

**M. TECH. THESIS**

On

**DESIGNING ALGORITHM FOR  
LIGHTWEIGHT STREAM CIPHER**

Author

**Sunny Samuel**



**M.TECH in CRYPTOLOGY AND SECURITY  
INDIAN STATISTICAL INSTITUTE, KOLKATA**

**July 2024**

# DESIGNING ALGORITHM FOR LIGHTWEIGHT STREAM CIPHER

A PROJECT REPORT

*Submitted in partial fulfillment of the  
requirements for the award of the degree*

*of*

MASTER OF TECHNOLOGY

*in*

CRYPTOLOGY AND SECURITY

*Submitted by:*

SUNNY SAMUEL

*Guided by:*

Dr. Dibyendu Roy (Primary Guide)

Prof. Subhamoy Maitra (Secondary Guide)

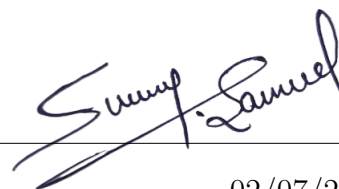
INDIAN STATISTICAL INSTITUTE, KOLKATA

July 2024

## CANDIDATE'S DECLARATION

I hereby declare that the project entitled **Designing Algorithm for Lightweight Stream Cipher** submitted in partial fulfillment for the award of the Master of Technology degree in Cryptology and Security completed under the supervision of **Dr. Dibyendu Roy (Primary Guide)**, Asst Prof, IIIT Vadodara and **Prof. Subhamoy Maitra (Secondary Guide)**, HoD ASU, ISI Kolkata is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.



02/07/2024

Sunny Samuel(Crs2221)

Indian Statistical Institute

Kolkata - 700108, India.

## CERTIFICATE by Guide(s)

It is certified that the above statement made by the student is correct to the best of my knowledge.



02/07/2024

Dibyendu Roy

Assistant Professor

Indian Institute of Information Technology, Vadodara



02/07/2024

Subhamoy Maitra

Professor & Head

Applied Statistics Unit

Indian Statistical Institute, Kolkata

## ACKNOWLEDGMENTS

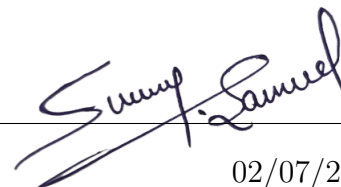
This thesis would not have been possible without the support and guidance of several remarkable individuals who generously devoted their time and energy to my academic journey.

Firstly, I would like to express my heartfelt gratitude to my wife for her unwavering support, kindness, and patience. My parents and sister have been my constant sources of inspiration, and I am deeply thankful for their encouragement.

I am profoundly grateful to my advisors, **Dr. Dibyendu Roy (Primary Guide)** and **Prof. Subhamoy Maitra (Secondary Guide)**, whose guidance, continuous support, and encouragement were invaluable throughout this project. They have taught me the principles of good research and inspired me with their insights and innovative ideas. Their mentorship has been instrumental in the completion of this work, and I owe them a great debt of gratitude. I also extend my thanks to the **Indian Navy, Capt Ritesh Wahi** for giving me the opportunity to pursue a Master of Technology in Cryptology and Security at ISI Kolkata.

My sincere thanks also go to **Mr. Sreejith Choudhury, Mr. R Radheshwar, Mr. Pritam Pal, Mr. Jyotirmoy Basak, Mr. Suman Datta, and Mr. Anup Kumar Kundu** for their wise counsel and support, which significantly contributed to various aspects of my thesis.

Finally, I assume full responsibility for any errors or shortcomings that may be present in this thesis.



---

02/07/2024

Sunny Samuel(Crs2221)

Indian Statistical Institute

Kolkata - 700108, India.

## Abstract

The role of embeddable cryptographic processors in revolutionizing defense communications for the Indian Navy bears immense significance. These processors serve as catalysts for a diverse range of novel applications critical to naval operations, encompassing tailored smartphones and robust tablet computers engineered for frontline tactical deployment. Moreover, they facilitate the establishment of secure tactical Wi-Fi networks, unmanned vehicle control systems, and real-time targeting capabilities, thereby bolstering operational efficacy and security.

Moreover, this technological evolution expedites the adoption of modernized cryptography, promising to furnish secure wireless computing solutions even in the most arduous maritime environments. In alignment with the Indian Navy's modernization endeavors, a concerted attempt is underway to harness commercial off-the-shelf (COTS) cryptographic algorithms and processing hardware. This strategic approach not only fortifies resilience against technological obsolescence but also bolsters support for network-centric operations, streamlines data dissemination and sharing, and advances interoperability with allied naval forces.

In response to these imperatives, the primary focus of the thesis is to develop a novel lightweight stream cipher algorithm expressly tailored for utilization within the Indian Navy, specifically for facilitating data communication among computers interconnected via internal LAN. Drawing inspiration from the FASTA, PASTA Lightweight Cryptography (LWC) algorithms and the esteemed NIST CEASER competition awardee, the ASCON family.

**Keywords:** Lightweight Stream Cipher, FASTA, PASTA, ASCON Algorithm

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Thesis Outline . . . . .	6
<b>2</b>	<b>Preliminaries</b>	<b>9</b>
2.1	Introduction to Cryptology . . . . .	9
2.2	Main Objectives of Cryptology . . . . .	10
2.3	Applications of Cryptology . . . . .	11
2.4	Symmetric and Asymmetric Cryptology . . . . .	12
2.4.1	Symmetric Cryptography . . . . .	12
2.4.2	Asymmetric Cryptography . . . . .	13
2.5	Need of Symmetric Cryptography . . . . .	13
2.6	Types of Ciphers . . . . .	14
2.6.1	Block Cipher . . . . .	14
2.6.2	Stream Cipher . . . . .	15
2.7	Block Cipher Algorithms . . . . .	16
2.7.1	Data Encryption Standard (DES) . . . . .	16
2.7.2	Triple Data Encryption Algorithm (3DES) . . . . .	17
2.7.3	Advanced Encryption Standard . . . . .	18
2.7.4	RC5 (Rivest Cipher) . . . . .	18
2.8	Stream Cipher Algorithms . . . . .	20
2.8.1	RC4 (Rivest Cipher 4) . . . . .	20
2.8.2	Salsa20 and ChaCha . . . . .	21

2.9	Comparison of Cipher Algorithms . . . . .	22
<b>3</b>	<b>Research Background</b>	<b>24</b>
3.1	Introduction to Lightweight Cryptology . . . . .	24
3.2	Understanding LWC . . . . .	25
3.3	Challenges in LWC . . . . .	26
3.4	LWC on Internet of Things (IoT) . . . . .	27
3.5	Known LWC Algorithms . . . . .	28
3.5.1	Lightweight Block Ciphers . . . . .	28
3.5.2	Lightweight Stream Ciphers . . . . .	29
<b>4</b>	<b>Proposed Lightweight Stream Cipher Algorithm</b>	<b>31</b>
4.1	Overview . . . . .	31
4.2	Detailed Structure . . . . .	32
4.2.1	Parameter Description . . . . .	32
4.2.2	Intialization Phase . . . . .	32
4.2.3	Linear Diffusion Phase . . . . .	33
4.2.4	Substitution Phase . . . . .	34
4.3	Encryption . . . . .	35
4.4	Decryption . . . . .	35
<b>5</b>	<b>Cryptanalysis of Algorithm</b>	<b>37</b>
5.1	Differential Cryptanalysis - Resistance to Differential Fault Attack . . . . .	37
5.1.1	DFA on Our Stream Cipher . . . . .	38
5.2	Linear Cryptanalysis - Resistance to Algebraic Attack . . . . .	41
5.3	Our Contribution . . . . .	41
<b>6</b>	<b>Conclusion and Future Work</b>	<b>45</b>
<b>A</b>	<b>C Code for implementation of the Lightweight Stream Cipher</b>	<b>51</b>
<b>B</b>	<b>Code for Differential Fault Attack</b>	<b>61</b>

# List of Figures

2.1	Symmetric Encryption . . . . .	12
2.2	Asymmetric Encryption . . . . .	13
2.3	Block Cipher . . . . .	15
2.4	Stream Cipher . . . . .	16
2.5	Data Encryption Standard (DES) . . . . .	17
2.6	Triple Data Encryption Standard (3DES) . . . . .	17
2.7	Advanced Encryption Standard (AES) . . . . .	18
2.8	RC5 Algorithm) . . . . .	19
2.9	RC4 . . . . .	20
2.10	Salsa Algorithm . . . . .	21
2.11	ChaCha20 Algorithm . . . . .	22
3.1	Internet of Things . . . . .	28
4.1	Detailed structure of LWC Encryption . . . . .	32
4.2	320 bits Internal State . . . . .	33
4.3	Encryption Algorithm Flow . . . . .	35
4.4	Decryption Algorithm Flow . . . . .	36



# List of Tables

4.1	AES S Box . . . . .	34
-----	---------------------	----

# Chapter 1

## Introduction

### 1.1 Introduction

Low-power devices are pervasive in various facets of modern life, including critical applications within the Indian Navy. These devices, ranging from communication equipment to surveillance systems, play a vital role in naval operations, often containing sensitive operational data. Ensuring adequate security measures for these devices is imperative, given the challenging low-power environments in which they operate. Conventional encryption methods may fall short in meeting the security needs of such devices due to constraints in both hardware and software capabilities.

For instance, tasks requiring real-time data processing or secure communication over limited bandwidths pose significant challenges for low-power devices in the Navy, which may lack the computational resources of larger naval assets. As a result, compromises must often be made, potentially compromising the overall security posture of the naval network. Lightweight cryptography emerges as a solution to this dilemma, aiming to provide robust security while minimizing resource consumption.

From a naval perspective, lightweight devices may encounter constraints such as limited memory, processing power, and energy availability. These limitations necessitate the development of cryptographic solutions tailored specifically to the Navy's operational environment. By

leveraging lightweight cryptographic algorithms, the Indian Navy can enhance the security of its communication systems, data transmissions, and onboard computing platforms without overburdening its low-power devices.

Recognizing the critical role of lightweight cryptography in naval operations, the Indian Navy has a vested interest in standardizing cryptographic protocols and algorithms suitable for low-power environments. This aligns with the Navy's overarching goal of modernizing its communication and information systems to meet the evolving challenges of maritime warfare. Moreover, initiatives such as NIST's framework [17] for benchmarking lightweight cryptographic algorithms offer valuable guidance in selecting and implementing secure cryptographic solutions tailored to the Navy's unique requirements.

Hence, lightweight cryptography holds immense significance for the Indian Navy, offering a pathway to enhance the security and resilience of its low-power devices in an increasingly complex maritime environment. By embracing lightweight cryptographic solutions, the Navy can safeguard its critical communications infrastructure and maintain operational superiority in the face of emerging threats.

In this thesis, we endeavor to craft an innovative Lightweight stream cipher, drawing inspiration from the esteemed ASCON, FASTA, and PASTA LWC algorithms. Our objective encompasses not only the meticulous design of this cipher but also its meticulous implementation in the C programming language. Furthermore, we shall conduct a comprehensive security analysis of the stream cipher, delving into its resilience against potential vulnerabilities and threats.

## **1.2 Thesis Outline**

In Chapter 2, this thesis lays the groundwork with a comprehensive overview of fundamental cryptographic concepts crucial for understanding subsequent discussions. It delves into encryption, decryption, and key management and introduces the distinction between symmetric

and asymmetric cryptographic algorithms. Moreover, Chapter 2 introduces lightweight cryptography (LWC), highlighting its significance in modern applications where efficiency and security are paramount.

Chapter 3 builds upon this foundation by conducting a detailed analysis of Lightweight Cryptography, identifying critical gaps in current research paradigms. This chapter meticulously reviews existing literature, discussing the specific challenges and requirements unique to lightweight cryptographic algorithms.

Chapter 4 presents the core contribution of this thesis—a meticulously crafted Lightweight Stream Cipher designed specifically for internal LAN communications within the Indian Navy. It elaborates on the cipher’s innovative design principles, including a unique algebraic permutation technique for diffusion and a robust 16x16 substitution box to ensure confusion properties. Detailed construction methodologies are outlined, supplemented by illustrative C code.

In Chapter 5, we cover the cryptanalysis of our proposed algorithm, focusing on its resistance to various attacks. We begin with an explanation of Differential Fault attacks and detail the measures implemented to ensure the algorithm’s robustness against such vulnerabilities using sage math. We then describe the diffusion layer, which uses XOR operations and right circular rotations with random bit shifts to enhance diffusion and complicate the internal state, making it difficult for attackers to predict. Additionally, we highlight the integration of the AES S-Box, modified with a bitwise affine transformation, to provide further security against cryptanalytic attacks. Next, we discuss algebraic attacks, which involve solving multivariate polynomial equations, and demonstrate that our algorithm is resistant to these attacks due to its high algebraic degree. It then brings out our contributions in designing this novel lightweight stream cipher. Overall, this chapter emphasizes the comprehensive security analysis conducted to validate the resilience of our lightweight stream cipher against both algebraic and Differential Fault attacks.

Finally, the concluding section synthesizes the thesis findings, offering a succinct summary of the scholarly journey undertaken. It underscores the significance of advancing lightweight cryptography and outlines future research directions to further strengthen cryptographic protocols for military and beyond.”

# Chapter 2

## Preliminaries

### 2.1 Introduction to Cryptology

Cryptography serves as a foundational cornerstone for ensuring the security of network and communication systems, enabling the attainment of data confidentiality and authentication. As a result, this field finds extensive applications across various domains, including the Internet, email communication, etc. Despite the existence of numerous cryptographic standards, their direct applicability to diverse applications is impeded by the significant variation in specific requirements unique to each application.

This thesis delves into the realm of lightweight cryptographic algorithms, a specialized category of cryptographic algorithms designed to cater to devices constrained by resource limitations while still maintaining a requisite level of security. Prior to delving into comprehensive insights about lightweight cryptography in Chapter 3, it is imperative to provide a comprehensive elucidation of cryptography itself, serving as the cornerstone of the thesis's conceptual framework. We shall understand the basics of this field, clarifying its fundamental concepts, diverse implementations, widespread applications, and the evolving necessity for its adoption over time.

Moreover, this chapter will undertake an in-depth exploration of types of encryption algorithms: symmetric and asymmetric key encryption. Lastly, a detailed exposition of the

thesis's objectives and scope will be presented, elucidating the overarching purpose and focal points of this dissertation.

## 2.2 Main Objectives of Cryptology

Cryptography can be defined in various ways, but generally, it is defined as the science of transmitting messages, information, and communications securely over insecure channels accessible to third parties, also known as adversaries [4]. This discipline offers a means of safeguarding information and communication so that only the planned recipient can get and interpret original information. This objective is realized by formulating regulations and protocols grounded in mathematical principles, concepts, and computations known as cryptographic algorithms.

Cryptography, in fact, originates from a broader domain known as cryptology. Cryptology encompasses the study and application of techniques and methodologies for securely communicating and storing data in a concealed and unintelligible manner. Within cryptology, two distinct fields emerge: cryptography, as described earlier, and cryptanalysis. Cryptanalysis is tasked with studying and analyzing cipher texts and codes to decipher them and get meaningful information.

In contemporary cryptography, the focus revolves around achieving four primary objectives outlined below:

1. **Confidentiality:** This objective guarantees that information and data are accessible and comprehensible only to authorized recipients, remaining unreadable to unauthorized parties [24]. Confidentiality is upheld by encrypting the message (plaintext) into ciphertext for transmission, with only the intended recipient holding the secret key to convert the ciphertext back into its initial plaintext form.

2. **Data Integrity:** Data integrity guarantees that information sent remains unaltered during storage or transmission between the source and destination, with any modification being

easily detectable. Techniques such as MD5 are utilized to create a smaller form of it and is sent with the plaintext. The receiver uses the same technique and gets the smaller form of the message and compares if there have been any changes in the received data as compared to the original data. If both are the same, then integrity is maintained, or else someone has altered the data.

3. **Non-Repudiation:** Non-repudiation guarantees that once a sender sends data, he cannot later deny that he had not sent it. [25] This is often accomplished through the use of digital signatures and Message Authentication Codes (MACs) containing a cryptographic key. These cryptographic primitives not only ensure data integrity but also bolster information security more robustly than simple hash functions.

4. **Authentication:** Authentication can be used to verify the parties involved in the exchange of data without requiring any exact information thereof. This assures that the sender is indeed who they claim to be. Digital certificates and appropriate digital signatures are commonly employed to achieve authentication.

## 2.3 Applications of Cryptology

Cryptography finds widespread utilization across various applications in today's digital landscape, being employed by individuals on a daily basis. In addition to its primary function of facilitating secure communication between systems, encryption plays a pivotal role in securing interactions between web browsers and servers, as well as between email clients and servers [10]. These services, commonly used by people worldwide, rely extensively on cryptographic algorithms.

With the rapid advancement of networking technologies in recent years, an increasing volume of sensitive and important information is sent through networks. Consequently, malicious entities actively seek to intercept and pilfer credentials and data. Cryptography serves as a vital deterrent against such nefarious actions, safeguarding against threats like identity theft.



This is particularly crucial in scenarios involving financial transactions personal information, etc.

The demand for robust cryptographic algorithms is exceptionally high, and their design poses significant challenges. This is exacerbated by the continual increase in computational power of computers, coupled with the relentless pursuit of new vulnerabilities and security breaches by malicious actors. Therefore, the development of resilient cryptographic algorithms remains an ongoing endeavor, essential to achieve integrity and confidentiality.

## 2.4 Symmetric and Asymmetric Cryptology

It can be classified based on the keys utilized for encryption and decryption: symmetric (private) key cryptography and asymmetric (public) key cryptography.

### 2.4.1 Symmetric Cryptography

In this type of cryptography, only one secret key will be used for both encrypting and decrypting, simplifying cryptographic operation [5]. Symmetric encryption offers high-speed performance compared to its asymmetric counterpart, primarily due to the shorter key lengths required. However, the challenge lies in securely sharing the key before initiating communication.

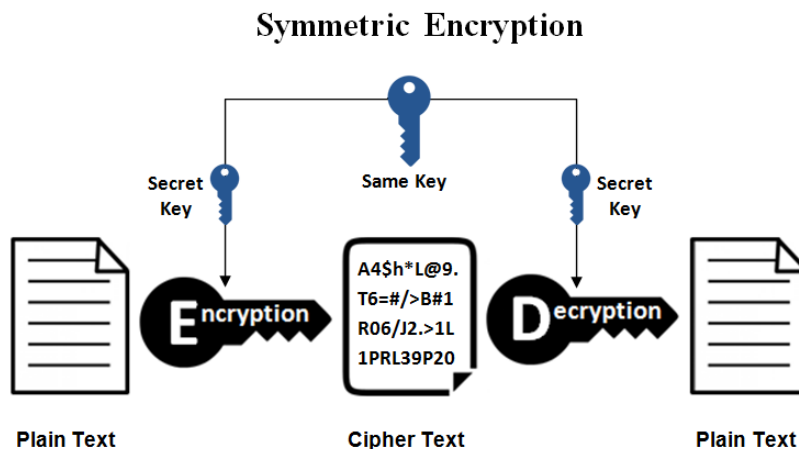


Figure 2.1: Symmetric Encryption

## 2.4.2 Asymmetric Cryptography

This encryption approach utilizes two distinct keys: a public key and a private key. This is done to increase the inherent security. Both the keys are designed in such a way that some interconnection is found mathematically, yet deriving one from the other is extremely challenging [5]. The sender and the receiver possess two keys: a public key that's openly accessible and visible to anyone within the network, necessitating no additional security measures, and a private key that remains confidential and known solely to its owner.

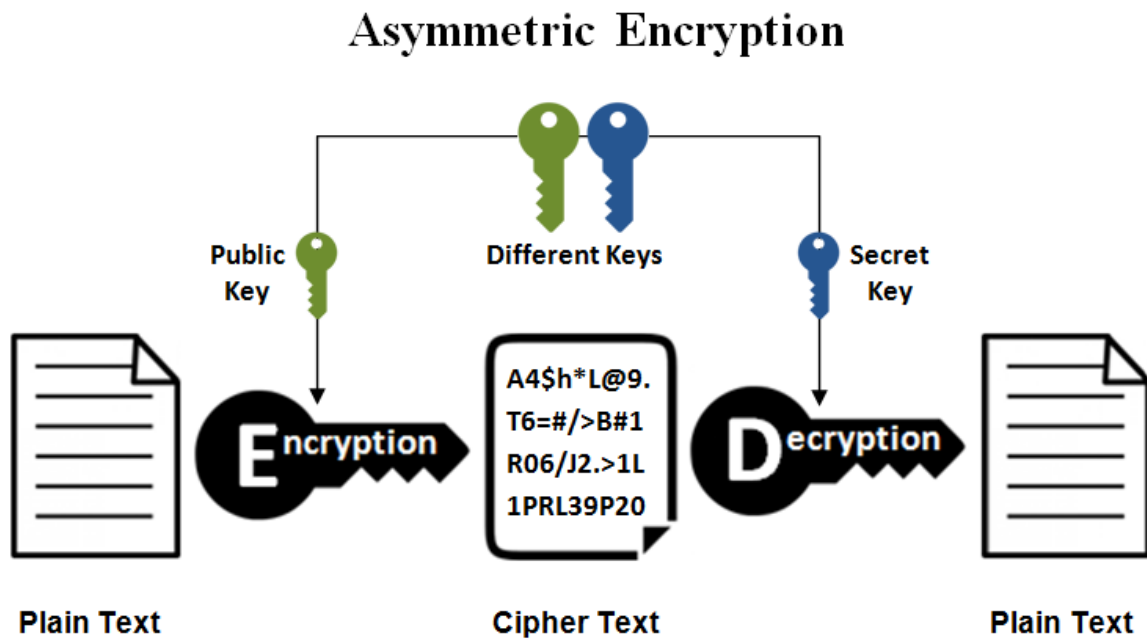


Figure 2.2: Asymmetric Encryption

## 2.5 Need of Symmetric Cryptography

In Lightweight cryptography, symmetric algorithms have demonstrated greater efficiency for several reasons, which will be elaborated upon later in this section. As previously mentioned, in this form of cryptography, there is only one secret key that will be used for both encryption and decryption processes, distinguishing it from asymmetric cryptography, where two keys are utilized. Here, Block and Stream Ciphers will be examined in detail. Their

respective functionalities and structures, as well as advantages and disadvantages, will be elucidated. Additionally, notable examples of these ciphers will be highlighted, along with pertinent details for each.

Primarily, asymmetric ciphers necessitate the use of two keys, contrasting with the single-key approach employed by symmetric ciphers. This inherent complexity and resource-intensive nature of asymmetric ciphers pose significant implementation challenges, particularly for resource-constrained IoT devices, where efficient operation is paramount [18]. However, it is essential to acknowledge that the utilization of a single key in symmetric cryptography introduces a key exchange dilemma between communicating parties. Unlike in public key encryption algorithms, where the private key remains secure and is never transmitted over the network, this issue arises with symmetric cryptography.

Nevertheless, in the context of Lightweight cryptography, symmetric algorithms emerge as the primary cryptographic method for limited-resource devices. It can be characterized by the ease of implementation, minimal key size requirements compared to asymmetric counterparts, reduced resource utilization resulting in low overhead, and a considerable level of security contingent upon the confidentiality of the key.

## **2.6 Types of Ciphers**

Symmetric cryptography encompasses two primary categories: Block and Stream ciphers. Broadly speaking, both of these ciphers serve as distinct methods for transforming plaintext (original message) into ciphertext (encrypted message).

### **2.6.1 Block Cipher**

In contrast to Stream Ciphers, Block Ciphers operate by processing the plaintext of one fixed-size [2]. For this, the plaintext is divided into blocks of equal size, and each block is encrypted individually. For example, if a block cipher is 256-bit, then it would encrypt each 256-bit block of message separately. If plaintext is smaller than the block size, then padding

techniques are employed to fill up the remaining space.

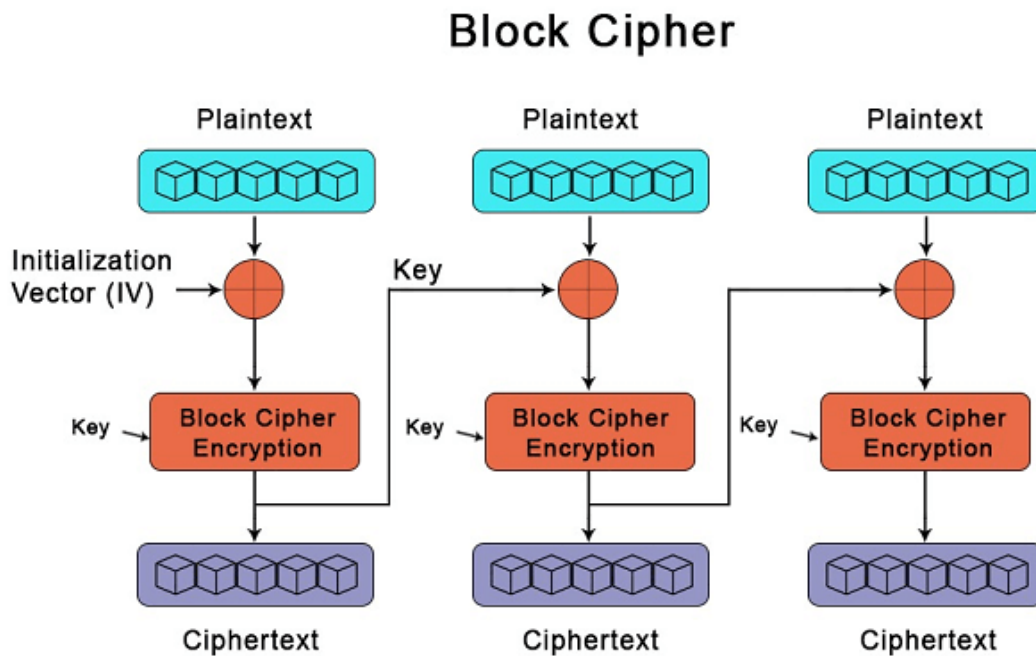


Figure 2.3: Block Cipher

## 2.6.2 Stream Cipher

Stream Ciphers encrypt the message either bit by bit or byte by byte [2]. It produces a keystream, which is then XORed with the plaintext to generate the ciphertext. Randomness of the keystream generator provides security to the stream cipher, however, there is a need to ensure that the secret key is utilized only once to encrypt the data. The fundamental objective of Stream Ciphers is to emulate the functionality of the One-Time Pad (OTP).

The OTP employs a completely random key generator, where the size of both the key and message are equal. However, this approach is impractical for real-world scenarios, where messages can be gigabytes in size, making key distribution and management infeasible. Consequently, key repetition becomes unavoidable, and perfect secrecy cannot be attained. Nevertheless, Stream Ciphers can still achieve a satisfactory level of security, albeit not perfect

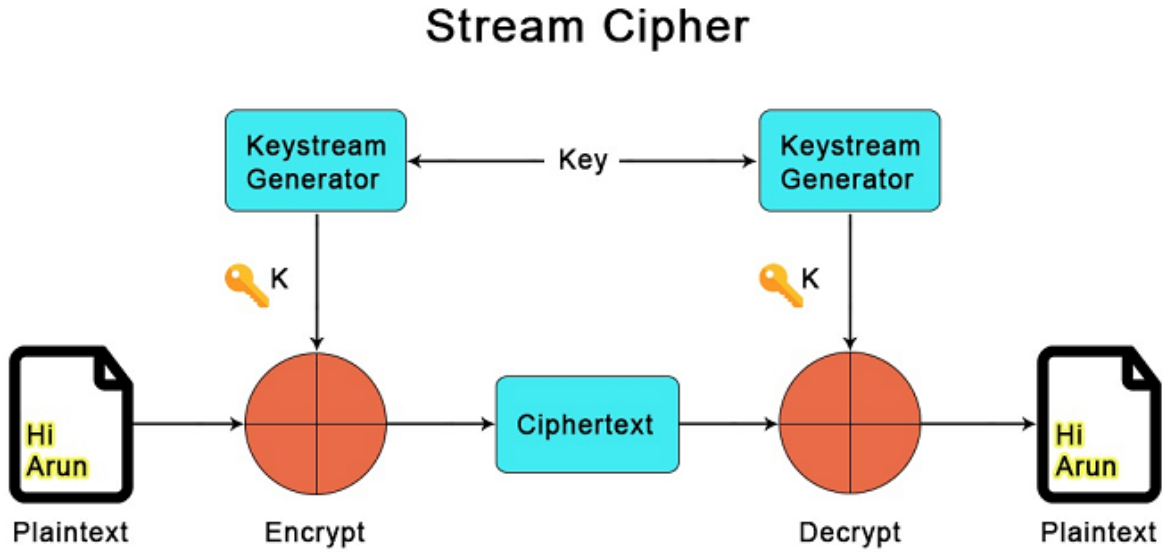


Figure 2.4: Stream Cipher

secrecy, by ensuring proper key management practices and avoiding key reuse.

## 2.7 Block Cipher Algorithms

Famous Block Cipher Algorithms are enumerated as follows.

### 2.7.1 Data Encryption Standard (DES)

Originating in the 1970s, DES was initially revered as a secure and standard encryption method for safeguarding data [1]. However, contemporary discourse surrounding DES primarily revolves around its historical significance, as it has become antiquated and vulnerable to various cryptographic attacks, particularly due to its insufficient key length.

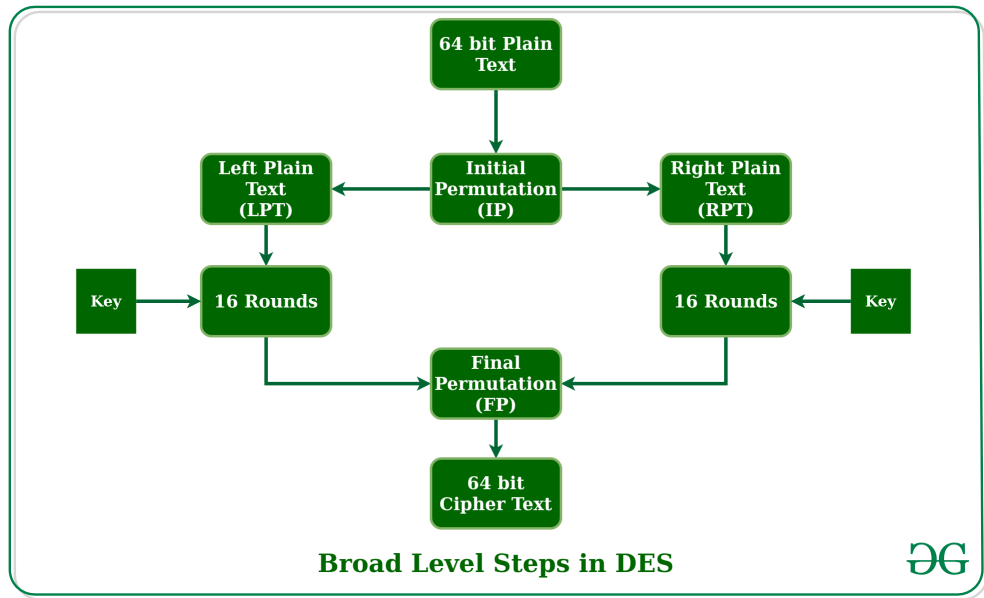


Figure 2.5: Data Encryption Standard (DES)

### 2.7.2 Triple Data Encryption Algorithm (3DES)

As its name suggests, in the Triple Data Encryption Algorithm (3DES), the DES cipher algorithm will be employed thrice for every block. It aims to enhance security by increasing the effective key length.

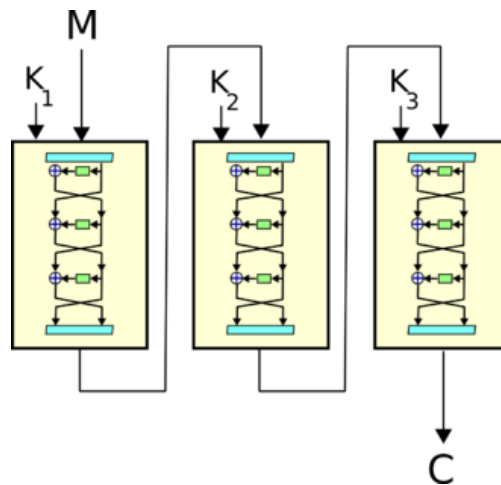


Figure 2.6: Triple Data Encryption Standard (3DES)

### 2.7.3 Advanced Encryption Standard

Advanced Encryption Standard (AES) emerged as the successor to DES, offering enhanced security and efficiency.[22] It functions as a symmetric block cipher, using a single key for both encryption and decryption. Unlike asymmetric encryption, which employs two keys, AES streamlines the process with just one secret key. Originally developed in the United States, AES has achieved international acclaim, emerging as one of the most extensively used and secure encryption algorithms. This underscores its effectiveness and versatility in protecting sensitive information worldwide.

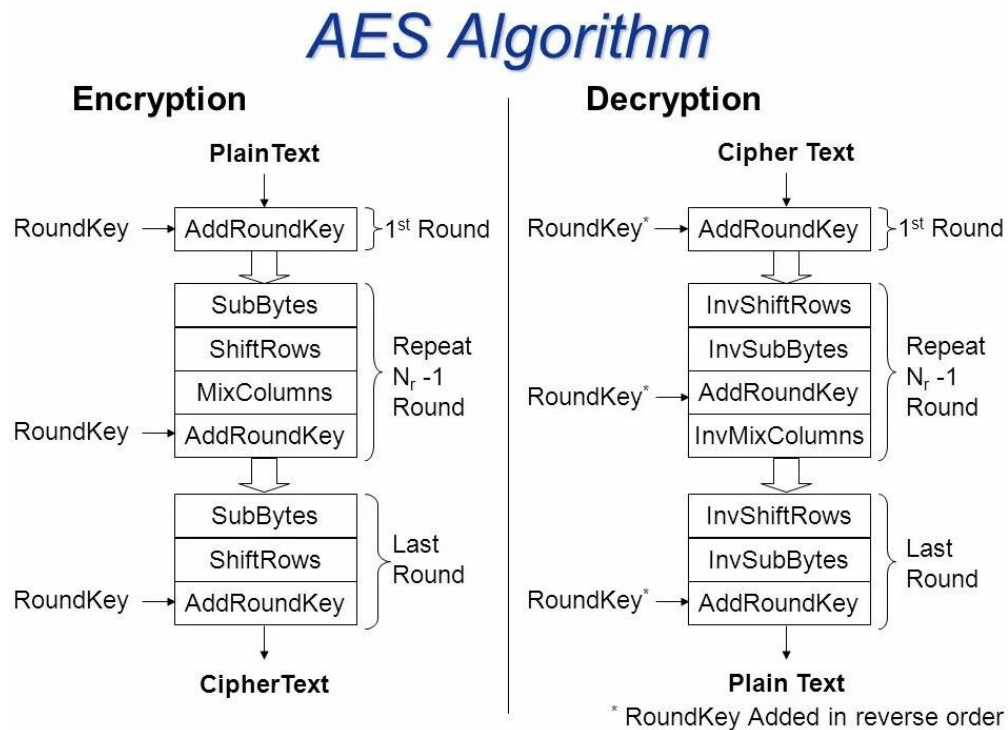


Figure 2.7: Advanced Encryption Standard (DES)

### 2.7.4 RC5 (Rivest Cipher)

RC5 is a symmetric-key block cipher designed by Ronald Rivest in 1994. [19] Here are some key features of RC5:

1. Variable Block Size: RC5 supports variable block sizes, allowing for encryption of data blocks of different lengths. The block size can be adjusted to 32, 64, or 128 bits.
2. Variable Key Size: RC5 supports variable key sizes, ranging from 0 to 2,040 bits, in increments of 8 bits. The recommended key size is between 0 and 256 bits.
3. Feistel Network Structure: RC5 employs a Feistel network structure, dividing the input block into two halves and applying a series of rounds to each half. The two halves are then combined to produce the ciphertext.
4. Security: RC5 has been widely studied and analyzed by cryptographers since its publication. While it has not been found to have any significant weaknesses, its security varies based on the choice of parameters and the quality of the key schedule.

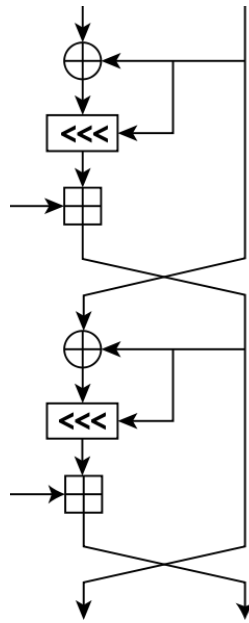


Figure 2.8: RC5 ALgorithm)



## 2.8 Stream Cipher Algorithms

### 2.8.1 RC4 (Rivest Cipher 4)

RC4 is a Cipher which is known for its simplicity and rapidity. It supports key lengths ranging from 40 to 2048 bits [13]. A pseudorandom bit generator produces a byte sequence known as the keystream. Each byte of the plaintext is then XORed with the corresponding byte of the keystream to generate the ciphertext. Predicting the keystream without knowledge of the key is deemed infeasible. However, RC4 is plagued by known vulnerabilities, particularly when the initial output keystream is retained or when non-random or related keys are employed.

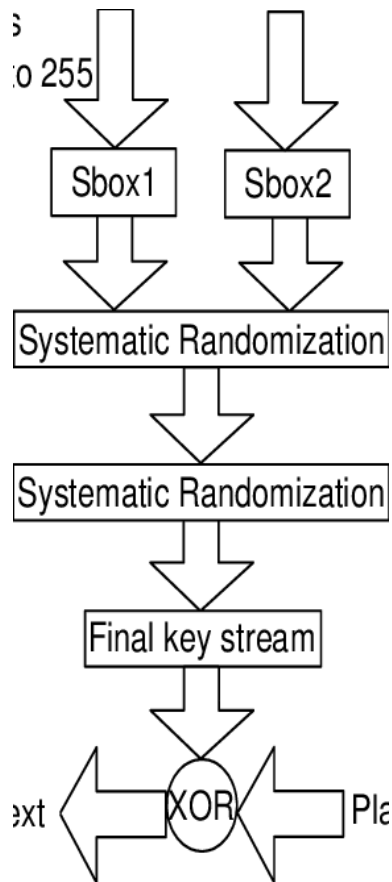


Figure 2.9: RC4

## 2.8.2 Salsa20 and ChaCha

Salsa20 and ChaCha, crafted by Daniel J. Bernstein, share striking similarities. Salsa20 debuted in 2005, with ChaCha following three years later in 2008 [8]. Both ciphers work on ARX (add-rotate-xor) operations, resulting in efficient performance and minimal hardware and software requirements. Key sizes for both ciphers are set at either 128 or 256 bits.

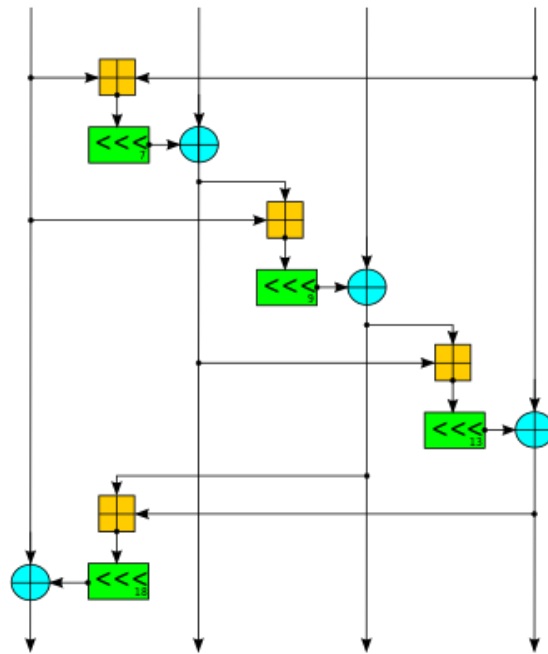


Figure 2.10: Salsa Algorithm

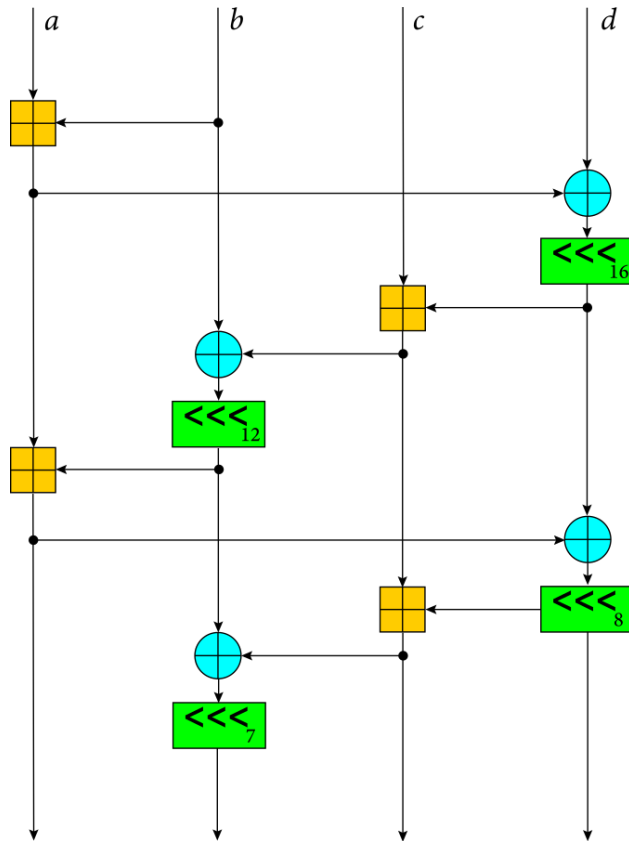


Figure 2.11: ChaCha20 Algorithm

## 2.9 Comparison of Cipher Algorithms

Determining the superior method for Lightweight cryptography presents a challenge, as each approach possesses distinct advantages and drawbacks, rendering a definitive answer elusive. Stream Ciphers, inherently swift, outpace Block Ciphers due to their ability to process data bit by bit, contrasting with the block-by-block processing required by the latter. Consequently, Block Ciphers necessitate larger memory allocations to handle sizable data chunks, often necessitating continued operations from preceding blocks. In contrast, Stream Ciphers, processing only a byte at most at a time, boast modest memory requirements, rendering them cost-effective for implementation, thus finding broader applicability in Lightweight cryptographic algorithms.

However, Stream Ciphers entail intricate development and design challenges and are susceptible to vulnerabilities contingent on their usage. The pseudorandom keystream generator

must adhere to stringent criteria resembling the One-Time Pad, demanding unpredictability and maximal randomness. Moreover, Stream Ciphers lack integrity protection and authentication, unlike certain Block Ciphers which, depending on their mode of operation, offer both confidentiality and integrity assurance. Furthermore, Block Ciphers, encrypting entire blocks at once and often utilizing feedback modes, are susceptible to introducing transmission noise that could distort data, rendering subsequent transmission unsuitable for the algorithm. In contrast, Stream Ciphers circumvent this issue by encrypting bits or bytes independently, with potential solutions available for connectivity issues.

In summary, Stream Ciphers excel in scenarios where data volume is indeterminate or continuous network streams prevail, such as live video streaming, while Block Ciphers shine in situations where data transmission volumes are predetermined, such as file transfers.

# Chapter 3

## Research Background

### 3.1 Introduction to Lightweight Cryptology

Cryptography serves as the cornerstone of modern information security, providing the means to protect sensitive data from unauthorized access and manipulation. Traditionally, cryptographic algorithms have been developed and optimized for high-performance computing devices such as servers, desktops, and smartphones [9]. However, the proliferation of resource-constrained devices in recent years, including sensors, RFID tags, and embedded systems, has necessitated the emergence of a specialized field known as Lightweight cryptography. Lightweight cryptography addresses the unique challenges posed by devices with limited computational resources, such as low-power processors, constrained memory, and energy efficiency requirements.

The significance of Lightweight cryptography stems from the fact that conventional cryptographic algorithms designed for high-performance devices are often unsuitable for deployment on resource-constrained platforms. These algorithms typically require extensive computational resources and memory, leading to inefficient performance and excessive energy consumption on lightweight devices.

In contrast, Lightweight cryptographic algorithms are specifically tailored to meet the stringent requirements of resource-constrained devices. They prioritize efficiency, minimizing

computational overhead and memory footprint.

Furthermore, the adoption of LWC is particularly crucial in upcoming future tech in the context of emerging technologies such as the Internet of Things (IoT)[9], where billions of interconnected devices collect, transmit, and process data in real time. Securing communication and data exchange among these devices is paramount to prevent unauthorized access, data breaches, and privacy violations.

In this context, [16] Lightweight cryptographic algorithms play a pivotal role in enabling secure communication, authentication, and data integrity in IoT ecosystems. They provide the necessary security mechanisms to safeguard sensitive information while meeting the stringent resource constraints of IoT devices.

Overall, LWC represents a critical area of research and development in modern cryptography, addressing the growing demand for efficient and secure cryptographic solutions for resource-constrained devices in an increasingly connected world.

## 3.2 Understanding LWC

Conventional cryptography, often referred to simply as cryptography, is tailored for devices like smartphones, tablets, laptops, and desktops. These devices typically boast substantial computational power, extensive memory capacity, and negligible constraints on disk space and other resources. Conversely, Lightweight cryptographic algorithms cater to devices such as embedded systems, RFID devices, and sensor networks, characterized by stringent limitations in memory, energy consumption, processing speed—critical in real-time applications—implementation costs, and more. Further constraining the capabilities and adaptability of potential algorithms. Consequently, these constraints significantly elevate the complexity and challenge of designing cryptographic algorithms tailored for such devices compared to conventional cryptography.

### 3.3 Challenges in LWC

In this section, we will explore the challenges encountered in the design of Lightweight cryptographic algorithms. These challenges involve trade-offs between performance, resources (often referred to as cost), and security levels required to be achieved.

Performance considerations can be broken down into several key metrics: power and energy consumption, latency, and throughput [21]. Firstly, the amount of power needed to operate the circuit. Lesser power requirement is better than it is for limited power sources like batteries or solar energy, where recharging or replacement may be impractical. Throughput, indicating the number of plaintexts processed per second, is another crucial metric. While high throughput is not always a primary design goal due to constraints, achieving an acceptable average throughput is necessary.

Latency refers to a measure of time needed for a certain design to complete a defined computational task, such as the time needed to generate ciphertext from plaintext. Lower latency is advantageous, particularly in real-time applications where speed is critical.

Resource allocation, categorized into hardware and software specifications, poses another challenge. Hardware resources are influenced by factors like gate equivalents, which specify the physical area required for the circuit. Software resources encompass ROM and RAM consumption, as well as code size. Both ROM and RAM usage should be minimized to optimize performance.

Security considerations are paramount and are closely intertwined with performance and resource allocation. A minimum security strength, typically a key length of at least 112 bits according to NIST recommendations, is essential to thwart attacks like brute force. However, using longer keys can increase memory requirements and potentially impact latency.

Furthermore, robustness against Side-Channel Attacks (SCAs) is crucial, particularly in

constrained devices common in Lightweight cryptography. SCAs exploit information about the algorithm's design and implementation, such as power consumption or decryption time, to reveal sensitive information. Preventative measures against SCAs involve randomizing inputs and standardizing encryption and decryption operation times to obfuscate attacker efforts.

In conclusion, designing Lightweight cryptographic algorithms is a complex endeavor fraught with challenges, given the delicate balance required among various performance metrics, resource constraints, and security imperatives.

### **3.4 LWC on Internet of Things (IoT)**

[9] This technology integrates a multitude of internet-connected devices, facilitating data exchange and communication regardless of geographical distance. Primarily comprised of sensors, IoT devices collect and transmit data, enabling various applications to enhance efficiency and convenience in people's lives.

Examples of IoT implementations abound, ranging from smart homes that automate heating and lighting based on sensor data to healthcare systems, automotive applications, automated factories, and smart cities. As IoT technology becomes increasingly pervasive, ensuring the security of these interconnected devices and the data they handle becomes paramount.

Sensitive information transmitted by IoT devices must remain confidential and unaltered, particularly in critical sectors like healthcare and automotive industries. However, conventional cryptographic methods are often unsuitable for IoT devices due to their resource constraints. Hence, Lightweight cryptographic algorithms are utilized to provide robust security in such constrained environments.

Despite the inherent challenges in achieving high-level security within these constraints, [14] the sensitivity and importance of the data necessitate the development of strong Lightweight



cryptographic algorithms.

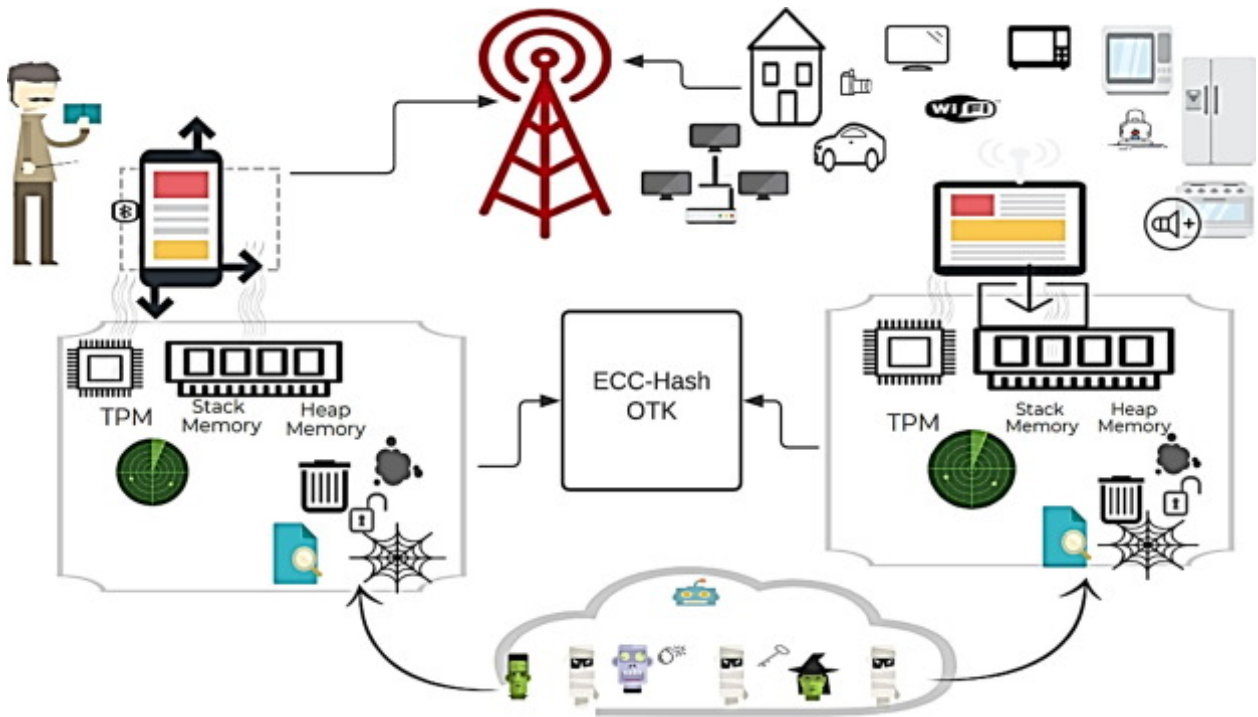


Figure 3.1: Internet of Things

## 3.5 Known LWC Algorithms

Numerous Lightweight cryptographic algorithms have been developed with the aim of striking a balance amidst the challenges and trade-offs discussed earlier. In this regard, a selection of the most prominent algorithms will be introduced to provide insight into the advancements made in this domain. This presentation will encompass both Block and Stream ciphers, offering a comprehensive overview of the progress achieved thus far.

### 3.5.1 Lightweight Block Ciphers

Several noteworthy Lightweight cryptographic algorithms have emerged, each addressing specific challenges while striving for optimal performance. Below are introductions to some of the most prominent algorithms, showcasing the progress made in this field:

**Present** - Developed jointly by Orange Labs (France), Ruhr University Bochum (Germany), and the Technical University of Denmark in 2007 [3], Present employs 64-bit blocks and supports keys of 80 and 128 bits. While the 80-bit key length may not be considered secure by contemporary standards, the algorithm is designed to meet moderate security requirements for specific applications. Present is admired for its simplicity and compact size. Moreover, it fulfills the design objectives outlined in the eSTREAM project, facilitating widespread adoption.

**Clefi**a - Introduced by SONY in 2007, Clefia is a proprietary Lightweight block cipher offering a choice of 128, 192, or 256-bit keys, operating on 128-bit blocks. Despite its proprietary nature, Clefia's specifications [23] are publicly available, encouraging evaluation and feedback from the global cryptographic community. Clefia maintains a high level of security suitable for both hardware and software implementations, with a focus on gate efficiency and shared functions between data processing and key scheduling components.

**Klein** - Developed in 2011, Klein operates on 64-bit blocks with key lengths of 64, 80, or 96 bits. [11] The choice of key length and block size is crucial in balancing security and performance trade-offs, with Klein's flexibility catering to various encryption and authentication scenarios.

While Present, Clefia, and Klein are internationally standardized ciphers, numerous other Lightweight block ciphers have been developed, including LED, Midori, Mantis, HIGHT, and GOS, among others.

### 3.5.2 Lightweight Stream Ciphers

**Enocoro** - developed by Hitachi, Ltd. in 2007 and updated to Enocoro-128v2 in 2010, offers a family of pseudorandom number generators. This algorithm, submitted to CRYPTREC, supports a 128-bit key and features an initialization function for generating keys. [26] Despite its low implementation cost, Enocoro is resilient against cryptanalytic attacks.

**Trivium** - [7] This flexible stream cipher generates up to 264 bits of the key stream from an 80-bit secret key and IV, making it a probabilistic model akin to Enocoro. While its key length may not offer ideal security, Trivium's resilience to cryptanalytic attacks ensures robust performance in various implementations.

**Grain** - Initially designed for the eSTREAM competition in 2004, Grain boasts a compact design suitable for hardware implementations. [12] With versions supporting 80-bit and 128-bit keys, Grain offers improved security while maintaining low gate area and power consumption. Its ability to enhance performance with additional hardware resources sets it apart from other stream ciphers.

# Chapter 4

## Proposed Lightweight Stream Cipher Algorithm

### 4.1 Overview

The proposed algorithm is designed for computers connected through the internal LAN of the Indian Navy, utilizing a 128-bit key over a block of the same size. The algorithm executes 4 rounds to generate the cipher. It is meticulously designed with innovative features, including a unique algebraic permutation technique to achieve diffusion and a 16x16 substitution box to establish the confusion property.

The algorithm contains the following main functions in each round which are:

1. Permutation
2. Substitution using 16X16 AES S-box
3. Add Key

## 4.2 Detailed Structure

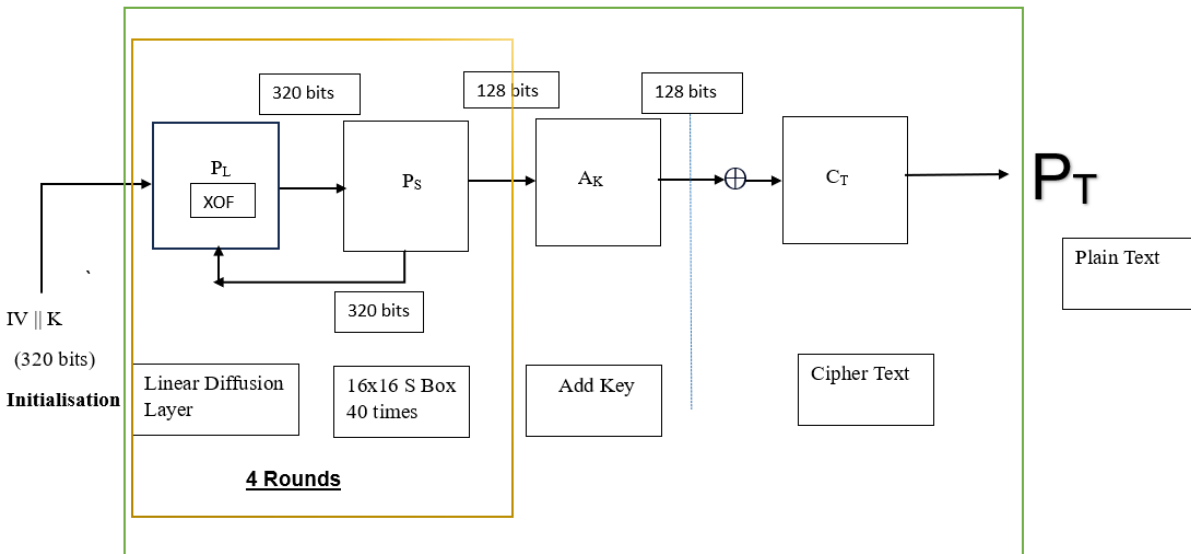


Figure 4.1: Detailed structure of LWC Encryption

The functions are elaborated in the following paragraphs:

### 4.2.1 Parameter Description

The parameters of the design are as follows: -

Secret Keys: 128 bits, Number used Once (to prevent replay attacks and ensure uniqueness of encryption): 64 bits, IV: 192 bits, Plain Text block size: 128 bits, and rounds: 4.

### 4.2.2 Initialization Phase

In the figure below, the Internal State is divided into 5 rows, each consisting of 64 bits (columns). In the Initialization Phase, the Internal State is structured as  $IV \parallel K$ . Consequently, the first three rows  $y_0, y_1, y_2$  contain the IV (192 bits). The last two rows  $y_3, y_4$  contain the secret key (128 bits). This stage takes the IV and K by the user and concatenates them to form a 320-bit internal state.

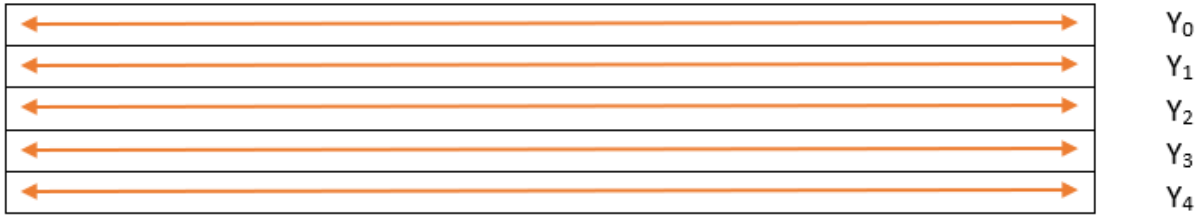


Figure 4.2: 320 bits Internal State

### 4.2.3 Linear Diffusion Phase

In this step, each row undergoes modification using XOR operations and right circular rotations ( $\ggg$ ), which involve rotating bits by random numbers to the right. These random numbers are generated by an XOF function that receives a Nonce and produces 10 integers, denoted as  $X_1$  to  $X_{10}$ . For each round, a new set of 10 integers is generated, where the Nonce is incremented by one, functioning like a counter.

For the operation on row  $y_0$ ,  $y_0$  is XOR'd with ( $y_1$  right circular rotated by the random number  $X_1$ ) XOR  $y_2$  (right circular rotated by the random number  $X_2$ ). This process repeats for the remaining rows, ensuring each row is adjusted to distribute the impact of a single bit. The result of this phase will yield 320 bits, which will then proceed to the next phase.

$$\begin{aligned}
 Y_0 &\rightarrow \Sigma_0(Y_0) = Y_0 \oplus (Y_1 \ggg X_1) \oplus (Y_2 \ggg X_2) \\
 Y_1 &\rightarrow \Sigma_0(Y_1) = Y_1 \oplus (Y_2 \ggg X_3) \oplus (Y_3 \ggg X_4) \\
 Y_2 &\rightarrow \Sigma_0(Y_2) = Y_2 \oplus (Y_3 \ggg X_5) \oplus (Y_4 \ggg X_6) \\
 Y_3 &\rightarrow \Sigma_0(Y_3) = Y_3 \oplus (Y_4 \ggg X_7) \oplus (Y_0 \ggg X_8) \\
 Y_4 &\rightarrow \Sigma_0(Y_4) = Y_4 \oplus (Y_0 \ggg X_9) \oplus (Y_1 \ggg X_{10})
 \end{aligned}$$

## 4.2.4 Substitution Phase

In this phase, we perform S-box substitution for undertaking transformation, which is non-linear, and it will be applied to the linearly diffused internal state. This involves using a predefined S-box, which serves as a lookup table to replace input bits with corresponding output bits. The substitution process introduces non-linearity and provides confusion, making it difficult for attackers to find patterns or relationships between the input and the output.

This AES S-box substitution is implemented as a lookup table and will be executed 40 times to generate a 320-bit output. By repeatedly applying the S-box substitution, we enhance the complexity and security of the cryptographic process, ensuring that the final output is thoroughly transformed and resistant to cryptanalysis.

	right (low-order) nibble															
left	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	3a	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	91	44	3c	88	07	c7	31	b1	12
c	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
d	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d
e	87	84	52	90	88	33	64	7f	24	b5	1e	a8	74	09	3f	ce
f	f9	20	c9	8e	5b	6f	19	b3	ea	39	42	27	f1	f6	e7	30

Table 4.1: AES S Box

### 4.3 Encryption

The user will define the initialization vector (IV), key (K), and nonce (N), and then the IV and Key will be concatenated as  $IV \parallel K$  to construct a 320-bit internal state. This 320-bit state will undergo a single round of Linear Diffusion phase. The output from this stage will then be passed through a 16x16 AES S-Box and will be iterated 40 times, yielding a 320-bit output.

Subsequently, a total of four such rounds will be executed. Following the fourth round, 128 significant bits of the resulting 320-bit output will be XORed with a 128-bit key. The result of this XOR operation will then be XORed with the plaintext to produce a 128-bit ciphertext. This process will be repeated for the remaining plaintext blocks, with the stipulation that a unique IV and Nonce must be provided for each 128-bit block of plaintext.

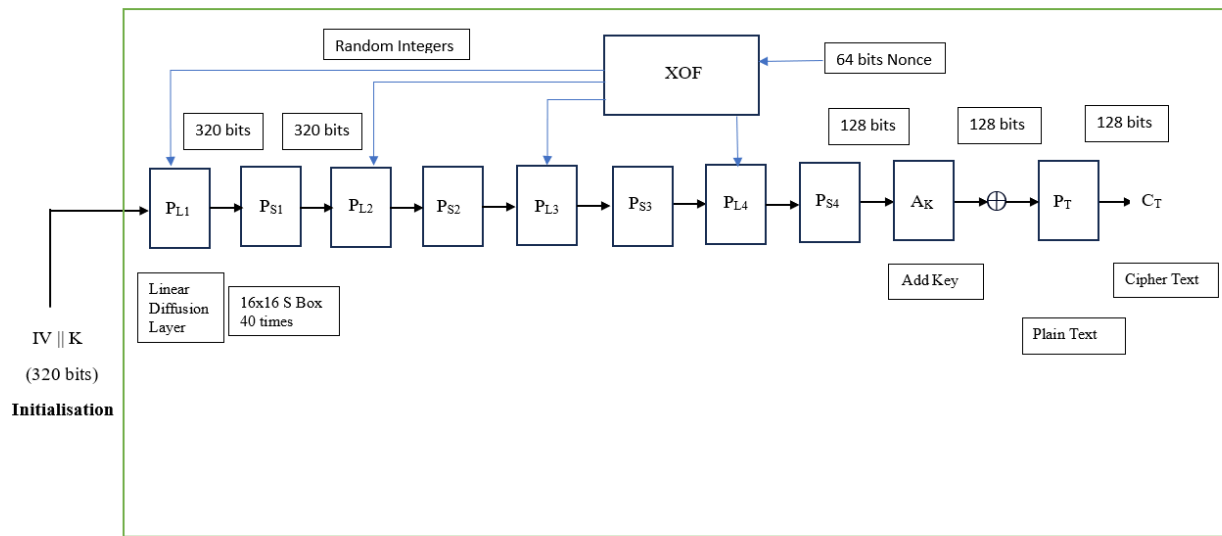


Figure 4.3: Encryption Algorithm Flow

### 4.4 Decryption

The user will define the initialization vector (IV), key (K), and nonce (N), and then the IV and Key will be concatenated as  $IV \parallel K$  to construct a 320-bit internal state. This 320-bit



state will undergo a single round of Linear Diffusion phase. The output from this stage will then be passed through a 16x16 AES S-Box and will be iterated 40 times, yielding a 320-bit output.

Subsequently, a total of four such rounds will be executed. Following the fourth round, 128 significant bits of the resulting 320-bit output will be XORed with a 128-bit key. The result of this XOR operation will then be XORed with the cipher text to produce a 128-bit plain text. This process will be repeated for the remaining cipher text blocks, with the stipulation that a unique IV and Nonce must be provided for each 128-bit block of cipher text.

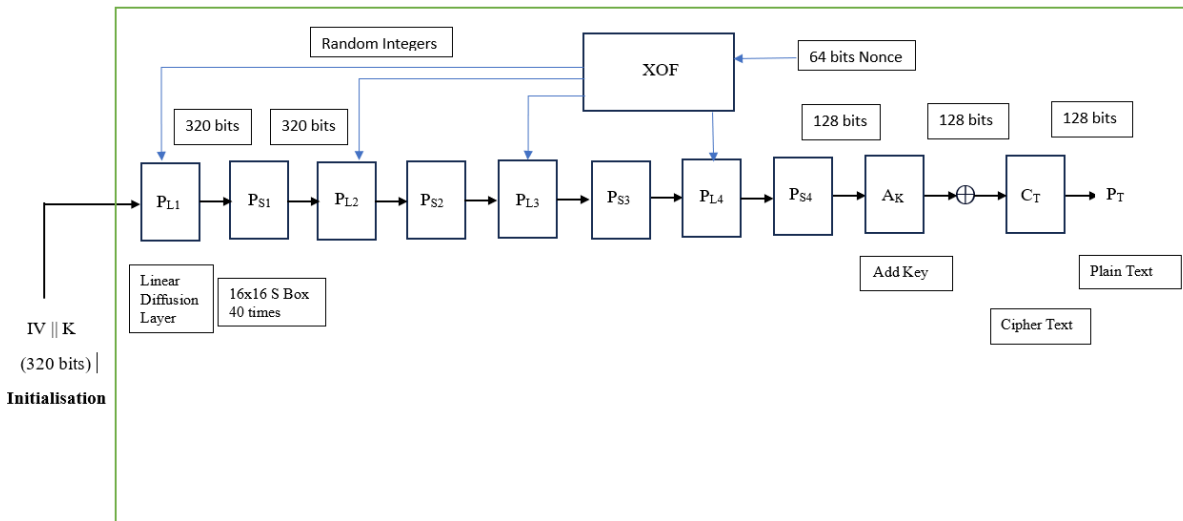


Figure 4.4: Decryption Algorithm Flow

# Chapter 5

## Cryptanalysis of Algorithm

An algebraic attack on an algorithm is performed to deduce the secret key by solving non-linear equations involving message, ciphertext, and key bit. [15].

In this chapter, we detail the methods for generating the system of polynomial equations and explain that our algorithm is resistant to the Differential Fault attack and the algebraic attack. We demonstrate that with the current computing power available, it is not feasible to execute an algebraic attack on our designed algorithm.

### 5.1 Differential Cryptanalysis - Resistance to Differential Fault Attack

Differential Fault Attack (DFA) exploits the vulnerability of cipher by injecting faults when the keys are being generated, and then the analysis of the difference is done in order to recover the key. DFA can exploit various weaknesses in the cipher's design and predict the behavior of the fault propagation. DFA relies on several critical assumptions to compromise the security of stream ciphers. Firstly, attackers assume they can repeat the attack on the cipher multiple times with the same secret key and public parameters, allowing them to collect both normal and faulty ciphertexts for analysis. Secondly, they need the capability to inject faults precisely at a specific time during the keystream generation, which necessitates

access to required tools capable of injecting the fault in a particular bit(s) of the cipher without causing damage. Once faults are injected, the attacker proceeds by identifying the fault position (if possible) and analyzing the initial and change. The attacker will then solve this system of equations in order to deduce the internal state and subsequently obtain the secret key.

### 5.1.1 DFA on Our Stream Cipher

To perform DFA on our proposed stream cipher, initially we will be injecting a single bit of fault in the key when it is being generated. We illustrate the attack in Sagemath.

```
1      X = key + iv + nonce
2      eX = T[0:v]
3      fkey[0] = fkey[0] + 1
4      eXf0 = T[0:v]
5      eXf0[0] = eXf0[0] + 1
6      Xf = fkey + iv + nonce
7      eXf = eXf0
8
```

In the above code snippet, X stores the 320-bit state concatenating the 128-bit key, a 64-bit initialization vector (IV), and the 128-bit nonce and similarly, Xf with the faulty key. eX and eXf store the state and faulty as the unknown variables, respectively. fkey stores the faulty key, and eXf0 stores the variables for the faulty key.

```
1      sys = []
2
3      for i in range(rounds):
4          x0, x1, x2, ..., x19 = random()
5          X = l1ayer(X, x0, x1, ..., x19)
6          eX = l1ayer(eX, x0, x1, ..., x19)
7          Xf = l1ayer(Xf, x0, x1, ..., x19)
8          eXf = l1ayer(eXf, x0, x1, ..., x19)
```

```

9
10     X = Sfunction(X)
11     eX = Sfunction(eX)
12     Xf = Sfunction(Xf)
13     eXf = Sfunction(eXf)
14
15     temp = T[1:1+320]
16     l = l + 320
17     tempf = T[1:1+320]
18     l = l + 320
19     for j in range(320):
20         sys.append(eX[j] + temp[j])
21         sys.append(eXf[j] + tempf[j])
22     eX = temp
23     eXf = tempf
24

```

sys[ ] is the list which is storing all the equations. The for loop acts as the round function generating the equations using normal and faulty keystreams. The random function produces twenty random numbers, which can be used in the linear layer. The llayer() function takes the state and random numbers as input and returns the state after linear operations. This layer propagates the fault to five positions due to the  $\sum(Y_i)$  function in the linear layer. The Sfunction() takes 8-bit input and produces 8-bit output, this mapping is based on the  $8 \times 8$  S-box. If one of the 8-bit in the input is fault affected then 8-bit output will be fault affected. This happens because each output bit is dependent on every input bit. Due to this, the fault propagates, and each output bit is fault-affected. We have a 7-degree output from this layer. So, After each round, we replace the state with new variables(temp and tempf) to generate the equations efficiently.

```

1     for i in range(ksize):
2         e = k0 + k0

```

```

3     e = eX[i] + X[i] + T[i] + key[i]
4     sys.append(e)
5     e = k0 + k0
6     e = eXf[i] + Xf[i] + eXf0[i] + fkey[i]
7     sys.append(e)
8

```

Now, we consider the first 128 bits in the state, perform the key XOR, and store the equations in `sys[ ]`.

```

1     print(len(sys))
2     print("SAT solver running")
3     Start_Time = time.time()
4     sol = solve(sys, n=10, target_variables=[T[0:ksize]])
5     print(sol)
6     print(len(sol))
7     Execution_Time = time.time() - Start_Time
8     print(Execution_Time)
9

```

Thus, `sys[ ]` has all the equations. We pass the system (`sys[ ]`) to the SAT solver (i.e., `solve()`) and try to obtain the secret by solving for the key variables that are set as the target variables. If there are multiple solutions we go for more number of keystreams.

After performing the above experiment, we couldn't find any solution. Even after optimizing the system of equations by introducing new variables, we have equations involving a large number of monomials, and at each round, they are almost 7 degrees. Due to this, the SAT solver is not able to produce any solution to the given system.

## 5.2 Linear Cryptanalysis - Resistance to Algebraic Attack

Our proposed stream cipher is based on a linear layer and a nonlinear layer. In the computation of nonlinear layer we are using AES-Subbyte S-box. This S-box is a mapping from  $\{0, 1\}^8 \rightarrow \{0, 1\}$ . It can be noted that the component functions of this S-box are of degree 7. After each linear layer, we perform the nonlinear layer. If we consider the secret key as unknown and try to estimate the degree of the Kesytreem bit equations, then we find that the degree of each state bit is increased by a multiple of 7. Thus, it is quite obvious that the degree of each keystream bit equation in terms of the unknown secret key will be 128. We construct  $N$  number of keystream bit equations, and each will be of degree 128. Our selection of  $N$  is done in such a way that the final system becomes over-defined. We finally linearize the entire system and solve the system using the Gauss elimination method. The number of variables required to linearize the system is  $M = \sum_{i=2}^{128} \binom{128}{i} \approx 2^{128}$ . The time complexity to solve the system is  $\approx N^{2.8} > 2^{128}$ . Thus, we strongly believe that our proposed stream cipher is secure against Algebraic attacks.

## 5.3 Our Contribution

In this thesis, we have embarked on an innovative journey by developing a sophisticated Lightweight Stream Cipher. Inspired by ASCON, FASTA, and PASTA Lightweight Cryptography (LWC) algorithms known for their ingenuity, we meticulously crafted our cipher design, drawing on cryptographic principles.

Our exploration goes beyond theory as we translate our concepts into reality. Using the C programming language, we brought our cipher to life, ensuring every line of code reflects our vision of cryptographic excellence.

Recognizing the critical importance of security in today's digital landscape, we subjected our creation to rigorous scrutiny. Through comprehensive security analysis, we thoroughly

evaluated our stream cipher’s resilience against various vulnerabilities and threats, including algebraic attacks—where adversaries attempt to break cryptographic algorithms by solving equations derived from their algebraic structure—and differential fault attacks—exploiting vulnerabilities introduced by faults in cryptographic computations. Our cipher’s robust design mitigates these threats, ensuring secure operation in the face of potential adversarial techniques.

### **Efficiency through Innovative Design: AES S-Box and Diffusion Layer**

In the realm of cryptographic design, achieving a balance between security and efficiency is an enduring challenge, particularly in resource-constrained environments like naval operations. This section explores the strategic design decisions behind a lightweight stream cipher customized for the Indian Navy, with a special focus on integrating the AES S-Box and introducing a novel diffusion layer.

#### **AES S-Box Integration**

AES is known for its strong security and efficiency, making it a natural choice for cryptographic primitives even in lightweight applications [6]. Central to the AES encryption process is its S-Box, a nonlinear substitution layer crucial for achieving diffusion and resistance against various cryptanalytic attacks. In our lightweight stream cipher design, we have used the AES S-Box. The S-Box transformation is designed to provide a high degree of nonlinearity, thereby enhancing resistance against differential and linear cryptanalysis. [20] By carefully selecting the coefficients and operations within the S-Box, we achieve a balance between cryptographic security and computational efficiency, crucial for real-time data processing within LAN environments of naval communication systems.

#### **Diffusion Layer with Random Right Rotational Shifting**

The diffusion layer plays a pivotal role in spreading the influence of each plaintext or ciphertext bit throughout the entire block of 320 bits, thereby increasing the cryptographic strength of the cipher. In our design, we employ a novel approach using random right rotational shifting of bits across three distinct rows. This strategy aims to enhance both confusion and diffusion properties, critical for thwarting statistical attacks and maintaining robustness against differential and linear cryptanalysis. Each row in the diffusion layer is assigned a unique rotation parameter, randomly selected during initialization. This randomization introduces variability in the bit-level transformations across successive rounds, thereby fortifying the cipher against known plaintext attacks and promoting resilience in the face of sophisticated cryptanalytic techniques. The choice of three different rows for rotational shifting ensures that the diffusion process is comprehensive and effectively mitigates any potential biases that might arise from deterministic shifting patterns.

### **Practical Implications and Naval Application**

The integration of these innovative design elements—AES S-Box with tailored optimizations and a diffusion layer employing random right rotational shifting—has profound implications for naval communication systems. By leveraging the efficiency and security benefits of the AES S-Box, our lightweight stream cipher ensures secure data transmission and processing within the Indian Navy’s internal LAN infrastructure.

Moreover, the adoption of a randomized approach to rotational shifting in the diffusion layer enhances the cipher’s resilience to cryptanalytic attacks while maintaining computational efficiency. This design choice aligns with the operational requirements of naval environments, where reliability, speed, and security are paramount. The cipher’s ability to operate effectively within these constraints underscores its suitability for safeguarding sensitive communications, supporting mission-critical operations, and enhancing overall cybersecurity posture.

Hence, the incorporation of the AES S-Box and the innovative diffusion layer in our lightweight stream cipher represents a significant advancement in cryptographic design for naval applica-



tions. By focusing on efficiency without compromising security, we have developed a robust encryption algorithm capable of meeting the stringent requirements of modern naval communication systems. Future research will continue to explore enhancements and optimizations to further strengthen the cipher's performance and adaptability in evolving cybersecurity landscapes.

This design approach not only addresses current challenges but also positions the Indian Navy at the forefront of secure communication technologies, ensuring operational readiness and resilience against emerging cyber threats.

In essence, our thesis stands as a testament to the relentless pursuit of innovation and excellence in the field of Lightweight Cryptography. Through our endeavors, We aim not only to push the boundaries of cryptographic research but also to empower individuals and organizations with the tools they need to safeguard their digital assets in an increasingly complex and interconnected world.

# Chapter 6

## Conclusion and Future Work

In this concluding chapter, we summarize our most important conclusions into three distinct parts.

Firstly, the design of this algorithm draws inspiration from ASCON, integrating randomization within the diffusion layer. This approach enhances security by making it extremely challenging for attackers to predict internal states. By introducing variability, the algorithm complicates potential attack vectors, ensuring a more robust cryptographic structure. This innovative use of randomization sets the algorithm apart, contributing to its novelty and effectiveness in safeguarding data.

Secondly, we conducted linear and differential cryptanalysis, demonstrating that our algorithm is resistant to these attacks. This highlights the robustness of our design against vulnerabilities. Additionally, we suggest future research directions, such as exploring correlation and difference propagation properties. By promoting transparency in design principles rather than obscurity, we aim to develop robust cryptographic solutions that address evolving cybersecurity challenges.

Thirdly, we highlight the application of our lightweight stream cipher in the context of the Indian Navy. Designed to operate efficiently on resource-constrained devices while ensuring secure communication and data integrity, our cipher integrates features from AES

S-box complexity and PASTA algorithm principles, supported by the underlying structure of ASCON. This tailored approach addresses the specific security needs of the Indian Navy's internal LAN communications, exemplifying the practical relevance and effectiveness of our cryptographic design.

Throughout this thesis, we have strived to transcend traditional cryptographic paradigms by proposing novel design strategies grounded in practical applicability and resilience against modern cryptographic attacks.

Our future work will involve conducting additional cryptographic attacks to further ensure the algorithm's security. We plan to submit the algorithm to the Indian Navy/WESEE and DRDO/SAG for review, feedback, and potential implementation. The focus will be on optimizing the algorithm for internal LAN environments and Motorola, ensuring robustness and efficiency in real-world applications.

# Bibliography

- [1] Barhoush, M., Abed-alguni, B., Hammad, R., Al-Fawa'reh, M., Hassan, R.: Des22: Des based algorithm with improved security. *Jordanian Journal of Computers and Information Technology* 8, 1 (03 2022), <https://10.5455/jjcit.71-1632868199>
- [2] Biryukov, A.: Block ciphers and stream ciphers: The state of the art. *IACR Cryptol. ePrint Arch.* 2004, 94 (2004), <https://api.semanticscholar.org/CorpusID:2393465>
- [3] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: Present: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2007*. pp. 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), [https://doi.org/10.1007/978-3-540-74735-2\\_31](https://doi.org/10.1007/978-3-540-74735-2_31)
- [4] Brown, A.: The basics of cryptography. *Towards Data Science* (Jan 2019), <https://towardsdatascience.com/the-basics-of-cryptography-80c7906ba2f7>, accessed: 12/12/2019
- [5] Chandra, S., Paira, S., Alam, S.S., Sanyal, G.: A comparative survey of symmetric and asymmetric key cryptography. *2014 International Conference on Electronics, Communication and Computational Engineering, ICECCE 2014* 11, 83–93 (2014), <https://doi.0.5455/jjcit.71-1632868199>
- [6] Das, S., Zaman, J.U., Ghosh, R.: Generation of aes s-boxes with various modulus and additive constant polynomials and testing their randomization. *Procedia Technology* 10, 957–962 (2013), <https://www.sciencedirect.com/science/article/>

- pii/S2212017313006051, first International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013
- [7] De Cannière, C.: Trivium: A stream cipher construction inspired by block cipher design principles. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) Information Security. pp. 171–186. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), [https://doi:10.1007/11836810\\_13](https://doi.org/10.1007/11836810_13)
- [8] Dey, S., Sarkar, S.: A theoretical investigation on the distinguishers of salsa and chacha. Discrete Applied Mathematics 302, 147–162 (10 2021)
- [9] Dutta, I.K., Ghosh, B., Bayoumi, M.: Lightweight cryptography for internet of insecure things: A survey. In: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). pp. 0475–0481 (2019)
- [10] El Assad, S., Lozi, R., Puech, W.: Cryptography and Its Applications in Information Security. Safwan El Assad (04 2022)
- [11] Gong, Z., Nikova, S., Law, Y.W.: Klein: A new family of lightweight block ciphers. In: Juels, A., Paar, C. (eds.) RFID. Security and Privacy. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
- [12] Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: 2006 IEEE International Symposium on Information Theory. pp. 1614–1618 (2006)
- [13] Jindal, P., Singh, B.: Rc4 encryption-a literature survey. Procedia Computer Science 46, 697–705 (2015), <https://www.sciencedirect.com/science/article/pii/S1877050915001933>, proceedings of the International Conference on Information and Communication Technologies, ICICT 2014, 3-5 December 2014 at Bolgatty Palace & Island Resort, Kochi, India
- [14] Khalifa, M., Algarni, F., Ayoub Khan, M., Ullah, A., Aloufi, K.: A lightweight cryptography (lwc) framework to secure memory heap in internet of things. Alexandria Engineering Journal 60(1), 1489–1497 (2021), <https://www.sciencedirect.com/science/article/pii/S1110016820305858>

- [15] Masoodi, F., Alam, S., Bokhari, M.U.: An analysis of linear feedback shift registers in stream ciphers (May 2012), [Online]. Available: <https://ijcaonline.org/archives/volume46/number17/7013-9714/>. DOI: 10.5120/7013-9714.
- [16] McKay, K., Bassham, L., Turan, M.S., Mouha, N.: Report on lightweight cryptography (2017-03-28 00:03:00 2017), [https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=922743](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=922743)
- [17] Mohajerani, K., Haeussler, R., Nagpal, R., Farahmand, F., Abdulgadir, A., Kaps, J.P., Gaj, K.: Fpga benchmarking of round 2 candidates in the nist lightweight cryptography standardization process: Methodology, metrics, tools, and results (2020), [Online]. Available: <https://eprint.iacr.org/2020/1207>
- [18] Rajesh, S., Paul, V., Menon, V.G., Khosravi, M.R.: A secure and efficient lightweight symmetric encryption scheme for transfer of text files between embedded iot devices. *Symmetry* 11(2) (2019), <https://www.mdpi.com/2073-8994/11/2/293>
- [19] Rivest, R.L.: The rc5 encryption algorithm. In: Preneel, B. (ed.) *Fast Software Encryption*. pp. 86–96. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
- [20] Roslan, M.F.B., Seman, K., Halim, A.H.A., Sayuti, M.N.A.S.M.: Substitution box design based from symmetric group composition. *Journal of Physics: Conference Series* 1366(1), 012001 (nov 2019), <https://dx.doi.org/10.1088/1742-6596/1366/1/012001>
- [21] Sadkhan, S.B., Salman, A.O.: A survey on lightweight-cryptography status and future challenges. In: *2018 International Conference on Advance of Sustainable Engineering and its Application (ICASEA)*. pp. 105–108 (2018)
- [22] Santhanalakshmi M, Lakshana K, S.G.M.: Enhanced aes-256 cipher round algorithm for iot applications. *The Scientific Temper* 14(01), 184–190 (Mar 2023), <https://scientifictemper.com/index.php/tst/article/view/371>

- [23] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher clefia (extended abstract). In: Biryukov, A. (ed.) Fast Software Encryption. pp. 181–195. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
- [24] Siegel, J.: Cryptography. umsl.edu (University of Missouri – St.Louis) (2019), [http://www.umsl.edu/~siegelj/information\\_theory/projects/des.netau.net/Cryptography%20and%20goals.html](http://www.umsl.edu/~siegelj/information_theory/projects/des.netau.net/Cryptography%20and%20goals.html), accessed: 12/12/2019
- [25] Team, F.: What is non-repudiation? Finjan blog (Feb 2017), <https://blog.finjan.com/what-is-non-repudiation/>, accessed: 12/12/2019
- [26] Watanabe, D., Owada, T., Okamoto, K., Igarashi, Y., Kaneko, T.: Update on encoro stream cipher. In: 2010 International Symposium On Information Theory & Its Applications. pp. 778–783 (2010)

# Appendix A

## C Code for implementation of the Lightweight Stream Cipher

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include <string.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #include <unistd.h>
7 #include <sys/types.h>
8 #include <math.h>
9 #include <inttypes.h>
10 #include "methods.h" // Methods.h contains all utility methods
11
12 #define IV_FILE "InitialVector.txt"
13 #define TEXT_FILE "Text.txt"
14 #define ENCRYPTED_FILE "Encrypted.txt"
15 #define DECRYPTED_FILE "Decrypted.txt"
16
17 typedef unsigned char uc;
```



```

18
19 int main() {
20     uc *key, *iv, *nonce, *state, *text, *cipher, *decr;
21
22     // Allocate memory for key, iv, and nonce
23     allocateByteMemory(&key, KEY_SIZE);
24     allocateByteMemory(&iv, IV_SIZE);
25     allocateByteMemory(&nonce, NONCE_SIZE);
26
27     // Prompt the user and read key, iv, and nonce directly
28     printf("Enter %d-bit Key in hexadecimal : ", KEY_SIZE);
29     scanf("%s", key);
30
31     // printf("Enter %d-bit IV in hexadecimal: ", IV_SIZE);
32     // scanf("%s", iv);
33
34     printf("Enter %d-bit Nonce in hexadecimal: ", NONCE_SIZE);
35     scanf("%s", nonce);
36
37     // Allocate memory for the state and input text
38     allocateByteMemory(&state, STATE_SIZE);
39
40     // Prompt user for input text (128 bits)
41     // printf("Enter 128-bit Input Text : ");
42     // scanf("%s", text);
43
44     // Open and read iv from file
45     FILE *ivFile = fopen(IV_FILE, "r");
46     readFile(iv, IV_SIZE, ivFile);
47     fclose(ivFile);

```

```

48
49 // Open and read text from file
50 FILE *textFile = fopen(TEXT_FILE, "r");
51 text = (uc*)malloc(KEY_SIZE); // Assuming input is 128 bits
    (16 bytes)
52 fread(text, sizeof(uc), KEY_SIZE, textFile);
53 fclose(textFile);
54
55 // Create starting state
56 createStartingState(state, key, iv);
57
58 // Perform internal operations
59 internalOperation(state, nonce);
60
61 // Encrypt the text using addKey method
62 cipher = addKey(text, state);
63
64 FILE *encrFile = fopen(ENCRYPTED_FILE, "w");
65 fprintf(encrFile, "%s", cipher);
66 fclose(encrFile);
67
68 // Display or use the resulting cipher as needed
69 printf("Encrypted Text:\n");
70 for (int i = 0; i < KEY_SIZE / BITS_IN_BYTE; ++i) {
71     printf("%02x ", cipher[i]);
72 }
73 printf("\n");
74
75 // Decrypt the text using addKey method
76 decr = addKey(cipher, state);

```

```
77
78     FILE *decrFile = fopen(DECRYPTED_FILE, "w");
79     fprintf(decrFile, "%s", decr);
80     fclose(decrFile);
81
82     // Displaying Decrypted Text
83     printf("Decrypted Text:\n");
84     printf("%s", decr);
85     printf("\n");
86
87     // Free allocated memory
88     free(key);
89     free(iv);
90     free(nonce);
91     free(state);
92     free(text);
93     free(cipher);
94
95     return 0;
96 }
```

## Methods.h

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include <string.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #include <unistd.h>
7 #include <sys/types.h>
8 #include <math.h>
9 #include <inttypes.h>
10 #include <openssl/evp.h>
11
12 typedef unsigned char uc;
13
14 #define STATE_SIZE (KEY_SIZE + IV_SIZE)
15 #define IV_SIZE 192
16 #define KEY_SIZE 128
17 #define NONCE_SIZE 64
18 #define LD1_NUM_SUBSTATES 5
19 #define LD1_SUBSTATES_SIZE 64
20 #define LD1_SUBSTATES_CONST1 2
21 #define LD1_SUBSTATES_CONST2 3
22 #define BITS_IN_BYTE 8
23 #define INTERNAL_OPERATION_LOOP 4
24 #define DIFFUSION_CONSTANT_SIZE 6
25 #define NUM_DIFFUSION_CONSTANT 10
26
27 // 16*16 S-box
```

```

28 static const uc sbox[256] = {
29     0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,
    0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
30     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4,
    0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
31     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5,
    0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
32     0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12,
    0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
33     0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b,
    0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
34     0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
    0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
35     0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9,
    0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
36     0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6,
    0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
37     0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
    0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
38     0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee,
    0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
39     0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3,
    0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
40     0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56,
    0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
41     0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd,
    0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
42     0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35,
    0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
43     0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,

```

```

    0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
44     0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99,
    0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
45 };
46
47 int DIFFUSION_CONSTANT[NUM_DIFFUSION_CONSTANT];
48
49 /*
50 Method to generate the Output from Extendable Output Function (
    XOF) - Using Shake128 algorithm
51 */
52 void generateXOFOutput(uc *nonce, uc *output) {
53     EVP_MD_CTX *mdctx;
54     const EVP_MD *md;
55     OpenSSL_add_all_algorithms();
56     // Using SHAKE128 XOF algorithm
57     md = EVP_shake128();
58
59     // Create and initialize context
60     if((mdctx = EVP_MD_CTX_new()) == NULL) {
61         printf("Error creating context\n");
62         return;
63     }
64     // Initialize digest operation
65     if(1 != EVP_DigestInit_ex(mdctx, md, NULL)) {
66         printf("Error initializing digest\n");
67         EVP_MD_CTX_free(mdctx);
68         return;
69     }
70     // Provide the nonce as input data

```

```

71     if(1 != EVP_DigestUpdate(mdctx, nonce, NONCE_SIZE)) {
72         printf("Error updating digest\n");
73         EVP_MD_CTX_free(mdctx);
74         return;
75     }
76     // Finalize the digest and obtain the output
77     if(1 != EVP_DigestFinalXOF(mdctx, output, 1)) {
78         printf("Error finalizing digest\n");
79         EVP_MD_CTX_free(mdctx);
80         return;
81     }
82     // Clean up
83     EVP_MD_CTX_free(mdctx);
84     EVP_cleanup();
85 }
86
87
88
89 /*
90 Method to generate the Diffusion Constants from Nonce - using
91 XOF output
92 */
93 void generateDiffusionConstant(uc *nonce) {
94     unsigned char xof_output[1]; // Output 1 byte at a time (8
95     bits)
96     int bit_index = 0;
97
98     for (int i = 0; i < NUM_DIFFUSION_CONSTANT; i++) {
99         DIFFUSION_CONSTANT[i] = 0;
100         for (int j = 0; j < DIFFUSION_CONSTANT_SIZE; j++) {

```

```

99         // Generate output from XOF based on the nonce
100         generateXOFOutput(nonce, xof_output);
101         // Use the first bit of the XOF output to set the
corresponding bit in the diffusion constant
102         DIFFUSION_CONSTANT[i] |= ((xof_output[0] & 1) <<
bit_index);
103         bit_index++;
104     }
105 }
106 }
107
108 int main() {
109     unsigned char nonce[NONCE_SIZE];
110     srand(time(NULL)); // Seed the random number generator with
current time
111     // Generate random nonce
112     for (int i = 0; i < NONCE_SIZE; i++) {
113         nonce[i] = rand() % 256;
114     }
115
116     generateDiffusionConstant(nonce);
117
118     // Print the diffusion constants
119     printf("Diffusion Constants:\n");
120     for (int i = 0; i < NUM_DIFFUSION_CONSTANT; i++) {
121         printf("Constant %d: %d\n", i + 1, DIFFUSION_CONSTANT[i
]);
122     }
123
124     return 0;

```



Listing A.1: Sample C Program

# Appendix B

## Code for Differential Fault Attack

```
1 import time
2 from sage.sat.boolean_polynomials import solve
3 from sage.crypto.sbox import SBox
4
5 ksize = 128
6 ivsize = 64
7 nsize = 128
8 v = ksize + ivsize + nsize
9 rounds = 4
10 nv = 2 * rounds * v + v
11 V = BooleanPolynomialRing(nv, ['k%d' % (i) for i in range(ksize)
    ]) + ['iv%d' % (i) for i in range(ivsize)] + ['n%d' % (i)
    for i in range(nsize)] + ['p%d' % (i) for i in range(2 *
    rounds * v)])
12 V.inject_variables(verbose=0)
13 T = list(V.gens()) # list of all variables
14 key = [k0 + k0] * ksize
15 fkey = [k0 + k0] * ksize
16 iv = [iv0 + iv0] * ivsize
```

```

17 nonce = [n0 + n0] * nsize
18
19 def random():
20     x = [ZZ.random_element(1, 64) for _ in range(20)]
21     return tuple(x)
22
23 # linear layer function
24 def llayer(state, *x):
25     y = [[k0 + k0] * 64 for _ in range(5)]
26     sy = [[k0 + k0] * 64 for _ in range(5)]
27     for i in range(64):
28         y[0][i] = state[i]
29         y[1][i] = state[i + 64]
30         y[2][i] = state[i + 128]
31         y[3][i] = state[i + 192]
32         y[4][i] = state[i + 256]
33
34     t = [None] * 5
35     for j in range(5):
36         t[j] = y[j][64 - x[j]:] + y[j][:64 - x[j]]
37     for i in range(64):
38         sy[0][i] = sum(t[j][i] for j in range(5))
39
40     for j in range(5):
41         t[j] = y[j][64 - x[j + 4]:] + y[j][:64 - x[j + 4]] if j
42         != 1 else y[j]
43     for i in range(64):
44         sy[1][i] = sum(t[j][i] for j in range(5))
45
46     for j in range(5):

```

```

46     t[j] = y[j][64 - x[j + 8]:] + y[j][:64 - x[j + 8]] if j
    != 2 else y[j]
47     for i in range(64):
48         sy[2][i] = sum(t[j][i] for j in range(5))
49
50     for j in range(5):
51         t[j] = y[j][64 - x[j + 12]:] + y[j][:64 - x[j + 12]] if
    j != 3 else y[j]
52     for i in range(64):
53         sy[3][i] = sum(t[j][i] for j in range(5))
54
55     for j in range(5):
56         t[j] = y[j][64 - x[j + 16]:] + y[j][:64 - x[j + 16]] if
    j != 4 else y[j]
57     for i in range(64):
58         sy[4][i] = sum(t[j][i] for j in range(5))
59
60     return sy[0] + sy[1] + sy[2] + sy[3] + sy[4]
61
62 S = SBox([0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30,
    0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
63     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad,
    0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
64     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34,
    0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
65     0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07,
    0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
66     0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52,
    0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
67     0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a,

```

```

    0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
68     0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45,
    0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
69     0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc,
    0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
70     0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4,
    0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
71     0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46,
    0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
72     0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2,
    0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
73     0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c,
    0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
74     0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8,
    0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
75     0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61,
    0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
76     0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b,
    0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
77     0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41,
    0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16])

78
79 def slayer(state):
80     state = V(S(state[0:8]) + S(state[8:16]) + S(state[16:24])
    + S(state[24:32]) + S(state[32:40]) + S(state[40:48]) + S(
    state[48:56]) + S(state[56:64]) + state[64:])
81     return state
82
83 state = nonce + iv + key
84 start = time.time()

```

```
85 for i in range(rounds):
86     state = slayer(layer(state, *random()))
87
88 print(f'Time taken: {time.time() - start} seconds')
```

# SunnySamuel\_CrsS221\_FinalReport

---

## ORIGINALITY REPORT

---

9%

SIMILARITY INDEX

7%

INTERNET SOURCES

6%

PUBLICATIONS

5%

STUDENT PAPERS

---

## PRIMARY SOURCES

---

1	<a href="http://hdl.handle.net">hdl.handle.net</a> Internet Source	3%
2	<a href="http://library.isical.ac.in:8080">library.isical.ac.in:8080</a> Internet Source	1%
3	<a href="http://academic.iiti.ac.in">academic.iiti.ac.in</a> Internet Source	1%
4	Kazuo Sakiyama, Yu Sasaki, Yang Li. "Security of Block Ciphers", Wiley, 2015 Publication	<1%
5	<a href="http://www.sannet.net">www.sannet.net</a> Internet Source	<1%
6	<a href="http://wikimili.com">wikimili.com</a> Internet Source	<1%
7	Submitted to University of Pretoria Student Paper	<1%
8	Joan Daemen, Vincent Rijmen. "The Design of Rijndael", Springer Science and Business Media LLC, 2020 Publication	<1%

---

9

Yefa Hu, Omer W. Taha, Kezhen Yang. "Fault Detection in Active Magnetic Bearings Using Digital Twin Technology", Applied Sciences, 2024

Publication

<1 %

10

Submitted to Nottingham Trent University

Student Paper

<1 %

11

dias.library.tuc.gr

Internet Source

<1 %

12

Submitted to Babes-Bolyai University

Student Paper

<1 %

13

dspace.mit.edu

Internet Source

<1 %

14

slideplayer.com

Internet Source

<1 %

15

Submitted to Southern Illinois University

Student Paper

<1 %

16

www.cypherpunk.at

Internet Source

<1 %

17

B.W. Boehm, R. Ross. "Theory-W software project management principles and examples", IEEE Transactions on Software Engineering, 1989

Publication

<1 %



18	Christof Paar, Jan Pelzl, Tim Güneysu. "Understanding Cryptography", Springer Science and Business Media LLC, 2024 Publication	<1 %
19	ir.uitm.edu.my Internet Source	<1 %
20	d-scholarship.pitt.edu Internet Source	<1 %
21	Pulkit Singh, Bibhudendra Acharya, Rahul Kumar Chaurasiya. "Lightweight cryptographic algorithms for resource- constrained IoT devices and sensor networks", Elsevier BV, 2021 Publication	<1 %
22	export.arxiv.org Internet Source	<1 %
23	webpages.eng.wayne.edu Internet Source	<1 %
24	www.mdpi.com Internet Source	<1 %
25	"Cryptography", Wiley, 2024 Publication	<1 %
26	edipermadi.files.wordpress.com Internet Source	<1 %
27	simple.wikipedia.org Internet Source	<1 %

<1 %

---

28

## Understanding Cryptography, 2010.

Publication

<1 %

---

---

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off