International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT22832

Bit Level Symmetric Key Encryption Algorithm using simple Shifting, Transposition Methods

Asoke Nath*, Shankhadeep Banerjee, Soumyadeep Lobwo, Proheli Paul

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

Article Info
Volume 8, Issue 2
Page Number : 309-318

Publication Issue :
March-April-2022

Article History
Accepted: 10 April 2022
Published: 30 April 2022

ABSTRACT

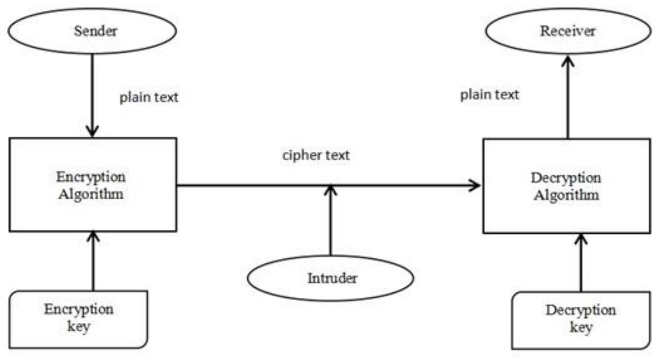
In today's digital world, 'information' is the key to almost 98% of all components in existence. This information, more commonly known as 'DATA' has immense potential and is one the most powerful resources in the world. The main idea behind this work is provide a data security measure which can ensure a much more refined methodology to protect data from any unauthorized access and thereby ensure its safety. In the present study the authors have used bit level symmetric key encryption algorithm to encrypt small pattern such as OTP, password, confidential message including Bank account details etc.

Keywords : Data, Data Security, Cryptography

I. INTRODUCTION

Cryptography is defined as the use of certain procedures to change the original content of a piece of information in a way, such that no one else except the intended recipient can understand the true meaning of this changed information. Traditional cryptography algorithms like Caesar Cipher, Playfair Cipher, etc. make use of symmetric key cryptography on account of its simplicity and ease of use. However, modern day cryptography demands a more secure data protection method and as such use asymmetric key cryptography for data encryption and decryption. One of the major algorithms in the field of cryptography is the 'Diffie - Hellman' algorithm which uses the concepts of prime numbers in order to exchange the keys between the sender and the receiver.

Another classification of cryptography methods is done on the basis of the mechanism used which can be divided into two groups - Substitution Cryptography and Transposition Cryptography. Transposition cryptography changes the position of the characters of the original message in order to encrypt it. This doesn't affect the size of the message, but renders it meaningless to a third party. Examples include - Rail Fence Cipher, Playfair Cipher, etc. On the other hand, substitution cryptography substitutes the characters of the original message with some other characters in order to encrypt it. This may affect the size of the message and also make it meaningless to a third party. Examples include - Cesar Cipher, Vernam Cipher, etc.



The arena of Bit level cryptography has already been explored to some level. Our project incorporates the fundamental principles mentioned in these existing works which are mentioned below:

1. Bit Level Symmetric Key Encryption Algorithm (BLSKEA)[3]:

This method deals with bit-level encryption and decryption methods. Nath et al (2014) already introduced a bit-level encryption method using feedback. But in the present paper, the authors have used some simple but very effective bit level encryption method. The plain text is initially converted to bits and after that bit-wise complement is done on some random prime positions. The entire bit stream is reversed and again applied bit complement operation in some random prime position. The bit complement is followed by bit-wise XOR operation and then the modified bit streams placed in a 2-dimensional array and perform some bit operations such as left-shift, up-shift, diagonal shift, cycling, right-shift number of times to make the bit patterns random. The bit operations are performed a number of times and finally, bits were converted to bytes and transferred to some output file. The results show that the present method is very much effective to encrypt passwords, SMS or any other confidential message. This algorithm has also beenimprovised.

2. Bit Level Encryption Standard (BLES) - Version-I[5]:

This method uses bit exchange and byte exchange methods with complements and XOR operation. The key element is the bit exchange depending on the randomized matrix which is generated every time and each one is unique. With different levels of extractions such as 2 bytes, 8bytes, 32 bytes and 128 bytes, the data finally gets shuffled to such an extent that without knowing the process and key, it would be impossible to decrypt. The authors have then implemented the bit-wise exchange method as follows:Firstly, they begin with initial transformation where the data is broken down to its corresponding bits and are then XOR-ed and complemented. These bits are stored in a reverse manner into a new file and this new file is now worked with. Secondly, randomization number and encryption number are calculated. Thirdly, first 2 bytes of data is extracted till the end of the file is read and is worked with, then 8 bytes, then 32 bytes and then 128 bytes. This process is executed till encryption number is reached. These multiple encryptions make their system more secure. This method will be most effective to encrypt short message such as SMS in mobile phones, password encryption and any type of confidential message. If the file size is large then the present method will take more time to encrypt.So therefore,BLES may be used in defence systems, Banking systems, Sensor networks, Mobile computing etc.

3. Bit Level Encryption Standard (BLES) - Version-II[6]:

In the method, the authors have introduced a new version of the previous symmetric key cryptography method called Bit Level Encryption Standard(BLES)Version-II which is basedonbit exchanging or bit reshuffling method from left to right as well as from right to left of theentire bit stream. In addition to that the authors have used a bitwise XOR operation to makethealgorithm more powerful. In BLES Version-I the authors had used bit exchange methods but with some fixed block sizes which were multiples of two. Due to the even power of two sometimes there were some repeat of characters in the encrypted file if the input plain text alsohad duplicate characters. To overcome this problem, in the present work the authors have takenblock size of squares of off numbers starting from three onwards. For scanning fromright toleft the authors used squares of even numbers starting from four onwards.After finishing the bit exchange the authors have performed bitwise XOR to make the cryptosystemalmost unbreakable. The authors have also introduced a special bit manipulation method so the encryption algorithm will work even for all characters with ASCII Code 0 or all characters withASCII Code 255. Most of the standard encryption algorithm will fail to encrypt a file whereall characters are ASCII '0' or all characters with ASCII '255' but the present method will be abletoencrypt a file where all characters are ASCII '0' or all characters are ASCII '255'. The present method will be effective for encrypting short messages, password, confidential key etc. Thespectral analysis in the result sections shows that the BLES version-II method is free fromknownplaintext attack, differential attack or any type brute force attack.

II. METHODS AND MATERIAL

The proposed algorithm uses the following two techniques - Columnar Transposition Method, and Shifting Technique.

A. Columnar Transposition Method

The Columnar transposition is a cipher technique in which the order of the position of the characters are re-arranged to obtain the cipher text. The necessary requirements for the columnar transposition are the key based on which the re-arrangement of the characters is done and secondly a transposition matrix.

Determining the dimensions of the transposition matrix

As mentioned before the number of rows and columns of the transposition matrix is determined from the key.

Column Length: In order to calculate the length of the row, it is necessary to calculate the maximum value in the key (say max). Then,

Number of columns(cols) = max + 1

Suppose the key is given as key = [4,1,0,3,2]. Then,
max = 4
cols = 4 + 1 = 5
Then the value of cols will be 5.

Row Length: In order to calculate the row length, we need to divide the length of the string content by the cols value and consider its ceiling value.

Number of rows (rows) = ceiling value of (length of the string / cols)

Suppose the string content is: “0111000000000010”.
Then the length of the string content (say n1) = 16.
Therefore, the value of rows will be:
rows = ceiling value of (16/5) = ceiling value of 3.2 = 4
Therefore, rows = 4

Encryption - Columnar Transposition

The algorithm for columnar transposition cipher is as follows:

Start
Calculate cols value and rows value

```
//Create arr (list of lists) of the dimension rows x cols
do arr = [[ '-' for j in range (cols)] for i in range (rows) ]
Enter the content of the string into arr.
for i ranging from 0 to length of the key
        do k = key [ i ]
        for j ranging from 0 to rows do
            if arr [ j ][ k ] != '-' is true then
            do encrypted = encrypted + arr [ j ][ k ]
Stop
```

Considering an example set to understand how Columnar Transposition Cipher works.

plaintext = “0100000101000010”
value of key, cols and rows mentioned above

Re-arrangement of the contents – After applying Columnar Transposition Cipher, the resultant cipher text that is obtained is-

	COL-0	COL-1	COL-2	COL-3	COL-4
ROW-0	0	1	0	0	0
ROW-1	0	0	1	0	1
ROW-2	0	0	0	0	1
ROW-3	0	-	-	-	-

Cipher text = “0111000000000010”

Decryption - Columnar Transposition

The algorithm for columnar transposition cipher is as follows:

```
Start
Calculate cols value and rows value
Do n1 = length of the encrypted string
//Create arr (list of lists) of the dimension rows x cols
do arr = [[ '-' for j in range( cols ) ] for i in
range( rows )]
do track = 0
for i ranging from 0 to length of the key
  col = key[i]
  for row ranging from 0 to rows
    do pos = row * cols + col
    if pos< n1 is true then
      if track < n1 is true then
        do arr[row][col] = encrypted[track]
        do track+=1
//extract the content of arr and store it in decrypted
for i ranging till rows
  for j ranging till cols
    if arr [ i ][ j ] != '-' is true then
      do decrypted = decrypted + arr[i ][ j ]
Stop
```

Considering the example set provided previously to understand how decryption of Columnar Transposition Cipher works.

Cipher text = “0111000000000010”
value of key, cols and rows mentioned above

Re-arrangement of the contents –After applying decryption of Columnar Transposition Cipher, the resultant decrypted text that is obtained is -

	COL-0	COL-1	COL-2	COL-3	COL-4
ROW-0	0	1	0	0	0
ROW-1	0	0	1	0	1
ROW-2	0	0	0	0	1
ROW-3	0	-	-	-	-

Decrypted text = “0100000101000010”

B. Shifting Algorithm

The shifting algorithm is a cipher technique in which the position of a particular character is shifted from one position to another and another character takes on its position.Theshifting algorithm implemented in the proposed cipher technique operates in two phases. The data set on which this technique is performed is a string of binary equivalent of the positions of all the 1s present in the cipher text obtained as a result of the

Columnar Transposition Cipher. The process is described further in the upcoming section.

Create string of binary equivalent of the position of all the 1s

Considering the cipher text obtained in the previous section,

Cipher text = "0111000000000010"

Next, the position of all the 1s is collected from the cipher text. They are,

Position of 1s – 1,2,3 and 14.

Now these position values are converted into their binary equivalent and then concatenated to a string.

1 ≡ 00000001; 2 ≡ 00000010; 3 ≡ 00000011; 14 ≡ 00001110

String = 00000001000000100000001100001110

The algorithm is as follows –

```
Start
c = 1
while c <= 2 do
      n2 = length of the string st
      do l= floor value of square-root
of(n2)
      do track=0
      st1=[['0' for j in range(l)]for
      i in range(l)]# creating a list
      of lists for shifting cipher
      for i ranging from 0 to l do
             for j ranging from 0 to l
                    if  track<(l*l)  is
      true then
       do st1[i][j]=st[track]
                                do
      track=track+1
      do res=st[track:n2]
      performing shifting cipher in
      the order - left, down, diagonal,
      up and then right shift
      store data of st1 in st
      do st = res + st
      do c = c + 1
Stop
```

Considering the string obtained in the previous section, the dimension of the list of lists needs to be calculated. This is done using the formula:

dimension = integral square root of (the length of the string)

Now divide the string into two parts:

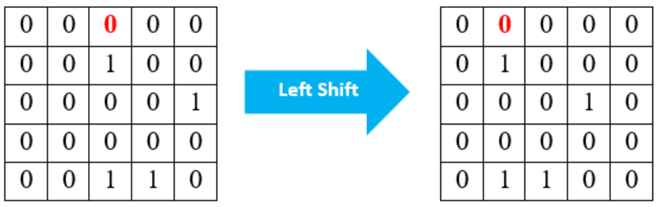
st = content of string from 0 to index which is value of dimension and residue(res) = rest of the string from index which is the value of dimension

Hence, st = **0000000100000010000000110** and res = **0001110**

The five types of shifting operations namely **left shift**, **down shift**, **diagonal shift**, **up shift** and **right shift** are performed in the proper order.

1. Left Shift

In this shift, all elements in every row of the shift matrix are shifted by one position to their left. For leftmost elements, they are shifted to the rightmost locations.

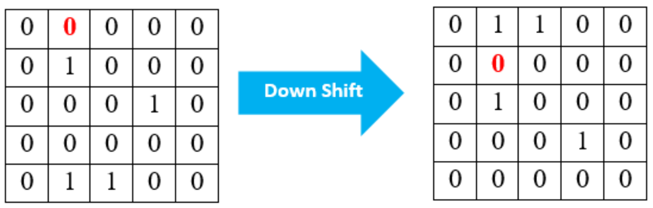


0	0	0	0	0
0	0	1	0	0
0	0	0	0	1
0	0	0	0	0
0	0	1	1	0

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

2. Down Shift

In this shift, all elements in every column of the shift matrix are shifted by one position downwards. For bottom most elements, they are shifted to the topmost locations.



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

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

3. Diagonal Shift

In this shift, all elements in opposite diagonal positions of the shift matrix are interchanged.

$$\begin{bmatrix}0&1&1&0&0\\0&0&0&0&0\\0&1&0&0&0\\0&0&0&1&0\\0&0&0&0&0\end{bmatrix} \xrightarrow{\text{Diagonal Shift}} \begin{bmatrix}0&1&1&0&0\\0&1&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$

4. Up Shift

In this shift, all elements in every column of the shift matrix are shifted by one position upwards. For topmost elements, they are shifted to the bottommost locations.

$$\begin{bmatrix}0&1&1&0&0\\0&1&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix} \xrightarrow{\text{Up Shift}} \begin{bmatrix}0&1&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&1&1&0&0\end{bmatrix}$$

5. Right Shift

In this shift, all elements in every row of the shift matrix are shifted by one position to their right. For rightmost elements, they are shifted to the leftmost locations.

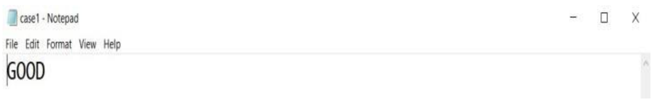
$$\begin{bmatrix}0&1&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&1&1&0&0\end{bmatrix} \xrightarrow{\text{Right Shift}} \begin{bmatrix}0&0&1&0&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&1&1&0\end{bmatrix}$$

III. RESULTS AND DISCUSSION

The proposed algorithm has been tested over a variety of input files and have generated satisfactory results.Accumulative study of the variation of the encrypted file content over the input file content has been performed. The analysis has been formulated on the basis of the change observed in the encrypted file, when certain character(s) of input file are changed, or when the input file characters exhibit a certain pattern. We have taken three test cases and drawn out the spectral analysis.

Case 1: Text used - **GOOD** (Key used: **2,0,1,3,4**)

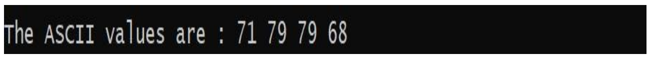
Input file:



GOOD

Fig III.1(a). Input file for Case 1(case1.txt)

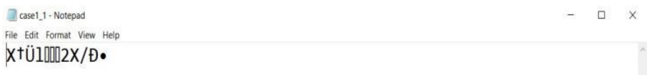
ASCII values:



```
The ASCII values are : 71 79 79 68
```

Fig III.1(b). ASCII values of input characters for Case 1

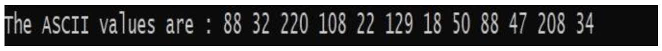
Encrypted file:



X†Ül□□2X/Ð•

Fig III.1(c). Encrypted file for Case1(case1_1.txt)

ASCII values:



```
The ASCII values are : 88 32 220 108 22 129 18 50 88 47 208 34
```

Fig III.1(d). ASCII values of encrypted characters for Case 1

From the above screenshots we can see that for **4** characters(**G, O, O, D**) of input file with ASCII values **71,79,79** and **68**,we get **12** characters in the output file with ASCII values **88, 32, 220, 108, 22, 129, 18, 50, 88, 47, 208** and **34** respectively. The graphical analysis of the same has been shown below:

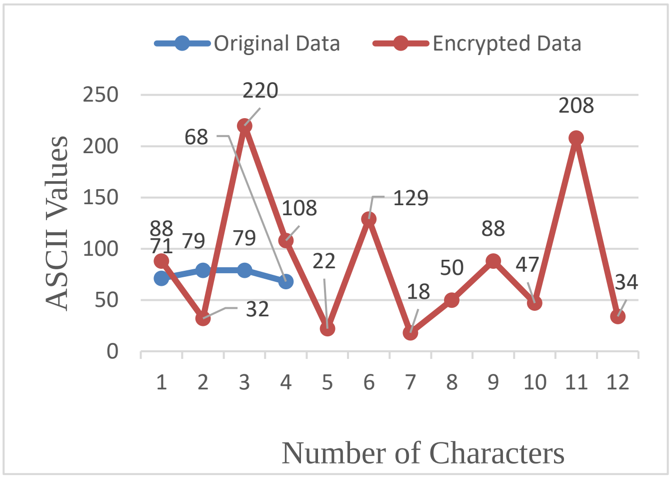
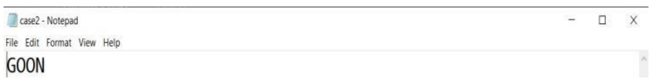


Fig III.1(e). Graphical representation of characters against their ASCII values for Case1

Case 2: Text used - **GOON** (Key used: **2,0,1,3,4**)

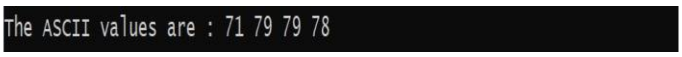
Input file:



GOON

Fig III.2(a). Input file for Case 2 (case2.txt)

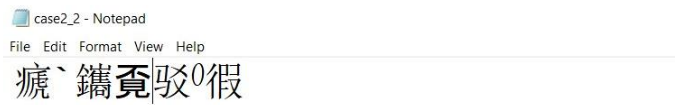
ASCII values:



```
The ASCII values are : 71 79 79 78
```

Fig III.2(b). ASCII values of input characters for Case 2

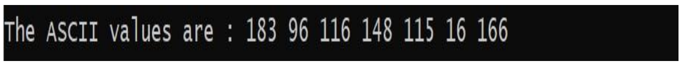
Encrypted file:



瘑` 鑴覔驳⁰假

Fig III.2(c). Encrypted file for Case 2 (case2_2.txt)

ASCII values:



```
The ASCII values are : 183 96 116 148 115 16 166
```

Fig III.2(d). ASCII values of encrypted characters for Case 2

From the above screenshots we can see that for **4** characters(**G, O, O, N**) of input file with ASCII values **71,79,79** and **78**,we get **7** characters in the output file with ASCII values **183, 96, 116, 148, 115, 16** and **166** respectively. The graphical analysis of the same has been shown below:

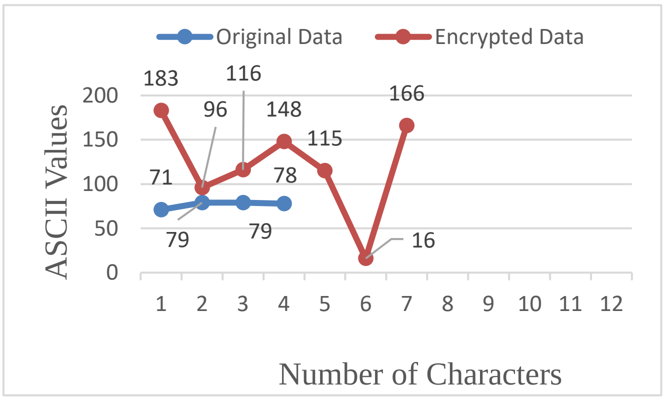
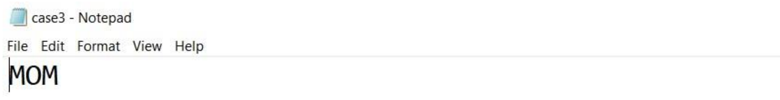


Fig III.2(e). Graphical representation of characters against their ASCII values for Case 2

Case 3: Text used - **MOM** (Key used: **2,0,1,3,4**)

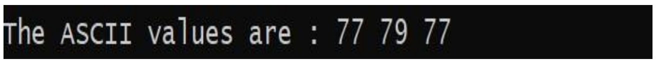
Input file:



MOM

Fig III.3(a). Input file for Case 3 (case3.txt)

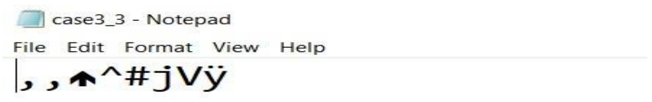
ASCII values:



```
The ASCII values are : 77 79 77
```

Fig III.3(b). ASCII values of input characters for Case 3

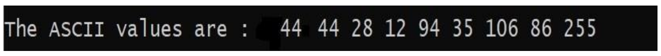
Encrypted file:



,,⬆^#jVÿ

Fig III.3(c). Encrypted file for Case 3 (case3_3.txt)

ASCII values:



```
The ASCII values are :  44 44 28 12 94 35 106 86 255
```

Fig III.3(d). ASCII values of encrypted characters for Case 3

From the above screenshots we can see that for **3** characters(**M, O, M**) of input file with ASCII values **77, 79,** and **77**, we get **9** characters in the output file with ASCII values **44, 44, 28, 12, 94, 35, 106, 86** and **255** respectively. The graphical analysis of the same has been shown below:

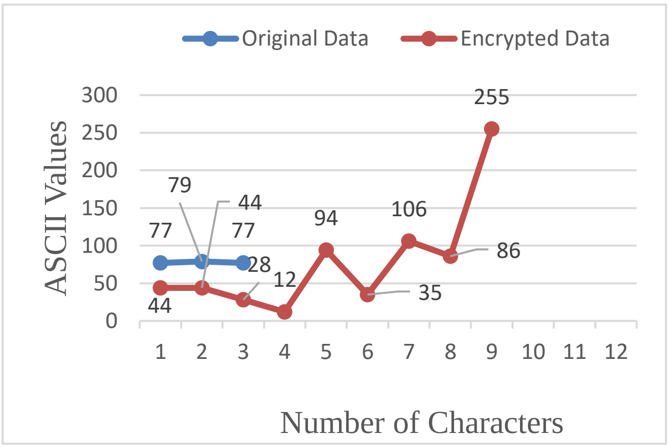


Fig III.3(e). Graphical representation of characters against their ASCII values for Case 3

After careful observation of all the three cases portrayed above, we conclude the following facts:

1. Unlike primitive cryptographic algorithms, like the RSA algorithm, where each input character is mapped to exactly one output character, our program has a random and dissimilar mapping technique.
2. From **Case 1** and **Case 2**, we observe that by just changing one character at the end of the input text, we get a complete new set of encrypted characters. This also proves the randomness of the algorithm used.
3. From **Case 3** we observe that for an input text exhibiting a palindrome pattern, the mapped characters are quite random even for same characters.
4. The variation between the number of characters in the input text and that in the encrypted text is not proportional. For **4** input characters we can get **9** or even **12** encrypted characters.

The proposed algorithm is greatly dependent on the size of the input file, on which it is being applied. As such, a spectral analysis could be drawn out to analyze the relation between the file size and the execution time of the proposed algorithm.

File type	File size(in bytes)	Encryption time(in seconds)	Decryption time(in seconds)
.txt	160	0.046875	0.0625
.png	67304	2.6875	2.984375
.xlsx	69568	2.578125	3.484375
.docx	88704	3.3125	3.796875
.pptx	254472	10.921875	11.96875

On the basis of the above table the graphical analysis can be done as follows:

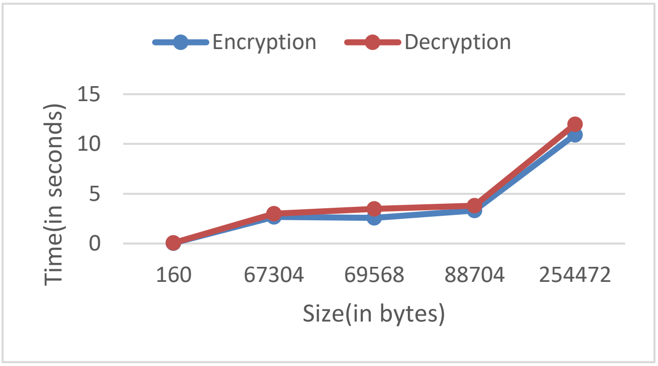


Fig III.4. Graphical analysis of execution time versus input file size

IV. CONCLUSION AND FUTURE SCOPE

The present method is extremely resilient to brute force attack.Since, there is a manipulation at the bit level, the program has a high-risk factor as even a single bit, if changed, can produce an entirely different result.The present algorithm can be applied to encrypt a variety of files ranging from as simple as a text file(.txt) to as complex as a video file(.mp4). Due to this, the applicability of this method is very wide. Majorly, this method of encryption and decryption can be used to secure OTPs, PINs, passwords, etc. There is much scope for the use of this technique and future improvements may allow that scope to expand even further.

V. ACKNOWLEDGEMENT

The authors are indebted to the Dept. of Computer Science, St. Xavier's College, Kolkata, for providing them an opportunity to work on cryptographic algorithms and the guidance and mental support throughout the course of the work. One of the authors AN is grateful to Rev. Dr. Dominic Savio, S.J., Principal, St. Xavier's College(Autonomous) for giving constant support to do research work in the field of Science and Engineering.

VI. REFERENCES

[1]. Asoke Nath, Sankar Das, Oishi Mazumder, AdrijaSaha, Monimoy Ghosh, "A New Bit Level

Positional Encryption Algorithm(NBLPEA ver-1)", International Journal ofComputer Sciences and Engineering, Vol. 8, Issue.4, pp.167-172, Apr 2020

[2]. Behrouz A. Forouzan, "Cryptography and Network Security", Special Indian edition2007, Tata Mc-Graw Hill Publishing LTD.

[3]. Nath, Asoke; Santra, Madhumita; Maji, Supriya; Fatema, Kanij Associate, Aleya Student, M "Bit Level Symmetric Key Encryption Algorithm (BLSKEA-1)Version-1". International Journal of Innovative Research in Computer and CommunicationEngineering (IJIRCCE ISSN 2320-9801). 3297. 10767-10773, 2015

[4]. Dan Boneh, Victor Shoup, "A Graduate Course in Applied Cryptography", StanfordUniversity, version 0.5, January 2020.

[5]. Neeraj Khanna, Dripto Chatterjee,Joyshree Nath,Asoke Nath, "Bit Level Encryption Standard(BLES):Version-I",International Journal of ComputerApplications(IJCA), (0975-8887) USA, Volume 52-No.2., Page.41-46, Aug2012.

[6]. Asoke Nath, Annie Chakraborty,Soumyaraj Maitra, "New Bit Level PositionalEncryption Algorithm (NBLPEA - ver 2)", International Journal of Scientific Researchin Computer Science, Engineering and Information Technology(IJSRCSEIT) | Volume 1 |Issue 1 | ISSN: 2456-3307, 2016

AUTHOR'S PROFILE

1. Dr. Asoke Nath is working as Associate Professor in the Department of Computer Science, St. Xavier's College (Autonomous), Kolkata. He is engaged in research work in the field of Cryptography and Network Security, Steganography, Green Computing, Big data analytics, Li-Fi Technology, Mathematical modelling of Social Area Networks, MOOCs, Quantum Computing etc. He has published more than 257 research articles in different Journals and conference proceedings.

2. Mr. Shankhadeep Banerjee is currently a 3rdyear undergraduate student in the Department of Computer Science, St.Xavier's College (Autonomous), Kolkata.His interests lie in Web Development, Cryptography, Data Science and real-world project implementations of these fields.

3. Mr. Soumyadeep Lobwo is currently a 3rd year UG student of the Department of Computer Science, St. Xavier's College, Kolkata. His interest lies in the field ofWeb Development, Software Engineering,Machine Learning, Cryptanalysis and their real-world implications.

4. Miss Proheli Pauliscurrently a 3rdyear undergraduate student in the Department of Computer Science, St.Xavier's College (Autonomous), Kolkata. Her interests lie in Web Designing, Cryptography and Network Security.

Cite this article as :

Asoke Nath, Shankhadeep Banerjee, Soumyadeep Lobwo, Proheli Paul, "Bit Level Symmetric Key Encryption Algorithm using simple Shifting, Transposition Methods", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 8 Issue 2, pp. 309-318, March-April 2022. Available at doi : https://doi.org/10.32628/CSEIT22832
Journal URL : https://ijsrcseit.com/CSEIT22832