Vol. 5 No. 3, 2024, pp. 286-298 **Pepadun Journal**

Text Data Embedding into Images Using Chaotic Least Significant Bit Encoding Steganography

[*1]Allwine and [2]Riza Sawitri

[1]Dept. of Computer Science, University of Lampung, Jl. Prof. Dr. Ir. Sumantri Brojonegoro No. 1, Bandar Lampung, Lampung Province, Indonesia, 35145
[2]Dept. of Mathematics, University of Lampung, Jl. Prof. Dr. Ir. Sumantri Brojonegoro No. 1, Bandar Lampung, Lampung Province, Indonesia, 35145
e-mail: [*1]**allwine@fmipa.unila.ac.id**, [2]rizasawitri@fmipa.unila.ac.id

Abstract - The rapid growth of computer networks and the internet has facilitated global communication and the exchange of various types of information, including text, images, videos, and audio. However, this technological advancement has also given rise to cybercrime, such as data theft. Steganography, the practice of concealing messages within another medium, offers a solution to protect sensitive information. With the evolution of digital technology, digital image files have become a popular medium for embedding hidden messages. This application employs a Chaotic Least Significant Bit (LSB) steganography method, which leverages randomness to determine the locations for embedding messages. This randomization enhances security by making the hidden data more difficult to detect during transmission over the internet. The LSB method ensures that only individuals with a specific key can access the concealed messages. Its strength lies in its ability to maintain the secrecy of the embedded data, effectively preventing unauthorized access. As a result, the LSB method serves as a critical tool for safeguarding confidential information and ensuring secure data transmission in a digital environment.

Keywords— Digital Image; Least Significant Bit; Steganography.

1. INTRODUCTION

Steganography is the art and science of hiding a message in such a way that no one else, apart from the intended recipient, knows about the message [1]. The advantage of steganography is its ability to make a secret message invisible, or not invite other people who do not know to care or be curious. Steganography usually consists of two systems, namely a system for hiding messages and a system for retrieving messages [2]. The steganography technique that will be used is to use digital images as a container for the message to be hidden. The use of a container in the form of a digital image is due to the limitations of human sensitivity in terms of visualization systems. The output of this steganography has the same form of perception as the original, of course, perception here is limited by the ability of human senses, but not by computers or other digital processors. With the development of the multimedia world, steganography uses these multimedia files as a cover to hide messages [3].

Considering the advantages of steganography on images, and also with the increasing development of hardware, a steganography application will be designed as one way to secure information. The steganography method used is a method based on a Chaos System [4]. In physics and mathematics, Chaos Theory is related to an activity or habit of a dynamic non-linear system, which for some conditions displays a random phenomenon (chaos). One of the existing chaos methods is Chaos Least Significant Bit Encoding (CLSBE) [5]. This method was taken because it is one of the simple methods and does not require an overly complicated algorithm [6][7]. One of the simplest methods in steganography is hiding messages in the Least Significant Bit of each pixel in its cover image, because in digital images changes of one or two bits in each pixel will not be visible to the naked eye [8][9]. That is the process that must be gone through by the method. Before the message hiding process is carried out, the hiding location is first determined.

2. RESEARCH METHODS

2.1 Steganography

Steganography is the science and art of hiding secret messages in such a way that the existence of the message is undetectable by human senses. Steganography requires two properties, the container and the secret data to be hidden. Digital steganography uses digital media as a container, such as images, sound, text, and video. The hidden secret data can also be images, sound, text, or video [10][11]. Steganography is different from cryptography, where a third party can detect the presence of data (ciphertext), because the results of cryptography are data that is different from its original form and usually the data seems messy, but can be returned to its original form [12]. An illustration of the difference between cryptography and steganography can be seen in Figure 1.

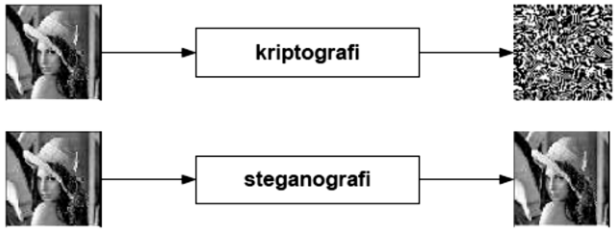

Figure 1. Illustration of cryptography and steganography in digital images.

Digital steganography uses digital media as a container, such as images, sound, text, and video. While the hidden secret data can be any file. Media that has been inserted with data is called stego-message. The process of hiding data into media is called embedding, while the reverse process is called extraction. The addition of an optional key is intended to further increase security [13]. Steganography is defined by [14] as follows, “Steganography is the art and science of communicating by concealing the existence of the communication itself”. Unlike cryptography, where an adversary can detect, capture and modify a message without being able to compromise the security of the cryptosystem, the goal of steganography is to hide a message within a “harmless” message in such a way as to prevent an adversary from detecting that the message is hidden. Figure 2 below shows the basic method of how steganography works.

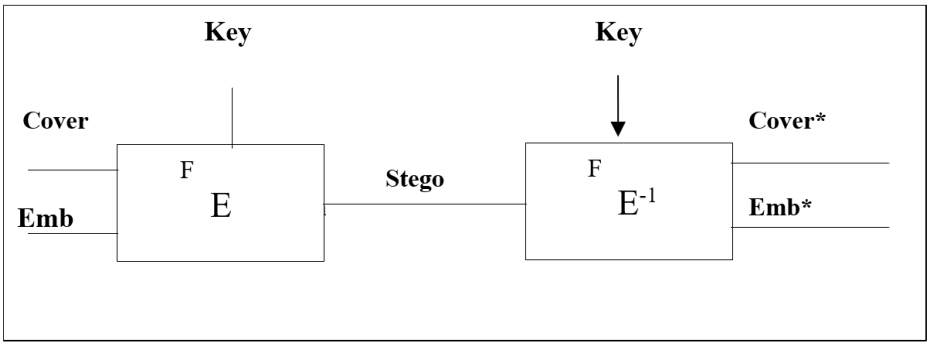

Figure 2. How steganography works in general [15].

Information:
FE : Embedding (Combining cover files with secret files)
FE^{-1} : Extraction (Extracting secret files from cover files)
Cover : Data file that covers the emb to be hidden
Emb : Message to be inserted
Stego : Data files that have hidden messages embedded in them.

A cover refers to an image, audio, or video file, while an Emb represents a secret message to be embedded within it. The Key is a parameter that guides the message embedding process, and the resulting Stego-file

is the one containing the hidden message. One critical aspect to consider is the size of the information to be embedded, as larger messages require greater modification of the cover data, increasing the likelihood of detection. This also impacts the quality of the cover image, as embedding larger text messages can lead to noticeable changes. Consequently, the image size must be proportionate to the message to ensure the hidden data remains inconspicuous. The simplest steganography technique involves concealing the message in the Least Significant Bit (LSB) of each pixel in the cover image. In digital images, altering one or two bits per pixel is virtually imperceptible to the human eye. Before embedding the message, the specific location for hiding is determined using a defined method to ensure secure and undetectable placement within the digital image. The method to determine the location is as follows:

1. For example, inserting the text "MELATI" with the key "SUKMA" into the image in Table 1 below. If represented in binary, the word "MELATI" becomes:

Table 1. Binary of MELATI.

Character	ASCII Code (Decimal)	Binary
M	77	01001101
E	69	01000101
L	76	01001100
A	65	01000001
T	84	01010100
I	73	01001001

While the keyword that will be used is "SUKMA", the key here functions as message security so that the message cannot be seen in Table 2. The following is the decimal value of the keyword that will be used.

Table 2. Binary of SUKMA.

Character	ASCII Code (Decimal)	Binary
S	83	01010011
U	85	01010101
K	75	01001011
M	77	01001101
A	65	01000001

2. For RGB cover-image $c(x, y) = [Rc, Gc, Bc]$ of size $M \times N$, determine a random seed and generate pseudorandom number then arrange them into an RGB pseudo-image $p(x, y) = [Rp, Gp, Bp]$ of size $M \times N$. Suppose there is an image such as Figure 3 below which has an image size of 412 x 270 pixels.

Figure 3. 412 x 270 pixels image (Ariyus, 2009).

Tabel 3. Cover-image RGB from Figure 3.

R	G	B
2	20	0
10	27	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
10	30	0
23	30	0
25	33	0
26	37	0
27	37	0
29	45	0
29	49	0
32	50	0
32	51	0
33	54	0

R	G	B
37	55	0
37	55	1
39	57	6
40	59	8
45	60	9
46	61	9
48	66	9
48	69	10
49	70	13
52	71	15
52	72	15
56	73	16
R	G	B
57	73	19
64	74	21
66	75	22
68	78	22
68	82	24
68	83	28
69	87	30
70	88	30
70	88	32
72	89	33
72	89	36
74	89	36
74	90	41
76	94	43

R	G	B
77	95	43
77	97	45
78	97	45
83	101	46
90	102	46
94	103	47
95	103	52
96	103	57
96	104	58
96	107	61
98	110	62
101	110	65
103	110	67
103	115	68
103	116	68
104	116	68
109	119	69
110	122	72
110	126	84
116	128	88
122	137	89
124	138	97
127	138	98
134	139	99
134	144	103
143	144	103
146	159	126

Table 4. Pseudo-image RGB from Figure 3.

R	G	B
72	208	0
104	204	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
104	56	0
60	56	0
132	164	0
168	52	0
204	52	0
20	84	0

R	G	B
20	228	0
128	8	0
128	44	0
164	152	0
52	188	0
52	188	36
124	4	216
160	76	32
84	112	68
120	148	68
192	72	68
192	180	104
228	216	212
80	252	28
80	32	28
224	68	64
R	G	B
4	68	172
0	104	244
72	140	24
144	248	24
144	136	96
144	172	240

R	G	B
180	60	56
216	96	56
216	96	128
32	132	164
32	132	16
104	132	16
104	168	196
176	56	12
212	92	12
212	164	84
248	164	84
172	52	120
168	88	120
56	124	156
92	124	80
128	124	4
128	160	40
128	12	148
200	120	184
52	120	36
124	120	108

3. Calculate the distance between $c\ (x, y)$ and $p\ (x, y)$ from Table 3 and Table 4 using the distance formula for two vectors presented in Equation (1). The results obtained are presented in Table 5.

$$d(p, c) = \sqrt{\left(R_c - R_p\right)^2 + \left(G_c - G_p\right)^2 + \left(G_c - G_p\right)^2} \quad (1)$$

Below are few examples from data to illustrate how the distances were calculated:

Example 1: Pixel at (0, 0)

Cover Image RGB: (2, 20, 0)

Pseudo Image RGB: (72, 208, 0)

Calculating the distance:

$$d = \sqrt{(2-72)^2 + (20-208)^2 + (0-0)^2} \approx 200.60907 \quad (2)$$

Example 2: Pixel at (1, 1)

Cover Image RGB: (10, 30, 0)

Pseudo Image RGB: (104, 56, 0)

Calculating the distance:

$$d = \sqrt{(10-104)^2 + (30-56)^2 + (0-0)^2} \approx 97.529483 \quad (3)$$

Table 5. Distance of two vectors.

Coordinate		Cover-image			Pseudo-image			Distance of two
x	y	R	G	B	R	G	B	Vectors (d)
0	0	2	20	0	72	208	0	200.60907
1	0	10	27	0	104	204	0	200.41208
2	0	10	30	0	104	56	0	97.529483
3	0	10	30	0	104	56	0	97.529483
4	0	10	30	0	104	56	0	97.529483
5	0	10	30	0	104	56	0	97.529483
6	0	10	30	0	104	56	0	97.529483
7	0	10	30	0	104	56	0	97.529483
8	0	10	30	0	104	56	0	97.529483
9	0	10	30	0	104	56	0	97.529483
0	1	10	30	0	104	56	0	97.529483
1	1	10	30	0	104	56	0	97.529483
2	1	10	30	0	104	56	0	97.529483
3	1	10	30	0	104	56	0	97.529483
4	1	10	30	0	104	56	0	97.529483
5	1	10	30	0	104	56	0	97.529483
6	1	10	30	0	104	56	0	97.529483
7	1	10	30	0	104	56	0	97.529483
8	1	23	30	0	60	56	0	45.221676
9	1	25	33	0	132	164	0	169,14491
0	2	26	37	0	168	52	0	142.79006
1	2	27	37	0	204	52	0	177.63446
2	2	29	45	0	20	84	0	40.024992
3	2	29	49	0	20	228	0	179,22611
4	2	32	50	0	128	8	0	104,7855
5	2	32	51	0	128	44	0	96.25487
6	2	33	54	0	164	152	0	163,60012
7	2	37	55	0	52	188	0	133.84319
8	2	37	55	1	52	188	36	138.34377
9	2	39	57	6	124	4	216	232,66714
0	3	40	59	8	160	76	32	123.55161
1	3	45	60	9	84	112	68	87.783825
2	3	46	61	9	120	148	68	128.55349
3	3	48	66	9	192	72	68	155,73375
4	3	48	69	10	192	180	104	204,6778
5	3	49	70	13	228	216	212	304.89014
6	3	52	71	15	80	252	28	183.61372
7	3	52	72	15	80	32	28	50.52722
8	3	56	73	16	224	68	64	174.79416
9	3	57	73	19	4	68	172	161.99691
0	4	64	74	21	0	104	244	233.93375
1	4	66	75	22	72	140	24	65.306967
2	4	68	78	22	144	248	24	186,22567

Coordinate		Cover-image			Pseudo-image			Distance of two Vectors (d)
x	y	R	G	B	R	G	B	
3	4	68	82	24	144	136	96	117.79643
4	4	68	83	28	144	172	240	242,15904
5	4	69	87	30	180	60	56	117,15801
6	4	70	88	30	216	96	56	148,51263
7	4	70	88	32	216	96	128	174.91712
8	4	72	89	33	32	132	164	143.56183
9	4	72	89	36	32	132	16	62.040309
0	5	74	89	36	104	132	16	56.115951
1	5	74	90	41	104	168	196	176.09373
2	5	76	94	43	176	56	12	111,37774
3	5	77	95	43	212	92	12	138.54602
4	5	77	97	45	212	164	84	155.67595
5	5	78	97	45	248	164	84	186.84218
6	5	83	101	46	172	52	120	125,6901
7	5	90	102	46	168	88	120	108.42509
8	5	94	103	47	56	124	156	117,3286
9	5	95	103	52	92	124	80	35,128336
0	6	96	103	57	128	124	4	65.375837
1	6	96	104	58	128	160	40	66.962676
2	6	96	107	61	128	12	148	132.73281
3	6	98	110	62	200	120	184	159.33612
4	6	101	110	65	52	120	36	57.810034
5	6	103	110	67	124	120	108	47.138095
6	6	103	115	68	124	44	144	106,10372
7	6	103	116	68	124	80	144	86.677563
8	6	104	116	68	160	80	144	101.03465
9	6	109	119	69	84	188	180	133.06765
0	7	110	122	72	120	40	32	91.782351
1	7	110	126	84	120	184	208	137.25888
2	7	116	128	88	80	0	96	133,20661
3	7	122	137	89	40	68	132	115.47294
4	7	124	138	97	112	104	164	76.085478
5	7	127	138	98	220	104	200	142,15836
6	7	134	139	99	216	140	236	159,66841
7	7	134	144	103	216	64	124	116.46888
8	7	143	144	103	28	64	124	141.65451
9	7	146	159	126	136	92	184	89.179594

4. Hiding starts from the location with the smallest distance to the largest distance. Distance sorting uses a data sorting algorithm. This LSB (Least Significant Bit) insertion method is to insert a message by replacing the 8th, 16th and 24th bits in the binary representation of the image file with the binary representation of the secret message to be hidden. Thus, in each pixel of a 24-bit BMP image file, 3 bits of message can be inserted as shown in Table 6.

Table 6. RGB values after text insertion.

RGB Bit Values Before Text Insertion			Bit Value After Text Insertion			RGB Values After Text Insertion		
R	G	B	R	G	B	R	G	B
01011111	01100111	00110100	01011110	01100111	00110100	94	103	52
00011101	00101101	00000000	00011100	00101101	00000000	28	45	0
00010111	00011110	00000000	00010110	00011110	00000000	22	30	0
01100111	01101110	01000011	01100111	01101110	01000010	103	110	66
00110100	01001000	00001111	00110100	01001000	00001110	52	72	14
01001010	01011001	00100100	01001011	01011000	00100101	75	88	37
01100101	01101110	01000001	01100100	01101111	01000000	100	111	64
01001000	01011001	00100100	01001000	01011001	00100100	72	89	36
01000010	01001011	00010110	01000010	01001011	00010110	66	75	22
01100000	01100111	00111001	01100000	01100110	00111001	96	102	57
01100000	01101000	00111010	01100000	01101000	00111010	96	104	58
01111100	10001010	01100001	01111101	10001010	01100000	125	138	96

RGB Bit Values Before Text Insertion			Bit Value After Text Insertion			RGB Values After Text Insertion		
R	G	B	R	G	B	R	G	B
01100111	01110100	01000100	01100111	01110100	01000100	103	116	68
00101101	00111100	00001001	00101101	00111100	00001001	45	60	9
10010010	10011111	01111110	10010010	10011110	01111110	146	158	126
01101110	01111010	01001000	01101110	01111011	01001000	110	123	72
00100000	00110011	00000000	00100001	00110010	00000000	33	50	0
00001010	00011110	00000000	00001011	00011111	00000001	11	31	1
00001010	00011110	00000000	00001010	00011110	00000001	10	30	1
00001010	00011110	00000000	00001010	00011111	00000000	10	31	0
00001010	00011110	00000000	00001010	00011111	00000001	10	31	1
00001010	00011110	00000000	00001010	00011111	00000000	10	31	0
00001010	00011110	00000000	00001011	00011111	00000000	11	31	0
00001010	00011110	00000000	00001010	00011111	00000000	10	31	0
00001010	00011110	00000000	00001011	00011110	00000000	11	30	0
00001010	00011110	00000000	00001010	00011110	00000000	10	30	0
00001010	00011110	00000000	00001011	00011110	00000001	11	30	1
00001010	00011110	00000000	00001010	00011110	00000001	10	30	1
00001010	00011110	00000000	00001010	00011110	00000000	10	30	0
00001010	00011110	00000000	00001010	00011110	00000000	10	30	0
00001010	00011110	00000000	00001010	00011110	00000000	10	30	0
00001010	00011110	00000000	00001010	00011110	00000000	10	30	0
00001010	00011110	00000000	00001010	00011110	00000000	10	30	0
01101000	01110100	01000100	01101000	01110100	01000100	104	116	68
00100000	00110010	00000000	00100000	00110010	00000000	32	50	0
0110111	01110011	01000100	01100111	01110011	01000100	103	115	68
01001100	01011110	00101011	01001100	01011110	00101011	76	94	43
01111010	10001001	01011001	01111010	10001001	01011001	122	137	89
10000110	10010000	01100111	10000110	10010000	01100111	134	144	103
01000101	01010111	00011110	01000101	01010111	00011110	69	87	30
01011110	01100111	00101111	01011110	01100111	00101111	94	103	47
01000100	01010010	00011000	01000100	01010010	00011000	68	82	24
00101000	00111011	00001000	00101000	00111011	00001000	40	59	8
01010011	01100101	00101110	01010011	01100101	00101110	83	101	46
00101110	00111101	00001001	00101110	00111101	00001001	46	61	9
01100000	01101011	00111101	01100000	01101011	00111101	96	107	61
01101101	01110111	01000101	01101101	01110111	01000101	109	119	69
01110100	10000000	01011000	01110100	10000000	01011000	116	128	88
00100101	00110111	00000000	00100101	00110111	00000000	37	55	0
01101110	01111110	01010100	01101110	01111110	01010100	110	126	84
00100101	00110111	00000001	00100101	00110111	00000001	37	55	1
01001101	01011111	00101011	01001101	01011111	00101011	77	95	43
10001111	10010000	01100111	10001111	10010000	01100111	143	144	103
01111111	10001010	01100010	01111111	10001010	01100010	127	138	98
00011010	00100101	00000000	00011010	00100101	00000000	26	37	0
01001000	01011001	00100001	01001000	01011001	00100001	72	89	33
01000110	01011000	00011110	01000110	01011000	00011110	70	88	30
01001101	01100001	00101101	01001101	01100001	00101101	77	97	45
00110000	01000010	00001001	00110000	01000010	00001001	48	66	9
01100010	01101110	00111110	01100010	01101110	00111110	98	110	62
10000110	10001011	01100011	10000110	10001011	01100011	134	139	99
00111001	01001001	00010011	00111001	01001001	00010011	57	73	19
00100001	00110110	00000000	00100001	00110110	00000000	33	54	0
00011001	00100001	00000000	00011001	00100001	00000000	25	33	0
00111000	01001001	00010000	00111000	01001001	00010000	56	73	16
01000110	01011000	00100000	01000110	01011000	00100000	70	88	32
01001010	01011010	00101001	01001010	01011010	00101001	74	90	41
00011011	00100101	00000000	00011011	00100101	00000000	27	37	0
00011101	00110001	00000000	00011101	00110001	00000000	29	49	0
00110100	01000111	00001111	00110100	01000111	00001111	52	71	15
01000100	01001110	00010110	01000100	01001110	00010110	68	78	22
01001110	01100001	00101101	01001110	01100001	00101101	78	97	45
00001010	00011011	00000000	00001010	00011011	00000000	10	27	0

RGB Bit Values Before Text Insertion			Bit Value After Text Insertion			RGB Values After Text Insertion		
R	G	B	R	G	B	R	G	B
00000010	00010100	00000000	00000010	00010100	00000000	2	20	0
00110000	01000101	00001010	00110000	01000101	00001010	48	69	10
00100111	00111001	00000110	00100111	00111001	00000110	39	57	6
01000000	01001010	00010101	01000000	01001010	00010101	64	74	21
01000100	01010011	00011100	01000100	01010011	00011100	68	83	28
00110001	01000110	00001101	00110001	01000110	00001101	49	70	13

2.2 Random Phenomenon (Chaos Insertion)

In the context of Chaotic Least Significant Bit (LSB) Encoding Steganography, the concept of "random phenomenon (chaos)" plays a significant role during the insertion of secret data into cover images. This method enhances data security and minimizes detection risks. Chaotic Systems are systems that exhibit sensitive dependence on initial conditions, meaning small changes can lead to vastly different outcomes. Examples include logistic maps and other chaotic functions. The chaotic behaviour introduces a level of randomness that makes it difficult to predict which pixels will be altered, thereby increasing the security of the embedded data.

Before embedding secret data into the cover image, the bits of the secret message are shuffled using a chaotic map. This shuffling ensures that the order of bits is randomized, making it harder for attackers to detect patterns. The chaotic sequence generated from the chaotic map determines which pixels will be modified for embedding. This selection is not linear or predictable, enhancing the overall security of the steganographic process. The selected pixels are modified at their least significant bits (LSBs) to embed the shuffled bits of the secret message. The LSB method allows for minimal visual distortion while still effectively hiding information.

2.2. Least Significant Bit (LSB) Method

The LSB method involves altering the least significant bit of each pixel's RGB value in an image to embed secret data. Since these bits have the least impact on the overall colour representation, changes are often imperceptible to human eyes. Each initial pixel's RGB values are represented in binary form before insertion. During the insertion process, for each bit of the secret message, a corresponding pixel is selected based on the chaotic sequence. The LSB of that pixel's RGB value is modified to reflect the bit being embedded. For example, consider a pixel with an RGB value before insertion represented as: R=01011111, G=01100111, B=00110100. If we want to embed a bit '1' into the LSB of R, then after insertion: R=01011110 (the last bit changes from '1' to '0' or remains '1' based on whether we're inserting a '0' or '1'). The original RGB values before text insertion and their corresponding values after modification illustrate how only minor changes occur, maintaining visual integrity while successfully embedding hidden information. This indicates that one bit was altered to embed part of the secret message while keeping other bits intact.

In general, a steganography program functions to conceal message data within image data, using a digital image as the medium. A critical aspect of this process is ensuring that the modifications made to the cover image are subtle and imperceptible to the naked eye. This minimizes noticeable differences between the original cover image and the modified stego-image, thereby preserving the confidentiality and integrity of the hidden information. To further secure the embedded message, access to it requires a specific key. Without this key, unauthorized individuals who are unaware of the keyword will be unable to extract the concealed information from the stego-image. Figure 4 shows the main menu design for the system developed in this research.

Vol. 5 No. 3, 2024, pp. 286-298 **Pepadun Journal**

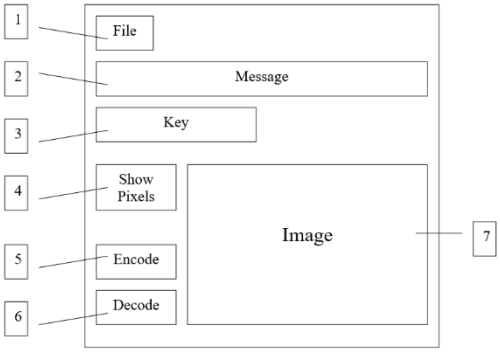

Figure 4. Main menu design.

Information of Figure 4:
1. File menu bar that can be accessed for various functions, there is a load image for entering messages.
2. The message to be inserted into the image.
3. Keys for encryption and decryption of text messages.
4. Shows pixels on the image.
5. Menu for encoding messages.
6. Menu for decoding messages.
7. Place the image where the text message will be inserted.

3. RESULTS AND DISCUSSION

3.1. Message Insertion Algorithm

In the message insertion program using Chaotic Steganography LSB is displayed and explained in detail so that it can be understood clearly. The display will be described in the following steps:

A. Image Input

IF select image = “finished” THEN print the image

B. Input Text Message
IF input message = “finished” THEN print message

C. Insertion process
IF input message and image = “finished” THEN input password
ELSE IF insertion process THEN change image and message into binary = “insert message”
ELSE input message and image = “not finished” THEN cancel the process
FINISH

D. Insertion Output
IF hiding process = “finished” THEN save to storage
FINISH

3.2. Message Retrieval Algorithm

In the message retrieval program using chaotic steganography LSB will be displayed and explained in detail so that users of this program can understand clearly and can use it easily. can be described more clearly in the following sequence of steps:

A. Image Input
IF select image = “finished” THEN print the image

B. Enter Password
IF input password = “done” THEN print password
C. Message retrieval process
IF input image and password = “finished” THEN input password
ELSE IF fetch process THEN change image and message into binary = “display message”
ELSE input password and image = “not finished” THEN cancel the process
FINISH
D. Message Retrieval Output
IF retrieval process = “finished” THEN print message
FINISH

The described algorithm outlines the process of retrieving a hidden message embedded within an image, serving as the container. The program utilizes chaotic least significant bit (LSB) encoding steganography to embed textual data into a digital image. Its design aims to ensure that users can comprehend and utilize the program efficiently. A computer program, by definition, is a series of systematic and logical instructions crafted to achieve specific objectives in accordance with predetermined rules. The implementation of this steganographic application was developed using Microsoft Visual Basic 6.0, comprising a single main form where all functionalities are integrated. The main form provides a detailed explanation of the operational process, enhancing user accessibility and understanding. The program accepts a 24-bit digital image as the container medium, selected via the file menu and open option. Upon selection, the embedding process begins, incorporating the secret message into the container image.

To ensure successful embedding, the size of the container image is carefully chosen to be at least eight times larger than the size of the secret message, ensuring that the text data fits without causing noticeable changes. This approach ensures that the visual difference between the original and modified images remains imperceptible to the naked eye, safeguarding the confidentiality of the embedded text data. Consequently, the program maintains the integrity of the image while securely hiding the secret message, effectively preventing unauthorized individuals from detecting or accessing the hidden information.

3.3. Writing Secret Messages

The container image and secret message in the form of files or text that have been previously selected will be combined in this process. So that later a new file will be created whose physical form is almost the same so as not to raise suspicions of the existence of secret information in the image.

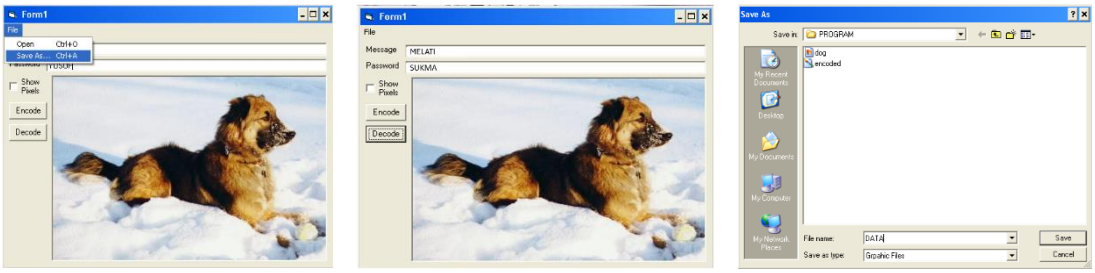

Figure 6. Process of merging text data with images.

The process of embedding secret messages into images, as illustrated in Figure 6, incorporates a security mechanism where the hidden message is protected with a keyword. This enhances the security of the embedded message, ensuring it cannot be accessed without the correct keyword. At this stage, the secret message is written into the container image, and a security key, consisting of at least five characters, is required to safeguard the message. To embed the message into the image, the user must press the encode button, initiating the process. The message is then automatically concealed within the file, ensuring its security by hiding it in

the image. As a result, individuals without the correct keyword will be unable to view the hidden message's content. Only authorized recipients with the correct key can access and understand the embedded message. The output of this process demonstrates the successful integration of the secret message into the image.

Figure 7. Image with inserted text data.

3.4. Secret Message Retrieval

The image that has been embedded with a secret message is selected to be separated again so that the secret message can be read by the authorized person. For this process, a key is needed as a requirement to reopen the hidden secret message.

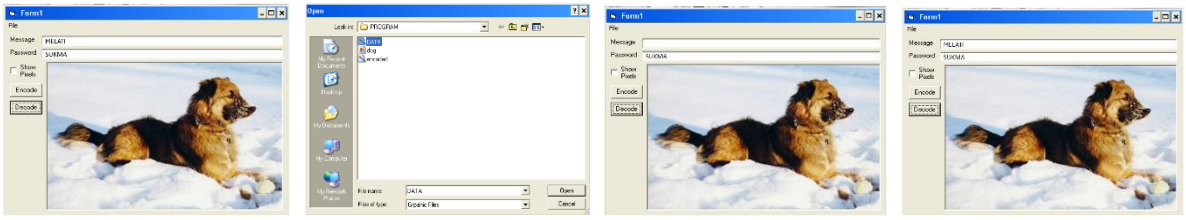

Figure 8. Secret message retrieval process.

The message insertion and retrieval algorithms using Chaotic Least Significant Bit (LSB) encoding steganography are designed to securely embed and extract secret messages within images. The chaotic system introduces unpredictability in selecting which pixels to modify for embedding the secret message. This randomness enhances security by making it difficult for unauthorized users to detect where data has been inserted. A chaotic map generates a sequence of numbers that determine pixel locations in a non-linear manner. This sequence helps in shuffling bits of the secret message before embedding them into selected pixels.

The least significant bit of each pixel's RGB value in the cover image is altered to embed bits from the secret message (see Section 2.2). This method ensures that changes are minimal and visually imperceptible, maintaining the integrity of the original image while securely hiding information. A password mechanism ensures that only authorized users can access or retrieve hidden messages, adding an additional layer of security. The chaotic nature of pixel selection makes it difficult for attackers to predict where data might be hidden, enhancing overall security against detection.

The described algorithms leverage chaotic systems combined with LSB techniques to create a robust steganographic method for securely embedding and retrieving messages within images. By ensuring randomness through chaos and maintaining visual fidelity through LSB modifications, this approach effectively protects sensitive information from unauthorized access while remaining user-friendly for legitimate users.

4. CONCLUSION

Based on the implemented system, several key conclusions can be drawn regarding the functionality and performance of the encoding and decoding processes. After the encoding process is completed, the size of the image increases compared to its original size before the secret message is embedded. This increase is due to the additional data introduced during the embedding process. Notably, after the decoding process is performed, the image size does not revert to its original state prior to encoding. Instead, the image retains the modified size resulting from the encoding stage. This consistency in size across the encoding and decoding processes highlights the effective use of the Least Significant Bit (LSB) algorithm, which is proven to reliably embed hidden messages into images while maintaining the integrity of the embedded data. Furthermore, the system ensures that the secret messages remain unchanged during both insertion and extraction processes. This guarantees that the extracted messages are identical to the original messages, without any alterations or data loss, thereby validating the robustness and accuracy of the implemented LSB-based steganographic method.

5. SUGGESTION

To improve the functionality and robustness of the steganography software described, several recommendations are proposed. First, enhancing data hiding efficiency by exploring advanced digital image processing techniques can significantly reduce the number of images required to embed larger data volumes. This optimization ensures practical usability and resource conservation. Second, integrating watermarking algorithms alongside steganography during the data embedding process can yield more robust outcomes. Such a combination provides resilience against various image manipulations, including alterations in brightness, blurring, resizing, or other potential attacks, thereby enhancing the security and reliability of hidden messages. These improvements would not only bolster the software's performance but also extend its applicability in real-world scenarios.

REFERENCES

[1] A. AbdelRaouf, "A new data hiding approach for image steganography based on visual color sensitivity," *Multimedia Tools and Applications*, pp. 1–25, 2021.

[2] N. Hopper, L. V. Ahn, & J. Langford, "Provably Secure Steganography," *IEEE Transactions on Computers*, vol. 58, pp. 662–676, 2002.

[3] S. Sharda & S. Budhiraja, "Image Steganography: A Review," *International Journal for Research in Applied Science and Engineering Technology (IJETAE)*, *3*(1), 707-710, 2023.

[4] A. Anees, A. Siddiqui, J. Ahmed, & I. Hussain, "A technique for digital steganography using chaotic maps," *Nonlinear Dynamics*, vol. 75, pp. 807–816, 2014.

[5] G. Gambhir & J. Mandal, "Multicore implementation and performance analysis of a chaos-based LSB steganography technique," *Microsystem Technologies*, vol. 27, pp. 4015–4025, 2020.

[6] X. Chai, H. Wu, Z. Gan, Y. Zhang, Y. Chen, & K. Nixon, "An efficient visually meaningful image compression and encryption scheme based on compressive sensing and dynamic LSB embedding," *Optics and Lasers in Engineering*, vol. 124, p. 105837, 2020.

[7] A. Mu'azu & K. Kabir, "Hybrid of Least Significant Bits and Most Significant Bits for Improving Security and Quality of Digital Image Steganography," *WSEAS Transactions on Computers*, 22, pp. 253-262, 2023.

[8] H. Ogras, "An Efficient Steganography Technique for Images using Chaotic Bitstream," *International Journal of Computer Network and Information Security*, 15 (2), 21, 2019.

[9] S. Rahman, J. Uddin, H. Khan, H. Hussain, A. Khan, & M. Zakarya, "A Novel Steganography Technique for Digital Images Using the Least Significant Bit Substitution Method," *IEEE Access*, vol. 10, pp. 124053–124075, 2022.

[10] A. Cheddad, J. Condell, K. Curran, & P. Kevitt, "Digital image steganography: Survey and analysis of current methods," *Signal Process*, vol. 90, pp. 727–752, 2010.

[11] H. Tayyeh & A. Al-Jumaili, "A combination of least significant bit and deflate compression for image steganography," *International Journal of Electrical and Computer Engineering (IJECE)*, 2022.

[12] Kavia & K. Anushudha, "Image Security using Steganography and Cryptography," *International Journal for Research in Applied Science and Engineering Technology*, Vol. 9, VII, pp. 373-378, 2021.

[13] M. Valandar, P. Ayubi, and M. Barani, "A new transform domain steganography based on modified logistic chaotic map for color images," *J. Inf. Secur. Appl.*, vol. 34, pp. 142–151, 2017.

[14] M. Kahn, Steganography Mailing List. [Online] Available: https://www.jjtc.com/Steganography/steglist.htm, 1995.

[15] D. Ariyus, *Multimedia Security*, Yogyakarta: Andi, 2009.