**Encryption Through Recursive Paired Parity**

**Operation (RPPO)**

To be submitted by

**SOUVIK SAHA**

In Fulfilment 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**

Kalyani-741235

Nadia, West Bengal, India

Phone - +91 - 33 – 5809617

(Direct)+91 - 33 - 25828750/4378

Extn.-304/256/225

Fax:-+91-33-5809617





Department of Computer Science and Engineering, University of Kalyani



This is to certify that the fulfilment of the project report entitled **“Encryption Through Recursive Paired Parity Operation (RPPO) Using Python** ”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**

## Prof. Jyotsna Kumar Mandal

Professor



**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,

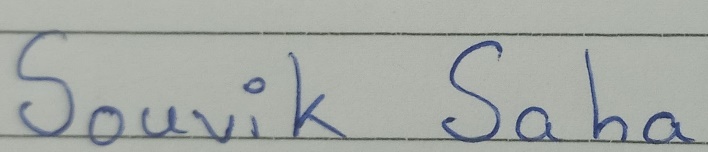
University of Kalyani,

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**

**Signature of Student**

**ACKNOWLEDGEMENT**

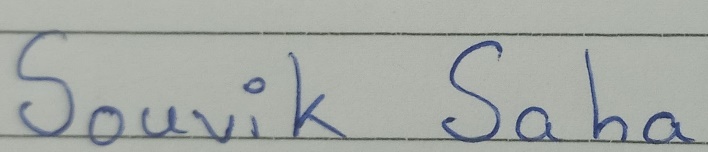
I would like to express my deep sense of gratitude and indebtedness to the many individuals who contributed to the success of my project. Foremost, I am fortunate to have had the guidance and support of my supervisor, Prof. Jyotsna Kumar Mandal. His uncountable advice and encouragement played a pivotal 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)

Content

1. Introduction 8
2. The Scheme 9
   1. Example 10-12
3. Implementation 13-18
4. Chi square Calculation 19
5. Results 20
   1. Results of .CPP Files 21-22
   2. Results of .SYS Files 23-24
   3. Results of .TXT Files 25-26
   4. Results of .DLL Files 27-28
   5. Results of .EXE Files 29-30
6. Comparison of Encryption And Decryption Time with respect to size of the individual blocks 31

6.1.1 Comparison between the Encryption Times of .TXT Files in

8,16,32,64 bits 31

6.1.2 Comparison between the Decryption Times of .TXT Files in

8,16,32,64 bits 32

6.1.3 Comparison between the Encryption Times of .SYS Files in

8,16,32,64 bits 32

6.1.4 Comparison between the Decryption Times of .SYS Files in

8,16,32,64 bits 33

6.1.5 Comparison between the Encryption Times of .DLL Files in

8,16,32,64 bits 33

6.1.6 Comparison between the Decryption Times of .DLL Files in

8,16,32,64 bits 34

6.1.7 Comparison between the Encryption Times of .CPP Files in

8,16,32,64 bits 34

6.1.8 Comparison between the Decryption Times of .CPP Files in

8,16,32,64 bits 35

6.1.9 Comparison between the Encryption Times of .EXE Files in

8,16,32,64 bits 35

6.2.0 Comparison between the Encryption Times of .EXE Files in

8,16,32,64 bits 36

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.

Section 2 of this report discusses the entire scheme of this technique with simple examples. 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 3 of this project report shows a simple implementation of the technique.

Section 4 of this project report is about how the chi square value is calculated.

Section 5 gives the results obtained after implementing the RPPO technique on the same set of real-life files of different categories and different sizes.

Section 6 of the project is a comparison of encryption and decryption time of files of different categories with respect to the size of the individual blocks

1. **The Scheme**

1. Initialization:

1. Define the plaintext as a stream of bits, P = s00 s01 s02 ... s0(n-1), where n is the block size.
2. Set the number of iterations required to regenerate the source block, I.

2. Generating the first intermediate block:

1. For each bit position i in the block (0 ≤ i ≤ n-1):
   1. Calculate si1 = s0(i-1) ⊕ s0i, where ⊕ denotes the exclusive-OR (XOR) operation.
2. The first intermediate block is l1 = s10 s11 s12 ... s1(n-1).

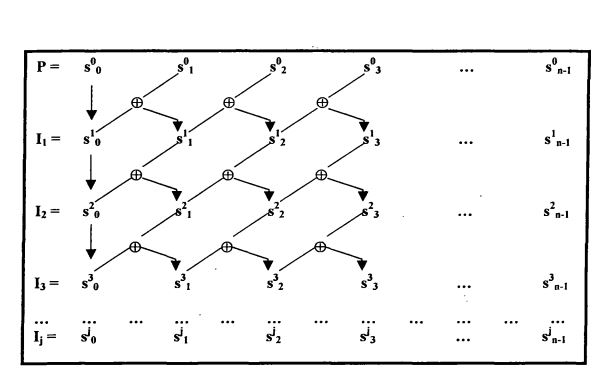
3. Generating subsequent intermediate blocks:

1. For each subsequent intermediate block j (2 ≤ j ≤ I):
   1. For each bit position i in the block (0 ≤ i ≤ n-1):
2. Calculate sij = si(j-1)(i-1) ⊕ si(j-1)i.
   1. 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):

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

Pictorial Representation of RPPO Technique



* 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.

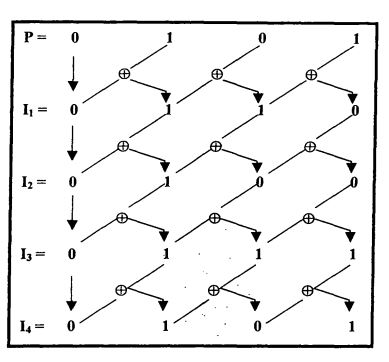


Figure 2.1

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

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. Concerning 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 pieces of 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 B1, B2, 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. [n 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 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.

1. **Implementation**

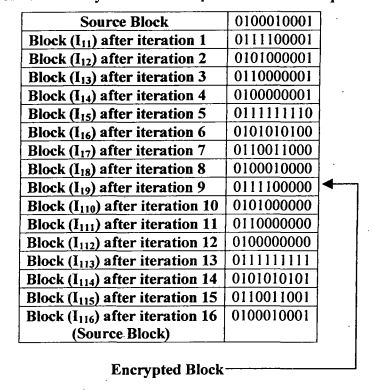
In this section, we explored the application of the Recursive Paired Parity Operation (RPPO) encryption technique on the plaintext "Data Encryption" represented as a bit stream (S) of length 120 bits. Unlike conventional approaches with fixed block sizes, we investigated the behavior of RPPO with blocks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

The chosen block sizes were:

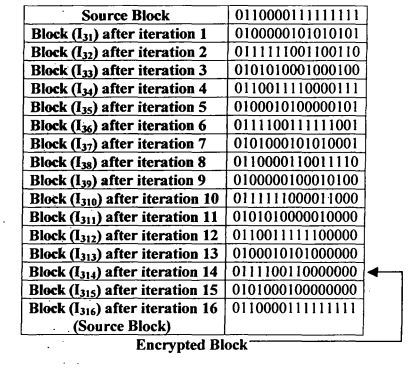
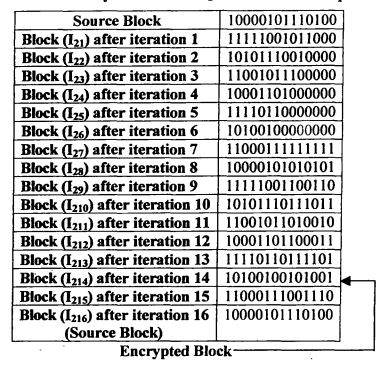
* S1: 0100010001(10 bits)
* S2: 10000101110100(14 bits)
* S3: 0110000111111111(16 bits)
* S4: 01110010(8 bits)
* S5: 010001010110111001100011(24 bits)
* S6: 01111001011100000111010001101001(32 bits)
* S7: 0110111101101110(16 bits)

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)**

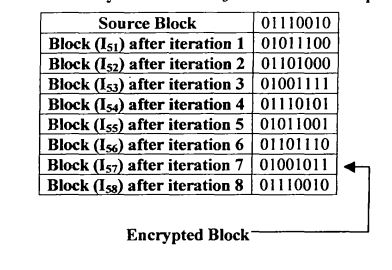


**Table 3.2 for S2:10000101110100(14 bits)**

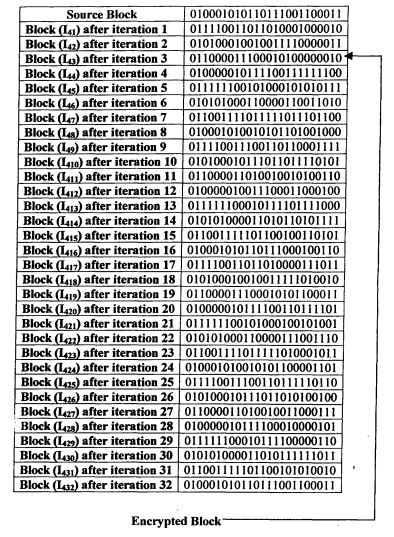


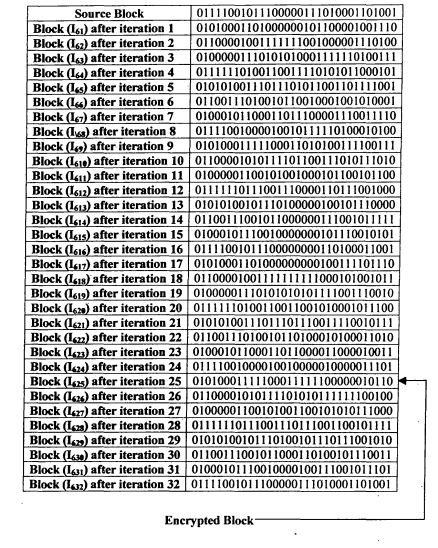
**Table 3.3 for S3: 0110000111111111(16 bits)**

**Table 3.4 for S4: 01110010(8 bits)**

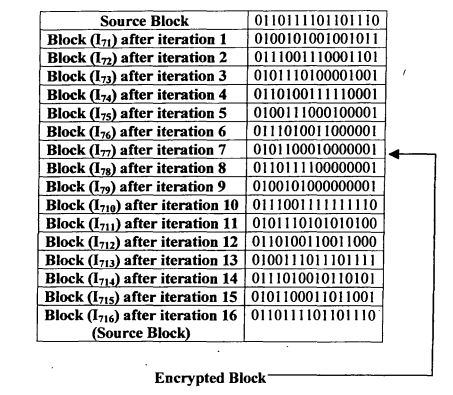


**Table 3.5 for S5:010001010110111001100011(24bits)**



**Table 3.6 for S6: 01111001011100000111010001101001(32 bits)**

**Table 3.7 for S7: 0110111101101110(16 bits)**



As indicated in tables 3.1 to 3.7, intermediate blocks I19 (0111100000), I214 (10100100101001), I314 (0111100110000000), 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: 0111100000/10100100101001/0111100110000000/011000011100010100000010/01001011/01010001111100011111100000010110 /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 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 119, 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 3.1, the encrypted block 119 was obtained after a total of 9 iterations. After obtaining all source blocks in this way, they are grouped to form what would be the source stream of bits, from which the plaintext is achieved.

1. **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:

**Χ2 = Σ {(ƒ 0 – ƒ e)2 / ƒ e}**

Here ƒe and ƒ0 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.

1. **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.

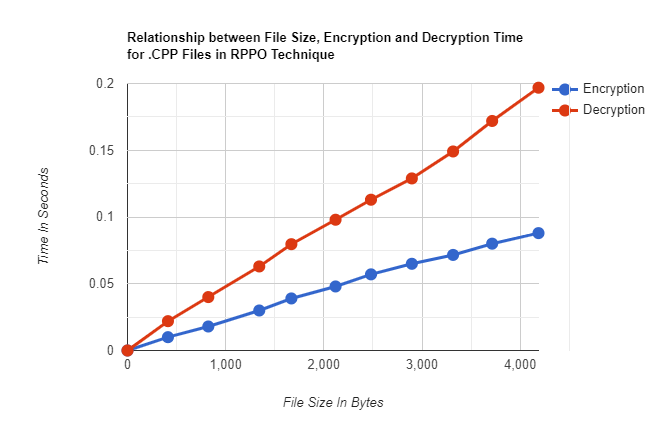
To experiment with the same set of sample files considered earlier, the technique of RPPO has been applied in a cascaded way with block sizes of 2, n increasing from 3 to 8. This means that first on the source file, the RPPO encryption technique is applied for blocks with the unique length of 8 bits. 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].

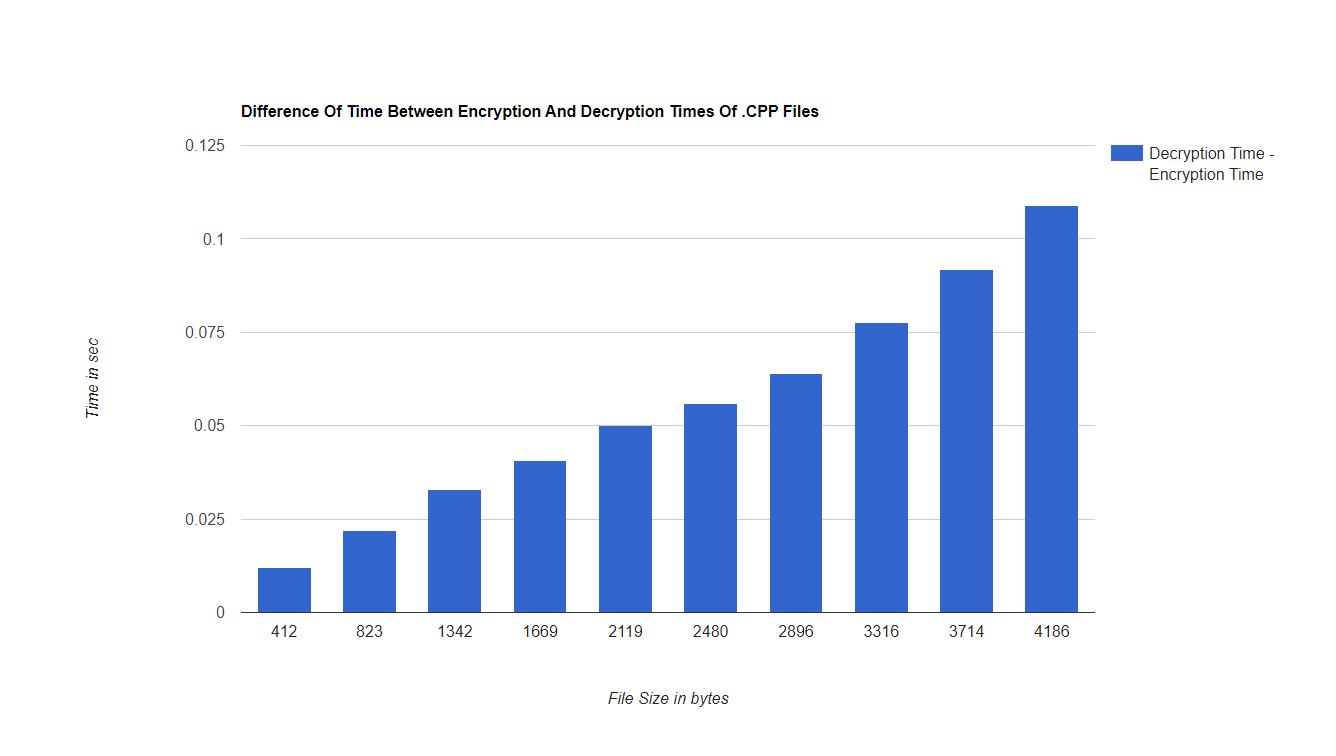
Section 5.1.1 shows the result on .EXE files, section 5.1.2 shows the result on .CPP files, section 5.1.3 shows the result on .SYS files, section 5.1.4 shows the result on .TXT files and section 5.1.5 shows the result on .DLL files.

* + 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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source 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 |
| input.cpp | 412 | 0.009989 | 0.021975 | 56736 | 85104 | 1265 | 55 |
| input2.cpp | 823 | 0.017982 | 0.040018 | 113904 | 170856 | 2536 | 68 |
| input3.cpp | 1342 | 0.029969 | 0.062961 | 186624 | 279936 | 4159 | 73 |
| input4.cpp | 1669 | 0.038993 | 0.079636 | 233424 | 350136 | 5287 | 81 |
| input5.cpp | 2119 | 0.047971 | 0.097949 | 296352 | 444528 | 6895 | 84 |
| input6.cpp | 2480 | 0.057038 | 0.112948 | 346320 | 519480 | 7934 | 80 |
| input7.cpp | 2896 | 0.064951 | 0.128897 | 404352 | 606528 | 9200 | 84 |
| input8.cpp | 3316 | 0.071593 | 0.149088 | 465408 | 698112 | 10359 | 91 |
| input9.cpp | 3714 | 0.080018 | 0.171940 | 522144 | 783216 | 11939 | 88 |
| input10.cpp | 4186 | 0.087928 | 0.196850 | 587088 | 880632 | 13002 | 91 |

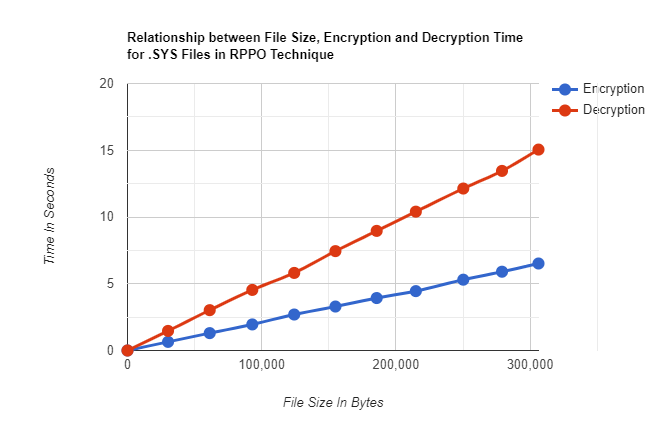


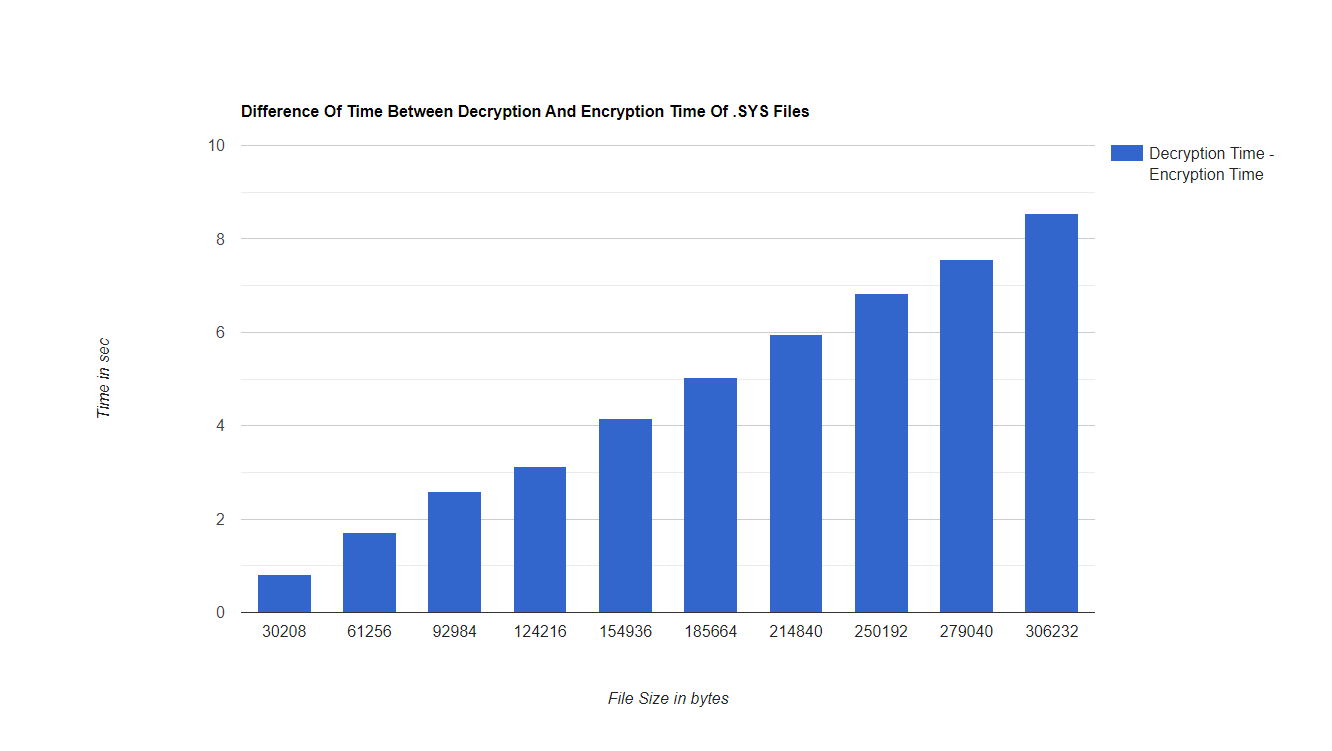


* + 1. **Results of .SYS files**

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source 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 |
| input.sys | 30208 | 0.646015 | 1.467025 | 4349808 | 6524712 | 78253 | 255 |
| input2.sys | 61256 | 1.301642 | 3.021426 | 8820720 | 13231080 | 155473 | 255 |
| input3.sys | 92984 | 1.952574 | 4.538795 | 13389264 | 20083896 | 234060 | 255 |
| input4.sys | 124216 | 2.695007 | 5.811565 | 17886960 | 26830440 | 318349 | 255 |
| input5.sys | 154936 | 3.29349 | 7.44377 | 22310640 | 33465960 | 398795 | 255 |
| input6.sys | 185664 | 3.929058 | 8.960408 | 26735328 | 40102992 | 485854 | 255 |
| input7.sys | 214840 | 4.447594 | 10.40030 | 30936384 | 46404576 | 542006 | 255 |
| input8.sys | 250192 | 5.304805 | 12.13168 | 36027360 | 54041040 | 644017 | 255 |
| input9.sys | 279040 | 5.901439 | 13.45305 | 40181328 | 60271992 | 740401 | 255 |
| input10.sys | 306232 | 6.516922 | 15.05040 | 44096832 | 66145248 | 743811 | 255 |

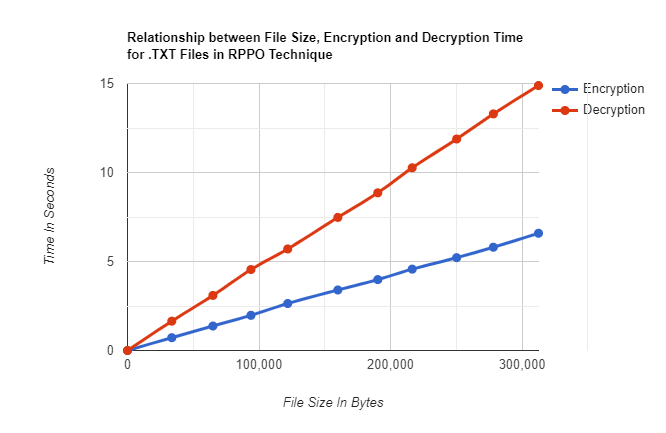


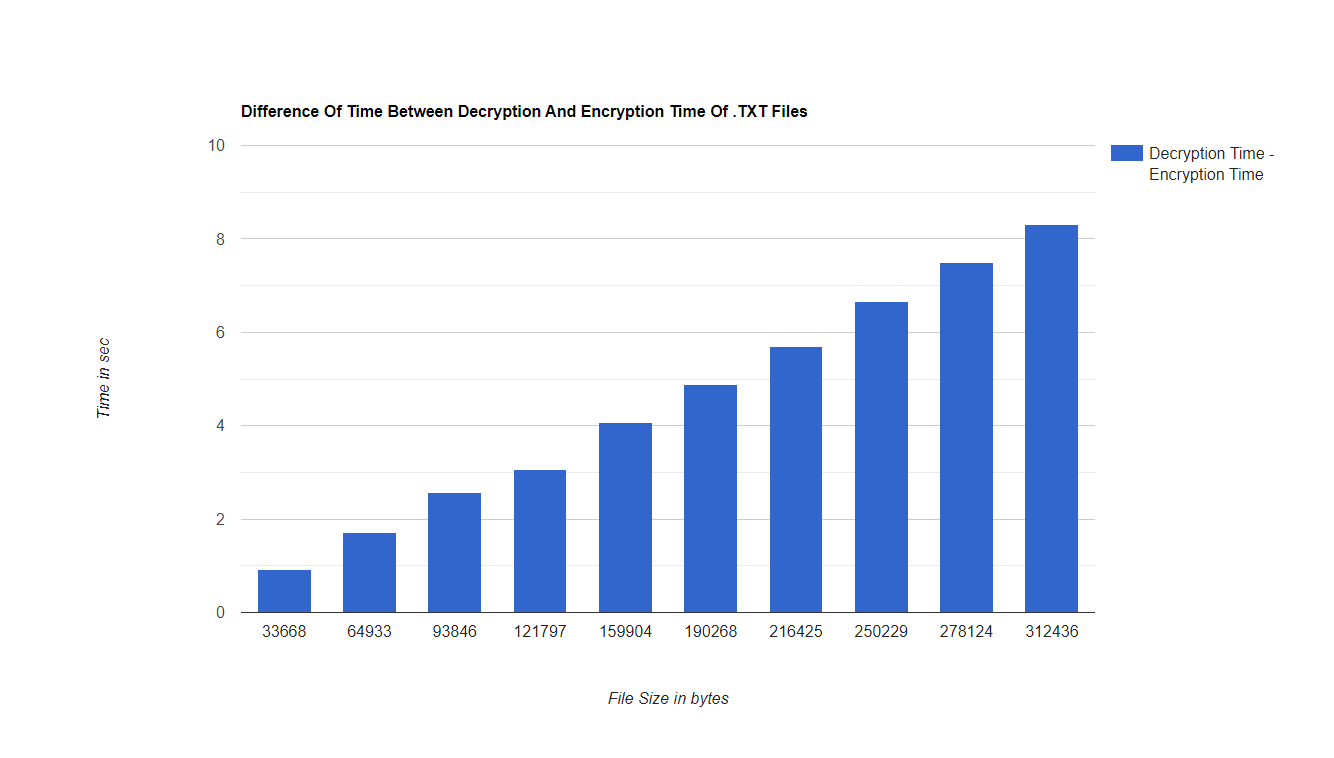


* + 1. **Results of .TXT files**

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source 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 |
| input.txt | 33668 | 0.719773 | 1.648159 | 4789152 | 7183728 | 98341 | 88 |
| input2.txt | 64933 | 1.374881 | 3.094640 | 9242496 | 13863744 | 189645 | 71 |
| input3.txt | 93846 | 1.975489 | 4.548501 | 13351104 | 20026656 | 271734 | 80 |
| input4.txt | 121797 | 2.639702 | 5.704205 | 17344368 | 26016552 | 353759 | 83 |
| input5.txt | 159904 | 3.399581 | 7.476212 | 22773888 | 34160832 | 464859 | 85 |
| input6.txt | 190268 | 3.983387 | 8.857363 | 27071424 | 40607136 | 559598 | 81 |
| input7.txt | 216425 | 4.571773 | 10.26466 | 30842784 | 46264176 | 634689 | 86 |
| input8.txt | 250229 | 5.215048 | 11.88080 | 34812288 | 52218432 | 730730 | 81 |
| input9.txt | 278124 | 5.802322 | 13.29207 | 39698208 | 59547312 | 824974 | 73 |
| input10.txt | 312436 | 6.586717 | 14.88655 | 44503488 | 66755232 | 936287 | 71 |

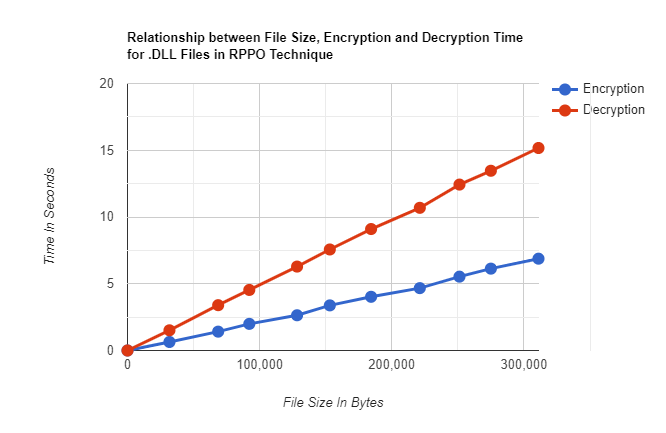


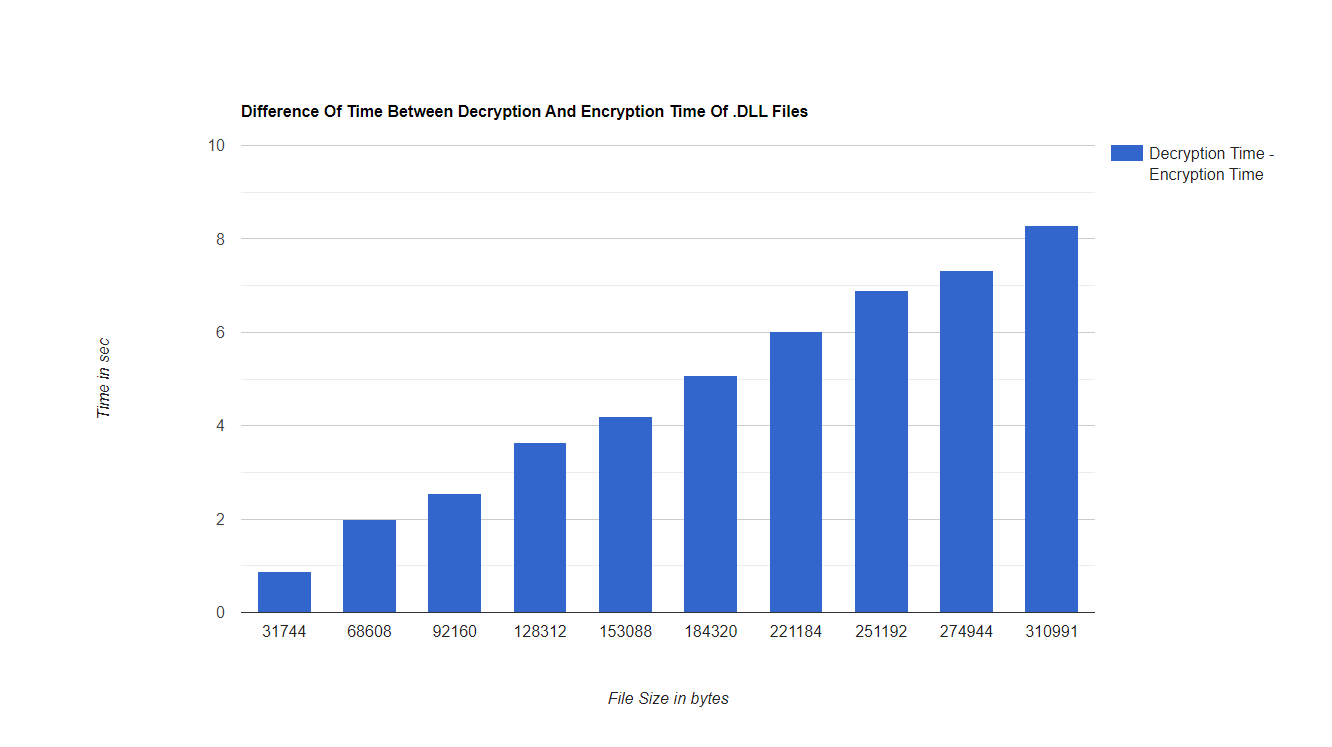


* + 1. **Results of .DLL files**

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source 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 |
| input.dll | 31744 | 0.640250 | 1.512897 | 4570848 | 6856272 | 81096 | 255 |
| input2.dll | 68608 | 1.412551 | 3.395116 | 9873648 | 14810472 | 174120 | 255 |
| input3.dll | 92160 | 1.995268 | 4.536923 | 13270752 | 19906128 | 232336 | 255 |
| input4.dll | 128312 | 2.6391823 | 6.2862 | 18476784 | 27715176 | 297415 | 255 |
| input5.dll | 153088 | 3.3796854 | 7.567380 | 22044240 | 33066360 | 407508 | 255 |
| input6.dll | 184320 | 4.022797 | 9.098409 | 26541936 | 39812904 | 461264 | 255 |
| input7.dll | 221184 | 4.672116 | 10.687936 | 31849920 | 47774880 | 581302 | 255 |
| input8.dll | 251192 | 5.5322067 | 12.428851 | 36171216 | 54256824 | 630248 | 255 |
| input9.dll | 274944 | 6.1317281 | 13.463537 | 39568896 | 59353344 | 728438 | 255 |
| input10.dll | 310991 | 6.8782029 | 15.168320 | 44162640 | 66243960 | 833057 | 255 |

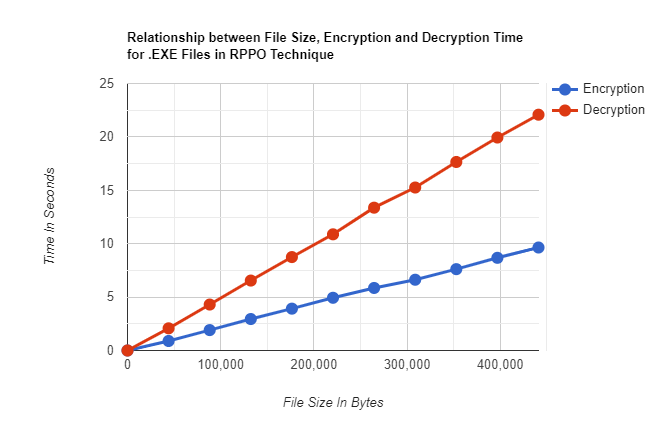
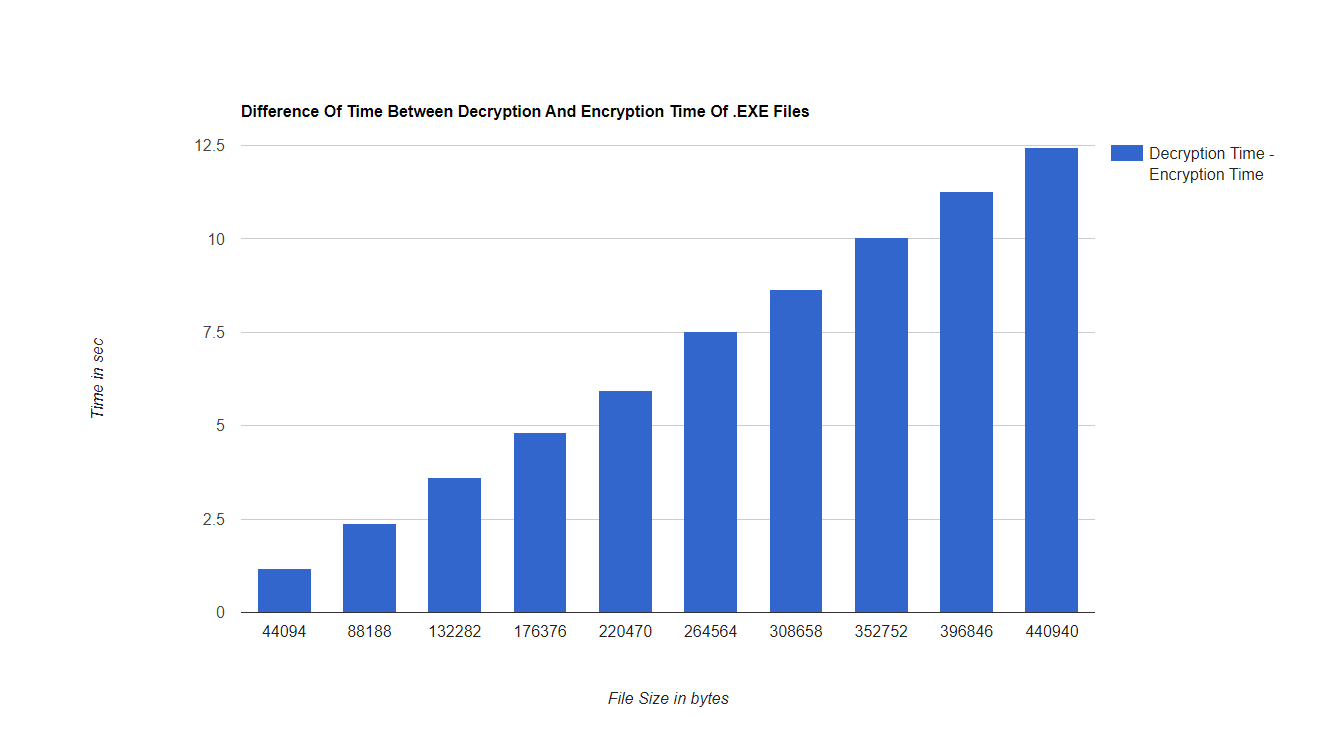




**5.1.5 Results of .EXE files**

Table 5.1.5 gives the result of implementing the technique on EXE 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 44094 bytes to 440940 bytes. The encryption time ranges 0.8904294 seconds to 9.6431651 seconds. The decryption time ranges from 2.0767014 seconds to 22.078723 seconds. The number of operations during the process of encryption ranges from 6349392 to 63495216, whereas the same during the process of decryption ranges from 9524088 to 95242824. The Chi Square value ranges from 108924 to 910429 and the degree of freedom remains 255 for all values.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source 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 |
| input.exe | 44094 | 0.8904294 | 2.0767014 | 6349392 | 9524088 | 108924 | 255 |
| input2.exe | 88188 | 1.9106922 | 4.3034296 | 12698928 | 19048392 | 197804 | 255 |
| input3.exe | 132282 | 2.9460866 | 6.5480928 | 19048464 | 28572696 | 286538 | 255 |
| input4.exe | 176376 | 3.9182028 | 8.7478909 | 25397856 | 38096784 | 375253 | 255 |
| input5.exe | 220470 | 4.9365074 | 10.878361 | 31747536 | 47621304 | 464077 | 255 |
| input6.exe | 264564 | 5.8555295 | 13.373048 | 38097072 | 57145608 | 552330 | 255 |
| input7.exe | 308658 | 6.6259434 | 15.260324 | 44446608 | 66669912 | 643365 | 255 |
| input8.exe | 352752 | 7.6159708 | 17.652042 | 50796144 | 76194216 | 729959 | 255 |
| input9.exe | 396846 | 8.6786994 | 19.941559 | 57145680 | 85718520 | 822091 | 255 |
| input10.exe | 440940 | 9.6431651 | 22.078723 | 63495216 | 95242824 | 910429 | 255 |



1. **Comparison Of Encryption And Decryption Time With**

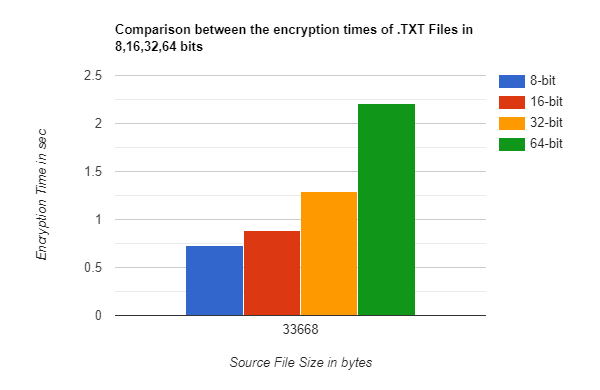
**Respect To Block Size(Bits)**

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.

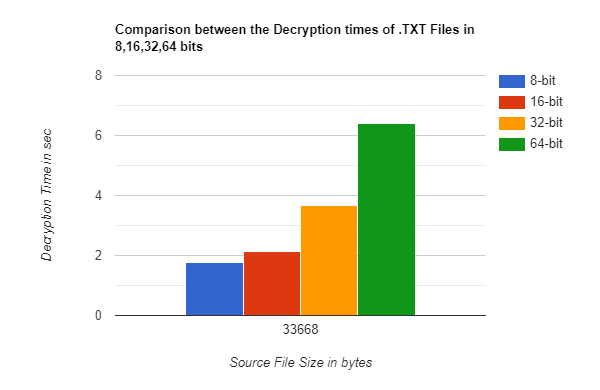
For comparing encryption times the block number being taken is 2 as the blocks will go through 2 iterations.

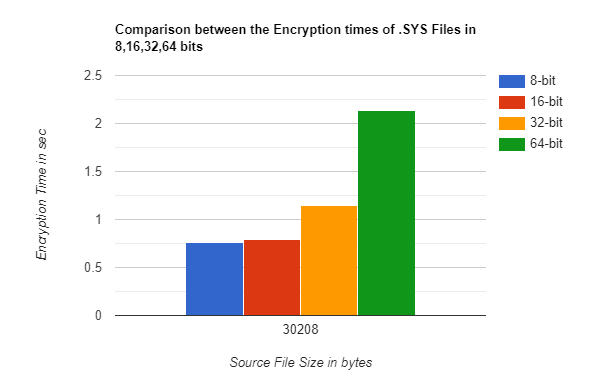
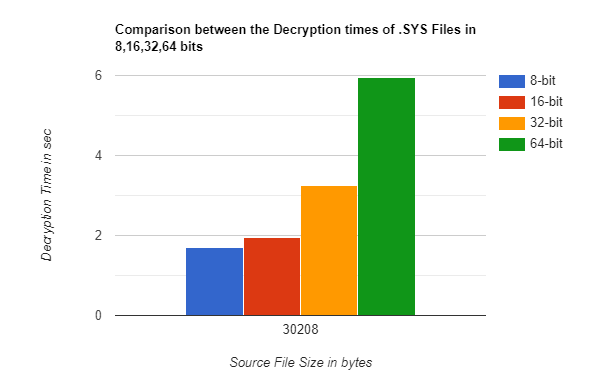
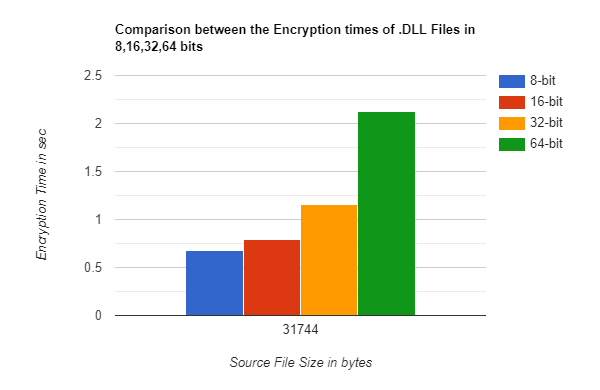
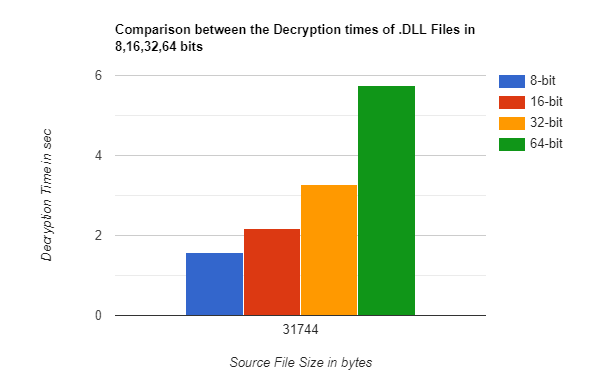
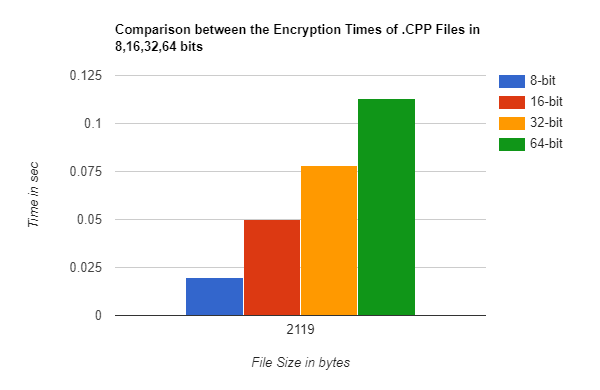
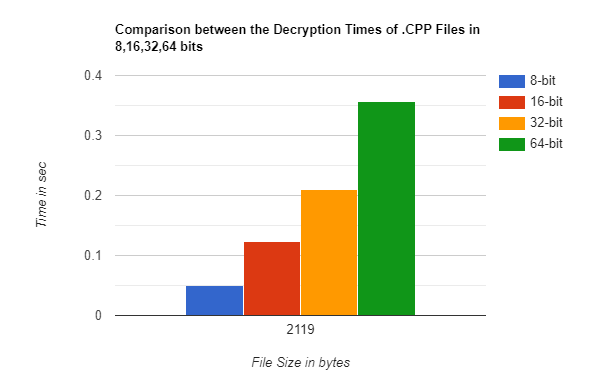
For comparing decryption times the block number is 2 for 8-bit,10 for 16-bit,26 for 32-bit and 58 for 64-bit. All of these have 6 iterations remaining for decryption which makes it easier to compare them.

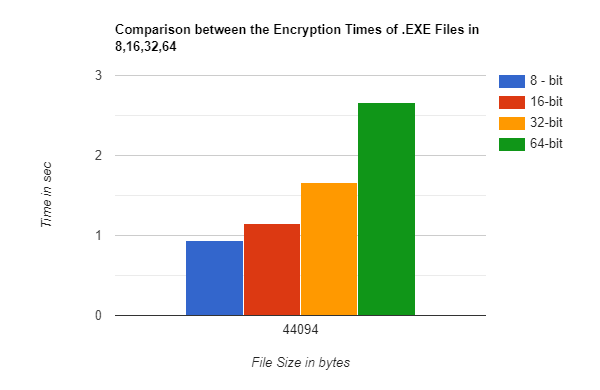
* + 1. **Comparison Between The Encryption Times Of .TXT Files in 8,16,32,64 bits**

****

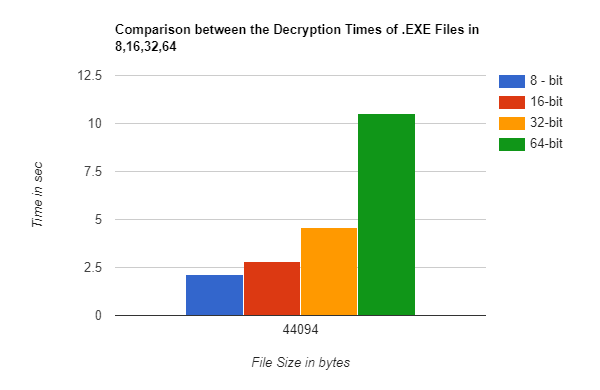
* + 1. **Comparison Between The Decryption Times of .TXT Files in 8,16,32,64 bits**



* + 1. ****Comparison Between The Encryption Times of .SYS Files in 8,16,32,64 bits**
    2. **Comparison Between The Decryption Times of .SYS Files in 8,16,32,64 bits**
    3. **Comparison Between The Encryption Times of .DLL Files in 8,16,32,64 bits**
    4. **Comparison Between The Decryption Times of .DLL Files in 8,16,32,64 bits**
    5. **Comparison Between The Encryption Times of .CPP Files in 8,16,32,64 bits**
    6. **Comparison Between The Decryption Times of .CPP Files in 8,16,32,64 bits**
    7. **Comparison Between The Encryption Times of .EXE Files in 8,16,32,64 bits**

****

* 1. **Comparison Between The Decryption Times of .EXE Files in 8,16,32,64 bits**

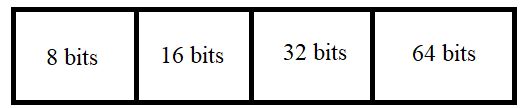
****

1. **RPPO With Session Key**

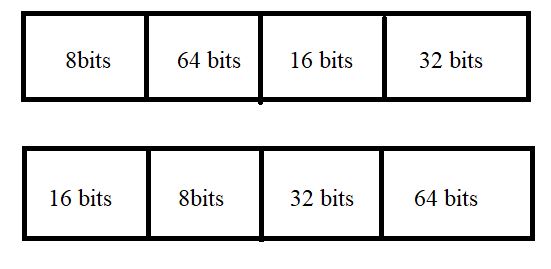
In this section we are implementing the RPPO algorithm with a session key and then testing the code with the previous files. The session key is 120 bits long with 4 segments consisting of 8bits, 16bits, 32bits, 64bits. Details on the making of the session key is given in section 7.1. Then the file is 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.

* 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 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.

* 1. **File Division and Implementation**

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).

1. **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.

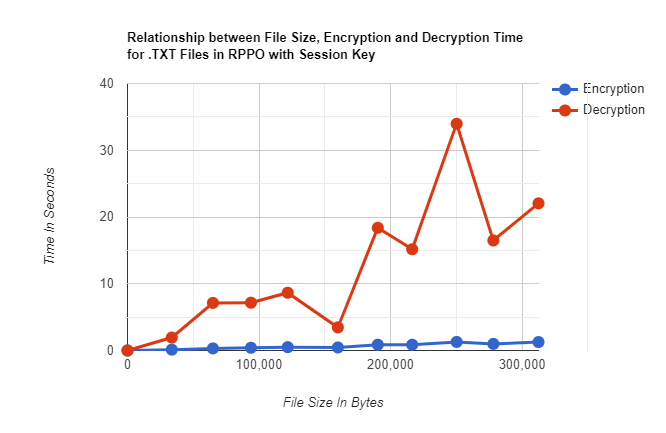
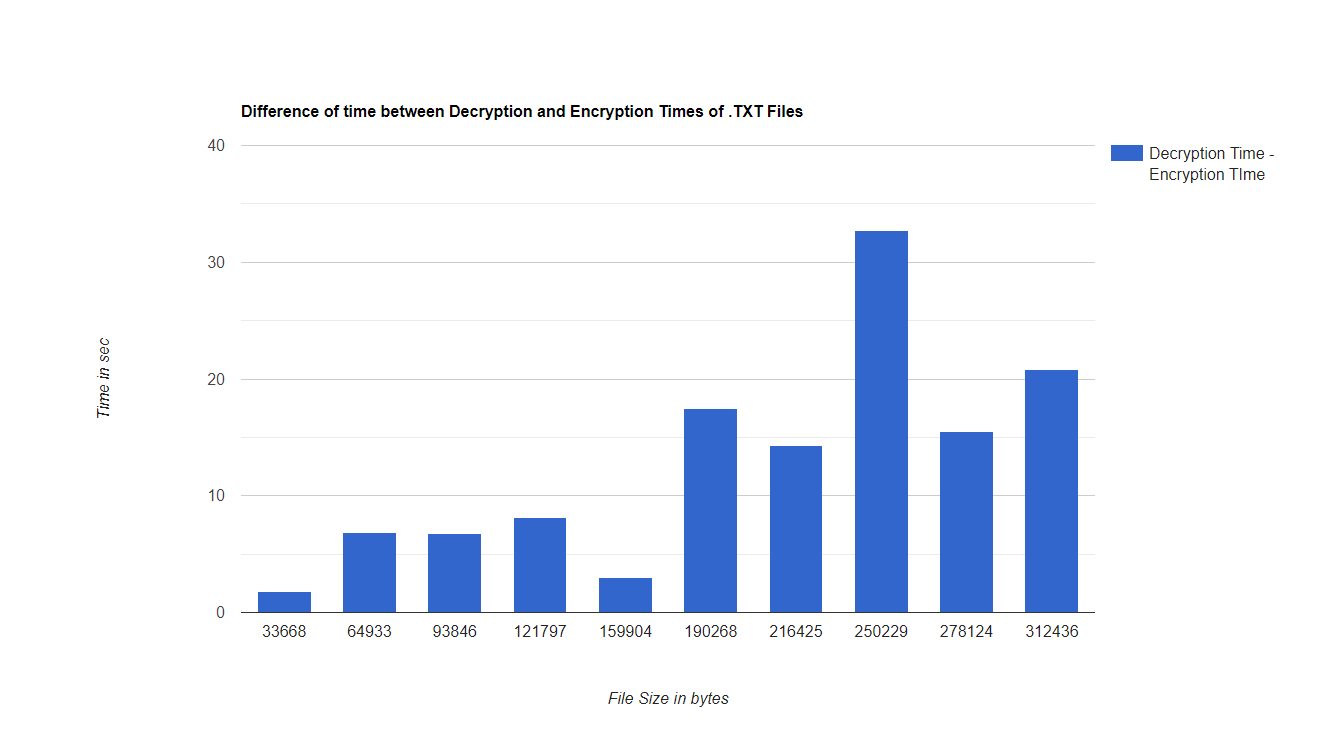
While testing this algorithm the encryption time, decryption time, chi-square value and degree of freedom varied greatly with no testing as the session key greatly differs each time. The number of xor operations while encrypting and decrypting are the only values that were similar in every test. So to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

* 1. **Results on TXT Files**

Table 8.1 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 3.Their sizes range from 33668 bytes to 312436 bytes. The encryption time ranges 0.13007592 seconds to 1.28433162 seconds. The decryption time ranges from 1.94097546 seconds to 33.96387959 seconds. The number of operations during the process of encryption ranges from 16495184 to 153291632, whereas the same during the process of decryption ranges from 176635576 to 1641705256. The Chi Square value ranges from 68893 to 688483 and the degree of freedom ranges from 135.6 to 197.6.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| File Name | 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 |
| input.txt | 33668 | 0.13007592 | 1.94097546 | 16495184 | 176635576 | 68893 | 153 |
| input2.txt | 64933 | 0.31378016 | 7.127503414 | 31838400 | 340985568 | 144318 | 177 |
| input3.txt | 93846 | 0.4076375 | 7.1596106 | 45993136 | 492592904 | 207403 | 197.6 |
| input4.txt | 121797 | 0.50273466 | 8.65286956 | 59743296 | 639823584 | 275098 | 167.6 |
| input5.txt | 159904 | 0.45289368 | 3.46574106 | 78450688 | 840208064 | 343335 | 154.8 |
| input6.txt | 190268 | 0.86700928 | 18.38096452 | 93247856 | 998655784 | 417169 | 168.6 |
| input7.txt | 216425 | 0.86452514 | 15.16420488 | 106235472 | 1137725112 | 472527 | 173.8 |
| input8.txt | 250229 | 1.28433162 | 33.96387959 | 119912128 | 1284226784 |  | 183.8 |
| input9.txt | 278124 | 0.98843628 | 16.48528834 | 136747312 | 1464555272 | 599994 | 150.2 |
| input10.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.



**8.2 Results on EXE Files**

Table 8.2 gives the result of implementing the technique on EXE files. Ten files have been considered. The block number for each encryption is considered to be 2.Their sizes range from 44094 bytes to 440940 bytes. The encryption time ranges 0.13602474 seconds to 2.1052053 seconds. The decryption time ranges from 1.25267992 seconds to 55.16533478 seconds. The number of operations during the process of encryption ranges from 21878848 to 218704352, whereas the same during the process of decryption ranges from 234363104 to 2342230480. The Chi Square value ranges from 85066 to 756011 and the degree of freedom ranges from 255 to 254.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| File Name | 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 |
| input.exe | 44094 | 0.13602474 | 1.25267992 | 21878848 | 234363104 | 85066 | 255 |
| input2.exe | 88188 | 0.29071038 | 4.16863364 | 43750832 | 468603784 | 156512 | 254 |
| input3.exe | 132282 | 0.73132816 | 18.6967529 | 65613760 | 702710048 | 230906 | 254 |
| input4.exe | 176376 | 0.65401944 | 10.1975277 | 87484544 | 936945664 | 305655 | 254 |
| input5.exe | 220470 | 0.8323786‬ | 13.3382738 | 109356160 | 1171186560 | 380644 | 254 |
| input6.exe | 264564 | 1.3424107 | 34.3915550 | 131228720 | 1405425160 | 455494 | 254 |
| input7.exe | 308658 | 1.16646904 | 20.09613762 | 153099248 | 1639659304 | 530824 | 254 |
| input8.exe | 352752 | 1.44276896 | 25.54680214 | 174967040 | 1873867264 | 605644 | 254 |
| input9.exe | 396846 | 2.1052053 | 55.16533478 | 196839680 | 2108117248 | 681066 | 254 |
| input10.exe | 440940 | 1.98222166 | 44.10361305 | 218704352 | 2342230480 | 756011 | 254 |

Note:- Encryption Time, Decryption Time are taken as an average of 5.

**9. Comparison of Encryption and Decryption Time of RPPO with 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 files using RPPO algorithm with Session Key.

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

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

Section 9.2.1 is comparison of Encryption Time of RPPO with RPPO Using Session Key of CPP Files

Section 9.2.2 is comparison of Decryption Time of RPPO with RPPO Using Session Key of CPP Files

Section 9.3.1 is comparison of Encryption Time of RPPO with RPPO Using Session Key of SYS Files

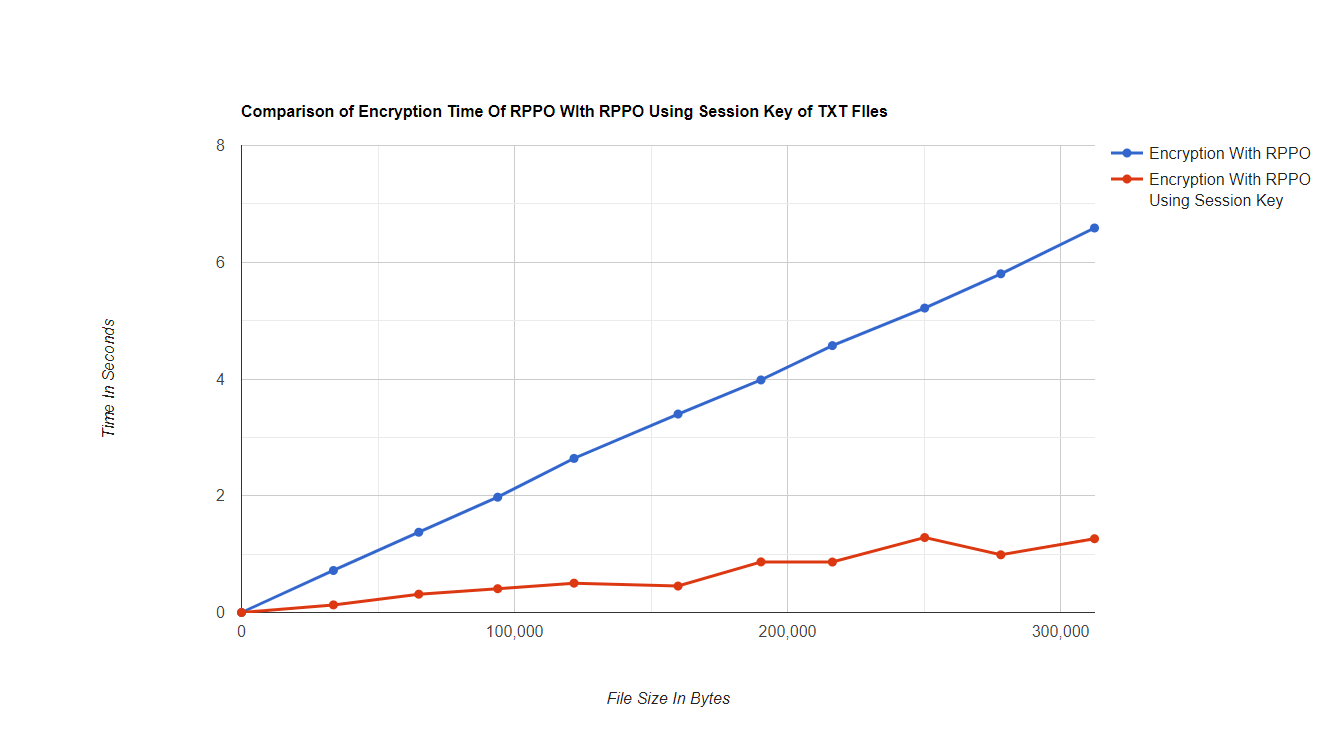
Section 9.3.2 is comparison of Encryption Time of RPPO with RPPO Using Session Key of SYS Files

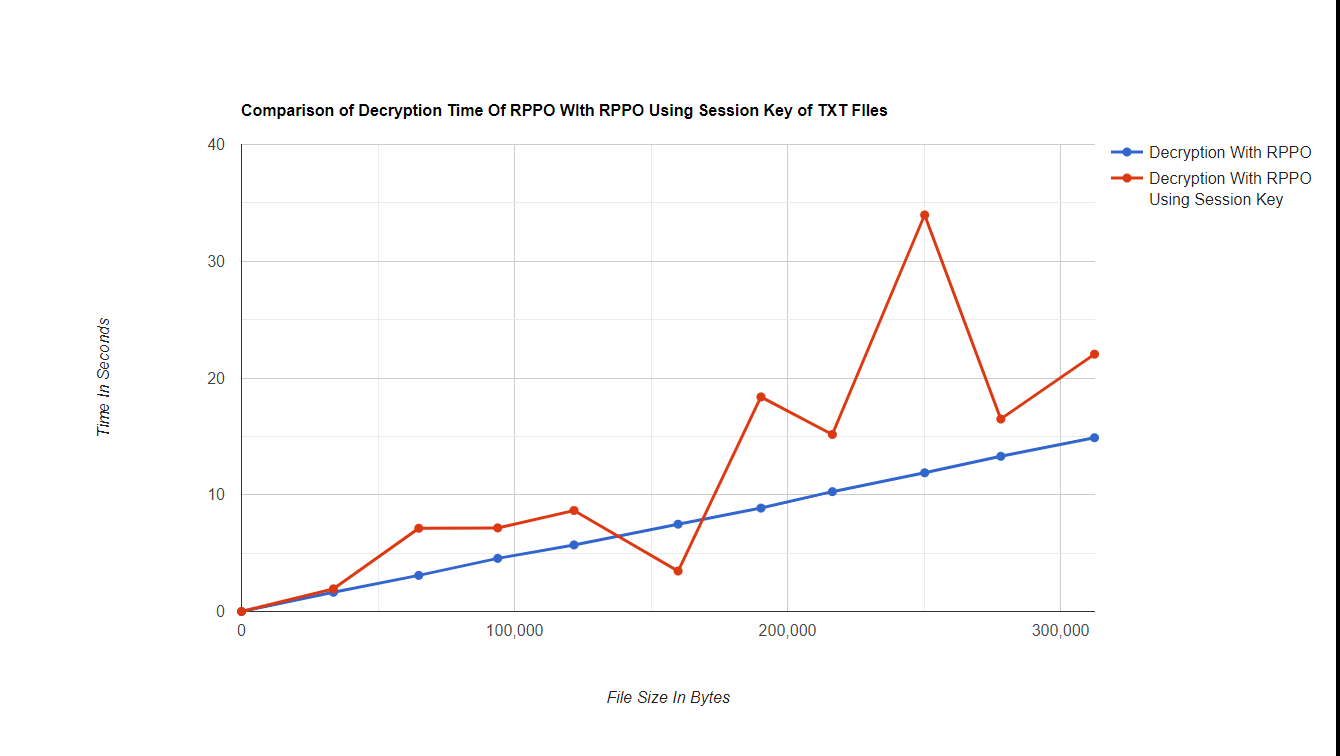
Section 9.4.1 is comparison of Encryption Time of RPPO with RPPO Using Session Key of DLL Files

Section 9.4.2 is comparison of Encryption Time of RPPO with RPPO Using Session Key of DLL Files

Section 9.5.1 is comparison of Encryption Time of RPPO with RPPO Using Session Key of EXE Files

Section 9.5.2 is comparison of Encryption Time of RPPO with RPPO Using Session Key of EXE Files

**9.1.1 Comparison of Encryption Time of RPPO with RPPO Using Session Key of TXT Files**

**9.1.2 Comparison of Decryption Time of RPPO with RPPO Using Session Key of TXT Files**