Enhancement of Security through an Efficient Substitution-based Block Cipher of Bit-level Implementation with Possible Lossless Compression

Pranam Paul
Dr. B. C. Roy Engineering College
Durgapur - 713206
West Bengal, India

Saurabh Dutta
Dr. B. C. Roy Engineering College
Durgapur - 713206
West Bengal, India

A K Bhattacharjee
National Institute of Technology
Durgapur - 713209
West Bengal, India

Abstract

This paper presents a substitution-based block cipher that considers a file to be encrypted as a bit-stream. The cipher implements a storage efficient algorithm through which along with encryption a reduction in size is also achieved. As encryption is done at bit label, this algorithm can be implemented on any kind of files. A tendency of increase in execution time is observed. The proposed technique is compared with the existing International Data Encryption Algorithm (IDEA) with respect to execution time and degree of non-homogeneity. A generalized expression for the key space is formularized.

***Key Words*:**
cryptography, encryption, decryption, cipher, private key, symmetric key, plain text, cryptographic modeling.

1. Introduction

Cryptography is an essential tool of data security through mathematical manipulation of data with an incomprehensible format for unauthorized person. Lossless data compression ensures storage efficiency. Incorporation of size-reduction while implementing a ciphering protocol introduces a special dimension in the scope of encryption.

In this paper, a substitution-based, private-key, block cipher, termed as "Block Substitution with the Sequential Position of Sorted Distinct blocks on Frequency (BSSPSDF)" is presented, which is modeled for implementing at bit-level with decomposing equal block length. This storage-efficient, cipher BSSPSDF is generated smaller encrypted bit-stream in size than the source one, but it depends on context of the plain text. If the difference between number of distinct blocks in source bit-stream and number of all possible combinations of blocks containing that particular length is at least 2, 1-bit compression is achieved for each and every occurrence of two particular blocks in source bit-stream.

Section 2 presents the scheme followed in the encryption technique. Section 3 describes an implementation of the technique. Section 4 presents results of executing the technique on some real files. Section 5 is an analytical discussion on the technique and draws a conclusion

Manuscript received April 5, 2008
Manuscript revised April 20, 2008

2. The Scheme

This section presents a description of the actual scheme, used during implementing BSSPSDF technique. To encrypt a source-bit-stream, we decomposed it into some blocks of equal length. Then each block is replaced with some other binary blocks. Different blocks in source-bit-stream, that is termed as Original Code are replaced by different binary blocks, which are termed in this particular paper as Replaced Codes, but for each same block, proposed replaced binary block is same during encryption. Generation of Replaced Code for each different block, i.e. Original Code is discussed in section 2.1. Section 2.2 describes the scheme used in ciphering technique, while Section 2.3 describes the scheme used in deciphering technique and in section 2.4, formation of Key is discussed

Replaced Code generation for source bit-stream-block.

At first, source file, means source bit stream is decomposed into blocks of equal length. Find out the distinct blocks of particular length with their sequence in original bit stream. Let us assume N be the number of distinct blocks if block length is L. So $0 \le N \le 2^L - 1$.

Step 1: We calculate the minimum numbers of bits for representing N, say n.
⇨ $0 \le N \le 2^L - 1$ and $n \le L$

Step 2: For all Original Code B_i ($\forall\ 0 \le i \le N$ and i is an integer) and i holds position in set of all distinct blocks. B_i, we check $i < 2^{n-1}$ or not and assume Replaced Code, RC = NULL.
$V = 2^{n-1}$, $P = V \div 2$, $Diff = 0$

Step 3: If $i < V$, we put 0 at the right most position of RC and then concatenate (n-1) bits representation of $(i - Diff)$ with recently generated Replaced Code

```
        at right end and it is Replaced code, RC for $B_i$,
        the Original Code of that blocks. Stop the
        processing.
        Else (means $i \geq 2^{n-1}$)
          we put 1 at the right most position of RC.
        End if
Step 4: If $V + P > N$
            We calculate required minimum number bit
            for representing $(N - V)$, say C bits. Then
            concatenate C bits representation of binary
            form of $(i - V)$, with recently generated
            incomplete Replaced Code, RC at right most
            place.
        Else (means $V + P \leq N$)
            Diff =V
            $V = V + P$
            $P = P \div 2$
            $n = n - 1$
            Go to Step 3
        End if
Step 5: Like this way, set of Replaced Code of all distinct
        blocks, occurred in source-bits-stream is formed.
```

If $N < (2^L - 2)$, there are at least 2 blocks, of which length of Replaced Code is less than length of source block (i.e. L) [9].

The scheme for ciphering technique

At first, source file is convert into binary form i.e. source bit stream which is decomposed into blocks of equal length; say L bits where L is an integer and $L \geq 2$. It may be happened that after decomposition of total source-bit-stream into some L bits blocks, a blocks, less than L bits is left at last, say U_B (=> length of $U_B < L$) which is totally unchanged during encryption.

Step 1: We are finding out all distinct bit-stream-blocks of L bits length appeared in source bit stream with their frequency of appearing. Say number of distinct block is N, So $0 \leq N \leq 2^L -1$.

Step 2: We sort the distinct source-bit-stream blocks as ascending order with their frequency of appearing and keep set of distinct blocks with recently arranged order.

Step 3: We are generating Replaced Code, discussed in section 2.1. for each distinct block which are already kept into key.

Step 5: Replace source bit stream with corresponding its Replaced Code. Then generate an intermediate cipher bit stream.

Step 6: Calculate required numbers of bits, say D_0 for which after concatenation of D_0 dummy bits with intermediate cipher bit stream and unchanged block i.e. U_B, its length will be multiple of 8.

Step 7: Finally to generate cipher bit stream concatenate maintaining following sequence: U_B, D_0 numbers of 0 (ZERO) as dummy bits and intermediate cipher bits stream. Accordingly from cipher bit stream, cipher text will be generated.

The scheme for deciphering technique

After receiving the target block and key, receiver comes to know information for each block, which was sending. After receiving the required information, the decryption authority performs the task of decryption.

From key, receiver comes to know source block length, say L, unchanged block length, say N_{UB}, number of dummy 0, say D_0 and number of distinct blocks, appeared in source bit stream, say N and obviously total set of distinct blocks in source bit stream with their sequence.

So $0 \leq N \leq 2^L -1$.

```
Step 1: Calculate minimum number of bits, required to
        represent N, say n.
        ⇨ $0 \leq N \leq 2^L -1$ and $n \leq L$.
Step 2: Calculate two integer C and P using following
        way:

        If $N = 2^L - 1$ or $N = 2^L - 2$ then
           $C = 0$ and $P = L$
        Else
           $C = 1$, $V = 2^{n-1}$
           increment = $V \div 2$, $P = 0$
           While (V + increment <N)
            {
              V = V + increment
              increment = increment ÷ 2
              C = C + 1
            }
            While ($2^P < N - V$)
            {
              P = P +1
            }
            P = P – 1
Step 3: Convert binary form of cipher text and get target
        bits stream. Take $N_{UB}$ bit from beginning of
        encrypted bits stream, say $U_B$. Leave next $D_0$ bits.
Step 4: Take next C bits stream from target bit stream.
Step 5: If there is at least one 0 (ZERO) in those C bits
            stream then take next $(n - C)$ bits and
            append at last of those C bits steam, it is
            treated as K, means K is bit stream block.
        Else (means all C bits are 1 or C = 0)
```

Take P next bits and append at last of those C bits steam, it is treated as K, means K is bit a stream block.

Step 6: Replace the Replaced Code K with Original Code, discussed into section 2.1.

Step 7: Take next C bit unprocessed blocks from target bit stream. Reaching end of the target bit stream exit from this process and getting incomplete deciphering bit stream, otherwise go to Step 4.

Step 6: Finally, concatenate U_B with incomplete deciphering bit stream at end and generate complete deciphering bits stream as well as also generate deciphering text.

Segment	Size	Description
1st	d	Binary form of L – 1 with d bits representation.
2nd	L	L bits binary form of the number of distinct blocks appeared in source-bit-stream counting starting from 0.
3rd	d	d bits binary form of number of unprocessed bits
4th	3	d bits binary form of number of dummy 0 (ZERO) to be added for making total encrypted string length as multiple of 8.
5th	$L \times 2^L$	All distinct blocks with their proper sequence and length, i.e. L

Total Size of Key: $2d + (2^L + 1) \times L + 3$

Picture 2.4.1
Key Structure

Key generation

For correct decryption for cipher text, correct Key is essential. For this algorithm key contents 5 segments. 1st segment denotes length of the blocks, while 2nd segment contents number of distinct blocks appeared in source bits stream. 3rd segment holds number of unchanged bits during encryption. How many dummy bits (i.e. 0) are been added to make length of encrypted bit stream as multiple of 8, is kept in 4th segment and last segment, means 5th segment holds set of all distinct bits blocks with their proper length and sequence, occurred in source bits stream.

If L bit blocks is taken for encrypting source bit stream. Find out an integer; say d for which $2^{d-1} < L \leq 2^d - 1$. Key structure is shown in figure 2.4.1.

3. An Implementation

We consider the plaintext P as "**Encrypt**". The stream of bits, S, representing P is as follows:
0100010101101110011000110111001001111001011100 0001110100.

Now S is decomposed into some block with 4 bits length. These are 0100, 0101, 0110, 1110, 0110, 0011, 0111, 0010, 0111, 1001, 0111, 0000, 0111 and 0100. Section 3.1 shows generation of Replaced Code of decomposed blocks. In section 3.2, encryption is been discussed while section 3.3 is used to discuss about key formation and at last, how correct decryption is been done, discussed in section 3.4.

3.1 Generation of Replaced Code

There are 9 distinct blocks in set of all decomposed blocks. Replaced Code for each distinct block is shown in table 3.1.1. by using the algorithm, discussed in section 2.1.

Table 3.1.1
Replaced Code for distinct blocks

Serial No. (Counting starting with 0)	Distinct Block in sorted ordered on their frequency	Frequency of the distinct blocks	Replaced Code
0	0101	1	0000
1	1110	1	0001
2	0011	1	0010
3	0010	1	0011
4	1001	1	0100
5	0000	1	0101
6	0100	2	0110
7	0110	2	0111
8	0111	4	1

3.2 Encryption

For the encryption Replaced Code, is very essential. Total encryption process has been shown in table 3.2.1.

Table 3.2.1
Encryption Process

Source Bit Block	**Encrypted bit Block by Replaced Code**	**Incomplete Encrypted bit Stream**
0100	0110	01100000 01110001 01110010 10011101 00101011 0110
0101	0000	
0110	0111	
1110	0001	
0110	0111	
0011	0010	
0111	1	
0010	0011	
0111	1	
1001	0100	
0111	1	
0000	0101	
0111	1	
0100	0110	

After generating encrypted bit stream, unchanged block and dummy bits will be added of with the incomplete encrypted bits stream at beginning. Here is no unchanged block and 4 dummy bits would be required for making length of encrypted bits stream, multiple of 8. So encrypted bits stream is **0000**01100000011100010111001010011101001010110110, from which cipher text, "♠•↨)╥┤║" is been generated.

3.3 Key Generation

For this particular example, source bit stream is decomposed into some blocks, having equal length i.e. 4 bits. So 3 bits are required to represent binary form of 4. Key generation is been shown table 3.3.1 As there 9 distinct 4 bits blocks, but for this example 5th segment contains $4 \times 2^4 = 64$, so after $4 \times 9 = 36$ bits, add $64 - 36 = 28$ numbers of 0 (ZERO) as 28 bits.

Table 3.3.1
Key Formation of the 6-bit block

Segment	**Description of the segment**	**Number of required bits**	**Significant Bits**
1st	Length of the blocks – 1	3	100
2nd	Number of distinct blocks	4	1000 (counting starting from 0)
3rd	Number of unchanged blocks	3	000
4th	Numbers of dummy bits	3	100
5th	Set of all distinct blocks	$4 \times 2^4 = 64$	0101111001110 0101001000001 0001100011000 0000000000000 000000000000

So for this particular example, 77 bits private key will be 1001000000100010111100111001010010000010001100011000000000000000000000000000.

3.4 Decryption

After receiving encrypted text, ♠•↨)╥┤║" and key which is very essential for decryption such that original text will back.

Step 1: From 1st, 2nd, 3rd and 4th segment of key, block length (say L), number of distinct blocks in target bits stream (say N), number unchanged bits (say N_{UB}) and number of added dummy bits (say D_0) are come to know. Here $L = 4$, $N = 9$, $N_{UB} = 0$, $D_0 = 4$.

Step 2: From 5th segment of key, first N numbers of L bits blocks are taken to generate Replaced Code table which same as table 3.1.1.

Step 3: Using step 2 of decryption process, discussed in section 2.3, calculate value of C and P.
Here $C = 1$ and $P = 0$

Step 4: Keep N_{UB} bits from beginning of target blocks into U_B which is unchanged blocks and leave next D_0 bits from target bits stream.
Here U_B = NULL

Then total decryption process for the target blocks, 01100000011100010111001010011101001010110110 is shown in table 3.4.1.

Table 3.4.1
Decryption Process

ROUND	Take next C (=1) bits	If all C bits are 1, take next P (=0) bits else next L-C (4-1=3) bits.	Replaced Code (Adding previous two columns)	Distinct Blocks code from table 3.1.1	Decrypted bits stream
1	0	110	0110	0100	0100 0101 0110 1110 0110 0011 0111 0010 0111 1001 0111 0000 0111 0100
2	0	000	0000	0101	
3	0	111	0111	0110	
4	0	001	0001	1110	
5	0	111	0111	0110	
6	0	010	0010	0011	
7	1		1	0111	
8	0	011	0011	0010	
9	1		1	0111	
10	0	100	0100	1001	
11	1		1	0111	
12	0	101	0101	0000	
13	1		1	0111	
14	0	110	0110	0100	

From table 3.4.1, received decrypted bits stream, 01000101011011100110001101 110010011110010111000001110100 is same as source bits stream so obviously decrypted text "**Encrypt**" is same as plane text.

4. Results

Here we have compared BSSPSDF with International Data Encryption Algorithm (IDEA) to establish a comparative analytical report that is helpful to understand strength and weakness of BSSPSDF. A brief idea on IDEA is discussed in section on 4.1 whereas section 4.2 describes comparative reports with IDEA on the basic of different parameters.

4.1 International Data Encryption Algorithm (IDEA)

International Data Encryption Algorithm is a 128-bit private key cipher, which is implemented in bit level with equally decomposed 64 bit blocks of plain text depending on modular arithmetic. There are regenerated 16-bit 52 sub keys from 128 bits key based on rotational replacement and some particular rules, say $k_1, k_2, k_3 \ldots k_{52}$. A 64-bit block of plain text is decomposed into 4 16-bit sub blocks; say P_1, P_2, P_3 and P_4. In IDEA, there are 8 rounds which have being followed same procedure. Each round takes output of previous blocks and 6 sub keys, according to the number of round, except 1st round, takes 4 sub blocks. Entire process of encryption through IDEA is shown in figure 4.1.1. After completing the total process we get a 64-bit block which is an encrypted block of that 64 bit block of plain text. Same things will be done on each and every generated blocks of plain text and finally we get encrypted text which is same in size of the plain text [7] [8].

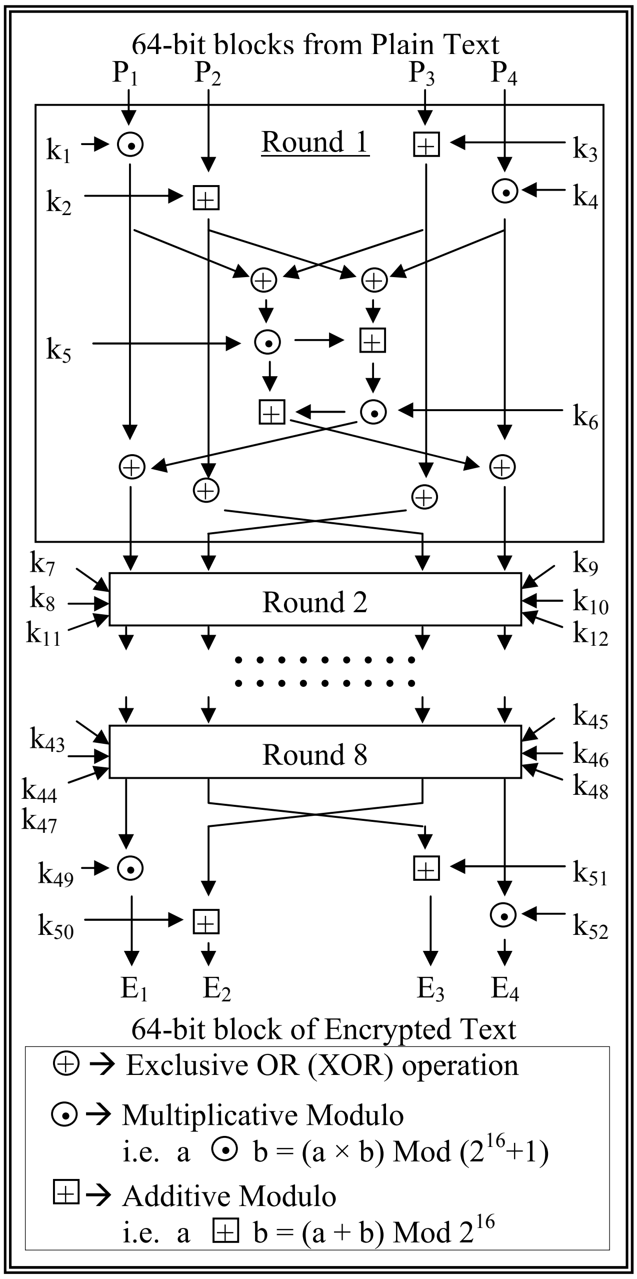
64-bit blocks from Plain Text
P1 P2 P3 P4
Round 1
k1 k2 k3 k4 k5 k6
k7 k8 k11 Round 2 k9 k10 k12
k43 k44 k47 Round 8 k45 k46 k48
k49 k50 k51 k52
E1 E2 E3 E4
64-bit block of Encrypted Text
⊕ → Exclusive OR (XOR) operation
⊙ → Multiplicative Modulo
i.e. a ⊙ b = (a × b) Mod (2^16+1)
⊞ → Additive Modulo
i.e. a ⊞ b = (a + b) Mod 2^16

Figure 4.1.1
Process of Encryption through IDEA

4.2 Comparative Report

Both the proposed BSSPSDF protocol (with 6-bit block size in this paper) and exiting IDEA have been implemented on a number of data files varying types of content and sizes of a wide range, shown in table 4.2.1.

Table 4.2.1
Set of Files with Size

Sl. No.	Source File Name	Original File Size
1	a.txt	14337
2	b.txt	73710
3	c.txt	131776
4	Des_56.cpp	14983
5	M.txt	48430
6	Calc.exe	114688
7	hh.exe	10752
8	Win.com	18432
9	Ansi.sys	9029
10	Watch.sys	14592
11	Blue.bmp	1272
12	ZAPO.BMP	9522

Accordingly the observations on the following points have been made:

1. Compression achieved in Percentage:

To prove, this technique is performed as encryption technique as well as compressed original file in size, table 4.2.2 shows the compression between proposed technique BSSPSDF and IDEA on the basis of rate of compression in size. No compression in encrypted file size is accorded during encryption for IDEA. Due to the dependency on context of source file, compression is achieved for some source files and some encrypted files are remained same in size during the implementation of BSSPSDF. Serial numbers for table 4.2.1 and table 4.2.2 are same for corresponding files.

Table 4.2.2
Relationship between Source and Encrypted File Size

Sl. No.	For BSSPSDF		For IDEA	
	Encrypted File Size	Rate of Compression	Encrypted File Size	Rate of Compression
1	11009	23.214%	14337	0.000%
2	55280	25.004%	73710	
3	102489	22.225%	131776	
4	13368	10.781%	14983	
5	36754	24.108%	48430	
6	114688	0.000%	114688	
7	10752	0.000%	10752	
8	18432	0.000%	18432	
9	8306	8.011%	9029	
10	14592	0.000%	14592	
11	1201	5.550%	1272	
12	9149	3.9021%	9522	

2. Encryption and Decryption Time* :

In Table 4.2.3, a comparison on basis of encryption time with their file size between BSSPSDF and IDEA has been shown here on the same set of files, used in Table 4.2.1 for evaluating the computational overhead.

Table 4.2.3
Encryption Time for BSSPSDF and IDEA

Sl. No.	File Size	Encryption time for BSSPSDF	Encryption time for IDEA
1	14337	1.10989011	2.4725274725
2	73710	4.50149011	12.6923076923
3	131776	10.3736264	22.5274725275
4	14983	1.39989011	2.5824175824
5	48430	3.39989011	8.3516483516
6	114688	6.18468112	17.5274725275
7	10752	1.10989011	1.6483516484
8	18432	1.89989011	2.8021978022
9	9029	1.00890115	1.3736263736
10	14592	1.39989011	2.2527472527
11	1272	0.05494505	0.2197802198
12	9522	1.10989011	1.4835164835

A graphical representation for the table 4.2.3 is shown in figure 4.2.1 with continues line and dotted line for encryption time of BSSPSDF and IDEA, respectively. According to the graph, there is a tendency that encryption time for BSSPSDF and IDEA increases with file size. But required time for the encryption through BSSPSDF is much smaller than encryption time for IDEA.

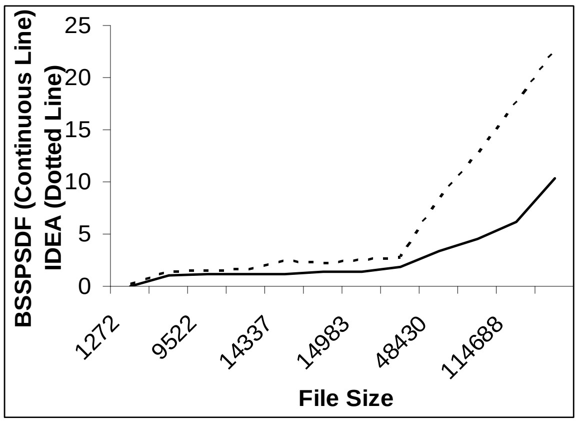

Figure 4.2.1
Relationship between Encryption Time of BSSPSDF and IDEA

*The observations were made using personal computer with specifications of 128 MB SD RAM, 1.5 GHz. processor and with Windows XP as the platform.

3. Pearsonian Chi-Square Value:

We check the non-homogeneity of the source file and the corresponding encrypted file through the "Pearsonian Chi-Square Value", also being termed as "Goodness-of-fit chi-square test", with the formula $\lambda^2 = \Sigma \{(f_o - f_e)^2 / f_e\}$, where f_e and f_o respectively being frequency of a character in source file and that of the same in the corresponding encrypted file

Table 5.1
Chi-Square Value for BSSPSDF and IDEA

Sl. No.	Chi-Square Value for BSSPSDF	Chi-Square Value for IDEA
1	12703.404583	11954.841494
2	65300.567094	61455.579243
3	118001.859896	111446.333333
4	35156.146845	33284.270177
5	43600.152096	40947.847106
6	42017959.919497	21150819.758794
7	328456.244148	229908.574337
8	658661.149455	477534.450116
9	624586.280769	14823.865385
10	246783.852664	204617.010065
11	1395.301800	1334.211477
12	214286.063093	15698.555711

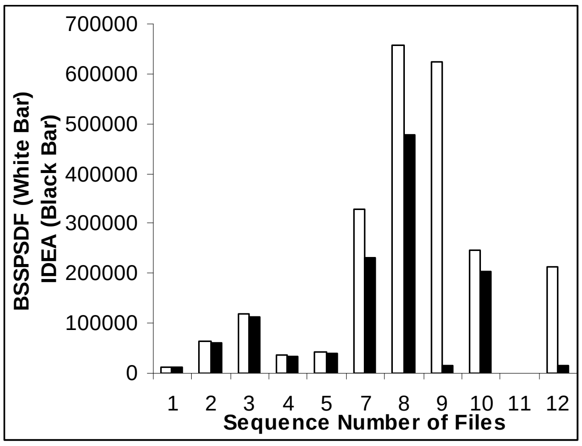

Figure 4.2.2
Comparison between Chi-Square value BSSPSDF (White Bar) and IDEA (Black Bar)

Chi-square value of BSSPSDF and IDEA for the set of same files which are used in table 4.2.1 with maintaining the Sl. No. for the respective files, have been compared in table 4.2.4. Among them figure 4.2.2 shows comparison of eleven files, except Calc.exe. As chi-square values of BSSPSDF and IDEA for Calc.exe are respectively 42017959.919497, 21150819.758794 which are very high with respect to other files, so for clear display of figure 4.2.2 comparisons on calc.exe file is not in the figure. White bar shows chi-square value of BSSPSDF of those eleven files, while black bar are been used for chi-square value of IDEA. That indicates chi-square value for the technique is generally higher than IDEA.

Apart form these there comparative observations; observation also on the following was made:

Graphical Test for Frequency Distribution :

We check the degree of security of the proposed protocol against the cryptanalytic attack using frequency distribution, where the frequency of the all 256 characters in source file and their corresponding encrypted file are compared graphically and we want to observe whether the exists any fixed relation ship between of the a character in both source and encrypted file.

To established this relationship between plain text and cipher text, figure 4.2.3 shows the distribution of the frequency (along Y axis) of the set of characters with their ASCII value (along X axis) of arbitrarily chosen a filer from table 4.2.1, ansi.sys and its encrypted file. Blue pillar represents the frequency of characters appeared in source file or plain text whereas red pillar is measured the frequency of characters in encrypted file or cipher text. From the figure 4.2.3 it is clearly shown that frequency of characters in plain text and cipher text are well distributed.

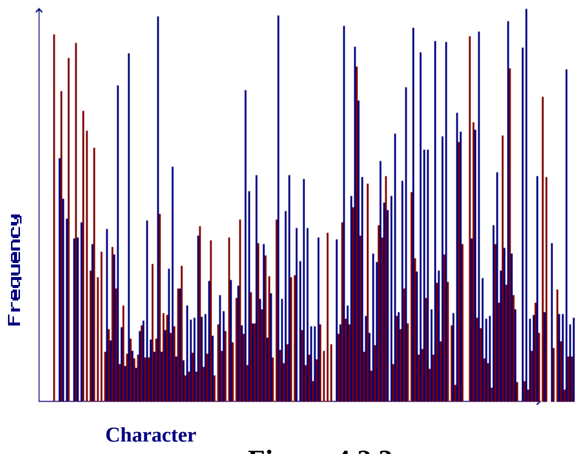

Figure 4.2.3
Frequency Distribution of Characters in "ansi.sys" and Corresponding Encrypted File

5. Analysis and Conclusion

Using this BSSPSDF private key technique, we can encrypt any size of file, as well as any kind of file, as BSSPSDF protocol is implemented in bit-level. This protocol not only encrypts the source bit-stream but the protocol is storage efficient also. Obviously the rate of compression will depend on the content of file. If number of distinct blocks appeared in source file is at least 2 less than number of all possible combinations of blocks, with that particular block size, there are at least 2 blocks for which compression will occur for 1 bit. Necessity of insertion of a few dummy bits to make the target file of size as the multiple of 8 bits makes the practical rate of compression as lesser than the rate of compression to be achieved theoretically.

Encryption with compression can also be achieved without sorting the distinct appeared blocks in plaintext with their frequency. But in an attempt to increase the rate of compression during encryption without hampering the level of security, we sort the distinct appeared blocks with their frequency in ascending order. There is a probability of having lesser number of bits in replaced codes for those original codes, which appear at the bottom of the sequence of the set of distinct appeared blocks from the plaintext. So for sorting in ascending order of distinct blocks with frequency of appearing, more frequently appeared blocks come in bottom of the sequence. Accordingly, in encrypted text more original blocks are replaced by the corresponding Replaced Codes which contain lesser bits than respective Original Codes. Obviously the rate of compression during encryption will be higher.

If difference of number of all possible combinations of blocks of particular block-size and number of distinct blocks occurred in source bit- stream is very high, it will result in a higher rate of compression during encryption. For large block size, probability of getting that difference is also very high. So compression during encryption is highly probable. Accordingly, it requires a larger key space. A set of sizes for the key with the respective block sizes is shown in table 5.1.

Table 5.1
Key Size for different blocks

Block Size (L)	**Minimum Number bits to represent block size (d)**	**Number of Possible different blocks**	**Key size** $2d + (2^L + 1) \times L + 3$
4	3	16	77
6	3	64	399
8	4	256	2067

Additionally, if we use X-bit key, 2^X number of possible keys may be generated, from which only one option is used for correct decryption. So, complexity of key breaking increases with increasing value of X. A processor, capable of doing 10^6 encryptions per millisecond, requires 5.9×10^{30} years to break a 168-bit key. Encryption time is not also increasing with increased key size, shown in table 5.1. So, 399-bit or more than 399-bit key is recommended for correct implementation of the technique [11].

For large block size, probability of rate of compression at the time of encryption is high. For bigger block size, more compression is achieved and key breaking is practically impossible. Due to that large block size is recommended for increasing complexity and getting more efficient effect. So this algorithm ensures high security during minimum overhead through network [2] [3] [5] [11].

Acknowledgement:

Let us express our heartiest gratitude to respective authorities of Dr. B. C. Roy Engineering College, Durgapur, West Bengal, INDIA and National Institute of Technology (NIT), Durgapur, West Bengal, INDIA for providing resources used during the entire development process.

References

[1] J. K. Mandal, S. Dutta, “A 256-bit recursive pair parity encoder for encryption”, Advances D -2004, Vol. 9 nº1, Association for the Advancement of Modelling and Simulation Techniques in Enterprises (AMSE, France), www. AMSE-Modeling.org, pp. 1-14

[2] Pranam Paul, Saurabh Dutta, “A Private-Key Storage-Efficient Ciphering Protocol for Information Communication Technology”, National Seminar on Research Issues in Technical Education (RITE), March 08-09, 2006, National Institute of Technical Teachers’ Training and Research, Kolkata, India

[3] Pranam Paul, Saurabh Dutta, “An Enhancement of Information Security Using Substitution of Bits Through Prime Detection in Blocks”, Proceedings of National Conference on Recent Trends in Information Systems (ReTIS-06), July 14-15, 2006, Organized by IEEE Gold Affinity Group, IEEE Calcutta Section, Computer Science & Engineering Department, CMATER & SRUVM Project-Jadavpur University and Computer Jagat

[4] Dutta S. and Mandal J. K., “A Space-Efficient Universal Encoder for Secured Transmission”, International Conference on Modelling and Simulation (MS’ 2000 – Egypt, Cairo, April 11-14, 2000

[5] Mandal J. K., Mal S., Dutta S., A 256 Bit Recursive Pair Parity Encoder for Encryption, accepted for publication in AMSE Journal, France, 2003

[6] Dutta S., Mal S., “A Multiplexing Triangular Encryption Technique – A move towards enhancing security in E-Commerce”, Proceedings of IT Conference (organized by Computer Association of Nepal), 26 and 27 January, 2002, BICC, Kathmandu

[7] William Stallings, Cryptography and Network security: Principles and practice (Second Edition), Pearson Education Asia, Sixth Indian Reprint 2002.

[8] Atul Kahate (Manager, i-flex solution limited, Pune, India), Cryptography and Network security, Tata McGraw-Hill Publishing Company Limited.

[9] Mark Nelson, Jean-Loup Gailly, The Data Compression Book. BPB Publication

[10] S Mal, J K Mandal and S Dutta, "A Microprocessor Based Encoder for Secured Transmission", Conference on Intelligent Computing on VLSI, Kalyani Govt. Engineering College, 1-17 Feb, 2001, pp 164-169

[11] Saurabh Dutta, "An Approch Towords Development of Efficient Encryption Technique", A Doctoral thesis submitted to the university of North Bengal for the Degree of Ph.D., 2004

[12] Pranam Paul, Saurabh Dutta, A. K. Bhattacherjee, "An Approach to ensure Security through Bit-level Encryption with Possible Lossless Compression", International Journal of Computer Science and Network Security, Vol. 8 No. 2, pp 291 – 299.

Pranam Paul is a Lecturer of Dr. B. C. Roy Engineering College, Durgapur, West Bengal, INDIA in the department of Master in Computer Application. He had completed his master degree in Computer Application in 2005 under West Bengal University of Technology, INDIA. He was lecturer in IT department of MCKV Institute of Engineering College, Liluah, West Bengal, INDIA. Then he had joined in MCA department in Bengal College of Engineering and Technology, Durgapur in same post. Now he is a registered Ph.D. scholar in Electronic and Communication Engineering department of National Institute of Technology, Durgapur in the field of Cryptography and Network Security.
He has total 11 publications in conference proceedings and journals of national and international level except this one.

Saurabh Dutta is Ph.D. (Computer Science) form University of North Bengal, India. His doctoral work was related to the cryptography. He was awarded Ph.D. 2006 for his dissertation entitled "An Approach Towards Development of Efficient Encryption Techniques". Having around 30 publications in conference proceedings and journal of national and international levels, he is involved in supervising scholars in cryptography-related areas. Having been in teaching profession for last 9 years, currently he is Associate Professor and Head in Department of Computer Application in Dr. B. C. Roy Engineering College, Durgapur-06, INDIA.

A. K. Bhattacharjee did his Ph.D. in Engineering from Jadavpur University, Kolkata, India in 1989. Presently he is associated with Department of Electronics and Communication Engineering in National Institute of Technology (NIT), Durgapur, INDIA as Professor. A B. E. (1983) from the then Shibpur B. E. College, INDIA, he did his M. E. from Jadavpur University, INDIA. Senior academicians, having been involved in teaching and research for last 20 years, his current areas of research are Microstrip Antenna and cryptography.