International Journal of Advanced Technology in Engineering and Science
Vol. No.3, Issue 11, November 2015
www.ijates.com
ijates
ISSN 2348 - 7550

SECURED DATA ENCRYPTION STANDARD (SDES-1) VERSION-I

Ayan Roy[1], Asoke Nath[2]

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

ABSTRACT

In the present paper the authors introduced a new symmetric key encryption method called Secured Data Encryption Standard (SDES) where for the first time the authors introduced something called noise in bit level. In SDES the plain text is to be converted to bits. Using random function one has to find the character position where the initial noise is to be introduced. In the present version the authors defined noise as the complement of the old bit. This noise is to be propagated in forward and in backward direction as well. However, the intensity of noise will be less slowly when it moves forward or in backward direction. The noise may be applied to any other character position and again it may be propagated in both forward and backward direction with less intensity. The present encryption method applied on some standard text and the output found totally unpredictable. The present method may be applied to encrypt password, sms or any other confidential data.

Keywords*: *Cipher Text, Data Encryption, Noise, Random Function, Secured Data

I. INTRODUCTION

Tremendous development in technology over the years has made it extremely necessary to encrypt the data that is in transit over a public network across the computers separated by distances. As the internet is accessible and open to any possible user, it becomes extremely important to emphasize on the security, integrity as well as the confidentiality of the data which is in transit over a network. It is extremely important to create a cipher text for a given plain text for transmission. However, the main intension of an attacker would be to break the cipher text and gain unauthorized access to the information in transit. It is therefore the job of a cryptographer to come up with certain techniques which would prevent the attacker from getting unauthorized access to the data. Nath et al had recently proposed MWFES-1[1], Modified MWFES-1[2], MWFES-2[3], Modified MWFES-2[4], MWFES-3[5], AFES-1[25].

In MWFES-1, the plain text character is added with the corresponding key character, the forward feedback and backward feedback and then the total sum (modulo 256) is taken as the corresponding cipher text character. The cipher text character is taken as the forward feedback value for the next byte (in case of forward operation) or backward feedback value for the previous byte(in case of backward operation). Forward and Backward operations are carried out on all the bytes starting from their respective ends.

In MWFES-2, the process is a little more general. Instead of propagating the feedback to the next byte (in case of forward feedback) and to the previous byte (in case of backward feedback) the feedback is propagated to the n^{th} byte where n is the 'skip factor'. In MWFES-2 the forward skip is kept equal to the backward skip (equal to n) and the initial forward feedback value and the backward feedback value was kept 0.

In MWFES-3, the authors introduced several changes in the algorithm. The plain text is broken into blocks and the encryption method is applied on each block separately. Each block has different Forward Skip (FS), Backward Skip (BS), initial Forward Feedback (FF) and initial Backward Feedback (BF) which are determined from the keypad counterpart of the block. These four important variables would decide the nature of the cipher text. The block size is different in every round of processing, causing these four important variables to change in every round. The total number of rounds (encryption no), and the block size value were also taken as a function of the key.

In AFES-1[25], 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. After this is done the authors convert the bits back to bytes and then apply MWFES-3[5] on those bytes. This entire process happens encryption_no (EN) times. Thorough tests were conducted on some standard plain text files and it was found that it is absolutely impossible for any intruder to extract any plain text from the generated encrypted text using any brute force method. The results show that the present method is free from any kind of plain text attack or differential attack.

In the present method, Secured Data Encryption Standard (SDES), the authors have introduced a new data encryption technique which extracts the bits of a file and represent them in layers. The proposed technique by the authors includes introducing noises at any random positions to generate the cipher text. The proposed method extracts the bits from a file and represents the bits in the multiple number of layers of a surface. Noise is introduced at any randomly generated position and it propagates through the entire surface. However, the algorithm devised by the authors is flexible, as it keeps the provision of introducing noises as many times desired by the user. After introducing the noises notable changes are observed and the bits are converted back to their ASCII character and sent as a cipher text to the intended receiver. The layers where the noises are introduced is the key for the entire communication. The indeed receiver on receiving the cipher text will convert the ASCII characters into its bits and store in the array. The intended receiver will introduce noises on those layers sent as a key in a reverse manner. The noises will be introduced for the same number of times introduced by the sender. After exhausting the layers indicated by the key, the receiver will convert the bits back to it's ASCII character to get the original text This method is an extremely strong method, the key changes for each communication and results in multiple cipher text for any given plain text at different communication sessions.

II. ALGORITHMS

In this section the algorithms used for encryption and decryption is discussed.

2.1 Algorithm for Encryption

Step-1: Accept the file.

Step-2: Estimate the size of the file and store it in a variable name 'size'.

Step-3:size1= (8*size)/8; // stores the number of layers

Step-4: Initialise 2 arrays a, b of dimension size1 X 2 X 4, decryp[100],index=0.

Step-5: Extract the bytes of the file.

ijates
ISSN 2348 - 7550

Step-6: Convert it into its bits.

Step-7: Store it in array a [][][].

Step-8: Copy the elements of array a[][][] to array b[][][]

Step-9: Randomly generate a number and store in a variable i1. //noise layer

Step-10: if i1>=size1, i1=i1-(size1/2). // noise adjustment.

Step-11: decryp[index]=i1. // stores the noisy layers as the key

Step-12: index++.

Step-13: m=i1, n1=1.

Step-14: for j11=0 to 2, j11++

Step-15: for k11=0 to 4, k11=k11+n1

Step-16: complement a[i1][j11][k11] and store it in b[i1][j11][k11].

Step-17: end for.

Step-18: end for. // noise is introduced in the backward direction

Step-19: i1--, n1++.

Step-20: if i1>0, goto step-14.

Step-21: n1=2, i11=m+1.

Step-22: for j11=0 to 2, j11++

Step-23: for k11=0 to 4, k11=k11+n1

Step-24: complement a[i11][j11][k11] and store it in b[i11][j11][k11].

Step-25: end for

Step-26: end for//noise is introduced in the forward direction

Step-27: i11++, n1++.

Step-28: i11<size1, goto step-22

Step-29: Copy the elements of array b[][][] to array a[][][]

Step-30: If sender wants to introduce more noise goto step-9

Step-31: Convert the bits of array a into corresponding ASCII characters.

Step-32: Send, decryp[],index,encrypted text,file size as key to the receiver.

Step-33: End

2.2 Algorithm for decryption

Step-1: Store the number of times noise is introduced, in a variable ind.

Step-2: Extract the characters from the encrypted text.

Step-3: Convert it into its bits

Step-4: index=ind., decrypt[] array stores the layers where noises are introduced.

Step-5: size1 stores the number of layers on the surface.

Step-6: Initialise an array b of dimension size1 X 2 X 4.

Step-7: Array a[][][] stores the bits of the encrypted text.

Step-8: Copy the elements of array a[][][] to array b[][][]

Step-9: i1=decrypt [index].

Step-10: m=i1, n=1.

Step-11: for j11=0 to 2, j11++

Step-12: for k11=0 to 4, k11=k11+n1

Step-13: complement a[i1][j11][k11] and store it in b[i1][j11][k11].

Step-14: end for.

Step-15: end for. // backward removal of noise from the encrypted text

Step-16: i1--, n1++.

Step-17: if i1>0, goto step-11

Step-18: n1=2, i11=m+1.

Step-19: for j11=0 to 2, j11++

Step-20: for k11=0 to 4, k11=k11+n1

Step-21: complement a[i11][j11][k11] and store it in b[i11][j11][k11].

Step-22: end for

Step-23: end for // forward removal of nise from the encrypted text

Step-24: i11++, n1++.

Step-25: i11<size1, goto step-19

Step-26: Copy the elements of array b[][][] to array a[][][]

Step-27: index--

Step-28: if index>=0 goto step-9.

Step-29: Array a contains the bits of the original plain text.

Step-30: Convert the bits into its corresponding ASCII character

Step-31: End

III. RESULTS AND DISCUSSIONS

We have tested the program against a variety of test cases.

In the following tables, we have shown the various cipher texts produced for the same plain text introducing noises different number of times.

Table-1: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **AAAAAAA**	**1**	
	2	
	3	
	4	

Table-2: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **BBBBBBB**	1	
	2	
	3	
	4	

Table-3: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **AAAAAAB**	1	
	2	
	3	
	4	

Table-4: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **BBBBBBA**	1	
	2	
	3	
	4	

Table-5: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **ABBBBBB**	**1**	
	2	
	3	
	4	

Table-6: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **BAAAAAA**	**1**	
	2	
	3	
	4	

It has been seen that it generated different cipher text for the same plain texts on different occasions. Therefore, it is difficult for an attacker to gain unauthorized access to the plain text without knowing the key. The cipher text on its own bears no meaning without the key. We have also observed that our algorithm could also encrypt ASCII-0 using the same method. The table below represents the encrypted texts for ASCII-0.

Table-7: Cipher texts vs. Number of times noise introduced

	Number of times Noise introduced	Corresponding Cipher text
Plain text :- **ASCII-0 (used 5 times)**	**1**	
	2	
	3	
	4	

IV. CONCLUSION AND FUTURE SCOPE

The present method is tested for various types of files and it has been able to deliver a satisfactory result. For a given communication, the encryption and decryption process worked smoothly. The result reveals that the process is successful in generating more than 1 cipher text for a given plain text. This method is free from any sort of Brute Force attack, Statistical attack, Known Plain text attack, Known cipher text attack and does not possess any threat to confidentiality. Such a technique can be used effectively for any android applications and any other fields as the process is simple, flexible yet efficient. The authors further aim to work on the present version and extend the present research to a more specific detail, dealing with specific point of a layer rather than the entire layer.

V. ACKNOWLEDGEMENT

The authors are grateful to Department of Computer Science for giving opportunity to do research work in Network Security. One of the authors AN is grateful to Fr. Dr. J. Felix Raj, Principal of St. Xavier's College, Kolkata for giving all time encouragement to do research work in the department.

REFERENCES

[1] Purnendu Mukherjee, Prabal Banerjee, Asoke Nath, Multi Way Feedback Encryption Standard Ver-I(MWFES-I) , International Journal of Advanced Computer Research(IJACR), Volume-3, Number-3, Issue-11, September 2013, Pages:176-182.

[2] Asoke Nath, Debdeep Basu, Surajit Bhowmik, Ankita Bose and Saptarshi Chatterjee, Multi Way Feedback Encryption Standard Ver-2(MWFES-2),. Paper submitted in International Conference : IEEE WICT 2013 to be held in December 15-18, 2013 at Hanoi, Vietnam.

[3] Asoke Nath, Payel Pal, Modern Encryption Standard Ver-IV(MES-IV), International Journal of Advanced Computer Research(IJACR), Volume-3, Number-3, Issue-11, September 2013, Page:216-223.

[4] Asoke Nath, Bidhusundar Samanta, Modern Encryption Standard Ver-V(MES-V), International Journal of Advanced Computer Research(IJACR), Volume-3, Number-3, Issue-11, September 2013, Pages:257-264.

[5] AsokeNath, Saima Ghosh, Symmetric Key Cryptography using Random Key generator: MeheboobAlamMallik:"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).

[6] DriptoChatterjee, JoyshreeNath, SoumitraMondal, SuvadeepDasgupta and AsokeNath, Advanced Symmetric key Cryptography using extended MSA method: DJSSA symmetric key algorithm: Journal of Computing, Vol 3, Issue-2, Page 66-71,Feb(2011).

[7] DriptoChatterjee, JoyshreeNath, SuvadeepDasgupta and AsokeNath, A new Symmetric key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm,: Proceedings of IEEE International Conference on Communication Systems and Network Technologies, held at SMVDU(Jammu) 03-06 June,2011, Page-89-94(2011).

[8] NeerajKhanna, JoelJames,JoyshreeNath, SayantanChakraborty, AmlanChakrabarti and AsokeNath, New Symmetric key Cryptographic algorithm using combined bit manipulation and MSA encryption algorithm: NJJSAA symmetric key algorithm: Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 03-06 June 2011, Page 125-130(2011).

[9] DriptoChatterjee, JoyshreeNath, Sankar Das, ShalabhAgarwal and AsokeNath, Symmetric key Cryptography using modified DJSSA symmetric key algorithm, Proceedings of International conference Worldcomp 2011 held at Las Vegas 18-21 July 2011, Page-306-311, Vol-1(2011).

[10] Debanjan Das, JoyshreeNath, Megholova Mukherjee, NehaChaudhury and AsokeNath, An Integrated symmetric key cryptography algorithm using generalized vernam cipher method and DJSA method: DJMNA symmetric key algorithm:: Proceedings of IEEE International conference: World Congress WICT-2011 held at Mumbai University 11-14 Dec, 2011, Page No.1203-1208(2011).

[11] Trisha Chatterjee, Tamodeep Das, JoyshreeNath, ShayanDey and AsokeNath, Symmetric key cryptosystem using combined cryptographic algorithms- generalized modified vernam cipher method, MSA method and NJJSAA method: TTJSA algorithm –, Proceedings of IEEE International conference: World Congress WICT-2011 t held at Mumbai University 11-14 Dec, 2011, Page No. 1179-1184(2011).