B. MWFES-II

Nath et al developed MWFES II which also encrypts plaintext using two feedback strings and keys. Encryption process starts from left hand side and proceed towards right hand side. Then encryption process starts from left hand side and proceed towards right hand side. Encryption from LHS to RHS is done by adding the ASCII value of the plaintext, forward feedback, backward feedback and key. If the resulting intermediate value exceed 256, modulo 256 operation performed on it. Then this modified intermediate cipher text is placed on the n^{th} position from current position. Then algorithm encrypt from RHS to LHS. ASCII value of plaintext, backward feedback, forward feedback and key are added together to generate intermediate cipher text. If this result exceeds 256, then modulo 256 operation takes place. After this, the intermediated cipher text is placed on the n^{th} position towards left. This "n^{th} place" moving from both LHS and RHS, contrasts with the version I. Control is passed between LHS and RHS alternately. This algorithm also expands its initial key. Forward feedback and backward feedback are initially set to 0. During execution, initial value changes to different values. Thus it hides the original feedback values. These two feedback strings acts as two additional key.

C. MWFES-III

In Multi way Feedback Encryption Standard III, the authors divided the plaintext into blocks. It uses keypad which is generated using the key expansion algorithm. In each iteration block size also changes. In each iteration, the block size, forward feedback, backward feedback, and the skip factor n for each block is taken as a function of the keypad. So, after every iteration, these four variables changes. The total number of the encryption iteration is also takes as the function of the user given seed and referred as encryption number. After completion of a single round, residual characters (if any) are moved to the beginning of the encrypted text and the same encryption process is applied. Authors showed that results completely hide the frequency. This is a novel method because the skip factor is being changed in a random number for every iteration. Thus multi way feedback encryption standard 3 becomes very hard to decrypt. Retrieval of plaintext from the encrypted message becomes impossible by any known attack method. The encryption process is non-linear and thus it is impossible to apply any brute force attack, differential attack on this present method.

III. METHOD USED TO GENERATE FEEDBACK

Before applying encryption method the plain text and the key are divided into square matrices. The number of layer is dependent on the length of the plain text. The length of the plain text and the length of the key must be equal. The feedback arrays are created which are of same size as of the plain text and the key. The feedback arrays are initialized by zero (0). Suppose for example we take a plain text: 111111111111111111 and also take a random no (say 11111) as our key. The string array will store the ASCII value of the corresponding plain text that the use gives as input. The key generation procedure will expand the key according to the length of the plain text. The length of the key and the plain text will be of same size. Then after dividing the plain text and the key according to our method it looks like,

TABLE I
STRARR (STRING ARRAY LAYER 0):

49	49	49
49	49	49
49	49	49

TABLE II
STRARR (STRING ARRAY LAYER 1):

49	49	49
49	49	49
49	49	49

After the expansion of the key the key array will look like,

TABLE III
KEYARR (KEY ARRAY LAYER 0):

137	107	167
47	32	62
2	122	137

TABLE IV
KEYARR (KEY ARRAY LAYER 1):

107	167	47
32	62	2
122	137	107

The forward and the backward feedback arrays are initialized by zero. According to the method the feedback arrays will be modified. The forward and the backward feedback are applied row wise in each square matrix simultaneously. The forward feedback starts from the first cell of the first square matrix of the lowest layer and the values stored in the same cell of the key array and feedback arrays are added and the resultant value is stored in the next cell of the forward feedback array. Similarly the backward feedback starts from the last cell of the first row of each square matrix the values stored in the same cell of key array and feedback arrays are added and stored in the previous cell of the backward feedback array. The string array and the key array will remain same. After the first iteration the modified feedback arrays will be,

TABLE V
IFXFARR (LAYER 0):

0	186	46
6	0	0
0	0	0

TABLE VI
IFXFARR (LAYER 1):

0	0	0
0	0	0
0	0	0

TABLE VII
IBXFARR (LAYER 0):

46	216	0
0	0	232
0	0	0

TABLE VIII
IBXFARR (LAYER 1):

0	0	0
0	0	0
0	0	0

The forward feedback of the last cell of the row will passed to the next row. Similarly the backward feedback of the first cell of the row will passed to the next row. After completion of feedback in first row the matrices will be look like,

TABLE IX
IFXFARR (LAYER 0):

160	186	46
6	102	14
101	152	113

TABLE X
IFXFARR (LAYER 1):

159	59	124
229	54	143
121	36	63

TABLE XI
IBXFARR (LAYER 0):

46	216	99
14	87	232
113	46	116

TABLE XII
IBXFARR (LAYER 1):

124	105	9
143	234	183
63	97	197

The main key point of this encryption method is that the feedback value of the last cell of the matrix of last layer is stored in the first cell from where feedback started. Therefore all the characters of the plain text is encrypted. That gives the encryption a different complexity.

The plain text array or string array (strarr) will be modified as:
strarr[k][i][j]=(strarr[k][i][j]+keyarr[k][i][j] +ifxfarr[k][i][j] + ibxfarr[k][i][j]). The matrix will look like this after modification:

TABLE XIII
STRARR (LAYER 0):

136	46	105
116	14	101
9	113	159

TABLE XIV
STRARR (LAYER 1):

183	124	229
197	143	121
99	63	160

After the completion of row wise encryption the column wise encryption is performed on the modified string array. For this we take the transpose of the string array and apply the row wise encryption on it. The feedback arrays corresponding to the column wise encryption is initialized by zero. Before the column wise encryption is applied the string array will look like,

TABLE XV.
STRARR (STRING ARRAY LAYER 0):

136	116	9
46	14	113
105	101	159

TABLE XVI
STRARR (STRING ARRAY LAYER 1):

183	197	99
124	143	63
229	121	160

The forward and the backward feedback of arrays corresponding to column wise encryption will be generated according to the method used above. After generation of the feedback arrays the final feedback arrays will be,

TABLE XVII
IFYFARR (FORWARD Y-AXIS FEEDBACK ARRAY LAYER 0):

107	17	160
80	173	59
155	6	245

TABLE XVIII
IFYFARR (FORWARD Y-AXIS FEEDBACK ARRAY LAYER 1):

5	39	32
173	73	158
38	133	121

TABLE XIX
IBYFARR (BACKWARD Y-AXIS FEEDBACK ARRAY LAYER 0):

160	176	254
59	96	177
245	16	232

TABLE XX
IBYFARR (BACKWARD Y-AXIS FEEDBACK ARRAY LAYER 1):

32	141	251
158	136	71
121	242	231

The plain text array or string array (strarr) will be modified as:
strarr[k][i][j]=(strarr[k][i][j]+keyarr[k][i][j]+ifyfarr[k][i][j] + ibyfarr[k][i][j]).
After getting the modified cipher text we have to re-transpose the string array to get the original order of the plain text. The modified string array will look like,

TABLE XXI
STRARR (LAYER 0):

28	232	251
160	59	245
78	155	5

TABLE XXII
STRARR (LAYER 1):

71	231	254
32	158	121
173	38	107

Before applying layer wise encryption method we have to transpose the string array layer wise and then we encrypt the string array row wise. The matrix that will be encrypted layer wise will look like,

TABLE XXIII
STRARR (LAYER 0):

28	71	232
231	251	254
160	32	59

TABLE XXIV
STRARR (LAYER 1):

158	245	121
78	173	155
38	5	107

This matrix is constructed by picking up values layer wise and storing them row wise. Therefore the arrays are in such order that we can apply row wise encryption on it. The key array will remain same. The forward and backward array of layer wise feedback is initialized by 0. And we apply the row wise feedback mechanism to generate the feedback matrices after the feedback matrices are generated they will look like,

TABLE XXV
IFZFARR (FORWARD Z-AXIS FEEDBACK ARRAY LAYER 0):

58	165	230
117	139	109
52	214	44

TABLE XXVI
IFZFARR (FORWARD Z-AXIS FEEDBACK ARRAY LAYER 1):

232	241	55
225	79	255
196	100	22

TABLE XXVII
IBZFARR (BACKWARD Z-AXIS FEEDBACK ARRAY LAYER 0):

230	143	122
109	199	139
44	188	248

TABLE XVIII
IBZFARR (BACKWARD Z-AXIS FEEDBACK ARRAY LAYER 1):

55	170	2
255	197	40
22	36	78

The final cipher array will be computed by the following formula
strarr[k][i][j]=(strarr[k][i][j]+keyarr[k][i][j] +ifzfarr[k][i][j] + ibzfarr[k][i][j]).
After the string array is modified we have to re-transpose it to get the original cipher text in proper order.

The final cipher text will be,

TABLE XXIX
STRARR (LAYER 0):

28	71	232
231	251	254
160	32	59

TABLE XXX
STRARR (LAYER 1):

158	245	121
78	173	155
38	5	107

If there is any character left in the plain text then we will put those in the front of the string array and fill up the rest of the arrays by taking the values from the previously modified array and store them sequentially. This modified string will again encrypted along three dimensions. Now the whole plain text is stored in a array. The complexity of the encryption method increased as it can be seen that the first and the last part of the modified array is encrypted once whereas the middle part is encrypted twice.

IV. ALGORITHM FOR 3DMWFES

A. *CONDITIONS TO FIND THE OPTIMAL ARRAY DIMENSIONS*

1) The number of columns and rows in the 2-D array must be the same.
2) The number of layers i.e. the number of 2-D arrays to be created must never exceed the dimension of the array itself except for when the length of the string is 2 or 3.

Example: For string length is 2 or 3 the array dimensions are {1,1,2} and {1,1,3}.
Else the dimension must always be of the form {a, a, b} where b<=a.

3)If length of the string is 1 or 8 it is changed to 2 or 9 by adding an extra ‘space’ at the end of the string.

B. *ALGORITHM FOR ENCRYPTION*

Step 1: Take the plain text as input from user and load it into strarr array.

Step 2: Take the key as input from user. The key expansion algorithm will expand the key according to the string length and load it into the keyarr array.

Step 3: Continue steps 4 to 6 until all the values in the cells of the ifxfarr and ibxfarr are modified. Alternately update ifxfarr and ibxfarr, after calculation of each cell of each matrix.

Step 4: Calculate the sum of the values in cell of strarr, keyarr, ifxfarr and ibxfarr and store the value in the next cell (column wise) of the forward feedback array (ifxfarr).The process starts from the first cell of the array.

Step 5: After each cell calculation of ifxfarr calculate the value of ibxfarr in the following way. Calculate the sum of strarr, keyarr, ifxfarr and ibxfarr and store it to the previous cell of ibxfarr starting from the last column of the first row.

Step 6: After each row calculation, the calculation of next row begins. The last backward and forward feedback values of each row are passed to the last and first column of next row. The same is true for all the next layers.

Step 7: The last forward and backward value is passed to starting cell of ifxfarr and ibxfarr i.e. the first cell of ifxfarr and the last cell of first row of ibxfarr.

Step 8: Calculate and modify all the values of string array in the following way: strarr[k][i][j]= strarr[k][i][j]+ keyarr[k][i][j]+ifxfarr[k][i][j]+ ibxfarr[k][i][j].

Step 9: Transpose the string array and store it into strarr.

Step 10: Repeat steps 3 to 7 and store the corresponding feedback values in ifyfarr (forward y-axis feedback array) and ibyfarr (backward y-axis feedback array).

Step 11: Calculate values of all cells of modified strarr as: strarr[k][i][j]=strarr[k][i][j]+ keyarr[k][i][j]+ ifyfarr[k][i][j]+ ibyfarr[k][i][j];

Step 12: Re-transpose the modified strarr to get the original order of the string array.

Step 13: Transpose the modified strarr layer wise. This matrix is constructed by picking up values layer wise and storing them row wise. Therefore the arrays are in such order that we can apply row wise encryption on it.

Step 14: Repeat steps 3 to 7 and store the corresponding feedback values in ifzfarr (forward z-axis feedback array) and ibzfarr (backward z-axis feedback array).

Step 15: Calculate values of all cells of modified strarr as: strarr[k][i][j]=strarr[k][i][j]+ keyarr[k][i][j]+ ifzfarr[k][i][j]+ ibzfarr[k][i][j].

Step 16: Re-transpose the modified strarr layer wise to get the original order.

Step 17: If there are any residual characters in the input string while allocating it to the strarr then add the residual characters to the beginning cells of the strarr and fill the rest of the matrix with cipher array characters. If there are no residual characters re-encrypt the modified strarr again.

Step 18: Encrypt the characters using the steps 3 to 15.

Step 19: After encryption rearrange the characters in correct order to get the final cipher.

Step 20: Repeat the steps from 1 to 19 for (string_length % 5 + 1) number of times to get the final encrypted string.

P.S:- If the input is of length 1 and 8 then an extra space is added to the plain text.

C. ALGORITHM FOR DECRYPTION

Step 1: Compute the dimensions of the array based on the length of the cipher text.

Step 2: Find the characters that might have been left over if the cipher text was copied to the array.

Step 3: Copy the residual characters into the array first and then fill rest of the array with characters starting from the first character of the cipher text. Do the same to the key array as well.

Step 4: The first task is the z-axis decryption. The whole array is transposed layer wise into another array of same dimensions i.e. in case of 2 layers the 1st character in layer 1 becomes the first character of transposed array, the 1st character of layer 2 becomes the 2nd character of transposed array, the 2nd character of layer 1 is the 3rd character of transposed array and so on.

Step 5: The decryption is done by following a pattern among the cipher array and the forward and backward axis arrays. The forward and backward axis arrays must be of the same dimensions as that of the cipher array. First calculate the position of the middle column of the arrays.

Step 6: If the current cell of the cipher array being traversed is greater than or equal to the middle column position but less than or equal to the last column position copy the values of the current cell to the next cell of the forward axis array.

Step 7: If current cell position is less than or equal to the middle column position but greater than first column position then subtract current cell value from the previous cell value and store it in the current cell position of the forward axis array.

Step 8: If the current cell of the cipher array being traversed is less than or equal to the middle column position but greater than or equal to the first column position copy the values of the current cell to the next cell of the backward axis array.

Step 9: If current cell position is greater than or equal to the middle column position but less than or equal to the last column position then subtract current cell value from the next cell value and store it in the current cell position of the backward axis array.

Step 10: Except for the positions (0, 0, 1), (0, 0, c1-2), (0, 1, 0), (0, 1, c1-1) of the cipher array, where c1 is the total no. of columns, the key, forward feedback and backward feedback values are subtracted from the cipher to get the required value.

Step 11: After the values have been found the left out values in the feedback arrays are found by adding the forward, backward, key and cipher values of the previous positions as was calculated during encryption and stored in the array strarr[][][]. The array is then re-transposed to get the original order.

Step 12: Now y-axis decryption is to be done. The strarr[][][] array is transposed first.

Step 13: Then steps 5 to 11 are to be repeated to get the decrypted text and is again stored in the strarr[][][] array. The array is then re-transposed layer wise to get the original order.

Step 14: Now the x-axis decryption is to be done.

Step 15: The steps 5 to 11 are to be repeated to get the decrypted text and is again stored in the strarr[][][] array.

Step 16: After the decryption has been done the decrypted string is reordered in its correct form and concatenated with the previously left out cipher text characters.

Step 17: Copy the maximum number of characters possible into the array from the newly created string starting from the beginning of the string.

Step 18: Repeat the steps from 4 to 16 to get the decrypted string.

Step 19: Repeat the steps from 1 to 18 for (string_length % 5 + 1) number of times to get the final decrypted string.

V. RESULTS AND DISCUSSIONS

The plain text is taken as following shown in the table. The key we have taken is constant in all the cases i.e. 1234. The encrypted value is shown over here. The corresponding number of matrix operations and the execution time is also shown in the table.

TABLE XXXI

Plain Text	Encrypted Text	Number of Matrix Operations	Time taken
12345678	Òñe†g¹S2	540	24ms
12349678	VeÉù>+Ág¦	540	10ms
abcdefghij	Ñ9Úë%å›Ž	108	7ms
1	«ž	72	4ms

The encryption was also applied on other files like executable file, image, docx files etc. The result was found to be quite satisfactory. A screenshot of an image file has been provided as an example:

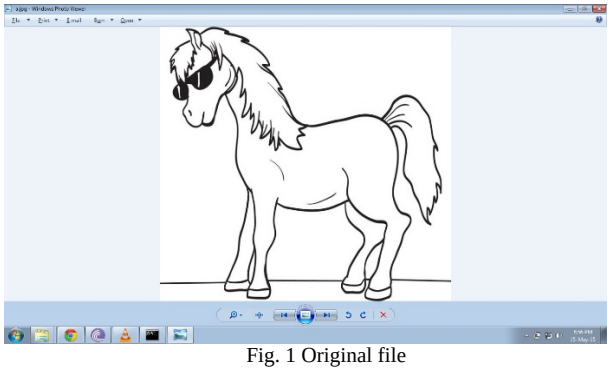

Fig. 1 Original file

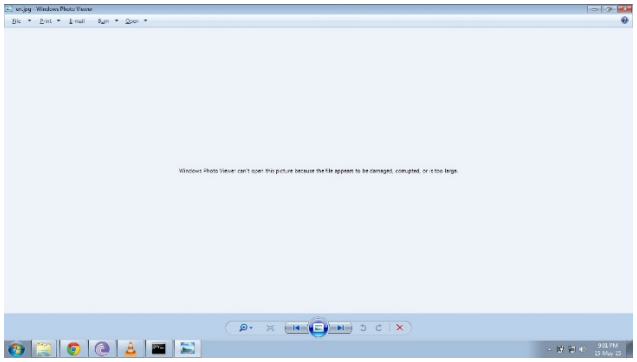

Fig. 2 Encrypted file

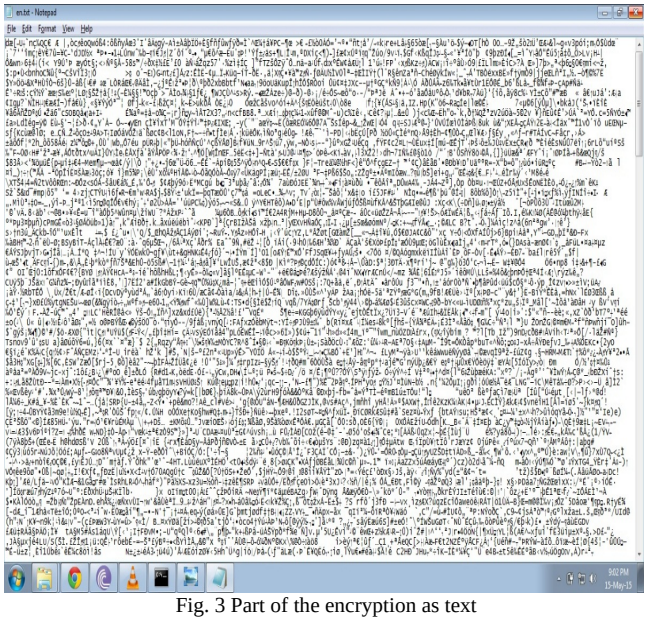

Fig. 3 Part of the encryption as text

Fig. 4 Decrypted file

The execution time and lines of code responsible for encryption was calculated and the graph was derived for each accordingly. The graphs are shown as below:

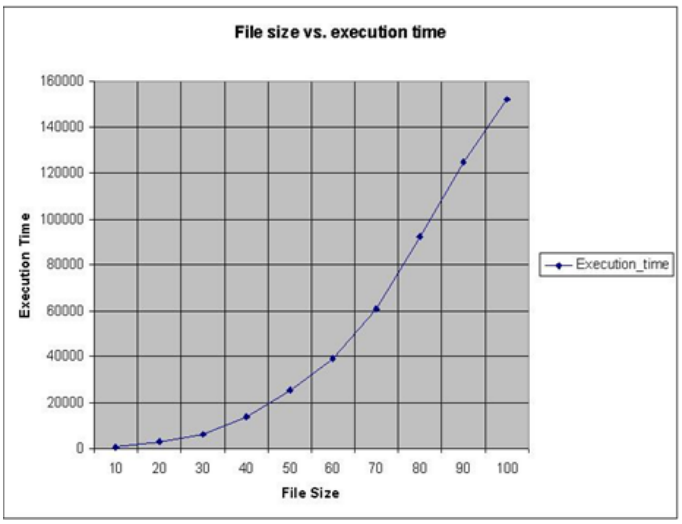

Fig. 5 File size vs. Execution Time graph

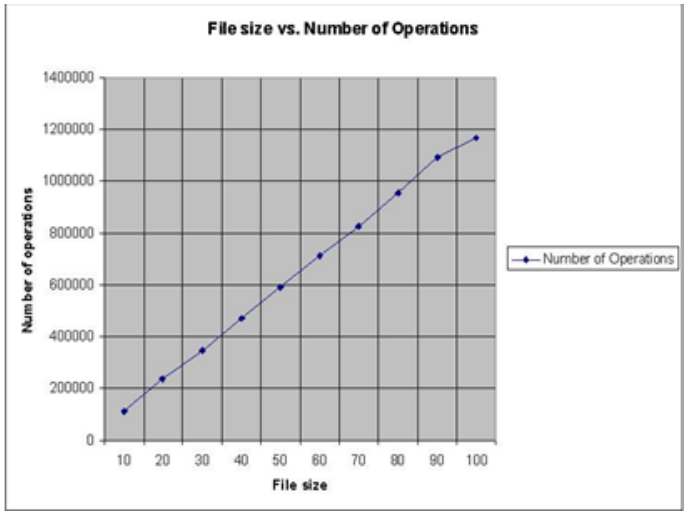

Fig. 6 File size vs. Number of Operations graph

VI. MERITS OF 3DMWFES METHOD

1) Invulnerable to traditional attacks like brute force or statistical attack will not be able to break the present method.
2) A single change in the plain text or key leads to a vast change in the cipher text which is almost chaotic in nature.
3) Due to the addition of an initial matrix creation process, the attacker needs to find out what the dimension of the matrix is. Without that any attack is bound to fail.
4) Even if the string and the key consists of same characters, the cipher text generated has no relation between its characters nor is there any problem of frequency.
5) It has been seen that even for a stream of characters with ASCII value '1' the cipher text generated is much complex.

VII. CONCLUSION AND FUTURE SCOPE

1) 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. The algorithm has the complexity of $O(n^3)$. However, for encryption of large files it takes large time.
2) This algorithm can be further developed to create more complex cipher text. One way can be to have a random selection of axes which can be decided by a sub function. The order of transition from one axis to another cannot be then easily tracked hence adding more chaos to the cipher text.
3) This algorithm has ample scopes for development. The algorithm can be modified to reduce its time complexity to a great extent without compromising with its quality.
4) The feedback arrays can be initialized with a random number instead of initializing it with some constant value. So, interpreting the plaintext values become more difficult.
5) Also the key generation algorithm can be modified so that for each iteration of the encryption or decryption function a new key is generated.

REFERENCES

[1] Purnendu Mukherjee, Prabal Banerjee, Asoke Nath, 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] Asoke Nath, Debdeep Basu, Surajit Bhowmik, Ankita Bose and Saptarshi Chatterjee, Multi Way Feedback Encryption Standard Ver-2(MWFES-2),. Paper submitted in International Conference : IEEE WICT 2013 to be held in December 15-18, 2013 at Hanoi, Vietnam.

[3] Asoke Nath, 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.

[4] Asoke Nath, Bidhusundar Samanta, 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.

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

[6] Prabal Banerjee, Asoke Nath, Advanced Symmetric Key Cryptosystem using bit and byte level encryption methods with feedback: Proceedings of International conference Worldcomp 2013 held at LasVegas, July 2013.

[7] Arijit Ghosh, Prabhakar Chakraborty, Asoke Nath, Shamindra Parui, "3d Multi Way Feedback Encryption Standard Version 1(3DMWFES-1)". International Journal of Advance Research in Computer Science and Management Studies, ISSN:2321-7782(online), Vol 2, Issue 10, Oct, Page:206-218(2014)