

Encryption Through Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB)

Thesis submitted in partial fulfillment of the requirements for the degree

of

Master of Technology

in

Computer Science and Engineering

By

Aditya Das

Roll 90/CSE No.: 230001

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

University of Kalyani

Kalyani, India

June, 2025

Declaration

I, Aditya Das, declare that this thesis titled “Encryption Through Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB)” is a bona fide record of the work carried out by me under the supervision of Prof. Jyotsna Kumar Mandal. This report represents my ideas in my own words and where others’ ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I further declare that the work reported in this thesis has not been and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or any other institute.

Date: 30/06/2025

Aditya Das

Roll: 90/CSE No.: 230001

Reg. No.: 508003 of 2023–2024

Acknowledgment

I want to take this opportunity to thank my supervisor Dr. Jyotsna Kumar Mandal, Department of Computer Science and Engineering, University of Kalyani, for his motivation and tireless efforts in helping me gain a deep understanding of the research area and for supporting me throughout the life cycle of my M. Tech. course. Especially, the extensive comments, healthy discussions, and fruitful interactions with the supervisors had a direct impact on the final form and quality of this dissertation work.

I am also thankful to Dr. Priya Ranjan Sinha Mahapatra, Professor and Head of the Department of Computer Science and Engineering, for his fruitful guidance through the early years of chaos and confusion. I extend my heartfelt thanks to the faculty members and supporting staff of the Department of Computer Science and Engineering for their generous support, guidance, and unwavering cooperation throughout this journey.

This thesis would not have been possible without the unwavering support of my friends. I am deeply grateful to my parents for their blessings, love, and constant encouragement.

Certificate

This is to certify that the thesis entitled “Encryption Through Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB)” submitted by “Aditya Das” (Reg. No. 5080003 of 2023-2024 , Roll. 90/CSE No. 230001) in partial fulfillment for the award of Master of Science in Computer Science and Engineering is a bona fide work carried out under my supervision. The thesis fulfills the requirements as per the regulations of this University and, in my opinion, meets the necessary standards for submission. The contents of this thesis have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma, and the same is certified. It is understood that by this approval, the undersigned does not necessarily endorse any of the statements made or opinions expressed therein but approves it only for the purpose for which it is submitted.

Internal Supervisor

External Supervisor

Head of the Department

External Examiner(s)

Abstract

In the evolving domain of data security, lightweight and efficient encryption techniques are crucial for ensuring confidentiality without introducing overhead in storage or computation. This project introduces a novel encryption technique titled “Encryption Through Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB)”, designed to enhance security through bit-level processing using recursive logic. Unlike traditional algorithms that often rely on complex mathematical transformations and fixed block structures, RMOPB performs a recursive bitwise XOR operation on paired bits within variable-sized blocks, offering a high degree of randomness and data obfuscation.

The proposed method begins by converting the source message into a stream of bits and decomposing this stream into blocks, where each block size is dynamically determined within a predefined range. Each block undergoes several rounds of recursive XOR operations on bit-pairs, forming intermediate encrypted blocks. A specific intermediate block from this cycle is selected as the encrypted output based on a session key, which also governs the block sizes and number of iterations, thereby providing strong key diversity and session uniqueness.

The decryption process mirrors the encryption, requiring only the session key to reverse the transformation and recover the original data. One of RMOPB’s key strengths is its resistance to frequency-based and statistical attacks, as validated by Chi-square analysis and frequency distribution evaluations across various file types (.DOC, .EXE, .DLL, .SYS, .CPP). Additionally, the RMOPB technique was benchmarked against established cryptographic standards like RSA and TDES. Results showed that RMOPB achieved comparable or superior encryption quality while maintaining lower computational overhead, especially in file formats commonly used in software systems.

This project demonstrates that RMOPB can serve as a practical and secure alternative to conventional encryption methods, particularly in resource-constrained environments where efficiency and adaptability are critical.

Table of Contents

Declaration	ii
Acknowledgment	iv
Certificate	vi
Abstract	viii
1 Introduction	1
2 Literature Review	2
3 Proposed Work	4
3.0.1 The Scheme	4
3.0.2 Methodology	4
3.0.3 Example	6
3.0.4 Implementation	7
3.0.5 Chi square Calculation	12
4 Results and Discussion	13
4.0.1 RMOPB without Session Key	13
4.0.1.1 Results of .CPP Files	13
4.0.1.2 Results of .SYS Files	15
4.0.1.3 Results of .TXT Files	17
4.0.1.4 Results of .DLL Files	19
4.0.1.5 Results of .EXE Files	21
4.0.2 RMOPB with Session Key Method	23

4.0.2.1	Generating the Session Key	23
4.0.2.2	Implementation	23
4.0.3	RMOPB with Session Key	24
4.0.3.1	Results of .CPP Files	25
4.0.3.2	Results of .SYS Files	27
4.0.3.3	Results of .TXT Files	29
4.0.3.4	Results of .DLL Files	31
4.0.3.5	Results of .EXE Files	33
4.0.4	Comparison between Encryption and Decryption Time of RMOPB and RMOPB using Session Key	35
4.0.4.1	Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .TXT files	35
4.0.4.2	Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .TXT files	36
4.0.4.3	Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .EXE files	36
4.0.4.4	Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .EXE files	37
4.0.4.5	Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .CPP files	37
4.0.4.6	Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .CPP files	38
4.0.4.7	Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .SYS files	38
4.0.4.8	Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .SYS files	39
4.0.4.9	Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .DLL files	39

4.0.4.10	Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .DLL files	40
4.0.5	NIST Test Report	41
5	Conclusion and Future Works	42

List of Figures

3.1	Pictorial representation of the RMOPB technique	5
3.2	Pictorial representation of the RMOPB technique for source block P = 10110111	6
3.3	Converting character into bytes for “Data Structure”	8
3.4	Different blocks considered for encryption	9
3.5	Formation of cycle for block S1 for RMOPB technique	9
3.6	Formation of cycle for block S2 for RMOPB technique	9
3.7	Formation of cycle for block S3 for RMOPB technique	10
3.8	Formation of cycle for block S4 for RMOPB technique	10
3.9	Formation of cycle for block S5 for RMOPB technique	10
3.10	Formation of cycle for block S6 for RMOPB technique	11
3.11	Formation of cycle for block S7 for RMOPB technique	11
4.1	Comparison of encryption and decryption time of .CPP files	14
4.2	16
4.3	18
4.4	20
4.5	22
4.6	Comparison of encryption and decryption time of RMOPB using session key of .CPP files	26

4.7	Comparison of encryption and decryption time of RMOPB using session key of .SYS files	28
4.8	Comparison of encryption and decryption time of RMOPB using session key of .TXT files	30
4.9	Comparison of encryption and decryption time of RMOPB using session key of .DLL files	32
4.10	Comparison of encryption and decryption time of RMOPB using session key of .EXE files	34
4.11	35
4.12	36
4.13	36
4.14	37
4.15	37
4.16	38
4.17	38
4.18	39
4.19	39
4.20	40

Chapter 1.

Introduction

In an attempt to attain a satisfactory level of security, the Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB) eliminates the possibility of overhead. In this RMOPB technique, the basic operation performed is the modulo-2 operation, rooted in the concepts of Boolean algebra.

Like all other proposed techniques, RMOPB is also a bit-level application. Therefore, after converting the source message to be transmitted into a stream of bits, the stream is decomposed into a finite number of blocks. However, in the RMOPB technique, since the basic mechanism is simpler than previous techniques and, moreover, there is no possibility of storage expansion, it is highly recommended to choose blocks of varying sizes. This approach ensures a larger key length, which almost nullifies the chance of breaking the cipher through cryptanalysis—especially when different block lengths are used.

For a given block of bits, a block of n bits is taken as an input stream (where n varies from 8 to 256) from the continuous bit stream, and the technique operates on it to generate the intermediate encrypted stream. The same operation is repeatedly performed for different block sizes, as specified in the session key, to generate the final encrypted stream.

The technique when implemented with varying block sizes offers good security, no expansion in size being the added advantage. Section 2.0 discusses the scheme. The technique is implemented in section 3.0 for the different confidential messages. Section 4.0 enlists the results from different perspectives after the technique is implemented for the same groups of files that were considered for the other proposed techniques and compared with RSA and TDES techniques. An analytical evaluation of the technique is presented in section 6.5.

Chapter 2.

Literature Review

Cryptography, as a fundamental pillar of information security, has evolved significantly with the increasing demand for lightweight, fast, and secure encryption methods. Traditional encryption techniques like RSA and TDES offer robust protection but are often associated with computational overhead, fixed key sizes, and storage expansion issues. To overcome these challenges, researchers have introduced several bit-level cryptographic techniques that operate directly on binary streams, offering enhanced control and flexibility.

One such advancement is the Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB). Unlike traditional schemes that use static block sizes or algebraic transformations, RMOPB introduces a recursive bitwise XOR mechanism on dynamically sized blocks of binary data. It derives inspiration from Boolean algebra and relies heavily on cyclic transformations where a stream of bits is divided into variable-sized blocks, and each block undergoes several recursive iterations using XOR operations. This cyclic behavior ensures that after a certain number of iterations, the original data can be reconstructed, enabling symmetric encryption and decryption using the same session key.

The concept of session key-based encryption in RMOPB introduces variability in encryption, as both block sizes and the number of iterations can differ per session, significantly expanding the key space. Comparative studies highlight RMOPB's advantages over RSA and TDES, particularly in terms of encryption time, decryption time, and

statistical randomness. In experimental results, RMOPB consistently produced lower encryption/decryption times while achieving comparable or better Chi-square values, indicating a high degree of non-uniformity in encrypted data—a desirable trait for strong encryption .

Moreover, frequency distribution tests conducted on various file types (.EXE, .DOC, .DLL, .SYS, .CPP) demonstrate that RMOPB-generated ciphertext exhibits excellent randomness, ensuring resistance against frequency analysis attacks. The technique’s inherent ability to support block size variability adds to its cryptanalytic resilience, making brute-force attempts impractical due to the expanded keyspace .

Previous techniques like CAOPB (Cascaded Application of Paired Bit Operations) served as precursors, yet RMOPB improves upon them by eliminating storage expansion and simplifying implementation without compromising security. While RSA is dependent on prime factorization and TDES on fixed permutation and substitution rounds, RMOPB relies on logic-based recursion, making it ideal for resource-constrained environments where lightweight cryptography is essential.

In summary, RMOPB represents a significant contribution to the field of symmetric encryption, particularly for bit-level and resource-aware secure communications, offering both computational efficiency and strong encryption guarantees over a variety of data formats.

Chapter 3.

Proposed Work

3.0.1 The Scheme

After generating the stream of bits from the source message to be transmitted, it is to be decomposed into a finite number of blocks. Till this point, the technique is the same as all the other proposed techniques. Section 3.0.1 discusses the policy of encryption and decryption.

3.0.2 Methodology

The technique, considers the plaintext as a stream of finite number of bits N , and is divided into a finite number of blocks, each also containing a finite number of bits n , Where, $1 \leq n \leq N$

Let $P = s_0^0 s_1^0 s_2^0 s_3^0 s_4^0 \dots s_{n-1}^0$ be a block of size n in the plaintext. Then the first intermediate block $I_1 = s_0^1 s_1^1 s_2^1 s_3^1 s_4^1 \dots s_{n-1}^1$ can be generated from P in the following way:

$$s_0^1 = s_0^0 \oplus s_1^0 \oplus s_2^0$$

$$s_1^1 = s_0^0 \oplus s_2^0 \oplus s_3^0$$

$$s_2^1 = s_0^0 \oplus s_3^0 \oplus s_4^0$$

$$s_j^1 = s_{i+j}^0 \oplus s_{i+j+1}^0 \oplus s_{i+j+2}^0 \oplus s_{i+j+3}^0, \quad 0 \leq i < (n-1), \quad 0 \leq j < (n-1);$$

\oplus stands for the exclusive-OR operation.

In the same way, the second intermediate block $I_2 = s_0^2 s_1^2 s_2^2 s_3^2 \dots s_{n-1}^2$ of the same size (n) can be generated by:

$$s_0^2 = s_0^1 \oplus s_1^1 \oplus s_2^1$$

$$s_2^2 = s_0^1 \oplus s_3^1 \oplus s_4^1$$

$$s_j^2 = s_{i+j}^1 \oplus s_{i+j+1}^1 \oplus s_{i+j+2}^1 \oplus s_{i+j+3}^1, \quad 0 \leq i < (n-1), \quad 1 < j < (n-1);$$

\oplus stands for the exclusive-OR operation.

If this process continues for a finite number of iterations, the source block P is regenerated forming a cycle, which depends on the value of block size n . Any intermediate block in the recursive process may be termed an intermediate encrypted block for that source block. The operation is repeated for the whole stream in the source block.

The same operation is performed for the whole stream number of times with varying block sizes. K such iteration is done and the final intermediate stream after k iterations generates the encrypted stream. All of the different block sizes and k constitute the key for the session. This key may be considered as a session key for that particular session. Figure 3.1 represents the technique pictorially (single step of the iteration process). The process of decryption is the same as that of encryption.

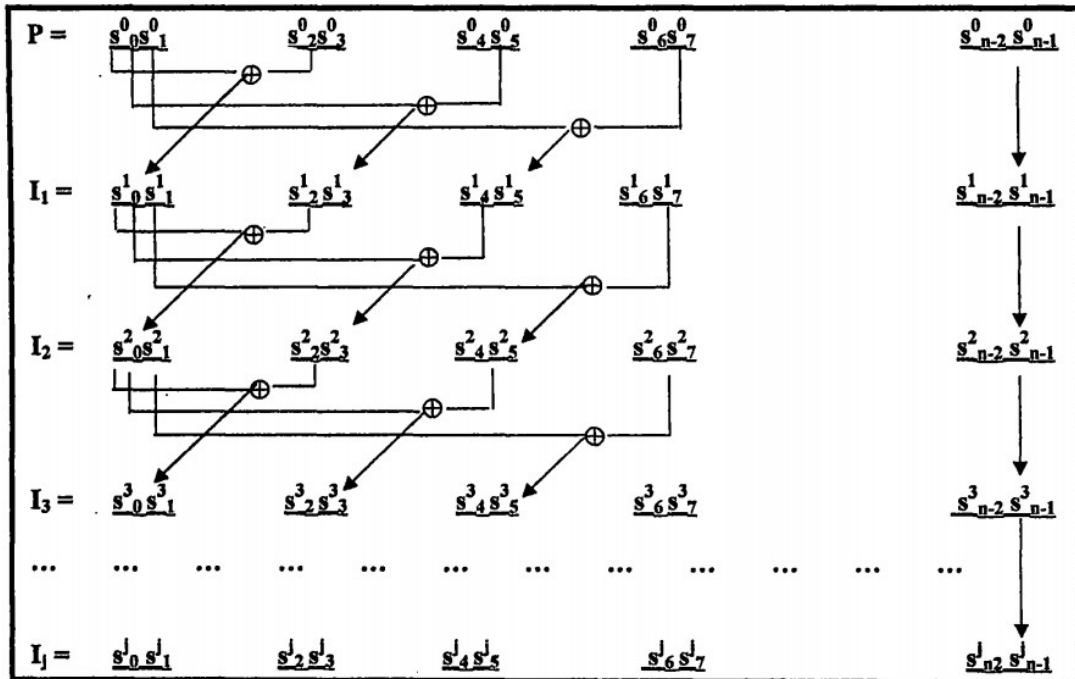


Figure 3.1: Pictorial representation of the RMOPB technique

3.0.3 Example

To illustrate the technique, let $P = 10110111$ be an 8-bit source block. Figure 3.2 shows the generation of the cycle for this sample block. Here it requires 4 iterations to regenerate the source block.

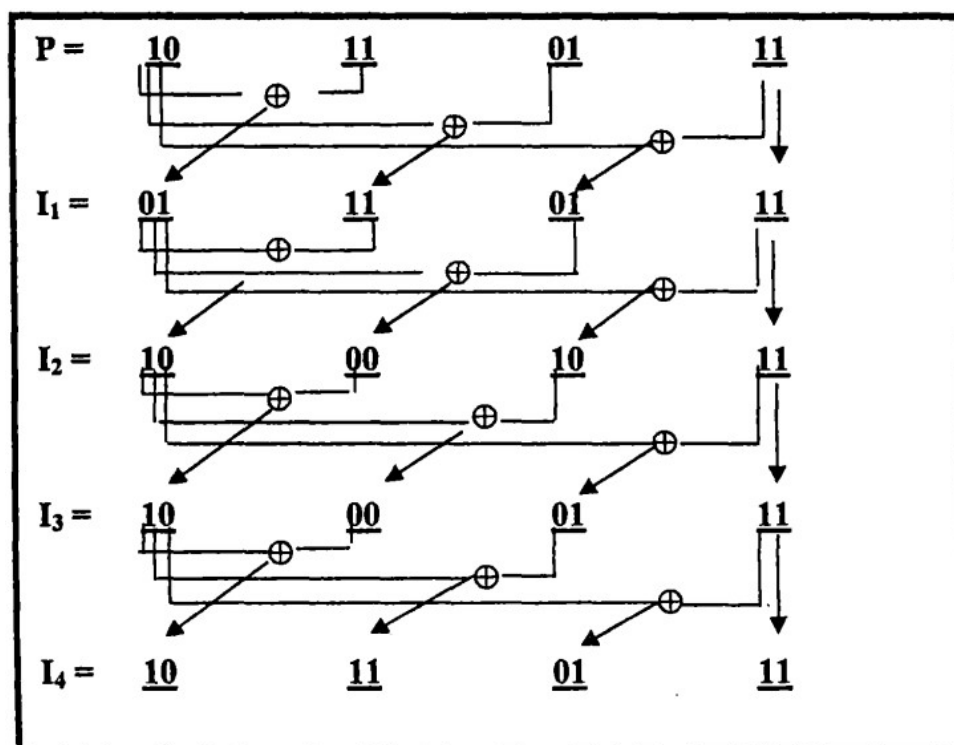


Figure 3.2: Pictorial representation of the RMOPB technique for source block $P = 10110111$

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

With respect to one single block of bits, any intermediate block during the process of forming the cycle can be considered as the encrypted block. If the total number of iterations required to complete the cycle is P and the 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 the 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 a source block of bits, the 4th, the 7th and the 5th intermediate blocks respectively are being considered as the encrypted blocks. In such a case, the key of the scheme will become much more complex, which in turn will ensure better security. In the case of blocks with varying lengths, different blocks will require different numbers of iteration to form the corresponding cycle. So, the LCM value, say, P , of all these numbers will give the actual number of iterations required to form the cycle for the entire stream. Now, if i number of iterations are performed to encrypt the entire stream, then a number of $(P - i)$ more iterations will be required to decrypt the encrypted stream.

3.0.4 Implementation

Whenever in any technique it is suggested to form blocks of varying lengths, the format of the secret key becomes very vital. This section chooses the different plaintext as compared to previous techniques. For the purpose of real implementation of the RMOPB technique, consider the plaintext (P) as: “Data Structure”

Table 3.3 and table 3.4 show the character into byte conversion of the above given plain text and different blocks considered for encryption. Tables 3.5 to 3.11 show the

formation of cycles for blocks $S_1, S_2, S_3, S_4, S_5, S_6, S_7$ respectively. Now, for each of the blocks, an arbitrary intermediate block, as indicated

in each table, is considered as the encrypted block.

Character	Byte
D	01000100
a	01000001
t	01010100
a	01000001
<Blank>	00110000
S	01010011
t	01010111
r	01010010
u	01010101
c	01000011
t	01010100
u	01010101
r	01010010
e	01000101

Figure 3.3: Converting character into bytes for “Data Structure”

Combining together all the bytes, the following stream of bits of the length of 112 bits are obtained. 01000100/01000001/01010100/01000001/00110000/01010011/01010111/01010010/01010101101000011/01010100/01010101/01010010/01000101 ”/” being working as the separator between two consecutive bytes.

Block 1 (S1)	0100010001000001
Block 2 (S2)	0101010001000001
Block 3 (S3)	0011000001010011
Block 4 (S4)	0101011101010010
Block 5 (S5)	0101010101000011
Block 6 (S6)	0101010001010101
Block 7 (S7)	0101001001000101

Figure 3.4: Different blocks considered for encryption

Different steps of the process of encryption are presented in tabular form in table 3.5 to 3.11.

Source block	0101010001000001
Block (I₂₁) after iteration 1	0000010001010001
Block (I₂₂) after iteration 2	0001000101000101
Block (I₂₃) after iteration 3	0100010100010101
Block (I₂₄) after iteration 4	0100000100000001
Block (I₂₅) after iteration 5	0101000101010001
Block (I₂₆) after iteration 6	0001000000010001
Block (I₂₇) after iteration 7	0100000001000101
Block (I₂₈) after iteration 8	0101010001000001

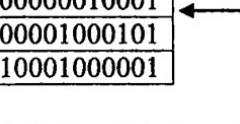
Encrypted block 

Figure 3.5: Formation of cycle for block S1 for RMOPB technique

Source block	0011000001010011
Block (I₃₁) after iteration 1	1100000101001111
Block (I₃₂) after iteration 2	1111101011000011
Block (I₃₃) after iteration 3	0001010011110011
Block (I₃₄) after iteration 4	0101001111001111
Block (I₃₅) after iteration 5	0001101001101011
Block (I₃₆) after iteration 6	0110100110101111
Block (I₃₇) after iteration 7	1111001111101011
Block (I₃₈) after iteration 8	0011000001010011

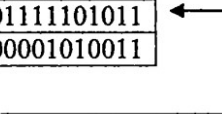
Encrypted block 

Figure 3.6: Formation of cycle for block S2 for RMOPB technique

Source block	0101011101010010
Block (I₄₁) after iteration 1	0000100000011110
Block (I₄₂) after iteration 2	0010000001111010
Block (I₄₃) after iteration 3	1000000111101010
Block (I₄₄) after iteration 4	1010110100000010
Block (I₄₅) after iteration 5	0001111010100010
Block (I₄₆) after iteration 6	0111101010001010
Block (I₄₇) after iteration 7	1011111011111110
Block (I₄₈) after iteration 8	0101011101010010

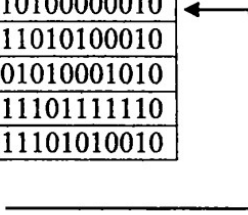
Encrypted block 

Figure 3.7: Formation of cycle for block S3 for RMOPB technique

Source block	0101010101000011
Block (I₅₁) after iteration 1	0000000001011011
Block (I₅₂) after iteration 2	0000000101101111
Block (I₅₃) after iteration 3	0000010110111111
Block (I₅₄) after iteration 4	0001011011111111
Block (I₅₅) after iteration 5	0101101111111111
Block (I₅₆) after iteration 6	0011101010101011
Block (I₅₇) after iteration 7	1110101010101111
Block (I₅₈) after iteration 8	0101010101000011

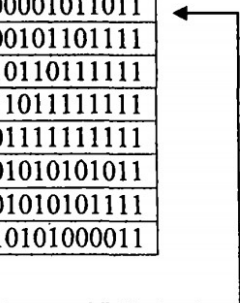
Encrypted block 

Figure 3.8: Formation of cycle for block S4 for RMOPB technique

Source block	0101010001010101
Block (I₆₁) after iteration 1	0000010000000001
Block (I₆₂) after iteration 2	0001000000000101
Block (I₆₃) after iteration 3	0100000000010101
Block (I₆₄) after iteration 4	0101010100000001
Block (I₆₅) after iteration 5	0000000101010001
Block (I₆₆) after iteration 6	0000010101000101
Block (I₆₇) after iteration 7	0001010100010101
Block (I₆₈) after iteration 8	0101010001010101

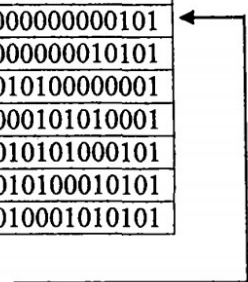
Encrypted block 

Figure 3.9: Formation of cycle for block S5 for RMOPB technique

Source block	0101001001000101
Block (I₇₁) after iteration 1	0001110001000001
Block (I₇₂) after iteration 2	0111000100000101
Block (I₇₃) after iteration 3	1001000101000001
Block (I₇₄) after iteration 4	1110111110101101
Block (I₇₅) after iteration 5	0100000101001001
Block (I₇₆) after iteration 6	0101000001110001
Block (I₇₇) after iteration 7	0001010010010001
Block (I₇₈) after iteration 8	0101001001000101

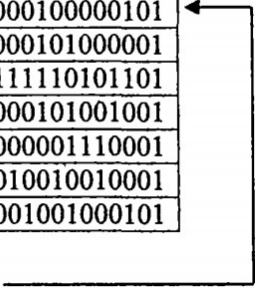
Encrypted block 

Figure 3.10: Formation of cycle for block S6 for RMOPB technique

Source block	0100010001000001
Block (I₁₁) after iteration 1	0100010001010001
Block (I₁₂) after iteration 2	0100010000010001
Block (I₁₃) after iteration 3	0100010100010001
Block (I₁₄) after iteration 4	0100000100010001
Block (I₁₅) after iteration 5	0101000100010001
Block (I₁₆) after iteration 6	0001000100010001
Block (I₁₇) after iteration 7	0100010001000101
Block (I₁₈) after iteration 8	0100010001000001

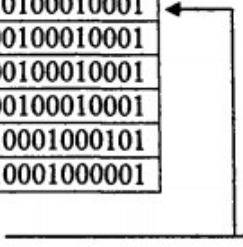
Encrypted block 

Figure 3.11: Formation of cycle for block S7 for RMOPB technique

As indicated in tables 3.5 to 3.11, intermediate blocks I13 (0100010100010001), I26 (0001000000010001), I37 (1111001111101011), I44 (1010110100000010), I51 (0000000001011011), I62 (000 10000000001 01), I72 (011000100000101) are considered as the encrypted blocks.

The encrypted stream can be rewritten as the series of bytes as follows: 01000101/0001000110001000/1111001111110101/10101101100000010/ 00000000/01011011/00010000/000001011011110001/00000101, "/" being used as only the separator.

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

$$C = \text{"\textcircled{a}@\textcircled{a}-\textcircled{z}\textcircled{O}\textcircled{i}\textcircled{\square}\textcircled{j}\textcircled{n}"}$$

After converting the cipher text C into a stream of bits, the technique of decomposition into several blocks of bits should follow the same way the source was decomposed.

Now, the process for the decryption is the same as the encryption. After converting the cipher text C into a stream of bits, it is to be decomposed into the different blocks like (C1, C2 ... C7). For each block, the same process as the encryption process is to be followed but for a varying number of iterations. For example, while decrypting the encrypted block Ct. the number of iterations should be $6 - 1 = 5$, iterations. The same logic is to be applied for the remaining blocks. After obtaining the source blocks in this way, it was kept as it is, in the same sequence and thus the source stream of bit is obtained, from which the source text is regenerated.

3.0.5 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:

$$\chi^2 = \sum \frac{(f_o - f_c)^2}{f_c} \quad (3.1)$$

Here f_c and f_o 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.

Chapter 4.

Results and Discussion

For the purpose of implementation a unique size of 64 bit has been considered, and for each such block, a symmetric technique has been implemented. In the report below the section represents the Encryption/Decryption times and Chi square values. Also section 5.1 shows the result on .CPP files, section 5.2 shows the result on .SYS files, section 5.3 shows the result on .TXT files, section 5.4 shows the result on .DLL files, and section 5.5 shows the result on .EXE files.

4.0.1 RMOPB without Session Key

4.0.1.1 Results of .CPP Files

Table 5.1.1 gives the result of implementing the technique on .CPP files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.04 seconds to 1.15 seconds. The decryption time ranges from 0.001 seconds to 2.87 seconds. The Chi Square value ranges from 525.3846154 to 2402080.133 and the degree of freedom ranges from 226 to 243.

Table 4.1: Encryption and Decryption Time with Chi-Square Analysis

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	525.3846154	243
test_file.02	5000	0.25	0.39	265413.7584	226
test_file.03	10000	0.32	0.81	533724.1181	226
test_file.04	14000	0.48	0.85	795583.5516	226
test_file.05	19000	0.50	1.87	1070268.931	226
test_file.06	14000	0.59	1.42	1333481.988	226
test_file.07	28000	0.73	1.84	1594081.196	226
test_file.08	33000	0.87	3.26	1863587.235	226
test_file.09	38000	0.96	2.53	2137950.012	226
test_file.10	42000	1.15	2.87	2402080.133	226

A part of the table diagrammatically represents in figure 5.1.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .CPP file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

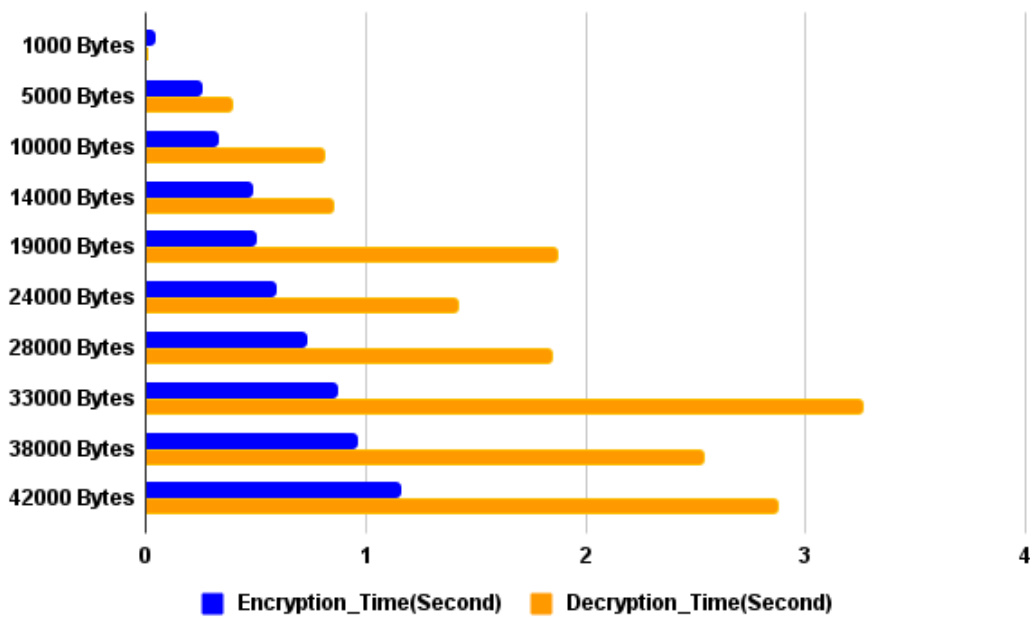


Figure 4.1: Comparison of encryption and decryption time of .CPP files

4.0.1.2 Results of .SYS Files

Table 5.2.1 gives the result of implementing the technique on .SYS files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.04 seconds to 1.19 seconds. The decryption time ranges from 0.001 seconds to 4.55 seconds. The Chi Square value ranges from 373.6363636 to 2399333.46 and the degree of freedom ranges from 226 to 243.

Table 4.2: Encryption and Decryption Time with Chi-Square Analysis

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	525.3846154	243
test_file.02	5000	0.25	0.39	265413.7584	226
test_file.03	10000	0.32	0.81	533724.1181	226
test_file.04	14000	0.48	0.85	795583.5516	226
test_file.05	19000	0.50	1.87	1070268.931	226
test_file.06	14000	0.59	1.42	1333481.988	226
test_file.07	28000	0.73	1.84	1594081.196	226
test_file.08	33000	0.87	3.26	1863587.235	226
test_file.09	38000	0.96	2.53	2137950.012	226
test_file.10	42000	1.15	2.87	2402080.133	226

A part of the table diagrammatically represents in figure 5.2.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .SYS file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

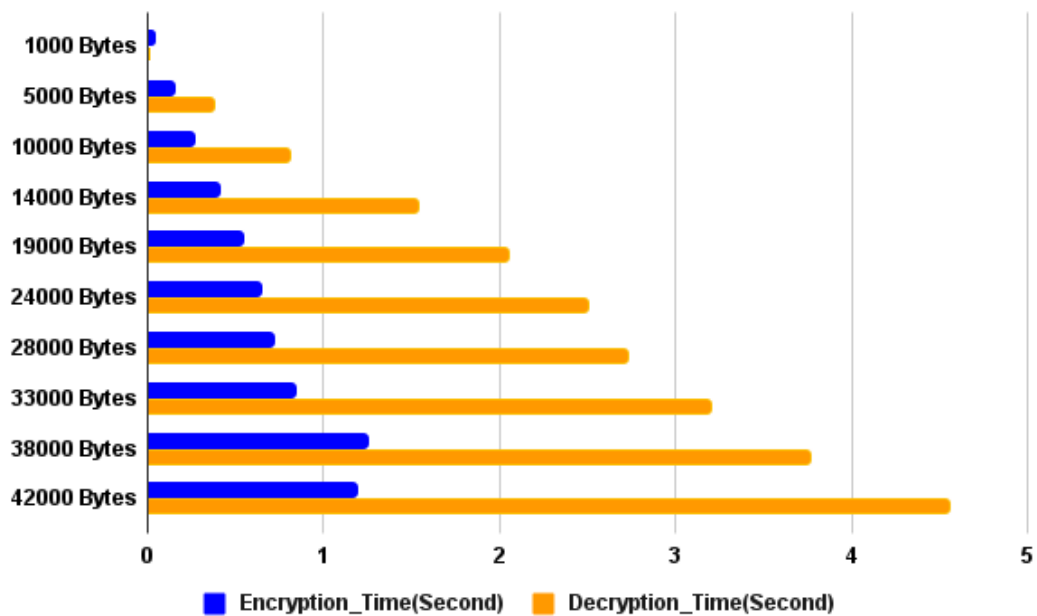


Figure 4.2

4.0.1.3 Results of .TXT Files

Table 5.3.1 gives the result of implementing the technique on .TXT files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.09 seconds to 1.57 seconds. The decryption time ranges from 0.04 seconds to 25.89 seconds. The Chi Square value ranges from 823.3333333 to 1528621.089 and the degree of freedom ranges from 226 to 243.

Table 4.3: Encryption and Decryption Time with Chi-Square Analysis

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	525.3846154	243
test_file.02	5000	0.25	0.39	265413.7584	226
test_file.03	10000	0.32	0.81	533724.1181	226
test_file.04	14000	0.48	0.85	795583.5516	226
test_file.05	19000	0.50	1.87	1070268.931	226
test_file.06	14000	0.59	1.42	1333481.988	226
test_file.07	28000	0.73	1.84	1594081.196	226
test_file.08	33000	0.87	3.26	1863587.235	226
test_file.09	38000	0.96	2.53	2137950.012	226
test_file.10	42000	1.15	2.87	2402080.133	226

A part of the table diagrammatically represents in figure 5.3.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .TXT file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

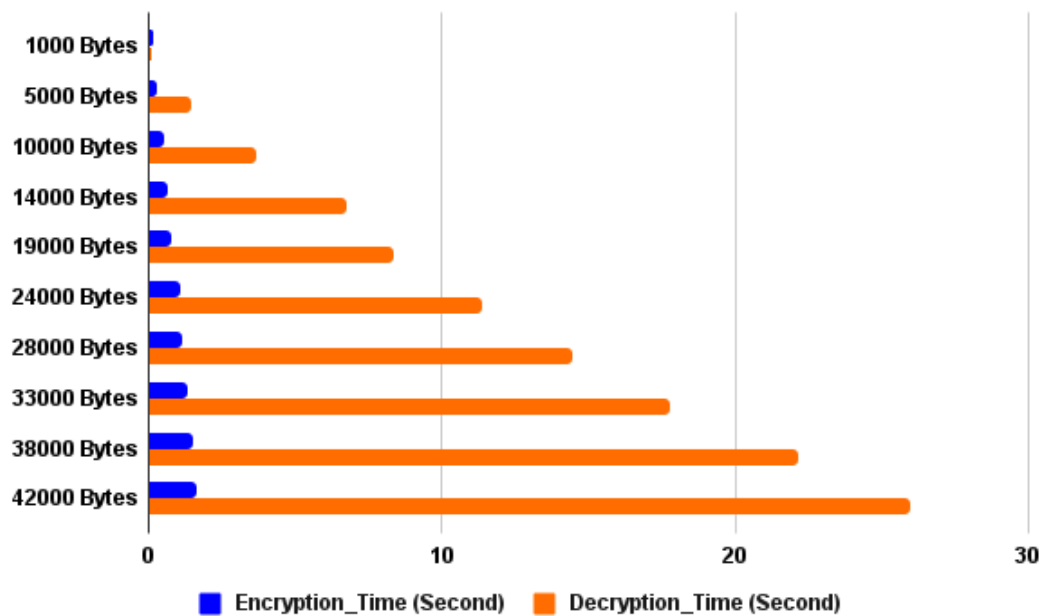


Figure 4.3

4.0.1.4 Results of .DLL Files

Table 5.4.1 gives the result of implementing the technique on .DLL files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.04 seconds to 1.14 seconds. The decryption time ranges from 0.001 seconds to 2.82 seconds. The Chi Square value ranges from 433.3478261 to 2400750.004 and the degree of freedom ranges from 226 to 243.

Table 4.4: Encryption and Decryption Time with Chi-Square Analysis

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	525.3846154	243
test_file.02	5000	0.25	0.39	265413.7584	226
test_file.03	10000	0.32	0.81	533724.1181	226
test_file.04	14000	0.48	0.85	795583.5516	226
test_file.05	19000	0.50	1.87	1070268.931	226
test_file.06	14000	0.59	1.42	1333481.988	226
test_file.07	28000	0.73	1.84	1594081.196	226
test_file.08	33000	0.87	3.26	1863587.235	226
test_file.09	38000	0.96	2.53	2137950.012	226
test_file.10	42000	1.15	2.87	2402080.133	226

A part of the table diagrammatically represents in figure 5.4.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .DLL file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

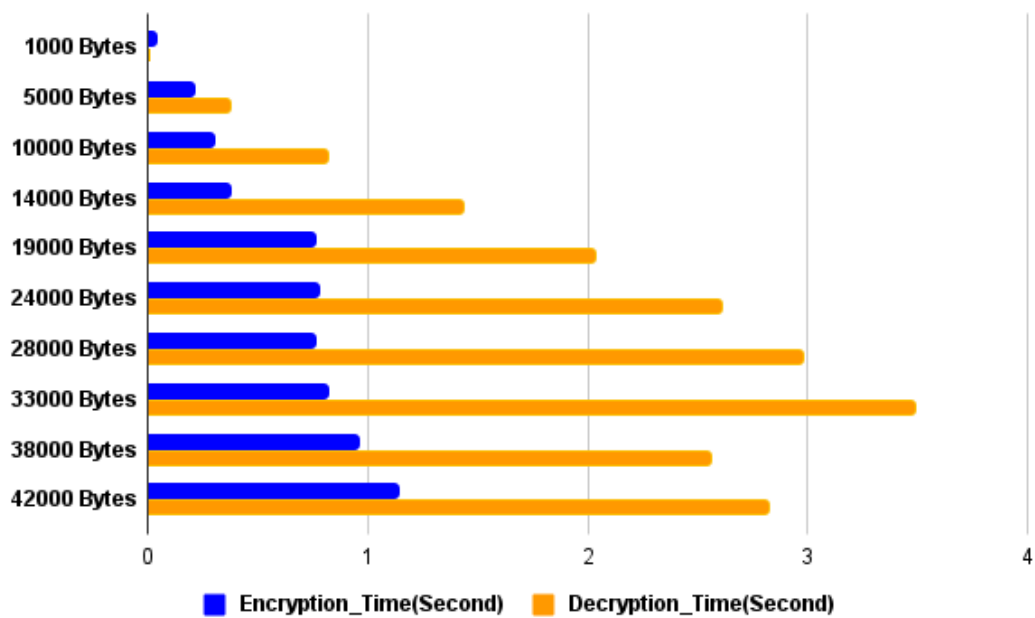


Figure 4.4

4.0.1.5 Results of .EXE Files

Table 5.5.1 gives the result of implementing the technique on .EXE files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0,1 seconds to 1.18 seconds. The decryption time ranges from 0.15 seconds to 4.37 seconds. The Chi Square value ranges from 456.28 to 2404288.876 and the degree of freedom ranges from 226 to 243.

Table 4.5: Encryption and Decryption Time with Chi-Square Analysis

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	525.3846154	243
test_file.02	5000	0.25	0.39	265413.7584	226
test_file.03	10000	0.32	0.81	533724.1181	226
test_file.04	14000	0.48	0.85	795583.5516	226
test_file.05	19000	0.50	1.87	1070268.931	226
test_file.06	14000	0.59	1.42	1333481.988	226
test_file.07	28000	0.73	1.84	1594081.196	226
test_file.08	33000	0.87	3.26	1863587.235	226
test_file.09	38000	0.96	2.53	2137950.012	226
test_file.10	42000	1.15	2.87	2402080.133	226

A part of the table diagrammatically represents in figure 5.5.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .EXE file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

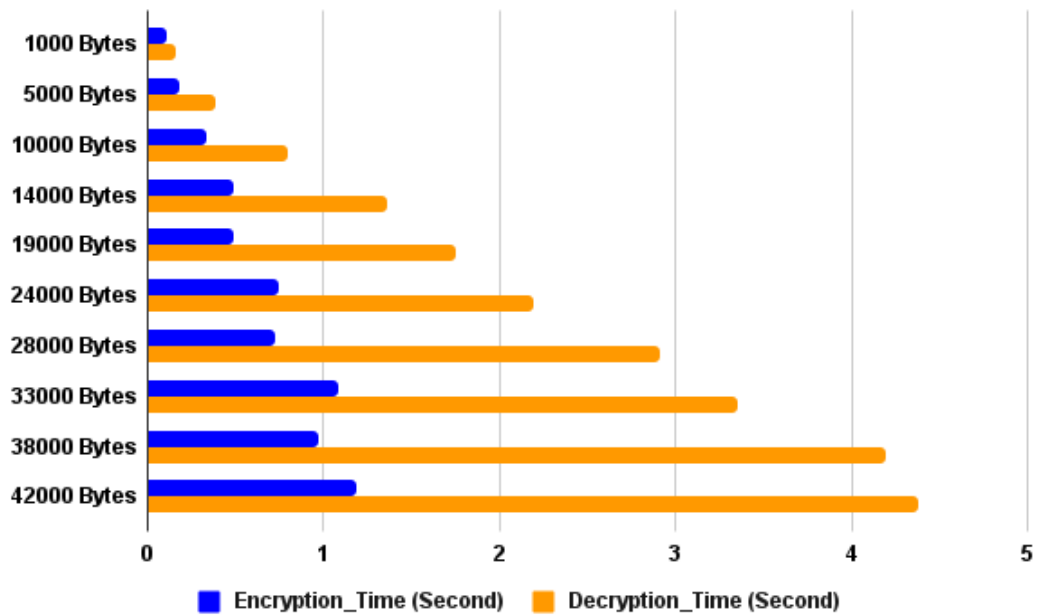


Figure 4.5

4.0.2 RMOPB with Session Key Method

In this section we are implementing the RMOPB algorithm with a session key and then implementing the code with the previous files. The session key is 128 bit long with 8 segment assignments of 16 bits, 16 bits, 16 bits, 16 bits, 16 bits, 16 bits, 16 bits, 16 bits. Details on the making of the session key is given below at section 6.1. Then the file is divided according to the session key. Details on the implementation are given in section 6.2. Finally we test the result of the code and the results are declared in section 7.

4.0.2.1 Generating the Session Key

A 128 bits session key is proposed with 8 segment of size 16 bits

16 bits	16 bits	16 bits	16 bits	16 bits	16 bits	16 bits	16 bits
---------	---------	---------	---------	---------	---------	---------	---------

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

4.0.2.2 Implementation

Each part of the content is converted into binary format. The length of each binary 0 of 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 corresponding to the shifted bit length.

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.

shuffled list : [16, 16, 16, 16, 16, 16, 16, 16]

The process goes through each key-value pair in the dictionary that maps bit keys to lists of binary chunks. :After this the implementation is the same as the implementation of the TE algorithm which was mentioned before.

4.0.3 RMOPB with Session Key

The RMOPB with session key is tested on different file formats with varying file sizes. The results of .txt files are in section 7.1, .sys files are in section 7.2, .cpp files are in section 7.3, .dll files are in section 7.4, and .exe files are in section 7.5.

This testing the files that were used for testing the RMOPB algorithms are used to test RMOPB with Session Key. There is no change of files in both testing. The session key is 128 bit long with 8 segments. Consisting of 16 bits each. The bit division of session key every segment can store minimum 0 to maximum $2^{16} - 1$ bit of block and each block can have minimum 0 to maximum $2^{16} - 1$ bits of data.

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.

4.0.3.1 Results of .CPP Files

Table 7.3.1 gives the result of implementing the technique on .CPP files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.04 seconds to 3.18 seconds. The decryption time ranges from 0.001 seconds to 6.03 seconds. The Chi Square value ranges from 4590 to 10849263.57 and the degree of freedom ranges from 226 to 243.

Table 4.6: Results for .TXT Files for RMOPB with Session Key Technique

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	4590.000	243
test_file.02	5000	0.35	0.54	1206658.792	226
test_file.03	10000	0.82	1.28	2416368.322	226
test_file.04	14000	0.90	2.24	3621501.101	226
test_file.05	19000	1.21	2.56	4820033.960	226
test_file.06	24000	1.60	3.28	6022632.612	226
test_file.07	28000	1.91	4.18	7246052.940	226
test_file.08	33000	2.45	4.65	8437470.098	226
test_file.09	38000	2.56	6.36	9641082.846	226
test_file.10	42000	3.18	6.03	10849263.570	226

A part of the table diagrammatically represents in figure 7.3.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .CPP file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

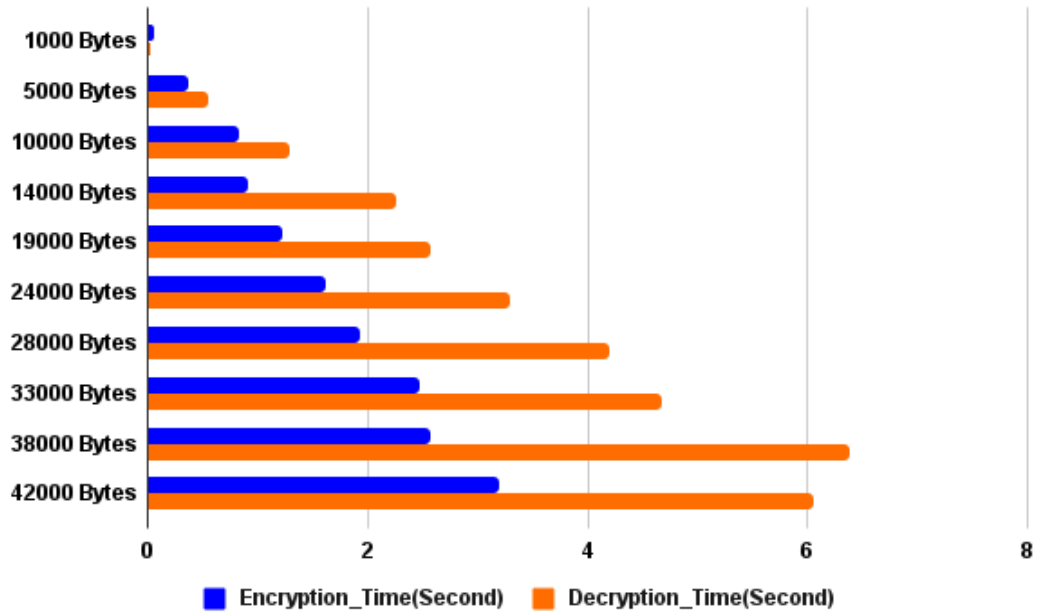


Figure 4.6: Comparison of encryption and decryption time of RMOPB using session key of .CPP files

4.0.3.2 Results of .SYS Files

Table 7.4.1 gives the result of implementing the technique on .SYS files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.04 seconds to 3.15 seconds. The decryption time ranges from 0.001 seconds to 6.65 seconds. The Chi Square value ranges from 4950 to 10854848.98 and the degree of freedom ranges from 226 to 243.

Table 4.7: Results for .SYS Files for RMOPB with Session Key Technique

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.04	0.001	4590.000	243
test_file.02	5000	0.74	0.54	1204627.973	226
test_file.03	10000	0.56	1.14	2410270.711	226
test_file.04	14000	0.92	2.17	3618960.251	226
test_file.05	19000	1.23	2.67	4826635.168	226
test_file.06	24000	1.57	3.51	6032785.847	226
test_file.07	28000	1.86	4.10	7237412.528	226
test_file.08	33000	2.39	5.33	8443563.291	226
test_file.09	38000	2.68	6.09	9645143.624	226
test_file.10	42000	3.15	6.65	10854848.980	226

A part of the table diagrammatically represents in figure 7.4.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .SYS file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

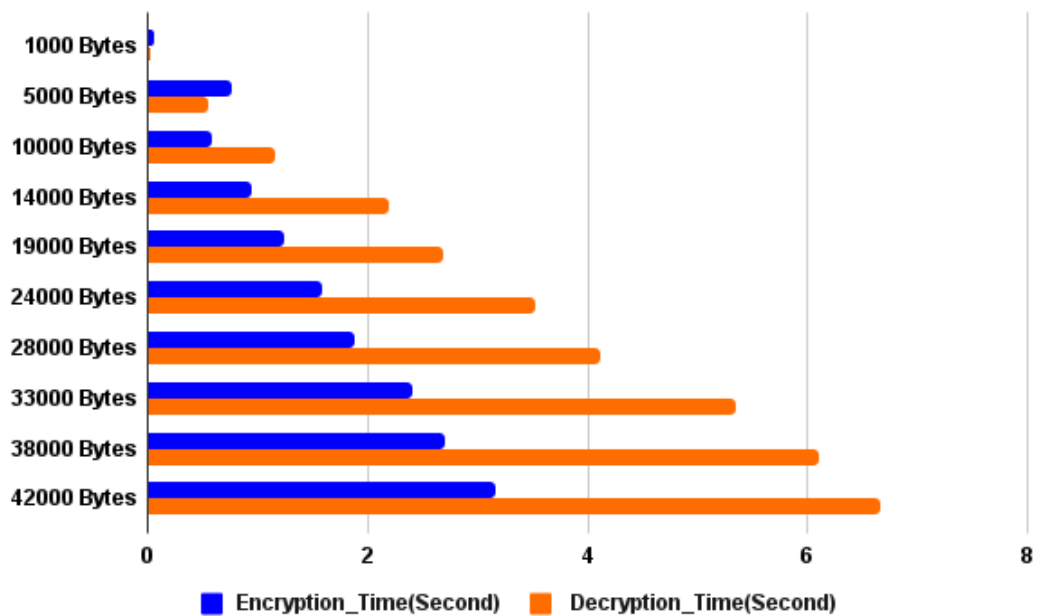


Figure 4.7: Comparison of encryption and decryption time of RMOPB using session key of .SYS files

4.0.3.3 Results of .TXT Files

Table 7.1.1 gives the result of implementing the technique on .TXT files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.14 seconds to 1.99 seconds. The decryption time ranges from 0.1 seconds to 35.08 seconds. The Chi Square value ranges from 4590 to 10863994.96 and the degree of freedom ranges from 226 to 243.

Table 4.8: Results for .TXT Files for RMOPB with Session Key Technique

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.14	0.10	4590.000	243
test_file.02	5000	0.28	2.29	1208184.161	226
test_file.03	10000	0.57	5.68	2417386.094	226
test_file.04	14000	0.73	9.53	3629134.389	226
test_file.05	19000	0.98	14.17	4823587.329	226
test_file.06	24000	1.32	19.88	6039394.663	226
test_file.07	28000	1.32	24.06	7240460.904	226
test_file.08	33000	1.65	24.52	8451694.420	226
test_file.09	38000	1.98	34.60	9665473.289	226
test_file.10	42000	1.99	35.08	10863994.960	226

A part of the table diagrammatically represents in figure 7.1.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .TXT file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

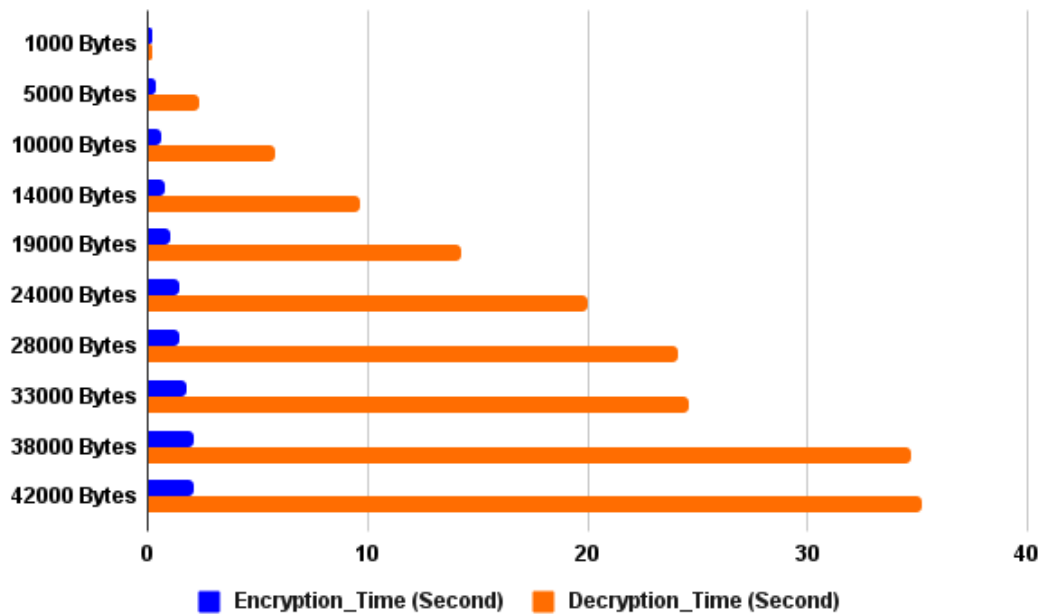


Figure 4.8: Comparison of encryption and decryption time of RMOPB using session key of .TXT files

4.0.3.4 Results of .DLL Files

Table 7.5.1 gives the result of implementing the technique on .DLL files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.03 seconds to 3.28 seconds. The decryption time ranges from 0.001 seconds to 6.25 seconds. The Chi Square value ranges from 4590 to 10845203.27 and the degree of freedom ranges from 226 to 243.

Table 4.9: Results for .DLL Files for RMOPB with Session Key Technique

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.03	0.001	4590.000	243
test_file.02	5000	0.67	0.54	1206150.765	226
test_file.03	10000	0.57	1.31	2415859.597	226
test_file.04	14000	0.85	1.89	3621501.101	226
test_file.05	19000	1.15	2.54	4816483.221	226
test_file.06	24000	1.62	3.43	6034310.314	226
test_file.07	28000	1.93	4.56	7233857.718	226
test_file.08	33000	2.40	5.50	8450677.599	226
test_file.09	38000	2.89	5.28	9653270.336	226
test_file.10	42000	3.28	6.25	10845203.270	226

A part of the table diagrammatically represents in figure 7.5.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .DLL file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

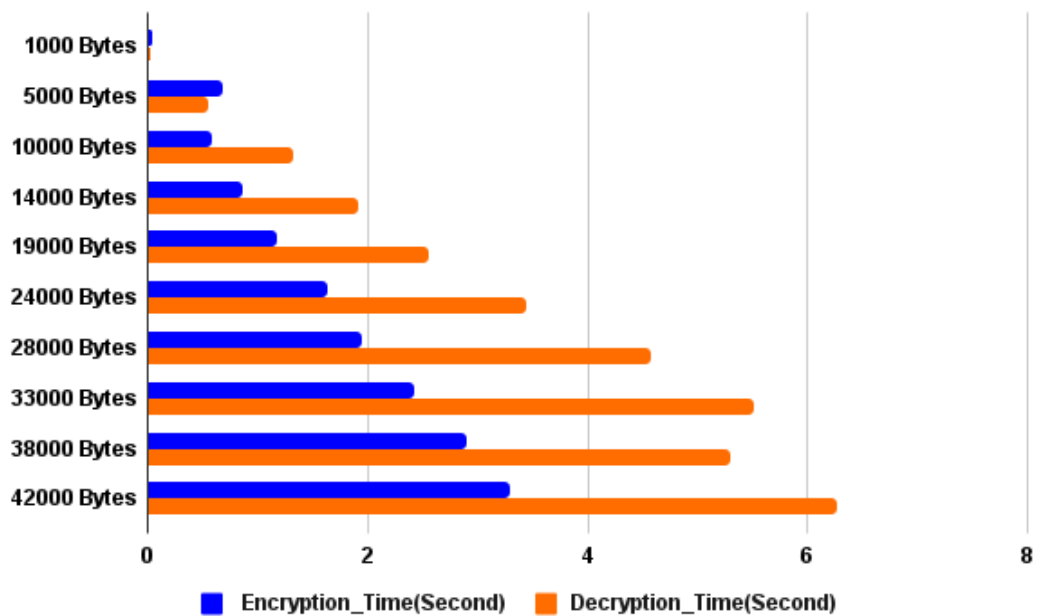


Figure 4.9: Comparison of encryption and decryption time of RMOPB using session key of .DLL files

4.0.3.5 Results of .EXE Files

Table 7.2.1 gives the result of implementing the technique on .EXE files. Ten files have been considered. Their sizes range from 1000 bytes to 42000 bytes. The encryption time ranges from 0.15 seconds to 2.86 seconds. The decryption time ranges from 0.3 seconds to 6.51 seconds. The Chi Square value ranges from 4590 to 10856372.77 and the degree of freedom ranges from 226 to 243.

Table 4.10: Results for .TXT Files for RMOPB with Session Key Technique

Source File	Size (Bytes)	Encryption Time (s)	Decryption Time (s)	Chi-Square Value	Degree of Freedom
test_file.01	1000	0.15	0.30	4590.000	243
test_file.02	5000	0.28	0.59	1207675.490	226
test_file.03	10000	0.59	1.15	2414334.067	226
test_file.04	14000	0.92	1.81	3617944.412	226
test_file.05	19000	1.20	2.56	4829684.940	226
test_file.06	24000	1.56	3.28	6025169.310	226
test_file.07	28000	1.81	4.24	7239444.635	226
test_file.08	33000	2.18	4.61	8451694.420	226
test_file.09	38000	2.59	5.65	9659878.349	226
test_file.10	42000	2.86	6.51	10856372.770	226

A part of the table diagrammatically represents in figure 7.2.2, where one graphical relationship is established between the source size and the encryption time and decryption time for the .EXE file. From the figure, it can be interpreted that there is a tendency that the encryption time changes almost linearly with the size of the source file.

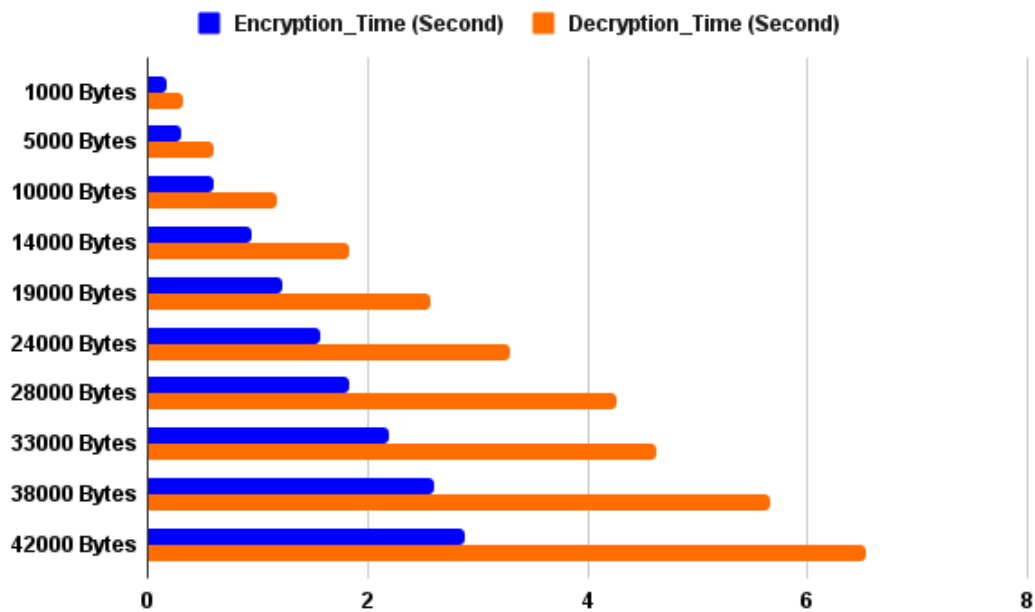


Figure 4.10: Comparison of encryption and decryption time of RMOPB using session key of .EXE files

4.0.4 Comparison between Encryption and Decryption Time of RMOPB and RMOPB using Session Key

In this section we will compare the encryption time and decryption time of various files using the RMOPB algorithm with the files using the RMOPB algorithm with session key.

4.0.4.1 Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .TXT files

We can see from the figure 8.1.1 that RMOPB with a session key takes more time in encryption than RMOPB technique.

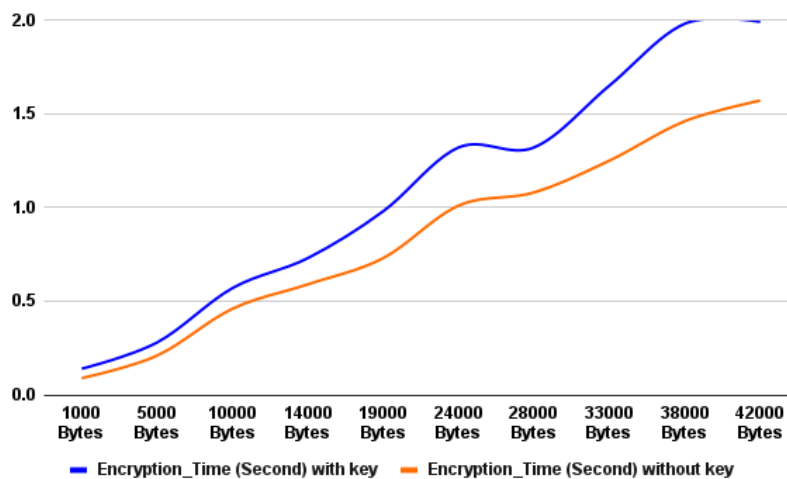


Figure 4.11

4.0.4.2 Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .TXT files

We can see from the figure 8.2.1 that RMOPB with session key has more time in decryption than RMOPB technique.

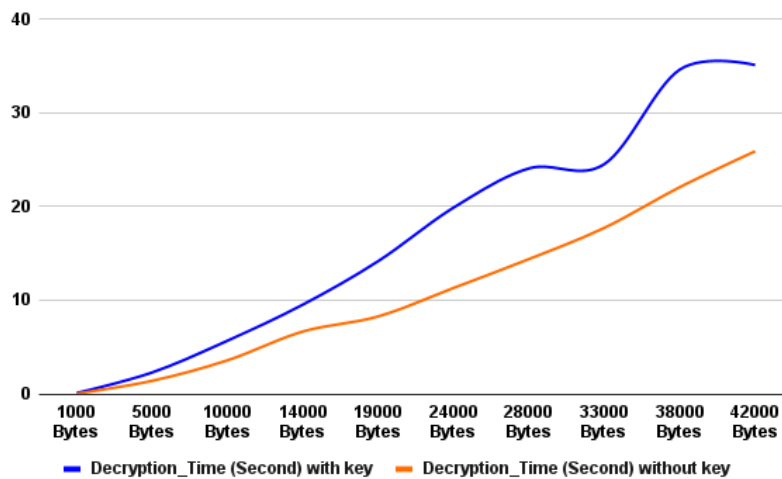


Figure 4.12

4.0.4.3 Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .EXE files

We can see from the figure 8.3.1 that RMOPB with session keys takes more time in encryption than RMOPB technique.

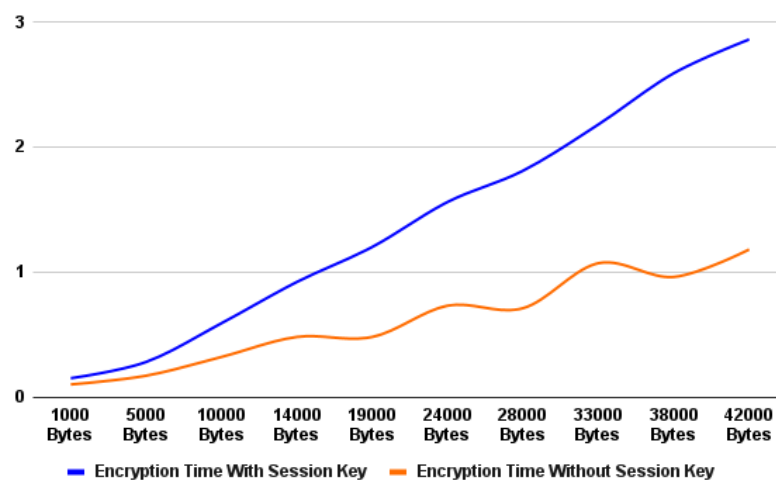


Figure 4.13

4.0.4.4 Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .EXE files

We can see from the figure 8.4.1 that RMOPB with session key has more time in decryption than RMOPB technique.

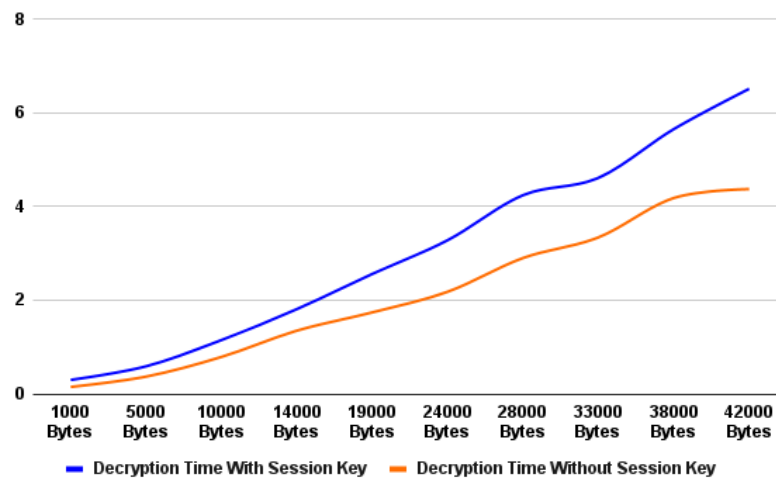


Figure 4.14

4.0.4.5 Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .CPP files

We can see from the figure 8.5.1 that RMOPB with session key takes more time in encryption than RMOPB technique.

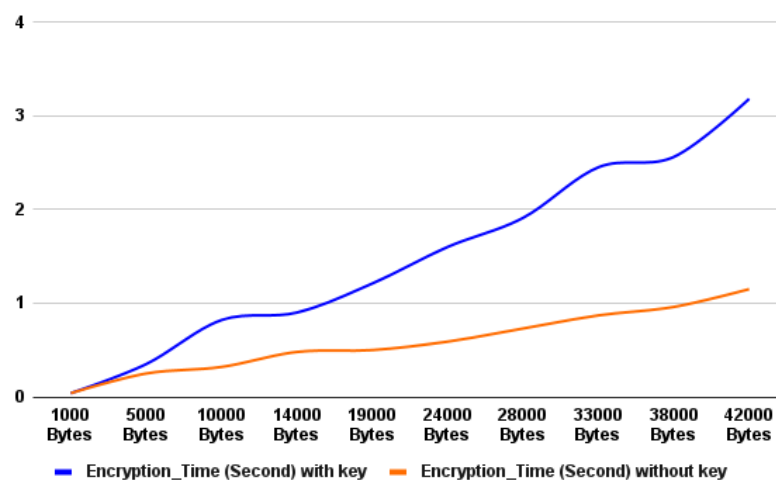


Figure 4.15

4.0.4.6 Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .CPP files

We can see from the figure 8.6.1 that a RMOPB with a session key has more time in decryption than RMOPB technique.

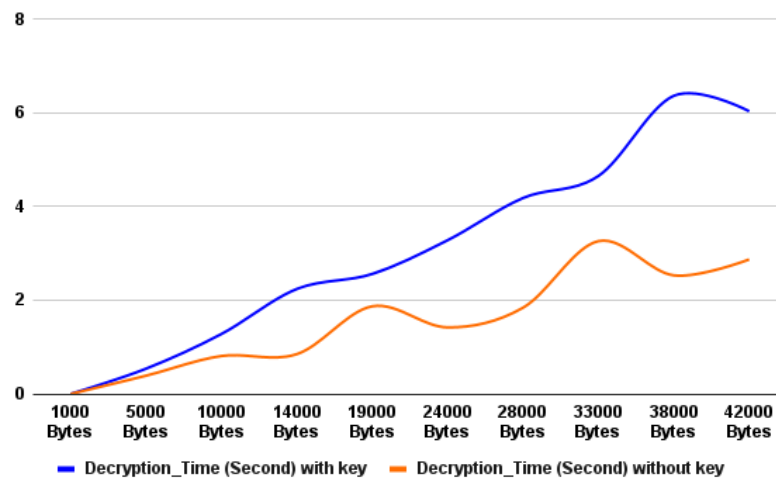


Figure 4.16

4.0.4.7 Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .SYS files

We can see from the figure 8.7.1 that RMOPB with a session key takes more time in encryption than RMOPB technique.

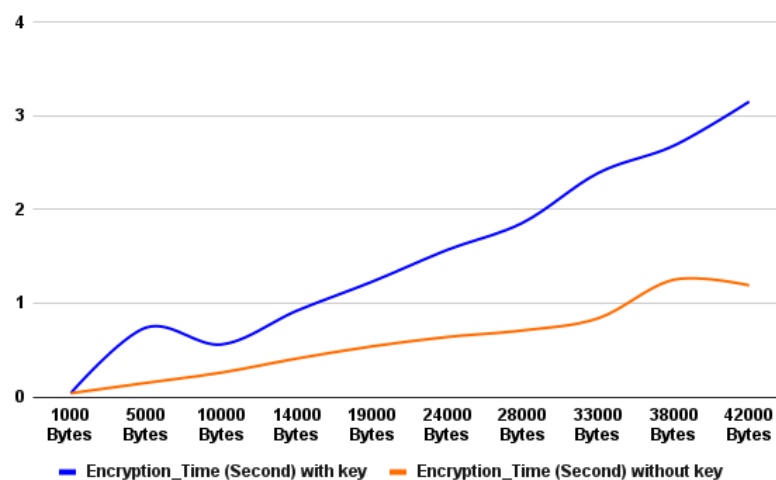


Figure 4.17

4.0.4.8 Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .SYS files

We can see from the figure 8.8.1 that a RMOPB with a session key has more time in decryption than RMOPB technique.

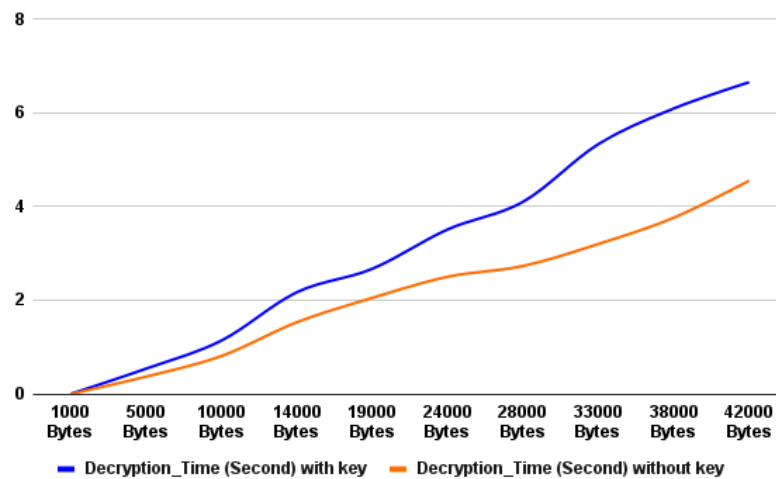


Figure 4.18

4.0.4.9 Comparison between Encryption Time of RMOPB and RMOPB using Session Key of .DLL files

We can see from the figure 8.9.1. that RMOPB with a session key takes more time in encryption than RMOPB technique.

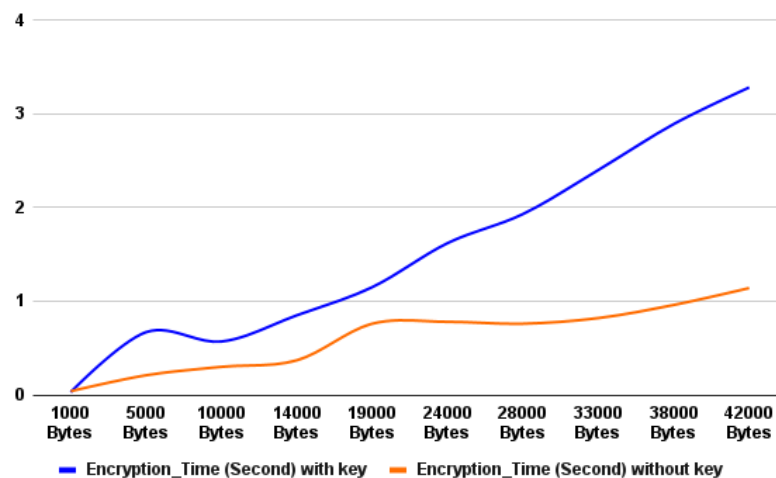


Figure 4.19

4.0.4.10 Comparison between Decryption Time of RMOPB and RMOPB using Session Key of .DLL files

We can see from the figure 8.10.1 that RMOPB with session key has more time in decryption than RMOPB technique.

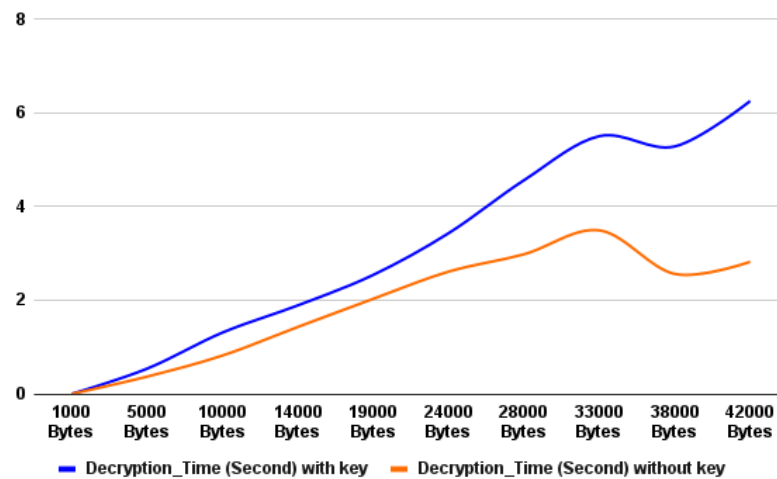


Figure 4.20

4.0.5 NIST Test Report

The National Institute of Standards and Technology (NIST) has developed a suite of statistical tests specifically designed to assess the randomness of cryptographic algorithms. These tests are widely used to verify whether a cryptographic output exhibits properties similar to a truly random sequence, which is crucial for ensuring data security. Implementing NIST tests on the Triangulation Encryption Algorithm ensures that the encryption mechanism produces unpredictable and secure outputs, contributing to a stronger cryptographic system.

Table 4.11: Status for Proportion of Passing and Uniformity of Distribution

Test No.	Test Name	Expected Proportion	Observed Proportion	Status for Proportion of Passing	P-value of P-values	Uniform / Non-uniform Status
1	Frequency_test	0.91022	1.0000	Success	1	1
2	Block_frequency_test	0.91022	1.0000	Success	1	1
3	Runs_test	0.91022	1.0000	Success	1	1
4	Longest_run_ones	0.91022	1.0000	Success	1	1
5	Rank_test	0.91022	0.0000	Unsuccess	1	1
6	Discrete_fourier_transform_test	0.91022	1.0000	Success	1	1
7	Cumulative_sums_test	0.93359	1.0000	Success	1	1
8	Approximate_entropy_test	0.91022	0.0000	Unsuccess	1	1
9	Serial_test	0.93359	1.0000	Success	1	1
10	Linear_complexity_test	0.91022	1.0000	Success	1	1
11	Non_overlapping_template_matching_test	0.91022	1.0000	Success	1	1
12	Overlapping_template_matching_test	0.91022	1.0000	Success	1	1
13	Maurers_universal_test	0.91022	0.0000	Unsuccess	1	1
14	Random_excursions_test	0.96179	0.0000	Unsuccess	1	1
15	Random_excursions_variant_test	0.9712	1.0000	Success	1	1

Chapter 5.

Conclusion and Future Works

In conclusion, the Encryption Through Recursive Modulo-2 Operation of Paired Bits of Streams (RMOPB) technique offers a significant advancement over traditional methods like RSA, with its unique geometric transformations providing enhanced security and resistance to common cryptographic attacks. RMOPB ensures data integrity and offers a versatile, scalable solution for various applications, making it a promising candidate for addressing evolving cyber threats. The future scope of RMOPB is vast, including securing real-time voice communications, protecting images through encryption and digital watermarking, and ensuring the confidentiality of video content in streaming services and video conferencing. As research and development continue, RMOPB is poised to become a standard in cryptographic practices, offering robust solutions for securely encrypting voice, images, and videos, thereby enhancing overall data security and privacy.

RMOPB can be used to encrypt text messages, ensuring that communications remain private and protected from unauthorized access. The RMOPB (Recursive Modulo-2 Operation of Paired Bits of Streams) 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, RMOPB 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.

Bibliography

- [1] Bruce Schneier, *Applied Cryptography. Second Edition, Protocols, Algorithms, and Source Code in C*, John Wiley & Sons Inc., 1996.
- [2] L.M. Adleman, C. Pomerance, and R. S. Rumeley, “On Distinguishing Prime Numbers from Composite Numbers,” *Annals of Mathematics*, vol. 117, no. 1, 1983, pp. 173–206.
- [3] H. Fiestel, “Cryptography and Computer Privacy,” *Scientific American*, vol. 228, no. 5, May 1973, pp. 15–23.
- [4] G. B. Agnew, “Random Sources of Cryptographic Systems,” *Advances in Cryptology: EUROCRYPT ’87 Proceedings*, Springer-Verlag, 1988, pp. 77–81.
- [5] S. G. Aki and H. Meijer, “A Fast Pseudo-Random Permutation Generator with Applications to Cryptology,” *Advances in Cryptology: EUROCRYPT ’84 Proceedings*, Springer-Verlag, 1985, pp. 269–275.
- [6] W. Alexi, B.Z. Chor, O. Goldreich, and C. P. Schnorr, “RSA and Rabin Functions: Certain Parts are as Hard as the Whole,” *SIAM Journal on Computing*, vol. 17, no. 2, Apr 1988, pp. 194–209.
- [7] W. Alexi, B.Z. Chor, O. Goldreich, and C. P. Schnorr, “RSA and Rabin Functions: Certain Parts are as Hard as the Whole,” *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science*, 1984, pp. 449–457.
- [8] R. J. Anderson, “Why Cryptosystems Fail,” *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 215–227.
- [9] R. J. Anderson, “Why Cryptosystems Fail,” *Communications of the ACM*, vol. 37, no. 11, Nov 1994, pp. 32–40.
- [10] J. Anderson and R. Needham, “Robustness of Principles for Public Key Protocols,” *Advances in Cryptology: EUROCRYPT ’95 Proceedings*, Springer-Verlag, 1995.

