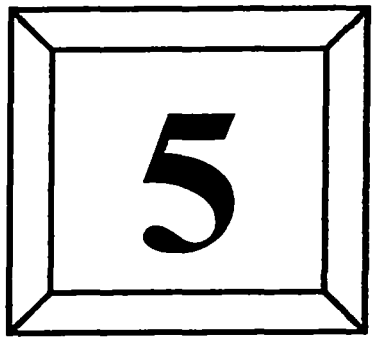

Encryption Through Recursive Positional Modulo-2 Substitution (RPMS) Technique

Contents

5.1 Introduction

Unlike the RPSP and the RPPO techniques, the Recursive Positional Modulo-2 Substitution (RPMS) technique is designed in such a manner that neither any cycle is formed nor the process of decryption is the same as that of the encryption.

Unlike the RPSP technique, here there is no any positional orientation of bits. In fact, through a generating function a new block is generated and the function is such that, unlike the generating function used in the RPSP technique, if an attempt is made to form a cycle it may require different number of iterations for two blocks of the same length. The generating function in the RPSP technique is related only with different bit-positions in a block, not with bits, so that any block of a fixed size requires the same number of iterations to form the cycle. But the generating function in the RPMS technique is directly related with different bits.

Unlike the RPPO technique discussed in chapter 4 and the TE technique discussed in chapter 3, here in the RPMS technique there is no application of Boolean algebra during encryption as well as decryption. During encryption, the decimal equivalent of the block of bits under consideration is one integral value from which the recursive modulo-2 operation starts. The modulo-2 operation is performed to check if the integral value is even or odd. Then the position of that integral value in the series of natural even or odd numbers is evaluated. The same process is started again with this positional value. Recursively this process is carried out to a finite number of times, which is exactly the length of the source block. After each modulo-2 operation, 0 or 1 is pushed to the output stream in MSB-to-LSB direction, depending on the fact whether the integral value is even or odd respectively. To generate the source code during decryption, bits in the target block are to be considered along LSB-to-MSB direction, where obviously 0 stands for even and 1 stands for odd. Following the same logic in reverse manner we are to reach to the MSB, after which we get an integral value, the binary equivalent of which is the source block [38, 42, 46, 47, 50, 54].

Section 5.2 is a detailed discussion on the scheme. Since the technique is an asymmetric one, the techniques of the scheme are implemented on a sample plaintext in section 5.3. Section 5.4 shows the results of applying this technique on the same set of

files that were considered earlier. The analysis of the RPMS technique from different angles on the basis of the results presented in section 5.4 is done in section 5.5.

5.2 The Scheme

Since the technique proposed is an asymmetric one, the scheme for encryption and that for decryption are being discussed separately along with relevant examples in sections 5.2.1 and 5.2.2.

5.2.1 The Encryption

A stream of bits is considered as the plaintext. Like in other proposed techniques. the plaintext is divided into a finite number of blocks, each having a finite fixed number of bits. The RPMS is then applied for each of the blocks.

For each block $S = s_0\ s_1\ s_2\ s_3\ s_4\ \ldots\ s_{L-1}$ of length L bits, the following technique is followed in a stepwise manner to generate the target block $T = t_0\ t_1\ t_2\ t_3\ t_4\ \ldots\ t_{L-1}$ of the same length (L).

Step 1: Corresponding to the source block $S = s_0\ s_1\ s_2\ s_3\ s_4\ \ldots\ s_{L-1}$, evaluate the equivalent decimal integer, D_L.

Step 2: Apply step 3 and step 4 exactly L number of times, for the values of the variable P ranging from 0 to (L-1) increasing by 1 after each execution of the loop.

Step 3: Apply modulo-2 operation on D_{L-P} to check if D_{L-P} is even or odd.

Step 4: If D_{L-P} is found to be even, compute $D_{L-P-1} = D_{L-P} / 2$, where D_{L-P-1} is its position in the series of natural even numbers. Assign $t_P = 0$.

If D_{L-P} is found to be odd, compute $D_{L-P-1} = (D_{L-P} + 1) / 2$, where D_{L-P-1} is its position in the series of natural odd numbers. Assign $t_P = 1$.

Step 5: With the values of all the t_P's being available, p ranging from 0 to (L-1), $T = t_0\ t_1\ t_2\ t_3\ t_4\ \ldots\ t_{L-1}$ constructs the target block corresponding to $S = s_0\ s_1\ s_2\ s_3\ s_4\ \ldots\ s_{L-1}$.

Figure 5.2.1.1 shows the pseudocode for this approach and the same has been presented through a flow diagram in figure 5.2.1.2.

```
Evaluate: D_L, the decimal equivalent, corresponding to the source
block S = s_0 s_1 s_2 s_3 s_4 ... s_{L-1}.
Set: P = 0.
LOOP:
        Evaluate: Temp = Remainder of D_{L-P} / 2.
        If Temp = 0
                Evaluate: D_{L-P-1} = D_{L-P} / 2.
                Set: t_P = 0.
        Else
        If Temp = 1
                Evaluate: D_{L-P-1} = (D_{L-P} + 1) / 2.
                Set: t_P = 1.
        Set: P = P + 1.
        If (P > (L – 1))
                Exit.
ENDLOOP
```

Figure 5.2.1.1
Pseudocode of the RPMS Scheme to encrypt a Source Block

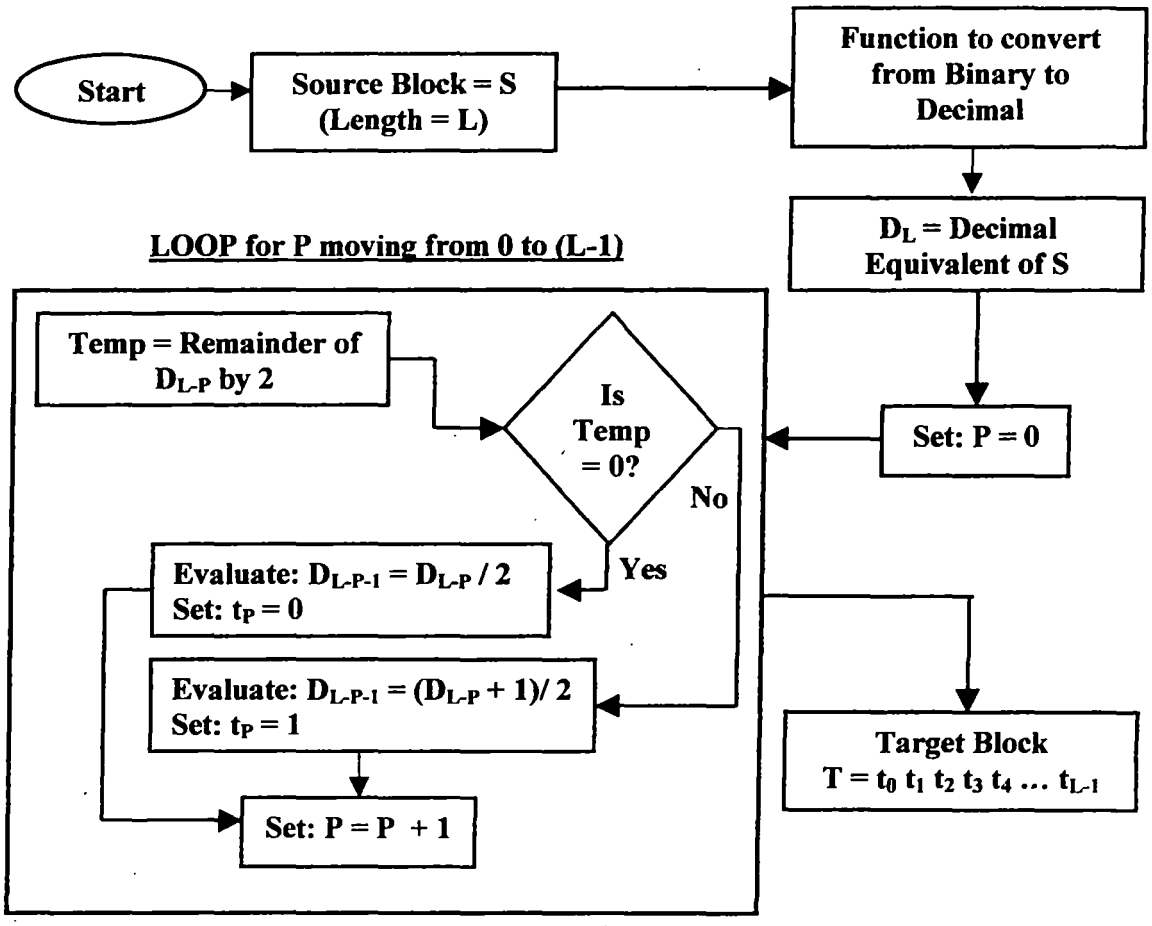

Figure 5.2.1.2
Flow Diagram of the RPMS Scheme to generate Target Block

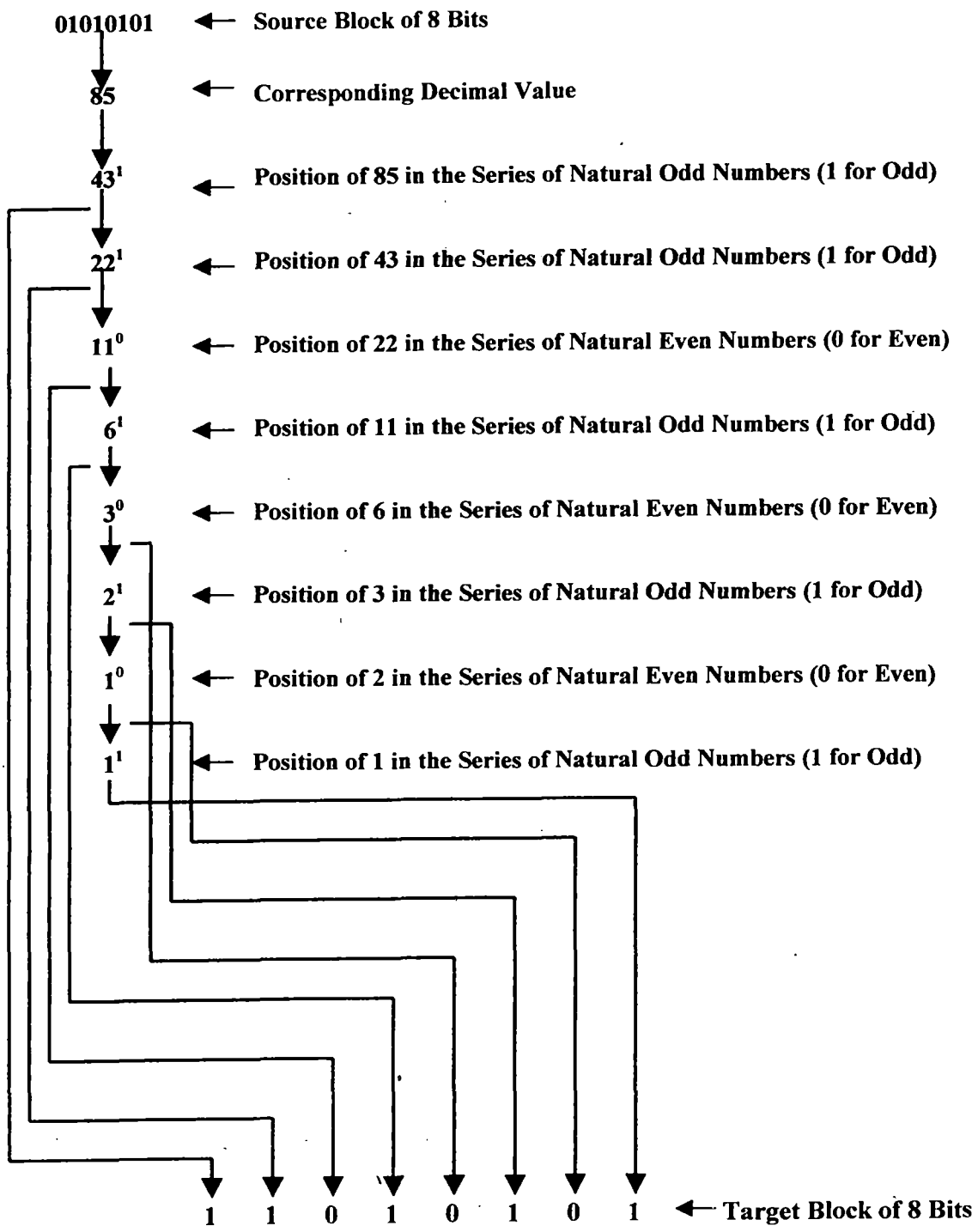

Figure 5.2.1.3
Formation of Target Block for Source Block 01010101 for RPMS Technique

Consider a source block of bits S=01010101 of length L=8. Now, following the technique shown in figures 5.2.1.1 and 5.2.1.2 the encrypted stream will be 11010101

and the corresponding flow diagram is shown in figure 5.2.1.3. In this figure, in any intermediate step after getting the position of an odd or even number in the series of natural numbers, either 1 or 0 is written on top of that position for indicating odd and even respectively. This bit may be termed as the **modulo-2 check bit**. Hence the encrypted block is T=11010101.

After generating all the target blocks, these are to be composed together in the same way the source stream was decomposed into different source blocks. As the result of this, the target stream of bits is obtained, the text corresponding to which is the encrypted message.

5.2.2 The Decryption

The initial step of work to be performed in the process of decryption is the same for all the proposed techniques.

In this initial stage, the encrypted message is to be converted into the corresponding original (source) stream of bits and then this stream is to be decomposed into a finite set of blocks, each consisting of a finite set of bits. Now, during this process of decomposition, the way by which the source stream was decomposed during encryption is to be followed, so that corresponding to each block, which is effectively the target block, the source block can be generated.

For each target block $T = t_0\ t_1\ t_2\ t_3\ t_4 \ldots t_{L-1}$ of length L bits, the following scheme is followed in a stepwise manner to generate the source block $S = s_0\ s_1\ s_2\ s_3\ s_4 \ldots s_{L-1}$ of the same length (L).

Step 1: Set P = L – 1 and T = 1.

Step 2: Repeat step 3 and step 4 for the value of P ranging from (L-1) to 0.

Step 3: If $t_P = 0$

T = T^{th} even number in the series of natural even numbers;

If $t_P = 1$

T = T^{th} odd number in the series of natural even numbers.

Step 4: Set P = P – 1.

Step 5: Convert T into the corresponding stream of bits S = s_0 s_1 s_2 s_3 s_4 … s_{L-1}, which is the source block.

```
Set: P = L – 1 and T = 1.
LOOP:
        If tP = 0
                Evaluate: T = Tth even number in the series of natural numbers.
        Else
        If tP = 1
                Evaluate: T = Tth odd number in the series of natural numbers.
        Set: P = P – 1.
        If P < 0
                Exit.
ENDLOOP
Evaluate: S = s0 s1 s2 s3 s4 … sL-1, which is the binary equivalent of T.
```

Figure 5.2.1.4
Pseudocode of RPMS Technique to decrypt a Target Block

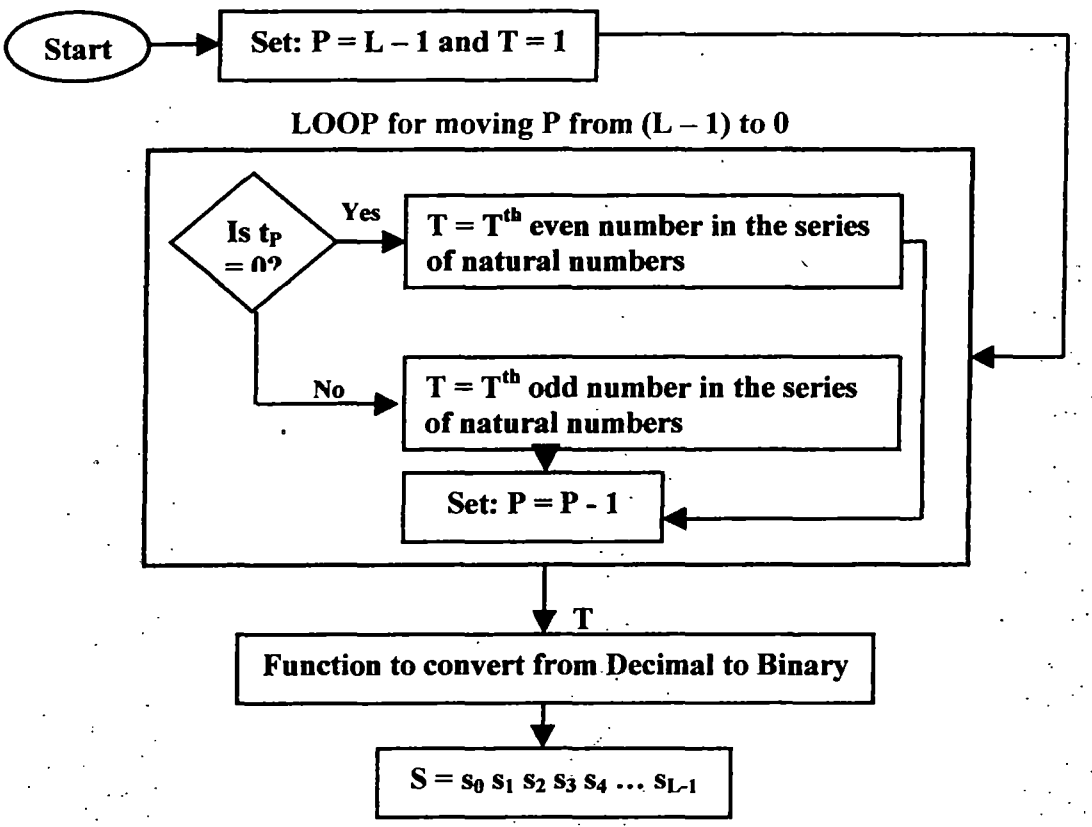

Figure 5.2.1.5
Flow Diagram of the RPMS Scheme to decrypt Target Block

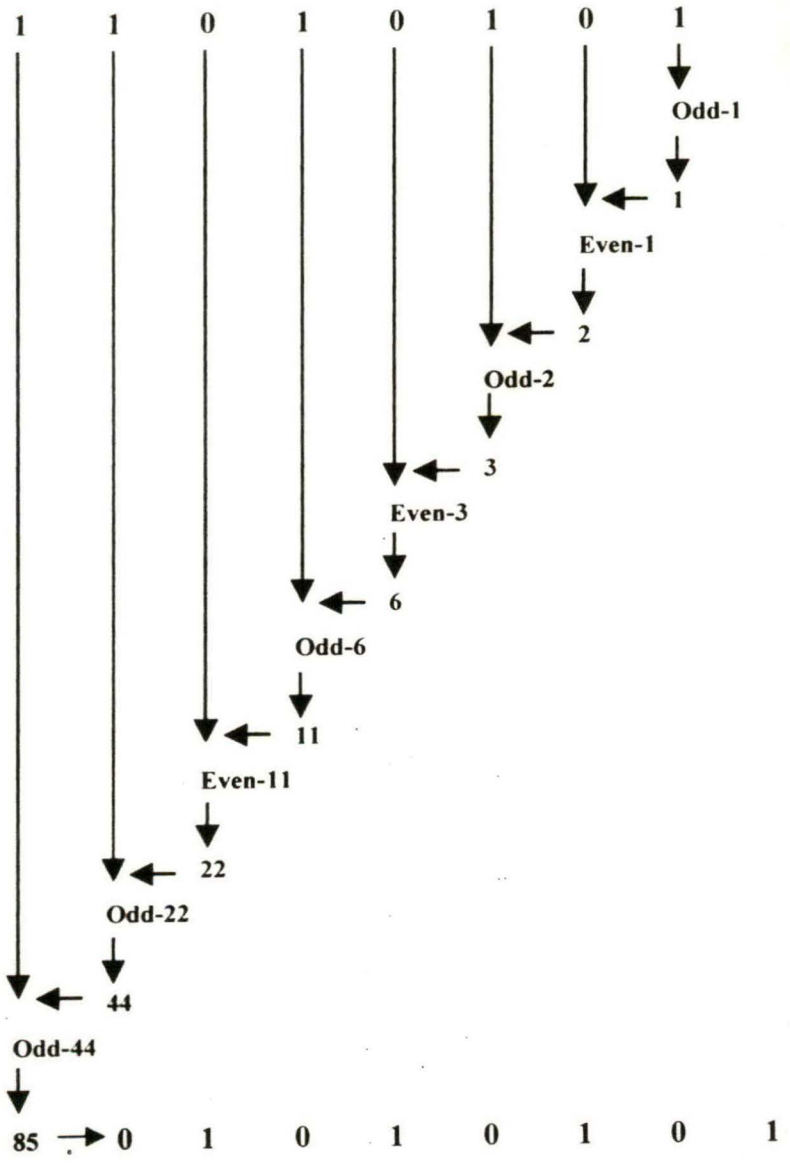

Figure 5.2.1.6

Formation of Source Block decrypting Target Block 11010101 in the RPMS Scheme

Figure 5.2.1.4 shows the approach written in the form of a pseudocode and the corresponding flow diagram is shown in figure 5.2.1.5.

Figure 5.2.1.6 diagrammatically shows the generation of the source block S=01010101 by decrypting the target block T=11010101. Here the process starts with the LSB, which is "1". Since "1" stands for "odd", the first odd number is to be considered first, which is 1. Then the previous-to-LSB bit is to be considered, which is here "0". Now, "0" stands for "even". Therefore the 1st even number is to be considered, which is

2. Consider the previous bit, which is "1". As "1" stands for "odd", 2nd odd number is to be considered, which is 3. Approaching in this way for the remaining bits till the MSB, the number "85" is obtained, for which the 8-bit block is "01010101".

5.3 Implementation

In this section, we consider a separate plaintext (P) as: "Local Area Network". The technique being an asymmetric one, the task of encryption and that of decryption are being discussed separately. Section 5.3.1 shows how the plaintext P is to be encrypted using this technique and section 5.3.2 describes the process of decryption.

5.3.1 The Process of Encryption

Consider the message string "Local Area Network". To construct the stream of bits table 5.3.1.1 is used, in which all the characters and their corresponding ASCII representations are enlisted.

Table 5.3.1.1
Character-to-Byte Conversion for the Text "Local Area Network"

Character	Byte
L	01001100
o	01101111
c	01100011
a	01100001
l	01101100
<Blank>	00100000
A	01000001
r	01110010
e	01100101
a	01100001
<Blank>	00100000
N	01001110
e	01100101
t	01110100
w	01110111
o	01101111
r	01110010
K	01101011

Putting together these bytes in the original sequence, we get the source stream of bits as the following:

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

Now, S is decomposed into a set of 5 blocks, each of the first four being of size 32 bits and the last one being of 16 bits. During this process of decomposition, S is scanned along the MSB-to-LSB direction and extract required number of bits for different block.

like the first 32 bits in this direction being for the block S_1, the next 32 bits being for the block S_2, and so on. In this way the blocks are generated as follows:

S_1=01001100011011110110001101100001, S_2=01101100001000000100000101110010, S_3=01100101011000010010000001001110, S_4=01100101011101000111011101101111, S_5=0111001001101011.

This way of decomposition is to be intimated as the key by the sender of the message to the receiver of the same through a secret channel. More about this has been discussed in section 5.5.

For the block S_1, corresponding to which the decimal value is $(1282368353)_{10}$, the process of encryption is shown below:

$1282368353 \rightarrow 641184177^1 \rightarrow 320592089^1 \rightarrow 160296045^1 \rightarrow 80148023^1 \rightarrow 40074012^1 \rightarrow 20037006^0 \rightarrow 100018503^0 \rightarrow 5009252^1 \rightarrow 2504626^0 \rightarrow 1252313^0 \rightarrow 626157^1 \rightarrow 313079^1 \rightarrow 156540^1 \rightarrow 78720^0 \rightarrow 39135^0 \rightarrow 19568^1 \rightarrow 9784^0 \rightarrow 4892^0 \rightarrow 2446^0 \rightarrow 1223^0 \rightarrow 612^1 \rightarrow 306^0 \rightarrow 153^0 \rightarrow 77^1 \rightarrow 39^1 \rightarrow 20^1 \rightarrow 10^0 \rightarrow 5^0 \rightarrow 3^1 \rightarrow 2^1 \rightarrow 1^0 \rightarrow 1^1$.

From this we generate the target block T_1 corresponding to S_1 as: T_1=11111001001110010000100111001101.

Applying the similar process, we generate target blocks T_1, T_2, T_3, T_4 and T_5 as follows corresponding to source blocks S_1, S_2, S_3, S_4 and S_5 respectively.

T_2=01110001011111011111101111001001

T_3=01001101111110110111100101011001

T_4=10001001000100011101000101011001

T_5=1110100110110001.

Now, combining target blocks in the same sequence, we get the target stream of bits T as the following:

T=11111001/00111001/00001001/11001101/01110001/01111101/111110
11/11001001/01001101/11111011/01111001/01011001/10001001/000100
01/11010001/01011001/11101001/10110001.

This stream (T) of bits, in the form of a stream of characters, is transmitted as the encrypted message (C), which looks like the following:

•9°=q}√╓M√yYė►╤YG

5.3.2 The Process of Decryption

At the destination point, this encrypted message or the ciphertext C reaches and for the purpose of decryption the receiver has only the secret key. Now, by that secret key, the suggested format of which is discussed in section 5.5, the receiver gets the information on different block lengths. Using that secret key, all the blocks T_1, T_2, T_3, T_4 and T_5 are formed as follows:

T_1=11111001001110010000100111001101

T_2=01110001011111011111101111001001

T_3=01001101111111011011100101011001

T_4=10001001000100011101000101011001

T_5=1110100110110001.

Now, applying the process of decryption shown in figures 5.2.1.4 and 5.2.1.5, the corresponding source blocks S_i are generated for all T_i, $1 \leq i \leq 5$.

As for example, for the target block T_1, we may proceed in the following way: "First odd number is 1, 1st even is 2, 2nd odd number is 3, 3rd odd number is 5. 5th even number is 10, 10th even number is 20, 20th odd number is 39, 39th odd number is 77, 77th odd number is 153, 153rd even number is 306, 306th even number is 612, 612th odd number is 1223, 1223rd even number is 2446, 2446th even number is 4892, 4892nd even number is9784, 9784th even number is 19568, 19568th odd number is 39135, 39135th even number is 78720, 78720th even number is 156540, 156540th odd number is 313079, 313079th odd number is 626157, 626157th odd number is 1252313, 1252313th even number is 2504626, 2504626th even number is 5009252, 5009252nd odd number is 100018503, 100018503rd even number is 20037006, 20037006th even number is 40074012, 40074012th odd number is 80148023, 80148023rd odd number is 160296045, 160296045th odd number is 320592089. 320592089th odd number is 641184177, and finally 641184177th odd number is 1282368353, for which the corresponding 32-bit stream is S_1=01001100011011110110001101100001."

In this way all the source blocks of bits are regenerated and combining those blocks in the same sequence. the source stream of bits are obtained to get the source message or the plaintext. The schematic diagram of this entire technique is shown in the figure 5.3.2.1.

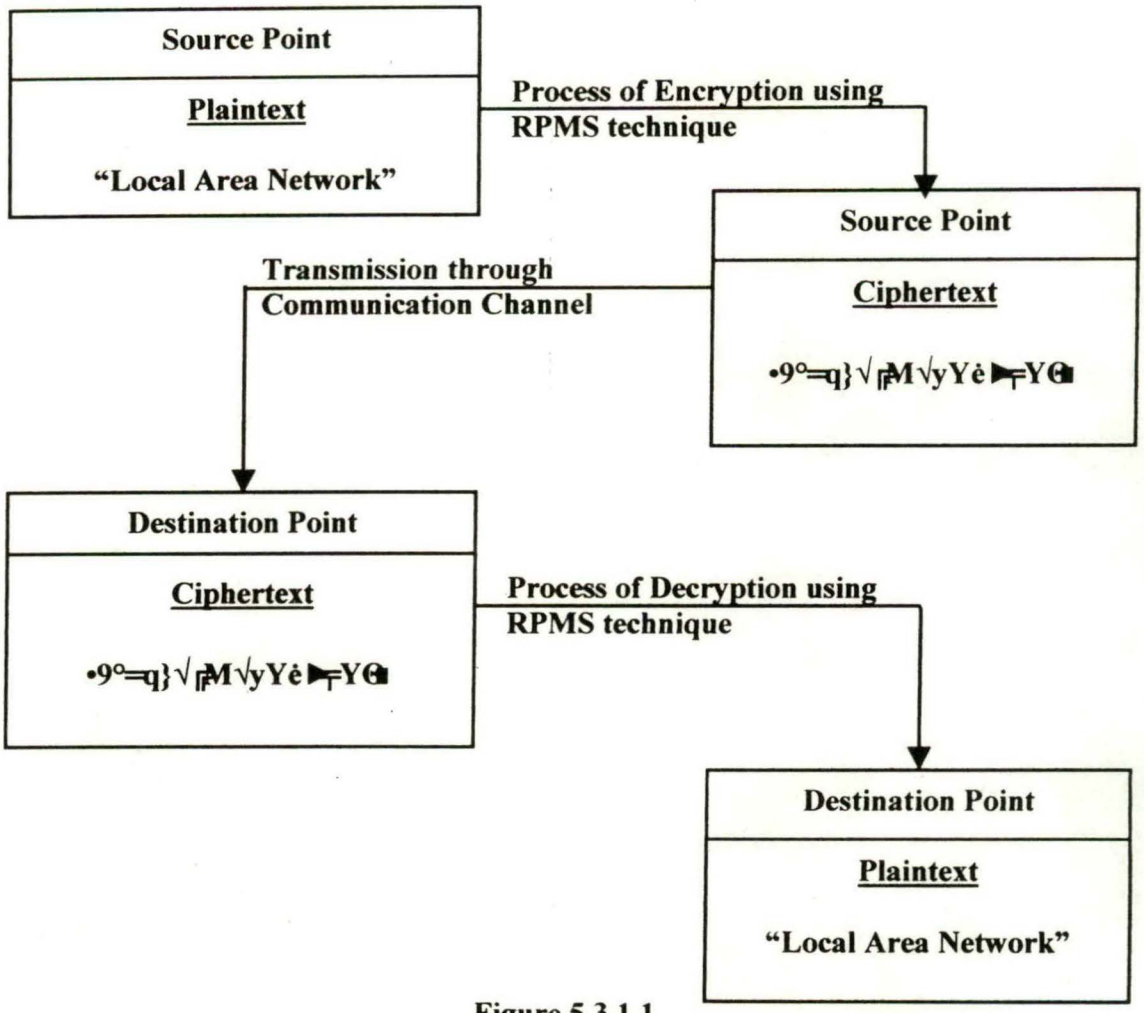

Figure 5.3.1.1
Schematic Diagram of Encryption/Decryption Techniques for Plaintext "Local Area Network"

5.4 Results

In this section results have been taken on the basis of the following factors:

- Computation of the encryption time and the decryption time, and hence establishing graphical relationships among the source size, the encryption time and/or the decryption time; also calculation of the Chi square value between the source and the encrypted files
- Performing the frequency distribution test
- Comparison with the RSA system

Experimentations on the basis of these four factors are respectively shown in section 5.4.1, section 5.4.2, and section 5.4.3.

In the next section, section 5.5, all these results have been analyzed from different perspectives.

5.4.1 Computing Encryption/Decryption Time

The same set of real-life files we have considered for the experimentation purpose. To ease the implementation, a unique block length has been considered. In this section all the results have been shown for block size of 16 bits [54, 55, 56].

Section 5.4.1.1 analyzes the results taken for the .EXE files, section 5.4.1.2 analyzes the results taken for the .COM files, section 5.4.1.3 analyzes the results taken for .DLL files, section 5.4.1.4 analyzes the results taken for the .SYS files, and section 5.4.1.5 analyzes the results taken for the .CPP files. In each of these sections the encryption time and the decryption time have been presented and graphically it has been shown how these execution times vary with the size of source file and that of the encrypted file. Each section also shows that in no case there is any storage overhead. Section 5.4.1.6 discusses about the results of the chi square tests.

5.4.1.1 Result for *EXE* Files

Table 5.4.1.1.1 shows the result for the .EXE files. Eight files have been considered. The size of each file is in the range of 23044 bytes to 59398 bytes. The encryption time is in the range of 0.2198 seconds to 0.6044 seconds. The decryption time lies in the range of 0.1648 seconds to 0.3846 seconds. The Chi Square value is in the range of 38480 to 444374 with the degree of freedom ranging from 248 to 255.

Table 5.4.1.1.1
Results for EXE Files in Tabular Form for RPMS Technique

Source File	Encrypted File	Source Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom
TLIB.EXE	A1.EXE	37220	0.3297	0.2198	364571	255
MAKER.EXE	A2.EXE	59398	0.6044	0.3846	444374	255
UNZIP.EXE	A3.EXE	23044	0.2747	0.1648	38480	255
RPPO.EXE	A4.EXE	35425	0.3846	0.2747	128642	255
PRIME.EXE	A5.EXE	37152	0.4945	0.3297	143696	255
TRIANGLE.EXE	A6.EXE	36242	0.4396	0.2198	136176	255
PING.EXE	A7.EXE	24576	0.2747	0.1648	127733	248
NETSTAT.EXE	A8.EXE	32768	0.3297	0.2198	387668	255
CLIPBRD.EXE	A9.EXE	18432	0.2198	0.1648	150396	255

The graphical relationship between the source file size and the encryption time on the basis of the results taken for the .EXE files is shown in figure 5.4.1.1.1. The figure establishes the fact that there exists a tendency that the encryption time increases linearly with the size of the source file.

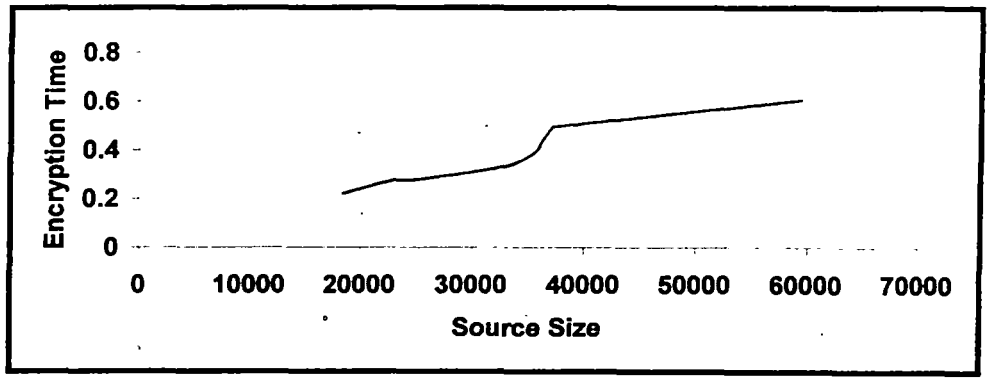

Figure 5.4.1.1.1
Relationship between Source Size and Encryption Time for EXE Files in RPMS Technique

5.4.1.2 Result for *COM* Files

Table 5.4.1.2.1 presents the results for the .COM files. Ten files have been considered. The size of each file is in the range of 5239 bytes to 29271 bytes. The encryption time is in the range of 0.1099 seconds to 0.3297 seconds. The decryption time

lies in the range of 0.0000 seconds to 0.2198 seconds. The Chi Square value is in the range of 13822 to 121318 with the degree of freedom ranging from 230 to 255.

Table 5.4.1.2.1
Results for COM Files in Tabular Form for RPMS Technique

Source File	Encrypted File	Source Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom
EMSTEST.COM	*A1.COM*	19664	0.2747	0.1099	68516	255
THELP.COM	*A2.COM*	11072	0.1648	0.0549	70590	250
WIN.COM	*A3.COM*	24791	0.2747	0.1648	79927	252
KEYB.COM	*A4.COM*	19927	0.2198	0.1099	121318	255
CHOICE.COM	*A5.COM*	5239	0.1099	0.0000	13822	232
DISKCOPY.COM	*A6.COM*	21975	0.2747	0.1099	91538	254
DOSKEY.COM	*A7.COM*	15495	0.1648	0.1099	51497	253
MODE.COM	*A8.COM*	29271	0.3297	0.2198	113529	255
MORE.COM	*A9.COM*	10471	0.1099	0.1099	15120	230
SYS.COM	*A10.COM*	18967	0.2198	0.1648	94004	254

Figure 5.4.1.2.1 graphically shows the relationship between the source file size and the decryption time for .COM files. In this case there exists no tendency of linear relationship between the decryption time and the size of the source file.

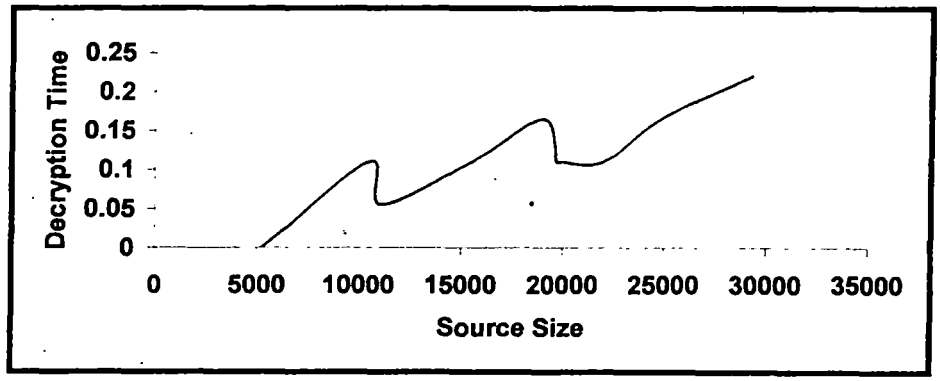

Figure 5.4.1.2.1
Relationship between Source Size and Encryption Time for COM Files in RPMS Technique

5.4.1.3 Result for *DLL* Files

Table 5.4.1.3.1 represents the results taken for the .DLL files. Ten files have been considered. The size of each file is in the range of 3216 bytes to 58368 bytes. The encryption time is in the range of 0.0549 seconds to 0.6044 seconds. The decryption time lies in the range of 0.0000 seconds to 0.3297 seconds. The Chi Square value is in the range of 10696 to 414717 with the degree of freedom ranging from 217 to 255.

Table 5.4.1.3.1
Result for DLL Files for RPMS Technique

Source File	Encrypted File	Source Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom
SNMPAPI.DLL	*A1.DLL*	32768	0.3846	0.2198	118235	253
KPSHARP.DLL	*A2.DLL*	31744	0.3846	0.2198	265630	254
WINSOCK.DLL	*A3.DLL*	21504	0.2198	0.1648	414717	252
SPWHPT.DLL	*A4.DLL*	32792	0.3846	0.2198	160065	255
HIDCI.DLL	*A5.DLL*	3216	0.0549	0.0000	10696	217
PFPICK.DLL	*A6.DLL*	58368	0.6044	0.3297	197903	255
NDDEAPI.DLL	*A7.DLL*	14032	0.1648	0.1099	128372	249
NDDENB.DLL	*A8.DLL*	10976	0.1099	0.0549	172239	251
ICCCODES.DLL	*A9.DLL*	20992	0.2198	0.1648	141924	252
KPSCALE.DLL	*A10.DLL*	31232	0.3297	0.2198	287292	255

Figure 5.4.1.3.1 represents the relationship of the source size with the encryption/decryption time for .DLL files. Here the continuous curve stands for the encryption time, whereas the dotted curve stands for the decryption time. It is clear from the figure that both the encryption time and the decryption time have the tendency of varying linearly with the source size.

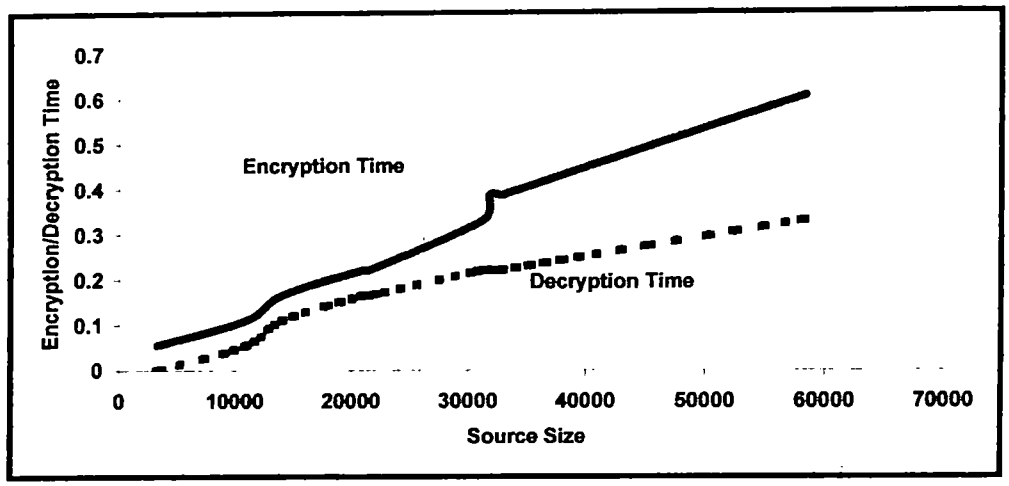

Figure 5.4.1.3.1
Relationship between Source Size and Encryption/Decryption Time for DLL Files in RPMS Technique

5.4.1.4 Result for *SYS* Files

Table 5.4.1.4.1 shows the results taken for the .SYS files. Ten files have been considered. The size of each file is in the range of 1105 bytes to 33191 bytes. The encryption time is in the range of 0.0549 seconds to 0.4397 seconds. The decryption time lies in the range of 0.0000 seconds to 0.3297 seconds. The Chi Square value is in the range of 2278 to 241968 with the degree of freedom ranging from 165 to 255.

Table 5.4.1.4.1
Result for SYS Files for RPMS Technique

Source File	Encrypted File	Source Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom
HIMEM.SYS	*A1.SYS*	33191	0.4397	0.3297	106943	255
RAMDRIVE.SYS	*A2.SYS*	12663	0.1099	0.1648	22352	241
USBD.SYS	*A3.SYS*	18912	0.2198	0.1648	158928	255
CMD640X.SYS	*A4.SYS*	24626	0.2747	0.2198	129672	255
CMD640X2.SYS	*A5.SYS*	20901	0.2198	0.2198	114070	255
REDBOOK.SYS	*A6.SYS*	5664	0.0549	0.0549	23469	230
IFSHLP.SYS	*A7.SYS*	3708	0.0549	0.0549	13116	237
ASPI2HLP.SYS	*A8.SYS*	1105	0.0549	0.0000	2278	165
DBLBUFF.SYS	*A9.SYS*	2614	0.0549	0.0549	4606	215
CCPORT.SYS	*A10.SYS*	31680	0.3297	0.3297	241968	255

Figure 5.4.1.4.1 shows the relationship between the source file size and the encryption time for .SYS files. The figure shows that the encryption time varies linearly enough with the source size.

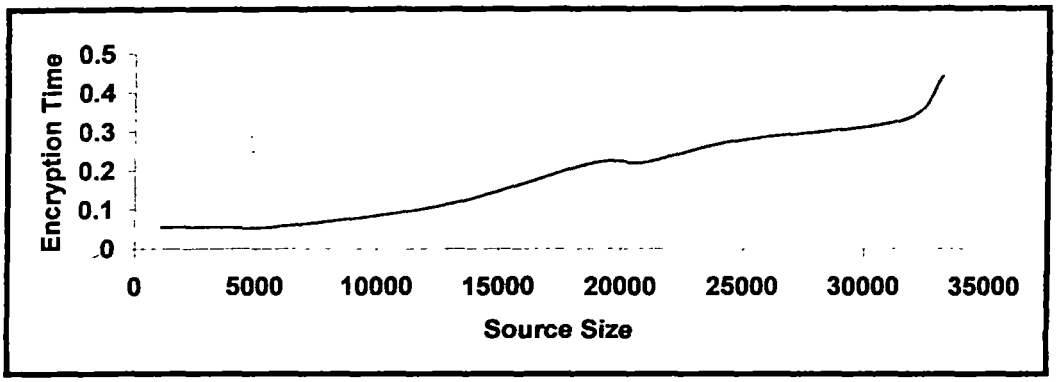

Figure.5.4.1.4.1
Relationship between Source Size and Encryption Time for SYS Files

Figure 5.4.1.4.2 establishes the graphical relationship between the encryption time and the decryption time for .SYS files. In the figure black horizontal pillars stand for different encryption times. Along the left-to-right direction the pillars have been arranged As per the increasing order of their heights. On each pillar, the corresponding decryption time has been marked as a point and all these points have been joined together to obtain a curve. Except on one occasion (on 3rd pillar), the curve is moving upward. So, it can be interpreted that there exists a tendency that the decryption time varies linearly with the encryption time, although some exceptions may also exist.

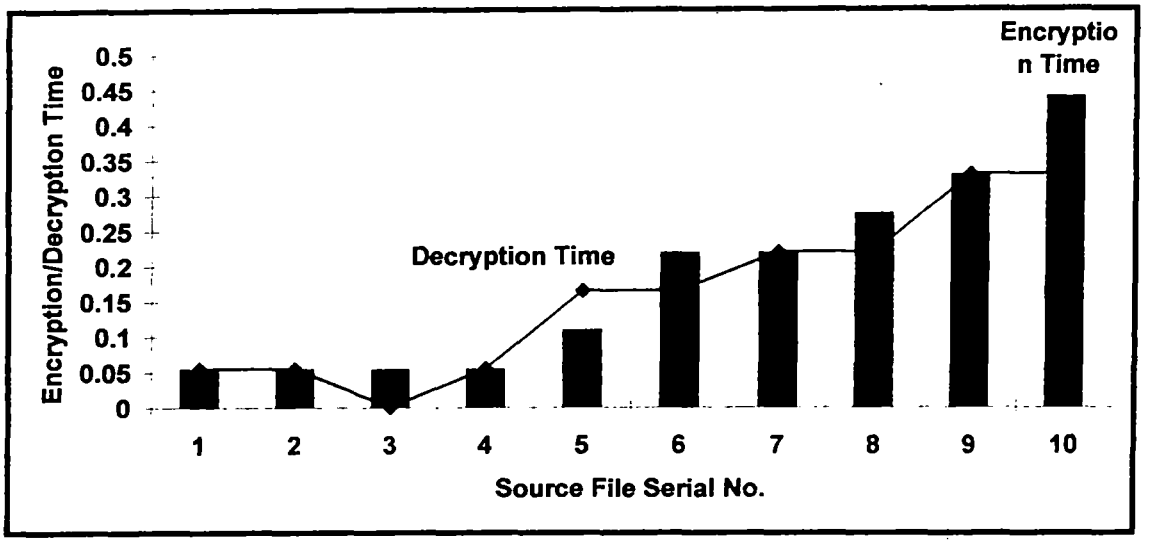

Figure 5.4.1.4.2
Relationship between Encryption Time and Decryption Time for .SYS Files for RPMS Technique

5.4.1.5 Result for *CPP* Files

Table 5.4.1.5.1 presents the result for the .CPP files. Ten files have been considered. The size of each file is in the range of 1257 bytes to 32150 bytes. The encryption time is in the range of 0.0000 seconds to 0.3846 seconds. The decryption time lies in the range of 0.0000 seconds to 0.2198 seconds. The Chi Square value is in the range of 1794 to 438133 with the degree of freedom ranging from 69 to 90.

Table 5.4.1.5.1
Result for .CPP Files

Source File	Encrypted File	Source Size	Encryption Time	Decryption Time	Chi Square Value	Degree of Freedom
BRICKS.CPP	*A1.CPP*	16723	0.2198	0.1099	113381	88
PROJECT.CPP	*A2.CPP*	32150	0.3846	0.2198	438133	90
ARITH.CPP	*A3.CPP*	9558	0.1099	0.0549	143723	77
START.CPP	*A4.CPP*	14557	0.1648	0.1099	297753	88
CHARTCOM.CPP	*A5.CPP*	14080	0.1648	0.0549	48929	84
BITIO.CPP	*A6.CPP*	4071	0.0549	0.0549	9101	70
MAINC.CPP	*A7.CPP*	4663	0.0549	0.0549	22485	83
TTEST.CPP	*A8.CPP*	1257	0.0000	0.0000	1794	69
DO.CPP	*A9.CPP*	14481	0.1648	0.1099	294607	88
CAL.CPP	*A10.CPP*	9540	0.1099	0.0549	143672	77

Graphically the relationship between the source size and the encryption/decryption time for .CPP files is shown in figure 5.4.1.5.1. Here the dotted curve stands for the encryption time and the other stands for the decryption time. It is observed that both the encryption time and the decryption time have the tendency of varying linearly with the size of the source file.

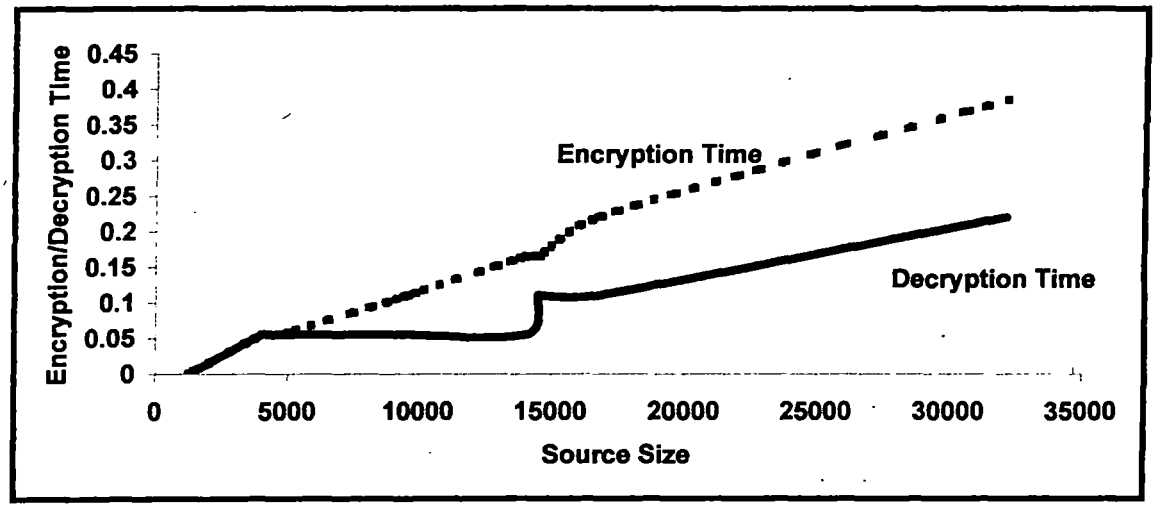

Figure 5.4.1.5.1
Graphical Relationship between Source Size and Encryption/Decryption Time for .CPP Files in RPMS Technique

5.4.1.6 Discussion on Chi Square Tests

As it is observed from the results of the chi square tests taken for .EXE, .COM, .DLL, and .SYS files, chi square values are much more less in comparison to values for .CPP files [44].

Figure 5.4.1.6.1 shows how chi square values change with source file size only for the category of .CPP files. From this figure any fixed conclusion hardly can be drawn. As the Chi Square value actually depends on the content of the source and the corresponding encrypted file; for files of almost same size Chi Square values may differ to a large extend. But it is observed that there exists a tendency that the Chi Square value increases with the file size, although the rate at which it increases is certainly not fixed and there exists a number of exceptions too.

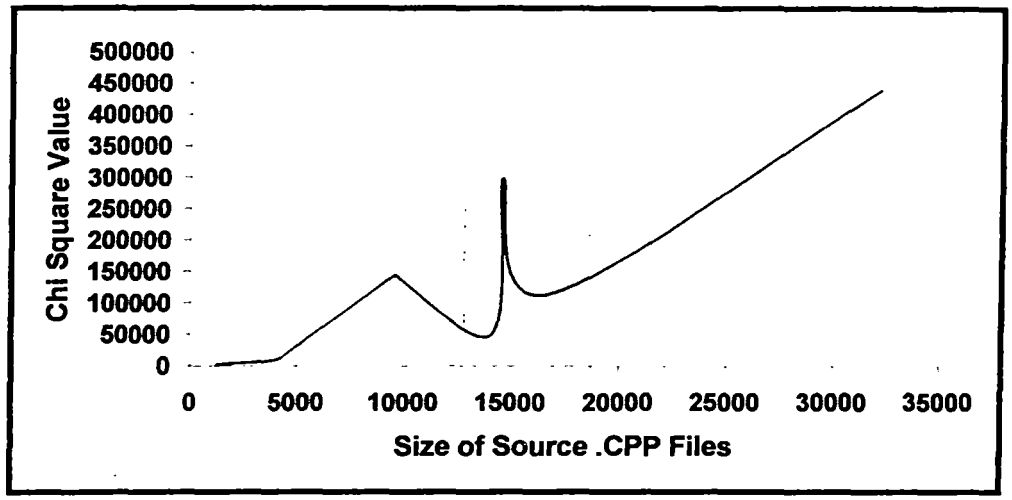

Figure 5.4.1.6.1
Variation of Chi Square Values with File Sizes (For .CPP Files)

Figure 5.4.1.6.2 attempts to represent a graphical outlook of a comparative analysis of the averages of all ratios of the chi square value and the source file size for all the five categories considered here. These average values are enlisted in table 5.4.1.6.1. From the table and the figure, it is observed that the result is most satisfactory in case of the .CPP files, where the average value is found to be 10.3263, in comparison with 3.9578 for .COM files, 4.3256 for .SYS files, 6.1545 for .EXE files, and 8.3660 for .DLL files.

Table 5.4.1.6.1
Ratios of Average Chi Square Value and Average Source Size for Different Categories of Files

Category of Files	Average of all Ratios of Chi Square Value and Source File Size
.EXE	6.1545
.COM	3.9578
.DLL	8.3660
.SYS	4.3256
.CPP	10.3263

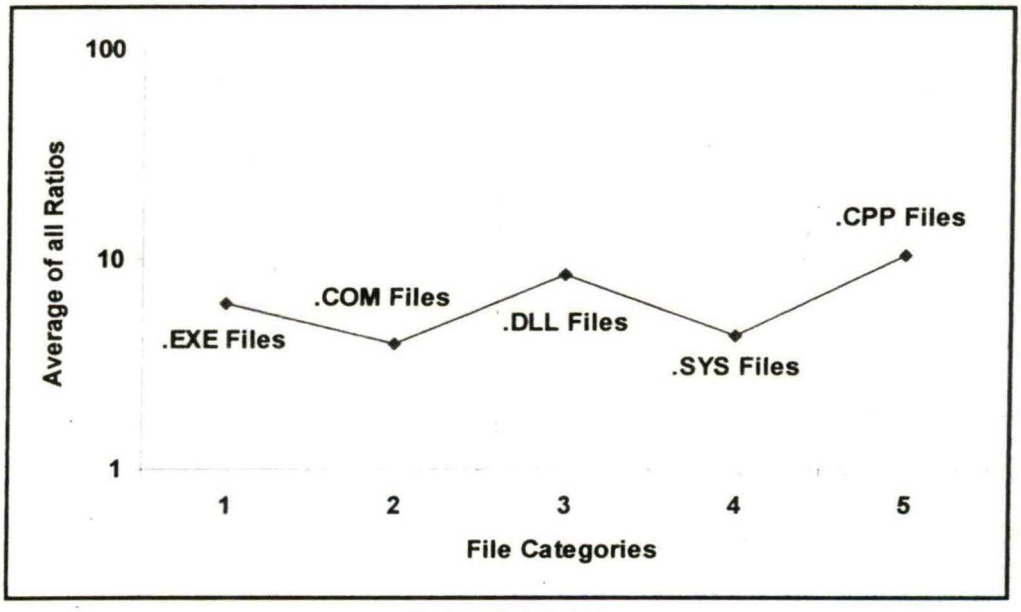

Table 5.4.1.6.2
Comparative Graphical Representation of Ratios of Average Chi Square Value and Average Source Size for Different Categories of Files

5.4.2 Result on Frequency Distribution Tests

Representing the result of the frequency distribution test for all the files considered in section 5.4.1 being an impractical task, here in this section, for the representation purpose only 5 files, one each from .EXE, .COM, .DLL, .SYS, and .CPP have been considered. In each case, frequency distribution is pictorially represented for the source file and the encrypted file. 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 5.4.2.1 to figure 5.4.2.5 show these results respectively.

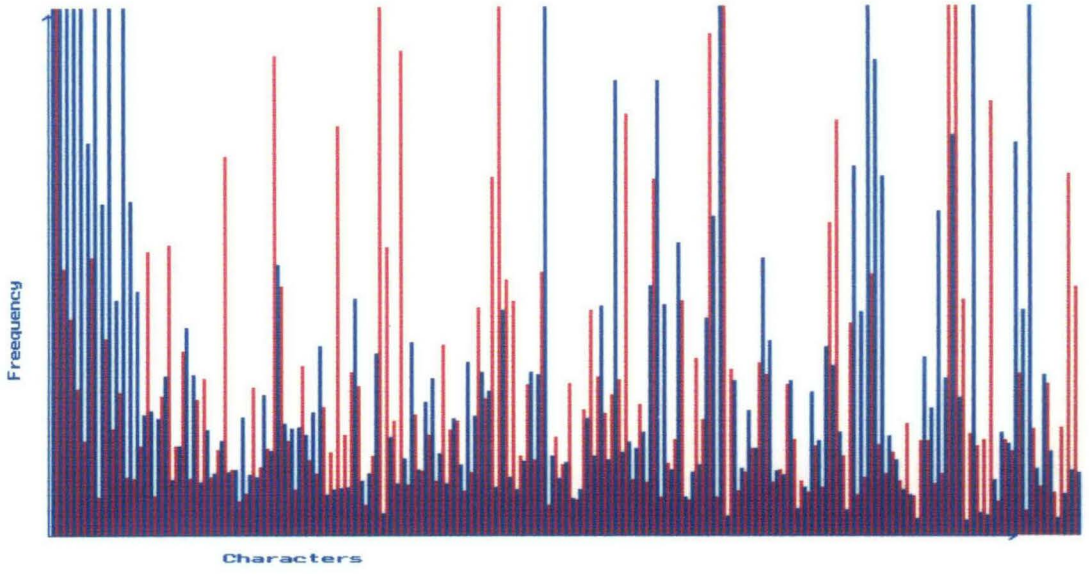

Figure 5.4.2.1
Frequency Distribution for PRIME.EXE and its Encrypted File for RPMS Technique

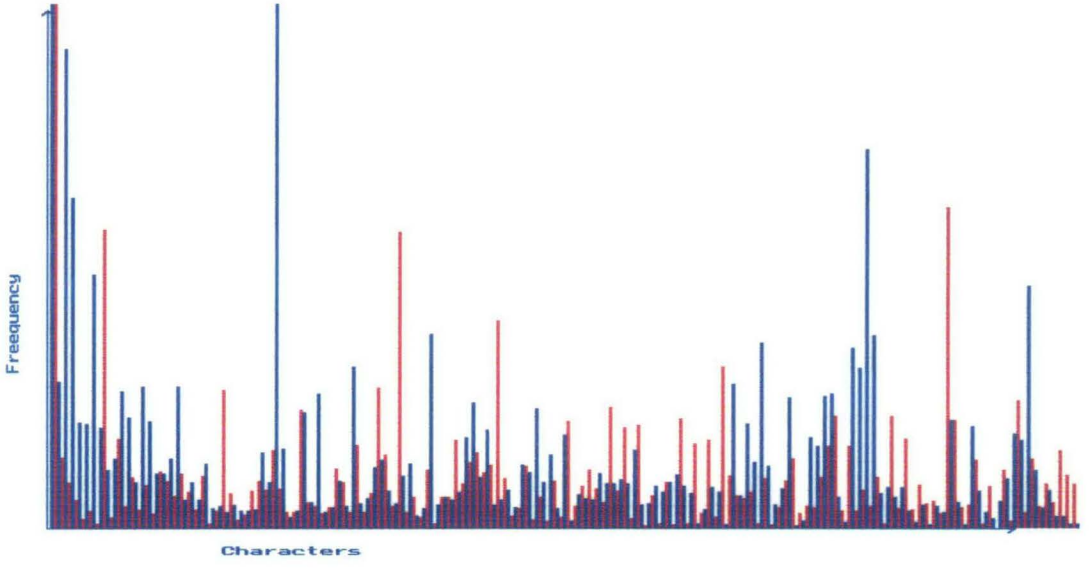

Figure 5.4.2.2
A Segment of Frequency Distribution for DOSKEY.COM and its Encrypted File for RPMS Technique

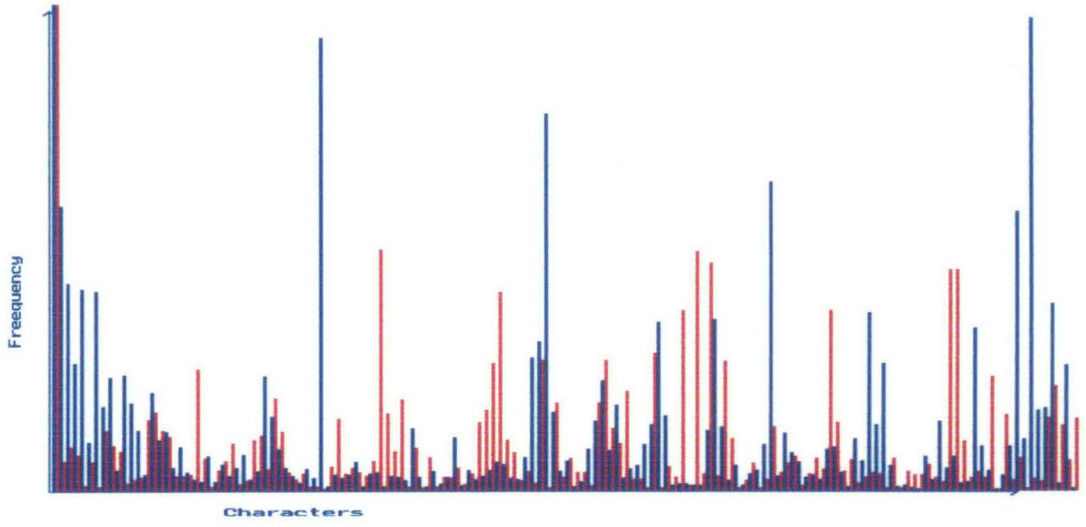

Figure 5.4.2.3
A Segment of Frequency Distribution for NDDENB.DLL and its Encrypted File for RPMS Technique

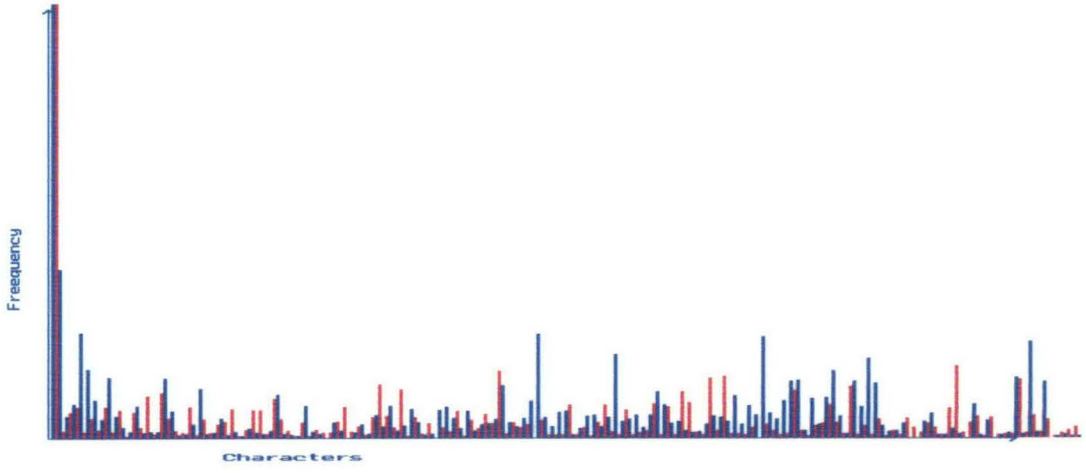

Figure 5.4.2.4
A Segment of Frequency Distribution for REDBOOK.SYS and its Encrypted File for RPMS Technique

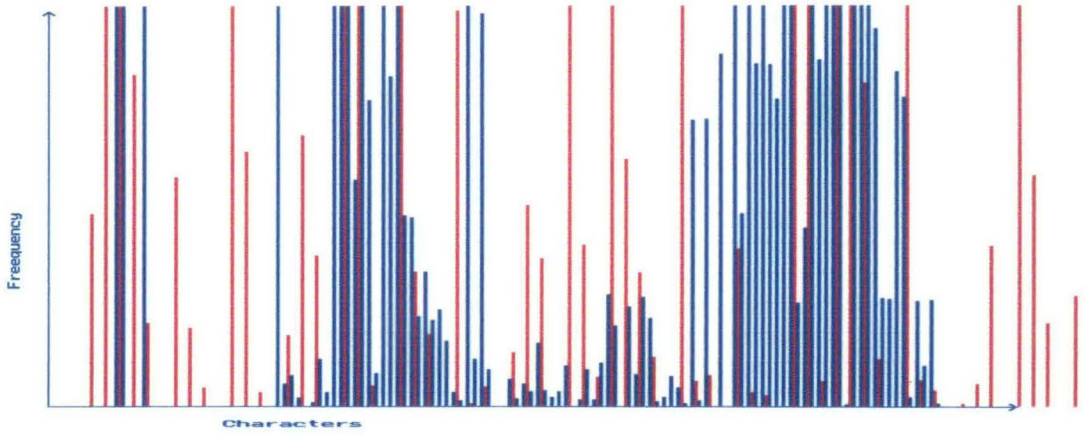

Figure 5.4.2.4
A Segment of Frequency Distribution for PROJECT.CPP and its Encrypted File for RPMS Technique

5.4.3 Comparison with RSA Technique

For the purpose of comparing the performance of the RPMS technique with the RSA system for a given set of files, the same set of 10 .CPP files have been considered. Table 5.4.3.1 represents this report. When the proposed RPMS technique is used for the encryption purpose, the Chi Square value is observed to be in the range of 1794 to 438133. If the existing RSA technique is used for the purpose of encryption, the Chi Square value is observed to be in the range of 3652 to 655734 [8, 40].

Table 5.4.3.1
Comparative Results between RPMS Technique and RSA System for .CPP Files

Source File	Encrypted File Using RPMS Technique	Encrypted File Using RSA Technique	Chi Square Value For RPMS Technique	Chi Square Value For RSA Technique	Degree of Freedom
BRICKS.CPP	*A1.CPP*	*CPP1.CPP*	113381	200221	88
PROJECT.CPP	*A2.CPP*	*CPP2.CPP*	438133	197728	90
ARITH.CPP	*A3.CPP*	*CPP3.CPP*	143723	273982	77
START.CPP	*A4.CPP*	*CPP4.CPP*	297753	49242	88
CHARTCOM.CPP	*A5.CPP*	*CPP5.CPP*	48929	105384	84
BITIO.CPP	*A6.CPP*	*CPP6.CPP*	9101	52529	70
MAINC.CPP	*A7.CPP*	*CPP7.CPP*	22485	4964	83
TTEST.CPP	*A8.CPP*	*CPP8.CPP*	1794	3652	69
DO.CPP	*A9.CPP*	*CPP9.CPP*	294607	655734	88
CAL.CPP	*A10.CPP*	*CPP10.CPP*	143672	216498	77

The information of table 5.4.3.1 has graphically been shown in figure 5.4.3.1. Here black pillars stand for Chi Square results for the RSA technique and red pillars stand for the same for the RPMS technique. As it is seen from the figure that red pillars are well compatible with black pillars and there exist three cases where red pillars are taller than black ones.

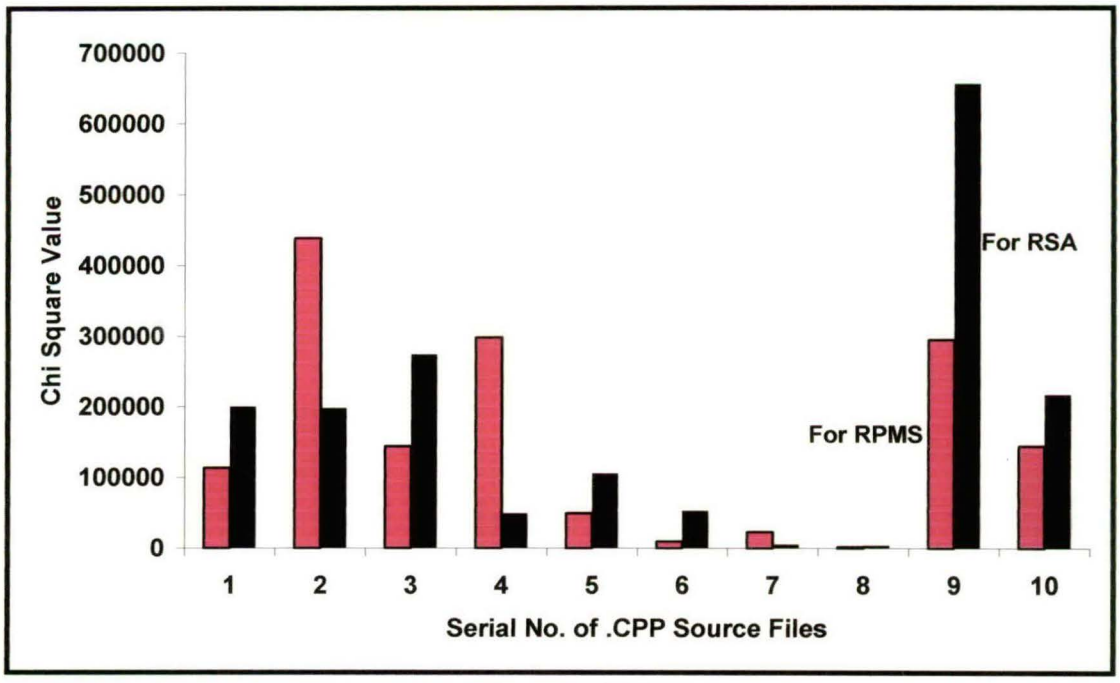

Figure 5.4.3.1
Graphical Comparison of Chi Square Values between RPMS and RSA Techniques

5.5 Analysis and Conclusion including Comparison with RPSP, TE, RPPO

Out of the four encryption techniques proposed so far in this thesis, it is observed that this RPMS technique produces the maximum Chi Square value on the average. Table 5.5.1 presents this information. The average Chi Square value observed for the RPMS technique is 140196.94, against 10701.70 for the RPSP technique, 64188.04 for the TE technique, and 85350.94 for the RPPO technique. In case of the RPMS technique, the average encryption time is observed to the lowest among the four techniques, which is 0.23659592 seconds, and the same is true for the average decryption time as well, which is 0.15137143 seconds [45, 48, 50, 52].

In chapter 10, graphical comparisons have been drawn.

Table 5.5.1
Average Encryption/Decryption Time and Chi Square Value obtained in RPSP, TE, RPPO, RPMS

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	

Since for the purpose of practical implementation blocks are constructed of unique size of only 16 bits, it is expected to achieve much better performance if bigger blocks are constructed, not necessarily of a unique size.

From the schematic point of view, the proposed RPMS technique is not different from at least some of the other proposed techniques like the RSBP technique and the RSBM technique.

Here the strength of the encryption policy becomes prominent if blocks are constructed of varying sizes. But, since here neither any cycle is formed, nor there exists any option for a source block to choose the corresponding encrypted block, the degree of variation in sizes of blocks should be made much high, so that a reasonably long key space becomes inevitable for a successful decryption.

Figure 8.2.4.1 in chapter 8 shows one proposed format of a 110-bit key constructed by strictly following a set of protocols for blocks formation. By allowing more flexibility in the approach of blocks formation, a much longer key space can be generated.

Evaluating from all perspectives, it can be concluded that the RPMS encryption policy is expected to offer a very satisfactory level of information security with providing construction of varying sizes of blocks. Also it may participate successfully in the cascaded approach of implementation discussed in chapter 9.