International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-4 Issue-13 December-2013

Multi Way Feedback Encryption Standard Ver-2(MWFES-2)

Asoke Nath[1], Debdeep Basu[2], Surajit Bhowmik[3], Ankita Bose[4], Saptarshi Chatterjee[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 some random key. In MWFES-I[18], 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 present method, the authors have used a much more general approach. In this paper, the FF and BF has been applied using skip by n-columns, where 'n' can be 0 to any number less than the length of the plain text. This skip, denoted here by 'n', can be generated dynamically from the key. So 'n' can be taken as a function of the key. A comparative study was also made for same plain text, same key and different skip value i.e. 'n'. The results indicate that the encrypted texts are coming totally different, just by varying 'n'. The present method gives almost unlimited scope to encrypt any message. 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 MSA[1] randomization 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 'n'. 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.

Asoke Nath Department of Computer Science, St.Xavier's,College(Autonomous) ,Kolkata, India.
Debdeep Basu Department of Computer Science, St.Xavier's,College(Autonomous),Kolkata, India.
Surajit Bhowmik Department of Computer Science, St.Xavier's,College(Autonomous),Kolkata, India.
Ankita Bose Department of Computer Science, St.Xavier's,College(Autonomous),Kolkata, India.
Saptarshi Chatterjee Department of Computer Science, St.Xavier's,College(Autonomous),Kolkata,India.

Keywords

MWFES, MSA, ASCII, Confidential Message, Encryption

1. Introduction

Data encryption is now-a-days a very important research area. Plain text or clear text should not be used for sending some confidential message 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. The researchers try to develop some effective cryptography method and the hackers try to break that method. Nath et al developed various cryptographic algorithms such as MSA, DJSA, DJMNA, TTJSA, MES-I,II,III,IV,V, UES-I,II,III,IV, BLES-I,II,III,IV [1-18]. 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 MWFES-II, the authors have made the system more general. Depending on the key entered by the user, one can skip 'n' number of characters while performing the feedback encryption process where n varies from 0 to any number less than the length of the file. The results show that the encryption process depends a lot on the skip factor. 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.

2. Algorithm of MWFES Ver-2:

A. Algorithm for Encryption

Step-1: Start.
Step-2: Input the plain text in a character string and then create an integer array which contains the ASCII code for each of the characters. Consider the array to be 'PT[]'.
Step-3: Generate a key using the MSA[1] method developed by Nath et al. Store the key in an array Key[].
Step-4: Also maintain a separate integer array for each of the following
Forward Feedback➔FF[]
Backward Feedback➔BF[]
Sum➔Sum[]
Cipher Text➔CT[]
Step-5: length=length of the plain text.
Step-6: skip= the no. of columns to skip.
Step-7: next=skip+1.
Step-8: i=1
Step-9: if i>length then go to Step-18
Step-10: Sum[i]=PT[i]+Key[i]+FF[i]+BF[i]
Step-11: CT[i]=mod(Sum[i],256)
Step-12: if (i+next) > length then do
FF[i+next-length]=CT[i]
Otherwise do
FF[i+next]=CT[i]
Step-13: j=(length-(i-1)) %to find out the index of backward operation
Step-14: Sum[j]=PT[j]+Key[j]+FF[j]+BF[j]
Step-15: CT[j]=mod(Sum[j],256)
Step-16: if (j-next)<1 then do
BF[length-(absolute(j-next))]=CT[j]
Otherwise do
BF[j-next]=CT[j]
Step-17: i=i+1
Step-18: goto Step-8
Step-19: End

B. Algorithm for Function Decryption()

Start
Step 1: The name of the Cipher Text file is stored in 'ct_file', key file is 'k_file' and output file is out_file.
Step 2: We store the file pointers in different variables.
Step 3: We create an array 'c_txt' which contains all characters of the Cipher Text and another array 'key' to store the key.
Step 4: len = length of c_txt.
Step 5: skip = input of number of characters to be skipped.
Step 6: next = skip+1.
Step 7: [u,v]=Call Generate_u_v(len,next).
Step 8: p_txt=array of length 'len' for decrypted Plain Text containing all zeros.
Step 9: k= (2*len).
Step 10:-[i,j] = Call what_Is_In (u[k],next,len,v).
Step11:-sub_i= Call is_Changed (i,u[k],next,len,v,ct). %% sub_i stores what is to be subtracted from 'i'
Step 12:- sub_j= Call is_Changed (j,u[k],next,len,v,ct). %%Stores what is to be subtracted from 'j'
Step 13:- check= c_txt[u[k]] -sub_i - sub_j- key[u[k]]. %%Un-optimized value of Plain Text
Step 14:-is check < 0; if yes go to Step 15, or else go to step 16
Step 15:-check = check + 256, go to Step 14
Step 16:- is check > 255; if yes go to Step 17, or else go to step 18
Step 17:- check = check - 256, go to Step 16
Step 18:-p_txt[u[k]] = check;
Step 19:-is k > (len+1), if yes go to step 21,or else go to step 20
Step 20:-k = k-1, go to step 10
Step 21:- Copy p_txt into out_file.
Step 22:-End.

C. Algorithm for function Generate_u_v(length,next)

%%u[] will contain the source of the Feedback Transfers
%%v[] will contain the destinations of the Feedback Transfers
Step 1:-source=1.
Step 2:- i=1.
Step 3:-u[i]=source. %%u contains the source of the Feedback Transfers.
Step 4:-if (u[i]+mod(next,length)) >length,then v[i]=u[i]+ mod(next,length) – length.
Step 5:- if (u[i]+mod(next,length)) <= length, then v[i]=u[i]+ mod(next,length).
Step 6:- source=source+1;
Step 7:- if i < (2*length); then i=i+2 and go to Step 3.
Step 8:-source= length.
Step 9:- i =2.
Step 10:-u[i]=source.
Step 11:-if (u[i]-mod(next,length)) < 1,then v[i]=u[i]-mod(next,length) + length.
Step 12:- if (u[i]-mod(next,length)) >= 1, then v[i]=u[i] - mod(next,length).
Step 13:- source=source-1;
Step 14:- if i < (2*length); then i= i+2 and go to Step 10.

Step 15:- Return Control to calling function, also return u[] and v[] to the calling function.

D. Algorithm for function what_Is_In (number,next,length,v[])

%%i and j will store the elements that have been shifted into 'number'
Step-1: if number+next<=length, then go to Step-2,otherwise Step-3
Step-2: i=number+ next
Step-3: i=number+ next-length,
Step-4: if number-next>=1 then go to Step-5,otherwise go to Step-6
Step-5: j=number-next
Step-6: j=number-next+length
Step-7: lastPos_number = Call last_Position _of(number, length)
Step-8: if(i=j and i!=0) then go to Step-9,otherwise go to Step-13
Step-9: if(Call last_Position_of(i,length)>lastPos_number) then go to Step-10,otherwise go to Step-11
Step-10: i=0
Step-11: if(Call first_Position_of(i,length)>lastPos_number) then go to Step-12,otherwise go to Step-23
Step-12: j=0
Step-13: if(i!=0) then go to Step-14 otherwise go to Step-18
Step-14: if(Call last_Position_of(i,length)>lastPos_number and v(Call last_Position_of(i,length))=number) then go to Step-15,otherwise go to Step-16
Step-15: i=0
Step-16: if(Call first_Position_of(i,length)>lastPos_number and v(Call first_Position_of(i,length))=number) then go to Step-17,otherwise go to Step-18
Step-17: i=0
Step-18: if(j!=0) then go to Step-19 otherwise go to Step-23
Step-19: if(Call last_Position_of(j,length)>lastPos_number and v(Call last_Position_of(j,length))=number) then go to Step-20,otherwise go to Step-21
Step-20: j=0
Step-21: if(Call first_Position_of(j,length)>lastPos_number and v(Call first_Position_of(j,length))=number) then go to Step-22,otherwise go to Step-23
Step-22: j=0
Step-23: Return i and j to the calling function

E. Algorithm for function is_Changed (number,mother,next,length,v[],c_txt)

%% 'sub' will store the value that is to be eventually subtracted from that element of the Cipher Text to %%get Plain Text.
Step 1: if number = 0, then, sub = 0. %% thus, a base case is reached
Step 2: else if mother =v(last_Position_of(number,length)), then, sub = c_txt(number). %% thus, a base case is reached.
Step 3: [in_bet_1,in_bet_2]=Call what_Lies_In_Between (number,next,length,v);
Step 4: if i=0 and j=0 , then, sub= c_txt(number). %% thus, a base case is reached
Step 5: else if in_bet_1=0 and in_bet_2~=0 then sub=c_txt(number) - is_Changed(in_bet_2,number,next,length,v,c_txt).
Step 6: else if in_bet_1~= 0 and in_bet_2 = 0, then sub=c_txt(number)-is_Changed(in_bet_1,number,next,length,v,c_txt).
Step 7: else if in_bet_1 ~= 0 and in_bet_2 ~= 0, then sub = c_txt(number) - is_Changed(in_bet_1,number,next,length,v,c_txt) - is_Changed(in_bet_2,number,next,length,v,c_txt).

F. Algorithm for function first_Position_of(number,length)

Step 1: current_pos = Call last_Position_of (number, length);
Step 2: first_pos = 2*length - current_pos+1;
Step 3: Return Control to calling function, and return 'first_pos' to the calling function.

G. Algorithm for function last_Position_of(number,length)

Step 1: if number <= ceil (length/2); go to Step 3
Step 2: if number >ceil (length/2); go to Step 4
Step 3: last_ pos = 2*block size - 2*(number-1);
Step 4: last_pos = 2*(number-1);
Step 5: Return last_pos to the calling function.

H. Algorithm for Function what_Is_In_Between (number,block_size,next,v[])

Step-1: (i,j) = Call whatIsIn (number,length,next,v[])
Step-2: if i=j and i!=0 and j!=0 then go to Step-3,otherwise go to Step-10
Step-3:condition=(Call lastPosition(i,length,)>Call oldPos(number,length,) and Call lastPosition(i,length,)<Call lastPosition(number,length,) and v(Cal lastPosition(i,length,))=number)
Step-4: if condition=0 then go to Step-5, otherwise go to Step-6
Step-5: i=0

Step-6: condition=(Call oldPosition(j,length,)>Call oldPosition(number,length,) and Call oldPosition(j,length,)<Call lastPosition(number,length,) and v(Call oldPosition(j,length,))=number)
Step-7: if condition=0 then got to Step-8, otherwise go to Step-9
Step-8: j=0
Step-9: go to Step-20
Step-10: if i!=0 then go to Step-11,otherwise go to Step-15
Step-11: condition1=Call lastPosition(i,length,)>Call oldPosition(number,length,) and Call lastPosition(i,length,)<Call lastPosition(number,length,) and v(Call lastPosition(i,length,))=number
Step-12: condition2=Call oldPosition(i,length,)>Call oldPos(number,length,) and Call oldPosition(i,length,)<Call lastPosition(number,length,) and v(Call oldPosition(i,length,))=number
Step-13: if condition1=0 and condition=0 then go to Step-14,otherwise go to Step-15
Step-14: i=0
Step-15: if j!=0 then go to Step-16,otherwise go to Step-20
Step-16: condition1=Call lastPosition(j,length,)>Call oldPosition(number,length,) and Call lastPosition(j,length,)<Call lastPosition(number,length,) and v(Call lastPosition(j,length,))=number
Step-17: condition2=Call oldPosition(j,length,)>Call oldPosition(number,length,) and Call oldPosition(j,length,)<Call lastPosition(number,length,) and v(Call oldPosition(j,length,))=number
Step-18: if condition1=0 and condition=0 then go to Step-19,otherwise go to Step-20
Step-19: j=0
Step-20: Return i and j to the calling function

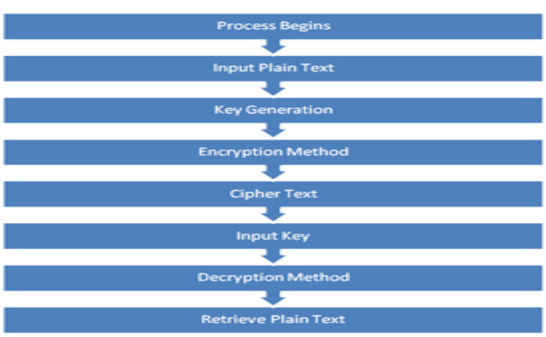


Fig-1: Block Diagram of MWFES Ver-2

3. 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 '1', ASCII '2', ASCII '3' is also given.

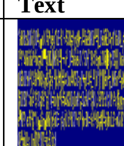
Table-1(a): Encryption of some Plain Texts using some key and skip number

Sl. No.	Plain Text	Key	Skip Number(N)	Encrypted Text
1.	ABCDE	╜ꞇ!θó	1	├⌂\U┬
2.	ABCDF	╜ꞇ!θó	1	─⌂]U─
3.	Ababbab	↓HîJ≈)ºJ		▲¶₧«╖5σ│
4.	Acacacac	↓HîJ≈)ºJ	3	▲↓F▒╖7σ╕
5.	Aabaa	¦=%1Ω	2	╒╫ç¬K
6.	Aacaa	¦=%1Ω	2	¤╫ê¬L
7.	Aacaa	¦=%1Ω	3	3╢êUJ

A. Inference

From the observations made in the table above we see that even for two seemingly similar Plain Texts (as shown in SL. NO. 1. And 2.), the Cipher Texts are drastically different owing to the fact that a change in the last character is rendering the initial backward feedback different, thus making the result completely haphazard even when compared to a Plain Text that is almost similar, even when the keys and the skips are taken to be the same. 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 that would be due to the key being same for both the test cases.

Table-1(b): Encryption of a small paragraph

Plain Text	Cipher Text
The Society of Jesus, a Christian Religious Order founded by Saint Ignatius of Loyola in 1540, has been active in the field of education throughout the world since its origin. In the world, the Jesuits are responsible for 3,897 Educational Institutions in 90 countries. These Jesuit Educational Institutions engage the efforts of approximately 1,34,303 teachers, educating approximately 29,28,806 students.	

B. 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 MSA algorithm we have managed to remove any discrepancies that may have occurred while using the same key for two Plain Texts.

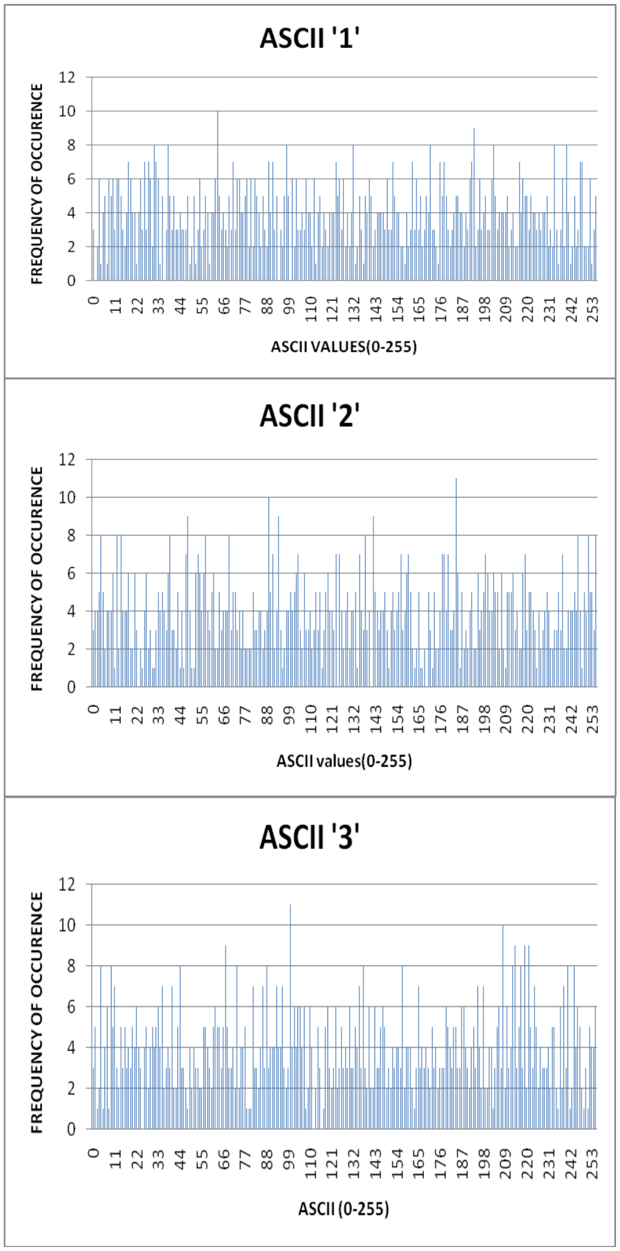
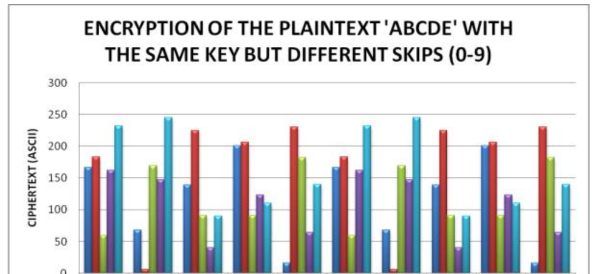


Fig-2: Frequency Spectral Analysis of Encryption of ASCII '1', ASCII '2', ASCII '3'(Top to Bottom)



	0	1	2	3	4	5	6	7	8	9
A	167	68	139	201	16	167	68	139	201	16
B	183	6	225	206	230	183	6	225	206	230
C	60	169	91	91	182	60	169	91	91	182
D	162	147	40	123	64	162	147	40	123	64
E	232	245	90	110	140	232	245	90	110	140

Fig-3: Encryption of Plain Text 'ABCDE' with same key and different skips

C. Inference

In the graph given above we get a pictorial representation of the randomness of the Plain Text characters for just a linear change in the skip. The key(generated by MSA[1] algorithm) is kept constant. We observe that for skips that are less than the length of the Plain Text, the result is completely haphazard, having no known simple relation with each other. However,it is to be noted that when the skip crosses the length of the Plain Text it has the same effect of being the mod of itself with the length of Plain Text thus duplicating the previously found results. In the real world, where the Plain Texts of images and even documents are of significant length and since the one skip is for one time use, this discrepancy will not be a factor during anyone's attempt at crypanalyzing MWFES Ver-II.

4. 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 16x16. The numbers in 16x16 may be arranged in 256! Ways. To complete the whole process the authors have chosen any of the random matrix depending on the user entered text-key. The results show that the set of strings where there is only difference in one character there also the encrypted texts are coming totally different. The present method is free from any kind of brute force attack or known plain text attack. The present MWFES Ver-2 method may be applied to encrypt any short message, password, confidential key. One can apply this method to encrypt data in sensor networks. The overall complexity of the encryption method may be increased even further by introducing random key in different blocks and also the shift may be made random.

Acknowledgment

The authors are very much grateful to the Department of Computer Science for giving the opportunity to work on symmetric key Cryptography. One of the authors (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,Jounal 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, Navajit Maitra, 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, AdvancedCaeser 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-2, 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-2, 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-2, Page-176-182(2013).

Asoke Nath is the Associate Professor in Department of Computer Science. Apart from his teaching assignment he is involved with various research works 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.

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.

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.

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.

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

.