx1–.docx4*

	Image resolution		
	(250×250)	(141×250)	(400×250)
1-bit LSB	10.95	19.42	6.85
2-bit LSB	5.48	9.71	3.42
3-bit LSB	3.65	6.47	2.28
4-bit LSB	2.74	4.86	1.71

REFERENCES

[1] T. Barnett, S. Jain, U. Andra, and T. Khurana, "Cisco visual networking index (vni) complete forecast update," https://www.cisco.com, 2018, [Online; accessed 19-August-2019].

[2] V. Saxena and J. Gupta, "Collusion attack resistant watermarking scheme for colored image using dct," *IAENG International Journal of Computer Science*, vol. 34, no. 2, pp. 171–177, 2007.

[3] D. Rachmawati, A. Sharif, and M. A. Abdurrazzaq, "Analysis of modified least significant bit polynomial function algorithm for securing digital image," in *Proceeding of the 3rd International Conference on Computing and Applied Informatics 2018*, pp. 1–9.

[4] A. Agusnady, O. S. Sitompul, Tulus, B. S. Sembiring, and T. Qowidho, "The effect of plaintext length on round trip time with client server based advanced encryption standard (aes)," in *Proceeding of International Conference on Computer Science and Applied Mathematic 2018*, pp. 1–5.

[5] O. S. Sitompul, Z. Situmorang, F. R. Naibaho, and E. B. Nababan, "Steganography with highly random linear congruential generator for security enhancement," in *IEEE Third International Conference on Informatics and Computing 2018*, pp. 1–6.

[6] O. S. Sitompul, Handrizal, N. H. Pasaribu, and E. B. Nababan, "Hybrid rc4 and affine ciphers to secure short message service on android," in *IEEE Third International Conference on Informatics and Computing 2018*, pp. 1–6.

[7] B. Datta, P. K. Pal, and S. K. Bandyopadhyay, "Multi-bit data hiding in randomly chosen lsb layers of an audio," in *International Conference on Information Technology 2016*, p. 283–287.

[8] M. Kaur and M. Juneja, "A new lsb embedding for 24-bit pixel using multi-layered bitwise xor," in *International Conference on Inventive Computation Technologies 2016*, pp. 1–5.

[9] R.-J. Chen, Y.-C. Chen, and S.-J. Horng, "Data hiding using flexible multi-bit mer," in *International Symposium on Biometrics and Security Technologies 2013*, pp. 24–31.

[10] S. Goel, S. Gupta, and N. Kaushik, "Image steganography – least significant bit with multiple progressions," in *Proceedings of the*

3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications 2014, pp. 105–112.
[11] P. Hellekalek and S. Wegenkittl, "Empirical evidence concerning aes," *ACM Transactions on Modeling and Computer Simulation*, vol. 13, no. 4, pp. 322–333, 2003.
[12] B. S. Champakamala, K. Padmini, and D. K. Radhika, "Least significant bit algorithm for image steganography," *Int. J. Adv. Comput. Technol.*, vol. 3, no. 4, pp. 34–38, 2014.
[13] S. A. Laskar and K. Hemachandran, "High capacity data hiding using lsb steganography and encryption," *Int. J. Database Manag. Syst.*, vol. 4, no. 6, pp. 57–68, 2012.
[14] A. Yahya, *Steganography Techniques for Digital Images*. Springer, 2019.
[15] L. Y. Por, D. Beh, T. F. Ang, and S. Y. Ong, "An enhanced mechanism for image steganography using sequential colour cycle algorithm," *Int. Arab J. Inf. Techn.*, vol. 10, no. 1, pp. 51–60, 2013.

Erna Budhiarti Nababan Completed her PhD on Science and Management Systems from Universiti Kebangsaan Malaysia, Selangor in 2010. She is currently a senior lecturer at Department of Information Technology, Universitas Sumatera Utara, Medan, Indonesia. She is a member of IAENG since 2017.

Goklas Tomu Simbolon Was formerly a master degree student at Department of Computer Science, Universitas Sumatera Utara. He completed his master's degree in 2019.

Opim Salim Sitompul Completed his PhD on Information Science from Universiti Kebangsaan Malaysia, Selangor in 2005. He is a professor at Department of Information Technology, Universitas Sumatera Utara, Medan, Indonesia. He is a member of IAENG since 2017.