I.J. Information Technology and Computer Science, **2013, 05, 67-73**
Published Online April 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2013.05.09

The Extensive Bit-level Encryption System (EBES)

Satyaki Roy
Department of Computer Science, St. Xavier's College, Kolkata, India
E-mail: unrivaledsatyaki@gmail.com

***Abstract*—** In the present work, the Extensive Bit-level Encryption System (EBES), a bit-level encryption mechanism has been introduced. It is a symmetric key cryptographic technique that combines advanced randomization of bits and serial bitwise feedback generation modules. After repeated testing with a variety of test inputs, frequency analysis, it would be safe to conclude that the algorithm is free from standard cryptographic attacks. It can effectively encrypt short messages and passwords.

***Index Terms*—** Randomization, Feedback, Byte-Extraction

I. Introduction

Due to the growing need to protect the confidentiality of information, there is a rising demand for an encryption algorithm that can protect data of every format, size and type. Protecting passwords from interception and unauthorized intrusion is of utmost importance. Some messages have repeated occurrences of the same characters and encryption of such texts may be rather difficult. Crypto analysis is fast becoming an integral part of cryptographic algorithms because measures must be taken to ensure that any file may be suitably encrypted.

Cryptography algorithms are largely of two types (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.

The present algorithm EBES is a symmetric key algorithm that performs encryption by advanced bitwise randomization and serial feedback generation. The prime objective is to ensure that even the rarest of text inputs like characters of ASCII 0, 1 and 2 may be encrypted to ciphers that are difficult to intercept and decode by standard cryptographic attack mechanisms.

II. The Extensive Bit-level Encryption System (EBES) Algorithm

The EBES algorithm includes a number of modules that may be largely classified under two algorithms (i) Bit-wise Randomization (ii) Serial bitwise feedback generation module. The first algorithm uses employs permutation of the plain text bits and the second applies new serial bit feedback generation to encrypt the randomized bytes. The two modules are managed with the help of integration modules for encryption and decryption respectively.

2.1 Integration Module

The features of the module are described below:

1. The integration module converts the plain bytes into bits.

2. It extracts bits of plain text depending on the size of the file. According to the algorithm 2, 8, 32 and 128 bytes may be extracted and encrypted at a time (described later).

3. The randomization module and the serial feedback generation modules are also invoked.

4. The plain bits are encrypted multiple times according to the user password.

Encryption

1. Enter the name of the Plain file, Cipher file and key=password (maximum size=64 bytes).
2. Define ma=n=Size of (Plain file).
3. Define cod=Σ key[i]* (i+1) where 0<=i<=64 and 1-d array arr [] = {2, 8, 32, 128} where arr[i] {0<=i<4} decides the number of bytes extracted at a time for encryption.
4. Perform cod=number of times encryption is repeated=modulus (cod, 30). If cod<10 then perform cod=10.
5. Split the plain file into bits. Define ii=0.
6. If ii>cod Goto 16

Copyright © 2013 MECS

7. Perform n=ma, ll=1, i=3
8. If n < arr[i] Goto 13
9. Extract (arr[i]*8) bits of plain text. Define m=square root (arr[i]*8)
10. Invoke function ran_en (m, ll) to perform bit-level randomization on the plain bits
11. Invoke module feed_en () to generate serial feedback to encrypt the randomized bits.
12. Perform n=n-arr[i] where n=unprocessed bytes. Perform ll=ll+1. Goto 14
13. Perform i = i – 1.
14. If n<=1 then copy the file to the cipher file else Goto 8.
15. Write the cipher bits back to the plain bit file, perform ii=ii+1 Goto 6 for another round of encryption.
16. Convert the encrypted bits back to bytes to obtain the final cipher file.
17. End

Decryption

The decryption process is almost the same as the encryption module. However the only point of difference is that the modules ran_en () and feed () modules are invoked in the opposite order during decryption.

Illustration of byte-extraction from the plain file:

1. If the size of the plain file is 43 bytes (ASSUME) then extraction of 128 bytes is not possible.

2. The algorithm extracts 32 bytes and encrypts it and writes it in the cipher file.

3. Therefore number of bytes remaining = 45-32=11.

4. Therefore extraction of 32 bytes is no longer possible. The algorithm extracts 8 bytes and encrypts it.

5. Number of bytes remaining=11-8=3.

6. Now, the algorithm extracts 2 bytes and encrypts it.

7. Number of bytes remaining=3-2=1. It copies the remaining byte into the cipher file.

2.2 Advanced Bit-wise Randomization Module

The features of this module are described below:

1. This module generates the key matrix for randomization of plain bits based on the value of'll' which counts the number of times the key matrix is randomized before the actual bit exchange.
2. The plain bits are randomized according to the key matrix.
3. The algorithm also performs selective compliment of the plain bits based on the key matrix entries so that the rare text inputs containing characters like ASCII 0 or 1 only may be randomized. So some plain bits of 0 become 1 and vice-versa.

Encryption ***rand_en (p, ll)***

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*, *ll=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.

Step 14: Increment i. Goto 11.

Copyright © 2013 MECS

Step 15: Repeat steps 11-14‘ll’ number of times. 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 in the 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]. If modulus (j,2) is not equal to 0 then compliment the bit stored in chararr2[k][l].

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.

Decryptio ***rand_de (m, ll)***

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, ll=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.

Step 14: Increment i. Goto 11.

Step 15: Repeat steps 11-14 ‘ll’ times. 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 the cipher bits in the 2d array chararr [][]. Define i=1

Step 20: Define j=1

Step 21: Define z=arr[i][j]

Step 22: Define k=z/m, l=modulus (z, m)

Step 23: If l is not equal to 0, k=k+1 else l=m.

Step 24: Compliment the bit stored in chararr[k][l].

Step 25: Perform j=j+2. If j<=m, Goto 21.

Step 26: Increment i. If i<=m, Goto 20

Copyright © 2013 MECS

Step 27: .Define n=1. Initialize i to 1.

Step 28: Initialize j to 1

Step 29: Initialize variables flag to 0, k to 0 and l to 0 where k=row index and l=column index for array chararr [] [].

Step 30: if arr[k] [l] is not equal to n Goto 32

Step 31: chararr2 [i][j] assumes the value in chararr[k][l], flag=1 and BREAK.

Step 32: If 'flag' is equal to 1 break

Step 33: Increment l.

Step 34: If l is less than or equal to m goto 30.

Step 35: Increment k

Step 36: If k is less than or equal to m goto 30.

Step 37: Increment n.

Step 38. Increment j.

Step 39. If j is less than or equal to m goto 29.

Step 40. Increment i

Step 41: If i is less than or equal to m goto 29.

Step 42: Write the decrypted elements in the bits array chararr2 [] [] in the output file.

Step 43: End

2.3 Serial Feedback Generation Module

The features of this module are described below:

A. This algorithm stores a starting feedback of 0.

B. It extracts plain bits and generates simple serial feedback (shown in the table-I below) by performing simple OR operation between the current feedback and plain bit.

C. The current cipher bit becomes the feedback for the next bit.

Encryption ***feed_en ()***

1. Enter the name of the file containing the plain bits.
2. Define character ch1 = Starting value of feedback=ASCII 48 (ASCII for character 0)
3. Define character ch2=1 extracted bit of plain text.
4. Perform ch1=ch1+ch2-96 to generate the serial bit feedback. Character ch1can have values 0 or 1 i.e. ASCII 48 or 49.
5. Write the encrypted bit ch1 into the cipher file.
6. Goto 3 until the entire plain text bits is processed.
7. End

Table 1: Serial feedback generation

Initial feedback=0
Plain bits: 1010
Cipher bits: 1100

Plain text	1	0	1	0
Feedback	0	1	1	0
Cipher Text	1	1	0	0

Decryption ***feed_de ()***

1. Enter the name of the file containing the cipher bits.
2. Define character ch1=1 extracted bit of cipher file.
3. Define character ch2=another extracted bit of cipher file.
4. Perform ch1=ch1+ch2-96 where ch1=decrypted bit. The variable ch1 may have values 0 or 1 i.e. ASCII 48 or 49.
5. Write the character ch1 into the cipher file.
6. Perform ch2=ch1. Goto 3 until the entire cipher file is processed.
7. End

III. The Working of EBES

The EBES algorithm computes n which is the size of the plain text. It defines an array arr [] = {2, 8, 32, 128} and variable i=3. If the value of n is greater than equal to arr [i] extract arr [i] bytes and perform the bit-wise randomization and serial feedback generation encryption on the arr [i] bytes. Perform n = n – arr [i]. Write the encrypted bytes in the cipher file. *Repeat till entire file is encrypted or 1 byte is remaining.*

Repeat the process 'enc' times where enc is the multiple encryption number.

Copyright © 2013 MECS

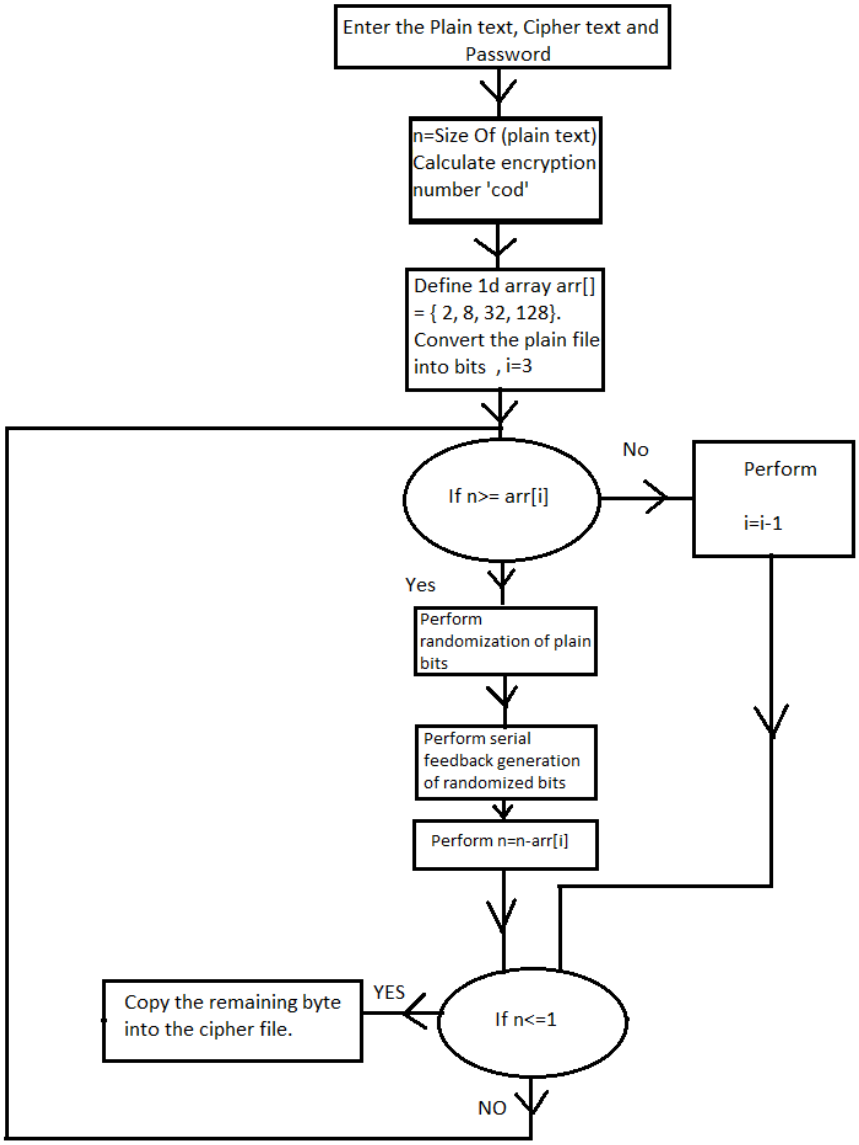


Fig. 1: The working of the EBES algorithm for every iteration

IV. Test Results and Cryptanalysis

In the present paper, two modules of advanced bit randomization and serial feedback generation method have been combined. The test results confirm that the algorithm not only works for every file format but also yields satisfactory test results for all possible file sizes.

The EBES algorithm has been tested with multiple files. The files have been altered subtly and the results have been recorded and analysed. Some of the results that have been included below are

A. Some general text inputs.

B. The variations of cipher file for different passwords.

C. Performance Analysis

D. Byte analysis of similar text inputs.

E. Frequency Analysis

4.1 Some general text inputs

The following table shows some miscellaneous text inputs as plain files and their corresponding ciphers. The text inputs have been made similar in terms of the constituting characters.

Table 2: Miscellaneous text inputs

Plain Text	Cipher Text
he is great	3Õr_±ñ‘[Si,
Aaaaaaaaaa	ãŽ/Öö_[FqR
bbbbbbbbbb	m_□iD¿(Ÿš@
Ccccccccccc	–3¹Š¯/ß¾Cb
Aabbbbaa	_Ðj ª_ »i ù_

4.2 The variation of cipher files for different password.

The idea behind this analysis is to study the effectiveness of the user password.

Table 3: The Variation Of Cipher File For Same Plain Text But Different Passwords

Plain Text	Password	Cipher Text
the Extensive Bit-level encryption mode	10	>ÿÄ £_ • Àg á_>Û ú DJØÜ\h ×Š- ["ô¨,__†!_€
the Extensive Bit-level encryption mode	11	«Ð·?Ç_õÝ~_òÇ€À jŠ¯ª ø¶ž– ¨Ûê ŸŒ„]} • »
the Extensive Bit-level encryption mode	12	> °_D__n õuh RK • _B_~è•Èßdé,_êwS • bWtM®
the Extensive Bit-level encryption mode	13	¿lŒUhþ?_d • ¨ï¯r Œp,³_n;_w.`¾ÌÓº Á ã¤C¿

4.3 Performance Analysis

The objective of this table is to study the encryption and decryption time for plain file for different sizes.

Table 4: Performance Analysis- the EBES algorithm has been tested with suitable time functions. The computation times for files of different sizes have been recorded for encryption and decryption for the same user password.

Plain Text	Time to Encrypt (in seconds)	Time to Decrypt (in seconds)
64 characters of 'A'	1	1
128 characters of 'A'	1	1
256 characters of 'A'	2	2
512 characters of 'A'	2	2

4.4 Byte Analysis of similar text inputs

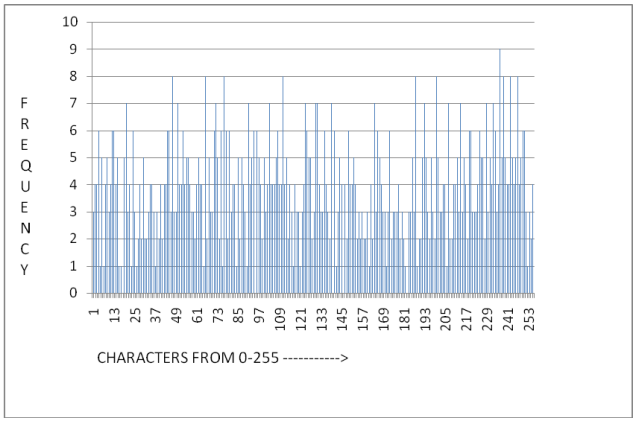
The EBES algorithm has been tested with similar but rare text inputs like 10 characters of ASCII 0, 1 and 2. This byte wise encryption confirms that for every byte of cipher file no repetitive patterns have been noticed.

Table 5: The following table performs byte wise comparison of 10 characters of ASCII 1, 2 and 3

Byte Number	Cipher byte for characters of ASCII 1	Cipher byte for characters of ASCII 2	Cipher byte for characters of ASCII 3
1	_	Œ	w
2		‚	_
3	~	Ü	è
4	•	"	Á
5	D	Ö	=
6	†	;	«
7	Í	¾	I
8	Þ		&
9	³	X	•
10	`	r	P

4.5 Frequency Analysis

This is the most crucial aspect of cryptanalysis as it explores the frequency spread of the characters in the cipher files. We check the occurrence of every character in the cipher file. It indicates the distribution of characters in the cipher file. In the graphs below x-axis represent the character set (0-255) whereas the y-axis represents the frequency of every character. For EBES method, the test results are quite remarkable for the text inputs of (i) 1024 characters of 'a' (ii) 1024 characters of ASCII 0.



Result-I corresponds to Figure-II (shown above). From the frequency analysis of 1024 'a', we can clearly understand that there are no clear dominance of any characters as the distribution of characters in the spectrum seems largely random.

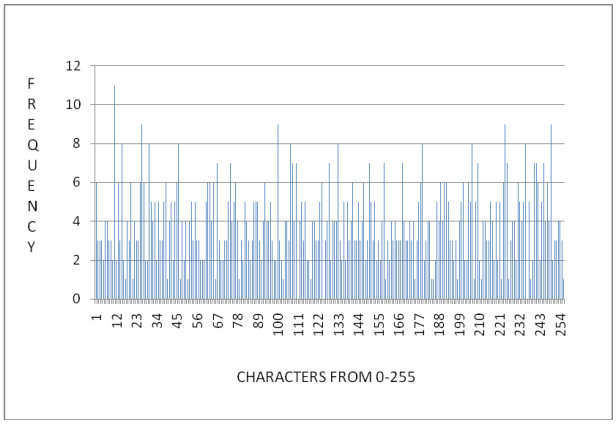


Fig. 3: The frequency analysis for plain file of 1024 bytes of ASCII 0

Fig-III corresponds to result-II (shown above). It is quite remarkable that 1024 occurrences of ASCII 0, that is 1024 X 8=9192 bits of ASCII 0 may have a frequency distribution as seen above. The reasons are not hard to fathom. The selective compliment of bits performed in the bit randomization models has yielded such a frequency distribution in EBES algorithm.

V. Conclusion and Future Scope

It is evident that the quality of encryption obtained at the bit level is significant as seen in this present algorithm EBES. The plain text files have been split into respective bits before we apply the aforementioned algorithms. The rare text files have been encrypted to test whether the algorithm can handle small messages as easily as long ones. Even when the same characters are provided as input, the cipher files have almost no occurrence of repetitive patterns. The use of multiple encryption and the role of the password provided by the user have also been demonstrated in the test results. Clearly, the user generated password is contributing greatly to the quality of encryption rendered.

Moreover the method of byte extraction based on the size of file is unique and efficient. The integration module follows the technique of extraction based on the size of the file. It adds to the effectiveness of the method. The idea of serial feedback is very new though it needs further attention for improvement.

Acknowledgement

The author is grateful to the Department of Computer Science of St. Xavier's College, Kolkata for their guidance and support.

References

[1] Ultra Encryption Standard(UES) Version-I :Symmetric Key Cryptosystem using generalized modified Vernam Cipher method, Permutation method and Columnar Transposition method, Satyaki Roy, Navajit Maitra, Shalabh Agarwal and Asoke Nath, Proceedings of RACCCT 2012, held at Surat , Mar 29-30, Page-81-88(2012)

[2] 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.

[3] Cryptography and Network, William Stallings, Prentice Hall of India.

[4] Cryptography & Network Security, B.A.Forouzan, Tata McGraw Hill Book Company.

[5] SD-AREE-I Cipher: Amalgamation of Bit Manipulation Modified VERNAM CIPHER & Modified Caesar Cipher (SD-AREE), International Journal of Modern Education and Computer Science (IJMECS), July, 2012.

[6] Ultra Encryption Standard Modified (UES) Version-I: Symmetric Key Cryptosystem with Multiple Encryption and Randomized Vernam Key Using Generalized Modified Vernam Cipher Method, Permutation Method, and Columnar Transposition Method, 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.

[7] 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.

Author's Profile

Satyaki Roy: He has recently graduated in computer Science from St. Xavier's College, Kolkata, India. He is currently starting to pursue his post-graduation. He has four publications in cryptography and recently presented his paper at an international conference. At present he is working on new encryption algorithms and improvement of his existent methods.

How to cite this paper: Satyaki Roy,"The Extensive Bit-level Encryption System (EBES)", International Journal of Information Technology and Computer Science(IJITCS), vol.5, no.5, pp.67-73, 2013.DOI: 10.5815/ijitcs.2013.05.09

Copyright © 2013 MECS