An Advanced Combined Symmetric Key Cryptographic Method using Bit Manipulation, Bit Reversal, Modified Caesar Cipher (SD-REE), DJSA method, TTJSA method: SJA-I Algorithm

Somdip Dey
St. Xavier's College
[Autonomous]
Kolkata, India.

Joyshree Nath
A.K.Chaudhuri School of IT,
Calcutta University
Kolkata, India.

Asoke Nath
St. Xavier's College
[Autonomous]
Kolkata, India.

ABSTRACT
In this paper, the authors propose a new combined symmetric key cryptographic method, SJA-I, which basically has four steps: Firstly, each byte is broken into its equivalent binary format and then single bit manipulation is executed on that; secondly Modified Caesar Cipher technique (SD-REE) and TTJSA cipher algorithm are applied on the data (message) randomly, which depends on the key provided during the time for encryption; thirdly DJSA method applied and in the final stage, Bit Reversal technique is applied to reach the final encrypted form of the original data. The method applied here is being tested on different plain text files and the results were analyzed very carefully. It was found that there was no pattern in the encrypted text and this combined cipher technique can not be broken by usual cryptanalysis attack like, differential attack, plain text attack, spectral analysis (frequency analysis), etc. The authors proposes that SJA-I can be applied for encryption of short message, password, secret key or any type confidential message.

General Terms
Information Security; Modified Caesar Cipher in form of SD-REE encryption method; TTJSA algorithm, which is a combination of Modified Vernam Cipher with feedback + MSA algorithm + NJJSA algorithm; DJSA algorithm, which is a modified form of MSA.

Keywords
Caesar Cipher, TTJSA, MSA, NJJSA, UES, DJMNA, SD-REE, SD-AREE, DJSA, Cryptography;

1. INTRODUCTION
In modern world, security is a big issue and securing important data is very essential, so that the data can not be intercepted or misused for illegal purposes. For example in Internet Banking system, e-reservation system the security of data is a very important issue. Under no circumstances the intruder should be able to get into the server database or the confidential data. In any type of service sectors the confidentiality of data is a very important issue. The primary goal of any system is that the data can not be modified by any external user or intruder. So, different cryptographic methods are used by different organizations and government institutions to protect their data online. But, cryptography hackers are always trying to break the cryptographic methods or retrieve keys by different means. For this reason cryptographers are always trying to invent different new cryptographic method to keep the data safe as far as possible.

Symmetric key algorithms are well accepted in the modern communication network. The main advantage of symmetric key cryptography is that the key management is very simple. Only one key is used for both encryption as well as for decryption purpose. There are many methods of implementing symmetric key. In case of symmetric key method, the key should never be revealed / disclosed to the outside world or to other user and should be kept secure. The key should be known to sender and the receiver only and no one else.

Our present work, SJA-I is also symmetric key cryptographic method, which is basically based on single bit manipulation technique; generalized modified Caesar Cipher method (SD-REE) [1] [14] [15]; TTJSA [2], which itself is based on generalized modified Vernam Cipher [2], MSA [3] and NJJSA [4]; DJSA method which is the extended version of MSA and bit reversal technique. Depending on the key entered by the user the functions of generalized modified Caesar Cipher and TTJSA are called randomly and then executed. The present method uses multiple times encryption to encrypt the plain text data.

2. SJA-I CIPHER METHOD
In this method, the authors initially apply *single bit manipulation* technique, where each byte is converted to its binary format i.e. in 0s and 1s format. In this step first, the bits are first shifted to its left and then reversed, which are equivalent to the length of the password provided, and secondly, the last 2 bits of the extreme ends are swapped with each other. In second stage, a *modified form of advanced Caesar Cipher (SD-REE)* [1] [14] [15] cryptographic method is applied. In cryptography, a Caesar cipher, also known as a Caesar's cipher or the shift cipher or Caesar's code or Caesar shift, is one of the simplest and basic known encryption techniques. It is a type of replacement cipher in which each letter in the plaintext is replaced by a letter with a fixed position separated by a numerical value used as a "key". But, in this method, SJA-I, any character (ASCII value 0-255) are not separated by a fixed numerical value, in fact it is a

variable numerical value, which is dependent on a non-linear polynomial function.

Along with the second stage, *TTJSA algorithm* [2] is also applied, which is basically a combined cryptographic method of modified Vernam Cipher with feedback, so that there can be existence of no pattern for different texts (messages), MSA [3] and NJJSA [4] algorithms. MSA and NJJSA algorithms are basically randomization cipher techniques, which are applied on blocks of data. Thus, MSA and NJJSA are block cipher techniques.

In the third stage, *DJSA technique*, which is a modified extended form of MSA, is applied on the encrypted file, which is an output of the previous stages.

And in the fourth and final stage, *bit reversal technique* is applied, where all the bytes are again converted to its binary format and then the whole bit sequence is reversed.

This present method is achieved by executing following three stages:

Stage 1: Single Bit Manipulation
Stage 2: In Random manner:
(i) Encrypt the data using generalized modified Caesar Cipher (SD-REE) method
(ii) Encrypt data using TTJSA method

Stage 3: DJSA method
Stage 4: Bit Reversal

In this present method, SJA-I, the user enters a secret key called as password and from that key we generate unique codes, which are successively used to encrypt the message. For decryption purpose we use just reverse process to get back the original plain text. During decryption the user has to enter the same secret key otherwise the decryption will not be successful. Now, we will describe in details the encryption procedure.

2.1. Single Bit Manipulation

In this stage, we convert each byte / character of the message to be encrypted, to its binary equivalent. Now, length of password is considered for bit left shift. i.e., Number of bits to be shifted to left will be decided by the length of password. Let L be the length of the password and L_R be the number of bits to be rotated to left and reversed (i.e. L_R is the effective length of password). The relation between L and L_R is represented by equation (1).

$L_R = L \bmod 7$ ------ eq. (1)

,where '7' is the number of iterations required to reverse entire input byte.

After this, the last two extreme positions of the bits are swapped with each other to generate the final output character, i.e. if the bit format is like $[B_8B_7B_6B_5B_4B_3B_2B_1]$ for input byte then B_8 will be swapped with B_1 and B_7 will be swapped with B_2. Thus, the binary equivalent of the output byte will become: $[B_1B_2B_6B_5B_4B_3B_7B_8]$ after swapping of bits.

For example,

Let Ch_{in} be any random character / byte from the message. Then its binary equivalent will be: $[B_8B_7B_6B_5B_4B_3B_2B_1]$. If the password provided for encryption is "sandi", then L_R=5 and the bits will be shifted by 5 positions to their left. Thus CH_{in} will become $[B_3B_2B_1B_8B_7B_6B_5B_4]$ after left shift and then $[B_3B_2B_1B_4B_5B_6B_7B_8]$ after reversing. Then, according to the algorithm the bits of extreme two sides are swapped with each other at a time. Thus, the resultant binary format is $[B_8B_7B_1B_4B_5B_6B_2B_3]$.

C_{in} → $[B_8B_7B_6B_5B_4B_3B_2B_1]$

$[B_8B_7B_1B_4B_5B_6B_2B_3]$ → C_{out}

2.2. Modified Caesar Cipher (SD-REE)

Dey et al proposed a new technique of modified Caesar Cipher [14][15], SD-REE, which is dependent on the key provided for encryption. From the key, we generate two unique numbers called 'code' and 'power_ex', and these numbers are used for the encryption purpose. This modified Caesar Cipher is based on a non-linear polynomial function and so, for this reason, every time it generates unique values, which will be added to each byte, i.e. each byte are shifted a different place every time using this modified Caesar Cipher (SD-REE).

2.2.1. Generation of Code and power_ex from the Secret Key:

The key is provided by the user in a string format and let the string be 'pwd[]'. From the given key we generate two numbers: 'code' and 'power_ex', which will be used for encrypting the message. First we generate the 'code' from the pass key.

Generation of code is as follows:

To generate the code, the ASCII value of each character of the key is multiplied with the string-length of the key and with 2^i, where 'i' is the position of the character in the key, starting from position '0' as the starting position. Then we sum up the resultant values of each character, which we got from multiplying, and then each digit of the resultant sum are added to form the 'pseudo_code'. Then we generate the code from the pseudo_code by doing modular operation of pseudo_code by 16, i.e.

code = Mod(pseudo_code, 16).

If code=0, then we set code =pseudo_code

Generation of the 'power_ex' is calculated as follows:

We generate power_ex from the pseudo_code generated from the above method. We add all the digits of the pseudo_code

and assign it as temporary_power_ex. Then we do modular operation on temporary_power_ex with code and save the resultant as power_ex. i.e.

power_ex =Mod (temporary_power_ex, code)

If power_ex = 0 OR power_ex = 1, then we set power_ex = code.

2.2.2. Encrypting the Message using 'code' and 'power_ex':

Now, we use the code and power_ex, generated from the key, to encrypt the main text (message). We extract the ASCII value of each character of the text (message to be encrypted) and add the code with the ASCII value of each character. Then with the resultant value of each character we add the (power_ex)^i, where i is the position of each character in the string, starting from '0' as the starting position and goes up to n, where n=position of end character of the message to be encrypted, and if position = 0, then (power_ex)^i = 0.

It can be given by the formula:

text[i] = text[i] + code + (power_ex)^i

If text[i] > 255 then, text[i] = Mod(text[i],256) : 'i' is the position of each character in the text and text[] is the message to be encrypted, where text[i] denotes each character of the text[] at position 'i'.

2.2.3. Algorithm for Decryption (Modified Caesar Cipher):

For this step we basically reverse the process of encryption technique used in the modified Caesar Cipher. And use the following formula:

text[i] = text[i] - code - (power_ex)^i

Note: If, ASCII value of text[i] < 0, then, set text[i] = Mod (text[i], 256); 'i' is the position of each character in the text and text[] is the message to be encrypted, where text[i] denotes each character of the text[] at position 'i'.

2.3. Encrypt the data using TTJSA:

TTJSA method is a combination of 3 distinct cryptographic methods, namely, (i) Generalized Modified Vernam Cipher Method, (ii) MSA method and (iii) NJJSA method. To begin the method a user has to enter a text-key, which may be at most 16 characters in length. From the text-key the randomization number and the encryption number is calculated using a method proposed by Nath et al. A minor change in the text-key will change the randomization number and the encryption number quite a lot. The method have also been tested on various types of known text files and have been found that, even if there is repetition in the input file, the encrypted file contains no repetition of patterns.

2.3.1. Algorithm of TTJSA (Encryption):

Step 1: Start
Step 2: Initialize the matrix mat[16][16] with numbers 0 to 255 in row major wise.
Step 3: call keygen() to calculate randomization number (=times), encryption number (=secure)
Step 4: call randomization() function to randomize the contents of mat[16][16].
Step 5: times2=times
Step 6: copy file f1 into file2
Step 7: k=1
Step 8: if k>secure go to Step 15
Step 9: p=k%6
Step 10: if p=0 then
call vernamenc(file2,outf1)
times=times2
call njjsaa(outf1,outf2)
call msa_encryption(outf2,file1)
else if p=1 then
call vernamenc(file2,outf1)
times=times2
call msa_encryption(outf1,file1)
call file_rev(file1,outf1)
call njjsaa(outf1,file2)
call msa_encryption(file2,outf1)
call vernamenc(outf1,file1)
times=times2
else if p=2 then
call msa_encryption(file2,outf1)
call vernamenc(outf1,outf2)
set times=times2
call njjsaa(outf2,file1)
else if p=3 then
call msa_encryption(file2,outf1)
call njjsaa(outf1,outf2)
call vernamenc(outf2,file1)
times=times2
else if p=4 then
call njjsaa(file2,outf1)
call vernamenc(outf1,outf2)
times=times2
call msa_encryption(outf2,file1)
else if p=5 then
call njjsaa(file2,outf1)
call msa_encryption(outf1,outf2)
call vernamenc(outf2,file1)
times=times2
Step 11: call function file_rev(file1,outf1)
Step 12: copy file outf1 into file2
Step 13: k=k+1
Step 14: goto Step 8
Step 15: End

2.3.2. Algorithm of vernamenc(f1,f2):

Step 1: Start vernamenc() function

Step 2: The matrix mat[16][16] is initialized with numbers 0-255 in row major wise order
Step 3: call function randomization() to
randomize the contents of mat[16][16].
Step 4: Copy the elements of random matrix
mat[16][16] into key[256] (row major wise)
Step 5: pass=1, times3=1, ch1=0
Step 6: Read a block from the input file f1 where number of characters in the block 256 characters
Step 7: If block size < 256 then goto Step 15
Step 8: copy all the characters of the block into an array str[256]
Step 9: call function encryption where str[] is passed as parameter along with the size of the current block
Step 10: if pass=1 then
times=(times+times3*11)%64
pass=pass+1
else if pass=2 then
times=(times+times3*3)%64
pass=pass+1
else if pass=3 then
times=(times+times3*7)%64
pass=pass+1
else if pass=4 then
times=(times+times3*13)%64
pass=pass+1
else if pass=5 then
times=(times+times3*times3)%64
pass=pass+1
else if pass=6 then
times=(times+times3*times3*times3)%64
pass=1
Step 11: call function randomization() with
current value of times
Step 12: copy the elements of mat[16][16] into
key[256]
Step 13: read the next block
Step 14: goto Step 7
Step 15: copy the last block (residual character if any) into str[]
Step 16: call function encryption() using str[] and the no. of residual characters
Step 17: Return

2.3.3. Algorithm of function encryption(str[],n):

Step 1: Start encryption() function
Step2: ch1=0
Step 3: calculate ch=(str[0]+key[0]+ch1)%256
Step 4: write ch into output file
Step 5: ch1=ch
Step 6: i=1
Step 7: if in then goto Step 13
Step 8: ch=(str[i]+key[i]+ch1)%256
Step 9: write ch into the output file
Step 10: ch1=ch
Step 11: i=i+1
Step 12: goto Step 7
Step 13: Return

2.3.4. Algorithm for Decryption:

Step 1: Start
Step 2: initialize mat[16][16] with 0-255 in row major wise
Step 3: call function keygen() to generate times
and secure
Step 4: call function randomization()
Step 5: set times2=times
Step 6: call file_rev(f1,outf1)
Step 7: set k=secure
Step 8: if k<1 go to Step 15
Step 9: call function file_rev(outf1,file2)
Step 10: set p=k%6
Step 11: if p=0 then
call msa_decryption(file2,outf1)
call njjsaa(outf1,outf2)
call vernamdec(outf2,file2)
times=times2
else if p=1 then
call function vernamdec(file2,outf1)
set times=times2
call function msa_decryption(outf1,outf2)
call fumction njjsaa(outf2,file2)
call function file_rev(file2,outf2)
call function msa_decryption(outf2,outf1)
call function vernamdec(outf1,file2)
times=times2
else if p=2 then
call njjsaa(file2,outf1)
call vernamdec(outf1,outf2)
times=times2
call msa_decryption(outf2,file2)
else if p=3 then
call vernamdec(file2,outf1)
times=times2
call njjsaa(outf1,outf2)
call msa_decryption(outf2,file2)
else if p=4 then
call msa_decryption(file2,outf1)
call vernamdec(outf1,outf2)
times=times2
call njjsaa(outf2,file2)
else if p=5 then
call vernamdec(file2,outf1)
times=times2
call msa_decryption(outf1,outf2)
call njjsaa(outf2,file2)
Step 12: copy the content of file2 to outf1
Step 13: set k=k-1
Step 14: Goto Step 8
Step 15: End

2.3.5. Algorithm of function vernamdec(f1,f2):

The algorithm of vernamdec() function is same as vernamenc() function. Here the only difference is that decryption() function is called instead of encryption() function.

2.3.6. Algorithm of decryption(str[],n):

Step 1: Start
Step 2: ch1=0
Step 3: ch=(256+str[0]-key[0]-ch1)%256
Step 4: write ch into the output file
Step 5: i=1
Step 6: if in then goto Step 12
Step 7: ch=(256+str[i]-key[i]-str[i-1]) %256
Step 8: write ch into the output file
Step 9: i=i+1
Step 10: goto Step 6
Step 11: ch1=str[n-1]
Step 12: Return

2.3.7. Algorithm of function file_rev(f1,f2):

Step 1: Start
Step 2: open the file f1 in input mode
Step 3: open the file f2 in output mode
Step 4: calculate n=sizeof(file f1)
Step 5: move file pointer to n
Step 6: read one byte
Step 7: write the byte on f2
Step 8: n=n-1
Step 9: if n>=1 then goto step-6
Step 10: close file f1, f2
Step 11: Return

2.3.8. NJJSAA ALGORITHM:

Nath et al. [2] proposed a method which is basically a bit manipulation method to encrypt or to decrypt any file.

The encryption number (=secure) and randomization number (=times) is calculated according to the method mentioned in MSA algorithm [1].

Step 1: Read 32 bytes at a time from the input file.
Step 2: Convert 32 bytes into 256 bits and store in some 1-dimensional array.
Step 3: Choose the first bit from the bit stream and also the corresponding number(n) from the key matrix. Interchange the 1st bit and the n-th bit of the bit stream.
Step 4: Repeat step-3 for 2nd bit, 3rd bit...256-th bit of the bit stream
Step 5: Perform right shift by one bit.
Step 6: Perform bit(1) XOR bit(2), bit(3) XOR bit(4),...,bit(255) XOR bit(256)

Step 7: Repeat Step 5 with 2 bit right, 3 bit right,...,n bit right shift followed by Step 6 after each completion of right bit shift.

2.3.9. MSA (MEHEBOOB, SAIMA, ASOKE) ENCRYPTION AND DECRYPTION ALGORITHM:

Nath et al. [3] proposed a symmetric key method where they have used a random key generator for generating the initial key and that key is used for encrypting the given source file. MSA method is basically a substitution method where we take 2 characters from any input file and then search the corresponding characters from the random key matrix and store the encrypted data in another file. MSA method provides us multiple encryptions and multiple decryptions. The key matrix (16x16) is formed from all characters (ASCII code 0 to 255) in a random order.

The randomization of key matrix is done using the following function calls:

Step-1: call Function cycling()
Step-2: call Function upshift()
Step-3: call Function downshift()
Step-4: call Function leftshift()
Step-5: call Function rightshift()

Note: Cycling, upshift, downshift, leftshift, rightshift are matrix operations performed (applied) on the matrix, formed from the key.

2.4. DJSA Algorithm

DJSA algorithm [6] is the extended version of MSA[3]. In DJSA the key matrix is defined as 256 x 256 and in each cell we store all possible two lettered words. The key matrix is randomized using extended MSA [6] randomization method. The encryption method of DJSA method is very similar to MSA except we read 4 characters from the plain text file and then split into two words. Then we have to search the key matrix whose size is 256x256. Here the encryption procedure is same as described in MSA. DJSA is very effective to encrypt small file. It works quite well provided the repetition of characters are less in a file. DJSA algorithm already tested on normal text file, image or audio or video file and we found it works quite successfully. But again this method is not free from cryptographic attack such as differential attack, known plain text attack etc.

2.5. Bit Reversal Technique

In this method we take the encrypted message, which we got from our previous stages and encrypt it for the last time. In this final stage, each byte is taken from the encrypted message and then converted to its binary format again. Then the whole sequence of the bits in the binary format is reversed.

i.e. If $[B_8B_7B_6B_5B_4B_3B_2B_1]$ be the binary format of each byte, then the sequence of bits are reversed and the final encrypted form of each byte is $[B_1B_2B_3B_4B_5B_6B_7B_8]$.

$[B_1B_2B_3B_4B_5B_6B_7B_8] \rightarrow$ FINAL ENCRYPTED FORM OF EACH BYTE

3. PROPOSED METHOD IN BLOCK DIAGRAM

In this section, we briefly show the idea of the proposed method in the form of block diagram:

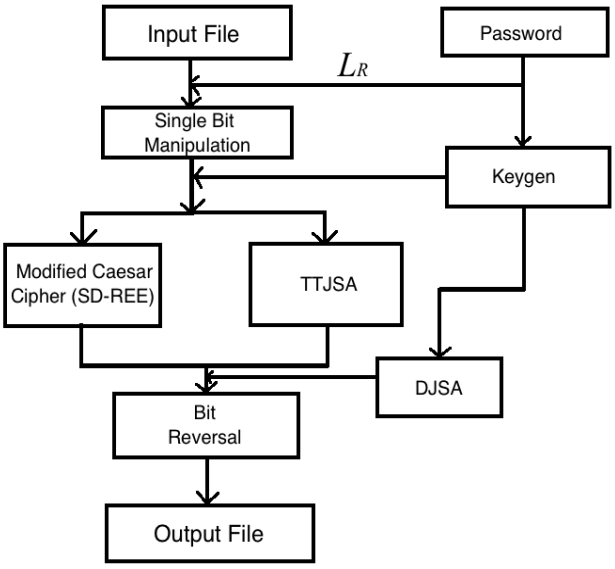

Figure: Block Diagram to Represent the Proposed Method SJA-I

4. RESULTS AND DISCUSSIONS

This method, SJA-I is used to encrypt different types of data and few results are given below:

Message	Encrypted Message
8 ASCII Value (1)	¿L_ü¬xäüÑ
8 ASCII Value (2)	0Ä„#nÐ□’ •
8 ASCII Value (4)	IªF‘šöyG
8 ASCII Value (8)	üo¹qâ0çg0
8 ASCII Value (255)	±z^Wü ¿´T
Lord Jesus Christ Son Of God Have A Mercy On Me A Sinner	□ Ý]ØE+ÿÏ□ Hšâ*G Ü›□ c,“‹Ê0— öãéðåÁ bÇNLE/Ê¾³ÉÏîW

5. SPECTRAL ANALYSIS AND CRYPTANALYSIS

One of the classical cryptanalysis method used is by detecting the frequency of characters in the encrypted text (message). So to test the effectiveness of SJA-I method, spectral analysis of the frequency of characters are closely observed. Using this method, SJA-I, we ran many analysis and tested different strings as input and used various methods of cryptanalysis. To show the usefulness and integrity of this cryptographic module, we used spectral analysis of the frequency of characters.

First, as a test case we chose, a file which contained 128 bytes of ASCII value (1) and used this method to encrypt the data. Fig 1.1 shows the spectral analysis of frequency of characters of 128 ASCII value (1) and Fig 1.2 shows the spectral analysis of frequency of characters of the encrypted data.

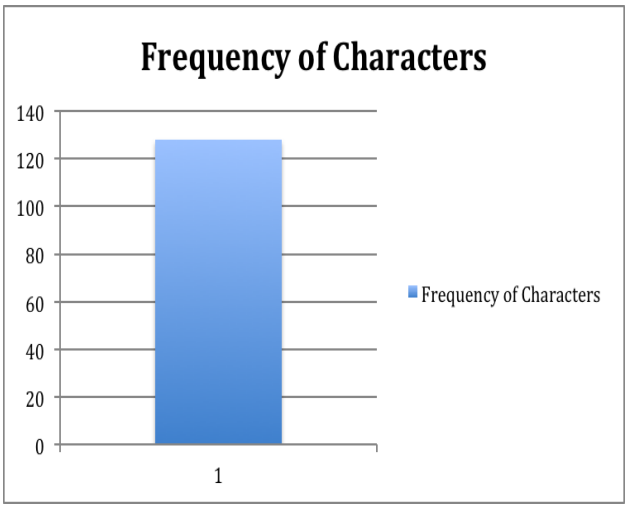

Fig 1.1: Spectral Analysis of Frequency of Characters of 128 ASCII Value (1)

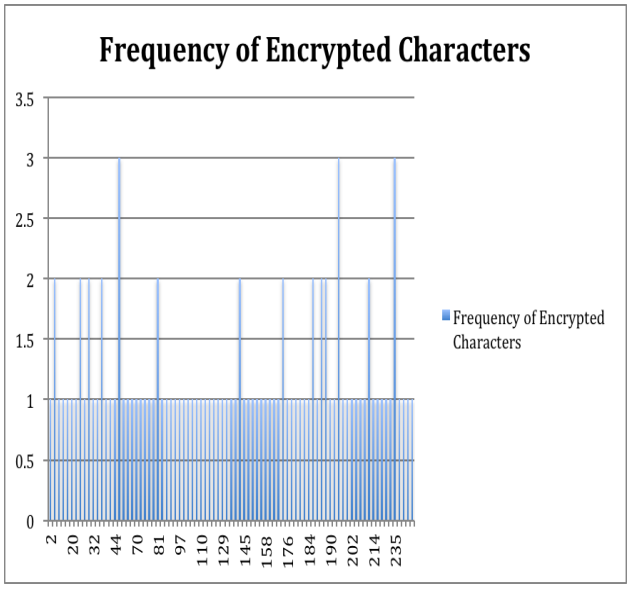

Fig 1.2: Spectral Analysis of Frequency of Characters of Encrypted data of 128 ASCII value (1)

As 2nd Test case, we chose a random text file of size 1024 bytes. The spectral analysis of the frequency of characters of the text file is shown in Fig 2.1 and Fig 2.2 shows the spectral analysis of frequency of characters of encrypted text.

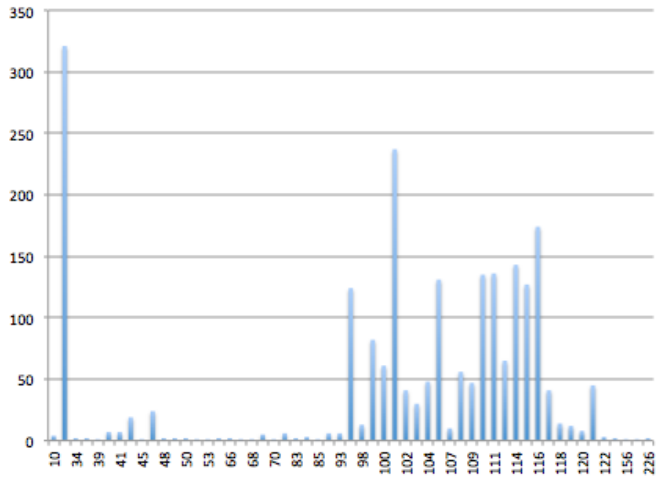

Fig 2.1: Spectral Analysis of frequency of characters of an ordinary text file

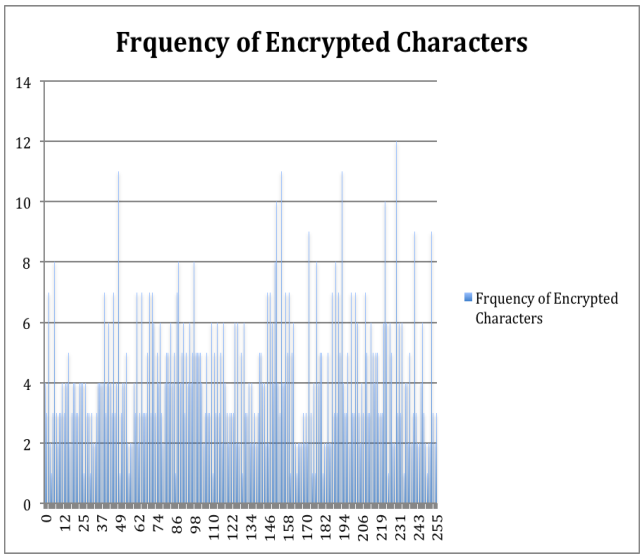

Fig 2.2: Spectral Analysis of frequency of characters of the encrypted text file

As 3rd Test case, we chose a file containing 128 bytes of ASCII value (255). The spectral analysis of the frequency of characters of the file is shown in Fig 3.1 and Fig 3.2 shows the spectral analysis of the frequency of characters of the encrypted file.

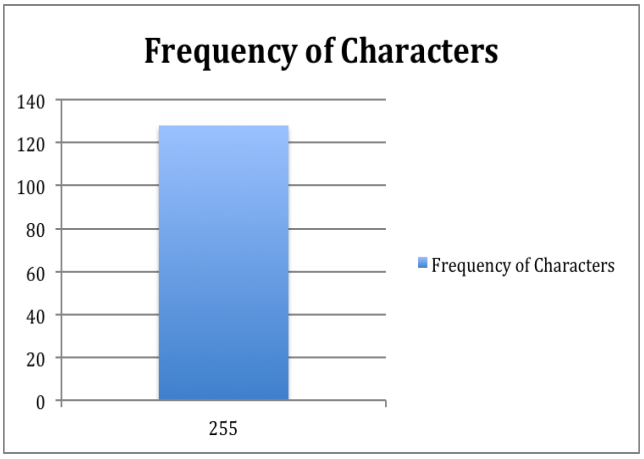

Fig 3.1: Spectral Analysis of Frequency of Characters of 128 bytes of ASCII value (255)

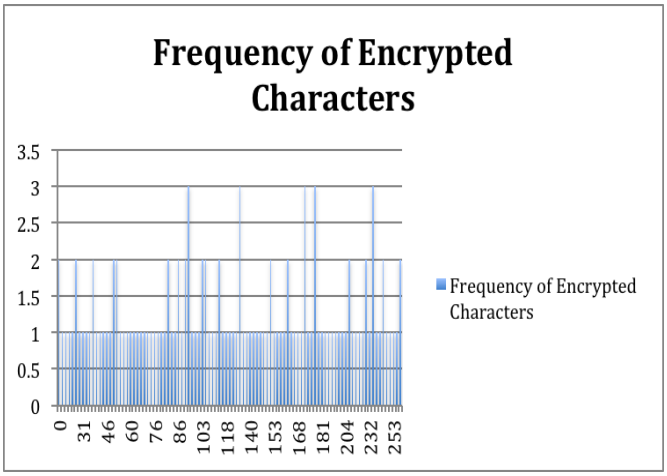

Fig 3.2: Spectral Analysis of Frequency of Characters of 128 bytes of Encrypted ASCII value (255)

Thus, from the above spectral analysis it is evident that the method, SJA-I, used here is very effective and there is no trace of any pattern in the encryption technique.

Since this cryptographic technique uses multiple bit and byte level encryption multiple times, for this reason, the method used here is unique and almost unbreakable because there is no trace of any pattern. And this method is also effective against both Differential Cryptanalysis (Differential Attack) and Brute-Force Attack.

6. CONCLUSION AND FUTURE SCOPE

In the present work, SJA-I, we use four different algorithms to make the encryption process unbreakable from standard cryptographic attacks. That is evident from our results also. We have applied our method on some known text, where the same character repeats for a number of times and we have found that after encryption there is no repetition of pattern in the output file. We have tested this feature

closely and have found satisfactory results in almost all cases. This has been possible as we have used modified Caesar Cipher method (SD-REE) with non-linear polynomial function, modified Vernam Cipher method with feedback mechanism and also NJJSAA, MSA and DJSA methods. The present method uses both byte-wise encryption, as well as bit-wise encryption. We propose that this encryption method can be applied for data encryption and decryption in banks, defense, mobile networks, ATM networks, government sectors, etc. for sending confidential data. The above method i,.e SJA-I method, may be further strengthened using additional bit manipulation method and we have already started to work on it.

7. ACKNOWLEDGMENTS

Somdip Dey (SD) expresses his gratitude to all his fellow students and faculty members of the Computer Science Department of St. Xavier's College [Autonomous], Kolkata, India, for their support and enthusiasm. AN is grateful to Dr. Fr. Felix Raj, Principal St. Xavier's College, Kolkata for giving opportunity to work in the field of data encryption, data hiding and retrieval.

8. REFERENCES

[1] http://www.purdue.edu/discoverypark/gk12 / downloads/Cryptography.pdf

[2] Symmetric key cryptosystem using combined cryptographic algorithms - Generalized modified Vernam Cipher method, MSA method and NJJSAA method: TTJSA algorithm " Proceedings of Information and Communication Technologies (WICT), 2011 " held at Mumbai, 11^{th} – 14^{th} Dec, 2011, Pages:1175-1180.

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

[4] New Symmetric key Cryptographic algorithm using combined bit manipulation and MSA encryption algorithm: NJJSAA symmetric key algorithm: Neeraj Khanna,Joel James,Joyshree Nath, Sayantan Chakraborty, Amlan Chakrabarti and Asoke Nath : Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 03-06 June 2011, Page 125-130(2011).

[5] Advanced Symmetric key Cryptography using extended MSA method: DJSSA symmetric key algorithm: Dripto Chatterjee, Joyshree Nath, Soumitra Mondal, Suvadeep Dasgupta and Asoke Nath, Jounal of Computing, Vol3, issue-2, Page 66-71,Feb(2011)

[6] A new Symmetric key Cryptography Algorithm using extended MSA method :DJSA symmetric key algorithm, Dripto Chatterjee, Joyshree Nath, Suvadeep Dasgupta and Asoke Nath : Proceedings of IEEE CSNT-2011 held at SMVDU(Jammu) 3-5 June,2011, Page-89-94(2011).

[7] Symmetric key Cryptography using modified DJSSA symmetric key algorithm, Dripto Chatterjee, Joyshree Nath, Sankar Das, Shalabh Agarwal and Asoke Nath, Proceedings of International conference Worldcomp 2011 held at LasVegas 18-21 July 2011, Page-306-311, Vol-1(2011).

[8] An Integrated symmetric key cryptography algorithm using generalized vernam cipher method and DJSA method: DJMNA symmetric key algorithm : Debanjan Das, Joyshree Nath, Megholova Mukherjee, Neha Chaudhury and Asoke Nath: Proceedings of IEEE International conference : World Congress WICT-2011 to be held at Mumbai University 11-14 Dec, 2011, Page No.1203-1208(2011). .

[9] Symmetric key Cryptography using two-way updated – Generalized Vernam Cipher method: TTSJA algorithm, Trisha Chatterjee, Tamodeep Das, Shayan dey, Joyshree Nath, Asoke Nath , International Journal of Computer Applications(IJCA, USA), Vol 42, No.1, March, Pg: 34 -39(2012).

[10] Ultra Encryption Standard(UES) Version-I: Symmetric Key Cryptosystem using generalized modified Vernam Cipher method, Permutation method and Columnar Transposition method,

[11] Satyaki Roy, Navajit Maitra, Joyshree Nath,Shalabh Agarwal and Asoke Nath, 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).

[12] Peter Montgomery, "Modular Multiplication Without Trial Division," Math. Computation, Vol. 44, pp. 519–521, 1985.

[13] W. Hasenplaugh, G. Gaubatz, and V. Gopal, "Fast Integer Reduction," 18th IEEE Symposium on Computer Arithmetic (ARITH '07), pp. 225– 229, 2007.

[14] Somdip Dey, "SD-REE: A Cryptographic Method To Exclude Repetition From a Message", Proceedings of The International Conference on Informatics & Applications (ICIA 2012), Malaysia, p. 182 – 189.

[15] Somdip Dey, "SD-AREE: A New Modified Caesar Cipher Cryptographic Method Along with Bit-Manipulation to Exclude Repetition from a Message to be Encrypted", Journal: Computing Research Repository - CoRR, vol. abs/1205.4279, 2012.