Advanced Steganography Algorithm using Encrypted secret message

Joyshree Nath
A.K.Chaudhuri School of IT
University of Calcutta
Kolkata, India
joyshreenath@gmail.com

Asoke Nath
Department of Computer Science
St. Xavier's College(Autonomous)
Kolkata, India
asokejoy@gmail.com

***Abstract*—In the present work the authors have introduced a new method for hiding any encrypted secret message inside a cover file. For encrypting secret message the authors have used new algorithm proposed by Nath et al(1). For hiding secret message we have used a method proposed by Nath et al (2). In MSA (1) method we have modified the idea of Play fair method into a new platform where we can encrypt or decrypt any file. We have introduced a new randomization method for generating the randomized key matrix to encrypt plain text file and to decrypt cipher text file. We have also introduced a new algorithm for encrypting the plain text multiple times. Our method is totally dependent on the random text_key which is to be supplied by the user. The maximum length of the text_key can be of 16 characters long and it may contain any character (ASCII code 0 to 255). We have developed an algorithm to calculate the randomization number and the encryption number from the given text_key. The size of the encryption key matrix is 16x16 and the total number of matrices can be formed from 16 x 16 is 256! which is quite large and hence if someone applies the brute force method then he/she has to give trail for 256! times which is quite absurd. Moreover the multiple encryption method makes the system further secured. For hiding secret message in the cover file we have inserted the 8 bits of each character of encrypted message file in 8 consecutive bytes of the cover file. We have introduced password for hiding data in the cover file. We propose that our new method could be most appropriate for hiding any file in any standard cover file such as image, audio, video files. Because the hidden message is encrypted hence it will be almost impossible for the intruder to unhide the actual secret message from the embedded cover file. This method may be the most secured method in digital water marking.**

Keywords- Steganography; MSA algorithm; Encryption; Decryption;

I. INTRODUCTION

In the present work we have used two (2) methods: (i) We encrypt the secret message(SM) using a method MSA(Meheboob,Saima and Asoke) proposed by Nath et al.(1). (ii) We insert the encrypted secret message inside the standard cover file(CF) by changing the least significant bit(LSB). Nath et al(2) already proposed different methods for embedding SM into CF but there the SF was inserted as it is in the CF and hence the security of steganography was not very high. In the present work we have basically tried to make the steganography method more secured. It means even if someone can extract SM from CF but he cannot be able to decrypt the message as he has to know the exact decryption method. In our present work we try to embed almost any type of file inside some standard cover file (CF) such as image file(.JPEG or .BMP) or any image file inside another image file. Here first we will describe our steganography method for embedding any type of file inside any type of file and then we will describe the encryption method which we have used to encrypt the secret message and to decrypt the extracted data from the embedded cover file.

(i) Least Significant Bit Insertion method: Least significant bit (LSB) insertion is a common, simple approach for embedding information in a cover image. The LSB or in other words 8-th bit of some or all the bytes inside an image is changed to a bit of the secret message. Let us consider a cover image contains the following bit patterns:

Byte-1	Byte-2	Byte-3	Byte-4
00101101	00011100	11011100	10100110

Byte-5	Byte-6	Byte-7	Byte-8
11000100	00001100	11010010	10101101

Suppose we want to embed a number 200 in the above bit pattern. Now the binary representation of 200 is 11001000. To embed this information we need at least 8 bytes in cover file. We have taken 8 bytes in the cover file. Now we modify the LSB of each byte of the cover file by each of the bit of embed text 11001000.

Now we want to show what happens to cover file text after we embed 11001000 in the LSB of all 8 bytes:

TABLE 1 CHANGING LSB

Before Replacement	After Replacement	Bit inserted	Remarks
00101101	0010110**1**	1	No change in bit pattern
00011100	0001110**1**	1	Change in bit pattern(i)
11011100	1101110**0**	0	No change in bit pattern
10100110	1010011**0**	0	No change in bit pattern
11000100	1100010**1**	1	Change in bit pattern(ii)
00001100	0000110**0**	0	No change in bit pattern
11010010	1101001**0**	0	No change in bit pattern

10101101	10101100	0	Change in bit pattern(iii)

Here we can see that out of 8 bytes only 3 bytes get changed only at the LSB position. Since we are changing the LSB hence we are either changing the corresponding character in forward direction or in backward direction by only one unit and depending on the situation there may not be any change also as we have seen in the above example. As our eye is not very sensitive so therefore after embedding a secret message in a cover file our eye may not be able to find the difference between the original message and the message after inserting some secret text or message on to it. To embed secret message we have to first skip 600 bytes from the last byte of the cover file. After that according to size of the secret message (say n bytes) we skip 8*n bytes. After that we start to insert the bits of the secret file into the cover file. Under no circumstances the size of the cover should not be less the 10*sizeof(secret message) then our method will fail. For extracting embedded file from the cover file we have to perform the following:

One has to enter the password while embedding a secret message ile. If password is correct then the program will read the file size from the cover file. Once we get the file size we follow simply the reverse process of embedding a file in the cover file. We read LSB of each byte and accumulate 8 bits to form a character and we immediately write that character on to a file.

We made a long experiment on different types of host files and also the secret messages and found the following combinations are most successful:

Table-2 COVER FILE TYPE AND SECRET MESSAGE FILE TYPE

Sl.No.	Cover file type	Secret file type used
1	.BMP	.BMP,.DOC,.TXT,.WAV,.MP3,.XLS,.PPT, .AVI,.JPG,.EXE..COM
2.	.JPG	.JPG,.BMP,.TXT,.WAV,.MP3,.XLS,,.PPT, .EXE,.COM
3.	.DOC	.TXT
4.	.WAV	.BMP,.JPG,.TXT,.DOC
5.	.AVI	.TXT,.WAV,.JPEG
6.	.PDF	.TXT

Only in case of .PDF and .JPEG file to insert secret message is a bit difficult job as those files are either compressed or encrypted. Even then we got success for inserting a small text file in .PDF file. So we can conclude that .PDF file is not a good cover file. On the other hand .BMP file is the most appropriate file which can be used for embedding any type of file without facing any problem.

(ii) Meheboob, Saima and Asoke(MSA) Symmetric key Cryptographic method:

Symmetric key cryptography is well known concept in modern cryptography. The plus point of symmetric key cryptography is that we need one key to encrypt a plain text and the same key can be used to decrypt the cipher text and the main problem is that the same key is used for encryption as well as decryption process. Hence the key must be secured. Because of this problem we have introduced public key cryptography such as RSA public key method. RSA method has got both merits as well as demerits. The problem of Public key cryptosystem is that we have to do massive computation for encrypting any plain text. Sometimes these methods may not be also suitable such as in sensor networks. However, there are quite a number of encryption methods have come up in the recent past appropriate for the sensor nodes. Nath et al.(1) proposed a symmetric key method where they have used a random key generator for generating the initial key and that key is used for encrypting the given source file. MSA method is basically a substitution method where we take 2 characters from any input file and then search the corresponding characters from the random key matrix and store the encrypted data in another file. In our work we have the provision for encrypting message multiple times. The key matrix contains all possible characters (ASCII code 0 to 255) in a random order. The pattern of the key matrix will depend on text_key entered by the user. Nath et al. proposed algorithm to obtain randomization number, encryption number and the shift parameter from the initial text_key. We have given a long trial run on text_key and we found that it is very difficult to match the three above parameters for 2 different Text_key which means if someone wants to break our encryption method then he/she has to know the exact pattern of the text_key otherwise it will not be possible to obtain two sets of identical parameters from two different text_key. We have given several trial runs to break our encryption method but we found it is almost unbreakable. For pure text file we can apply brute force method to decrypt small text but for any other file such as binary file we cannot apply any brute force method.

II. RANDOM KEY GENERATION AND MSA ENCRYPTION ALGORITHM

Before we embed the secret message in a cover file we first encrypt the secret message using MSA method. The detail method is given in our previous publication (1). Here we will describe the MSA algorithm in brief:

To create Random key Matrix of size(16x16) we have to take any key. The size of key must be less than or equal to 16 characters long. These 16 characters can be any of the 256 characters(ASCII code 0 to 255). The relative position and the character itself is very important in our method to calculate the randomization number , the encryption number and the relative shift of characters in the starting key matrix. We take an example how to calculate randomization number, the encryption number and relative shift from a given key. Here we are demonstrating our method:

Suppose key=AB

Choose the following table for calculating the place value and the power of characters of the incoming key:

TABLE-3

Length of key(n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Base value(b)	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2

$$\text{Step-1: Sum}=\sum_{m=1}^{n} \text{ASCII Code} * b^{m} \quad \text{----(1)}$$

Example-1:

Now we calculate the sum for key="**AB**" using equation(1)

$\text{Sum}=65*16^1 + 66 * 16^2 = 17936$

Now we have to calculate 3 parameters from this sum (i) Randomization number(n1), (ii) Encryption number(n2) and (iii)Relative shift(n3) using the following method:

a) Randomization number(n1):

num1=1*1+7*2+9*3+3*4+6*5=84

n1=sum mod num1=17936 mod 84=**44**

Note: if n1=0 then n1=num1 and n1<=128

b) Encryption number(n2):

num2=6*1+3*2+9*3+7*4+1*5=72

n2=sum mod num2 =17936 mod 72 =**8**

Note: if n2=0 then n2=num2 and n2<=64

c) Relative shift(n3):

n3= Σall digits in sum=1+7+9+3+6=**26**

Now we show the original key matrix(16 x 16) which contains all characters(ASCII code 0-255):

TABLE –4 : THE ORIGINAL MATRIX:

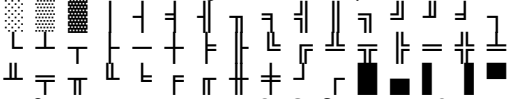
☺ ☻ ♥ ♦ ♣ ♠
♫ ☼
► ◄ ↕ ‼ ¶ § ▬ ↨ ↑ ↓ ← ∟ ↔ ▲ ▼
! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~ ⌂
Ç ü é â ä à å ç ê ë è ï î ì Ä Å
É æ Æ ô ö ò û ù ÿ Ö Ü ¢ £ ¥ Pts *f*
á í ó ú ñ Ñ ª º ¿ ⌐ ¬ ½ ¼ ¡ « »

α ß Γ π Σ σ µ τ Φ Θ Ω δ ∞ φ ε ∩
≡ ± ≥ ≤ ⌠ ⌡ ÷ ≈ ° · · √ ⁿ ²

TABLE-5 : THE MATRIX AFTER RELATIVE SHIFT(N3=26) IS:

4 N h é £ ╣ ╩ Ω ☺ ← 5 O i â
¥ ╖ ╤ δ ☻ ∟ 6 P j ä Pts ╕ ╥ ∞ ♥ ↔
7 Q k à *f* ╣ ╚ φ ♦ ▲ 8 R l å á ╠
└ ε ♣ ▼ 9 S m ç í ╖ ╒ ∩ ♠ : T
n ê ó ╝ ╓ ≡ ! ; U o ë ú ╩ ╬ ±
" < V p è ñ ╛ ╪ ≥ # = W q ï
Ñ ╖ ┘ ≤
$ > X r î ª └ ┌ ⌠ ♂ %
? Y s ì º ┴ ▉ ⌡ ♀ & @ Z t Ä ¿ ┬
' A [u Å ⌐ ├ ▌ ≈ ♫ (B \
v É ¬ ─ ▐ ° ☼) C] w æ ½ ┼ ■ ·
► * D ^ x Æ ¼ ╞ α · ◄ + E _ y ô
¡ ╠ ß √ ↕ , F ` z ö « ╚ Γ ⁿ ‼ -
G a { ò » ╓ π ² ¶ . H b | û ░ ╩
Σ ■ § / I c } ù ▒ ╦ σ ═ 0 J d
~ ÿ ▓ ╠ µ ↨ 1 K e ⌂ Ö ▏ ═ τ ↑ 2
L f Ç Ü ┤ ╬ Φ ↓ 3 M g ü ¢ ╡ ╧ Θ

Now we apply the following randomization methods one after another in a serial manner:

Step-1: Function cycling()

Step-2: Function upshift()

Step-3: Function downshift()

Step-4:Function leftshift()

Step-5:Function rightshift()

Step-6:Function random()

Step-7:Function random_diagonal_right()

Step-8:Function random_diagonal_left()

For detail randomization methods we refer to our previous work(1).

After finishing above shifting process we perform

a) column randomization
b) row randomization and
c) diagonal rotation and
d) reverse diagonal rotation

Each operation will continue for n3 number of times.

Now we apply encryption process on any text file. Our encryption process is as follows:

We choose a 4X4 simple key matrix:

TABLE-6

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Case-I: Suppose we want to encrypt **FF** then it will take as **GG** which is just one character after F in the same row.

Case –II: Suppose we want to encrypt **FK** where **F** and **K** appears in two different rows and two different columns. **FK** will be encrypted to **KH (FK→GJ→HK→KH).**

Case-III: Suppose we want to encrypt **EF** where **EF** occurs in the same row. Here **EF** will be converted to **HG**

III. CHANGING LSB BITS OF COVER FILE USING ENCRYPTED SECRET MESSAGE FILE

In the present work we have made an exhaustive study on embedding (i) text, (ii)sound, (iii)image in different cover files such as image file, sound file, word document file, .PDF file. The size of the cover file must be at least 10-times more than secret message file which is to be embedded within the cover file. The last 500 bytes of the cover file we reserved for storing the password and the size of the secret message file. After that we subtract n*(size of the secret message file) from the size of the cover file. Here n=8 depending on how many bytes we have used to embed one byte of the secret message file in the cover file. For strong password we have used a simple algorithm as follows: We take XOR operation with each byte of the password with 255 and insert it into the cover file. To retrieve the password we read the byte from the cover file and apply XOR operation with 255 to get back original password. To embed any any secret message we have to enter the password and to extract message we have to enter the same password. The size of the secret message file we convert into 32 bits binary and then convert it into 4 characters and write onto cover file. When we want to extract encrypted secret message from a cover file then we first extract the file size from the cover file and extract the same amount of bytes from cover file. Now we will describe the algorithms which we have used in our present study:

We read one byte at a time from the encrypted secret message file(ESMF) and then we extract 8 bits from that byte. After that we read 8 consecutive bytes from the cover file(CF). We check the LSB of each byte of that 8 byte chunk whether it is different from the bits of ESMF. If it different then we replace that bit by the bit we obtain from the ESMF. Our program also counts how many bits we change and how many bytes we change and then we also calculate the percentage of bits changed and percentage of bytes changed in the CF. Now we will demonstrate in a simple case.:

Suppose we want to embed "A" in the cover text "BBCDEFGH". Now we will show how this cover text will be modified after we insert "A" within it.

TABLE -7 CHANGING LSB

Original Text	Bit string	Bit to be inserted in LSB	Changed Bit string	Changed Text
B	01000010	0	0100001**0**	B
B	01000010	1	0100001**1**	C
C	01000011	0	0100001**0**	B
D	01000100	0	0100010**0**	D
E	01000101	0	0100010**0**	D
F	01000110	0	0100011**0**	F
G	01000111	0	0100011**0**	F
H	01001000	1	0100100**1**	I

Here we can see that to embed "A" we modify 5 bits out of 64 bits. After embedding "A" in cover text "BBCDEFGH" the cover text converts to "BCBDDFFI". We can see that the change in cover text is prominent as we are trying to embed text within text which is actually not possible using LSB method. But when we do it in some image or audio file then it will not be so prominent.

To extract byte from the cover file we follow the reverse process which we apply in case of encoding the message. We simply extract serially one by one from the cover file and then we club 8 bits and convert it to a character and then we write it to another file. But this extracted file is now in encrypted form and hence we apply decryption process which will be the reverse of encryption process to get back original secret message file.

IV. RESULTS AND DISCUSSION

Case-1: Cover File type=.jpg Secret File type=.jpg

Fig_1:Cover file name: sxcn.jpg Size=1155378 Bytes

+

Fig_2:Secret message File:joy1.jpg Size=1870 Bytes

=

Fig_3: Embedded Cover file name :sxcn.jpg Size=1155378 Bytes

(secret message encrypted before embedding)

Case-2: Cover File type=.BMP secret message file =.doc

Fig_4: Cover File name : tvshow.bmp Size=688856 Bytes.

Fig_5: Embedded Cover File name : tvshow.bmp Size=688856 Bytes
In this file an encrypted word file xxfile2.doc(size=19456B) is embedded

Case-3: Cover File type=.BMP secret message file =.jpg

Fig_6: Cover file name = tvshow1.bmp (size=688856B)

+

Fig_7: Secret message file= tuktuk1.jpg(size=50880B) (The secret message file was Encrypted while embedding)

=

Fig_8: Embedded cover file name=tvshow1.bmp (size=688856B)

Case-4: Cover File type=..AVI(Movie File) secret message file =.jpg

Fig_9: Cover File Name= Name=rhinos.avi(size=76800B)

+

Fig_10: Secret message File name = tuktuk_bw.jpg (Size=1870B)

=

Fig_11:Embedded Cover File Name=rhionos.avi. (Size=76800B)
(The encrypted secret message file is embedded)

V. CONCLUSION

In the present work we try to embed some secret message inside any cover file in encrypted form so that no one will be able to extract actual secret message. Here we use the standard steganographic method i.e. changing LSB bits of the cover file. Our encryption method can use maximum encryption number=64 and maximum randomization number=128. The key matrix may be generated in 256! Ways. So in principle it will be difficult for anyone to decrypt the encrypted message without knowing the exact key matrix. Our method is essentially stream cipher method and it may take huge amount of time if the files size is large and the encryption number is also large. The merit of this method is that if we change the key_text little bit then the whole encryption and decryption process will change. This method may most suitable for water marking. The steganography method may be further secured if we compress the secret message first and then encrypt it and then finally embed inside the cover file.

VI. ACKNOWLEDGEMENT

AN sincerely expresses his gratitude to Department of Computer Science for providing necessary help and assistance. AN is also extremely grateful to University Grants Commission for providing fund for continuing minor research project on Data encryption using symmetric key and public key crypto system. JN is grateful to A.K. Chaudhury School of I.T. for giving inspiration for research work.

REFERENCES

[1] Symmetric key cryptography using random key generator, A.Nath , S.Ghosh, M.A.Mallik, Proceedings of International conference on SAM-2010 held at Las Vegas(USA) 12-15 July,2010, Vol-2,P-239-244

[2] Data Hiding and Retrieval, A.Nath, S.Das, A.Chakrabarti, Proceedings of IEEE International conference on Computer Intelligence and Computer Network held at Bhopal from 26-28 Nov, Page-392-397, 2010.

[3] Advanced steganographic approach for hiding encrypted secret message in LSB ,LSB+1,LSB+2 and LSB+3 bits in non standard cover files, Joyshree Nath, Sankar Das, Shalabh Agarwal and Asoke Nath , to be published in IJCA(USA),Vol 14-No.7, P-31-35, February 2011.

[4] Cryptography and Network , William Stallings , Prectice Hall of India

[5] Modified Version of Playfair Cipher using Linear Feedback Shift Register, P. Murali and Gandhidoss Senthilkumar, UCSNS International journal of Computer Science and Network Security, Vol-8 No.12, Dec 2008.

[6] Jpeg20000 Standard for Image Compression Concepts algorithms and VLSI Architectures by Tinku Acharya and Ping-Sing Tsai, Wiley Interscience.

[7] Steganography and Seganalysis by Moerland, T , Leiden Institute of Advanced Computing Science.

[8] SSB-4 System of Steganography using bit 4 by J.M.Rodrigues Et. Al.

[9] An Overview of Image Steganography by T.Morkel, J.H.P. Eloff and M.S.Oliver.

[10] An Overview of Steganography by Shawn D. Dickman

[11] Hamdy, S., El-messiry, H., Roushdy, M., & Kahlifa, E. (2010). Quantization Table Estimation in JPEG Images. International Journal of Advanced Computer Science and Applications - IJACSA, 1(6), 17-23.

[12] Lalith, T. (2010). Key Management Techniques for Controlling the Distribution and Update of Cryptographic keys. International Journal of Advanced Computer Science and Applications - IJACSA, 1(6), 163-166.

[13] Hajami, A., & Elkoutbi, M. (2010). A Council-based Distributed Key Management Scheme for MANETs. International Journal of Advanced Computer Science and Applications - IJACSA, 1(3).

[14] Meshram, C. (2010). Modified ID-Based Public key Cryptosystem using Double Discrete Logarithm Problem. International Journal of Advanced Computer Science and Applications - IJACSA, 1(6). Retrieved from http://ijacsa.thesai.org/.