Ministry of Higher Education and
Scientific Research
University of Babylon
College of Science for Women
Department of Computer Science

Parallel Lightweight Encryption Method

A Thesis

Submitted to the Council of the College of Science for Women at University of Babylon in Partial Fulfillment of the Requirements for the Degree of Master in Sciences \ Computer Sciences

By

Hawraa Jabir Hamiza

Supervised by

Asst. Prof. Dr. Ahmed Badri Muslim

2023 A.D. **1445 A.H**

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

فَتَعَٰلَى ٱللَّهُ ٱلْمَلِكُ ٱلْحَقُّ ۗ وَلَا تَعْجَلْ بِٱلْقُرْءَانِ مِن قَبْلِ أَن يُقْضَىٰٓ إِلَيْكَ وَحْيُهُۥ ۖ وَقُل رَّبِّ زِدْنِى عِلْمًا ﴿١١٤﴾

صدق الله العلي العظيم

(سورة طه – الآية ١١٤)

Supervisor Certification

I certify that this thesis titled "***Parallel Lightweight Encryption Method*** " was done by (**Hawraa Jabir Hamiza**) under my supervision.

Signature:

Name : Asst. prof. Dr. Ahmed Badri Muslim

Date: / / 2023

Address: University of Babylon /College of Science for Women

The Head of the Department Certification

In view of the available recommendations, I forward the dissertation entitled "***Parallel Lightweight Encryption Method*** " for debate by the examination committee.

Signature:

Name : Dr. Saif M. Kh. Al- Alak

Date: / / 2023

Address: University of Babylon /College of Science for Women

CERTIFICATION OF THE EXAMINATION COMMITTEE

We are the members of the examination committee, certify that we have read this thesis entitled (***Parallel Lightweight Encryption Method***) and after examining the master student (***Hawraa Jabir Hamiza***) in its contents 19/10/2023 and that in our opinion it is adequate as a thesis for the degree of Master in Science \ Computer Science with degree (Excellence).

Committee Chairman
Signature:
Name: **Samaher Al - Janabi**
Scientific Order: **Prof. Dr**.
Address: University of Babylon\ College of Science for Women
Date: / / 2023

Committee Member
Signature:
Name: **Rana J . S Al - Janabi**
Scientific Order: **Assist. Prof. Dr.**
Address: University of Al-Qadisiyah\ College of Computer Science and Information Technology
Date: / / 2023

Committee Member
Signature:
Name: **Muhammad A. Mahdi**
Scientific Order: **Assist. Prof. Dr.**
Address: University of Babylon\ College of Science for Women
Date: / / 2023

Committee Member(**Supervisor**)
Signature:
Name**: Ahmed Badri Muslim**
Scientific Order: **Assist. Prof. Dr.**
Address: University of Babylon\ College of Science for Women
Date: / / 2023

Date of Examination: 19 / 10 / 2023
Deanship Authentication of college of Science for Women
Approved for the College Committee of graduate studies.

Signature
Name: **Abeer Fauzi Al-Rubaye**
Scientific Order: **Prof. Dr.**
Address: Dean of College of Science for Women
Date: / / 2023

Dedication

To my father (May Allah have mercy on him)…

To the source of my happiness and strength (my beloved mother)…

To my support in this life (dear brother)...

To my friends, teachers, and everyone who has contributed to my education throughout this period…

To all of them, I dedicate this work. I ask Almighty Allah to accept it.

Hawraa (2023)

Acknowledgments

In the name of Allah, the most merciful the most compassionate all praise be to Allah the lord of the worlds and prayers and peace be upon Mohamed his servant and messenger and to his good and pure household.

First and foremost, I would like to express my gratitude to Almighty Allah for assisting me and providing me with the strength to finish my thesis.

I would like to express my sincere thanks and gratitude to my teacher and supervisor, Assistant Professor Dr. Ahmed Badri Muslim, for his continuous support. His experience, insightful comments, and helpful notes have decisively contributed to my work.

Also, I would like to express my most sincere gratitude and appreciation to my family for their kindness and support through challenging times.

Also, I owe my thanks to the person who caused me suffering, but who also gave me determination and patience(may Allah have mercy on my father).

Special thanks are extended to the College of the Science for women at University of Babylon for giving me the chance to complete my postgraduate study.

Finally, I want to thank my friends for their valuable advice and support throughout the duration of my study.

Hawraa(2023)

Abstract

Data protection has become one of the top issues, despite major advancements in communications and technology. For web-based technology to send data quickly and safely, the data must be encrypted. Encryption is the process of turning plain text into ciphered text, which bad people can't read or change. The problem of strengthening the implementation of reliable cryptographic algorithms is the focus of this study. The ubiquity of digital documents and business transactions has demonstrated how urgently encryption methods are needed. Strong cryptographic methods require a lot of time to implement effectively. However, a number of researchers developed the cryptography approach in parallel in order to reduce the amount of time needed for the encryption and decryption procedures to be finished. The investigation of the issue has produced a number of viable solutions. Researchers were able to attain improved performance levels on the encryption technique by using parallelism to increase throughput and boost the efficiency of encryption methods.

Nowadays, multi-core methods are used to accomplish the same result with more cores in a single processor. To make the most of these processors, however, the application running on them needs to be designed in a concurrent manner; this calls for the use of high-level parallel programming languages and explicit parallel techniques to make it easier for programmers to create multi-threaded or multi-process parallel applications. A lot of techniques (libraries, message-passing interfaces, etc.) have been evolved to assist in the development of parallel programs. Modern encryption algorithms usually rely heavily on mathematics to produce the cipher text. Thus, in order to guarantee a higher level of security, lightweight cryptography required numerous rounds. The computational cost rises and encryption speeds are slowed when the number of rounds is increased.

Thus, in this work, a lightweight cipher scheme is proposed that only employs one round of block cipher technique that is applied in parallel over a multicore processor. The proposed message encryption algorithm encrypts two subblocks of 128 bits of plain message per round. It uses an initial vector, substitution box, and splitmix64 PRNG and

uses a dynamic key with a size of 512 bits to increase the security level. Therefore, it encrypts the plain message and obtains two encrypted subblocks, making it a fast technique to encrypt and decrypt blocks of messages in comparison to the existing method. This proposed method is a probabilistic method since it depends on a dynamic key.

According to the performance findings, it is able to reach a high data throughput in comparison to some lightweight methods that already exist, with a throughput that is higher than 25 and 10 Gigabits per second on two Intel Core i7 central CPU. The proposed encryption method outperforms the parallel speck method by an average of 14.10 and 17.09 times faster when executed over two multicore CPUs. The average speedup compared to the sequential version of the proposed algorithm and its parallel implementation is 4.70 and 5.26, respectively. Also, the proposed encryption method offers a substantial amount of randomness and passes PractRand's statistical tests. Thus, the suggested method is a strong contender for high-security implementation on multicore processors. However, this work advances the field of data encryption by presenting a strong contender for high-security one-round encryption implemented on multicore processors, offering improved throughput, efficiency, and statistical robustness compared to existing speck method.

Table of Contents

Chapter Three: The Proposed Parallel System

Chapter Four: Proposed Method Implementation and Experimental Results

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbreviations	Full Meaning
IOT	Internet of Things
FPGA	Field Programmable Gate Arrays
ASIC	Application Specific Integrated Circuit
NSA	National Security Agency
CPU	Central Processing Unit
GPU	Graphics Processing Unit
NIST	National Institute of Standards and Technology
SPN	Substitution Permutation Network
NBS	National Bureau of Standards
FIPS	Federal Information Processing Standard
LFSRs	linear feedback Shift Registers
NLFSRs	Non-Linear Feedback Shift Registers
ARX	Add-Rotate-XOR
SSL	Secure Sockets Layer
WEP	Wired Equivalent Privacy
KSA	Key Scheduling Algorithm
MACs	Message Authentication Codes
PDF	Probability Density Function
RNGA	Pseudo-Random Generation Algorithm

PRNGs	Pseudo-Random Number Generators
LCGs	Linear Congruential Generators
GUI	Graphical User Interface
STS	Statistical Test Suite
DLP	Data Level Parallelism
ILP	Instruction Level Parallelism
TLP	Thread Level Parallelism
HPC	High Performance Computing
WAN	Wide Area Network
LAN	local Area Network
WSN	Wireless Sensor Network

List of Publications

This work has resulted in the following publication:

[1] H. Jabir, and A. Fanfakh, “An Overview of Parallel Symmetric Cipher of Messages,” J. Univ. BABYLON Pure Appl. Sci., vol. 31, no. 2, pp. 19–33, 2023, doi: 10.29196/jubpas.v31i2.4652.

[2] H. Jabir, and A. Fanfakh, “Parallel lightweight Block Cipher algorithm for Multicore CPUs,” Accepted for publication in Baghdad Sci J., July,2024.

Chapter One

General Introduction

1.1. Introduction

Security has emerged as one of the most important areas focusing on businesses, academic institutions, and government research. Our data is now disseminated and stored on servers, in the cloud, and at billions of other sites all over the globe. Various types of data (videos, images, messages, etc.) are being sent back and forth between users via a wide variety of channels, including wireless and satellite communication. These channels support a wide range of data transfer protocols. Additionally, encryption must be applicable to any type of data, implying that the cipher utilized has to safeguard not just text but also photos and videos [1].

The field of lightweight cryptography is rapidly evolving to meet the rising needs of networked systems with severely constrained software and hardware equipment. This development is necessary to fulfill growing expectations[2]. Over the past few decades, there has been a significant increase in the development of lightweight cryptography, often surpassing its predecessors due to technologies such as ASIC, FPGA, microcontrollers, or microprocessors[3]. Lightweight cryptology should be suitable for a wide variety of computer systems, considering both their hardware and software. Simon and Speck are two new symmetric cryptographic variations recently made available by the National Security Agency (NSA) of the United States of America [4-5]. These variants are simple, lightweight, and perform very well in both software and hardware.

The primary goal of many models and programming techniques is to reduce the time and cost of computation required for parallel systems to run parallel programs. Efficiency and acceleration are the measures used to quantify program performance based on software implementation durations for concurrent or distributed applications. One way to increase speed is by utilizing a technique known as Thread-Level Parallelism (TLP), where multiple processors run different sections of a program simultaneously and exchange data via shared memory locations during program execution [6].

This study focuses on the issue of strengthening the application of trustworthy cryptographic methods. The widespread use of digital documents in commercial transactions has highlighted the critical need for encryption techniques. Robust cryptographic techniques take a long time to deploy efficiently. also , focuses on optimizing the execution of cryptographic calculations with the ultimate objective of accelerating the encryption process and effectively utilizing modern multi-core processors. Parallel dialect technology, which employs threads as the main executing unit, performs very well in a multi-core computer environment.

1.2. Research Problem

Lightweight algorithms make significant contributions to the field of encryption in parallel systems; additional research and analysis are needed to address the highlighted research gaps and provide a more comprehensive understanding of the performance, security, and practical applicability of the used encryption techniques. Because of these challenges, nowadays, a lot of lightweight encryption techniques leverage multi-core technology to achieve the same goal with more CPU cores. However, to fully utilize.

In order to generate the encrypted text, modern encryption algorithms typically rely substantially on mathematics. Thus, more rounds were needed for lightweight cryptography to ensure a higher level of security. Increasing the number of rounds causes the computational cost to increase and the encryption rate to slow down.

1.3 Aims of the Thesis

The aims of thesis include several aspects such as:

1. It provides a high level of security by protecting data and not allowing hackers to recover the secret key.

2. Reduce the cost of computation: Many encryption algorithms currently have a low computational cost. To achieve this, the thesis will focus on reducing the

encryption-decryption complexity. However, propose an effective parallel method that is single-round, and then compare it with Speck's algorithm. this leads to reduce cost computational.

3. increasing method throughput (measured in manipulated bytes per second) to enhance efficiency. This will be accomplished by minimizing encryption-decryption times.

1.4 Limitation and Hypothesis of The Thesis

One method for improving the efficiency of popular lightweight algorithms is to design a specific piece of hardware to handle the issue. Nonetheless, there are a number of restrictions that may apply, including:

1. Synchronization Overheads: Parallel lightweight encryption methods may encounter limitations due to the need for synchronization between parallel processes. When communication between the threads is increased, performance decreases.
2. Data Dependency: Limitations may arise due to data dependencies between parallel processing units, leading to potential bottlenecks and reduced efficiency in scenarios where data dependencies are high.

Furthermore, Scalability is much easier, especially when the parallel algorithm is well-designed. It might be hypothesized that the scalability of the parallel lightweight encryption method will depend on factors such as the size of the dataset, suggesting that the method's effectiveness scales with larger datasets. Moreover, a typical personal computer system contains a multi-core processor that has eight cores or more; therefore, it is hypothesized to make use of all of them to enhance performance.

1.5 Question of thesis

The lightweight proposed parallel encryption method is designed to efficiently encrypt data while allowing parallel processing. Questions related to such a thesis could include:

- What considerations were taken into account to ensure the encryption method is lightweight?
- How does the proposed encryption method compare to Speck lightweight methods in terms of security?
- What are the performance metrics used to evaluate the efficiency of the parallel lightweight encryption method?
- What challenges were encountered in parallelizing the encryption method, and how were they addressed?

1.6 Literature Review

In this section, we present related works focusing on encryption methods executed on various parallel platforms with the aim of increasing throughput. Throughput refers to manipulating bytes per second to maximize efficiency, with the main goal of accelerating previous encryption methods while maintaining a high level of security.

In 2022, A. Fanfakh et al. [7] introduced a dynamic resource technique for a new encryption called "ORSCA" that only required one round. The suggested cryptosystem was specifically designed to consider GPU peculiarities and is suitable for large-scale applications. Results showed it could process more data than other approaches, achieving a capacity of about 5 terabits per second on a Tesla A100 GPU. This cryptography surpasses the strongest GPU counterparts of AES, Simons, and Speck, making it highly applicable in real-world scenarios.

In 2021, L. Sleem and R. Couturier [1][8] The proposed method that is used in the IOT field is Speck, which was designed by the National Security Agency (NSA) in June 2013. In this paper, we propose a new ultra-lightweight cryptographic algorithm based on Speck, known as Speck-R. Speck-R is a hybrid cipher, combining the ARX architecture with a dynamic substitution layer. It used 64-bit blocks, CTR mode, and the 96-bit Speck version. The study's primary success is the decrease in the number of Speck rounds from 26 to 7, while still retaining a good degree of security. By reducing the number of repeats, execution times will decrease.

In 2021, M. Gupta and A. Sinha [9] presented Enhanced AES (EAES), a unique way of partitioning the AES S-box to perform simultaneous multithreaded replacement of two bytes at once. The proposed EAES algorithm outperforms other encryption methods, including RC5, Blowfish, and Skipjack, against various base processes, key lengths, and rounds, enhancing the network lifetime and throughput of Wireless Sensor Networks (WSN).

In 2021, A. Adil Yazdeen and in 2020, M. Nabil et al. [10][11] proposed pipelining and parallelization techniques to accelerate AES encryption on FPGA devices. By breaking up the AES encryption process into smaller stages and using multiple AES encryption cores, they were able to achieve higher clock frequencies and improved performance.

In 2020, R. Couturier et al. [12] proposed "ESSENCE," a lightweight stream cipher scheme based on a dynamic key approach. ESSENCE outperforms available AES implementations on GPUs and offers a high level of unpredictability, frequency, and key responsiveness, making it a viable stream cipher option.

In 2020, W. El Hadj et al. [13] presented 32-bit datapath, LED 64/128, SIMON 64/128, and SIMECK 64/128 algorithms with optimized hardware and compact cryptographic architectures for devices with constrained resources. The architectures

were compared based on size, speed, efficiency, and power consumption on different FPGA systems.

In 2019, A. S. Anil et al.[14] proposed The block cipher families of Speck and Simon are especially made to provide security on devices with few resources, where design simplicity is crucial. Both Simon and Spock are available in multiple key sizes and widths. Furthermore, by allowing a variable block size, it lessens the inefficiencies of encrypting marginally longer texts. As part of our project, we simulate and synthesize this algorithm after analyzing the "Simon and Speck" block cipher family.

In 2019, N. Alassaf et al. [15] introduce a SIMON-based lightweight cryptography algorithm for IoT systems, emphasizing speed improvement while maintaining a balance of security and efficiency. Their proposed work was compared to AES and basic SIMON cryptosystems in terms of execution time and memory consumption.

In 2018, A. Rajallah et al.[16][17] assessed the performance of the blowfish method in a parallel environment using the MPI standard. The experiments showed that the blowfish system's runtime decreased and speed increased as the number of processors grew.

In 2018, W. Xuguang et al. [18] introduced the Parallel Encryption Scheme (PLMES) for encrypting large data sets or long messages in parallel. PLMES divides a long message into smaller blocks, encrypts each block independently in parallel, and then concatenates them to form the final encrypted message. This technique has been widely used in various applications, including secure communication, cloud computing, and big data processing.

Table(1.1): Comparison between Parallel Encryption Techniques

Ref	Algorithm	Structure	Key Size	Block Size	#Rounds	architecture parallel	Throughput
[1][8]	**Speck-R**	FN	96 bits	64 bits	26	-	-
[9]	**EAES**	SPN	128,192, 256 bits	128 bits	10,12, 14	Multi core processors	-
[10][11]	**AES**	SPN	128,192, 256 bits	128 bits	10,12, 14	FPGA platform, Pipeline	-
[12]	**ESSENCE**	-	512 bits	128 bits	One	GPU	higher than 115 GB/s
[13]	**LED**	SPN	128 bits	64 bits	48	FPGA Platform	891.99 Mbps
[13]	**SIMECK**	FN	128 bits	64 bits	44	FPGA Platform	210.13 Mbps
[14]	**Simon**	FN	128 bits	64 bits	44	FPGA Platform	838.95 Mbps
[15]	**Speck**	FN	128 bits	128 bits	32	-	-
[16][17]	**Blowfish**	FN	32-448 bits	64 bits	16,18	Multi core processors	-
Proposed	**One round**	-	512 bits	128 bits	One	Multi core CPU	higher than 25 and 10 GB/s

1.7 Thesis Outline

In addition to this chapter, the thesis consists of four chapters a follows:

Chapter Two: This chapter presents the theoretical background of encryption, covering the foundational aspects of the research. It includes definitions of symmetric and asymmetric ciphers, as well as lightweight parallel cryptography algorithms. Additionally, cryptography fundamentals are studied, encompassing security analysis, performance metrics, and randomness tests used to evaluate parallel encryption. Furthermore, a description of the parallel architectures applicable at both hardware and software levels is provided.

Chapter Three: This chapter outlines the research approach and the suggested methods. It involves examining the programs created and implemented to achieve optimal results in terms of compute execution time, throughput, and speedup, followed by a detailed analysis of the experimental findings.

Chapter Four: This chapter presents the results of the proposed method along with in-depth discussions and interpretations.

Chapter Five: This chapter contains the conclusions drawn from the research findings and outlines potential avenues for future work and improvements.

Chapter Two

Theoretical Background

2.1 Introduction

With the development of new security attacks that take advantage of the attackers' increased processing capacity, data security is experiencing growing difficulties. Cryptography may help you protect your data from hackers more efficiently[19]. Cryptography is the practice and study of techniques for secure communication in the presence of third parties, known as adversaries[20]. It involves transforming information (referred to as plaintext) into an unreadable form (cipher text) to prevent unauthorized access and then transforming the cipher text back into the original form (plaintext) for authorized access [21–22].

Cryptology, whose primary goal is to safeguard data from malicious users, can be divided into two main branches: symmetric and asymmetric cryptography[23], which our works focuses on symmetric cryptography that explain below in this chapter. Using the principles of parallel computing, a large message may be broken down into more manageable chunks that can then be distributed among several processors. Numerous different parallel cryptographic techniques are being investigated to provide secure communication. New lightweight cryptography methods for parallel implementation over multicore processors are covered in this chapter, as described below in figure 2.1. Also, this chapter provides the cryptography foundations, which include security analysis, cryptography performance metrics, and randomness tests. Finally, this chapter provides the classification of parallel architectures using Flynn’s classification. Both the shared and distributed parallel memory architectures and their types are demonstrated too.

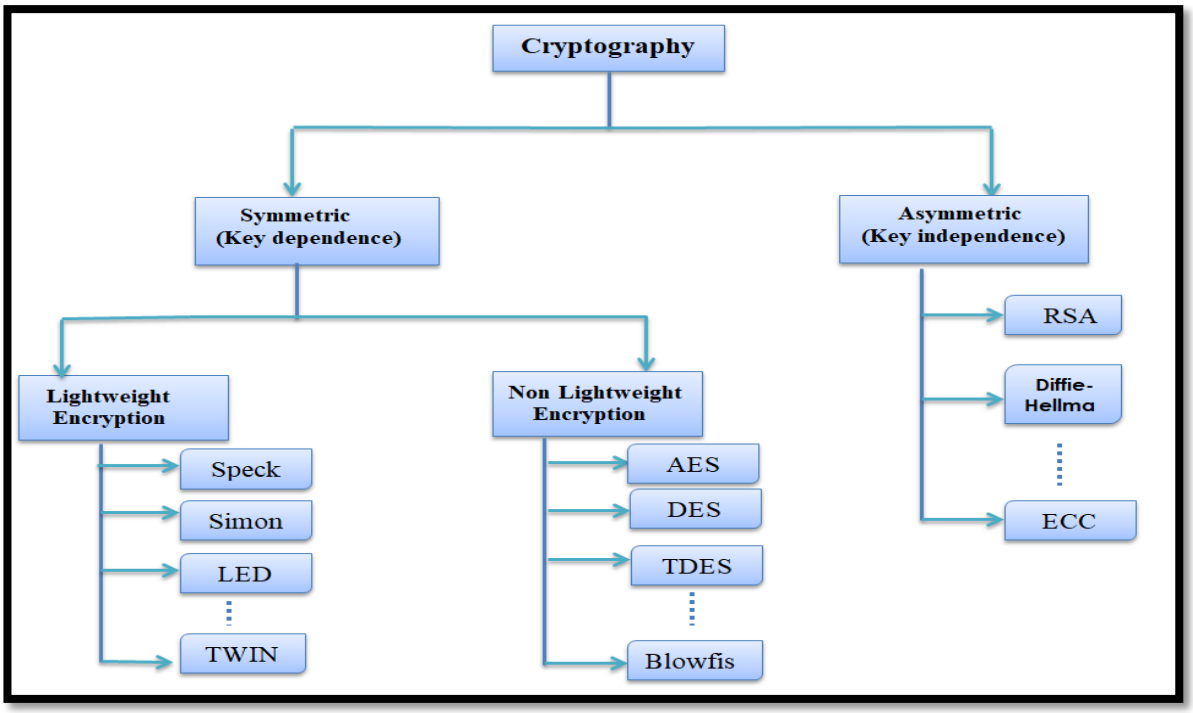


Figure (2.1): illustrates the categorized Encryption Techniques

2.2 lightweight Cryptography

Lightweight cryptography is the branch of new cryptography, which is the cover of cryptographic algorithms intended for using pervasive devices with low resources. LWC does not locate tough criteria for classifying a cryptographic algorithm as lightweight, but the common countenance of light-weight algorithms has very low requirements for main resources for target devices [23]. Lightweight cryptography (LWC) aims to facilitate a wide range of applications, such as smart cards and RFID tags, wireless sensor networks, embedded systems, wireless patient monitoring, the Internet of Things (IoT), vehicle security systems, intelligent transportation systems (ITS), and other applications [24–25]. This is made more difficult by the fact that these devices often exhibit direct physical interaction with the world through corresponding actuators, which could jeopardize user security in the event of misuse [26–27]. In this section, the provided literature survey on the use of lightweight symmetric

cryptosystems covers recently released hardware and software. Symmetric lightweight ciphers are encryption methods that are made to be lightweight, which means they use little computing power and have a small amount of code. These ciphers are often used in places with limited memory, processing power, and energy, such as embedded systems, Internet of Things (IoT) devices, and low-power devices[28]. In this section, Lightweight Cryptography is suggested as a way to deal with these problems that normal and traditional ciphers can't handle, especially in the Internet of Things (IoT). These programs must work with the limitations of the hardware and software and keep a high level of protection at the same time. Below, we list most of the lightweight methods that are described in the figure (2.2). Symmetric ciphers include block and stream. We describe many different methods for lightweight cryptography primitives.

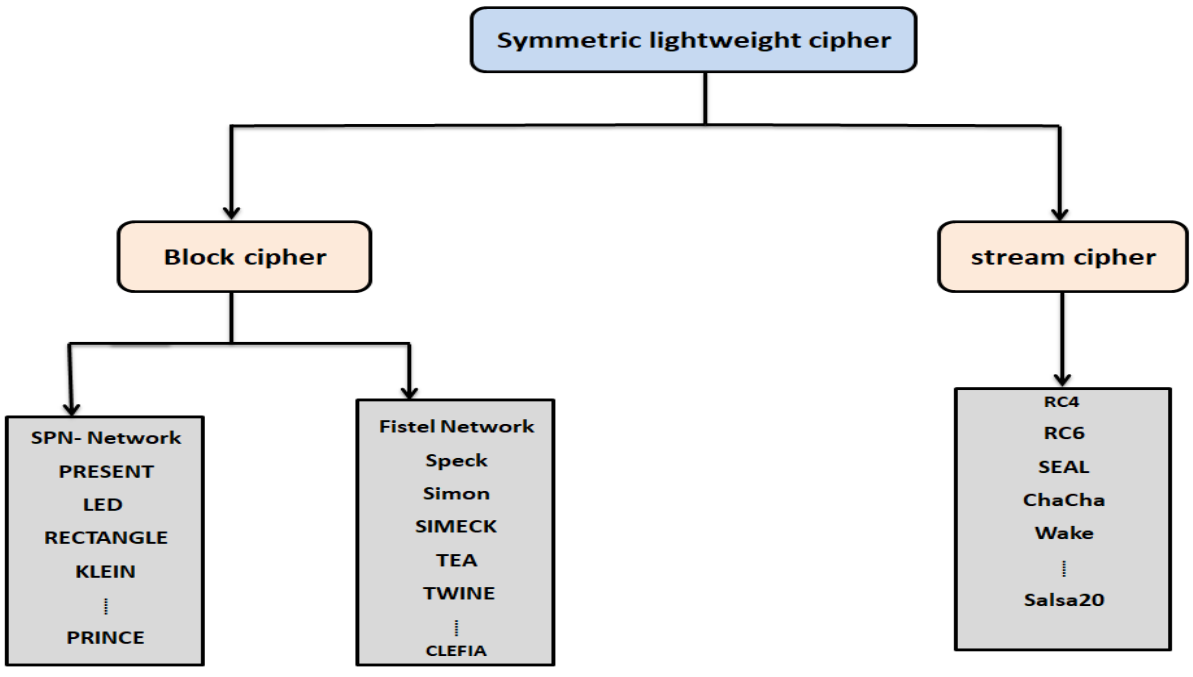


Figure (2.2): Symmetric Lightweight Cryptography

2.2.1 Lightweight Block Ciphers

Block ciphers are cryptographic algorithms that function by encrypting data in blocks of a predetermined size. They encrypt the message by first separating it into separate blocks, which are generally of a predetermined length (such as 64 or 128 bits), and then encrypting each block individually. In order to conduct a number of encryption rounds, block ciphers need a key. These encryption rounds often entail operations such as substitution and permutation. The output of one block is dependent on the output of the blocks that came before it as the encryption key. Most lightweight block ciphers have smaller block sizes and key lengths than their traditional versions. This makes encryption and decoding faster and uses less memory. One famous example is the SIMON and SPECK block cipher families, which were released by the National Security Agency (NSA) in 2015. SIMON is optimized for hardware implementation, while SPECK is optimized for software implementation. Block ciphers use a secret key to do both encryption and decryption. The same key is used for both steps, which is why the process is called "symmetric encryption." Most of the time, the key has a set amount that fits the needs of the block cipher method, as shown in figure 2.3. The following are the five different categories that are used to classify the different construction types of block ciphers: substitution permutation networks (SPN), Feistel networks (FN), add-rotate-XOR (ARX), NLFSR-based, and hybrid [29], and they are widely used for various security applications. Here are some common applications of block ciphers, such as virtual private networks (VPNs), secure email and messaging, secure web communication, authentication and digital signatures, IoT security, military and government communications, and other applications. Block ciphers are a fundamental building block of modern cryptography and play a crucial role in securing a wide range of applications and systems, providing confidentiality and data integrity[30]. However, the main reason behind using block cipher is that it is easy to implement in parallel and has the possibility of processing more than one block in parallel at the same time, unlike stream cipher, and there are several modes in it, namely:

1. Modes of Block Encryption: In order to properly encrypt messages, block ciphers are required to be paired with other security methods of functioning. An additional prerequisite is that an adversary should not be able to ascertain, by virtue of the fact that the key is changed for each block, that the same two blocks have been encrypted with the same key (which is an action that cannot be treated as trivia). There are five different kinds of operations that may be performed using block cipher modes: the ECB (Electronic Code Block) mode, the CBC (Cipher Block Chaining) mode, the CFB (Cipher Feedback) mode, the OFB (Output Feedback) mode, and the CTR (Counter) mode. Each of these modes performs a different function. The counter (CTR [LRW00]) and cipher block chaining (CBC [EMST76]) modes are two of the most used modes. In addition, authentication modes such as the O-Set Encoding Mode and the OCB [KR11] developed by Krovetz and Rogaway may be used to generate encryption algorithm blocks [31].

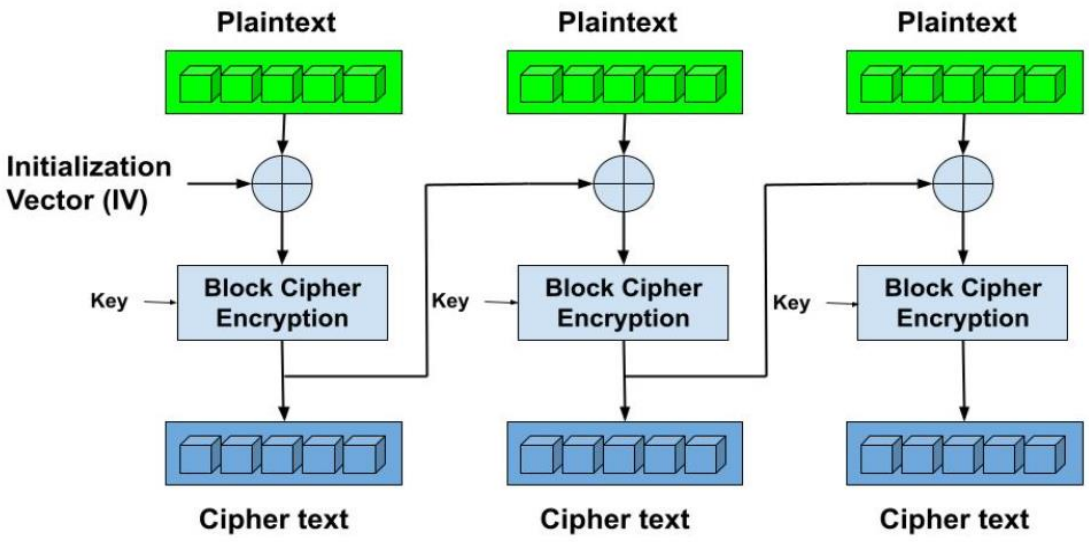


Figure (2.3): Block ciphers [31]

2.2.2 Lightweight Stream Cipher

Stream ciphers encrypt data in a continuous stream, usually one bit or one byte at a time. This is different from block ciphers, which encrypt data in blocks of a set size. Lightweight stream ciphers are also seen as efficient and promising for IoT environments with limited resources. Compared to lightweight block methods, the stream cipher is said to be faster, easier, and more customizable in hardware applications. It is based on two constructions, such as linear feedback shift registers (LFSRs) and non-linear feedback shift registers (NLFSRs)[28]. The cipher text is created by performing a bitwise exclusive (XOR) operation on the plaintext after it has been mixed with a stream of pseudo-random bits or bytes that are generated by the algorithm. When data is encrypted or decrypted, the same stream of bits is used for both processes. Stream ciphers were developed as a result, and they proved to be the answer to all of these issues. They achieve this by using a small secret key that can be readily passed around between the parties in order to produce a pseudo-random stream of bit values, as shown in figure (2.4). The RC4 technique, which is used in protocols such as SSL and WEP, is an example of one of the stream ciphers that is used the most often. However, it has been discovered that RC4 has several security flaws, and as a result, more recent stream ciphers such as ChaCha20 and Salsa20 have been created as substitutes. uses a 256-bit key and a 64-bit nonce (number used once) to generate a stream of pseudorandom bits that are combined with the plaintext stream using a bitwise XOR operation[31].

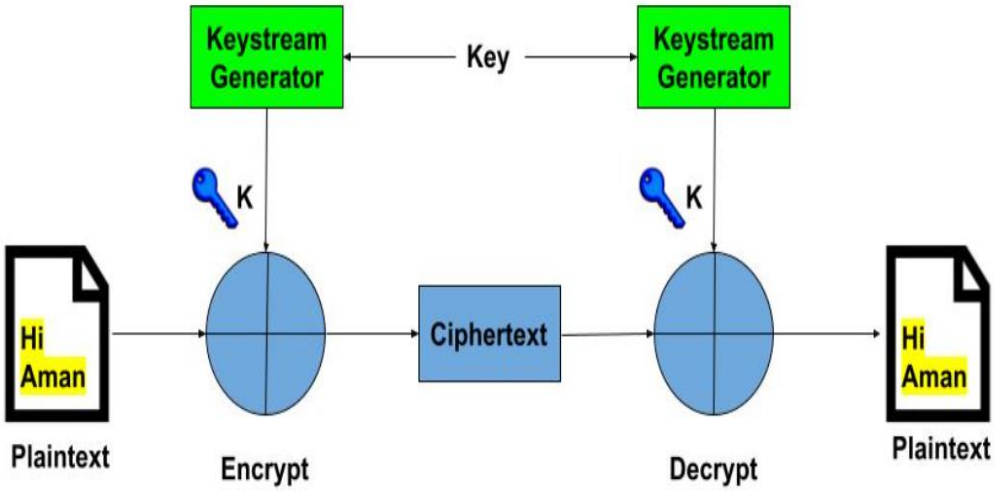


Figure (2.4): Stream Ciphers[31]

2.3 Lightweight Cryptography Algorithm

2.3.1 Speck Cryptography

The NSA (National Security Agency) made the lightweight block cipher family Speck public in June 2013. Speck has undergone performance improvements for software applications. The add-rotate-xor (ARX) cipher is what the Speck cipher uses[32]. Speck is a family of block ciphers with different key sizes and block sizes. The most common versions of Speck are Speck32/64, Speck48/72, and Speck64/128. The numbers in the names refer to the block size and key size of the cipher. The Speck cipher is represented as Speck2n/wn, where it has a 2n-bit block and a wn-bit key [33-34]. The generalized Feistel structure was implemented in Speck and has five distinct block sizes (32, 48, 64, 96, and 128). These can be further subdivided into ten versions using a size key (see Table 2.1). The XOR, modulo addition, and rotation operations make up the round function. In Figure 2.5, a general round function of Speck is shown, with Li and Ri representing the left and right half intermediate values of the input for the i_{th} iteration, respectively. The n-bit key used in the i_{th} round is called Ki. The circular left

and right shifts are indicated by the bits α and β, respectively. The XOR operation is called L, and the modulo addition is called Li+1 and Ri+1 are the outputs for the ith round, and the round function is as follows[1][8]:

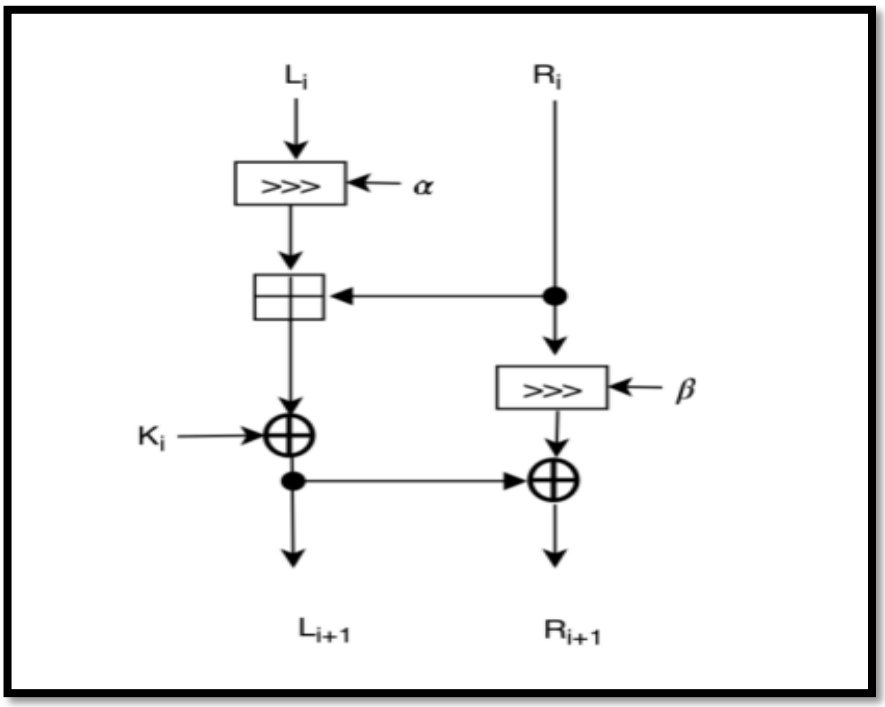


Figure (2.5): Speak Round Function [1][8]

Table 2.1: Speck parameter

block size	key sizes	Rounds
32	64	22
48	72,96	22,23
64	96, 128	26,27
96	96, 144	28,29
128	128, 192, 256	32,33,34

2.3.2 Simon Cryptography

Simon is a lightweight block cipher designed for hardware and software environments with constrained resources, such as Internet of Things (IoT) devices. In 2013, the National Security Agency (NSA) developed it and made it available to the general public[35]. The 10 Speck instances were developed with exceptional hardware and software performance in mind, with particular attention placed on the performance of the microcontrollers. Employ the same nomenclature for each of the various types of speck that Speck does. Simon comes in different versions with varying block sizes and key sizes, including Simon 32/64, Simon 48/72, and Simon 64/128, where the numbers in their names refer to the cipher's block size and key size. The Simon cipher uses a Feistel network structure and involves the use of bitwise XOR, bitwise rotation, and bitwise AND operations, as shown in Figure 2.6. Simon is known for its low memory requirements, high speed, and security against various attacks[36].

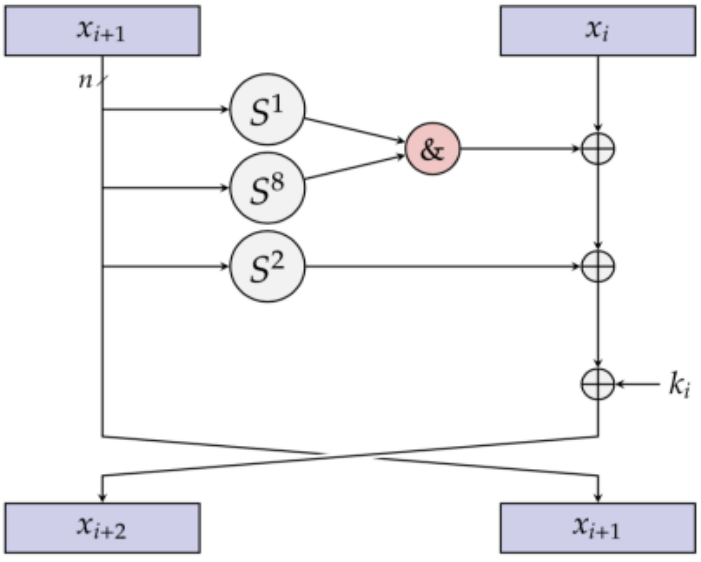


Figure (2.6): Simon Round Function [35]

2.3.3 RC4 Cryptography

RC4 is a stream cipher that was invented by Ron Rivest in 1987, specifically for use by RSA Security. It is a stream cipher with byte-oriented operations that may use keys of varying sizes. The technique relies on the use of a random permutation as its foundation. It is a lightweight encryption technique, which implies that it is meant to be implemented on devices with limited processing power and memory. Some examples of such devices are embedded systems or smart cards. RC4 encrypts and decrypts data one byte at a time by using a key stream that it creates based on a key with a configurable length (usually between 40 and 256 bits)[36]. The key stream is formed by repeatedly applying a key-scheduling algorithm, also known as a KSA(see algorithm 2.1), to the key. This results in a permutation of the integers 0 to 255 being generated, as shown in figure (2.7). The key stream is then generated by applying a pseudo-random generation algorithm (PRGA) to the permutation in order to produce the key stream [37-38].

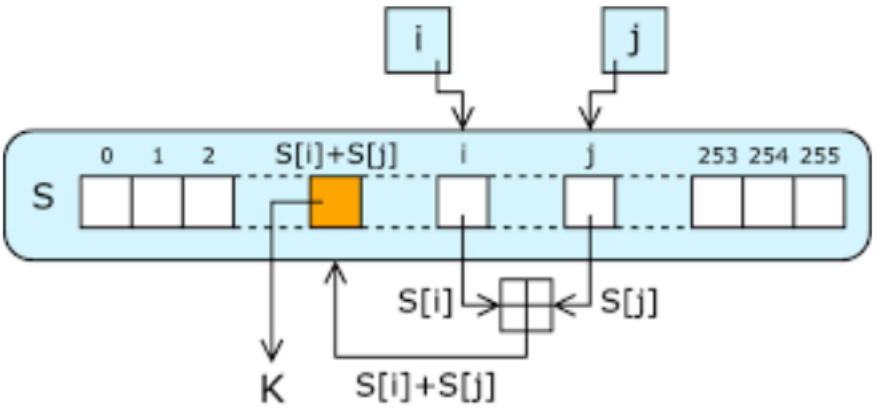


Figure (2.7): RC4 Encryption [39]

The algorithm (2.1) presented takes 256-byte permutation array referred to as "S" and an identically sized array referred to as "T," both of which are initialized with the key. Both arrays have the same number of bytes. After the elements of array T have been initialized with values ranging from 0 to 255 in ascending order, an algorithm that is

key-dependent is used to initialize the elements of array S. This algorithm conducts a series of swap operations depending on the key bytes. The term "key scheduling algorithm" is often used to refer to this operation. The RC4 approach that is provided below is called algorithm 5, and it begins with the initialization of an array SC that has 256 elements and values ranging from 0 to 255. Proceed through the components of SC in order, starting with 0 and ending with 255. After doing so, calculate the index j0 by adding the value of sc[i0] to the key byte size_DK that corresponds to the current index i. Finally, assign the current index i to the variable i0. Take the result of the addition and divide it by 256 (so that the result will wrap around if it is more than 255). Last but not least, you need to switch the values that are stored in sc[i0] and sc[j0]. The array sc represents the state that RC4 is in now that it has been key-scheduled, and it is prepared to produce the pseudo-random stream.

Algorithm 2.1: RC4 Algorithm

Inputs: key , size_DK

output: *SC*

1: For i = 0 to 255 (increment by 1):

2: SC[i] ⟵ *i*

3: End For

4: initialize variable J ⟵ *0*

5: For L = 0 to 255 *(increment by 1):*

6: *Compute* J = (J + SC[L] + key[L % size_DK])& 0Xff

7: *initialize variable* tmp ⟵ SC[L]

8: Sawp SC[L] = SC[J]

9: SC[J] ⟵ tmp

10: End For

2.4 Pseudo Random Generator Algorithms

2.4.1 Xorshift64 PRNG

The xorshift64 is a sort of pseudorandom number generator (PRNG) that generates a series of random numbers via the use of bitwise XOR (exclusive OR) and bit-shifting operations. This type of PRNG is also known as a cryptographic random number generator. It uses a 64-bit state to do its operations, and at each step, it modifies the state in order to generate the next random number. This technique relies on linear-feedback shift registers, sometimes known as LFSRs, which are outlined in algorithm 2.2. Because of Xorshift, an efficient implementation may be achieved without resorting to sparse polynomials in excess. Because of this, they are very fast on any contemporary computer architecture[40-42].

Algorithm 2.2: Xorshift64 PRNG Algorithm [40]

Inputs: *state (reference to an unsigned, 64-bit integer).*

output: *x (64-bit unsigned integer*

Start

1. *Replace the value that state is pointing to with x.*
2. *Combine x with the 13-bit left shift of x by performing an XOR operation.*
3. *Combine the output of step 2 with x's right shift of 7 bits using the XOR process.*
4. *Combine the output of step 3 with x's left shift of 17 bits using the XOR process.*
5. *Add the value of x to the value to which state is pointing.*
6. *Return the value of x.*

End

For more information on the Xorshift64 PRNG, a digital example explains the steps of the algorithm in detail, as shown in figure 2.8.

Step 1: if state is 0x5A4A6927F56EFD5B(64 bit in hexadecimal)

Step 2: After this step, x becomes 0x5A4A6927F56EFD5B ⊕ (0x5A4A6927F56EFD5B << 13) = 0x247B3DD8AED2E0E1.

Step 3: After this step, x becomes 0x247B3DD8AED2E0E1⊕ (0x247B3DD8AED2E0E1 >> 7) = 0x21E96B862BD59382.

Step 4: After this step, x becomes 0x21E96B862BD59382 ⊕ (0x21E96B862BD59382 << 17) = 0xCAF0ED9EDEE41E82.

Step 5: x = 0xCAF0ED9EDEE41E82.

Figure (2.8): Xorshift64 PRNG

2.4.1 Splitmix64 PRNG

SplitMix64 is a fast and highly robust 64-bit random number generator. The "Splitmix-64" PRNG has a lot of advantages, including a relatively short execution time and an easy implementation. These are only two of its many merits. Algorithm 2.3 describes the SplitMix64 PRNG, which is a quick split table PRNG. It has input and output states that are both 64 bits in size. However, because the generated sequences of pseudo-random values can be predicted, they are not recommended for use in cryptography or other security applications. This is because the mixing functions are invertible, and two successive outputs provide enough information to reconstruct the internal state of the generator. The suggested method uses SplitMix64 and dynamic seeds to create a transformation that builds a linear transformation depending on shift and rotation. As a result, there is a substantial decrease in latency and a substantial amount of unpredictability in the resources that are pertinent. The suggest block cipher uses a huge key space, and the use of dynamic cryptographic primitives provides a high degree of security[40-42].

Algorithm 2.3: Splitmix6464 PRNG Algorithm [40]

Inputs: *index (a 64-bit unsigned integer pointer)*

Outputs*:* z *(64-bit unsigned integer)*

Start

1. *A value referred to by index is added by UINT64_C (0x9E3779B97F4A7C15), and the result is then assigned to z.*
2. *Execute an XOR operation between z and z with a 30-bit right shift.*
3. *Assign the result of step 2 to z after multiplying it by UINT64_C (0xBF58476D1CE4E5B9).*
4. *Use an XOR operation to combine z with the 27-bit right shift of z.*
5. *Assign the result of step 4 to z after multiplying it by UINT64_C (0x94D049BB133111EB).*
6. *Execute an XOR operation using z and the 31-bit right shift of z.*
7. *Return the value of z.*
8. *End*

2.5 Parallel Computing

The term "parallel programming" refers to a method of computing that depends on the simultaneous execution of program components. This improves program performance and minimizes the amount of time needed to complete the program. Especially in recent years, the function of synchronization has grown significantly in order to boost computer performance. Simulations, data processing, scientific modeling, and rendering are all examples of computationally demanding operations that may greatly benefit from parallel computing[43]. First, the development of the processors adopted Moore's law to increase performance by increasing the number of transistors in the integrated chip. Next, another technique was adopted that attempted to implement implicit parallelism by executing more than one instruction in one cycle, creating

pipelines, and increasing the frequency of processors. At this time, multi-core technologies are being used as a means of accomplishing the goal while simultaneously expanding the number of cores contained inside a single CPU. However, the level of parallelism in parallel computing refers to the degree of concurrency or the number of tasks that can be executed simultaneously [6]. The parallelism level can be categorized into different types that are described below in this section: additional, there are different models and architectures for parallel computing, including shared memory systems and distributed memory systems:

2.5.1. Shared Memory of Parallel Architectures

A shared memory architecture is a kind of parallel computing that enables several processors or cores to access and operate on a shared address space. Shared memory architectures are also known as memory sharing. In this design, each processor has its own private cache or local memory, but in addition to this, they are able to access a shared portion of memory that is located in the same physical location[44]. A specialized, high-speed connection network, such as a high-speed bus, enables parallel access to any portion of the memory from any CPU. Because a single operating system is responsible for managing all of the processors, it is possible for them to interact and stay in sync with one another by reading and writing to shared variables. Shard memory, which also include multicore and GPU architecture, are an example of this kind of design[45].

2.5.1.1 Multi-core Processors

A multi-core processor is a kind of computer processor that has numerous independent processing units, also known as cores, on a single integrated circuit (IC) or chip[46]. The multi-core design integrates many processing cores onto a single chip in order to do multiple tasks simultaneously. As a result of this, it is also known as a chip

multiprocessor. Each core in a multi-core architectural design has its own execution pipeline, and each core has the resources required to operate without obstructing the resources that the other software threads require. The one and only computing component is a multi-core processor, which consists of two or more CPUs that read and execute the actual program instructions[47]. All of these multicores are able to interact with one another by using shared memory, as can be seen in figure 2.9. The ability of individual cores to carry out numerous instructions in parallel results in an improvement in the overall performance of software that has been optimized to make the most of the specific architecture. Calculations and the execution of programs may take place more quickly on a multi-core processor than on a single processor chip due to the presence of many processing units inside the chip[40]. The majority of today's personal computers, tablets, and smartphones all make use of multi-core processors, which allow these devices to operate far more swiftly than they could with just one central processing chip [7].

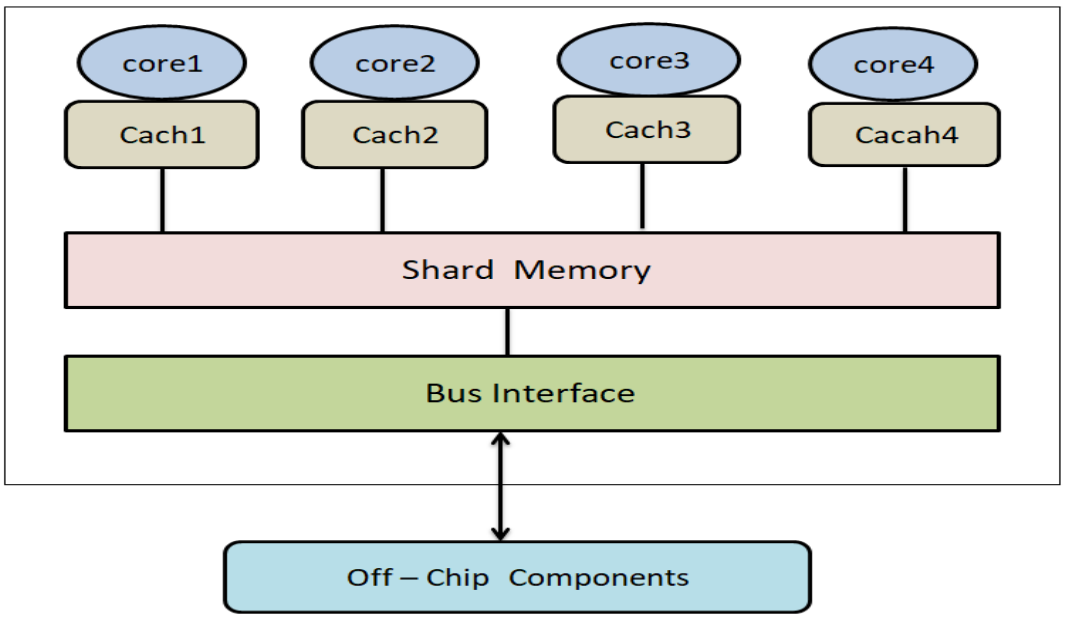


Figure (2.9): Multi-core Architecture

2.6 Cryptography Tests

2.6.1 Randomness tests

In this section, we'll talk about how crucial statistical randomization tests are for assessing the randomness of the encrypted output that cryptographic algorithms produce. These tests are crucial in assessing the crypt's ability to withstand cryptographic analysis and attacks. The testing of cryptographic keychain randomness is a critical and valuable activity that helps determine the quality of implementation. By conducting these tests, we can identify any weaknesses or vulnerabilities in the cryptographic algorithm and take steps to improve its implementation. Therefore, statistical randomization tests play a vital role in ensuring the security and effectiveness of cryptographic systems. Randomness is a critical component of cryptographic algorithms, and statistical randomization tests can help evaluate whether the generated key stream meets the necessary randomized and consistency requirements. Randomness is achieved through a probabilistic process that produces independent, uniformly dispersed, and unpredictable qualities that cannot be consistently repeated. Random sequences, however, may have distinct statistical characteristics that can be evaluated using various statistical methods. There are several tools available for conducting these statistical tests, including TestU01, Practrand, Diehard, ENT, Die Harder, and NIST STS, among others. Let us now discuss these random tests in more detail [48][49]. The most popular method for calculating randomness is the PractRand test.

- **PractRand** (Practically Random) is a software package written in C that includes both pseudo-random number generators (RNGs) and statistical measures for evaluating RNGs. It is the only test suite that can conduct tests with functionally limitless durations. PractRand also requires extra data to detect skew compared to other test suites. PractRand provides a valuable tool for developers to test the quality of their RNGs and ensure that they meet the necessary randomness requirements for use in cryptographic algorithms. However, this test is regarded as the most difficult system[48].

2.6.2 Cryptography Performance Metrics

The performance measurement stage is a highly significant one since it enables the identification of the bottlenecks in the cryptography and the subsequent attempt to reduce those bottlenecks in order to increase the cryptography algorithm's performance. Throughput capacity, execution time, and speed are three metrics that are used to evaluate the performance of the system. The amount of time an application takes to complete its tasks is a critical factor for each of these metrics[2]. The main goal of this work is to study the performance cost of the suggested encryption method in parallel and compare it to different lightweight approaches to encrypting data on CPU-based devices. This section presents an analysis of the performance based on parallel throughput, encryption time, and speedup metrics for both the Speck, Simon, and proposed ciphers. In addition, we have the ability to choose the amount of accuracy of the algorithm as well as the level of parallelism that will provide the best results in a parallel environment. When increasing the number of processing units that are working together to solve the problem, it is important to gain a deeper understanding of the scalability factor in the algorithm.

2.6.2.1 The Execution Time

The execution time of parallel cryptographic algorithms can vary widely depending on several factors, such as the specific algorithm used, the hardware platform on which it is executed, the size of the input data, and the specific implementation of the algorithm. For sequential execution for an algorithm that encrypts or decrypts the message of size *n*, where each block takes time *Ti*, which is the time to encrypt all blocks as follows:

$$\boldsymbol{Tseq} = \sum_{\boldsymbol{i=0}}^{\boldsymbol{n-1}} \boldsymbol{Ti} \qquad (\mathbf{2.1})$$

In the case of parallel execution, the execution time is calculated as the maximum time of the slower task that encrypts or decrypts the message block, as follows:

$$T_{Par} = Max_{i=0}^{n-1} (T_i) \qquad (2.2)$$

However, modern hardware platforms and optimized software implementations can help reduce the execution time of cryptographic algorithms. For example, hardware-accelerated cryptographic modules or specialized cryptographic co-processors can significantly speed up the execution of certain cryptographic operations.

2.6.2.2 Throughput

The throughput of a parallel cryptographic algorithm is a measure of the amount of data that can be encrypted or decrypted per unit of time. Typically, the goal of parallel algorithms is to achieve a higher throughput than sequential algorithms such as speck and Simon by leveraging parallel processing architectures, such as multi-core CPUs or GPUs. In general, the more processing cores available, the higher the potential throughput of the algorithm. On the other hand, Another way to increase the throughput of a parallel cryptographic algorithm is to use specialized hardware, such as FPGAs or ASICs, that is optimized for specific cryptographic algorithms and can perform encryption and decryption operations in parallel[31]. During the period of time T devoted to the execution of the algorithm, the throughput of encrypting or decrypting a message with a length of *N* bits is calculated as follows:

$$Throughput\ (GigabitPerSecond) = \frac{Size\ of\ message\ (Gigabit)}{Execution\ \ time\ (sec)} \qquad (2.3)$$

The equation shown above is used by a large number of researchers within the scholarly literature, including.

2.6.2.3 Speedup

Parallel cryptography methods are cryptographic algorithms that are designed to take advantage of parallel processing architectures, such as multi-core CPUs or GPUs, to achieve faster encryption and decryption speeds[16]. The speedup of parallel cryptography methods can be measured in terms of the reduction in computational time compared to traditional sequential cryptographic algorithms. Typically, the goal is to achieve a significant speedup without sacrificing security. The ratio of the delay time required for sequential execution to the delay time required for the parallel version's execution represents the speedup provided by the parallel algorithm[16]:

$$\boldsymbol{Speedup = \frac{Tseq}{Tpar}} \qquad (\mathbf{2.4})$$

Where *Tseq* represents the amount of time spent in the execution delay of the sequential version, and *Tpar* represents the amount of time spent in the execution delay of the parallel version. The value of the speedup, on the other hand, may also be expressed as a multiple, for instance. 2 times, or 2 times the current speed. This signifies that the speed will be twice as fast. This will offer a sense of the scalability of the method by assessing the degree to which the speedup may alter when the number of processing units is varied[16].

2.6.3 Security Analysis Tests

In this section, a security analysis is carried out to verify the resiliency of the suggested encryption. In fact, these experiments demonstrate that the cipher that was presented may also be used for the purpose of encrypting images that include data that is strongly linked but has distinct intrinsic properties. These tests will demonstrate its resistance to a variety of secrecy assaults, including statistical, differential, linear, non-linear, and brute-force attacks, among others. Extensive tests are carried out in this part to demonstrate the resilience of the suggested cipher system. Although the proposed

encryption method can be applied to any type of data, only the results for multimedia content are shown here for clarity[1].

2.6.3.1 Statistical Analysis Test

The effectiveness of a cipher may often be evaluated through the use of statistical analysis techniques. In these tests, the cipher text is scrutinized in search of statistical patterns or abnormalities that have the potential to divulge information on the plaintext or the encryption key. A cipher scheme requires specific random properties in order to efficiently resist statistical attacks. Randomness and uniformity are the two most important characteristics that a cipher has to have in order to be considered secure against statistical attacks. Text cryptanalysis should make use of techniques such as probability density function (PDF) analysis, histogram analysis, entropy analysis, and correlation between the original and encrypted texts in order to be carried out effectively[7].

2.6.3.1.1 Histogram Analysis

Histogram analysis is one of the statistical analysis tests used to analyze the cipher message of a cipher. It entails producing a histogram of the frequency bits or other components found in the cipher text and comparing it to the distribution that would be predicted for the language that was used in the plain massage. This encryption satisfies the uniformity condition only if the encrypted image has a histogram with a uniform distribution. This indicates that the frequency with which each symbol appears is proportional to the total number of symbols in the message. In other words, must close to message-size / number of symbols. if we have a two-dimensional image with a size n*n , the analysis of the histogram of the original plain compares with their cipher-image counterparts, it's shown, demonstrated to be quite close to a uniform distribution. If the histogram of the cipher text reveals a non-uniform distribution of symbols or other components, this may be an indication that the encryption technique does not use a randomization method that is robust enough.

2.6.3.1.2 Uniformity Analysis

Uniformity analysis of cipher messages is a technique used to determine the randomness and uniformity of the distribution of characters in the encrypted message. The study of the probability density function (PDF) was the parameter that was used so that the uniformity of the ciphered picture could be shown. The likelihood of an individual value appearing in the cipher text is described in the PDF. One common PDF analysis test is to calculate the frequency distribution of each symbol in the cipher text. This can be done by counting the number of times each letter appears in the text and dividing by the total number of letters in the text. This means it is necessary for each symbol to have a probability that is somewhat near 1/n, where n is the total number of symbols [7]. The encrypted image PDFs can be thought of as having a nearly uniform distribution.

2.6.3.1.3 Entropy Analysis

Entropy analysis is a popular statistical analysis test used to assess the randomness or unpredictability of cipher text. Entropy analysis's fundamental goal is to gauge the degree of uncertainty or information present in a transmission. The message's informational entropy *M* is a metric that defines the degree of a random variable's uncertainty as follows[1]:

$$\boldsymbol{H(m) = -\sum_{i=1}^{n} p(mi)\, log_2 \frac{1}{p(mi)}} \qquad \textbf{(2.5)}$$

Where *H* is the entropy, *p(Mi)* is the probability of the mi symbol occurring in the message, and $\log_2$ is the logarithm base 2. The binary secret data should have an entropy value of 1 or close to it for optimum encryption. It is important to take into account that the entropy is stated in bits. The entropy is measured at the sub-matrix level using the suggested entropy test, where each sub-matrix has a size of h^2 bytes, where h^2 is the total number of symbols. This enables homogeneity to be measured at the sub-matrix level rather than the whole picture. If a block's entropy is equal to or near $log_2 (h^2)$, it may be seen as a genuine random source that follows a standard distribution pattern[1][8].

2.6.3.1.4 Correlation Test

The correlation test is an essential test, particularly in the area of encryption, due to the significance of concealing information from an attacker and preventing an attacker from reading plain message information from a ciphered message. The correlation test, which compares cipher and plain message versions of the same information, achieves this. In light of this, the correlation of the suggested cryptosystem needs to be as low as is practically achievable in order to accomplish the goal of achieving security [7]. If the correlation coefficient is close to 1, it suggests a strong correlation between the cipher message and the expected plain message frequencies, which in turn suggests that the encryption method may be vulnerable to frequency analysis attacks. If the correlation coefficient is close to 0, it suggests no correlation between the cipher and the expected plain message frequencies, which indicates a stronger encryption method. This test is carried out by selecting *N* neighboring pixels from both the plain picture and the ciphered image and putting them into the following equation as inputs[50]:

$$Pxy = \frac{Cov(x,y)}{\sqrt{E(x) \times E(y)}} \quad (2.6)$$

$$Cov\ (x,y) = \frac{1}{N} \times \sum_{k=1}^{n} (xi - C(x))\ (yi - C(y)) \quad (2.7)$$

$$Cx = \frac{1}{N} \times \sum_{k=1}^{n} xi \quad (2.8)$$

$$Ex = \frac{1}{N} \times \sum_{k=1}^{n} (xi - C(x))^2 \quad (2.9)$$

Pxy is the correlation coefficient between the two sequences *x* and *y*, and it was used in the preceding formulae. *xi* and *yi*, respectively, show what the values of *x* and *y* are. We utilized a correlation analysis of nearby pixels for both the ciphered picture and the original image in order to demonstrate that the ciphered image is an entirely unique representation of the original image[50].

2.6.3.2 Sensitivity Tests

Performing a sensitivity analysis on cipher text often entails conducting research on the effect that alterations or adjustments made to the cipher text have on the plaintext that was obtained by decrypting the ciphertext. The purpose of this investigation is to evaluate the cryptographic system's safety and authenticity while also gaining an understanding of how changes to the cipher text could influence the plaintext that is ultimately produced. Differential attacks include analyzing the relationship that exists between two encrypted communications that have resulted from a tiny alteration, such as a difference of one bit, that has been introduced between the original messages. The sensitivity tests have to demonstrate that a very small alteration to either the plain image or the key causes a distinct variation in the cipher image that is produced. The sensitivity of the encryption technique improves in direct proportion to the difference between the two values[7].

2.6.3.2.1 Key Avalanche Effect

The main avalanche test is considered one of the most significant tests. Using this test, the sensitivity of the ciphertext (or the sensitivity of the key stream in the case of block cipher) to a tiny variation in the secret key may be measured and quantified. Since the suggested key derivation function is based on the secret key and the nonce, it is able to guarantee a high degree of sensitivity for both the secret key and the nonce. Two different dynamic keys, *DK1* and *DK2*, which differ from one another by a single random bit, are employed in this investigation of the sensitivity of the dynamic key. After that, the identical plaintext is encrypted in two different ways, and the Hamming distance between the resulting cipher texts, *C1* and *C2*, is calculated using a total of one thousand random dynamic keys[7][12].

2.6.3.2.2 Message Avalanche Effect (Sensitivity)

The same plain picture is used as input to the method, but it generates a totally new cipher image each time since a different dynamic key is utilized each time. Therefore, the encryption that has been presented successfully meets the avalanche criterion[7][12].

2.9 Summary

This chapter provides in-depth information on symmetric lightweight algorithms, such as speck and Simon algorithms. Also, pseudo-random generator algorithms. This chapter describes the parallel architectures that may be used on the hardware or software level in parallelism. These architectures can be found at different levels of parallelism. After that, Flynn's classification is broken down and discussed in terms of the link that exists between the memory model and the compute units. The components of parallel memory architectures might be categorized according to whether they are distributed or shared. Finally, it also includes a description of the security analysis and performance analysis to assess the cryptography algorithm. Furthermore, this chapter offers an overview of the statistical ciphers' randomness tests that define the randomness of the output cipher text.

Chapter Three

The Proposed Method

3.1 Introduction

This chapter provides an introduction to the proposed parallel cipher, which makes use of a multi-core processor in parallel to improve encryption performance. Through the use of the Linux operating system and the message-passing interface MPI for the purpose of parallelization on a computer system. However, this chapter describes the proposed one-round block encryption algorithm in depth and explains its structure in terms of compute elements and parallel platform implementation. in order to achieve ideal results in terms of compute execution time, throughput, and speedup, followed by a thorough examination of the experiment's results.

3.2 Proposed One Round Cryptography Algorithm

This section presents the proposed one-round cipher algorithm, designed to outperform the multi-round Speck cipher. The algorithm has been implemented in parallel to enhance encryption performance. To assess its effectiveness, the proposed cipher underwent randomness tests, including the PractRand test, and its performance was compared to well-known lightweight encryption algorithms such as Speck. The proposed system consists of three main steps: mixing, pseudo-random number generation (PNRG), and substitution. Each step has been meticulously designed to ensure both the security and efficiency of the cipher. Algorithm (3.1) illustrates the general encryption process, highlighting the various steps involved. Figure (3.1) showcases the scheme of the proposed one-round block cipher.

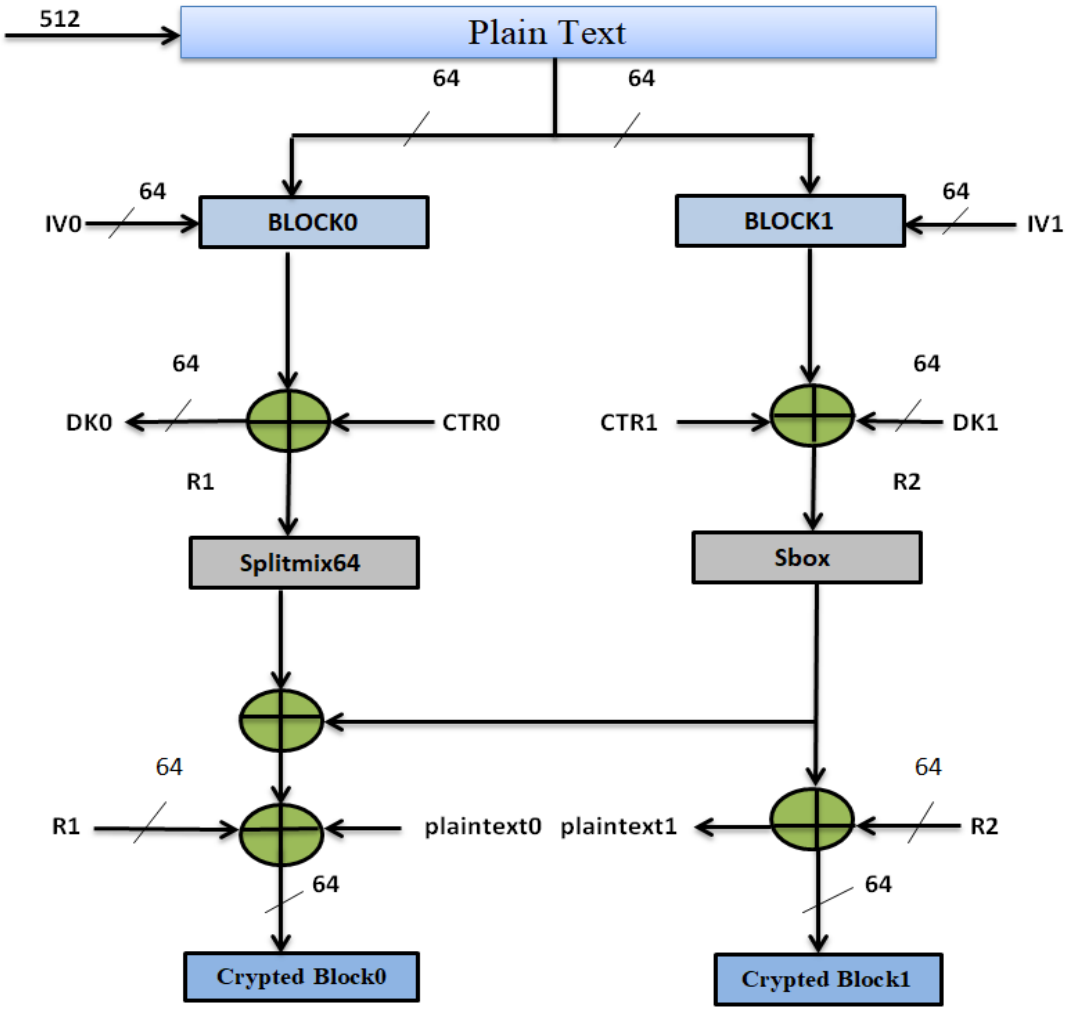


Figure 3.1: Scheme of the Proposed One-Round Block Cipher

Algorithm (3.1) is intended to carry out a one-round encryption on an input message that is separated into two 64-bit blocks, denoted by the variables (R1,R2). The input undergoes bitwise XOR operations with the key, block counter CTR, and initial vector using the PRNG xorshift64. The resulting value is stored in another variable. Temporary variables R1 and R2 are then calculated by performing XOR operations between the initial variable (i) and the dynamic key. The initial vector is generated through the splitmix64 algorithm with a 64-bit size, as described below in Chapter 2 of Section 2.4.2. Also, the dynamic key is generated through the xorshift64 algorithm, as described below in Chapter 2 of Section 2.4.1. This is followed by applying the splitmix64

function to R1. R2 undergoes byte substitution using Sbox1, which replaces each byte with a new value based on a lookup table that is generated by the RC4 algorithm, which is described below in Chapter 2 of Section 2.3.3. Subsequently, XOR operations are performed between R1 and R2, and the result is stored in R1. Finally, the encrypted values are stored in the output array called "out". The parameters presented in algorithm are (*, &), which means as follows:

1. * is the pointer operator of the original array.
2. & means mode operation

Algorithm (3.1): Proposed Encryption Algorithm

Input:

in: plain block array of size 64-bit

Sbox1: arrays of size 256 bytes representing substitution boxes

DK: array of size 512 bytes representing dynamic key

v: array of initial vector of size 64-bit

start: Starting index for the encryption loop

end: Ending index for the encryption loop

Output:

out: encrypted block array of size 64-bit

Start

Encryption

1: For i = start to end (increment by 2):

2: R1 = (i) ⊕ (((ulong) DK) [i&63])) ⊕ v[i]*

3: R2 = ((i+1) ⊕ (((ulong) DK) [i&63])) ⊕ v[i+1]*

4: R1 = Splitmix64 (R1)

5: R2 = Substitution (R2, Sbox)

6: R1 = R1 ⊕ R2

7: out [k] = R2 ⊕ in [k]

8: out [k+1] = R1 ⊕ in [k+1]

9: increasing k by 2

10:End For

11: End Encryption

12: End

3.2.1 Dynamic Key Generation Method

The proposed solution is based on the dynamic key-dependent methodology, where a set of dynamic cryptographic primitives is produced by using a dynamic key DK. (Substitution tables along with a collection of N seeds, where each seed may be a word made up of 32 or 64 bits.) An initial vector and CTR (counter mode), which need to be updated and unique for each transmission, are combined to create the dynamic key Dk. This procedure is illustrated in Figure 3.2, and it is outlined in the equation as follows:

$$DK = (IV \oplus CTR \oplus key) \qquad (3.1)$$

Where DK is the dynamic key with size 512, IV is the initial vector of size 64 bit that is generated by splitmix64, CTR is the index of the block number, and key is the static array of size 64 bit, then bitwise operation (XOR) of them. This dynamic key, also known as DK, is subdivided into three different subkeys, the first two of which are identical. (K1 and K2) have a length of 128 bits, while the third sub-key (Kseed) has a length of 256 bits. To generate a dynamic key (see algorithm 2.1 of chapter 2).

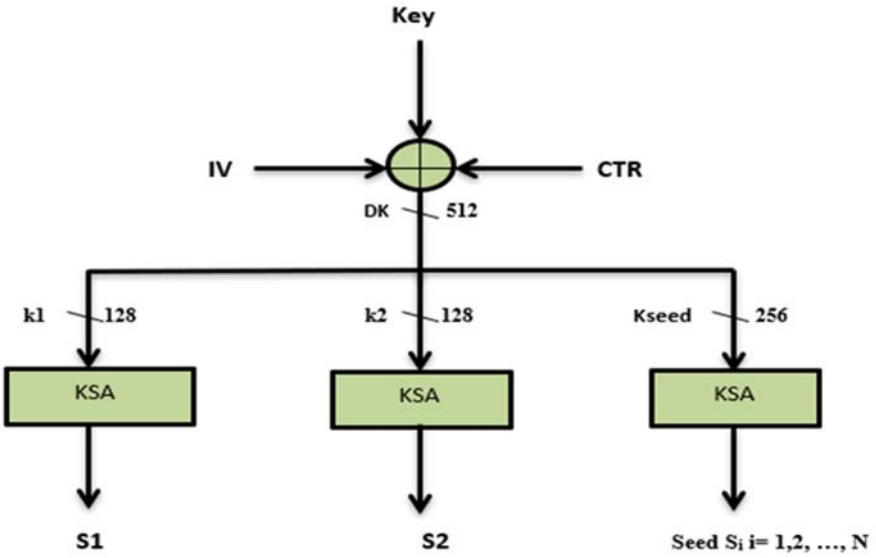


Figure 3.2: Displays the Proposed Dynamic Key Generation and Construction Cryptographic Primitives

The following provides an explanation of these sub-keys:

1. K1 represents the first 128 most important bits of DK and is the first replacement sub-key. This is the case because K1 is the first subkey. S1 stands for the first substitution table, which makes use of this sub-key. In this stage, you are free to employ any technique you choose to generate dynamic-key-dependent substitution tables. For instance, the Key Setup Algorithm (KSA) of RC4 was implemented as described in (chapter 2 of section 2.3.3). Therefore, dynamic substitution tables could be created. At this point, we also make use of the KSA algorithm that is used by RC4.

2. K2 stands for the second of DK's 128 least significant bits and is the second replacement sub-key. After K1 comes K2. The second substitution table, known as S2, is generated using the same procedure that was used to create K1, which is the KSA for RC4.

3. KSeed represents the initial 256 most significant bits (MSB) of DK. A key stream with a length of N words is created using this sub-key as a hidden seed in conjunction with any PRNG, where each word's length can range between 32 and 64 bits. Because of this, KSeed is where one can acquire N seeds. Each process will choose one of these produced samples, which will be generated in a manner that is dynamically simulated to be random.

The below-mentioned algorithm is a function named intial_seed, which looks to initialize various variables and data structures using a combination of the xorshift64 and splitmix64 pseudorandom number generators in addition to an RC4 key generation algorithm (RC4). The algorithm takes a 64-bit unsigned integer seed as input, where the seed value is the random value of size 64 that is used as input in the algorithm. Within the loop, it makes a call to the xorshift64 function that is described below in (Chapter 2 of Section 2.4.1), with seed+i as the argument, and then it assigns the value that is returned to the element of the array DK that is located at index i. It would seem that DK is an array of unsigned integers with 64 bits of precision. Following the iteration of the loop, the value of the seed is increased by 512. The malloc function is used at this point in the algorithm to allocate memory for an array IV that contains n 64-bit unsigned integers. Using the malloc memory allocation method, it also allots memory for an array Sbox that contains 256 unsigned characters (uchar). A new loop is started, and this one will iterate for the specified number of times (from j = 0 to n). Within the loop, it makes a call to the splitmix64 function described below (Chapter 2 of Section 2.4.2), with seed +j as the input, and then it assigns the value that is returned to the iv array at index j. After the iteration, n is added to the value of the seed variable. At last, the code executes the RC4 function as described below in (Chapter 2 of Section 2.3.3), supplying as inputs the number 256, the Sbox variable, and the address of the beginning of the DK array, starting at index 256

Algorithm(3.2): Intial_Seed Algorithm

Inputs: *Seed , n*

output: *DK , initial vector (IV), Sbox*

1: For i = 0 to 511 (increment by 1):

*2: DK[i] ⟵ xorshift64 (Seed +i) …… **(see algorithm 2 of chapter 2)***

3: Increasing Seed by 512

4: Iv =(uint64_t)malloc(sizeof(uint64_t)*n)*

5: Sbox=(uchar)malloc(256*sizeof(uchar))*

6: For j = 0 to n (increment by 1):

*7: IV[j] ⟵ Splitmix64(Seed + j) …… (**see algorithm 3 of chapter 2)***

*8: **End For***

9: increasing Seed by n

*10: Call RC4 (DK[256], Sbox, 256) …… **(see algorithm 1 of chapter 2)***

*11: **End For***

3.3 The Proposed Parallel Encryption Method

To provide clarity on the operational process of a one-round cipher, Figure 3.3 presents a graphical illustration of the parallel implementation of the proposed cipher. The plain messages are divided into groups of blocks, with each block having a size of 64 bits. The block size, blsize, is computed by dividing the message size by the number of parallel threads. Subsequently, all sub-blocks have the same size, and each one is sent to its respective thread. The encryption algorithm is then applied to each thread, generating a set of encrypted blocks. Moreover, all encrypted blocks are gathered asynchronously, as shown in algorithm (3.3).

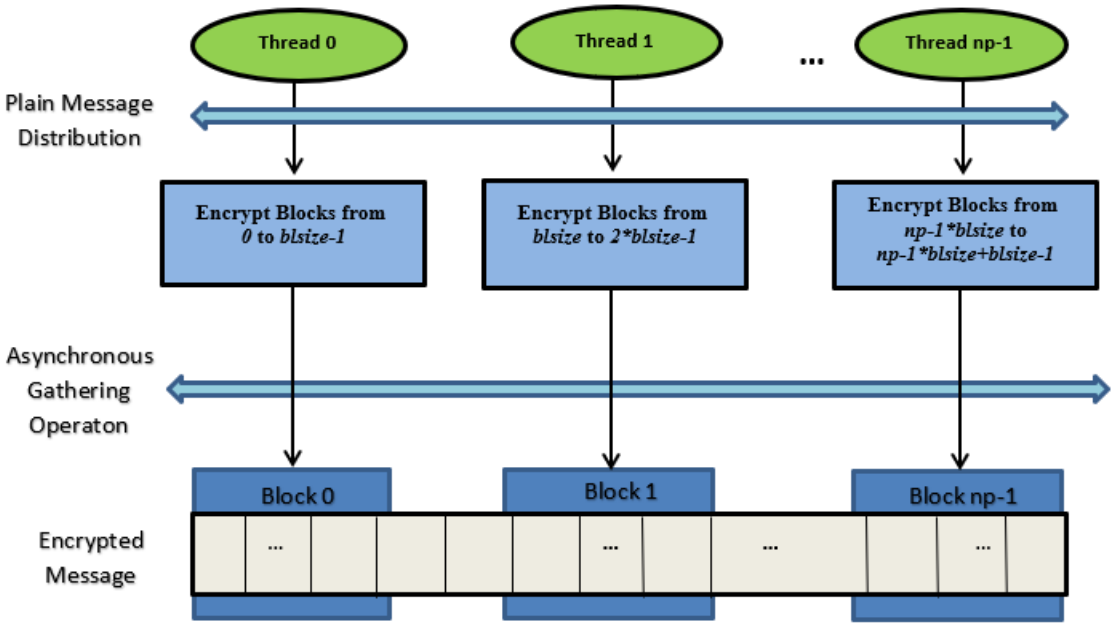


Figure 3.3: The Parallel Encryption Method

The proposed parallel encryption method is a function that takes in several parameters, as shown in algorithm 6. In the following description of the algorithm parameters:

- IV: A pointer to the 64-bit initialization vector
- Sbox: A pointer to an array of 256 bytes that represents an S-box substitution table.
- ThreadId: the index of the parallel thread dedicated to each block of data.
- Blocksize is the size of block data that each parallel thread computes.
- plain is an array of 64-bit elements representing the plain message.
- encrypt: A subarray of a 64-bit output buffer to store the encrypted cipher text.
- crypted_Block0 and crypted_Block1 are the final output for the encrypted blocks per iteration.

Algorithm (3.3): Parallel One-Round Cipher Encryption Algorithm

Inputs: initial vector (V), Sbox, ThreadId, Blocksize, DK, plain, cipher

output: crypted_Block0, crypted_Block1

1: Set k = 0

2: Start = ThreadId*blocksize;

3: End = start+(blocksize-1)

4: Call Encryption ……. (See algorithm 3.1)

12: End For

13: Call MPI_Igather (cipher, bl_size, MPI_INT64_T, allcipher, blocksize, MPI_INT64_T,0, MPI_COMM_WORLD)

The algorithm (3.3) presented above is a parallel one-round encryption algorithm for block ciphers. The algorithm takes an initial vector (V), an Sbox substitution table, a ThreadId, a Blocksize, a DK (derived key), plaintext (plain), and an encryption function as inputs and produces two crypted blocks (crypted_Block0, crypted_Block1). The algorithm operates on each block of the plain message in parallel by partitioning the plain message into sub-blocks of size 'blocksize' and assigning each sub-block to a different thread identified by 'ThreadId'. Each thread processes its assigned sub-block in parallel as output. The algorithm iteratively processes each sub-block by first computing R1 and R2 based on the initial vector V, the derived key DK, and the sub-block index j. R1 and R2 are then subject to Splitmix64 and Substitution operations, respectively. The resulting R1 and R2 values are then combined using an XOR operation, and the resulting values are used to encrypt two plain values (j and j+1) using the provided

encryption function. The resulting cipher message values are stored in the 'encrypt' array. This process is repeated for all plaintext sub-blocks assigned to the current thread. After processing all sub-blocks assigned to the current thread, the algorithm uses MPI_Igather to collect the resulting cipher message values from all threads and store them in the 'allcipher' array. The gathered cipher message values are of size 'blocksize' and are of type MPI_INT64_T from all threads in the buffer called allcipher. Asynchronous gathering is used to reduce communication times. The presented algorithm is a parallel block cipher encryption algorithm that operates on plain messages in parallel, making it suitable for use in systems that require efficient and scalable encryption of large volumes of data.

3.4 Summary

This chapter presents the details of the proposed one-round block encryption algorithm and explains its structure in terms of the parts of the computation and its implementation on a parallel platform. The applications of parallel message-passing operating on a multicore CPU are used. It includes reviewing and implementing programmers to get optimal outcomes in terms of compute execution time, throughput, and speedup, followed by a complete analysis of the results of the experiment.

Chapter Four

Experimental Results

4.1 Introduction

This chapter presents the results of testing the Speck parallel algorithm in comparison with the results of testing the parallel suggested algorithm using two distinct processors, namely the Intel(R) Core (TM) i7-7700HQ and the Intel(R) Core (TM) i7-6600U, which they denoted in this chapter CPU1 and CPU2 respectively. In order to validate the efficacy of the parallel procedure suggested, many kinds of tests, such as analyses of performance and security, were carried out. Examples of these tests include The PDF test, the histogram test, the entropy test, and the correlation test. These are just a few examples of the several tests that fall under this category. Tests for sensitivity and randomization are also included in the additional security checks. Other testing included measuring the speed, throughput, and execution time of the performance analysis. With this research, we want to investigate how the different sizes affect the metrics, specifically.

4.2 Experiments Setting

This section discusses the findings of Speck parallel cryptography, which made use of two unique types of processors and just a single key length of 128/128 bits. There is a distinct number of rounds for the speck cryptography (32 rounds), and these results are compared with the parallel recommended algorithm results with one round and a dynamic key of 512 sizes. The process that led to these findings was carried out on a computer running the Linux operating system and making use of the MPI platform. In particular, three performance metrics are investigated in the trials by varying the number of threads (2, 4, 6, and 8) and message sizes (4, 8, 16, 32, 64, 128, and 256 megabytes). The proposed parallel encryption and lightweight Speck algorithms are evaluated on CPU1, which features an Intel (R) core (TM) i7-7700HQ CPU, to determine their performance. The processor has four physical cores and eight logical ones. The frequency of each core is 2.8 GHz. In addition, the rating includes the CPU2, which

features an Intel (R) Core (TM) i7-6600U CPU with a base frequency of 2.60 GHz and a maximum frequency of 2.81 GHz, respectively, and is used in this study. These are the two versions of Intel Core i7 chips that are currently available. The information on the two processors that were used in the studies may be found in the table that follows (4.1).

Table (4.1): Technical Details of the Computing Processors

Processor Name	Frequency speed (GHz)	main memory Size (GB)	Operational Use
Intel Core i7- 7700HQ	2.8	16	Laptop Processor
Intel Core i7-6600U	2.8	8	Desktop Processor

4.3 Experiments Results

The results were divided into two parts according to the stages obtained them, security and performance analysis results.

4.3.1 Performance Analysis Results

In this section, the suggested encryption method is conducted in parallel and compared to the speck lightweight method for encrypting data on two CPUs. To assess the performance of this newly proposed encryption algorithm in a parallel computing environment, two different processors were used to conduct these experiments. This section presents an analysis of the results based on parallel throughput, encryption time,

and speedup metrics for both the Speck and proposed ciphers, which were implemented on (2, 4, 6, and 8 threads) in conjunction with different message sizes (4, 8, 16, 32, 64, 128, and 256 megabytes). As a concluding note, Tables (4.2) and (4.3) display the average of the comparison results for execution time and throughput across all message sizes.

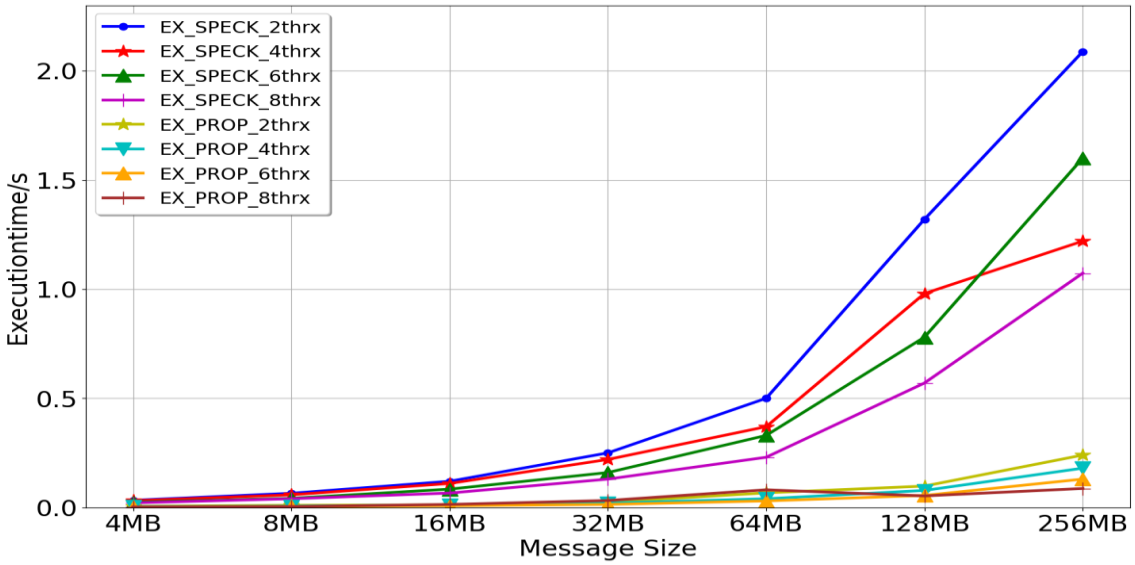
Table (4.2): Comparison Results on CPU1

#threads	Average overall message sizes			
	Execution time (s)		Throughput Gbits/s	
	Proposed	Speck	Proposed	Speck
2	0.098	2.088	11.062	1.069
4	0.18	1.22	19.69	1.95
6	0.13	1.60	20. 31	1.94
8	0.087	1.073	25.69	2.44

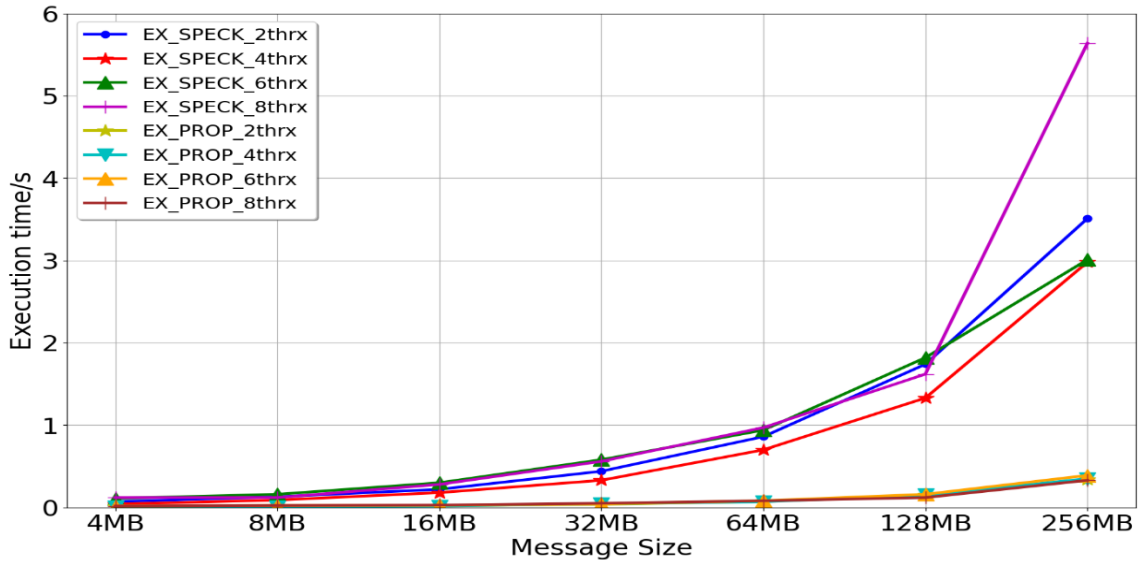
Table (4.3): Comparison Results on CPU2

#threads	Average overall message sizes			
	Execution time (s)		Throughput Gbits/s	
	Proposed	Speck	Proposed	Speck
2	0.34	2.088	7.420	3.51
4	0.35	1.220	9.00٤	2.98
6	0.39	1.600	8.044	3.01
8	0.33	1.073	8.790	5.64

Figure (4.1): Display the results of the execution time to encrypt or decrypt a message of different sizes executed over different numbers of parallel threads. We chose seven different sizes of data, from 4 megabytes to 256 megabytes. According to the figure, we can see that the encryption time goes down as the number of threads goes up. This can be seen clearly when executing over more parallel threads. But the collected data showed that the proposed consecutive algorithm for the one-round cipher took less time to process than the Speck algorithm for different message sizes when it was run on two multicore processors.



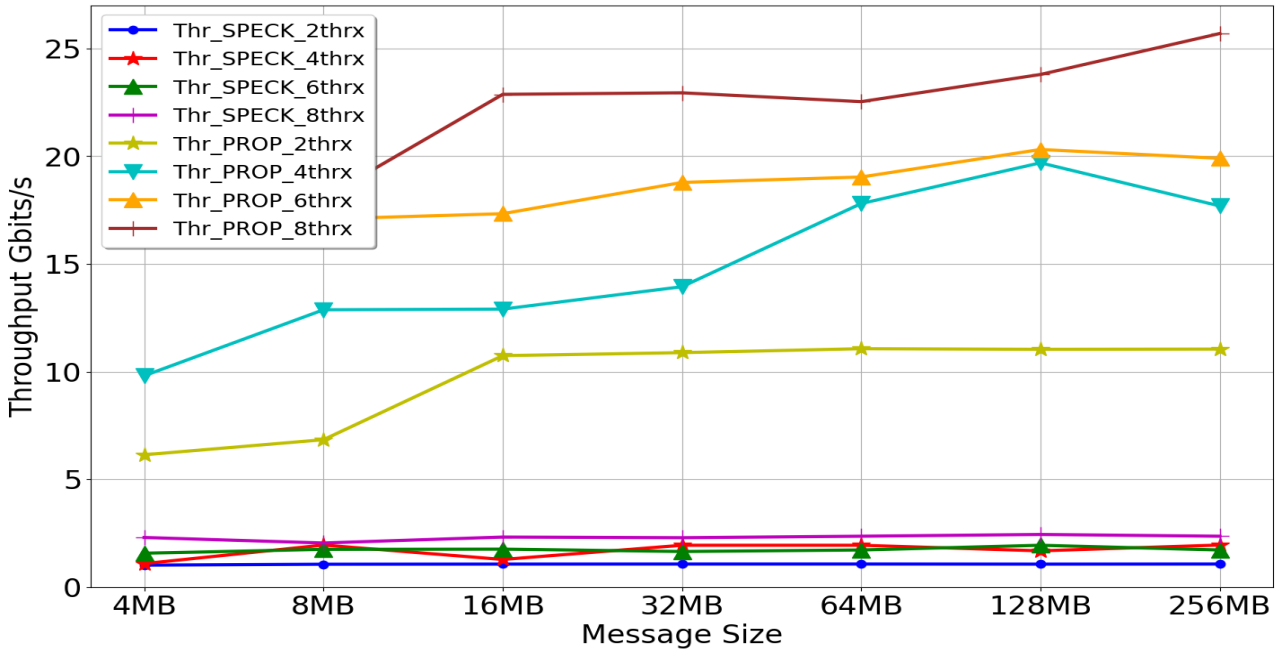
CPU1



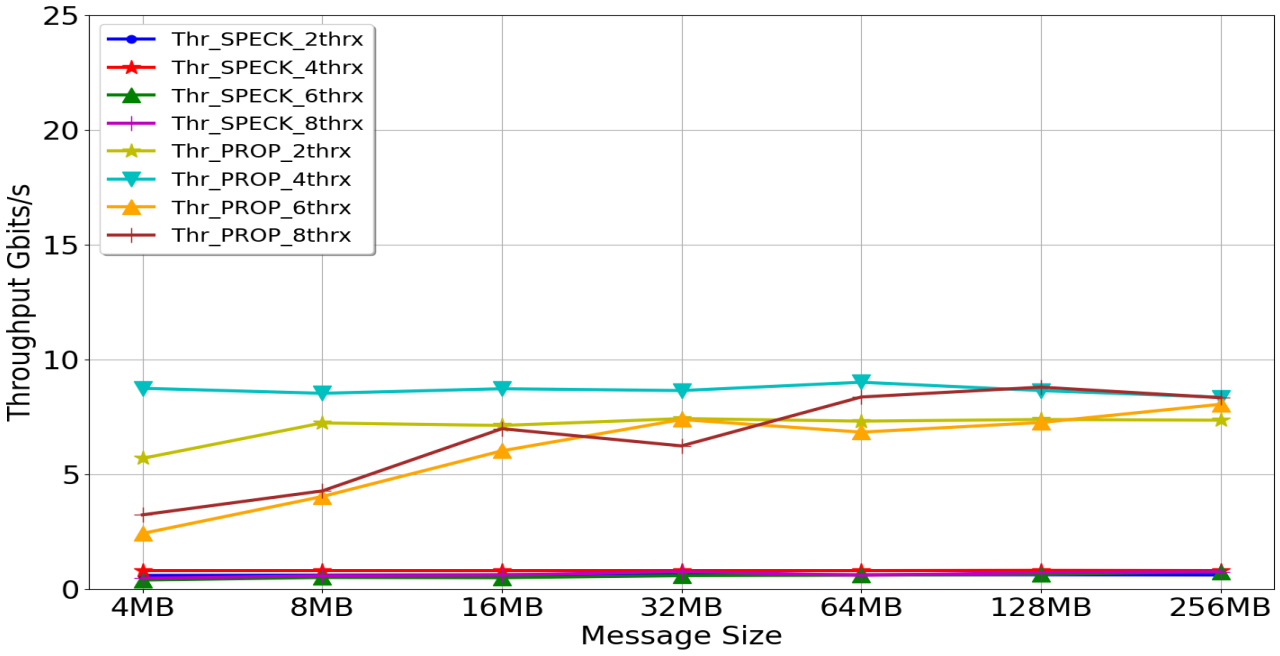
CPU2

Figure (4.1): The Execution Time Comparison Results

The presented results in Figure 4.2 show that the proposed encryption technique is able to reach a high data throughput in comparison to some lightweight methods that already exist, with a throughput that is higher than 25 and 9 Gigabits per second on two Intel Core i7 central processing units. The actual transmission rates achieved are subject to variability based on the quantity of data subjected to encryption and the computational ability of the CPU. The obtained results demonstrate that the suggested algorithm for the one-round cypher offers superior performance in comparison to stream encryption algorithms utilized for a range of message sizes when it is run on two multicore processors.



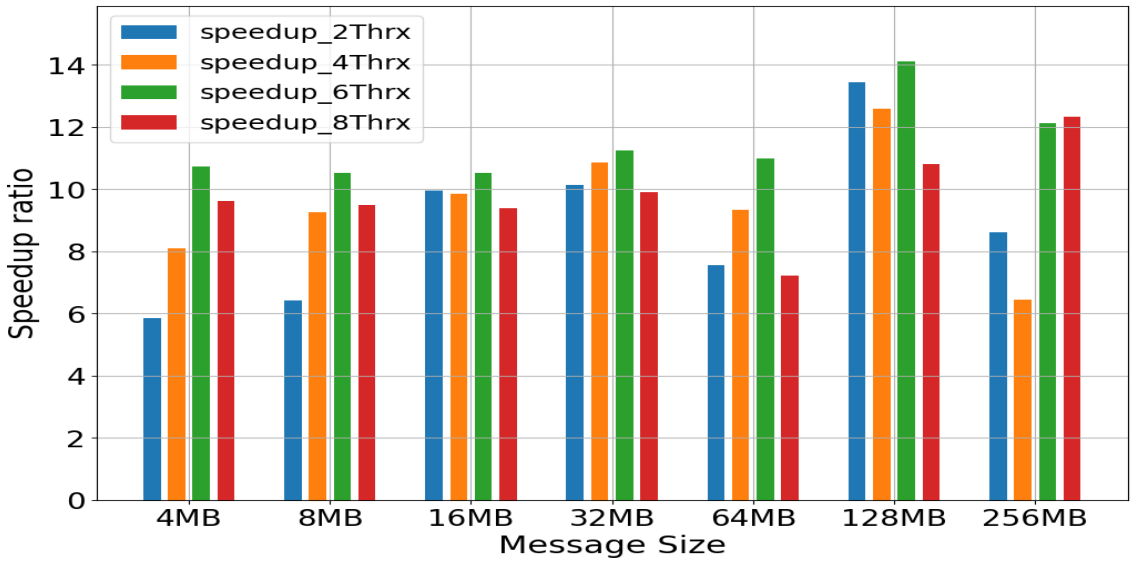
CPU1



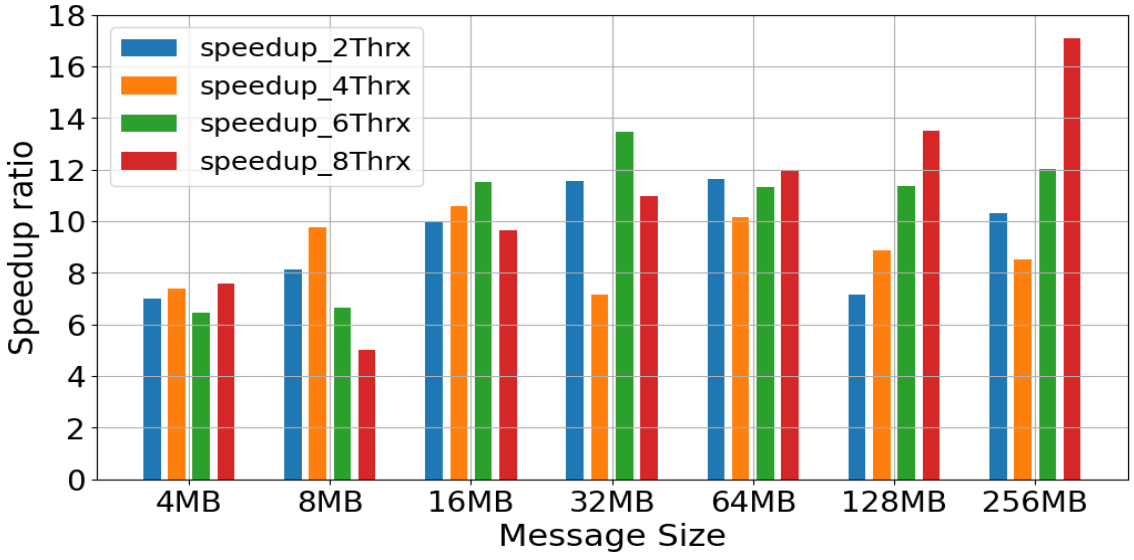
CPU2

Figure (4.2): Shows the Throughput Results of the Proposed Encryption in Comparison to Speck Ciphers

In figure 4.3, it is demonstrated that, on average, the proposed parallel encryption algorithm is significantly faster than the parallel implementations of the Speck cypher. Specifically, in CPU1, the proposed parallel encryption algorithm is 14.10 times faster than the parallel implementation of Speck, while in CPU2, it is 17.09 times faster than the parallel implementation of Speck.



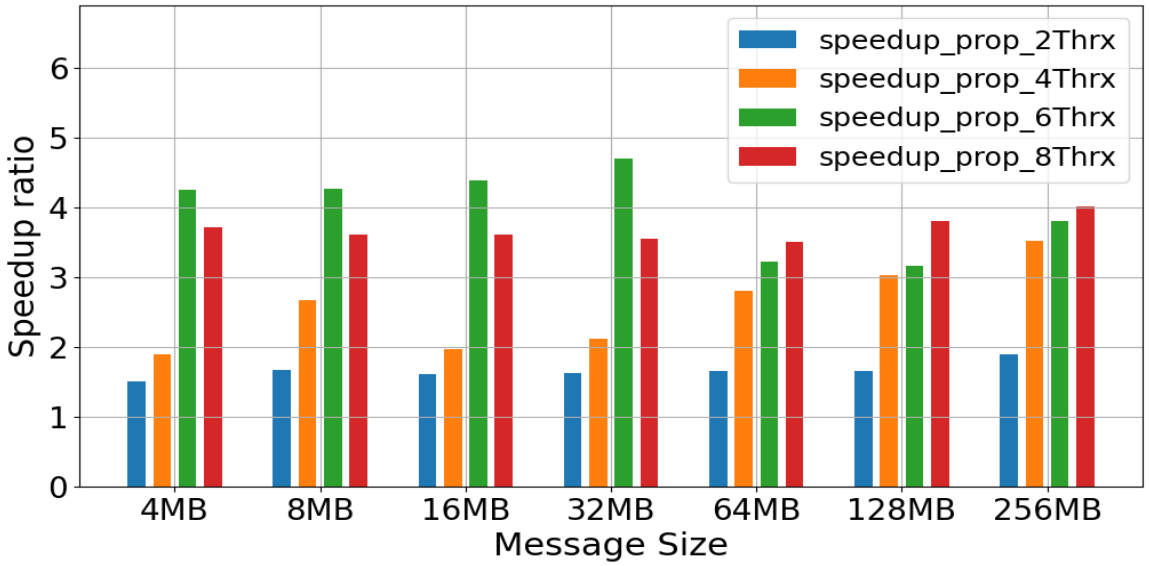
CPU1



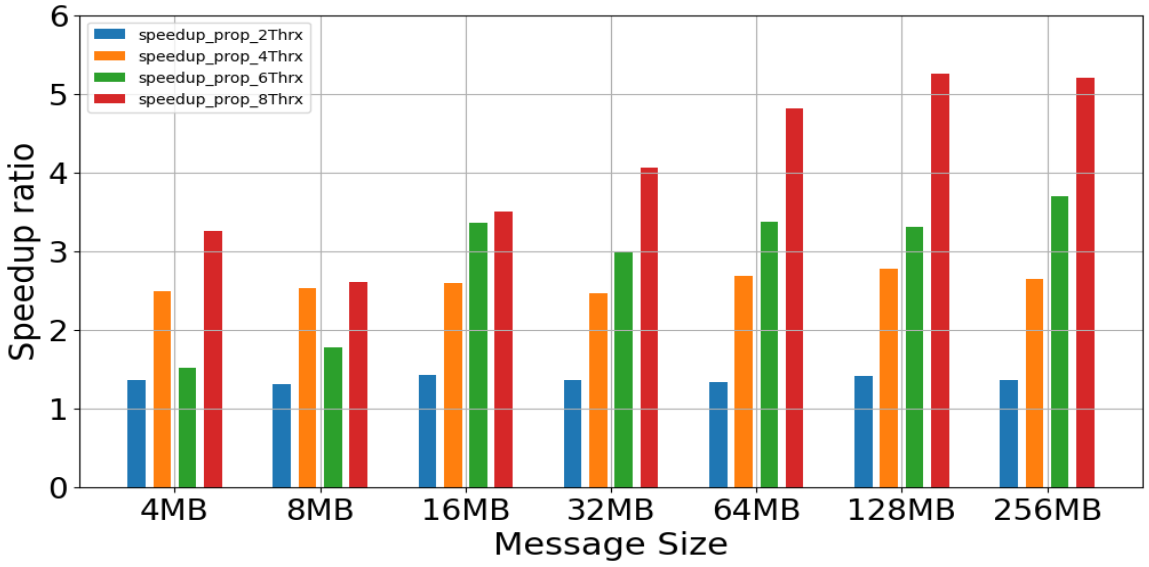
CPU2

Figure (4.3): The Speedup Ratio of the Parallel Implementation of the Speck and Proposed

Figure 4.4 compares the suggested one-round cipher's concurrent execution versus its sequential execution on CPUs 1 and 2. The latter shows that, on average, it is 4.70 and 5.26 times faster than a sequential implementation.



CPU1



CPU2

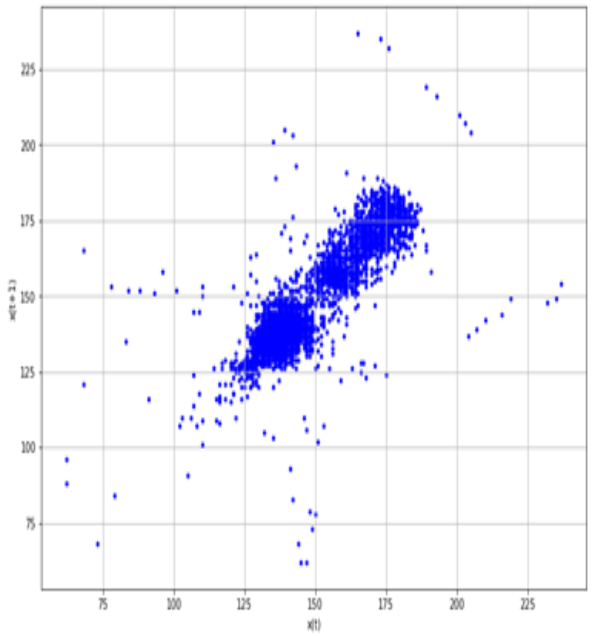
Figure (4.4): The Speedup Ratio of the Sequential and Parallel Proposed Methods

4.3.2 Security Analysis Results

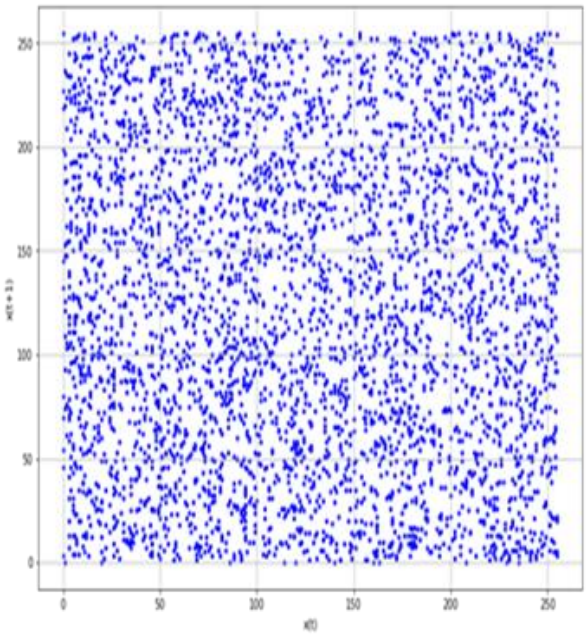
Security analysis test as given in Section 2.6.3 of Chapter 2, Which is evaluated using two laboratory images, which are a picture of a boat and Lena with a size of 512×512, and these tests are shown below.

4.3.2.1 Statistical Security Analysis Tests

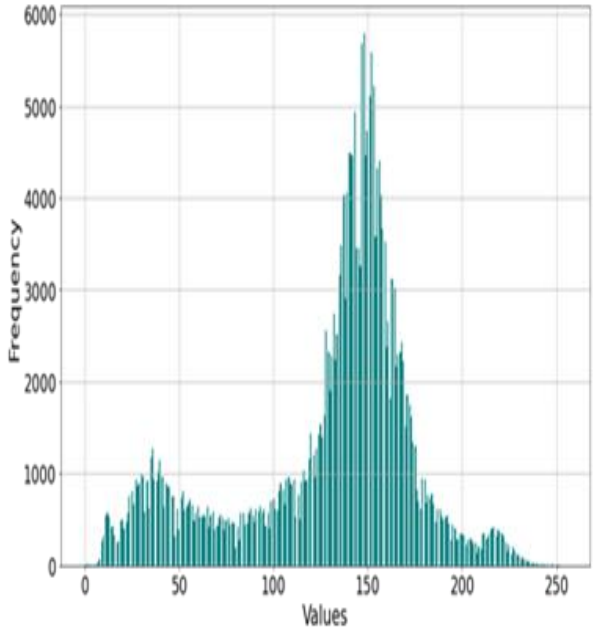
This part includes a number of different tests, such as an analysis of the histogram, a pdf analysis, an entropy analysis, and a correlation coefficient analysis, all of which are described in detail in Section 2.6.3.1 of Chapter 2, and the results of these tests are shown in Figures 5, 6, 7 and 8. Also, Table 4 shows the comparison of the security results of boat and lena images between the proposed and speck ciphers. It indicates that both ciphers are very close in terms of security level.



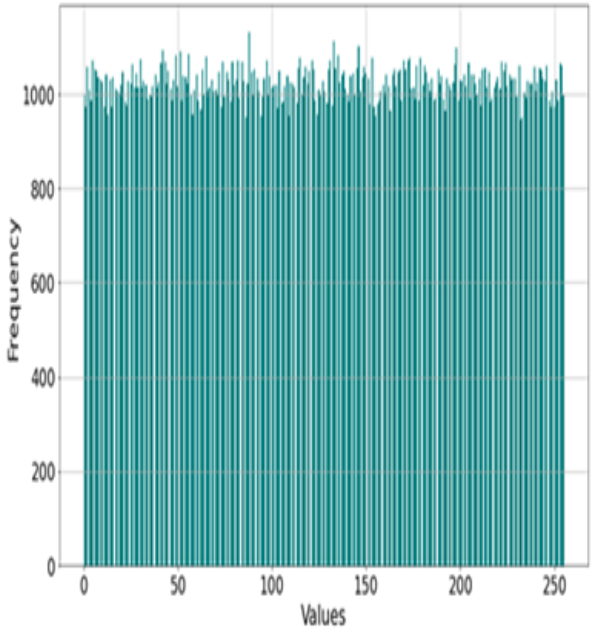
(a)



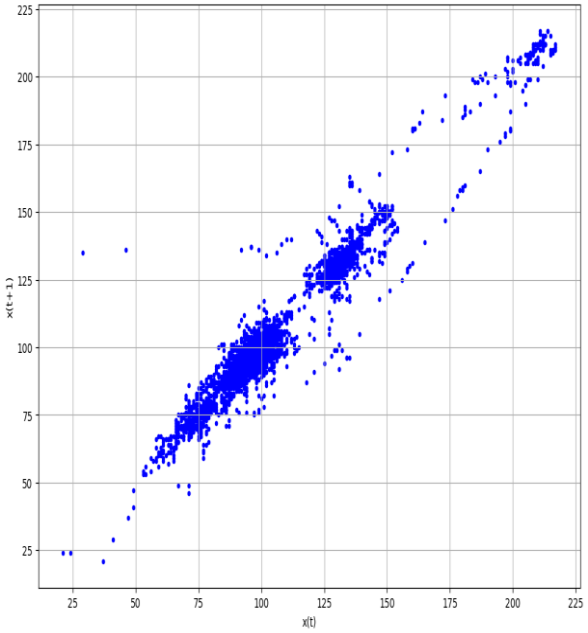
(b)


Frequency
Values
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200
250

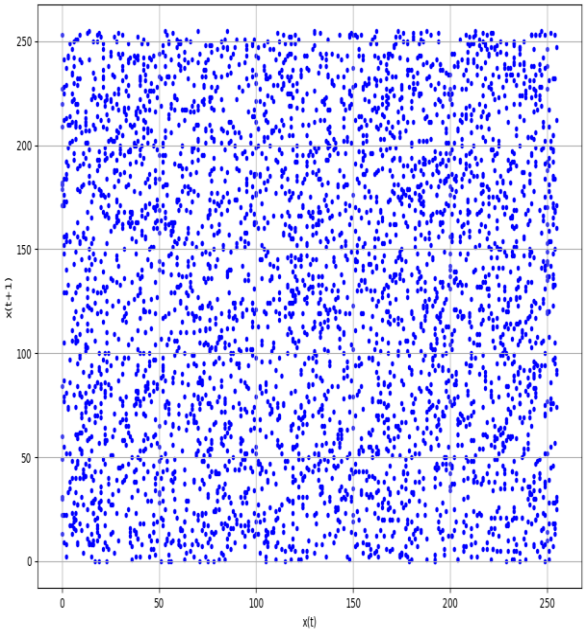
(c)


Frequency
Values
0
200
400
600
800
1000
0
50
100
150
200
250

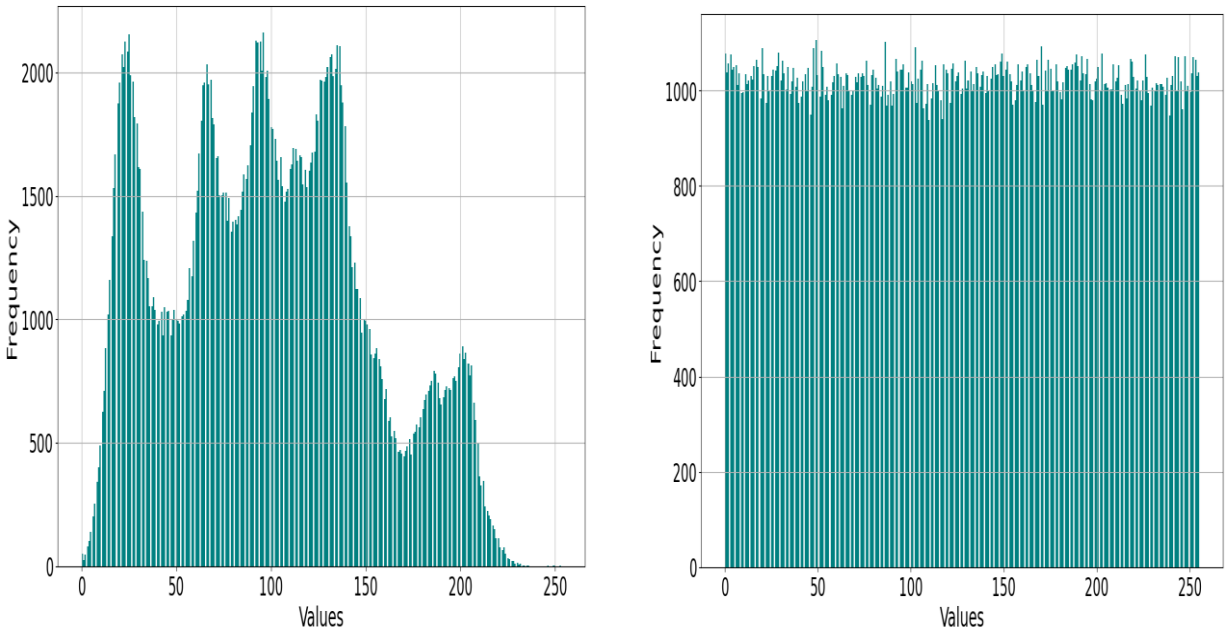
(d)


x(t+1)
x(t)
25
50
75
100
125
150
175
200
225

(e)


x(t+1)
x(t)
0
50
100
150
200
250

(f)



(g) **(h)**

Figure (4.5): Images of the Boat and Lena as a Result (a) Recurrence of the Boat Image (b) Recurrence of Boat-Ciphered (c) Histogram of the Boat Image (d) Histogram of Boat-Ciphered (e) Recurrence of Lena's Image (f) Recurrence of Lena-Ciphered (g) Histogram of Lena's Image (h) Histogram of Lena-Ciphered

The histogram analysis results are presented in Figure 4. 5.c,d, g, and h are comparing the original plain images of size 512 x 512 with their cipher-image counterparts. Figure 4.5 (d)(h) The histogram of the boat and Lena encrypted images is demonstrated to be quite close to a uniform distribution (512x512)/256, which about 1024.

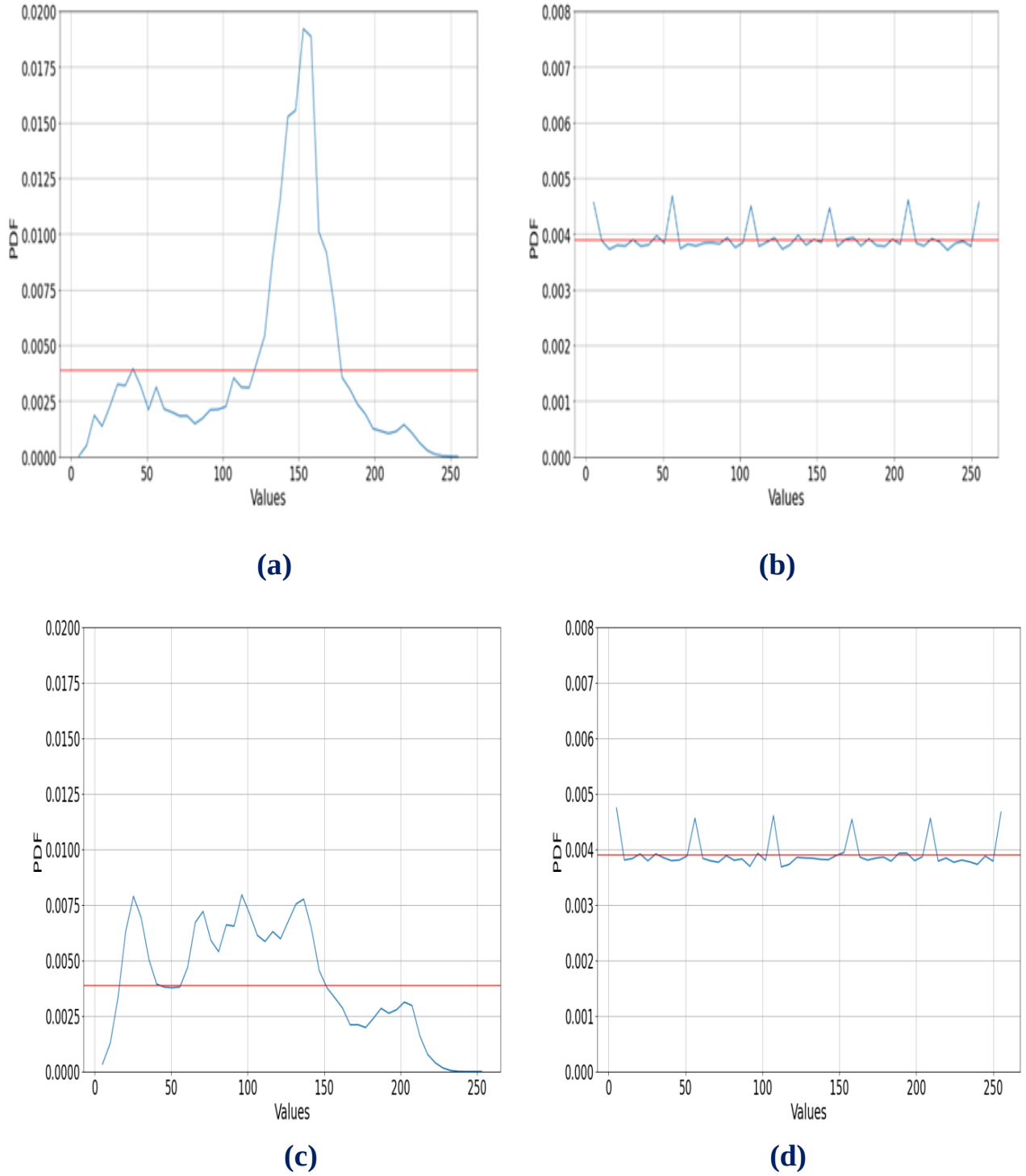


Figure (4.6): Show The PDFs of the Original and Cipher Image are Presented in (a) (b)(c)(d), Respectively.

As shown in Figure 4.6 (b) and (d), it is clear that the PDF of the ciphered boat and ciphered Lena images with a value of around 0.039 ($1/256 = 3.9 \times 10^{3}$) for all cypher

text symbols, PDFs of the images that were encrypted can be thought of as having a close to uniform distribution.

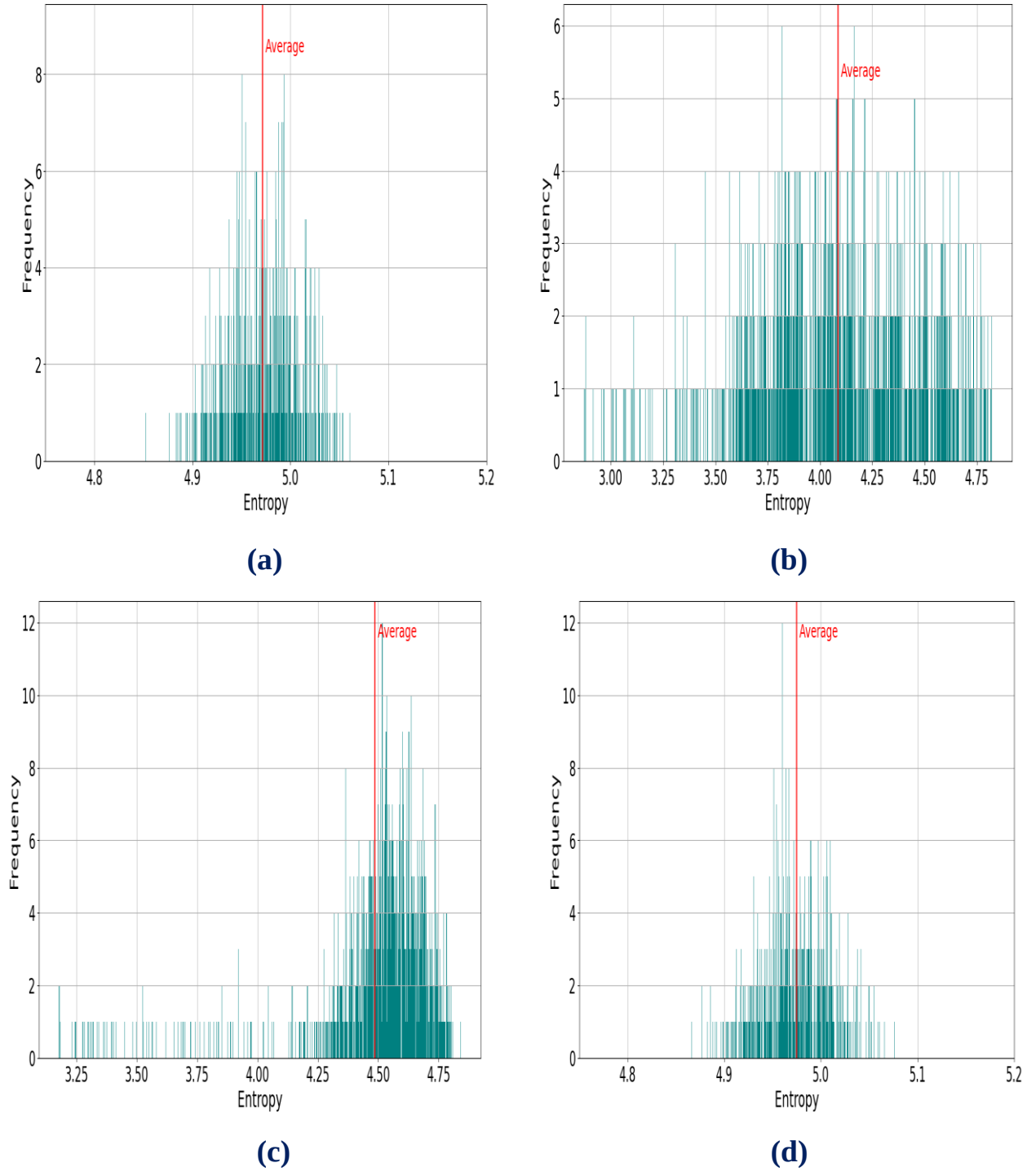
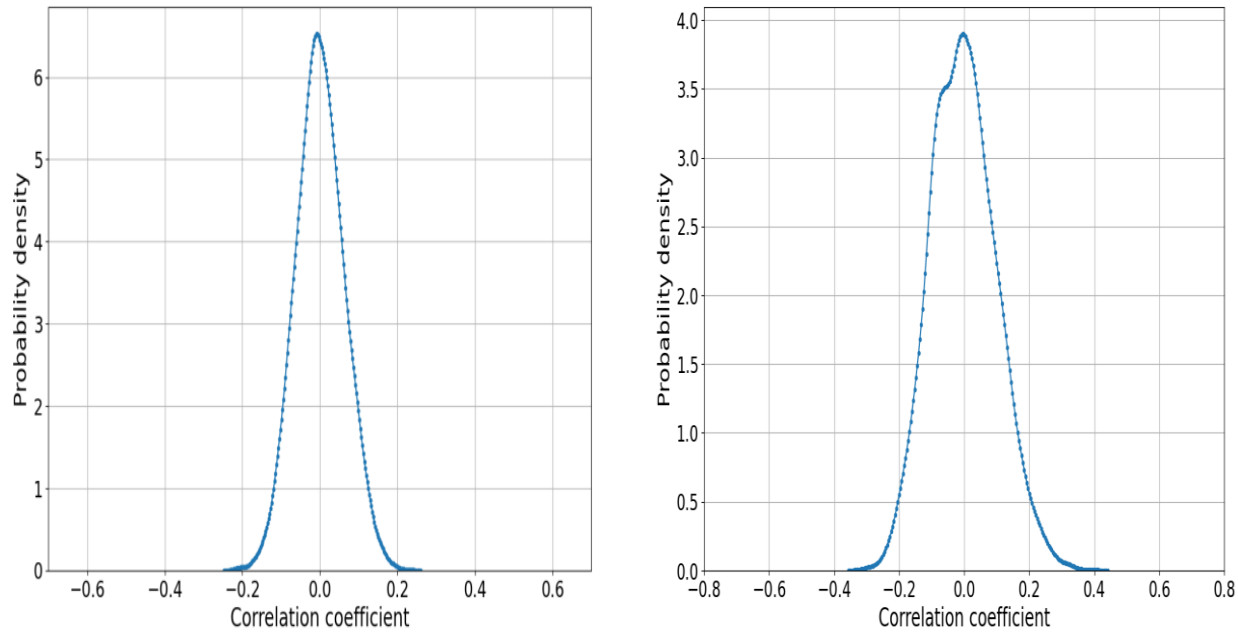


Figure (4.7): Analysis of the Entropy of (a) (c) the Boat and Lena Plain Images and (b) (d) The Produced Cipher Pictures at the 16*16 Sub-Matrix Level Compared to 1,000 Randomly Private Keys

The ordering of the values of entropy discovered by analyzing encrypted text that was gathered at the byte level is depicted in Figures 4.7(b) and (d). The entropy values exhibit a normal distribution with an average of 4.97, almost exactly at the theoretical maximum (log2 (Tb/8) = 5 for Tb = 256), where Tb represents the total number of pixels in messages, the occurrence probability of the pixels in the message is 8, and the entropy value is always close to the desired value of 5. Figure 4.7(b)(d)'s uniform distribution of entropy values shows that the encrypted message follows the same pattern. Therefore, the suggested encryption method is sufficiently safe from any entropy attack.



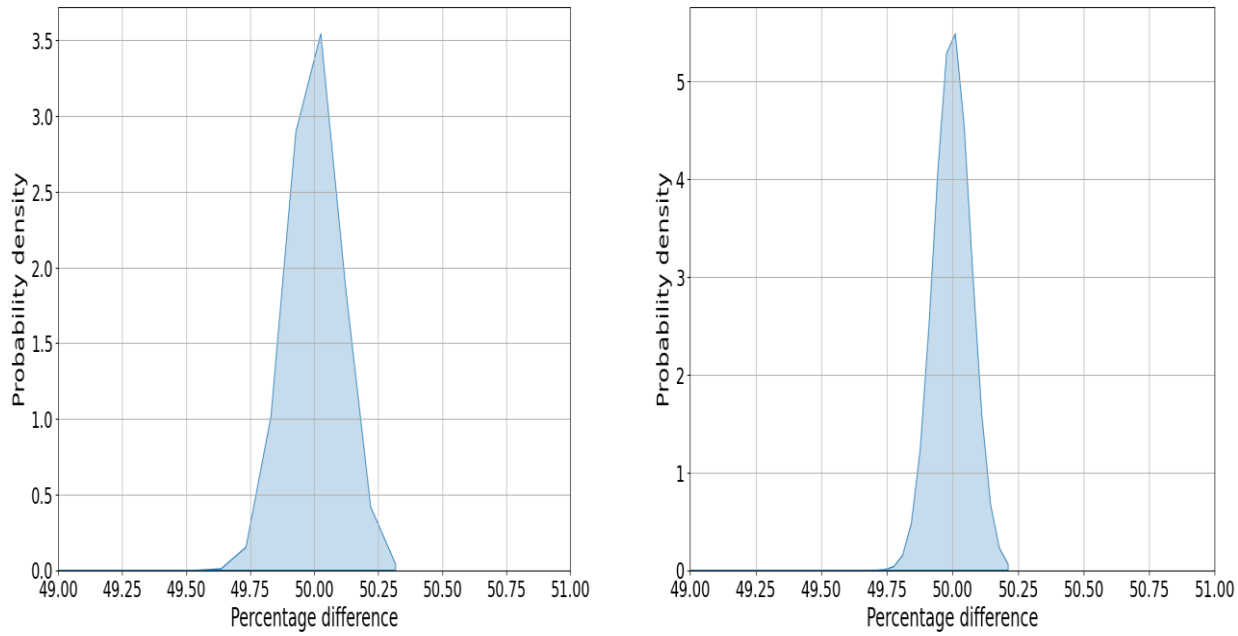
(a) **(b)**

Figure (4.8):(a) and (b) PDF of the Correlation Coefficients between Plain and Cipher Images for a Group of 1024 bytes

Figure 4.8 shows the outcomes of the association analysis between the unencrypted and encrypted images for 1,000 different keys overall and one random key each iteration. The findings unequivocally demonstrate that the association ratio is extremely small, nearly at zero, supporting the independence and unpredictable nature of the created cypher text.

4.3.2.2 Avalanche Effect Tests (Sensitivities Test)

This section includes two important sensitivities tests, which are detailed in Section 2.6.3.2 of Chapter 2, and the results of the key Avalanche effect are shown in Figure 4.9.



(a) **(b)**

Figure (4.9): (a) and (b) The Percentage Difference of the Sensitivity Test between 1000 Cipher Message Values Generated by Changing Randomly One Bit in the Key for the Same Message.

Figure 4.9 shows that for Nb = 256 and similarly for greater values of Nb, the difference between the values of the cipher message is between 1000 ciphered values for the same message, where each value is formed by arbitrarily changing a single bit of the key. According to the findings, at least Nb/2 bits, or 50% of the cipher message bits, have been altered, where Nb represents the number of bits used in the image. The suggested encryption did, however, provide great unpredictability against statistical assaults.

Table (4.4): The Comparison of the Security Results of Boat and Lena Images

Image Name	Average results				
	Histogram Test	PDF Test	Entropy Test	Correlation Test	Sensitivities Test
Boat Cipher	748.98	0.003921	4.9712	0.000696	49.801
Lena Cipher	748.98	0.003921	4.9747	-0.00185	49.911

As can be seen in Table 4.4, the statistical analysis test comparison results for the boat and Lina's images show that, on average, they are both very similar to a uniform distribution, but that they differ slightly in terms of values. For the correlation test at the boat image, the best from the Lena image is very low, and its value is zero, which indicates the strength of the proposed method. As for testing the sensitivity of the boat image, it is better than the Lena image because the less sensitive the system is, the more powerful it is.

4.3.2.3 Randomness test using PractRand

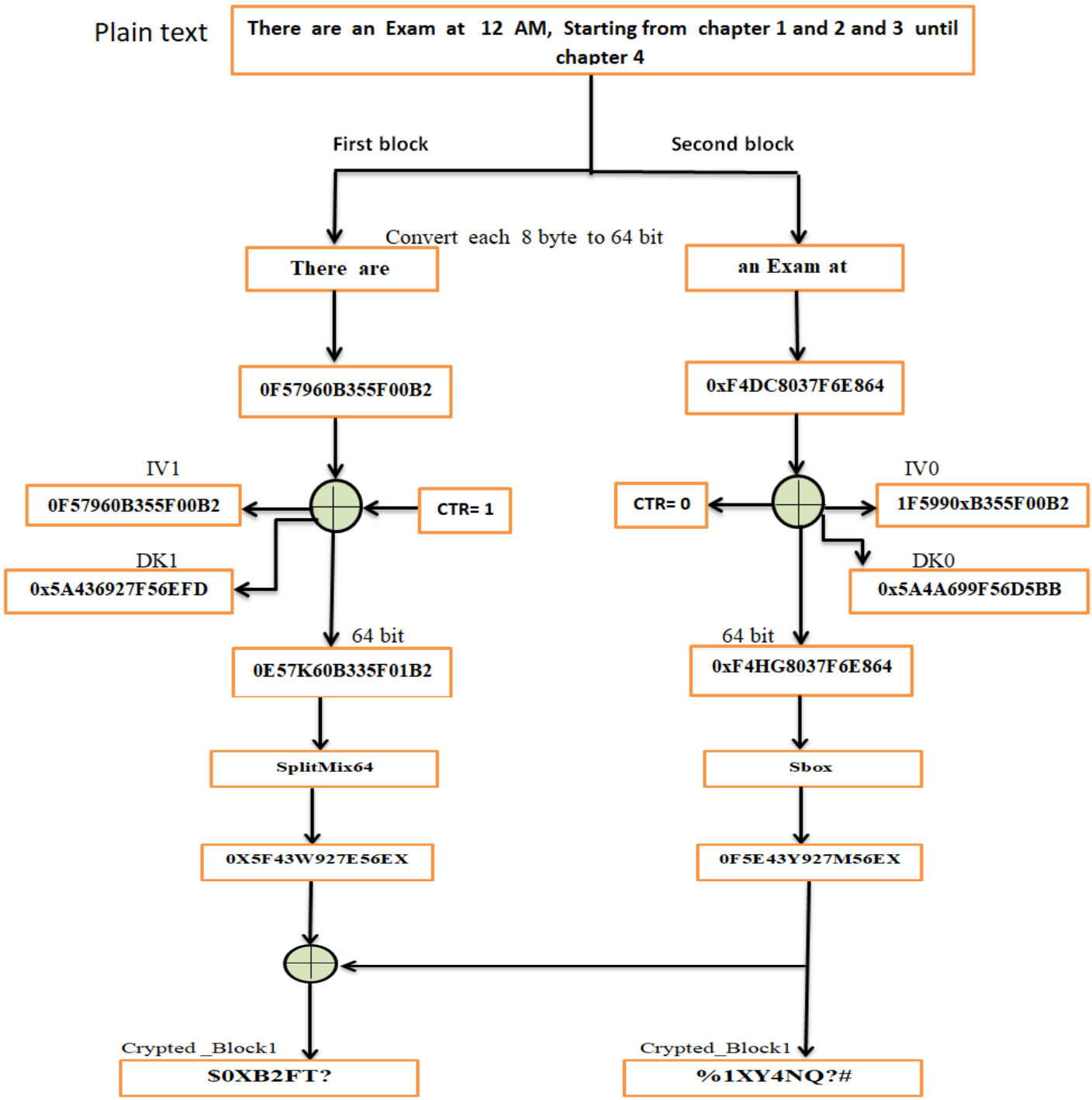
The proposed one-round block cipher was tested using 64 seeds and Practrand, and it passed each test with flying colors. In order to determine if the created sequence passes the tests or not, PractRand will evaluate it and provide a report. In actuality, simply a message of 512 by 512 dimensions is used. The message's components are all set to zero at the beginning, and the key is only ever set up once. Furthermore, each additional variable is only initialized once. In order to analyze the encrypted message that resulted, PractRand, one of the trickiest statistical tests, was used. It will be feasible to determine via these tests whether or not the keystream that was created satisfies the

appropriate randomization and uniformity criteria. Additionally, the practice test suite focuses on identifying certain varieties of non-randomness that may escape detection by more conventional statistical tests. The statistical tests used by PractRand include birthday spacing tests, gap tests, and others. Additionally, it offers statistical quality indicators, such as the quantity of output under "1TB" (terabytes) required to identify a certain degree of non-randomness. Table (4.5): shows the proposed method results that were obtained by PractRand tests.

Table (4.5): The Result of the proposed method as determined by Practrand tests

Size of message(bytes)	Test name	perfect value
2^29	FPF	pass
2^30	FPF	pass
2^31	FPF	pass
2^32	FPF	pass
2^33	FPF	pass
2^34	FPF	pass
2^35	FPF	pass
2^36	BCFN	pass
2^37	BCFN	pass
2^38	FPF	pass

For example, to include a new case study on a message that only contains 512 characters (64 byte), study each stage of the proposed work in detail as shown in figure 4.10. also shows in detail the results of randomness that were obtained by PractRand tests in figure 4.11.



Figure(4.10): Real Study of the proposed one round

```
hawra@hp:~$ cd PractRand
hawra@hp:~/PractRand$ g++ -std=c++14 -O3 -Wall -w -o test test1.cpp
hawra@hp:~/PractRand$ ./test  | ./RNG_test stdin64 -tlmax "512GB" -multithreaded
RNG_test using PractRand version 0.93
RNG = RNG_stdin64, seed = 0x6d34ea66
test set = normal, folding = standard (64 bit)

rng=RNG_stdin64, seed=0x6d34ea66
length= 256 megabytes (2^28 bytes), time= 2.5 seconds
  no anomalies in 159 test result(s)

rng=RNG_stdin64, seed=0x6d34ea66
length= 512 megabytes (2^29 bytes), time= 5.4 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/64]FPF-14+6/16:all          R=  +4.8  p =  6.2e-4   unusual
  ...and 168 test result(s) without anomalies

rng=RNG_stdin64, seed=0x6d34ea66
length= 1 gigabyte (2^30 bytes), time= 10.3 seconds
  no anomalies in 180 test result(s)

rng=RNG_stdin64, seed=0x6d34ea66
length= 2 gigabytes (2^31 bytes), time= 20.8 seconds
  no anomalies in 191 test result(s)

rng=RNG_stdin64, seed=0x6d34ea66
length= 4 gigabytes (2^32 bytes), time= 43.7 seconds
  no anomalies in 201 test result(s)

rng=RNG_stdin64, seed=0x6d34ea66
length= 8 gigabytes (2^33 bytes), time= 79.7 seconds
  no anomalies in 212 test result(s)

rng=RNG_stdin64, seed=0x6d34ea66
length= 16 gigabytes (2^34 bytes), time= 151 seconds
  no anomalies in 223 test result(s)

rng=RNG_stdin64, seed=0x6d34ea66
length= 32 gigabytes (2^35 bytes), time= 285 seconds
  no anomalies in 233 test result(s)
```

(a)

```
rng=RNG_stdin64, seed=0x6d34ea66
length= 64 gigabytes (2^36 bytes), time= 559 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/64]BCFN(2+0,13-1,T)         R=  +8.1  p =  7.9e-4   unusual
  ...and 243 test result(s) without anomalies

rng=RNG_stdin64, seed=0x6d34ea66
length= 128 gigabytes (2^37 bytes), time= 1103 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low16/64]BCFN(2+2,13-0,T)        R=  -7.3  p =1-1.0e-3   unusual
  ...and 254 test result(s) without anomalies

rng=RNG_stdin64, seed=0x6d34ea66
length= 256 gigabytes (2^38 bytes), time= 2157 seconds
  Test Name                         Raw       Processed     Evaluation
  FPF-14+6/16:(0,14-0)              R= +11.2  p =  5.6e-10  very suspicious
  FPF-14+6/16:(1,14-0)              R=  +8.3  p =  2.5e-7   suspicious
  FPF-14+6/16:all                   R=  +8.6  p =  1.6e-7   very suspicious
  FPF-14+6/16:all2                  R= +29.2  p =  1.2e-14    FAIL
  [Low16/64]FPF-14+6/16:(0,14-0)    R=  +9.5  p =  2.3e-8   very suspicious
  [Low16/64]FPF-14+6/16:all         R=  +6.2  p =  2.7e-5   mildly suspicious
  [Low16/64]FPF-14+6/16:all2        R= +14.8  p =  5.0e-8   very suspicious
  ...and 258 test result(s) without anomalies
```

(b)

Figure(4.11): (a) and (b) The result of Randomness Tests using PractRand

The figure (4.11) presented for the randomness result in Practrand has four columns: (1) ”Test Name”, a name for the sub-test and the corresponding (2) "Raw" is not of much use to end users. (3) "processed,", either a p-value, "pass," or ”fail”. (4) ”Evaluation”, describing the result. ”FAIL” means that the tested cipher un-vaguely failed that sub-test, while ”suspicious” means that that result should not happen often on a good RNG or cipher but should happen occasionally. The following is a summary of the unsatisfactory outcomes. After one terabyte of data, the suggested approach failed, and the error code was FPF-14+6/16:cross. Additionally, it failed after 4 terabytes of data, revealing the precise error—the BCFN (2+0,13-0,T) error—that both the ordinary and optimized versions had. In essence, BCFN looks for long-range trends in the 0s and

1s distributions. Hence, failing BCFN often indicates that you have sufficient information to determine which is more common in the nth with an accuracy of above 50%, given whether 0s or 1s were more common in the preceding 32 or 64 byte (n-1) blocks. In this case, the error number is 13.

4.3 Summary

This chapter presents the comparison results of the proposed cipher with Speck's light-weight encryption. In order to compare the Speck algorithm with the proposed technique, two different computer processors—the Intel Core i7-7700HQ and the Intel Core i7-6600U—were employed. The findings of the security study are presented in terms of statistical analysis tests, sensitivity testing, and randomness. Additionally, it shows the execution time, throughput, and speedup for two different CPUs.

Chapter Five

Conclusion and Future WorkS

5.1 Conclusion

This thesis proposed one-round lightweight cipher method that applied in parallel over multi-core CPUs. In the following are the conclusions of this work:

1. This work proposed an efficient one-round cipher scheme that encrypts 128 bits per round. It is designed to specifically fulfilled on parallel platforms. For multicore CPUs, the proposed encryption performs better than the most optimized Speck implementations, making it more appropriate for real-time applications.

2. Compared to other methods, like the Speck method, which requires 32 rounds and a 128-bit key size, our method requires just one round and a dynamic key size of 512 bits, demonstrating improved speed and performance, according to empirical evaluations.

3. Moreover, as compared to other encryption methods now in use, the suggested cipher implementation is exceedingly complicated and challenging. Using some benchmark tests and cryptanalysis, the robustness of the proposed cipher was evaluated and confirmed.

4. The suggested technique is able to reach a high data throughput in comparison to some lightweight methods that already exist, with a throughput that is higher than 25 and 10 Gigabits per second on two Intel Core i7 central processing units. The proposed encryption method outperforms the parallel speck method by an average of 14.10 and 17.09 times faster when executed over the two multicore CPUs used. The average speedup of the two CPUs use compared to the sequential version of the proposed algorithm and its parallel implementation is 4.70 and 5.26.

5.2 Future Works

Some concepts will be proposed for use in further works, as seen in the following examples:

1. We recommend more elaboration on the "Multicore CPUs" property platforms of the proposed cipher compared with other related lightweight ciphers.

4. Applying the method that was recommended over a wide variety of parallel platforms, such as grid and cloud, while finding more enhancements for node-to-node communications.

5. Compare it with other lightweight cryptography methods.

References

Reference

[1] L. Sleem, “Design and implementation of lightweight and secure cryptographic algorithms for embedded devices Lama Sleem To cite this version : HAL Id : tel-03101356,” 2021.

[2] I. K. Dutta, B. Ghosh, and M. Bayoumi, “Lightweight cryptography for internet of insecure things: A survey,” 2019 IEEE 9th Annu. Comput. Commun. Work. Conf. CCWC 2019, pp. 475–481, 2019, doi: 10.1109/CCWC.2019.8666557.

[3] L. W. Kong, “High Speed Computation of Advanced Cryptographic Algorithms on Massively Parallel Architecture Doctor of Philosophy in Engineering Lee Kong Chian Faculty of Engineering and Science Universiti Tunku Abdul Rahman February 2018,” no. February, p. 235, 2018.

[4] R. F. Atiyah and I. Al-mejibli, “Lightweight secure Approach for IOT Devices,” vol. 13, no. 3, pp. 4069–4078, 2022.

[5] S. Surendran, A. Nassef, and B. D. Beheshti, “A survey of cryptographic algorithms for IoT devices,” *2018 IEEE Long Isl. Syst. Appl. Technol. Conf. LISAT 2018*, pp. 1–8, 2018, doi: 10.1109/LISAT.2018.8378034.

[6] Z. A.Alhameed, A. B.M.Fanfaakh, and E.Hadi, “Prediction of CPU Frequency for Energy Saving in Heterogeneous Cluster using Genetic Algorithm,” 2020.

[7] A. Fanfakh, H. Noura, and R. Couturier, “ORSCA-GPU: one round stream cipher algorithm for GPU implementation,” J. Supercomput., vol. 78, no. 9, pp. 11744–11767, 2022, doi: 10.1007/s11227-022-04335-4.

[8] L. Sleem and R. Couturier, “Speck-R: An ultra light-weight cryptographic scheme for Internet of Things,” *Multimed. Tools Appl.*, vol. 80, no. 11, pp. 17067–17102, 2021, doi: 10.1007/s11042-020-09625-8.

[9] M. Gupta and A. Sinha, “Enhanced-AES encryption mechanism with S-box splitting for wireless sensor networks,” *Int. J. Inf. Technol.*, vol. 13, no. 3, pp. 933–941, 2021, doi: 10.1007/s41870-021-00626-w.

[10] A. Adil Yazdeen, S. R. M. Zeebaree, M. Mohammed Sadeeq, S. F. Kak, O. M. Ahmed, and R. R. Zebari, "FPGA Implementations for Data Encryption and Decryption via Concurrent and Parallel Computation: A Review," Qubahan Acad. J., vol. 1, no. 2, pp. 8–16, 2021, doi: 10.48161/qaj.v1n2a38.

[11] M. Nabil, A. A. M. Khalaf, and S. M. Hassan, "Design and implementation of pipelined and parallel AES encryption systems using FPGA," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 20, no. 1, pp. 287–299, 2020, doi: 10.11591/ijeecs.v20.i1.pp287-299.

[12] R. Couturier, H. N. Noura, and A. Chehab, "ESSENCE: GPU-based and dynamic key-dependent efficient stream cipher for multimedia contents," Multimed. Tools Appl., vol. 79, no. 19–20, pp. 13559–13579, 2020, doi: 10.1007/s11042-020-08613-2.

[13] W. El Hadj Youssef, A. Abdelli, F. Dridi, and M. Machhout, "Hardware implementation of secure lightweight cryptographic designs for IoT applications," Secur. Commun. Networks, vol. 2020, 2020, doi: 10.1155/2020/8860598.

[14] A. S. Anil, G. Sawant, Sayali Kamthe, Yashasvini Shaha , Bapu Morajkar, "Implementation of SIMON & SPECK Algorithm," vol. 6, no. 1, pp. 292–296, 2019.

[15] N. Alassaf, A. Gutub, S. A. Parah, and M. Al Ghamdi, "Enhancing speed of SIMON: A light-weight-cryptographic algorithm for IoT applications," *Multimed. Tools Appl.*, vol. 78, no. 23, pp. 32633–32657, 2019, doi: 10.1007/s11042-018-6801-z.

[16] R. M. N. Aldahdooh and A. Y. Mahmoud, "Parallel Implementation and Analysis of Encryption Algorithms," no. April, 2018.

[17] M. R. Asassfeh, M. Qatawneh, and F. M. Al Azzeh, "Performance evaluation of blowfish algorithm on supercomputer IMAN1," *Int. J. Comput. Networks Commun.*, vol. 10, no. 2, pp. 43–53, 2018, doi: 10.5121/ijcnc.2018.10205.

[18] X. Wu, Y. Han, M. Zhang, and S. Zhu, "Parallel Long Messages Encryption Scheme Based on Certificateless Cryptosystem for Big Data," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10726 LNCS, pp. 211–222, 2018, doi: 10.1007/978-3-319-75160-3_14.

[19] W. Stallings, Cryptography and Network Security: Principles and Practice, International Edition: Principles and Practice. 2014. [Online]. Available: https://books.google.com/books?id=q_6pBwAAQBAJ&pgis=1

[20] W. Stallings, L Earning in N Etwork S Ecurity: Principles and Practice, International Edition: Principles and Practice. vol. 11, no. 5,pp.1-14, 2019.

[21] K. B. Logunleko, O. D. Adeniji, and A. Logunleko, "A Comparative Study of Symmetric Cryptography Mechanism on DES , AES and EB64 for Information Security A Comparative Study of Symmetric Cryptography Mechanism on DES , AES and EB64 for Information Security," no. June, 2020.

[22] R. Asaad, S. Abdulrahman, and A. Hani, "Advanced Encryption Standard Enhancement with Output Feedback Block Mode Operation," Acad. J. Nawroz Univ., vol. 6, no. 3, pp. 1–10, 2017, doi: 10.25007/ajnu.v6n3a70.

[23] M. N. Hasan and V. R. Sharma, "FPGA IMPLEMENTATION OF CRYPTOGRAPHY USING AES ALGORITHM Submitted by," vol. 7, no. November 2012, pp. 167–172, 2018.

[24] S. K. Mousavi, A. Ghaffari, S. Besharat, and H. Afshari, Security of internet of things based on cryptographic algorithms: a survey, vol. 27, no. 2. Springer US, 2021. doi: 10.1007/s11276-020-02535-5.

[25] S. B. Sadkhan and A. O. Salman, "A survey on lightweight-cryptography status and future challenges," Int. Conf. Adv. Sustain. Eng. Appl. ICASEA 2018 - Proc., pp. 105–108, 2018, doi: 10.1109/ICASEA.2018.8370965.

[26] M. A. Kale and P. S. Dhamdhere, "Survey Paper on Different Type of Hashing Algorithm," Int. J. Adv. Sci. Res., vol. 3, no. 2, pp. 14–16, 2018.

[27] J. Jebrane and S. Lazaar, “A performance comparison of lightweight cryptographic algorithms suitable for IoT transmissions,” Gen. Lett. Math., vol. 10, no. 2, pp. 46–53, 2021, doi: 10.31559/glm2021.10.2.5.

[28] L. W. Kong, “High Speed Computation of Advanced Cryptographic Algorithms on Massively Parallel Architecture Doctor of Philosophy in Engineering Lee Kong Chian Faculty of Engineering and Science Universiti Tunku Abdul Rahman February 2018,” no. February, p. 235, 2018.

[29] M. Naya-plasencia, M. N. Symmetric, and L. S. Cryptography, “Symmetric Cryptography for Long-Term Security To cite this version : HAL Id : tel-01656036 Presented by Mar ´ Inria de Paris Symmetric Cryptography for Long-Term Security,” 2017.

[30] D. Sehrawat and N. S. Gill, “Lightweight Block Ciphers for IoT based applications: A Review,” *Int. J. Appl. Eng. Res.*, vol. 13, no. 5, pp. 2258–2270, 2018, [Online]. Available: http://www.ripublication.com

[31] A. Project, “Performance Analysis of the Speck Cryptography Algorithm Supervisors Certification I certify that this Project entitled ‘ Performance Analysis of the Speck cryptography algorithm ’ was done by (Dunay Abd Al- Hussain,” 2022.

[32] C. L. D. A. SILVA, “Desenvolvimento De Um Módulo De Criptografia Leve Para Fpga Utilizando O Algoritmo Simon and Speck,” 2019, [Online]. Available: https://www.inf.pucrs.br/moraes/docs/tcc/tcc_cristovam.pdf

[33] I. Assurance and N. L. Cryptography, “Notes on the design and analysis of Simon and Speck,” no. January, pp. 1–23, 2018.

[34] P. E. A. Adriaanse and Z. Erkin, “A Comparative Study of the TEA , XTEA , PRESENT and SPECK lightweight cryptographic schemes,” pp. 1–21, 2021.

[35] S. L. Yeo, D. P. Le, and K. Khoo, “Improved algebraic attacks on lightweight block ciphers,” J. Cryptogr. Eng., vol. 11, no. 1, 2021, doi: 10.1007/s13389-020-00237-4.

[36] M. A. Al-Shabi, “A Survey on Symmetric and Asymmetric Cryptography Algorithms in information Security,” Int. J. Sci. Res. Publ., vol. 9, no. 3, p. p8779, 2019, doi: 10.29322/ijsrp.9.03.2019.p8779.

[37] F. Thabit, S. Alhomdy, and S. Jagtap, “Security analysis and performance evaluation of a new lightweight cryptographic algorithm for cloud computing,” Glob. Transitions Proc., vol. 2, no. 1, pp. 100–110, 2021, doi: 10.1016/j.gltp.2021.01.014.

[38] S. M. Salim Reza, A. Ayob, M. M. Arifeen, N. Amin, M. H. M. Saad, and A. Hussain, “A lightweight security scheme for advanced metering infrastructures in smart grid,” Bull. Electr. Eng. Informatics, vol. 9, no. 2, pp. 777–784, 2020, doi: 10.11591/eei.v9i2.2086.

[39] O. G. ABood and S. K. . Guirguis, “A Survey on Cryptography Algorithms,” Int. J. Sci. Res. Publ., vol. 8, no. 7, 2018, doi: 10.29322/ijsrp.8.7.2018.p7978.

[40] A. Alaa, A. Fanfakh, “Parallel Message Authentication Algorithm Implemented Over Multicore CPU,” Int. J. Intell. Eng. Syst., vol. 16, no. 4, pp. 642–654, 2023, doi: 10.22266/ijies2023.0831.52.

[41] H. N. Noura, R. Couturier, O. Salman, and K. Mazouzi, “DKEMA: GPU-based and dynamic key-dependent efficient message authentication algorithm,” J. Supercomput., vol. 78, no. 12, pp. 14034–14071, 2022, doi: 10.1007/s11227-022-04433-3.

[42] A. Fanfakh, H. Noura, and R. Couturier, “Simultaneous encryption and authentication of messages over GPUs”, Multimedia Tools and Applications, pp. 1-33, 2023. https://doi.org/10.1007/s11042-023-15451-5.

[43] R. Trobec, B. Slivnik, P. Bulić, and B. Robič, Introduction to parallel computing: from algorithms to programming on state-of-the-art platforms. Springer, 2018.

[44] T. Rauber and G. Rünger, Parallel programming: For multicore and cluster systems. 2013. doi: 10.1007/978-3-642-37801-0.

[45] T. S. Alsulimani, N. K. Almazmomi, and K. B. Arour, “Emerging Trends in management information systems : The use of new Parallel Distributed Technologies,” vol. 20, no. 6, pp. 160–165, 2020.

[46] C. Sekhar, “Comparative Study on CPU , GPU and TPU,” Vol.5, No.1 (2020), 2021, pp.31-38, doi: 10.21742/IJCSITE.2020.5.1.04.

[47] A. Galletti, G. Giunta, L. Marcellino, and D. Parlato, “An algorithm for Gaussian recursive filters in a multicore architecture,” Proc. 2017 Fed. Conf. Comput. Sci. Inf. Syst. FedCSIS 2017, no. October, pp. 507–511, 2017, doi: 10.15439/2017F428.

[48] L. Sleem and R. Couturier, “TestU01 and Practrand: Tools for a randomness evaluation for famous multimedia ciphers,” Multimed. Tools Appl., vol. 79, no. 33–34, pp. 24075–24088, 2020, doi: 10.1007/s11042-020-09108-w.

[49] H. Jabir, and A. Fanfakh, “An Overview of Parallel Symmetric Cipher of Messages,” J. Univ. BABYLON Pure Appl. Sci., vol. 31, no. 2, pp. 19–33, 2023, doi: 10.29196/jubpas.v31i2.4652.

[50] E. Based, P. Interface, M. Abutaha, I. Amar, and S. Alqahtani, “Parallel and Practical Approach of Efficient Image Chaotic,” entropy Artic., pp. 1–22, 2022.

الخلاصة

أصبحت حماية البيانات واحدة من أهم القضايا على الرغم من التقدم الكبير في الاتصالات والتكنولوجيا. لكي تتمكن التكنولوجيا المستندة إلى الويب من إرسال البيانات بسرعة وأمان، يجب تشفير البيانات. التشفير هو عملية تحويل النص العادي إلى نص مشفر، والذي لا يستطيع الأشخاص السيئون قراءته أو تغييره. إن مشكلة تعزيز تنفيذ خوارزميات التشفير الموثوقة هي محور هذه الدراسة. لقد أظهر انتشار المستندات الرقمية والمعاملات التجارية في كل مكان مدى الحاجة الملحة إلى أساليب التشفير. تتطلب أساليب التشفير القوية الكثير من الوقت لتنفيذها بفعالية ومع ذلك، قام عدد من الباحثين بتطوير أسلوب التشفير بشكل متوازٍ من أجل تقليل مقدار الوقت اللازم لإنهاء إجراءات التشفير وفك التشفير. وقد أسفر التحقيق في هذه القضية عن عدد من الحلول القابلة للتطبيق. تمكن الباحثون من تحقيق مستويات أداء محسنة في تقنية التشفير باستخدام التوازي لزيادة الإنتاجية وتعزيز كفاءة طرق التشفير.

في الوقت الحاضر، يتم استخدام أساليب متعددة النواة لتحقيق نفس النتيجة مع المزيد من النوى في معالج واحد. ومع ذلك، لتحقيق أقصى استفادة من هذه المعالجات، يجب تصميم التطبيق الذي يعمل عليها بطريقة متزامنة؛ وهذا يستدعي استخدام لغات برمجة متوازية عالية المستوى وتقنيات متوازية صريحة لتسهيل إنشاء تطبيقات متوازية متعددة الخيوط أو متعددة العمليات على المبرمجين. تم تطوير الكثير من التقنيات (المكتبات وواجهات تمرير الرسائل وما إلى ذلك) للمساعدة في تطوير البرامج الموازية. عادةً ما تعتمد خوارزميات التشفير الحديثة بشكل كبير على الرياضيات لإنتاج النص المشفر. وبالتالي، من أجل ضمان مستوى أعلى من الأمان، يتطلب التشفير الخفيف جولات عديدة. ترتفع التكلفة الحسابية وتتباطأ سرعات التشفير عند زيادة عدد الجولات.

وبالتالي، في هذا العمل، تم اقتراح نظام تشفير خفيف الوزن يستخدم فقط جولة واحدة من تقنية التشفير الكتلي التي يتم تطبيقها بالتوازي على معالج متعدد النواة. تقوم خوارزمية تشفير الرسائل المقترحة بتشفير كتلتين فرعتين من ١٢٨ بت من الرسالة العادية في كل جولة. يستخدم المتجه الاولي وصناديق الاستبدال و splitmix64 PRNG ويستخدم مفتاحًا ديناميكيًا بحجم ٥١٢ بت لزيادة مستوى الأمان. ولذلك، فأنها تقوم بتشفير الرسالة العادية والحصول على كتلتين فرعيتين مشفرتين، مما يجعلها تقنية سريعة لتشفير وفك تشفير كتل الرسائل. بالمقارنة مع الطريقة الحالية. هذه الطريقة المقترحة هي طريقة احتمالية لأنها تعتمد على مفتاح ديناميكي.

ووفقاً لنتائج الأداء، فهو قادر على الوصول إلى إنتاجية عالية للبيانات مقارنة ببعض الطرق خفيفة الوزن الموجودة بالفعل، حيث يصل إنتاجية أعلى من ٢٥ و١٠ جيجابت في الثانية على وحدتي معالجة مركزية Intel Core i7. تتفوق طريقة التشفير المقترحة على طريقة Speck المتوازية بمعدل ١٤,١٠ و ١٧,٠٩ اسرع عند تنفيذها عبر وحدتي معالجة مركزية متعددة النواة. متوسط التسريع مقارنة بالنسخة التسلسلية للخوارزمية المقترحة وتنفيذها الموازي هو ٤,٧٠ و٥,٢٦ على التوالي. كما أن طريقة التشفير المقترحة توفر قدرًا كبيرًا من العشوائية وتجتاز اختبارات PractRand الاحصائية.

وبالتالي، فإن الطريقة المقترحة تعتبر منافسًا قويًا لتطبيق الأمان العالي على المعالجات متعددة النواة. ومع ذلك، فإن هذا العمل يطور مجال تشفير البيانات من خلال تقديم منافس قوي للتشفير أحادي الجولة عالي الأمان المطبق على معالجات متعددة النواة، مما يوفر إنتاجية محسنة والكفاءة والمتانة الإحصائية مقارنة بطريقة شبيك الحالية.

وزارة التعليم العالي و
البحث العلمي
جامعة بابل
كلية العلوم للبنات
قسم علوم الحاسوب

طريقة تشفير خفيفة الوزن الموازية

رسالة

مقدمة إلى مجلس كلية العلوم للبنات – جامعة بابل وهي جزء من

متطلبات

نيل شهادة الماجستير في العلوم / علوم الحاسوب

من قبل

حوراء جابر حمزة

بإشراف

الاستاذ المساعد الدكتور

احمد بدري مسلم

٢٠٢٣م ١٤٤٥هـ