Key Based Bit Level Genetic Cryptographic Technique (KBGCT)

Subhranil Som
Dept. of Computer Application, JIS College of Engineering, India
E-mail: subhranil.som@gmail.com

Niladri Shekhar Chatergee
WebSpiders India Pvt. Ltd, India
E-mail: niladris.chatterjee@gmail.com

J. K. Mandal
Dept. of Computer Sc. & Eng.
University of Kalyani, India
E-mail: jkm.cse@gmail.com

***Abstract*—This is an encryption and decryption algorithm with the help of genetic functions cryptography. This new algorithm is developed for encryption and decryption process. This algorithm combines the features of Genetic Algorithm in Cryptography. Here we generate random numbers for "Crossover" and "Mutation". The encryption and decryption algorithms will be made public. The algorithm contains a key, which is known to only sender and receiver. In this technique the input file is broken down into different blocks of various sizes. The main algorithm works in two stages. Bit Level XOR operation followed by Genetic Crossover and Mutation.**

Keywords-Key Based Bit Level Genetic Cryptographic Technique (KBGCT), Encryption, Decryption, Cipher text, plain text, Genetic Algorithm.

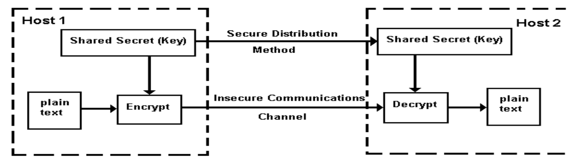
I. INTRODUCTION

Cryptography is an age old art of sending secret messages between two entities. Now with the advancement in internet technologies the requirement of cryptography has changed, but the basis is same transmission of secret message between two entities. The early 1980s saw the proposal of a fundamental and radical idea to get beyond iterative, attack-responsive design of cryptographic algorithms: Goldwasser and Micali [2,5,7,10], followed by Blum-Micali [3,6,8,9] and Yao [4,11,12,13,14,15], suggested that security could be proved under "standard" and well-believed complexity theoretic assumptions[1]. The ultimate strength of a cryptographic algorithm lies in the extent of difficulty faced in breaking the cryptographic system in which the algorithm has been used. The lesser the possibility of breaking the cryptographic system, the higher is the strength of the algorithm. The ultimate responsibility of a system designer is to design an unbreakable cryptographic system [16].

In section II of this paper is discussed about the algorithm. In section III is discussed results and analysis; in section IV will discuss the conclusions followed by references.

II. THE SCHEME

The main algorithm works in two stages, namely Bit Level XOR operation followed by Genetic Crossover and Mutation.



Stages of KBGCT algorithm are a) Bit level XOR operation b) Genetic Crossover and Mutation.

Working process of KBGCT

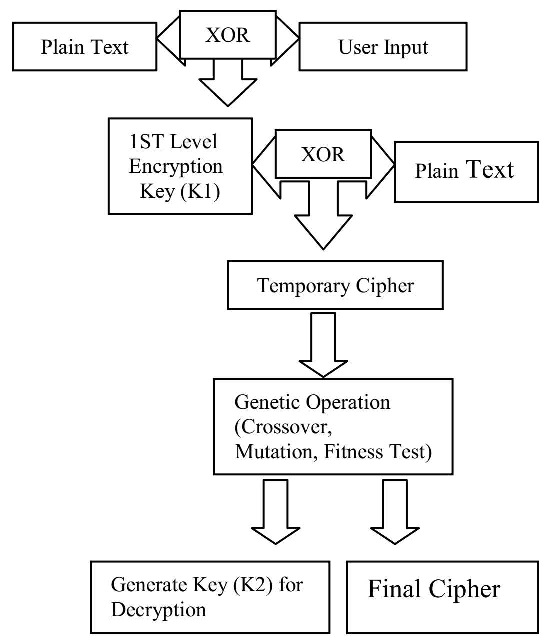


Figure: 1
Working Process of KBGCT

Encryption Algorithm

Stage 1:

a) Calculation of the N length of bits stream is to be made.

b) Division of the bit stream into small blocks of different lengths is to be done.

c) Calculate of the length (n) of each block is to be performed.

d) Two numbers randomly between 0 and n1 for each block are to be chosen or an input password from user is to be taken.

e) Bits from each position of each block are to be picked up and the key (K) is to be generated.

f) Division of plain text into length of key (K) termed as S is to be made.

g) XOR operation between S and K has to be performed.

h) Some transformation technique with the key (Optional) is to be used.

Explanation of step a) to h):

110000101010001010010101010001011101001010101001010100101001010000111111

Plain Text [Size (N) = 72]

11000010 10100010 10010101 01000101 11010010 10101001 01010010 10010100 00111111

The plain text is divided into 9 blocks. Each block size is 8 (n)

Two index positions randomly from each block. [0 to 7 (0 to n-1)] are chosen.

Say: 1st block 1 and 7

2nd block 5 and 7 and so on.

1 7 5 7 3 6 0 4 1 5

1**1**00001**0** 10100**0**1**0** 100**1**01**0**1 **0**100**0**101 1**1**010**0**10

2 7 1 6 45 0 3

10**1**0100**1** 0**1**0100**1**0 1001**01**00 **0**01**1**1111

Generated key is (K):

100011111101001001

XOR operation with the key with each block S is made. The results are:

110000101010001010 010101010001011101
001010101001010100 101001010000111111

100011111101001001

010011010111000011 110110101100010100
101001010100011101 011010101001110110

Cipher Text of Level – 1

i) Repeatation of the XOR operation with each block has been made again reversing the key.

j) Reversing key (K1) is the key that will be required to decrypt the cipher.

Stage 2:

a) In this step, the cipher of level-1 as input stream will be used.

b) The bit stream into pair of blocks (N) of variable length is to be divided.

c) The length of each pair (Li) of block will be chosen randomly.

d) The fitness (F1) of the bits stream pair is to be calculated.

e) A single point crossover function on the first pair of bit stream is to be applied to produce Child bit stream. After crossover, mutation on these child chromosomes is to be applied. For each pair of bits stream several Child for different Crossover point (CP1) produced and their fitness will be calculated.

Fitness Function:

a. Each child with their parents is to be compared.

b. The child which is having maximum difference from its parents will be chosen.

f) From these child chromosomes one chromosome (Cr1) which has the maximum fitness will be chosen.

g) Cr1 is reversed to produce first child chromosome C1.

h) The same process with next pair of chromosome will be applied and produced the fittest child chromosome (Cr2) with crossover point (CP2) and will be reversed it to produce second child chromosome C2.

i) The process from step d) to step h) will be continued until all child chromosomes are produced.

j) After completion of these steps we will get our final cipher C1,C2,C,,C4,…………Cn1, Cn.

k) And key for level-2 will be NL1CP1L2CP2 L3CP3L4CP4…………..Ln1CPn1LnCPn. After completing the total process our final cipher is C1C2C3C4…………Cn1Cn and the secret key is NL1CP1L2CP2L3CP3L4CP4…………..Ln1CPn1LnCPnK.

Example:

First we consider that we have input stream of bits (plain text) containing a word "COMPUTER' and user input is "ROM".

Key Generation Process:

In this example we generated key from user input (Stage 1 (d)) has be generated.

1. The algorithm receives an input stream of bits from the user "COMPUTER" and a user input is "ROM".

2. It is converted into equivalent binary stream of bits and shown in Table 1 and Table 2

Table 1

Binary equivalent of "Computer"

Input Stream		
Character	ASCII value	Binary Value
'C'	67	01000011
'O'	79	01001111
'M'	77	01001101
'P'	80	01010000
'U'	85	01010101
'T'	84	01010100
'E'	69	01000101
'R'	82	01010010

Table 2

Binary equivalent of "ROM"

User Input		
Character	ASCII value	Binary value
'R'	82	01010010
'O'	79	01001111
'M'	77	01001101

Binary stream of bits for "COMPUTER" are:

01000011010011110100110101010000010101010101000100010101010010

Binary stream of bits for "ROM" are:

010100100100111101001101

3. Length of key is 24. Now a part of input stream of length 24 is taken arbitrarily.

Say the part is 101001101010100000101010. XOR is operated with the user input.
(010100100100111101001101)XOR(101001101010100000101010)
= 111101001110011101100111 (Key K)

Encryption:

1. Binary stream of bits for "COMPUTER" are taken:
0100001101001111010011010101010000010101010101010100 0100010101010010
2. XOR operation of the key (K) with binary stream of plain text has been made. Result is:
1011011110101000001010101010010010110010001100111011000110110101
3. The key (K) is revered and again XOR operation is made with the above result. Now we get
0101000101001111000001010100001001010101000111000101011101010010
4. The length of the bit stream is 64. Now a random number which is a factor of the length is taken. Say 16 is the number.
5. Previous bit stream has been divided into four blocks. The size of each block is 16 as shown in Figure 2

0101000101001111	0000010101000010	0101010100011100	0101011101010010

Figure 2

6. First two blocks from Figure 2 are taken and crossover between them has been made. The crossover point is chosen randomly ranging from 1 to 16. Some of the child chromosomes are given below in the Table 3

Table 3

Crossover between first pair of chromosomes

Chromosome 1	Chromosome 2	Child Chromosome	Cross over Pt
0101000101001111	0000010101000010	01010001010000100000010101001111	7
0101000101001111	0000010101000010	01010001010000100000010101001111	11
0101000101001111	0000010101000010	01010101010000100000000101001111	4
0101000101001111	0000010101000010	01010001010000100000010101001111	9
0101000101001111	0000010101000010	01010001010011100000010101000011	14

From the above child chromosomes we select the fittest child chromosome. The fittest chromosome is one which has having maximum difference from parent chromosomes. Now we select the child
01010001010000100000010101001111
whose crossover point is 9.
Now a mutation function is applied on this child chromosome which just inverts the bit at the crossover point. So the final child chromosome is
01010001000000100000010101001111.
7. Step 6 is repeated for next pair of blocks (chromosomes) from Figure 3.8. Say, the second child chromosome is
01110111010100100101010100011100 whose crossover point is 3.
8. These child chromosomes are concatenated and reversed to produce final cipher text. So our final cipher text is:
0011100010101010010010101110111011110010101000000100000010001010
If we represent the bits stream as string then our cipher is 8ªJîò¿½u and the final key is "ROM1693", where "ROM" is user input, each block is 16 and 9, 3 is crossover point.

Decryption

1. The decryption algorithm receives the Cipher and the Key from user.
2. Key 1, Block size and crossover points are extracted from the key.
3. The cipher is converted into binary stream. The cipher is:
0011100010101010010010101110111011110010101111111011110101110101
4. The binary stream is reversed and the result is:
0101000100000001000000101010011110111011101010010 0101010100011100
5. The bit stream is divided into Blocks. Block size is extracted from key. In this example the block size is 16.
0101000100000010 0100101011101110 1111001010111111 1011110101110101
6. The bits at crossover point in each pair of blocks are inverted. These are:
0101000101000010 0000010101001111 0101011101010010 0101010100011100
7. Crossover between each pair of block with supplied crossover points has been made. The result is:
0101000101001111000001010100001001010101000111000101011101010010
8. XOR operation of the above bits stream with reversed Key 1 has been made. The result is:
1011011110101000001010101010010010110010001100111011000110110101
9. Again XOR operation of the above bits stream with Key 1 has been made. The result is:
0100001101001111010011010101010000010101010101010100 0100010101010010
10. To get back the original Plain texts, just represent the above bit stream into string and we will get "COMPUTER".

III. RESULTS AND ANALYSIS

In this section the implementation of different types of files are presented. Files are chosen at random comprising of various file sizes and file types. Analysis includes comparing encryption and decryption times, Chi-Square values. Implementation of all algorithms and different types of tests has been done using JAVA.

A. *Studies on Text File*

The ten text files of different sizes are taken for testing. The encryption time, the decryption time and source file sizes are noted for T-DES, RSA and KBGCT algorithms. Table 4 and Table 5 show the encryption/decryption time of increasing size of text files for the proposed RSA, TDES, and KBGCT technique. Pictorial form is given in Figure 3.

Table 4
File size v/s Encryption Time for TXT files
(For RSA, TDES and KBGCT algorithm)

Source File Size (*.TXT)	Encryption Time (Seconds)		
	RSA	T-DES	KBGCT
9232	~0	2	2.8
2166	~0	4	6.61
29344	~0	5	7.74
34029	~0	5	8.2
41992	~0	7	9.3
53100	1.89	9	16.9
67140	2.10	11	21.2
71864	2.89	12	21.4
80376	3.78	13	23.78
119840	5.34	20	33.9

Table 5
File size Decryption Time for TXT files
(For RSA, TDES and KBGCT algorithm)

Source File Size (*.TXT)	Decryption Time (Seconds)		
	RSA	T-DES	KBGCT
9232	~0	2	0.4
2166	~0	4	1.32
29344	~0	5	1.87
34029	~0	6	3.42
41992	~0	8	6.21
53100	1.90	9	10.32
67140	2.01	11	13.78
71864	2.71	12	14.01
80376	3.12	13	16.78
119840	4.11	21	20.34

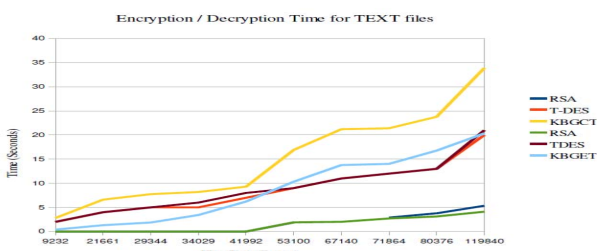


Figure 3
Encryption / Decryption Time for KBGCT, RSA, and TDES Techniques for TEXT files

B. Studies on Executable Files

The ten executable files of different sizes are taken for testing. The encryption time, the decryption time and source file sizes are noted for T-DES, RSA and KBGCT algorithms. Table 6 and Table 7 show the encryption/decryption time of increasing size of text files for the proposed RSA, TDES, and KBGCT technique. Pictorial form is given in Figure 4.

Table 6
File size v/s Encryption Time for EXE files
(For KBGCT, RSA and TDES algorithm)

Source File Size (*.EXE)	Encryption Time (Seconds)		
	RSA	T-DES	KBGCT
8192	~0	1	2.48
12768	~0	2	4.28
13824	~0	2	4.6
20480	~0	4	7.21
28160	~0	4	9.07
55296	~0	9	18
71680	`0	11	23.2
93184	1	16	30.6
110278	3	21	37.8
123672	5	24	44.7

Table 7
File size v/s Decryption Time for EXE files
(For KBGCT, RSA and TDES algorithm)

Source File Size (*.EXE)	Decryption Time (Seconds)		
	RSA	T-DES	KBGCT
8192	~0	1	1.46
12768	~0	2	3.70
13824	~0	2	3.67
20480	~0	3	6.85
28160	~0	5	8.98
55296	~0	10	17.01
71680	~0	12	22.01
93184	1	16	28.11
110278	3	22	36.12
123672	5	25	42.34

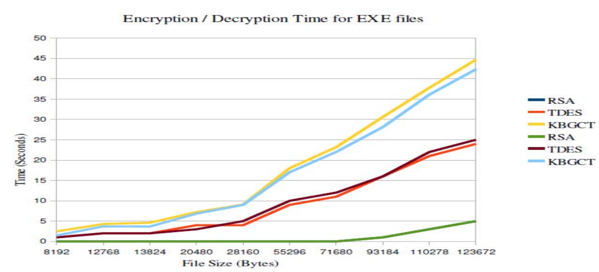


Figure 4
Encryption / Decryption Time for RSA, T-DES and KBGCT techniques for EXE files

C. Studies on DLL Files

Time analysis has also been done for dynamic link libraries. Ten files of different sizes are taken for consideration. Table 8 and 9 show the encryption and decryption time in seconds taken for proposed KBGCT, TDES and RSA techniques. The pictorial effects of the same are shown in Figure 5.

Table 8
File size v/s Encryption Time for DLL files
(For KBGCT, RSA and TDES algorithm)

Source File Size (*.DLL)	Encryption Time (Seconds)		
	RSA	T-DES	KBGCT
8704	~0	2	3.45
14672	~0	3	4.7
18944	~0	5	7.765
25032	~0	5	9.965
30152	~0	6	9.797
35840	~0	9	13.875
35848	~0	9	13.924
42440	1	10	16.984
67072	1	13	26.031
67080	1	13	26.875

Table 9
File size v/s Decryption Time for DLL files
(For KBGCT, RSA and TDES algorithm)

Source File Size (*.DLL)	Decryption Time (Seconds)		
	RSA	T-DES	KBGCT
8704	~0	2	2.21
14672	~0	3	3.70
18944	~0	4	6.31
25032	~0	5	7.980
30152	~0	7	8.102
35840	~0	7	12.011
35848	~0	8	12.132
42440	1	9	14.241
67072	1	14	25.011
67080	1	15	25.121

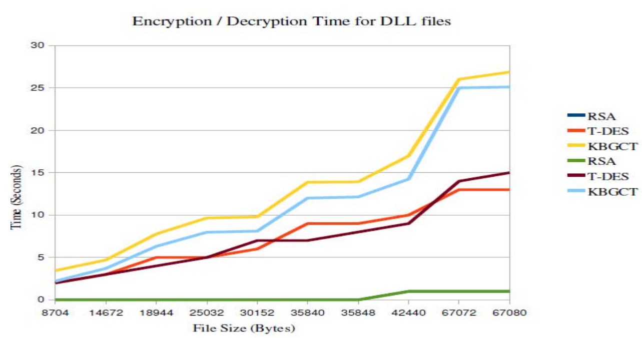


Figure 5
Encryption / Decryption Time for KBGCT, RSA, and TDES Techniques for DLL files

D. Analysis of Character Frequencies

Distribution of character frequencies are analyzed for text file for the proposed KBGCT, RSA and TDES algorithms. Figure 6 shows the pictorial representation of distribution of character frequencies for different techniques. Figure 6(a) shows the distribution of characters in the source file “redist.txt”. Figure 6(b) and 6(c) show the distribution of characters in encrypted files both for RSA and TDES respectively. Figure 6(d) gives the distribution of characters in encrypted file using the proposed technique KBGCT. It’s seen from the picture that in the case of RSA the distribution of characters in encrypted file is concentrated in a small region, whereas both for TDES and the proposed technique KBGCT frequencies of encrypted file are distributed over the complete spectrum of characters. From this observation it may be concluded that the proposed technique may obtain good security.

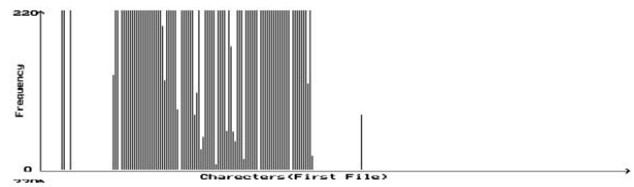


Figure 6(a): Distribution of characters in the source file “**redist.txt**”

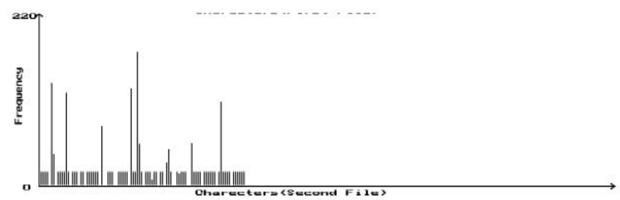


Figure 6(b): Distribution of characters in the encrypted file of RSA

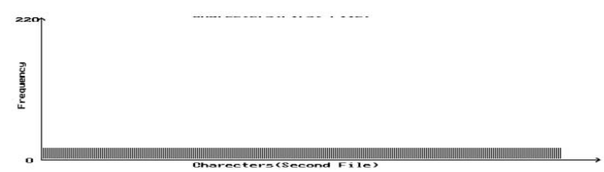


Figure 6(c): Distribution of characters in the encrypted file of T-DES

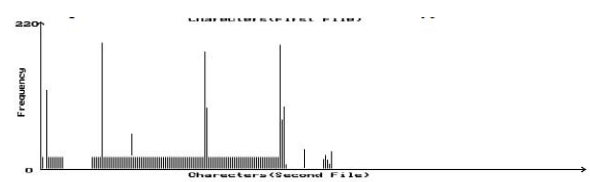


Figure 6(d): Distribution of characters in the encrypted file of KBGCT

Figure 6
Frequency distribution graph of the file “redist.txt” for RSA, TDES, and KBGCT as the source file

E. Tests for Non-Homogeneity

The well accepted parametric tests have been performed to test the non-homogeneity between source and encrypted files. The large Chi-Square values may confirm the heterogeneity of the source and encrypted files. Text files are taken for experiment. The Chi-Square test has been performed using source file and encrypted files for KBGCT technique and existing RSA and TDES techniques. For non-homogeneity the value of the Chi-Square should increase for increasing the file size. Ten files of different sizes are taken. Further the high Chi-Square value may ensure the non-homogeneity between source and encrypted files. In all three cases of implementation a good degree of non-homogeneity has been observed. So it may be inferred that proposed KBGCT technique may ensure optimal security in transmission. The pictorial representations of Chi-Square values are given in Figure 7**.**

Table 10
Chi-Square values

Source File Name	**Chi-Square values**		
	RSA	**T-DES**	**KBGCT**
ils.txt	2034.26	1925.98	2199.20
leesmy.txt	4016.59	3832.11	5129.80
ukraine.txt	8858.46	8350.29	10005.32
cp1257.txt	17515.38	17505.79	20209.41
roman.txt	26342.44	25816.80	28068.21
corpchar.txt	35554.74	34303.53	38010.12
oemeula.txt	43383.58	41574.40	43979.31
incidia.txt	51774.95	49557.47	54083.49
readme.txt	60525.72	57889.44	62601.62
schedule.txt	67193.24	64269.12	69301.37

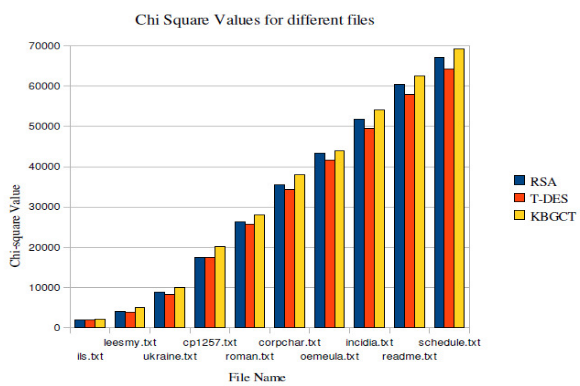


Figure 7
Chi Square Values for RSA, KBGCT and TDES

IV. Conclusion

The above technique KBGCT is showing the good result in different result analysis. Chi-Squire value show the good result comparing with RSA and TDES, which is indicating the high security of the proposed technique. Analysis of character frequency of KBGCT also shows the better result in comparing with RSA. It is taking little bit more encryption and decryption time in comparing with RSA and TDES. It will be no matter because it shows good results in character frequency analysis and Chi Square value to support the high security.

References

[1] Bellare Mihir and Rogaway Phillip", (1993) Entity Authentication and Key Distribution", Advances in Cryptology [Crypto '93 Proceedings, August].

[2] Goldwasser S. and Micali S., (1984) "Probabilistic encryption," Journal of Computer and System tem Sciences Vol. 28, 270-299 (April).

[3] Blum M. and Micali S., (1984) "How to generate cryptographically strong sequences of pseudorandom bits," SIAM Journal on Computing 13(4), 850-864 (November). Yao, A. C., \Theory and applications of trapdoor functions," Proceedings of the Twenty.

[4] Cryptography and Network Security, Atul Kahate, Fourth Reprint.

[5] A cipher based on 3D Array Rotation, P. R. Suri, Sukhvinder Singh Deora.

[6] Byron S. Gottfried, "Programming with C" TATA McGraw HILL, Second Edition, 1998.

[7] Herbert Schildt, "Java: The Complete Reference", Tata McGrawHill Publishing Company Limited, 5th Edition.

[8] E.Balagurusamy, "Programming with Java", Tata McGrawHill Publishing Company Limited, 3rd Edition.

[9] Ankit Fadia, "Network Security", Macmillan India Ltd.

[10] William Stallings, "Cryptography and Network Security", Prentice Hall, 3rd Edition.

[11] Subhranil Som, Joytsna Kumar Mandal, (2009) "Random Byte Value Shift (RBVS) Algorithm", JIS Management Vista, Vol. III, No. 1, pp.81-88.

[12] Som S., Mitra D., Halder J., (2008) "Session Key Based Manipulated Iteration Encryption Technique (SKBMIET)", the 2008 International Conference on Advanced Computer Theory And Engineering ICACTE 2008), 20-22, December 2008, Phuket, Thailand.

[13] Som S., Bhattacharyya K, Roy Guha R., Mandal J.K., (2009) "Block Wise Bits Manipulations Technique (BBMT)", The 2009 International Conference On Advanced Computing, 6-8, August 2009, Tiruchirappalli, India.

[14] Som S., Mandal J. K., (2008) "A Session Key Based SecureBit Encryption Technique (SBET)", National Conference (INDIACom2008) on Computing For Nation Development, February 08-09, 2008, New Delhi, India.

[15] Som S., Mitra D., Halder J., (2008) "SecureBit Rotate and Swapped Encryption Technique (SBRSET)", National Conference on Trend in Modern Engineering System (IConTiMES 2008), February 23-24, 2008, WB, India.

[16] Ankit Fadia and Jaya Bhattacharjee, "Encryption protecting your data", Vikas Publishing House Pvt. Ltd.".