IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-4, April 2019 E-ISSN: 2347-2693

Bit Level Encryption Algorithm Using Chaos Version 1.0(BLEAUC-1.0)

Asoke Nath[1*], Suchandra Datta[2], Souptik Kumar Majumdar[3]

[1,2,3]Dept. of Computer Science, St. Xavier's College (Autonomous), Kolkata, India

*Corresponding Author: asokejoy1@gmail.com, Tel.: +033-

DOI: https://doi.org/10.26438/ijcse/v7i4.451456 | Available online at: www.ijcseonline.org

Accepted: 12/Apr/2019, Published: 30/Apr/2019

Abstract- Cryptographic algorithms attempt to generate ciphertexts which are as seemingly unrelated to the plain text as possible. We attempt to add a further improvement to existing bit level encryption by incorporating chaos through Game of Life. Chaos is introduced in our algorithm to make the enciphering process even more random. This is made possible by using a reversible encryption process using XOR operation in the encryption algorithm. The algorithm is very bit sensitive i.e. any change in plain text bits will produce a completely different result: cipher text using the same key. The main focus of our algorithm was to introduce randomness in encryption which has been achieved by generating a random matrix consisting of dead and alive cells. The said matrix is used for encryption. The matrix generated is seemingly random depending upon the key given. Thus, achieving the desired target of making the encryption and decryption algorithm even more complex.

Keyword- Bit level encryption; differential attack; brute force attack; leftshift; rightshift; chaos using game of life matrix; DNAsequence.

I. INTRODUCTION

The exponential growth in technologies, network traffic and digital communication has resulted in augmenting the importance of data as one of the most valuable and vulnerable resources which affects companies and individuals alike. It is imperative to disguise data whenever it is in transit, a feat achieved by encryption, to prevent unauthorized access. Over the years, encryption techniques have evolved from simple shift ciphers to modern DNA encryption algorithms. DNA cryptography is one of the rapid emerging technology which works on concepts of DNA computing. A new technique for securing data was introduced using the biological structure of DNA called DNA Computing (aka molecular computing or biological computing). It was invented by Leonard Max Adleman in the year 1994, for solving the complex problems such as directed Hamilton path problem, NP-complete problem similar to The Travelling Salesman problem[1].DNA stands for deoxyribonucleic acid. It consists of four chemical bases A(adenine), G(guanine), C(cytosine) and T(thymine). The sequence of the bases determines the characteristics of the organism. A always pairs with T and C with G to form base pairs. Each base has a sugar and phosphate molecule attached to it(together called a nucleotide). Nucleotides are arranged in a double helical structure. An encryption algorithm takes the information known as plaintext as input, performs a series of operations on it to generate a seemingly random sequence of bits, known as cipher text, which is totally unrelated to the input plaintext [2]. The process of converting the cipher text back to the plaintext is known as decryption. If the keys used in both these processes are mathematical inverses of each other, it is symmetric key encryption else asymmetric key encryption. Symmetric key encryption algorithms include mono-alphabetic and poly-alphabetic substitution ciphers; the latter is less susceptible to cryptanalysis attacks, both brute force and statistical analysis [3]. The former involves trying out all possible keys in the domain whilst the latter involves exploiting language characteristics. For instance, the probability of occurrence of each letter in the English alphabet is listed with 'e' having maximum probability, followed by't' then 'a'. Other types of attacks include known cipher text, chosen plaintext, chosen ciphertext and ciphertext only.

John Conway's game of life is essentially a series of rules which forms the backbone of a cellular automaton. It needs an initial configuration to start, how the cells multiply or die is solely controlled by the rules with generation of complex patterns. If one is provided with an initial and final pattern, there is no algorithm which exists to predict whether the final pattern will ever appear.

Our algorithm attempts to incorporate Conway's Game of Life rules during the encryption process to generate a sufficiently random output. This algorithm is to be a module which can be included in [1] or [3]. It hides individual letter frequencies. Change in one bit in cipher text prevents decryption to obtain original plaintext. Two plaintexts of same length using the same key but differing in only one bit generate completely random and different cipher texts. It includes randomizations as described in [4]. Section II, III describes the methodologies used, Section IV, V describes the algorithm to be implemented, Section VI highlights the results and Section VII includes conclusion and future scope.

II. LITERATURE SURVEY

A. DNA cryptography

DNA cryptography is one of the emerging trends in data security and the possible future of data storage. The usage of this biological phenomenon in fields of cryptography and steganography may give rise to unbreakable algorithms. Although there are many drawbacks of DNA cryptography, efforts are being made to solve these as it is believed that they offer more advantages than traditional cryptographic techniques.

The methodologies include bio-molecular structure usage, OTP, DNA chip technology, DNA fragmentation and PCR(Polymerase chain reaction). The input plaintext is converted to bits for example if the input is “A” then the binary string is 01000001. This binary data is encoded to a DNA strand as follows:

00 is converted to A. 01 is converted to C.
10 is converted to G. 11 is converted to T.

So the result is “CAAC” after encryption. DNA cryptography is based on DNA computing, whose main advantages are parallelism and reducing bulk of data as opposed to conventional silicon-based machines.

B. Game of Life

It was devised by the mathematician John Horton Conway in 1970. The working environment for the game of life is an infinite, two-dimensional grid of cells which can be either alive or dead. The initial pattern forms the seed of the system. Generations are created by applying the following rules simultaneously to every cell in the seed. Each cell interacts with its eight neighbors which are the cells which are horizontally, vertically or diagonally adjacent. The rules to apply are as follow:

- Any live cell with less than two live neighbors dies, as if by underpopulation.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The key is used to randomly populate the grid to serve as the seed. It is ensured that not too many cells are alive as then probability of most of the cells dying increases. The final pattern obtained by applying these rules is XOR-ed bitwise with the plaintext for introducing sufficient randomization.

III. GAME OF LIFE MATRIX GENERATION FOR RANDOMIZATION

To describe the method user for generation of the initial starting game of life matrix, let us assume the plain text=” AB”. Therefore, the number of plaintext bits=2*8=16=m.
Convert Plaintext to bits=0100000101000010
A square matrix of dimension n*n is initialized where n=sqrt(m=16). Therefore n=4.
Initially all the cells (each element of the matrix) is dead i.e.=0.
Depending upon the given key: A random number is generated using the random class of Java to generate the cells which will become alive (=1).

Since the initial matrix is a sparse matrix consisting mainly of 0s (dead cells), the number of cells which will become alive is an only a fraction of the total number of bits in the plaintext. This fraction is essentially between $1/3^{rd}$ and $1/5^{th}$. The first matrix generated is the initial state. Once the first matrix with live cells are generated, depending upon the rules of Conway’s Game of Life, subsequent matrices are generated with live and dead cells as specified.

IV. ALGORITHM FOR FUNCTION ENCRYPTION ():

Step-1: Start
Step-2: Input the Key
Step-3: Convert the bytes of the input file to bits
Step-4: Calculate the total number of bits in input text
Step-5: Initialize the Game of Life Matrix using the total number of bits in plaintext and generate the live cells using the given Key.
Step-6: Final Matrix is XORed bitwise the plaintext bits. If the number of plaintext bits is a perfect square, then all the bits are XORed. However, if that is not so, only the first n^2 of plaintext bits are XORed with the matrix (of dimension nXn). Remaining bits are unchanged.
Step-7: Store the XORed bits in a 2-dimensional array(nXn) and residual bits into a 1-dimensional array.
Step-9: Perform the following shifting operations on the 2-dimensional array.
Step-10: Perform bitwise left-shift //To shift all bits in each row by 1 unit on LHS
Step-11: Perform bitwise up-shift //To shift all bits in each column by 1 unit towards up
Step-12: Perform bitwise right-shift //To shift all bits in each row by 1 unit on RHS
Step-13: Perform Cyclic Operation on the matrix:
Clockwise operation on the outermost layer. Anti-Clockwise operation on the next layer and so on. (// To perform circular shift of bits anti clock wise and then clock-wise in alternate periphery of the square)

Step-14: Perform bitwise down-shift //To shift all bits in each column by 1 unit towards down.

Step-15: Repeat the Steps 10-14 ‘k’ number of times depending upon the random number generated as follows:
(i)Initialize counter to index of last character in bit string obtained in step 6. Initialize term to 1. Initialize sum to zero.
(ii)Check the bit present in the bit string obtained in step 6 at position whose value is given by the value in the counter. If this bit is zero, go to step (iv) otherwise proceed to step (iii).
(iii)Add the current value of term to the sum.
(iv)Update the term as term=term+2 and counter=counter-1
(v)If the value of counter is greater than or equal to zero go to step (ii)
(vi)The last two digits of the sum thus obtained gives the value of ‘k’.
Step-16: Store the final result in a 1-Dimensional array along with the residual bits.
Step-17: Take the first 8 bits from the 1-D array. These 8-bits are XORed with the each of next 8-Bits (Blocks) of the 1-D array. The resultant 8-bits replace each of the corresponding 8-bits with which the first 8-bits are XORed.
Step-18: Take the final 8 bits from the 1-D array. These 8-bits are XORed with the each of previous 8-Bits (Blocks) of the 1-D array. The resultant 8-bits replace each of the corresponding 8-bits with which the last 8-bits are XORed. Final 8-bits are shifted to be the First 8-bits of the array.
Step-19: Take the final 4 bits from the 1-D array. These4-bits are XORed with the each of previous 4-Bits (Blocks) of the 1-D array. The resultant 4-bits replace each of the corresponding 4-bits with which the last 4-bits are XORed.
Step-20: Take the first 4 bits from the 1-D array. These 4-bits are XORed with the each of next 4-Bits (Blocks) of the 1-D array. The resultant 4-bits replace each of the corresponding 4-bits with which the first 4-bits are XORed.First 4-bits are shifted to be the Final 4-bits of the array.
Note that after the first 4 bits are taken for XOR, the bit at 5^{th} position is skipped randomly and then the rest of the bits are divided into blocks of 4 bits and XOR-ed as described.
Step-21: Take 2 bits at a time and convert it into DNA sequences: 00 ->A, 01 ->C, 10->G, 11->T
Step-21(a): Randomize it using following shifting operations:
Leftshift(), Upshift(), Diagonalshift(), Cycling(), Rightshift(), Downshift ()
Step-22: Convert the DNA sequence into bits.
Step-23: Convert bits to bytes and store it into a file as cipher text. Store the value of ‘k’ as the last byte of the file.
Step 24: End

V. ALGORITHM FOR FUNCTION DECRYPTION()

Step-1: Start
Step-2: Input the Key
Step-3: Convert the bytes of input file to bits
Step-4: Take 2 bits at a time and convert it into DNA sequences: 00 ->A, 01 ->C, 10->G, 11->T
Step-5: Upshift (), Leftshift(), Cycling(), Diagonalshift(), Downshift (), Rightshift().
Step-6: Convert DNA sequence into bits.
Step-7: Take the first 4 bits from the 1-D array. These 4-bits are XORed with the each of next 4-Bits (Blocks) of the 1-D array. The resultant 4-bits replace each of the corresponding 4-bits with which the first 4-bits are XORed.First 4-bits are shifted to be the Final 4-bits of the array.
Note that prior to XOR, the bit in the first position is skipped and then the remaining bits are divided into blocks of 4 and XOR-ed as described.
Step-8: Take the final 4 bits from the 1-D array. These 4-bits are XORed with the each of previous 4-Bits (Blocks) of the 1-D array. The resultant 4-bits replace each of the corresponding 4-bits with which the last 4-bits are XORed.
Step-9: Take the final 8 bits from the 1-D array. These 8-bits are XORed with the each of previous 8-Bits (Blocks) of the 1-D array. The resultant 8-bits replace each of the corresponding 8-bits with which the last 8-bits are XORed. Final 8-bits are shifted to be the First 8-bits of the array.
Step-10: Take the first 8 bits from the 1-D array. These 8-bits are XORed with the each of next 8-Bits (Blocks) of the 1-D array. The resultant 8-bits replace each of the corresponding 8-bits with which the first 8-bits are XORed.
Step-11: Store the final result in a 1-Dimensional array along with the residual bits.
Step-12: Store the XORed bits in a 2-dimensional array(nXn) and residual bits into a 1-dimensional array.
Step-13: Perform the following shifting operations on the 2-dimensional array.
Step-14: Perform bitwise up-shift //To shift all bits in each column by 1 unit towards up.
Step-15: Perform Cyclic Operation on the matrix: Anti-Clockwise operation on the outermost layer. Clockwise operation on the next layer and so on. (// To perform circular shift of bits anti clock wise and then clock-wise in alternateperiphery of the square)

Step-16: Perform bitwise left-shift //To shift all bits in each row by 1 unit on LHS
Step-17: Perform bitwise down-shift //To shift all bits in each column by 1 unit towards down.
Step-18: Perform bitwise right-shift // To shift all bits in each row by 1 unit on RHS
Step-19: Repeat the Steps 14-18 ‘k’ number of times which is present as the last byte read from the file.
Step-20: Calculate the total number of bits in cipher text
Step-21: Initialize the Game of Life Matrix using the total number of bits in ciphertext and generate the live cells using the given Key.
Step-22: Final Matrix is XORed bitwise the ciphertext bits. If the number of ciphertext bits is a perfect square, then all the bits are XORed. However, if that is not so, only the first n^2 of ciphertext bits are XORed with the matrix (of dimension nXn). Remaining bits are unchanged.
Step-23: Store the XORed bits in a 2-dimensional array(nXn) and residual bits into a 1-dimensional array.
Step-24: Convert bits to bytes.

Step-25: Plaintext is obtained.
Step-26: End.

VI. RESULTS AND DISCUSSION

In the table given below some plain texts and the corresponding ASCII value of cipher text are shown. There are many instances where it was observed for the same key, almost similar plain texts, the cipher texts are totally different. So, without knowing the secret text-key and the actual decryption process it is quite impossible for the intruder to generate the plain text from the cipher text. The present algorithm can even encrypt ASCII 0, ASCII 1, and ASCII 255 which normally impossible in standard encryption methods like DES, RSA etc.

Table -1: Some Plain texts and ASCII code of Encrypted Texts

Plaintext	Key	ASCII Number of Ciphertext
16 ASCII ‘0’	10	4,68,68,68,71,200,70,20 1,244,99,81,132,108,204 ,210,82
16 ASCII ‘1’	10	0,16,184,20,211,226,34, 133,161,164,5,160,32,53 ,128,0
16 ASCII ‘2’	10	160,160,32,162,32,129,1 60,173,241,32,147,108,9 7,65,37,182
16 ASCII ‘255’	10	240,15,255,236,191,1,25 5,30,191,118,100,79,21, 175,255,170
8 ASCII ‘1’+8 ASCII ‘2’	10	2,6,162,40,2,86,6,6,98,1 62,103,162,62, 125,27,4
7 ASCII ‘1’+ 2 ASCII ‘0’+ 7 ASCII ‘2’	10	223,223,254,205,157,33, 222,62,141,215,76,127,5 3,143,221,170
ASCII ‘1’ + ASCII ‘2’ + ASCII ‘4’ +ASCII ‘8’	10	85,208,29,25

Table-2: Input characters vs. corresponding cipher text.

PLAIN TEXT	KEY	CIPHERTEXT (ASCII VALUE)	CIPHERTEXT (CHAR)
HE IS GOOD	10	16,12,177,100,118,85,36,166,10 0,63	±dvU$? d?
HE IS GOON	10	198,206,88,163,234,110,160,168 ,66,22	Æ?X£ên ?B
8 A + B + 8 A	10	22,55,182,180,155,120,3,188,23, 27,13,10,54,187,82,242,4	7???x¼ 6»Rò
16 A	10	190,58,142,11,126,81,16,172,10 6,12,112,123,126,249,166,5	?:? ~Q¬jp{ ~ù?

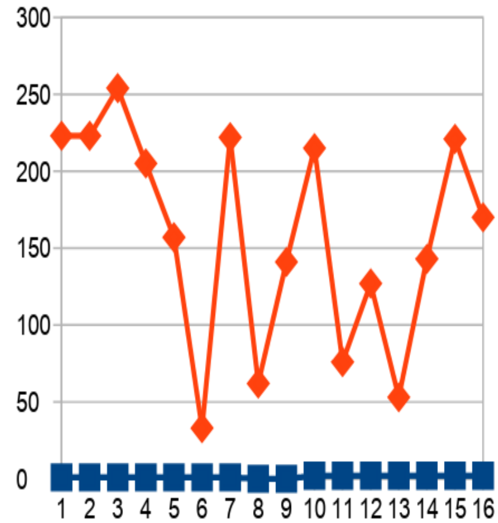

Fig 1: Relationship between Ciphertext and Plaintext (Plaintext:7 ASCII ‘1’+ 2 ASCII ‘0’+ 7 ASCII ‘2’)

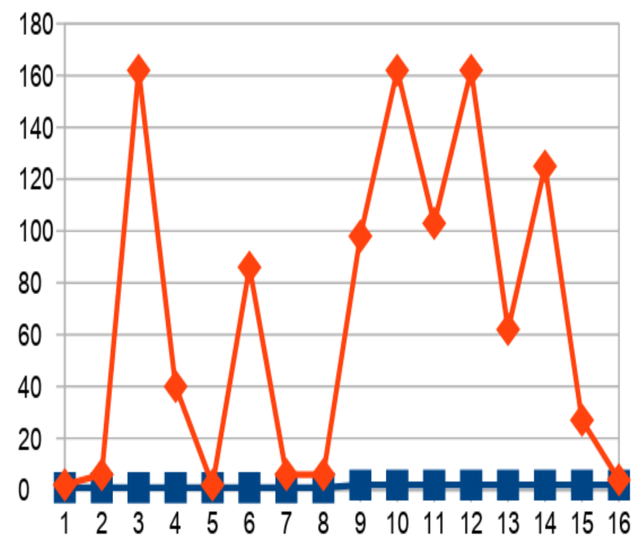

Fig 2: Relationship between Ciphertext and Plaintext (Plaintext:8ASCII ‘1’ + 8 ASCII ‘2’)

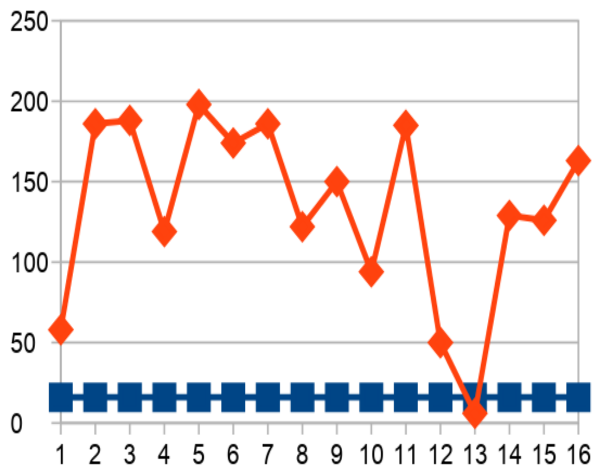

Fig 3: Relationship between Ciphertext and Plaintext (Plaintext:16ASCII '16')

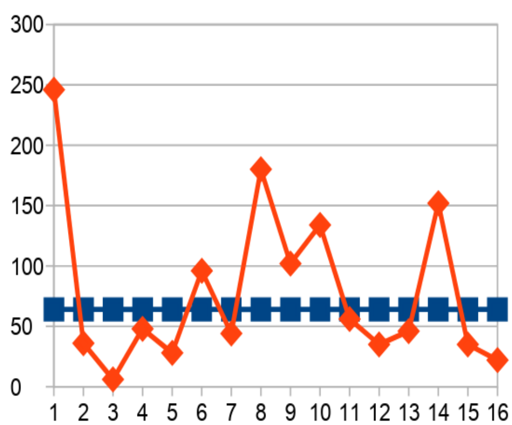

Fig 4: Relationship between Ciphertext and Plaintext (Plaintext:16ASCII '64')

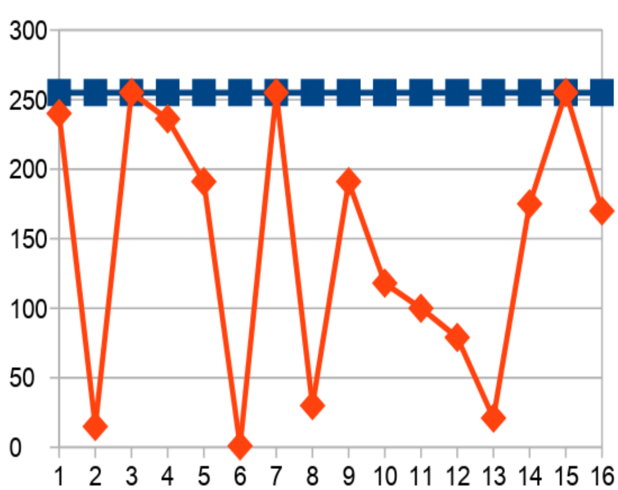

Fig 5: Relationship between Ciphertext and Plaintext (Plaintext:16ASCII '255')

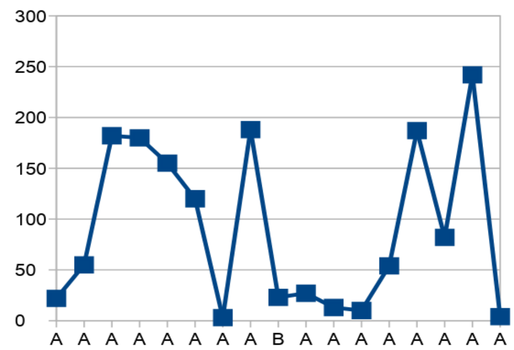

Fig 5: Relationship between Ciphertext and Plaintext (Plaintext:8 A + B + 8A)

In above table the results show that the cipher texts ate totally unpredictable even though the Plain texts contain some trivial patterns. The present method shows cipher texts always different even if input plain contains all characters same.In Figures 1 to 5 the encrypted data and also plain text data are shown. The results show that the Cipher texts patterns are totally unpredictable. The hackers will not be able apply any kind of brute force method to find Plain Text without knowing secret key. The present may be used to encrypt confidential message such as password, key etc.

VII. CONCLUSION AND FUTURE SCOPE

The present method applied on different files like .txt, .png, .jpg, .ddl, .exe etc. and results were quite satisfactory onany type of file. The user has to input some initial secret key for encryption and decryption. One cannot decrypt the encrypted text without knowing the initial secret key. Many standard method like left shift (), right shift (), downshift (),upshift(), cycling(),diagonal shift(),xor(), are applied in plain text in the bit level so if two plain texts differ slightly, the encrypted text differs huge and so it is free from any type of brute force attack. In addition to above functions, bits are XOR-ed with a randomly generated matrix (Game of Life) which makes the ciphertext even more difficult for decryption without proper key. To make this system further complex bitwise operations were used. Every application has its merits and demerits. The present method has covered almost all requirements. Further requirements and improvements can easily be done since coding is mainly structured or modular in nature. This method can be extended using DNA computing.

REFERENCES

[1]. Asoke Nath, Soumyadip Ray, Salil Anthony Dhara, Sourav Hazra *"3-DIMENSIONAL BIT LEVEL ENCRYPTION ALGORITHM VERSION-3 (3DBLEA-3)"* International Journal of Latest Trends

in Engineering and Technology Vol.(10)Issue(2), pp.347-353 May 2018

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

[3]. Asoke Nath, Ayan Ghosh, Enakshi Ghosh and Jayisha Saha, *"3-Dimensional Bit Level Encryption Algorithm Ver-2(3DBLEA-2)",* , International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE), Vol. 4, page: : 8611-8618 Issue 5, MAY 2017.

[4]. Asoke Nath, Saima Ghosh, MeheboobAlam Mallik, "Symmetric Key Cryptography using Random Key generator" Proceedings of International conference on security and management (SAM-10) held at Las Vegas, USA, July 12-15, 2010, Vol-2, Page: 239-244(2010).

AUTHORS PROFILE

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 etc. He has published more than **242** research articles in different Journals and conference proceedings.

Souptik Kumar Majumdar is currently pursuing his Bachelor's degree in Computer Science from St. Xavier's College(Autonomous), Kolkata, which will be completed by May 2019. He is adept at programming with frequent participation in coding competitions and internships to his credit.

Suchandra Datta is currently pursuing her Bachelor's degree in Computer Science from St. Xavier's College(Autonomous), Kolkata, which will be completed by May 2019. She is passionate about programming and areas of interest include cryptography and machine learning.

