International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT217350

New Bit Level Positional Encryption Algorithm (NBPLEA - Ver 2)

Asoke Nath*[1], Annie Chakraborty[1], Soumyaraj Maitra[1]
[1]Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India

Article Info
Volume 7, Issue 3
Page Number: 245-257
Publication Issue :
May-June-2021
Article History
Accepted : 15 May 2021
Published : 22 May 2021

ABSTRACT

In the age of the digital world, cyber security and protecting the digital assets of a person and organization are of the utmost importance. Various types of attackers and foreign entities attempt to break in and steal or modify records and valuable assets for their personal gain which is why designing efficient and unbreakable cryptographic algorithms is a very crucial field of Computer Science.[2] In this project, we develop a cryptographic algorithm that is able to encrypt small texts like passwords or pins.

Keywords: Plain Text, Cipher Text, Encryption, Decryption, Key.

I. INTRODUCTION

Nowadays the danger of an intruder intercepting the message sent by a sender to a receiver is very much present. No communication channel is ever truly secure. One of the chief ways of maintaining confidentiality is to convert the message into some seemingly gibberish text so that when the intruder intercepts it, he/she won't be able to make sense of it. The receiver, when he/she receives the gibberish text, will be able to convert it to get the original message. The process of converting the message to gibberish text is called encryption[2] and it uses a cryptographic algorithm that takes in the message as input and produces a ciphertext as output. The gibberish text is called the ciphertext. It is used by the sender. The process of converting the gibberish text to a message is called decryption[2] and it uses a cryptographic algorithm that takes in the ciphertext as input and produces the original message as output. It is used by the receiver.

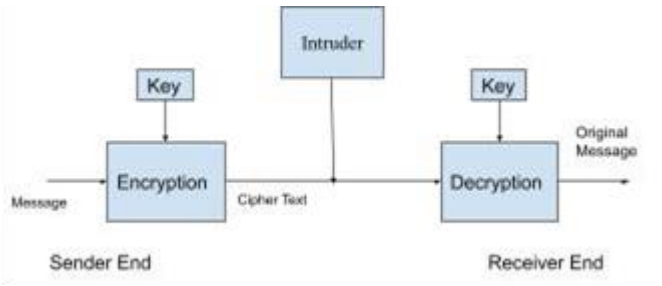


Fig 1: A pictorial demonstration of the encryption and decryption process.

In most cases nowadays, the cryptographic algorithms used are public. Hence, to secure the procedure even further, the algorithms use a secret key in the encryption and decryption process. This key is only shared between the sender and the intended receivers. When only one key is involved in both the encryption and the decryption process it is called Symmetric Key Cryptography. However, there is a different form of Cryptography called Asymmetric Key Cryptography where the sender encrypts the message using one key and the sender decrypts the message using a different key [7].

Traditional Ciphers worked by encrypting text characters from text files. Some examples of Traditional Ciphers are Substitution Ciphers which substitutes every character with one of the other 26 English alphabets like Monoalphabetic Substitution Cipher (eg. Caesar Cipher, Additive Cipher, Affine Cipher, Multiplicative Cipher, etc.) and Polyalphabetic Substitution Ciphers (eg. Autokey Cipher, Playfair Cipher, Vigenere Cipher, etc.)[2] and then there are Transposition Ciphers which instead change the location of the symbols like Rail Fence Cipher. [4]

Modern ciphers work by converting files to bits and then performing the encipherment and decipherment process. This allows the sender to encrypt not only text files but image files, video files, audio files, executable files etc. since information, with the advent of the digital age, is not limited to text. Additionally, when we convert data, say text, to bits every character gets replaced by 8 (or 16) bits which means the number of symbols increases 8 (or 16 times) and this makes the work of an intruder tougher thereby increasing security.

Some of the works already done in the field of Bit Level Encryption have been discussed below.

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

This method deals with bit-level encryption and decryption methods. Nath et al(2014)[3] 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 bitwise complement is done on some random prime positions. The entire bitstream is reversed and again applied bit complement operation in some random prime position. The bit complement is followed by bitwise XOR operation and then the modified bitstreams 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 been improvised.

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

This method uses bit exchange and byte exchange methods with complements and xor operations. The key element is the bit exchange depending on the randomized matrix which is generated every time making certain that each one is unique. With different levels of extractions such as 2 bytes, 8bytes, 32 bytes, and 128 bytes, in powers of 2, 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 in the following manner:

Firstly, they begin with initial transformation where the data is broken down to its corresponding bits and are then xored and complemented. These bits are stored in a reverse manner into a new file and then they work on this new file. Secondly, they calculate the randomization number and encryption number. Thirdly, the first 2 bytes of data are extracted till the end of the file is read and is worked with, then they extract 8 bytes, then 32 bytes, and then 128 bytes. This process is executed till the encryption number is reached. These multiple encryptions make their system more secure. This method will be most effective to encrypt short messages 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 defense 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 cryptographic method called Bit Level Encryption Standard(BLES) Version-II which is based on bit exchanging or bit reshuffling method from left to right as well as from right to left of the entire bitstream. In addition to that, the authors have used a bitwise XOR operation to make the algorithm more powerful. In BLES Version-I[5] 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 also had duplicate characters. To overcome this problem, in the present work the authors have taken block size of squares of off numbers starting from three onwards. For scanning from right to left 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 cryptosystem almost 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 with ASCII Code 255. Most of the standard encryption algorithm will fail to encrypt a file where all characters are ASCII '0' or all characters with ASCII '255' but the present method will be able to encrypt 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. The spectral analysis in the result sections shows that the BLES version-II method is free from known-plaintext attack, differential attack or any type brute force attack. [Quoted from the abstract of the paper[6]].

II. METHODS AND MATERIAL

The encryption functions used in the algorithm are - One's complement operations, bitwise XOR operations and shift operations namely left shift, right shift, down shift, up shift and diagonal shift. A brief description of these operations have been given below.

A. One's Complement Operation:

In the Binary Number Representation System, there are only two possible values: 0(off) and 1(on). The 1's complement of a given binary number is obtained by inverting every individual digit of the number, i.e the 1's are converted to 0 and 0's are converted to 1.

Example:
Consider the following binary number, 10100101. When 1's complement operation is performed on this number, we convert all the 1's to 0 and 0's to 1 to obtain the result as follows:

Before 1's Complement: 10100101
After 1's Complement: 01011010

B. Bitwise XOR Operation:

The Bitwise XOR Operation between two binary numbers generates a binary number where the resultant bit is 1 if the corresponding bits of the two operands are opposite, otherwise the resultant bit is 0. The following truth table describes this concept:

A	B	AXORB
0	0	0
0	1	1
1	0	1
1	1	0

Example:
Consider the following binary numbers, 1010 and 0101.
The Bitwise XOR Operation on these two binary numbers generates the output 11110 as follows:
Operand 1: 10100
Operand 2: 01010

Result: 11110

C. Left Shift Operation:

The Bitwise Left Shift Operation on a binary number is performed by shifting every individual bit by one position to the left.
If the operation is performed on a matrix, all elements in each row are shifted by one position to the left.

Example:
Consider a binary number, 00101.
The Bitwise Left Shift Operation on this number generates the number 01010 as the Least Significant Bit(1) changes to 0 however the Most Significant Bit(MSB) remains the same.

D. Right Shift Operation:

The Bitwise Right Shift Operation on a binary number is performed by shifting every individual bit by one position to the right.
If the operation is performed on a matrix, all elements in each row are shifted by one position to the right.
Example:
Consider a binary number, 00101.
The Bitwise Right Shift Operation on this number generates the number 10010 as the Least Significant Bit(1) changes to 0 and the Most Significant Bit(MSB) becomes 1.

E. Up Shift Operation:

The Up Shift Operation on a matrix is performed by shifting all elements in each column by one position in the upwards direction.

Example:
Original Matrix:

1	0	1
0	1	0
1	1	1

After Up-Shift Operation:

0	1	0
1	1	1
1	0	1

F. Down Shift Operation:

The Down Shift Operation on a matrix is performed by shifting all elements in each column by one position in the downwards direction.

Example:

Original Matrix:

1	0	1
0	1	0
1	1	1

After Down-Shift Operation:

1	1	1
1	0	1
0	1	0

G. Diagonal Shift:

The Diagonal Shift Operation on a matrix is performed by shifting all elements in each diagonal by one position in the diagonal direction.

Example:
Original Matrix:

1	1	0
1	0	1
1	0	0

After Diagonal-Shift Operation:

0	1	1
1	0	1
0	0	1

H. THE ENCRYPTION ALGORITHM

1. The name of the input plain text file and the output cipher text file, where the encrypted text is to be written, is taken as input from the user.
2. Every character from the plain text file is extracted and converted into an 8 bit data.
3. The bitstream obtained from extracting all the characters is traversed and the positions of 0s are stored in an array. For example, if the bit stream obtained from the previous step is 01011010 then the positions of 0s are 1, 3, 6and 8.
4. Each of the positions obtained from the previous step is converted into its equivalent 32 bit binary format and then appended to a string.
5. After all the positions of 0s are converted into their equivalent 32 bit format, the length of the resultant bit string is calculated.
6. The user is prompted to enter a key that lies in the range of 0 to the square root of the resultant bit string length. This ensures that the matrix capacity doesn't exceed the bitstream length.
7. The bits in the non-prime positions are complemented.
8. The bit stream obtained from the previous step is reversed and then the bits in the prime positions are complemented.
9. The bits in the odd positions are XORed with the bits in the even positions and the result is then stored in the even position, i.e., we XOR the bits in positions 1 and 2 and place the result in position 2, then we XOR the bits in positions 3 and 4 and place the result in position 4 and so on.
10. The bit stream obtained from the previous step is reversed.
11. The bits in the first and last position are XORed and the result is stored in the last position, i.e., first we XOR bits in position 1 and position n and place the result in position n, then we XOR bits in position 2 and position (n-2) and place the result in position (n-2) and so on.
12. A matrix of dimension key X key is declared and filled with the first key X key number of bits (obtained from the previous step). The following rotation operations are performed on the matrix-

- **Left shift** where all the elements ofevery column are shifted to the left and the elements of the last column shift to the first column.
- **Diagonal shift** where all the elementsof both diagonals are shifted by one unit diagonally along both diagonals.
- **Down shift** where the entire elements ofevery column are shifted towards the bottom by one unit and the elements of the bottom most row move to the top.
- **Right shift** where all the elements of every column are shifted to the right and the elements of the first column shift to the last column.
- **Up shift** where all the elements of every column are shifted upwards and the elements of the top most row shift to the bottom.

After performing these operations we store the modified key X key bits in a new string.

13. Step 12 is executed with every successive batch of key X key elements until the number of bits left in the bitstream is less than the value of key X key.
14. If after n number of iterations of step 12, k bits remain where k < key X key then these k bits are called residual bits. These residual bits are appended to the new string.
15. The bits in the non prime positions of the new modified bitstream which was obtained after the series of rotations, are complemented.
16. The bitstream obtained from the previous step is reversed. Then the bits in the prime positions are complemented.
17. We then extract 8 bits from the final bitstream obtained from the above steps and calculate its decimal equivalent which will give the ASCII value of the cipher text character to be written to the output cipher file. This step is repeated for all the successive 8 bits until the end of the bitstream is reached.

I. THE DECRYPTION ALGORITHM

1. The name of the input cipher text file and the output plain text file, where the decrypted text is to be written, is taken as input from the user.
2. Every character from the cipher text file is extracted and converted into an 8 bit data.
3. The user is prompted to enter a key that lies in the range of 0 to the square root of the resultant bit string length. This ensures that the matrix capacity doesn't exceed the bitstream length.
4. The bits in the prime positions of the bitstream obtained from the previous step are complemented.
5. The bitstream is then reversed and then the non prime positions are complemented.
6. A matrix of dimension key X key is declared and filled with the first key X key number of bits from the bitstream obtained from step 4.
7. The following rotation operations are performed on the matrix-

- **Left shift** where all the elements ofevery column are shifted to the left and the elements of the last column shift to the first column.
- **Diagonal shift** where all the elementsof both diagonals are shifted by one unit diagonally along both diagonals.
- **Down shift** where all the elements ofevery column are shifted towards the bottom by one unit and the elements of the bottom most row move to the top.
- **Right shift** where all the elements ofevery column are shifted to the right and the elements of the first column shift to the last column.
- **Up shift** where all the elements ofevery column are shifted upwards and the elements of the top most row shift to the bottom.

After performing these operations the modified key X key bits are appended to a new string.

8. Step 7 is executed with every successive batch of key X key elements until the number of bits left in the bitstream is less than key X key.
9. If after n number of iterations of step 7, k bits remain where k < key X key then these k bits are called residual bits. These residual bits are appended to the new string.
10. XOR operation is performed on the bits in the first and last positions of the bitstream, i.e., first we XOR bits in position 1 and position n and place the result in position n, then we XOR bits in position 2 and position (n-2) and place the result in position (n-2) and so on.
11. The bitstream is reversed and the bits in the even and odd positions are XORed and the result is placed in the even positions, i.e., we XOR the bits

in positions 1 and 2 and place the result in position 2, then we XOR the bits in positions 3 and 4 and place the result in position 4 and so on.

12. The bits in the prime positions of the above bitstream are complemented and then the bitstream is reversed.
13. We complement the bits in the non prime positions.
14. The bitstream obtained from the previous step has the positions of 0s. We extract 32 bits at a time and convert it to get their decimal equivalent. The decimal equivalent gives the position of the 0s.
15. The original bitstream is formed by substituting 0s in the positions mentioned in the above array and 1s in the other positions.
16. Extract 8 bits at a time from the above bitstream and convert it to its decimal equivalent. This gives the ASCII value of the decrypted plaintext character. The character is written to the output file. This step is performed until all the bits have been read.

III. RESULTS AND DISCUSSION

The testing of this algorithm was conducted with the following five(5) sets of data, each with a purpose to test a specific aspect of the implemented approach:

A. Set-1:

1) Inputs in this set: (i)AAAA, (ii)AAAAA
2) Key: 16 (for both the inputs).
3) The Aspect of the algorithm being tested: Changes observed in the ASCII values of the encrypted text due to a minor change in the Input(presence of one extra A in case (ii))
4) Case-1:

INPUT - AAAA:

Fig (1). (i). (a): Input file containing the data AAAA

ASCII VALUES OF THE GENERATED CIPHERTEXT:

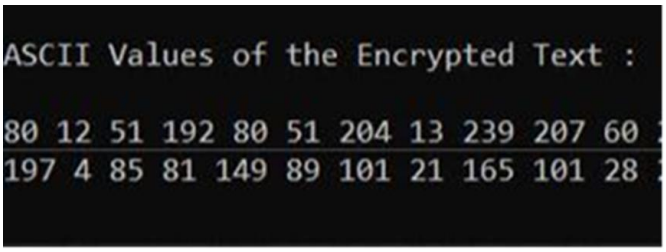


Fig (1).(i).(b): Command-Line Output showing the ASCII values of the generated ciphertext

GENERATED CIPHERTEXT:

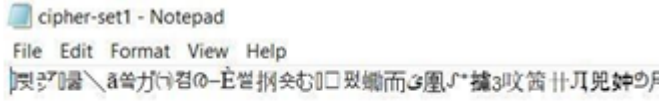


Fig (1).(i).(c): File containing the Generated Ciphertext

5) Case-2:

INPUT - AAAAA:

Fig (1).(ii).(a): Input file containing the data AAAAA

ASCII VALUES OF THE GENERATED CIPHERTEXT:

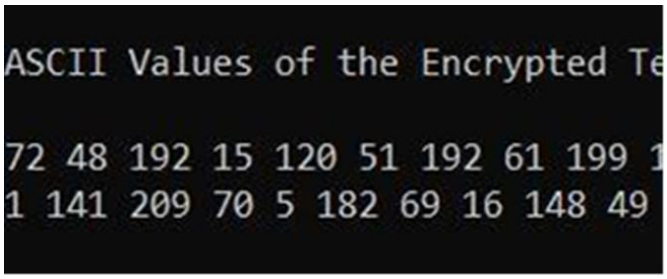
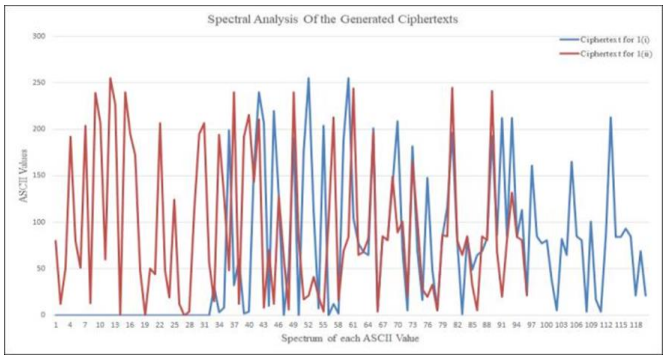


Fig (1).(ii).(b): Command-Line Output showing the ASCII values of the generated ciphertext

GENERATED CIPHERTEXT:

Fig (1).(ii).(c): File containing the Generated Ciphertext

6) Spectral Analysis for Set-1:



7) Conclusion for Set-1:

The first four ASCII values of the generated ciphertext for (i) and (ii) are {80,12,51,192}and {72,48,192,15}. This proves that the ASCII values of the generated ciphertexts are completely unique even though the first four(4) alphabets of the input text are identical.

B. Set-2:

1) Inputs in this set: (i)A, (ii)B
2) Key: 10 (for both the inputs).
3) The Aspect of the algorithm being tested: Changes observed in the ciphertext for two distinct single alphabets Inputs.
4) Case-1:

INPUT - A:

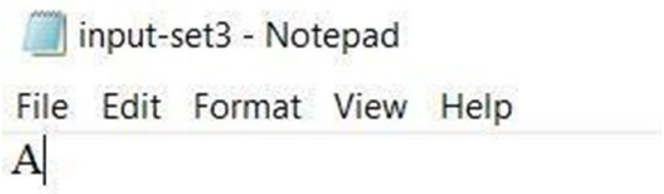


Fig (2).(i).(a): Input file containing the data A

ASCII VALUES OF THE GENERATED CIPHERTEXT:

Fig (2).(i).(b): Command-Line Output showing the ASCII values of the generated ciphertext

GENERATED CIPHERTEXT:

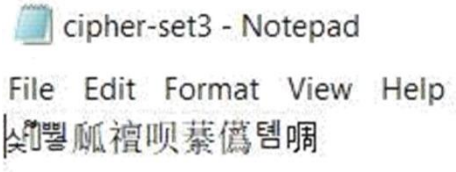


Fig (2).(ii).(c): File containing the Generated Ciphertext

5) Case-2:

INPUT - B:

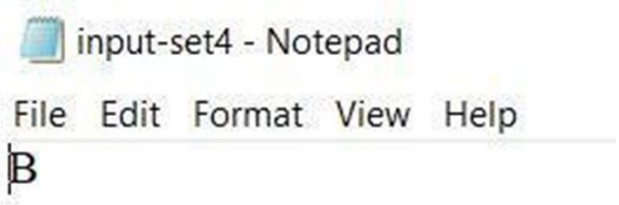


Fig (2).(ii).(a): Input file containing the data B

ASCII VALUES OF THE GENERATED CIPHERTEXT:

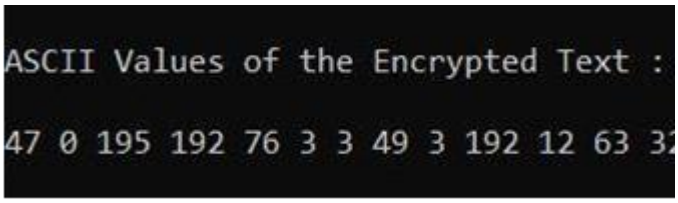
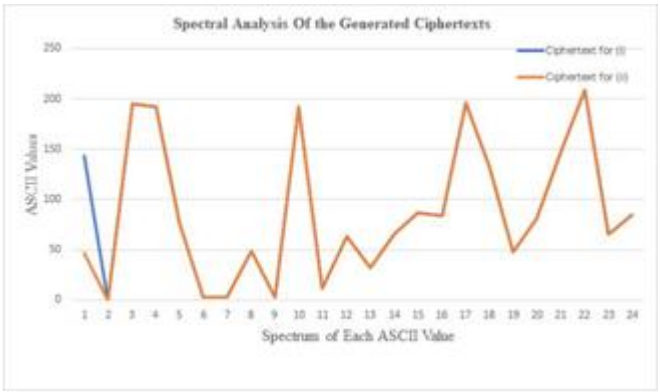


Fig (2).(ii).(b): Command-Line Output showing the ASCII values of the generated ciphertext

GENERATED CIPHERTEXT:

6) Spectral Analysis for Set-2:



7) Conclusion for Set-2:

Due to the complexity of the encryption algorithm, the ciphertexts generated in case of two distinct Inputs of 1 byte only, are still completely unique.

C. Set-3:

1) Inputs in this set:
   (i) CMSA SEM-6,
   (ii)ARTIFICIAL INTELLIGENCE
2) Key: 12 (for both the inputs).
3) The Aspect of the algorithm being tested: Effectiveness of the algorithm for encrypting alphanumeric data from real life.
4) Case-1:

INPUT - CMSA SEM-6:

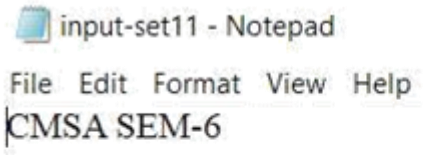


Fig (3).(i).(a): Input file containing the data CMSA SEM-6

ASCII VALUES OF THE GENERATED CIPHERTEXT:

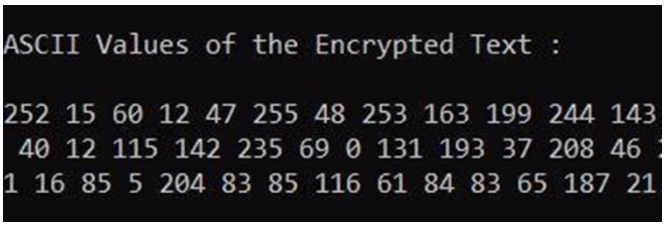


Fig (3).(i).(b): Command-Line Output showing the ASCII values of the generated ciphertext

GENERATED CIPHERTEXT:

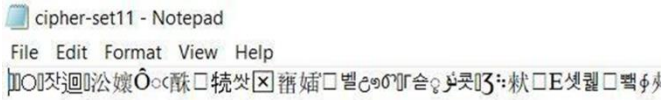


Fig (3).(i).(c): File containing the Generated Ciphertext

5) Case-2:

INPUT - ARTIFICIAL INTELLIGENCE:

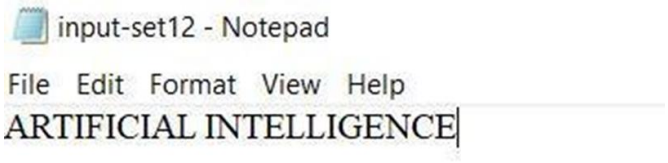


Fig (3).(ii).(a): Input file containing the data ARTIFICIAL INTELLIGENCE

ASCII VALUES OF THE GENERATED CIPHERTEXT:

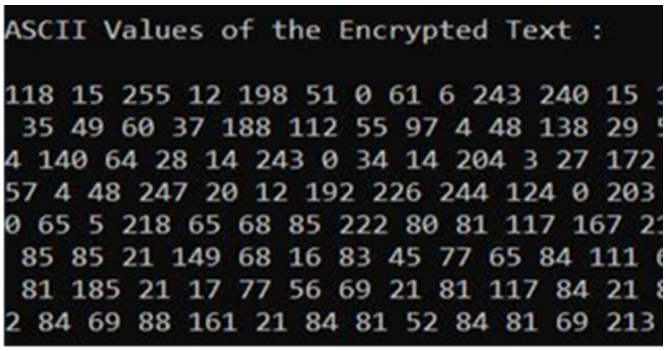


Fig (3).(ii).(b): Command-Line Output showing the ASCII values of the generated ciphertext

GENERATED CIPHERTEXT

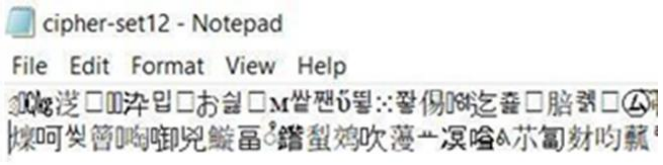
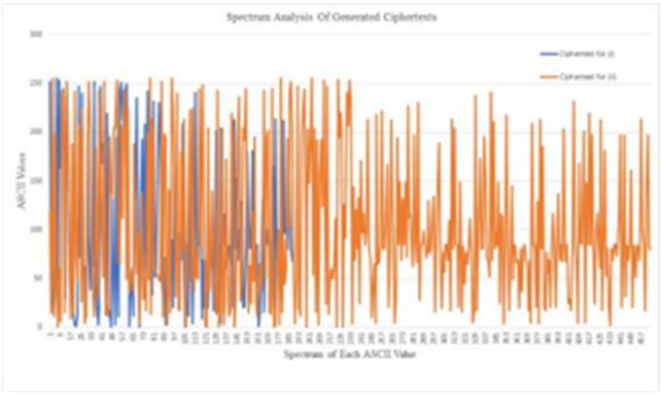


Fig (3).(ii).(c): File containing the Generated Ciphertext

6) Spectral Analysis for Set-3:



7) Conclusion for Set-3:

The algorithm successfully performs all the expected functions on the Input files containing Alphanumeric data from real life as inputs.

D. Set-4:

1) Inputs in this set: (i) ♦♦♦♦☻ ☻ ☺ (4 ASCII 4's, 2 ASCII 2's, 1 ASCII 1)
2) Key: 15 (for the first iteration), 5 (for the second iteration).
3) The Aspect of the algorithm being tested: Change in ASCII values of the Encrypted Ciphertext due to change in Key during different iterations of the algorithm encrypting the same Input data.

TABLE- I. COMPARATIVE ANALYSIS OF INPUTS BASED ON DIFFERENT KEYS

Subject	Key: 15	Key: 5
Input	input-set13 - N File Edit Forma ♦♦♦♦☻ ☻ ☺	input-set13 - I File Edit Form ♦♦♦♦☻ ☻ ☺

Subject	Key: 15	Key: 5
ASCII values for the Generated Ciphertext (3 Values)	217 63 243	4 185 191
Generated Ciphertext	*cipher-forkey15 - Notepad File Edit Format View Hel	*cipher-forkey5 - Nc File Edit Format Vi

4) Conclusion for Set-4:

The algorithm generates completely unique ciphertexts due to a change in the Key, and successfully decrypts the ciphertext provided that the Keys given as Input as identical during both the Encryption and Decryption process.

E. Set-5:

1) Input(s) in this set: (i) ▪▪▪▪▪▪▪▪ (8 ASCII 8's)
2) Key: 22 (for the first iteration), 17 (for the second iteration).
3) The Aspect of the algorithm being tested: Inability to Decrypt the Encrypted Ciphertext unless the Receiver has the correct Key used during the Encryption phase of the process.

INPUT - ▪▪▪▪▪▪▪▪ :

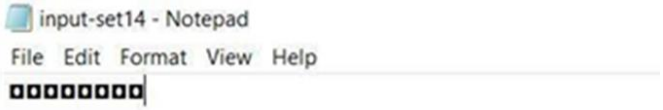


Fig (5).(a): Input file containing the data 8 ASCII 8's

ASCII VALUES OF THE GENERATED CIPHERTEXT:

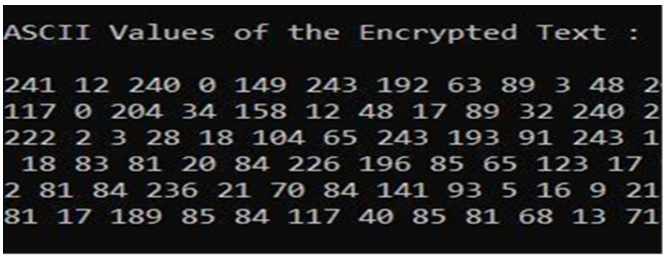


Fig (5).(b): ASCII values of the Generated Ciphertext

GENERATED CIPHERTEXT:

Fig (5).(c): Encrypted file containing Generated Ciphertext

DECRYPTED TEXT(FOR KEY 22):

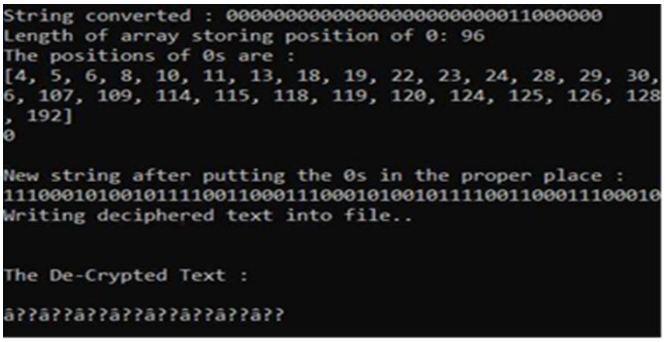


Fig (5).(d): Command Line Output Of the Decryption Algorithm (for Key 22)

DECRYPTED TEXT (FOR KEY 17):

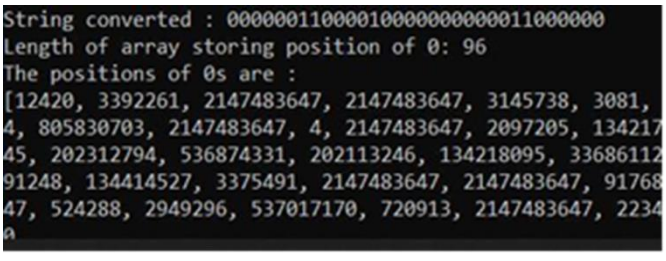


Fig (5).(e): Command Line Output Of the Decryption Algorithm (for Key 17)

4) Conclusion for Set-5:

The Algorithm effectively performs all phases of the process and generates the Decrypted text successfully if and only if the Key provided during both phases of the process (Encryption and Decryption) are the same. In any other case, the Algorithm does not generate the expected Decrypted text thus preserving the security aspect of the Encryption process where a Ciphertext can only be decrypted using the unique Key used in the Encryption phase of the algorithm.

IV.CONCLUSION

The text cannot be decrypted without the knowledge of the key used in encryption. The key domain is variable and depends on the size of the input plaintext file. The spectral analysis shows that our present method is free from standard cryptography attacks namely brute force attack, known plaintext attack and differential attack.

This cryptographic algorithm can be used to encrypt small texts like OTPs, PINs and Passwords.

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 in these tough and uncertain scenarios.

VI. REFERENCES

[1]. Asoke Nath, Sankar Das, Oishi Mazumder, Adrija Saha, Monimoy Ghosh, " A New Bit Level PositionalEncryption Algorithm(NBLPEA ver-1)", International Journal of Computer Sciences and Engineering, Vol. 8, Issue.4, pp.167-172, Apr 2020

[2]. Behrouz A. Forouzan, "Cryptography and Network Security", Special Indian edition 2007, Tata Mc-Graw Hill Publishing LTD.

[3]. Nath, Asoke & Santra, Madhumita & Maji, Supriya & Fatema, Kanij & Associate, Aleya & Student, M.(2015). Bit Level Symmetric Key Encryption Algorithm (BLSKEA-1) Version-1. International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE ISSN 2320-9801). 3297.10767-10773.

[4]. Dan Boneh, Victor Shoup, "A Graduate Course in Applied Cryptography." from Stanford University.

[5]. Bit Level Encryption Standard(BLES) : Version-I, Neeraj Khanna, Dripto Chatterjee, Joyshree Nath and Asoke Nath, International Journal of Computer Applications(IJCA)(0975- 8887) USA Volume 52-No.2.,Aug, Page.41-46(2012).

[6]. Bhadra, Gaurav & Baia, Tanya & Banik, Samik & Nath, Asoke & Nath, Joyshree. (2012). Bit Level Encryption Standard (BLES): Version-II. 121-127. 10.1109/WICT.2012.6409061.

[7]. Symmetric Key Cryptography by Security Encyclopedia

AUTHOR'S PROFILE

1. Dr. Asoke Nath is workingas 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 etc.

He has published 253 research articles in different Journals and conference proceedings.

2. Miss Annie Chakrabortyiscurrently a third year undergraduate student in the Department of Computer Science, St.Xavier's College (Autonomous), Kolkata. Her interests lie in Web Development, Cryptography, Data Science and real world project implementations of these fields.

3. Mr. Soumyaraj Maitra is a student of the Department of Computer Science, St. Xavier's College, Kolkata. He is interested in the fields of Cryptography, Database Management System (DBMS), Automata Theory and is inclined to be a part of the integration of the various aspects of these technologies in our day-to-day lives.

Asoke Nath, Annie Chakraborty, Soumyaraj Maitra, "New Bit Level Positional Encryption Algorithm (NBPLEA - Ver 2)", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 3, pp. 245-257, May-June 2021. Available at
doi : https://doi.org/10.32628/CSEIT217350
Journal URL : https://ijsrcseit.com/CSEIT217350

Cite this Article