

ASCON INSIGHT:EXPLORING THE DEPTHS OF LIGHTWEIGHT ENCRYPTION ALGORITHMS.

*Dissertation submitted in the Partial fulfillment of the requirements for the award
of the degree of*

BACHELOR OF TECHNOLOGY

By

M.MOHAN DURGA PRADEEP - 20VV1A1232

M.SAI SITA MAHALAXMI - 20VV1A1234

R.BINDU HARSHITA - 20VV1A1254

D.SAI AJITH KUMAR - 21VV5A1268

Under the esteemed Guidance of

Mr.W.ANIL M.Tech(PhD)

Assistant Professor

Department of Information Technology



DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUGV College of Engineering Vizianagaram (Autonomous)

Jawaharlal Nehru Technological University Gurajada Vizianagaram

Dwarapudi, Vizianagaram-535003, Andhra Pradesh, India

April 2024

**DEPARTMENT OF INFORMATION TECHNOLOGY
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
GURAJADA VIZIANAGARAM**



CERTIFICATE

This is to certify that the dissertation entitled "**Ascon Insight:Exploring the Depths of Lightweight Encryption Algorithms.**" submitted by **M.MOHAN DURGA PRADEEP(20VV1A1232)**, **M.SAI SITA MAHALAXMI (20VV1A1234)**, **R.BINDU HARSHITA(20VV1A1254)**, **D.SAI AJITH KUMAR(21VV5A1268)** in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** from Jawaharlal Nehru Technological University Gurajada Vizianagaram-College of Engineering Vizianagaram is a record of Bonafide work carried out by them under my guidance and supervision during the year April 2024. The results embedded in this dissertation have not been submitted by any other university or institution for the award of any degree.

Signature of the Supervisor

Mr.W.ANIL

Assistant Professor

Dept. of Information Technology

JNTUGV-CEV.

Signature of the Head of the Department

Dr. B. TIRIMULA RAO

Assistant Professor & HOD

Dept. of Information Technology

JNTUGV-CEV.

External Signature

Declaration

We, **M.MOHAN DURGA PRADEEP(20VV1A1232), M.SAI SITA MAHALAXMI (20VV1A1234),R.BINDU HARSHITA(20VV1A1254), D.SAI AJITH KUMAR(21VV5A1268)** declared that the dissertation report entitled "Ascon Insight:Exploring the Depths of Lightweight Encryption Algorithms." is no more than 70,040 words in length including quotes and exclusive of tables, figures, bibliography, and references. This dissertation contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated this dissertation is in our own work

Roll No	Name	Signature
20VV1A1232	M.MOHAN DURGA PRADEEP	_____
20VV1A1234	M.SAI SITA MAHALAXMI	_____
20VV1A1254	R.BINDU HARSHITA	_____
21VV5A1268	D.SAI AJITH KUMAR	_____



JNTUGV COLLEGE OF ENGINEERING,VIZIANAGARAM(AUTONOMOUS)
JAWAHARLAL NEHRU TECHNOLOGY UNIVERSITY GURUJADA,VIZIANAGARAM
DWARAPUDI (POST),VIZIANAGARAM,ANDHRA PRADESH- 535003
 A constitute college of JNTUGV & approved by AICTE,new Delhi)(Recognized by UGC under section 2(f) &12(B) of UGC Act 1956)

Subject Name:Major Project
Academic Year:2024

Course Code:R204212PR01
Regulation:R20

CO'S
Course Outcomes

Course Outcomes	
CO1	Formulate solutions to computing problems using latest technologies and tools.
CO2	Work effectively in teams to design and implement solutions to computational problems and socially relevant issues
CO3	Recognize the social and ethical responsibilities of a professional working in the discipline
CO4	Apply advanced algorithmic and mathematical concepts to the design and analysis of software
CO5	Devise a communication strategy(language,content and medium) to deliver messages according to the situation and need of audience
CO6	Deliver effective presentations,extemporaneous or impromptu oral presentations,setting up technical reports using technical tools

CO-PO Mapping
Mapping of Course Outcomes (COs) with Program Outcomes (POs)

	Program Outcomes (POs)														
	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	3	2	-	2	3	-	3	-	-	-	-	-	2	-
CO2	2	3	2	-	3	-	-	3	3	-	-	2	2	2	2
CO3	-	-	-	-	-	2	-	3	-	-	-	2	-	-	-
CO4	3	-	-	-	2	-	-	-	2	-	-	3	3	2	-
CO5	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-
CO6	-	-	-	-	2	-	-	-	-	-	3	2	-	-	-

Enter correlation levels 1,2 and 3 as defined below:
 1:Slight(Low) 2:Moderate(Medium) 3:Substantial(High) If there is no correlation, put "0"

Signature of the Students

Signature of the Guide

Signature of the HOD

Acknowledgements

This report dissertation could not have been written without the support of our guide **Mr. W. Anil, Assistant Professor, Information Technology Department** who not only served as our superior but also encouraged and challenged us throughout our academic program. Our foremost thanks go to him. Without him, this dissertation would not have been possible. We appreciate his vast knowledge in many areas and his insights, suggestions, and guidance that helped to shape our research skills.

We are thankful to our project coordinator **Dr. Ch. Bindu Madhuri**, Assistant Professor, Department of Information Technology, for her continuous support.

We are thankful to the Project Review Committee Members **Dr. B. TIRIMULA RAO**, Assistant Professor, Head of the Department Of Information Technology, **Mr W. Anil** Assistant Professor, Information Technology Department, **Dr. Ch. Bindu Madhuri**, Assistant Professor, Information Technology Department for their valuable support

We express our sincere thanks to our respected **Dr. B. TIRIMULA RAO, Assistant Professor Head of the Department Of Information Technology** of JNTUGV College of Engineering Vizianagaram for bet valuable suggestion and constant motivation that greatly helped us in successful completion of project We also take the privilege to express our heartfelt gratitude to **Prof. K. SRI KUMAR, Principal**, JNTUGV University College of Engineering Vizianagaram.

We are thankful to all faculty members for extending their kind cooperation and assistance. Finally, we are extremely thankful to our parents and friends for their constant moral support.

20VV1A1232

M.MOHAN DURGA PRADEEP

20VV1A1234

M.SAI SITA MAHALAXMI

20VV1A1254

R.BINDU HARSHITA

21VV5A1268

D.SAI AJITH KUMAR

ABSTRACT

Lightweight encryption algorithms play a pivotal role in securing data across a myriad of applications, particularly in resource-constrained environments such as IoT devices, embedded systems. Among these algorithms, ASCON has emerged as a notable contender due to its blend of efficiency, security, and versatility. This paper delves into the depths of ASCON, providing an insightful exploration of its design principles, cryptographic features, and practical applications. We dissect ASCON's underlying mechanisms, including its sponge construction, elucidating the strengths and trade-offs associated with each. Furthermore, we examine ASCON's performance characteristics, highlighting its suitability for both hardware and software implementations.

In the realm of hardware implementations, ASCON demonstrates commendable execution time and throughput metrics. Through benchmarking and analysis, we calculate the execution time of ASCON on hardware platform, showcasing its efficiency in real-world scenarios. Additionally, we evaluate ASCON's throughput, revealing its capacity to process data swiftly while maintaining robust security guarantees. Furthermore, we investigate ASCON's avalanche effect, quantifying its ability to propagate changes in input data throughout the output, thus ensuring cryptographic strength.

Similarly, in software implementations, ASCON exhibits impressive execution time and throughput figures. By profiling ASCON on different software environments, we uncover its computational efficiency and suitability for applications requiring lightweight encryption. Additionally, we assess ASCON's avalanche effect in software, providing insights into its cryptographic properties and resistance against attacks.

KEY WORDS: ASCON, IOT DEVICES, CRYPTOGRAPHY, THROUGHPUT, EXECUTION TIME, AVALANCHE EFFECT

Contents

Acknowledgements	iv
Abstract	v
List of Figures	ix
List of Tables	x
1 INTRODUCTION	1
1.1 Cryptography	2
1.1.1 Introduction to Cryptography	2
1.1.2 Application of Cryptography	3
1.1.3 Introduction to Light Weight Security:	4
1.2 IOT Devices	5
1.2.1 Introduction to IOT Devices	5
1.2.2 Intigration to Cryptography algorithms	6
1.3 Objectives	7
2 LITERATURE SURVEY	8
2.1 LITERATURE SURVEY	9
3 REQUIREMENTS	11
3.1 REQUIREMENTS	12

3.1.1	Software Requirements	12
3.1.2	Hardware Requirements	14
4	METHODOLOGY	16
4.1	METHODOLOGY	17
4.2	Setup and Hardware selection.	17
4.2.1	Objective:	17
4.2.2	Elaboration:	17
4.3	Algorithm Implementation:	18
4.3.1	Objective:	18
4.3.2	Elaboration:	18
4.4	Data Generation:	18
4.4.1	Objective:	18
4.4.2	Elaboration:	18
4.5	Flowchart:	19
5	ALGORITHM	21
5.1	ALGORITHM	22
5.2	ASCON	22
5.3	Varients of ASCON	24
5.4	Working Process of ASCON	26
6	IMPLEMENTATION	37
6.1	IMPLEMENTATION	38

6.1.1	Formula Of Metrics Used	38
7	CODING AND RESULT ANALYSIS	45
7.1	Results	46
7.1.1	Results in windows configuration	46
7.1.2	Results in Ardunio UNO Board	47
7.2	Result Analysis	49
7.2.1	Analysing the Result	49
7.3	Code	50
7.3.1	code of ASCON in windows configuration.	50
7.3.2	code of ASCON in Ardunio IDE	58
8	CONCLUSION AND FUTURE WORK	62
8.1	CONCLUSION	63
8.2	FUTURE WORK	64
	BIBLOGRAPHY	65

List of Figures

1.1	cryptography	2
1.2	Light Weight Security	4
3.3	Development Environment Of Dev C++	12
3.4	Development Environment Of Ardunio	14
3.5	Ardunio UNO Board	15
4.6	Flowchat	20
5.7	Hardware implementation	32
5.8	FlowChart	33
7.9	ASCON-128 in windows	46
7.10	ASCON-128a in windows	46
7.11	ASCON-80pq in windows	47
7.12	ASCON-128 in Ardunio	47
7.13	ASCON-128a in Ardunio	48
7.14	ASCON-80pq in Ardunio	48
7.15	Limited size of Arduino	50
7.16	Output Of Ascon in Windows	53
7.17	Output Of Ascon in Windows	57
7.18	Output Of Ascon in ardunio	61

List of Tables

5.1	Comparison of ASCON-128, ASCON-128a, and ASCON-80pq	25
5.2	Comparison between lightweight AES and ASCON	34
5.3	Comparison between Feistel variant and Non Feistel variant	36

Chapter 1: INTRODUCTION

1.1 Cryptography

1.1.1 Introduction to Cryptography

A crucial component of human communication and security for ages has been cryptography, a fascinating science. At its foundation, cryptography is the art and science of safeguarding information by encoding it in an incomprehensible format, and then, using a variety of methods and algorithms, decrypting it to restore it to its original condition ([12]). Cryptography is essential for preserving the secrecy, integrity, and authenticity of data in today's hyper connected world where data is sent over digital networks at an unparalleled rate.

Ancient civilizations employed cyphers and secret codes to shield private communications from prying eyes, which is where cryptography got its start ([2]). Because of the increasing necessity to protect information in the age of digital information, cryptography has developed through time from straightforward substitution cyphers to intricate mathematical algorithms. It is now a crucial instrument for protecting our private data, financial activities, and national security ([6]). It includes several cryptographic protocols, symmetric and asymmetric encryption, and digital signatures ([14]). Asymmetric in contrast to symmetric encryption, which uses a single secret key for both encryption and decryption, encryption relies on a pair of keys, one for encryption and the other for decryption. This dual key system improves security and makes it possible to communicate securely over untrusted networks.

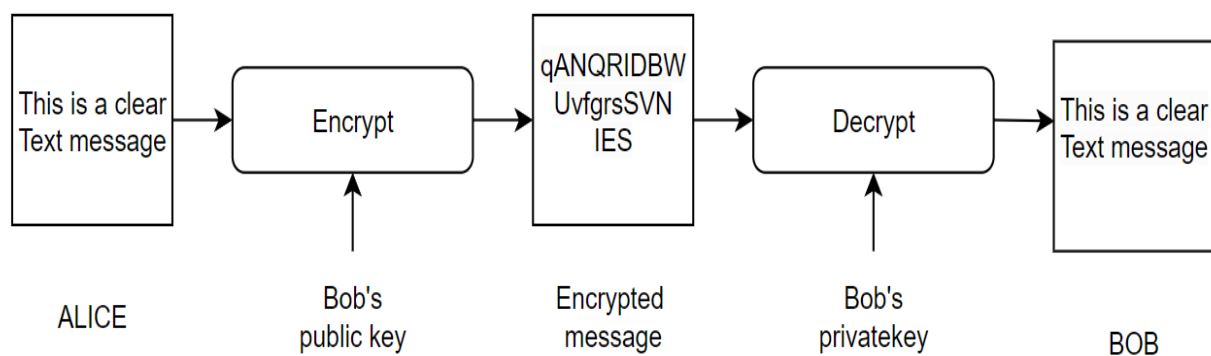


Figure 1.1: cryptography

1.1.2 Application of Cryptography

Data Encryption: Describe how cryptography is used to encrypt sensitive data, such as personal information, financial transactions, and corporate communications. Explain popular encryption algorithms like AES (Advanced Encryption Standard).

Secure Communication: Discuss how cryptography ensures secure communication over the internet, including protocols like SSL/TLS (Secure Sockets Layer/Transport Layer Security) used in web browsing, email encryption (e.g., PGP - Pretty Good Privacy), and secure messaging applications.

Authentication and Digital Signatures: Explain how cryptography is used for authentication and verification, such as digital signatures in electronic documents and certificates in secure websites (e.g., SSL certificates).

Identity Protection: Discuss the role of cryptography in identity protection, including password hashing techniques, biometric encryption, and multi-factor authentication systems.

Data Integrity: Explain how cryptographic hash functions are used to verify the integrity of data, such as file integrity checks and message authentication codes (MACs).

Cybersecurity: Highlight the importance of cryptography in cybersecurity strategies, including intrusion detection systems, network security protocols, and secure software development practices.

1.1.3 Introduction to Light Weight Security:

A form of encryption known as lightweight cryptography has a tiny computational complexity or footprint.

Its international standardization and guidelines compilation are now under way, and it aims to increase the applicability of cryptography on restricted devices ([3]). In simple words designed to consume minimal resources Such as

- Faster
- Power Consumption
- Memory
- Processing Capability

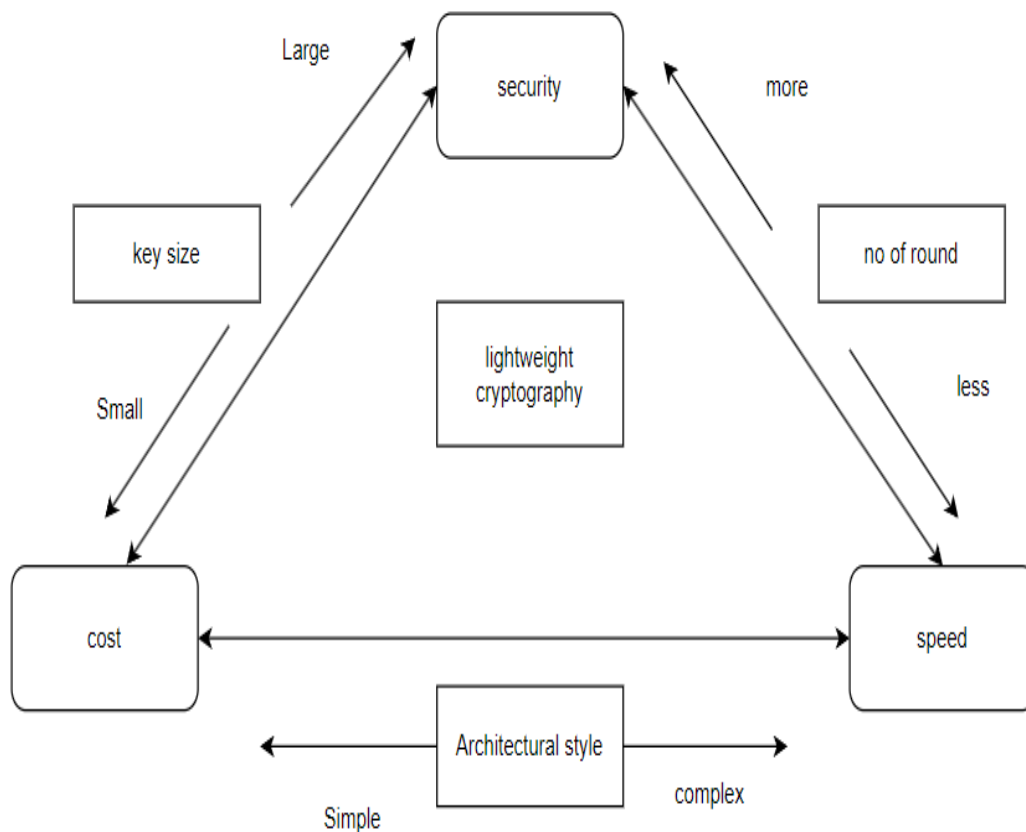


Figure 1.2: Light Weight Security

1.2 IOT Devices

1.2.1 Introduction to IOT Devices

In the contemporary era, the Internet of Things (IoT) stands as a revolutionary concept reshaping the fabric of our technological landscape. At its core, IoT represents a network of interconnected devices, equipped with sensors, actuators, and communication technologies, all orchestrated to collect, transmit, and exchange data. These devices span a vast spectrum, ranging from everyday objects like household appliances and wearable gadgets to industrial machinery and smart infrastructure components. The essence of IoT lies in its ability to bridge the physical and digital realms, enabling seamless interactions between humans, devices, and systems. From the convenience of smart homes, where IoT devices govern lighting, heating, and security systems based on user preferences and environmental conditions, to the intricacies of industrial automation, where IoT-enabled sensors monitor equipment health, optimize production processes, and predict maintenance needs, the applications of IoT are diverse and far-reaching.

Moreover, the transformative potential of IoT extends beyond mere convenience and efficiency gains; it holds the promise of revolutionizing entire industries and redefining business models. In the realm of healthcare, IoT devices facilitate remote patient monitoring, medication adherence tracking, and personalized health interventions, ushering in an era of preventative and proactive healthcare delivery. In agriculture, IoT sensors deployed in fields and greenhouses monitor soil moisture levels, weather conditions, and crop health, empowering farmers to make data-driven decisions, optimize resource usage, and increase crop yields sustainably. Similarly, in transportation and logistics, IoT-enabled solutions streamline supply chain operations, enhance fleet management, and improve last-mile delivery efficiency, driving down costs and improving service reliability.

1.2.2 Intigration to Cryptography algorithms

IoT devices play a crucial role in enhancing the performance of cryptography by providing a foundation for secure communication, data protection, and authentication in interconnected environments. At the core of this synergy lies the seamless integration of IoT devices with cryptographic algorithms and protocols, enabling robust security measures to be implemented at the edge of the network.

IoT devices enable the implementation of cryptographic mechanisms for data protection and integrity verification. By employing techniques such as digital signatures and hash functions, IoT devices can authenticate data sources, verify the integrity of received data, and detect any unauthorized modifications or tampering attempts. This ensures the trustworthiness and reliability of data exchanged within IoT ecosystems, bolstering confidence in the integrity of critical operations and decision-making processes.

Moreover, IoT devices contribute to the scalability and efficiency of cryptographic operations by offloading computational tasks to distributed edge devices. By distributing cryptographic processing across a network of interconnected devices, computational overheads can be minimized, and performance bottlenecks can be alleviated. This decentralized approach to cryptography not only enhances the scalability of IoT deployments but also improves response times and resource utilization, especially in latency-sensitive applications.

In conclusion, IoT devices play a pivotal role in optimizing the performance of cryptography within interconnected environments, enabling secure communication, data protection, and authentication mechanisms to be seamlessly integrated at the edge of the network. By leveraging cryptographic algorithms and protocols, IoT devices ensure the confidentiality, integrity, and authenticity of data exchanged within IoT ecosystems, thereby bolstering the overall security posture of interconnected systems and applications.

1.3 Objectives

The fundamental goal of our study is to undertake a rigorous and systematic evaluation of the performance of cryptographic algorithms when referring to Internet of Things (IoT) gadgets Smart homes and wearables, as well as industrial sensors and driverless cars, have become vital elements of our everyday lives as the IoT environment continues to develop fast. The limited resources of IoT devices, on the other hand, provide significant obstacles for creating safe cryptographic solutions. Our objective is to address these issues and give useful information to help guide the selection and optimization of cryptographic algorithms in IoT applications.

Performance Assessment: Performance assessment of cryptographic algorithms ASCON depends on various factors including key size,Nonce Size,Rate,time,Throughput,no.of.rounds, requirements.ASCON is a lightweight encryption algorithm,it uses Substitution box for secure communication over an insecure channel.S-Box provides Non-linearity and Diffusion properties which makes cryptographic attacks more complex.

Platform Versatility: To confirm the applicability and relevance of our findings, we run our tests on two separate platforms. Windows and Arduino UNO. This dual-platform technique enables us to evaluate algorithm behavior in a variety of computer settings. Windows offers a more resource-rich environment, whereas Arduino UNO mimics the settings of IoT devices, which are recognized for their low processing capabilities. Our goal is to deliver insights that are applicable to both traditional computer systems and IoT situations.

Security and Efficiency Balance: We analyze the security elements of ASCON algorithm using the Avalanche Effect(slight change in input produces a vastly different output, enhancing security by making it difficult to predict without knowledge of the input).we observed in every computation the avalanche effect is more than 45 percentage.Which makes our code more complex to be attacked.

Chapter 2: LITERATURE SURVEY

2.1 LITERATURE SURVEY

In recent years, the integration of cryptographic algorithms with Internet of Things (IoT) devices has gained significant attention due to the growing concern for securing data transmission and communication in IoT ecosystems. Among these cryptographic algorithms, ASCON (Army Static Switched Communication Network) has emerged as a promising candidate for ensuring secure and efficient communication in military and civilian applications alike. This literature survey aims to explore the existing research on integrating the ASCON algorithm with IoT devices, focusing on its implications, challenges, and potential applications.

A tailored framework designed for benchmarking software implementations sourced from the National Institute of Standards and Technology (NIST) Lightweight Cryptography (LWC) project, particularly on embedded devices. We detail the framework's architecture and fundamental functionalities, applying it comprehensively to assess various NIST LWC authenticated encryption with associated data (AEAD) ciphers. Our evaluation encompasses the speed assessment of 213 algorithm variants submitted, executed across four distinct microcontroller units (MCUs), encompassing 32-bit ARM and 8-bit AVR architectures. Additionally, to facilitate a comprehensive comparison, we conduct code size evaluations across all four boards and RAM utilization assessments on a designated test platform.

[1] Christoph Dobraunig et.al This document outlines the Ascon cipher suite, featuring AEAD and hashing functionalities, including Ascon-128 and Ascon-128a, which are favored in the CAESAR competition's final selection for lightweight encryption. It introduces Ascon-80pq for enhanced quantum resistance, alongside hash functions Ascon-Hash and Ascon-Hasha, and output functions Ascon-Xof and Ascon-Xofa. Recommended for NIST, these schemes offer 128-bit security using a shared 320-bit permutation, enabling efficient implementation of both AEAD and hashing with a single lightweight primitive.

[2] Tran, S.-N et.al The rapid expansion of the Internet of Things (IoT) has integrated it into diverse aspects of daily life, necessitating high data throughput and minimal latency for real-time communication. However, heightened data transmission raises security concerns. To address this, we propose a hardware architecture for Ascon, facilitating high-throughput, low-latency security services in IPSec protocols. Our results demonstrate the ESP protocol's capability of

achieving 8.806 Gbps throughput and 427ns latency with minimal hardware usage, ideal for securing high-speed, low-latency IoT networks at IoT gateways.

[3] Aneesh Kandi et.al In this study, we detail multiple hardware implementations for ASCON, covering both encryption/tag generation and decryption/tag verification for ASCON AEAD, along with the ASCON hash function. We enhance standard implementations with side channel protection (threshold countermeasure) and triplication/majority-based fault protection, which operate independently, allowing either to be activated without affecting the other. Additionally, we provide ASIC and FPGA benchmarks for these implementations.

[4] Mickael Cazorla et al In this article, we explore the suitability of various block ciphers for wireless sensor networks (WSNs), focusing on energy efficiency and minimal memory requirements. We benchmark recent lightweight and traditional block ciphers on the TI 16-bit MSP430 microcontroller, previously testing 12 block ciphers on the ATMEL AVR ATtiny45. Our study includes a security and implementation overview of the selected ciphers and presents implementation results on our dedicated platform.

[5] sebastian1.renner et.al This paper introduces a tailored framework for evaluating NIST Lightweight Cryptography (LWC) project software implementations on embedded devices. We detail the framework's design and functions, applying it to several NIST LWC AEAD ciphers, and assess the performance of 213 algorithm variants across four microcontrollers, including 32-bit ARM and 8-bit AVR architectures. Additionally, we conduct comparative analyses by performing code size tests on all platforms and RAM utilization tests on one specific test platform.

Chapter 3: REQUIREMENTS

3.1 REQUIREMENTS

3.1.1 Software Requirements

Operating System (Windows 11): Windows is a popular platform for software creation, and Windows 11 includes several enhancements to the user experience and security measures. It emphasizes improved performance, enhanced security features, and better support for touchscreen and hybrid devices. Because many IoT devices connect with Windows-based computers, it's critical that your cryptographic methods function effectively on this platform.

Development Environment (Dev C++): Dev-C++ is a free integrated development environment (IDE) distributed under the GNU General Public License for programming in C and C++. It offers features like code completion, syntax highlighting, and a built-in debugger. Dev-C++ runs on Windows and has been widely used by beginners for its simplicity and ease of use.

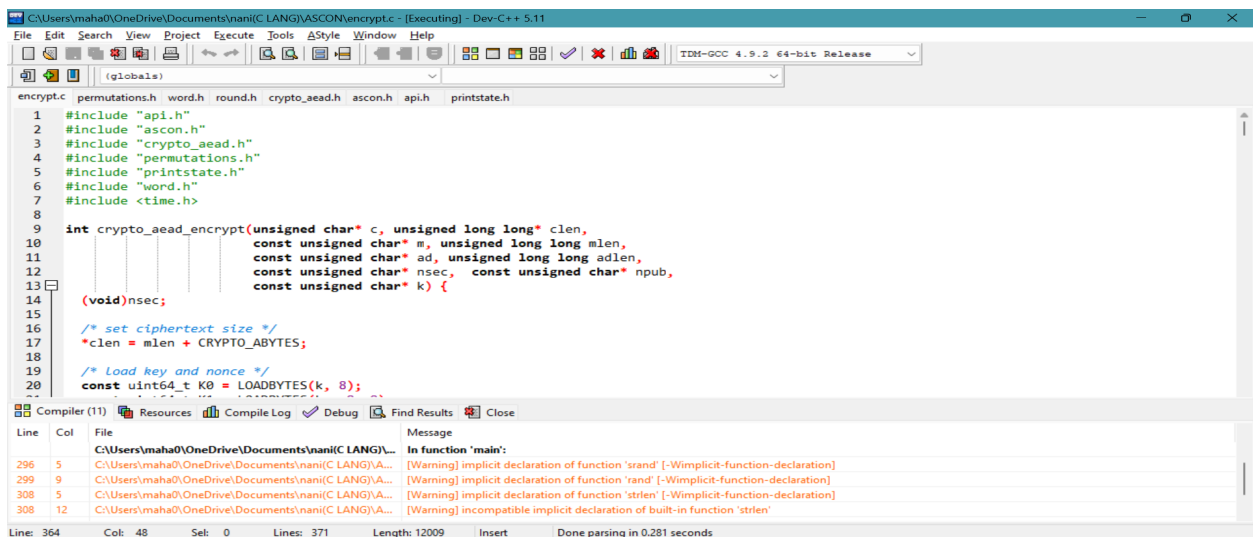


Figure 3.3: Development Environment Of Dev C++

C Language: C is a procedural programming language developed in the early 1970s by Dennis Ritchie at Bell Labs. Known for its efficiency and versatility, C is widely used in system programming, embedded systems, and software development. It offers low-level access to memory, making it suitable for programming close to the hardware, and it serves as the foundation for many other programming languages.

Resource Monitoring: It is recommended that you monitor resource consumption, such as

CPU and memory utilization, during your tests to understand how your cryptographic methods effect the performance of your system. This data might be useful for analyzing the outcomes.

3.1.2 Hardware Requirements

System Configuration: The hardware requirements of my laptop, which include 8GB of RAM and a 512GB SSD, should be suitable for cryptographic algorithm testing. However, it is critical to keep these constraints in mind while performing resource-intensive operations, since they might have an influence on the accuracy of our performance metrics.

Development Environment (Arduino IDE) : The Arduino IDE (Integrated Development Environment) is a software platform used for programming Arduino microcontroller boards. It provides a simplified interface for writing, compiling, and uploading code to Arduino devices, making it accessible to beginners and experienced developers alike. The IDE supports the Arduino programming language based on C/C++, and it includes libraries and examples to facilitate rapid prototyping and development of electronic projects. Additionally, the Arduino IDE offers tools for monitoring serial communication and debugging code for efficient troubleshooting.

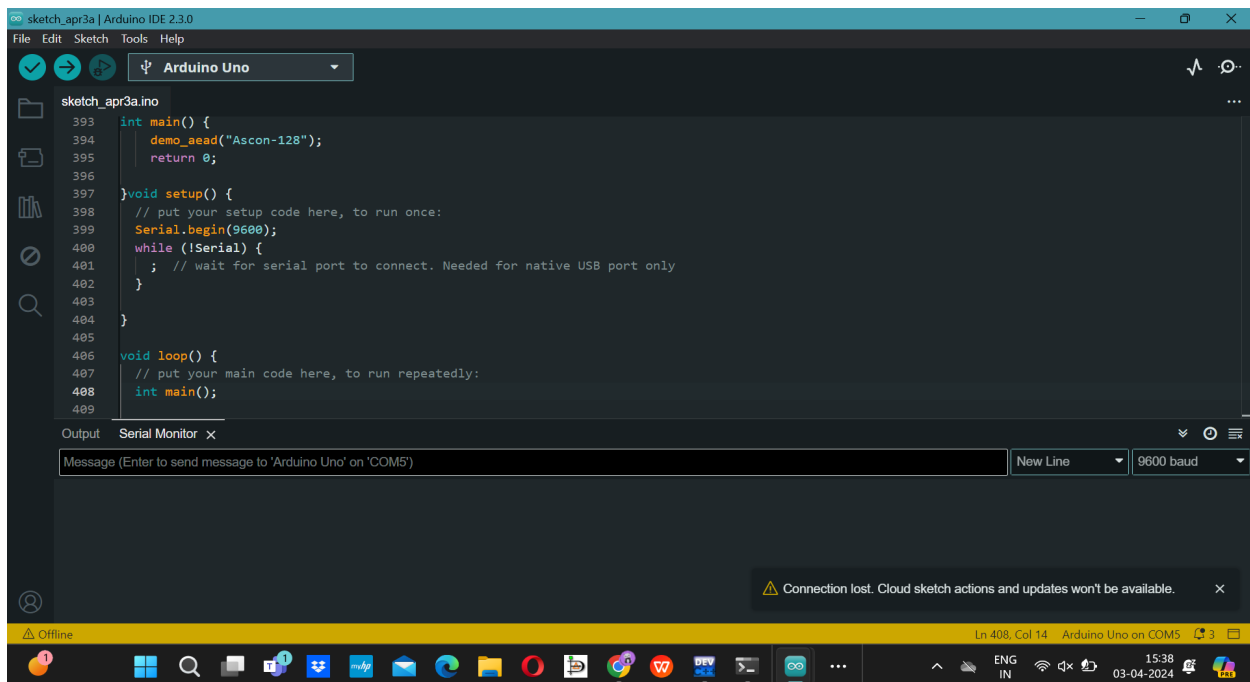


Figure 3.4: Development Environment Of Arduino

Arduino board UNO :The Arduino Uno is a popular microcontroller board featuring an Atmega328P chip, offering 14 digital input/output pins, 6 analog inputs, and a USB connection for programming and power. It's widely used for prototyping and DIY electronics projects due to its simplicity and versatility. The Uno is compatible with a wide range of sensors, actuators, and shields, making it suitable for beginners and advanced users alike. With its open-source nature and extensive community support, the Arduino Uno remains a cornerstone of the maker movement. Its ease of use and affordability make it an ideal choice for learning about electronics and programming.



Figure 3.5: Arduino UNO Board

Chapter 4: METHODOLOGY

4.1 METHODOLOGY

4.2 Setup and Hardware selection.

4.2.1 Objective:

The purpose of this first stage is to provide the groundwork for your project by configuring your research environment and selecting the proper hardware for your investigations. This is an important step in ensuring that your tests are carried out on a consistent and controlled platform.

4.2.2 Elaboration:

a) IoT Device Setup: Begin by selecting the IoT devices that will be used in your tests. Ascertain that they meet the project's standards and limits, such as processing power, memory, and communication capabilities

b) Software Setup: Using Arduino IDE set the connection between the Arduino UNO board and your computer. Make sure you selected correct port number and board name.

c) Hardware Setup: Set up the chosen IoT devices in a safe setting. To achieve consistent findings throughout your tests, verify that they are correctly designed and calibrated.

4.3 Algorithm Implementation:

4.3.1 Objective:

In this stage, we will put the cryptographic methods that we want to compare on our IoT devices into action. The objective is to design optimized implementations that take into consideration our devices' hardware constraints.

4.3.2 Elaboration:

a) Algorithm Selection: Choose the cryptographic algorithms you wish to learn (for example ASCON)
b) Implementation Optimization: Optimize each algorithm for resource-constrained contexts before implementing it on your IoT devices. This may include encryption time, decryption time, throughput, avalanche effect.

4.4 Data Generation:

4.4.1 Objective:

In your IoT application, generate test data that replicates real-world circumstances. This information will be used for encryption and decryption during your tests.

4.4.2 Elaboration:

a) Output: we have generated outputs like encryption, decryption time, throughput, avalanche effect, memory usage. These outputs will be helpful to know the efficiency of the algorithm on the IOT device.

4.5 Flowchart:

a) Start Project: The first step in every endeavor is to get it started. This entails the project team or stakeholders formally initiating project activity

b) Setup Hardware: Once the project is started, the next crucial step is to setup and set up the appropriate gear. This might include computers, servers, or any other equipment necessary for the project's execution.

c) Algorithm Implementation: The project enters the implementation phase now that the hardware is in place. Depending on the nature of the project, the project goals and requirements are turned into code or software at this stage.

d) Performance Metrics: It is critical to set performance indicators to ensure that the project stays on schedule and meets its objectives. These indicators serve as a baseline for analyzing progress and determining project success.

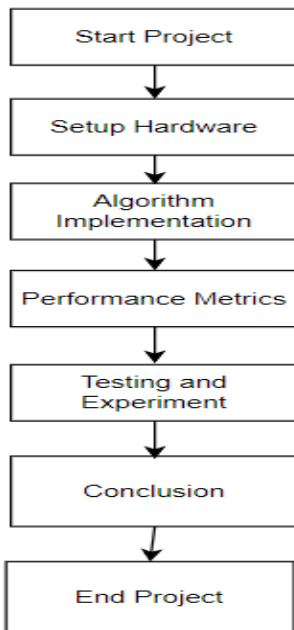


Figure 4.6: Flowchat

e) Testing and Experiment: Following the implementation of the algorithm, the project moves onto the testing and experimentation phase. The algorithm is thoroughly tested here to guarantee that it works and provides the intended results.

f) Conclusion: When the project's goals have been completed or the expected outcomes have been achieved, it's time to call it a day. This phase entails presenting the project's conclusions and findings, which is usually done through documentation or a final report.

g) End Project: The project's life cycle concludes with its formal closure. This include surrendering resources, finalizing paperwork, and formally terminating project efforts.

Chapter 5: ALGORITHM

5.1 ALGORITHM

5.2 ASCON

ASCON is a family of cryptographic algorithms designed primarily for lightweight applications, where resources such as power, memory, and processing capabilities are limited. It emerged as part of the CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) competition, aiming to establish secure and efficient authenticated encryption methods. ASCON is noteworthy for its simplicity, efficiency in hardware and software, and its robust security credentials.

The ASCON encryption scheme is built around a sponge construction, which is a flexible method for arbitrary-length input processing used for both hashing and encryption. This method allows ASCON to perform both encryption and authentication in a single cryptographic primitive, streamlining operations in constrained environments. The core of ASCON is a 320-bit state permutation function designed to be both lightweight and secure. The permutation exploits bitwise operations, rotations, and modular additions, balancing security with simplicity and low-resource usage.

ASCON operates in various modes, with the primary ones being ASCON-128 and ASCON-128a, indicating different security levels and performance trade-offs. These modes are differentiated by their initialization parameters and number of rounds during encryption, impacting their security margin and throughput.

The security of ASCON has been extensively analyzed within the scope of the CAESAR competition and beyond. The primary concerns addressed include resistance against differential and linear cryptanalysis, side-channel attacks, and forgery attacks. ASCON's design has been particularly noted for its resistance to differential cryptanalysis, leveraging a strong confusion-diffusion mechanism inherent in its permutation-based design.

Several academic papers and industry analyses have confirmed the robustness of ASCON against known cryptographic attacks, highlighting its suitability for environments susceptible to physical security threats, such as IoT devices and embedded systems. The simplicity of the algorithm also

reduces the surface area for potential vulnerabilities, a critical factor in maintaining security in lightweight applications.

ASCON's performance has been evaluated in terms of both hardware and software efficiency. In hardware, ASCON demonstrates excellent performance on FPGAs and ASICs, characterized by low gate count and minimal power consumption, making it suitable for RFID tags and smart cards. In software, while ASCON is not the fastest among CAESAR candidates, its performance is competitive, particularly in environments where the resource constraints justify a slight trade-off in speed for enhanced security and lower power usage.

Studies comparing ASCON with other lightweight cryptographic algorithms like SPECK, SIMON, and AES indicate that while ASCON may not always be the leader in raw performance metrics, it offers a balanced profile of security, efficiency, and implementation simplicity, essential for its target applications.

The application areas for ASCON are vast, particularly in sectors where security needs to be embedded in hardware with stringent resource constraints. These include automotive systems, IoT devices, industrial control systems, and small-scale embedded devices, where the balance of security, power, and performance is critical.

The future research directions might focus on further optimizing ASCON's implementation in ultra-constrained devices and exploring its integration with emerging technologies like quantum-resistant cryptographic systems and multi-party computation environments.

ASCON represents a significant step forward in the design of lightweight cryptographic solutions, providing a high level of security and efficiency tailored for environments where both are traditionally difficult to achieve. Its adoption and continued validation in academic and practical fields underscore its relevance and potential as a cornerstone for future secure systems in the lightweight cryptography domain.

5.3 Variants of ASCON

a) ASCON-128

ASCON-128 is a lightweight cryptographic algorithm designed for secure communication in resource-constrained environments. It provides strong encryption and authentication with minimal computational overhead. ASCON-128 operates on 128-bit blocks and supports key sizes ranging from 128 to 320 bits. Its efficient design makes it suitable for applications in IoT devices, embedded systems, and constrained environments where performance and security are paramount. With its simplicity and robustness, ASCON-128 offers a practical solution for ensuring data confidentiality and integrity in various real-world scenarios.

b) ASCON-128(a)

ASCON-128(a) is a lightweight cryptographic algorithm designed for secure communication in resource-constrained environments. It offers a 128-bit security level, ensuring robust protection against various cryptographic attacks. ASCON-128(a) features a compact implementation with low computational overhead, making it suitable for embedded systems and IoT devices. With its permutation-based design and simple round function, ASCON-128(a) provides efficient encryption and authentication capabilities. Its versatility and resistance to side-channel attacks make it a promising solution for securing data transmission in diverse applications.

c) ASCON-80pq

ASCON-80pq is a variant of the ASCON cryptographic algorithm optimized for lightweight and efficient performance. It operates on 80-bit plaintext and ciphertext blocks, with a variable key size of 80 to 160 bits. ASCON-80pq is designed for applications requiring strong security in resource-constrained environments, such as IoT devices and embedded systems. Its lightweight design and low computational overhead make it suitable for scenarios where memory and processing power are limited. ASCON-80pq provides robust encryption and authentication capabilities, ensuring data confidentiality and integrity in constrained environments.

Table 5.1: Comparison of ASCON-128, ASCON-128a, and ASCON-80pq

Feature	ASCON-128	ASCON-128a	ASCON-80pq
Block Size	128 bits	128 bits	80 bits
Key Size	128 to 320 bits	128 to 320 bits	80 to 160 bits
Variant	Original	Improved	Lightweight
Optimization	Balanced between security and speed	Enhanced security	Optimized for lightweight
Use Cases	General-purpose cryptography	Applications requiring extra security	Resource-constrained environments
Performance	Moderate	Improved	Efficient
Implementation	Standard implementation available	Enhanced implementation available	Lightweight implementation available
Suitable for	Various applications	Applications with higher security requirements	IoT devices, embedded systems

5.4 Working Process of ASCON

Ascon is a cipher suite that provides authenticated encryption with associated data (AEAD). The Ascon cipher suite utilizes the sponge design methodology, which shares some construction similarities with the SHA-3 contest winner Keccak [26]. The design rationale behind Ascon is to provide the best trade-off between security, size and speed in both software and hardware.

The Ascon cipher suite has two versions, Ascon-128 and Ascon-128a [27]. Ascon-128a processes data in 8 rounds, as opposed to 6 rounds in Ascon-128, but it also has a larger block size of 128 bits, double that of Ascon-128's 64 bits. So Ascon-128a cipher suite has a higher throughput than the Ascon-128 version. Since we are targeting high-speed IoT networks, the Ascon-128a algorithm is more suitable. Note that the Ascon-128a is secondary recommendation after Ascon-128 and it is currently uncertain whether finalized standard will include both of them. However, since the two algorithms are not significantly different, transitioning from one to the other should not pose any major challenges.

Ascon has a state of 320 bits and two permutations p^a and p^b . The 320-bit state S is divided into an outer part S^r of r bits and an inner part S^c of c bits, where the rate r and capacity $c = 320 - r$ depend on the Ascon variant. The 320-bit state S is split into five 64-bit registers words:

$$S = S^r \parallel S^c = x0 \parallel x1 \parallel x2 \parallel x3 \parallel x4$$

Ascon authenticated encryption or decryption consists of four phases – Initialization, Associated Data (AD), Processing Plaintext/Ciphertext, and Finalization.

1) INITIALIZATION

The 320-bit initial state of Ascon is constructed from secret key K, nonce N, and an initialization vector (IV) that is predefined by the algorithm.

$$IV_{k,r,a,b} \leftarrow k \parallel r \parallel a \parallel b \parallel 0^{(160-k)} = \{80400c060000000000 \\ \{80800c080000000000$$

$$S \leftarrow IV_{k,r,a,b} \parallel K \parallel N$$

The initial state goes through a round of the round transformation and then is XORed with the secret key K.

$$S \leftarrow p^a(S) \oplus (0^{(320-k)} \parallel K)$$

2) ASSOCIATED DATA

Ascon appends a single 1 and the smallest number of 0s to associated data A to obtain a multiple of r bits and split it into s blocks of r bits, A_1, \dots, A_s . In case A is empty, no padding is applied and $s = 0$.

Each block A_i with $i = 1, \dots, s$ is xored to the first r bits S_r of the state S, followed by b round of the round transformation:

$$S \leftarrow p^b((S^r \oplus A^i) \text{---} S^c), 1 \leq i \leq s$$

A 1-bit domain separation constant is xored to state S after processing A_s

$$S \leftarrow S \oplus (0^{(319)} \parallel 1)$$

3) PROCESSING PLAINTEXT/CIPHERTEXT

The same padding rule with single 1 and the smallest number of 0s is applied to the plaintext P. The ciphertext is generated by XORing plaintext with the state S, followed by b round of the round transformation:

$$\begin{aligned} C_i &\leftarrow S_r \oplus P_i \\ S &\leftarrow p^b (C_i \parallel S_c) \text{ if } 1 \leq i < t \\ &C_i \parallel S_c \text{ if } 1 \leq i = t \end{aligned}$$

The last ciphertext block is truncated to the length of the unpadded last plaintext:

$$C^r \leftarrow C_t |P| \bmod r$$

The decryption process is similar:

$$\begin{aligned} P_i &\leftarrow S_r \oplus C_i \\ S &\leftarrow p^p(C_i \parallel S_c), 1 \leq i < t \\ P_t &\leftarrow S_r \oplus C_t^r \\ S &\leftarrow (S_r \oplus (P_t \parallel 1 \parallel 0^{(r-1-)})) \parallel S_c \end{aligned}$$

4) FINALIZATION

In the finalization, the secret key K is xored to the internal state and the state is transformed by the permutation p a using a rounds. The tag T consists of the last (least significant) 128 bits of the state XORed with the last 128 bits of the key K :

$$\begin{aligned} S &\leftarrow p^a(S \oplus (0^r \parallel K \parallel 0^{(c-k)})) \\ T &\leftarrow S^{(128)} \oplus K^{(128)}. \end{aligned}$$

The encryption algorithm returns T together with the ciphertext. The decryption algorithm returns the plaintext only if the calculated tag value matches the received tag value.

C. PERMUTATIONS OF ASCON

The permutations of Ascon, known as the round transformation p , are based on the SPN structure and consist of three steps p_C, p_S, p_L . These steps include the addition of constants, substitution layer, and linear diffusion layer. The permutations p^a and p^b are applied to 320-bit state S and differ only in the number of rounds.

The constant addition step p_C adds a round constant c_r to register word x_2 of the state S in round i .

$$x_2 \leftarrow x_2 \oplus c_r$$

The S-box can also be implemented efficiently using bit sliced technique[27]:

$$\begin{aligned} x_0 &= x_0 \oplus x_4; x_4 = x_4 \oplus t_1; x_2 = x_2 \oplus t_1; \\ t_0 &= x_0 \oplus (x_4); t_1 = x_2 \oplus (x_1); \\ x_0 &= x_0 \oplus t_1; t_1 = x_4 \oplus (x_3); \\ x_2 &= x_2 \oplus x_1; t_1 = x_1 \oplus (x_0); \\ x_4 &= x_4 \oplus x_3; t_1 = x_3 \oplus (x_2); \\ x_1 &= x_1 \oplus t_1; x_3 = x_3 \oplus x_4; \\ x_1 &= x_1 \oplus x_0; x_3 = x_3 \oplus x_2; x_0 = x_0 \oplus x_4; x_2 = x_2; \end{aligned}$$

The linear diffusion layer p_L provides diffusion within each 64-bit register word x_i by XOR and right-rotation (circular shift) operations:

$$\begin{aligned} x_0 &= x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\ x_1 &= x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\ x_2 &= x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6) \\ x_3 &= x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\ x_4 &= x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41) \end{aligned}$$

Invertible components in ASCON

In ASCON (Authenticated Sponge Construction), several components are invertible, meaning they can be reversed to recover the original input from the output. These invertible components play a crucial role in both the encryption and authentication processes. Here are some key invertible components in ASCON:

1.S-boxes: ASCON uses invertible S-boxes (substitution boxes) in its round function. These S-boxes map input bit sequences to output bit sequences in a reversible manner. In ASCON, these S-boxes contribute to the confusion aspect of the cryptographic transformation.

2.Linear Layer: The linear layer of ASCON is a reversible matrix multiplication operation. It is responsible for achieving diffusion in the state. The linear layer ensures that changes to one part of the state propagate throughout the entire state, increasing the security of the algorithm.

3.Permutation Layer: ASCON employs a permutation layer that rearranges the bits of the state in a reversible manner. This permutation ensures that the state undergoes a series of reversible transformations during each round, contributing to both diffusion and confusion.

4.XOR Operations: XOR (exclusive OR) operations are used extensively throughout ASCON. XOR operations are inherently reversible; given the result of an XOR operation and one of the operands, you can recover the other operand. XOR operations are used to combine different components of the state and round constants.

5.Keccak-f Permutation (in non-Feistel variant): In the non-Feistel variant of ASCON, the Keccak-f permutation is used as part of the underlying sponge construction. The Keccak-f permutation is reversible and contributes to both diffusion and confusion within the state.

These invertible components ensure that both the encryption and authentication processes in ASCON can be performed efficiently and securely while maintaining reversibility where necessary. This reversibility is essential for decryption and verification processes, where the original input needs to be recovered from the ciphertext or authentication tag.

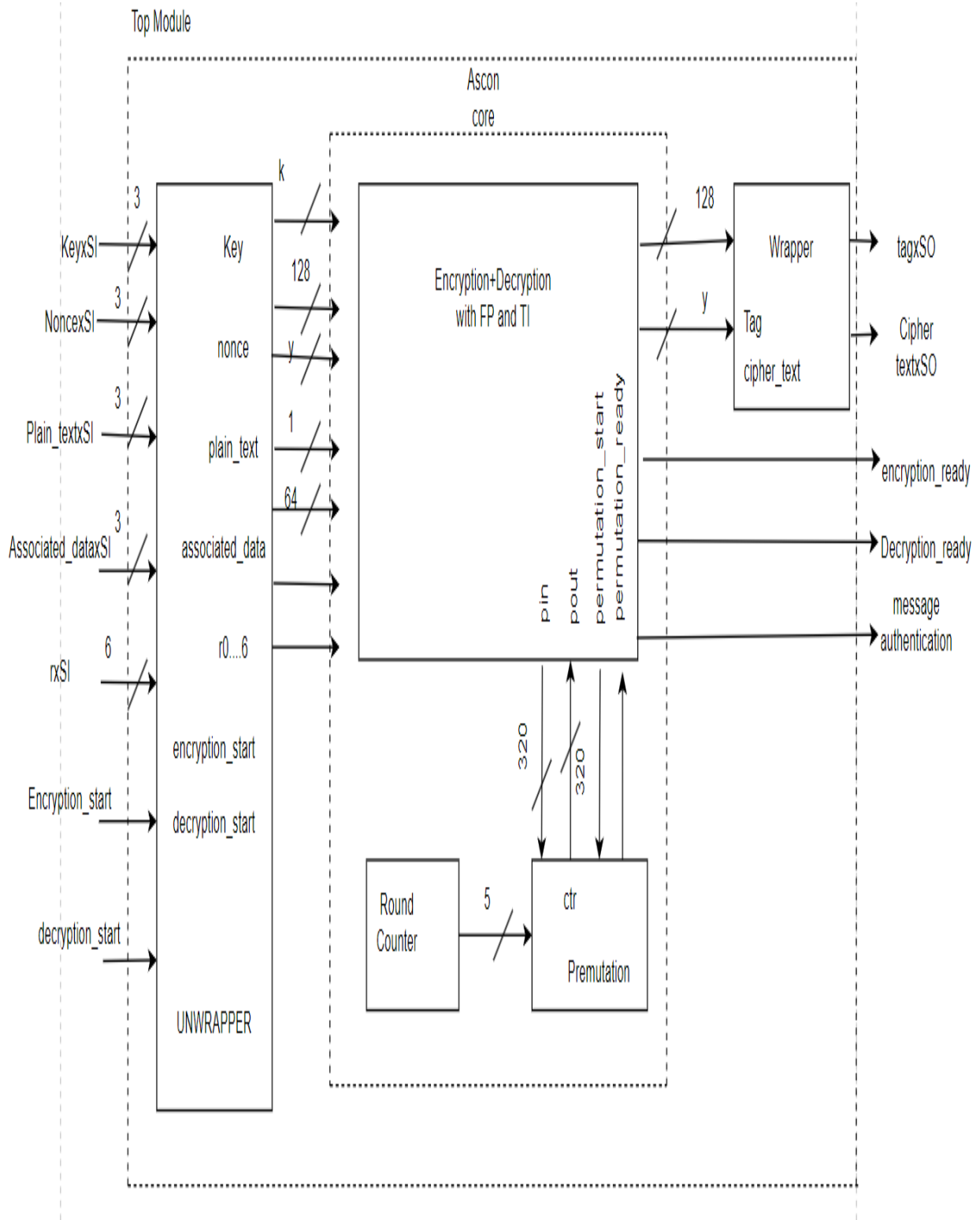


Figure 5.7: Hardware implementation

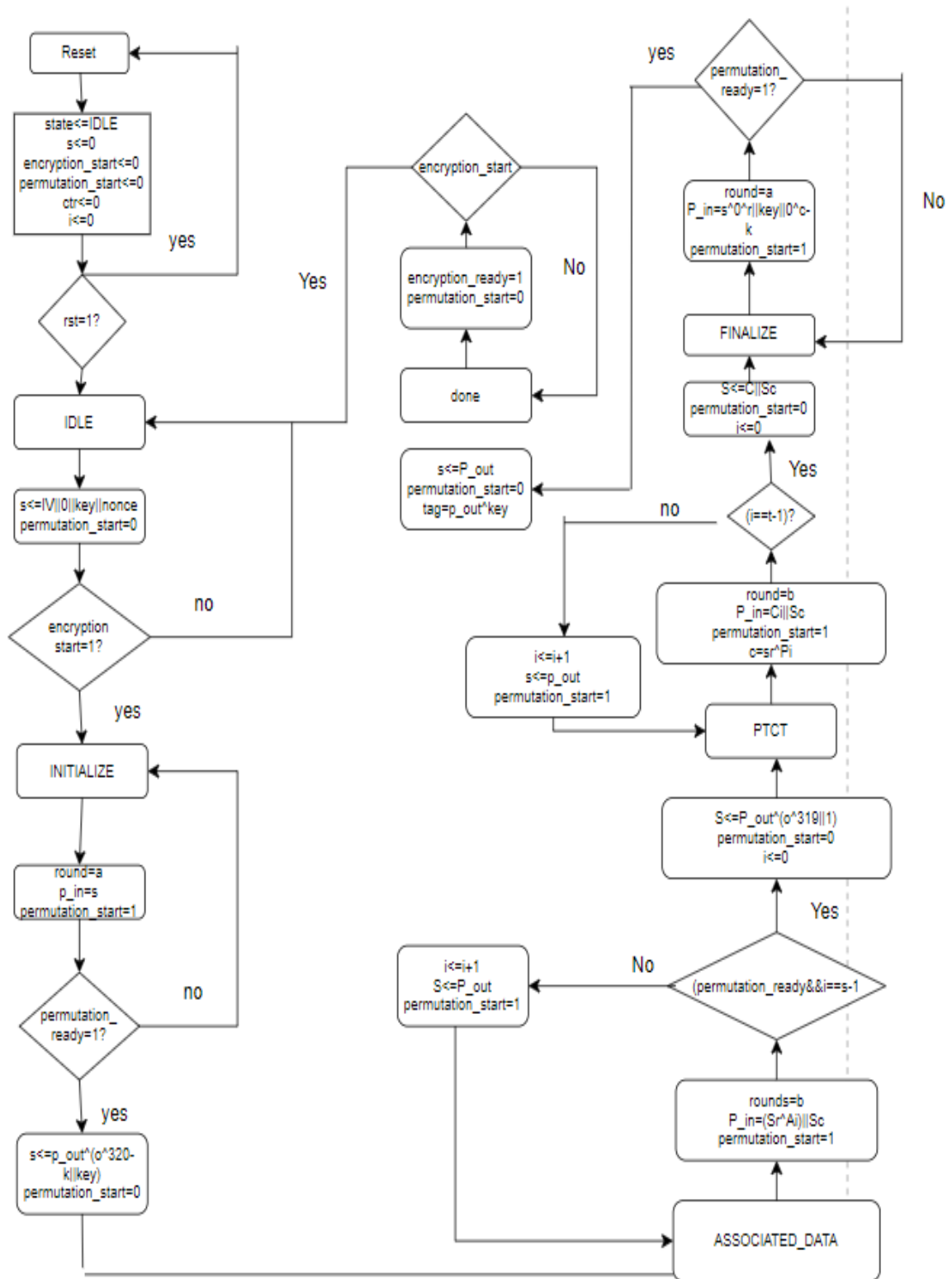


Figure 5.8: FlowChart

Table 5.2: Comparison between lightweight AES and ASCON

Feature	Lightweight AES	ASCON
Type	Block Cipher	Authenticated Encryption Algorithm
Algorithm Family	Symmetric-key	Symmetric-key
Key Size	128, 192, or 256 bits	128, 192, or 256 bits
Block Size	128 bits	Variable (typically 128 bits)
Design Philosophy	Substitution-permutation network	Sponge Construction
Feistel Structure	No	Optional (Feistel Variant)
Security Strength	Generally considered secure	Secure against known attacks
Performance	Fast, optimized implementations	Designed for lightweight platforms
Hardware Efficiency	Often implemented in hardware	Suitable for hardware implementations
Software Efficiency	Efficient software implementations	Designed for software efficiency
Applications	Wide range of applications	Constrained environments, IoT, RFID
Cryptographic Primitives	Encryption only	Encryption and authentication
Round Operations	Substitution, Permutation	Substitution, Permutation, XOR
Security Analysis	Extensively studied	Subject to ongoing analysis
Standards	Included in various standards	Considered for standardization

Non invertable components in ASCON

In ASCON (Authenticated Sponge Construction), certain components are designed to be non-invertible, meaning they cannot be directly reversed to recover the original input from the output. These non-invertible components are crucial for achieving security properties such as collision resistance and preventing attackers from easily reversing the cryptographic transformations. Here are some key non-invertible components in ASCON:

1.S-boxes: While ASCON uses invertible S-boxes in its round function (for confusion), these S-boxes are designed to be non-linear and non-invertible. This means that it's computationally infeasible to efficiently compute the pre-image of an output without knowledge of the original input.

2.Non-linear Operations: Apart from S-boxes, ASCON incorporates other non-linear operations such as addition modulo 2^n (XOR) and bitwise rotations. These operations contribute to the non-linearity of the algorithm, making it resistant to linear and differential cryptanalysis.

3.Round Constants: ASCON uses round constants that are introduced into the state at different rounds of the permutation layer. These round constants are designed to be non-reversible and play a crucial role in ensuring the security of the algorithm against various attacks.

4.Keccak-f Permutation (in non-Feistel variant): In the non-Feistel variant of ASCON, the Keccak-f permutation is used as part of the underlying sponge construction. Keccak-f is a non-linear, non-invertible permutation designed to provide strong cryptographic properties.

5.Diffusion Layer: ASCON employs a diffusion layer that ensures changes to one part of the state propagate throughout the entire state. This diffusion is achieved through a combination of linear and non-linear operations, making it difficult to recover the original input from the output.

These non-invertible components are carefully designed to enhance the security of ASCON against various cryptanalytic attacks while maintaining efficiency in both encryption and authentication processes. They contribute to the overall robustness of the algorithm and ensure that attackers cannot easily reverse the cryptographic transformations to recover sensitive information.

Table 5.3: Comparison between Feistel variant and Non Feistel variant

Feature	Feistel Variant	Non-Feistel Variant
Permutation Layer	Follows Feistel structure (divides state into halves, multiple rounds)	Employs alternative techniques (may involve substitutions and permutations directly)
Round Function	Typically Feistel-based round function (S-boxes, permuta- tions)	May be different, optimized for performance and security
Implementation	Potentially faster hardware implementations	May offer flexibility in software implementations
Design Complexity	May be simpler to analyze and implement	May offer different trade-offs in terms of security and effi- ciency
Security Analysis	Traditional Feistel structure and analysis apply	May require separate analysis due to different structure
Suitability	Potential advantages in hard- ware implementations	May be more adaptable for certain environments or plat- forms

Chapter 6: IMPLEMENTATION

6.1 IMPLEMENTATION

6.1.1 Formula Of Metrics Used

textbfa)Execution Time: The time it takes for a given operation or function to complete its execution is referred to as execution time. It is a finer-grained measurement than elapsed time and focuses on the time spent on a specific portion of a program.

Formula:

The execution time for ASCON depends on a number of factors, including the length, message, message length, additional data, its length, nonce, public key, and secret key.

In this algorithm we calculate Both Encryption Time and Decryption time.

Encryption Time:

```
//Perform encryption and calculate execution time clock_gettime(CLOCK_MONOTONIC,
&start_encrypt);
int encrypt_result = crypto_aead_encrypt(c, &clen, m, mlen, ad, adlen, nsec, npub, k);
clock_gettime(CLOCK_MONOTONIC, &end_encrypt);
encrypt_time = (end_encrypt.tv_sec - start_encrypt.tv_sec) + (end_encrypt.tv_nsec -
start_encrypt.tv_nsec) / 1e9;
```

Explanation:

This code measures how long it takes to encrypt data. It starts by recording the time before calling an encryption function named 'crypto_aead_encrypt'. The function likely encrypts some data ('m') with additional information ('ad') using a secret key ('k'). After encryption, the code records the time again. Finally, it calculates the difference between the two timestamps to determine the encryption time in seconds, accounting for both whole seconds and nanoseconds.

Decryption Time:

```
// Perform decryption and calculate execution time clock_gettime(CLOCK_MONOTONIC,
&start_decrypt);
int decrypt_result = crypto_aead_decrypt(m, &milen, nsec, c, clen, ad, adlen, npub, k);
clock_gettime(CLOCK_MONOTONIC, &end_decrypt);
decrypt_time = (end_decrypt.tv_sec - start_decrypt.tv_sec) * 1e9 + (end_decrypt.tv_nsec -
start_decrypt.tv_nsec) / 1e9;
```

Explanation:

This code mirrors the encryption measurement but focuses on decryption time. It starts by grabbing the current time before calling 'crypto_aead_decrypt', which likely takes ciphertext ('c'), additional data ('ad'), and secret key ('k') to retrieve the original message ('m'). Similar to encryption, the code captures the time after decryption and calculates the elapsed time in seconds. Here, the difference between 'end_decrypt' and 'start_decrypt' considers both whole seconds and nanoseconds (converted to seconds) to determine the decryption time.

b)Program Throughput:

Throughput refers to the rate at which a system processes data or completes tasks over a specific period of time. It is commonly measured in units such as transactions per second, data bits per second, or items per hour. Throughput is a crucial metric in evaluating the performance and efficiency of both computing and manufacturing systems.

Formula: The execution time for ASCON depends on a number of factors, including the message length and encrypt_time which is calculated before .

In this algorithm we calculate Both Encryption Throughput and Decryption Throughput.

Encryption Throughput:

```
double throughput_encrypt = mlen / encrypt_time;
```

Explanation:

This line of code calculates how much data the encryption process can handle per second, also known as the throughput. It does this by dividing the size of the original message (stored in mlen and measured in bytes) by the time it took to encrypt the message (stored in encrypt_time and measured in seconds). The result, stored in throughput_encrypt, is expressed in bytes per second (B/s). A higher throughput value indicates faster encryption. In simpler terms, it tells you how much data the encryption function can process in a single second.

Decryption Throughput:

```
double throughput_decrypt = mlen / decrypt_time;
```

Explanation: This line of code mirrors the throughput calculation for encryption but focuses on decryption. Throughput, in this context, refers to how much data the decryption process can handle per second. The code achieves this by dividing the size of the original message (stored in 'mlen' and measured in bytes) by the time it took to decrypt the message (stored in 'decrypt_time' and measured in seconds). The result, placed in 'throughput_decrypt', is expressed in bytes per second (B/s). A higher throughput value indicates faster decryption. In simpler terms, it calculates how quickly the decryption function can process the encrypted data (ciphertext) to retrieve the original message.

c)Avalanche effect:

The Avalanche effect refers to the property in cryptography where small changes in input result in significant and unpredictable changes in output, ensuring that minor alterations in plaintext or key produce vastly different ciphertext, enhancing security by obscuring relationships between input and output.

Formula:

the avalanche effect of ASCON depends on the number of factors including message,message length,additional data,additional data length,public key,secret key,cyphertext length,

Avalanche effect:

```
void calculate_avalanche_effect(const unsigned char* m, unsigned long long mlen,
const unsigned char* ad, unsigned long long
adlen
const unsigned char* npub, const unsigned char* k)
unsigned char c[512]; // Assuming a maximum ciphertext length of 512 bytes unsigned long long
clen;
unsigned char m_copy[256]; // Copy of the plaintext for modification
unsigned char k_copy[32]; // Copy of the key for modification

// Copy the original plaintext and key
memcpy(m_copy, m, mlen);
memcpy(k_copy, k, 32);

// Variables to store avalanche effect percentages
double plaintext_avalanche = 0.0;
double key_avalanche = 0.0;

// Perform encryption with original inputs
crypto_aead_encrypt(c, &clen, m, mlen, ad, adlen, NULL, npub, k);

// Modify each bit of the plaintext and calculate avalanche effect
unsigned long long i;
```

```

for (i = 0; i < mlen; ++i)
// Modify a single bit in the plaintext copy
m_copy[i]^= 0x01;

    // Encrypt the modified plaintext
    crypto_aead_encrypt(c, &clen, m_copy, mlen, ad, adlen, NULL, npub, k);

    // Calculate the percentage of bits that changed
    unsigned long long bit_changes = 0;
    unsigned long long j;
    for (j = 0; j < clen; ++j)
    unsigned char diff = c[j] ^ c[j - mlen]; // Compare corresponding bytes
    unsigned long long k ;
    for (k = 0; k < 8; ++k)
    if ((diff >> k) & 0x01)
    bit_changes++;
    }
    }
    }

    // Calculate and accumulate avalanche effect for plaintext bits
    plaintext_avalanche += (double)bit_changes / (clen * 8) * 100.0;

    // Restore the modified plaintext for the next iteration
    m_copy[i]^= 0x01;

    // Modify each bit of the key and calculate avalanche effect
    // unsigned long long i;
    for (i = 0; i < 32; ++i)
    // Modify a single bit in the key copy
    k_copy[i]^= 0x01;

```

```

    // Encrypt the plaintext with the modified key
    crypto_aead_encrypt(c, &clen, m, mlen, ad, adlen, NULL, npub, k_copy);

    // Calculate the percentage of bits that changed
    unsigned long long bit_changes = 0;
    unsigned long long j;
    for (j = 0; j < clen; ++j)
        unsigned char diff = c[j] ^ c[j - mlen]; // Compare corresponding bytes
    unsigned long long k;
    for (k = 0; k < 8; ++k)
        if ((diff >> k) & 0x01)
            bit_changes++;
    }
    }
    }

    // Calculate and accumulate avalanche effect for key bits
    key_avalanche += (double)bit_changes / (clen * 8) * 100.0;

    // Restore the modified key for the next iteration
    k_copy[i] ^= 0x01;

    // Calculate average avalanche effect percentages
    plaintext_avalanche /= mlen;
    key_avalanche /= 32;
    // Print the avalanche effect percentages
    printf("Avalanche Effect for Plaintext Bits: %.2f%%\n", plaintext_avalanche);
    printf("Avalanche Effect for Key Bits: %.2f%% \n", key_avalanche);

```

Explanation:

This code defines a function `calculate_avalanche_effect` that measures the avalanche effect of a cryptographic algorithm, specifically targeting the plaintext and key inputs. It encrypts a plaintext message using the provided key and then iterates through each bit of the plaintext and key, modifying one bit at a time. After each modification, it re-encrypts the plaintext with the modified key or plaintext bit and calculates the percentage of bits that change in the resulting ciphertext compared to the original ciphertext. This process allows it to determine how much each bit in the plaintext or key influences the output ciphertext. Finally, it calculates the average avalanche effect percentages for both the plaintext and key and prints the results. The avalanche effect is a crucial property in cryptography, ensuring that small changes in input lead to significant changes in output, enhancing security by obscuring relationships between input and output.

Chapter 7: CODING AND RESULT ANALYSIS

7.1 Results

7.1.1 Results in windows configuration

key	rate	encryption time(nanosec)	decryption time(nanosec)	encryption throughput	decryption throughput	alavanche effect
8	8	4.7000000	8.1000000	5106382.97872	2962962.966296	49
16	8	5.1000000	3.4000000	475882.35294	7058823.52941	48
32	8	4.4000000	3.5000000	5454545.45455	6857142.85714	49
8	16	4.2000000	3.0000000	5714285.71429	8000000.00000	45
16	16	3.4000000	3.0000000	7058823.52941	8000000.00000	45
32	16	3.6000000	3.1000000	6666666.66667	7741935.48387	45
8	32	2.9000000	5.4000000	8275862.06897	4444444.44444	51
16	32	3.0000000	2.9000000	800000.00000	8275862.06897	49
32	32	4.1000000	4.4000000	5853658.53659	5454545.45455	54

Figure 7.9: ASCON-128 in windows

key	rate	encryption time(nanosec)	decryption time(nanosec)	encryption throughput	decryption throughput	alavanche effect
8	8	5.2000000	3.9000000	4615384.61538	6153846.15385	51
16	8	5.0000000	3.5000000	4800000.00000	6857142.85714	49
32	8	4.9000000	3.5000000	4897959.18367	6857142.85714	52
8	16	5.5000000	3.5000000	4363636.36364	6857142.85714	46
16	16	4.1000000	3.0000000	5853658.53659	8000000.00000	43
32	16	4.9000000	2.9000000	4897959.18367	8257862.06897	42
8	32	3.8000000	2.9000000	6315789.47368	8257862.06897	54
16	32	3.3000000	2.9000000	7272727.27273	8275862.06897	52
32	32	3.0000000	2.9000000	8000000.00000	8257862.06897	55

Figure 7.10: ASCON-128a in windows

key	rate	encryption time(nanosec)	decryption time(nanosec)	encryption throughput	decryption throughput	avalanche effect
8	8	6.1000000	3.40000000	3934426.22951	7058823.52941	50
16	8	3.8000000	6.3000000	6315789.47368	3809523.80952	49
32	8	4.8000000	3.6000000	5000000.00000	66666666.66667	48
8	16	4.0000000	2.9000000	6000000.00000	8275862.06897	44
16	16	4.7000000	3.0000000	5106.38297872	8000000.00000	44
32	16	3.6000000	2.9000000	6666666.66667	8275862.06897	42
8	32	3.4000000	2.9000000	7058823.52941	8275860.06897	53
16	32	3.2000000	2.7000000	7500000.00000	8888888.88889	47
32	32	3.4000000	2.7000000	7058823.52941	8888888.88889	50

Figure 7.11: ASCON-80pq in windows

7.1.2 Results in Arduinio UNO Board

key	rate	encryption time (nanosec)	decryption time(nanosec)	encryption throughput(bytes per sec)	decryption throughput(bytes per sec)	avalanche effect%
8	8	7.3800001	7.6479997	2.9810297	2.8765689	49
16	8	7.3800001	7.6479997	2.9810297	2.8765689	53
32	8	7.3800001	7.6399998	2.9810297	2.8795812	50
8	16	6.3400001	6.5640001	3.4700314	3.3516147	42
16	16	6.3359999	6.559999	3.4722223	3.3536586	42
32	16	6.3359999	6.559999	3.4722223	3.3536586	42
8	32	5.9520001	6.5720000	3.6962363	3.3475348	39
16	32	5.9520001	6.5720000	3.6962363	3.3475348	39
32	32	5.9520001	6.5640001	3.6962363	3.3516147	39

Figure 7.12: ASCON-128 in Arduinio

key	rate	encryption time (nanosec)	decryption time(nanosec)	encryption throughput(bytes per sec)	decryption throughput(bytes per sec)	avalanche effect%
8	8	7.3800001	7.6440000	2.9810297	2.8780741	49
16	8	7.3800001	7.6479997	2.9810297	2.8765689	48
32	8	7.3800001	7.6440000	2.9810297	2.8780741	50
8	16	6.3359999	6.5640001	3.4722223	3.3516147	42
16	16	6.3359999	6.559999	3.4722223	3.3536586	43
32	16	6.3400001	6.559999	3.4700314	3.3536586	41
8	32	5.9520001	6.5640001	3.6962363	3.3516147	36
16	32	5.9520001	6.5679998	3.6962363	3.3495738	34
32	32	5.9520001	6.5640001	3.6962363	3.3516147	39

Figure 7.13: ASCON-128a in Arduino

key	rate	encryption time (nanosec)	decryption time(nanosec)	encryption throughput(bytes per sec)	decryption throughput(bytes per sec)	avalanche effect%
8	8	7.3800001	7.6519999	2.9810297	2.8750653	48
16	8	7.3800001	7.6479997	2.9810297	2.8765689	48
32	8	7.3800001	7.6479997	2.9810297	2.8765689	48
8	16	6.3359999	6.5560002	3.4722223	3.3557045	42
16	16	6.3400001	6.5640001	3.4700314	3.3516147	42
32	16	6.3359999	6.5560002	3.4722223	3.3557045	40
8	32	5.9520001	6.5720000	3.6962363	3.3475348	40
16	32	5.9520001	6.5640001	3.6962363	3.3516147	36
32	32	5.9520001	6.5640001	3.6962363	3.3516147	39

Figure 7.14: ASCON-80pq in Arduino

7.2 Result Analysis

7.2.1 Analysing the Result

a)comparing the results on windows and Arduino.

The discrepancy in performance between the Ascon algorithm on Windows and Arduino platforms can be attributed to differences in hardware capabilities and system architectures. Windows operating systems typically run on more powerful hardware with faster processors and more memory compared to Arduino microcontrollers. As a result, executing cryptographic operations, such as those performed by the Ascon algorithm, is inherently faster and more efficient on Windows. Conversely, Arduino boards have limited computational resources, including processing power and memory, which can lead to slower execution times and reduced throughput for complex algorithms like Ascon. Therefore, while Ascon may deliver impressive results on Windows systems, its performance on Arduino boards is constrained by the hardware limitations inherent to microcontroller platforms.

b)Input Size is limited for Arduino

When working with an Arduino board, it's important to be aware of its memory constraints, particularly when attempting to encrypt data. If the data size exceeds 2048 bytes, the board may not be able to complete the encryption process. This limitation arises because Arduino boards have limited RAM, which needs to accommodate not only the data but also the encryption algorithm and its overhead. During encryption, the data often increases in size due to processes such as padding, exacerbating the memory issue. Additionally, cryptographic operations require extra temporary memory for processing, which can quickly exceed the available space if the data is too large. In scenarios where memory capacity is a concern, it's crucial to either use a board with more RAM or implement data handling strategies such as breaking the data into smaller segments to ensure successful encryption.

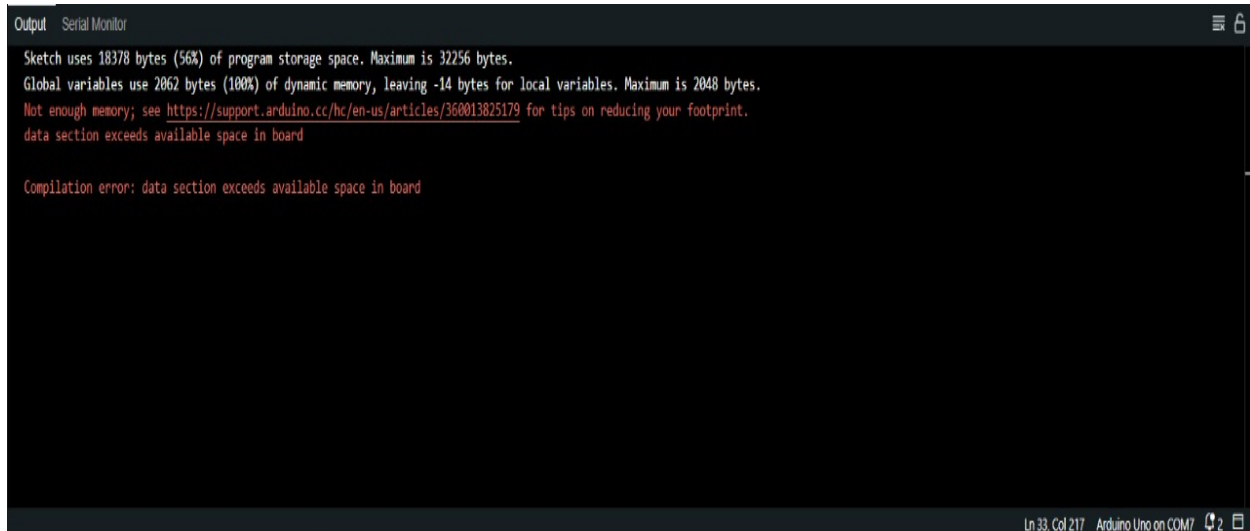


Figure 7.15: Limited size of Arduino

7.3 Code

7.3.1 code of ASCON in windows configuration.

a.Encryption Code

```
include "api.h"
include "ascon.h"
include "crypto_aead.h"
include "permutations.h"
include "printstate.h"
include "word.h"
include time.h

int crypto_aead_encrypt(unsigned char* c, unsigned long long* clen,
const unsigned char* m, unsigned long long mlen,
const unsigned char* ad, unsigned long long adlen,
const unsigned char* nsec, const unsigned char* npub,
const unsigned char* k) {
(void)nsec;
```

```

    /* set ciphertext size */
    *clen = mlen + CRYPTO_ABYTES;

    /* load key and nonce */
    const uint64_t K0 = LOADBYTES(k, 8);
    const uint64_t K1 = LOADBYTES(k + 8, 8);
    const uint64_t N0 = LOADBYTES(npub, 8);
    const uint64_t N1 = LOADBYTES(npub + 8, 8);

    /* initialize */
    state_t s;
    s.x0 = ASCON_80PQ_IV;
    s.x1 = K0;
    s.x2 = K1;
    s.x3 = N0;
    s.x4 = N1;
    P12(&s);
    s.x3  $\hat{=}$  K0;
    s.x4  $\hat{=}$  K1;
    printstate("initialization", &s);

    if (adlen) {
    /* full associated data blocks */
    while (adlen  $\bar{\geq}$  ASCON_128_RATE)
        s.x0  $\hat{=}$  LOADBYTES(ad, 8);
        P6(&s);
        ad += ASCON_128_RATE;
        adlen -= ASCON_128_RATE;
    }
    /* final associated data block */
    s.x0  $\hat{=}$  LOADBYTES(ad, adlen);
    s.x0  $\hat{=}$  PAD(adlen);

```

```

P6(&s);
}
/* domain separation */
s.x4  $\hat{=}$  1;
printstate("process associated data", &s);

```

```

    /* full plaintext blocks */
while (mLEN  $\hat{=}$  ASCON_128_RATE) {
s.x0  $\hat{=}$  LOADBYTES(m, 8);
STOREBYTES(c, s.x0, 8);
P6(&s); m += ASCON_128_RATE;
c += ASCON_128_RATE;
mLEN -= ASCON_128_RATE;
}
/* final plaintext block */
s.x0  $\hat{=}$  LOADBYTES(m, mLEN);
STOREBYTES(c, s.x0, mLEN);
s.x0  $\hat{=}$  PAD(mLEN);
c += mLEN;
printstate("process plaintext", &s);

```

```

    /* finalize */
s.x1  $\hat{=}$  K0;
s.x2  $\hat{=}$  K1;
P12(&s);
s.x3  $\hat{=}$  K0;
s.x4  $\hat{=}$  K1;
printstate("finalization", &s);

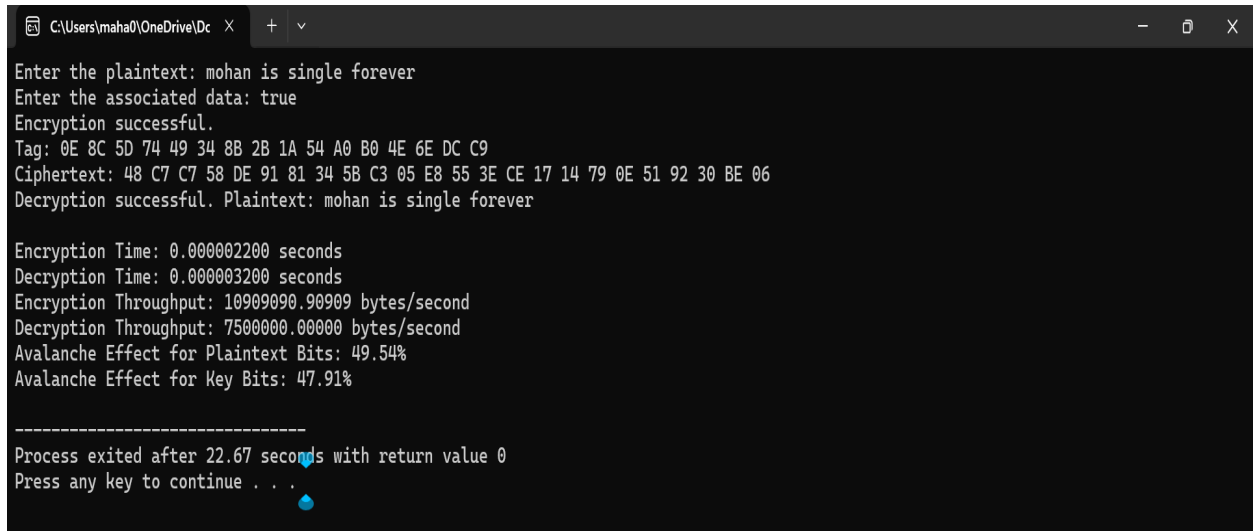
```

```

    /* set tag */
STOREBYTES(c, s.x3, 8);
STOREBYTES(c + 8, s.x4, 8);

```

```
    return 0;
}
```



```
C:\Users\maha0\OneDrive\Doc X + v
Enter the plaintext: mohan is single forever
Enter the associated data: true
Encryption successful.
Tag: 0E 8C 5D 74 49 34 8B 2B 1A 54 A0 B0 4E 6E DC C9
Ciphertext: 48 C7 C7 58 DE 91 81 34 5B C3 05 E8 55 3E CE 17 14 79 0E 51 92 30 BE 06
Decryption successful. Plaintext: mohan is single forever

Encryption Time: 0.000002200 seconds
Decryption Time: 0.000003200 seconds
Encryption Throughput: 10909090.90909 bytes/second
Decryption Throughput: 7500000.00000 bytes/second
Avalanche Effect for Plaintext Bits: 49.54%
Avalanche Effect for Key Bits: 47.91%

-----
Process exited after 22.67 seconds with return value 0
Press any key to continue . . .
```

Figure 7.16: Output Of Ascon in Windows

b.Decryption Code

```
include "api.h"
include "ascon.h"
include "crypto_aead.h"
include "permutations.h"
include "printstate.h"
include "word.h"
include time.h

int crypto_aead_decrypt(unsigned char* m, unsigned long long* mlen,
unsigned char* nsec, const unsigned char* c,
unsigned long long clen, const unsigned char* ad,
unsigned long long adlen, const unsigned char* npub,
const unsigned char* k) (void)nsec;

if (clen < CRYPTO_ABYTES) return -1;

/* set plaintext size */
*mlen = clen - CRYPTO_ABYTES;

/* load key and nonce */
const uint64_t K0 = LOADBYTES(k, 8);
const uint64_t K1 = LOADBYTES(k + 8, 8);
const uint64_t N0 = LOADBYTES(npub, 8);
const uint64_t N1 = LOADBYTES(npub + 8, 8);

/* initialize */
state_t s;
s.x0 = ASCON_80PQ_IV;
s.x1 = K0;
s.x2 = K1;
```

```

s.x3 = N0;
s.x4 = N1;
P12(&s);
s.x3  $\hat{=}$  K0;
s.x4  $\hat{=}$  K1;
printstate("initialization", &s);

```

```

    if (adlen)
/* full associated data blocks */
while (adlen  $\bar{\wedge}$  ASCON_128_RATE)
s.x0  $\hat{=}$  LOADBYTES(ad, 8);
P6(&s);
ad += ASCON_128_RATE;
adlen -= ASCON_128_RATE;
}
/* final associated data block */ s.x0  $\hat{=}$  LOADBYTES(ad, adlen);
s.x0  $\hat{=}$  PAD(adlen);
P6(&s);
}
/* domain separation */
s.x4  $\hat{=}$  1;
printstate("process associated data", &s);

```

```

/* full ciphertext blocks */
clen -= CRYPTO_ABYTES;
while (clen  $\bar{\wedge}$  ASCON_128_RATE)
uint64_t c0 = LOADBYTES(c, 8);
STOREBYTES(m, s.x0  $\hat{c}$  0, 8);
s.x0 = c0;
P6(&s);
m += ASCON_128_RATE;
c += ASCON_128_RATE;
clen -= ASCON_128_RATE;
}

```

```

    /* final ciphertext block */
    uint64_t c0 = LOADBYTES(c, clen);
    STOREBYTES(m, s.x0 ^ c0, clen);
    s.x0 = CLEARBYTES(s.x0, clen);
    s.x0 || = c0;
    s.x0  $\hat{=}$  PAD(clen);
    c+ = clen;
    printstate("processciphertext", &s);

    /* finalize */
    s.x1 = K0;
    s.x2  $\hat{=}$  K1;
    P12(&s);
    s.x3  $\hat{=}$  K0;
    s.x4  $\hat{=}$  K1;
    printstate("finalization", &s);

    /* set tag */
    uint8_t t[16];
    STOREBYTES(t, s.x3, 8);
    STOREBYTES(t + 8, s.x4, 8);

    /* verify tag (should be constant time, check compiler output) */
    int result = 0;
    int i;
    for (i = 0; i < CRYPTO_ABYTES; ++i) result || = c[i] ^ t[i];
    result = (((result - 1) ^ 8) & 1) - 1;

    return result;
}

```

```
C:\Users\ymaha0\OneDrive\De  X + v
Enter the plaintext: this is our ASCON project
Enter the associated data: completed
Encryption successful.
Tag: C9 65 FA CC 71 6C BE 55 7A 6D E3 F3 54 D6 06 51
Ciphertext: 1C 4E B0 BA 31 8A 8F 72 C0 34 4E 10 A3 28 36 98 99 F9 67 12 0A 5D FE 04 9B 50
Decryption successful. Plaintext: this is our ASCON project

Encryption Time: 0.000002300 seconds
Decryption Time: 0.000002700 seconds
Encryption Throughput: 11304347.82609 bytes/second
Decryption Throughput: 9629629.62963 bytes/second
Avalanche Effect for Plaintext Bits: 48.50%
Avalanche Effect for Key Bits: 49.44%

-----
Process exited after 34.13 seconds with return value 0
Press any key to continue . . .
```

Figure 7.17: Output Of Ascon in Windows

7.3.2 code of ASCON in Arduinio IDE

```
include Arduino.h
include string.h
include "api.h"
include "ascon.h"
include "crypto_aead.h"
include "permutations.h"
include "printstate.h"
include "word.h"
include "encrypt.c"

unsigned char m[256]; // Assuming a maximum message length of 256 bytes
unsigned long long mlen;
unsigned char ad[256]; // Assuming a maximum associated data length of 256 bytes
unsigned long long adlen;
unsigned char c[512]; // Assuming a maximum ciphertext length of 512 bytes
unsigned long long clen;
const unsigned char nsec[16] = {0}; // Nonce security, assuming 16 bytes (all zeros)
unsigned char npub[16]; // Nonce, to be generated randomly
unsigned char k[16]; // Key, to be generated randomly


void setup() {
Serial.begin(9600);


    // Generate random key (16 bytes)
    randomSeed(analogRead(A0)); // Seed for random number generation
    for (int i = 0; i < 16; ++i) {
        k[i] = random(256); // Random byte between 0 and 255
    }


    // Input the plaintext
    Serial.println("the plaintext:");
    while (Serial.available() == 0);
    String plaintext = Serial.readStringUntil("");
    plaintext.toCharArray((char*)m, sizeof(m));
```

```

mlen = plaintext.length();

    // Input the associated data
    Serial.println("Enter the associated data:");
    while (Serial.available() == 0);
    String associated_data = Serial.readStringUntil("");
    associated_data.toCharArray((char*)ad, sizeof(ad));
    adlen = associated_data.length();

    // Perform encryption
    unsigned long start_encrypt = millis();
    int encrypt_result = crypto_aead_encrypt(c, clen, m, mlen, ad, adlen, nsec, npub, k);
    unsigned long end_encrypt = millis();

    // Check if encryption was successful
    if (encrypt_result == 0)
    // Encryption successful, print the tag and ciphertext
    Serial.println("Encryption successful.");
    print_hex(c + clen - CRYPTO_ABYTES, CRYPTO_ABYTES);
    Serial.println(":");
    for (unsigned long long i = 0; i < clen; ++i)
    Serial.print(c[i], HEX);
    Serial.print(" ");
    }

    // Perform decryption
    unsigned long start_decrypt = millis();
    int decrypt_result = crypto_aead_decrypt(m, &mlen, nsec, c, clen, ad, adlen, npub, k);
    unsigned long end_decrypt = millis();

    // Check if decryption was successful
    if (decrypt_result == 0) {

```

```

// Decryption successful, print the plaintext
Serial.print("Decryption successful. Plaintext: ");
for (unsigned long long i = 0; i < mlen; ++i)
Serial.print((char)m[i]);
}
// Calculate and print throughput
double encrypt_time = (end_encrypt - start_encrypt) / 1000.0; // in seconds
double decrypt_time = (end_decrypt - start_decrypt) / 1000.0; // in seconds
double throughput_encrypt = mlen / encrypt_time;
double throughput_decrypt = mlen / decrypt_time;
Serial.print("Encryption Time: ");
Serial.print(encrypt_time, 9);
Serial.println(" seconds");
Serial.print("Decryption Time: ");
Serial.print(decrypt_time, 9);
Serial.println(" seconds");
Serial.print("Encryption Throughput: ");
Serial.print(throughput_encrypt);
Serial.println(" bytes/second");
Serial.print("Decryption Throughput: ");
Serial.print(throughput_decrypt);
Serial.println(" bytes/second");
} else {
// Decryption failed
Serial.println("Decryption failed.");
} } else { // Encryption failed
Serial.println("Encryption failed.");
}

// Calculate and print the avalanche effect
calculate_avalanche_effect(m, mlen, ad, adlen, npub, k);
}

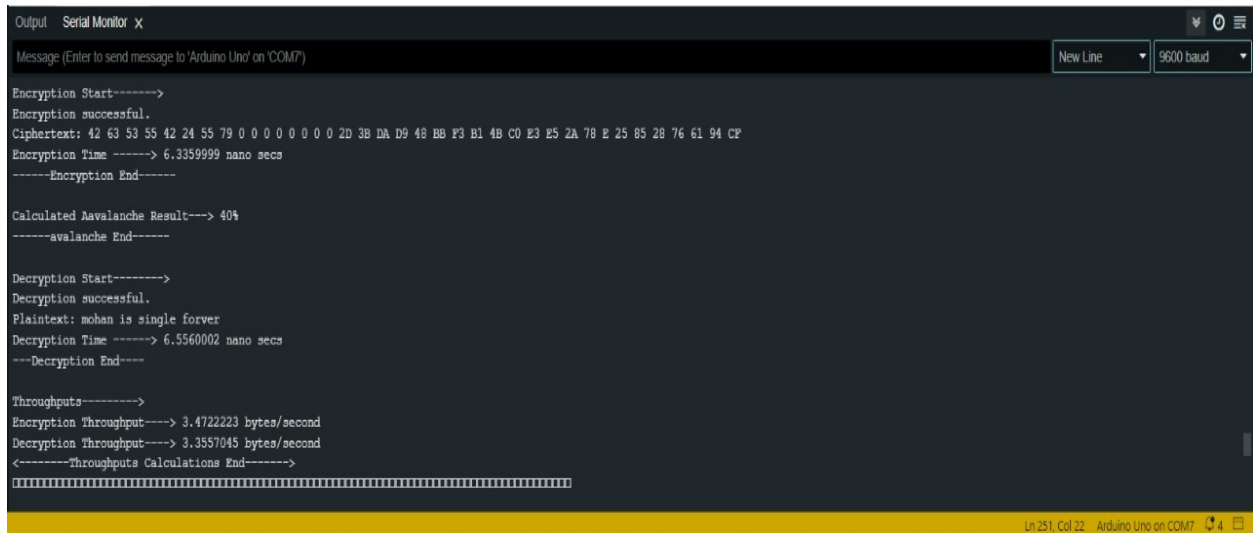
```

```

void loop() {

```

```
// Empty loop, as the code is designed to run once in the setup function
}
```



```

Output  Serial Monitor x
Message (Enter to send message to 'Arduino Uno' on 'COM7')
New Line 9600 baud

Encryption Start----->
Encryption successful.
Ciphertext: 42 63 53 55 42 24 55 79 0 0 0 0 0 0 0 0 2D 3B DA D9 4B BB F3 B1 4B C0 E3 E5 2A 78 E 25 85 28 76 61 94 CF
Encryption Time -----> 6.3359999 nano secs
-----Encryption End-----

Calculated Avalanche Result--> 40%
-----avalanche End-----

Decryption Start----->
Decryption successful.
Plaintext: mohan is single forever
Decryption Time -----> 6.5560002 nano secs
---Decryption End---

Throughputs----->
Encryption Throughput-----> 3.4722223 bytes/second
Decryption Throughput-----> 3.3557045 bytes/second
<-----Throughputs Calculations End----->

```

Figure 7.18: Output Of Ascon in arduino

Chapter 8: CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

In conclusion, the implementation and performance evaluation of the lightweight encryption algorithm Ascon on Arduino boards have yielded promising results for securing IoT devices. The project successfully integrated Ascon into the Arduino development environment, optimizing its implementation in C language for microprocessor-level efficiency. Through rigorous testing, Ascon demonstrated commendable performance metrics, including encryption decryption speed, memory usage, and energy consumption. These results underscore Ascon's suitability for resource-constrained environments, providing robust security without compromising performance. Overall, this project contributes valuable insights into the practical applicability and effectiveness of Ascon in ensuring the security of IoT deployments, paving the way for its adoption in various IoT applications.

Our project combining ASCON with Arduino is all about making small devices, like those in embedded systems, more secure. By using ASCON with Arduino, we're adding a layer of protection to IoT devices, keeping them safe from hackers. This project helps developers understand and use cybersecurity better, making encryption easier for everyone to use. It's like giving these devices a shield to keep them safe in the digital world, and in doing so, we're making the technology we rely on every day more trustworthy and secure .

8.2 FUTURE WORK

In our forthcoming endeavors, our primary objective is to augment both the security and efficiency of Ascon, a cryptographic algorithm, through the implementation of strategic modifications to its individual rounds. By refining and fortifying its cryptographic properties, we aim to bolster its resilience against potential attacks, thereby enhancing its overall security posture. It is crucial to note that these enhancements will be meticulously designed to ensure seamless compatibility with existing implementations of Ascon, minimizing disruption to established systems and workflows.

One pivotal aspect of our proposed enhancements involves the integration of a secret key into the decryption process of Ascon. This strategic addition serves to amplify the algorithm's security measures by imposing an additional authentication step, thereby imposing an additional layer of protection against unauthorized access to encrypted data. By requiring the possession of a secret key for decryption, we aim to significantly heighten the barrier to potential adversaries seeking to exploit vulnerabilities within the cryptographic system. This additional authentication mechanism holds promise for reinforcing the confidentiality and integrity of sensitive information safeguarded by Ascon, thereby elevating its effectiveness as a cryptographic tool in various applications and environments.

BIBLIOGRAPHY

- [1] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  ffer. Ascon v1.2. Submission to NIST, 2019. <https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf>
- [2] Tran, S.-N., Hoang, V.-T., Bui, D.-H., A Hardware Architecture of NIST Lightweight Cryptography Applied in IPSec to Secure High-Throughput Low-Latency IoT Networks. <https://ieeexplore.ieee.org/document/10224261> DOI: 10.1109/ACCESS.2023.3306420
- [3] Aneesh Kandi, Anubhab Baksi, Tomas Gerlich, Sylvain Guilley, Peizhou Gan, Hardware Implementation of ASCON, 2023. <https://csrc.nist.gov/csrc/media/Events/2023/lightweight-cryptography-workshop-2023/documents/accepted-papers/07-hardware-implementation-of-ascon.pdf>
- [4] Mickael Cazorla et al. “Survey and Benchmark of Lightweight Block Ciphers for MSP430 16-bit Microcontroller”. In: *Sec. and Commun. Netw.* 8.18 (Dec. 2015), pp. 3564–3579. ISSN: 1939-0114. DOI: 10.1002/sec.1281. URL: <http://dx.doi.org/10.1002/sec.1281>
- [5] sebastian1.renner, enrico.pozzobon, juergen.mottok. “Benchmarking Software Implementations of 1st Round Candidates of the NIST LWC Project on Microcontrollers”. 2019, OTH Regensburg, <https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/benchmarking-software-implementations-lwc2019.pdf>