International Journal of Innovative Research in Advanced Engineering (IJIRAE) ISSN: 2349-2763
Issue 12, Volume 2 (December 2015) www.ijirae.com

Noise Driven Encryption Algorithm (NDEA) Version-1

Aashijit Mukhopadhyay
St. Xavier's College (Autonomous) Kolkata

Somnath Saha
St. Xavier's College (Autonomous) Kolkata

Naved Ahmed Tagala
St. Xavier's College (Autonomous) Kolkata

Asoke Nath
St. Xavier's College (Autonomous) Kolkata

Abstract*— In the present paper the authors introduced an algorithm for encrypting useful data at bit level. The authors have used a new technique of introducing noise wave over the bit patterns of the plain text to encrypt it. Noise has been defined as the complement of the present bit pattern. Certain window layers have been selected over the bit patterns and noise wave has been made to propagate over those pattern in concentric circular fashion thus encrypting the whole bit text present. Windows are picked up from the bit patterns randomly and noise wave is made to propagate from any random position with a random intensity. Noise propagates inside the medium with decreased intensity as one move farther from the source of noise using natural laws. The method has been applied on some standard text and the output has been found to be totally unpredictable. The method can be used to encrypt OTP (One Time Password) and other bank transactions.

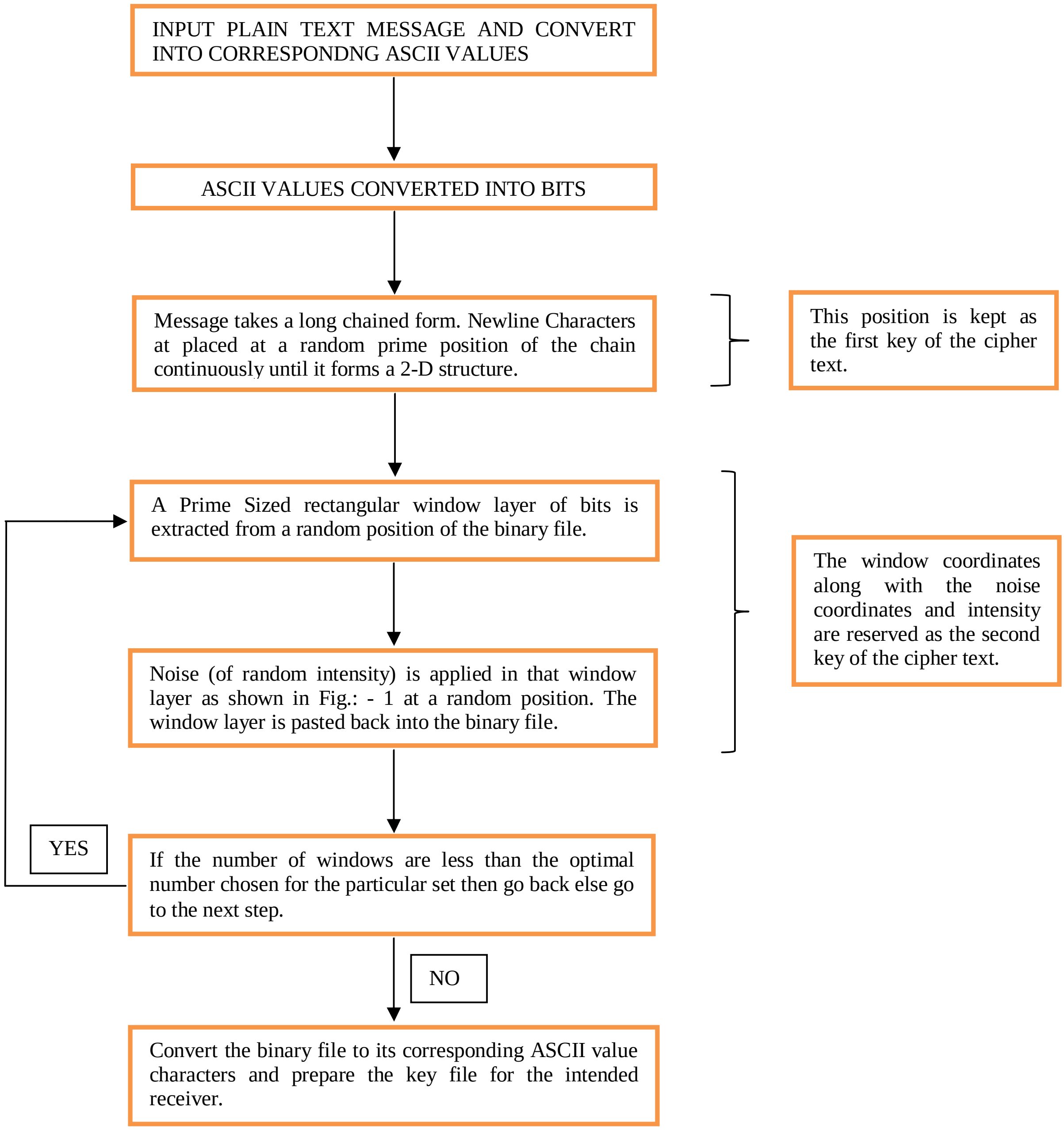
***Keywords*— Noise, Plain Text, Encryption, Decryption, AFES**

I. INTRODUCTION

With the advancement of internet technologies, information flow over the internet has increased manifold. Due to the extensive information flow, the security of each bit of the information becomes very vital. The main applications of secured data transfer are in money transfer applications. Online banking has been the most important sector of fund transfer. During fund transfer, vital information such as the pin of the user is present in the network for a considerable period of time. This information must be properly encrypted in order to secure this vital information from the hackers. The algorithm AFES-1 that has been recently proposed has been a great help for the logical birth of the present algorithm. In AFES-1, the Plain Text is converted to its corresponding bits and stored in a square matrix of size equal to the integral square root of the number of bits. The residual bits remain untouched. Then the bits are arranged by calling 24 different shifting functions. Now, the order of calling the 24 functions change at each iteration and that order is taken as a function of the keypad. The bits extracted from the above algorithm are then encrypted more using MWFES-3. This algorithm has been found to be very useful against any brute force attack. In the present paper, the authors have suggested a new technique where concentric propagation of noise over the useful data is used as the prime authority of the method. Now the medium through which the noise traverses becomes encrypted and can be revived back through a particular decryption technique involving reverse noise propagation. To make deciphering more complex, the authors introduce the concept of Windows in this technique. The plain text is given as input by the user. Each character of the text is converted into its respective ASCII value which in turn is converted into binary bits and stored in the RAM. Introducing a newline character in a particular position of the binary medium makes the medium rectangular in shape. This position is stored as the first key of decryption. Noise is not applied directly over the whole medium.

The medium is divided into prime sized windows whose co-ordinates are chosen randomly all over the binary medium. Each window layer is extracted from the medium and noise is applied at a randomly chosen position, at a randomly chosen intensity. Noise spreads along the surface layer of the medium affecting all the bits along its path in concentric circles only inside the medium with diminishing intensity. This window layer is pasted back in the particular position from where it was lifted. The authors keep the algorithm flexible in a way that the user can choose the number of windows he/she wants to use to encrypt his/her message. After the ongoing process ends according to the user's wish, the bits are converted back into their corresponding ASCII character and send to the intended receiver. The window layers and the randomly chosen position and intensities is stored as the second key of the system for decryption. The intended receiver converts the ASCII characters into its corresponding binary digits and store in an array. The receiver will then apply noise in a reverse manner in those window layers at the particular position and intensity sent as the second key by the user. After using up the windows as keys the intended receiver gets back the original binary file of the ASCII code that has been sent and in turn getting back the original text. This method is one of the strongest methods of encryption, as the key changes after each time and dual key system makes the method un-decipherable for brute forces.

II. GENERAL WORKING OF THE ALGORITHM



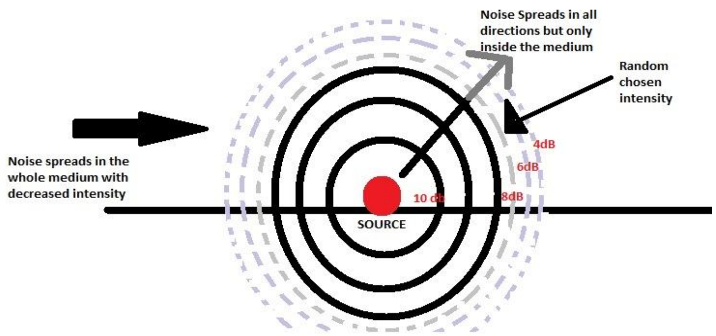


Fig: - 01 (Propagation of Noise through the medium)

IV. ALGORITHMS

In this section the algorithms for encryption and decryption are discussed.

IV.1 ALGORITHM FOR ENCRYPTION

Step 1: Accept the plain text in the form of a .txt file.
Step 2: Convert each character into its respective ASCII code and store in an array.
Step 3: Randomly choose a number, n such that n > (8*size of the plain text) and n = prime
Step 4: Convert each character into bits.
Step 5: Store it in an array a [] [] such that number of columns of a = n.
Step 6: Randomly generate four numbers n1, n2, n3, n4. //for the window coordinates
Step 7: Here (n3-n1) = prime and (n4-n2) = prime //prime sized window coordinates
Step 8: The origin of the window (x, y) = (n1, n2)
Step 9: if (n1+n3) > (width of a [] []) n3 = n1 – n3 //going the opposite direction to create window
Step 10: if (n2+n4) > (height of a [] []) n4=n2-n4 //the opposite vertical direction as well
Step 11: The end position of the window (x_end, y_end) = (n3, n4)
Step 12: Randomly generate to numbers n5, n6
Step 13: if n5 > (n3 – n1) || n6 > (n4 - n2) goto step-12
Step 14: Origin of noise (x_n, y_n) = (n5, n6)
Step 15: Randomly generate the last number n7
Step 16: if n7! = prime goto step- 15 //radius of noise intensity should be of prime magnitude
Step 17: Radius of noise to be generated r = n7
Step 18: Extract the bits inside the window
Step 19: Store the extracted bits in an array window [] []
Step 20: Let radius = 0, centre of propagation (h, k) = (x_n, y_n) // starting to propagate noise from the source
Step 21: while radius <= r
Step 22: Let r_tmp=radius, x = 0, gap=1 //noise to be propagated densely
Step 23: while x < (r_tmp/√2) //drawing $1/8^{th}$ of the circle
Step 24: y= √ (r^2 – x^2) //using circle draw equation
Step 25: complement bit (x+h, y+k) of a [] [] // generating the concentric circles
Step 26: complement bit (x-h, y+k) of a [] []
Step 27: complement bit (x+h, y-k) of a [] []
Step 28: complement bit (y+h, x+k) of a [] []
Step 29: complement bit (y-h, x+k) of a [] []
Step 30: complement bit (y+h, x-k) of a [] []
Step 31: complement bit (x-h, y-k) of a [] []
Step 32: complement bit (y-h, x-k) of a [] []
Step 33: x= x + gap
Step 34: end while //while loop for generating a single circle
Step 35: radius= radius + 1
Step 36: end while //while loop for generating the full intensity noise
Step 37: radius = r+1 //sound propagation inside the rest of the window layer
Step 38: while radius < (width of window [] [])
Step 39: r_tmp = radius, x= 0, gap = 2
Step 40: while x < r_tmp/√2 //drawing $1/8^{th}$ of the circle which propagates sparse noise
Step 41: y=√ (r^2 – x^ 2)
Step 42: complement bit (x+h, y+k) of a [] [] // generating the concentric circles
Step 43: complement bit (x-h, y+k) of a [] []
Step 44: complement bit (x+h, y-k) of a [] []
Step 45: complement bit (y+h, x+k) of a [] []
Step 46: complement bit (y-h, x+k) of a [] []
Step 47: complement bit (y+h, x-k) of a [] []
Step 48: complement bit (x-h, y-k) of a [] []
Step 49: complement bit (y-h, x-k) of a [] []
Step 50: x= x + gap
Step 51 end while // for generating each circle
Step 52: radius = radius + 1
Step 53: gap = gap + 1 // decrease the intensity continuously
Step 54: end while //for the overall generation
Step 55: for i=0 to number of rows in window, i=i+1

Step 56: for j=0 to number of columns in window, j=j+1
Step 57: a [n1+i] [n2+j] = window [i] [j] //paste the window back into the particular position
Step 58: Store n as the first key
Step 59: Store n1, n2, n3, n4, n5, n6, n7 as the second key file for decryption goto step 6 for the optimum number of times decided according to the size of the message.
Step 60: Convert bits of array a [] [] into bits of corresponding ASCII character
Step 61: Send encrypted text, key1, key2 to the receiver
Step 62: End

IV.2 ALGORITHM FOR DECRYPTION

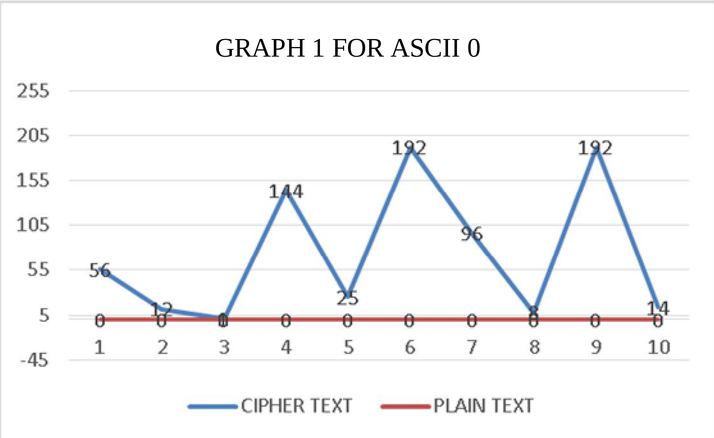
Step 1: Accept the encrypted file
Step 2: Extract the ASCII characters of the encrypted text file
Step 3: Convert the characters into bits
Step 4: Extract the value of the first key file and store in variable n
Step 5: Store the bits in array a [] [] such that number of columns of a [] [] = n
Step 6: Extract all the set of values in the second key file and insert them into a 1-d array key []
Step 7: Reverse key []
Step 8: for i=0 to (length of key []), i=i+1
Step 9: String str_tmp = key[i] //store the first set of keys
Step 10: Extract each values in the ith index of key []
Step 11: Store all the seven values in different variables n1, n2, n3, n4, n5, n6, n7
Step 12: Let window origin (win_x, win_y) = (n1, n2)
Step 13: Let window end position (end_x, end_y) = (n3, n4)
Step 14: Extract the window bits from array a [] []
Step 15: Store the bits in the array win [] [] //arranging the window layer to apply noise
Step 16: Let centre of propagation of noise (h, k) = (n5, n6)
Step 17: radius of intensity of noise r = n7
Step 18: Let radius = 0 // starting to propagate noise from the source
Step 19: while radius <= r
Step 20: Let r_tmp=radius, x = 0, gap=1 //noise to be propagated densely
Step 21: while x < (r_tmp/√2) //drawing 1/8th of the circle
Step 22: y= √ (r^2 – x^2) //using circle draw equation
Step 23: complement bit (x+h, y+k) of a [] [] // generating the concentric circles
Step 24: complement bit (x-h, y+k) of a [] []
Step 25: complement bit (x+h, y-k) of a [] []
Step 26: complement bit (y+h, x+k) of a [] []
Step 27: complement bit (y-h, x+k) of a [] []
Step 28: complement bit (y+h, x-k) of a [] []
Step 29: complement bit (x-h, y-k) of a [] []
Step 30: complement bit (y-h, x-k) of a [] []
Step 31: x= x + gap
Step 32: end while //while loop for generating a single circle
Step 33: radius= radius + 1
Step 34: end while //while loop for generating the full intensity noise
Step 35: radius = r+1 //sound propagation inside the rest of the window layer
Step 36: while radius < (width of window [] [])
Step 37: r_tmp = radius, x= 0, gap = 2
Step 38: while x < r_tmp/√2 //drawing 1/8th of the circle which propagates sparse noise
Step 39: y=√ (r^2 – x^ 2)
Step 40: complement bit (x+h, y+k) of a [] [] // generating the concentric circles
Step 41: complement bit (x-h, y+k) of a [] []
Step 42: complement bit (x+h, y-k) of a [] []
Step 43: complement bit (y+h, x+k) of a [] []
Step 44: complement bit (y-h, x+k) of a [] []
Step 45: complement bit (y+h, x-k) of a [] []
Step 46: complement bit (x-h, y-k) of a [] []
Step 47: complement bit (y-h, x-k) of a [] []
Step 48: x= x + gap
Step 49: end while // for generating each circle

Step 50: radius = radius + 1
Step 51: gap = gap + 1 // decrease the intensity continuously
Step 52: end while //for the overall generation
Step 53: for ii=0 to number of rows in window, ii=ii+1
Step 54: for j=0 to number of columns in window, j=j+1
Step 55: a [n1+ii] [n2+j] = window [ii] [j] //paste the window back into the particular position
Step 56: end for //for loop rotating the key array to decode the file
Step 57: Array a [] [] contains the bits of the original text
Step 58: Convert the bits into the corresponding ASCII characters
Step 59: End

V. RESULTS AND DISCUSSIONS

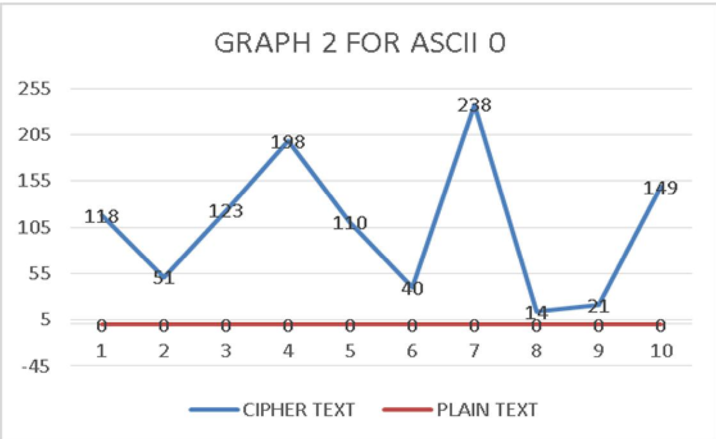
We have tested the proposed algorithm against a variety of test cases

TEST CASE 1: - ASCII 0 OF 10 CHARACTERS



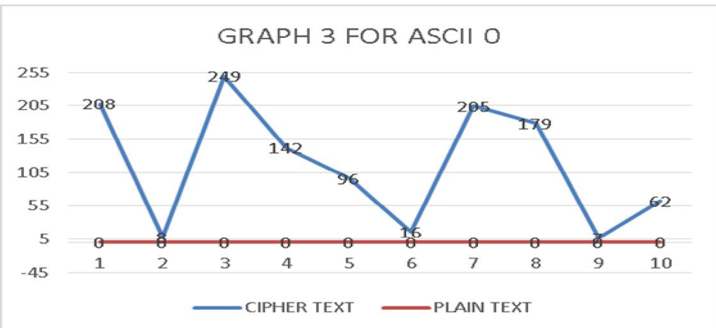

```
ORIGINAL ASCII values
0    0    0    0    0    0    0    0    0    0
ENCRYPTED ASCII values
56   12   1    144  25   192  96   8    192  14
```

Fig -02 (Graph between ASCII 0 Cipher Text after 1st Execution)



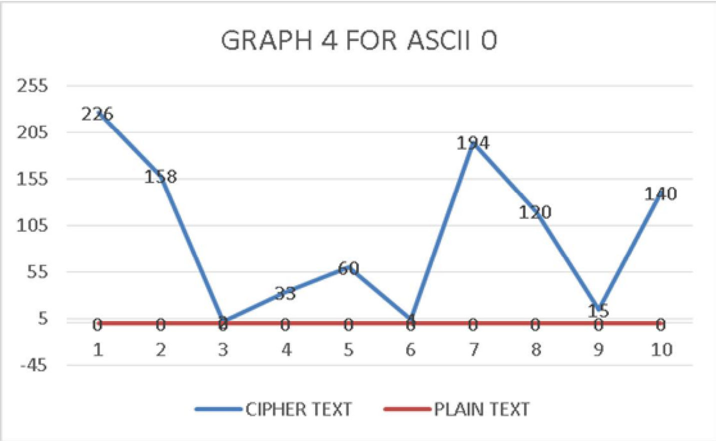

```
ORIGINAL ASCII values
0    0    0    0    0    0    0    0    0    0
ENCRYPTED ASCII values
118  51   123  198  110  40   238  14   21   149
```

Fig -03(Graph between ASCII 0 Cipher Text after 2nd Execution)




```
ORIGINAL ASCII values
0    0    0    0    0    0    0    0    0    0
ENCRYPTED ASCII values
204  8    249  142  96   16   205  179  7    62
```

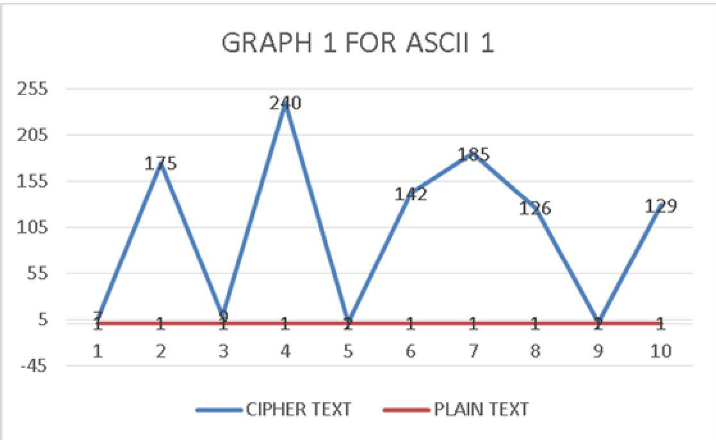
Fig -04 (Graph between ASCII 0 Cipher Text after 3rd Execution)




```
ORIGINAL ASCII values
0   0   0   0   0   0   0   0   0   0
ENCRYPTED ASCII values
226 158 2   33  60  4   194 120 15  140
```

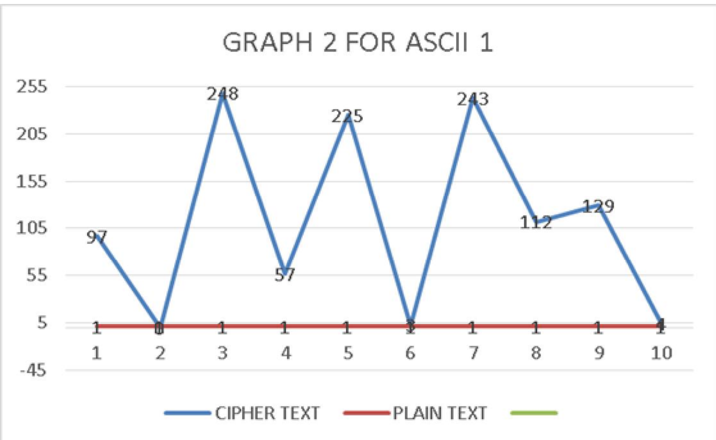
Fig -05 (Graph between ASCII 0 Cipher Text after 4th Execution)

TEST CASE 2: - ASCII 1 OF 10 CHARACTERS



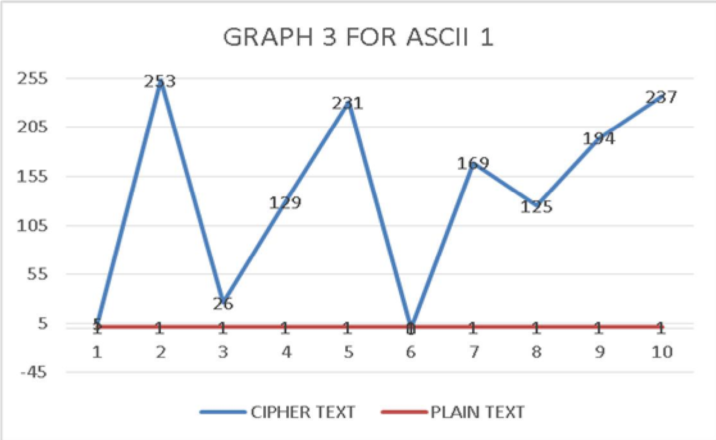

```
ORIGINAL ASCII values
1   1   1   1   1   1   1   1   1   1
ENCRYPTED ASCII values
7   175 9   240 2   142 185 126 2   129
```

Fig -06 (Graph between ASCII 1 Cipher Text after 1st Execution)



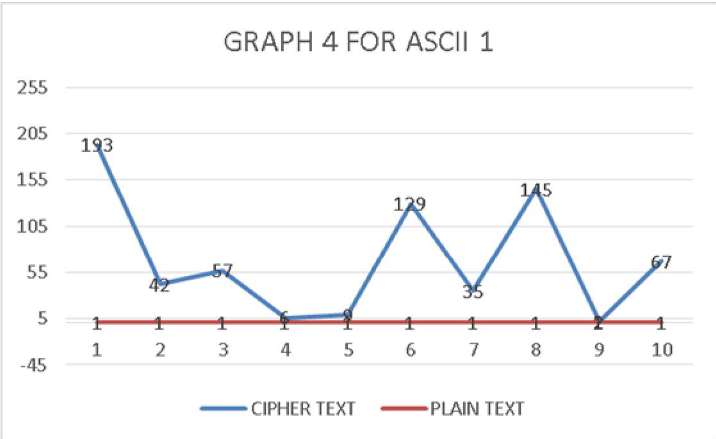

```
ORIGINAL ASCII values
1   1   1   1   1   1   1   1   1   1
ENCRYPTED ASCII values
97  0   248 57  225 3   243 112 129 4
```

Fig -07 (Graph between ASCII 1 Cipher Text after 2nd Execution)




```
ORIGINAL ASCII values
1   1   1   1   1   1   1   1   1   1
ENCRYPTED ASCII values
5   253 26  129 231 0   169 125 194 237
```

Fig -08 (Graph between ASCII 1 Cipher Text after 3rd Execution)

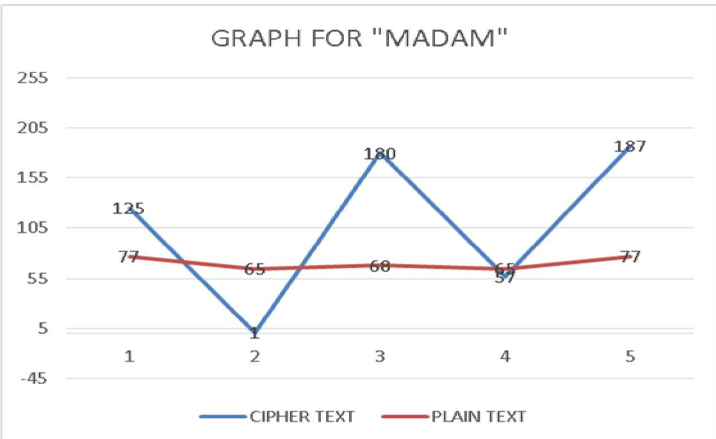



```
ORIGINAL ASCII values
1   1   1   1   1   1   1   1   1   1
ENCRYPTED ASCII values
193 42  57  6   9   129 35  145 2   67
```

Fig -09 (Graph between ASCII 1 Cipher Text after 4th Execution)

TEST CASE 3: - PLAINDROME WORD CHECK

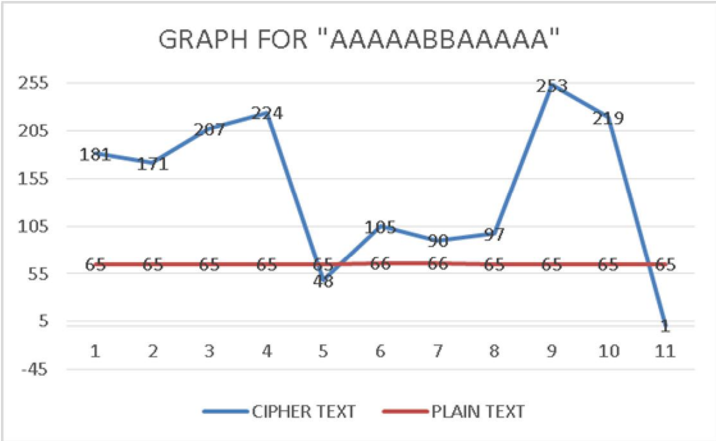
MESSAGE 1: - MADAM




```
ORIGINAL ASCII values
77  65  68  65  77
ENCRYPTED ASCII values
125 1   180 57  187
```

Fig -10 (Graph between palindromic word “MADAM” and its corresponding Cipher Text)

MESSAGE 2: - AAAAABBAAAAA




```
ORIGINAL ASCII values
65  65  65  65  65  66  66  65  65  65  65  65
ENCRYPTED ASCII values
181 171 207 224 48  105 90  5   97  253 219 1
```

Fig -11 (Graph between palindromic word "AAAAABBAAAAA" and its corresponding Cipher Text)

TEST CASE 4: - TWO SIMILAR LINES ARE TAKEN AS TEST CASES.

Graph between the two lines are shown as the result.
Line 1: - HE IS GOOD
Line 2: - HE IS GOON

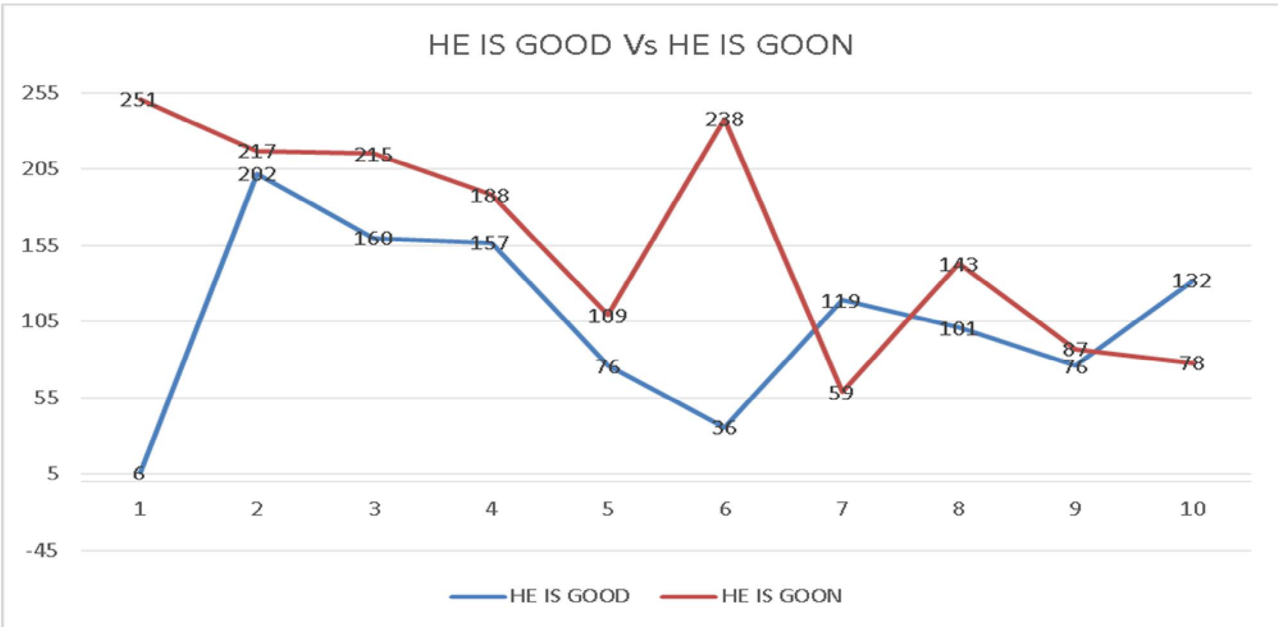


Fig -12 (Graph between two almost similar sentences "HE IS GOOD" and "HE IS GOON")

VI. CONCLUSIONS

The present algorithm used to encrypt data has been used for various files and it has yielded good results. Keeping the above produced results in mind, it is un-deniable that the algorithm is successful in producing random possibilities for same messages and intensely different possibilities even when same characters are given as input to the system. Thus, it is easy to conclude that the message that will be transmitted after flowing through this algorithm will be free from all brute force attacks, known plain text attack, known cipher text attack, statistical attack securing the most important information of the user while sending it to the intended receiver. The authors have thought of improvising new techniques in creation of windows and reducing the overlapping windows by discarding THEM IN the next versions of the work.

ACKNOWLEDGEMENT

The authors are grateful to Computer Science Department of St. Xavier's College (Autonomous), Kolkata for giving them opportunity to do research in Network Security.

REFERENCES

[1] Symmetric Key Cryptography using Random Key generator: Asoke Nath, Saima Ghosh, MeheboobAlamMallik: "Proceedings of International conference on security and management (SAM'10" held at Las Vegas, USA Jull 12-15, 2010), Vol-2, Page: 239-244(2010).
[2] A new Symmetric key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm, DriptoChatterjee, JoyshreeNath, SuvadeepDasgupta and AsokeNath : Proceedings of IEEE International Conference on Communication Systems and Network Technologies, held at SMVDU(Jammu) 03-06 June,2011, Page-89-94(2011).
[3] Symmetric key Cryptographic algorithm using combined bit manipulation and MSA encryption algorithm: NJJSAA symmetric key algorithm:NeerajKhanna, Joel James,JoyshreeNath, SayantanChakraborty, AmlanChakrabarti and AsokeNath : Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 03-06 June 2011, Page 125-130(2011).
[4] Symmetric key Cryptography using modified DJSSA symmetric key algorithm, DriptoChatterjee, JoyshreeNath, Sankar Das, ShalabhAgarwal and AsokeNath, Proceedings of International conference Worldcomp 2011 held at LasVegas 18-21 July 2011, Page-306-311, Vol-1(2011).
[5] Symmetric key Cryptography using two-way updated – Generalized Vernam Cipher method: TTSJA algorithm, International Journal of Computer Applications (IJCA, USA), Vol 42, No.1, March, Pg: 34 -39(2012).
[6] Ultra Encryption Standard(UES) Version-I: Symmetric Key Cryptosystem using generalized modified Vernam Cipher method, Permutation method and Columnar Transposition method, Satyaki Roy, NavajitMaitra, JoyshreeNath,ShalabhAgarwal and AsokeNath, Proceedings of IEEE sponsored National Conference on Recent Advances in Communication, Control and Computing Technology-RACCCT 2012, 29-30 March held at Surat, Page 81-88(2012)
[7] Advanced Digital Steganography using Encrypted Secret Message and Encrypted Embedded Cover File, Joyshree Nath, Saima Ghosh and Asoke Nath, International Journal of Computer Applications(IJCA 0975-8887), Vol 46, No-14, May ,(2012).
[8] Ultra Encryption Standard(UES) Version-II: Symmetric key Cryptosystem using generalized modified vernam method, permutation method, colum,nar transposition method and TTJSA method, Satyaki Roy, NavajitMoitra, Joyshree Nath, Shalabh Agarwal and Asoke Nath, Proceedings of International Conference Worldcomp 2012 held at Las Vegas, USA, FCS-12, Page-97 – 104(2012).
[9] Ultra Encryption Standard(UES) Version-IV: New Symmetric Key Cryptosystem with bit-level columnar Transposition and Reshuffling of Bits, Satyaki Roy, Navajit Maitra, Joyshree Nath, Shalabh Agarwal and Asoke Nath, International Journal of Computer Applications(IJCA)(0975-8887) USA Volume 51-No.1.,Aug, Page. 28-35(2012).
[10] Bit Level Encryption Standard (BLES) : Version-I, Neeraj Khanna, Dripto Chatterjee, Joyshree Nath and AsokeNath, International Journal of Computer Applications(IJCA)(0975-8887) USA Volume 52-No.2.,Aug, Page.41-46(2012).
[11] Bit Level Encryption Standard(BLES) : Versiob-II, GauravBhadra, Tanya Bala, Samaik Banik, Joyshree Nath and Asoke Nath, Proceedings of IEEE International Conference WICT-2012 held at IIITM-K, Trivandrum Oct 30 to Nov 1, 2012, Page No. 121- 127(2012).
[12] Modern Encryption Standard(MES) version-I : An Advanced Cryptographic Method, Somdip Dey, Asoke Nath, Proceedings of IEEE International Conference WICT- 2012 held at IIITM-K, Trivandrum Oct 30 to Nov 1, 2012, Page No. 242-247(2012).
[13] Asoke Nath, Bit Level Generalized Modified Vernam Cipher Method with Feedback, International Journal of Advanced Computer Research (ISSN(print):2249- 7277 ISSN(online): 2277-7970), Volume-2, Number-4 Issue-6, Page-24-30, Dec(2012).
[14] Bit Level Encryption Standard (BLES): Ver-III, Gaurav Bhadra, Tanya Bala, Samik Banik, Joyshree Nath, Asoke Nath, Proceedings of International Conference Worldcomp 2013 held at Las Vegas, USA in Jul 22-25, 2013. Proceedings page 99-105(2013).
[15] Advanced Symmetric Key Cryptosystem using Bit and Byte Level Encryption Methods with Feedback dvanced Symmetric Key Cryptosystem using Bit and Byte Level Encryption methods with Feedback, Prabal Banerjee, Asoke Nath, Proceedings of InInternational Conference Worldcomp 2013 held at Las Vegas, USA in Jul 22-25, 2013.Proceedings Page 120-126(2013).

[16] Multi Way Feedback Encryption Standard Ver-3(MWFES-3), Asoke Nath, Debdeep Basu, Ankita Bose, Saptarshi Chatterjee, Surajit Bhowmik published in IEEE conference proceedings: WICT-2013 held at Hanoi in Dec 14-18(2013), page 318-325(2013).
[17] Multi Way Feedback Encryption Standard Ver-2(MWFES-2), Asoke Nath, Debdeep Basu, Ankita Bose, Saptarshi Chatterjee, Surojit Bhowmik , International Journal of Advanced Computer Research(IJACR), Vol 3, Number-1, Issue-13, Page-29-35, Dec(2013).
[18] Ankita Basu, Debdeep Basu, Saptarshi Chatterjee, Asoke Nath, Surajit Bhowmik, Bit Level Multi Way Feedback Encryption Standard Ver-1(BLMWFES-1), published in Proceedings of IEEE conference CSNT-2014 held at Bhopal, page-601-605, April 7(2014).page-793-799, April 7(2014).
[19] Asoke Nath , Bit level Multi Way Feedback Encryption Standard Ver- 2(BLMWFES-2) , proceedings of International IEEE conference Advanced Communication, Control & Computing Technologies(ICACCCT) 2014 held at Syed Ammal Engineering College, Page 1702-1707(2014).
[20] Arijit Ghosh, PrabhakarChakraborty, AsokeNath, ShamindraParui, ―3d Multi Way Feedback Encryption Standard Version I(3dMWFES-1), International Journal of Advance Research in Computer Science and Management Studies, ISSN:2321-7782(Online), Vol 2, Issue 10,Oct, Page:206-218(2014).
[21] Arijit Ghosh, Prabhakar Chakraborty, Asoke Nath, “3d Multi Way Feedback Encryption Standard Version II(3dMWFES-II)”, International Journal of Computer Science and Information Technologies(IJCSIT), Vol.6(3), Page 2990-2997(June 2015).
[22] Asoke Nath, Ranjini Mukherjee, Dona Sarkar, Chaitali Patra, “2-Dimensional Multi Way Feedback Encryption Standard Version-1(2dMWFES-1)”, International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE), Vol-3, Issue 6, Page 5024-5033(30-th June 2015).