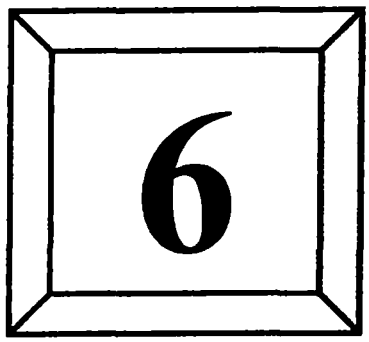


Encryption Through Recursive Substitutions of Bits Through Prime-Nonprime (RSBP) Detection of Sub-stream

	Contents	Pages
6.1	Introduction	197
6.2	The Scheme	198
6.3	Implementation	206
6.4	Results	217
6.5	Analysis and Conclusion including Comparison with RPSP, TE, RPPO, RPMS	231

6.1 Introduction

The Recursive Substitutions of Bits through Prime-nonprime (RSBP) detection of sub-stream is a completely different technique in comparison to the other proposed techniques because this is the only proposed technique, in which in anticipation of getting the maximum security a little storage overhead has been accepted.

Like the RPSP technique, described in chapter 2, in this RSBP technique also the basic operation to be performed is prime-oriented. So, in this regard, this technique is quite different from the RPPO technique, described in chapter 4, and the TAE/TSE technique, described in chapter 3, because in those two techniques the basic operations performed were Boolean operations. Also, unlike the RPSP technique and the RPPO technique, here there is no formation of cycle. That means, directly by applying this technique once the target file can be generated from the source one.

Since, like all the other proposed techniques, this is also a bit-level technique, the source file is to be converted into a stream of bits. To avoid the increasing complexity of the implementation, it is suggested to decompose the stream into blocks of unit length. For a block of bits, the corresponding decimal number is to be calculated, like the RPMS technique, described in chapter 5. Depending on whether that decimal number is prime or not, a corresponding 1-bit code value is assigned for that block and, to identify the source block uniquely, a rank value is also assigned to the block.

To form the encrypted file, code values of all the source blocks are put together first, followed by all the rank values, preferably in the reverse sequence. In between these two, a few extra 0's may have to be inserted to form size of the encrypted file as a multiple of 8 bits.

During the process of decryption, the key of the technique helps in getting the unique block size and hence the total number of blocks, assuming that the size of the source file is known to the receiver. From these, it is to be calculated the possible length of the rank values corresponding to each of the two code values. Combining these, all the source blocks are generated one by one [3, 29, 30, 52, 53].

Section 6.2 discusses the scheme. The technique is implemented for a certain text in section 6.3. Section 6.4 enlists the results from different perspectives after the technique is implemented for the same groups of files that were considered for the other

proposed techniques. An analytical evaluation of the technique is presented in section 6.5.

6.2 The Scheme

Like all the other techniques proposed, this RSBP technique is also of bit-level implementation. Naturally, given a source file to be encrypted using this technique, the first task to be performed is to obtain the corresponding stream of bits. Section 6.2.1 discusses how this source bit of stream is encrypted and section 6.2.2 discusses the process of decryption to regenerate the original stream.

6.2.1 The Encryption Technique

One major difference of this technique with all the other proposed techniques is that here corresponding to one block, there is no so called "target block". It is because of the fact that for each block there is one 1-bit code value and one rank value, and these two are not put together. For ensuring the error-free decryption, the code values of all the blocks are put together one by one as to start from the MSB position and the corresponding rank values are put together one by one from last as to start from the LSB position. Between these two, a few extra bits, preferably 0's, the number of which should not exceed 7, may have to be inserted, so that the size of the encrypted stream becomes a multiple of 8.

Following is the stepwise approach to be followed to encrypt the source stream of bits.

Step 1: Decompose the source stream, say, into a finite number of blocks, each preferably of the same size, say, L.

Step 2: Calculate the total number of primes and nonprimes in the range of 0 to (2^L-1). Accordingly, find minimum how many bits are required to represent each of these two numbers.

Step 3 to step 5 are to be applied for all the blocks.

Step 3: For the block under consideration, calculate the decimal number corresponding to that. Say, it is D.

Step 4: Find out if D is prime or nonprime. If D is prime, the code value for that block is 1 and if not so, it is 0.

Step 5: In the series of primes or nonprimes (whichever be applicable for D) in the range of 0 to (2^L-1), find the position of D. Represent this position in terms of binary values. This is the rank of this block.

After repeating these steps (3, 4 and 5) for all the blocks, following steps are to be followed.

Step 6: Say, there are N number of blocks. In the target stream of bits, put all the N code values one by one starting from the MSB position. So, in the target stream, the first N bits are code values for N blocks.

Step 7: For putting all the rank values in the target stream, we are to start from the N^{th} bit from the MSB position and then to come back bit-by-bit. Immediately after the N^{th} bit, put the rank value of the N^{th} block, followed by the rank value of the $(N-1)^{th}$ block, and so on. In this way, the rank value of the first block will be placed at the last.

Step 8: Combining all the code values as well as the rank values, if the total number of bits in the target stream is not a multiple of 8, then to make it so, at most 7 bits may have to be inserted. Insertion of these extra bits is to be started from the $(N+1)^{th}$ position. So, a maximum of 7 right shifting operations may have to be performed in the $(N+1)^{th}$ position, where that many 0's are inserted.

Consider a stream S=1010100101010010 of only 16 bits. We apply these steps to get the target stream T corresponding to S using the RSBP technique.

Following step 1, we decompose S into four 4-bit blocks taking bits four by four from the MSB, which are D_1=1010, D_2=1001, D_3=0101 and D_4=0010. So, as per step 1, L=4.

Following step 2, we find that the total number of primes in the range of 0 to 2^4-1=15 is 6 (2, 3, 5, 7, 11, 13) and that of nonprimes is 10 (0, 1, 4, 6, 8, 9, 10, 12, 14, 15). So, to represent the position of a prime number the number of bits required is 3,

because since there are 6 primes, their positions range from 0 to 5, Similarly, to represent the position of a nonprime number the number of bits required is 4 because their positions range from 0 to 9 as there are 10 nonprime numbers.

Now, we apply the next three steps (3, 4, 5) for blocks D_1, D_2, D_3 and D_4.

The decimal equivalent of D_1=1010 is 10, which is the 6^{th} nonprime. So, the code value of D_1 is C_1=0 and the rank is R_1=0110.

The decimal equivalent of D_2=1001 is 9, which is the 5^{th} nonprime. So, the code value of D_2 is C_2=0 and the rank is R_2=0101.

The decimal equivalent of D_3=0101 is 5, which is the 2^{nd} prime. So, the code value of D_3 is C_3=1 and the rank is R_3=010.

The decimal equivalent of D_4=0010 is 2, which is the 0^{th} prime. So, the code value of D_4 is C_4=1 and the rank is R_4=000.

Following step 6 and step 7, to form the target stream, first we put all the code values one by one starting from the MSB position to get 0/0/1/1 and they are followed by the rank values of all the blocks starting from the last, i.e., 000/010/0101/0110. Here "/" works just as the separator. So, combining these code values and rank values we obtain 0011000010010101110, a stream of length 18.

Following step 8, to make the length a multiple of 8, a block "000000" is to be inserted between the code values and the rank values, so that the stream 0011/000000/000010010101110 is formed. Therefore corresponding to the 16-bit source stream S=1010100101010010, the 24-bit target stream is as follows:

T=001100000000001001010110.

Figure 6.2.1.1 gives a pictorial representation of this example.

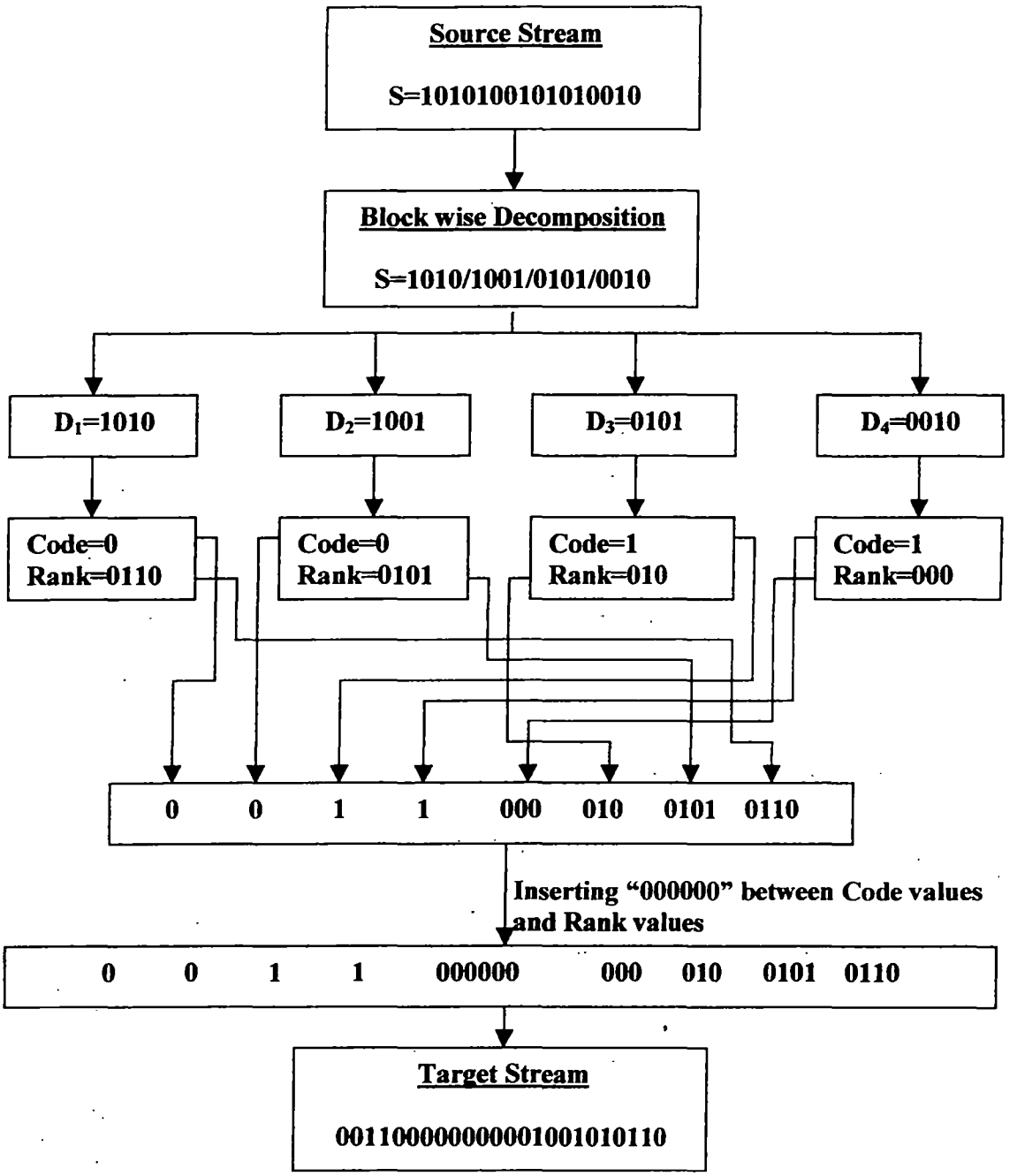


Figure 6.2.1.1
Pictorial Representation of Encrypting S=1010100101010010 using RSBP Technique

6.2.2 The Decryption Technique

In the policy of encryption, the strategy of distributing the code values and the rank values of all the source blocks was adopted with the objective of having an unambiguous decryption. As the result of that, the policy of decryption discussed in this section offers an absolute correct outcome.

Following are the steps to be followed during the decryption:

Step 1: Get the unique block length from the key. Say, it is L.

Step 2: Calculate the total number of blocks generated from the source stream of bits. This calculation is done by the following:

Total Number of Blocks (B) = Source Stream Size / Unique Block Length, "/" denoting the integer division.

So, the first B number of bits, starting from position 0 (MSB position) to position (B-1) in the encrypted stream denotes the code values of B blocks.

Step 3: Calculate the total number of primes in the range of 0 to (2^L-1). Say, it is P. Hence calculate how many maximum bits are required to express P in binary form. Say, it is X. Then $X = \lfloor \log_2 P \rfloor + 1$, where $\lfloor \log_2 P \rfloor$ denotes the integral part of $\log_2 P$.

Step 4: Calculate the total number of nonprimes in the range of 0 to (2^L-1). Say, it is Q. Hence calculate how many maximum bits are required to express Q in binary form. Say, it is Y. Then $Y = \lfloor \log_2 Q \rfloor + 1$, where $\lfloor \log_2 Q \rfloor$ denotes the integral part of $\log_2 Q$. It is mentionable here that $Q = 2^L - P$.

Step 5: Consider the MSB. It is the code value of the first source block.

If MSB=1,

1. Consider the last block of X bits, convert the binary number represented by this block of bits into the corresponding decimal, Say, it is M. Mark this block as being processed.

2. Find the M^{th} prime number in the series of natural numbers (with the assumption that the position of the first prime number is 0, not 1).

3. The L-bit binary number corresponding to the decimal prime number obtained in 2 is the first source block.

4. Mark the MSB as being processed.

If MSB=0,

1. Consider the last block of Y bits; convert the binary number represented by this block of bits into the corresponding decimal, Say, it is M. Mark this block as being processed.

2. Find the M^{th} prime number in the series of natural numbers (with the assumption that the position of the first prime number is 0, not 1).

3. The L-bit binary number corresponding to the decimal nonprime number obtained in 2 is the first source block.

4. Mark the MSB as being processed.

Step 6: Repeat step 7 and step 8 for (B-1) number of times for the values of I ranging from 1 to (B-1) as there are (B-1) more blocks left to be considered. Set I = 1.

Step 7: Consider the I^{th} bit from the MSB position. Let it be denoted by T_I.

If $T_I = 1$,

1. Consider the first unprocessed block of P bits in the LSB-to-MSB direction, convert the binary number represented by this block of bits into the corresponding decimal, Say, it is M. Mark this block being processed.

2. Find the M^{th} prime number in the series of natural numbers (with the assumption that the position of the first prime number is 0, not 1).

3. The L-bit binary number corresponding to the decimal prime number obtained in 2 is the I^{th} source block.

If $T_I = 0$,

1. Consider the first unprocessed block of Q bits in the LSB-to-MSB direction, convert the binary number represented by this block of bits into the corresponding decimal, Say, it is M. Mark this block being processed.

2. Find the M^{th} nonprime number in the series of natural numbers (with the assumption that the position of the first prime number is 0, not 1).

3. The L-bit binary number corresponding to the decimal nonprime number obtained in 2 is the I^{th} source block.

Step 8: Let I = I + 1.

Step 9: Concatenate all the blocks obtained so far in the sequence of their generation and this is the source stream.

The length of the source stream is (L * B) and accordingly L_T – (L * B) number of 0's in the positions between the code values and the target values in the target stream will remain being unmarked, as these 0's were inserted at the end of the encryption process; L_T being considered as the length of the target stream.

Continue with the same example for illustration, where the target stream obtained was T = 001100000000001001010110.

Now, from step 1, from the key we get the unique block length L = 4.

Following step 2, we obtain the total number of blocks as B = 16 / 4 = 4, as it is assumed to be known to the receiver that the source stream before being encrypted was of

length 16 bits. Therefore in the encrypted stream, the first four bits are the code values of four blocks.

Following step 3, we calculate the total number of primes in the range of 0 to 15 (i.e., $2^4 - 1$), which is P = 6, and to represent it by a binary number the maximum number of bits needed is X = 3.

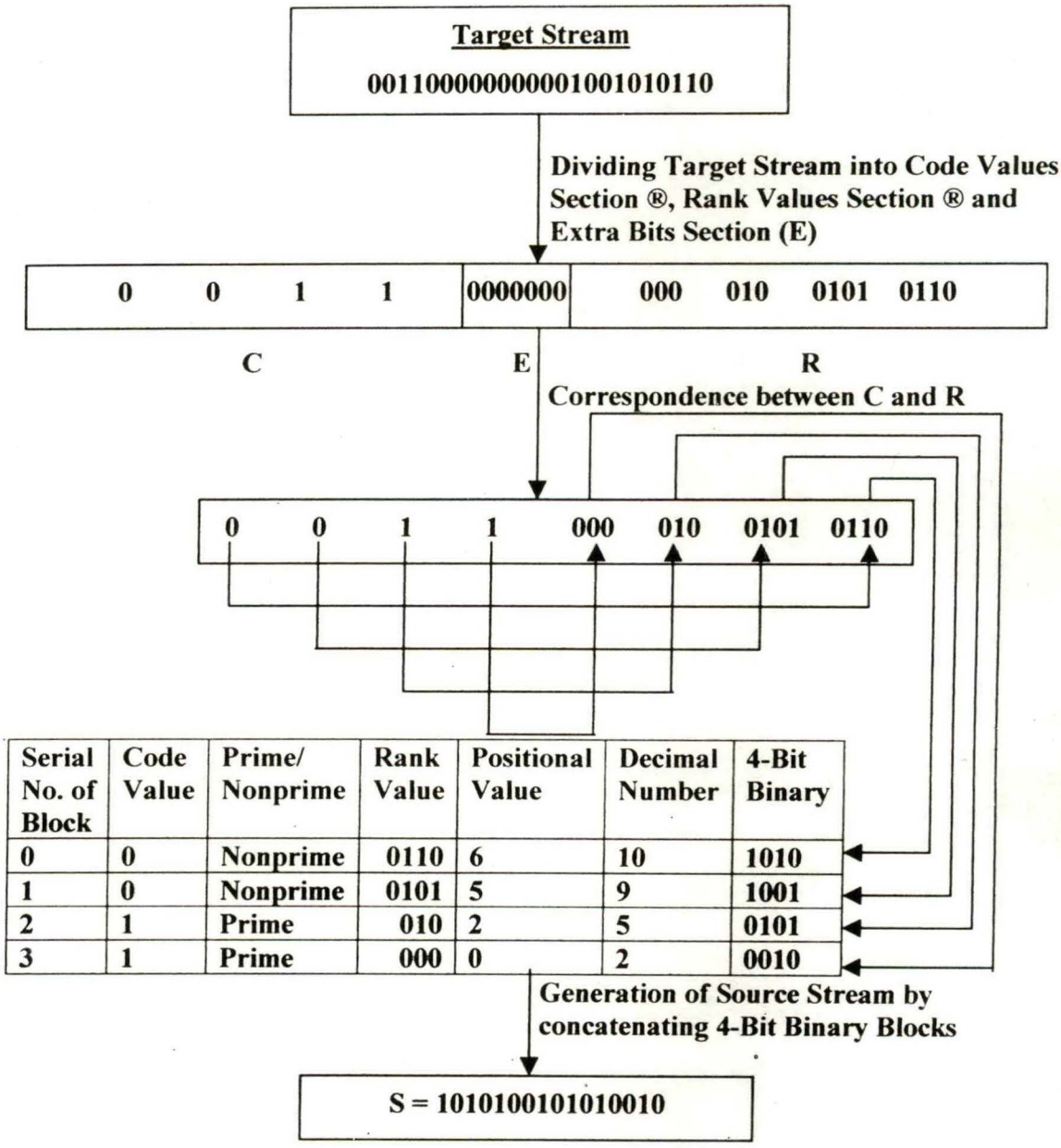
Similarly, following step 4, we calculate the total number of nonprimes in the range of 0 to 15 (i.e., $2^4 - 1$), which is P = 10, and to represent it by a binary number the maximum number of bits needed is Y = 4.

Now, following step 5, we find the MSB as 0, so that we are to consider the block of the last Y = 4 number of bits, which is 0110, the decimal of which is M = 6. So, we are to find the 6^{th} nonprime number in the series of natural numbers. It is 10 (assuming that 0 is the 0^{th} nonprime, 1 is the 1^{st} nonprime, and so on), the 4-bit binary of which is 1010. Hence the first source block is 1010.

Using step 6, we can say that step 7 and step 8 are to be repeated for 3 times as there are still 3 blocks left. Step 7 only does the job of moving from one block to another and, in fact, step 8 works in the same way as step 5. So, proceeding in the same way, we obtain the remaining blocks as 1001, 0101 and 0010.

Following step 9, we concatenate all the blocks in the same sequence of their generation to obtain the source stream S = 1010100101010010.

Figure 6.2.2.1 represents this example diagrammatically.



Serial No. of Block	Code Value	Prime/ Nonprime	Rank Value	Positional Value	Decimal Number	4-Bit Binary
0	0	Nonprime	0110	6	10	1010
1	0	Nonprime	0101	5	9	1001
2	1	Prime	010	2	5	0101
3	1	Prime	000	0	2	0010

Figure 6.2.2.1
Pictorial Representation of Decrypting T = 00110000000000001001010110 using RSBP Technique

6.3 Implementation

For the purpose of real implementation of the RSBP technique, we take the plaintext "Local Area Network" that was also considered in the RPMS technique in chapter 5 to have a comparative look at the two ciphertexts.

Section 6.3.1 shows the process of encryption and the process of decryption is described in section 6.3.2.

6.3.1 Implementation of Encryption Technique of RSBP

Corresponding to the plaintext "Local Area Network" the 144-bit stream obtained in section 5.3 in chapter 5 is as follows:

S=01001100/01101111/01100011/01100001/01101100/00100000/01000001/01110010/01100101/01100001/00100000/01001110/01100101/01110100/01110111/01101111/01110010/01101011, "/" being used as the separator of two consecutive bytes.

Now, the unique block size is taken as 8 bits. In the range of 0 to 255 (= $2^8 - 1$), there are 54 primes and 202 nonprimes. So, the rank value corresponding to the code value of 1 (prime) should be of 6 bits and the same corresponding to the code value of 0 (nonprime) should be of 8 bits. Table 6.3.1.1 shows the status of all the blocks in terms of the RSBP technique.

Table 6.3.1.1
Status of Different Blocks for the Message Stream of Bits of "Local Area Network"

Block Serial No.	Block Value (Binary)	Block Value (Decimal)	Prime/ Nonprime	Code Value	Positional Value	Rank Value
0	01001100	76	Nonprime	0	55	00110111
1	01101111	111	Nonprime	0	82	01010010
2	01100011	99	Nonprime	0	74	01001010
3	01100001	97	Prime	1	24	011000
4	01101100	108	Nonprime	0	80	01010000
5	00100000	32	Nonprime	0	21	00010101
6	01000001	65	Nonprime	0	47	00101111
7	01110010	114	Nonprime	0	84	01010100
8	01100101	101	Prime	1	25	011001
9	01100001	97	Prime	1	24	011000
10	00100000	32	Nonprime	0	21	00010101
11	01001110	78	Nonprime	0	57	00111001
12	01100101	101	Prime	1	25	011001
13	01110100	116	Nonprime	0	86	01010110
14	01110111	119	Nonprime	0	89	01011001
15	01101111	111	Nonprime	0	82	01010010
16	01110010	114	Nonprime	0	84	01010100
17	01101011	107	Prime	1	27	011011

From the column corresponding to "Code Value" in the table 6.3.1.1, we get code values of all the blocks and placing these values one after the other we get the stream C = 000100001100100001 of the length of 18 bits.

From the column corresponding to "Rank Value" in the same table, we take rank values of blocks in the opposite direction, which means that the rank value of the last block is taken first, followed by the rank value of the previous block, and so. In this way, all these rank values are placed together to form the stream R as the following: 011011010101000101001001011001010101100110010011100100010101011000011001

01010100001011110001010101010000011000010010100101001000110111, the length of which is 134 bits.

Now, since the combined length of the two streams C and R is 18 + 134 = 152 bits, which is already a multiple of 8, so that there is no need to insert any extra block of bits between C and R. to form the target stream T. Hence T = C + R, where "+" denotes the concatenation operator.

Therefore the 152-bit target stream T corresponding to the 136-bit source stream S is as follows:

000100001100100001011011010101000101001001011001010101100110010011100100010101011000011001010101000010111100010101010100000110000100101001010010001101111.

For the purpose of converting it into the ciphertext, we re-write it in the byte wise manner, so that it looks like the following:

00010000/11001000/01011011/01010100/01010010/01011001/01010110/01100100/11100100/01010101/10000110/01010101/00001011/11000101/01010100/00011000/01001010/01010010/00110111.

Finally, we obtain the ciphertext, which looks like the following:

▶╣TRYVd∑UåU╡T↑ JR7

Figure 6.3.1.1 pictorially shows the ciphertexts obtained for "Local Area Network" for this RSBP technique as well as for the RPMS technique discussed in chapter 5.

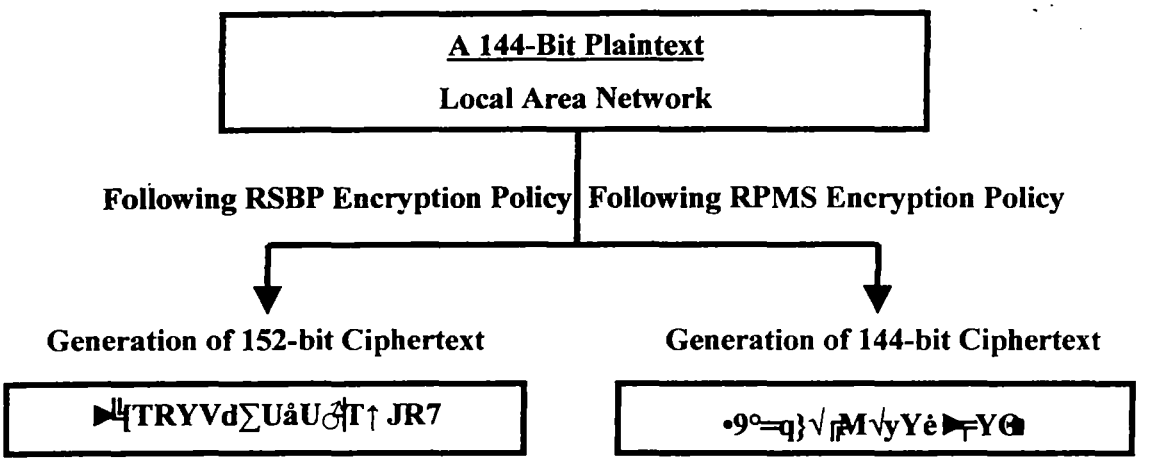


Figure 6.3.1.1
Ciphertexts for RSBP and RPMS for Plaintext "Local Area Network"

6.3.2 Implementation of Decryption Technique of RSBP

The process of decryption in this technique of RSBP is started with the 152-bit ciphertext, which, in the form of a binary stream is as follows:

000100001100100001011011010101000101001001011001010101100110010011100100
010101011000011001010101000010111100010101010100000110000100101001010010
00110111.

Now, combining the knowledge of the key and the size of the source file, it is known to the receiver of the ciphertext that the total number of blocks that had been created from the source stream prior to the encryption is 18. Hence out of the 152 bits in the received ciphertext, the first 18 bits stand for the code values of the 18 blocks.

Therefore the received stream of bits can be decomposed into the following two sections:

Code Value Section:

000100001100100001

Rank Value Section (Including Extra Bits, if any):

011011010101000101001001011001010101100110010011100100010101011000
011001010101000010111100010101010100000110000100101001010010000110111

Now, the receiver calculates that the total numbers of possible primes and nonprimes in the range of 0 to 255 (= $2^8 - 1$) are 54 and 202 respectively, so that blocks of 6 bits and 8 bits respectively are enough to represent their positions using bits.

Now, the scanning of code values from the beginning starts one after the other and accordingly the corresponding rank values are also consider using the following steps:

Step 1: We consider the first unmarked code bit in the target stream, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "00110111". From this we conclude that it indicates the 55th nonprime number, which is 76, i.e., 01001100 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`00100001100100001011011010101000101001001011001010101100110010011100100010101011000011001010101000010111100010101010100000110000100101001010010**(00110111)**$^{\surd}$

Step 2: We consider the first unmarked code bit in the stream of step 1, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "1010010". From this we conclude that it indicates the 82nd nonprime number, which is 111, i.e., 01101111 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`**0**$^{\surd}$0100001100100001011011010101000101001001011001010101100110010011100100010101011000011001010101000010111100010101010100000110000100101001010**(01010010)**$^{\surd}$ **(00110111)**$^{\surd}$

Step 3: We consider the first unmarked code bit in the stream of step 2, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01001010". From this we conclude that it indicates the 74th nonprime number, which is 99, i.e., 01100011 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`**0**$^{\surd}$**0**`10000110010000101101101010100010100100101100101010110011001001110010001010101100001100101010100001011110001010101010000011000 **(01001010)**$^{\surd}$ **(01010010)**$^{\surd}$ **(00110111)**`

Step 4: We consider the first unmarked code bit in the stream of step 3, which is 1, so that the first unmarked 6-bit block is considered from the LSB position, which is "011000". From this we conclude that it indicates the 24th prime number, which is 97, i.e., 01100001 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1`00001100100000101110110101010000101001001011001010
1011001100100111001000101010110000110010101010000101111
1000101010101000**(011000) (01001010)` (01010010)√ (00110111)`**

Step 5: We consider the first unmarked code bit in the stream of step 4, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01010000". From this we conclude that it indicates the 80th nonprime number, which is 108, i.e., 01101100 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1`0`00011001000010110110101010000101001001011001010
1011001100100111001000101010110000110010101010000101111
1100010101**(01010000)√ (011000)` (01001010)√ (01010010)√ (00110111)`**

Step 6: We consider the first unmarked code bit in the stream of step 5, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "00010101". From this we conclude that it indicates the 21st nonprime number, which is 32, i.e., 00100000 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1√0`0`00110010000010110110101010000101001001011001010
1010110011001001111001000101010110000110010101010000010
111**(00010101)` (01010000)` (011000)` (01001010)√ (01010010)√ (00110111)`**

Step 7: We consider the first unmarked code bit in the stream of step 6, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "00101111". From this we conclude that it indicates the 47th nonprime number, which is 65, i.e., 01000001 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`0`0`1`0`0`0`0`01100100001011011010101000101001001011001
0101011001100100111001000101010110000110010101010**0(001
01111)`(00010101)`(01010000)`(011000)`(01001010)`
(01010010)`(00110111)`**

Step 8: We consider the first unmarked code bit in the stream of step 7, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01010100". From this we conclude that it indicates the 84th nonprime number, which is 114, i.e., 01110010 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`0`0`1`0`0`0`0`11001000010110110101010001010010010110 01
0101011001100100111001000101010110000110 01**(01010100)(00
101111)`(00010101)`(01010000)`(011000)`(01001010)`
(01010010)`(00110111)`**

Step 9: We consider the first unmarked code bit in the stream of step 8, which is 1, so that the first unmarked 6-bit block is considered from the LSB position, which is "011001". From this we conclude that it indicates the 25th prime number. which is 101, i.e., 01100101 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`0`0`1`0`0`0`0`1`10010000101101101010100010100100101100
1010101100110010011100100010101011000**(011001)`
(01010100)`(00101111)`(00010101)`(01010000)`(011000)`
(01001010)`(01010010)`(00110111)`**

Step 10: We consider the first unmarked code bit in the stream of step 9. which is 1, so that the first unmarked 6-bit block is considered from the LSB position, which is "011000". From this we conclude that it indicates the 24th prime number, which is 97, i.e., 01100001 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1`0√0√0√0√1√1√**0010000101101101010100010100100101100101010111001100100111001000101010**(011000)`(011001)`(01010100)`(00101111)√(00010101)√(01010000)√(011000)√(01001010)`(01010010)`(00110111)√

Step 11: We consider the first unmarked code bit in the stream of step 10. which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "00010101". From this we conclude that it indicates the 21st nonprime number, which is 32. i.e., 00100000 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0`0√0√1`0`0`0√0`1√1√0`0100001011011010101000101001001011.00101010110011001001110010**(00010101)`(011000)`(011001)`(01010100)`(00101111)√(00010101)√(01010000)`(011000)`(01001010)`(01010010)`(00110111)√**

Step 12: We consider the first unmarked code bit in the stream of step 11. which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "00111001". From this we conclude that it indicates the 57th nonprime number, which is 78. i.e., 01001110 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1`0`0`0√0`1√1√0√0`**1000010111011010101000101001001011001010101100110011 **(00111001)`(00010101)√(011000)√(011001)`(01010100)√(00101111)`(00010101)√(01010000)√(011000)`(01001010)`(01010010)`(00110111)`

Step 13: We consider the first unmarked code bit in the stream of step 12, which is 1, so that the first unmarked 6-bit block is considered from the LSB position, which is "011001". From this we conclude that it indicates the 25th prime number, which is 101, i.e., 01100101 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1√0√0√0√0√1√1√0√0√1√0000101101101010100010100100101100101010110 **(011001)√ (00111001)√ (00010101)√ (011000)√ (011001)√ (01010100)√ (00101111)√ (00010101)√ (01010000)√ (011000)√ (01001010)√ (01010010)√ (00110111)√**

Step 14: We consider the first unmarked code bit in the stream of step 13, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01010110". From this we conclude that it indicates the 86th nonprime number, which is 116, i.e., 01110100 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1√0√0√0√0√1√1√0√0√1√0√0001011011010101000101001001011001**(01010110)√ (011001)√ (00111001)√ (00010101)√ (011000)√ (011001)√ (01010100)√ (00101111)√ (00010101)√ (01010000)√ (011000)√ (01001010)√ (01010010)√ (00110111)√**

Step 15: We consider the first unmarked code bit in the stream of step 14, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01011001". From this we conclude that it indicates the 89th nonprime number, which is 119, i.e., 01110111 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0√0√1√0√0√0√0√1√1√0√0√1√0√0√0010110110101010001010010**(01011001)√ (01010110)√ (011001)√ (00111001)√ (00010101)√ (011000)√ (011001)√ (01010100)√ (00101111)√ (00010101)√ (01010000)√ (011000)√ (01001010)√ (01010010)√ (00110111)√**

Step 16: We consider the first unmarked code bit in the stream of step 15, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01010010". From this we conclude that it indicates the 82nd nonprime number, which is 111, i.e., 01101111 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0`0√1√0√0√0√0√1√1√0√0`1√0√0√0√0101101101010100**(01010010) √(01011001)√(01010110)√(011001)√(00111001)√(00010101)` (011000)`(011001)√(01010100)√(00101111)√(00010101)√ (01010000)√(011000)√(01001010)√(01010010)`(00110111)√**

Step 17: We consider the first unmarked code bit in the stream of 16, which is 0, so that the first unmarked 8-bit block is considered from the LSB position, which is "01010100". From this we conclude that it indicates the 84th nonprime number, which is 114, i.e., 01110010 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0`0√1√0√0√0`0√1√1√0√0`1√0√0√0√0√**1011011**(01010100) √(01010010)√(01011001)`(01010110)√(011001)√(00111001)` (00010101)√(011000)√(011001)√(01010100)√(00101111)√ (00010101)√(01010000)`(011000)√(01001010)√(01010010)√ (00110111)√

Step 18: We consider the first unmarked code bit in the stream of step 17, which is 1, so that the first unmarked 8-bit block is considered from the LSB position, which is "011011". From this we conclude that it indicates the 27th prime number, which is 107, i.e., 01101011 in 8-bit format. We mark the code bit and the block of the rank value as follows:

0√0`0√1`0√0√0`0√1√1√0√0`1√0√0√0√0√1√(011011)√(01010100) √(01010010)√(01011001)√(01010110)√(011001)√(00111001)√ (00010101)√(011000)√(011001)√(01010100)√(00101111)√

(00010101)$^{\backslash}$ (01010000)$^{\surd}$ (011000)$^{\surd}$ (01001010)$^{\surd}$ (01010010)$^{\surd}$ (00110111)$^{\backslash}$

Since in the source stream there are 18 blocks, the decryption process is over and incidentally there is no unmarked block of bits left because while encrypting no extra bits were needed to place between the code values and the rank values.

Now, combining all the 8-bit values generated from step 1 to step 18, we get the following stream:

S=01001100/01101111/01100011/01100001/01101100/00100000/01000001/01110010/01100101/01100001/00100000/01001110/01100101/01110100/01110111/01101111/01110010/01101011.

Converting the bytes of S into characters, the plaintext is regenerated as **"Local Area Network"**.

6.4 Results

In this section different sample files have been encrypted using the RSBP technique for blocks of unique length of only 8 bits. One characteristic of using small blocks is the fast implementation, but as it is being discussed throughout the thesis, that for ensuring the security of the highest level, blocks of lengths at least of 64 bits are to be considered. The overall performance in terms of ensuring security increases with the increment in block size. This improvement in performance is reflected in chi square values and frequency distribution tests [44, 54].

The execution time being so less because of considering small blocks, this section emphasizes on the alteration of file size rather than on the encryption/decryption time.

Section 6.4.1 shows results of the encryption/decryption time, and the chi square value, this section also analyzes the alteration in file size. Section 6.4.2 depicts pictorial result of the frequency distribution tests, section 6.4.3 presents results of the comparison with the RSA system.

6.4.1 Result of Encryption/Decryption Time, Chi Square Value and File Size Alteration

Section 6.4.1.1 to section 6.4.1.5 presents results for files of different categories, namely, .EXE, .COM, .DLL, .SYS, and .CPP. Each section consists of tables with information on the source file name and size, the encrypted file size, the encryption time, the decryption time, average size alteration, and the chi square value, with the degree of freedom [55, 56].

6.4.1.1 Result for *EXE* Files

Table 6.4.1.1.1 presents the result for .EXE files. Here nine files have been considered. There sizes are in the range of 11611 bytes to 37220 bytes. Sizes of the encrypted files are in the range of 11816 to 39262. Out of nine, in seven cases sizes increase through encryption, whereas in remaining two cases reduction in size occurs. On the average, 2.03% of size is expanded. The encryption time ranges between 0.5500 seconds to 1.3700 seconds, whereas the decryption time ranges from 0.0500 seconds to 0.2800 seconds. The Chi Square value for each of the cases lies in the range of 32199 to 1501384 with the degree of freedom ranging from 248 to 255.

Table 6.4.1.1.1
Result for EXE Files for RSBP Techniques

Source File	Source File Size	Encrypted File Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom	Average Size Alteration
TLIB.EXE	37220	39262	1.3200	0.1700	193871	255	
UNZIP.EXE	23044	24222	0.9900	0.1700	32199	255	
RPPO.EXE	35425	36390	1.2700	0.2700	108005	255	
PRIME.EXE	37152	39124	1.3700	0.2200	106441	255	
TCDEF.EXE	11611	11816	0.5500	0.0500	55944	254	+2. 03 %
TRIANGLE.EXE	36242	37550	1.3100	0.1600	103763	255	
PING.EXE	24576	23107	0.9300	0.1100	1501384	248	
NETSTAT.EXE	32768	31544	1.1500	0.2200	422135	255	
CLIPBRD.EXE	18432	18673	0.8300	0.2800	171438	255	

Figure 6.4.1.1.1 graphically compares the size between the source file and the encrypted file for each pair for .EXE files. Each horizontal bar of color blue stands for the source size, whereas that of color brown stands for the encrypted file size. It is observed

from the figure that whatever be the case, size reduction or size expansion, never there exists a huge change in size.

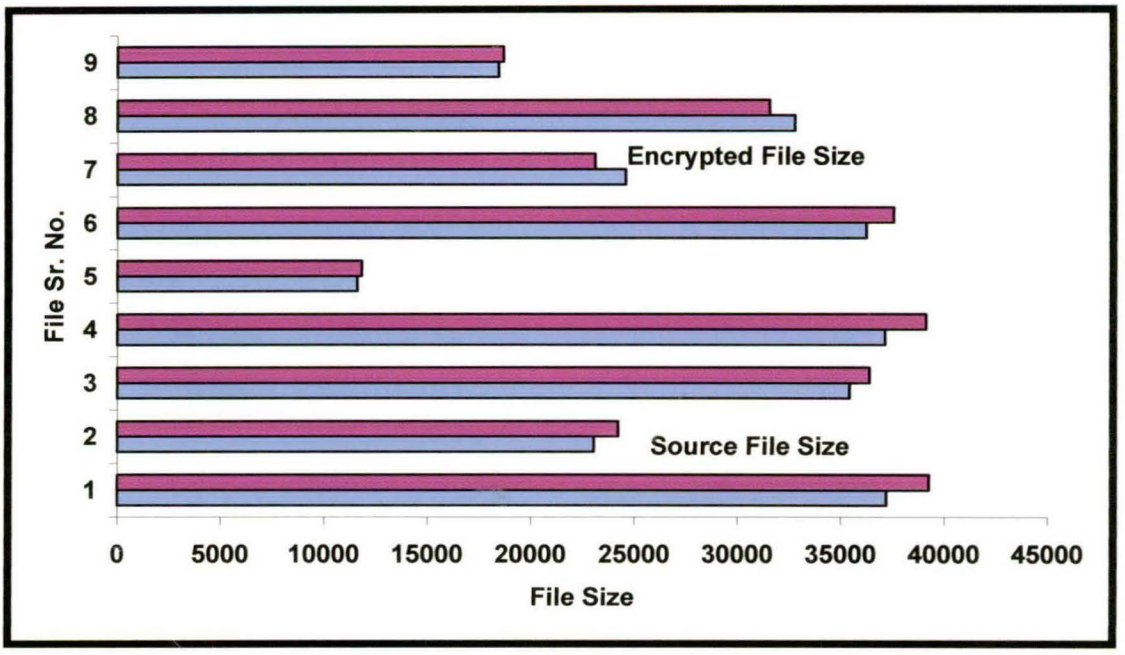


Figure 6.4.1.1.1
A Comparison between Source Size and Encrypted File Size for EXE Files in RSBP Technique

6.4.1.2 Result for *COM* Files

Table 6.4.1.2.1 presents the result for .COM files. Here ten files have been considered. There sizes are in the range of 5239 bytes to 29271 bytes. Sizes of the encrypted files are in the range of 5225 to 29612. Out of ten, in only four cases sizes increase through encryption, whereas in remaining six cases reduction in size occurs. On the average, 0.36% of size is reduced. The encryption time ranges between 0.2800 seconds to 1.0900 seconds, whereas the decryption time ranges from 0.0500 seconds to 0.2200 seconds. The Chi Square value for each of the cases lies in the range of 53441 to 667403 with the degree of freedom ranging from 230 to 255.

Table 6.4.1.2.1
Result for COM Files for RSBP Technique

Source File	Source File Size	Encrypted File Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom	Average Size Alteration
EMSTEST.COM	19664	20728	0.9300	0.0600	59855	255	
THELP.COM	11072	11552	0.6100	0.0500	53441	250	
WIN.COM	24791	24101	0.9900	0.1700	434812	252	
KEYB.COM	19927	20045	0.8800	0.2200	132264	255	
CHOICE.COM	5239	5225	0.2800	0.1100	72041	232	**-0. 36%**
DISKCOPY.COM	21975	21299	0.9300	0.1100	325978	254	
DOSKEY.COM	15495	15325	0.6100	0.1100	179966	253	
MODE.COM	29271	29612	1.0900	0.2200	147782	255	
MORE.COM	10471	9708	0.4900	0.0500	667403	230	
SYS.COM	18967	18633	0.8700	0.1700	232277	254	

Figure 6.4.1.2.1 graphically compares the size between the source file and the encrypted file for each pair for .COM files. Each horizontal bar of color blue stands for the source size, whereas that of color brown stands for the encrypted file size. Here also it is observed from the figure that whatever be the case, size reduction or size expansion, never there exists a huge change in size.

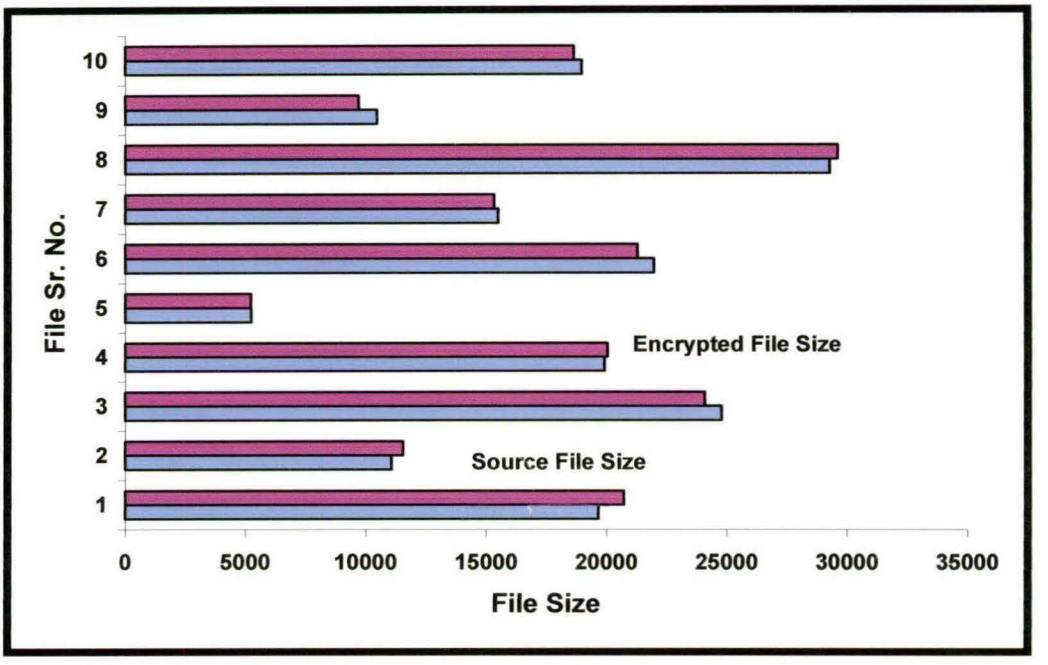


Figure 6.4.1.2.1
A Comparison between Source Size and Encrypted File Size for COM Files in RSBP Technique

6.4.1.3 Result for *DLL* Files

Table 6.4.1.3.1 presents the result for .DLL files. Here ten files have been considered. There sizes are in the range of 3216 bytes to 58368 bytes. Sizes of the encrypted files are in the range of 3251 to 59425. Out of ten, in six cases sizes increase through encryption, whereas in remaining four cases reduction in size occurs. On the average, 0.65% of size is expanded. The encryption time ranges between 0.1600 seconds to 1.6500 seconds, whereas the decryption time ranges from 0.0000 seconds to 0.2200 seconds. The Chi Square value for each of the cases lies in the range of 15369 to 725753 with the degree of freedom ranging from 217 to 255.

Table 6.4.1.3.1
Result for DLL Files for RSBP Technique

Source File	Source File Size	Encrypted File Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom	Average Size Alteration
SNMPAPI.DLL	32768	31311	1.1000	0.1600	725753	253	
KPSHARP.DLL	31744	32060	1.1000	0.1600	231924	254	
WINSOCK.DLL	21504	22309	0.8800	0.1100	97233	252	
SPWHPT.DLL	32792	32634	1.1600	0.1100	171222	255	
HIDCI.DLL	3216	3251	0.1600	0.0000	15369	217	**+0. 65%**
PFPICK.DLL	58368	59425	1.6500	0.2200	244965	255	
NDDEAPI.DLL	14032	14453	0.5500	0.0500	63622	249	
NDDENB.DLL	10976	11349	0.4400	0.0500	68331	251	
ICCCODES.DLL	20992	20927	0.7100	0.0600	158949	252	
KPSCALE.DLL	31232	31575	1.1500	0.1600	208467	255	

Figure 6.4.1.3.1 graphically compares the size between the source file and the encrypted file for each pair for .DLL files. Each horizontal bar of color blue stands for the source size, whereas that of color brown stands for the encrypted file size. Here also it is observed from the figure that whatever be the case, size reduction or size expansion, never there exists a huge change in size.

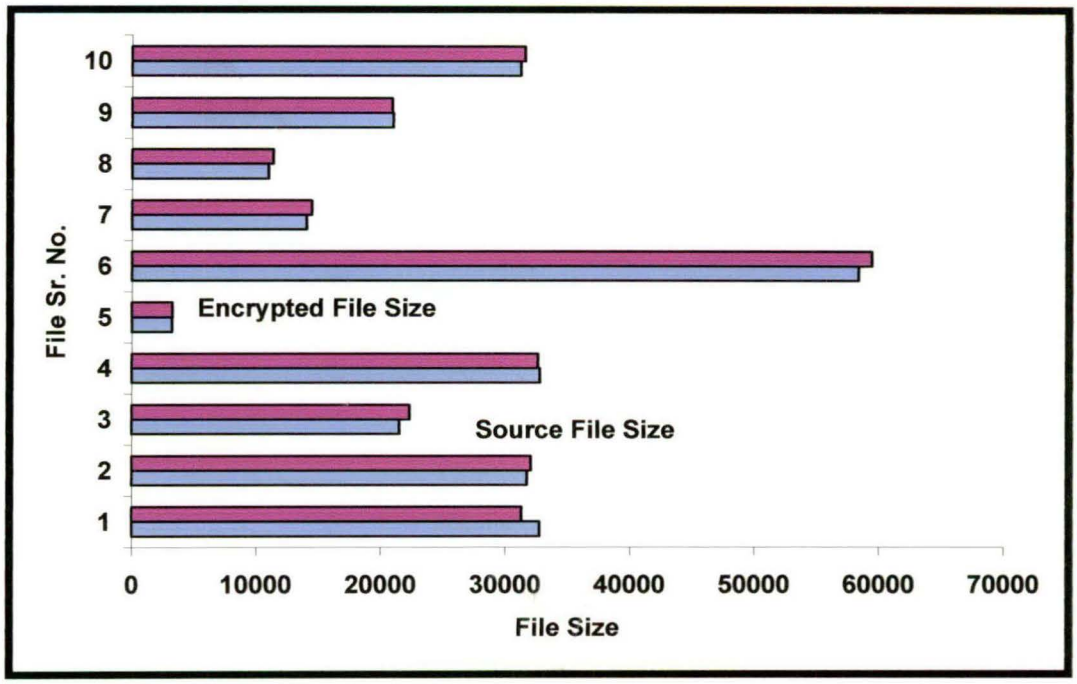


Figure 6.4.1.3.1
A Comparison between Source Size and Encrypted File Size for DLL Files in RSBP Technique

6.4.1.4 Result for *SYS* Files

Table 7.4.1.4.1 presents the result for .SYS files. Here ten files have been considered. There sizes are in the range of 5664 bytes to 33191 bytes. Sizes of the encrypted files are in the range of 5673 to 32453. Out of ten, in eight cases sizes increase through encryption, whereas in remaining two cases reduction in size occurs. On the average, 0.46% of size is expanded. The encryption time ranges between 0.1100 seconds to 1.1000 seconds, whereas the decryption time ranges from 0.0000 seconds to 0.1600 seconds. The Chi Square value for each of the cases lies in the range of 2755 to 830584 with the degree of freedom ranging from 165 to 255.

Table 6.4.1.4.1
Result for SYS Files for RSBP Technique

Source File	Source File Size	Encrypted File Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom	Average Size Alteration
HIMEM.SYS	33191	32453	1.1000	0.1600	830584	255	+0. 46%
RAMDRIVE.SYS	12663	12229	0.4400	0.0600	140587	241	
USBD.SYS	18912	19384	0.7100	0.1100	69223	255	
CMD640X.SYS	24626	25200	0.8800	0.1100	156293	255	
CMD640X2.SYS	20901	21300	0.9300	0.0500	163185	255	
REDBOOK.SYS	5664	5673	0.2800	0.0000	23680	230	
IFSHLP.SYS	3708	3807	0.2700	0.0500	22751	237	
ASPI2HLP.SYS	1105	1123	0.1100	0.0000	2755	165	
DBLBUFF.SYS	2614	2642	0.1100	0.0500	8396	215	
CCPORT.SYS	31680	31965	1.0900	0.1600	157014	255	

Figure 6.4.1.4.1 graphically compares the size between the source file and the encrypted file for each pair for .SYS files. Each horizontal bar of color blue stands for the source size, whereas that of color brown stands for the encrypted file size. Here also it is observed from the figure that whatever be the case, size reduction or size expansion, never there exists a huge change in size.

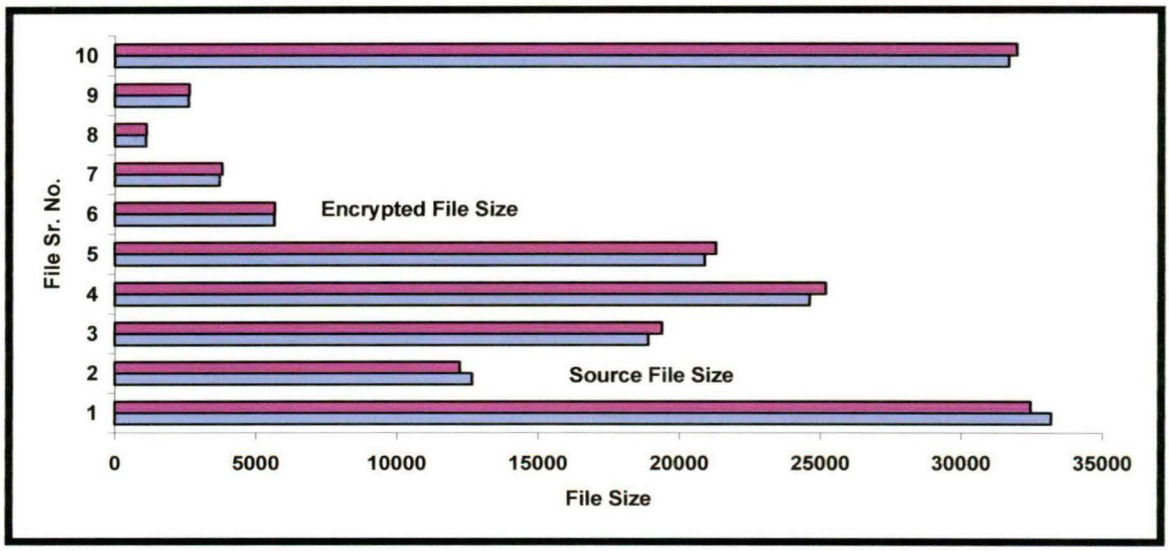


Figure 6.4.1.4.1
A Comparison between Source Size and Encrypted File Size for SYS Files in RSBP Technique

6.4.1.5 Result for *CPP* Files

Table 6.4.1.4.1 presents the result for .CPP files. Here ten files have been considered. There sizes are in the range of 1257 bytes to 32150 bytes. Sizes of the encrypted files are in the range of 1338 to 34250. Out of ten, in all cases, sizes increase through encryption. On the average, the highest among the five categories of files, expansion occurs, which is 6.92%. The encryption time ranges between 0.0600 seconds to 1.0400 seconds, whereas the decryption time ranges from 0.0000 seconds to 0.2200 seconds. The Chi Square value for each of the cases lies in the range of 2088 to 469578 with the degree of freedom ranging from 69 to 90.

Table 6.4.1.5.1
Result for CPP Files for RSBP Technique

Source File	Source File Size	Encrypted File Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom	Average Size Alteration
BRICKS.CPP	16723	17934	0.5400	0.1100	162768	88	+6. 92%
PROJECT.CPP	32150	34250	1.0400	0.2200	469578	90	
ARITH.CPP	9558	10101	0.3300	0.0600	40089	77	
START.CPP	14557	15690	0.4400	0.1100	182263	88	
CHARTCOM.CPP	14080	15039	0.6000	0.0500	212197	84	
BITIO.CPP	4071	4381	0.1600	0.0000	13544	70	
MAINC.CPP	4663	5024	0.1600	0.0000	24048	83	
TTEST.CPP	1257	1338	0.0600	0.0000	2088	69	
DO.CPP	14481	15614	0.6000	0.0500	195541	88	
CAL.CPP	9540	10083	0.4400	0.0600	34129	77	

Figure 6.4.1.5.1 graphically compares the size between the source file and the encrypted file for each pair for .CPP files. Each horizontal bar of color blue stands for the source size, whereas that of color brown stands for the encrypted file size. Here also it is observed from the figure that whatever be the case, size reduction or size expansion, never there exists a huge change in size. But even through a glance at the figure one can understand that each brown bars are longer than the corresponding blue bar since in each of the cases file size has been increased.

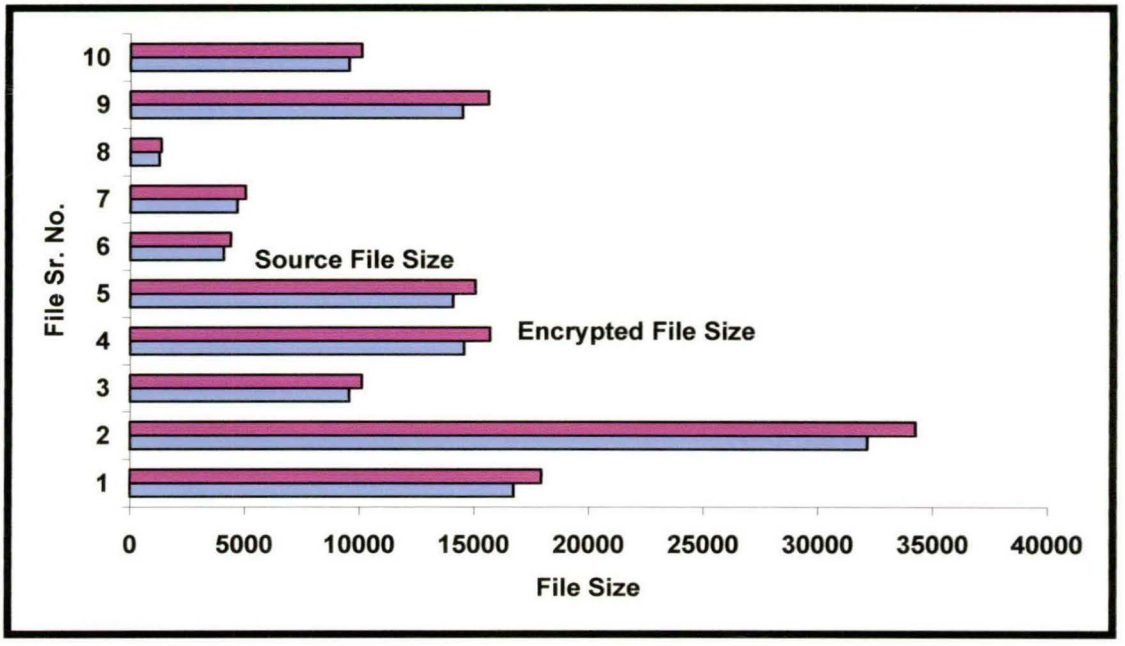


Figure 6.4.1.5.1
A Comparison between Source Size and Encrypted File Size for CPP Files in RSBP Technique

6.4.2 Result for Frequency Distribution Tests

The frequency distribution tests have been performed for all types of sample files under consideration. It is seen for all cases that the characters in the encrypted files are well distributed, which indicates that the technique proposed here is quite compatible with existing techniques. The red bars represent frequencies of characters in the encrypted file and those in blue color represent frequencies of characters in the source file. The frequencies of characters in encrypted files are evenly distributed. Therefore the source and the corresponding encrypted file are heterogeneous in nature. Hence it can be interpreted that through the proposed technique, a good quality of encryption is obtained.

Figure 6.4.2.1 to figure 6.4.2.5 are pictorial representations of a few of these tests.

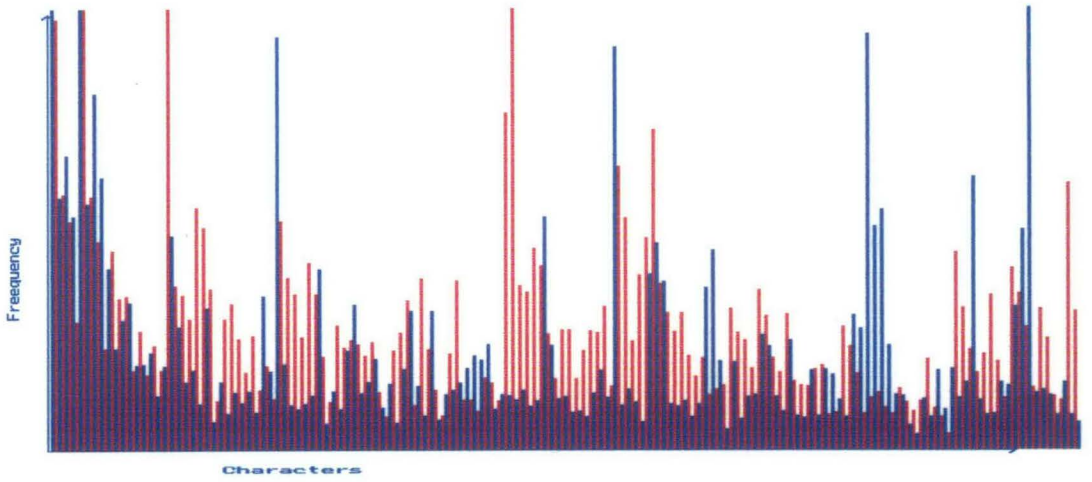


Figure 6.4.2.1
Frequency Distribution in UNZIP.EXE and Encrypted File for RSBP Technique

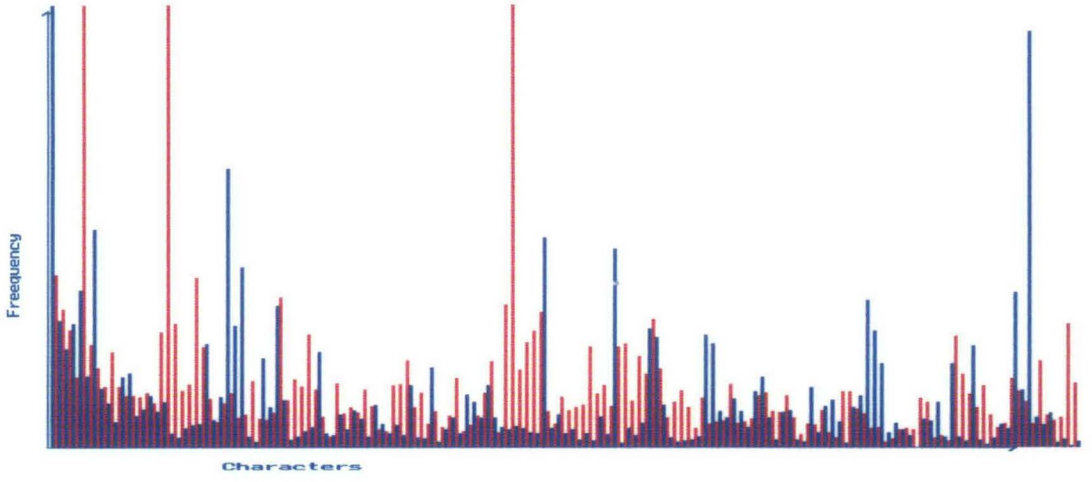


Figure 6.4.2.2
Frequency Distribution in THELP.COM and Encrypted File for RSBP Technique

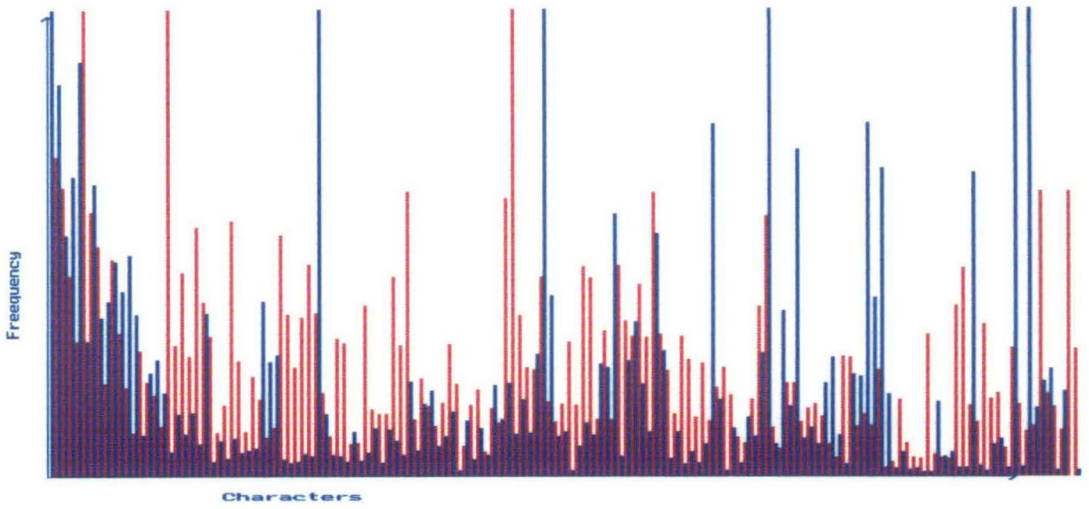


Figure 6.4.2.3
Frequency Distribution in WINSOCK.DLL and Encrypted File for RSBP Technique

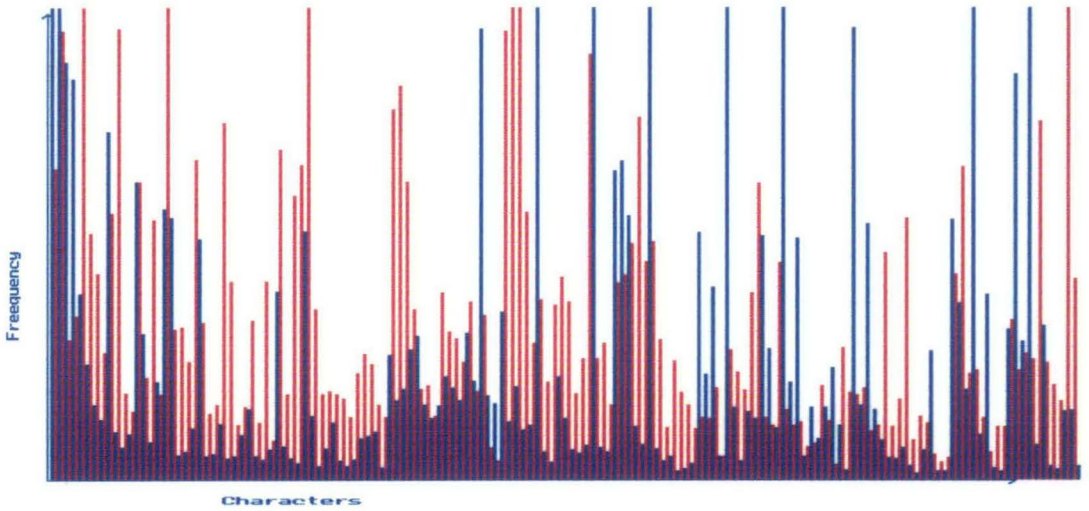


Figure 6.4.2.4
Frequency Distribution in CCPORT.SYS and Encrypted File for RSBP Technique

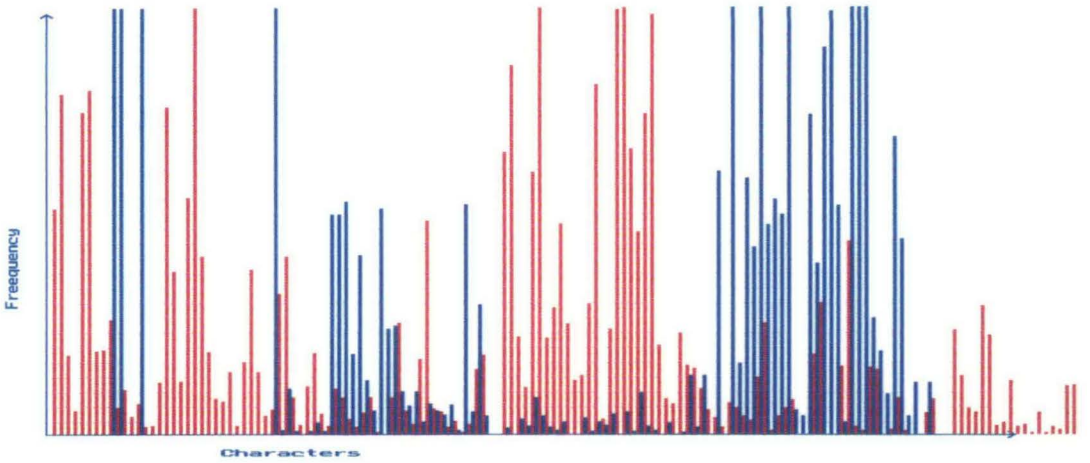


Figure 6.4.2.5
Frequency Distribution in CHARTCOM.CPP and Encrypted File for RSBP Technique

6.4.3 Comparison with RSA Technique

The result of implementing the proposed RSBP technique on the sample .CPP files has been compared with the result of implementing the existing RSA technique on the same set of files. The comparison is made in terms of chi square values. Table 6.4.3.1 enlists this comparative performance. Here the same ten files of type .CPP have been considered, sizes of which are ranging from 1257 bytes to 32150 bytes. The Chi Square value between a source file and the corresponding encrypted file, encrypted using the proposed RPSP technique, is in the range of 2088 to 469578, whereas the same between a source file and the corresponding encrypted file, encrypted using the existing RSA technique, is in the range of 3652 to 655734. The degrees of freedom are in the range of 69 to 90 [41].

Table 6.4.3.1
Comparative Result between Proposed RSBP and Existing RSA on the basis of Chi Square Values

File Name	File Size	Chi Square Value In RPSP	Chi Square Value In RSA	Degree of Freedom
BRICKS.CPP	16723	162768	200221	88
PROJECT.CPP	32150	469578	197728	90
ARITH.CPP	9558	40089	273982	77
START.CPP	14557	182263	49242	88
CHARTCOM.CPP	14080	212197	105384	84
BITIO.CPP	4071	13544	52529	70
MAINC.CPP	4663	24048	4964	83
TTEST.CPP	1257	2088	3652	69
DO.CPP	14481	195541	655734	88
CAL.CPP	9540	34129	216498	77

Graphically, using horizontal bars, it has been exhibited in figure 6.4.3.1. Each blue bar stands for the Chi Square value obtained implementing the RSBP technique, and each brown bar stands for the Chi Square value obtained implementing the RSA technique. There exist three blue bars, the height of each of which is more than the corresponding brown bar.

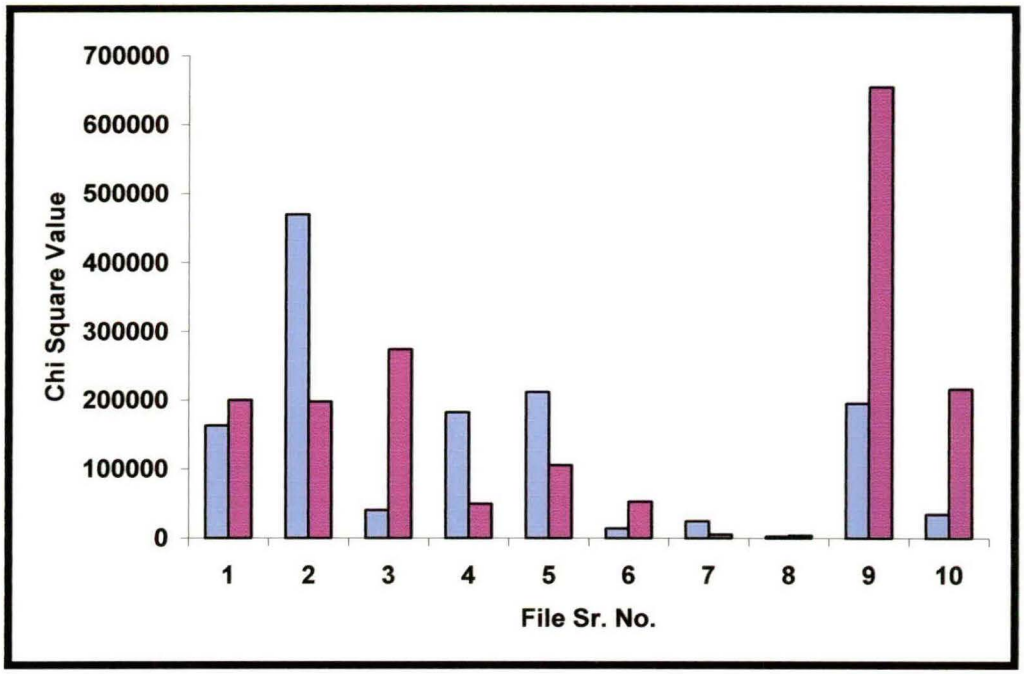


Figure 6.4.3.1
Graphical Comparison between Chi Square Values for RPSP (in Blue) and RSA (in Red)

6.5 Analysis and Conclusion including Comparison with RPSP, TE, RPPO, RPMS

Out of the five techniques proposed so far in the thesis, this RSBP technique, on the basis of the practical implementation done, offers the maximum Chi Square value on the average. Table 6.5.1 shows these results. It is observed from the table that the average Chi Square value of 201990.76 is obtained by implementing the RSBP technique, which is much more than 10701.70, obtained for the RPSP technique, 64188.04, obtained for the TE technique, 85350.94, obtained for the RPPO technique, and 140196.94, obtained for RPMS technique. The average encryption time during implementing the RSBP technique is observed to be 0.74673470 seconds, which is more than times taken implementing the RPMS (0.23659592 seconds) and the RPPO (0.73186806 seconds) techniques, whereas it is lesser than times taken implementing the TE (0.86703290 seconds) and the RPSP (8.75713800 seconds) techniques. The decryption time on the average is observed to be the lowest among these five proposed techniques, which is 0.11040816 seconds [48, 49, 50, 51, 54].

Graphical comparison in this aspect is drawn in chapter 10.

Table 6.5.1
Average Encryption/Decryption Time and Chi Square Value obtained in RPSP, TE, RPPO, RPMS, RSBP

Proposed Technique	Average Encryption Time	Average Decryption Time	Average Chi Square Value	Average Degree of Freedom
RPSP	8.75713800	8.73955200	10701.70	214
TE	0.86703290	0.94175818	64188.04	
RPPO	0.73186806	7.03076904	85350.94	
RPMS	0.23659592	0.15137143	140196.94	
RSBP	0.74673470	0.11040816	201990.76	

The proposed RSBP encryption policy is expected to be highly fruitful in ensuring information security to its best. The marginal alteration in size enhances the security. The requirement of the source file size during the process of decryption may be considered as an overhead, but, in turn, in helps in forming a larger key space as it is proposed to accommodate the source size in the secret key. Figure 8.2.3.1 in chapter 8 shows one proposed 123-bit key format for this RSBP policy strictly following a set of protocols used to form different blocks from the source stream of bits.

Forming blocks of varying lengths in this case makes the process of implementation a bit more difficult, but, if can be implemented faultlessly, ensures much better performance than what is observed in section 6.4.

The process of decryption involves much calculation, especially in the case of varying sizes of different blocks, as for each block size, it is required to calculate the lengths of different rank value sections. This computation overhead, in turn, promises better security.

Besides being highly effective while being implemented independently, if the RSBP encryption technique participates in the cascaded approach, it plays an important role in forming larger key space to enhance the security.