Volume 3, No. 4, April 2012
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

A NEW DATA HIDING ALGORITHM WITH ENCRYPTED SECRET MESSAGE USING TTJSA SYMMETRIC KEY CRYPTO SYSTEM

Sayak Guha[1], Tamodeep Das[2], Saima Ghosh[3] , Joyshree Nath[4], Sankar Das[5] and Asoke Nath[6]
[1,2,5,6]Department of Computer Science,St. Xavier's College(Autonomous), Kolkata,India
[3]Cognizant Technology Solutions, Kolkata, India
[4]A.K.Chaudhuri School of IT,Raja Bazar Science College,Calcutta University
[1]sayak0290@gmail.com, [2]tamodeep1@yahoo.co.in [3]saima.ghosh@gmail.com, [4]joyshreenath@gmail.com,
[5]dassankar16@yahoo.co.in, [6]asokejoy@gmail.com

Abstract: In the present work we are proposing a new steganogarphy method to hide any encrypted secret message inside a cover file by substituting in the LSB. For encrypting secret message we have used new algorithm namely TTJSA developed by Nath et al [10]. For hiding secret message we have used a method proposed by Nath et al [2]. The TTJSA method comprises of 3 distinct methods which are also developed by Nath et al[1,7]. The methods are MSA[Meheboob, Saima and Asoke][1], NJJSAA[Neeraj, Joel, Joyshree, Amlan, Asoke][7] and Generalised Modified Vernam Cipher method developed by Nath et al[10]. The authors have used TTJSA method for encryption purpose as it is already proved that TTJSA is very effective even if we have small pattern such as digital watermark or password etc. Moreover the cryptanalysis of TTJSA shows that the standard attack like differential attack or simple plain text attack will not be able to break the encryption method. So the main advantage of this method is that even if the hacker can extract the embedded data from a host file but they can not get back the original secret message. While embedding encrypted secret message we have used the standard LSB substitution method [2]. The present method may be used for hiding very confidential message or password or any private key from one machine to another machine or from one machine to server etc. For sending question papers normally the teachers are sending it through e-mail as normal plain text. Instead of that now they can encrypt it first using TTJSA method and hide the encrypted message in some popular image and send it to destination with full confidence like in between no one will be able to hack it. In defense or in Banking sector also the present method may be used for sending some crucial and important message. The present method may be used to hide any confidential message such as text, audio, image in any image or audio or video file.

Keywords: MSA,TTJSA,NJJSAA,LSB,Vernam,Steganography

INTRODUCTION

In the present work we have used two(2) distinct algorithms (i) to encrypt secret message(SM) using TTJSA(Trisha,Tamodeep,Joyshree,Shayan,Asoke) proposed by Nath et al.[10]. (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. One can extract SM from CF but it can not be decrypted as one has to execute 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(.BMP or .JPEG) 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.

a. ***LSB insertion method****:* Here we substitute the bits of the secret message into LSB position of every byte of the cover file. Now we choose some bit pattern where we want to embed some secret text:

11000100 00001100 11010010 10101101
00101101 00011100 11011100 10100110

Suppose we want to embed a number 224 in the above bit pattern. Now the binary representation of 224 is 11100000. To embed this information we need at least 8 bytes in cover file. Here we have taken 8 bytes in the cover file.

Now we modify LSB of each byte of the cover file by each of the bit of secret text 11100000. Now we want to show what happens to cover file text after we embed 11100000 in the LSB of all 8 bytes:

Table 1 Changing LSB

Before Replacement	After Replacement	Bit inserted	Remarks
00101101	00101101	1	No change in bit pattern
00011100	00011101	1	Change in bit pattern(1)
11011100	11011101	1	Change in bit pattern(1)
10100110	10100110	0	No change in bit pattern
11000100	11000100	0	Change in bit pattern(0)
00001100	00000100	0	No change in bit pattern
11010010	11010010	0	No change in bit pattern
10101101	10100100	0	No change in bit pattern

Here we can see that out of 8 bytes only 3 bytes(red marked) get changed only at the LSB position. That means the change of byte is minimum. The human 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 first skip 5000 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. The size of the cover file should not be less than 10*sizeof(secret message). For extracting embedded file from the cover file we have to perform the following:

We have to enter the password while embedding a secret message file. Once we get the file size we follow simply the reverse process of embedding a file in the cover file. We read bit from LSB of each byte and accumulate 8 bits to form a character and we immediately write that character on to a file.

We made an exhaustive 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	Any file type provided the size of the secret message file is very small in compare to cover file
3.	.DOC	Any small file
4.	.WAV	.BMP,.JPG,.TXT,.DOC
5.	.AVI	.TXT,.WAV,.JPEG
6.	.PDF	Any small file

After doing exhaustive study on all possible type of files we conclude that the .BMP file is the most appropriate file which can be used for embedding any type of file.

TTJSA ALGORITHM:

TTJSA is a combination of 3 distinct cryptographic methods, namely, (i) Generalized Modified Vernam Cipher Method, (ii) MSA method [1] and (iii) NJJSA method [7]. To initiate the encryption process a user has to enter a text-key which may be at most 16 characters in length. From the text-key the randomization number and the encryption number is calculated using a method proposed by Nath et al. [1]. A minor change in the text-key will change the randomization number and the encryption number quite a lot. We have tested this method on various types of known text files and we have found that, even if there is repetition in the input file, the encrypted file contains no repetition of patterns.

Now here we will describe TTJSA algorithm:

Algorithm for Encryption:

Step 1: Start
Step 2: Initialize the matrix mat[16][16] with numbers 0 to 255 in row major wise.
Step 3: call keygen() to calculate randomization number (=times), encryption number (=secure)
Step 4: call randomization () function to randomize the contents of mat [16][16].
Step 5: set times2=times
Step 6: copy file f1 into file2
Step 7: set k=1
Step 8: if k>secure go to Step 15
Step 9: p=k%6
Step 10: if p=0
call vernamenc(file2,outf1)
set times=times2
call njjsaa(outf1,outf2)
call msa_encryption(outf2,file1)
else if p=1
call vernamenc(file2,outf1)
set times=times2
call msa_encryption(outf1,file1)
call file_rev(file1,outf1)
call njjsaa(outf1,file2)
call msa_encryption(file2,outf1)
call vernamenc(outf1,file1)
set times=times2
else if p=2
call msa_encryption(file2,outf1)
call vernamenc(outf1,outf2)
set times=times2
call njjsaa(outf2,file1)
else if p=3
call msa_encryption(file2,outf1)
call njjsaa(outf1,outf2)
call vernamenc(outf2,file1)
set times=times2
else if p=4
call njjsaa(file2,outf1)
call vernamenc(outf1,outf2)
set times=times2
call msa_encryption(outf2,file1)
else if p=5
call njjsaa(file2,outf1)
call msa_encryption(outf1,outf2)
call vernamenc(outf2,file1)
set times=times2
Step 11: call function file_rev(file1,outf1)
Step 12: copy file outf1 into file2
Step 13: k=k+1
Step 14: goto Step 8
Step 15: End

Algorithm of Vernamenc (f1,f2)

The algorithm of vernamenc() function is a block cipher method. The block-wise encryption procedure is shown in Figure 2. 'Feedback' of each character is used for the encryption of the next character.
Step 1: Start vernamenc() function
Step 2: The matrix mat[16][16] is initialized with numbers 0-255 in row major wise order
Step 3: call function randomization() to randomize the contents of mat[16][16].
Step 4: Copy the elements of random matrix mat[16][16] into key[256] (row major wise)
Step 5: set pass=1, times3=1, ch1=0
Step 6: Read a block from the input file f1 where number of characters in the block $\leq$ 256 characters
Step 7: If block size < 256 then goto Step 15
Step 8: copy all the characters of the block into an array str[256]
Step 9: call function encryption() where str[] is passed as parameter along with the size of the current block
Step 10: if pass=1
set times=(times+times3*11)%64
set pass=pass+1
else if pass=2
set times=(times+times3*3)%64
set pass=pass+1
else if pass=3
set times=(times+times3*7)%64

```
            set pass=pass+1
          else if pass=4
            set times=(times+times3*13)%64
            set pass=pass+1
else if pass=5
            set times=(times+times3*times3)%64
            set pass=pass+1
          else if pass=6
set times=(times+times3*times3*times3)%64
            set pass=1
```

Step 11: call function randomization() with current value of times
Step 12: copy the elements of mat[16][16] into key[256]
Step 13: read the next block
Step 14: goto Step 7
Step 15: copy the last block (residual characters, if any) into str[]
Step 16: call function encryption() using str[] and the no. of residual characters
Step 17: Return

Algorithm of function Encryption (str[],n):

Step 1: Start encryption() function
Step 2: set ch1=0
Step 3: calculate ch=(str[0]+key[0]+ch1)%256
Step 4: write ch into output file
Step 5: set ch1=ch
Step 6: set i=1
Step 7: if i≥n then goto Step 13
Step 8: ch=(str[i]+key[i]+ch1)%256
Step 9: write ch into the output file
Step 10: ch1=ch
Step 11: i=i+1
Step 12: goto Step 7
Step 13: Return

Algorithm for Decryption:

Step 1: Start
Step 2: initialize mat[16][16] with 0-255 in row major wise
Step 3: call function keygen() to generate times and secure
Step 4: call function randomization()
Step 5: set times2=times
Step 6: call file_rev(f1,outf1)
Step 7: set k=secure
Step 8: if k<1 go to Step 15
Step 9: call function file_rev(outf1,file2)
Step 10: set p=k%6
Step 11: if p=0

```
            call msa_decryption(file2,outf1)
            call njjsaa(outf1,outf2)
            call vernamdec(outf2,file2)
            set times=times2
        else if p=1
            call function vernamdec(file2,outf1)
            set times=times2
            call function msa_decryption(outf1,outf2)
            call fumction njjsaa(outf2,file2)
            call function file_rev(file2,outf2)
            call function msa_decryption(outf2,outf1)
            call function vernamdec(outf1,file2)
            set times=times2
        else if p=2
            call njjsaa(file2,outf1)
            call vernamdec(outf1,outf2)
            set times=times2
            call msa_decryption(outf2,file2)
        else if p=3
            call vernamdec(file2,outf1)
            set times=times2
            call njjsaa(outf1,outf2)
call msa_decryption(outf2,file2)
        else if p=4
            call msa_decryption(file2,outf1)
            call vernamdec(outf1,outf2)
            set times=times2
            call njjsaa(outf2,file2)
        else if p=5
            call vernamdec(file2,outf1)
            set times=times2
            call msa_decryption(outf1,outf2)
            call njjsaa(outf2,file2)
```

Step 12: copy the content of file2 to outf1
Step 13: set k=k-1
Step 14: Goto Step 8
Step 15: End

Algorithm of function Vernamdec (f1,f2):

The algorithm of vernamdec() function is same as vernamenc() function. Here the only difference is that decryption() function is called instead of encryption() function.

Algorithm of Decryption (str[],n):

Step 1: Start
Step 2: ch1=0
Step 3: ch=(256+str[0]-key[0]-ch1)%256
Step 4: write ch into the output file
Step 5: i=1
Step 6: if i≥n then goto Step 12
Step 7: ch=(256+str[i]-key[i]-str[i-1]) %256
Step 8: write ch into the output file
Step 9: i=i+1
Step 10: goto Step 6
Step 11: ch1=str[n-1]
Step 12: Return

Algorithm of Function file_rev (f1,f2) :

Step 1: Start
Step 2: open the file f1 in input mode
Step 3: open the file f2 in output mode
Step 4: calculate n=sizeof(file f1)
Step 5: move file pointer to n
Step 6: read one byte
Step 7: write the byte on f2
Step 8: n=n-1
Step 9: if n>=1 then goto step-6
Step 10: close file f1, f2
Step 11: Return

The encryption number (=secure) and randomization number (=times) is calculated according to the method mentioned in MSA algorithm [1].

NJJSAA ALGORITHM

Nath et al. [2] proposed a method which is basically a bit manipulation method to encrypt or to decrypt any file.

Step 1: Read 32 bytes at a time from the input file.
Step 2: Convert 32 bytes into 256 bits and store in some 1-dimensional array.
Step 3: Choose the first bit from the bit stream and also the corresponding number(n) from the key matrix.
Interchange the 1st bit and the n-th bit of the bit stream.
Step 4: Repeat step-3 for 2nd bit, 3rd bit…256-th bit of the bit stream
Step 5: Perform right shift by one bit.
Step 6: Perform bit(1) XOR bit(2), bit(3) XOR bit(4),…,bit(255) XOR bit(256)
Step 7: Repeat Step 5 with 2 bit right, 3 bit right,…,n bit right shift followed by Step 6 after each completion of right bit shift.

MSA (MEHEBOOB, SAIMA, ASOKE) ENCRYPTION AND DECRYPTION ALGORITHM

Nath et al. [2] 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. MSA method provides us multiple encryptions and multiple decryptions. The key matrix (16x16) is formed from all characters (ASCII code 0 to 255) in a random order.

The randomization of key matrix is done using the following function calls:

Step-1: call Function cycling()
Step-2: call Function upshift()
Step-3: call Function downshift()
Step-4: call Function leftshift()
Step-5: call Function rightshift()

For detail randomization methods we refer to the done by Nath et al. [1].

Table-4→

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

Table-3 Key Matrix

Now we will describe how we perform the encryption process using MSA algorithm.

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).

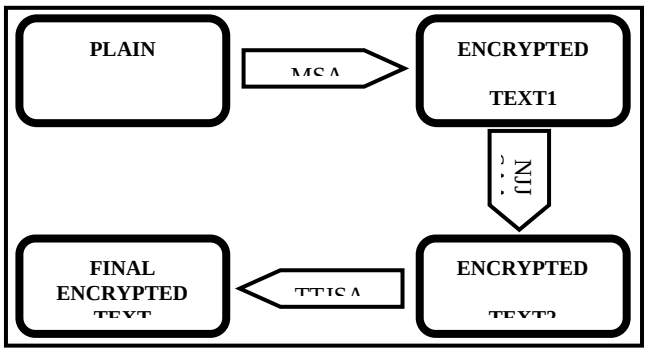
Case III: Suppose we want to encrypt EF where EF occurs in the same row. Here EF will be converted to HG.

After encrypting 2 bytes we write the encrypted bytes on a new output file. We apply the entire encryption method multiple times. The encryption number will be determined by the process described in the method described by Nath et al [1].

The decryption method will be just the reverse process of the encryption method as described above.

THE OVERALL ENCRYPTION AND DECRYPTION METHOD

The three methods are applied on the plain text based on a decision-making parameter. The order in which the three methods are used in a pass is dynamic. Again each method can be used in a pass more than one time. We suggest that the TTJSA method should be used more than one time in a pass to give better result.



CHANGING LSB BIT OF COVER FILE USING ENCRYPTED SECRET MESSAGE FILE

In the present work the last 5000 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 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 -5 Changing Lsb

Original Text	Bit string	Bit to be inserted in LSB position	Changed Bit string	Changed Text
B	01000010	0	01000010	B
B	01000010	1	01000011	C
C	01000011	0	01000010	B
D	01000100	0	01000100	D
E	01000101	0	01000100	D
F	01000110	0	01000110	F
G	01000111	0	01000110	F
H	01001000	1	01001001	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. Total 5 characters is been modified. For text file this change is noticeable 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. Now this extracted file is encrypted form and hence we apply decryption process which will be the reverse of encryption process to get back original secret message file.

RESULTS AND DISCUSSION

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

Figure_1:Cover file name: sxc.jpg (Size=4KB)

+ MSAERC1.c

Figure _2:Secret message File:msaerc1.c File name: sxc1.jpg (secret message encrypted before embedding) (Size=1129KB)

=

Figure _3: Embedded Cover file (Size=1129KB)

Case-2: Cover File type=.AVI Secret message file =.jpg

Figure _4: Cover File name : rhinos.avi (Size=768000 B)

+

Figure _5:Secret message File : joy1.jpg (Size=1870 B) (secret message encrypted before embedding)

=

Figure_6:Embedded Cover File name :rhinos.avi (size=768000 B)

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

Figure _7: Cover file name =image2.bmp (size=673KB)

+

Figure _8: Secret message file= sg3.bmp(size=50KB) (The secret message file was Encrypted while embedding)

=

Figure _9: Embedded cover file name=image2n.bmp (size=673KB)

Case-4: Cover File type=..DOC(MS-Word File) secret message file =.C

Mydoc.doc

Figure _10: Cover File Name= mydoc.doc (Size=22528 B)

+ xxprog2.c

Figure _11: Secret message File name =xxprog2.c (size=136 B) (The encrypted secret message file is embedded)

= Mydoc.doc

Figure _12:Embedded Cover File name= mydoc.doc (Size=22528B)

CONCLUSION

In the present work we hide some secret message inside any cover file in encrypted form so that no one will be able to extract actual secret message. Here we change LSB bit of the cover file. Our encryption mechanism is too hard to break by any intruder. Without knowing the actual encryption process no one can unhide the actual secret message. TTJSA is free from differential attack or simple plain text attack. Even if the intruder could extract the data from the embed cover file but he/she will not be able to decrypt it just by using some brute force method. In the present method there two way protection one at the time of unhide data and a second key at the time decrypting the data. These two keys to preserved by user in safe custody to extract secret message from any host file.

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 be most suitable for water marking. The steganography method may be further enhanced by using QR-code initially for data hiding and then the entire QR-code may be inserted in some image. In QR-code again we can insert data in encrypted form. This will give more strength in steganography method. If we compress the secret message first and then encrypt it and then finally embed it then we can insert more data in same host file. Presently we are working on last two methods for data hiding.

ACKNOWLEDGEMENT

The authors are sincerely express their gratitude to Department of Computer Science, St. Xavier's College(Autonomous) 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 doing research work.

REFERENCES

[1]. Symmetric Key Cryptography using Random Key generator : Asoke Nath, Saima Ghosh, Meheboob Alam Mallik: "Proceedings of International conference on security and management(SAM'10" held at Las Vegas, USA Jull 12-15, 2010), P-Vol-2, 239-244(2010).

[2]. Data Hiding and Retrieval : Asoke Nath, Sankar Das, Amlan Chakraborti, published in IEEE "Proceedings of International Conference on Computational Intelligence and Communication Networks (CICN 2010)" held from 26-28 NOV'2010 at Bhupal.

[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, International Journal of Computer Applications, Vol14-No.7,Page-31-35, Feb(2011).

[4]. Advanced Symmetric key Cryptography using extended MSA method: DJSSA symmetric key algorithm: Dripto Chatterjee, Joyshree Nath, Soumitra Mondal, Suvadeep Dasgupta and Asoke Nath, Jounal of Computing, Vol3, issue-2, Page 66-71,Feb(2011).

[5]. Advanced Steganography Algorithm using encrypted secret message : Joyshree Nath and Asoke Nath, International Journal of Advanced Computer Science and Applications, Vol-2, No-3, Page-19-24, March(2011).

[6]. A Challenge in hiding encrypted message in LSB and LSB+1 bit positions in any cover files : executable files, Microsoft office files and database files, image files, audio files and video files : Joyshree Nath, Sankar Das, Shalabh Agarwal and Asoke Nath : JGRCS, Vol-2, No.4, Page:180-185,April (2011)

[7]. New Symmetric key Cryptographic algorithm using combined bit manipulation and MSA encryption algorithm: NJJSAA symmetric key algorithm :Neeraj Khanna, Joel James,Joyshree Nath, Sayantan Chakraborty, Amlan Chakrabarti and Asoke Nath : Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 03-06 June 2011, Page 125-130(2011)..

[8]. New Data Hiding Algorithm in MATLAB using Encrypted secret message :Agniswar Dutta, Abhirup Kumar Sen, Sankar Das,Shalabh Agarwal and Asoke Nath : Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 03-06 June 2011, Page 262-267(2011).

[9]. An efficient data hiding method using encrypted secret message obtained by MSA algorithm: Joyshree Nath, Meheboob Alam Mallik , Saima Ghosh and Asoke Nath : Proceedings of the International conference Worldcomp 2011 held at Las Vegas(USA), 18-21 Jul(2011), Page 312-318, Vol-1(2011)

[10]. Symmetric key cryptosystem using combined cryptographic algorithms- generalized modified vernam cipher method, MSA method and NJJSAA method: TTJSA algorithm – Trisha Chatterjee, Tamodeep Das, Joyshree Nath, Shayan Dey and Asoke Nath, Proceedings of IEEE International conference : World Congress WICT-2011 t held at Mumbai University 11-14 Dec, 2011, Page No. 1179-1184(2011).

[11]. A new randomized data hiding algorithm with encrypted secret message using modified generalized Vernam Cipher Method: RAN-SEC algorithm, Rishav Ray, Jeeyan Sanyal, Tripti Das, Kaushik Goswami, Sankar Das and Asoke Nath, Proceedings of IEEE International conference : World Congress WICT-2011 held at Mumbai University 11-14 Dec, 2011, Page No. 1215-1220 (2011).

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

