4

Encryption Through Recursive Paired Parity Operation (RPPO)

Contents

Pages

4.1	**Introduction**	**132**
4.2	**The Scheme**	**133**
4.3	**Implementation**	**136**
4.4	**Results**	**143**
4.5	**Analysis and Conclusion with RPSP and TE**	**156**

4.1 Introduction

Like RPSP, described in chapter 2, the Recursive Paired Parity Operation or the RPPO is also a secret-key cipher system and it also generates a cycle to regenerate the source block. Here also during the process of forming the cycle, any intermediate block can be considered as the encrypted block. After running the same technique for a finite number of more iterations, the source block is regenerated. This is under the part of decryption. In RPSP, one generating function was used to re-orient the positions of different bits in each of the iterations in forming the cycle. In RPPO, the bits are not re-oriented only in their positions but a special Boolean operation is performed on the source and the subsequent blocks of bits. The operation, called the Recursive Paired Parity Operation, is such that after a finite number of iterations, the source block is regenerated. In RPPO, the number of iterations required to complete the cycle follows a certain mathematical policy, while in RPSP this number was not as per a fixed policy.

After decomposing the source stream of bits into a finite number of blocks, the RPPO technique can be applied on each block. Depending on the size of a block, it is fixed that after how many iterations the source block will be regenerated. Accordingly, any intermediate block can be considered as the corresponding encrypted block. It is a wise strategy to take different blocks of varying sizes, so that the key space becomes large enough to almost nullify the chance of breaking the cipher through cryptanalysis. The same type of strategy was also advised in chapter 2 for the RPSP technique.

The technique does not cause any storage overhead. The implementation is well proven with the positive outcome [45, 46, 49].

Section 4.2 of this chapter discusses the entire scheme of this technique with simple examples. Like the RPSP technique, since the entire scheme is the combination of the encryption and the decryption processes, this section also includes how one part of the scheme can be used for the encryption and how the remaining part can be used for the decryption. Section 4.3 shows a simple implementation of the technique, where the same text message as in section 2.3 has been considered for the purpose of transmission using the encryption. Section 4.4 gives the results obtained after implementing the RPPO technique on the same set of real-life files of different categories. Section 4.5 is an analytical presentation of the technique with the concluding remark, where the RPPO technique has been analyzed from different perspectives and also the mathematical policy that exists in finding the number of iterations required to complete a cycle has been presented.

4.2 The Scheme

The technique, like all other techniques described in the thesis, considers the plaintext as a stream of finite number of bits N, and is divided into a finite number of blocks, each also containing a finite number of bits n, where 1 <= n <= N [45].

Let $P = s^0_0\ s^0_1\ s^0_2\ s^0_3\ s^0_4 \ldots s^0_{n-1}$ is a block of size n in the plaintext. Then the first intermediate block $I_1 = s^1_0\ s^1_1\ s^1_2\ s^1_3\ s^1_4 \ldots s^1_{n-1}$ can be generated from P in the following way:

$s^1_0 = s^0_0$

$s^1_i = s^1_{i-1} \oplus s^0_i,\ 1 * i * (n-1)$; $\oplus$ stands for the exclusive OR operation.

Now, in the same way, the second intermediate block $I_2 = s^2_0\ s^2_1\ s^2_2\ s^2_3\ s^2_4 \ldots s^2_{n-1}$ of the same size (n) can be generated by:

$s^2_0 = s^1_0$

$s^2_i = s^2_{i-1} \oplus s^1_i,\ 1 * i * (n-1)$.

After this process continues for a finite number of iterations, which depends on the value of n, the source block P is regenerated.

If the number of iterations required to regenerate the source block is assumed to be I, the generation of any intermediate or the final block can be generalized as follows:

$s^j_0 = s^{j-1}_0$

$s^j_i = s^2_{i-1} \oplus s^{j-1}_i,\ 1 * i * (n-1)$; where $1 * j * i$.

In this generalized formulation system, the final block, which in turn is the source block, is generated when j = i.

Figure 4.2.1 pictorially represents this technique.

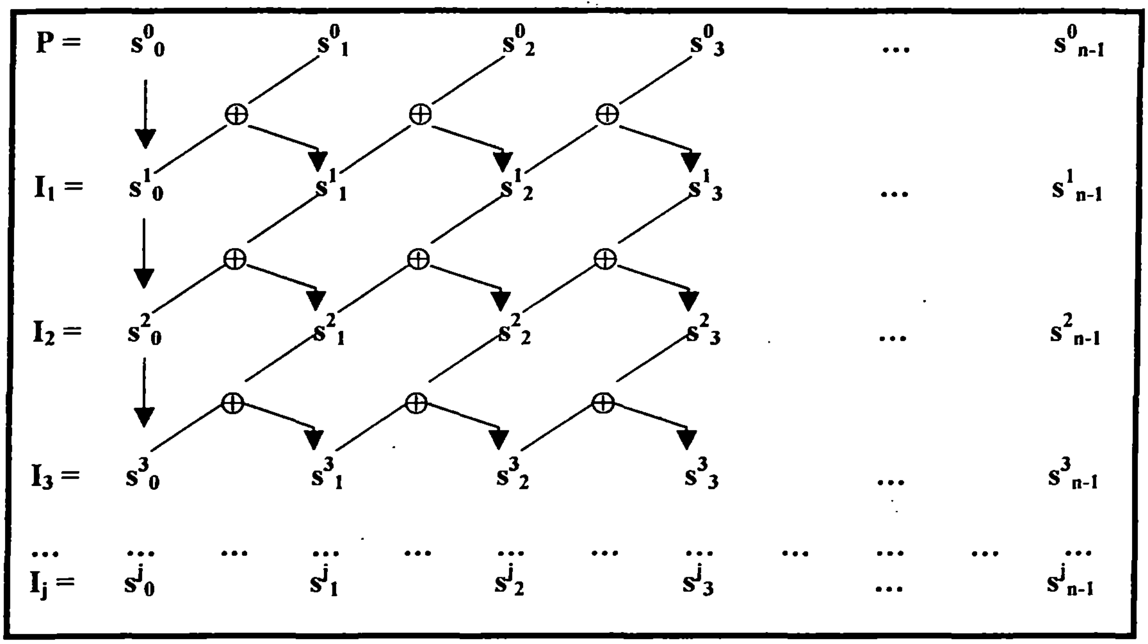

Figure 4.2.1
Pictorial Representation of the RPPO Technique

4.2.1 Example

To illustrate the technique, let P = 0101 be a 4-bit source block. Figure 4.2.1.1 shows the generation of the cycle for this sample block. Here it requires 4 iterations to regenerate the source block.

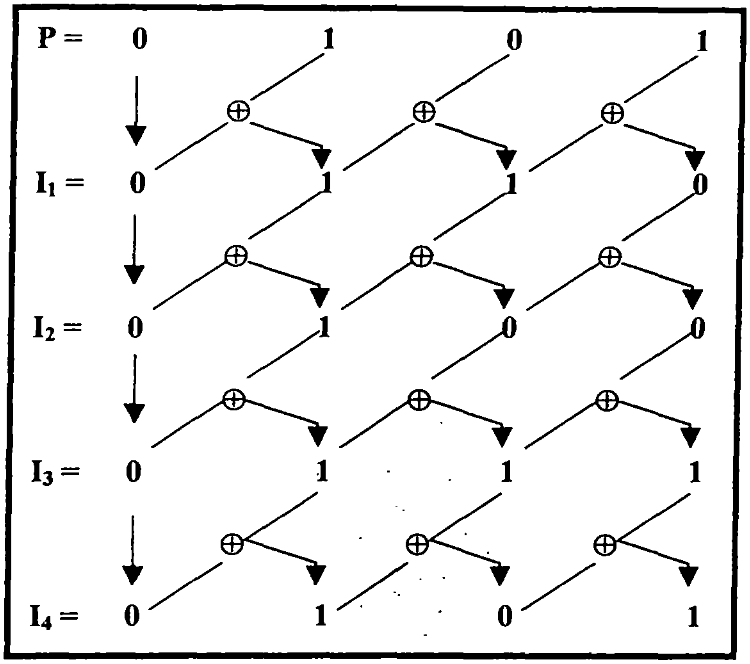

Figure 4.2.1.1
Pictorial Representation of the RPPO Technique for Source Block P = 0101

In this way, for different blocks in the plaintext corresponding cycles are formed. If the blocks are taken of the same size, the number of iterations required in forming the cycles will be equal and hence that number of iterations will be required to complete the cycle for the entire stream of bits.

With respect to one single block of bits, any intermediate block during the process of forming the cycle can be considered as the encrypted block. If the total number of iterations required to complete the cycle is P and the ith block is considered to be the encrypted block, then a number of (P – i) more iterations will be required to decrypt the encrypted block, i.e., to regenerate the source block.

Now, if the process of encryption is considered for the entire stream of bits, then it depends on how the blocks have been formed. Out of the entire stream of bits, different blocks can be formed in two ways:

1. Blocks with equal size
2. Blocks with different sizes.

In the case of blocks with equal length, if for all blocks, intermediate blocks after a fixed number of iterations are considered as the corresponding encrypted blocks, then that very number of iterations will be required for encrypting the entire stream of bits. The key of the scheme will be quite simple, consisting of only two information, one being the fixed block size and the other being the fixed number of iterations for all the blocks used during the encryption. On the other hand, for different source blocks different intermediate blocks may be considered as the corresponding encrypted blocks. For example, the policy may be something like that out of three source blocks B_1, B_2, B_3 in a source block of bits, the 4th, the 7th and the 5th intermediate blocks respectively are being considered as the encrypted blocks. In such a case, the key of the scheme will become much more complex, which in turn will ensure better security.

In the case of blocks with varying lengths, different blocks will require different numbers of iteration to form the corresponding cycle. So, the LCM value, say, P, of all these numbers will give the actual number of iterations required to form the cycle for the entire stream. Now, if i number of iterations are performed to encrypt the entire stream, then a number of (P – i) more iterations will be required to decrypt the encrypted stream.

4.3 Implementation

In this section, let us consider the same plaintext (P): Data Encryption to encrypt it using the RPPO technique. The corresponding stream of bits (S) of length 120 bits is as follows:

01000100/01100001/01110100/01100001/11111111/01000101/01101110/01100011/
01110010/01111001/01110000/01110100/01101001/01101111/01101110

Here "/" is used as the separator between successive bytes.

Blocks can be chosen in any manner. Here we choose blocks to be of varying sizes. Say, following are the different blocks constructed from S:

S_1 = 0100010001 (10 bits)

S_2 = 10000101110100 (14 bits)

S_3 = 0110000111111111 (16 bits)

S_4 = 010001010110111001100011 (24 bits)

S_5 = 01110010 (8 bits)

S_6 = 01111001011100000111010001101001 (32 bits)

S_7 = 0110111101101110 (16 bits)

Tables 4.3.1 to 4.3.7 show the formation of cycles for blocks S_1, S_2, S_3, S_4, S_5, S_6 and S_7 respectively. Now, for each of the blocks, an arbitrary intermediate block, as indicated in each table, is considered as the encrypted block.

Table 4.3.1
Formation of Cycle for Block S_1 for RPPO Technique

Source Block	0100010001
Block (I_{11}) after iteration 1	0111100001
Block (I_{12}) after iteration 2	0101000001
Block (I_{13}) after iteration 3	0110000001
Block (I_{14}) after iteration 4	0100000001
Block (I_{15}) after iteration 5	0111111110
Block (I_{16}) after iteration 6	0101010100
Block (I_{17}) after iteration 7	0110011000
Block (I_{18}) after iteration 8	0100010000
Block (I_{19}) after iteration 9	0111100000
Block (I_{110}) after iteration 10	0101000000
Block (I_{111}) after iteration 11	0110000000
Block (I_{112}) after iteration 12	0100000000
Block (I_{113}) after iteration 13	0111111111
Block (I_{114}) after iteration 14	0101010101
Block (I_{115}) after iteration 15	0110011001
Block (I_{116}) after iteration 16 (Source Block)	0100010001

Encrypted Block

Table 4.3.2
Formation of Cycle for Block S_2 for RPPO Technique

Source Block	10000101110100
Block (I_{21}) after iteration 1	11111001011000
Block (I_{22}) after iteration 2	10101110010000
Block (I_{23}) after iteration 3	11001011100000
Block (I_{24}) after iteration 4	10001101000000
Block (I_{25}) after iteration 5	11110110000000
Block (I_{26}) after iteration 6	10100100000000
Block (I_{27}) after iteration 7	11000111111111
Block (I_{28}) after iteration 8	10000101010101
Block (I_{29}) after iteration 9	11111001100110
Block (I_{210}) after iteration 10	10101110111011
Block (I_{211}) after iteration 11	11001011010010
Block (I_{212}) after iteration 12	10001101100011
Block (I_{213}) after iteration 13	11110110111101
Block (I_{214}) after iteration 14	10100100101001
Block (I_{215}) after iteration 15	11000111001110
Block (I_{216}) after iteration 16 (Source Block)	10000101110100

Encrypted Block

Table 4.3.3
Formation of Cycle for Block S_3 for RPPO Technique

Source Block	0110000111111111
Block (I_{31}) after iteration 1	0100000101010101
Block (I_{32}) after iteration 2	0111111001100110
Block (I_{33}) after iteration 3	0101010001000100
Block (I_{34}) after iteration 4	0110011110000111
Block (I_{35}) after iteration 5	0100010100000101
Block (I_{36}) after iteration 6	0111100111111001
Block (I_{37}) after iteration 7	0101000101010001
Block (I_{38}) after iteration 8	0110000110011110
Block (I_{39}) after iteration 9	0100000100010100
Block (I_{310}) after iteration 10	0111111000011000
Block (I_{311}) after iteration 11	0101010000010000
Block (I_{312}) after iteration 12	0110011111100000
Block (I_{313}) after iteration 13	0100010101000000
Block (I_{314}) after iteration 14	0111100110000000
Block (I_{315}) after iteration 15	0101000100000000
Block (I_{316}) after iteration 16 (Source Block)	0110000111111111

Encrypted Block

Table 4.3.4
Formation of Cycle for Block S_4 for RPPO Technique

Source Block	010001010110111001100011
Block (I_{41}) after iteration 1	011110011011010001000010
Block (I_{42}) after iteration 2	010100010010011110000011
Block (I_{43}) after iteration 3	011000011100010100000010
Block (I_{44}) after iteration 4	010000010111100111111100
Block (I_{45}) after iteration 5	011111100101000101010111
Block (I_{46}) after iteration 6	010101000110000110011010
Block (I_{47}) after iteration 7	011001111011111011101100
Block (I_{48}) after iteration 8	010001010010101101001000
Block (I_{49}) after iteration 9	011110011100110110001111
Block (I_{410}) after iteration 10	010100010111011011110101
Block (I_{411}) after iteration 11	011000011010010010100110
Block (I_{412}) after iteration 12	010000010011100011000100
Block (I_{413}) after iteration 13	011111100010111101111000
Block (I_{414}) after iteration 14	010101000011010110101111
Block (I_{415}) after iteration 15	011001111101100100110101
Block (I_{416}) after iteration 16	010001010110111000100110
Block (I_{417}) after iteration 17	011110011011010000111011
Block (I_{418}) after iteration 18	010100010010011111010010
Block (I_{419}) after iteration 19	011000011100010101100011
Block (I_{420}) after iteration 20	010000010111100110111101
Block (I_{421}) after iteration 21	011111100101000100101001
Block (I_{422}) after iteration 22	010101000110000111001110
Block (I_{423}) after iteration 23	011001111011111010001011
Block (I_{424}) after iteration 24	010001010010101100001101
Block (I_{425}) after iteration 25	011110011100110111110110
Block (I_{426}) after iteration 26	010100010111011010100100
Block (I_{427}) after iteration 27	011000011010010011000111
Block (I_{428}) after iteration 28	010000010111100010000101
Block (I_{429}) after iteration 29	011111100010111100000110
Block (I_{430}) after iteration 30	010101000011010111111011
Block (I_{431}) after iteration 31	011001111101100101010010
Block (I_{432}) after iteration 32	010001010110111001100011

Encrypted Block

Table 4.3.5
Formation of Cycle for Block S_5 for RPPO Technique

Source Block	01110010
Block (I_{51}) after iteration 1	01011100
Block (I_{52}) after iteration 2	01101000
Block (I_{53}) after iteration 3	01001111
Block (I_{54}) after iteration 4	01110101
Block (I_{55}) after iteration 5	01011001
Block (I_{56}) after iteration 6	01101110
Block (I_{57}) after iteration 7	01001011
Block (I_{58}) after iteration 8	01110010

Encrypted Block

Table 4.3.6

Formation of Cycle for Block S_6 for RPPO Technique

Source Block	01111001011100000111010001101001
Block (I_{61}) after iteration 1	01010001101000000101100001001110
Block (I_{62}) after iteration 2	01100001001111111001000001110100
Block (I_{63}) after iteration 3	01000001110101010001111110100111
Block (I_{64}) after iteration 4	01111110100110011110101011000101
Block (I_{65}) after iteration 5	01010100111011101011001101111001
Block (I_{66}) after iteration 6	01100111010010110010001001010001
Block (I_{67}) after iteration 7	01000101100011011100001110011110
Block (I_{68}) after iteration 8	01111001000010010111110100010100
Block (I_{69}) after iteration 9	01010001111100011010100111100111
Block (I_{610}) after iteration 10	01100001010111101100111010111010
Block (I_{611}) after iteration 11	01000001100101001000101100101100
Block (I_{612}) after iteration 12	01111110111001110000110111001000
Block (I_{613}) after iteration 13	01010100101110100000100101110000
Block (I_{614}) after iteration 14	01100111001011000000111001011111
Block (I_{615}) after iteration 15	01000101110010000000101110010101
Block (I_{616}) after iteration 16	01111001011100000000110100011001
Block (I_{617}) after iteration 17	01010001101000000000100111101110
Block (I_{618}) after iteration 18	01100001001111111111000101001011
Block (I_{619}) after iteration 19	01000001110101010101111001110010
Block (I_{620}) after iteration 20	01111110100110011001010001011100
Block (I_{621}) after iteration 21	01010100111011101110011110010111
Block (I_{622}) after iteration 22	01100111010010110100010100011010
Block (I_{623}) after iteration 23	01000101100011011000011000010011
Block (I_{624}) after iteration 24	01111001000010010000010000011101
Block (I_{625}) after iteration 25	01010001111100011111100000010110
Block (I_{626}) after iteration 26	01100001010111101010111111100100
Block (I_{627}) after iteration 27	01000001100101001100101010111000
Block (I_{628}) after iteration 28	01111110111001110111001100101111
Block (I_{629}) after iteration 29	01010100101110100101110111001010
Block (I_{630}) after iteration 30	01100111001011000110100101110011
Block (I_{631}) after iteration 31	01000101110010000100111001011101
Block (I_{632}) after iteration 32	01111001011100000111010001101001

Encrypted Block

Table 4.3.7
Formation of Cycle for Block S_7 for RPPO Technique

Source Block	0110111101101110
Block (I_{71}) after iteration 1	0100101001001011
Block (I_{72}) after iteration 2	0111001110001101
Block (I_{73}) after iteration 3	0101110100001001
Block (I_{74}) after iteration 4	0110100111110001
Block (I_{75}) after iteration 5	0100111000100001
Block (I_{76}) after iteration 6	0111010011000001
Block (I_{77}) after iteration 7	0101100010000001
Block (I_{78}) after iteration 8	0110111100000001
Block (I_{79}) after iteration 9	0100101000000001
Block (I_{710}) after iteration 10	0111001111111110
Block (I_{711}) after iteration 11	0101110101010100
Block (I_{712}) after iteration 12	0110100110011000
Block (I_{713}) after iteration 13	0100111011101111
Block (I_{714}) after iteration 14	0111010010110101
Block (I_{715}) after iteration 15	0101100011011001
Block (I_{716}) after iteration 16 (Source Block)	0110111101101110

Encrypted Block

As indicated in tables 4.3.1 to 4.3.7, intermediate blocks I_{19} (0111100000), I_{214} (10100100101001), I_{314} (0111100110000000), I_{43} (011000011100010100000010), I_{57} (01001011), I_{625} (01010001111100011111000000010110) and I_{77} (0101100010000001) are considered as the encrypted blocks, so that these blocks form the encrypted stream as follows:

0111100000/10100100101001/0111100110000000/011000011100010100000010/01001 011/01010001111100011111000000010110/0101100010000001, "/" being used as only the separator.

The encrypted stream can be rewritten as the series of bytes as follows:

01111000/00101001/00101001/01111001/10000000/01100001/11000101/00000010/010 01011/01010001/11110001/11111000/00010110/01011000/10000001.

Converting the bytes into the corresponding characters, the following text is obtained as the encrypted text, which is to be transmitted/stored:

C = x))y a┼KQ±°■Xú

Now, since while encrypting in this case, the source stream is decomposed into sub-streams in a different way than what was done in section 2.3 for the same example, the process of decryption also in this case is much more complicated.

After converting the ciphertext C into a stream of bits, the technique of decomposition into several blocks of bits should follow the same way the source was decomposed. Then for each block the necessary number of iterations is to be performed to get the corresponding source block. For example, to get the source block corresponding to the encrypted block I_{19}, the same iterations are to be applied $(16 - 9) = 7$ times because as per the mathematical policy a total of 16 iterations are required to complete the cycle, and as was shown in table 4.3.1, the encrypted block I_{19} was obtained after a total of 9 iterations. After obtaining all source blocks in this way, they are grouped together to form what would be the source stream of bits, from which the plaintext is achieved.

4.4 Results

Section 4.4.1 shows results of the encryption/decryption time, the number of operations for encryption and decryption, and the chi square value, section 4.4.2 depicts pictorial result of the frequency distribution tests, section 4.4.3 presents results of the comparison with the RSA system.

4.4.1 Result of Encryption/Decryption Time, Total Number of Operations, Chi Square Value

To experiment with the same set of sample files considered earlier, the technique of RPPO has been applied in a cascaded way with block sizes of 2^n, n increasing from 3 to 8. This means that first on the source file, the RPPO encryption technique is applied for blocks with the unique length of 8 bits. On the generated stream of bits, the same technique is applied with blocks with the unique length of 16 bits, and this process continues till the generation of stream of bits for blocks of the unique length of 256 bits. In each case, intermediate blocks generated after only one iteration are considered as target blocks, so that the process of decryption requires much more time and involves much more number of operations than the process of encryption. [36, 44, 46, 55, 56]

Section 4.4.1.1 shows the result on *EXE* files, section 4.4.1.2 shows the result on *COM* files, section 4.4.1.3 shows the result on *DLL* files, section 4.4.1.4 shows the result on *SYS* files and section 4.4.1.5 shows the result on *CPP* files.

4.4.1.1 Result for *EXE* Files

Table 4.4.1.1.1 gives the result of implementing the technique on *EXE* files. Ten files have been considered. There sizes range from 18432 bytes to 59398 bytes. The encryption time ranges from 0.604396 seconds to 1.978022 seconds. The decryption time ranges from 3.956044 seconds to 20.274725 seconds. The number of operations during the process of encryption ranges from 5562057 to 4256157816, whereas the same during the process of decryption ranges from 725704903 to 4630879800. The Chi Square value is observed to be between 20479 and 202973 with the degree of freedom ranging from 248 to 255.

Table 4.4.1.1.1
Result for *EXE* Files for RPPO Technique

Source File	Source Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In Seconds)	No. of Operations During Encryption	No. of Operations During Decryption	Chi Square Value	Degree of Freedom
TLIB.EXE	37220	1.263736	12.692307	5879407	762483595	146690	255
MAKER.EXE	59398	1.978022	20.274725	899004121	2106446259	166659	255
UNZIP.EXE	23044	0.769231	7.912087	2312980138	2781386689	20479	255
RPPO.EXE	35425	1.208791	12.087912	5562057	725704903	56083	255
PRIME.EXE	37152	1.263736	12.692307	852691357	1608069906	66283	255
TCDEF.EXE	26983	0.934066	3.956044	1739245468	1976720300	43640	254
TRIANGLE.EXE	36385	1.263736	12.362637	2022115101	2758695665	57049	255
PING.EXE	24576	0.824176	8.351648	2886377400	3386006712	142814	248
NETSTAT.EXE	32768	1.098901	11.153846	3475136184	4141308600	202973	255
CLIPBRD.EXE	18432	0.604396	6.318681	4256157816	4630879800	90561	255

A part of the table is diagrammatically represented in figure 4.4.1.1.1, where one graphical relationship is established between the source size and the encryption time for *EXE* files. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

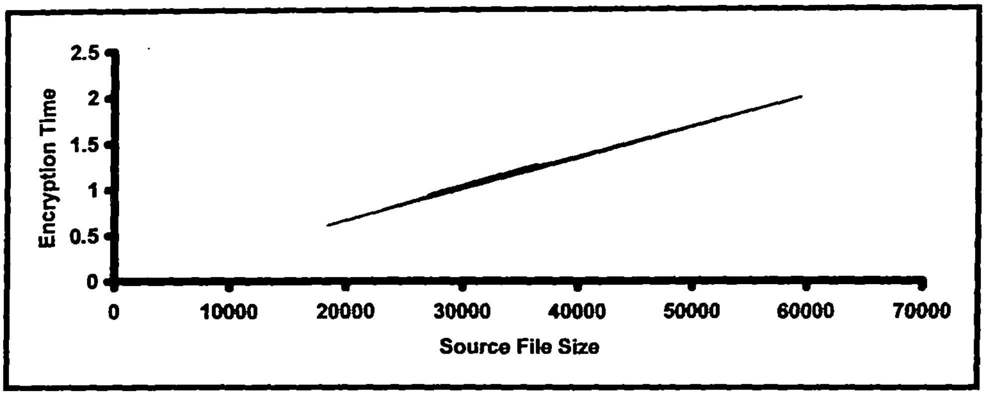

Figure 4.4.1.1.1
Relationship between Source Size and Encryption Time for .EXE Files in RPPO Technique

4.4.1.2 Result for *COM* Files

Table 4.4.1.2.1 gives the result of implementing the technique on different *COM* files. Ten files have been considered. There sizes range from 5239 bytes to 29271 bytes. The encryption time ranges from 0.219780 seconds to 1.043956 seconds. The decryption time ranges from 1.868132 seconds to 10.054945 seconds. The number of operations during the process of encryption ranges from 827008 to 1972543267. whereas the same during the process of decryption ranges from 107002375 to 2357808445. The Chi Square value is observed to be between 8801 and 80497 with the degree of freedom ranging from 230 to 255.

Table 4.4.1.2.1
Result for *COM* Files for RPPO Technique

Source File	Source Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In Seconds)	No. of Operations During Encryption	No. of Operations During Decryption	Chi Square Value	Degree of Freedom
EMSTEST.COM	19664	0.714286	6.758242	3106170	402678963	40182	255
THELP.COM	11072	0.494505	3.791209	471601446	696694860	29624	250
WIN.COM	24791	0.879121	8.516483	738433117	1242100033	53529	252
KEYB.COM	19927	0.769231	6.868132	1329918277	1734700225	61875	255
CHOICE.COM	5239	0.219780	1.868132	827008	107002375	11607	232
DISKCOPY.COM	21975	0.769231	7.527472	128353285	574771009	80497	254
DOSKEY.COM	15495	0.549451	5.329670	652269631	967144870	37393	253
MODE.COM	29271	1.043956	10.054945	1024683457	1619428633	80065	255
MORE.COM	10471	0.384615	3.626374	1721057212	1933794688	8801	230
SYS.COM	18967	0.659341	6.538461	1972543267	2357808445	47097	254

Figure 4.4.1.2.1 is constructed to establish the relationship between the source size and the decryption time for *COM* files. As it is observed from the figure, there exists a linear relationship between the source file size and the encryption time.

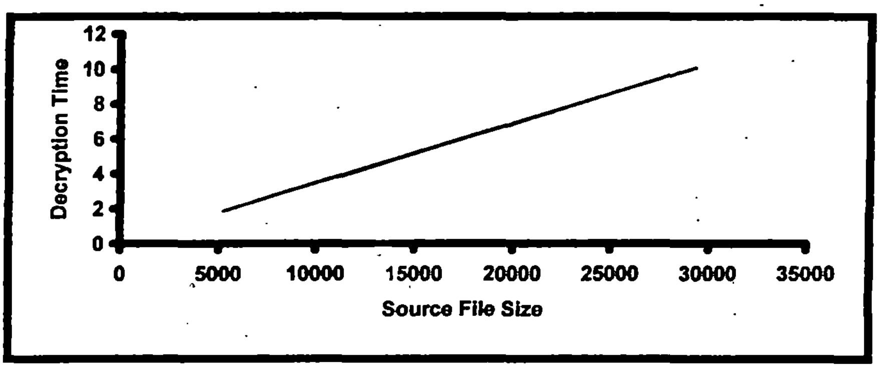

Figure 4.4.1.2.1
Relationship between Source Size and Decryption Time for .COM Files in RPPO Technique

4.4.1.3 Result for *DLL* Files

Table 4.4.1.3.1 gives the result of implementing the technique on different *DLL* files. Ten files have been considered. There sizes range from 3216 bytes to 58368 bytes. The encryption time ranges from 0.164835 seconds to 2.032967 seconds. The decryption time ranges from 1.098901 seconds to 19.945055 seconds. The number of operations during the process of encryption ranges from 5176320 to 5415873840, whereas the same during the process of decryption ranges from 5176320 to 6050819424. The Chi Square value is observed to be between 8907 and 534891 with the degree of freedom ranging from 217 to 255.

Table 4.4.1.3.1
Result for *DLL* Files for RPPO Technique

Source File	Source Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In Seconds)	No. of Operations During Encryption	No. of Operations During Decryption	Chi Square Value	Degree of Freedom
SNMPAPI.DLL	32768	1.153846	11.153846	5176320	5176320	99714	253
KPSHARP.DLL	31744	1.098901	10.879121	788300832	1433655360	291423	254
WINSOCK.DLL	21504	0.714286	7.362637	1545491808	1982667456	101308	252
SPWHPT.DLL	32792	1.153846	11.263736	2061306257	2727632447	320131	255
HIDCI.DLL	3216	0.164835	1.098901	2840151804	2905337271	8907	217
PFPICK.DLL	58368	2.032967	19.945055	2925543648	4112163264	201908	255
NDDEAPI.DLL	14032	0.494505	4.835165	4313768490	4598842899	78677	249
NDDENB.DLL	10976	0.384615	3.791209	4648510953	4871652690	56741	251
ICCCODES.DLL	20992	0.769231	7.197802	4912463472	5339230176	534891	252
KPSCALE.DLL	31232	1.043956	10.714285	5415873840	6050819424	242761	255

The relationship between the source size and the number of operations during encryption for *DLL* files is shown in figure 4.4.1.3.1, from which it is found that there is a tendency that the number of operations increases with the size of the source file considered for encryption, but the relationship is not exactly linear.

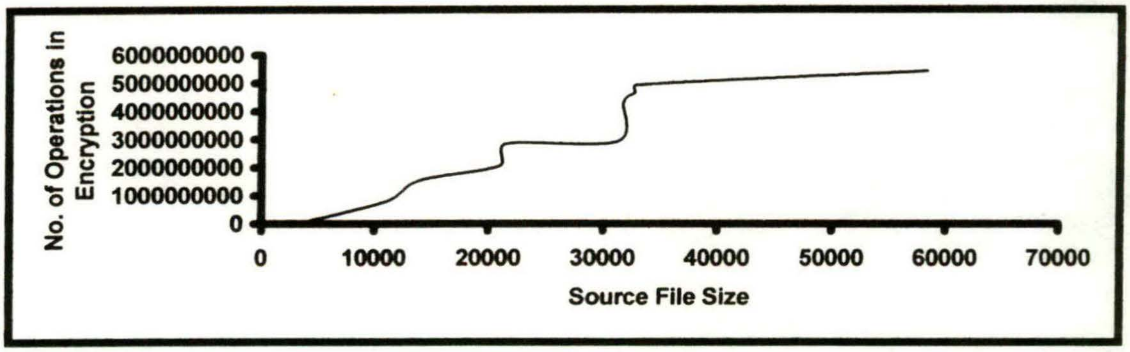

Figure 4.4.1.3.1
Relationship between Source Size and No. of Operations during Encryption for *.DLL* Files in RPPO Technique

4.4.1.4 Result for *SYS* Files

Table 4.4.1.4.1 gives the result of implementing the technique on different *SYS* files. Ten files have been considered. There sizes range from 1105 bytes to 33191 bytes. The encryption time ranges from 0.109890 seconds to 1.098901 seconds. The decryption time ranges from 0.384615 seconds to 11.318681 seconds. The number of operations during the process of encryption ranges from 412456 to 2859362712, whereas the same during the process of decryption ranges from 53241936 to 2881611334. The Chi Square value is observed to be between 1532 and 170454 with the degree of freedom ranging from 165 to 255.

Table 4.4.1.4.1
Result for *SYS* Files for RPPO Technique

Source File	Source Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In Seconds)	No. of Operations During Encryption	No. of Operations During Decryption	Chi Square Value	Degree of Freedom
HIMEM.SYS	33191	1.098901	11.318681	5242678	679877044	115511	255
RAMDRIVE.SYS	12663	0.439560	4.340659	795242728	1052347783	22506	241
USBD.SYS	18912	0.659341	6.428571	1098576273	1483056642	134840	255
CMD640X.SYS	24626	0.879121	8.406593	1551551070	2051961071	56728	255
CMD640X2.SYS	20901	0.714286	7.142857	2139380983	2564200130	54934	255
REDBOOK.SYS	5664	0.219780	1.978022	2636482815	2751631758	29329	230
IFSHLP.SYS	3708	0.164835	1.263736	2771565576	2846537706	7791	237
ASPI2HLP.SYS	1105	0.109890	0.384615	2859362712	2881611334	1532	165
DBLBUFF.SYS	2614	0.109890	0.934066	412456	53241936	4536	215
CCPORT.SYS	31680	1.098901	10.824176	5004450	649057860	170454	255

The linear relationship between the source size and the decryption time for *SYS* files is shown in figure 4.4.1.4.1. The figure establishes the fact that the decryption time varies almost linearly with the size of the source file.

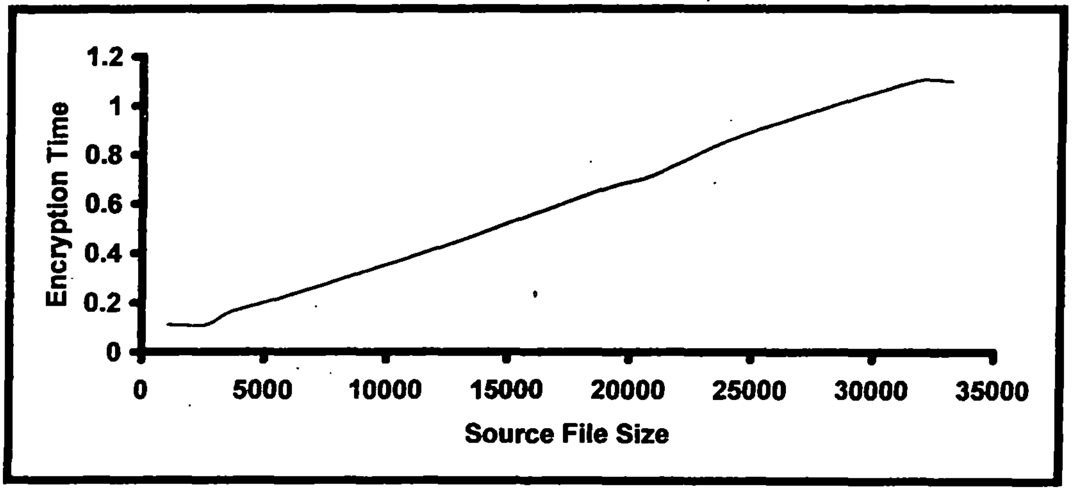

Figure 4.4.1.4.1
Relationship between Source Size and Decryption Time for *SYS* Files in RPPO Technique

4.4.1.5 Result for *CPP* Files

Table 4.4.5.1 gives the result of implementing the technique on different *CPP* files. Ten files have been considered. There sizes range from 1257 bytes to 32150 bytes. The encryption time ranges from 0.109890 seconds to 1.098901 seconds. The decryption time ranges from 0.439560 seconds to 11.043956 seconds. The number of operations during the process of encryption ranges from 2641311 to 2665493488, whereas the same during the process of decryption ranges from 342363288 to 2859364141. The Chi Square value is observed to be between 1644 and 74726 with the degree of freedom ranging from 69 to 90.

Table 4.4.1.5.1
Result for *CPP* Files for RPPO Technique

Source File	Source Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In Seconds)	No. of Operations During Encryption	No. of Operations During Decryption	Chi Square Value	Degree of Freedom
BRICKS.CPP	16723	0.604396	5.714285	2641311	342363288	53583	88
PROJECT.CPP	32150	1.098901	11.043956	404559709	1057855146	74726	90
ARITH.CPP	9558	0.384615	3.296703	1169177503	1363178286	18910	77
START.CPP	14557	0.494505	5.000000	1398114495	1693626175	31930	88
CHARTCOM.CPP	14080	0.494505	4.835165	1745558760	2031804720	39848	84
BITIO.CPP	4071	0.164835	1.428571	2080545508	2163171184	10608	70
MAINC.CPP	4663	0.164835	1.593407	2177797378	2272262683	9920	83
TTEST.CPP	1257	0.109890	0.439560	2288373428	2313769485	1644	69
DO.CPP	14481	0.494505	5.000000	2320339542	2614521826	31359	88
CAL.CPP	9540	0.384615	3.241758	2665493488	2859364141	16496	77

Figure 4.4.5.1 shows how the decryption time almost linearly changes with the encryption time for a given source size for *CPP* files.

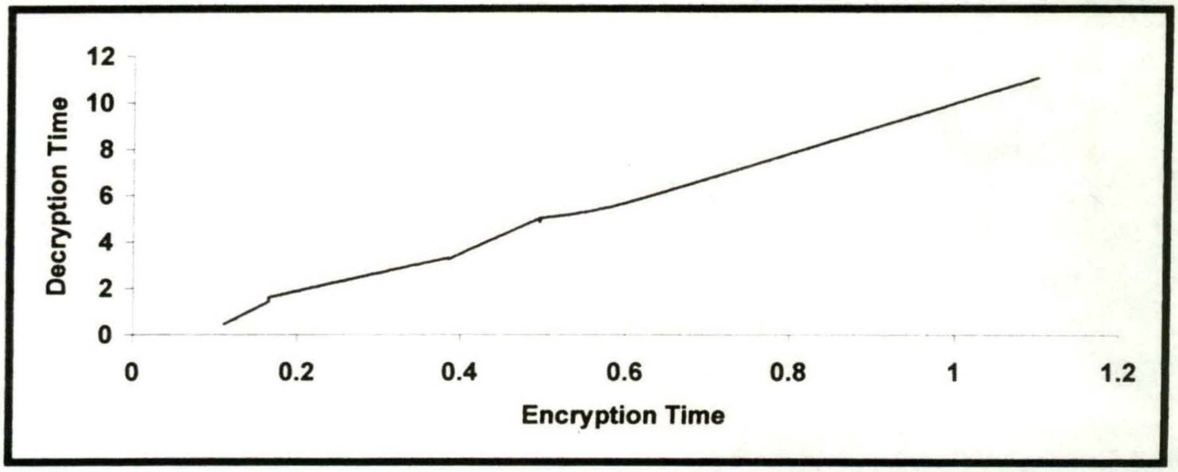

Figure 4.4.1.5.1
Graphical Relationship between Encryption Time and Decryption Time for *.CPP* Files in RPPO Technique

4.4.2 Result of Frequency Distribution Tests

For all sample files, frequency distribution tests for all characters have been performed. 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.

Each of the figures from 4.4.2.1 to 4.4.2.5 exhibits frequency distribution for different files, one from each category.

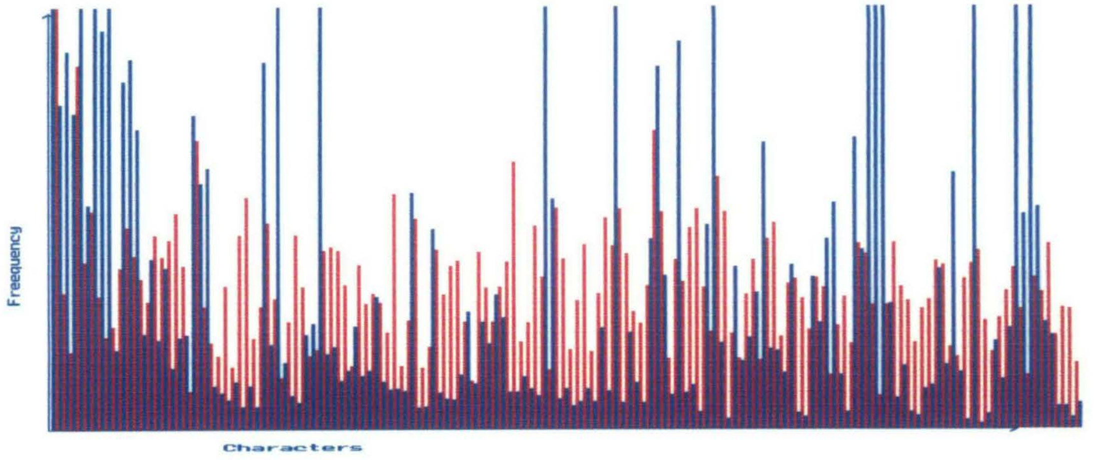

Figure 4.4.2.1
Frequency Distribution between TLIB.EXE and its Encrypted File for RPPO Technique

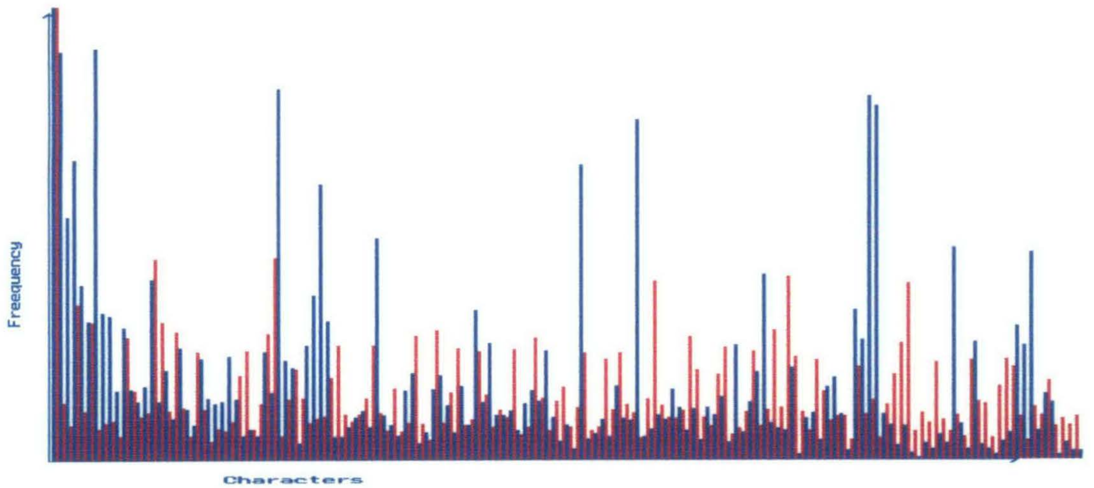

Figure 4.4.2.2
Frequency Distribution between KEYB.COM and its Encrypted File for RPPO Technique

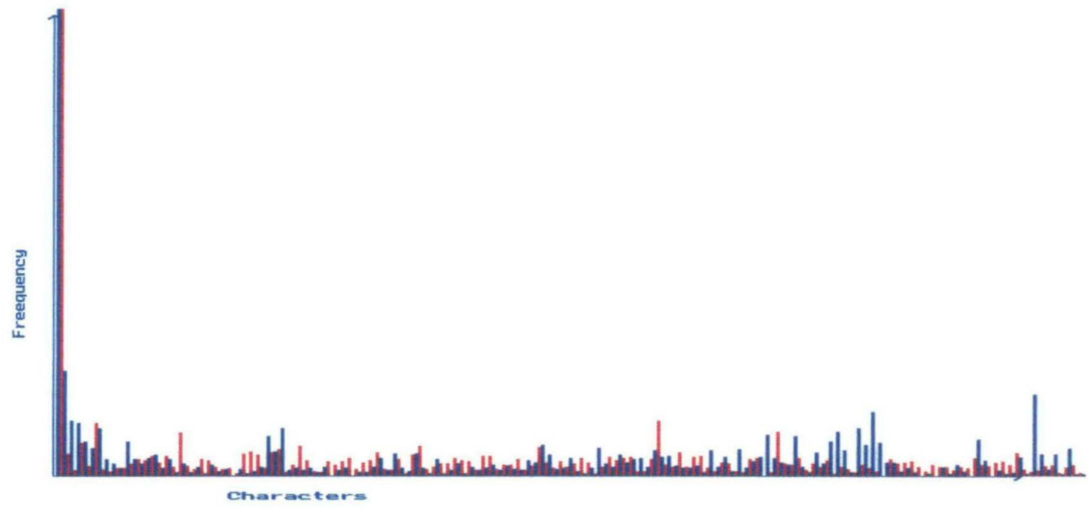

Figure 4.4.2.3
Frequency Distribution between HIDCI.DLL and its Encrypted File for RPPO Technique

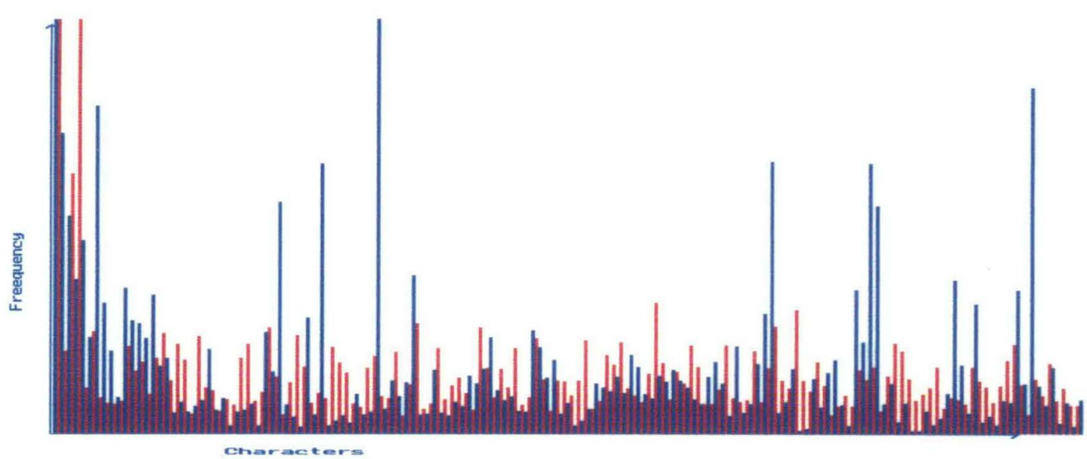

Figure 4.4.2.4
Frequency Distribution between HIMEM.SYS and its Encrypted File for RPPO Technique

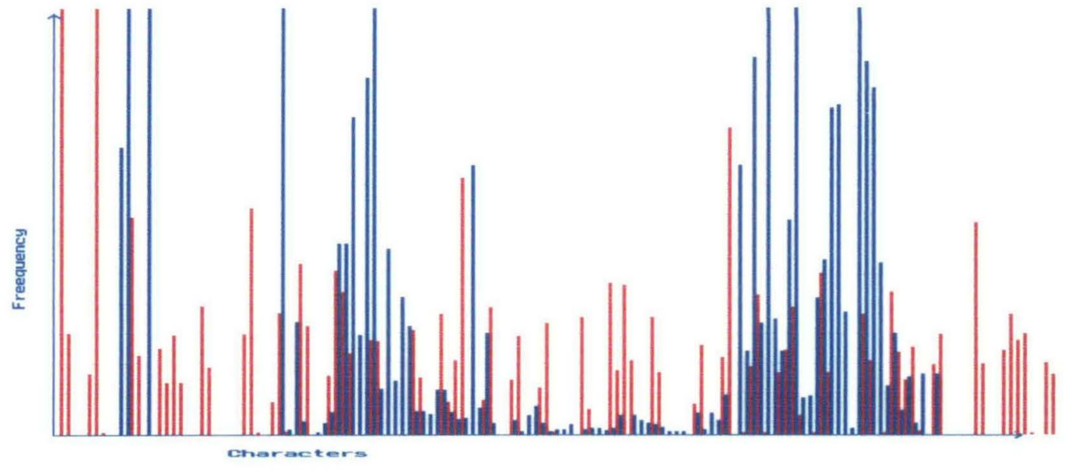

Figure 4.4.2.5
Frequency Distribution between START.CPP and its Encrypted File for RPPO Technique

4.4.3 Comparison with RSA Technique

The comparison between the proposed RPPO technique and the existing RSA system has been performed in terms of chi square values between .CPP sample files and respective encrypted files implementing both the techniques. Table 4.4.3.1 enlists the result. The same ten sample files have been considered. The Chi Square value between each of these files and the corresponding encrypted file using the proposed RPPO technique ranges from 1644 to 74726, whereas the same between each of the source files and the corresponding encrypted file using the existing RSA technique ranges from 3652 to 655734. The degree of freedom for these files ranges from 69 to 90 [40, 41].

Table 4.4.3.1
Comparison of Chi Square Values between RPPO and RSA

Source File (1)	Source Size	Encrypted File using RPPO Technique (2)	Encrypted File using RSA Technique (3)	Chi Square Value between (1) and (2)	Chi Square Value between (1) and (3)	Degree of Freedom
BRICKS.CPP	16723	*FOX1.CPP*	*CPP1.CPP*	53583	200221	88
PROJECT.CPP	32150	*FOX2.CPP*	*CPP2.CPP*	74726	197728	90
ARITH.CPP	9558	*FOX3.CPP*	*CPP3.CPP*	18910	273982	77
START.CPP	14557	*FOX4.CPP*	*CPP4.CPP*	31930	49242	88
CHARTCOM.CPP	14080	*FOX5.CPP*	*CPP5.CPP*	39848	105384	84
BITIO.CPP	4071	*FOX6.CPP*	*CPP6.CPP*	10608	52529	70
MAINC.CPP	4663	*FOX7.CPP*	*CPP7.CPP*	9920	4964	83
TTEST.CPP	1257	*FOX8.CPP*	*CPP8.CPP*	1644	3652	69
DO.CPP	14481	*FOX9.CPP*	*CPP9.CPP*	31359	655734	88
CAL.CPP	9540	*FOX10.CPP*	*CPP10.CPP*	16496	216498	77

Figure 4.4.3.1 presents the comparison graphically. Here red horizontal bars stand for Chi Square values for the RSA technique and blue horizontal bars stand for those for the proposed RPPO technique.

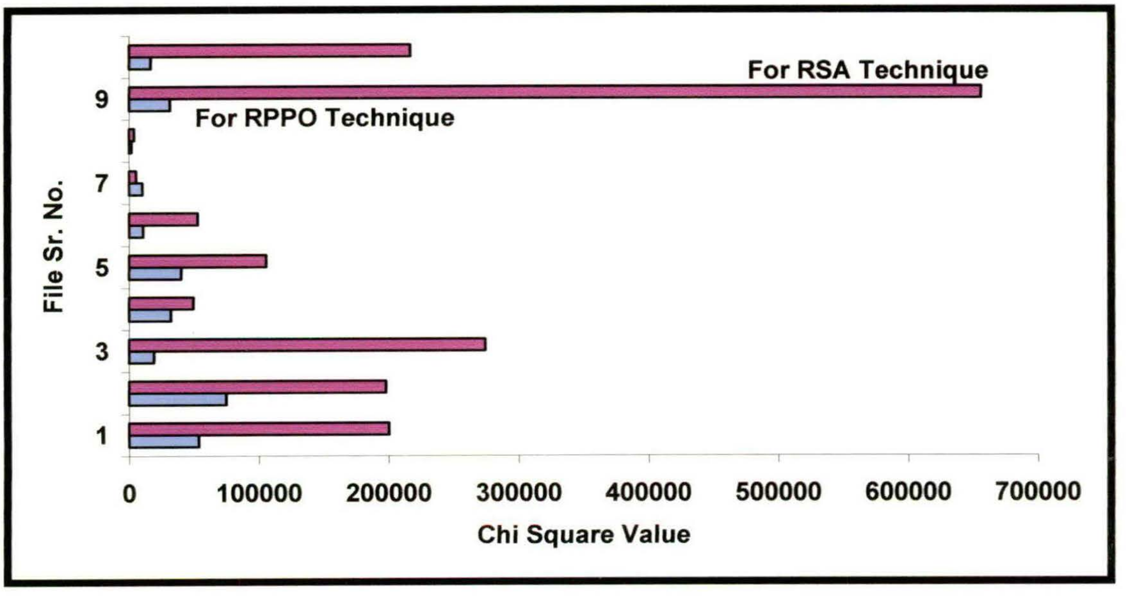

Figure 4.4.3.1
Comparison between Proposed RPPO Technique and Existing RSA Technique

4.5 Analysis and Conclusion including Comparison with RPSP, TE

On the basis of the practical implementation, the proposed RPPO technique has been assessed in comparison with the RPSP and the TE techniques, discussed in the last two chapters, in section 4.5.1. The total number of iterations required to complete the cycle, i.e., to regenerate the source block, follows a certain mathematical policy. Here first of all this policy has been presented in section 4.5.2. Section 4.5.3 presents the proof of the finiteness in regenerating the source block. Section 4.5.4 analyzes the results obtained for different sample files from different perspectives, and also draws a conclusion on the technique.

4.5.1 Comparative Analysis with RPSP and TE Techniques

The average of all the Chi Square values between all fifty sample files and their corresponding encrypted files, encrypted using the RPPO technique, is observed to be the maximum so far. Table 4.5.1.1 shows these results. From the table, it is observed that this value is 85350.94, comparing to 64188.04, obtained for the TE technique, and 10701.70, obtained for the RPSP technique. The average encryption time, 0.73186806 seconds, is the lowest, and the average decryption time, 7.03076904 seconds, is lesser than that for the RPSP technique but much more than that for the TE technique [48, 52]

In chapter 10, graphical comparisons have been drawn.

Table 4.5.1.1
Average Encryption/Decryption Time and Chi Square Value obtained in RPSP, TE, RPPO Techniques

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	

4.5.2 Formation of Cycle

The maximum number of iterations, I_{max}, required to complete a cycle for a certain block of bits of any length, say, N, can be evaluated using the following mathematical policy [46, 49]:

I_{max} $= N$, if $N = 2^p$, p being a finite positive integer;

$= 2^{p+1}$, if $2^p < N < 2^{p+1}$, p being a finite positive integer.

Table 4.5.2.1 shows values of I_{max} corresponding to different block lengths. In table 4.5.2.1, the value of N has been considered in the range of 4 to 259. When N is 4, the value of I_{max} is 4. For N ranging from 5 to 8, I_{max} is 8. For N ranging from 9 to 16, I_{max} is 16. In the range of values of 17 to 32 of N, I_{max} is 32. The value of I_{max} is 64 as N ranges from 33 to 64. For N ranging from 65 to 128, I_{max} is 128. If N is in the range of 129 to 256, I_{max} is 256. Finally, in the range of 257 to 259 of N, the value of I_{max} is 512.

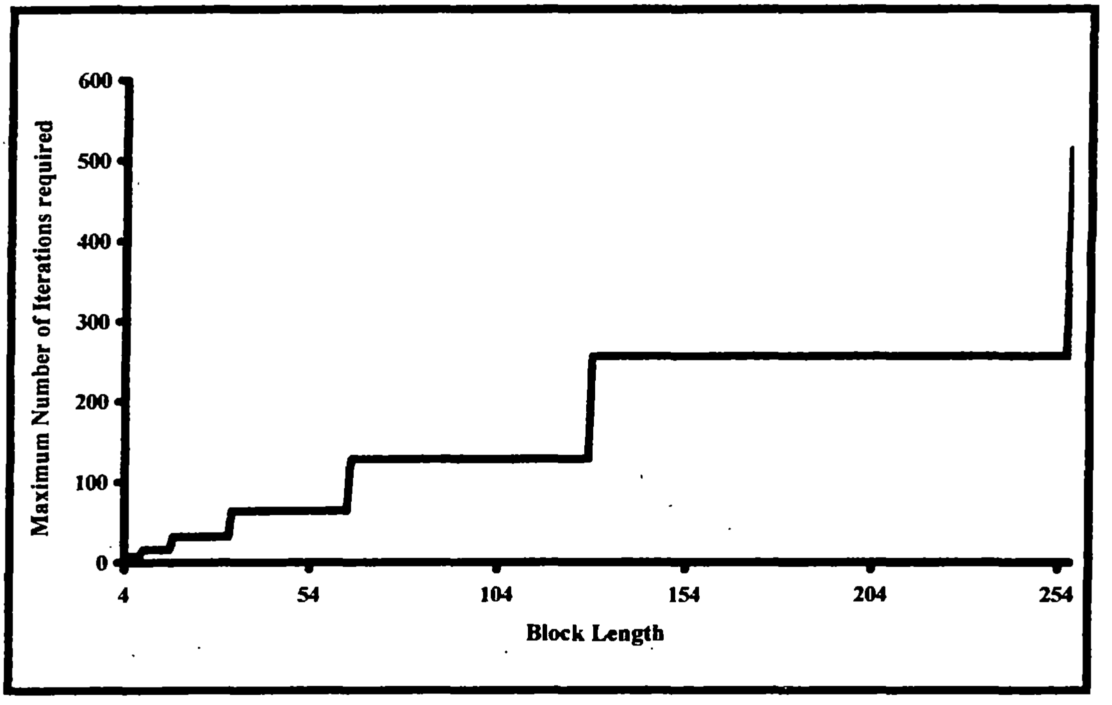

Figure 4.5.2.1
Diagrammatic Representation between N and I_{max}

Figure 4.5.2.1 depicts the diagrammatic relationship between N and I_{max}. The figure indicates that the value of I_{max} does not vary linearly with N. I_{max} is non-decreasing with the increment of N.

Table 4.5.2.1
Values of I_{max} for Different Block Lengths (N)

N	I_{max}	N	I_{max}	N	I_{max}	N	I_{max}	N	I_{max}	N	I_{max}	N	I_{max}	N	I_{max}
4	4	36	64	68	128	100	128	132	256	164	256	196	256	228	256
5	8	37	64	69	128	101	128	133	256	165	256	197	256	229	256
6	8	38	64	70	128	102	128	134	256	166	256	198	256	230	256
7	8	39	64	71	128	103	128	135	256	167	256	199	256	231	256
8	8	40	64	72	128	104	128	136	256	168	256	200	256	232	256
9	16	41	64	73	128	105	128	137	256	169	256	201	256	233	256
10	16	42	64	74	128	106	128	138	256	170	256	202	256	234	256
11	16	43	64	75	128	107	128	139	256	171	256	203	256	235	256
12	16	44	64	76	128	108	128	140	256	172	256	204	256	236	256
13	16	45	64	77	128	109	128	141	256	173	256	205	256	237	256
14	16	46	64	78	128	110	128	142	256	174	256	206	256	238	256
15	16	47	64	79	128	111	128	143	256	175	256	207	256	239	256
16	16	48	64	80	128	112	128	144	256	176	256	208	256	240	256
17	32	49	64	81	128	113	128	145	256	177	256	209	256	241	256
18	32	50	64	82	128	114	128	146	256	178	256	210	256	242	256
19	32	51	64	83	128	115	128	147	256	179	256	211	256	243	256
20	32	52	64	84	128	116	128	148	256	180	256	212	256	244	256
21	32	53	64	85	128	117	128	149	256	181	256	213	256	245	256
22	32	54	64	86	128	118	128	150	256	182	256	214	256	246	256
23	32	55	64	87	128	119	128	151	256	183	256	215	256	247	256
24	32	56	64	88	128	120	128	152	256	184	256	216	256	248	256
25	32	57	64	89	128	121	128	153	256	185	256	217	256	249	256
26	32	58	64	90	128	122	128	154	256	186	256	218	256	250	256
27	32	59	64	91	128	123	128	155	256	187	256	219	256	351	256
28	32	60	64	92	128	124	128	156	256	188	256	220	256	252	256
29	32	61	64	93	128	125	128	157	256	189	256	221	256	253	256
30	32	62	64	94	128	126	128	158	256	190	256	222	256	254	256
31	32	63	64	95	128	127	128	159	256	191	256	223	256	255	256
32	32	64	64	96	128	128	128	160	256	192	256	224	256	256	256
33	64	65	128	97	128	129	256	161	256	193	256	225	256	257	512
34	64	66	128	98	128	130	256	162	256	194	256	226	256	258	512
35	64	67	128	99	128	131	256	163	256	195	256	227	256	259	512

4.5.3 Proof of the Finiteness in Re-generating Source Block

This section only presents the proof through the empirical evidences.

4.5.3.1 Proof for Block Size of 2 Bits

Consider a 2-bit block P = AB. Then the block after the first iteration is $Y^2_1 = A\ [A \oplus B]$. Accordingly, the block after the second iteration is $Y^2_2 = A\ [A \oplus (A \oplus B)] = AB$, which is the source block itself. Hence after 2 iterations, the source block is regenerated.

4.5.3.2 Proof for Block Size of 3 Bits

Consider a 3-bit block Q = ABC, by adding an extra bit (C) to the block P considered in case 1.

Then after the first iteration, the block $Y^3_1 = A\ [A \oplus B]\ [(A \oplus B) \oplus C]$ is obtained.

Through the second iteration, the block $Y^3_2 = A\ [A \oplus (A \oplus B)]\ [(A \oplus (A \oplus B) \oplus (A \oplus B) \oplus C)]$ is generated.

Through the third iteration, the block $Y^3_3 = A\ [A \oplus (A \oplus (A \oplus B))]\ [(A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B) \oplus (A \oplus B) \oplus C))]$ is generated.

Finally, through the fourth iteration, the block $Y^3_4 = A\ [A \oplus (A \oplus (A \oplus (A \oplus B)))]\ [A \oplus (A \oplus (A \oplus (A \oplus B))) \oplus (A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B) \oplus (A \oplus B) \oplus C))]$ is generated, which is nothing but ABC, the source block Q.

Hence after 4 iterations the source block is regenerated.

4.5.3.3 Proof for Block Size of 4 Bits

Consider a 4-bit block R = ABCD, by adding an extra bit (D) to the block Q considered in case 2.

Then the block generated after the first iteration is $Y^4_1 = A\ [A \oplus B]\ [(A \oplus B) \oplus C]\ [((A \oplus B) \oplus C) \oplus D]$.

The block generated after the second iteration is $Y^4_2 = A [A \oplus (A \oplus B)] [(A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C)] [((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C)) \oplus (((A \oplus B) \oplus C) \oplus D)]$.

The block generated after the third iteration is $Y^4_3 = A [A \oplus (A \oplus (A \oplus B))] [(A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C))] [((A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C))) \oplus (((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C)) \oplus (((A \oplus B) \oplus C) \oplus D))]$.

Finally, the block generated after the fourth iteration is $Y^4_4 = A [A \oplus (A \oplus (A \oplus (A \oplus B)))] [(A \oplus (A \oplus (A \oplus (A \oplus B)))) \oplus ((A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C)))] [((A \oplus (A \oplus (A \oplus (A \oplus B)))) \oplus ((A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C)))) \oplus (((A \oplus (A \oplus (A \oplus B))) \oplus ((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C))) \oplus (((A \oplus (A \oplus B)) \oplus ((A \oplus B) \oplus C)) \oplus (((A \oplus B) \oplus C) \oplus D)))]$, which is nothing but ABCD, the source block R.

Hence after 4 iterations the source block is regenerated.

In this manner, the finiteness of the number of iterations to regenerate the source block can be proved for the source block of any length.

4.5.4 A Conclusive Analysis of Different Results Obtained

The encryption/decryption time is related to the size of the source file, and, being a bit-level application, it does not depend on the type of the file. On the other hand, the chi square value is entirely file-dependent.

Table 4.5.4.1 considers results for some files with almost same sizes but of different types. Here seven files have been considered. Their sizes range from 31232 bytes to 33191 bytes. It is observed from the table that encryption times are highly compatible to each other, ranging from 1.043956 seconds to 1.153846. Decryption times are also lying in the small range of 10.714285 seconds to 11.318681 seconds. But different Chi Square values are in the much bigger range of 74726 to 320131.

Table 4.5.4.1
Result of Encryption/Decryption Time and Chi Square value for Different Types of Files of Almost Same Sizes

File Name	File Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In seconds)	Chi Square Value	Degree of Freedom
PROJECT.CPP	32150	1.098901	11.043956	74726	90
CCPORT.SYS	31680	1.098901	10.824176	170454	255
HIMEM.SYS	33191	1.098901	11.318681	115511	255
KPSCALE.DLL	31232	1.043956	10.714285	242761	255
SPWHPT.DLL	32792	1.153846	11.263736	320131	255
SNMPAPI.DLL	32768	1.153846	11.153846	99714	253
NETSTAT.EXE	32768	1.098901	11.153846	202973	255

Figure 4.5.4.1 exhibits the sameness of encryption/decryption times for the files considered in table 4.5.4.1. Here gray vertical pillars stand for different encryption times and black vertical pillars stand for different decryption times. From the figure, it is observed that gray pillars are almost of the same height and the same is true for black pillars also. This indicates that whatever be the types of files considered, being of almost same size, encryption/decryption times do not differ too much.

Figure 4.5.4.2 establishes the file-dependency of the chi square value. Here it is observed that although files are of almost similar sizes, different vertical pillars are of different lengths. This indicates that Chi Square values are related to contents of files [44].

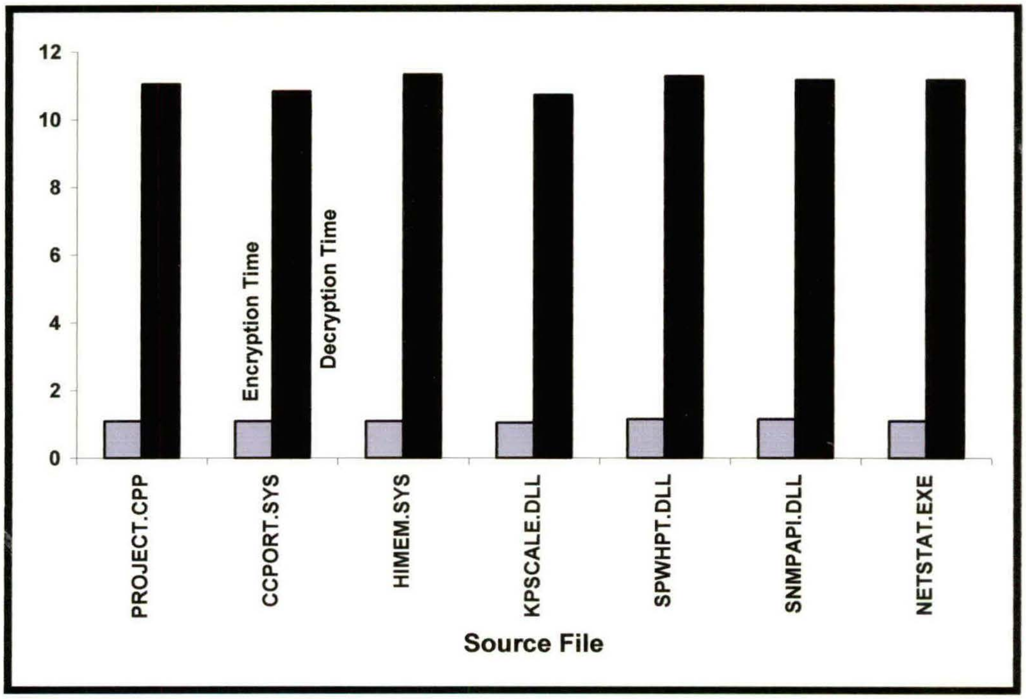

Figure 4.5.4.1
The Sameness in Encryption/Decryption Time for Files of Almost Similar Sizes

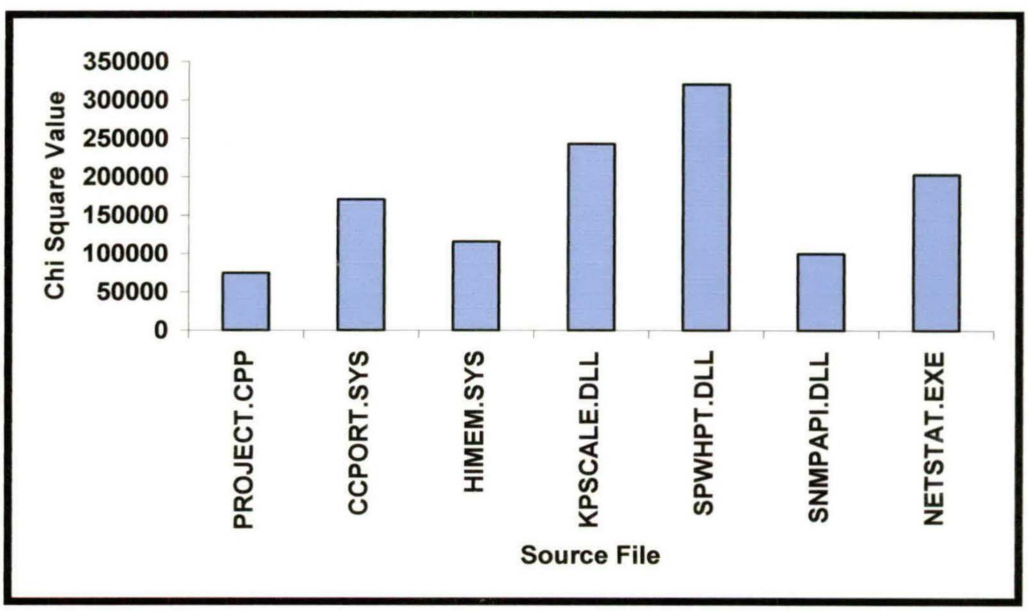

Figure 4.5.3.2
The File-Dependency in Chi Square Value for Files of Almost Similar Sizes

Results obtained from the frequency distribution tests indicate that the encrypted characters are well distributed. Figure 4.4.3.1 suggests that the performance of the RPPO technique in terms of the chi square value is not as good as of the existing RSA technique. But the real strength of this proposed technique, like the other proposed techniques, lies on the formation of a long key space, which can be achieved by constructing blocks of varying sizes, and by arbitrary choice of a block as the encrypted block from any of the intermediate blocks generated during the formation of the cycle. The RPPO encryption policy is expected to ensure a highly satisfactory performance mainly due to the level of flexibility it offers in encrypting a file.