Ultra Encryption Standard (UES) Version-IV: New Symmetric Key Cryptosystem with bit-level columnar Transposition and Reshuffling of Bits

Satyaki Roy
Department of Computer Science,
St. Xavier's College
(Autonomous), Kolkata, India

Shalabh Agarwal
Department of Computer Science
St. Xavier's College
(Autonomous), Kolkata, India

Navajit Maitra
Department of Computer Science
St. Xavier's College
(Autonomous), Kolkata, India

Joyshree Nath
A.K. Chaudhuri School of IT,
Raja Bazar Science College,
Calcutta University, Kolkata,
India

Asoke Nath
Department of Computer Science
St. Xavier's College
(Autonomous), Kolkata, India

ABSTRACT

The present paper is an extension of the previous work of the authors i.e. UES version-II and III. Roy et al recently developed few efficient encryption methods such as UES version-I, Modified UES-I, UES version-II, UES version-III. Nath et al developed some methods such as TTJSA, TTSJA and DJMNA which are most suitable methods to encrypt password or any small message. The introduction of multiple feedbacks in TTJSA and TTSJA it was found that the methods were free from any brute force attack or differential attack or simple plain text attack. The authors proposed the present method i.e. Ultra Encryption Standard Version-IV. It is a Symmetric key Cryptosystem which includes multiple encryption, bit-wise reshuffling method and bit-wise columnar transposition method. In the present work the authors have performed the encryption process at the bit-level to achieve greater strength of encryption. In the result section the spectral analysis is done on repeated characters. The authors proposed method i.e. UES-IV can be used to encrypt short message, password or any confidential key.

General Terms

UES-I, UES-II, randomization, bit-wise, feedback, password, shift

1. INTRODUCTION

The problem of data security is now manifold due to tremendous progress in internet technology. To break a password nowadays is not at all a very difficult task. The educational sectors, banking sectors, government sectors, IT industries, share market, medical sectors all are computerized. So the most important issue is to protect confidential data from any kind of intrusion. Especially in the Banking sectors hacking of data means the system will simply collapse. There are many websites developed by the hackers where one can get all those software which can help to break any kind of weak password. It means only user id and password are not enough to protect any confidential data. Suppose an intruder intercepts the confidential data of a company and sells it to a rival company, then it will be a big damage for the company from where the data has been intercepted. The confidential data must be always stored in encrypted form.

To encrypt data or to transform data from readable format to unreadable format we use two types of cryptography algorithms (i) Symmetric key cryptography where we use single key for encryption and decryption purpose. (ii) Public key cryptography where we use one key for encryption purpose and one key for decryption purpose. Both the methods have their advantages as well as disadvantages. Nath et al. had developed some symmetric key algorithm [1-8]. Most of the methods operate in byte level. In the present paper, the authors have attempted to take encryption one step further by introducing bit-level encryption. The authors have already attempted bit-level encryption in UES III. The algorithm provides the combined strength of bit-level reshuffling and a Bit-wise columnar transposition method. We have tested this method on various types of known text files and we have found that, even if there is repetition in the input file, the encrypted file contains no repetition of patterns. Undoubtedly, it provides stronger encryption than the byte-level encryption method attempted so far.

2. ALGORITHM- UES IV

The UES Version- IV algorithm comprises of two distinct methods (i) Bit-level Encryption Technique with Columnar Transposition method which the author used in UES-I in byte level, (ii) Bit-level reshuffling method. Now we will describe in detail UES IV algorithm.

UES IV Encryption Algorithm

The algorithm integrates bit-level columnar transposition and bit wise reshuffling. It computes 'cod' which controls the multiple encryption number and 'v' which is the columnar sequence generator. It splits the plain files into bits, encrypts it and then converts it back to bits.

Step-1: Input plain text file, cipher text file and password which may be maximum 64-byte long

Step-2: Calculate n=sizeOf (plain file).

Step-3: Calculate cod=key[i] * (i+1) where i=position of character, 0<=i<=n.

Step-4: Calculate cod=cod%50 and if cod<15 then cod=15. Cod= Number of times the encryption process is repeated.

Step-5: Decompose the entire plain file into its constituent bits and define z=0.

Step-6: Define 1-d array arr [] = {2, 8, 32,128} where the elements of 'arr'= number of bytes of plain text extracted and encrypted at a time in each iteration.

Step-7: If z>=4 Goto17.

Step-8: If arr[z]>n then break else calculate no=n/arr[z] and define i=0.

Step-9: Extract arr[z] bytes of plain text and if i>=n then goto step-15

Step 10: Define u=square_root (arr[z]*8), (arr[z]*8) = number of bits encrypted at once.

Step 11: Define v=0, where v= counter variable to perform multiple encryption.

Step-12: If v>=cod Goto step-15.

Step-13: Call function ran_en (u, i) where the key matrix will be randomized i-times.

Step-14: Call function colum (v) where the value v=the columnar sequence generator for every iteration. Increment v. Goto step-12

Step-15: Calculate i=i+1, Goto step-9

Step-15: Write the encrypted bits in the output file together with the unrandomized residual bits and convert into respective bytes of encrypted text.

Step-16: Calculate

Step-17: Goto step-7

Step-17: End

UES IV Decryption Algorithm

Step-1: Input plain text file, cipher text file, password (maximum size is 64-byte).

Step-2: Calculate n=sizeOf (plain text file).

Step-3: Calculate cod=key [i] * (i+1) where i=position of character, 0<=i<=n.

Step-4: Calculate cod=cod%50 and if cod<15 then cod=15. cod= Number of times the decryption process is repeated.

Step-5: Decompose the entire plain file into its constituent bits and define z=0.

Step-6: Define 1-d array arr [] = {2, 8, 32,128} where the elements of 'arr'= number of bytes of plain text extracted and encrypted at a time in each iteration.

Step-7: If z>=4 Goto17.

Step-8: If arr[z]>n then z=z+1 and continue, else calculate no=n/arr[z] and define i=0.

Step-9: Extract arr[z] bytes of plain text and if i>=n then goto step-15

Step 10: Define u=square root (arr[z]*8), (arr[z]*8) =number of bits encrypted at once.

Step 11: Define v=0, where v= counter variable to perform multiple encryption.

Step-12: If v>=cod Goto step-15.

Step-13: Call function colum (v, cod) where the value v=the columnar sequence generator for every iteration.

Step-14: Call function ran_de (u,i) where the key matrix will be randomized i-times. Increment v. Goto step-12

Step-15: Perform i=i+1, Goto step-9

Step-15: Write the encrypted bits in the output file together with the unrandomized residual bits and convert into respective bytes of decrypted text.

Step-16: calculate z=z+1

Step-17: Go to step-7.

Step-18: End

2.1Bit Level Columnar Transposition Algorithm

This module generates the random sequence of column extraction in every iteration. It extracts the plain bits according to the sequence set by an array.

Encryption column (v, cod)

Step 1: Start

Step 2: The columnar transposition is performed on the bits of plain file. The value cod=unique key generated from user password, v=the columnar transposition column generator. The initial value of arra[]={7,6,5,4,3,2,1,0} and subsequently becomes arra[i]=(arra[i]+v)%8 where 0<i<8. Therefore the value of arra which stores the sequence of column extraction, changes in every iteration.

Step 3: Initialize the variable n to an arbitrary value which represents the number of columns of the columnar transposition array in which the plain file bits will be stored. Typically n may have any value.

Step 4: Initialize both variables row=0 and col=0

Step 5: Initialize all the elements in the specified array arr [] [] to NULL('\0')

Step 6: Store the plain text file byte by byte in the array arr [] [] where the row and column positions are determined by 'row' and 'col'.

Step 7: Perform col=col+1 once a byte is read and placed in the array

Step 8: If col=n then row=row+1, and initialize col=0 to keep a check on the row and column parameters.

Step 9: Goto 7 until the storing of the intermediate plain text in the array is complete.

Step 10: If col==0 then we decrement the row index by 1 to ensure that the character array arr[] does not produce an extra row. This happens when a bit is placed at the last column of a particular row.

Step 11: Initialize both variables count=index=0.

Step 12: If (count>=n) Goto 17

Step 13: count=count+1

Table-I (a): Plain Text bits

Plain text (ASSUME): aa = 0110000101100001
The plain text is placed in array 'arr'.
arra [i]=(arra[i]+v)%8
p=arra [index]
Where index=0 and subsequently index=index+1
p=6, 7, 5, 4, 3, 1, 2, 0 (ASSUME) respectively where p stands for the sequence of extracted column.

0	1	1	0	0	0	0	1
0	1	1	0	0	0	0	1

Table-I (b): Cipher Text bits: After bit-wise columnar transposition Encryption

0	1	0	0	0	1	1	0
0	1	0	0	0	1	1	0

Step 14: Perform p= arra [index]. Here the 'arra []' stores the order in which the columns will be transported to the same columnar transposition array arr [] to implement the columnar transposition encryption method. The variable 'index' is subsequently incremented to transport the rest of the columns of the columnar transposition array.

Step 15: Write the bits in the p-th column in the intermediate encrypted output file.

Step 16: Goto 12.

Step 17: Once the control flows outside the loop. The encryption process is complete.

Step 18: End.

Decryption column (v, cod)

Step 1: Start

Step 2: The algorithm de-randomizes the cipher bits. Cod = the unique key calculated from user password.

Step 3: Initialize the value v=columnar transposition column generator and array num[]={7,6,5,4,3,2,1,0} which stores the order in which the columns will be decrypted to get the decrypted file. Then arra[i]=(arra[i]+(cod-v-1))%8 where 0<i<8. Therefore the value of arra which stores the sequence of column extraction, changes in every iteration. The algorithm states that the order of the columns in the columnar transposition method must follow the same sequence.

Step 4: Initialize three variables row=col=count=0.

Step 5: Initialize all the elements in the array arr[][] to NULL('\0')

Step 6: Read the encrypted file byte by byte and increment count by 1.Count will give us the number of characters that needs to be decrypted.

Step 7: Compute no=count/8. The algorithm (one such case) has used 8 columns to encrypt the plain text and hence divide count by 8 to get the number of rows in which the decrypted text has to be stored in the array arr [][].

Step 8: If count%8! = 0 then an extra row is produced which will accommodate the extra characters. Therefore the variable 'no' is incremented.

Step 9: Initialized i to ' 0'

Step 10: If i>=8 then Goto step 17.

Step 11: Compute p=num[i] and initialize k to 0, where k will control the iterations within variable 'no' that has been calculated.

Step 12: If 'k' is greater than or equal to 'no' Goto 16

Step 13: Extract the characters from the encrypted file and store in character 'ch'

Step 14: Save the character in arr[k][p].

Step 15: Increment k.

Step 16: Goto 13.

Step 17: Increment i.

Step 18: Goto 10.

Step 19: Once the control flows out of the while loop, the array 'arr [] []' holds the decrypted file that is ready to be written into the intermediate output file.

Step 20: Write the characters from the array arr [][] in the output file.

Step 21: End

2.2Bit- level Reshufflings Algorithm

The algorithm randomizes the key matrix based on the value of the multiple encryption number (cod). It performs randomization of the plain bits based on the arrangement of numbers in the key matrix.

Encryption rand_en (m, in)

Step 1: Start

Step 2: Create a key matrix which is used to randomize the bits of plain text where *m=number of rows / columns in the square matrix of plain bits, in=number of times the key matrix is randomized*.

Step 3: Define 2-d arrays arr=the randomization key. Define 2-d bits arrays chararr [] [] =plains bits and chararr2 [] [] =randomized bits.

Step 4: Initialize all the elements in the bits arrays chararr [][] and chararr2[][] to 'null'.

Step 5: 'm'=number of rows and columns in the square matrix of chararr [] [], chararr2 [] [], arr [] [].

Step 6: Input the numbers 1, 2, 3..., (m*8) to the array arr [] [] by incrementing the value of n.

Step 7: Copy the input file bits to 2-d array chararr [] [].

Step 8: The program invokes function 'leftshift ()' which shifts every column in the array to one place left thus the leftmost column goes to the extreme right.

Step 9: Invoke function top shift () which shifts very row to the row above. Therefore the elements in first row are displaced to the corresponding position of the last row.

Step 10: Subsequently perform cycling operation on the array arr [] []. Initialize i to 1.

Step 11: If i > m/2 Goto 15.

Step 12: If i is odd, perform clockwise cycling of the ith cycle of the key matrix array. Invoke functions :

rights(),downs(), lefts(),tops() to implement the clockwise displacement of the elements in arr[][].

Step 13: If i is even, perform anti-clockwise cycling of the i-th cycle of the bits array. Invoke functions *ac_rights (), ac_downs (), ac_lefts (), ac_tops ()* to implement the anti-clockwise displacement of the elements in arr [] []. Therefore the array arr [] [] is alternately randomized in clockwise and anti-clockwise cycles. Repeat step 13 'in' number of times.

Step 14: Increment i. Goto 11.

Step 15: The program invokes function '*rightshift ()*' which shifts every column in the array to one place right thus the last column is displaced to the position of the first column.

Step 16: Invoke function '*downshift ()* which shifts very row to the row below. Therefore the elements last row are displaced in the corresponding position of the first row.

Step 17: Invoke the function '*leftdiagonal ()*' that performs downshift on the elements in the left diagonal such that the lowermost element is displaced to the position of the topmost element in the left diagonal.

Step 18: Invoke the function '*rightdiagonal ()*' that performs downshift on the elements in the right diagonal such that the lowermost element is displaced to the position of the topmost element in the right diagonal.

Step 19: To arrange the elements in the bits array chararr [] [] according to the randomized array arr [] []. Initialize i to 1.

Step 19: Initialize j to 1

Step 20: Store element arr[i] [j] in z.

Step 21: Compute the k=row position=z/m and l=column position=modulus (z, m) pointed by the element z

Step 22: Place chararr[k][l] in auxiliary bits array

chararr2 [][] in positions chararr2[i][j].

Step 23: Increment j.

Step 24: If j<=m Goto 20

Step 25: Increment i

Step 26: If j<=m Goto 20

Step 27: Write the randomized elements in bits array chararr2 [i] [j] to the output file.

Step 28: End.

Decryption rand_de (m, in)

Step 1: Start

Step 2: Create a key matrix which is used to randomize the bits of plain text where m=number of rows / columns in the square matrix of plain bits, in=number of times the key matrix is randomized.

Step 3: Define 2-d array 'arr' = randomized key. Define 2-d bits arrays 'chararr [][]' = bits in encrypted file and chararr2[][] = decrypted bits.

Step 4: Initialize all the elements in the bits arrays chararr [][] and chararr2 [][] to 'null'.

Step 5: 'm' = number of rows and columns in the square matrix of chararr [] [], chararr2 [] [], arr [] [].

Step 6: Input the numbers 1, 2, 3..., (m*8) to the array arr [] [] by incrementing the value of n. The bits in the input file are copied to the bits array 'chararr [][]'.

Step 7: Use the numbers in the randomized array created with the help of the functions subsequently defined in the program to obtain key matrix.

Step 8: The program invokes function *'leftshift ()*' which shifts every column in the array to one place left.

Step 9: Invoke function *'topshift ()* which shifts every row to the row above.

Step 10: Perform cycling operation on the array 'arr []' . Initialize i to 1.

Step 11: If i > m/2 goto 15.

Step 12: If i is odd, perform clockwise cycling of the i-th cycle of the bits array. Invoke functions *rights (), downs (), lefts (), tops ()* to implement the clockwise displacement of the elements in arr [] [].

Step 13: If i is even, perform anti-clockwise cycling of the ith cycle of the bits array. Invoke functions :

ac_rights (), ac_downs (), ac_lefts (), ac_tops () to implement the anti-clockwise displacement of the elements in arr[][]. Therefore the array arr [] [] is alternately randomized in clockwise and anti-clockwise cycles. Repeat step-13 'in' times.

Step 14: Increment i. Goto 11.

Step 15: Invoke function *'rightshift ()'* which shifts every column in the array to one place right.

Step 16: Invoke function 'downshift () which shifts very row to the row below.

Step 17: Invoke the function 'leftdiagonal ()' that performs downshift on the elements in the left diagonal.

Step 18: Invoke the function 'rightdiagonal ()' that performs downshift on the elements in the right diagonal.

Step 19: Store decrypted bits in auxiliary array chararr [] [].Initialize i to 1.

Step 20: Initialize j to 1

Step 21: Initialize variables flag to 0, k to 0 and l to 0 where k=row index and l=column index for array chararr [] [].

Step 22: if arr[k] [l] is not equal to n goto 24

Step 23: chararr2 [i][j] assumes the value in chararr[k][l], flag=1 and BREAK.

Step 24: If 'flag' is equal to 1 break

Step 25: Increment l.

Step 26: If l is less than or equal to m goto 22.

Step 27: Increment k

Step 28: If k is less than or equal to m goto 22.

Step 29: Increment n.

Step 30. Increment j.

Step 31. If j is less than or equal to m goto 21.

Step 32. Increment i

Step 33: If i is less than or equal to m goto 22.

Step 34: Write the decrypted elements in the bits array chararr2 [] [] in the output file.

Step 35: End

2.3Diagrammatic Representation of Ultra Encryption Standard (UES) Version IV: New Symmetric key Cryptosystem with bit-level columnar transposition and reshuffling of bits

The above diagram shows the working of the UES IV algorithm. It initially extracts 2 bytes at a time and performs bitwise reshuffling and columnar transposition of the extracted data. It then extracts the next 2 bytes and performs the same process until the entire file is encrypted or the number of residual bytes is less than the number of extracted bytes. It then repeats the same procedure by extracting 8, 32 and 128 bytes of plain text bytes at a time.

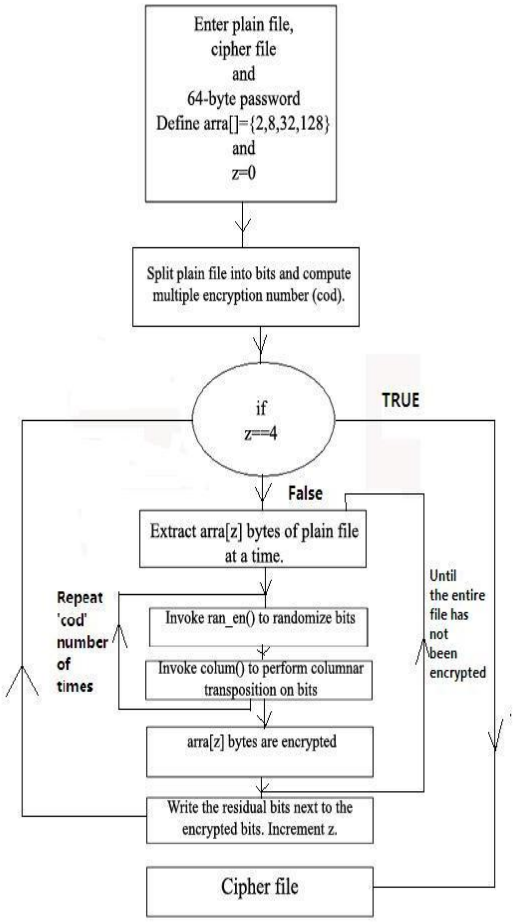

Figure 1: Diagrammatic Representation of UES IV.

3.TEST RESULTS

In the present paper, the authors have combined the two modules of bit-level randomization and Advanced Bit-wise Encryption Technique with columnar transposition. The test results have been recorded with great care to ensure that the algorithm not only works for every file format but also yields satisfactory test results for all possible file sizes. The algorithm works at the bit-level and the test results show that the quality and strength of encryption obtained is significantly higher than the techniques that work with bytes. The test results include *(i) the change in the cipher text when applied on the same plain text but with different passwords and (ii) Frequency analysis of some rare test cases. (iii) Byte by byte comparison between the ciphers of text file containing only characters 'a', 'b', 'c' respectively.* The following results effectively demonstrate the quality of encryption rendered by UES-IV.

(i) Table-II: The change in the cipher text when applied on the same plain text but with different password

Plain Text	Password	Cipher File
Ultra Encryption Standard IV is a bit-level encryption technique.	1	…ë«3í__□È_Yàü±_ä€j æ,½_éž'D®U‹•²fÃª£Ù H'$&y_†}ïmÙ!©à¥`Ê_ DÏujz_ö1è
Ultra Encryption Standard IV is a bit-level encryption technique.	11	p±L•µH_{I□"__Å0+à¼ V_ÕV>õ…`_ó,ºË•)›)_É üÄ_\Z¸DØy+é_[W\øÐ' Ê ô@¤g]_
Ultra Encryption Standard IV is a bit-level encryption technique.	111	hð°ÏXÅ__}#ô*Âp8‹ia_ 4□!l+_¦_¤Œlh.__pù±_Ír S}T^,"¯¨º,«Œh_@³û3T¦ AÚ

The above table studies the effectiveness of the user password by applying different passwords on the same plain text input and cataloguing the variation in the resultant cipher files.

Frequency Analysis of rare text inputs

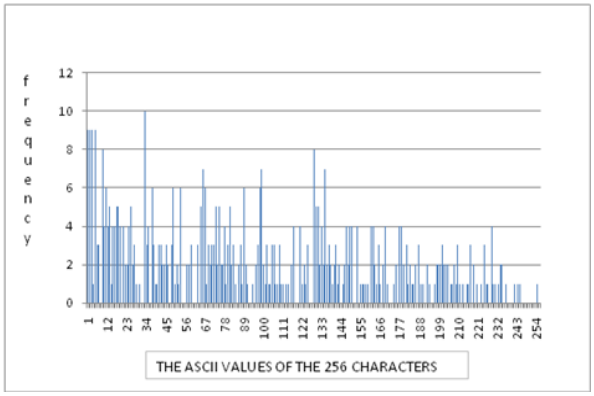

Figure II: Spectrum Analysis for cipher file for 256 characters of ASCII 'a'

Generally plain inputs like 512 bytes of 'a' or 256 bytes of ASCII value 2 are hard to encrypt. However frequency analysis reveals that this algorithm can encrypt such plain file inputs.

The figure above (RESULT-I) represents the frequency of each character in the cipher file corresponding the plain text which is **512 characters of a**. The bit-configuration of the character is therefore 01100001. However, due to the effectiveness of bit-level randomization and columnar transposition, it has been possible to encrypt such a plain file.

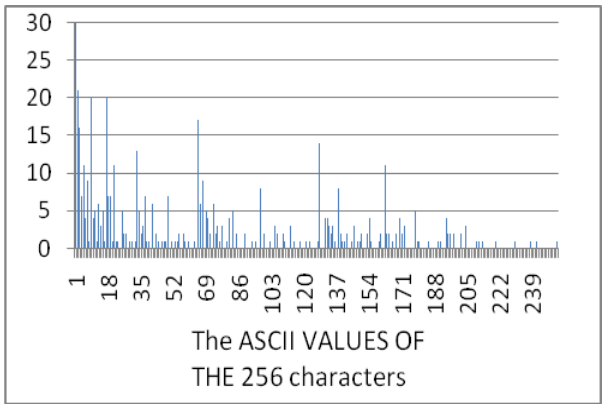

Fig-III: RESULT – II: 256 characters of ASCII '2', the graph representing the frequency of each character (of ASCII 0-255) in the cipher file

The graph (RESULT II) corresponds to 256 characters of ASCII value '2' of bit sequence 00000010. Since majority of bits in the plain file is 0 therefore the encryption of such a file is equally difficult without feedback. However it is only possible due to repeating transposition and randomization operations.

Byte by byte comparison between the ciphers of text files 20 containing only characters 'a', 'b', 'c'

Byte Number	ASCII value of character in cipher files for 40 characters of 'A'.	ASCII value of character in cipher file for 40 characters of 'B'.	ASCII value of character in cipher file for 40 characters of 'C'.
1	165	32	160
2	169	0	34
3	132	18	22
4	0	169	173
5	0	32	32
6	28	64	64
7	65	0	50
8	7	92	124
9	128	20	157
10	132	157	136
11	65	128	163

12	36	163	36
13	32	32	42
14	74	32	58
15	50	26	7
16	36	7	97
17	10	97	44
18	147	44	184
19	64	152	2
20	44	2	201

Table - III: Byte by byte comparison of similar text inputs. (In this case 20 characters of 'A', 'B' and 'C'.

The objective of this test is to verify whether the cipher file for similar inputs (in this case, **40 bytes of 'A', 'B' and 'C'** are similar. The results in columns 2,3 and 4 of the table verify that almost no row has the same character for the three plain text files.

4. CONCLUSION AND FUTURE SCOPE

In the previous endeavours of UES Version-I, UES Modified Version-I and UES Version-II, the authors have worked exclusively on bytes. In the present work the entire encryption process is performed at the bit-level. The plain text files have been split into respective bits before we apply the aforementioned algorithms. From the test results shown before, it is evident that the algorithm takes care of plain text inputs such as ASCII 2 and many occurrences of same character. Even when the same characters are provided as input, the cipher files have almost no occurrence of repetitive patterns.The columnar transposition module with bits has been utilized for the first time. The use of multiple encryption and the role of the password provided by the user have also been demonstrated in the test results.

The results show that this method is too hard to break by using any kind of brute force method. As mentioned before have applied our method on some known text where the single character repeats itself for a number of times and we have found that after encryption there is no repetition of pattern in the output file. Moreover, it must be remembered, if the cipher file is tampered and certain character(s) in the file get altered, it would be impossible to retrieve the plain file, since the feedback generated will be different for different characters. The present method will not work if the plain text file contains all ASCII character 255 or ASCII character 0.

5. ACKNOWLEDGEMENT

We are grateful to the Department of Computer Science for giving us the unique opportunity to work on Symmetric Key Cryptography. One of the authors (AN) sincerely expresses his gratitude to Fr. Dr. Felix Raj and Fr. Jimmy Keepuram for allowing us to carry out research work. AN is thankful to the University Grant Commission for their support and financial assistance. JN is grateful to A.K. Chaudhuri School of IT and SR, NM, SA and AN are thankful to St. Xavier's College.

6. REFERENCES

[1] Symmetric Key Cryptography using Random Key generator: Asoke Nath, Saima Ghosh, Meheboob Alam Mallik: ŖProceedings of International conference on security and management (SAMř10ŗ held at Las Vegas, USA Jull 12-15, 2010), P-Vol-2, 239-244 (2010).

[2] A new Symmetric key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm, Dripto Chatterjee, Joyshree Nath, Suvadeep Dasgupta and Asoke Nath : Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 3-5 June,2011, Page-89-94.

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

[4] Advanced Symmetric key Cryptography using extended MSA method: DJSSA symmetric key algorithm: Dripto Chatterjee, Joyshree Nath, Soumitra Mondal, Suvadeep Dasgupta and Asoke Nath, Jounal of Computing, Vol3, issue-2, Page 66-71,Feb(2011).

[5] Advanced Steganography Algorithm using encrypted secret message : Joyshree Nath and Asoke Nath, International Journal of Advanced Computer Science and Applications, Vol-2, No-3, Page-19-24, March(2011).

[6] Symmetric key Cryptography using modified DJSSA symmetric key algorithm, Dripto Chatterjee, Joyshree Nath, Sankar Das, Shalabh Agarwal and Asoke nath, Proceedings of International conference Worldcomp 2011 held at Las Vegas, USA, July 18-21, Page 312-318, Vol-I(2011).

[7] Cryptography and Network, Willian Stallings, Prentice Hall of India.

[8] Cryptography & Network Security, B.A.Forouzan, Tata Mcgraw Hill Book Company.

[9] An Integrated symmetric key cryptography algorithm using generalized vernam cipher method and DJSA method: DJMNA symmetric key algorithm, Debanjan Das, Joyshree Nath, Megholova Mukherjee, Neha Chaudhury and Asoke Nath, Proceedings of IEEE conference WICT-2011 held at Mumbai University Dec 11-14,2011

[10] Symmetric key cryptosystem using combined cryptographic algorithms-generalized modified Vernam cipher method, MSA method and NJJSAA method: TTJSA algorithm, Trisha Chatterjee, Tamodeep Das, Joyshree Nath, Shyan dey and asoke Nath, Proceedings of IEEE conference WICT-2011 held at Mumbai University Dec 11-14, 2011.

[11] Ultra Encryption Standard (UES) Version-II: Symmetric Key Cryptosystem using generalized modified Vernam Cipher method, Permutation method, Columnar Transposition method and TTJSA Method, Satyaki Roy, Navajit Maitra, Shalabh Agarwal and Asoke Nath,

Proceedings of the 2012 International Conference on Foundation of Computer Science, held at Las Vegas, July 14-19, Page 97-104.

[12] Ultra Encryption Standard (UES) Version-III: Symmetric Key Cryptosystem With Bit-level Encryption Algorithm, Satyaki Roy, Navajit Maitra, Shalabh Agarwal, Joyshree Nath, Asoke Nath, International Journal of Modern Education and Computer Science (IJMECS), Volume 4 Number 7, July 2012.

[13] Ultra Encryption Algorithm (UEA): Bit level Symmetric key Cryptosystem with randomized bits and feedback mechanism International Journal of Computer Application (IJCA) Volume 49 number 5, Satyaki Roy, Navajit Maitra, Shalabh Agarwal, Joyshree Nath, Asoke Nath, July, 2012

.