Arab American University

Faculty of Graduate Studies

Software Implementation of Modified PRESENT Lightweight Cryptography for Low Power Consumption in AVR Microcontroller

By

Aref Rateb Khalil

Supervisor

Dr. Mohammad M. N. Hamarsheh

This thesis was submitted in partial fulfillment of the requirements for the Master`s degree in

Cyber Crime and Digital Evidence

September/ 2023

© Arab American University – Jenin year. All rights reserved.

Thesis Approval

Software Implementation of Modified PRESENT Lightweight Cryptography for Low Power Consumption in AVR Microcontroller

By

Aref Khalil

This thesis was defended successfully on 10/12/2023 and approved by:

Committee members	Signature
1. Dr. Mohammad M.N. Hamarsheh, Supervisor	.....
2. Dr. Adwan Yasin, Internal Examiner	.....
3. Dr. Saeed Salah, External Examiner	.....

Declaration

Aref Rateb Khalil with a university ID 202012172 declares that this thesis, "Software Implementation of Modified PRESENT Lightweight Cryptography for Low Power Consumption in AVR Microcontroller " is his own original work and that all informational and materials sources have been appropriately acknowledged.

I confirm that the material, facts, and concepts utilized in this thesis have all been properly cited and recognized in accordance with Arab American University citation guidelines.

I provide permission for this thesis to be archived and made accessible by Arab American University for research and educational purposes. Any additional uses or copies of this thesis must have my permission.

Name of Student: Aref Rateb Khalil.

University ID: 202012172

Signature: 

Date: 2/19/2024

Acknowledgment

I am sincerely grateful to my supervisor Dr. Mohammed Hamarsheh, for his help, assistance, suggestions, and patience. I would like to thank the Department of Cyber Crimes and Digital Forensics for their help and guidance. Thanks to all my friends and colleagues at Arab American University. I am grateful to my father, Dr. Rateb Othman, for his support and encouragement.

Abstract

The research aims to establish a modified version of the PRESENT lightweight cipher to consume less power in the AVR 8-bit microcontroller Atmel 128 with respect to code size metric and security metrics due to the Internet of Things hardware limitations. FELICS (Fair Evaluation of Lightweight Cryptographic Systems) framework simulated power consumption and code size. The results showed that the PRESENT cipher consumes many clock cycles compared to all other SPN ciphers in C software implementation. The P-layer of PRESENT consumed the majority of total power. The proposed MPRESENT cipher decreased the number of clock cycles from 3×10^7 clock cycles to 1.5×10^6 cycles to encrypt 128-bit. The code size of our proposed MPRESENT needed 2754 bytes, while other ciphers need more code sizes, such as LED, AES, Robin*, and Fantomas. The avalanche effect evaluated the result of security metrics by changing one bit in the plain text to see the relationship between the cipher and the plain text and changing one bit in the key. By analyzing the 8,384-bit dataset, the security results showed that changing one bit in the plain text of the original PRESENT changes around 32 bits; however, our proposed MPRESENT changes 31.38 bits over 64 bits. The proposed MPRESENT showed good results in terms of power consumption, code size, and avalanche effect.

Table of Contents

List of Tables

List of Figures

Chapter One: Introduction

Research on the Internet of Things (IoT) has a long tradition. This chapter discusses its restriction and mentions PRESENT cipher limitations in software implementation. Additionally, it details the specific knowledge gap that this topic aims to address. The chapter will also state the study's objectives and the scope of the thesis.

1.1 Introduction to Lightweight Ciphers

In the past several years, lightweight encryption has played an essential role in people's lives. Lightweight encryption algorithms are needed to fit and accommodate restricted devices like IoT devices. Moreover, lightweight cryptography aims to achieve acceptable security simultaneously with its hardware limitations. The trade-offs between the encryption process and performance exist. Equally important, IoT devices are increasingly ubiquitous across various domains of contemporary life. Smart homes can communicate together and act without any human interactions. Smart appliances like washing machines, smart TVs, and gas oven sensors can work effectively together. IoT is also involved in the medical and health fields. For example, IoT can be inserted inside a sick person to monitor their vital organs, so securing IoT becomes sensitive and necessary (Stajano).

Previous work concluded that IoT is a modern term that has appeared (Ashton and others). This is to say that any device with internet access that can communicate with another device with internet or network access can be referred to as an object or a thing. The refrigerator can be IoT. Nowadays, a smart refrigerator can be equipped with internet connectivity. It alerts you when your milk supply is running low (Daily Mail Online).

The IoT and Radio Frequency Identification (RFID) tags have power and

hardware limitations, and the cost of replacing the battery could cost more than a new device. Designing or enhancing encryption processing can be useful. Security researchers (ZDNET) noticed that over 100,000 IoT web camera devices could be easily exploited and hacked. A smartwatch is also vulnerable to hackers and can trace the location of the person who wears it. To prevent hackers from reading these valuable pieces of information, lightweight encryption is the main solution that accommodates the restricted resources to reach the desired level of cryptography.

We cannot apply traditional encryption algorithms to IoT devices due to resource limitations. Some limitations in IoT devices are the hard disk size, the Random Access Memory (RAM), the microcontroller's capacity, and the extremely limited battery capacity. Replacing these components may be very costly. Lightweight encryption has been developed to fit its small key size and operation limitations, which do not require much processing. It will require a lower power consumption, thereby allocating resources for other operations rather than expending significant power on encryption.

The PRESENT cipher is the most power-consuming compared to all other lightweight ciphers for software implementation. A study by (Dinu et al.) compared nineteen block ciphers and mentioned some of the performance metrics in a set of microcontrollers. Dinu et al. reported that PRESENT is an unfriendly software implementation, so Dinu et al. compared the result of PRESENT in assembly language instead of C language. Earlier, (Eisenbarth, Kumar, et al., “A Survey of Lightweight-Cryptography Implementations”) conducted a survey of lightweight ciphers and mentioned that PRESETN is hardware efficient and needs many clock cycles compared to other ciphers.

1.2 Scope of Study

This thesis focused on one of the most famous lightweight ciphers, the PRESENT cipher. The PRESENT cipher consumes more clock cycles than any lightweight cipher in encryption and decryption in C implementation. The thesis presented a new approach called modified MPRESENT, which reduces the clock cycles compared to the original PRESENT cipher. A modified MPRESENT version was created by first measuring the consumption of clock cycles of each PRESENT cipher layer and replacing the PRESENT permutation layer with the RECTANGLE permutation layer for its low consumption and ease of implementation in addition to adding circular shift for the first 16-bit in permutation layer. The research focused on the AVR 8-bit microcontroller by simulating the results of performance metrics (clock cycles and code size) by the FELICS framework in Ubuntu virtual machine (VM) and shows the impact of the proposed MPRESENT algorithm on the security by analyzing the avalanche effect for confusion and diffusion by testing 1920-bit. The code of encryption and decryption was written in C language.

1.3 Objectives

Based on the short preview above, the study's main objective was to establish a new modified PRESENT cipher that consumes low power compared to the original PRESENT in C software language. The proposed cipher is called MPRESENT. The main objective was further divided into several specific objectives as the following:

- Establishing a new modified MPRESENT cipher consumes a lower power cost than the original PRESENT cipher in C software implementation.
- Study each layer's power consumption in Substitution and Permutation Network (SPN) ciphers.
- Studying the impact of modified MPRESENT on code size metric.

- Comparison study of different lightweight ciphers and modified MPRESENT in terms of confusion and diffusion.

- Examining the relationship between modified MPRESENT and security metrics.

1.4 Thesis Outline

In addition to the introductory chapter, the thesis contains the following five chapters. Chapter Two introduces concepts and background related to cryptography and lightweight cryptography algorithms. Chapter Three provides a literature review of the recent papers on lightweight ciphers, performance metrics, and recent research on security aspects. Chapter Four delineates the research methodology and approach followed to conduct the research. Chapter Five analyzes our findings for performance metrics and security metrics, and Chapter Six states the conclusions and future work based on the results.

Chapter Two: Background

Cryptology comes from the Greek word "kryptos," which means secret. It is divided into two parts. The first part is encryption, the science of converting text to an unreadable format. The second part is cryptanalysis, the science of breaking the cipher by finding a weakness in the design.

In 4000 B.C., the Egyptians used cryptography, an art in place of science, to encode hieroglyphic messages (Kahn). The father taught the art of hieroglyphs to his sons, and Champollion broke it later on (Vaudenay). Later, Caesar (the Roman emperor) created a new technique for encrypting the data of his messages when communicating with others. By shifting the letters to the third next alphabet, the decryption is the inverse of the letter with a left shift to the third alphabet. This technique, now called substitution, means substituting one letter with another.

As PCs have grown, the modern microprocessor and its ability to multitask processes simultaneously have helped facilitate and shorten the time it takes to decrypt and perform cryptanalysis. The Caeser cipher was easily broken. However, PCs did the cryptanalysis in a few seconds by trying all the shifting possibilities.

In later decades and with the rise of the Internet, all parties are communicating with each other and taking part in financial transactions, necessitating robust cryptography. Cryptographic algorithms provide acceptable security while considering their effects on performance. As a result, encryption and security have become essential on the Internet. Now, most websites use secure connections by applying encryption algorithms.

Cryptography is based on the encryption of information and the inability of any

outside party to read the information. Encryption is based on a secret key for encryption and decryption, allowing only the sender and receiver to read the information sent between them. Cryptanalysis is a part of cryptography that tries to break down the encryption cycle and read the content without knowing the key. (Shannon) showed how cryptography can be applied in communication media, where the sender is Alice and the receiver is Bob, and they communicate in a channel. Eve is the third party trying to intercept the messages between Alice and Bob and did some cryptanalysis. The encryption algorithm's strength will make it harder for Eve to break down the encryption.

Some work (Kerckhoffs and Cryptographie Militaire ») introduced the Kerckhoffs principle, stating, "A cryptographic system should be secure even if everything about the system, except the key, is public knowledge.". The encryption algorithm must be public, except for the encryption key, which must be secret and unknown to others (Ferguson et al.). The algorithm's secrecy can be determined through reverse engineering, as it is implemented in either software or hardware.

It is hard to keep the encryption algorithm secret, unlike the security of the encryption key. Another factor to consider is the difficulty of replacing the entire algorithm in case of a leak or exposure. Moreover, using a different key for every communication is more logical than continuously changing the algorithm. Designing new self-algorithms is complex, and it could be vulnerable. The certified algorithms have passed intensive security tests to be trusted and marked as secure algorithms.

The claim that published algorithms are secure is not 100% correct. Many published algorithms have discovered weaknesses or been broken after they passed the National Institute of Standards and Technology (NIST) tests. For example, PC1

algorithms were introduced in 1997 and broken in 2012.

In summary, this chapter gives the basic concepts and terminology of cryptography. Section 1 introduces the main security concepts and how they relate to cryptography. Section 2 presents the two main cipher techniques. Section 3 briefly introduces encryption modes; Section 4 lists the required level of security; Section 5 presents the FELICS framework. Section 6 lists lightweight ciphers, and Section 7 states the conclusions.

2.1 Information Security's Main Components

Confidentiality, integrity, non-repudiation, and authentication are the four main security services that can be achieved by implementing cryptography (Alfred et al.). In addition, this main security service is the backbone of other security services, such as digital signatures and access control. Confidentiality, also known as secrecy, is a service that makes data only readable by the sender and receiver. Integrity is designed to expose data manipulation when storing or transmitting a message; changing one bit of data will greatly change the results. Authentication consists of two parts: data authentication and entity authentication. Data authentication guarantees that the intended sender sent the message, whereas entity authentication is also called user authentication. Non-repudiation prevents the sender from denying sending a message in cases of disagreement, such as financial transactions.

Cryptography can be mainly divided into two groups: symmetric encryption and asymmetric encryption. Symmetric encryption uses the same key to encrypt and decrypt, while asymmetric encryption uses one key for encryption and another for decryption (Alimzhanova et al.). Each of these techniques is used in one of the security service domains.

The symmetric key cipher uses the same key for encryption and decryption; symmetric is older than asymmetric encryption. It is used in block ciphers, stream ciphers, hash functions, and message authentication. One main challenge facing this technique is how to transmit the secret key.

Authentication ciphers achieve integrity, confidentiality, and authenticity in a symmetric chipper using one technique: AtE Authenticate and encrypt. The authenticate computation is based on a plain text tag. The tag is appended to the plain text, which will encrypt both the auth ID and the plain text. EtA Encrypt and then authenticate. The plain text is encrypted first, and authentication computes based on the cipher text and appends it to the cipher. E&A encryption and authentication are done at the same time.

Asymmetric encryption is referred to as "public key cryptography." It uses two keys: the public key and the private key. The private key cannot be determined from the public key. The private key must not be shared with anyone. The public key is public and can be shared with everyone. The sender will encrypt the transmitted data using the receiver's public key for confidentiality since it is shared with everybody, and the receiver is the only one who can decrypt it using the private key. RSA Graham et al. (1978) and ElGama (1985) are examples of public-key encryption.

2.2 Block and Stream Ciphers

Symmetric encryption can be used in block ciphers by dividing the message into blocks and encrypting each block with a secret key. The message must be reversible for decryption. For example, consider M a message; it will be divided into several blocks: $b0,\ b1, \ldots\ bn-1$, and the cipher is $c0,\ c1, \ldots\ cn-1$. The block cipher consists of many rounds. Each time it iterates, a set of layers is applied to ensure linear and non-linear security aspects using a subkey derivative from the main or master key.

Block cipher rounds have five different categories. (i) SPN: Known as the substitution (S) and permutation (P) network, the S layer tries to eliminate any linearity relation between the plain text and the cipher result by applying non-linear functions, like the S-box. The P-layer changes the position of bits in the block; (ii) Feistel Network (FN): It takes two inputs, the data, and the subkey, and the data is divided into two halves; one half is used in the round function, and the output is XORed with another half; the output size is the same as the input; (iii) ARX: This technique uses addition and rotation with XOR rather than an S-box layer; (iv) NLFSR: This is commonly used in stream cipher instead of block cipher; it uses nonlinear-feedback shift registers; and (v) Hybrid ciphers: It is a mix of two techniques that usually have performance or security metrics advantages.

The stream cipher is like an OTP (One Time Pad), and by XORing the plain text with the key. The key must be fully randomized for each plain text and unpredictable for other parties. However, it is difficult to share that large key if the input is so large, so the stream cipher creates the idea of sharing the pseudo-random key, XORing with the Initial Vector (IV), and then XORing with plain text.

2.3 Encryption Modes

Choosing the correct mode can speed up the encryption and decryption processes and make it hard for third parties to expose the plain text. The modes are Cipher Block Chaining (CBC), Electronic Code Book (ECB), Ciphertext Feedback (CFB), Output Feedback (OFB), and Counter Mode (CTR), Figure 1 shows how each block cipher mode works. However, the ECB mode has a disadvantage over other modes. It simply takes plain text and a key, as shown in Figure 1. A shows, whereas other modes have IV as input to the algorithm, that XOR with plain text before the encryption process happens, as in

Figures 1. B, 1. C, and 1.D show.

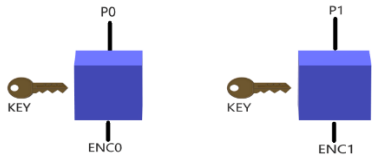
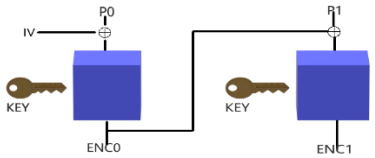


Figure 1.A: ECB Mode of Encryption



. Figure 1.B: CBC Mode of Encryption

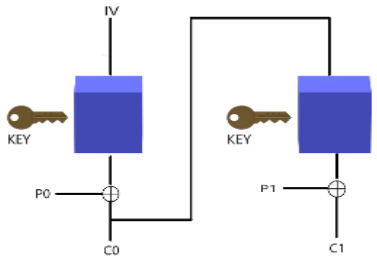


Figure 1.C: CFB Mode of Encryption.

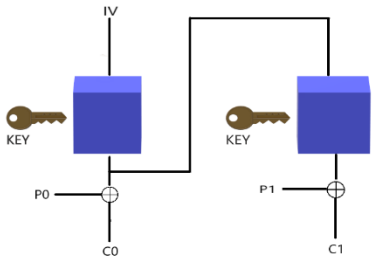


Figure 1.D: OFB Mode of Encryption.

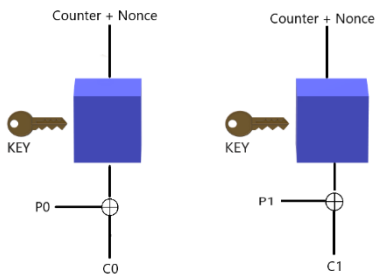


Figure 1.E: CTR Mode of Encryption.

Figure 1: Block Cipher Modes

2.4 Required Level of Security

Early work by Shannon (1949) introduced confusion and diffusion to ensure the

encryption algorithms met the security requirements and were secure enough to use. Most block ciphers apply confusion and diffusion in one of their layers; confusion can be preserved by a non-linear layer like the S-box layer, and diffusion maintains the linearity layer like the permutation or mixing layer. Increasing security in algorithms will lead to more processes, which means more power and time consumption.

The imperative trade-offs between security and performance are increasing. Confusion is between the cipher and the key. A slight change in the input key results in complex changes in the cipher output. Diffusion uses bit permutations to make the relationship between the cipher and plaintext complex.

2.5 FELICS Framework

Fair Evaluation of Lightweight Cryptographic Systems was used to compare software implementation when applied to IoT devices. The most important advantage is that the framework is open source. Thus, anyone can deal with the source code, make any modifications, or run to find the results.

The framework has comprehensive results. A comparison of metrics has been conducted for more than 19 block ciphers, and each cipher has more than one version. This enables us to choose the most appropriate version, considering their trade-offs. All results and details can be viewed via the FELICS website (*CryptoLUX > FELICS*). It is also flexible since the framework allows you to implement a new cryptographic primitive by following the guidelines and understanding its structure. The framework can do test verification among all ciphers and newly updated cipher primitives by a test vector.

The FELICS framework was developed in many programming languages. The source code of ciphers was written in C and C inline with assembly for supporting portability. The framework was also built in Python and bash script for ease of use. The

scenarios were built in C. Each primitive or cipher algorithm has its make file that must be run as the first step of generating the selected cipher results. Python was used for advanced scripting over bash script to make complicated scripts. The test vector is employed to check the output of the cipher to determine its correctness; the framework was usable for implementing new algorithms.

2.5.1 Sub-Core Module

The core module is the backbone of the framework, and each implementation will directly connect to the backbone core module to extract its metrics with different optimization compilers. Figure 2 shows the core module, implementing ciphers, and its components. Each microcontroller (AVR, MSP, and PC) has three performance metrics. When implementing a new cipher, the user can select which scenario to run (S0, S1, or S2) concerning performance metrics. This process is done by running a bash script for each performance metric.

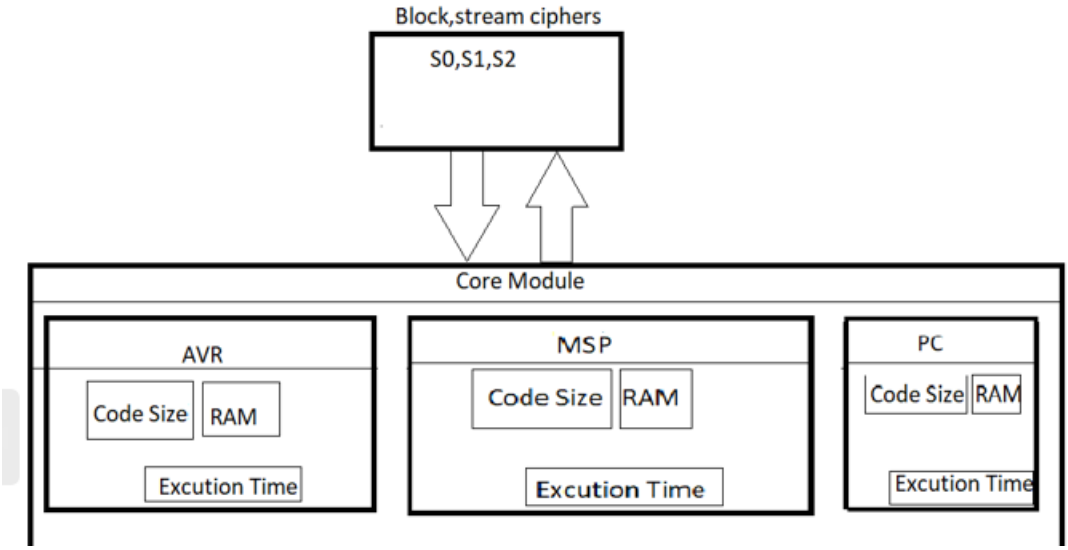


Figure 2:Core Module Components of FELICS Framework

The framework is not limited to embedded devices. It supports the evaluation of ciphers built for PCs. The core depends on a configuration file named conf.sh to extract the metrics from target devices and get_result.sh exists for all algorithms'

implementations to get exact performance.

2.5.2 Ciphers Implementation

Block encryption is implemented by defining a file that contains the components of the main encryption process, such as key scheduling, cryptography, and decryption, in a file written in C, known as the prototype. As for the implementation, it occurs inside another file, in which the parts and components of each of the 19 algorithms are written, all constants such as block size and key size are defined in a file named constant.h, and the implementation.info file contains a set of information for each algorithm.

The framework contains three scenarios. The first scenario is scenario zero, the traditional scenario in which a block of information is encrypted and then decrypted. The second scenario is scenario 1, where encryption and decryption are done using the CBC mode. The third scenario, scenario 2, is in CTR authentication mode.

The streaming cipher is similar to the block cipher, but the implementation occurs in a different C language file containing the constants file. As for scenarios, there are two types: scenario 0 and scenario 1. The framework also supports authenticated ciphers and hash functions. The hashing module has three scenarios: scenario 0, scenario 1, and scenario 2.

2.5.3 Output Formats and Supported Devices

This framework provides the flexibility to select the type and format of the output for any chosen scenario or coding, whether block, streaming, or any other type. It enables the visualization of the output in either the form of a CSV table or an XML format that is compatible with Microsoft applications.

In this framework, the simulation process is carried out on three appropriate and suitable types for IoT needs. First, AVR ATmega128 is used as an 8-bit architecture

(ATmega128), and TI MSP430F1611 (*MSP430F15*) is an example of using a 16-bit architecture. On the other hand, ARM and Cortex-M3 (*Arm Developer*) microcontrollers are examples of using 32-bit.

2.5.4 Metrics

The *collect_cipher_metrics.sh* file contains the three metrics used for all ciphers. The minor metrics can be derived from the three major metrics, and power consumption can be concluded for the number of clock cycles required to perform encryption or decryption. Considerable data can be beneficial for engineers to decide which algorithm to implement under a certain scenario, optimize compilers, and meet business requirements.

Code size is the number of bytes the algorithm stores in volatile memory, like RAM. Code size metrics are extracted by a tool called GNU. The code size results are extracted by running the script (*cipher_code_size.sh*). RAM metric is the size of the constants stored in static RAM, such as those needed in encryption, decryption, and key scheduling, and is calculated by the GNU tool. Execution time is the number of CPU cycles that must complete a certain process, such as encryption or decryption. It is calculated by subtracting the number of cycles at the end of execution from its beginning. For example, the Avrora (Titzer et al.; *Avrora - The AVR Simulation and Analysis Framework*) emulator calculates the number of cycles for encoding and decoding for AVR while the MSPDebug emulator is used for the MSP microcontroller (*MSPDebug*). The results are extracted through an automatic script, cipher_execution_time.sh, for any selected scenario.

2.6 Lightweight Block Encryption Algorithms

Block cryptography is widely used in lightweight encryption; the following are

the most commonly used lightweight block ciphers. HIGHT was first introduced in 1998 with a block size of 128 bits and 128 key bits. It is based on FN combined with ARX, and the output of FN is combined with XOR (Hong, Sung, et al.). Another cipher is LBlock, which was introduced in 2011 and is built upon FN with 32 rounds, tailored for use in resource-constrained hardware (Wu and Zhang). It has a 64-bit block size with an 80-bit key length to be efficient for hardware and software. LEA represents a relatively recent addition to the family of ciphers. It was introduced in 2013 and based on the ARX structure. LEA offers flexibility in terms of the number of rounds, with options of 24, 28, or 32, and is designed with a key and block size of 128 bits. A study by (Hong, Lee, et al.) showed that LEA is faster and more efficient than AES on different platforms. LED, first proposed in 2011, is like AES and supports 64-bit blocks with two versions of keys (64- and 128-bit keys) based on SPN structure (Guo et al.). Later, (Wang and Sako) present deferential cryptographic analysis success in LED. Piccolo is well suited for ultra-lightweight devices. The algorithm was first proposed in 2011 (Shibutani et al.). Based on the Feistel cipher, it can be 25 or 31 rounds with a 64-bit block size and 80 or 128-bit key; Shibutani proves it is immune against linear and differential attacks.

PRESENT cipher is based on the SPN structure. It consists of 32 rounds and is considered one of the leading ciphers in the lightweight field (Bogdanov et al.). PRESENT was introduced in 2007. It is built with a 64-bit block size and an 80- or 128-bit key size, and its S-box is 4-bits, designed to be more efficient for limited hardware. PRIDE cipher is considered an SPN cipher with a linear liar and an S-box liar; it has simple key scheduling, and the implementation shows that it fits well for software (Albrecht et al.). It has a 64-bit block size and a 128-bit key size. PRINCE an FS cipher; it consists of 12 rounds with a 64-bit block size and a 128-bit key size. PRINCE is

susceptible to a differential fault attack (Borghoff et al.). RC5 was introduced in 1994, it is a Feistel cipher (Preneel).

The 64-bit key size was vulnerable to being cracked; the 128-bit key size was strong against attacks. RECTANGLE is SPN cipher structure that can be efficient for hardware and software implementations (Zhang et al.). The differential attack success for 18 out of 25 rounds, and the P-layer consists of three rotations. RoadRunneR, introduced in 2015 (Baysal Adnan and Şahin) proposed a new lightweight cipher algorithm based on the FN structure with 10 or 12 rounds, an input block size of 64 bits, and an 80- or 128-bit key size. It has high performance on 8-bit processors.

Robin cipher is based on an SPN design; it accepts 128-bit block sizes and is suitable for software implementations (Grosso et al.). Simon is based on FN, and its design makes it perform very efficiently for hardware and software implementations (Ashur). The input is 64 bits, and the key can be either 92 or 128 bits in size, along with 28 or 32 rounds. SPARX considers an SPN structure with LTS (Long Trial Strategy), uses a large key size of 128 or 256 bits with 24 or 32 rounds, and the algorithm is vulnerable to an integral attack (Dinu Daniel and Perrin). Speck works well with hardware and software implementations; however, many attacks compromised it and showed the cipher algorithm's weakness (Beaulieu et al.).

TWINE considers an FN structure. The algorithm showed efficiency in software deployment with low RAM consumption, and the mentioned algorithm is vulnerable to a linear attack (Suzaki et al.). NIST accepted AES, and the structure is based on SPN. The meet-in-the-middle attack was successful for seven rounds out of 10 (Xing et al.). Chaskey, based on the FN structure with 128-bit input and 128-bit key size, along with 8

or 16 iterations, the deferential-linear attack succeeds after seven iterations (Mouha et al.). Fantomas considers that the SPN structure consists of S-box and L-box. The input data and key size are 128 bits and iterations (Grosso et al.).

Chapter Three: Related Works

The researcher presents the literature review related to the thesis field in this section. The use of the IoT in research, technology, and military fields makes it one of the most popular things now. The IoT needs to encrypt the data to make it unreadable. This will ensure confidentiality.

The method of implementing encryption algorithms can be done in two different ways: the first is a software implementation, and the second is applied directly to the hardware parts. The two methods have some tradeoffs: the hardware implementation could lead to better results in terms of power consumption. However, it is not considered portable code, unlike the software implementation, which is easy to write and portable. Therefore, implementing software or hardware depends on various aspects. Hardware implementations are based on the description of the hardware part. VHDL (VHSIC Hardware Description Language) is an example of a hardware description language (Navabi), FPGA (Kilts) and ASIC (Einspruch) are methods of implementing digital circuits on hardware chips. Software implementations are another method of implementing cryptography algorithms by writing a program for specific tasks like encryption and decryption. C language (Engler et al.) and assembly (Hyde) are the two languages commonly used in software implementations of encryption algorithms.

Software implementation is much easier than hardware implementation. Code can be written, and if a bug occurs at any time, the code can be updated by releasing fixed versions. Software implementation is easy to handle, fix bugs, and learn. Hardware implementation is faster in operating tasks than software implementation, but the cost of updating bugs or fixing errors is much more complicated with hardware implementation. It is not portable, and in some cases, it can be hard to implement a complex design.

Whether the choice is hardware or software implementation, it is important to make the algorithm achieve a high rate of efficiency that does not consume many resources. A good efficiency rate can be achieved by making the algorithm have a high usage rate. A good understanding of the algorithm and its structure is crucial. Hardware and software metrics are important parameters to consider when building or modifying encryption algorithms. Hardware metrics are area, latency or execution time, throughput, and power consumption (Mohd et al.). The area is quantified in GE (Gate Equivalents), where one GE corresponds to a two-input NAND gate size. Alternatively, it can be expressed as the physical size of the implementing circuit in square micrometers (μm^2).

Latency metric also means the execution time it measures in the clock cycle, while the throughput is the highest data rate that can be reached. Power consumption is the amount of power consumed to complete an operation and is normally measured in mW. While software metrics are code size, RAM, throughput, and clock cycles (Dinu et al.). Code size is the number of bytes needed to store binary code in non-volatile memory. RAM consumption is the maximum number of bytes a specific software implementation uses. Throughput can be defined as the highest data rate that can be reached, while power consumption is the amount of power the microcontroller or processor consumes. Execution time simulates the number of clock cycles needed for encryption and decryption.

This chapter is organized into three main sections: 3.1 presents related works for lightweight cipher frameworks; the software implementations of different ciphers on a set of microcontrollers are discussed in Section 3.2; and Section 3.3 presents recent research related to modified algorithms with their security aspects.

3.1 Lightweight Ciphers Frameworks

The NIST is based on issuing and approving encryption algorithms (*Hash Functions | CSRC*; Dworkin). The algorithms are initially evaluated in the first stage, and the weak ones are excluded. In the next stage, the algorithms are at a degree of convergence and suspicious of their characteristics, so their efficiency and characteristics in terms of design are considered in their application through software and hardware implementations (Gaj et al.).

NIST organized two workshops (*Lightweight Cryptography Workshop 2015 | NIST*; *Lightweight Cryptography Workshop 2016 | NIST*)to discuss light encryption, limited resources, and protection. It sets foundations and standards that consider the increase in demand on the Internet to provide the necessary and acceptable protection for the compatibility of limited resources. Certain tools have been used to break the tie among the algorithms participating in NIST. In 2018, NIST published a test vector generation code for an appropriate lightweight encryption option with authenticated encryption with associated data (AEAD) and optional hashing functionalities (*Lightweight Cryptography | CSRC*).

The BLOC project (*BLOC - Library*) measures criteria for evaluating the performance of 17 lightweight encryption algorithms. A research paper (Cazorla et al.) related to this project has been published to evaluate many algorithms related to lightweight encryption where the C programming language is used. This project measured the criteria for a 16-bit device such as the MSP430F1611 microcontroller. The criteria are execution time, RAM, and code size (*MSP430F16x, MSP430F161x MICROCONTROLLER*). The bash script extracts the result. The BLOC Project does not cover AVR, ARM, or PC. It only covers MSP.

eBACS project (*EBACS*) is based on measuring the speed of execution on each of the computers and servers needed by encryption algorithms and handling the speed of execution in some hash algorithms, stream ciphers, and authenticated and public key ciphers. It was developed in C in line with the assembly and Python programming languages, and it is open source; the tools are available on their website (*EBACS*). Unfortunately, the project only extracts the execution time and is limited to PCs.

XBX (Wenzel-Benner and Gräf) is an improvement of SUPERCOP and adds some features that scale hash functions on different microcontrollers. In addition, it added two properties, namely the code size and the RAM consumption. To know the size of the code, it analyzes the resulting binary file. To know the rate of RAM consumption, it collects the amount of consumption by the algorithm for the stack. It was written in C, Perl, and Bash, and encryption algorithms for 6, 16, and 32-bit microcontrollers were tested. Unfortunately, the project is offline, but the source code is still available (*XBX Embedded Hashing*). It's close to FELCS (*CryptoLUX > FELICS*); however, PC platforms are not included, and the owners do not continuously maintain them.

The Athena framework aimed to evaluate and report the results of implementing cipher algorithms. It evaluates them using the hardware description language, VHDL. It is open source (*ATHENa*), an extended version of the eBACS project (*EBACS*). The scripts were written in both Perl and Bash; they evaluated and measured execution time. Throughput and area indicate that the primary goal devices are Xlinx, FPGAs, and ACTEL; the main results of their basic tests can be found on their website (ATHENa).

The ECRYPT II evaluates the top three common metrics for code size, RAM, and execution time from the source code written in Assembly (Eisenbarth, Gong, et al.;

Balasch et al.). The source code and results are available at (*ECRYPT Lightweight Hash Functions*). It evaluates lightweight hash functions (Shrimpton, Hirose based on AES-256, S-Quark, and PHOTON-256/32/32). It extracts the three main metrics for the AVR platform: RAM, execution time, and code size.

The FELICS (Dinu et al.)framework is designed to evaluate a set of metrics for fair evaluation (Fair Evaluation of Lightweight Cryptographic Systems). It implemented nineteen lightweight block ciphers and around four stream ciphers with different scenarios. The source code of lightweight ciphers was designed with C and C with inlined Assembly for three architectures: 8, 16, and 32 bits. The FELICS framework is free, open-source under the GPL license, and easy to use. Sets of frameworks have been discussed by many researchers in the field (Law et al.; Eisenbarth, Kumar, et al., "A Survey of Lightweight-Cryptography Implementations"; Knežević et al.; Kerckhof et al.; Matsui and Murakami). Nineteen block ciphers and around four stream ciphers were measured by code size, RAM consumption, and clock cycle. By using three microcontrollers, the AVR ATmega128, TI MSP430F1611, and ARM Cortex-M3, for 8-, 16-, and 32-bit architectures, respectively, the source code of block ciphers were written in C and C inline with assembly for better results, resulting in the framework named FELICS (*CryptoLUX > FELICS*; Dinu et al.). FELICS, unlike the other frameworks mentioned, covers a wide range of platforms and performance metrics.

The best results for scenarios 1 (CBC mode) and 2 (authentication mode) were achieved by Chaskey, Chaskey-LTS, and Speck, who obtained a high FoM (Figure of Merit) ranking in contrast to Piccolo, TWINE, and LED. The results can vary based on the code version and FoM weight. Overall, the results contain mixed assembly and C implementations. Tradeoffs can be seen when considering execution time and code size

separately. We can realize that Chaskey is better than Speck, but the code size metric shows that Speck is consuming less space than Chasekey. The execution times in LED are longer than others, but the RAM consumption in TWINE is higher than in LED.

Based on the foregoing, the FELICS framework will be used in this thesis to extract and implement a new modified algorithm. FELICS shows comprehensive results. It covers all the main software performance metrics and its ease of usage. However, it does not cover basic security metrics like immunity against security standards like the avalanche effect, confusion, or diffusion.

3.2 Software Ciphers Implementation

A recent study by (Sevin and Mohammed) compared 50 lightweight encryption algorithms to understand what each algorithm needs regarding memory consumption, execution time, and throughput. The author has compared 8-bit architecture using a software implementation with various key sizes and a fixed input length. The IoT (Internet of Things) concept can be divided into many domains, such as sensors, clouds, devices (smart TVs, smart washer machines), and gateways (collectors that aggregate data from multiple sensors) (Asghari et al.). Execution time, RAM consumption, and code size are considered primary metrics, unlike power consumption and throughput, which are grouped as secondary metrics because they are derivatives of the main metrics.

Throughput is the amount of encrypted data transferred over a unit of time, usually a second. At the same time, energy consumption refers to the amount of power required to encrypt a single bit. Throughput can be extracted from the number of cycles, as Equation 1 shows CD is the number of cycles to encrypt the 128 bytes, f refers to the frequency (8 MHz), and NB is the block size. The amount of power consumption is a multiplication of encryption time by AVG power consumption. Equation 2 shows the

energy consumption, Vcc is the supply voltage of the microcontroller, and I is the number of currents per second (Eisenbarth, Kumar, et al., “A Survey of Lightweight-Cryptography Implementations”).

$$\text{Throughput} = \frac{\text{NB} * \text{F}}{\text{CD}} \quad (1)$$

$$\text{Energy} = \frac{\text{I} * \text{Vcc} * \text{CD}}{\text{f}} \quad (2)$$

The Atmel AVR 8-bit RISC microcontroller tested the 50 lightweight block ciphers. The Atmel Studio 7 platform simulated all 50 ciphers; the ATmega128 microcontroller was selected for its low power consumption and high performance. It consisted of 4 kilobits of RAM, 128 kilobits of memory, 4 kilobits of EEPROM, and 16 MHz. CBC mode was selected to encrypt and decrypt the data during the simulation. The results showed that the CHASKEY (Mouha et al.) has the lowest encryption and decryption times, which consumes about 10K cycles. In addition, it has fewer rounds and a simple ARX permutation layer, making it less power-consuming. This makes sense due to the correlation between the number of cycles and energy consumption, as Equation 2 shows. SPECK (Beaulieu et al.) ranks in second place; it has a simple round function; it is like CHASKEY; it is based on ARX; LEA (Hong, Lee, et al.) and SIMON (Ashur) are in third and fourth rank, respectively. PRESENT (Bogdanov et al.) fails to satisfy in terms of power. The author mentioned that it could not be more software-friendly due to the bit-oriented permutations. CRAFT consumes less code size than others, and PRESENT comes in third rank; it has a small S-box, and the P-layer does not have a table to store in. The third metric result is RAM consumption. The author stores the data of plain text, IV, key, and round key in the RAM, and the resulting output, which is a cipher, is placed over the plain text to make the RAM more efficient.

(Sevin and Mohammed) did a similar work related to FELICS; they tested the results on the Atmel Studio 7 platform for AVR and did not cover any security metrics.

A recent work by (Jangra and Singh) made compression between CLEFIA (Shirai et al.) and PRESENT (Bogdanov et al.) lightweight block ciphers, unlike the paper (Sevin and Mohammed) which used the Python programming language to extract throughput, memory usage, and security strength. He said that most traffic is not encrypted, and PRESENT and CLEFIA are light block ciphers; PRESENT is based on an SPN structure, whereas CLFIA is considered a Feistel network. Implementations for CLEFIA and PRESENT are done by the Intel(R) Core TM i7-7700@3.60 GHz to achieve security, performance, and resource utilization.

The results of the Avalanche effect completed the security evaluation, which is the percentage change in output for fewer changes in the input. For example, if one bit was changed in the input, there should be more change in the output. The security results showed that PRESENT overcomes CLEFIA. The experiment was performed by changing one bit in different positions to see the corresponding effect on the output cipher. Regarding performance, the author collected data from five files, and the results were based on them. By taking the average, PRESENT consumes many cycles compared to CLEFIA, with about 600 kilocycles/byte for PRESENT and 370 kilocycles/byte for CLEFIA. CLEFIA also shows the best throughput results at about 18 kbps and 11 kbps for PRESENT. It also consumes more time than CLEFIA, around 1440 s for PRESENT and about 870 s for CLEFIA. PRESENT is better than CLEFIA in terms of security and memory usage, whereas CLEFIA showed better results in throughput and time.

The results show that PRESENT performs poorly over CLEFIA and needs

modification to be a competitive cipher among other ciphers.

A study by (Edwar et al.) implemented a PRESENT lightweight block cipher on different platforms. Edwar used 16F and 18F from microchips as representatives of 8-bits, 24F as a 16-bit, and Cortex-M0 as a 32-bit microcontroller. The platform is KL25Z, which supports C and C++. Flash memory needs 2210 bytes, data memory RAM consumes 73 bytes (without pointers) and 768 bytes (with pointers), and the throughput is 12 m/s as a result of 8-bit microcontroller in encryption. The 16-bit microcontroller consumed 3276 bytes of flash memory; 245 bytes were consumed during testing RAM, and the throughput was 18 b/s. The 32-bit microcontroller consumed 20.5 K for flash memory, and 600 bytes for RAM, and the throughput was close to 234 bytes.

Previous work completed recently (Panahi et al.) handled ten lightweight encryption algorithms (AES (Xing et al.), PRESENT (Bogdanov et al.), LBlock (Wu and Zhang), Skipjack (Kim and Phan), SIMON (Ashur), XTEA (Moon et al.), PRINCE (Borghoff et al.), Piccolo (Shibutani et al.), HIGHT (Hong, Sung, et al.), and RECTANGLE (Zhang et al.)) to determine how much each algorithm needs in terms of RAM and ROM consumption, the time required for the encryption and decryption process, and the amount of energy consumed for cipher, decipher, and throughput on both Raspberry Pi 3 and Arduino Mega 2560. To obtain the result, the simulation was performed using a Pi 3 with requirements of 1 GB RAM, a 64-bit ARM Cortex-A53 processor that was powered by 5-V Micro USB and could also be powered by a power bank, and an Arduino Mega 2560 that has 256 KB of flash memory, 8 KB of SRAM, and 4 KB of EEPROM with a 16 MHz ATmega2560 8-bit microcontroller, and a cloud service of Dropbox. Encryption is performed on the sender side, then sent to the cloud service, and deciphering is done on the destination side. The simulation was written in C to extract

the results. Equations 3 and 4 measured the power consumed.

$$\text{Charge (C)} = \text{Current Stream (A)} * \text{Time (Seconds)} \quad (3)$$

$$\text{Energy (J)} = \text{Charge ©} * \text{Voltage (V)} \quad (4)$$

The results showed that PRESENT consumes more power on both devices than other cipher algorithms because it is hardware efficient. On the other hand, XTEA, Skipjack, and RECTANGLE have less power consumption for PI 3. For the Mega 2560 device, Skipjack, Hight, and RECTANGLE have the lowest power consumption. Atmel Studio 7 is used for RAM and ROM simulation; XTEA and RECTANGLE currently hold a prominent position for RAM consumption. They do not consume much RAM for PI 3 devices, while Mega Hight and PRESENT consume the highest amount of RAM. For ROM usage, SIMON is more efficient in both devices, and RECTANGLE and AES are less efficient in PI 3 and Mega 2560, respectively. For the execution time, PRESENT takes the longest time compared to other algorithms in the encryption and decryption processes for PI 3 and Arduino Mega 2560. Equation 5 calculated the throughput metric. For a 16-byte payload, the XTEA (Moon et al.) has the highest throughput in the Pi 3 device, and the PRESENT (Bogdanov et al.) has the worst values in all modes in Pi 3 and Arduino 2560.

$$\text{Throughput} = \frac{\text{Number of Bytes}}{\text{End Time} - \text{Start Time}} \quad (5)$$

Another study (Ghorashi et al.) presented two methods for enhancing Klein lightweight encryption metrics: CPU usage, RAM consumption, and processing time. The two methods were written in Python on the Raspberry Pi 3. The first approach is to subrogate the less efficient resource consumption with others that do not require high

hardware resource consumption. The Mix-Nibble consumes more resources than others, which is replaced with the 3-stage version of the S-box. The other method is increasing the number of iterations (12, 16, 20, and 32) to harden against key recovery with the same key and input block size. The result of the first method in Pi 3 shows its superior performance in terms of RAM utilization (17076 KB) and time (0.088 sec) compared with the original version (RAM: 17324 KB and 0.260921 sec). The result of the second method is repeated four times for accuracy. The result showed that with 12 iterations, Klein has the highest clock speed for the CPU.

A review paper by (Sehrawat and Gill) introduced a set of lightweight block cipher algorithms through a comprehensive study of these lightweight block ciphers that match the IoT requirements and conditions. Some lightweight encryption algorithms can perform better in hardware than software implementation; both techniques have advantages. Software deployment can be more flexible and easier to patch, unlike hardware implementation, which is not portable. According to (Sehrawat and Gill), around 47% of the block's ciphers are SPN, 40% are Feistel, and 12% are mixed ciphers. Well-designed algorithms must achieve high throughput, be simple, consume less power, have low RAM and power consumption, and apply cryptography standards. The paper briefly explained about 40 algorithms named Improved Lilliput (Pookuzhy Ali et al.), GIFT (Banik et al.), DLBCA (AlDabbagh), LiCi (Patil et al.), SKINNY (Ankele Ralphand Banik), MANTIS (Beierle Christofand Jean), SPARX (Dinu Danieland Perrin), and other 30 lightweight encryption algorithm.

In conclusion, PRESENT (Bogdanov et al.), mCrypton (Lim and Korkishko), and KLEIN (Ghorashi et al.) are poorly performing in software implementations. CLEFIA (Shirai et al.) also requires high RAM due to the large lookup table and key scheduling

stores in RAM.

Recently published work (Lee et al.) implemented standard AES along with PRESENT, SPECK, SIMON, and CAEFIA on both the CPU in ARM processors (A15 and A7) and GPU (Mali T628 (*ODROID*)). The authors discuss the power consumption for the stated lightweight block ciphers. The simulation was done on the ODROID-XU3 platform, which has an embedded power analysis tool. The evaluation and simulation processes were performed on the ODROID-XU3 on the Linux LUBUNTU 14.04 version, with A15 running at 2.0 GHz, A7 at 1.4 GHz, and T628 at 600 MHz. The five selected ciphers were encrypted with the C programming language in single, multiple, and GPU modes in the CTR cipher mode. All five ciphers were simulated ahead with a 128-bit key size for fair evaluations, and the execution time was computed for the key scheduling and encryption processes. The simulation process was performed ten times, and then the average was taken for execution times and power consumption, computed by a function supported in Linux called “get the time of day”. The results of the energy efficiency study show that PRESENT is less efficient than other ciphers for both the CPU and GPU. In the IoT field, the microcontroller is widely used over others.

3.3 Modified Algorithms with Security Aspects

Some researchers (Rana and Abul Kashem) discuss a new approach to modifying Split-Radix called MSBK (Modified Split Butterfly for Key). The author presented its security strength as a proof of concept by measuring its avalanche effect, diffusion, and confusion. In addition to performance metrics such as execution time and RAM usage measured by the FELICS (*CryptoLUX* > *FELICS*; Dinu et al.) tool, MATLAB was used in 2021a to measure its security. The proposed MSBK considers the limitations of IoT devices. The MSBK was designed to be lightweight and secure simultaneously, and they

added the butterfly architecture to the key scheduling process to make the key for each round highly secure. XOR and XNOR are the two main factors of MSBK. The algorithm is considered non-linear because it randomly generates output without linear relation. The structure of MSBK is mainly based on a split-radix butterfly structure. It takes four inputs and outputs the same size, which is 4 bits, by XOR and XNOR operations in addition to two random numbers. By analyzing the results using FELICS (*CryptoLUX > FELICS*), it requires fewer clock cycles than PRESENT (Bogdanov et al.), AES (Xing et al.), HIGH (Hong, Sung, et al.), LEA (Hong, Lee, et al.), SIMON (Ashur), SIT (Usman et al.), and G-cipher (Pliam) for the key generation process. In addition to performance metrics, MATLAB was used to evaluate its security. The avalanche effect test by encrypting an image using MATLAB. Encrypting an image with a key and changing one bit in the key makes the image unrecognizable, proving the avalanche effect.

Other work (Rana et al.) proposes a new mixed lightweight algorithm, mixing SP with Feistel cipher architecture; it is a symmetric block cipher, FELICS (*CryptoLUX > FELICS*) simulates the new algorithm in addition to MATLAB for security simulations. Usually, lightweight ciphers can have 10 to 32 rounds; this algorithm only has five rounds. The suggested algorithm contains two parts: the first is key scheduling, and the second is the encryption process. The algorithm is implemented in the C programming language. The result of RAM usage is about 34. The cycle required for generating the key is 1630, and the encryption cycle is 792. The MATLAB test shows that it achieved the avalanche effect since the image is not recognizable after changing one bit.

(Usman et al.) submitted a neSIT (Secure IoT) algorithm. It is a mix of SPN and Feistel cipher. It takes 64 data blocks as input and the same for output, with a 64-bit key size. The simulation was done using an 8-bit microcontroller. Certainly, increasing the

security will directly affect the performance metric (Chandramouli et al.), as ciphering more securely means more arithmetic operations, which leads to consuming resources. The algorithm consists of five rounds; however, it consists of shifting, swapping, and substations to increase security by achieving high levels of confusion and diffusion. The simulation was performed using MATLAB on an Intel Core i7-3770@3.40 GHz processor to check the avalanche effect. The hardware performance metrics are simulated based on ATmega 328 Arduino. The execution time of the SIT algorithm is 0.188 msec for encryption and 0.187 msec for decryption. While RAM usage is 22 bytes, and the number of clock cycles needed for encryption is 3006 in hardware implementation. The result of changing one bit in the key result of changing 49% of total bits makes the output unrecognized.

Chapter Four: Design and Methodology

This chapter presents proposed MPRESENT lightweight cryptography. The primary goal is to ensure the MPRESENT version consumes less power than the original PRESENT. The new cipher has passed through a set of performance and security tests. Since Section 3.1 discusses a set of frameworks used for the performance evaluation of sets of encryption algorithms, we choose FELICS (Dinu et al.). Framework since it has advantages over other frameworks (*BLOC - Library*)(*EBACS*) (Wenzel-Benner and Gräf), FELICS covers a wide range of platforms and performance metrics, it supports multiple scenarios and updates continuously. The performance metrics (clock cycles and code size) are simulated by an 8-bit microcontroller (AVR ATmega128) for both scenario 1 and scenario 2 (details about the scenarios in Section 1.5.2). The proposed version of the MPRESENT algorithm still has the structure of SPN. This chapter presents in Section 4.1 the design of the MPRESENT algorithm. Section 4.2 lists the performance metrics, and Section 4.3 states the security metrics.

4.1 Design of MPRESENT

To enhance the power efficiency of MPRESENT over PRESENT cipher, we conducted an analysis of nine Substitution-Permutation Network (SPN) ciphers, namely, AES (Xing et al.), Fantomas (Grosso et al.), LED (Guo et al.), Pride (Albrecht et al.), Prince (Borghoff et al.), Rectangle (Zhang et al.), Robin (Grosso et al.), and PRESENT (Bogdanov et al.). We analyzed separately the execution time of each layer (S-box and P-box) to determine which layer is consuming more power. The following is a general pseudo-code for all SPN ciphers.

Encrypt

{

S-Box

P-Box

XOR with Key

}

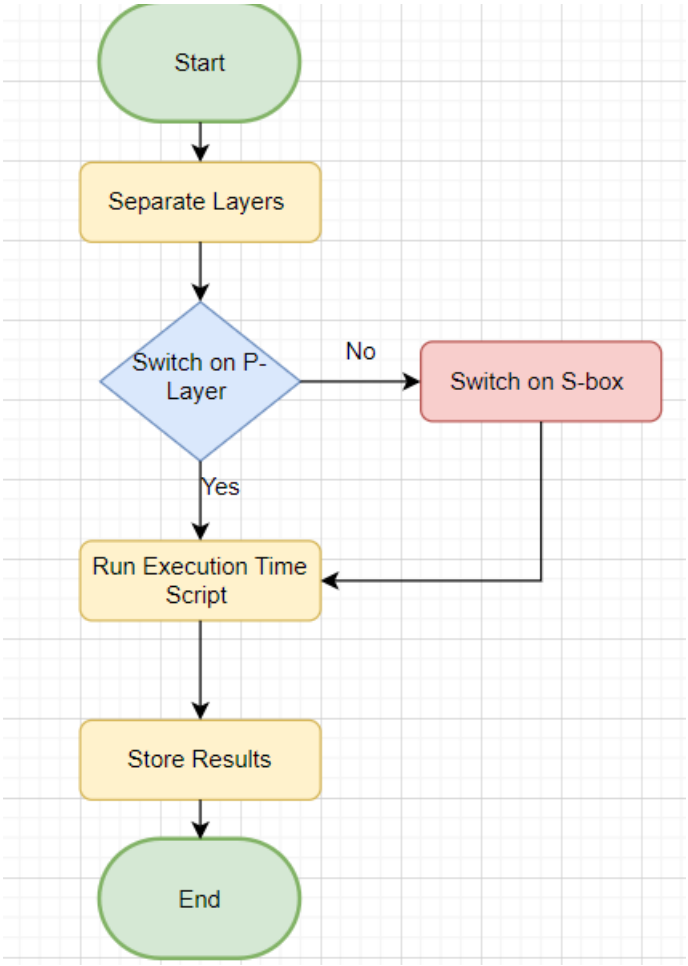


Figure 3:Workflow of Script for SPN Layers

To make a power analysis of each layer, we simply comment on all other parts of the code. For example, we comment on the P-box and run the cipher to know how much the S-box consumes power. Figure 3 shows the flow chart of the execution time extraction of each layer.

The Figure shows the steps of how we analyzed each SPN layer of encryption code separately. The first step is to switch the P-box on and turn off the S-box by commenting the S-box layers and running the FELICS script to extract the result of the active layer (P-box), then switch the S-box on and Switch the P-box off by commenting the P-box layer and extract the execution time for the S-box. After running the scripts, the output is stored in a file.

After analyzing each layer of 10 SPN cipher algorithms, we propose the modified MPRESENT algorithm based on the power result and simplicity of implementation. The proposed MPRESENT consists of the original S-box of PRESENT and P-box of RECTANGLE cipher in addition to the circular shift layer. The number of rounds still consists of 31, the same as the original PRESENT cipher, and each round's key schedule is still like the PRESENT. It accepts a data block of 64 bits and a key size of 80 bits. Figure 4.A shows the layer of PRESENT Cipher which consists of S-Box and P-Box; Figure 4.B shows MPRESENT cipher which consists of three layers: the S-Box of PRESENT cipher, P-Box of RECTANGLE cipher, and the circular shift layer. The proposed cipher still preserves the structure of the original PRESENT. The following is the pseudo-code of the new cipher. The full encryption and decryption code is illustrated in Appendix A and Appendix D, respectively.

```
Encrypt (block, roundkey)
{
    For (I=0,I<31,I++)
    {
        block XOR roundkey
```

S-box // same as original PRESENT

P-box // RECTANGLE layer

Circular shift

}

block XOR key[62,63]

}

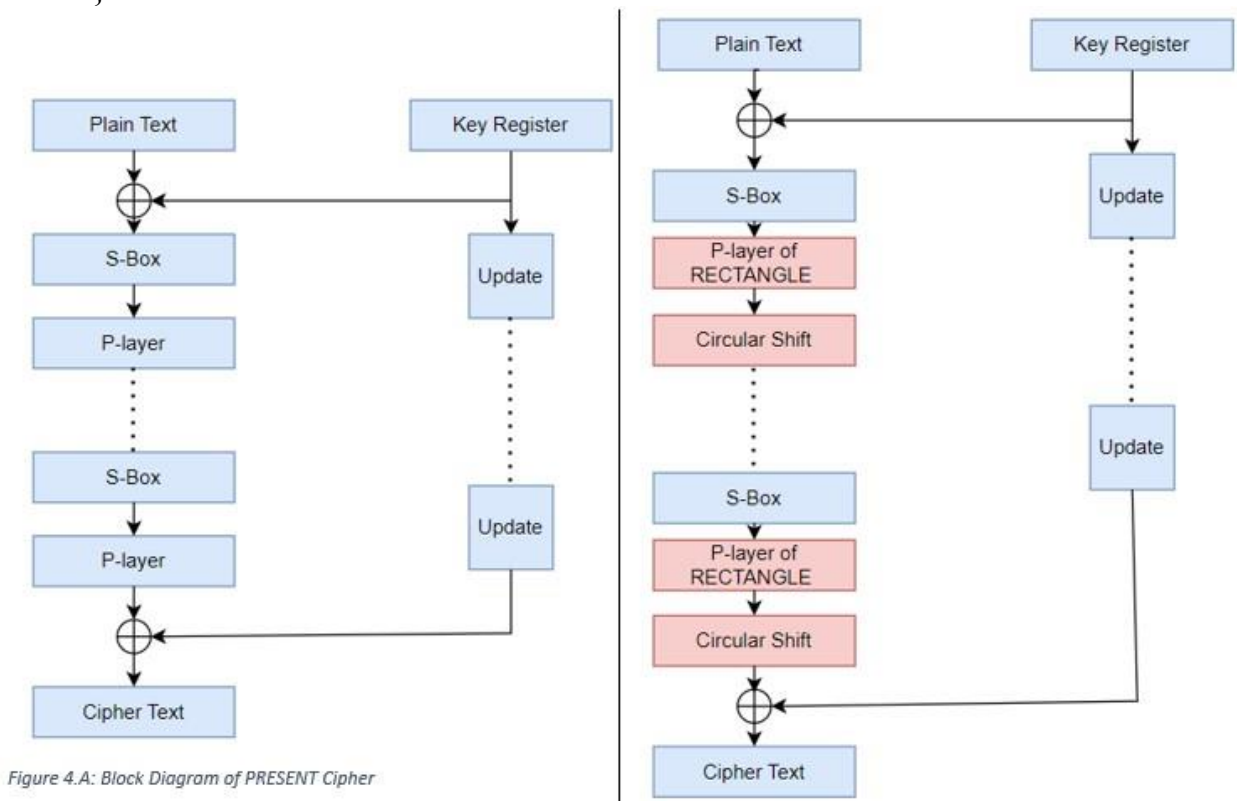


Figure 4:PRESENT and MPRESENT Block Diagram

The S-box layer is 4-bit input, resulting in 4-bits of output. The S-box table is represented in hexadecimal. Table I shows the S-box table. The S layer aims to increase the confusion by its nonlinearity relation between the inputs and outputs.

Table I:S-Box layer of PRESENT Cipher

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[X]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

The output $S[x] = S[Wi]$, whereas $Wi = b4 * i + 3||b4 * i + 2||b4 * i + 1||b4 * i$.

(Guo et al.) discuss in detail the design criteria of the S-box layer and proof of the avalanche effect for the S-box.

The output of the S layer is the input of the P-box. The P-layer is based on shifting bits and is considered linear. The P-layer consists of three definitions: offset 0 is unchanged, and no rotation applies. Data of offset 1 will be left shift by 1 bit and then right shift by 15 bits. Offset 2 has a left shift to 12 bits followed by a right shift of 4 bits and then XOR (^), offset three consists of a left shift of 13 bits and 3 bits right shift and XOR with the bit at position 13 and position 3, the offset 3, the definition of Player as follow:

```
#define ROTL1(x)     (((x)<<1)^((x)>>15))

#define ROTL12(x)    (((x)<<12)^((x)>>4))

#define ROTL13(x)    (((x)<<13)^((x)>>3 ))
```

The last layer is the circular shift layer. The P-layer's output will be directed as input to the circular shift. It will shift the first byte in the next round to the last byte, which means the P-layer will be applied to the last 3 bytes beside the one shifted byte. The following part of the code $\#define\ ROTL13(x)(((x) << 13)\text{^}((x) >> 3))$ that will increase the diffusion along with the P-box.

The proposed MPRESENT was written in C language, and the full encryption code is in Appendix A. The core of the code is an encrypted function “void Encrypt(uint8_t *block, uint8_t *roundKeys)” it accepts two inputs, the block and the round key. The state variable has the value of the 64-bit block, and the operation starts for a loop that iterates 31 times. The XOR operation applies to the state and key(subkey_lo and subkey_hi). The key casts to 32-bit for each subkey_lo and_hi, which means that the round key consists of 64-bit. The first layer in the code is the S-box. This part of the code now has a for loop for accessing all 16 bytes. It handles 1 hex at a time, 4-bit input, and 4-bit output as Table I, and the P-layer is the next layer in the code. It takes the “state” variable as input and passes it to “P_layer(uint16_t *data).” This function is implemented to handle 48-bit, data[1], data[2], and data[3]. Each is 16-bit, and data [0] does not pass to the P_layer function. Figure 5 shows how P_layer function handle data[1], data[2], data[3].

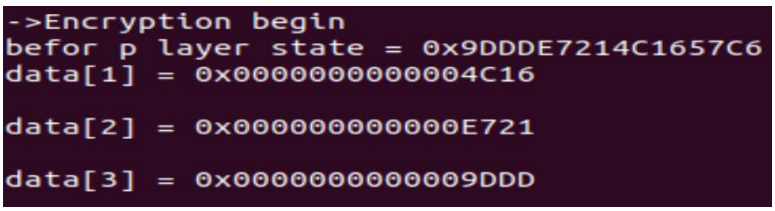


Figure 5: Output of P_layer for MPRESENT Cipher

The P_layer (ROTL1, ROTL12, and ROTL13) discussed in the previous section, the last layer is a simple circular shift by applying the “uint64_t shift(uint64_t value)” function as Figures 6 and 7 show.

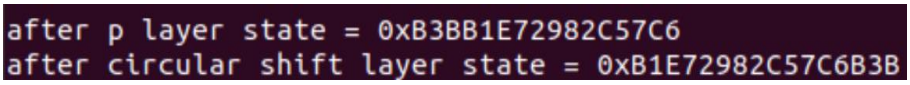


Figure 6: Output of Circular Shift Layer of MPRESENT Cipher

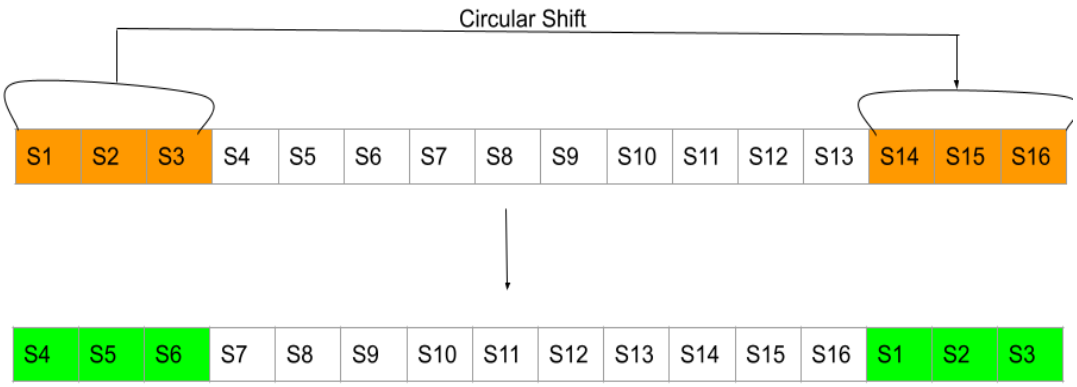


Figure 7: Circular Shift Process of MPRESENT

This function will shift the first 3 bytes to be the last 3 bytes, and the S4 byte will be left shift to be the first byte, as Figure 7 shows. The last step after the loop completes 31 rounds when the "state" variable will XOR with round-key of (subkey_lo[62] and subkey_hi[63])).

4.2 Performance Metrics

We extract the execution time or clock cycle needed for encryption and decryption as it considers the primary metric that gives insight into the power consumed by a cipher algorithm. The result of execution time includes clock cycles for the encryption process, decryption process, encryption key schedule, and key schedule for the decryption process.

The second performance metric is code size, as optimizing the power consumption of our new proposed MPRESENT cipher cloud directly affects the code size. We analyzed our modified cipher using an AVR 8-bit ATmega128 microcontroller with option O3 for the compiler and scenarios 1 (CBC mode) and 2 (authentication mode).

4.3 Security Metrics

Optimizing a performance metric could directly affect other metrics. Security metrics were analyzed for the impact of enhancing power on the cipher's security performance. We select confusion and diffusion as two primary security metrics.

Our modification of the P-layer will affect the diffusion of the cipher for evaluation. We analyzed 1920-bit as input to test the confusion and diffusion by changing one bit for all input blocks. The datasets are random data. Appendix B shows all datasets used as input to ten SPN ciphers.

SPN ciphers are divided into two groups: Group A has a block size of 64-bit and a key size of 80-bit, and Group B has a 64-bit block size and 128-bit key length. When testing confusion for group A, the size of the block is 64-bit while the key is 80-bit size, the data block input equals “123456789ABCDEF0” for all 131 inputs, and the key input equals the datasets in Appendix B, with the appending two-byte “00” at the end to make the size of key equals to 80-bit. The data block input is the 131 datasets to analyze the diffusion, and the key is “123456789ABCDEF01234”. Group B has a key size of 128-bit and a data block of 64-bit. Group B will have a static value of key equal to “123456789ABCDEF0123456789ABCDEF0” when testing diffusion for datasets in Appendix B and a block size value equal to 123456789ABCDEF0 when testing confusion for datasets in Appendix C.

Chapter Five: Experimental Results and Discussion

This chapter presents the analysis of the results for our proposed MPRESENT cipher. The results and the discussion compared to a set of lightweight ciphers. The result showed the tradeoffs between security metrics and performance metrics.

First, the execution time metric (clock cycle) consumed by each SPN layer of the SPN ciphers was presented to determine which layer had the most power consumption. Second, we compared the code size and execution time of our proposed MPRESENT cipher with other SPN ciphers for scenarios 1 and 2, comparing the results with the recent papers. Third, a discussion of security metrics for SPN lightweight ciphers.

This chapter is presented in Section 5.1, which discusses the execution time of the SPN layers. Section 5.2 stated the performance metrics of our proposed cipher along with other ciphers, and Section 5.3 presented the result of our security metrics for SPN ciphers.

5.1 Clock Cycles for SPN Layers

Table ii and Table iii showed the number of clock cycles needed to encrypt and decrypt the block cipher for each layer of given ciphers in CBC mode. This analysis helps us know which layer consumes more power than the other layer for SPN ciphers. Some of the given ciphers operate with block sizes of 64-bit or 128-bit.

Table II:Clock Cycles for Each Layer of Encryption

SPN Cipher	S-Box	P-Layer	L-Layer	Mix-Layer	Shift-Layer
PRESENT	695713	14906737	—	—	—
LED	230977	—	—	3472289	142529
FANTOMAS	12913	—	23777	—	—
PRIDE	90353	—	86465	—	—
RECTANGLE	62225	52065	—	—	—
ROBIN	43809	—	54201	—	—
ROBIN*	69121	—	74737	—	—

Table III:Clock Cycles for Each Layer of Decryption

SPN Cipher	S-Box	P-Layer	L-Layer	Mix-Layer	Shift-Layer
PRESENT	689861	14877941	—	—	—
LED	230789	—	—	3419137	143989
FANTOMAS	15020	—	26092	—	—
PRIDE	91781	—	88805	—	—
RECTANGLE	42773	52645	—	—	—
ROBIN	41580	—	56964	—	—
ROBIN*	66140	—	75956	—	—

Figure 8 represents the number of clock cycles needed for each layer separately in a chart form. The y-axis in the Figure represents the number of clock cycles needed to encrypt one data block. The y-axis in Figure 9 represents the number of clock cycles needed for decryption. The X-axis represents the name of the SPN cipher and its corresponding layers.

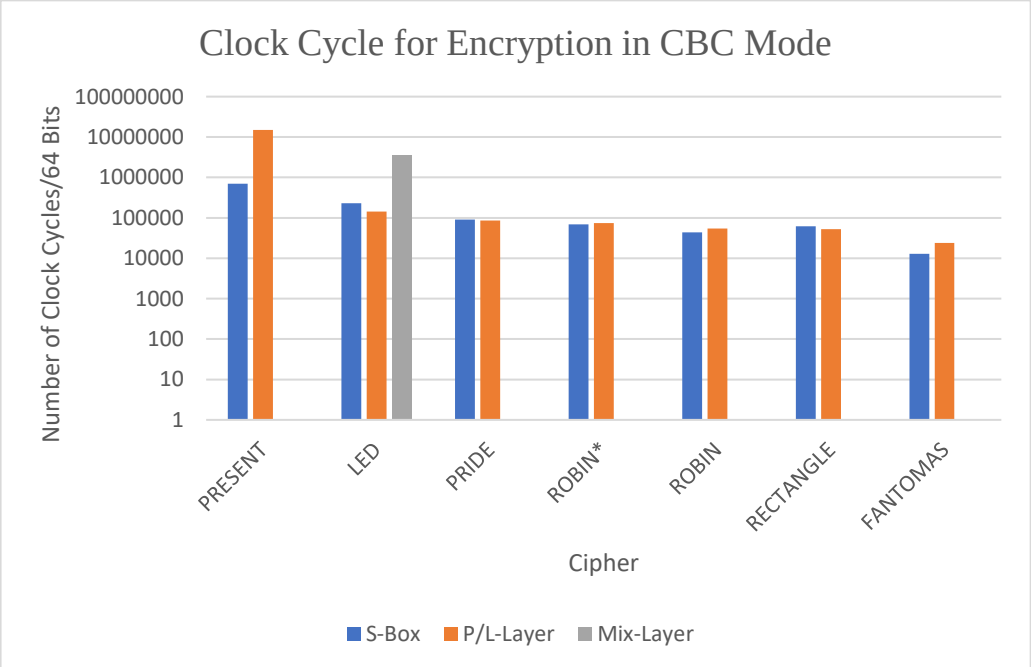


Figure 8:Clock Cycles Consumed for S/P-layer in Encryption

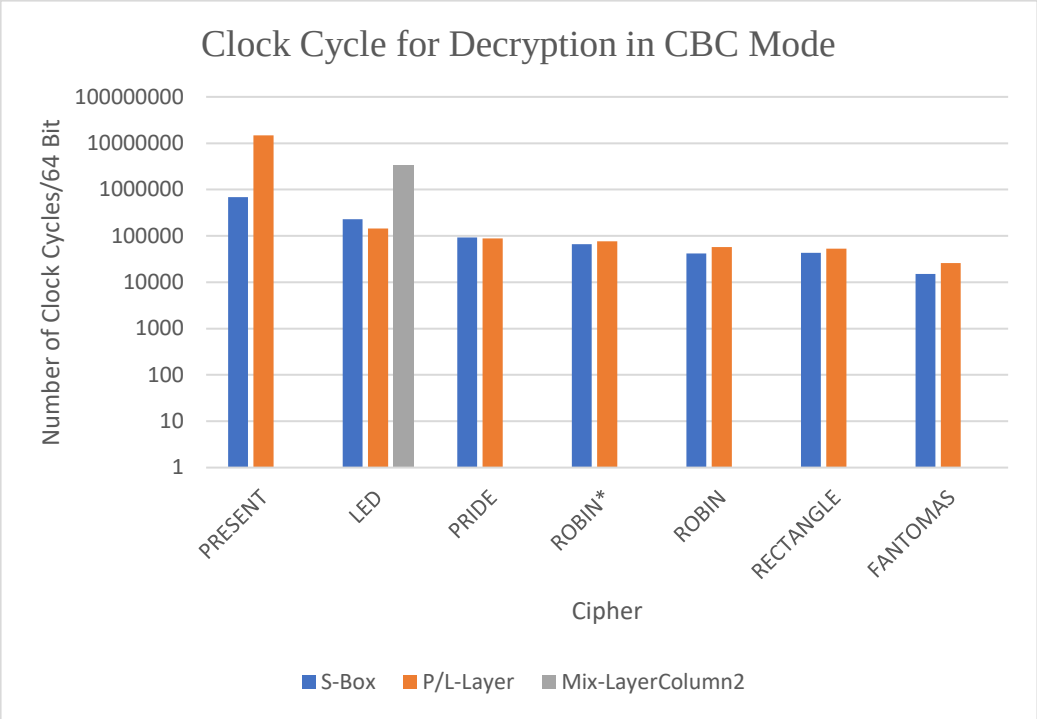


Figure 9:Clock Cycles for S/P-layer in Decryption

As we conclude from the given charts, the P/l layer of 7 SPN ciphers consumes more power than the S-layer as they require more processing. The results of S-box and P/L layer encryption and decryption for the given ciphers are similar except for the PRESENT and LED ciphers. Their P-layers need much more of a clock cycle to encrypt the 64-bit block size. The P-layer of PRESENT is 21 times higher than the S-box layer, and the P-layer of LED needs 1403% more clock cycles than the LED S-box. These inconsistent results of the PRESENT cipher make us modify the P-layer by replacing it with another P-layer that doesn't require many clock cycles.

The L-layer of FANTOMAS requires more clock cycles than the S-box. The S-box is based on bit-sliced and consists of simple XOR operations, while the L-layer is based on two large lookup tables; when accessing Lbox[i], it will index to the corresponding table and introduce some latencies compared to the S-box. PRIDE cipher consists of two layers, S-box and L-layer. The results of the clock cycles of each layer are

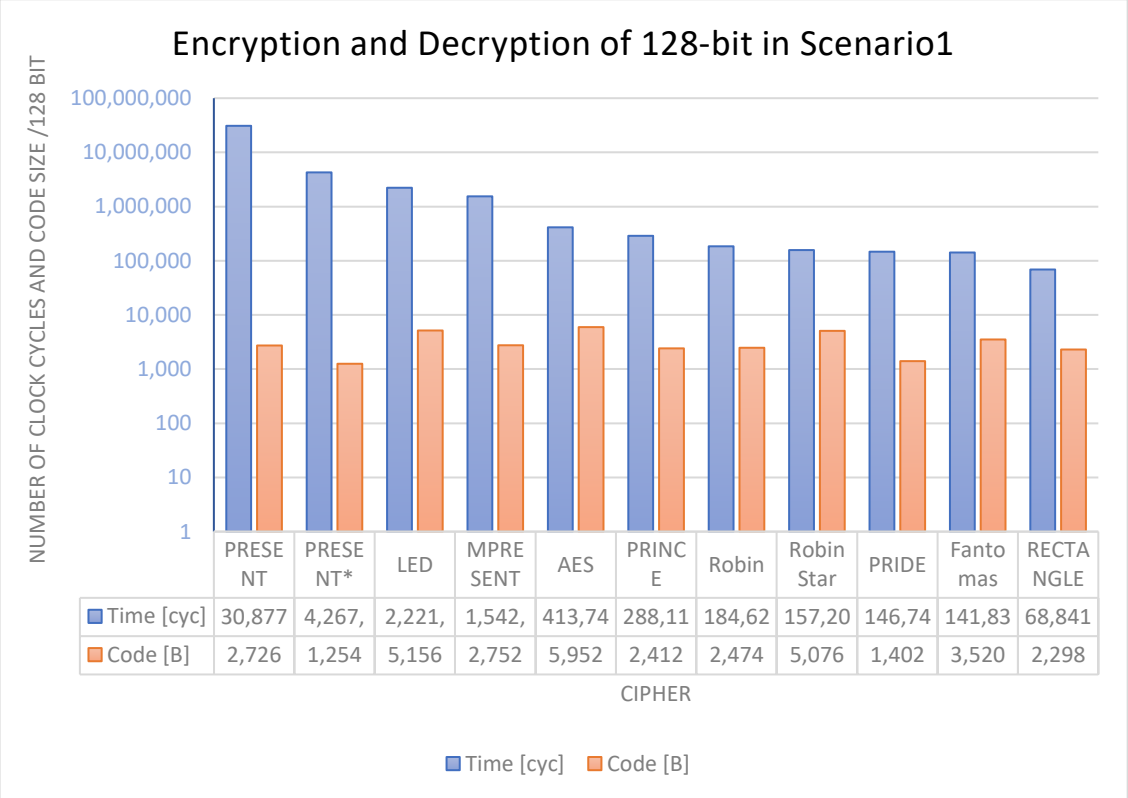
close to each other. The S-box layer shows slightly more than the L-layer regarding clock cycles. The P-layer consists of bitwise rotation, which makes it faster than the S-box. S-box has more bitwise operations than L_layer. The S-box consists of XOR and AND operations, making it consume more power than the P-layer. The number of bitwise operations in both the S-box and L-layer is almost similar. This explains the similarity of the results for the S and L layers. The S-box of the RECANGLE cipher consumes more clock cycles than the P-laye. The S-layer consists of bitwise XOR, AND, and OR operations, while the P-layer is a simple shifting operation that doesn't require high clock cycles. The permutation layer of Robin and Robin* consumes more clock cycles than the S-layer. Two lookup tables implemented a p-layer, each with a size of 256 bits, that requires access and addresses the memory.

LED cipher consists of three main layers. The mix layer requires a lot of clock cycles compared to other layers because of the complexity of the operation. The result of the PRESENT cipher shows that the P-layer increases 21 times compared to the S-layer. The P-layer iterates over each bit (bit by bit permutation) with arithmetic calculation to determine the new position of the specified bit. It repeats 64 times for each 32 rounds.

5.2 Results of Performance Metrics

After analyzing the results in section 5.1, we used the P-layer of the RECTANGLE cipher in our proposed MPRESENT as it shows low clock cycles for encryption and decryption 64-bit block size. RECTANGLE is unlike FANTOMAS and can easily implanted since it has the same block and key size as PRESENT. The P-layer of RECTANGLE design to fit well with 64 block size Figures 10 and 11 show the performance metrics for SPN cipher from (Dinu et al.) and (Sevin and Mohammed) along with our proposed MPRESENT.

Figure 10 shows that PRESENT needs a considerable number of clock cycles due to the permutation layer. As section 5.1 shows, our proposed MPRESENT consumes fewer clock cycles than the original PRESENT, LED cipher, and PRESENT* in terms of clock cycles. The results show that our proposed cipher does not surpass other SPN ciphers due to the consumption of S-box to clock cycles. It handles 4-bit at a time while the P-layer handles 16-bit with a low cost of arithmetic operation.



	PRESENT	PRESENT*	LED	MPRESENT	AES	PRINCE	Robin	Robin Star	PRIDE	Fantomas	RECTANGLE
Time [cyc]	30,877	4,267,	2,221,	1,542,	413,74	288,11	184,62	157,20	146,74	141,83	68,841
Code [B]	2,726	1,254	5,156	2,752	5,952	2,412	2,474	5,076	1,402	3,520	2,298

Figure 10 [1]*:Performance Metrics of SPN Ciphers in Scenario 1*

Figure 11 shows the clock cycles and the code size of implementing encryption of

[1] **Note: PRESENT* refers to PRESENT implemented by* (Sevin and Mohammed)

128-bit for eleven Lightweight ciphers in scenario 2. MPRESENT cipher shows better results than the PRESENT and RECTANGLE regarding Clock Cycle numbers. The elimination of the P-Layer of the original PRESENT in MPRESENT yields better performance in terms of clock cycle numbers. The code size of MPRESENT increases by 12-bytes since the source code is changed; in addition, the encryption and decryption codes have an extra circular shift layer, which leads to an increase in code size.

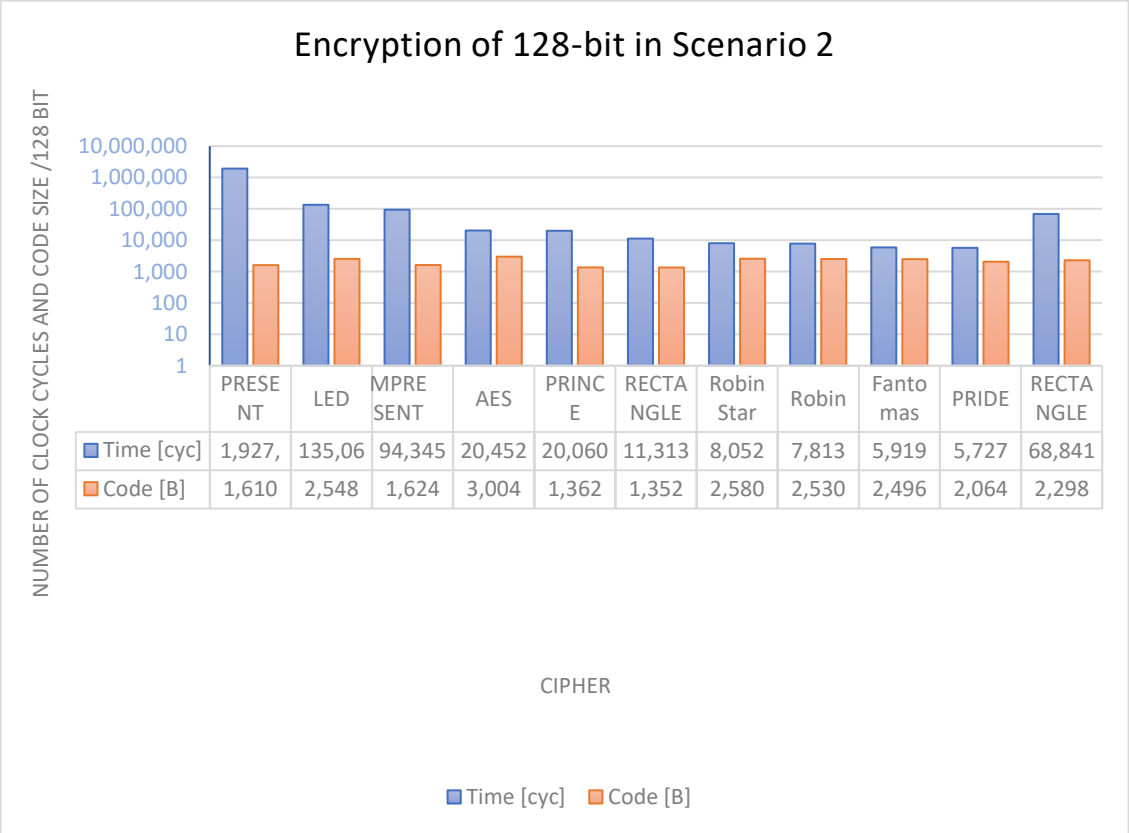


Figure 11:Performance Metrics of SPN Ciphers in Scenario 2

The code size of our Modified MPRESENT is close to the original PRESENT code size. It does not contain large lookup tables, and the result of the proposed MPRESENT surpasses AES, AESa, LED, Robin*, and Fantomas.

(Jangra and Singh) present an evaluation for PRESENT and CLEFIA ciphers by evaluating the number of clock cycles. Our performance metrics of the proposed

MPRESENT show it needs around 7.57×10^5 cycles for 64-bit, while the results of (Jangra and Singh) show the PRESENT requires 4.896×10^6 clock cycles, our implementation was done on AVR 8-bit microcontroller, at the variance of (Jangra and Singh). The implementation was done on Intel(R) core TM i7-7700@3.60, which is considered high-speed and made for PCs.

(Panahi et al.) show that PRESENT is the most cipher that needs execution time to encrypt and decrypt, which means it is the most cipher-consuming power. The result was done by testing ciphers on Raspberry Pi 3 with 1.2 Ghz and Arduino Mega 2560. The Panahi Pejman calculates the execution time in seconds, in contrast to our simulation, by simulating the cipher and evaluating the number of clock cycles. We drive the number of clock cycles by the following equations.

$$Clock\ Cycle\ Time\ = \frac{1}{Clock\ Frequency} \tag{6}$$

$$Clock\ Cycles\ (N) = \frac{Execution\ Time}{Clock\ Cycle\ Time} \tag{7}$$

Therefore, the number of clock cycles used to encrypt a 64-bit block of PRESENT by Raspberry Pi 3 is approximately 6×10^6 clock cycles, while our Modified PRESENT requires 7.57×10^5 clock cycles in AVR 8-bit.

(Lee et al.) analyze the power efficiency and the time required to encrypt AES and PRESENT using A15 and A7 processors. We calculate clock cycles using equations Equation 6 and Equation 7. A15 requires 2.25×10^6 clock cycles to encrypt 64-bit, Table IV shows the MPRESENT along with other ciphers.

Table IV:Clock Cycles to Encrypt 64-bit PRESENT Cipher

Cipher	Clock Cycles	Microcontroller/Processor
PRESENT by (Jangra and Singh)	4.896×10^6	Intel(R) core TM i7-7700@3.60
PRESENT by (Panahi et al.)	6×10^6	Raspberry Pi 3
PRESENT by (Lee et al.)	2.25×10^6	A15
MPRESENT	7.57×10^5	AVR 8-bit ATmega128

In conclusion, the result of the proposed MPRESENT cipher algorithm in the AVR 8-bit microcontroller shows better results over the original PRESENT in AVR, i7-7700, Raspberry Pi 3, and A15 microcontrollers in terms of clock cycles. The result of the code size of the proposed MPRESENT shows a similar result to the original PRESENT cipher. Enhancing the clock cycles does not directly affect the proposed MPRESENT code size metric.

5.3 Results of Security Metrics

Security metrics are essential to consider when designing or enhancing the performance of encryption algorithms. The two security metrics, diffusion, and confusion, were chosen for this research. Diffusion indicates the avalanche effect by ensuring that a slight change in the input results in a large change in the output. It assisted in defending multiple forms of cryptanalysis, including differential and linear cryptanalysis, which target attackers by looking for correlations or patterns in the bits. In contrast, the encryption function's non-linearity increases due to confusion. Attackers can take advantage of linearity; therefore, creating confusion helps. This increased the security of the encryption technique by making it more difficult for an attacker to predict the relationship between the input and output. Figure 12 shows the results of simulating 131 rounds of 6 SPN ciphers by representing the security metric for a data block of 64-bit.

PRINCE and PRIDE keys have a length of 128 bits, and their confusion simulated 65 rounds. Appendix B and C show the datasets.

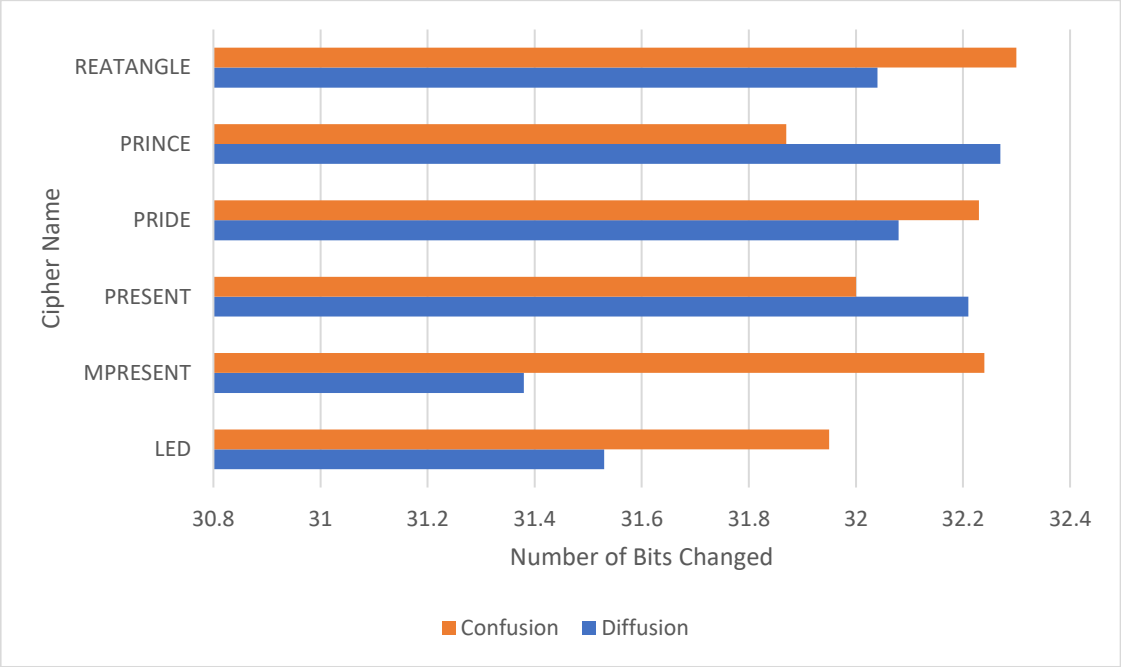


Figure 12: Avalanche Effect for SPN Ciphers

The results are close to each other. Changing one bit in the plain or key leads to significant changes in cipher output. The proposed MPRESENT changed 31.38 out 64 bit in diffusion for 131 datasets. However, the confusion of our proposed cipher kept the same value of confusion as the original PRESENT cipher. While our modification of the linear layer directly affected the diffusion since the P-layer is considered a diffusion layer and the original P-layer was replaced with RECTANGLE P-layer in addition to circular shift, diffusion is produced by permutation, which distributes an input bit's impact among several output bits. The confusion does not affect it because the nonlinear layer (S-box) was not modified. The proposed MPRESENT achieved an avalanche effect for both the data block and key. The proposed MPRESENT showed the result of diffusion had 31.38 out of 64-bit modified, while the original PRESENT achieved 32.21-bit changes over 64-

bit.

(Jangra and Singh) showed that the CLEFIA cipher had a 24-bit change over 64-bit. However, our proposed cipher changed to 31.38-bit over 64-bit. (Rana and Abul Kashem) Proposed a new key scheduling. Their proposed approach provides confusion of 36-bit change over 64-bit. The confusion of our proposed MPRESENT had a 32.21-bit change. Both ciphers had an avalanche effect on the confusion. (Rana et al.) showed a new lightweight cipher, the strength of security tested by diffusion. Changing one bit in the plain changes 49% of the outputs. However, our MPRESENT also changes 49%.

Chapter Six: Conclusions and Future Works

IoT has become widespread in recent days. It is integrated into various fields of medicine, science, military, and smart devices, and it connects with each other to exchange data. Traditional cryptography does not apply to IoT because of hardware limitations. The capacity of the power is a major challenge when applying an encryption algorithm to an IoT device. Many lightweight ciphers are introduced to handle the limitation. We analyzed each SPN layer of 7 ciphers.

The PRESENT cipher shows very high power consumption in C software implementation due to bit-level permutation. We simulate the performance results on the AVR 8-bit microcontroller using the FELISC framework, and the Modified cipher reduces the result 20 times compared to the original PRESENT. The proposed MPRESENT showed lower power consumption than PRESENT implemented by other researchers (Sevin and Mohammed; Jangra and Singh; Panahi et al.; Lee et al.) without significant effect on code size.

Enhancing power consumption can directly affect the avalanche of plain text since it is not a complicated permutation with a small number of arithmetic operations. However, MPRESENT still has the avalanche effect. While changing one bit in a key achieved 32 bits out of 64 bits, changing one bit in plain text achieved 49%. MPRESENT shows better results than (Jangra and Singh). Based on the foregoing, we can say the proposed MPRESENT is adequate with low and mid-level security requirements. Future work can focus on enhancing the S-box implementation on 8,16-and 32-bit microcontrollers with respect to side-channel attacks.

References

Albrecht, Martin R., et al. “Block Ciphers–Focus on the Linear Layer (Feat. PRIDE).” Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34, 2014, pp. 57–76.

AlDabbagh, Sufyan Salim Mahmood. “Design 32-Bit Lightweight Block Cipher Algorithm (DLBCA).” International Journal of Computer Applications, vol. 166, no. 8, May 2017, pp. 17–20, https://doi.org/10.5120/ijca2017914088.

Alfred, Menezes, et al. Handbook of Applied Cryptography. CRC press, 1997.

Alimzhanova, Zhanna, et al. “Analysis of Block Ciphers Characteristics for CBC and OFB Modes of Operation When Input Data Are Shifted.” 2022 International Conference on Smart Information Systems and Technologies (SIST), 2022, pp. 1–4.

Ankele Ralph and Banik, Subhadeep and Chakraborti Avik and List Eik and Mendel Florian and Sim Siang Meng and Wang Gaoli. “Related-Key Impossible-Differential Attack on Reduced-Round Skinny.” Applied Cryptography and Network Security, edited by Atsuko and Kikuchi Hiroaki Gollmann Dieter and Miyaji, Springer International Publishing, 2017, pp. 208–28.

Arm Developer. https://developer.arm.com/documentation. Accessed 27 Mar. 2023.

Asghari, Parvaneh, et al. “Internet of Things Applications: A Systematic Review.” Computer Networks, vol. 148, 2019, pp. 241–61.

Ashton, Kevin, and others. “That ‘Internet of Things’ Thing.” RFID Journal, vol. 22, no. 7, 2009, pp. 97–114.

Ashur, Tomer. “Improved Linear Trails for the Block Cipher Simon.” Cryptology EPrint Archive, 2015.

ATHENa. http://cryptography.gmu.edu/athena/. Accessed 27 Dec. 2022.

“ATmega128.” ATmega128, pp. 1–386, https://ww1.microchip.com/downloads/en/DeviceDoc/ doc2467.pdf. Accessed 16 May 2023.

Avrora - The AVR Simulation and Analysis Framework. http://compilers.cs.ucla.edu/avrora/. Accessed 20 June 2023.

Balasch, Josep, et al. “Compact Implementation and Performance Evaluation of Hash Functions in Attiny Devices.” Smart Card Research and Advanced Applications: 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers 11, 2013, pp. 158–72.

Banik, Subhadeep, et al. “GIFT: A Small Present: Towards Reaching the Limit of Lightweight Encryption.” Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, 2017, pp. 321–45.

Baysal Adnan and Şahin, Sühap. “RoadRunneR: A Small and Fast Bitslice Block Cipher for Low Cost 8-Bit Processors.” Lightweight Cryptography for Security and Privacy, edited by Gregor and Moradi Amir Güneysu Tim and Leander, Springer International Publishing, 2016, pp. 58–76.

Beaulieu, Ray, et al. “The SIMON and SPECK Families of Lightweight Block Ciphers.” Cryptology Eprint Archive, 2013.

Beierle Christof and Jean, Jérémy and Kölbl Stefan and Leander Gregor and Moradi Amir and Peyrin Thomas and Sasaki Yu and Sasdrich Pascal and Sim Siang Meng. “The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS.” Advances in Cryptology – CRYPTO 2016, edited by Jonathan Robshaw Matthew and Katz, Springer Berlin Heidelberg, 2016, pp. 123–53.

BLOC - Library. http://bloc.project.citi-lab.fr/library.html. Accessed 28 June 2023.

Bogdanov, Andrey, et al. “PRESENT: An Ultra-Lightweight Block Cipher.” Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9, 2007, pp. 450–66.

Borghoff, Julia, et al. “PRINCE–a Low-Latency Block Cipher for Pervasive Computing Applications.” Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18, 2012, pp. 208–25.

Cazorla, Mickaël, et al. “Survey and Benchmark of Lightweight Block Ciphers for Wireless Sensor Networks.” 2013 International Conference on Security and Cryptography (SECRYPT), 2013, pp. 1–6.

Chandramouli, Rajarathnam, et al. “Battery Power-Aware Encryption.” ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 2, 2006, pp. 162–80.

CryptoLUX > FELICS. https://www.cryptolux.org/index.php/FELICS. Accessed 27 Oct. 2023.

CryptoLUX > FELICS---. https://www.cryptolux.org/index.php/FELICS. Accessed 9 Dec. 2022.

Daily Mail Online. https://www.dailymail.co.uk/sciencetech/article-3385278/Never-run-food-Smart-mat-tells-low-milk-fridge-cam-shows-s-inside-shopping.html. Accessed 28 May 2023.

Dinu, Daniel, et al. “Triathlon of Lightweight Block Ciphers for the Internet of Things.” Journal of Cryptographic Engineering, vol. 9, 2019, pp. 283–302.

Dinu Daniel and Perrin, Léo and Udovenko Aleksei and Velichkov Vesselin and Großschädl Johann and Biryukov Alex. “Design Strategies for ARX with Provable

Bounds: Sparx and LAX.” Advances in Cryptology – ASIACRYPT 2016, edited by Tsuyoshi Cheon Jung Hee and Takagi, Springer Berlin Heidelberg, 2016, pp. 484–513.

Dworkin, Morris J. Advanced Encryption Standard (AES). 2023, https://doi.org/10.6028/NIST.FIPS.197-upd1.

EBACS. http://bench.cr.yp.to/. Accessed 12 Feb. 2023.

ECRYPT Lightweight Hash Functions. https://perso.uclouvain.be/fstandae/source_codes/hash_atmel/. Accessed 25 Feb. 2023.

Edwar, Jacinto G., et al. “Implementation of the Cryptographic Algorithm ‘Present’ in Different Microcontroller Type Embedded Software Platforms.” International Journal of Applied Engineering Research, vol. 12, no. 19, Research India Publications, 2017, pp. 8092–96.

Einspruch, Norman. Application Specific Integrated Circuit (ASIC) Technology. Academic Press, 2012.

Eisenbarth, Thomas, Sandeep Kumar, et al. “A Survey of Lightweight-Cryptography Implementations.” IEEE Design & Test of Computers, vol. 24, no. 6, 2007, pp. 522–33.

---. “A Survey of Lightweight-Cryptography Implementations.” IEEE Design & Test of Computers, vol. 24, no. 6, 2007, pp. 522–33.

Eisenbarth, Thomas, Zheng Gong, et al. “Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices.” Progress in Cryptology-AFRICACRYPT 2012: 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings 5, 2012, pp. 172–87.

ElGamal, Taher. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.” IEEE Transactions on Information Theory, vol. 31, no. 4, 1985, pp. 469–72.

Engler, Dawson R., et al. “C: A Language for High-Level, Efficient, and Machine-Independent Dynamic Code Generation.” Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1996, pp. 131–44.

Ferguson, Niels, et al. Cryptography Engineering: Design Principles and Practical Applications. John Wiley & Sons, 2011.

Gaj, Kris, et al. “Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs.” International Workshop on Cryptographic Hardware and Embedded Systems, 2010, pp. 264–78.

Ghorashi, Seyed Ramin, et al. Software Optimisation of Lightweight Klein Encryption in the Internet of Things. 2021.

Graham, S. L., et al. Programming Techniques A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. 1978.

Grosso, Vincent, et al. “LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations.” Fast Software Encryption: 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers 21, 2015, pp. 18–37.

Guo, Jian, et al. “The LED Block Cipher.” Cryptographic Hardware and Embedded Systems–CHES 2011: 13th International Workshop, Nara, Japan, September 28–October 1, 2011. Proceedings 13, 2011, pp. 326–41.

Hash Functions | CSRC. https://csrc.nist.gov/projects/hash-functions/sha-3-project. Accessed 2 Mar. 2023.

Hong, Deukjo, Jaechul Sung, et al. “HIGHT: A New Block Cipher Suitable for Low-Resource Device.” Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8, 2006, pp. 46–59.

Hong, Deukjo, Jung-Keun Lee, et al. “LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors.” Information Security Applications: 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers 14, 2014, pp. 3–27.

Hyde, Randall. The Art of Assembly Language. No Starch Press San Francisco, CA, USA, 2003.

IoT Cameras Can Be Remotely Hacked Security Researcher | ZDNET. https://www.zdnet.com/article/175000-iot-cameras-can-be-remotely-hacked-thanks-to-flaw-says-security-researcher/. Accessed 28 Jan. 2023.

Jangra, Monika, and Buddha Singh. “Performance Analysis of CLEFIA and PRESENT Lightweight Block Ciphers.” Journal of Discrete Mathematical Sciences and Cryptography, vol. 22, no. 8, 2019, pp. 1489–99.

Kahn, David. The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet. Simon and Schuster, 1996.

Kerckhof, Stéphanie, et al. “Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint.” Cryptographic Hardware and Embedded Systems–CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings 14, 2012, pp. 390–407.

Kerckhoffs, Auguste, and La Cryptographie Militaire ». “La Cryptographie Militaire (Seconde Partie).” Journal Des Sciences Militaires, vol. IX, pp. 161–91, http://petitcolas.net/kerckhoffs/crypto_militaire_2.pdfOriginalPDFat:http://petitcol

as.net/kerckhoffs/crypto_militaire_2.pdfhttp://petitcolas.net/kerckhoffs/crypto_militaire_2.pdf. Accessed 27 Mar. 2023.

Kilts, Steve. Advanced FPGA Design: Architecture, Implementation, and Optimization. John Wiley & Sons, 2007.

Kim, Jongsung, and Raphael C. W. Phan. “A Cryptanalytic View of the NSA’s Skipjack Block Cipher Design.” Advances in Information Security and Assurance: Third International Conference and Workshops, ISA 2009, Seoul, Korea, June 25-27, 2009. Proceedings 3, 2009, pp. 368–81.

Knežević, Miroslav, et al. “Low-Latency Encryption–Is ‘Lightweight= Light+ Wait’?” Cryptographic Hardware and Embedded Systems–CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings 14, 2012, pp. 426–46.

Law, Yee Wei, et al. “Survey and Benchmark of Block Ciphers for Wireless Sensor Networks.” ACM Transactions on Sensor Networks (TOSN), vol. 2, no. 1, 2006, pp. 65–93.

Lee, Wai Kong, et al. “Fast and Energy-Efficient Block Ciphers Implementations in ARM Processors and Mali GPU.” IETE Journal of Research, vol. 68, no. 4, 2022, pp. 2675–82.

Lightweight Cryptography | CSRC. https://csrc.nist.gov/Projects/Lightweight-Cryptography. Accessed 27 Mar. 2023.

Lightweight Cryptography Workshop 2015 | NIST. https://www.nist.gov/news-events/events/2015/07/lightweight-cryptography-workshop-2015. Accessed 27 Mar. 2023.

Lightweight Cryptography Workshop 2016 | NIST. https://www.nist.gov/news-events/events/2016/10/lightweight-cryptography-workshop-2016. Accessed 27 Mar. 2023.

Lim, Chae Hoon, and Tymur Korkishko. “MCrypton–a Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors.” International Workshop on Information Security Applications, 2005, pp. 243–58.

Matsui, Mitsuru, and Yumiko Murakami. “Minimalism of Software Implementation: Extensive Performance Analysis of Symmetric Primitives on the RL78 Microcontroller.” Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers 20, 2014, pp. 393–409.

Mohd, Bassam J., et al. “A Survey on Lightweight Block Ciphers for Low-Resource Devices: Comparative Study and Open Issues.” Journal of Network and Computer Applications, vol. 58, 2015, pp. 73–93.

Moon, Dukjae, et al. “Impossible Differential Cryptanalysis of Reduced Round XTEA and TEA.” Fast Software Encryption: 9th International Workshop, FSE 2002 Leuven, Belgium, February 4–6, 2002 Revised Papers 9, 2002, pp. 49–60.

Mouha, Nicky, et al. “Chaskey: An Efficient MAC Algorithm for 32-Bit Microcontrollers.” Selected Areas in Cryptography–SAC 2014: 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers 21, 2014, pp. 306–23.

MSP430F15. pp. 1–80, https://www.ti.com/lit/ds/symlink/msp430f1611.pdf. Accessed 1 Apr. 2023.

MSP430F16x, MSP430F161x MICROCONTROLLER. 2002, https://www.ti.com/lit/ds/symlink/msp430f1611.pdf.

MSPDebug. https://dlbeer.co.nz/mspdebug/. Accessed 1 Apr. 2023.

Navabi, Zainalabedin. VHDL: Analysis and Modeling of Digital Systems. McGraw-Hill, Inc., 1992.

ODROID. https://www.hardkernel.com/. Accessed 1 Apr. 2023.

Panahi, Pejman, et al. “Performance Evaluation of Lightweight Encryption Algorithms for IoT-Based Applications.” Arabian Journal for Science and Engineering, vol. 46, 2021, pp. 4015–37.

Patil, Jagdish, et al. “LiCi: A New Ultra-Lightweight Block Cipher.” 2017 International Conference on Emerging Trends & Innovation in ICT (ICEI), 2017, pp. 40–45.

Pliam, John O. “Guesswork and Variation Distance as Measures of Cipher Security.” International Workshop on Selected Areas in Cryptography, 1999, pp. 62–77.

Pookuzhy Ali, Mumthaz, et al. Optimised Design of Light Weight Block Cipher Lilliput with Extended Generalised Feistal Network (EGFN). 2017, https://doi.org/10.15680/IJIRSET.2017.0604072.

Preneel, Bart. Fast Software Encryption: Second International Workshop, Leuven, Belgium, December 14-16, 1994. Proceedings. Springer Science & Business Media, 1995.

Rana, Sohel, et al. “An Effective Lightweight Cryptographic Algorithm to Secure Resource-Constrained Devices.” International Journal of Advanced Computer Science and Applications, vol. 9, no. 11, 2018.

Rana, Sohel, and Mohammod Abul Kashem. “A Modified Split-Radix Architecture-Based Key Scheduling Technique for Lightweight Block Ciphers.” International Journal Of Computing and Digital System, 2021, pp. 1327–35.

Sehrawat, Deepti, and Nasib Singh Gill. “Lightweight Block Ciphers for IoT Based Applications: A Review.” International Journal of Applied Engineering Research, vol. 13, no. 5, 2018, pp. 2258–70.

Sevin, Abdullah, and Abdu Ahmed Osman Mohammed. “A Survey on Software Implementation of Lightweight Block Ciphers for IoT Devices.” Journal of Ambient Intelligence and Humanized Computing, 2021, pp. 1–15.

Shannon, Claude E. “Communication Theory of Secrecy Systems.” The Bell System Technical Journal, vol. 28, no. 4, 1949, pp. 656–715.

Shibutani, Kyoji, et al. “Piccolo: An Ultra-Lightweight Blockcipher.” Cryptographic Hardware and Embedded Systems–CHES 2011: 13th International Workshop, Nara, Japan, September 28–October 1, 2011. Proceedings 13, 2011, pp. 342–57.

Shirai, Taizo, et al. “The 128-Bit Blockcipher CLEFIA.” Fast Software Encryption: 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers 14, 2007, pp. 181–95.

Stajano, Frank. Security for Ubiquitous Computing. Wiley Chichester, 2002.

Suzaki, Tomoyasu, et al. “Twine: A Lightweight, Versatile Block Cipher.” ECRYPT Workshop on Lightweight Cryptography, vol. 2011, 2011.

Titzer, Ben L., et al. “Avrora: Scalable Sensor Network Simulation with Precise Timing.” IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005., 2005, pp. 477–82.

Usman, Muhammad, et al. “SIT: A Lightweight Encryption Algorithm for Secure Internet of Things.” ArXiv Preprint ArXiv:1704.08688, 2017.

Vaudenay, Serge. A Classical Introduction to Cryptography: Applications for Communications Security. Springer Science & Business Media, 2005.

Wang, Xiaoyun, and Kazue Sako. Advances in Cryptology–ASiACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012, Proceedings. Springer Science & Business Media, 2012.

Wenzel-Benner, Christian, and Jens Gräf. “XBX: EXternal Benchmarking EXtension for the SUPERCOP Crypto Benchmarking Framework.” Cryptographic Hardware and Embedded Systems, CHES 2010: 12th International Workshop, Santa Barbara, USA, August 17-20, 2010. Proceedings 12, 2010, pp. 294–305.

Wu, Wenling, and Lei Zhang. “LBlock: A Lightweight Block Cipher.” Applied Cryptography and Network Security: 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings 9, 2011, pp. 327–44.

XBX Embedded Hashing. https://github.com/das-labor/xbx/. Accessed 27 Apr. 2023.

Xing, Biao, et al. “Accelerating DES and AES Algorithms for a Heterogeneous Many-Core Processor.” International Journal of Parallel Programming, vol. 49, 2021, pp. 463–86.

Zhang, Wentao, et al. “RECTANGLE: A Bit-Slice Lightweight Block Cipher Suitable for Multiple Platforms.” Cryptology EPrint Archive, 2014.

Appendices

Appendix A: Encryption Code of MPRESENT

```
#include <stdint.h>
#include "cipher.h"
#include "constants.h"
#include "rotate.h"
#include "rot.h"

void P_layer(uint16_t *data) {
data[1] = ROTL1(data[1]);
data[2] = ROTL12(data[2]);
data[3] = ROTL13(data[3]);
}
uint64_t shift(uint64_t value) {
return ((value << 12) | (value >> 52));
}
void Encrypt(uint8_t *block, uint8_t *roundKeys) {
uint64_t state = *(uint64_t*)block;
uint8_t round, k;
for (round = 0; round < 31; round++) {
        uint32_t subkey_lo =
READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[2 * round]);
        uint32_t subkey_hi =
READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[2 * round + 1]);

        state ^= (uint64_t)subkey_lo ^ (((uint64_t)subkey_hi) << 32);
        for (k = 0; k < 16; k++) {
        uint16_t sBoxValue = state & 0xF;
        state &= 0xFFFFFFFFFFFFFFF0;
        state |= READ_SBOX_BYTE(sBox4[sBoxValue]);
        state = rotate4l_64(state);
        }
        P_layer((uint16_t*)&state);
        uint64_t shiftedState = shift(state);
        state = shiftedState;
        }
uint32_t subkey_lo =
READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[62]);
uint32_t subkey_hi =
READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[63]);


state ^= (uint64_t)subkey_lo ^ (((uint64_t)subkey_hi) << 32);
*(uint64_t*)block = state;
}
```

Appendix B: Data for 64-bit input block

DATA	One Bit Change
0x54,0x65,0x78,0x74,0x20,0x6D,0x65,0x73	0x54,0x65,0x78,0x7c,0x20,0x6D,0x65,0x73
0x6F,0x72,0x20,0x74,0x65,0x78,0x74,0x69	0x6F,0x72,0x20,0x7c,0x65,0x78,0x74,0x69
0x6E,0x67,0x2C,0x20,0x69,0x73,0x20,0x74	0x6F,0x67,0x2C,0x20,0x69,0x73,0x20,0x74
0x68,0x65,0x20,0x61,0x63,0x74,0x20,0x6F	0x69,0x65,0x20,0x61,0x63,0x74,0x20,0x6F
0x66,0x20,0x63,0x6F,0x6D,0x70,0x6F,0x73	0x66,0x20,0x62,0x6F,0x6D,0x70,0x6F,0x73
0x69,0x6E,0x67,0x20,0x61,0x6E,0x64,0x20	0x69,0x6E,0x67,0x20,0x61,0x6E,0x64,0x21
0x73,0x65,0x6E,0x64,0x69,0x6E,0x67,0x20	0x73,0x64,0x6E,0x64,0x69,0x6E,0x67,0x20
0x69,0x63,0x20,0x6D,0x65,0x73,0x73,0x61	0x69,0x63,0x20,0x6F,0x65,0x73,0x73,0x6
0x67,0x65,0x73,0x2C,0x20,0x74,0x79,0x70	0x67,0x65,0x73,0x3C,0x20,0x74,0x79,0x70
0x69,0x63,0x61,0x6C,0x6C,0x79,0x20,0x63	0x69,0x63,0x61,0x6C,0x6C,0x79,0x22,0x63
0x6F,0x6E,0x73,0x69,0x73,0x74,0x69,0x6E	0x6D,0x6E,0x73,0x69,0x73,0x74,0x69,0x6E
0x68,0x61,0x62,0x65,0x74,0x69,0x63,0x20	0x60,0x61,0x62,0x65,0x74,0x69,0x63,0x20
0x61,0x6E,0x64,0x20,0x6E,0x75,0x6D,0x65	0x61,0x6E,0x64,0x20,0x6C,0x75,0x6D,0x65
0x72,0x69,0x63,0x20,0x63,0x68,0x61,0x72	0x72,0x68,0x63,0x20,0x63,0x68,0x61,0x72
0x61,0x63,0x74,0x65,0x72,0x73,0x2C,0x20	0x61,0x23,0x74,0x65,0x72,0x73,0x2C,0x20
0x62,0x65,0x74,0x77,0x65,0x65,0x6E,0x20	0x62,0x65,0x74,0x77,0x65,0x65,0x6A,0x20
0x74,0x77,0x6F,0x20,0x6F,0x72,0x20,0x6D	0x74,0x77,0x6F,0x21,0x6F,0x72,0x20,0x6D
0x73,0x20,0x6F,0x66,0x20,0x6D,0x6F,0x62	0x73,0x20,0x6F,0x26,0x20,0x6D,0x6F,0x62
0x69,0x6C,0x65,0x20,0x64,0x65,0x76,0x69	0x69,0x6C,0x65,0x21,0x64,0x65,0x76,0x69
0x63,0x65,0x73,0x2C,0x20,0x64,0x65,0x73	0x63,0x65,0x73,0x2C,0x22,0x64,0x65,0x73
0x6B,0x74,0x6F,0x70,0x73,0x2F,0x6C,0x61	0x6B,0x74,0x6F,0x70,0x7B,0x2F,0x6C,0x61
0x72,0x20,0x61,0x6E,0x6F,0x74,0x68,0x65	0x72,0x20,0x61,0x6E,0x6E,0x74,0x68,0x65
0x66,0x20,0x63,0x6F,0x6D,0x70,0x61,0x74	0x66,0x20,0x63,0x6F,0x6D,0x70,0x61,0x70
0x69,0x62,0x6C,0x65,0x20,0x63,0x6F,0x6D	0x69,0x62,0x68,0x65,0x20,0x63,0x6F,0x6D
0x70,0x75,0x74,0x65,0x72,0x2E,0x20,0x54	0x70,0x75,0x74,0x65,0x72,0x2E,0x22,0x54
0x65,0x78,0x74,0x20,0x6D,0x65,0x73,0x73	0x65,0x79,0x74,0x20,0x6D,0x65,0x73,0x73
0x61,0x67,0x65,0x73,0x20,0x6D,0x61,0x79	0x60,0x67,0x65,0x73,0x20,0x6D,0x61,0x79
0x20,0x62,0x65,0x20,0x73,0x65,0x6E,0x74	0x20,0x62,0x65,0x20,0x72,0x65,0x6E,0x74
0x20,0x6F,0x76,0x65,0x72,0x20,0x61,0x20	0x20,0x6D,0x76,0x65,0x72,0x20,0x61,0x20
0x63,0x65,0x6C,0x6C,0x75,0x6C,0x61,0x72	0x63,0x65,0x6C,0x6C,0x7D,0x6C,0x61,0x72
0x4A,0x7F,0xE2,0x91,0x3D,0xB6,0x58,0x0A	0x4A,0x7F,0xE2,0x91,0x3D,0xB6,0x58,0x0B
0x8C,0xD4,0x2F,0x5E,0x70,0x9B,0x13,0xA6	0x8C,0xD4,0x2F,0x5E,0x70,0x9B,0x13,0xA7
0x26,0x18,0xEC,0x7F,0x49,0x05,0xB2,0xD8	0x26,0x18,0xEC,0x7F,0x49,0x05,0xB2,0xD9
0xF3,0x81,0xA7,0x6D,0x2C,0x40,0x9E,0x5B	0xF3,0x81,0xA7,0x6D,0x2C,0xC0,0x9E,0x5B
0x7A,0x3E,0x59,0xC1,0x20,0x8D,0xF7,0x64	0x7A,0x3E,0x59,0xC1,0x20,0x8D,0xF7,0x65
0xB9,0x1F,0x46,0x35,0xD2,0x87,0xAE,0x02	0xB9,0x1F,0x46,0x35,0xD2,0x87,0xAE,0x03
0xE4,0xC7,0x10,0x9A,0x53,0xF8,0x6B,0x24	0xE4,0xC7,0x10,0x9A,0x53,0xF8,0x6B,0x25
0x68,0xDB,0x8E,0x15,0xA0,0x7C,0x34,0xF9	0x68,0xDB,0x8E,0x15,0xA0,0x7C,0x34,0xF8
0x97,0x2D,0x5A,0x0B,0xE8,0x14,0xFC,0x36	0x97,0x2D,0x5A,0x0B,0xA8,0x14,0xFC,0x36

0x50,0x1C,0xB3,0x76,0xCA,0x29,0x8F,0xE0	0x50,0x1C,0xB3,0x76,0xCA,0x29,0x8D,0xE0
0x6E,0xA5,0x38,0xFD,0x80,0xD7,0x0C,0x94	0x6E,0xA5,0x38,0xBD,0x80,0xD7,0x0C,0x94
0xC2,0xF6,0x73,0x22,0x5D,0x09,0xB8,0x4F	0xC2,0xB6,0x73,0x22,0x5D,0x09,0xB8,0x4F
0xAF,0x72,0x01,0xCD,0x98,0x54,0x2B,0xE6	0xAF,0x72,0x01,0xCD,0x98,0x55,0x2B,0xE6
0x31,0x6A,0xD5,0x88,0x07,0xBB,0x42,0xFC	0x31,0x6A,0xD5,0x88,0x07,0xBB,0x43,0xFC
0x4D,0x83,0xEE,0x19,0xB6,0x50,0xA2,0x0F	0x4D,0x83,0xEE,0x19,0xB6,0x50,0xA0,0x0F
0x9F,0x24,0x6B,0xC8,0x15,0xFA,0x37,0xD0	0x9F,0x24,0x6A,0xC8,0x15,0xFA,0x37,0xD0
0x62,0x3C,0x7F,0x58,0xA4,0xE9,0x01,0xD6	0x62,0x3C,0x7F,0x58,0xA4,0xE9,0x00,0xD6
0x8B,0xF0,0x44,0x9E,0x3D,0x70,0xC2,0xA6	0x8B,0xF0,0x44,0x9E,0x3D,0x71,0xC2,0xA6
0x11,0x6D,0xB8,0x5F,0xE0,0x23,0xCA,0x97	0x11,0x6D,0xB8,0x5F,0xE0,0x23,0xCB,0x97
0x75,0xA9,0x30,0xFE,0x46,0x8B,0xD2,0x0C	0x75,0xA9,0x30,0xFE,0x46,0xAB,0xD2,0x0C
0xDB,0x16,0xC9,0x5E,0xA3,0x7F,0x20,0x84	0xDB,0x16,0xC9,0x5E,0xAB,0x7F,0x20,0x84
0x5F,0x82,0x3A,0xE7,0x48,0x0D,0xB4,0x69	0x5F,0x82,0x3A,0xE7,0xC8,0x0D,0xB4,0x69
0x9C,0x42,0xFD,0x35,0xBA,0x67,0x1E,0x80	0x9C,0x42,0xFC,0x35,0xBA,0x67,0x1E,0x80
0xE2,0xAF,0x71,0x0A,0xD5,0x28,0xFC,0x43	0xE2,0xAF,0x75,0x0A,0xD5,0x28,0xFC,0x43
0x3E,0x50,0x9D,0x06,0xB3,0x7C,0xA1,0xEF	0x3E,0x50,0x9D,0x06,0xB2,0x7C,0xA1,0xEF
0x89,0xC7,0x2A,0xF4,0x51,0x1E,0xD0,0xB3	0x89,0xC7,0x2A,0xF4,0x51,0x1E,0xD0,0xF3
0x27,0x79,0xB4,0x63,0xDC,0x01,0xAE,0xE8	0x27,0x79,0xB4,0x63,0xDC,0x01,0xAE,0xE0
0xF1,0x2C,0x98,0x43,0xAE,0x65,0x37,0xD0	0xD1,0x2C,0x98,0x43,0xAE,0x65,0x37,0xD0
0x5A,0x84,0xD1,0x3E,0xE9,0x47,0x2F,0xB0	0x5A,0x80,0xD1,0x3E,0xE9,0x47,0x2F,0xB0
0x76,0xBB,0x20,0x8D,0xC1,0x94,0x5E,0x3A	0x76,0x9B,0x20,0x8D,0xC1,0x94,0x5E,0x3A
0xCA,0x15,0x68,0xB7,0x02,0xDF,0x94,0x3E	0xCA,0x15,0x68,0xB7,0x03,0xDF,0x94,0x3E
0x0F,0x53,0x8E,0x21,0xD4,0x7B,0xA6,0xC0	0x0F,0x51,0x8E,0x21,0xD4,0x7B,0xA6,0xC0
0x4B,0x96,0x21,0xEC,0x37,0xD8,0x0F,0xA5	0x4B,0x96,0x21,0xEC,0x37,0xDC,0x0F,0xA5
0x83,0xDE,0x57,0x2A,0xB1,0x0C,0xF4,0x69	0x93,0xDE,0x57,0x2A,0xB1,0x0C,0xF4,0x69
0x2D,0x71,0xAC,0x03,0xC6,0x59,0x94,0xE8	0x2D,0x71,0xAC,0x03,0xCE,0x59,0x94,0xE8
0xE9,0x34,0xFB,0x46,0x81,0xCC,0x17,0xAD	0xF9,0x34,0xFB,0x46,0x81,0xCC,0x17,0xAD
0x40,0x7E,0xA3,0x1C,0xD8,0x52,0x9F,0x6B	0x40,0x7E,0xA3,0x3C,0xD8,0x52,0x9F,0x6B
0x74,0xC8,0x25,0xFA,0x5B,0x90,0x0D,0xE6	0x54,0xC8,0x25,0xFA,0x5B,0x90,0x0D,0xE6
0xAE,0x61,0x8C,0x30,0xF5,0x4B,0x97,0x02	0xAE,0x61,0x8C,0x30,0xF5,0x4B,0x97,0x22
0x19,0x55,0x8A,0xF1,0x3E,0x60,0xBD,0xC7	0x19,0x55,0x8A,0xF1,0x3E,0x60,0xBD,0xE7
0x45,0x98,0x2F,0xE3,0x78,0xA1,0xD4,0x0C	0x45,0x98,0x2F,0xE3,0x78,0xA1,0xD4,0x04
0x91,0xDC,0x01,0x7B,0x56,0x2A,0xC3,0xFE	0x91,0xDC,0x01,0x7B,0x56,0x2A,0xC3,0xFA
0x2B,0x76,0xC9,0x14,0xA0,0x5D,0xE3,0x80	0x23,0x76,0xC9,0x14,0xA0,0x5D,0xE3,0x80
0xF6,0x38,0x87,0xDA,0x14,0x61,0xBC,0x59	0xF6,0x38,0x87,0xDA,0x14,0x61,0xAC,0x59
0x52,0x9D,0xE0,0x3F,0xA7,0x6A,0x14,0xC8	0x52,0x9D,0xE0,0x3F,0xA7,0x6A,0x10,0xC8
0x8A,0xD4,0x01,0x5F,0x30,0x6E,0xB2,0xC7	0x8A,0xF4,0x01,0x5F,0x30,0x6E,0xB2,0xC7
0xC5,0x20,0x9F,0x74,0xDB,0x01,0xA8,0x3E	0xC5,0x20,0x9F,0x70,0xDB,0x01,0xA8,0x3E
0x10,0x4E,0x93,0x65,0xB8,0x2B,0xD7,0xFA	0x10,0x4E,0x93,0x65,0xB8,0x2B,0xD7,0xFE
0x3B,0x84,0xC1,0x5E,0xA9,0x70,0xFE,0x26	0x3B,0x84,0xC3,0x5E,0xA9,0x70,0xFE,0x26
0x67,0xBA,0x0F,0x52,0x8D,0xE1,0x34,0xC8	0x67,0xBA,0x0F,0x52,0x85,0xE1,0x34,0xC8
0x95,0xD9,0x24,0x6F,0xB0,0x8A,0x1E,0xC7	0x95,0xD1,0x24,0x6F,0xB0,0x8A,0x1E,0xC7
0xDF,0x21,0x7C,0xA9,0x54,0x0E,0xB3,0x68	0xDF,0x21,0x7C,0xA9,0x54,0x0E,0xB3,0x6A

0x0A,0x56,0x89,0xD3,0x2E,0xF1,0xB7,0x4C	0x0A,0x56,0x09,0xD3,0x2E,0xF1,0xB7,0x4C
0x47,0x9B,0xC4,0x2F,0xE1,0x1A,0xB0,0x56	0x47,0x9B,0xC4,0x0F,0xE1,0x1A,0xB0,0x56
0x83,0xDC,0x29,0x74,0xB9,0x01,0xF6,0x5E	0x83,0xDC,0x29,0xF4,0xB9,0x01,0xF6,0x5E
0x2F,0x65,0xB8,0x13,0xCE,0x04,0x97,0xDA	0x2F,0x65,0xB8,0x12,0xCE,0x04,0x97,0xDA
0xEA,0x25,0x9B,0x47,0x72,0xDF,0x06,0xC8	0xEA,0x25,0x9B,0x47,0x72,0xDF,0x06,0x48
0xBC,0xF1,0x4E,0x83,0x1A,0xD7,0x60,0x2F	0xBC,0xF1,0x4E,0x83,0x1B,0xD7,0x60,0x2F
0x6D,0xA1,0x34,0xC8,0x1F,0xF6,0x5B,0x80	0x6D,0xA1,0x34,0xC8,0x1F,0xF6,0xDB,0x80
0xAB,0x16,0xC0,0x7F,0x58,0x82,0xD4,0x29	0xAB,0x16,0xC0,0x7F,0x58,0x82,0x94,0x29
0xF0,0x2A,0x9D,0x57,0x8B,0xC4,0x11,0xE6	0xF0,0x2A,0x9D,0x55,0x8B,0xC4,0x11,0xE6
0x45,0x7E,0xB3,0x08,0xD1,0x6C,0xA9,0x20	0x45,0x7E,0xB3,0x08,0xD1,0xEC,0xA9,0x20
0x90,0xDC,0x15,0x6A,0xBF,0x40,0xE7,0x82	0x90,0xD8,0x15,0x6A,0xBF,0x40,0xE7,0x82
0x3D,0x61,0xBE,0x24,0xC8,0x97,0x50,0xAF	0x3D,0x61,0xBE,0x24,0xD8,0x97,0x50,0xAF
0x88,0xD3,0x0E,0x72,0x5F,0xA1,0xC4,0xB9	0x88,0xD3,0x0E,0x72,0x5F,0xA1,0xC6,0xB9
0x2A,0x74,0xDF,0x01,0x95,0x6B,0xC8,0x30	0x2A,0x74,0xDF,0x03,0x95,0x6B,0xC8,0x30
0xE5,0x38,0x8B,0xD4,0x01,0x76,0xC2,0xAF	0xE5,0x28,0x8B,0xD4,0x01,0x76,0xC2,0xAF
0x0C,0x53,0x9E,0x20,0xD7,0x68,0xB1,0xFC	0x0C,0x53,0x1E,0x20,0xD7,0x68,0xB1,0xFC
0x41,0x85,0xF0,0x3E,0xBA,0x56,0x29,0xD4	0x41,0x85,0xF0,0x3E,0xBA,0x52,0x29,0xD4
0x95,0xCA,0x17,0xF3,0x4E,0x80,0xDB,0x62	0x95,0xCA,0x17,0xF3,0x4E,0x80,0xFB,0x62
0x1a,0x2b,0x3c,0x4d,0x5e,0x6f,0x80,0x91	0x1A,0x2B,0x3C,0x45,0x5E,0x6F,0x80,0x91
0xa2,0xb3,0xc4,0xd5,0xe6,0xf7,0x08,0x19	0xA2,0xB3,0xC4,0xD5,0xE6,0xF7,0x09,0x19
0x2a,0x3b,0x4c,0x5d,0x6e,0x7f,0x90,0xa1	0x2A,0x3B,0x4C,0x5D,0x6A,0x7F,0x90,0xA1
0xb2,0xc3,0xd4,0xe5,0xf6,0x07,0x18,0x29	0xB2,0xC3,0xD4,0xE5,0xF6,0x87,0x18,0x29
0x3a,0x4b,0x5c,0x6d,0x7e,0x8f,0xa0,0xb1	0x3E,0x4B,0x5C,0x6D,0x7E,0x8F,0xA0,0xB1
0xc2,0xd3,0xe4,0xf5,0x06,0x17,0x28,0x39	0xC2,0xD3,0x64,0xF5,0x06,0x17,0x28,0x39
0x4a,0x5b,0x6c,0x7d,0x8e,0x9f,0xb0,0xc1	0x4A,0x5B,0x6C,0x7D,0x8E,0x9F,0xB0,0xC0
0xd2,0xe3,0xf4,0x05,0x16,0x27,0x38,0x49	0xD2,0xE3,0xFC,0x05,0x16,0x27,0x38,0x49
0x5a,0x6b,0x7c,0x8d,0x9e,0xaf,0xc0,0xd1	0x5A,0x6B,0x7C,0xCD,0x9E,0xAF,0xC0,0xD1
0xe2,0xf3,0x04,0x15,0x26,0x37,0x48,0x59	0xE2,0xF3,0x44,0x15,0x26,0x37,0x48,0x59
0x6a,0x7b,0x8c,0x9d,0xae,0xbf,0xd0,0xe1	0x6A,0x3B,0x8C,0x9D,0xAE,0xBF,0xD0,0xE1
0xf2,0x03,0x14,0x25,0x36,0x47,0x58,0x69	0xF2,0x03,0x14,0x25,0xB6,0x47,0x58,0x69
0x7a,0x8b,0x9c,0xad,0xbe,0xcf,0xe0,0xf1	0x72,0x8B,0x9C,0xAD,0xBE,0xCF,0xE0,0xF1
0x02,0x13,0x24,0x35,0x46,0x57,0x68,0x79	0x02,0x13,0xA4,0x35,0x46,0x57,0x68,0x79
0x8a,0x9b,0xac,0xbd,0xce,0xdf,0xf0,0x01	0x8A,0x8B,0xAC,0xBD,0xCE,0xDF,0xF0,0x01
0x12,0x23,0x34,0x45,0x56,0x67,0x78,0x89	0x12,0x23,0x34,0x45,0x56,0x67,0x78,0xA9
0x9a,0xab,0xbc,0xcd,0xde,0xef,0x00,0x11	0x98,0xAB,0xBC,0xCD,0xDE,0xEF,0x00,0x11
0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99	0x32,0x33,0x44,0x55,0x66,0x77,0x88,0x99
0xaa,0xbb,0xcc,0xdd,0xee,0xff,0x10,0x21	0xAA,0xBB,0xCC,0xDD,0xEC,0xFF,0x10,0x21
0x32,0x43,0x54,0x65,0x76,0x87,0x98,0xa9	0x32,0x43,0x54,0xE5,0x76,0x87,0x98,0xA9
0xba,0xcb,0xdc,0xed,0xfe,0x0f,0x20,0x31	0xBA,0xCB,0xDD,0xED,0xFE,0x0F,0x20,0x31
0x42,0x53,0x64,0x75,0x86,0x97,0xa8,0xb9	0x42,0x53,0x44,0x75,0x86,0x97,0xA8,0xB9
0xca,0xdb,0xec,0xfd,0x0e,0x1f,0x30,0x41	0xCA,0xDB,0xEC,0xFF,0x0E,0x1F,0x30,0x41
0x52,0x63,0x74,0x85,0x96,0xa7,0xb8,0xc9	0x52,0x23,0x74,0x85,0x96,0xA7,0xB8,0xC9
0xda,0xeb,0xfc,0x0d,0x1e,0x2f,0x40,0x51	0xDA,0xEB,0xEC,0x0D,0x1E,0x2F,0x40,0x51

0x62,0x73,0x84,0x95,0xa6,0xb7,0xc8,0xd9	0x6A,0x73,0x84,0x95,0xA6,0xB7,0xC8,0xD9
0xea,0xfb,0x0c,0x1d,0x2e,0x3f,0x50,0x61	0xEA,0xFB,0x0C,0x1D,0x2E,0x3F,0x50,0x69
0x72,0x83,0x94,0xa5,0xb6,0xc7,0xd8,0xe9	0x72,0x83,0x94,0xA5,0xB6,0xC7,0xD8,0xE8
0xfa,0x0b,0x1c,0x2d,0x3e,0x4f,0x60,0x71	0xFA,0x0B,0x3C,0x2D,0x3E,0x4F,0x60,0x71
0x82,0x93,0xa4,0xb5,0xc6,0xd7,0xe8,0xf9	0x82,0x93,0xA4,0xB5,0xC6,0xD7,0xF8,0xF9
0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79	0x70,0x73,0x74,0x75,0x76,0x77,0x78,0x79

Appendix C: Data for 128-bit input block

Input	One bit Change
0x6F,0x72,0x20,0x74,0x65,0x78,0x74,0x69, 0x6E,0x67,0x2C,0x20,0x69,0x73,0x20,0x74	0x6F,0x72,0x20,0x74,0x65,0x78,0x74,0x69, 0x6F,0x67,0x2C,0x20,0x69,0x73,0x20,0x74
0x68,0x65,0x20,0x61,0x63,0x74,0x20,0x6F, 0x66,0x20,0x63,0x6F,0x6D,0x70,0x6F,0x73	0x68,0x65,0x20,0x61,0x63,0x74,0x20,0x6F, 0x66,0x20,0x62,0x6F,0x6D,0x70,0x6F,0x73
0x69,0x6E,0x67,0x20,0x61,0x6E,0x64,0x20, 0x73,0x65,0x6E,0x64,0x69,0x6E,0x67,0x20	0x69,0x6E,0x67,0x20,0x61,0x6E,0x64,0x20, 0x73,0x64,0x6E,0x64,0x69,0x6E,0x67,0x20
0x65,0x6C,0x65,0x63,0x74,0x72,0x6F,0x6E, 0x69,0x63,0x20,0x6D,0x65,0x73,0x73,0x61	0x65,0x6C,0x65,0x63,0x74,0x72,0x6F,0x6E, 0x69,0x63,0x20,0x6F,0x65,0x73,0x73,0x61
0x67,0x65,0x73,0x2C,0x20,0x74,0x79,0x70, 0x69,0x63,0x61,0x6C,0x6C,0x79,0x20,0x63	0x67,0x65,0x73,0x2C,0x20,0x74,0x79,0x70, 0x69,0x63,0x61,0x6C,0x6C,0x79,0x22,0x63
0x6F,0x6E,0x73,0x69,0x73,0x74,0x69,0x6E, 0x67,0x20,0x6F,0x66,0x20,0x61,0x6C,0x70	0x6F,0x6E,0x73,0x69,0x73,0x74,0x69,0x6E, 0x67,0x20,0x6F,0x66,0x20,0x60,0x6C,0x70
0x68,0x61,0x62,0x65,0x74,0x69,0x63,0x20, 0x61,0x6E,0x64,0x20,0x6E,0x75,0x6D,0x65	0x68,0x61,0x62,0x65,0x74,0x69,0x63,0x20, 0x61,0x6E,0x64,0x20,0x6C,0x75,0x6D,0x65
0x72,0x69,0x63,0x20,0x63,0x68,0x61,0x72, 0x61,0x63,0x74,0x65,0x72,0x73,0x2C,0x20	0x72,0x69,0x63,0x20,0x63,0x68,0x61,0x72, 0x61,0x23,0x74,0x65,0x72,0x73,0x2C,0x20
0x62,0x65,0x74,0x77,0x65,0x65,0x6E,0x20, 0x74,0x77,0x6F,0x20,0x6F,0x72,0x20,0x6D	0x62,0x65,0x74,0x77,0x65,0x65,0x6E,0x20, 0x74,0x77,0x6F,0x21,0x6F,0x72,0x20,0x6D
0x69,0x6C,0x65,0x20,0x64,0x65,0x76,0x69, 0x63,0x65,0x73,0x2C,0x20,0x64,0x65,0x73	0x69,0x6C,0x65,0x20,0x64,0x65,0x76,0x69, 0x63,0x65,0x73,0x2C,0x22,0x64,0x65,0x73
0x6B,0x74,0x6F,0x70,0x73,0x2F,0x6C,0x61, 0x70,0x74,0x6F,0x70,0x73,0x2C,0x20,0x6F	0x6B,0x74,0x6F,0x70,0x73,0x2F,0x6C,0x61, 0x70,0x74,0x6F,0x70,0x73,0x6C,0x20,0x6F
0x66,0x20,0x63,0x6F,0x6D,0x70,0x61,0x74, 0x69,0x62,0x6C,0x65,0x20,0x63,0x6F,0x6D	0x66,0x20,0x63,0x6F,0x6D,0x70,0x61,0x74, 0x69,0x62,0x68,0x65,0x20,0x63,0x6F,0x6D
0x70,0x75,0x74,0x65,0x72,0x2E,0x20,0x54, 0x65,0x78,0x74,0x20,0x6D,0x65,0x73,0x73	0x70,0x75,0x74,0x65,0x72,0x2E,0x20,0x54, 0x65,0x79,0x74,0x20,0x6D,0x65,0x73,0x73
0x61,0x67,0x65,0x73,0x20,0x6D,0x61,0x79, 0x20,0x62,0x65,0x20,0x73,0x65,0x6E,0x74	0x61,0x67,0x65,0x73,0x20,0x6D,0x61,0x79, 0x20,0x62,0x65,0x20,0x72,0x65,0x6E,0x74
0x20,0x6F,0x76,0x65,0x72,0x20,0x61,0x20, 0x63,0x65,0x6C,0x6C,0x75,0x6C,0x61,0x72	0x20,0x6F,0x76,0x65,0x72,0x20,0x61,0x20, 0x63,0x65,0x6C,0x6C,0x7D,0x6C,0x61,0x72
0x4A,0x7F,0xE2,0x91,0x3D,0xB6,0x58,0x0A, 0x8C,0xD4,0x2F,0x5E,0x70,0x9B,0x13,0xA6	0x4A,0x7F,0xE2,0x91,0x3D,0xB6,0x58,0x0B, 0x8C,0xD4,0x2F,0x5E,0x70,0x9B,0x13,0xA6
0x26,0x18,0xEC,0x7F,0x49,0x05,0xB2,0xD8,0 xF3,0x81,0xA7,0x6D,0x2C,0x40,0x9E,0x5B	0x26,0x18,0xEC,0x7F,0x49,0x05,0xB2,0xD9,0 xF3,0x81,0xA7,0x6D,0x2C,0x40,0x9E,0x5B
0x7A,0x3E,0x59,0xC1,0x20,0x8D,0xF7,0x64, 0xB9,0x1F,0x46,0x35,0xD2,0x87,0xAE,0x02	0x7A,0x3E,0x59,0xC1,0x20,0x8D,0xF7,0x65, 0xB9,0x1F,0x46,0x35,0xD2,0x87,0xAE,0x02
0xE4,0xC7,0x10,0x9A,0x53,0xF8,0x6B,0x24, 0x68,0xDB,0x8E,0x15,0xA0,0x7C,0x34,0xF9	0xE4,0xC7,0x10,0x9A,0x53,0xF8,0x6B,0x25, 0x68,0xDB,0x8E,0x15,0xA0,0x7C,0x34,0xF9
0x97,0x2D,0x5A,0x0B,0xE8,0x14,0xFC,0x36, 0x50,0x1C,0xB3,0x76,0xCA,0x29,0x8F,0xE0	0x97,0x2D,0x5A,0x0B,0xA8,0x14,0xFC,0x36, 0x50,0x1C,0xB3,0x76,0xCA,0x29,0x8F,0xE0
0x6E,0xA5,0x38,0xFD,0x80,0xD7,0x0C,0x94, 0xC2,0xF6,0x73,0x22,0x5D,0x09,0xB8,0x4F	0x6E,0xA5,0x38,0xBD,0x80,0xD7,0x0C,0x94, 0xC2,0xF6,0x73,0x22,0x5D,0x09,0xB8,0x4F

0xAF,0x72,0x01,0xCD,0x98,0x54,0x2B,0xE6, 0x31,0x6A,0xD5,0x88,0x07,0xBB,0x42,0xFC	0xAF,0x72,0x01,0xCD,0x98,0x55,0x2B,0xE6, 0x31,0x6A,0xD5,0x88,0x07,0xBB,0x42,0xFC
0x4D,0x83,0xEE,0x19,0xB6,0x50,0xA2,0x0F, 0x9F,0x24,0x6B,0xC8,0x15,0xFA,0x37,0xD0	0x4D,0x83,0xEE,0x19,0xB6,0x50,0xA0,0x0F, 0x9F,0x24,0x6B,0xC8,0x15,0xFA,0x37,0xD0
0x62,0x3C,0x7F,0x58,0xA4,0xE9,0x01,0xD6, 0x8B,0xF0,0x44,0x9E,0x3D,0x70,0xC2,0xA6	0x62,0x3C,0x7F,0x58,0xA4,0xE9,0x00,0xD6,0 x8B,0xF0,0x44,0x9E,0x3D,0x70,0xC2,0xA6
0x11,0x6D,0xB8,0x5F,0xE0,0x23,0xCA,0x97, 0x75,0xA9,0x30,0xFE,0x46,0x8B,0xD2,0x0C	0x11,0x6D,0xB8,0x5F,0xE0,0x23,0xCB,0x97, 0x75,0xA9,0x30,0xFE,0x46,0x8B,0xD2,0x0C
0xDB,0x16,0xC9,0x5E,0xA3,0x7F,0x20,0x84, 0x5F,0x82,0x3A,0xE7,0x48,0x0D,0xB4,0x69	0xDB,0x16,0xC9,0x5E,0xAB,0x7F,0x20,0x84, 0x5F,0x82,0x3A,0xE7,0x48,0x0D,0xB4,0x69
0x9C,0x42,0xFD,0x35,0xBA,0x67,0x1E,0x80, 0xE2,0xAF,0x71,0x0A,0xD5,0x28,0xFC,0x43	0x9C,0x42,0xFC,0x35,0xBA,0x67,0x1E,0x80, 0xE2,0xAF,0x71,0x0A,0xD5,0x28,0xFC,0x43
0x3E,0x50,0x9D,0x06,0xB3,0x7C,0xA1,0xEF, 0x89,0xC7,0x2A,0xF4,0x51,0x1E,0xD0,0xB3	0x3E,0x50,0x9D,0x06,0xB2,0x7C,0xA1,0xEF, 0x89,0xC7,0x2A,0xF4,0x51,0x1E,0xD0,0xB3
0x27,0x79,0xB4,0x63,0xDC,0x01,0xAE,0xE8, 0xF1,0x2C,0x98,0x43,0xAE,0x65,0x37,0xD0	0x27,0x79,0xB4,0x63,0xDC,0x01,0xAE,0xE0, 0xF1,0x2C,0x98,0x43,0xAE,0x65,0x37,0xD0
0x5A,0x84,0xD1,0x3E,0xE9,0x47,0x2F,0xB0, 0x76,0xBB,0x20,0x8D,0xC1,0x94,0x5E,0x3A	0x5A,0x80,0xD1,0x3E,0xE9,0x47,0x2F,0xB0, 0x76,0xBB,0x20,0x8D,0xC1,0x94,0x5E,0x3A
0xCA,0x15,0x68,0xB7,0x02,0xDF,0x94,0x3E, 0x0F,0x53,0x8E,0x21,0xD4,0x7B,0xA6,0xC0	0xCA,0x15,0x68,0xB7,0x03,0xDF,0x94,0x3E, 0x0F,0x53,0x8E,0x21,0xD4,0x7B,0xA6,0xC0
0x4B,0x96,0x21,0xEC,0x37,0xD8,0x0F,0xA5, 0x83,0xDE,0x57,0x2A,0xB1,0x0C,0xF4,0x69	0x4B,0x96,0x21,0xEC,0x37,0xDC,0x0F,0xA5, 0x83,0xDE,0x57,0x2A,0xB1,0x0C,0xF4,0x69
0x2D,0x71,0xAC,0x03,0xC6,0x59,0x94,0xE8, 0xE9,0x34,0xFB,0x46,0x81,0xCC,0x17,0xAD	0x2D,0x71,0xAC,0x03,0xCE,0x59,0x94,0xE8, 0xE9,0x34,0xFB,0x46,0x81,0xCC,0x17,0xAD
0x40,0x7E,0xA3,0x1C,0xD8,0x52,0x9F,0x6B, 0x74,0xC8,0x25,0xFA,0x5B,0x90,0x0D,0xE6	0x40,0x7E,0xA3,0x3C,0xD8,0x52,0x9F,0x6B, 0x74,0xC8,0x25,0xFA,0x5B,0x90,0x0D,0xE6
0xAE,0x61,0x8C,0x30,0xF5,0x4B,0x97,0x02, 0x19,0x55,0x8A,0xF1,0x3E,0x60,0xBD,0xC7	0xAE,0x61,0x8C,0x30,0xF5,0x4B,0x97,0x22, 0x19,0x55,0x8A,0xF1,0x3E,0x60,0xBD,0xC7
0x45,0x98,0x2F,0xE3,0x78,0xA1,0xD4,0x0C, 0x91,0xDC,0x01,0x7B,0x56,0x2A,0xC3,0xFE	0x45,0x98,0x2F,0xE3,0x78,0xA1,0xD4,0x04,0 x91,0xDC,0x01,0x7B,0x56,0x2A,0xC3,0xFE
0x2B,0x76,0xC9,0x14,0xA0,0x5D,0xE3,0x80, 0xF6,0x38,0x87,0xDA,0x14,0x61,0xBC,0x59	0x23,0x76,0xC9,0x14,0xA0,0x5D,0xE3,0x80, 0xF6,0x38,0x87,0xDA,0x14,0x61,0xBC,0x59
0x52,0x9D,0xE0,0x3F,0xA7,0x6A,0x14,0xC8, 0x8A,0xD4,0x01,0x5F,0x30,0x6E,0xB2,0xC7	0x52,0x9D,0xE0,0x3F,0xA7,0x6A,0x10,0xC8, 0x8A,0xD4,0x01,0x5F,0x30,0x6E,0xB2,0xC7
0xC5,0x20,0x9F,0x74,0xDB,0x01,0xA8,0x3E, 0x10,0x4E,0x93,0x65,0xB8,0x2B,0xD7,0xFA	0xC5,0x20,0x9F,0x70,0xDB,0x01,0xA8,0x3E, 0x10,0x4E,0x93,0x65,0xB8,0x2B,0xD7,0xFA
0x3B,0x84,0xC1,0x5E,0xA9,0x70,0xFE,0x26, 0x67,0xBA,0x0F,0x52,0x8D,0xE1,0x34,0xC8	0x3B,0x84,0xC3,0x5E,0xA9,0x70,0xFE,0x26, 0x67,0xBA,0x0F,0x52,0x8D,0xE1,0x34,0xC8
0x95,0xD9,0x24,0x6F,0xB0,0x8A,0x1E,0xC7, 0xDF,0x21,0x7C,0xA9,0x54,0x0E,0xB3,0x68	0x95,0xD1,0x24,0x6F,0xB0,0x8A,0x1E,0xC7, 0xDF,0x21,0x7C,0xA9,0x54,0x0E,0xB3,0x68
0x0A,0x56,0x89,0xD3,0x2E,0xF1,0xB7,0x4C, 0x47,0x9B,0xC4,0x2F,0xE1,0x1A,0xB0,0x56	0x0A,0x56,0x09,0xD3,0x2E,0xF1,0xB7,0x4C, 0x47,0x9B,0xC4,0x2F,0xE1,0x1A,0xB0,0x56
0x83,0xDC,0x29,0x74,0xB9,0x01,0xF6,0x5E, 0x2F,0x65,0xB8,0x13,0xCE,0x04,0x97,0xDA	0x83,0xDC,0x29,0xF4,0xB9,0x01,0xF6,0x5E, 0x2F,0x65,0xB8,0x13,0xCE,0x04,0x97,0xDA
0xEA,0x25,0x9B,0x47,0x72,0xDF,0x06,0xC8, 0xBC,0xF1,0x4E,0x83,0x1A,0xD7,0x60,0x2F	0xEA,0x25,0x9B,0x47,0x72,0xDF,0x06,0x48,0 xBC,0xF1,0x4E,0x83,0x1A,0xD7,0x60,0x2F

0x6D,0xA1,0x34,0xC8,0x1F,0xF6,0x5B,0x80, 0xAB,0x16,0xC0,0x7F,0x58,0x82,0xD4,0x29	0x6D,0xA1,0x34,0xC8,0x1F,0xF6,0xDB,0x80, 0xAB,0x16,0xC0,0x7F,0x58,0x82,0xD4,0x29
0xF0,0x2A,0x9D,0x57,0x8B,0xC4,0x11,0xE6, 0x45,0x7E,0xB3,0x08,0xD1,0x6C,0xA9,0x20	0xF0,0x2A,0x9D,0x55,0x8B,0xC4,0x11,0xE6, 0x45,0x7E,0xB3,0x08,0xD1,0x6C,0xA9,0x20
0x90,0xDC,0x15,0x6A,0xBF,0x40,0xE7,0x82, 0x3D,0x61,0xBE,0x24,0xC8,0x97,0x50,0xAF	0x90,0xD8,0x15,0x6A,0xBF,0x40,0xE7,0x82,0 x3D,0x61,0xBE,0x24,0xC8,0x97,0x50,0xAF
0x88,0xD3,0x0E,0x72,0x5F,0xA1,0xC4,0xB9, 0x2A,0x74,0xDF,0x01,0x95,0x6B,0xC8,0x30	0x88,0xD3,0x0E,0x72,0x5F,0xA1,0xC6,0xB9, 0x2A,0x74,0xDF,0x01,0x95,0x6B,0xC8,0x30
0xE5,0x38,0x8B,0xD4,0x01,0x76,0xC2,0xAF, 0x0C,0x53,0x9E,0x20,0xD7,0x68,0xB1,0xFC	0xE5,0x28,0x8B,0xD4,0x01,0x76,0xC2,0xAF, 0x0C,0x53,0x9E,0x20,0xD7,0x68,0xB1,0xFC
0x41,0x85,0xF0,0x3E,0xBA,0x56,0x29,0xD4, 0x95,0xCA,0x17,0xF3,0x4E,0x80,0xDB,0x62	0x41,0x85,0xF0,0x3E,0xBA,0x52,0x29,0xD4, 0x95,0xCA,0x17,0xF3,0x4E,0x80,0xDB,0x62
0x1a,0x2b,0x3c,0x4d,0x5e,0x6f,0x80,0x91,0xa 2,0xb3,0xc4,0xd5,0xe6,0xf7,0x08,0x19	0x1A,0x2B,0x3C,0x45,0x5E,0x6F,0x80,0x91,0 xA2,0xB3,0xC4,0xD5,0xE6,0xF7,0x08,0x19
0x2a,0x3b,0x4c,0x5d,0x6e,0x7f,0x90,0xa1,0xb 2,0xc3,0xd4,0xe5,0xf6,0x07,0x18,0x29	0x2A,0x3B,0x4C,0x5D,0x6A,0x7F,0x90,0xA1, 0xB2,0xC3,0xD4,0xE5,0xF6,0x07,0x18,0x29
0x3a,0x4b,0x5c,0x6d,0x7e,0x8f,0xa0,0xb1,0xc 2,0xd3,0xe4,0xf5,0x06,0x17,0x28,0x39	0x3E,0x4B,0x5C,0x6D,0x7E,0x8F,0xA0,0xB1, 0xC2,0xD3,0xe4,0xF5,0x06,0x17,0x28,0x39
0x4a,0x5b,0x6c,0x7d,0x8e,0x9f,0xb0,0xc1,0xd 2,0xe3,0xf4,0x05,0x16,0x27,0x38,0x49	0x4A,0x5B,0x6C,0x7D,0x8E,0x9F,0xB0,0xC0, 0xD2,0xE3,0xF4,0x05,0x16,0x27,0x38,0x49
0x5a,0x6b,0x7c,0x8d,0x9e,0xaf,0xc0,0xd1,0xe 2,0xf3,0x04,0x15,0x26,0x37,0x48,0x59	0x5A,0x6B,0x7C,0xCD,0x9E,0xAF,0xC0,0xD 1,0xE2,0xF3,0x04,0x15,0x26,0x37,0x48,0x59
0x6a,0x7b,0x8c,0x9d,0xae,0xbf,0xd0,0xe1,0xf 2,0x03,0x14,0x25,0x36,0x47,0x58,0x69	0x6A,0x3B,0x8C,0x9D,0xAE,0xBF,0xD0,0xE 1,0xF2,0x03,0x14,0x25,0x36,0x47,0x58,0x69
0x7a,0x8b,0x9c,0xad,0xbe,0xcf,0xe0,0xf1,0x0 2,0x13,0x24,0x35,0x46,0x57,0x68,0x79	0x72,0x8B,0x9C,0xAD,0xBE,0xCF,0xE0,0xF1 ,0x02,0x13,0x24,0x35,0x46,0x57,0x68,0x79
0x8a,0x9b,0xac,0xbd,0xce,0xdf,0xf0,0x01,0x1 2,0x23,0x34,0x45,0x56,0x67,0x78,0x89	0x8A,0x8B,0xAC,0xBD,0xCE,0xDF,0xF0,0x0 1,0x12,0x23,0x34,0x45,0x56,0x67,0x78,0x89
0x9a,0xab,0xbc,0xcd,0xde,0xef,0x00,0x11,0x2 2,0x33,0x44,0x55,0x66,0x77,0x88,0x99	0x98,0xAB,0xBC,0xCD,0xDE,0xEF,0x00,0x1 1,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99
0xaa,0xbb,0xcc,0xdd,0xee,0xff,0x10,0x21,0x3 2,0x43,0x54,0x65,0x76,0x87,0x98,0xa9	0xAA,0xBB,0xCC,0xDD,0xEC,0xFF,0x10,0x2 1,0x32,0x43,0x54,0x65,0x76,0x87,0x98,0xA9
0xba,0xcb,0xdc,0xed,0xfe,0x0f,0x20,0x31,0x4 2,0x53,0x64,0x75,0x86,0x97,0xa8,0xb9	0xBA,0xCB,0xDD,0xED,0xFE,0x0F,0x20,0x3 1,0x42,0x53,0x64,0x75,0x86,0x97,0xA8,0xB9
0xca,0xdb,0xec,0xfd,0x0e,0x1f,0x30,0x41,0x5 2,0x63,0x74,0x85,0x96,0xa7,0xb8,0xc9	0xCA,0xDB,0xEC,0xFF,0x0E,0x1F,0x30,0x41, 0x52,0x63,0x74,0x85,0x96,0xA7,0xB8,0xC9
0xda,0xeb,0xfc,0x0d,0x1e,0x2f,0x40,0x51,0x6 2,0x73,0x84,0x95,0xa6,0xb7,0xc8,0xd9	0xDA,0xEB,0xEC,0x0D,0x1E,0x2F,0x40,0x51 ,0x62,0x73,0x84,0x95,0xA6,0xB7,0xC8,0xD9
0xea,0xfb,0x0c,0x1d,0x2e,0x3f,0x50,0x61,0x7 2,0x83,0x94,0xa5,0xb6,0xc7,0xd8,0xe9	0xEA,0xFB,0x0C,0x1D,0x2E,0x3F,0x50,0x69, 0x72,0x83,0x94,0xA5,0xB6,0xC7,0xD8,0xE9
0xfa,0x0b,0x1c,0x2d,0x3e,0x4f,0x60,0x71,0x8 2,0x93,0xa4,0xb5,0xc6,0xd7,0xe8,0xf9	0xFA,0x0B,0x3C,0x2D,0x3E,0x4F,0x60,0x71, 0x82,0x93,0xA4,0xB5,0xC6,0xD7,0xe8,0xF9

Appendix D: Decryption Code of MPRESENT

```
#include <stdint.h>
#include <string.h> // Include this for memcpy
#include "cipher.h"
#include "constants.h"
#include "rotate.h"
#include "rot.h"

void IP_layer(uint16_t *state){
  state[1] = ROTR1(state[1]);
  state[2] = ROTR12(state[2]);
  state[3] = ROTR13(state[3]);
}
uint64_t shiftback(uint64_t value) {
   return ((value >> 12) | (value << 52));
}

void Decrypt(uint8_t *block, uint8_t *roundKeys)
{
   uint64_t state = *(uint64_t*)block;
uint64_t temp;
uint32_t subkey_lo, subkey_hi;
uint8_t keyindex = 31;
uint8_t i, k;

   for (i = 0; i < 31; i++)
   {
      /* addRoundkey */
subkey_lo = READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[2 *
keyindex]);
subkey_hi = READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[2 *
keyindex + 1]);

state ^= ((uint64_t)subkey_lo) | (((uint64_t)subkey_hi) << 32);
keyindex--;
uint64_t reversedState = shiftback(state);
state = reversedState;

 IP_layer((uint16_t*)&state);

      /* sBoxLayer */
for (k = 0; k < 16; k++)
{

uint16_t sBoxValue = state & 0xF;
state &= 0xFFFFFFFFFFFFFFF0;
state |= READ_SBOX_BYTE(invsBox4[sBoxValue]);
```

```
state = rotate4l_64(state);

}

   }

   subkey_lo = READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[2 *
keyindex]);
subkey_hi = READ_ROUND_KEY_DOUBLE_WORD(((uint32_t*)roundKeys)[2 *
keyindex + 1]);


   state ^= ((uint64_t)subkey_lo) | (((uint64_t)subkey_hi) << 32);
```

الملخص

يهدف البحث إلى انشاء نسخة معدلة من التشفير الخفيف الوزن المسمى برزنت، لاستهلاك طاقة أقل في المتحكم الدقيق من نوع أتمل أي في ار 128 - 8 خانات مع الاخذ بعين الاعتبار حجم البرنامج ودرجة الأمان، حيث أن الشبكة العنكبوتية للأشياء لديها قدرات محدودة. تمت المحاكة وتم استخراج نتائج استهلاك الطاقة وحجم الخوارزمية باستخدام إطار العمل المسمى فلكس. تظهر النتائج أن التشفير الحالي يحتاج الى العديد من النبضات للمتحكم مقارنة بجميع الخوارزميات التي تتكون من طبقة واحدة من التعويض والتبديل، حيث تستهلك طبقة التبديل معظم الطاقة. يقلل تعديل التشفير المقترح المسمى ام - برزنت عدد نبضات المتحكم اللازمة من 30 مليون نبضة إلى 1.5 نبضة لتشفير 128 خانه. اما حجم الخوارزمية ام - برزنت فتحتاج إلى 2754 بايت بينما تحتاج العديد من الخوارزميات الأخرى مثل ليد وروبين وفانتوماس الى حجم أكبر من ذلك. تم تقييم درجة الأمان من خلال قياس درجة الانهيار، وذلك عن طريق تغيير خانه واحده في النص العادي او المفتاح لرؤية عدد الخانات التي تتأثر في الشيفرة. تم تحليل مجموعة من البيانات مكونة من 8384 خانه وأظهرت النتائج ان تغيير خانه واحد في النص العادي برزنت الأصلي يؤثر على 32 خانه، بينما إذا تغير خانه واحده في ام – برزنت المقترح فان ذلك يغير 31.38 خانه بالمعدل من أصل 64 خانه. وفي المحصلة يُظهر النظام المقترح ام – برزنت نتائج جيده من حيث استهلاك الطاقة وحجم البرنامج ودرجة الحماية.