Parallel Implementation and Analysis of Encryption Algorithms

By:
Rami M. N. Aldahdooh

Supervisor
Dr. Ahmed Y. Mahmoud
Associate Professor

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Computing and Information Systems

April – 2018
التحليل والتطبيق المتوازي لخوارزميات التشفير

إعداد الباحث:
رامي محمد نصيف الدحدوح

إشراف:
د. أحمد يحيى محمود

أستاذ مشارك

قدمت هذه الرسالة استناداً لمتطلبات الحصول على درجة الماجستير في الحوسبة ونظم المعلومات

أبريل 2018
Abstract

This research addresses the issue of improving the execution of robust cryptographic algorithms. The proliferation of digital documents and commercial transactions have shown an urgent need for the presence of encryption algorithms. Robust cryptographic algorithms are very time consuming; efficient implementation is necessary for an online application. A promising approach has seen in a parallel implementation. This research presents a parallel implementation and analyze various encryption algorithms (Advanced Encryption Standard AES, Blowfish, Twofish, Data Encryption Standard DES, Triple DES, and Serpent). The parallel processing is used to enhance efficiency. The experiments show that the parallel implementation is significantly better than the sequential implementation. The algorithms are examined based on encryption time and speedup. The results show that Triple DES is more efficient algorithm among the examined algorithms.

Keywords:
Block cipher, Parallel encryption implementation, AES, Blowfish, Twofish, DES, Triple DES, Serpent.
يتناول هذا البحث مسألة تحسين تنفيذ خوارزميات التشفير القوية. أظهر انتشار الوثائق الرقمية والمعاملات التجارية حاجة ملحة لوجود خوارزميات التشفير. أن خوارزميات التشفير القوية تستهلك وقتا طويلا جدا، حيث أن التنفيذ الفعال لخوارزميات التشفير ضروري لتطبيقات الإنترنت.

المعالجة المتوازية من الأساليب الواعدة في تطبيق خوارزميات التشفير لتحسين الوقت. في هذا البحث، قمنا بتحليل وتطبيق المعالجة المتوازية للعديد من خوارزميات التشفير مثل AES، Blowfish، Twofish، DES، Triple DES، Serpent. تم استخدام المعالجة المتوازية لتعزيز وتحسين كفاءة تنفيذ خوارزميات التشفير. بينت التجارب أن التنفيذ المتوازي أفضل بكثير من التنفيذ المتسلسل حيث تم اختبار الخوارزميات على أساس وقت التشفير والسرعة. أظهرت النتائج أن أفضل خوارزمية من بين الخوارزميات التي تم اختبارها هي Triple DES.
Declaration

I “Rami Aldahdooh” confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form e.g., ideas, equations, figures, text, tables, programs etc. are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: ................................................

Date: ...................................................
Dedication

This thesis is dedicated to:

The sake of Allah, my Creator and my Master,

My great teacher and messenger, Mohammed (May Allah bless

and grant him), who taught us the purpose of life,

My homeland Palestine,

Al-Azhar University; my second magnificent home;

My great parents, who never stop giving of themselves in countless ways,

My dearest wife, who leads me through the valley of darkness

with light of hope and support,

My beloved brothers and sisters,

My beloved kids, whom I can't force myself

to stop loving. To all my family, the symbol of love and giving,

My friends who encourage and support me,

All the people in my life who touch my heart,

I dedicate this research.
Acknowledgment

I would like to express my appreciation for all the efforts of my supervisor, Assoc. Prof. Dr. Ahmed Y. Mahmoud, for all the help, suggestions and guidance that he offered to me. I am thankful to him for his advice and the time that he spent to make this work better. I would also like to thank my Family for their support and their willingness to help at any time. I am truly grateful.
# Table of Contents

Abstract .......................................................................................................................... ii
الملخص ......................................................................................................................... iii
Declaration ...................................................................................................................... iv
Dedication ....................................................................................................................... v
Acknowledgment ........................................................................................................ vi
Table of Contents ........................................................................................................ vii
List of Abbreviations ...................................................................................................... viii
List of Tables ................................................................................................................ ix
List of Figures ................................................................................................................ x
Introduction .................................................................................................................. 2
  1.1 Statement of the Problem ....................................................................................... 3
  1.2 Objectives ............................................................................................................. 4
  1.3 Importance of the Research ................................................................................ 4
  1.4 Scope and the Limitations of the Research .......................................................... 5
  1.5 Methodology ....................................................................................................... 5
  1.6 Thesis Structure .................................................................................................. 6
Background ................................................................................................................... 8
  2.1 Definition ............................................................................................................ 9
  2.2 Principles of encryption ...................................................................................... 11
  2.3 Block ciphers ..................................................................................................... 13
  2.4 Implemented block ciphers ................................................................................ 15
    2.3.1. Advance Encryption Standard (AES) ............................................................. 16
    2.3.2. Blowfish .................................................................................................... 18
    2.3.3. Twofish .................................................................................................... 20
    2.3.4. Serpent ..................................................................................................... 21
    2.3.5. Data Encryption Standard DES ................................................................. 22
    2.3.6. Triple DES ............................................................................................... 23
Literature Review .......................................................................................................... 26
Proposed Parallel System ............................................................................................ 31
  4.1. Parallel Programming Languages ...................................................................... 31
  4.2. Designing Parallel Encryption Algorithms ....................................................... 32
    4.3.1. Prototyping ............................................................................................... 32
    4.3.2. Parallel Implementation ............................................................................ 33
    4.3.3. Performance Benchmarking ..................................................................... 34
Experimental Results and Discussion ........................................................................ 37
  5.1. Algorithm(s) to be parallelized ........................................................................ 37
  5.2. Experimental Setup .......................................................................................... 37
  5.3. Implementation and Benchmark ...................................................................... 38
Conclusion and Future Work ...................................................................................... 59
References ...................................................................................................................... 61
**List of Abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td><strong>Triple DES</strong></td>
<td>Triple Data Encryption Standard</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td><strong>HTTPs</strong></td>
<td>Hypertext Transfer Protocol Secure</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>CIA</td>
<td>Confidentiality, Integrity and Availability;</td>
</tr>
</tbody>
</table>
List of Tables

Table 1. DES Implementation run time as Sequential and Parallel using Multi Cores ................ 40
Table 2. DES Implementation Speedup as Sequential and Parallel using Multi Cores ............... 41
Table 3. DES Implementation Efficiency as Sequential and Parallel using Multi Cores ............ 42
Table 4. 3DES Implementation Runtime as Sequential and Parallel using Multi Cores .......... 43
Table 5. 3DES Implementation Speedup as Sequential and Parallel using Multi Cores .......... 44
Table 6. 3DES Implementation Efficiency as Sequential and Parallel using Multi Cores .......... 45
Table 7. AES Implementation Runtime as Sequential and Parallel using Multi Cores ............... 46
Table 8. AES Implementation Speedup as Sequential and Parallel using Multi Cores ............... 47
Table 9. AES Implementation Efficiency as Sequential and Parallel using Multi Cores .......... 48
Table 10. Blowfish Implementation Runtime as Sequential and Parallel using Multi Cores ....... 49
Table 11. Blowfish Implementation Speedup as Sequential and Parallel using Multi Cores ...... 50
Table 12. Blowfish Implementation Efficiency as Sequential and Parallel using Multi Cores ...... 51
Table 13. Twofish Implementation Runtime as Sequential and Parallel using Multi Cores ...... 52
Table 14. Twofish Implementation Speedup as Sequential and Parallel using Multi Cores ...... 53
Table 15. Twofish Implementation Efficiency as Sequential and Parallel using Multi Cores ...... 54
Table 16. Serpent Implementation Runtime as Sequential and Parallel using Multi Cores ........ 55
Table 17. Serpent Implementation Speedup as Sequential and Parallel using Multi Cores .......... 56
Table 18. Serpent Implementation Efficiency as Sequential and Parallel using Multi Cores ...... 57
List of Figures

Figure 1. Feistel Encryption and Decryption (16 rounds). (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017) ................................................................. 15

Figure 2. AES Encryption Process. (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017) ........................................................................ 18

Figure 3: Blowfish Encryption and Decryption (Stallings, Cryptography and Network Security: Principles and Practice, 6th Edition, 2014) ................................................................. 19

Figure 4 Twofish algorithm (Schneier, Products that Use Twofish, 2013b) ........................................... 20

Figure 5 Serpent Encryption Algorithm (Naeemabadi, 2015) ........................................................... 22

Figure 6 General Depiction of DES Encryption Algorithm (Stallings, Cryptography and Network Security: Principles and Practice, 6th Edition, 2014) .................................................. 23

Figure 7 3DES encryption and decryption algorithm (Mohammad, 2017) ........................................... 24

Figure 10. Sequential Implementation Tool Figure 11. Parallel Implementation Tool ..... 38

Figure 12. Sequential CPU utilization Figure 13. Parallel CPU utilization ........................................ 39

Figure 16. DES Implementation run time as Sequential and Parallel using Multi Cores.................. 40

Figure 17. DES Implementation Speedup as Sequential and Parallel using Multi Cores .............. 41

Figure 18. DES Implementation Efficiency as Sequential and Parallel using Multi Cores .......... 42

Figure 19. 3DES Implementation Runtime as Sequential and Parallel using Multi Cores .......... 43

Figure 20. 3DES Implementation Speedup as Sequential and Parallel using Multi Cores .......... 44

Figure 21. 3DES Implementation Efficiency as Sequential and Parallel using Multi Cores .......... 45

Figure 22. AES Implementation Runtime as Sequential and Parallel using Multi Cores .......... 46

Figure 23. AES Implementation Speedup as Sequential and Parallel using Multi Cores .......... 47

Figure 24. AES Implementation Efficiency as Sequential and Parallel using Multi Cores .......... 48

Figure 25. Blowfish Implementation Runtime as Sequential and Parallel using Multi Cores .... 49

Figure 26. Blowfish Implementation Speedup as Sequential and Parallel using Multi Cores .... 50

Figure 27. Blowfish Implementation Efficiency as Sequential and Parallel using Multi Cores ... 51

Figure 28. Twofish Implementation Runtime as Sequential and Parallel using Multi Cores .... 52

Figure 29. Twofish Implementation Speedup as Sequential and Parallel using Multi Cores .... 53

Figure 30. Twofish Implementation Efficiency as Sequential and Parallel using Multi Cores..... 54

Figure 31. Serpent Implementation Runtime as Sequential and Parallel using Multi Cores ....... 55

Figure 32. Serpent Implementation Speedup as Sequential and Parallel using Multi Cores....... 56

Figure 33. Serpent Implementation Efficiency as Sequential and Parallel using Multi Cores...... 57
Chapter 1

Introduction
Introduction

Nowadays, Desktop PC contains a multi-core processor. This parallel equipment makes the product creators reevaluate the product configuration to get the most conceivable computational power from these effective processors. This should be possible by using parallel programming strategies and utilizing them to make the simultaneous execution of encryption algorithms. The absolute most regularly executed calculations by PC clients these days are cryptographic calculations, which are utilized to scramble and decode information keeping in mind the end goal to send it securely and safely over a danger domain like the web.

The research concentrates on improving the execution of the cryptographic calculations with a specific end goal to accelerate the encryption procedure and utilize the new multi-core processors productively. The approach received to improve the execution is the utilization of parallel programming to accelerate the solid cryptographic calculations. The parallel dialect innovation that is utilizations string as a primary executing unit and works superbly in a multi-core PC condition.

By improving the execution time of these ciphers algorithms, they can be effortlessly designed to give a more elevated amount of security by utilizing longer keys or expanding the calculations inside them. This will improve the security of each web client everywhere throughout the world. (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017)

Encryption is a part of our everyday life although mostly invisible. It is used to prevent eavesdropping on our communications over cell phone calls and Internet, securing network connections, making e-commerce and e-banking possible and generally hiding information from unwanted eyes. Through the history of civilization, new encryption functions have been constructed as old ones have been broken (Mollin, 2005).

Better implementations, optimized for specific processor architectures, can bring performance improvements of order of magnitude thus giving higher throughput and, in case throughput bottleneck being elsewhere, reduce CPU usage (Potlapally, Ravi, & Raghunathan, 2003). Less CPU usage also results to smaller power usage. Most commonly
used ciphers have hardware implementations available and today even on consumer processors (Akdemir & et, al., 2010)

However, software implementations remain important for less commonly used algorithms and for new ciphers in newer encryption standards that will become more used in the future but yet do not have gained enough popularity for wide hardware implementations and acceleration.

1.1 Statement of the Problem

Cryptographic operations are broadly used to keep up a level of security between gatherings interfacing over the web. Trust is fundamental, particularly in the web-based business field, where occasionally an entire business is directed from the web. Our Purpose is increasing the speed of these methods that protect our data as much as we can by using parallelism for improving cryptographic algorithms.

Nowadays, multi-core techniques are adopted to achieve the goal with an increased number of cores inside one processor. However, to get the most benefit of these processors, the application that is being executed should be designed in a concurrent way; this is the need of explicit parallel techniques and high-level parallel programming languages to make the mission easier to the programmer in order to develop multi-threaded or multi-process parallel applications (Grama, Karypis, Kumar, & Gupta, 2003), (Stephen, Olukotun, & Hofstee, 2009).

“Parallel Programming is programming in a language that allows you to explicitly indicate how different portions of the computation may be executed concurrently by different processors” (Quinn, 2004). A lot of techniques (libraries, application program interface API, parallel models and languages) have been evolved to assist the development of parallel programs.

Modern encryption algorithms usually rely heavily on mathematics to produce the cipher text. Consequently, those algorithms can contain heavy mathematical calculations repetition on the text data segments (bytes, blocks etc.). Therefore, the single program multiple data (SPMD) mode can be the most suitable mode to parallelize these algorithms.
1.2 Objectives

The general point of the review is to accelerate the encryption process, and to expand the security level of the web end-client. This can be done by exploiting parallel programming techniques and providing a parallel version of strong cryptographic algorithm which are used nowadays in the security protocols. This general goal can be identified by a few points:

✓ Investigate the security level of cryptographic algorithms.
✓ Discover the best cryptographic algorithms.
✓ Develop a parallel version of the (AES, Blowfish, Twofish, DES, Triple DES, Serpent) algorithms.
✓ Adjust the performance of the parallel algorithm.
✓ Evaluate the performance of the parallel algorithm across multi-core parallel devices.

1.3 Importance of the Research

This research aims to increase the speed of the former encryption algorithms especially to protect big data as much as possible. In this research, we use parallelism for encryption to emphasize the benefit of it; this is achieved through the implementation and improving of cryptographic algorithms.

On the other hand, this approach may not reach the same performance enhancement level of a dedicated hardware; however, it addresses the first approach limitations by providing a relatively cheap alternative to the dedicated hardware, can be applied even to normal personal computers. Furthermore, the scalability is much easier, especially when the parallel algorithm is well-designed. Moreover, a typical personal computer system contains a multi-core processor which has eight cores or more, thus, it is a wise decision to make use of all of them to enhance the overall performance.

However, when using longer keys, we get higher security and slower speed. This leads to the need to use parallel computing to increase the security level without adversely affecting
the performance. However, performance improvement can be necessary for Internet servers that provide secure channels with the client since the bottleneck there may be the encryption/decryption of the large file stream.

1.4 Scope and the Limitations of the Research

One approach used to enhance the performance of commonly-used algorithms is to devise a dedicated piece of hardware to solve the problem. However, there are several limitations that can be faced, such as:

- This is a cost-effective approach that can be adopted by the big companies and organizations, but not by normal computer users.
- Usually there is a scalability limitation of the dedicated hardware, so after a period of time another device will be required.

On the other hand, another approach can be used which exploits parallel programming to solve the problem. This approach may not reach the same performance enhancement level of a dedicated hardware; however, it addresses the first approach limitations by providing a relatively cheap alternative to the dedicated hardware, can be applied even to normal personal computers. Furthermore, the scalability is much easier, especially when the parallel algorithm is well-designed.

Moreover, a typical personal computer system contains a multi-core processor which has eight cores or more, thus, it is a wise decision to make use of all of them to enhance the overall performance.

However, Encryption of Multimedia Large files are obtaining a slower performance when the longer key was used for more security. This leads to the need to use parallel computing to increase the security level without adversely affecting the performance.

1.5 Methodology

To achieve the objectives of the research, the following methodology is followed, we have reviewed several articles, the review presents the most frequently used cryptographic algorithms, a full description for the revised algorithms has been introduced. The steps of
designing a parallel program from sequential code are highlighted. An investigation and analysis of sequential algorithm are introduced.

By using RAD Studio to develop multi-device applications that run on other platforms. The RTL (run-time library) provides the Parallel Programming Library (PPL), giving applications the ability to have tasks running in parallel taking advantage of working across multiple CPU devices and computers. We develop, implement, test and evaluate a parallel version of the considered (AES, Blowfish, Twofish, DES, Triple DES, Serpent) algorithms. Finally, we analyzed and discussed the obtained results.

1.6 Thesis Structure

This thesis consists of six chapters,

Chapter 2 Background: presents theoretical background of encryption algorithms and the theory needed for thesis work.

Chapter 3 Literature Review: includes related work of parallel encryption and highlights the differences between parallel and sequential implementation

Chapter 4 introduces the Proposed Parallel System.

Chapter 5 presents Experimental Results and Discussion

Chapter 6 Conclusion and Future Work
Chapter 2

Background
Background

In this chapter, a short introduction into encryption functions and block ciphers are given. We also discuss briefly about the considered x86-64 architecture with optimization techniques, methods and details.

First, we give a survey of the definitions and the terms which are used throughout the research. **Plaintext** is the original message in a form that is understandable by all parties, including the sender of message and the receiver. Plaintext is also understandable by anyone eavesdropping on the communication between the two (Anderson R. J., 2008, Chefranov and Mahmoud, 2010, Chefranov and Mahmoud, 2013). Ciphertext on the other hand is the encrypted message that should not be understood by other parties except the sender and the receiver of the message. Plaintext is converted to ciphertext by the process of encryption and ciphertext is converted back to plaintext by decryption (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

The system for performing the encryption and the decryption of messages is called a cipher or a cryptographic system. The science and art of constructing cryptographic systems is called cryptography. The attacker tries to find out the original plaintext message (or the key) without knowing all the details of the system. This area of scientific research is called cryptanalysis and the research area combining cryptography and cryptanalysis is called cryptology (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017, Mahmoud and Chefranov, 2012).

Several different cryptographic primitives exist, such as stream ciphers and block ciphers. Now key in cryptography is an entity that is used to customize the used cipher so that only the parties that have the correct key can expose the original plaintext message from the encrypted one (Mollin, 2005). A cryptographic system that uses the same key in both the sending and receiving sides of the communication is called a shared-key, secret-key or symmetric-key cipher. A system that uses different keys for encryption and decryption is called a public-key or an asymmetric-key cipher (Anderson R. J., 2008).
2.1 Definition

Parallel programming is a form of computing that relies on the simultaneous execution of program components; this enhances program performance and reduces execution time. The role of synchronization has increased performance to increase computing performance, especially recently.

Firstly, the development of the processors, adopted Moore's law to increase performance by increasing the number of transistors in the integrated chip, and then another technique was adopted that attempted to implement the implicit parallelism by executing more than one instruction in one cycle, pipelines and increasing frequency of processors. However, these technologies also reached their limits when problems related to energy and heat consumption began to emerge.

At present, multi-core technologies are being adopted to achieve the target while increasing the number of cores within a single processor. However, to make the most of these processors, the application should be designed in a synchronous manner; this is the need for parallel technologies and high-level parallel programming languages to facilitate the task of the programmer to develop multi-threaded or multi-process applications (Katz & Lindell, 2014).

A multi-core system is a system which consists of two or more cores within a single processor. Here, core is nothing but a processing or execution unit. Multi-core architecture consists of two or more processing cores on the same chip.

So, it is also referred as Chip Multiprocessor. In multi-core architecture design, each core has its own execution pipeline and each core has the resources required to run without blocking the resources needed by the other software threads.
Figure 1. Architecture of Multicore System (Roma, 2011)

Figure 1 shows the architecture of multi-core systems it has n number of processing cores integrated onto a single Chip. Each processing cores has its own private L1 cache and share a common L2 cache. The bandwidth between the L2 cache and main memory is shared by all the processing cores.

The dual core processor contains two-cores, quad-core processor contains four cores and so on. Multi-core processor supports multiprocessing into a single physical package. Different cores in a multi-core system can be coupled together loosely or tightly. Some of the common network topologies to interconnect cores are ring, bus, 2-dimensional mesh, crossbar etc. Multi-core system supports the concept of simultaneous multithreading.

Figure 2. Multi-core System with Simultaneous Multithreading (Knoop, Karl, Schulz, & Ino, 2017)
Figure 2 shows the multi-core system with simultaneous multithreading. It permits several independent threads to execute simultaneously on the same core. So, in multi-core systems no. of threads, can execute multiple numbers of tasks simultaneously.

2.2 Principles of encryption

In symmetric-key cipher the same key is used for both encryption and decryption. The encryption function for symmetric-key cipher is a bijective function transforming the input plaintext message to ciphertext. The plaintext message belongs to finite message space M and the ciphertext message to finite ciphertext message space C (Mollin, 2005).

A keyed encryption function can be presented as

\[ c = E_e(m); m \in M, e \in K_e \ldots \ldots \ldots (1) \]

where e is the encryption key, m is the input plaintext message and c the output ciphertext message. The key e is selected from the available key space Ke and determines how the function Ek maps plaintext messages to ciphertext messages (Mollin, 2005). A good cipher should perform a different transform with each different key from the key space (Ferguson & Schneier, 2003).

Likewise, the decryption function is bijective function for transforming the input ciphertext to a plaintext message. The decryption function can be presented as

\[ m = D_d(c); c \in C, d \in K_d \ldots \ldots \ldots (2) \]

where d in the decryption key, c is the input ciphertext message and m is the output plaintext message. The decryption function with the correct corresponding decryption key d is an inverse of the encryption function with the corresponding encryption key e,

\[ m = D_d(c) = E^{-1}(c).K_d K_e \ldots \ldots \ldots (3) \]

combining these corresponding keys with the encryption and decryption functions determines the cryptographic system or cipher. In a symmetric-key cipher the encryption key can easily be transformed into the decryption key and vice versa. Typically, the encryption key and the decryption keys are the same, e = d, thus clarifying the use of the term symmetric-key (Mollin, 2005).
For asymmetric-key ciphers, the decryption key is different from the encryption key and the encryption key cannot easily be transformed into the decryption key without some extra knowledge (Katz & Lindell, 2014). Asymmetric-key ciphers allow the encryption key to be made public so anyone having access to the public key may produce a ciphertext message that only the holder of the “private” decryption key can transform to readable plaintext (Mollin, 2005). The symmetric-key encryption, contrariwise, requires the key to be shared by some method between the sender and the receiver, without making the key public (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). In this light the asymmetric-key encryption might seem more desirable for communication as the encryption keys can be made public without risking critical information being leaked. However, asymmetric-key ciphers require much computational processing and as a result are at least 2 to 3 orders of magnitude slower than symmetric-key ciphers (Katz & Lindell, 2014). Therefore, the symmetric-key ciphers are used for the bulk encryption of large data. Asymmetric-key ciphers and their related key exchange schemes are used to transmit the shared symmetric-key between the sending and receiving sides (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

Modern ciphers are designed so that the security of the encryption scheme depends only on the secrecy of the encryption key. The cipher algorithm must be designed in such a way that the algorithm can be allowed to fall into the hands of the enemy without risking the security. This is one of the cipher design principles that was presented by Kerckhoffs (Kerckhoffs, 1883) and it is now known as Kerckhoffs’ principle (Mollin, 2005). The reason why such a principle is valuable is that algorithms are hard to change after the deployment to use in software and hardware.

By making security depend on the key, one can use the same algorithm for a much longer time and make the use of encryption more practical (Ferguson & Schneier, 2003). By allowing the cipher algorithm to be public, the algorithm also gains attention from the cryptanalysis research. In this way the algorithm gets public scrutiny and the possible weaknesses of the cipher can be revealed before the cipher is deployed into use (Mollin, 2005) (Ferguson & Schneier, 2003).
Different classes of symmetric-key encryption systems exist, such as block ciphers and stream ciphers (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). Block ciphers encrypt fixed-size plaintext messages to ciphertext messages of the same size (Mollin, 2005). Stream ciphers on the other hand encrypt messages one bit or byte at a time (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). Typically stream ciphers generate a pseudo-random key stream based on the used encryption key, which is then XORed with the plaintext message to produce the ciphertext message (Katz & Lindell, 2014). We shall detail the block ciphers on the next section. Later in the section 2.2 we look at a way of constructing an optimal cipher.

2.3 Block ciphers

Block ciphers can be considered to be the basic building block of the symmetric-key encryption schemes, rather than being used only to encrypt messages (Katz & Lindell, 2014). As the performance of encryption is critical especially on the server side of communications, good block ciphers are designed to be fast in software and hardware (Ferguson & Schneier, 2003) (Robshaw, 1995). The input for a block cipher encryption is a fixed-size plaintext message, which is transformed to a same size ciphertext message. This fixed message length is called the block size. The most common block sizes used in the current block ciphers are 64 bits and 128 bits (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

A block cipher system contains the encryption function, the decryption function and the encryption key. The length of the key can vary, and the accepted key lengths depend on the cipher algorithm (Katz & Lindell, 2014). For each different key the block cipher encryption function will perform different permutation on the same input plaintext block. With the encryption key being secret to the outsiders, the permutation appears as random. Therefore, block ciphers can be described as a pseudo-random permutation. The reason why block ciphers are pseudo-random instead of truly random is that with knowledge of the key, the ciphertext is no longer random (Katz & Lindell, 2014).
Although the construction of the encryption function is different for each cipher algorithm, there are founding principles on which the design and the construction of the block ciphers are based (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). The fundamental cryptographic techniques of confusion and diffusion were introduced by Shannon (Shannon, 1949).

The goal of confusion in the cipher design is to make the statistical analysis of the ciphertext difficult for cryptanalysts. This is achieved by obscuring the ciphertext so that statistical redundancies or patterns disappear. Strong confusion in a cipher can be accomplished by using a complex non-linear substitution algorithm (Mollin, 2005). The goal of diffusion is to spread the information of the plaintext block to whole width of the ciphertext block. In a binary cipher, this means that the change in one plaintext bit results to a change in the majority of the ciphertext bits (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). This spreading of the plaintext information over the whole width of the ciphertext makes cryptanalysis much harder. Confusion attempts to prevent successful search for redundancies in the plaintext through observing statistical patterns from the ciphertext (Mollin, 2005).

Strong block ciphers are typically constructed from weaker parts, rounds (Ferguson & Schneier, 2003). The structure of such strong block cipher is such that the weak block cipher – round – is repeated several times with different sub-keys or round keys. These multiple round keys are derived from the encryption key of the block cipher, which is sometimes called the master key. The process of constructing the sub-keys from the master key is called key schedule. The strength of block cipher against cryptanalysis grows with the number of rounds as the amount of diffusion and confusion increases (Katz & Lindell, 2014).

The two most common classes of block ciphers are substitution-permutation networks and Feistel ciphers (Ferguson & Schneier, 2003) (Katz & Lindell, 2014). We will detail these two different block cipher constructs in the following two subsections. For a note, we call the intermediate input data moving through between different rounds as the block state.
The input plaintext or ciphertext block to a block cipher is the first block state, and it is replaced by the round with new block state. This new block state may be completely replaced or contain part of the previous block state unmodified. The block state at the end of the encryption (or the decryption) process is then output as the ciphertext (or the plaintext) block.

![Feistel Encryption and Decryption Diagram](image)

*Figure 1. Feistel Encryption and Decryption (16 rounds). (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017)*

### 2.4 Implemented block ciphers

In this section, we give short descriptions of the five block ciphers that are implemented in this thesis. Each of these ciphers has a structure of its own; this variation makes a more detailed examination interesting from fast implementation point of view. Each of these
block ciphers is used in practice at least to some extent. AES, DES and Triple DES have been endorsed by different standardization organizations.

AES is widely used in different embedded applications such as encrypted Wi-Fi connections (Mathews & Hunt, 2007). Blowfish, Twofish and Serpent are also used in several different cryptographic products (Schneier, Products that Use Blowfish., 2013a), (Schneier, Products that Use Twofish, 2013b), (Fruhwirth, 2011), (Chan, 2015).

2.3.1. Advance Encryption Standard (AES)

In 2000, the Rijndael cipher was announced as the winner of the Advance Encryption Standard contest (Katz & Lindell, 2014). The contest was held by the National Institute of Standards and Technology for the search of new standard block cipher for the U.S. government (Ferguson & Schneier, 2003). Fifteen proposals were submitted by the cryptographic community, out of which the five finalists were selected for the final round (Ferguson & Schneier, 2003). As the result of winning the contest, the Rijndael cipher is now known as AES.

However, the Rijndael cipher is not exactly the same as the standardized version of AES. The Rijndael cipher supports three different block sizes: 128, 192 and 256 bits (Daemen & Rijmen, The Block Cipher Rijndael, 2000). However, the standard AES cipher defines only 128 bits block size (Katz & Lindell, 2014). The AES cipher supports three different key lengths that were required by the National Institute of Standards and Technology NIST: 128, 192 and 256 bits (Daemen & Rijmen, The Design of Rijndael: AES - The Advanced Encryption Standard., 2002). Next, we will give short overview of the structure of the AES cipher of 128 bits block size.

The structure of AES encryption function can be seen as a substitution-permutation network (Katz & Lindell, 2014). The SP-network structure is illustrated in Figure 1. The 128-bit round keys are produced by key schedule from the master key. The round keys are mixed in the round function with the block state using exclusive-or. This key-mixing phase of round function is called AddRoundKey. The round function performs the substitution phase by passing the sixteen bytes of the block state through an $8 \times 8$-bit S-box function.
This parallel S-box phase is called SubBytes. The S-box of AES performs multiplicative inverse in particular finite field GF(28) and affine transform in the form of bit-matrix multiplication. As a result, the S-box function is a bijection as required for a substitution-permutation network construct. The inverse of S-box for decryption is constructed from the inverse of the affine transform and the same multiplicative inverse in GF(28) (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). The substitution phase of decryption that performs the sixteen-parallel inverse S-box operations is called InvSubBytes (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

The permutation phase of AES is split to two different phases, ShiftRows and MixColumns (Katz & Lindell, 2014). For these phases, the block state can be seen as 4 × 4-byte matrix. The ShiftRows rotates the values in the state matrix by different amounts along the rows. The first row is not rotated (or can be said to be rotated by zero places), the second row by one place, the third by two and the last row by three places. The MixColumns phase performs matrix multiplication in the particular finite field GF(28) with each column individually as input and output. Both of these operations can be reversed and these inverses used in the decryption are called InvShiftRows and InvMixColumns (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).
The AES encryption process starts with initial transformation, which is an additional AddRoundKey operation (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017). The decryption can be performed by reversing the order of operations and replacing the operations with their inverses (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

2.3.2. Blowfish

Blowfish was designed by (Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher Blowfish, 1994) and it was one of the first ciphers to be published as patent-free and royal-free (Mollin, 2005) (Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher Blowfish, 1994). As a result, Blowfish has gained popularity and it is utilized by various products (Mollin, 2005).
Blowfish is a Feistel cipher with block size of 64 bits or, in the other words, 8 bytes. The supported master key lengths for Blowfish range from 32 to 448 bits. The key schedule for Blowfish is performed before encryption and it generates four $8 \times 32$-bit arrays that are used as the four S-boxes by the round function. This makes the S-boxes directly dependent of the master key. Additional eighteen 32-bit round keys are also generated by the key schedule (Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher Blowfish, 1994).

The decryption can be executed with the encryption function just by reversing the order of the round keys. Thus, the implementation of Blowfish is in this sense easy because implementing the encryption function also yields the implementation of the decryption function (Mollin, 2005).

Figure 3: Blowfish Encryption and Decryption (Stallings, Cryptography and Network Security: Principles and Practice, 6th Edition, 2014)
2.3.3. Twofish

Twofish is a Feistel cipher and was one of the five finalists in the AES contest (Ferguson & Schneier, 2003). Like AES, the block size of Twofish is 128 bits and it supports three key lengths 128, 192 and 256 bits (Schneier & et, al., Twofish: A 128-Bit Block Cipher, 1998). The input plaintext to encryption functions is split to two halves as with typical Feistel ciphers. Moreover, the halves are split to two 32-bit values on which most of the operations are performed (Ferguson & Schneier, 2003).

The Twofish cipher customizes its S-boxes with the master key and therefore practical software implementations compute the look-up tables in advance of encryption or decryption (Schneier & et, al., Twofish: A 128-Bit Block Cipher, 1998).

The use of look-up tables as an optimization technique. The round function F, the g-function is the part where the S-box functions are executed. Since the g-function only uses a 32-bit input and the input to F-function is the 64-bit block state, the g-function is executed twice.

*Figure 4 Twofish algorithm (Schneier, Products that Use Twofish, 2013b)*
Outputs of the both g-functions are then mixed in the PHT-function and the round key is mixed with the state using 32-bit integer addition (Schneier & et, al., Twofish: A 128-Bit Block Cipher, 1998).

The result of F-function is output as two 32-bit values that are mixed with the right-side block state using exclusive-or operation. The 32-bit values of the right-side state are additionally rotated one-bit left and right (Schneier & et, al., Twofish: A 128-Bit Block Cipher, 1998).

These rotations were introduced to make the cryptanalysis more difficult, but they have disadvantages. The software implementations are slowed by about 5% by the introduction of the one-bit rotations. These rotations also break the typical Feistel structure so that the decryption function cannot be constructed from the encryption function simply reversing the order of the round keys (Ferguson & Schneier, 2003).

2.3.4. Serpent

The Serpent cipher was one of the five finalists in the AES contest (Ferguson & Schneier, 2003). As all the ciphers in the AES contest, Serpent has 128-bit block size and supports key sizes of 128, 192 and 256 bits. The Serpent encryption function uses the substitution-permutation network structure. Serpent is said to be designed with a stronger emphasis on security, whereas AES puts emphasis on efficiency.

Furthermore, as Serpent is designed the bit-sliced optimization in mind, the input and the output to the encryption function do not have to be transformed to a bit-sliced form (Anderson, Biham, & Knudsen, 1998). Since the same S-box is applied 32 times in parallel in each round, the substitution phase can be implemented in bit-sliced manner by treating the block state as 32 separate 4-bit blocks. Also, the permutation phase is designed so that it can be implemented efficiently for bit-sliced implementation (Anderson, Biham, & Knudsen, 1998).
2.3.5. **Data Encryption Standard DES**

Data Encryption Standard (DES) was the developed in 1974 by an IBM candidate based on earlier algorithm, Horst Feistel’s Lucifer cipher. This algorithm was developed for US government’s computer security needs. Later, many attacks were recorded that confronted several weaknesses of DES Algorithm. Brute force attack became major reason for the failure of DES algorithm as there was a fixed key size of 56bits (+ 8 parity bits) (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

*Figure 5 Serpent Encryption Algorithm (Naeemabadi, 2015)*
2.3.6. Triple DES

Triple DES Algorithm is same as DES Algorithm except we apply it three time. So, in order to understand Triple DES, we need to understand how DES Algorithm is used to encrypt data and generating key.

DES performs an initial permutation on the 64 bits block of data. Then it splits it into two parts named L and R, each 32-bit sub-blocks. Then the encryption of block of message takes place in 16 rounds. From the input key, sixteen 48-bit keys are generated, one for each round. The right half is expanded from 32 to 48 bits. The result is combined with the sub-key for that round using the XOR operation.

Using the S-boxes the 48 resulting bits are then transformed again to 32 bits, which are subsequently permutated again using yet another fixed table. This by now thoroughly
shuffled right half is now combined with the left half using the XOR operation. In the next round, this combination is used as the new left half. This process is conducted for all 16 rounds. The function f in following figure makes all mapping in all rounds (M. & et.al., 2018).

*Figure 7 3DES encryption and decryption algorithm (Mohammad, 2017)*
Chapter 3

Literature Review
Literature Review

The expression "computer security" used to describe the process of maintaining the confidentiality of information stored in computers. However, at present with the massive proliferation of computers worldwide and with increased Internet access, computer security now deals with other issues such as protecting the privacy of users and intellectual property, which have become serious issues because of the increasing number of people who use e-commerce technologies and services Digital banking, and also because of the most organized crimes online.

This will result in information security in terms of computers, so the principles of information security must be satisfactory in every organization or even every computer system. Thus, information security is responsible for protecting data from unauthorized access. Information security consists of three main pillars: confidentiality, integrity and availability; some people refer to these corners as the acronym "CIA".

Confidentiality means that unauthorized parties are denied access to information, while integrity is to ensure that data is not changed after it is sent and received, that is, it has not been changed by a third party. Availability ensures that data is available at any time if the requesting party is authorized (Stallings, Cryptography and Network Security: Principles and Practice, 7th edition, 2017).

Confidentiality and integrity are implemented using encryption, however, the achievement of availability needs to be done by another function.

In this section, some mathematical background is presented, and a full description of the cryptographic algorithms is clarified.

There are many algorithms that can be used to encrypt and decrypt data. In the past, the strength of encryption was based on the algorithm's secret; however, many scenarios have shown that this is not true, because using the same algorithm all the time, regardless of its power, will reduce system security and the possibility for cracking the algorithm will increase. At this time, the most powerful algorithms are well known. However, the secret part is the key encryption / decryption, which obviously can be easily changed on a regular basis (Lehtinen, Russell, & Gangemi, 2006).
Encryption algorithms can be classified into two main categories: symmetric key algorithms and public key algorithms.

The key feature of the symmetric key algorithms is that the decryption key can be calculated from the encryption key, however, they are equal in most practical applications. The main problem here is the key transmitting issue.

Symmetric key algorithms can be separated into two branches: block cipher scheme and stream cipher scheme.

The block cipher encryption scheme breaks the message down into blocks with a specific length and encrypts each one separately. The cryptographic block system divides the message into fixed length blocks and encrypts each of them separately. In its simplest form, when the cluster size is equal to one, the block cipher scheme becomes a stream cipher scheme. The stream encryption scheme has an advantage over the cluster cryptography scheme. In fault-prone transmission, the transmitter can transmit the wrong data without affecting the decryption process, and it can be easily used with telephone conversations and wireless network connections. Also, it is considered less secure than the block cipher scheme (Menezes, van Oorschot, & Vanstone, 2001 Updated 2014).

The most well-known symmetric key algorithms are: AES, Blowfish, Twofish, DES, 3DES, Serpent. (Qu & Yang, 2012, Mahmoud, 2012)

Navalgund (Navalgund, Desai, Ankalgi, & Yamanur, 2013) proposed parallelization version of AES algorithm using (OpenMP) programming model to express complex pipelined computations. The proposed parallelization version of AES algorithm is enhanced at data level and control level. According to parallel calculation models, the independent parts of the algorithms must be determined and then intended to work in separate threads. The algorithms divided into parallelizable and un-parallelizable parts. The parallelized part and unparalleled part are combined using fork-join model. This work is concentrate on parallelization at data level and control level (Navalgund, Desai, Ankalgi, & Yamanur, 2013).

Baas suggested another version of the parallelization of AES algorithm in their work by improving the performance of AES algorithm at task level and data level. The
system is designed for fine-grained many core systems. The design of this system works with Two, Four and six cores. The fastest design obtains the most throughput when the processors are running at a specified frequency. This system assigns the workload of each processor, which reduces the number of cores (Liu & Baas, 2013).

Nagendra and Chandra Sekhar suggested the Parallelization of AES algorithm in their work by improving the performance of AES using parallel computation that depends on divide and conquer approach. Such approach is used in parallel computation to solve algorithms in parallel by partitioning and allocating number of given subtask to available processing units.

The text file is given as input. The text file is decomposed into number of blocks. Each block is executed in a single core. The implementation of cryptography algorithm is done on dual core processor. The programming model of express complex pipelined computations Application Programming Interface (OpenMP API) was used to reduce the implementation time.

The system is specially intended for dual core processors, but recent processors have many cores, thereby performance of the system is not improved much. The system is specially designed for encryption algorithm, under Symmetric cryptosystems. But most of the applications in internet use only asymmetric cryptography to achieve confidentiality, authentication and integrity. The blocks of text file are executed in dual core processor, so that the CPU resources are not fully utilized in looping statements. The text file is executed in parallel, but the source code is not parallelized (Nagendra & Chandra Sekhar, 2014).

Parallel processing is a form of computation in which many calculations are carried out simultaneously, (Almasi & Gottlieb, 1993) operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (“in parallel”). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism.

As power consumption (and consequently heat generation) by computers has become a concern in recent years, (Adve & et, al., 2008) parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors. “Parallel computers can be roughly classified according to the level at which
the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks” (Asanovic & et, al., 2008).

The fundamental parts of the study, which can be divided into two essential parts: security and parallel parts. The focus was on the description of the cryptographic algorithms relies on math’s and need mathematical knowledge with a specific end goal to have the capacity to program them in the most ideal way. This is especially true when the target mission is to parallelize these algorithms, which need a very deep understanding not only for their steps and stages, but also how they work mathematically. The Parallel System proposed focuses on the technologies which are available to build a parallel program. Besides, it concentrates on the performance measures that can be applied to improve the parallel algorithm.
Chapter 4

Proposed Parallel System
Proposed Parallel System

This chapter introduces the Proposed Parallel System using Multi-core processor in parallel to enhance the encryption performance. By using Parallel File Encrypt with API Libraries to perform the encryption/decryption itself, and Parallel Programming Library (PPL) for parallelization on a computer system. With generated fixed file sizes for experiment we develop, implement, test and evaluate a parallel version of the considered (AES, Blowfish, Twofish, DES, Triple DES, Serpent) algorithms.

4.1. Parallel Programming Languages

“Parallel Programming is programming in a language that allows you to explicitly indicate how different portions of the computation may be executed concurrently by different processors.” (Quinn, 2004). A lot of techniques (libraries, API, parallel models and languages) have been evolved to assist the development of parallel programs.

By using RAD Studio to develop multi-device applications that run on other platforms. The RTL (run-time library) provides the Parallel Programming Library (PPL), providing your applications/systems the ability to allocate tasks running in parallel taking advantage of working across computers with multiple CPU. The PPL contains several advanced features for executing tasks/jobs, joining tasks, waiting on groups of tasks, etc. to process. For all this, there is a thread pool that self-tunes itself automatically (based on the load on the CPU’s).

Using the PPL, the proposed applications can easily:

- Make looping faster with Sub Class(TParallel.For).
- Run multiple tasks in parallel using Sub Class (TTask) and (ITask).
- Leave a process running focusing on other tasks and then get the result of that process at the point you want. Sub Class (IFuture) allows you to establish a priority for the running code blocks and still return the results when needed.
4.2. Designing Parallel Encryption Algorithms

Modern cryptographic algorithms often rely on mathematics to convert the plaintext into the ciphertext. Thus, those algorithms can include heavy mathematical calculation repetition on the data segments (bytes, blocks etc.). The Single Program Multiple Data (SPMD) mode can be the most relevant mode to parallelize the cryptographic algorithms. However, according to (Chan, 2015) the general stages of designing a parallel program from sequential code are: analyses the sequential algorithm to identify the potential parallelize, design and implement the parallel version (At this stage the prototyping techniques may be used), Test to detect synchronization errors that may be caused by multiple threads and adjust the algorithm for better performance by removing all performance bottlenecks.

In this study, six of the most commonly used cryptographic algorithms have been described and identified. The previous steps will be adopted when designing parallel algorithms; However, it is important to clarify some of the factors that will be used in the project.

4.3.1. Prototyping

“Prototype is an initial version of a software system that is used to demonstrate concepts, try out design options and, generally, to find out more about the problem and its possible solutions” (Sommerville, 2015). According to (d’Auriol, Lee, & Lee, 2008) prototyping is a common activity during the design stage of systems. It focuses on key ideas and basic requirements, and results in a reduction in the total coding that is allocated to solve the problem. In addition, the prototype allows for the exploration of various solutions and critical resource aspects associated with the problem. The prototypes should be built rapidly and modified rapidly as well to get the most benefits from the prototyping process. To achieve this, the language should provide very high-level semantic constructs which enable the designer to model any data structure or idea.

In parallel prototypes used in the design phase of parallel algorithms to produce the initial parallel version of the program that contains the main parallel properties. Later, learn from this version and improve it until we get to the final version. Therefore, the primary
objective of the prototypes is not to achieve ideal speed but to provide us ideas before drawing the algorithm into a specific parallel structure for optimal speed as a final result.

4.3.2. Parallel Implementation

According to (Breshears, 2009), there are eight rules that must be adopted in the design of multi-threaded parallel algorithms:

1. Determine the independent components of the serialized algorithm since some components of the algorithm may not be parallel because of the dependencies between them.

2. To perform concurrency using the highest level of parallelism, there are two approaches to define the highest-level top-down and bottom-up.

3. Plan for scalability in the early design phase and take into account the increase in the number of processing units.


5. Use the implicit threading model over the explicit model that provides the required functionality.

6. Do not assume that the algorithm components should be executed in a particular order.

7. Use local thread variables as much as possible and provide locking on shared data to ensure synchronization.

8. Change the algorithm if this can provide more synchronization, even if this increases the complexity of the algorithm.

However, the cryptographic algorithms can be parallelized to a higher level, when we use the encryption mode ECB or CTR modes, which may provide higher performance (Mahmoud and Chefranov, 2014, Doukhnitch, & et.al, 2013).
4.3.3. Performance Benchmarking

Performance measurement is a very important step that allows identifying the bottlenecks in the algorithm and trying to eliminate them in order to improve overall performance. Furthermore, we can decide on the best level of parallelism or level of accuracy of the algorithm and the parallel environment. Learn more about the scalability factor in the algorithm when increasing the number of processing units that work together to solve the problem.

In this study we use two factors to evaluate performance: a speedup factor and efficiency factor as described below:

a. Speedup

According to (Robshaw, 1995), the speedup of the parallel algorithm is the ratio between the delay time of sequential execution and the delay time of the parallel version execution:

\[ S = \frac{T_{seq}}{T_{par}} \] \hspace{1cm} (4)

Where the value \( T_{seq} \) is the execution delay time of the sequential version, and \( T_{par} \) is the execution delay of the parallel version. However, the value of the speedup can be represented as a multiple as well, for example, 2x, which means that the speed will be doubled (Chan, 2015). The speedup should be changed by changing the number of processing units; this will give a sense of algorithm scalability by measuring the extent of change in speed when changing the number of processing units.

b. Efficiency

Efficiency tells the designer how well the computational resources are being utilized (Chan, 2015). Efficiency can be measured by dividing the Speedup value by the number of processing units.
\[
\text{Efficiency} = \frac{\text{Speedup}}{P} \quad \ldots \quad (5)
\]

The number of processing units \( P \) is a key factor in the efficiency equation; moreover, the efficiency of some programs may be adversely affected when the number of processing units increases. Although reducing the number of processing units may enhance efficiency, the main question remains “why the parallel program cannot get the most benefit from the processing units?” Additionally, increasing the number of threads and assigning more than one thread per processing unit may slightly improve program performance (Chan, 2015). However, speed and efficiency values can provide clues about the level of accuracy for the best performance.
Chapter 5

Experimental Results and Discussion
Experimental Results and Discussion

In this chapter, the results of parallelized encryption algorithms are presented. The generated files with different sizes on the identical dedicated computers with different processors are introduced. We introduce the analysis and discussion of the obtained results for the parallel and sequential implementation.

5.1. Algorithm(s) to be parallelized

From the previous literature review it can be seen targeted algorithms AES, Blowfish, Twofish, DES, 3DES, Serpent are the strongest algorithms and the most frequently used ones nowadays. Consequently, those algorithms are very strong candidates to be implemented and parallelized.

Consequently, due to time limitations other encryption algorithms are not implemented or parallelized. Moreover, it would be more interesting to parallel algorithms, and thus must provide precisely parallel algorithms.

5.2. Experimental Setup

It is possible to benchmark the performance of the various algorithms by running the program Parallel File Encrypt with API Libraries Used to perform the encryption/decryption itself, and PPL for parallelization on a computer system.

This program executes all currently implemented sequential versions and writes the execution time to an external Excel file. The file sizes that are used in the experiment are: 5MB, 10MB, 100MB, 1000MB which are generated previously and stored in external files.

All the sequential and parallel versions have been measured on multi-core machine, comprising performance analysis was run under windows 10 operating system with 16GB of RAM available using three different Processors as following:

- Processor Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 2592 Mhz, 2 Core,
- Processor Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Core,
- Processor Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz, 2000 Mhz, 6 Core(s),
So, the program was executed to compare the performance of the various algorithms. To obtain reliable readings the measurement is repeated three times and the median is taken.

### 5.3. Implementation and Benchmark

In this Section, a sequential and parallel version will be developed using Delphi with PPL as shown in Figure 10 and Figure 11. It starts working by the selection of input and output files then identify the algorithm to be used with the encryption key. Finally, execute and get the time elapsed for the parallel and sequential implementation.

The performance will be compared with the best sequential. As a result of this Section there will be some analysis for the parallel and sequential implementation.

![Sequential File Encryption](image1)

![Parallel File Encryption](image2)

*Figure 8. Sequential Implementation Tool*

*Figure 9. Parallel Implementation Tool*

Firstly, the sequential performance will be measured. Secondly, the parallel performance will be measured by tuning the number of processors and various Files size. We notice CPU utilization differences between sequential and parallel Implementations as shown:
The following tables show the obtained values of the elapsed run time, speedup and efficiency of sequential and parallel Implementation by different processor cores in milliseconds with four different files size.
### Table 1. DES Implementation run time as Sequential and Parallel using Multi Cores

<table>
<thead>
<tr>
<th>DES Runtime</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>112</td>
<td>224</td>
<td>2251</td>
<td>22428</td>
</tr>
<tr>
<td>2 Core</td>
<td>92</td>
<td>192</td>
<td>1872</td>
<td>18463</td>
</tr>
<tr>
<td>4 Cores</td>
<td>78</td>
<td>89</td>
<td>846</td>
<td>8600</td>
</tr>
<tr>
<td>6 Cores</td>
<td>56</td>
<td>64</td>
<td>557</td>
<td>5592</td>
</tr>
</tbody>
</table>

Table 1: show the obtained values of elapsed **run time** of sequential and parallel implementation by different processors cores in milliseconds with four different files size for **DES** Algorithm.

![Des runtime graph](image)

*Figure 12. DES Implementation run time as Sequential and Parallel using Multi Cores*

Figure 16: illustrates the differences in **run time** values of sequential and parallel by different processor cores in milliseconds with four different files size for **DES** Algorithm. It shows that the parallel encryption time is better than the sequential encryption time.
Table 2: show the obtained values for Speedup calculations regarding the sequential and parallel implementation by different processors cores with four different files size for DES Algorithm.

Figure 17: Illustrates the differences in Speedup values of sequential and parallel implementation by different processors cores with four different files size for DES algorithm.
### Table 3: DES Implementation Efficiency as Sequential and Parallel using Multi Cores

<table>
<thead>
<tr>
<th>DES Efficiency</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Core</td>
<td>0.61</td>
<td>0.58</td>
<td>0.60</td>
<td>0.61</td>
</tr>
<tr>
<td>4 Cores</td>
<td>0.36</td>
<td>0.63</td>
<td>0.67</td>
<td>0.65</td>
</tr>
<tr>
<td>6 Cores</td>
<td>0.33</td>
<td>0.58</td>
<td>0.67</td>
<td>0.67</td>
</tr>
</tbody>
</table>

Table 3: show the obtained results of **Efficiency** calculations for sequential and parallel implementation by different processors cores with four different files size for **DES** Algorithm.

![DES Efficiency Graph](image)

**Figure 14. DES Implementation Efficiency as Sequential and Parallel using Multi Cores**

Figure 18: illustrates the differences in **Efficiency** values of sequential and parallel implementation by different processors cores with four different files size for **DES** Algorithm.
<table>
<thead>
<tr>
<th>3DES Runtime</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential</strong></td>
<td>298</td>
<td>582</td>
<td>5657</td>
<td>56987</td>
</tr>
<tr>
<td><strong>2 Core</strong></td>
<td>110</td>
<td>229</td>
<td>2148</td>
<td>20585</td>
</tr>
<tr>
<td><strong>4 Cores</strong></td>
<td>116</td>
<td>209</td>
<td>1910</td>
<td>19001</td>
</tr>
<tr>
<td><strong>6 Cores</strong></td>
<td>66</td>
<td>139</td>
<td>1099</td>
<td>11085</td>
</tr>
</tbody>
</table>

Table 4: 3DES Implementation Runtime as Sequential and Parallel using Multi Cores

Table 4: show the values of elapsed run time of sequential and parallel implementation by different processors cores in milliseconds with four different files size for 3DES Algorithm.

Figure 15: 3DES Implementation Runtime as Sequential and Parallel using Multi Cores

Figure 19: illustrates the differences in run time values of sequential and parallel implementation by different processors cores in milliseconds with four different files size for 3DES Algorithm.
Table 5: show the obtained values of Speedup calculations for sequential and parallel implementation by different processors cores with four different files size for 3DES Algorithm.

![3DES Speedup Chart](chart.png)

Figure 16. 3DES Implementation Speedup as Sequential and Parallel using Multi Cores

Figure 20: illustrates the differences in Speedup values of sequential and parallel by different processor cores with four different files size for 3DES Algorithm.
Table 6: shows the obtained values for Efficiency calculations of sequential and parallel implementation by different processors cores with four different files size for 3DES Algorithm.

<table>
<thead>
<tr>
<th>3DES Efficiency</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Core</td>
<td>1.35</td>
<td>1.27</td>
<td>1.32</td>
<td>1.38</td>
</tr>
<tr>
<td>4 Cores</td>
<td>0.64</td>
<td>0.70</td>
<td>0.74</td>
<td>0.75</td>
</tr>
<tr>
<td>6 Cores</td>
<td>0.75</td>
<td>0.70</td>
<td>0.86</td>
<td>0.86</td>
</tr>
</tbody>
</table>

Figure 17. 3DES Implementation Efficiency as Sequential and Parallel using Multi Cores

Figure 21: illustrates the differences in Efficiency values of sequential and parallel implementation by different processor cores with four different files size for 3DES Algorithm.
<table>
<thead>
<tr>
<th>AES Runtime</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>77</td>
<td>148</td>
<td>1332</td>
<td>13597</td>
</tr>
<tr>
<td>2 Core</td>
<td>71</td>
<td>142</td>
<td>1445</td>
<td>14703</td>
</tr>
<tr>
<td>4 Cores</td>
<td>35</td>
<td>63</td>
<td>617</td>
<td>6543</td>
</tr>
<tr>
<td>6 Cores</td>
<td>25</td>
<td>47</td>
<td>405</td>
<td>5304</td>
</tr>
</tbody>
</table>

Table 7. AES Implementation Runtime as Sequential and Parallel using Multi Cores

Table 7: shows the values of the elapsed **run time** of sequential and parallel implementation by different processor cores in milliseconds with four different files size for AES Algorithm.

![AES Runtime](image)

Figure 18. AES Implementation Runtime as Sequential and Parallel using Multi Cores

Figure 22: illustrates the differences in elapsed **run time** values of sequential and parallel implementation by different processor cores in milliseconds with four different files size for AES Algorithm.
Table 8: shows the values of Speedup calculations of sequential and parallel implementation by different processor cores with four different files size for AES Algorithm.

Figure 19: AES Implementation Speedup as Sequential and Parallel using Multi Cores

Figure 20: illustrates the differences in Speedup values of sequential and parallel implementation by different processor cores with four different files size for AES Algorithm.
<table>
<thead>
<tr>
<th>AES Efficiency</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Core</td>
<td>0.54</td>
<td>0.52</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td>4 Cores</td>
<td>0.55</td>
<td>0.59</td>
<td>0.54</td>
<td>0.52</td>
</tr>
<tr>
<td>6 Cores</td>
<td>0.51</td>
<td>0.52</td>
<td>0.55</td>
<td>0.43</td>
</tr>
</tbody>
</table>

*Table 9. AES Implementation Efficiency as Sequential and Parallel using Multi Cores*

Table 9: shows the values of **Efficiency** calculations of sequential and parallel implementation by different processor cores with four different files size for AES Algorithm.

![AES EFFICIENCY](image)

*Figure 20. AES Implementation Efficiency as Sequential and Parallel using Multi Cores*

Figure 24: illustrates the differences in **Efficiency** values of sequential and parallel implementation by different processor cores with four different files size for AES Algorithm.
Table 10: shows the values of elapsed run time of sequential and parallel implementation by different processor cores in milliseconds with four different files size for Blowfish Algorithm.

Figure 21: Blowfish Implementation Runtime as Sequential and Parallel using Multi Cores

Figure 25: illustrates the differences in run time values of sequential and parallel by different processor cores in milliseconds with four different files size for Blowfish Algorithm.
### Table 11. Blowfish Implementation Speedup as Sequential and Parallel using Multi Cores

<table>
<thead>
<tr>
<th>Blowfish Speedup</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Core</td>
<td>0.57</td>
<td>0.60</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>4 Cores</td>
<td>1.55</td>
<td>2.00</td>
<td>2.09</td>
<td>2.07</td>
</tr>
<tr>
<td>6 Cores</td>
<td>1.46</td>
<td>1.51</td>
<td>1.62</td>
<td>1.49</td>
</tr>
</tbody>
</table>

Table 11: shows the obtained values of Speedup calculations of sequential and parallel implementation by different processor cores with four different files size for **Blowfish** Algorithm.

*Figure 22. Blowfish Implementation Speedup as Sequential and Parallel using Multi Cores*

Figure 26: illustrates the differences in Speedup values of sequential and parallel implementation by different processor cores with four different files size for **Blowfish** Algorithm.
<table>
<thead>
<tr>
<th>Blowfish Efficiency</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Core</td>
<td>0.29</td>
<td>0.30</td>
<td>0.29</td>
<td>0.30</td>
</tr>
<tr>
<td>4 Cores</td>
<td>0.39</td>
<td>0.50</td>
<td>0.52</td>
<td>0.52</td>
</tr>
<tr>
<td>6 Cores</td>
<td>0.24</td>
<td>0.25</td>
<td>0.27</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 12. Blowfish Implementation Efficiency as Sequential and Parallel using Multi Cores

Table 12: shows the values of **Efficiency** calculations of sequential and parallel implementation by different processor cores with four different files size for **Blowfish** Algorithm.

![BLOWFISH EFFICIENCY](image)

Figure 23. Blowfish Implementation Efficiency as Sequential and Parallel using Multi Cores

Figure 27: illustrates the differences in **Efficiency** values of sequential and parallel implementation by different processor cores with four different files size for **Blowfish** Algorithm.
Table 13: shows the values of elapsed run time of sequential and parallel implementation by different processor cores in milliseconds with four different files size for Twofish Algorithm.

![TWOFISH RUNTIME](image)

Figure 24. Twofish Implementation Runtime as Sequential and Parallel using Multi Cores

Figure 28: illustrates the differences in elapsed run time values of sequential and parallel by different processor cores in milliseconds with four different files size for Twofish Algorithm.
Table 14: show the values of Speedup calculations of sequential and parallel implementation by different processor cores with four different files size for Twofish Algorithm.

![TWOFISH SPEEDUP](image)

**Figure 25. Twofish Implementation Speedup as Sequential and Parallel using Multi Cores**

Figure 29: illustrates the differences in Speedup values of sequential and parallel implementation by different processor cores with four different files size for Twofish Algorithm.
<table>
<thead>
<tr>
<th>Twofish Efficiency</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Core</td>
<td>0.10</td>
<td>0.09</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>4 Cores</td>
<td>0.30</td>
<td>0.29</td>
<td>0.30</td>
<td>0.30</td>
</tr>
<tr>
<td>6 Cores</td>
<td>0.16</td>
<td>0.13</td>
<td>0.15</td>
<td>0.14</td>
</tr>
</tbody>
</table>

*Table 15. Twofish Implementation Efficiency as Sequential and Parallel using Multi Cores*

Table 15: shows the values of Efficiency calculations of sequential and parallel implementation by different processor cores with four different files size for Twofish Algorithm.

![TWOFISH EFFICIENCY](image)

*Figure 26. Twofish Implementation Efficiency as Sequential and Parallel using Multi Cores*

Figure 30: illustrates the differences in Efficiency values of sequential and parallel implementation by different processor cores with four different files size for Twofish Algorithm.
Table 16: shows the values of elapsed run time of sequential and parallel implementation by different processor cores in milliseconds with four different files size for Serpent Algorithm.

![Serpent Runtime Table](image)

Figure 27. Serpent Implementation Runtime as Sequential and Parallel using Multi Cores

Figure 31: illustrates the differences in run time values of sequential and parallel implementation by different processor cores in milliseconds with four different files size for Serpent Algorithm.
Table 17: shows the obtained values of Speedup calculations of sequential and parallel implementation by different processor cores with four different files size for Serpent Algorithm.

**Table 17. Serpent Implementation Speedup as Sequential and Parallel using Multi Cores**

<table>
<thead>
<tr>
<th>Serpent Speedup</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential</strong></td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>2 Core</strong></td>
<td>2.32</td>
<td>2.33</td>
<td>2.38</td>
<td>2.47</td>
</tr>
<tr>
<td><strong>4 Cores</strong></td>
<td>2.59</td>
<td>2.58</td>
<td>2.61</td>
<td>2.60</td>
</tr>
<tr>
<td><strong>6 Cores</strong></td>
<td>4.23</td>
<td>4.07</td>
<td>4.83</td>
<td>4.80</td>
</tr>
</tbody>
</table>

**Figure 28. Serpent Implementation Speedup as Sequential and Parallel using Multi Cores**

Figure 32: illustrates the differences in Speedup values of sequential and parallel implementation by different processor cores with four different files size for Serpent Algorithm.
Table 18: shows the obtained values of **Efficiency** calculations of sequential and parallel implementation by different processor cores with four different files size for **Serpent Algorithm**.

**SERPENT EFFICIENCY**

<table>
<thead>
<tr>
<th>Serpent Efficiency</th>
<th>5MB</th>
<th>10MB</th>
<th>100MB</th>
<th>1000MB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential</strong></td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>2 Core</strong></td>
<td>1.16</td>
<td>1.16</td>
<td>1.19</td>
<td>1.23</td>
</tr>
<tr>
<td><strong>4 Cores</strong></td>
<td>0.65</td>
<td>0.64</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td><strong>6 Cores</strong></td>
<td>0.70</td>
<td>0.68</td>
<td>0.81</td>
<td>0.80</td>
</tr>
</tbody>
</table>

*Table 18. Serpent Implementation Efficiency as Sequential and Parallel using Multi Cores*

**Figure 29. Serpent Implementation Efficiency as Sequential and Parallel using Multi Cores**

Figure 33: illustrates the differences in **Efficiency** values of sequential and parallel implementation by different processor cores with four different files size for **Serpent Algorithm**.
Chapter 6

Conclusion and Future Work
Conclusion and Future Work

This thesis was about reviewing different optimization techniques used on the x86-64 architecture when implementing different block ciphers and applying these techniques when constructing new faster implementations of the selected block ciphers. The use of different techniques on particular block cipher algorithm depends greatly on the actual design of the algorithm. Typically block ciphers are designed so that fast software implementations can utilize the use of large look-up tables for the substitution and permutation phases found in round function.

This review covers the main aspects of the project, which can be divided into two essential parts: security and parallel parts. The main target was on the description of the cryptographic algorithms as they rely on math’s and need mathematical understanding to be able to program them in the best way. This is especially true when the target mission is to parallelize these algorithms, which need a very deep understanding not only for their steps and stages, but also how they work mathematically. The Parallel part focuses on the technologies which are available to build a parallel program. Furthermore, it focuses on the performance measures that can be applied to improve the parallel algorithm.

The parallel processing can be accomplished with specific block cipher modes of operation, such as counter mode, that allow it. Modes of operation that have ciphertext block feedback, require the previous plaintext block to be encrypted before, serializing the encryption process. Such modes of operation cannot benefit from the additional performance introduced by parallel processing. Therefore, most of our implementations process blocks in parallel to reach higher performance.

The out-of-order scheduling, that is available in most x86-64 processors today, can find instruction level parallelism and execute independent instruction streams in parallel. Matsui (Matsui, 2006) presented this technique for implementing the Camellia cipher in two-way parallel manner, interleaving two block encryption operations to introduce more parallelism for processor to exploit and yield higher performance. We successfully applied this technique to implementations of Blowfish, AES, DES, 3DES and Twofish. The parallel implementation of 3DES was faster than sequential implementation on Intel
processor. For **Twofish** implementations was **slower** than was **slower** than reference implementation on Intel processor.

There is still room for improvement in our implementations. For example, counter mode caching was used on algorithms implementations. This optimization technique might be possible to apply to other ciphers. We applied this technique to get comparable results with previous research.

Some of the processors used in this research were quite old, and we did not have the latest generation of processors available for measurements. It would have been interesting to get measurements on new AMD processors. The newer Intel is closely related to the old Intel and differences in measurements between them can be expected to be small. On the other hand, the use of older processors allowed us to compare our results with previous research and verify that our measure methods gave identical results.
References


42. Schneier, B. (1994). Description of a New Variable-Length Key, 64-bit Block Cipher Blowfish. ACM Digital Library.


