Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 9 (2014) pp. 244-254

Modified Advanced Feedback Encryption Standard Version-1 (MAFES-1)

Surajit Bhowmik[1], Debdeep Basu[2], Ankita Bose[3], Saptarshi Chatterjee[3] and Asoke Nath[5]
[1,2,3,4 and 5]Department of Computer Science
St. Xavier's College (Autonomous)
Kolkata, India
bhowmik1994@gmail.com[1],debdeepbasucal@hotmail.com[2], ankkitabose@hotmail.com[3], sapishere.chatterjee@gmail.com[4], asokejoy1@gmail.com[5]

***Abstract*: Nath et al had recently published Advanced Feedback Encryption Standard Version 1, AFES-1[25] where they had combined both bit-level and byte level operations on the plain text. In AFES-1[25], the authors had capitalized on the strength of MWFES-3[5] by introducing a bit-shuffling operation at the beginning of every iteration. At the beginning of every iteration, the plain text bits of that iteration are shuffled by using 24 different shuffling functions. Now, the order in which the 24 different functions are called, changes at each iteration, and that order is taken as a function of the key. After the initial shuffling of the bits, the bits are converted back to bytes and MWFES-3 is applied on the bytes. This process goes on Encryption Number (EN) times, where EN is also taken as a function of the key. So, at the beginning of each iteration, the bits obtained from the last iteration are shuffled in a different way. In the present paper, Modified Advanced Encryption Standard Version 1 (MAFES-1), the authors have used a different key expansion algorithm which makes the method much stronger than the previous method (AFES-1). The method has been tested on standard plain texts such as ASCII '0', ASCII '1' and the results are quite satisfactory. This method is immune to any classical form of attacks.**

Keywords: MWFES-I,MWFES-2,MWFES-3, AFES-1, Encryption, Encryption Number, Decryption.

I. Introduction

Due to the tremendous development in internet technologies it is essential to encrypt any kind of confidential message before sending the message from one computer to another computer. Data security is an extremely important issue and many algorithms have been developed which are almost impossible to break. The intention of the trespasser is to break the cipher and to retrieve unauthorized information. It is the job of the cryptographers to restrict the trespassers from achieving unauthorized access. Nath et al had recently proposed MWFES-1[1], Modified MWFES-1[2], MWFES-2[3], Modified MWFES-2[4], MWFES-3[5], AFES-1[25].

In MWFES-1, the plain text character is added with the corresponding key character, the forward feedback and backward feedback and then the total sum (modulo 256) is taken as the corresponding cipher text character. The cipher text character is taken as the forward feedback value for the next byte (in case of forward operation) or backward feedback value for the previous byte(in case of backward operation). Forward and Backward operations are carried out on all the bytes starting from their respective ends.

In MWFES-2, the process is a little more general. Instead of propagating the feedback to the next byte (in case of forward feedback) and to the previous byte (in case of backward feedback) the feedback is propagated to the n^{th} byte where n is the 'skip factor'. In MWFES-2 the forward skip is kept equal to the backward skip (equal to n) and the initial forward feedback value and the backward feedback value was kept 0.

In MWFES-3, the authors introduced several changes in the algorithm. The plain text is broken into blocks and the encryption method is applied on each block separately. Each block has different Forward Skip (FS), Backward Skip (BS), initial Forward Feedback (FF) and initial Backward Feedback (BF) which are determined from the keypad counterpart of the block. These four important variables would decide the nature of the cipher text. The block size is different in every round of processing, causing these four important variables to change in every round. The total number of rounds (encryption no), and the block size value were also taken as a function of the key.

In AFES-1[25], the Plain Text is converted to its corresponding bits and stored in a square matrix of size equal to the integral square root of the number of bits. The residual bits remain untouched. Then the bits are arranged by calling 24 different shifting functions. Now, the order of calling the 24 functions change at each iteration and that order is taken as a function of the keypad. After this is done the authors convert the bits back to bytes and then apply MWFES-3[5] on those bytes. This entire process happens encryption_no (EN) times. 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 the generated encrypted text using any brute force method. The results show that the

present method is free from any kind of plain text attack or differential attack.

In the present method, Modified Advanced Feedback Encryption Standard (MAFES-1), the authors have kept the same encryption algorithm as that of AFES-1[25] but have improved the key generation algorithm. This change has improved the security of the method to a huge extent. The key expansion algorithm has been described in section II C. The present method is an extremely strong method as all controlling parameters change at every round. The present method may be applied in any Corporate sector, Defense sector, Government sector etc. The entire encryption and decryption software have been developed using MATLAB.

II. Algorithm For MAFES-1

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

A. Algorithm For Function Encryption()

Step 1: Start
Step 2: Input PlainText,User Provided Seed and CipherText filenames
Step 3: length=length(pt) /*pt is the PlainText*/
Step 4: seed[]=Stores content of seed given by User
Step 5: n=16
Step 6: If n*n<length, the go to Step 7,otherwise go to Step 8
Step 7: n=n+1 and go to Step 6
Step 8: key[]=Call key_generation(seed[],n)
Step 9: encryption_no=key[fix((n*n)/2)]
Step 10: encryption_no=mod(encryption_no,64)
Step 11: if encryption_no=0 then encryption_no=1
Step 12: e=1
Step 13: If e<=encryption_no then go to Step 14 otherwise go to Step 64
Step 14: Initialisesum,ff[length],bf[length],ct[length] with zeros /*ct=CipherText,ff=Forward Feedbacks,bf=Backward Feedbacks*/.
Step 15: Call pt= pt_Shift(key[e]).
Step 16: block_size=key[e]
Step 17: If block_size>length, then go to Step 18, otherwise go to Step 19
Step 18: block_size=block_size-4
Step 19: If block_size<4,thenblock_size=4
Step 20: Initialise k,low and no_of_block with 1
Step 21: high=block_size
Step 22: If k>=block_size, go to Step 23,otherwise go to Step 36
Step 23: k=k-block_size
Step 24: forward_next=mod(key[low]+1,block_size)
Step 25: backward_next=mod(key[high]+1,block_size)
Step 26: If forward_next=0,thenforward_next=1
Step 27: If backward_next=0,thenbackward_next=1
Step 28: forward_feedback=key[low+1]
Step 29: backward_feedback=key[high-1]
Step 30: ff[low]=forward_feedback
Step 31: bf[high]=backward_feedback
Step 32: Call encryption_block(low,high)
Step 33: low=high+1
Step 34: high=high+block_size
Step 35: no_of_block=no_of_block+1 and go to Step 22
Step 36: i=low
Step 37: If i<=length, go to Step 38, otherwise go to Step 40
Step 38: ct[i]=pt[i]
Step 39: i=i+1 and go to Step 37
Step 40: If k>0,go to Step 41,otherwise go to Step 62
Step 41: i=length-k
Step 42: If i>=1,go to Step 43,otherwise go to Step 45
Step 43: ct[i+k]=ct[i]
Step 44: i=i-1 and go to Step 42
Step 45: j=low
Step 46: i=1
Step 47: If i<=k,go to Step 48,otherwise go to Step 51
Step 48: ct[i]=pt[j]
Step 49: j=j+1
Step 50: i=i+1 and go to Step 47
Step 51: pt[]=ct[]
Step 52: forward_next=mod((key[1]+1),256)
Step 53: backward_next=mod((key[block_size]+1),256)
Step 54: If forward_next=0,thenforward_next=1
Step 55: If backward_next=0,thenbackward_next=1
Step 56: forward_feedback=key[2]
Step 57: backward_feedback=key[block_size-1]
Step 58: Initialiseff[length] and bf[length] with zeros
Step 59: ff[1]=forward_feedback
Step 60: bf[block_size]=backward_feedback
Step 61: Call encryption_block(1,block_size)
Step 62: pt[]=ct[] /*Copying converted PlainText into CipherText array */
Step 63: e=e+1 and go to Step 13
Step 64: End

B. Algorithm For Function Encryption_Block (low,high)

Step 1: Initialize i=low
Step 2: sum[i]=pt[i]+key[i]+ff[i]+bf[i]
Step 3: ct[i]=mod(sum[i],256);
Step 4: if i+forward_next>high go to step 5,else go to step 6
Step 5: ff[low+(i+forward_next-high)-1]=ct[i]
Step 6: ff[i+forward_next]=ct[i];
Step 7: index=high-(i-low)
Step 8: sum[index]=pt[index]+key[index]+ff[index]+bf[index]
Step 9: ct[index]=mod(sum(index),256)
Step 10: If index-backward_next<low go to step 11,else go to step 12
Step 11: bf[high-(low-(index-backward_next))+1]=ct[index]
Step 12: Return control to the calling function

C. Algorithm For Function Key_Generation(seed[],n):

From the user defined secret_key (seed), the program will generation enlarged keypad. The requirement of this keypad is that it must be a square matrix having dimensions equal to the nearest greater perfect square of the Plain Text length.

The keypad is an 1-D array at first where we apply 'Fold' to the keypad. The folding concept cleaves the existing key string from the middle and the two halves are made to collapse on one another in order to produce a new set of characters which are appended with to the existing key string. The characters of the keypad are taken modulo 256 and then the keypad is converted into a 2-D matrix. Now, another variable called 'Randomization Number' is calculated from the generated matrix by adding the diagonal values and then bringing it down to 0-255 by modular operation with 256.

The generated 2-D matrix is permuted by calling the 24 shifting and shuffling functions. All of these functions have been discussed in section II R (a). These functions are called "Randomization Number" of times in an order which is to be provided by whoever is implementing the algorithm. Security can be further enhanced by permuting the order in which the functions are called. 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.

1)Key generation method:

Step-1: Step 1: l = length of the seed
Step-2: if l>n*n then go to Step-12
Step-3: i=1
Step-4: j=l
Step-5: if i<j && l<n*n then go to Step-6 otherwise go to Step-11
Step-6: l=l+1;
Step-7: key[l]=mod([key[i]+key[j]],256)
Step-8: i=i+1;
Step-9: j=j-1;
Step-10: Go to Step-5
Step-11: Go to Step-2
Step-12: Copy the keypad into a 2-D array.
Step-13: Add all the diagonal terms and store the sum in 'randomization_number'
Step-14: Call the shifting and shuffling functions according to the order provided by the organization implementing the method 'randomization_number' number of times
Step-15: Convert the 2-D array to a 1-D key string
Return 'key' array to the calling function

D. Algorithm For Function Decryption()

Step 1: Start
Step 2: Input the CipherText,User Provided Seed and decrypted PlainText(Output) filenames.
Step 3: len=length(CipherText)
Step 4: seed[]=User Provided Key
Step 5: n=16
Step 6: Is (n*n)<len?
Step 7: If Step 6=True,then n=n+1 and go to Step 6.
Step 8: If Step 6=False,then go to Step 9
Step 9: key=Call key_generation(seed,n)
Step 10: encryption_no=key[fix((n*n)/2)]
Step 11: e=encryption_no
Step 12: block_size=key[e]
Step 13: Is block_size>len?
Step 14: If Step 12=True,then block_size=block_size-4 and go to Step 13.
Step 15: If Step 12=False,then go to Step 16.
Step 16: Is block_size<4?
Step 17: If Step 16=True,then block_size=4
Step 18: If Step 16=False,then go to Step 20.
Step 19: remainder=mod(len,block_size);
Step 20: initialise the array ct_temp[block_size] with all zeros;
Step 21: initialise the array key_temp[block_size] with all zeros;
Step 22: Is remainder<>0?
Step 23: If Step 22=True, then go to Step 24,else go to Step 44.
Step 24: t=1
Step 25: ct_temp[t]=ct[t]
Step 26: key_temp[t]=key[t]
Step 27: If t>block_size,go to step 28,else t=t+1 and go to Step 25.
Step 28: forward_next=key_temp[1]
Step 29: backward_next=key_temp[block_size]
Step 30: forward_feedback=key_temp[2]
Step 31: backward_feedback=key_temp[block_size-1]
Step32: pt=Call decryption_block(ct_temp,key_tempforward_next,backward_next,forward_feedback,backward_feedback,block_size)
Step 33: j=len-remainder+1
Step 34: i=1
Step 35: pt_main[j]=pt[i]
Step 36: j=j+1
Step 37: If i>remainder,then go to Step 38,else i=i+1 and go to Step 35.
Step 38: i=1
Step 39: ct[i]=pt[i]
Step 40: If i>block_size,then go to Step 41 else i=i+1 go to Step 39.
Step 41: i=remainder+1
Step 42: ct[i-remainder]=ct[i]
Step 43: If i>len,then go to Step 44,else i=i+1 and go to Step 42.
Step 44: begin=1
Step 45: tot_div=floor(len/block_size)
Step 46: i=1
Step 47: set all elements of ct_temp[block_size] by zero
Step 48: set all key_temp[block_size] with all zeros
Step 49: j=1
Step 50: t=begin
Step 51: ct_temp[j]=ct[t]
Step 52: key_temp[j]=key[t]
Step 53: j=j+1
Step 54: If t>begin+block_size-1 then go to Step 55 else t=t+1 , go to Step 51.
Step 55: forward_next=key_temp[1]
Step 56: backward_next=key_temp[block_size]
Step 57: forward_feedback=key_temp[2]

Step 58: backward_feedback=key_temp[block_size-1]
Step59: pt=Call decryption_block(ct_temp,key_temp,forward_next,backward_next,forward_feedback,bacward_feedback,block_size)
Step 60: j=1
Step 61: k=begin
Step 62: pt_main[k]=pt[j]
Step 63: j=j+1
Step 64: If k>begin+block_size-1,then go to Step 65,else k=k+1 and go to Step 62.
Step 65: begin=begin+block_size
Step 66: If i>tot_div,then go to Step 67,else i=i+1 and go to Step 46.
Step 67: ct[]=pt_main[]
Step 68: ct=Call ct_shift(pt_main[],key[e])
Step 69: If e<1,then go to Step 70,else e=e-1 and go to Step 12.
Step 70: Write contents of pt_main into output file.
Step 71: End.

E. Algorithm For Function Decryption_Block(ct[],key[],forward_next,backward_next,ff[],bf[],block_size)

Step-1: forward_next=forward_next+1
Step-2: forward_next=mod(forward_next,block_size)
Step-3: if forward_next=0 then forward_next=1 otherwise go to Step-4
Step-4: backward_next=backward_next+1
Step-5: backward_next=mod(backward_next,block_size)
Step-6: if backward_next=0 then backward_next=1 otherwise go to Step-7
Step-7: (u,v)= Call generateList(block_size,forward_next,backward_next);
Step-8: initialise the array pt[block_size] with all zeros
Step-9: k=2*block_size
Step-10: if k > block_size+1 then go to Step-45 otherwise go to Step-11
Step-11: (i,j)=Call whatIsIn(u[k],block_size,forward_next,backward_next,v[])
Step-12: if i!=j then go to Step-13 otherwise Step-25
Step-13: if i=0 then go to Step-14 otherwise Step-15
Step-14: pos_i=0
Step-15: if v[Call oldPosition(i,block_size)]=u[k] then go to Step-16 otherwise go to Step-17
Step-16: pos_i=Call oldPosition(i,block_size)
Step-17: pos_i=Call lastPosition(i,block_size)
Step-18: if j=0 the go to Step-19 otherwise go to Step-20
Step-19: pos_j=0
Step-20: if v(Call oldPosition(j,block_size))=u[k] then go to Step-21 otherwise go to Step-22
Step-21: pos_j=Call oldPosition(j,block_size)
Step-22: pos_j=Call lastPosition(j,block_size)
Step-23: sub1=Call isChanged(i,pos_i)
Step-24: sub2=Call isChanged(j,pos_j) go to Step-27
Step-25: sub1=Call isChanged(i,oldPosition(i,block_size))
Step-26: sub2=Call isChanged(i,lastPosition(i,block_size))
Step-27: check1=ct[u[k]]-sub1-sub2-key[u[k]]
Step-28: if i=0 and j=0 then go to Step-29 otherwise go to Step-34
Step-29: if u[k]=block_size then go to Step-30 otherwise Step-31
Step-30: check=check1-bf
Step-31: if u[k]=1 then go to Step-32 otherwise go to Step-33
Step-32: check=check1-ff go to Step-39
Step-33: check=check1 go to Step-39
Step-34: if u[k]=block_size and (Call conditionCheck(u[k],i,j,block_size,v)=1) then go to Step-35 otherwise go to Step-36
Step-35: check=check1-bf
Step-36: if u[k]=1 and (Call conditionCheck(u[k],i,j,block_size,v)=2) then go to Step-37 otherwise go to Step-38
Step-37: check=check1-ff
Step-38: check=check1
Step-39: if (check < 0) then go to Step-40 otherwise go to Step-42
Step-40: check=check+256
Step-41: go to Step-39
Step-42: check=mod(check,256)
Step-43: pt[u[k]]=check
Step-44: k=k+1 and go to Step-10
Step-45: return pt[] to the calling function

F. Algorithm For Function Conditioncheck (number,i,j,block_size,v[])

Step 1: if i*j!=0, go to step 2, else 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,block_size)]==number, go to step 5, else go to step 6
Step 5: pos=Call oldPosition(i,block_size)
Step 6: pos=Call lastPosition(i,block_size)
Step 8: if v[Call oldPosition(j,block_size)]==number, go to step 9,else go to step 10
Step 9: pos=Call oldPosition(j,block_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

G. Algorithm For Function Is_Changed (number,position,forward_next,backward_next,block_size,u[],v[],ct[],ff[],bf[])

Step-1: is number==0?
Step-2: if Step 1=TRUE,then go to sub=0 else go to Step 3
Step-3: if position=Call lastPosition(number,block_size) then sub=ct[number] else go to Step 4
Step-4: [i,j]=Call whatIsInBetween(number,length,v,next1,next2)
Step-5: if i <>j then go to step 6, else go to step 24
Step-6: if i=0 then set position_i=0 else go to step 7

Step-7: if v[Call lastPosition(i,block_size)]=number then set position_i=Call lastPosition(i,block_size) else go to Step-8
Step-8: set position_i=Call oldPosition(i,block_size)
Step-9: if j==0 then set position_j=0,else go to step 10
Step-10: if v[Call lastPosition(i,block_size)]=number, set position_j=Call lastPosition(j,block_size),else go to step 11
Step-11: set position_j=Call oldPosition(j,block_size)
Step-12: sub1=ct[number]-isChanged(i,position_i,next1,next2,block_size,u,v,ct,ff,bf)-isChanged(j,position_j,next1,next2,block_size,u,v,ct,ff,bf)
Step-13: [a,b]=CallwhatIsIn(number,block_size,next1,next2,v)
Step-14: if a=0 and b=0 then go to step 15,else go to step 18
Step-15: if number=block_size,set sub=sub1+bf else go to step 16
Step-16: if number=1 then set sub=sub1+ff else go to step 17
Step-17: sub=sub1
Step-18: flag= Call conditionCheck(number,a,b,block_size,v)
Step-19: if number=block_size and position=Call oldPosition(number) and flag<>1 then go to Step 20, else go to Step-21
Step-20: sub=sub1+bf
Step-21: Step 21: if number=1 and position=Call oldPosition(number) and flag!=2 then go to Step 22, else go to step 23
Step-22: sub=sub1+ff
Step-23: sub=sub1
Step-24: sub1=ct[number]-isChanged(i,oldPosition(i),next1,next2,block_size,u,v,ct,ff,bf)-isChanged(i,lastPosition(i),next1,next2,block_size,u,v,ct,ff,bf)
Step-25: [a,b]=whatIsIn(number,block_size,next1,next2,v)
Step-26: if a=0 and b=0 then go to step 27,else go to step 30
Step-27: if number==block_size then set sub=sub1+bf else go to step 28
Step-28: if number==1 then set sub=sub1+ff,else go to step 29
Step-29: sub=sub1
Step-30: flag=Call conditionCheck(number,a,b,block_size,v)
Step-31: if number==block_size and position==Call oldPosition(number,block_size) and flag!=1 go to Step-32, else go to Step-33
Step-32: sub=sub1+bf
Step-33: if number==1 and position==Call oldPosition(number,block_size) and flag<>2 then go to Step 34, else go to step 35
Step-34: sub=sub1+ff
Step-35: sub=sub1
Step-36: Return sub to the calling function

H. Algorithm For Function What_Is_In (number,block_size,forward_next,backward_next,v[])

Step-1: if number+backward_next<=block_size, then go to Step-2,otherwise Step-3
Step-2: i=number+backward_next
Step-3: i=number+backward_next-block_size,
Step-4: if number-forward_next>=1 then go to Step-5,otherwise go to Step-6
Step-5: j=number-forward_next
Step-6: j=number-forward_next+block_size,
Step-7: lastPos_number=Call lastPosition (number, block_size)
Step-8: if(i=j and i!=0) then go to Step-9,otherwise go to Step-13
Step-9: if(Call lastPosition(i,block_size)>lastPos_number) then go to Step-10,otherwise go to Step-11
Step-10: i=0
Step-11: if(Call oldPos(i,block_size)>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 lastPosition(i,block_size)>lastPos_number and v(Call lastPosition(i,block_size))=number) then go to Step-15,otherwise go to Step-16
Step-15: i=0
Step-16: if(Call oldPos(i,block_size,)>lastPos_number and v(Call oldPos(i,block_size,))=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 lastPosition(j,block_size,)>lastPos_number and v(Call lastPosition(j,block_size,))=number) then go to Step-20,otherwise go to Step-21
Step-20: j=0
Step-21: if(Call oldPos(j,block_size)>lastPos_number and v(Call oldPos(j,block_size))=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

I. Algorithm For Function What_Is_In_Between (number,block_size,forward_next,backward_next,v[])

Step-1: (i,j)= Call whatIsIn(number,block_size,,forward_next,backward_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,block_size,)>Call oldPos(number,block_size,) and Call lastPosition(i,block_size,)<Call lastPosition(number,block_size,) and v(Cal lastPosition(i,block_size,))=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,block_size,)>Call oldPosition(number,block_size,) and Call oldPosition(j,block_size,)<Call lastPosition(number,block_size,) and v(Call oldPosition(j,block_size,))=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,block_size,)>Call oldPosition(number,block_size,) and Call lastPosition(i,block_size,)<Call lastPosition(number,block_size,) and v(Call lastPosition(i,block_size,))=number
Step-12: condition2=Call oldPosition(i,block_size,)>Call oldPos(number,block_size,) and Call oldPosition(i,block_size,)<Call lastPosition(number,block_size,) and v(Call oldPosition(i,block_size,))=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,block_size,)>Call oldPosition(number,block_size,) and Call lastPosition(j,block_size,)<Call lastPosition(number,block_size,) and v(Call lastPosition(j,block_size,))=number
Step-17: condition2=Call oldPosition(j,block_size,)>Call oldPosition(number,block_size,) and Call oldPosition(j,block_size,)<Call lastPosition(number,block_size,) and v(Call oldPosition(j,block_size,))=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

J. Algorithm For Function GenerateList(block_size,forward_next,backward_next)
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.

K. Algorithm For Function OldPosition(number,block_size)
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.

L. Algorithm For Function LastPosition(Number,Block_Size)
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*length - 2*(number-1);'
Step 4:- last_pos = 2*(number-1);
Step 5:- Return Control to calling function,and return last_pos to the calling function.

M. Algorithm For Function Pt_Shift(num)

Step 1:- seq=Call generate_sequence(num)
Step 2:-pt_bits=Call convertToBits(pt)
Step 3:-shifted_pt_bits=Bit_Rotation(pt,bits,seq[],0)
Step 4:- shifted_pt_bytes=convertToBytes(shifted_pt_bits)
Step 5:-Return shifted_pt_bytes to calling function.

N.Algorithm For Function Ct_Shift(ct[],num)
Step 1:- seq=Call generate_sequence(num)
Step 2:-ct_bits=Call convertToBits(ct)
Step 3:-shifted_ct_bits=Bit_Rotation(ct,bits,seq[],1)
Step 4:- shifted_ct_bytes=convertToBytes(shifted_pt_bits)
Step 5:-Return shifted_ct_bytes to calling function.

O. Algorithm For Function convertToBits(a[])
Step 1:- k=1
Step 2:- i=1 to length of a[] Step=1 do
Step 3:-j=8 to 1 step=-1 do
Step 4:-aux[k]=Call bitget(a[i],j)
Step 5:=k=k+1;
Step 6:-If j>1 then go to Step 4 else go to Step 7
Step 7:-If i<length of a[] then go to Step 3 else go to Step 8
Step 8:-Return aux[] to the calling function.

P. Algorithm For Function convertToBytes(a[])
Step 1:- k=1
Step 2:- i=1 to (length of a[])/8 Step=1 do
Step 3:-sum=0
Step 4:-j=7 to 0 step=-1 do
Step 5:-sum= sum+a[k]*2^j
Step 6:-k=k+1
Step 7:-If j>0 then go to Step 5 else go to Step 8
Step 8:-b[i]=sum
Step 9:-If i<(length of a[])/8 then go to Step 3 else go to Step 10
Step 10:-Return b[] to the calling function.

Q. Algorithm For Function Generate_Sequence(num)
This function simply generates a sequence array according to the generated keypad in order to make sure that the bits are rotated in a dynamic fashion rather than in the same way every round, which would render the rotation of bits impractical.

R. Algorithm For Function Bit_Rotation(b[],seq[],flag)
Step 1:-len= length of b[] array
Step 2:- n=Integral part of square root of len.

Step 3:-Array a[n][n] is filled row-major wise with the bits in b[] array.
Step 4:-If flag=1(signifying Decryption), the seq[] array is reversed.
Step 5:- The 24 different bit shifting and shuffling functions are called in a sequence given by the seq[] array.
Step 6:- Jumbled bits are copied back into the b[] array.
Step 7:- b[] array is returned to the calling function.

The shifting and shuffling functions are:

1) *Diagonal1_Down_Shift*:- In this function, the major diagonal is shifted one place downwards, the shifting being cyclic.

2) *Diagonal2_Down_Shift*:- In this function, the second diagonal is shifted one place downwards, the shifting being cyclic.

3) *Diagonal1_Up_Shift*:- In this function, the major diagonal is shifted one place upwards, the shifting being cyclic.

4) *Diagonal2_Up_Shift*:- In this function, the second diagonal is shifted one place upwards, the shifting being cyclic.

5) *Exchange_Diagonals_ColumnWise*:- In this function, the two diagonals in the bit matrix are exchanged with each other column wise.

6) *Exchange_Diagonals_RowWise*:- In this function, the two diagonals in the bit matrix are exchanged with each other row wise.

7) *Flip_Diagonal1*:- In this function, the order of the major diagonal elements is reversed.

8) *Flip_Diagonal2*:- In this function, the order of the second diagonal elements is reversed.

9) *Up_Shift_Even*:- In this function, Even rows are shifted upwards by one.

10) *Down_Shift_Even*:- In this function, Even rows are shifted downwards by one.

11) *Up_Shift_Odd:*-In this function, Odd rows are shifted upwards by one.

12) *Down_Shift_Odd:*-In this function, Odd rows are shifted downwards by one.

13) *Exchange_Even_Column:*-In this function, Even columns are exchanged with each other.

14) *Exchange_Odd_Column:*-In this function, Odd columns are exchanged with each other.

15) *Exchange_Even_Row*:- In this function, Even rows are exchanged with each other.

16) *Exchange_Odd_Row*:- In this function, Odd rows are exchanged with each other.

17) *Left_Shift_Even*:- In this function, Even rows are shifted one place to the left.

18) *Left_Shift_Odd*:- In this function,Odd rows are shifted one place to the left.

19) *Right_Shift_Even*:- In this function,Even rows are shifted one place to the right.

20) *Right_Shift_Odd*:- In this function,Odd rows are shifted one place to the right.

21) *Rotate_Even_AntiClockwise*:- In this function, Even interior circles are rotated AntiClockwise.

22) *Rotate_Odd_AntiClockwise*:- In this function,Odd interior circles are rotated AntiClockwise.

23) *Rotate_Even_Clockwise*:- In this function,Even interior circles are rotated Clockwise.

24) *Rotate_Odd_Clockwise*:- In this function,Odd interior circles are rotated Clockwise.

III. Results And Discussions

A. Encryption Of Small Plain Texts With Given Seed

In the table given below, there are many instances where we observe for the same seed, almost similar Plain Texts in SL. NO 1 , 2 and 3, NO. 4 ,5 and 6, NO. 9 and 10, the Cipher Texts are totally haphazard thus rendering the Plain Text irretrievable. Therefore, for slightly bigger Plain Texts the retrieval of the Plain Text becomes almost impossible for any machine as well unless the key, i.e seed is known.

SL. NO.	PLAIN TEXT	SEED	CIPHER TEXT (Ver-1)
1	AAAAAA	Xaviers	_N†ÛH•
2	BAAAAA	Xaviers	Z_Ã_v °
3	CAAAAA	Xaviers	_/Æ ß__
4	ABABAB	Xaviers	Å ³+_U_
5	ACACAC	Xaviers	~Œ'ôï
6	BCBCBCB	Xaviers	,ÖV £À_ø
7	AAABAAA	Xaviers	…”³ 8Ÿµ
8	AAACAAA	Xaviers	‰U6â? 7
9	AABAABA	Xaviers	“Ø¸æF_>
10	AACAACA	Xaviers	’_5¦@_?

Table-I. Small test cases

B. Encryption Of a Paragraph With Given Seed

In the encrypted Cipher Text shown below we get a better example of the efficiency of our algorithm since the size of each block in every iteration of the encryption process plays an important role in completely diffusing the Plain Text into a seemingly random and completely incomprehensible Cipher Text.

Table-II: Encryption of a paragraph with seed “Xaviers”

Plain text	Cipher text
St. Xavier's has always been known for his cosmopolitan and national character. Much before the expression "national integration" gained currency, St. Xavier's had tried to foster among its students the spirit and practice of it. Coming as they do from all over India and from various communities, they live in complete harmony, understanding and mutual respect. Thus they are encouraged to develop beyond local and group affinities, loyalties to the country and the society at large.	M({Æ(ýoÈƒõãÃ®¾7_@ž‹_~Žã Ïíƒ??£_~fõë_Cùº—¨YÎi,_v ?@_v?yWOû_ºÜ¨ÙµO_¨;ŠÆî\b __?FÅž>_^•Jv¶æ%‰'¨là²”Ç– DöÞð¼ük_ŒûJUyÛ_ñ_ø¸Á¼õR ¿zè±ö<pG_³cãwºš_ÆS°üÀ¶LêÉ @u_Y¡²Ì_y®E”«?+úE0_P †«ým-'5 èç_ – ç¢Îêþ¨¹Ü?__¨òuZ-am?(ÒŠÓ!¶1n «V[,~ºO^½Œé})Ÿ¯àŸ!_P,Š,±,ü ùD_¤E5B[_6.?Â_I?À ù \“p¨!ü5bŠ•ãü}~¹fp_‰_?”Ò_÷ùÙ _;/œþ–žj?- @HŽ$,åµE_ÈzÉ,,¹BÄ2_5$I__s ©ÇÂ/E?Øå…—4f¶è> ÊØ s ôA_HýOîÅ_þÃ_?Þ,þ~àécö z_h[»b$ªÍÜ,úçãZ.#åù¡™Ä_«TN _,çp8?À¯ªÙxPF ¼_…uÏ_%>‘ê¿ Š¾¡¡"#(mãñÀÄa{3‹÷%Øn_^;¨gT _Déý§[â‘û³¼É¤·!2°î)H_é“,ø?Ï?ù _œ‡ E èg É üè °_»»(ûP âÂ_ íé_@

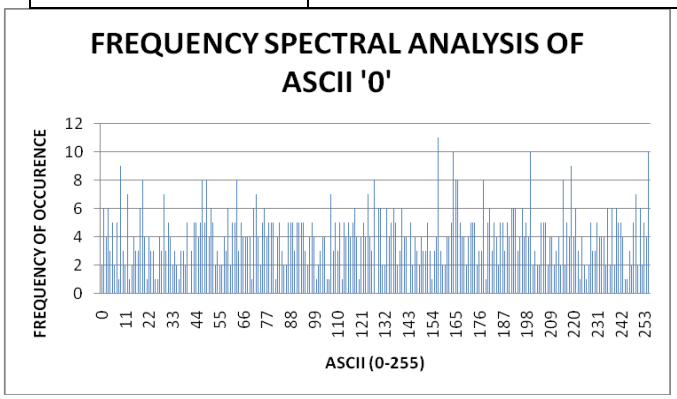

Figure 1. Frequency Spectral Analysis of ASCII ‘0’ for Modified AFES Ver-1

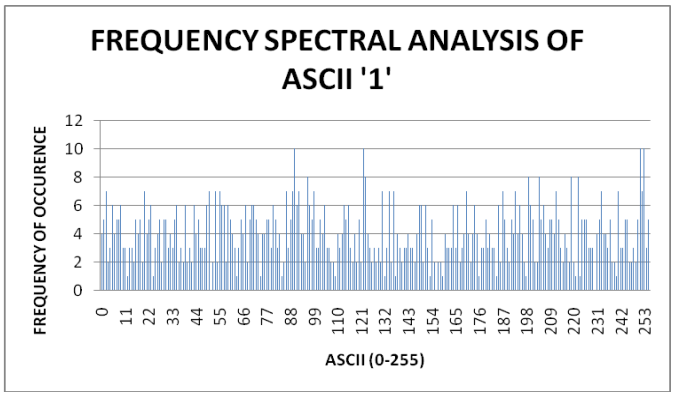

Figure 2. Frequency Spectral Analysis of ASCII ‘1’ for Modified AFES Ver-1

C. Graphs and Frequency Analysis Charts

The graphs given below give us a good estimate of the randomness of the occurrence frequencies of the different ASCII characters. In the ASCII ‘0’, ASCII ‘1’ charts we get varied results even when we encrypt the same character over and over again.

IV. Conclusion And Future Scope

The present method has been tested on various types of files such as .doc, .jpg, .bmp, .exe, .com, .dbf, .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 keypad. The results show that, the set of strings where there is a difference in only one character in the plain text, the encrypted texts are coming totally different from each other. The present method is free from any kind of brute force attack or known plain text attack. The present Modified AFES Ver-1 may be applied to encrypt any short message, password, confidential key and even images and other file types as well. One can apply this method to encrypt data in sensor networks as well.

V. Acknowledgment

The authors are extremely 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 regarding research in the field of cryptography.

VI. References

[1] Purnendu Mukherjee, Prabal Banerjee, AsokeNath, "Multi Way Feedback Encryption Standard Ver-I(MWFES-I)", International Journal of Advanced Computer Research(IJACR), Volume-3, Number-3, Issue-11, September 2013, Pages:176-182.

[2] Prabal Banerjee, Purnendu Mukherjee, AsokeNath, "Modified Multi Way FeedbackEncryptionStandard :Ver-I (MMWFES-I)", International Journal of Advanced Computer Research(IJACR),Vol-3, No.1, Issue-13, Page 352-360, Dec(2013).

[3] AsokeNath, DebdeepBasu, Ankita Bose, SaptarshiChatterjee, SurajitBhowmik,, "Multi Way Feedback Encryption Standard Ver-2(MWFES-2)," International Journal of Advanced Computer Research(IJACR), Vol-3, Number-1, Issue-13, Page-29-35, Dec(2013).

[4] SaptarshiChatterjee, DebdeepBasu, Ankita Bose, SurajitBhowmik, AsokeNath,, "Modified Multi Way Feedback Encryption Standard Ver-2(MMWFES-2)," JGRCS, Vol-4, No. 12, December, 2013, Page 8-13(2013).

[5] AsokeNath,DebdeepBasu, Ankita Bose, SaptarshiChatterjee and SurajitBhowmik "Multi Way Feedback Encryption Standard Ver-3(MWFES-3)," published in IEEE conference proceedings: WICT-2013 held at Hanoi in Dec 14-18(2013), page 318-325(2013).

[6] AsokeNath, Payel Pal, "Modern Encryption Standard Ver-IV(MES-IV)," International Journal of Advanced Computer Research(IJACR), Volume-3, Number-3, Issue-11, September 2013, Page:216-223.

[7] AsokeNath, BidhusundarSamanta, "Modern Encryption Standard Ver-V(MES-V)," International Journal of Advanced Computer Research(IJACR), Volume-3, Number-3, Issue-11, September 2013, Pages:257-264.

[8] AsokeNath,SaimaGhosh,MeheboobAlamMallik,"Symmetric Key Cryptography using Random Key generator" 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).

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

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

[11] NeerajKhanna, JoelJames,JoyshreeNath, SayantanChakraborty, AmlanChakrabarti, AsokeNath, "New Symmetric key Cryptographic algorithm using combined bit manipulation and MSA encryption algorithm: NJJSAA symmetric key algorithm", Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 03-06 June 2011, Page 125-130(2011).

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

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

[14] Trisha Chatterjee, Tamodeep Das, JoyshreeNath, ShayanDey and AsokeNath, "Symmetric key cryptosystem using combined cryptographic algorithms-generalized modified vernam cipher method, MSA method and NJJSAA method: TTJSA algorithm", Proceedings of IEEE International conference: World

Congress WICT-2011 t held at MumbaiUniversity 11-14 Dec, 2011, Page No. 1179-1184(2011).

[15] 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).

[16] Satyaki Roy, NavajitMaitra, JoyshreeNath,ShalabhAgarwal and AsokeNath, "Ultra Encryption Standard(UES) Version-I: Symmetric Key Cryptosystem using generalized modified Vernam Cipher method, Permutation method and Columnar Transposition method", 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).

[17] SomdipDey, JoyshreeNath, AsokeNath, "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, (IJMECS), ISSN: 2075-0161 (Print), ISSN: 2075-017X (Online), Vol-4, No-5, Page 1-9,2012.

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

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

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

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

[22] Prabal Banerjee, AsokeNath, "Advanced Symmetric Key cryptosystem using Bit andByte Level encryption methods with Feedback", Proceedings of International conference Worldcomp 2013 held at Las Vegas, July 2013

[23] Ankita Bose, DebdeepBasu, SaptarshiChatterjee, AsokeNath, SurajitBhowmik, "Bit Level Multi Way Feedback Encryption Standard Version-1(BLMWFES-1)", Paper presented at IEEE conference proceedings CSNT 2014 held at Bhopal on 7-9th April, 2014.

[24] AsokeNath, SurajitBhowmik, DebdeepBasu, Ankita Bose, SaptarshiChatterjee, "Bit Level Multi Way Feedback Encryption Standard Version-1(BLMWFES-2)", Paper presented in IEEE conference proceedings ICACCCT2014, held at Ramanathapuramon 8-10th May, 2014.

[25] DebdeepBasu, Ankita Bose, SurajitBhowmik, SaptarshiChatterjee, AsokeNath, "Advanced Feedback Encryption Standard Version – 1 (AFES-1)", Paper accepted for publication in IEEE conference International Conference on Advances in Computing, Communications and Informatics (ICACCI-2014)to be held at GCET, Greater Noida, Delhi, India during September 24-27, 2014.

Short BiodataOf All The Authors

[1]**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.

[2]**Debdeep Basu** after passing her B.Sc. in Computer Science from St. Xavier's College(Autonomous) now is pursuing his M.Sc. in Computer Science from Benaras Hindu University, India. He was born in Kolkata on 03.08.1993. He is presently involved in research work in Cryptography.

[3]**Ankita Bose** after passing her B.Sc. in Computer Science from St. Xavier's College(Autonomous) now is pursuing her M.Sc. Computer Science from Benaras Hindu University. She was born in Kolkata on 15.02.1993. She is presently involved in research work in Cryptography.

[4]**Saptarshi Chatterjee** after passing her B.Sc. in Computer Science from St. Xavier's College(Autonomous) now is pursuing his M.Sc Computer Science at St.Xavier's,College(Autonomous),Kolkata. He was born in Kolkata on 17.04.1993. He is presently involved in research work in Cryptography.

[5]**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, Big Data handling. He has presented papers and invited talk, keynote address and tutorials in different International and National conferences in India and in US, Vietnam etc.