

---

# 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

---

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 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:** 25/01/2024

---

**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

## **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

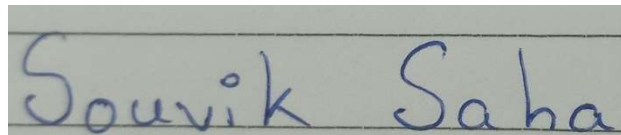
Place of Project: Department of Computer Science and Engineering,  
University of Kalyani,  
Kalyani, Nadia, West Bengal, India.

## **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: 25/01/2024**



---

**Signature of Student**

# ACKNOWLEDGEMENT

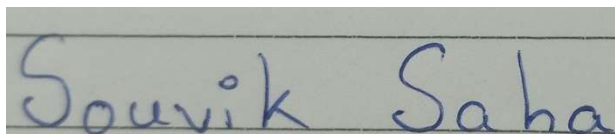
I would like to express my deep sense of gratitude and indebtedness to the many individuals who contributed to the success of my project. Foremost, I am fortunate to have had the guidance and support of my supervisor, Prof. Jyotsna Kumar Mandal. His uncountable advice and encouragement played a pivotal 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
2.1 Example	10-12
3. Implementation	13-18
4. Chi square Calculation	19
5. Results	20
5.1 Results of .CPP Files	21-22
5.2 Results of .SYS Files	23-24
5.3 Results of .TXT Files	25-26
5.4 Results of .DLL Files	27-28
5.5 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 shows a simple implementation of the technique, Section 4 of this project is about how the chi square value is calculated and Section 5 gives the results obtained after implementing the RPPO technique on the same set of real-life files of different categories.



## 2. The Scheme

### 1. Initialization:

- I. Define the plaintext as a stream of bits,  $P = s_{00} s_{01} s_{02} \dots s_{0(n-1)}$ , 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:

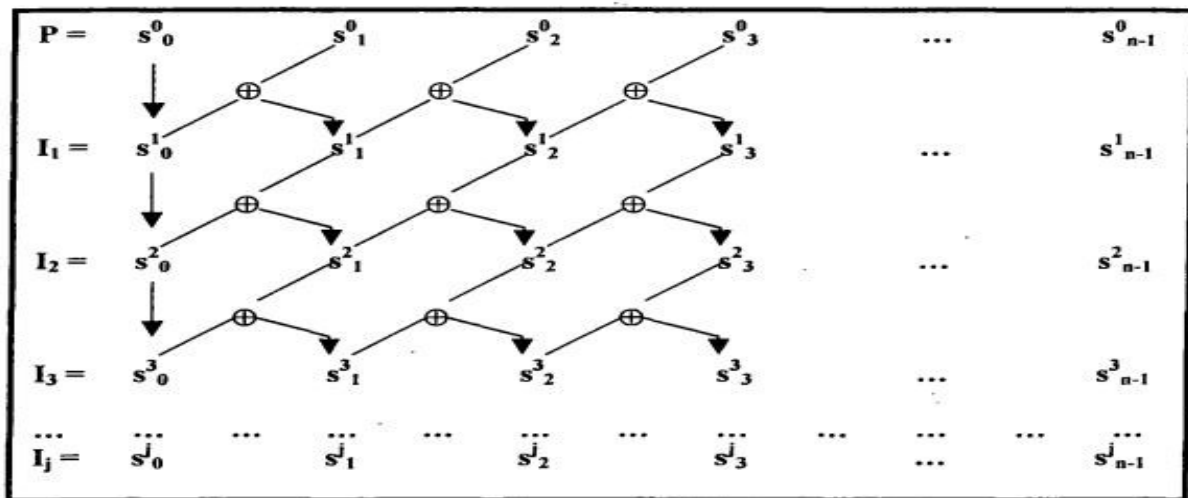
- I. For each bit position  $i$  in the block ( $0 \leq i \leq n-1$ ):
  - a. Calculate  $s_{i1} = s_{0(i-1)} \oplus s_{0i}$ , where  $\oplus$  denotes the exclusive-OR (XOR) operation.
- II. The first intermediate block is  $l_1 = s_{10} s_{11} s_{12} \dots s_{1(n-1)}$ .

### 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  $s_{ij} = s_{i(j-1)(i-1)} \oplus s_{i(j-1)i}$ .
  - b. The  $j$ -th intermediate block is  $l_j = s_{(j)0} s_{(j)1} s_{(j)2} \dots s_{(j)(n-1)}$ .

### 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.

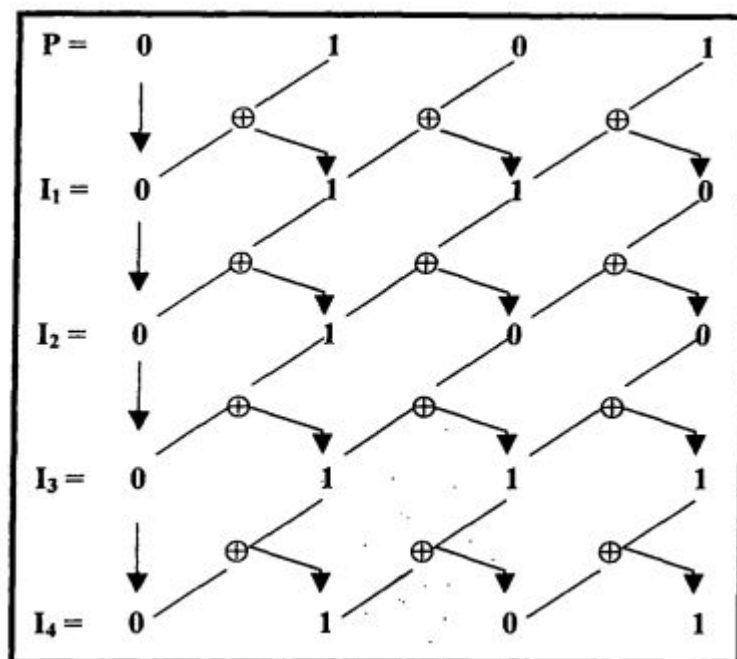


Figure 2.1

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

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  $i$ th 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  $B_1$ ,  $B_2$ ,  $B_3$  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 iterations 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 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.

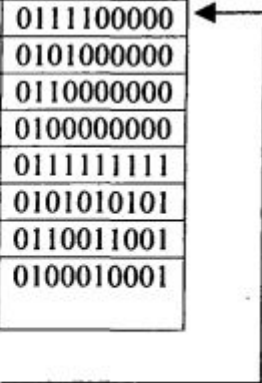
The chosen block sizes were:

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

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

**Encrypted Block** 

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

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

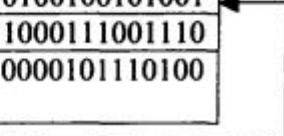
**Encrypted Block** 

Table 3.3 for S3: 0110000111111111(16 bits)

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

Encrypted Block

Table 3.4 for S4: 01110010(8 bits)

<b>Source Block</b>	01110010
<b>Block (I<sub>51</sub>) after iteration 1</b>	01011100
<b>Block (I<sub>52</sub>) after iteration 2</b>	01101000
<b>Block (I<sub>53</sub>) after iteration 3</b>	01001111
<b>Block (I<sub>54</sub>) after iteration 4</b>	01110101
<b>Block (I<sub>55</sub>) after iteration 5</b>	01011001
<b>Block (I<sub>56</sub>) after iteration 6</b>	01101110
<b>Block (I<sub>57</sub>) after iteration 7</b>	01001011
<b>Block (I<sub>58</sub>) after iteration 8</b>	01110010

Encrypted Block

Table 3.5 for S5:010001010110111001100011(24bits)

Source Block	010001010110111001100011
Block (L <sub>41</sub> ) after iteration 1	011110011011010001000010
Block (L <sub>42</sub> ) after iteration 2	010100010010011110000011
Block (L <sub>43</sub> ) after iteration 3	011000011100010100000010
Block (L <sub>44</sub> ) after iteration 4	010000010111100111111100
Block (L <sub>45</sub> ) after iteration 5	011111100101000101010111
Block (L <sub>46</sub> ) after iteration 6	010101000110000110011010
Block (L <sub>47</sub> ) after iteration 7	011001111011111011101100
Block (L <sub>48</sub> ) after iteration 8	010001010010101101001000
Block (L <sub>49</sub> ) after iteration 9	011110011100110110001111
Block (L <sub>410</sub> ) after iteration 10	010100010111011011110101
Block (L <sub>411</sub> ) after iteration 11	011000011010010010100110
Block (L <sub>412</sub> ) after iteration 12	010000010011100011000100
Block (L <sub>413</sub> ) after iteration 13	011111100010111101111000
Block (L <sub>414</sub> ) after iteration 14	010101000011010110101111
Block (L <sub>415</sub> ) after iteration 15	011001111101100100110101
Block (L <sub>416</sub> ) after iteration 16	010001010110111000100110
Block (L <sub>417</sub> ) after iteration 17	011110011011010000111011
Block (L <sub>418</sub> ) after iteration 18	010100010010011111010010
Block (L <sub>419</sub> ) after iteration 19	011000011100010101100011
Block (L <sub>420</sub> ) after iteration 20	010000010111100110111101
Block (L <sub>421</sub> ) after iteration 21	011111100101000100101001
Block (L <sub>422</sub> ) after iteration 22	010101000110000111001110
Block (L <sub>423</sub> ) after iteration 23	011001111011111010001011
Block (L <sub>424</sub> ) after iteration 24	010001010010101100001101
Block (L <sub>425</sub> ) after iteration 25	011110011100110111110110
Block (L <sub>426</sub> ) after iteration 26	010100010111011010100100
Block (L <sub>427</sub> ) after iteration 27	011000011010010011000111
Block (L <sub>428</sub> ) after iteration 28	010000010111100010000101
Block (L <sub>429</sub> ) after iteration 29	011111100010111100000110
Block (L <sub>430</sub> ) after iteration 30	010101000011010111111011
Block (L <sub>431</sub> ) after iteration 31	011001111101100101010010
Block (L <sub>432</sub> ) 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	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 3.7 for S7: 0110111101101110(16 bits)**

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



**Encrypted Block**

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.

## 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:

$$X^2 = \sum \{(f_0 - f_e)^2 / f_e\}$$

Here  $f_e$  and  $f_0$  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, section.

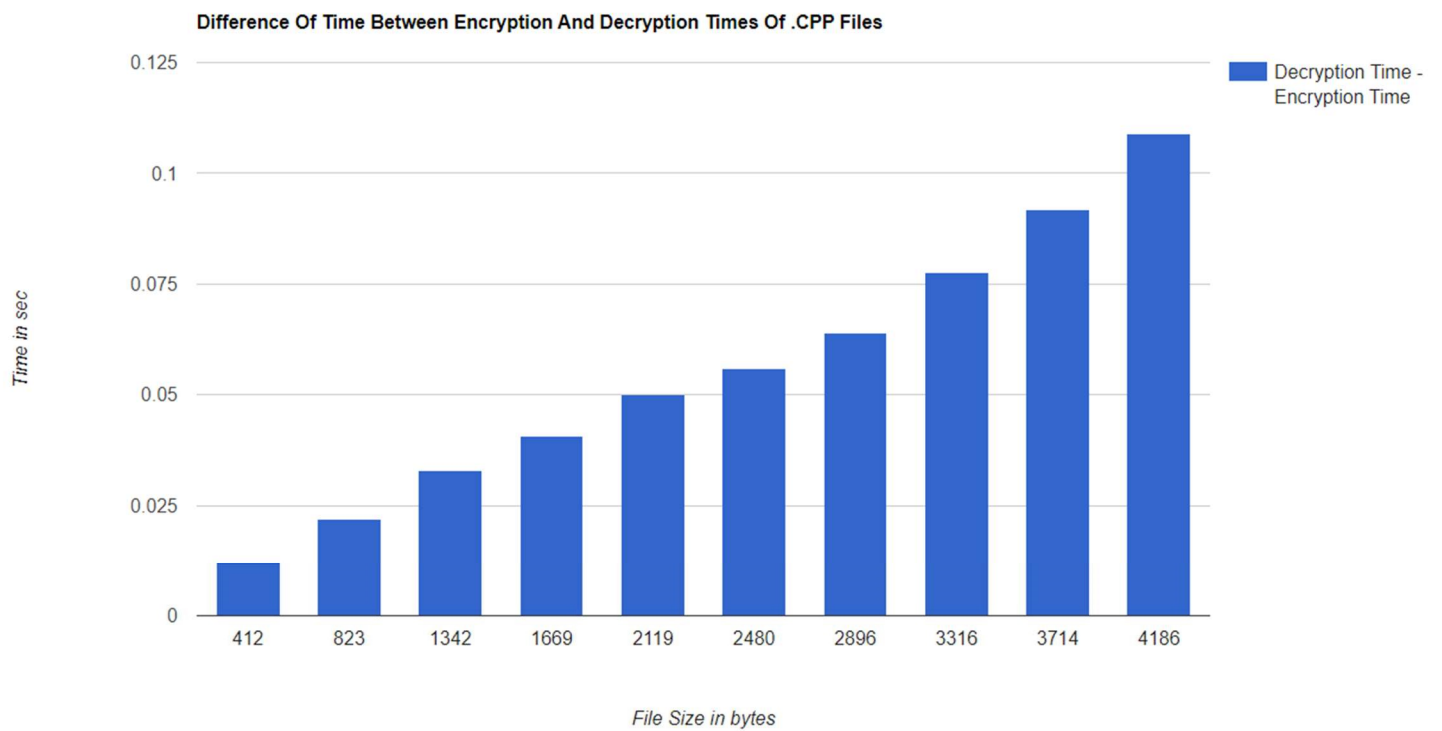
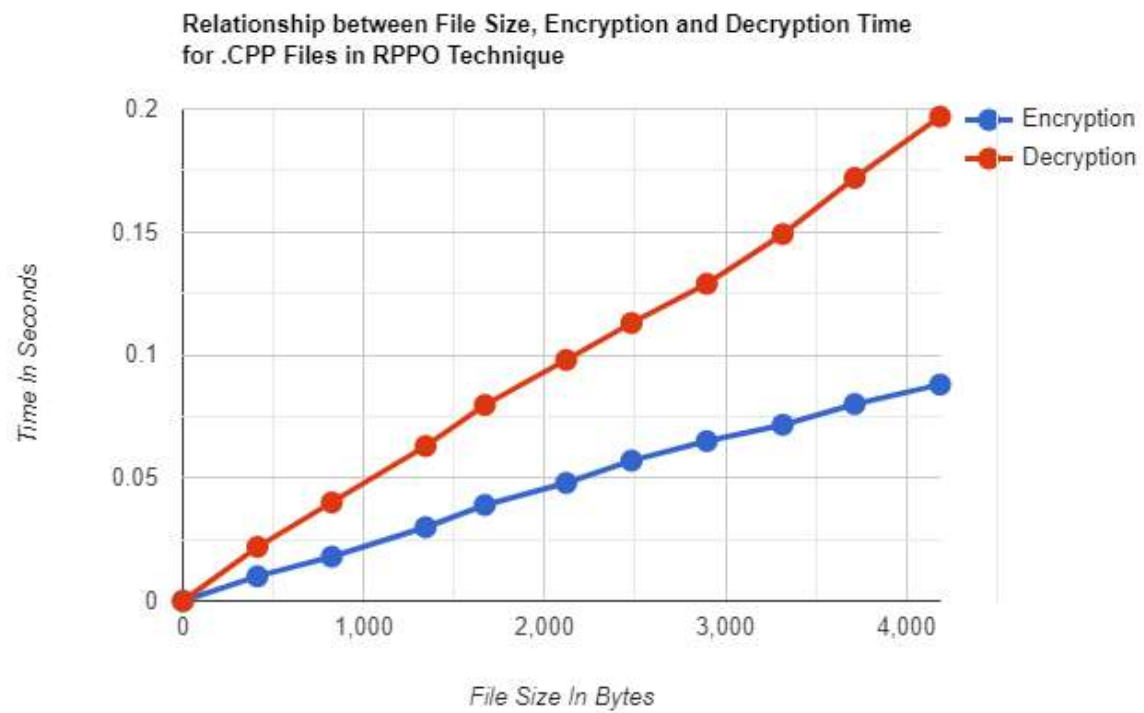
To experiment with the same set of sample files considered earlier, the technique of RPPO has been applied in a cascaded way with block sizes of 2, n increasing from 3 to 8. This means that first on the source file, the RPPO encryption technique is applied for blocks with the unique length of 8 bits.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

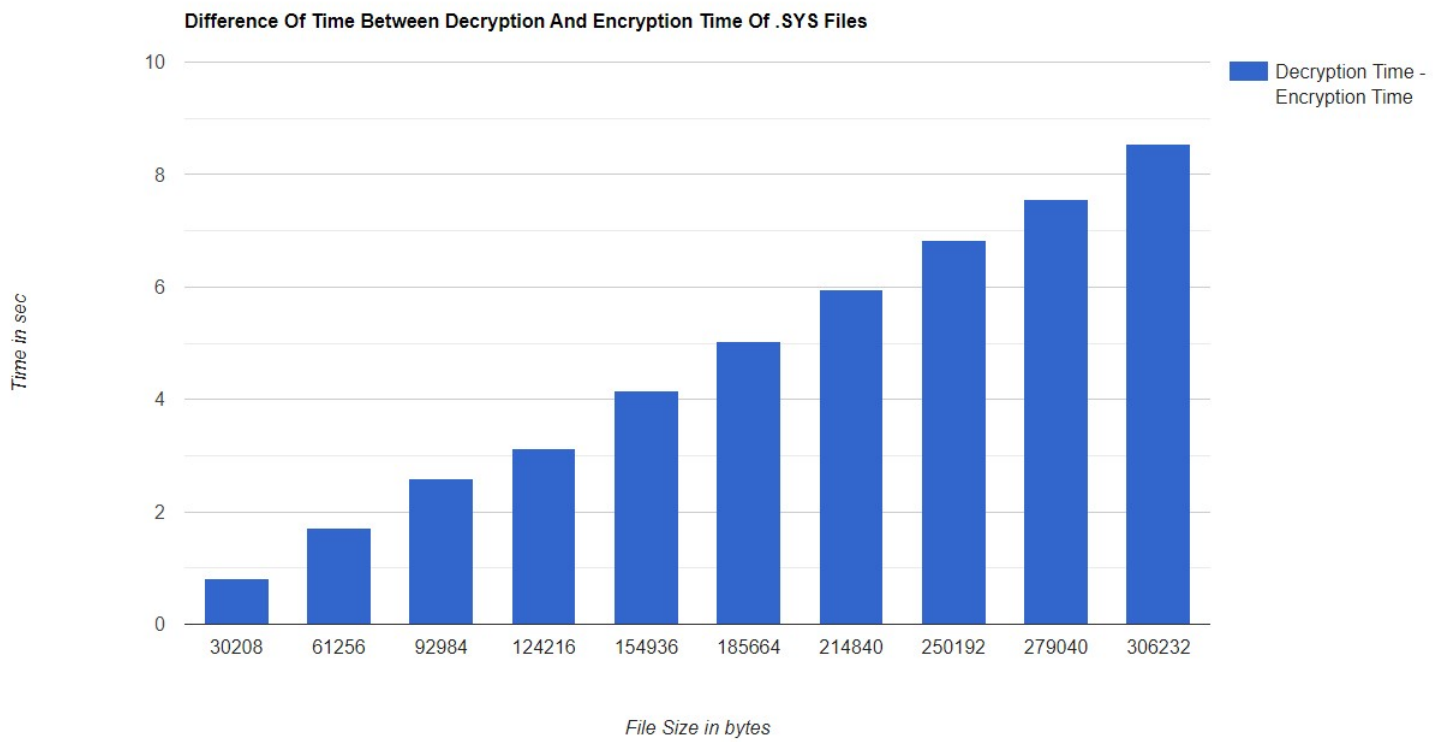
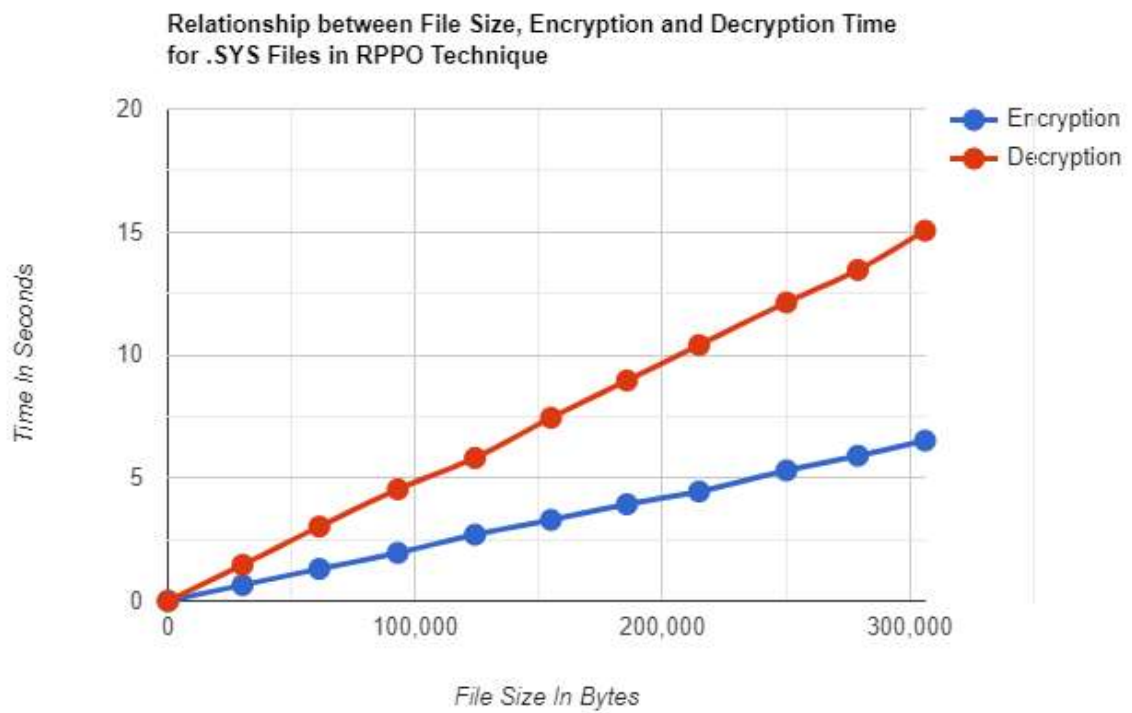
Source File	Source Size (In Bytes)	Encryption Time (In seconds)	Decryption Time (In seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
input.cpp	412	0.009989	0.021975	56736	85104	1265	55
input2.cpp	823	0.017982	0.040018	113904	170856	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



## 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.

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



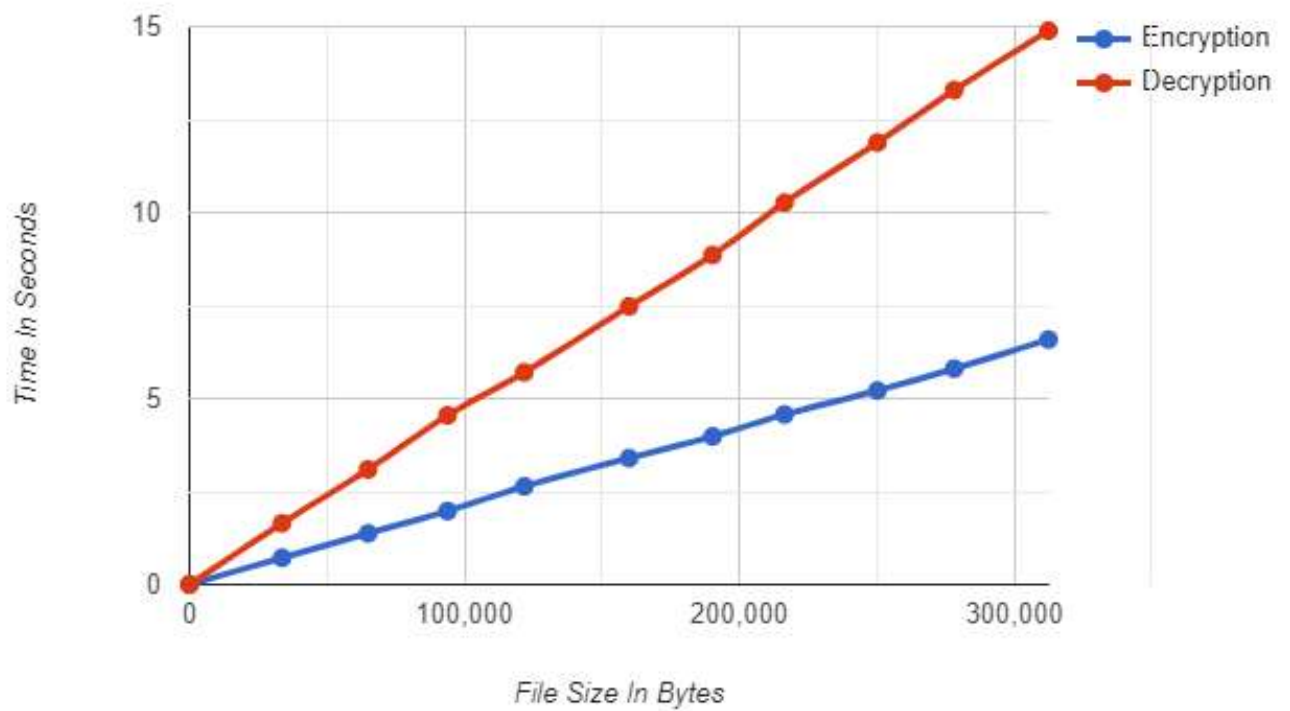


### 5.1.3 Results of .TXT files

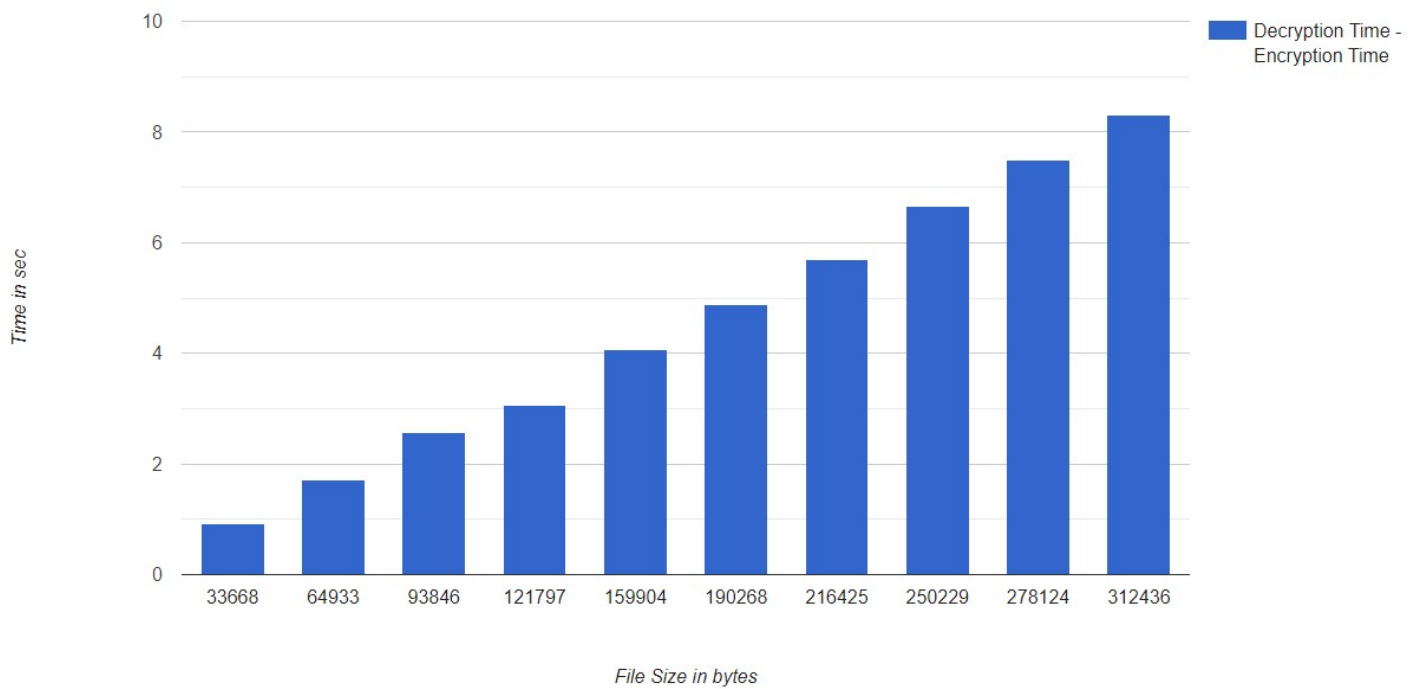
Table 5.1.3 gives the result of implementing the technique on TXT files. Ten files have been considered. The block number for each encryption is considered to be 2 where block size is 8. Their sizes range from 33668 bytes to 312436 bytes. The encryption time ranges 0.719773 seconds to 6.586717 seconds. 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.

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

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



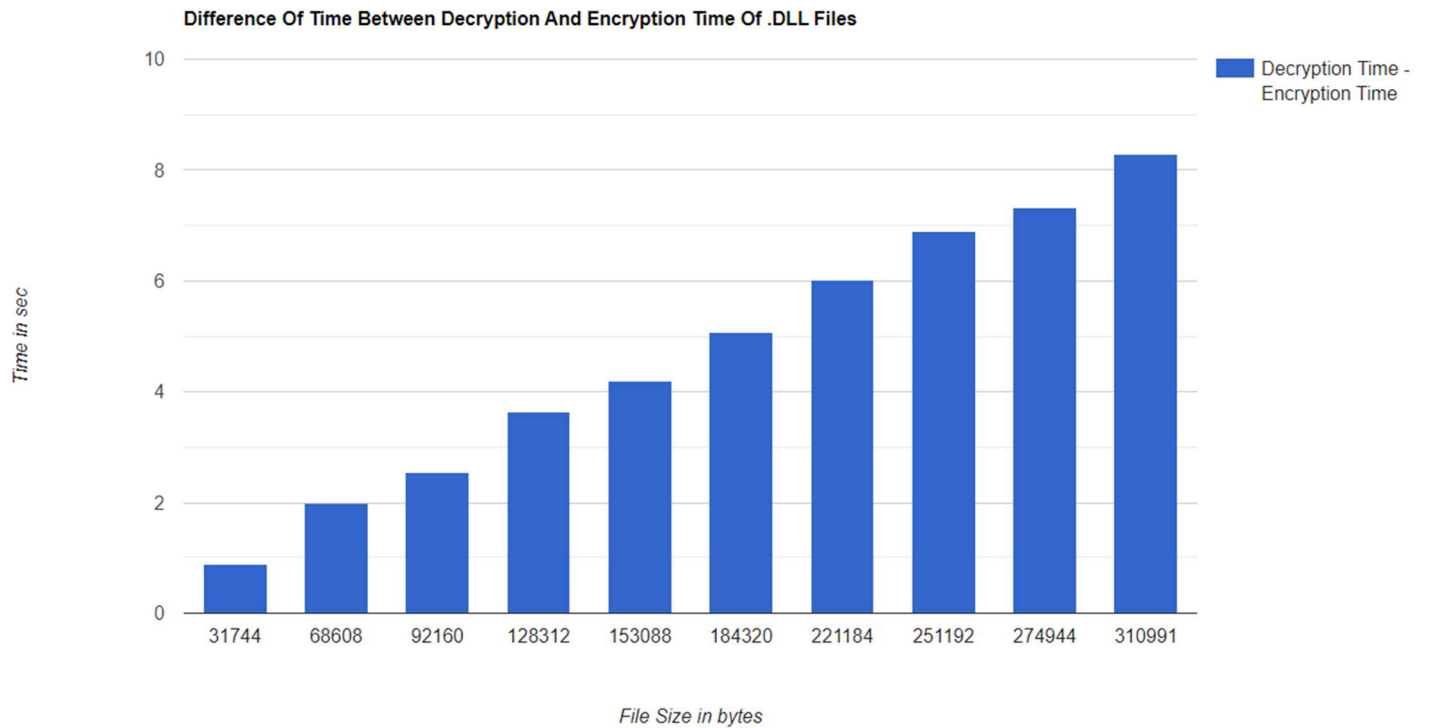
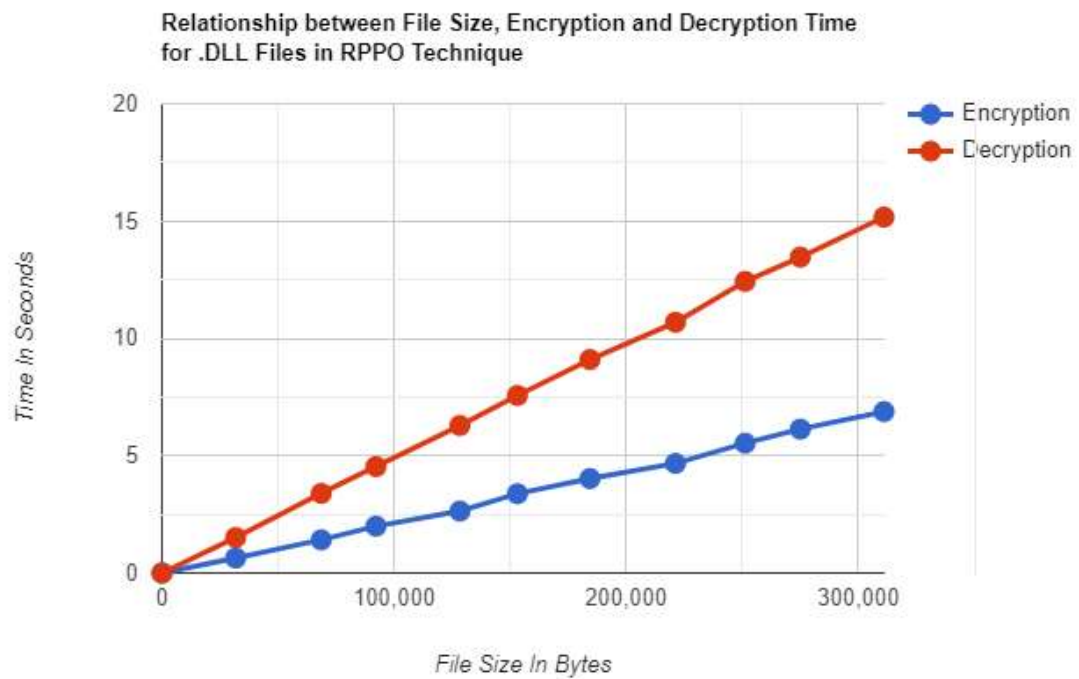
Difference Of Time Between Decryption And Encryption Time Of .TXT Files



## 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.

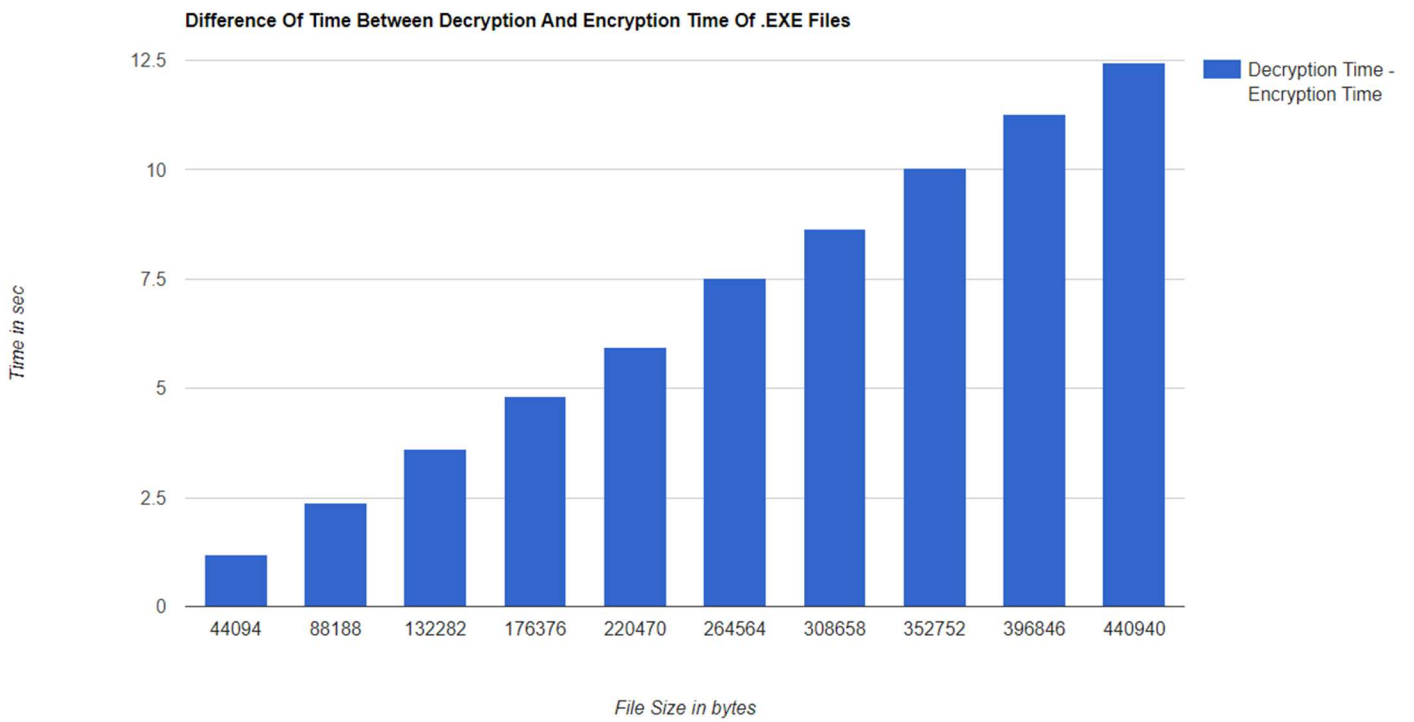
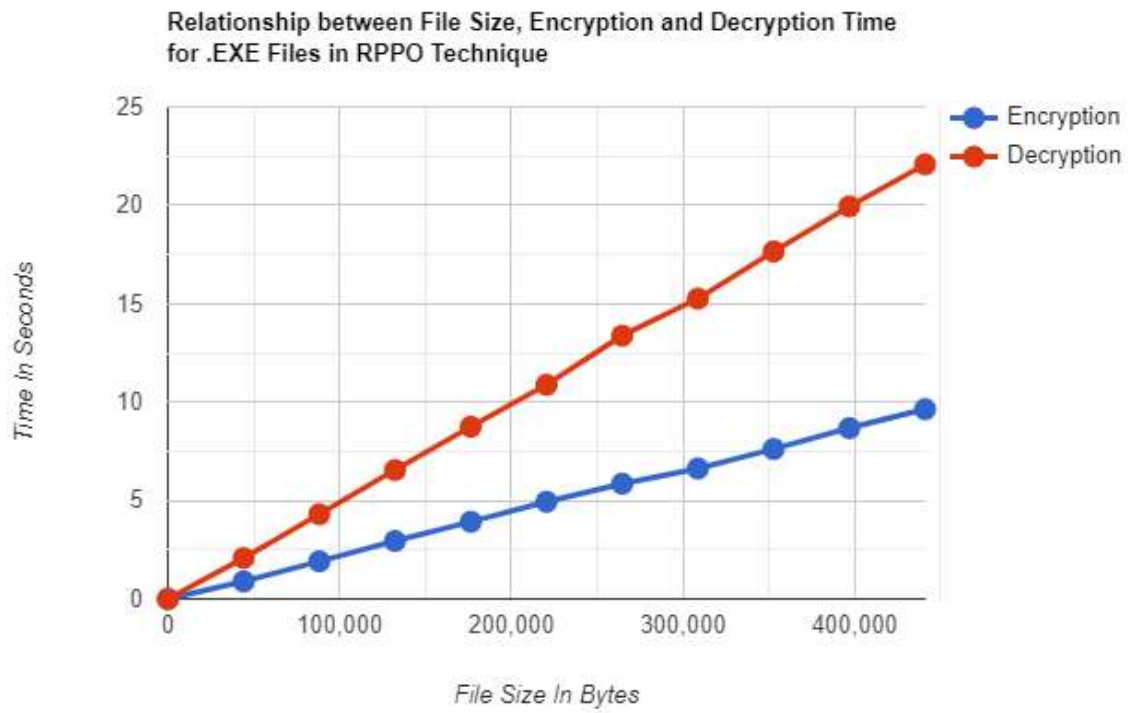
Source File	Source Size (In Bytes)	Encryption Time (In seconds)	Decryption Time (In seconds)	Number of Operations During Encryption	Number of Operations During Decryption	Chi Square Value	Degree of freedom
input.dll	31744	0.640250	1.512897	4570848	6856272	81096	255
input2.dll	68608	1.412551	3.395116	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



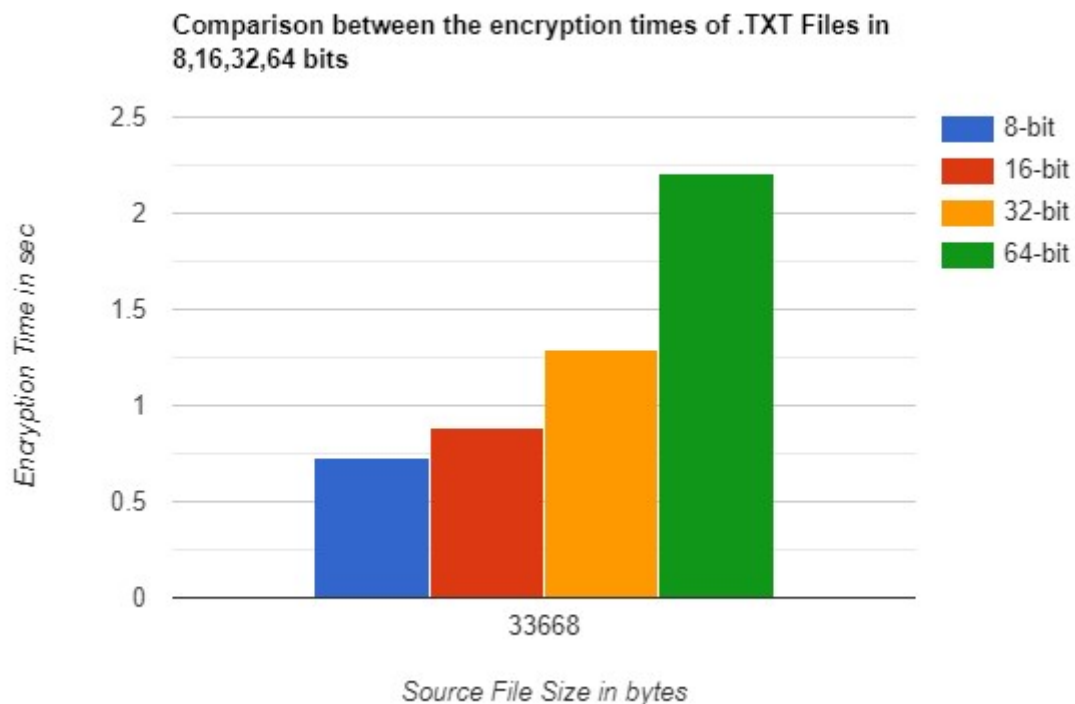
## 6. Comparison Of Encryption And Decryption Time With Respect To Block Size(Bits)

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.

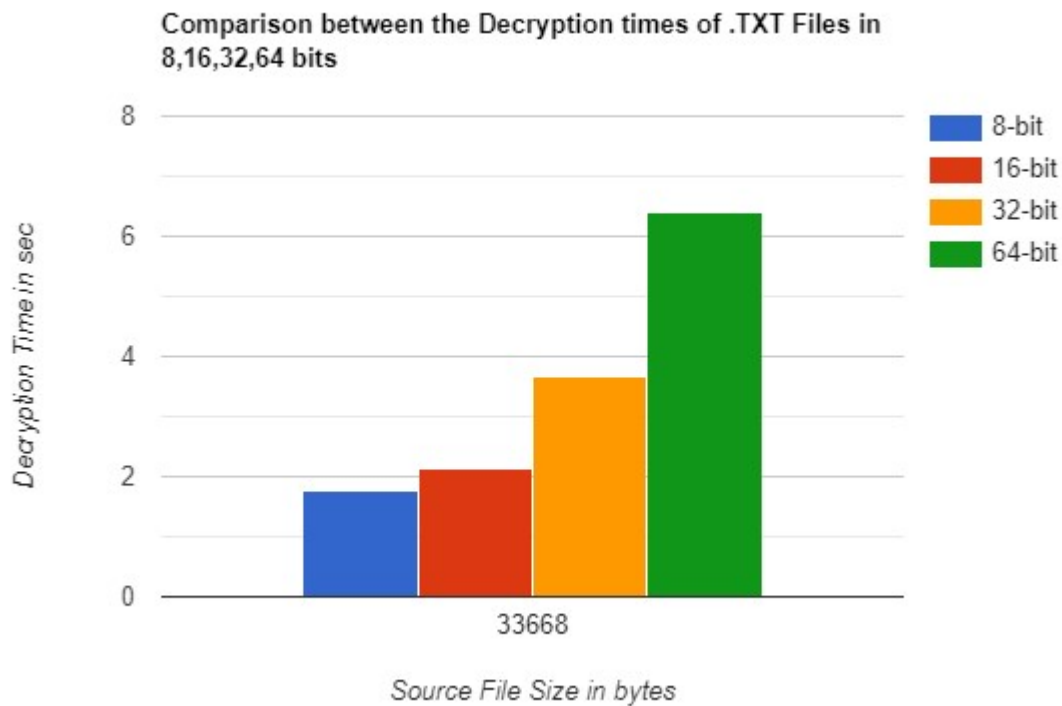
For comparing encryption times the block number being taken is 2 as the blocks will go through 2 iterations.

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.

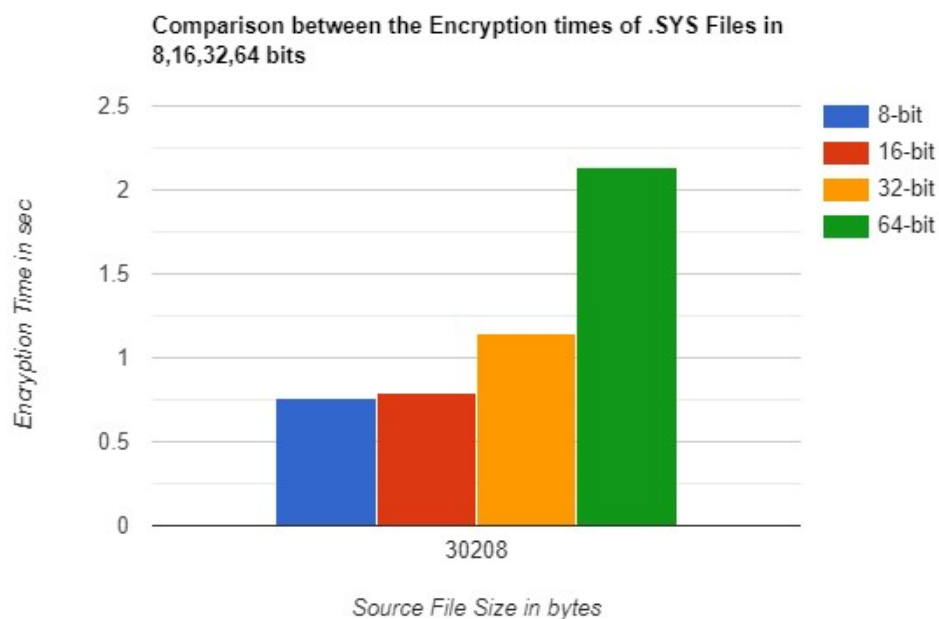
### 6.1.1 Comparison Between The Encryption Times Of .TXT Files in 8,16,32,64 bits



### 6.1.2 Comparison Between The Decryption Times of .TXT Files in 8,16,32,64 bits

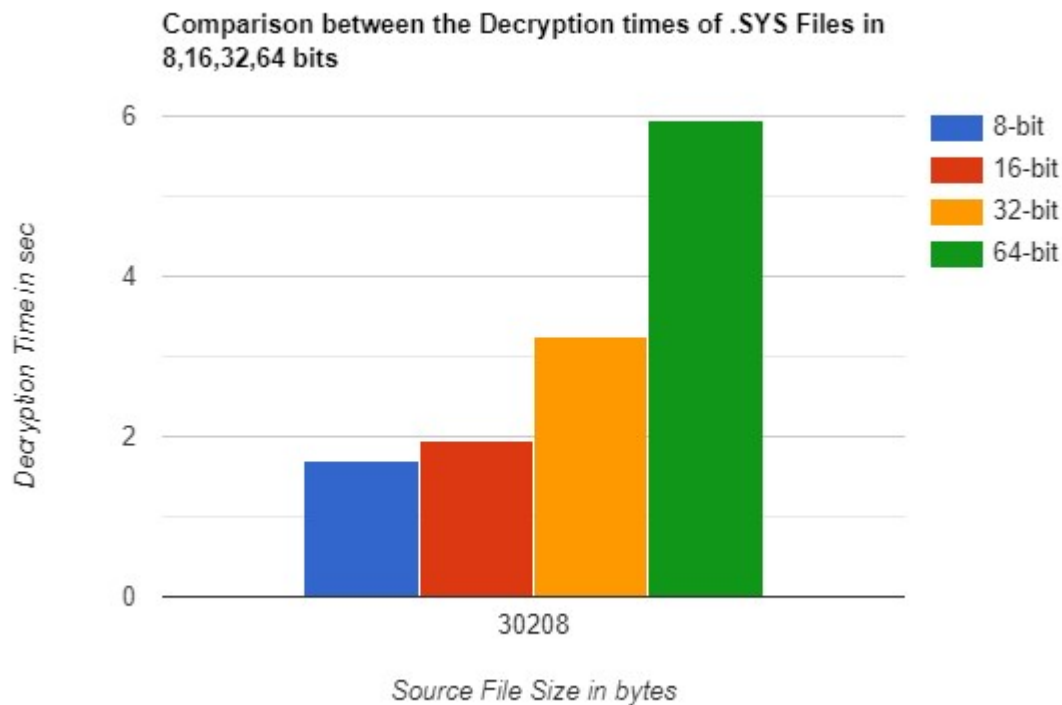


### 6.1.3 Comparison Between The Encryption Times of .SYS Files in 8,16,32,64 bits

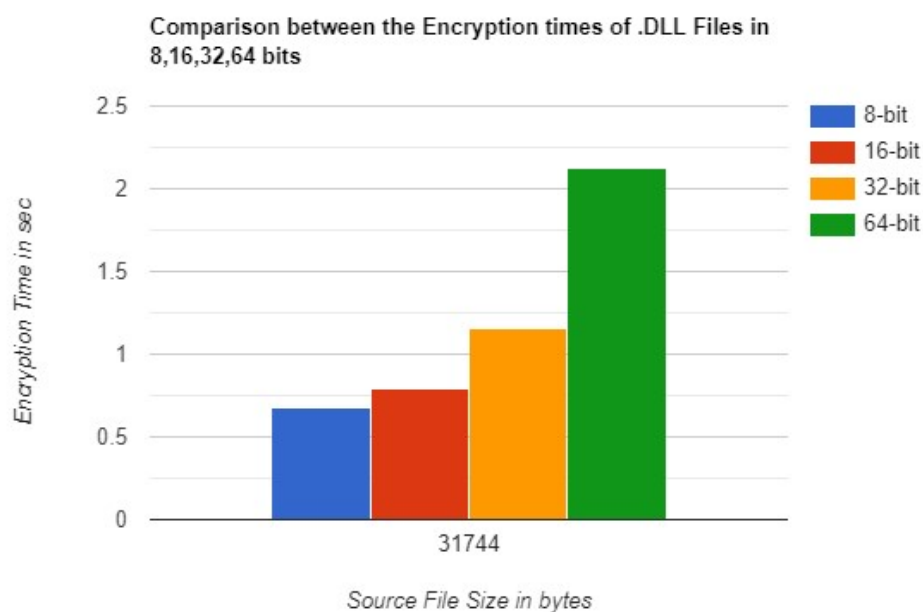




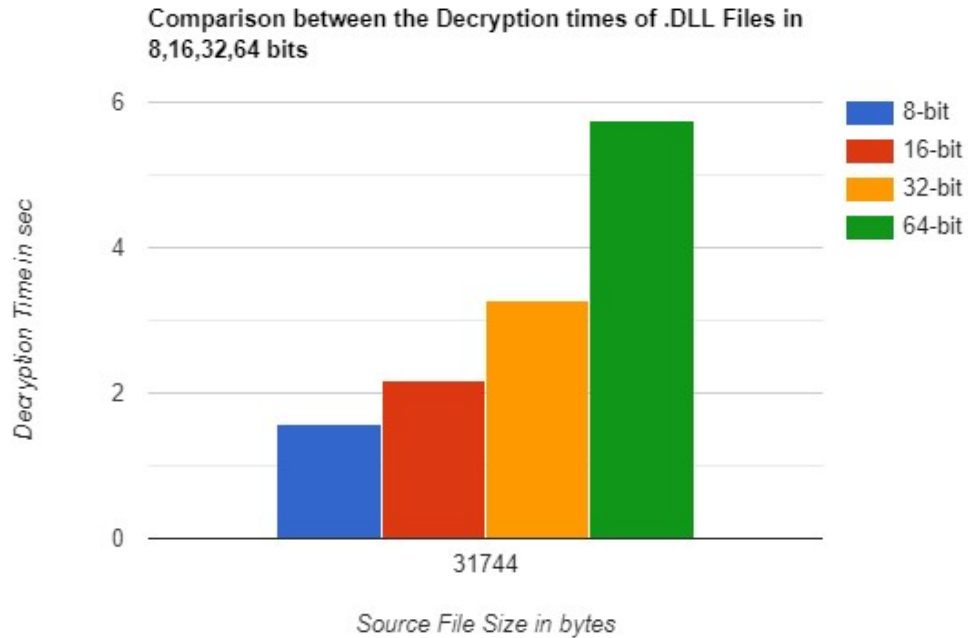
### 6.1.4 Comparison Between The Decryption Times of .SYS Files in 8,16,32,64 bits



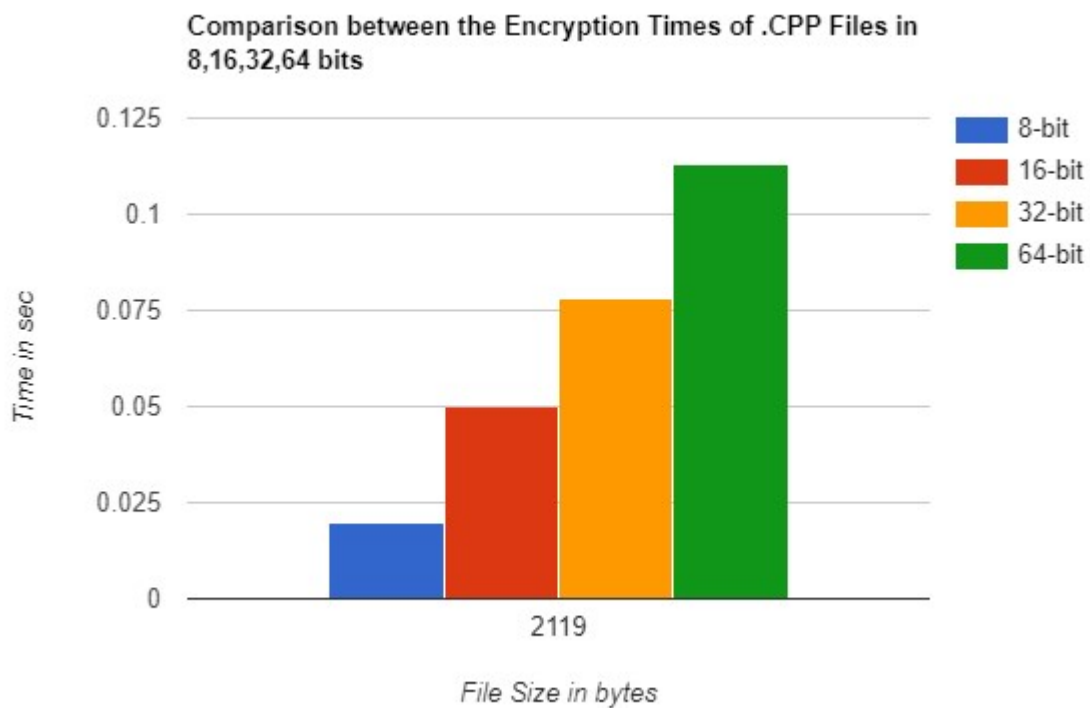
### 6.1.5 Comparison Between The Encryption Times of .DLL Files in 8,16,32,64 bits



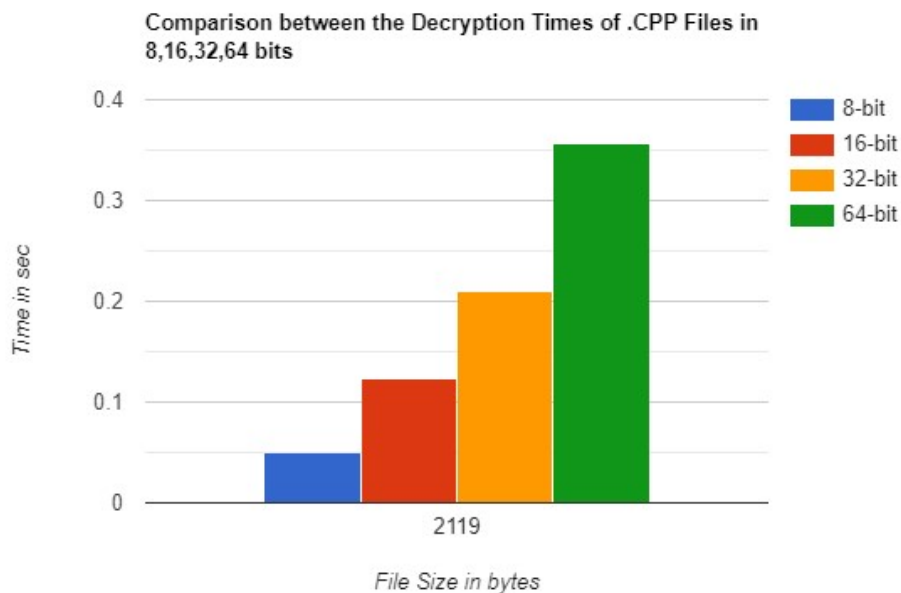
### 6.1.6 Comparison Between The Decryption Times of .DLL Files in 8,16,32,64 bits



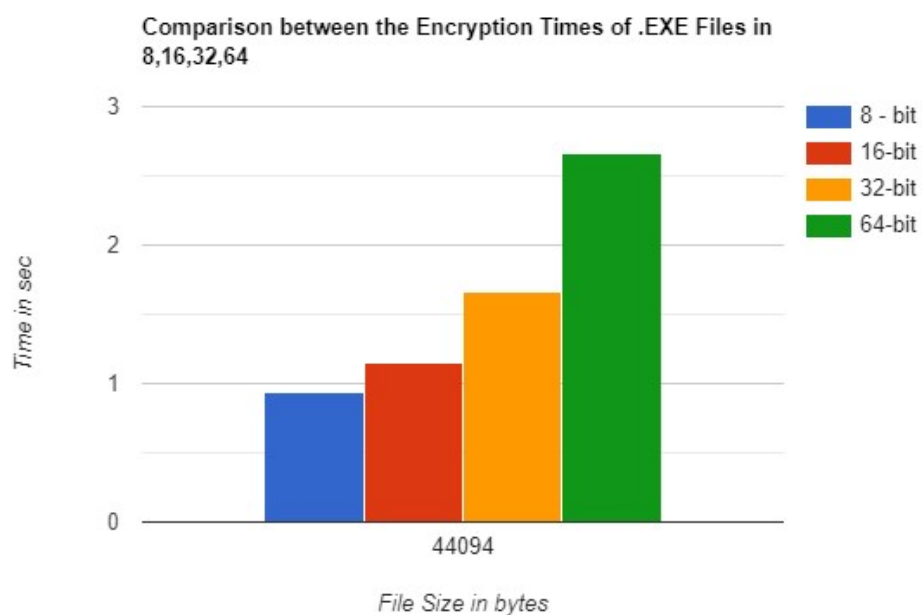
### 6.1.7 Comparison Between The Encryption Times of .CPP Files in 8,16,32,64 bits



### 6.1.8 Comparison Between The Decryption Times of .CPP Files in 8,16,32,64 bits



### 6.1.9 Comparison Between The Encryption Times of .EXE Files in 8,16,32,64 bits



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

