Encryption Through Recursive Paired Parity Operation (RPPO)

To be submitted by

SOUVIK SAHA

In Partial Fulfillment for the degree of

Master of Computer Applications

Under the Supervision of

Prof. Jyotsna Kumar Mandal

Department of Computer Science and Engineering

University of Kalyani

Department of Computer Science and Engineering

University of Kalyani

Kalyani, Nadia, Pin code: 741235,West Bengal, India

June 2024

University of Kalyani

Faculty Of Engineering, Technology And Management

Prof. Jyotsna Kumar Mandal
Professor
Department of Computer Science and Engineering,
University of Kalyani

Kalyani-741235
Nadia, West Bengal, India
Phone - +91 - 33 – 5809617
(Direct) +91 - 33 - 25828750/4378

Extn.-304/256/225
Fax:-+91-33-5809617

Supervisor's Certificate

This is to certify that the partial fulfilment of the project report entitled **"Encryption Through Recursive Paired Parity Operation (RPPO)"** submitted by Mr Souvik Saha, bearing Registration Number 2080013 of 2022-2023 and Roll No: 90/MCA/220026, a student of MCA in Department of Computer Science and Engineering under The University of Kalyani, is based upon his own work under my supervision and that neither his project nor any part of the project has been submitted for any degree or diploma or any other academic award anywhere before. I wish him all the success.

Place: Kalyani

Date: 25/06/2024

25/06/2024
Professor
Department of Computer Sc. & Engg.
University of Kalyani
Kalyani-741 235, India

Prof. Jyotsna Kumar Mandal
Professor
Department of Computer Science and Engineering
Faculty of Engineering, Technology and Management
University of Kalyani, Kalyani-741235, Nadia, West Bengal, India

Kousik Dasgupta 25/6/24

25/06/24

Statutory Declarations

Name of the Candidate: Souvik Saha

Title of the Project: Encryption Through Recursive Paired Parity Operation (RPPO)

Degree: Masters of Computer Application (M.C.A)

Name of the Guide: Prof. Jyotsna Kumar Mandal

Registration Number: 2080013 of 2022-2023

Roll Number: 90/MCA/220026

Place of Project: Department of Computer Science and Engineering,
University of Kalyani,
Kalyani, Nadia, West Bengal, India.

Declaration by the Student

I hereby declare that the work reported in the M.C.A Project entitled "**Encryption Through Recursive Paired Parity Operation (RPPO)**" is an authentic record of my work carried out under the supervision of Prof. Jyotsna Kumar Mandal. I have not submitted this work elsewhere for any other degree or diploma.

Place: Kalyani

Date: 25/06/2024

Souvik Saha

Souvik Saha

Signature of Student

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude and indebtedness to the many individuals who contributed to the success of my project. Foremost, I am fortunate to have had the guidance and support of my supervisor, Prof. Jyotsna Kumar Mandal. His uncountable advice and encouragement played a pivotal role in the systematic completion of my project, and I am confident that his influence will resonate throughout my career.

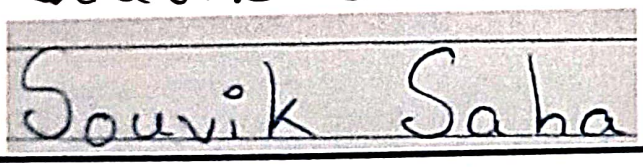
I extend my thanks to the Head of the Department, Prof. Kalyani Mali, as well as Prof. Anirban Mukhopadhyay, Prof. Utpal Biswas, Prof. Priya Ranjan Sinha Mahapatra, Dr. Debabrata Sarddar, Mr. Sukanta Majumdar, Mr Jaydeep Paul and Mrs. Shrabanti Kundu from the Department of Computer Science and Engineering at the University of Kalyani. Their support and cooperation were invaluable.

I am grateful to the entire Department of Computer Science and Engineering at the University of Kalyani for their assistance and collaborative efforts.

During the course of my project, I received tremendous support from my friends and classmates. My sincerest thanks go to them for their invaluable assistance.

No success would be complete without acknowledging the role of my parents. Their unwavering support and encouragement have been the driving force behind my education and achievements. They made it their life's mission to ensure I stayed focused on my goals. With their blessings and wise words, I found the strength to overcome challenges. I deeply appreciate their foresight and simplicity, which have been instrumental in helping me achieve my aspirations.

Souvik Saha



(Souvik Saha)

Content

1. Introduction

The Recursive Paired Parity Operation or the RPPO is a secret-key cipher system and it generates a cycle to regenerate the source block. Here 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 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. 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 technique does not cause any storage overhead.

2. The Scheme

1. Initialization:

   I. Define the plaintext as a stream of bits, P = s00 s01 s02 ... s0(n-1), where n is the block size.

   II. Set the number of iterations required to regenerate the source block, I.

2. Generating the first intermediate block:

   I. For each bit position i in the block ($0 \leq i \leq n-1$):

      a. Calculate si1 = s0(i-1) $\oplus$ s0i, where $\oplus$ denotes the exclusive-OR (XOR) operation.

   II. The first intermediate block is l1 = s10 s11 s12 ... s1(n-1).

3. Generating subsequent intermediate blocks:

   I. For each subsequent intermediate block j ($2 \leq j \leq I$):

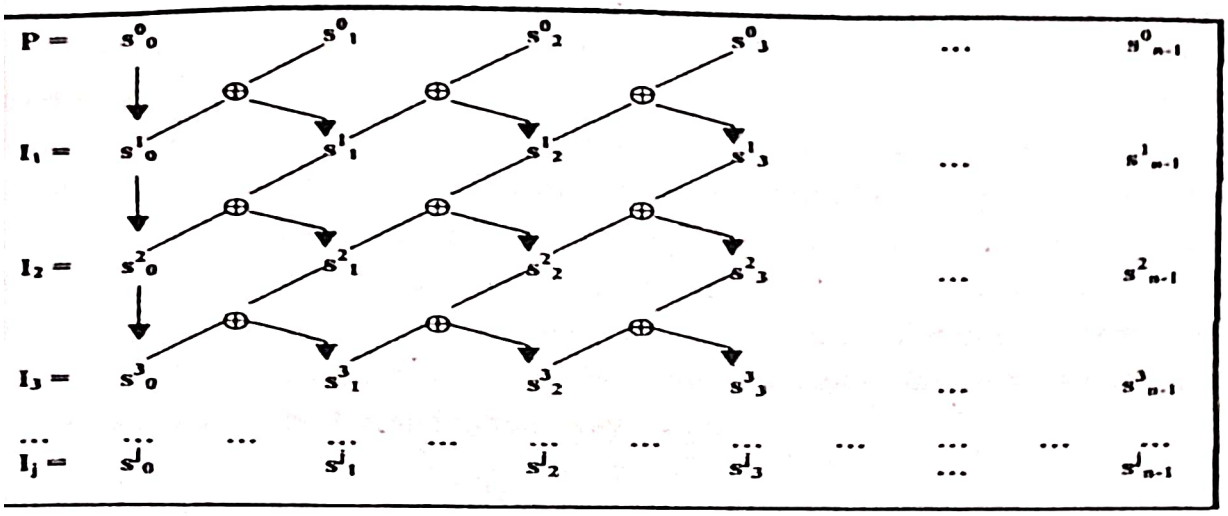
      a. For each bit position i in the block ($0 \leq i \leq n-1$):

         A. Calculate sij = si(j-1)(i-1) $\oplus$ si(j-1)i.

      b. The j-th intermediate block is lj = s(j)0 s(j)1 s(j)2 ... s(j)(n-1).

4. Generating the final block (source block regeneration):

   I. The final block, which is the regenerated source block, is obtained when j = i.



Pictorial Representation of RPPO Technique

2.1 Example

To illustrate the technique, let P = 0101 be a 4-bit source block. Figure 2.1 shows the generation of the cycle for this sample block. Here it requires 4 iterations to regenerate the source block.

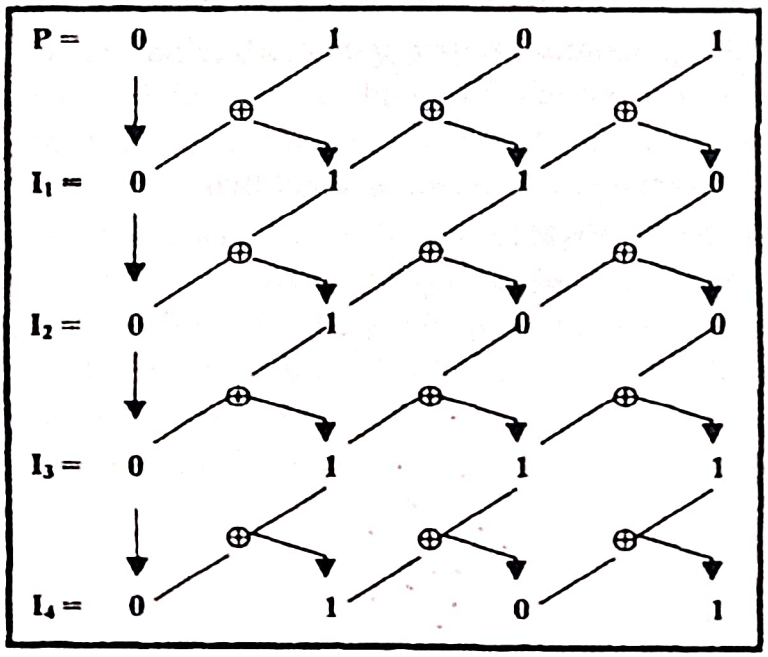


Figure 2.1

Pictorial Representation of the RPPO Technique or Source Block P = 0101

n this way, for different blocks in the plaintext corresponding cycles are formed. If ıe blocks are taken of the same size, the number of iterations required in forming the ycles will be equal and hence that number of iterations will be required to complete he cycle for the entire stream of bits. Concerning one single block of bits, any ıtermediate block during the process of forming the cycle can be considered as the ncrypted block. If the total number of iterations required to complete the cycle is P nd the ith block is considered to be the encrypted block, then a number of (P- i)more :erations will be required to decrypt the encrypted block, i.e, to regenerate the source lock. Now, if the process of encryption is considered for the entire stream of bits ıen it depends on how the blocks have been formed. Out of the entire stream of bits, ifferent blocks can be formed in two ways:

. Blocks with equal size.
. Blocks with different sizes.

n the case of blocks with equal length, if for all blocks, intermediate blocks after a ixed number of iterations are considered as the corresponding encrypted blocks then hat very number of iterations will be required for encrypting the entire stream of bits. he key of the scheme will be quite simple, consisting of only two pieces of nformation, one being the fixed block size and the other being the fixed number of terations for all the blocks used during the encryption. On the other hand, for different ource blocks different intermediate blocks may be considered as the corresponding :ncrypted blocks. For example, the policy may be something like that out of three :ource blocks B1, B2, B 3 in a source block of bits, the 4th, the 7th and the 5th ntermediate blocks respectively are being considered as the encrypted blocks. [n such ı case, the key of the scheme will become much more complex, which in turn will :nsure better security. In the case of blocks with varying lengths, different- blocks will require different numbers of iterations to form the corresponding cycle. So, the LCM value, say, P. of all these numbers will give the actual number of iterations required "red to form the cycle for the entire stream. Now, if i number of iterations are performed to encrypt the entire stream. then several (P- i) more iterations will be required to decrypt the encrypted stream.

3.Implementation

In this section, we explored the application of the Recursive Paired Parity Operation (RPPO) encryption technique on the plaintext "Data Encryption" represented as a bit stream (S) of length 120 bits. Unlike conventional approaches with fixed block sizes, we investigated the behavior of RPPO with blocks of differing lengths, aiming to analyze its adaptability and potential security benefits.

The chosen block sizes were:

- S1: 0100010001(10 bits)
- S2: 10000101110100(14 bits)
- S3: 0110000111111111(16 bits)
- S4: 01110010(8 bits)
- S5: 010001010110111001100011(24 bits)
- S6: 011110010111000001110100011010001(32 bits)
- S7: 01101111010110(16 bits)

Tables 3.1 to 3.7 show the formation of cycles for blocks S1, S2, S3, S4, S5, S6 and S7 respectively. Now, for each of the blocks, an arbitrary intermediate block, as indicated in each table, is considered as the encrypted block.

Table 3.1 for S1: 0100010001(10 bits)

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 3.2 for S2:10000101110100(14 bits)

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 3.5 for S5:010001010110111001100011(24bits)

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

Encrypted Block

Table 3.6 for S6: 01111001011100000111010001101001(32 bits)

Source Block	01111001011100000111010001101001
Block (I_{61}) after iteration 1	01010001101000000101100001001110
Block (I_{62}) after iteration 2	01100001001111111100100000111010 0
Block (I_{63}) after iteration 3	01000001110101010001111111010011 1
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	01010100101111010000010010111 0000
Block (I_{614}) after iteration 14	01100111001011000000111001011111
Block (I_{615}) after iteration 15	01000101110010000000101110010101
Block (I_{616}) after iteration 16	01111001011100000000011010001 1001
Block (I_{617}) after iteration 17	01010001101000000000100111101110
Block (I_{618}) after iteration 18	01100001001111111111100010100101 1
Block (I_{619}) after iteration 19	01000001110101010101111001110010
Block (I_{620}) after iteration 20	01111110100110011001010001011100
Block (I_{621}) after iteration 21	01010100111011101111001111100101 11
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	01010001111110001111110000001011 0
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 3.7 for S7: 0110111101101110(16 bits)

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 3.1 to 3.7, intermediate blocks I19 (0111100000), I214 (10100100101001), I314 (01111001100000000), I43 (011000011100010100000010), I57 (01001011), I625 (01010001111100011111100000010110) and I77 (0101100010000001) are considered as the encrypted blocks, so that these blocks form the encrypted stream as follows:
0111100000/10100100101001/0111100110000000/0110000111000101000000100/01001011/01010001111100011111100000010110 /f0101100010000001, "/" being used as only the separator. The encrypted stream can be rewritten as the series of bytes as follows:
01111000/00101001/001010001/0111001/10000000/01100001/11000101/00000010/01001011/01010001/11110001/11111000/00010110/01011000/10000001.
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 a1-KQ±~u.

Now, since while encrypting in this case, the source stream is decomposed into sub-streams. After converting the ciphertext C into a stream of bits, the technique

4. Chi-square Calculation:

Through the chi square test performed between the original and the encrypted files, the non-homogeneity of the two files is tested. The "Pearsonian Chi-square test" or the "Goodness-of-fit Chi-square test" has been performed here to decide whether the observations onto encrypted files are in good agreement with a hypothetical distribution, which means whether the sample of encrypted files may be supposed to have arisen from a specified population. In this case, the chi square distribution is being performed with (2-1) = 1 degree of freedom, 2 being the total number of classes of possible characters in the source as well as in the encrypted files. If the observed value of the statistic exceeds the tabulated value at a given level, the null hypothesis is rejected.

The "Pearsonian Chi-square" or the "Goodness-of-fit Chi-square" is defined as follows:

$$X^2 = \Sigma \{(f0 - fe)^2 / fe\}$$

Here fe and $f0$ respectively stand for the frequencies of '0' and '1' in the source file and that of the same character in the corresponding encrypted file. On the basis of this formula, the Chi-square values have been calculated for sample pairs of source and encrypted files.

5. Results

Section 5.1.1 shows results of the encryption/decryption time, the number of operations for encryption and decryption, and the chi square value, degree of freedom.

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 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 5.1.1 shows the result on .EXE files, section 5.1.2 shows the result on .CPP files, section 5.1.3 shows the result on .SYS files, section 5.1.4 shows the result on .TXT files and section 5.1.5 shows the result on .DLL files.

5.1.1 Results of .CPP files

Table 5.1.1 gives the result of implementing the technique on CPP files. Ten files have been considered. The block number for each encryption is considered to be 2 with a block size of 8. There sizes range from 1332 bytes to 34048 bytes. The encryption time ranges from 0.029034 seconds to 0.767309 seconds. The decryption time ranges from 0.074776 seconds to 1.743418 seconds. The number of operations during the process of encryption ranges from 200304 to 4875120, whereas the same during the process of decryption ranges from 300456 to 7312680. The Chi Square value ranges from 4208 to 106410 and the degree of freedom ranges from 48 to 95.

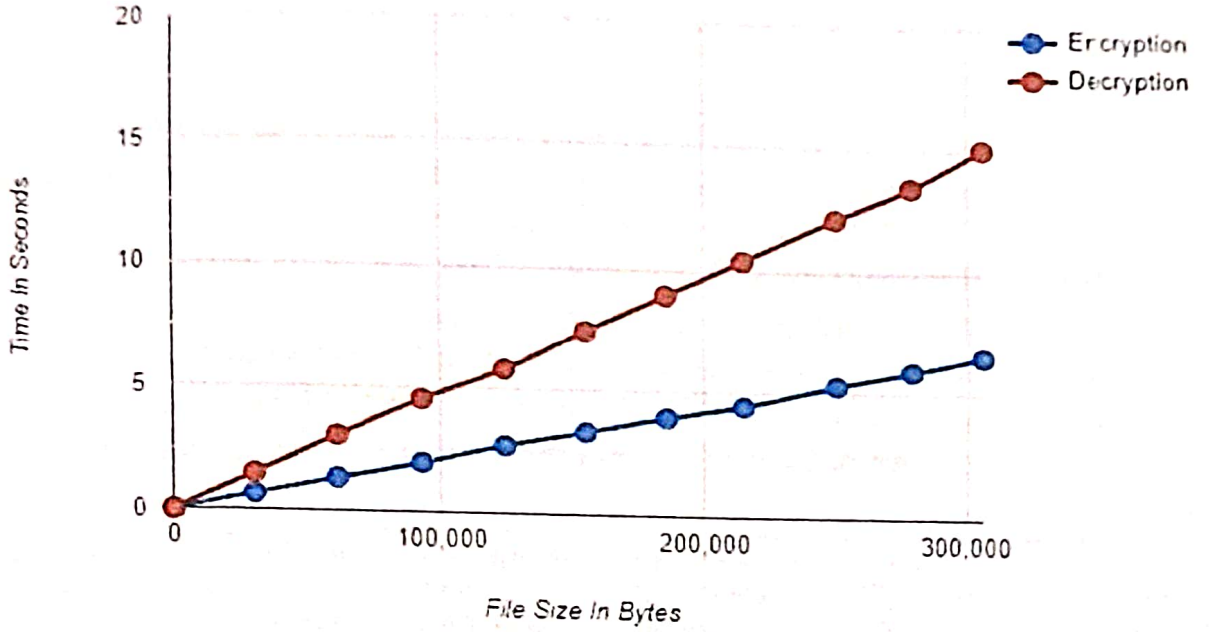
ce File	Source Size (In Bytes)	Encryption Time (In seconds)	Decryption Time (In seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
ut.cpp	412	0.009989	0.021975	56736	85104	1265	55
ut2.cpp	823	0.017982	0.040018	113904	170856	2536	68
ut3.cpp	1342	0.029969	0.062961	186624	279936	4159	73
ut4.cpp	1669	0.038993	0.079636	233424	350136	5287	81
ut5.cpp	2119	0.047971	0.097949	296352	444528	6895	84
ut6.cpp	2480	0.057038	0.112948	346320	519480	7934	80
ut7.cpp	2896	0.064951	0.128897	404352	606528	9200	84
ut8.cpp	3316	0.071593	0.149088	465408	698112	10359	91
ut9.cpp	3714	0.080018	0.171940	522144	783216	11939	88
ut10.cpp	4186	0.087928	0.196850	587088	880632	13002	91

5.1.2 Results of .SYS files

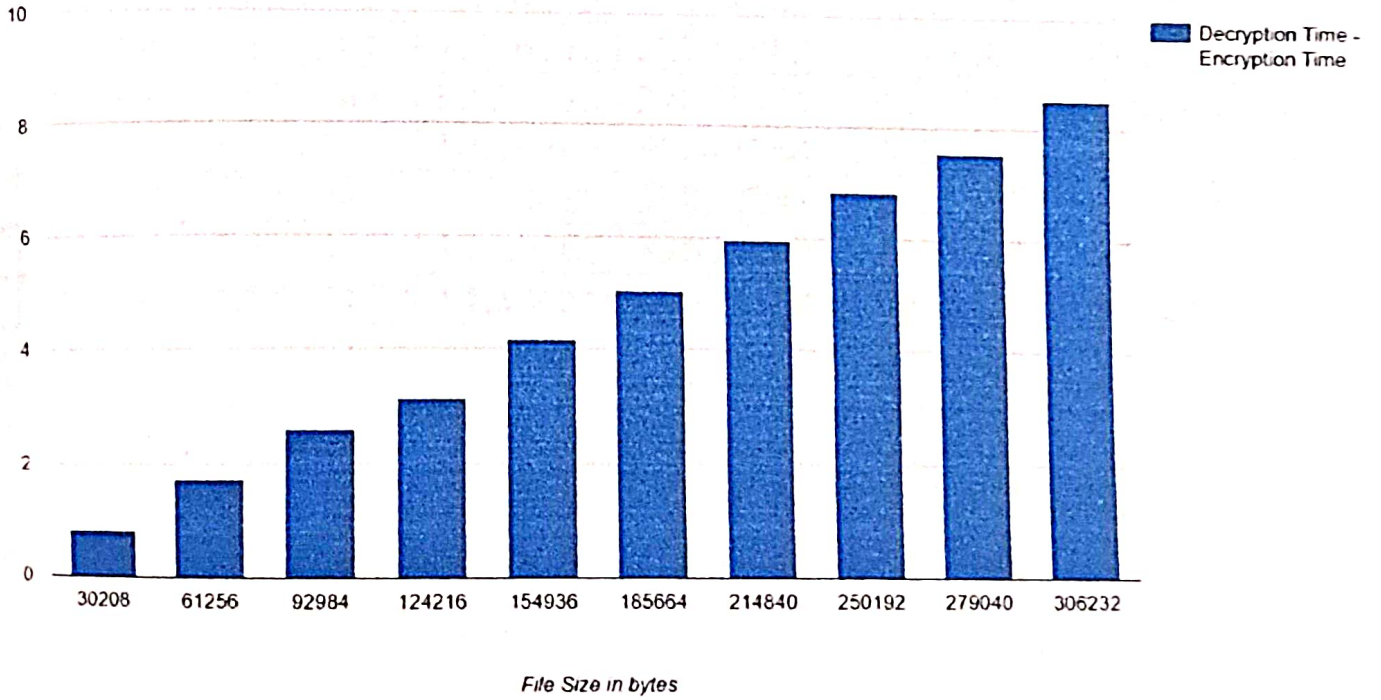
Table 5.1.2 gives the result of implementing the technique on SYS files. Ten files have been considered. The block number for each encryption is considered to be 2 where block size is 8. Their sizes range from 30208 bytes to 306232 bytes. The encryption time ranges 0.646015 seconds to 6.516922 seconds. The decryption time ranges from 1.467025seconds to 2.5359 seconds. The number of operations during the process of encryption ranges from 4349808 to 44096832, whereas the same during the process of decryption ranges from 6524712 to 66145248. The Chi Square value ranges from 78253 to 743811 and the degree of freedom has no change and remains 255.

rce File	Source Size (In Bytes)	Encryption Time (In seconds)	Decryption Time (In seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
ut.sys	30208	0.646015	1.467025	4349808	6524712	78253	255
ut2.sys	61256	1.301642	3.021426	8820720	13231080	155473	255
ut3.sys	92984	1.952574	4.538795	13389264	20083896	234060	255
ut4.sys	124216	2.695007	5.811565	17886960	26830440	318349	255
ut5.sys	154936	3.29349	7.44377	22310640	33465960	398795	255
ut6.sys	185664	3.929058	8.960408	26735328	40102992	485854	255
ut7.sys	214840	4.447594	10.40030	30936384	46404576	542006	255
ut8.sys	250192	5.304805	12.13168	36027360	54041040	644017	255
ut9.sys	279040	5.901439	13.45305	40181328	60271992	740401	255
ut10.sys	306232	6.516922	15.05040	44096832	66145248	743811	255

Relationship between File Size, Encryption and Decryption Time for .SYS Files in RPPO Technique



Difference Of Time Between Decryption And Encryption Time Of .SYS Files

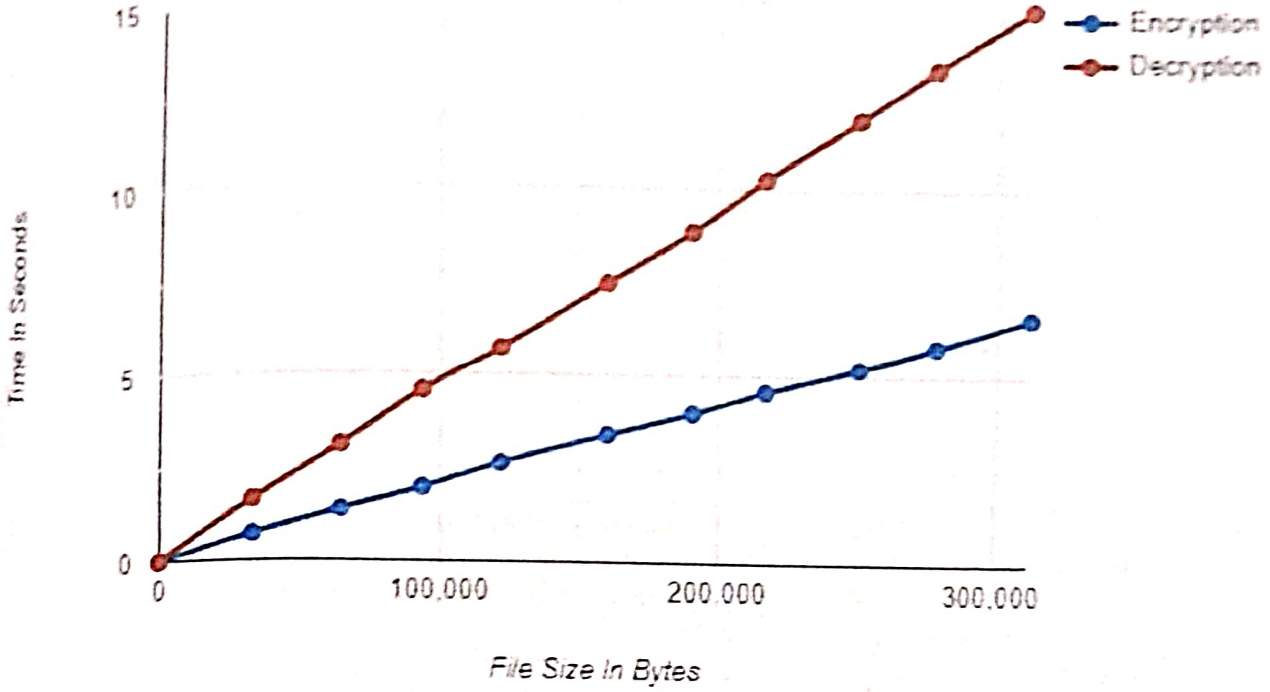


5.1.3 Results of .TXT files

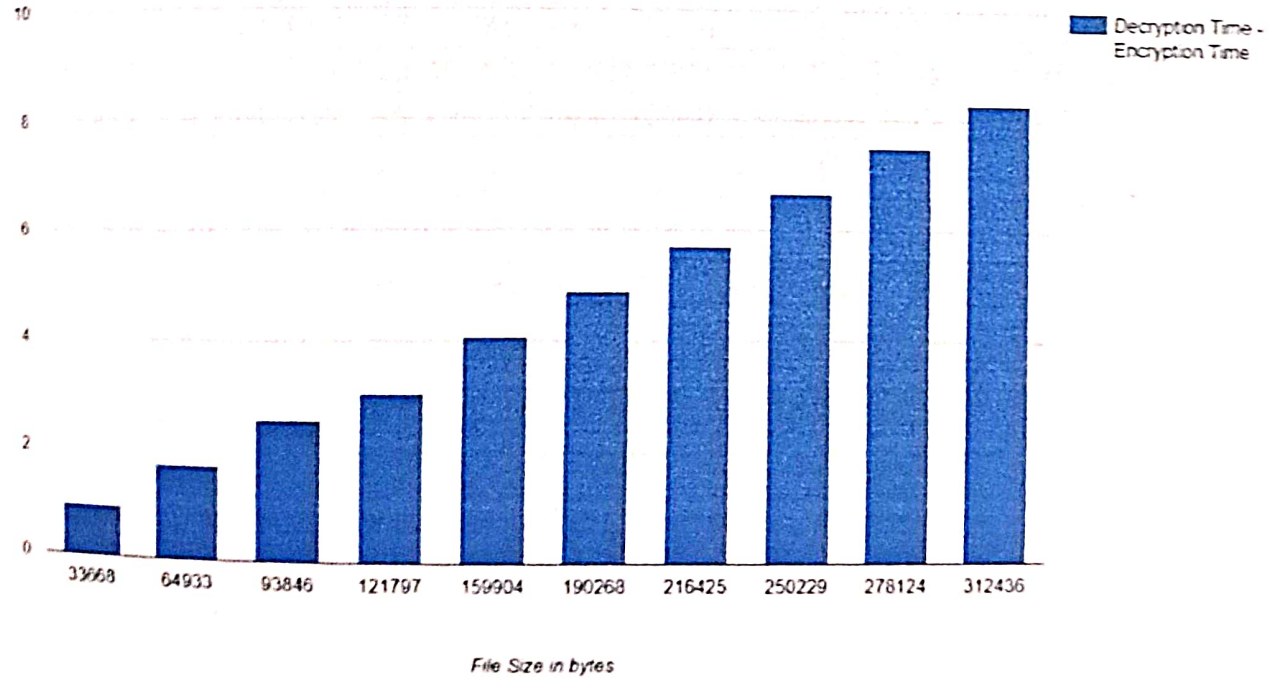
Table 5.1.3 gives the result of implementing the technique on TXT files. Ten files have been considered. The block number for each encryption is considered to be 2 where block size is 8. Their sizes range from 33668 bytes to 312436 bytes. The encryption time ranges 0.719773 seconds to 6.586717 seconds. The decryption time ranges from 1.648159 seconds to 14.88655 seconds. The number of operations during the process of encryption ranges from 4789152 to 44503488, whereas the same during the process of decryption ranges from 7183728 to 66755232. The Chi Square value ranges from 98341 to 936287 and the degree of freedom ranges from 71 to 88.

ce File	Source Size (In Bytes)	Encryption Time (In seconds)	Decryption Time (In seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
ut.txt	33668	0.719773	1.648159	4789152	7183728	98341	88
ut2.txt	64933	1.374881	3.094640	9242496	13863744	189645	71
ut3.txt	93846	1.975489	4.548501	13351104	20026656	271734	80
ut4.txt	121797	2.639702	5.704205	17344368	26016552	353759	83
ut5.txt	159904	3.399581	7.476212	22773888	34160832	464859	85
ut6.txt	190268	3.983387	8.857363	27071424	40607136	559598	81
ut7.txt	216425	4.571773	10.26466	30842784	46264176	634689	86
ut8.txt	250229	5.215048	11.88080	34812288	52218432	730730	81
ut9.txt	278124	5.802322	13.29207	39698208	59547312	824974	73
ut10.txt	312436	6.586717	14.88655	44503488	66755232	936287	71

Relationship between File Size, Encryption and Decryption Time for .TXT Files in RPPO Technique



Difference Of Time Between Decryption And Encryption Time Of .TXT Files



5.1.4 Results of .DLL files

Table 5.1.4 gives the result of implementing the technique on DLL files. Ten files have been considered. The block number for each encryption is considered to be 2 where the block size is 8. Their sizes range from 31744 bytes to 310991 bytes. The encryption time ranges 0.640250 seconds to 6.8782029 seconds. The decryption time ranges from 1.512897 seconds to 15.168320 seconds. The number of operations during the process of encryption ranges from 4570848 to 44162640, whereas the same during the process of decryption ranges from 6856272 to 66243960. The Chi Square value ranges from 81096 to 833057 and the degree of freedom remains 255 for all values.

ree File	Source Size (In Bytes)	Encryption Time (In seconds)	Decryption Time (In seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
it.dll	31744	0.640250	1.512897	4570848	6856272	81096	255
it2.dll	68608	1.412551	3.395116	9873648	14810472	174120	255
it3.dll	92160	1.995268	4.536923	13270752	19906128	232336	255
it4.dll	128312	2.6391823	6.2862	18476784	27715176	297415	255
it5.dll	153088	3.3796854	7.567380	22044240	33066360	407508	255
it6.dll	184320	4.022797	9.098409	26541936	39812904	461264	255
it7.dll	221184	4.672116	10.687936	31849920	47774880	581302	255
it8.dll	251192	5.5322067	12.428851	36171216	54256824	630248	255
it9.dll	274944	6.1317281	13.463537	39568896	59353344	728438	255
it10.dll	310991	6.8782029	15.168320	44162640	66243960	833057	255

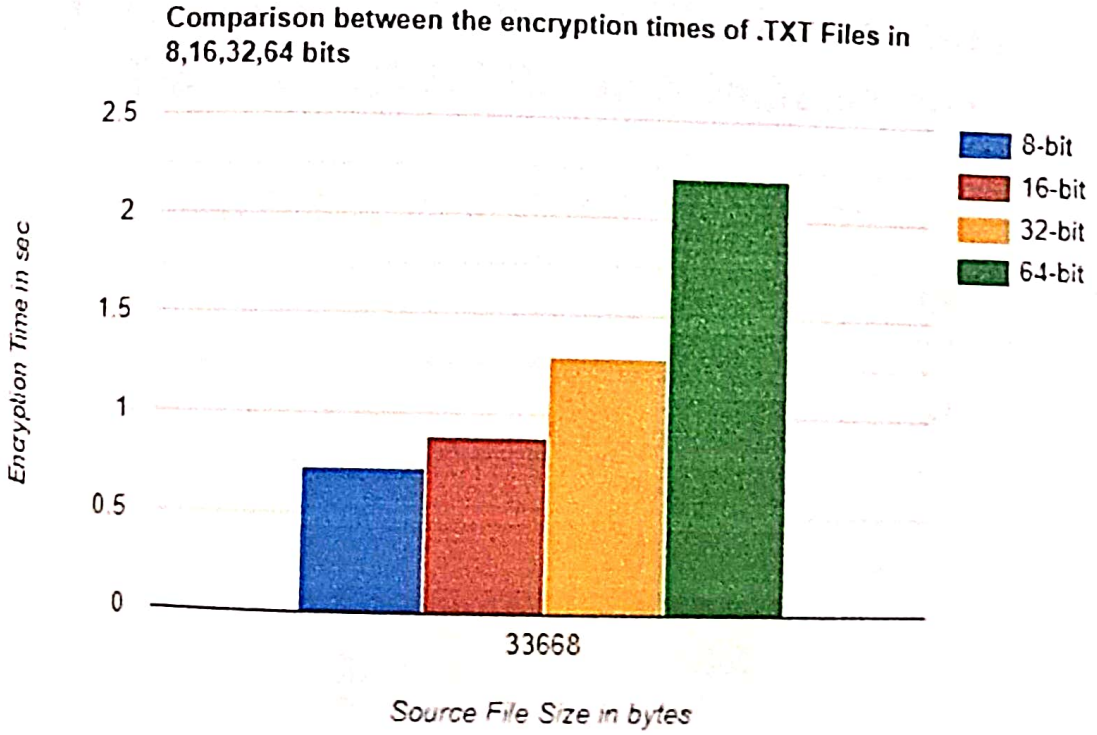
6. Comparison Of Encryption And Decryption Time With Respect To Block Size(Bits)

This section entails the detailed study of the change of encryption and decryption time with variable block size (8-bit, 16-bit, 32-bit and 64-bit). The comparison is done on .TXT, .DLL, .CPP, .SYS files.

For comparing encryption times the block number being taken is 2 as the blocks will go through 2 iterations.

For comparing decryption times the block number is 2 for 8-bit,10 for 16-bit,26 for 32-bit and 58 for 64-bit. All of these have 6 iterations remaining for decryption which makes it easier to compare them.

6.1.1 Comparison Between The Encryption Times Of .TXT Files in 8,16,32,64 bits

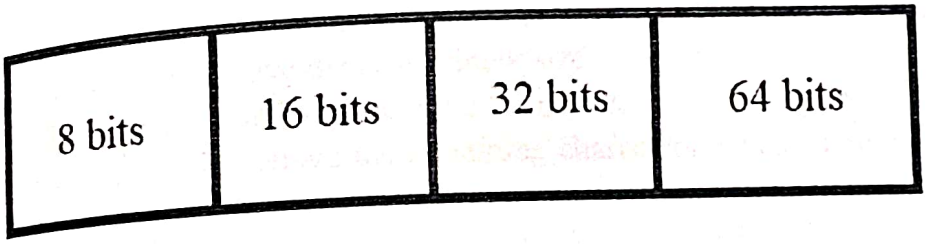


7.RPPO With Session Key

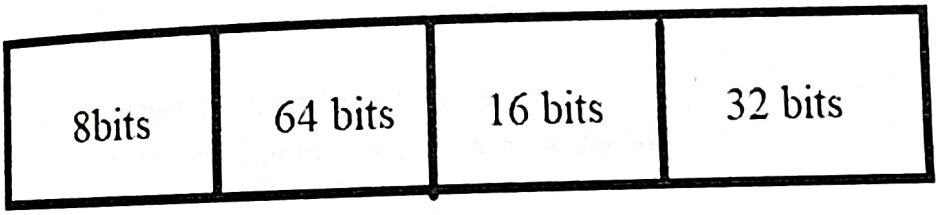
In this section we are implementing the RPPO algorithm with a session key and then passing the code with the previous files. The session key is 120 bits long with 4 segments consisting of 8bits, 16bits, 32bits, 64bits. Details on the making of the session key is given in section 7.1. Then the file is divided according to the session key. Details on the implementation are given in section 7.2. Finally we test the result of the code and the results are declared in section 8.

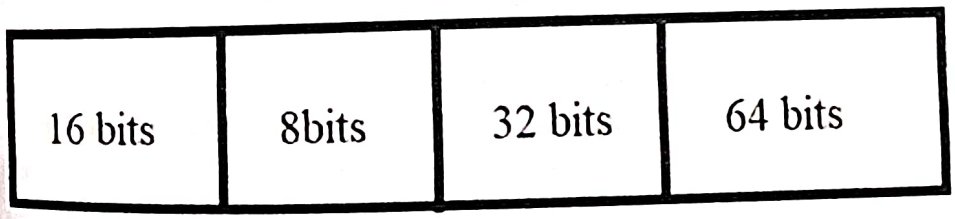
7.1 Generating The Session Key

A 120 bit session key is proposed with 4 segments of size 8 bits, 16 bits, 32 bits, 64 bits.



The segments in the session key will be random every time





According to the formation of the session key we are dividing the input file. The dividing logic is detailed in the next section.

7.2 File Division Logic

The algorithm for dividing a string into four approximately equal parts involves a series of systematic steps to ensure a balanced distribution of the string's content.

Firstly we calculate the total number of characters in the string. Next we divide the total number of characters by 4 to get the basic chunk size. This is the number of characters each part should ideally have. Then we compute the remainder, which represents the number of characters that won't fit evenly into the four parts.

Finally we distribute the remainder characters to the first part to ensure that it can handle the extra characters. This helps in keeping the other parts as equal as possible. So the string is divided into four equal (approximately) parts.

First Part: This part will have the basic chunk size plus all the remainder characters.
Second Part: This part will have the basic chunk size.
Third Part: This part will also have the basic chunk size.
Fourth Part: This part will contain the remaining characters, which should be equal to the basic chunk size.

Example:-Dividing a 33668-byte String

The total length of the string is 33668 characters.

Calculating Basic Chunk Size and Remainder:

- Basic Chunk Size (x): 33668 divided by 4 equals 8417.
- Remainder (y): 33668 modulus 4 equals 0.

Since the remainder is 0, the division is perfectly even, and we do not need to adjust any parts to accommodate extra characters.

Allocating characters to parts:

- First Part: 8417 characters.
- Second Part: 8417 characters.
- Third Part: 8417 characters.
- Fourth Part: 8417 characters.

Example:-Dividing a 33670-byte String

The total length of the string is 33670 characters.

Calculating **Basic Chunk Size and Remainder**:

- Basic Chunk Size (x): 33670 divided by 4 equals 8417.5, which we round up to 8418.
- Remainder (y): 33670 modulus 4 equals 2.

Allocating **Characters to Parts**:

- First Part: Basic chunk size plus the remainder, so 8418 + 2 = 8420 characters.
- Second Part: 8418 characters.
- Third Part: 8418 characters.
- Fourth Part: 8414 characters (since the total is 33670 and the other three parts sum to 8420 + 8418 + 8418 = 25256, thus 33670 – 25256 = 8414).

7.3 Implementation

Each part of the content is converted into binary format. The length of each binary chunk is determined by the shuffled bit lengths. For example, if the bit length is 8, each chunk is one character long; if it's 16, each chunk is two characters long, and so on.

The binary chunks from each part of the content are stored in a new list.This new list holds the binary representation of each content part, corresponding to the shuffled bit lengths.

The binary chunks are then mapped to their respective bit lengths using a dictionary. This dictionary associates each bit length with the corresponding list of binary chunks.

Example of the shuffled list:-

```
Shuffled list: [16, 8, 64, 32]
```

Example of the string version of dictionary:-

```
g version of the dictionary:

'In the hush of dawn's first light,\nW", 8: 'hispers of dreams take flight.\nSi
: 'lent echoes, soft and clear,\nMorn', 32: 'ings bloom, dispelling fear.\n'}
```

The process goes through each key-value pair in the dictionary that maps bit lengths to lists of binary chunks.

After this the implementation is same as the implementation of RPPO algorithm which is mentioned before.

8. Results Of RPPO with Session Key

The RPPO with session key is tested on different file formats with varying file sizes. Results of .txt files are in section 8.1, .sys files are in section 8.2, .cpp files are in section 8.3, .dll files are in section 8.4, and .exe files are in section 8.5.

In this testing the files that were used for testing the RPPO algorithms are used to test RPPO with Session Key. There is no change of files in both testing.

In all the testing the block number of encryption is taken as 3 and the maximum block number that can be taken is 7 as it has a session key with varying individual block sizes according to the session key which is explained before. The 3rd block is considered as the encryption block while decryption depends on the number of iterations left which depends on the individual block size.

While testing this algorithm the encryption time, decryption time varied greatly, chi-square value and degree of freedom sometimes varied greatly with no testing as the session key greatly differs each time. The number of xor operations while encrypting and decrypting are the only values that were similar in every test. So to keep consistent I have tested every file five times and all the results are the average of the five times testing.

8.1 Results on TXT Files

8.1 gives the result of implementing the technique on TXT files. Ten files have considered. The block number for each encryption is considered to be 3.Their sizes from 33668 bytes to 312436 bytes. The encryption time ranges 0.13007592 ds to 1.28433162 seconds. The decryption time ranges from 1.94097546 seconds 96387959 seconds. The number of operations during the process of encryption s from 16495184 to 153291632, whereas the same during the process of decryption s from 176635576 to 1641705256. The Chi Square value ranges from 68893 to 83 and the degree of freedom ranges from 135.6 to 197.6.

me	File Size (In Bytes)	Encryption Time (In Seconds)	Decryption Time (In Seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
xt	33668	0.13007592	1.94097546	16495184	176635576	68893	153
.txt	64933	0.31378016	7.127503414	31838400	340985568	144318	177
.txt	93846	0.4076375	7.1596106	45993136	492592904	207403	197.6
.txt	121797	0.50273466	8.65286956	59743296	639823584	275098	167.6
.txt	159904	0.45289368	3.46574106	78450688	840208064	343335	154.8
6.txt	190268	0.86700928	18.38096452	93247856	998655784	417169	168.6
7.txt	216425	0.86452514	15.16420488	106235472	1137725112	472527	173.8
8.txt	250229	1.28433162	33.96387959	119912128	1284226784	546298	183.8
9.txt	278124	0.98843628	16.48528834	136747312	1464555272	599994	150.2
10.txt	312436	1.26387602	22.04331966	153291632	1641705256	688483	135.6

Note:- Encryption Time, Decryption Time, Chi Square Value and Degree of Freedom are taken as an average of 5.

Comparison between Encryption and Decryption Time of RPPO and RPPO using Session Key

In this section we will compare the encryption time and decryption time of various files using the RPPO algorithm with the same files using RPPO algorithm with Session Key.

In this comparison all files are same with the block number that is used to encrypt and decrypt the files in both methods.

Section 9.1.1 is comparison between Encryption Time of RPPO and RPPO Using Session Key of TXT Files

Section 9.1.2 is comparison between Decryption Time of RPPO and RPPO Using Session Key of TXT Files

Section 9.2.1 is comparison between Encryption Time of RPPO and RPPO Using Session Key of EXE Files

Section 9.2.2 is comparison between Decryption Time of RPPO and RPPO Using Session Key of EXE Files

Section 9.3.1 is comparison between Encryption Time of RPPO and RPPO Using Session Key of CPP Files

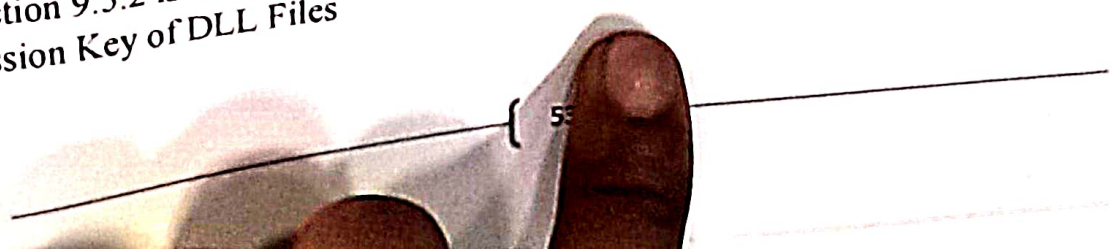
Section 9.3.2 is comparison between Decryption Time of RPPO and RPPO Using Session Key of CPP Files

Section 9.4.1 is comparison between Encryption Time of RPPO and RPPO Using Session Key of SYS Files

Section 9.4.2 is comparison between Decryption Time of RPPO and RPPO Using Session Key of SYS Files

Section 9.5.1 is comparison between Encryption Time of RPPO and RPPO Using Session Key of DLL Files

Section 9.5.2 is comparison between Decryption Time of RPPO and RPPO Using Session Key of DLL Files



10. Conclusion

The RPPO (Recursive Paired Parity Operation) technique presents a novel approach in the realm of cryptographic methods by leveraging Boolean algebra to generate secure encryption.

Key attributes of the RPPO technique include its implementation at the bit level, symmetric nature, use as a block cipher, reliance on substitution techniques, and the fundamental application of Boolean operations. Notably, the RPPO technique maintains the original block size, reinforcing its utility in environments where data size consistency is critical.

This makes RPPO a viable option for secure private key systems and algorithm a vital component in modern encryption practices, providing a reliable solution for protecting critical data.

11. Future Works

RPPO can be used to encrypt text messages, ensuring that communications remain private and protected from unauthorized access

The RPPO (Recursive Paired Parity Operation) algorithm can be further tested on multimedia formats like images, videos and audios.

After testing it we can use the algorithm on multimedia-sharing platforms, RPPO can ensure that shared content remains confidential and is accessible only to authorized users. This is particularly important for platforms dealing with sensitive or proprietary media content.