Volume 4, No. 12, December 2013
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

MODIFIED MULTI WAY FEEDBACK ENCRYPTION STANDARD VER-2(MWFES-2)

Saptarshi Chatterjee[*1], Debdeep Basu [2], Ankita Bose[3], Surajit Bhowmik [4] and Asoke Nath[5]

*[1]Department of Computer Science, St. Xavier's College(Autonomous), Kolkata, West Bengal, India
sapishere.chatterjee@gmail.com [1]
[2]Department of Computer Science, St. Xavier's College(Autonomous), Kolkata, West Bengal, India
debdeepbasucal@hotmail.com[2]
[3]Department of Computer Science, St. Xavier's College(Autonomous), Kolkata, West Bengal, India
ankkitabose@hotmail.com [3]
[4]Department of Computer Science, St. Xavier's College(Autonomous), Kolkata, West Bengal, India
bhowmik1994@gmail.com[4]
[5]Department of Computer Science, St. Xavier's College(Autonomous), Kolkata, West Bengal, India
asokejoy1@gmail.com[5]

Abstract: Nath et al developed a method Multi Way Feedback Encryption Standard Version-I [18] recently, where the authors used both forward and backward feedback from left to right and from right to left on the plain text along with the key. In MWFES-I, the ASCII value of plain text is added with key and forward feedback (FF) and backward feedback (BF) to obtain intermediate cipher text. The initial FF and BF are taken to be 0. The intermediate cipher text is taken modulo operation with 256 to get cipher text. This cipher text is taken as feedback for the next column. In the second round we calculate the cipher text from the RHS. In the MWFES-II, the authors have used a much more general approach. The FF and BF have been applied using skip by n-columns; where 'n' can be 0 to any number less than the length of the plain text. In the present method (Modified Multiway Feedback Encryption Standard, Version -2), the authors have introduced two different skips 'n1' and 'n2'. We perform 'n1' skips for forward feedback (that is, from the left), and 'n2' skip for backward feedback (that is, from the right). 'n1' and 'n2' may or may not be equal and is taken as a function of the generated keypad. The authors applied the present method on some standard plain texts such as 1024 ASCII '0', 1024 ASCII '1', 1024 ASCII '2' and 1024 ASCII '3' and the frequency analysis shows the encrypted texts are totally random. Initially, the user has to enter a secret key (seed). The key-expansion algorithm generates an enlarged keypad of the size of the plain-text from the seed. This keypad is used for further encryption and decryption. The present method is very effective as the encrypted text changes drastically on varying the skip 'n1' and 'n2'. Modified MWFES-2 can be applied to encrypt any short message, password, confidential message or any other important document. The results show that the present method is free from standard attacks such as differential attack, known plain text attack etc.

Keywords: *MWFES, ASCII, Confidential Message, Encryption*

INTRODUCTION

Data encryption is a very important research area now-a-days. Plain text or clear text should not be used for sending confidential messages because the security might get compromised. In the last two decades, quite a number of encryption algorithms have been developed. Some of the methods are almost unbreakable and are used widely in different sectors like business, academic etc. There is also a parallel process going on, that is, to break the encryption algorithm using some common attacks such as middle-man attack, differential attack, known plain text attack, brute force attack etc. Nath et al developed various cryptographic algorithms such as MSA, DJSA, DJMNA, TTJSA, MES-I,II,III, UES-I,II,III,IV, BLES-I,II,III,IV [1-15]. Nath et al for the first time introduce feedback in Vernam cipher method to develop generalized Vernam Cipher Method. Nath et al developed Multi Way Feedback Encryption Standard Ver-I(MWFES-I)[18] where the authors used plain texts, randomized key, forward feedback(FF) and backward feedback(BF) simultaneously to encrypt any plain text. The authors used FF from LHS and BF from RHS and in this way the entire file was encrypted. In the present method, Modified MWFES-II [19] e authors have made the system even more general. Depending on the key entered by the user, one can skip 'n1' number of characters while performing forward feedback and 'n2' number of characters while performing backward feedback. 'n1' and 'n2' varies from 0 to any number less than the length of the file. The results show that the encryption process depends a lot on theses skip factors. This method is a novel method because the skip characters can be different in different blocks of characters. The present encryption method can be applied multiple times to make the system fully secured. Thorough tests were conducted on some standard plain text files and it was found that it is absolutely impossible for any intruder to extract any plain text from encrypted text using any brute force method. The results show that the present method is also free from any kind of known plain text or differential attack.

ALGORITHMS FOR MODIFIED MWFES-2

In the present section we will be discussing the encryption algorithm, key generation algorithm as well as decryption algorithm.

Algorithm for Function Encryption()

Step 1: Input 'pt'(Plain Text),'seed'(secret key)
Step 2: Calculate integral part of square root of Plain Text length
Step 3: Call KeyGeneration function to get complete key in 'key'
Step 4: Get forward_skip, backward _skip from the key.
Step 5: Call encryption_method(pt,key,forward_skip, backward_skip) to get Cipher Text 'ct'
Step 6: End

Algorithm For Function Encryption_Method (pt,key,forward_skip, backward_skip)

Step 1: i=1 to length(pt) do
Step 2: sum[i]=sum of ASCII codes of pt, key, ff, bf
Step 3: modulo of (sum[i] ,256) added to ct
Step 4: If forward skip exceeds length(pt) bring it back to the front
Step 5: Initialize 'ind' to keep tab of backward feedback
Step 6: sum[ind]=sum of ASCII codes of pt, key, ff, bf
Step 7: modulo of (sum[ind],256) added to ct
Step 8: If backward skip exceeds length(pt) bring it back to the front
Step 9: If i<length(pt) then i=i+1 and go to Step 2 else return ct array to calling function

Algorithm for Function Key_Generation(seed[],n)

From the secret_key (seed), the program will generate an enlarged keypad. We keep this keypad in a square matrix having dimensions equal to the nearest greater perfect square of the Plain Text length. The first element of this keypad is taken to be the sum of the ASCII codes of the characters in the seed. Now the next elements are simply values that are linearly multiplied with the sum, i.e k*sum for k=1, 2, 3 etc. If there is any repetition of the original sum, then the program will modify the initial sum to be sum=sum+1 and k=1 and the process will continue. The value of sum and its subsequent multiplications are kept within the boundaries of 0-255 by applying the modulo operations on the numbers at every iteration and only after that, the numbers are added to the keypad matrix. The elements of the matrix were reshuffled using some simple operations such as upshift(), leftshift(), diagonal1_shift(), downshift(), rightshift(), diagonal2_shift() etc. These functions were at first called seed length number of times. The order of calling these functions should follow the constraint that leftshift() shouldn't be preceded or succeeded by rightshift(). Similarly upshift() shouldn't precede or succeed downshift(). Now, another variable called Randomization Number was calculated from the generated matrix by adding the diagonal values and then bringing it down to 0-255 by modular operation. The generated matrix was permuted again by calling the various functions Randomization Number of times. Security can be further enhanced by permuting the order in which the functions are called, but maintaining the basic calling constraints. The final key was thus developed using various properties of the seed as well as intrinsic properties of the keypad which is then used by the encryption and decryption methods to find out the different required parameters at each stage of the individual processes.

Key generation method:

Step 1: Initialise seed_len with value of the length of seed
Step 2: Add ASCII codes of the seed characters and store it in sum
Step 3: Initialise a 2 dimensional 'key' array with nxn elements and initialize 'sum1' with the value of sum
Step 4: for i=1 to n do
Step 5: for j=1 to n do
Step 6: key[i,j]= modulo of(sum1,256)
Step 7: If modulo of(sum1,256)==modulo of(sum,256),then go to Step 8 else j=j+1 and go to Step 6
Step 8: sum =modulo of(sum+1,256)
Step 9: sum1=mod(sum,256);
Step 10: If j<n then j=j+1 and go to Step 6 else go to Step 11
Step 11: If i<n then i=i+1 and go to Step 5 else go to Step 12
Step 12: for i=1 to seed_len do
Step 13: Call all Shifting and shuffling operations one by one.
Step 14: If i<seed_len then i=i+1 and go to Step 13 else go to
Step 15:Add all the diagonal terms and store the sum in 'randomization_number'
Step 16: Call all Shifting and shuffling operations one by one 'randomization_number' number of times
Step 17: Return 'key' array to calling function

Algorithm for Function Clockwise(a[],n,i)

Step 1: Start from the top left corner of 'i'th row of matrix a[][]
Step 2: Keeping that row and column as the circular boundary of the matrix and neglecting all elements that do not fall in this circle,we shift each element to the right
Step 3: Return array a[][] to the calling function

Algorithm for Function Anti_Clockwise (a[],n,i)

Step 1: Start from the top left corner of 'i'th row of matrix a[][]
Step 2: Keeping that row and column as the circular boundary of the matrix and neglecting all elements that do not fall in this circle,we shift each element to the left
Step 3: Return array a[][] to the calling function

Algorithm for Function Left_Shift(a[],n)

Step 1: for row =1 to n do
Step 2: Copy all elements of the row to the right of this row to this row of array a[][]
Step 3: If row<=n then row =row-1 and go to Step 2 else go to Step 4
Step 5: Return array a[][] to the calling function

Algorithm for Function Right_Shift(a[],n)

Step 1: for row =1 to n do
Step 2: Copy all elements of the row to the left of this row to this row of array a[][]
Step 3: If row<=n then row =row-1 and go to Step 2 else go to Step 4
Step 5: Return array a[][] to the calling function

Algorithm for Function Up_Shift(a[],n)

Step 1: for column =1 to n do
Step 2: Copy all elements of the column below this column to this column of array a[][]
Step 3: If column<=n then column =column-1 and go to Step 2 else go to Step 4
Step 5: Return array a[][] to the calling function

Algorithm for Function Down_Shift(a[],n)

Step 1: for column =1 to n do
Step 2: Copy all elements of the column above this column to this column of array a[][]
Step 3: If column<=n then column =column-1 and go to Step 2 else go to Step 4

Step 4: Return array a[][] to the calling function

Algorithm for Function Diagonal1_Shift(a[],n)
Step 1: Beginning from top left corner of array a[][] copy all diagonal elements onto their adjacent elements
Step 2: Return array a[][] to Calling function

Algorithm for Function Diagonal2_Shift(a[],n)
Step 1: Beginning from top left corner of array a[][] copy all diagonal elements onto their adjacent elements
Step 2: Return array a[][] to Calling function

Algorithm For Function Decryption()
Step 1: Input encrypted 'ct'(Cipher Text) and 'seed'(secret key).
Step 2: Calculate integral part of square root of Cipher Text length
Step 3: Call Key Generation function to get complete key in 'key' that was obtained during encryption process
Step 4: Get forward_skip and backward_skip from the generated key.
Step 5: Call decryption_method (ct,key,forward_skip,backward_skip) to get retrieved plaintext.
Step 6: End

Algorithm for Function Decryption_Method (ct[],key[], forward_skip,backward_skip)
Step 1: We generate the list of feedback transfers for the block using function Generate List.
Step 2: for k=2*length(ct):-1:length(ct)+1 do
Step 3: We call the various functions like whatIsIn and whatIsInBetween in order to figure out the contents of each elements and what they may contain.
Step 4: Since each element can have at most 2 contents at the time we store the final contents of each in sub1 and sub2
Step 5: The final subtracted value is stored in check.
Step 6: If k>=length(ct)+1then k=k-1 and go to Step 3 else go to step 7
Step 7: Return decrypted PlainText to calling function

Algorithm for Function ConditionCheck (number,i,j,size,v[])
Step 1: if i*j!=0, go to step 2,else go to step 3
Step 2: flag=0
Step 3: if i!=0 go to Step 4,else go to step 8
Step 4: if v[Call oldPosition(i,size)]==number, go to step 5, else go to step 6
Step 5: pos=Call oldPosition(i,size)
Step 6: pos=Call lastPosition(i,size)
Step 8: if v[Call oldPosition(j,size)]==number, go to step 9,else go to step 10
Step 9: pos=Call oldPosition(j,size)
Step 10: pos=Call lastPosition(j,length)
Step 11: flag1=mod(pos,2)
Step 12: if flag1=0, go to step 13,else go to step 14
Step 13: flag=2
Step 14: flag=flag1
Step 15: Return flag to the calling function

Algorithm for Function Is_Changed (number,position,forward_skip,backward_skip,size,u[],v[], ct[],ff[],bf[])
Step 1: Now we begin by identifying the last change that has occurred in number.
Step 2: We store the actual value to be subtracted in 'sub'
Step 3: [i,j]=Call function whatIsInBetween to determine the intermediate values
Step 4: We also determine the immediate values by calling the function whatIsIn
Step 5: Depending on the value of the immediate values we either add the initial forward or backward feedback or we proceed to add all the previously added values into sub.
Step 6: After all the values that have been added in the encryption process are determined and their values added to sub we return sub to the Calling function.

Algorithm for Function What_Is_In (number,size,forward_skip,backward_skip,v[])
Step 1: Call function last position of in order to determine the start from the last occurrence of the character 'number 'destination array list v[].
Step 2: From that point we try to work out whether that is the changed value of that element or not out the beginning of the tree of values that have been inserted into
Step 3: If it is a changed occurrence we try to figure this element thus resulting in the final value of the element
Step 4: Return the values in the element to the calling function

Algorithm for Function What_Is_In_Between (number,size,forward_skip,backward_skip,v[])
Step 1: Begin by calling the function whatIsIn and figuring out what the immediate content of 'number' is
Step 2: Next we try to determine what are intermediate values that have crept into this 'number' during the encryption process
Step 3: We need these intermediate contents in order to subtract these contents and decrypt the result
Step 4: Return the intermediate values to the calling function.

Algorithm for Function GenerateList (size,forward_skip,backward_skip)
Step 1: Initialise source with 1
SteP 2: Input source serially into the u[] array
Step 3: Calculate destination for particular source depending on forward_skip and store destination in v[] array
Step 4: Now initialize source to size
Step 5: Repeat Steps 2,3,4 for backward_skip
Step 6: Return u[] and v[] to the calling function

Algorithm For Function OldPosition (number,size)
Step 1:currrent_pos = Call function last_Position_of (number, size)
Step 2: first_pos = 2*size - current_pos+1
Step 3:-Return first_pos to the calling function.

Algorithm for Function LastPosition (number,size)
Step 1: if number <= ceiling of (size/2); go to Step 3
Step 2: if number >ceiling (size/2); go to Step 4
Step 3: last_ pos = 2*size - 2*(number-1);'
Step 4: last_pos = 2*(number-1);
Step 5: Return last_pos to the calling function.

RESULTS AND DISCUSSIONS

The following list contains the result of some test cases using our encryption method. For each case, we have the plain text, the key, the number of shifts and their corresponding cipher text. The test cases we have shown here are mostly assorted

and random phrases or texts. The spectral analysis of standard ASCII ‘0’, ASCII ‘1’, ASCII ‘2’ , ASCII ‘3’ are also given.

Table 1. Encryption of some Plain Texts using some key and skip

Sl. No.	Plain Text	Seed	Forward Feedback	Backward Feedback	Cipher Text
1	AAAAB AAAA	SXC	1	1	a'ª*ßFåÑ
2	AAAAB AAAA	SXC	1	2	ë=°•@åñ„.
3	AAAAC AAAA	SXC	1	2	ë>°•Aåò„/
4	ABABA BABAB	SXC	2	5	bH†6ÓJç©ª-
5	ACACA CACAC	SXC	2	5	hL†8ÔKé-¬±
6	ACACA CACAC	SXC	3	8	½3ùé§Än4ÕÎ

Inference

From the observations made in the table above we see that even for two similar Plain Texts (as shown in SL. NO. 1. And 2.), the Cipher Texts are drastically different owing to the fact that there is just a slight change in the backward skip making the result completely haphazard. We repeat the experiment for different Plain Texts keeping a few constraints in mind, such as the skip and key and even then we do not see any seemingly visible pattern for deciphering the Plain Text. Even if one character does turn out to be similar in a rare case that would be due to the key being same for both the test cases.

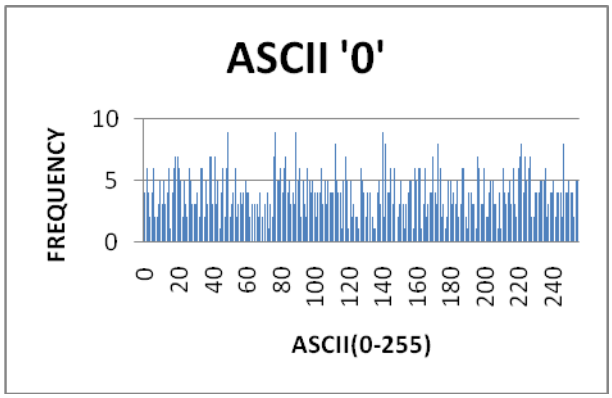

Figure 1. Frequency Spectral Analysis of ASCII ‘0’ Encryption with Seed= ‘SXC’, Forward Skip=12 and Backward Skip=7

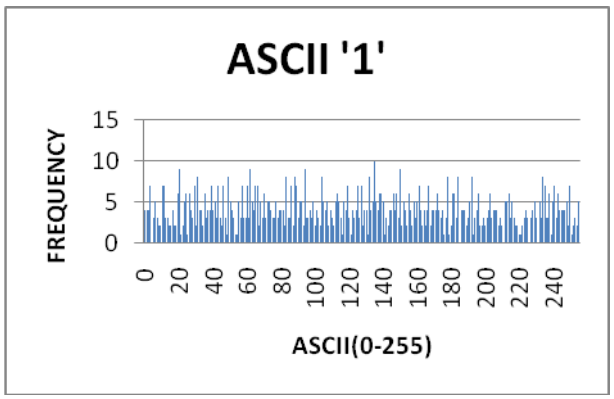

Figure 2. Frequency Spectral Analysis of ASCII ‘1’ Encryption with Seed= ‘SXC’, Forward Skip=12 and Backward Skip=7

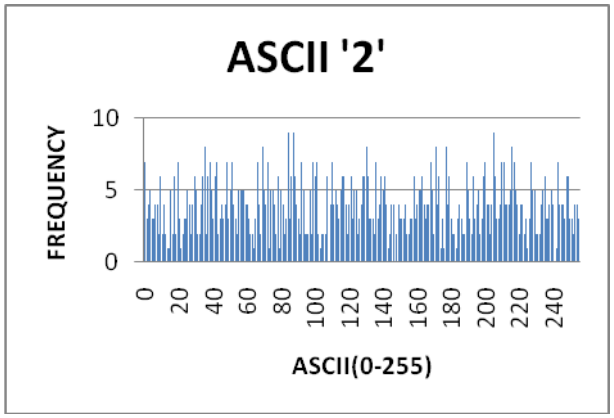

Figure 3. Frequency Spectral Analysis of ASCII ‘2’ Encryption with Seed= ‘SXC’, Forward Skip=12 and Backward Skip=7

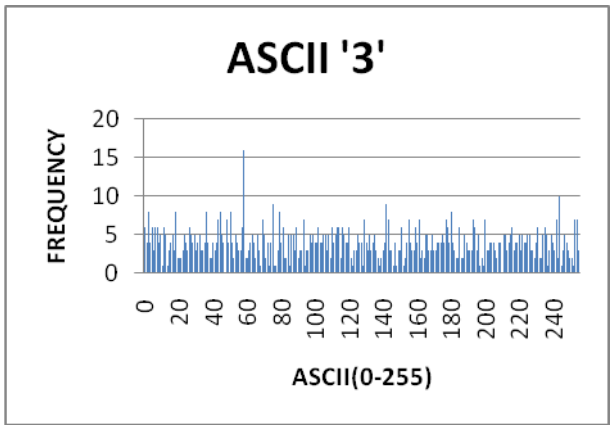

Figure 4. Frequency Spectral Analysis of ASCII ‘3’ Encryption with Seed= ‘SXC’, Forward Skip=12 and Backward Skip=7

Table 2. Encryption of a Small Paragraph

Paragraph	Cipher
St. Francis Xavier is the Patron of St. Xavier's College. There is not an educated Indian who has not heard the name of Francis Xavier. It is to India that Ignatius of Loyola, the Founder of the Society of Jesus, sent his greatest son, Francis Xavier in 1542. Xavier was a zealous "missionary on the move".	èÖ²)M□ #L¾Ê]À?¹Ÿný _°x__ç-òûà"<- €¨£™äC_})j[Ë,DßÀÓ Ü‡ëä®Štt_,CÊûÕr§Ž é2X+Ó%Â_9wú_ñ\Ý=_ }¤AXQO_Mû'û!ÝÖt&- u>í-š_?kÑb¼_ìèÙ_ ž— _ê?__¥\•°EÈ_/é‡°_ ,y?x©Y?? T³wwÔ¤ôÎIhÜû«CSx< Ú5öÑ5÷õ"?Þ¿□Yþ_ê_ è°†-Ã4e"`öÏ_£õå? Yœæ«— 8ªõ_NÄ¨d1RTŒ\4ôSî _šÁ®®_'ñý~Çóñ_Š_* ó×}_-÷¥^×___ÜŠÞ_‰ _ö2-ý^™l(_á"Éß__E ¹•_£ÅL{ç€K%äÜ»°¢? □ ½_R__cé¼]'àÀ¡p_B _º_

Inference

Comparing the result of this paragraph with the table that we had obtained before we see that there is no way of linking the Plain Text with the Cipher Text. If we scrutinize the result in the table above for similar Plain Text characters too we find that in no two places are the Cipher Text characters same. Using a randomized key generated by the key generation algorithm we have managed to remove any discrepancies that may have occurred while using the same key for two Plain Texts.

CONCLUSION AND FUTURE SCOPE

The present method is tested on various types of files such as .doc, .jpg, .bmp, .exe, .com, .dbf, .xls, .wav, .avi and the results were quite satisfactory. The encryption and decryption methods work smoothly. In the present method the encrypted text cannot be decrypted without knowing the exact initial random matrix. The size of random matrix taken is at least 16x16. The numbers in 16x16 may be arranged in 256! Ways. The present method is free from any kind of brute force attack or known plain text attack. The present Modified MWFES Ver-2 method may be applied to encrypt any short message, password, confidential key. One can also apply this method to encrypt data in sensor networks.

ACKNOWLEDGMENT

The authors are very much grateful to the Department of Computer Science for giving the opportunity to work on symmetric key Cryptography. A.N sincerely expresses his gratitude to Fr. Dr. Felix Raj, Principal of St. Xavier's College (Autonomous) for giving constant encouragement in doing research in the field of cryptography.

REFERENCES

[1] Symmetric Key Cryptography using Random Key generator: AsokeNath, Saima Ghosh, MeheboobAlamMallik:"Proceedings of International conference on security and management(SAM '10)" held at Las Vegas, USA July 12-15, 2010), Vol-2, Page: 239-244(2010).

[2] Advanced Symmetric key Cryptography using extended MSA method: DJSSA symmetric key algorithm: DriptoChatterjee, JoyshreeNath, SoumitraMondal, SuvadeepDasgupta and AsokeNath,Journal of Computing, Vol 3, Issue-2, Page 66-71,Feb(2011).

[3] A new Symmetric key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm, DriptoChatterjee, JoyshreeNath, SuvadeepDasgupta and AsokeNath : Proceedings of IEEE International Conference on Communication Systems and Network Technologies, held at SMVDU(Jammu) 03-06 June,2011, Page-89-94(2011).

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

[5] Symmetric key Cryptography using modified DJSSA symmetric key algorithm, DriptoChatterjee, JoyshreeNath, Sankar Das, ShalabhAgarwal and AsokeNath, Proceedings of International conference Worldcomp 2011 held at Las Vegas 18-21 July 2011, Page-306-311, Vol-1(2011).

[6] An Integrated symmetric key cryptography algorithm using generalized vernam cipher method and DJSA method: DJMNA symmetric key algorithm: Debanjan Das, JoyshreeNath, Megholova Mukherjee, NehaChaudhury and AsokeNath: Proceedings of IEEE International conference: World Congress WICT-2011 held at Mumbai University 11-14 Dec, 2011, Page No.1203-1208(2011).

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

[8] Symmetric key Cryptography using two-way updated Generalized Vernam Cipher method: TTSJA algorithm, International Journal of Computer Applications (IJCA, USA), Vol 42, No.1, March, Pg: 34 -39(2012).

[9] Ultra Encryption Standard(UES) Version-I: Symmetric Key Cryptosystem using generalized modified Vernam Cipher method, Permutation method and Columnar Transposition method, Satyaki Roy, NavajitMaitra, JoyshreeNath,ShalabhAgarwal and AsokeNath, Proceedings of IEEE sponsored National Conference on Recent Advances in Communication, Control and Computing Technology - RACCCT 2012, 29-30 March held at Surat, Page 81-88(2012).

[10] An Integrated Symmetric Key Cryptographic Method – Amalgamation of TTJSA Algorithm, Advanced Caeser Cipher Algorithm, Bit Rotation and reversal Method: SJA Algorithm., International Journal of Modern Education and Computer Science, SomdipDey, JoyshreeNath, AsokeNath,(IJMECS), ISSN: 2075-0161 (Print), ISSN: 2075-017X (Online), Vol-4, No-5, Page 1-9,2012.

[11] An Advanced Combined Symmetric Key Cryptographic Method using Bit manipulation, Bit Reversal, Modified Caeser Cipher(SD-REE), DJSA method, TTJSA method: SJA-I Algorithm, Somdip Dey, Joyshree Nath, Asoke Nath, International Journal of Computer Applications(IJCA 0975-8887, USA), Vol. 46, No.20, Page- 46-53,May, 2012.

[12] Ultra Encryption Standard(UES) Version-IV: New Symmetric Key Cryptosystem with bit-level columnar Transposition and Reshuffling of Bits, Satyaki Roy, NavajitMaitra, JoyshreeNath, ShalabhAgarwal and AsokeNath, International Journal of Computer Applications(IJCA)(0975-8887) USA Volume 51-No.1.,Aug, Page. 28-35(2012)

[13] Bit Level Encryption Standard(BLES) : Version-I, NeerajKhanna, DriptoChatterjee, JoyshreeNath and AsokeNath, International Journal of Computer Applications(IJCA)(0975-8887) USA Volume 52-No.2.,Aug, Page.41-46(2012).

[14] Bit LevelGeneralized Modified Vernam Cipher Method with Feedback : Prabal Banerjee, AsokeNath, Proceedings of International Conference on Emerging Trends and Technologies held at Indore, Dec 15-16,2012.

[15] Advanced Symmetric Key cryptosystem using Bit and Byte Level encryption methods with Feedback : Prabal Banerjee, Asoke Nath, Proceedings of International conference Worldcomp 2013 held at Las Vegas, July 2013.

[16] Modern Encryption Standard Ver-IV(MES-IV), Asoke Nath, Payel Pal, International Journal of Advanced Computer Research, Volume-3, Number-II, Page-216-223(2013).

[17] Modern Encryption Standard Ver-IV(MES-V), Asoke Nath, Bidhusunder Samanta, International Journal of Advanced Computer Research, Volume-3, Number-II, Page-257-264(2013).

[18] Multi Way Feedback Encryption Standard Ver-I(MWFES-I) , Purnendu Mukherjee, Prabal Banerjee, Asoke Nath, International Journal of Advanced Computer Research, Volume-3, Number-II, Page-176-182(2013).

[19] Multi Way Feedbach Encryption Standard Ver-2(MWFES ver-2) , Asoke Nath, Debadeep Basu, Ankita Bose, Saptarshi Chatterjee, Surojit Bhowmik, published in IEEE conference proceedings WICT-2013 held at Hanoi, Viertnam, Dec 15-18, Page 318-325(2013).

SHORT BIODATA OF ALL THE AUTHOR

Saptarshi Chatterjee is pursuing his Bachelor Of Science (Computer Science Honors) at St.Xavier's,College is involved (Autonomous),Kolkata, India. He was born in Kolkata on 17.04.1993. He is presently involved in research work in Cryptography.

Debdeep Basu is pursuing his Bachelor Of Science (Computer Science Honors) at St.Xavier's,College (Autonomous),Kolkata, India. He was born in Kolkata on 03.08.1993. He is presently involved in research work in Cryptography.

Ankita Bose is pursuing her Bachelor Of Science (Computer Science Honors) at St.Xavier's,College (Autonomous),Kolkata, India. She was born in Kolkata on 15.02.1993. She is presently involved in research work in Cryptography.

Surajit Bhowmik is pursuing his Bachelor Of Science (Computer Science Honors) at St.Xavier's,College (Autonomous),Kolkata, India. He was born in Kolkata on 24.05.1994. He is presently involved in research work in Cryptography.

Asoke Nath is the Associate Professor in Department of Computer Science. Apart from his teaching assignment he is involved with various research work in Cryptography, Steganography, Green Computing, E-learning. He has presented papers and invited tutorials in different International and National conferences in India and in abroad.

