Hiding Data in a Colored Image using the Similarity and Knapsack Algorithm

Sana Ahmed Kadhom
Al Maamon College University, Computer Science Department

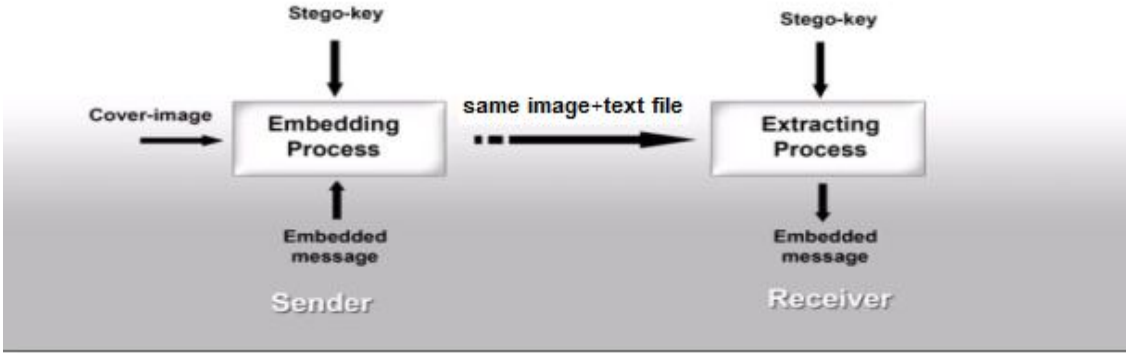
Abstract
In this paper, a new algorithm for hiding data in a binary image is introduced. This technique allows embedding data in a binary image without making any changes in the cover image; thus getting 100% similarity this technique does not need any key for embedding and extracting data. Also, it allows hiding many bits by matching values. The proposed algorithm ensures security and the hiding effect is quite invisible.

Keywords: Data Hiding, Steganography, Binary Images, Security, Similarity.

1- INTRODUCTION

The staggering growth in communication technology and usage of public domain channels (i.e. Internet) has greatly facilitated transfer of data. However, such open communication channels have greater vulnerability to security threats causing unauthorized information access. This might lead to disclosure, modification or even deletion of classified/unclassified data files or copyright violation [1]. Therefore preventing unauthorized information access has been a prime consideration for growing use of steganography techniques [2] for applications like copyright protection, feature tagging and secret communication [3]. As a result, steganography has become an interesting and challenging field of research striving to achieve greater immunity of hidden data against signal processing operations on the host cover media; e.g. a good steganography technique should offer immunity of hidden data against lossy compression, scaling, interception, modification, or removal etc. and ensure that embedded data remains inviolate and recoverable [4]. However, a trade-off between the quantity of hidden data and its degree of immunity to host signal modification is needed in most cases [5]. Data hiding requires embedding data into digital media like image, audio, or text, however, due to providing high embedding efficiency the still images are preferred as hosts as compared to others.

The general model of hiding data in other data can be described as follows: The embedded data is the message that one wishes to send secretly. It is usually hidden in an innocuous message referred to as a cover-text, cover image or cover-audio as appropriate, producing the stego-text or other stego object. A stego-key is used to control the hiding process so as to restrict detection and/or recovery of the embedded data to parties who know it (or who know some derived key value)[1]. Figure(1) shows the core part Information Hiding.



2- KNAPSACK ALGORITHM

The encryption and decryption algorithms are based on solving a knapsack problem. The knapsack problem consists of finding a way to select some of the items to be packed such that their sum (the amount of space they take up) exactly equals the knapsack capacity (the target). Formally the knapsack problems is as follows: given a set S = {a1,a2,...an} and a target sum T, where each $a_i \geq 0$, is there a selection vector V = {v_1, v_2,...v_n} each of whose elements is 0 or 1, such that $\sum_i (a_i * v_i) = T$

The idea of scheme is to encode a binary message as a solution to a knapsack problem reducing the cipher text to the target sum obtained by adding terms corresponding to 1s in the plaintext. That is, blocks of plain text are converted to knapsack sums by adding into the sum the terms that match with 1bits in the plaintext.

The receiver will have to use the cipher text (the target sum) to recover the plaintext. A knapsack problem is said to be ``easy'' if $\forall k\ a_k > \sum_j^{k-1} aj$. The solution for the easy knapsack problem is straightforward, and can be achieved in O(n). Furthermore, there exists a way for transforming an easy knapsack set S into a non-easy knapsack H simply by multiplying the elements of S by w mod n where w and n are well chosen integers. Formally the terms of H are: [6]

$$h_i = (w * s_i) \bmod n$$

2-1 Encryption algorithm (executed by the sender)

H is called the public key and is made public by the receiver

1. Choose w and n such that n > max(S) and n is prime, and w < n. Construct H from S, w, and n
2. Sender uses H to encipher blocks of m bits P_0 = [p_1, p_2...pm], P_1= [p_{m+1}, p_{m+2}...p_{2m}] and so forth (m is the number of terms in H), as follows: $T_i = P_i * H = \sum_j (p_j * h_j)$. Thus $T_0 = P_0 * H$, $T_1 = P_1 * H$ and so on. T_i are then transmitted to the receiver via a reliable channel.

2-2 Decryption algorithm (executed by the receiver)

The tuple (S, n, w) is called the private key and is kept secret by the receiver

1- Use w and n to compute w^{-1} where $w^{-1} * w = 1 \bmod n$. if n is prime then $w^{-1} = w^{(n-2)} \bmod n$.
2- Compute $A_i = w^{-1} * T_i \bmod n$.
3- Find P_i by solving $A_i = P_i * S$.

3- THE SUGGESTED METHOD

The embedding algorithm has two parts, one with the plaintext to be hidden and the second with the image (covered image).

3-1 First part (plain text):

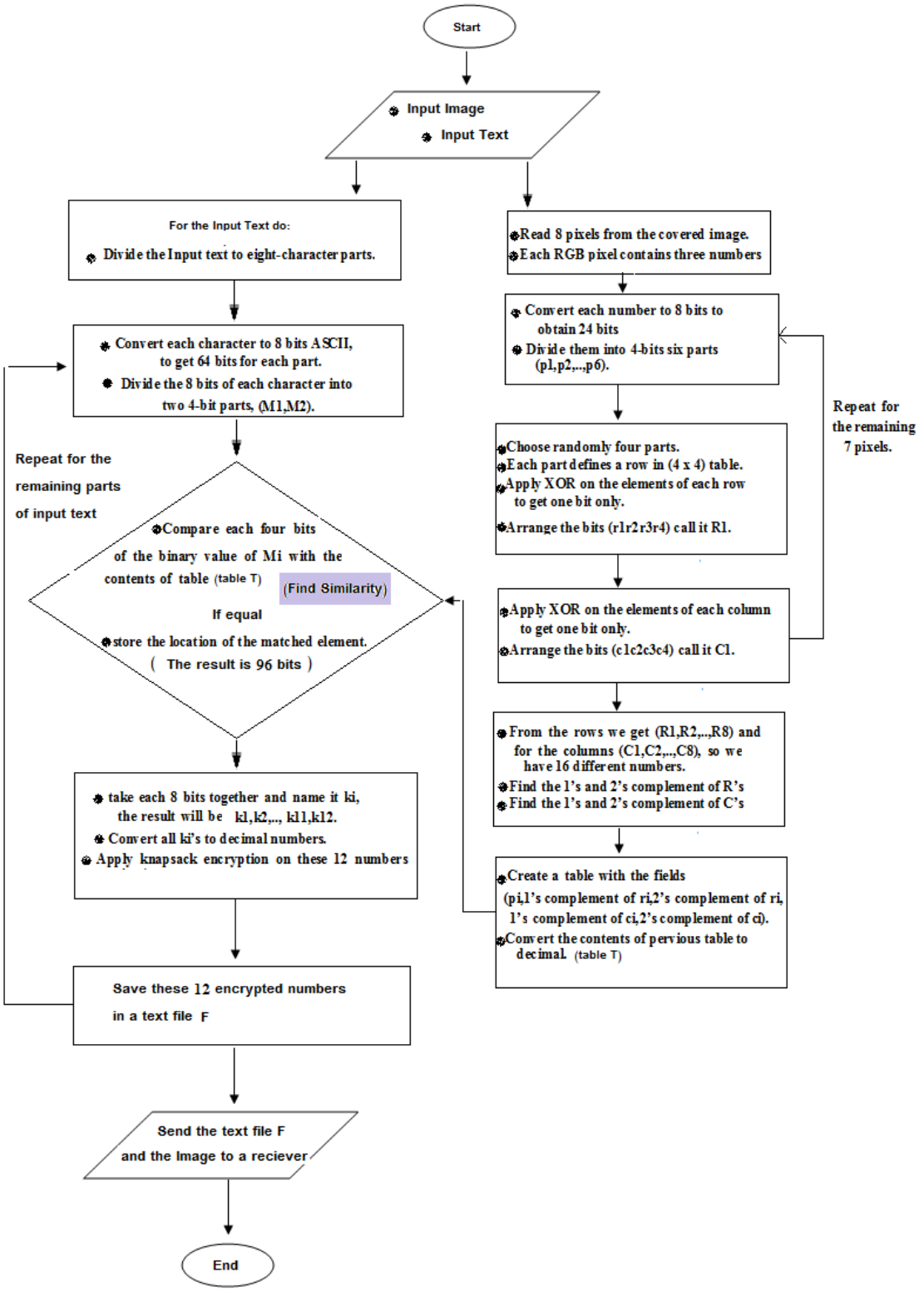
1- Divide the plain text to eight-character parts.
2- Convert each character to 8 bits ASCII, to get 64 bits for each part.
3- Divide the 8 bits of each character into two 4-bit parts, (M1,M2).

3-2 Second part (covered Image):

To hide For each part of the plain text, the following process will be done:

1- Read 8 pixels from the covered image.
2- Each RGB pixel contains three numbers (Red, Green, Blue).
3- Convert each number to 8 bits to obtain 24 bits.
4- Divide them into 4-bits six parts (p1,p2,..,p6).
5- Choose randomly four parts. Each part defines a row in (4 x 4) table.
6- Apply XOR on the elements of each row to get one bit only.
7- Arrange the bits (r1r2r3r4) call it R1.
8- Apply XOR on the elements of each column to get one bit only.
9- Arrange the bits (c1c2c3c4) call it C1.
10- Repeat the steps from 1 to 9 for the remaining 7 pixels.
11- From the rows we get (R1,R2,..,R8) and for the columns (C1,C2,..,C8), so we have 16 different numbers.
12- Find the 1's and 2's complement of R's (without carry).
13- Find the 1's and 2's complement of C's (without carry).
14- Create a table with the fields (pi,1's complement of ri,2's complement of ri, 1's complement of ci,2's complement of ci).
15- Convert the contents of pervious table to decimal.
16- Compare each four bits of the binary value of Mi from plaintext with the contents of pervious table and store the location(row number, column number) of the matched element. So, each 4 bits of Mi will give 6 bits and for a part of 8 characters the number of bits will be 96.
17- From these 96 bits, take each 8 bits together and name it ki, the result will be k1,k2,.., k11,k12.
18- Convert all ki's to decimal numbers.
19- Apply knapsack encryption on these 12 numbers (use the public key of the receiver).
20- The output will be send as a text file with the untouched covered image.

4- THE SUGGESTED METHOD FLOW CHART:



5- DECRYPTION PROCESS

1- The receiver will apply the same steps of part two (from step 1..15) on the image.
2- Decrypt the numbers in the cipher text file (in the text file received with the image) using knapsack decryption (use the privet key of the receiver).
3- Convert the resulting numbers to binary (12 number * 8 bits for each number = 96)
4- Divide these bits into 16 parts, each part of 6 bits.
5- Divide each 6 bits into two parts each of 3 bits.
6- The first three bits defines the row number and the second three bits defines the column number.
7- Take the element value of intersection (row,column). Convert to 4 bits binary.
8- These are the value of the first 4 bits of the first character from the sent secret message.
9- Repeat the steps 5..8 for the next 6 bits of the 96 bits to get the second part of the first character.
10- Each 12 bits will give one character and at the end of the process you will get the whole 8 characters of the sent message.

6- IMPLEMENTATION OF THE SUGGESTED ALGORITHM

Let the covered image be image (1) below

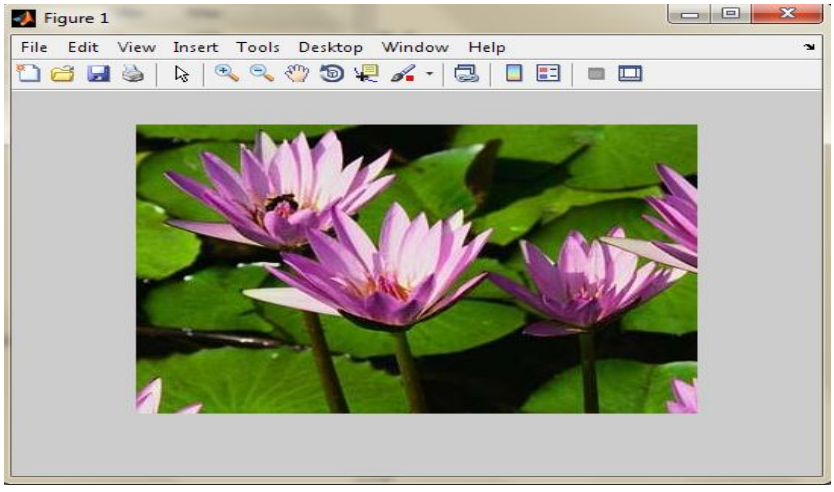


Image (1)

And let the message be the word "ENCRYPTION"

- Take the first 8 characters.
- Convert the ASCII code of each character to binary
- Divide it into two parts each of 4 bits, as shown in table (1) below

	Char	ASCII	Binary	First 4 bits	Second 4 bits
1	E	69	01000101	0100	0101
2	N	78	01001110	0100	1101
3	C	67	01000011	0100	0011
4	R	82	01010010	0101	0001
5	Y	89	01011001	0101	1001
6	P	80	01010000	0100	1111
7	T	84	01010100	0101	0100
8	I	73	01001010	0100	1010

Table (1)

Take 8 pixels from the covered image:
(P1= RGB(80,128,16) , P2 = RGB(81,130,15) , P3 = RGB(81,131,16) , P4 = RGB(80,134,14)
P5 = RGB(75,121,13) , P6 = RGB(76,124,14) , P7 = RGB(78,126,14) , P8 = RGB(79,130,12))
For each pixel apply the following:
Pixel 1 (P1) : RGB(80,128,16)

80 = 01010000 p1= 0101 p2 = 0000
128 = 10000000 p3= 1000 p4 = 0000
16 = 00010000 p5= 0001 p6 = 0000

Choose p1,p4,p5,p3 (using random function)

p1	0	1	0	1
p4	0	0	0	0
p5	0	0	0	1
p3	1	0	0	0

0⊕1⊕0⊕1=0
0⊕0⊕0⊕0=0
0⊕0⊕0⊕1=1
1⊕0⊕0⊕0=1

XOR of rows

0⊕0⊕0⊕1=1
1⊕0⊕0⊕0=1
0⊕0⊕0⊕0=0
1⊕0⊕1⊕0=0

XOR of columns

r1 = 0011 , c1 = 1100

Pixel 2 (P2) = RGB(81,130,15)

81 = 01010001 p1=0101 p2=0001
130 = 10000010 p3=1000 p4=0010
15 = 00001111 p5=0000 p6=1111

p1	0	1	0	1
p4	0	0	1	0
p5	0	0	0	0
p3	1	0	0	0

0⊕1⊕0⊕1=0
0⊕0⊕1⊕0=1
0⊕0⊕0⊕0=0
1⊕0⊕0⊕0=1

0⊕0⊕0⊕1=1
1⊕0⊕0⊕0=1
0⊕1⊕0⊕0=1
1⊕0⊕0⊕0=1

r2 = 0101 , c2 = 1111

Then continuo the same process with other pixels to get the following results:

Pixel 3 (P3) = RGB(81,131,16) , r3 = 1101 , c3 = 1110
Pixel 4 (P4) = RGB(80,134,14) , r4= 0001 , c4 = 1011
Pixel 5 (P5) = RGB(75,121,13) , r5= 1000 , c5 = 1010
Pixel 6 (P6) = RGB(76,124,14) , r6 = 1001 , c6 = 1111
Pixel 7 (P7) = RGB(78,126,14) , r7 = 1101 , c7 = 1101
Pixel 8 (P8) = RGB(79,130,12) , r8 = 1101 , c8 = 1111

Calculate for each value of ri and ci, the 1's complement $(ri)^{1c}$ and the 2's complement $(ri)^{2c}$, put the results in a table. Give a sequence for each row and column in the table as follows:

	Column number	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)
Row number		Ri	$(ri)^c$	$(ri)^{2c}$	ci	$(ci)^c$	$(ci)^{2c}$
0 (000)	P1	0011	1100	1101	1100	0011	0100
1(001)	P2	0101	1010	1011	1111	0000	0001
2 (010)	P3	0011	1100	1101	1110	0001	0010
3 (011)	P4	0001	1110	1111	1011	0100	0101
4 (100)	P5	1000	0111	1000	1010	0101	0110
5 (101)	P6	1001	0110	0111	1111	0000	0001
6 (110)	P7	1101	0010	0011	1101	0010	0011
7 (111)	P8	1101	0010	0011	1110	0001	0010

Table (2)

Convert the contents of the previous table to decimal:

	Column number	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)
Row number		Ri	$(ri)^c$	$(ri)^{2c}$	ci	$(ci)^c$	$(ci)^{2c}$
0 (000)	P1	3	12	13	12	13	4
1(001)	P2	5	10	11	15	0	1
2 (010)	P3	3	12	13	14	1	3
3 (011)	P4	1	14	15	11	4	5
4 (100)	P5	8	7	8	10	5	6
5 (101)	P6	9	6	7	15	0	1
6 (110)	P7	13	0	3	13	2	3
7 (111)	P8	13	2	3	14	1	2

Table (3)

For the characters in table (1) :

- Convert the first 4 bits and the second 4 bits to decimal.
- Match the numbers with the values of table (2). (i.e find the similarity between the values of the data to be hidden and the values of the image pixels).
- Take the position of each match (row number, column number). If more than one match is found then any of these matches could be used.
- For each match 6 bits will be gotten.

CHAR	ASCII	FIRST 4 bits	LOCATION	SECOND 4 bits	LOCATION
E	69	0100 = 4	$P1C^{2C}$=000101	0101=5	P2R=001000
N	78	0100 = 4	$P1C^{2C}$=000101	1110=13	$P3R^{2C}$=010010
C	67	0100 = 4	$P4C^{C}$=011100	0011=3	$P3C^{2C}$=010101
R	82	0101 = 5	$P4C^{2C}$=011101	0010=2	$P7C^{C}$=110100
Y	89	0101 = 5	$P5C^{C}$=100100	1001=9	P6R=101000
P	80	0101 = 5	P2R =001000	0000=0	$P2C^{C}$=001100
T	84	0101 = 5	P2R =001000	0100=4	$P4C^{C}$=011100
I	73	0100 = 4	$P1C^{2C}$=000101	1001=9	P6R=101000

As a final result for the 8 characters, a sequence of 96 bits will be gotten.

OUTPUT =000101 001000 000101 010010 011100 010101 011101 110100
100100 101000 001000 001100 001100 001000 000101 101000

Divide that sequence of bits into 8-bits parts:

k1= 00010100 = 20 k2 = 10000001 =129 k3 =01010010 = 82

k4= 01110001 = 113 k5 = 01010111= 87 k6=01110100 = 116

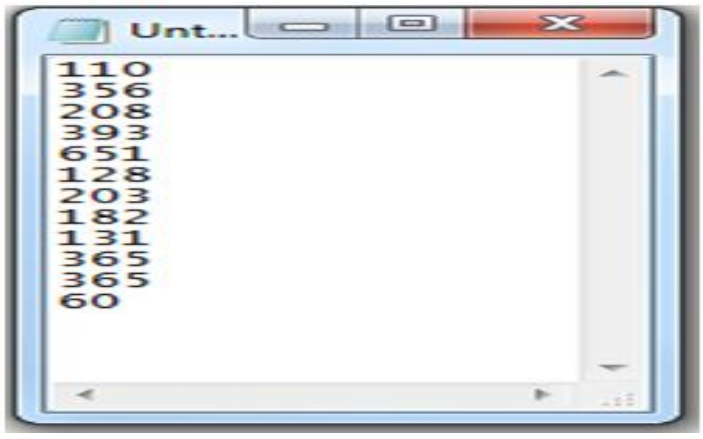
k7=10010010 = 146 k8 = 10000010 =130 k9=00001100 =12

k10=00100001 = 33 k11=10000001 = 129 k12=01101000 =104

To get more secure data transfer, Encrypt the numbers using knapsack algorithm.

	Plain text	Cipher text
k1	20	110
k2	129	356
k3	82	208
k4	113	393
k5	87	651
k6	116	128
k7	146	203
k8	130	182
k9	12	131
k10	33	365
k11	129	365
k12	104	60

Send the encrypted numbers as a text file to the receiver with the covered image



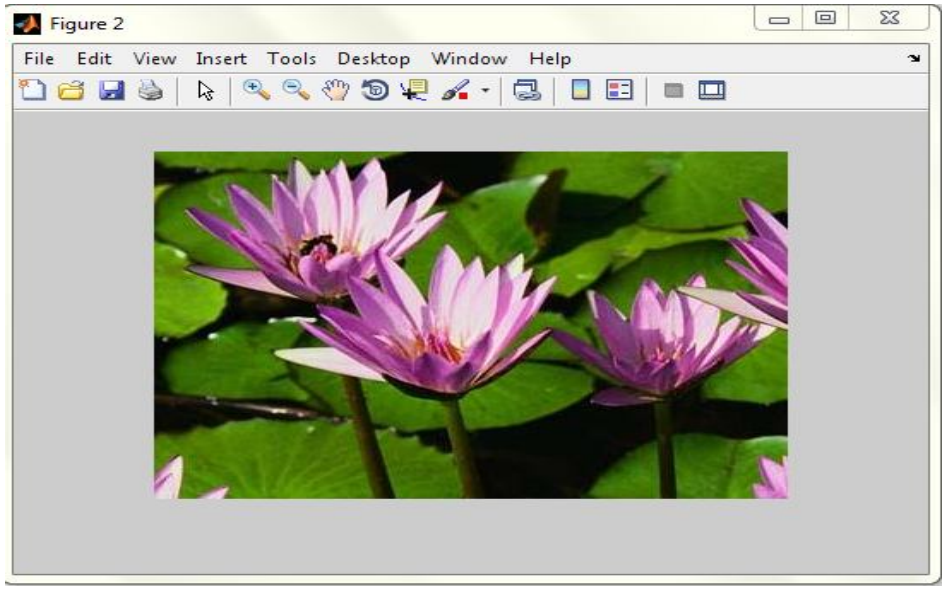


Image (2)

CONCLUSIONS

The suggested method in this paper will hide the data in the colored RGB image without any change in the values of the image pixels, because the method will use the likeness between the value of pixel in image and the data we want to hide.

The output is a group of decimal encrypted numbers which is send as a text file with the image, so even if this file is opened, no data and no information about the data will be discovered. This gives the power and difficulty for opponent to discover the data in the binary image.

REFERENCES

[1]- W. Bender, D. Gruhl, N. Morimoto and A. Lu "Techniques for Data Hiding", IBM Systems Journals, Vol.35, NOS 3-4, pp. 313-336, 1996.

[2]- M.M Amin, M. Salleh, S.Ibrahim, M. R. Katmin, and M.Z.I .Shamsuddin "Information hiding using Steganography", Proc. 4th IEEE Conference on Telecommunication Technology, Shah Alam, Malaysia, pp. 21-25, 2003.

[3]- Cox et al. "A Secure, Robust Watermark for Multimedia", Lecture Notes in Computer Science Vol. 1, 174, Springer- Verlag, Berlin, pp. 185-206, 1996.

[4]- F.A.P. Peticolas, R.J.Anderson and M.G.Kuhn, "Inforamition Hiding- A survey", Proc. of IEEE, Vol. 87, Issue: 7, pp. 1062-1078, 1999.

[5]- Harsh Vikram Singh, A. K. Singh, S.K. Balasubramanian and Anand Mohan, "Minimizing Security Threats in Multimedia Systems", Proc. 2nd IEEE International Conf. on Distributed Framework for Multimedia Applications Penang, Malaysia, pp. 102-107, 2006.

[6]- 2172-Merkle-Hellman knapsack public key cryptosystem, Africa and Middle East – Africa and Arab – 2000/2001.