An Approach to ensure Security through Bit-level Encryption 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 block cipher based on private key to be implemented in bit-level. The scheme used in the proposed cipher is substitution-based. Encryption through this proposed cipher also results in possible lossless compression. Efficiencies of the proposed cipher are observed and compared with the existing cipher IDEA on execution time, rate of compression achieved and chi-square value. Graphically frequency-distribution of characters in source and corresponding encrypted file are observed for a sample file. On the basis of all observations made, proposed cipher is found to be efficient. Requirements of key space of least 166-bit makes it invulnerable to attacks. It is highly compatible with existing cipher like IDEA.

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

1. Introduction

A ciphering protocol not vulnerable to cryptanalytic attacks is considered to be a strong tool in ensuring security. If a lossless compression is achieved while encrypting using such a protocol, an extra flavor of effectiveness is added to it. The paper presents a block-cipher, BSDMB, "Block Substitution Differentiation for Minimum-valued Block". The BSDMS protocol can encrypt file of any size and any type, and operates in bit-level. Its performances have been observed and compared with-known IDEA, International Data Encryption Algorithm to achieve a potentially satisfactory level of efficiency.

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 with a conclusion.

2. The Scheme

This section presents a description of the actual scheme, used during implementing BSDMB 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. Section 2.1 describes the scheme used in ciphering technique, while Section 2.2, formation of key is discussed and in section 2.3, describes the scheme used in deciphering technique [1] [2] [3] [4] [9].

2.1 Stepwise Presentation of Ciphering Model

At first, source file is converted 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-bit blocks, a blocks, less than L bits is left at last, say U_B (means length of $U_B < L$) which is kept unchanged during encryption.

Let an intermediate cipher bits stream, E be null at initial stage.

Step 1: We find out minimum and maximum valued block among all blocks of L bits length in source bit-stream, say "min" and "max" respectively. We keep min into Key.

Step 2: Let D = max – min. We find out the number of bits in D, say t, after which first 1 is appeared in D from left most bit. Let

EL be maximum required bit to represent encrypted blocks. So EL = L – t. This EL is also kept into Key.

Step 3: We calculate number of bits in D from $(t + 1)^{th}$ bit, until first appearance of 0, say CB. So CB numbers of bit in D from $(t+1)^{th}$ are all 1. This CB is also kept in Key.

Step 4: We again calculate the numbers of bits in D, say t_1, after which first 1 is appeared in D from $(t+CB)^{th}$ bit. Let us calculate $C\_B = L – (t + CB + t_1)$ and t_1 is also kept in key.

Step 5: We take first L-bit from source bit-stream and assigned it into B.

Step 6: First t bits are left from B at beginning (means from MSB bit) and next CB bits from B are assigned into TB.

If all bits in TB are 1,

Next t_1 bits from B are not considered.

After that, next C_B bits from B are appended with TB at end.

Else (at least one 0 is in TB)

Next (EL – CB) bits are appended with TB at end.

End if

Step 7: Now TB is concatenated with E at last and recently generated E is used as intermediate ciphered bit-stream for next round.

Step 8: We take next L-bit from source bit-stream in B and go to Step 6. This process will continue till the end of the ciphered bit-stream and finally, we get intermediate cipher bit-stream, E.

Step 9: 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, E. Accordingly from cipher bit-stream, cipher text will be generated.

2.2 Proposed Structure of 166-bit Key

For this algorithm key consists of 7 segments.

If L-bit block is taken for the encryption of source bit-stream, let us find out an integer; say d for which $2^{d-1} < L\text{-}1 \leq 2^d – 1$. Key structure is shown in figure 2.2.1.

Segment	Size	Description
1st	d	Binary form of L with d bits representation.
2nd	d	d bits binary form of number of unprocessed bits
3rd	3	3 bits binary form of number of dummy 0 (ZERO) to be added for making total encrypted string length as multiple of 8.
4th	L	L-bit minimum valued blocks, that is min
5th	d	Minimum required bit to represent (maximum valued block – min)
6th	d	Numbers of compared bits, in which all are 1 for compressed blocks, otherwise there is at least one 0.
7th	d	Numbers of compressed bits in each compressed of block.

Total Size of Key: 5d + L + 3

Figure 2.2.1
Key Structure

We implement the protocol with decomposition of plane text into some blocks having each 128-bit length. So L = 128 and $2^{7-1} < 128 – 1 \leq 2^7 – 1$, so d = 7. For the cause, $5 \times 7 + 128 + 3 = 166$- bit key is generated. On the discussion in section 5, less than 166-bit key in length may be generated, but the small key is very easy to break. So, we considers only 166-bit or greater than 166-bit key for this protocol.

2.3 Stepwise Presentation of Deciphering Model

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

From key with the reference of figure 2.2.1, receiver comes to know source block length, say L, length of unchanged block, say N_{UB}, number of dummy 0s, say D_0, minimum valued block with L bits, say “min”, number of compared bits i.e. CB

and number of compressed bits, say t_1 and RB that is minimum required bits to represent deference between maximum valued block and "min" during encryption.

Let an incomplete deciphered bit- stream, IDB be null initially.

Step 1: We convert binary form of cipher text and get target bit-stream. Then we take N_{UB} bits from beginning of encrypted bits stream, say U_B and leave next D_0 bits.

Step 2: Next CB bits stream from target bit-stream are taken.

Step 3: If there is at least one 0 (ZERO) in those CB bits stream, we take next (RB – CB) bits and append at last of those CB bits steam, it is treated as T, a temporary bit-stream.

Else (means all CB bits are 1 or CB = 0)

We take next C_B = {RB – (CB + t_1)} bits.

Now we concatenate with the sequence CB bits, t_1 numbers of 0s and recently taken C_B bits. It is treated as T, a temporary bit-stream.

End if

Step 4: We append (L – RB) numbers of 0s at left position with T, say TB.

Step 5: Let A = min + TB. Now A is concatenated with IDB at right most end and recently generated IDB is used as incomplete deciphered bit-stream for next round.

Step 6: Go to Step 2. This process will continue till the end of the target bit-stream.

Step 7: Finally, we concatenate U_B with incomplete deciphered bit-stream, IDB at the end and generate complete deciphering bits stream as well as also generate deciphering text.

3. An Implementation

We consider a small plaintext P as "e**ncrypt**". The stream of bits, S, representing P is as follows:

01100101011011100110001101110010011110010111000001110100

Now S is decomposed into some blocks with 16 bits length, so refer to section 2.1, L = 16. These are 0110010101101110, 0110001101110010, 0111100101110000 and 01110100 that last block is remained unchanged, say U_B. In section 3.1, encryption is been discussed while section 3.2 is used to discuss about key formation and at last, how correct decryption is been done, discussed in section 3.3.

3.1 Encryption

We find out minimum and maximum valued blocks from set of decomposed block, say "min" and "max" respectively.

min = 0110001101110010

max = 0111100101110000

D = max – min = 0001010111111110

EB = Required Minimum bit to represent D = 13

CB = Number of Compared Bits = 1

t_1 = Number of Compressed Bits = 1

Process of encryption for each block except unchanged block, U_B is shown in table 3.1.1

Table 3.1.1
Difference of Each Block

Source Bit Block. Let's consider this column as A	**Difference of block from D (A – D) Let's consider this column as B**	**Leave (L – EB) i.e. 3 bits from left most end of B Let's consider this column as T**	**If first CB i.e. 1 bits all are 1, leave next t_1 i.e. 1 bit and take next {EB – (CB+t_1)} i.e. 11 bits, else take next (EB – CB) i.e. 12 bits**	**Bits in target block**
0110010101101110	0000000111111100	0000111111100	00001111111100	13
0110001101110010	0000000000000000	0000000000000	00000000000000	13
0111100101110000	0001010111111110	1010111111110	11011111111110	12

So the intermediate target bit-stream is 00001111111000000000000000110111111110, say E, length of which is (13+13+12) = 34 bits. Length of U_B is 8 bits so the total length of together E and U_B is 46 bits. That means 2 dummy bits i.e. 0 are required to make length of 46-bit-stream, multiple of 8. So concatenating with the sequence U_B, 2 dummy 0 s and E, target bit-stream is

01110100**00**00001111111000000000000000011011111111110. The generated equivalent plain text is "t♥° ♪■".

3.2 Key Generation

For this particular example, source bit-stream is decomposed into some blocks, having equal length 16 bits. Refer to section 2.2, L = 16 and 4 i.e. d bits are required to represent binary form of (16-1) = 15. Key generation has been shown table 3.2.1.

Table 3.2.1
Replacing Code for distinct blocks

Segment (in MSB-to-LSB direction)	Description of the segment	Number of required bits	Significant Bits
1	Length of the blocks	4	1111
2	Number of bits in unchanged block	4	1000
3	Number of dummy bits	3	010
4	Minimum valued block	16	01100011 01110010
5	Required minimum bits to represent (Max – Min) valued block	4	1100
6	Number of compared bits	4	0001
7	Number of compressed bits	4	0001

So for this particular example, 5×4+16+3 = 39 bits private key will be 111110000100110001101110010110000010001.

3.3 Decryption

After receiving encrypted text, "t♥°♪■" and key following will be performed to get back original text.

Step 1: From 1st, 2nd, 3rd, 4th, 5th, 6th and 7th segments of key, block length (say L), number unchanged bits (say N_{UB}) and number of added dummy bits (say D_0), minimum valued blocks (say "min"), required minimum bits to represent difference each block from "min" (say RB), number of compared bits (say CB) and number of compressed bits (say t_1) are come to know.

Here L = 16, N_{UB} = 0, D_0 = 3, RB = 11, CB = 1, t_1 = 1 and min = 0110001101110010.

Step 2: 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 = 01110100

Then total decryption process for the target blocks, 00001111111000000000000000110111111110 is shown in table 3.3.1.

Table 3.3.1
Decryption Process

ROUND	Take next CB (=1) bits Let's consider this column as A	If first CB bits all are 1, concatenate t_1 i.e. 1 number of 0 and take next {EB – (CB+t_1)} i.e. 11 bits with A, else take next (EB – CB) i.e. 12 bits with A Let's consider this column as B	Add B with min, means B + min Let's consider this column as C	After Concaten-ating with all rows of column C, Interme-diate Decrypted bit-stream is generated
1	0	00001111 11100	01100101 01101110	01111001 01110000
2	0	00000000 00000	01100011 01110010	01100011 01110010
3	1	1**0**101111 11110	01111001 01110000	01100101 01101110

From table 3.3.1, being concatenated with the sequence received intermediate decrypted bits stream, 0100010101101110011000110111001001111001011100000111010 0 and unchanged block, U_B = 01110100, decrypted bit-stream, 0100010101101110011000110111001001111001011100000111010001110100 is generated; text format of which is "enc**rypt**" which is same as plane text.

4. Results

Here we have compared BSDMB with International Data Encryption Algorithm (IDEA)

to establish a comparative analytical report that is helpful to understand strength and weakness of BSDMB. 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 … 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].

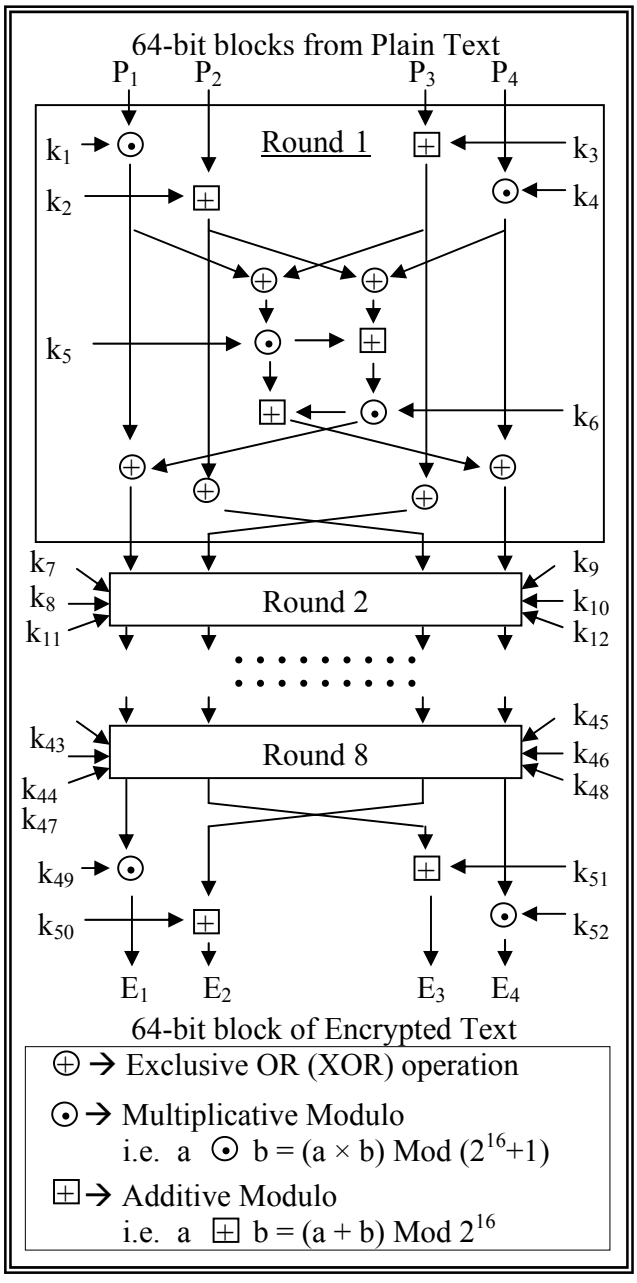

Figure 4.1.1
Process of Encryption through IDEA

4.2 Comparative Report

Both the proposed BSDMB protocol (with 128-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. **Percentage of Compression achieved :** *To check storage efficiency after a been encrypted*

Table 4.2.2 shows the compression between proposed technique BSDMB 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 BSDMB. Serial numbers for table 4.2.1 and table 4.2.2 are same for corresponding files.

Table 4.2.2
Relationship between Source File Size and Encrypted File Size

Sl. No.	For BSDMB		For IDEA	
	Encrypted File Size	Rate of Compression	Encrypted File Size	Rate of Compression
1	14159	1.241%	14337	0.000%
2	72794	1.242%	73710	
3	130143	1.239%	131776	
4	14866	0.781%	14983	
5	47830	0.000%	48430	
6	114688	0.000%	114688	
7	10752	0.000%	10752	
8	18432	0.000%	18432	
9	9028	0.011%	9029	
10	14592	0.000%	14592	
11	1265	0.550%	1272	
12	9520	0.021%	9522	

2. **Encryption and Decryption Time* :** *To evaluate computational overhead*

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

Table 4.2.3
Encryption Time for BSDMB and IDEA

Sl. No.	File Size	Encryption time for BSDMB	Encryption time for IDEA
1	14337	0.1098901099	2.4725274725
2	73710	0.8241758242	12.6923076923
3	131776	1.3736263736	22.5274725275
4	14983	0.1648351648	2.5824175824
5	48430	0.5494505495	8.3516483516
6	114688	1.2087912088	17.5274725275
7	10752	0.1098901099	1.6483516484
8	18432	0.2197802198	2.8021978022
9	9029	0.1098901099	1.3736263736
10	14592	0.1648351648	2.2527472527
11	1272	0.0549450549	0.2197802198
12	9522	0.1098901099	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 BSDMB and IDEA, respectively. According to the graph, there is a tendency that encryption time for BSDMB and IDEA increases with file size. But required time for the encryption through BSDMB is much smaller than encryption time for IDEA.

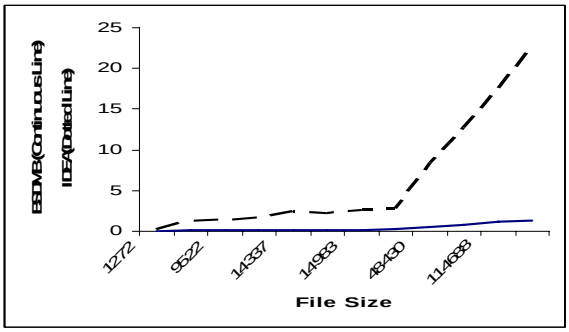

Figure 4.2.1

Relationship between Encryption Time of BSDMB 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: T***o check the non-homogeneity of the source file and the corresponding encrypted file and 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 BSDMB and IDEA

Sl. No.	Chi-Square Value for BSDMB	Chi-Square Value for IDEA
1	2703.404583	11954.841494
2	65300.567094	61455.579243
3	118951.809896	111446.333333
4	35151.146255	33284.270177
5	43693.152096	40947.847106
6	52617929.919497	21150819.758794
7	329746.244148	229908.574337
8	750661.149455	477534.450116
9	924586.280769	14823.865385
10	246543.852664	204617.010065
11	395.302800	1334.211477
12	219756.063093	15698.555711

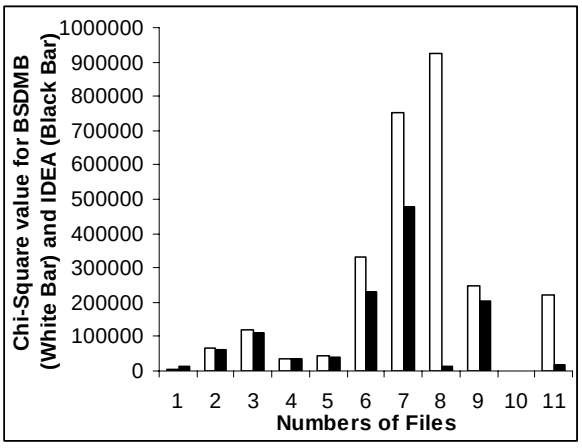

Figure 4.2.2
Comparison between Chi-Square value BSDMB (White Bar) and IDEA (Black Bar)

Chi-square value of BSDMB 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 BSDMB and IDEA for Calc.exe are respectively 52617929.919497, 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 BSDMB 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: *To test the degree of security of the proposed protocol against the cryptanalytic attack, where the frequency of the all 256 characters in source file and their corresponding encrypted file are compared graphically, the observation being to show 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.

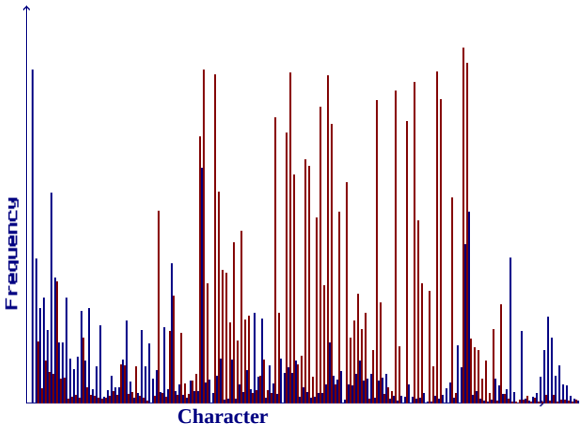

Figure 4.2.3
Frequency Distribution of Characters in "ansi.sys" and Corresponding Encrypted File

5. Analysis and Conclusion

Using this BSDMB private key technique, we can encrypt any size of file, as well as any kind of file, as BSDMB protocol is implemented in bit level. This protocol not only encrypts the source bit-stream but the protocol is storage efficient also, which means that encrypted file size is less than or equal to the source file size, but obviously rate of compression will depend on context of file. If there is at least one 0 in the difference between maximum valued block and minimum valued block in source file, minimum 1 bit compression should be achieved. Until minimum 8 bits compression is occurred during encryption, occurrence of compression in file size, which is a byte format, is not reflected into byte, because if any achieved compression is less than 8 bits, dummy bits should be required to make total length of file, multiple of 8 bits for converting bit format to byte format.

If total range, in which all appeared blocks belong, is small, rate of compression is high, which means that rate of compression is inversely proportional to the span of appeared blocks. That is not depending on blocks size. For large block size, decryption complexity is increased. On the basis of these two logics, for more compression and more security, large blocks size is recommended. So accordingly, key size will be 5d + L +3 where L-bit be the block length, d is the minimum required bit to present L – 1. Some key size and their corresponding block size with their encryption time for the particular file, calc.exe which is arbitrarily chosen from table 4.2.1 are shown in table **5**.1.

Table 2.2.1
Key Size for different blocks

Block Size (L)	Minimum Number bits to represent block size (d)	Key size 5d + L + 3	Encryption Time
16	4	39	1.2637362637
32	5	60	1.2087912088
64	6	97	1.2087912088
128	7	166	1.2087912088
256	8	299	1.2087912088
512	9	560	1.2087912088
1024	10	1077	1.2087912088

Additionally, if we use X-bit key, 2^X number of key may be generated, from which only one option is used for correct decryption. So, complexity of key breaking increases according to increased value of X. Refer to the references' number 11, 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, 166-bit or more than 166-bit key is recommended for correct implementation of the technique.

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 thesis submitted to the university of North Bengal for the Degree of Ph.D., 2004