International Journal of Computer Applications (0975 – 8887)
Volume 46– No.14, May 2012

Advanced Digital Steganography using Encrypted Secret Message and Encrypted Embedded Cover File

Joyshree Nath
A.K.Chaudhuri School of I.T.
Calcutta University
Kolkata, India

Saima Ghosh
Cognizent Technology Solutions
Kolkata, India

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

ABSTRACT

In the present work the authors are proposing a new steganography method to hide any encrypted secret message in multiple steps. In step-1 the secret message is encrypted using TTJSA algorithm. In step-2 we embed this encrypted message inside one known image or audio file using 4-th bit from LSB substitution method. In step-3 we again encrypt the embedded cover file using TTJSA algorithm . Finally embed this encrypted cover file in final known cover file which may be some image or audio file. In the present method the authors used data hiding and encryption in two steps to make the entire steganography method almost unbreakable. The sender and the receiver have to share a set of secret keys which should not be shared by any outsider who is not the sender or the receiver. Due to multiple time encryption and multiple time data hiding it is almost impossible for any intruder to extract secret message from embedded cover file. Nath et al developed several steganography algorithm where the encrypted secret message is embedded inside some standard or non-standard cover file. It means the data hiding was done in one step but in the present study the encryption is done in two steps and data hiding is also done in two steps.. For encrypting secret message the authors have used new algorithm namely TTJSA developed by Nath et al. In TTJSA method the authors have used bit-exchange, bit-xor and modified playfair method i.e. MSA.. For hiding encrypted message the authors have used substitution of bits in 4-th bit from LSB of the cover file. The present method may be used for sharing secret keys between sender and receiver and also to send some very important confidential message from sender to receiver. In defense or in Banking sector also the present method may be used for sending some crucial and important message.

General Terms

Data hiding and Retrieval.

Keywords

MSA, TTJSA ,NJJSAA, LSB, Steganography, encryption

1. INTRODUCTION

In the present work we have used two(2) distinct algorithms (i) to encrypt secret message(SM) using NJJSAA (Neeraj, Joel, Joyshree, Sayantan, Amlan, Asoke) proposed by Nath et al.[7]. (ii) We insert the encrypted secret message inside the standard cover file(CF) by changing the 4-th bit from LSB proposed by Nath et al[2]. In the present work we have basically tried to make the steganography method more secured. Instead of taking one cover file we have taken here 2 cover files and one secret message. The size of secret message must be less than 5-10% of cover file-1. The size of cover file-1 must be less than 10% of the 2nd or the final cover file. The cover file can be both image files(preferably .bmp file) or audio(.wav or .mp3) or video file(.avi) and the secret message may be image or text or any small file. In our present work we first encrypt secret message using TTJSA algorithm. Then we insert that secret message inside some known cover file such known image or known audio file. After embedding the encrypted secret message inside cover file-1 we again encrypt the embedded cover file-1 using TTJSA method. Finally we embed the encrypted cover file-1 where the secret message is already embedded inside the 2nd cover file using substitution of 4-th bit from LSB. Now we will describe our method:

(i) 4-th from LSB insertion method: Here we substitute the bits of the secret message into 4-th bit from LSB positions of a byte in a cover file. Here we choose some bit pattern where we want to embed some secret text:

Suppose we want to embed a character '**a'** in the above bit pattern. Now the binary representation of 'a'(ASCII=97) is **01100001**. To embed this information we need 8 bytes in cover file. We assume that the cover file contains a pattern **"ABCDEFGH"**.. Now we want to show what happens to cover file text after we embed **01100001** in the 4-th bit of each byte of the cover file:

Table-1: 4-th bit substitution in cover File

Before Insertion	Bit to be inserted	After Insertion	Remarks
01000001=A	1	01001001=**I**	Change Byte
01000010=B	0	01000010=B	No Change
01000011=C	0	01000011=C	No change
01000100=D	0	01000100=D	No change
01000101=E	0	01000101=E	No change

01000110=F	1	01001110=**N**	Change Byte
01000111=G	1	01001111=**O**	Change Byte
01001000=H	0	01000000=**@**	Change Byte

It shows **ABCDEFGH is** modified to **IBCDENO@**

Here we can see that out of 8 bytes only 4 bytes(red marked) get changed.. It means the change of byte is about 50%. 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 few 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 15*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.

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

2. TTJSA ALGORITHM

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 ≤ 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].

3. 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.

4. 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-3 KEY MATRIX

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

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

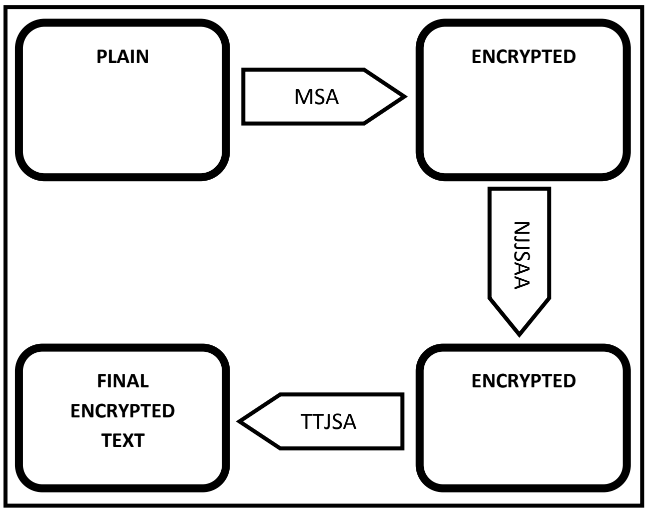

Fig-1: TTJSA Encryption Process

4. Changing 4-th bit from LSB of Cover File using encrypted secret message file:

In the present work the last 200 bytes of the cover file we reserve 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=4 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 4-th bit from LSB bytes 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. 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.

5. RESULTS AND DISCUSSION

We apply the present proposed data encryption and data hiding method on various type of host files and secret file and we found that the method is working absolutely ok. In the present study we will be presenting two different cases:

Case-I: Cover File-1(type): .bmp file, Cover File-2(type: .bmp file and Secret message file) : .bmp file.

(i) secret message File=joy1.jpg(size=1870 bytes)

(ii) Embedded Cover message file-1 : sxc.bmp
(size=(492,210 bytes)

(iii)Cover File-2(Final cover file) :
image1.bmp(size=29,942,838 bytes)

Now we will describe how we hide the secret message inside a cover file :

Step-1: joy1.jpg→Apply TTJSA→ joy1n.jpg (encrypted).

Step-2: Combine joy1n.jpg + logo.bmp → embedded logo.bmp(visible).

Step-3: logo.bmp(embedded)→Apply TTJSA→ logo1.bmp (encrypted).

Step-4:Combine logo1.bmp+image1.bmp→embedded image1.bmp(visible as shown above)

To unhide the secret message we have to follow the following steps.

Step-1: Unhide from image1.bmp → logo1n.bmp (encrypted embedded cover file-1).

Step-2: logo1n.bmp → Applly TTJSA decryption algorithm → logon.bmp(embedded cover file-1 which is visible).

Step-3: Unhide joy1c1.bmp from logon.bmp,
Joy1c1.bmp→Unhide joy1c1.jpg(encrypted secret image file)

Step-4: Decrypt joy1c1.jpg using TTJSA decryption algorithm to obtain original secret image file:
Joy1c1.jpg → Apply TTJSA decryption algorithm→ joy1d.jpg(Original secret image and it is visible).

So to hide some secret message we have to perform 4 steps and to unhide the same secret message we have to perform 4 steps.

Case-II:
(i) **Secret message file : algo.txt(size=1528 bytes)**
Content of secret message file:-
ALGORITHM

```
        ENCRYPTION
Step 1          : Start
Step 2          : mat[][] is initialized (values from 0 to
255) in row major manner
Step 3          : keygen() and randomization() function
called
Step 4          : times2=times
Step 5          : file f1 is copied into file2
Step 6          : k=1
Step 7          : if k>secure go to Step 13
Step 8          : p=k%6
Step 9          : if   p=0
```

```
            vernamenc(file2,outf1)
            times=times2
            njjsaa(outf1,file1)
            msa_encryption(file1,file2)
            njjsaa(file2,outf1)
            vernamenc(outf1,file1)
            times=times2
     else if  p=1
            vernamenc(file2,outf1)
            times=times2
            msa_encryption(outf1,file1)
            file_rev(file1,outf1)
            njjsaa(outf1,file2)
            msa_encryption(file2,outf1)
            vernamenc(outf1,file1)
            times=times2
     else if  p=2
            msa_encryption(file2,outf1)
            vernamenc(outf1,outf2)
            times=times2
            njjsaa(outf2,file2)
            vernamenc(file2,outf1)
            times=times2
            msa_encryption(outf1,file1)

     else if  p=3
            msa_encryption(file2,outf1)
            njjsaa(outf1,outf2)
            vernamenc(outf2,file2)
            times=times2
            njjsaa(file2,outf1)
            msa_encryption(outf1,file1)

     else if  p=4
            njjsaa(file2,outf1)
            vernamenc(outf1,outf2)
            times=times2
            msa_encryption(outf2,file2)
            vernamenc(file2,outf1)
            times=times2
            njjsaa(outf1,file1)

     else if  p=5
            njjsaa(file2,outf1)
            msa_encryption(outf1,outf2)
            vernamenc(outf2,file2)
            times=times2
            msa_encryption(file2,outf1)
            njjsaa(outf1,file1)
Step 10     : file_rev(file1,outf1)
Step 11     : file outf1 is copied into file2
Step 12     : k=k+1 Goto Step 7
Step 13     : Stop
```

(ii) Cover File-1(embedded with encrypted algo.txt): aud2.mp3(size=55633 bytes),

(iii) Cover File-2(embedded cover file with encrypted audio file aud2.mp3) : image7.bmp (size=5760054 bytes)

To embed secret image(joy1.jpg) Here we have done the following:

(i) Embed first one small text file algo.txt in encrypted form in an audio file aud2.mp3.

(ii) Embed the encrypted audio file aud2.mp3 is embedded inside an image image7.bmp and we can see there is no distortion in the image or even in the sound file.

Step-1: Convert algo.txt→Apply TTJSA→ algo1.txt (encrypted)

Step-2: Combine algo1.txt + aud2.mp3 → embedded aud2.mp3(playable)

Step-3:Convert aud2.mp3(embedded)→Apply TTJSA→ aud2n(encrypted and not playable)

Step-4: Combine aud2n.mp3 + iamge7.bmp→embedded image7.bmp (clear and visible)

For un hiding secret message from the last cover file we have to follow the same process as we have mentioned in case-I.

We have tested our method to hide any type of file in any cover file such as image or audio or video and we found it is working 100% ok.

6. CONCLUSION

In the present work we hide some secret message inside any cover file in encrypted form and then we encrypt that cover file again and finally we embed that encrypted cover file inside another cover file so that final cover file should remain intact. This way we propose a double protection so that no one will be able to extract actual secret message. Here we change LSB and LSB+1 bits 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 embedded cover file but he/she will not be able to decrypt it just by using some brute force method. In the present method there are two way protections keys 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.

7. 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.

8. 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.