Lightweight Cryptography for Resource Constrained IoT devices

Vishalkumar Arjunsinh Thakor

Supervisor: Dr M. A. Razzaque

School of Computing, Engineering and Digital Technologies
Teesside University, UK

The thesis is submitted for the degree of
Doctor of Philosophy

September 2022

I would like to dedicate this thesis to my beloved family that includes my adore grandmother, my loving parents, my uncles & aunties, my cousins, my better half and my little hearts, Krishna and Parth…

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This thesis contains fewer than 40,000 words including appendices, bibliography, footnotes, tables and equations.

Vishalkumar Arjunsinh Thakor
September 2022

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors, Dr Razzaque for their invaluable advice, continuous support, and patience during my PhD study. I am also grateful to my co-supervisor, Dr Usman for his guidance and support during this journey. Their immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. In fact, it has made my study and stays in the UK an amazing time. Also, I am thankful to Dr Anand and Mr Aksh for their laboratory support to conduct experiments during my stay in India due to the Covid-19 pandemic. Finally, I would like to express my gratitude to my parents, my wife and my children. Without their tremendous understanding and scarification (in many ways), it would be impossible for me to complete my study.

Paper Publications

Paper 1: Lightweight Cryptography Algorithms for Resource Constrained IoT devices: A Review, Comparison and Research Opportunities (IEEE Access)

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

Lightweight Cryptography Algorithms for Resource-constrained IoT devices: A Review, comparison and research opportunities

VISHAL A. THAKOR[1], M.A.RAZZAQUE[2*], and MUHAMMAD R. A. KHANDAKER[3],
[1]School of Computing, Engineering and Digital Technologies, Teesside University, UK (e-mail: v.thakor@tees.ac.uk)
[2]School of Computing, Engineering and Digital Technologies, Teesside University, UK (e-mail: m.razzaque@tees.ac.uk)
[*]Corresponding author: m.razzaque@tees.ac.uk
[3]School of Engineering and Physical Sciences, Heriot-Watt University, UK (email: m.khandaker@hw.ac.uk)

ABSTRACT IoT devices are becoming more common and popular choice due to their wide range of applications in various domains. They collect data from the real environment and transfer it over the

Paper 2: A novel 5-bit S-box design for lightweight cryptography algorithms (Journal of Information Security and Applications (JISA), Elsevier)-Submitted

A novel 5-bit S-box design for lightweight cryptography algorithms

Vishal A. Thakor[a], Mohammad A. Razzaque[a], Anand D. Darji[b] and Aksh R. Patel[b]

[a]*School of Computing, Engineering and Digital Technologies, Teesside University, UK*
[b]*Department of Electronics Engineering, Sardar Vallabhbhai National Institute of Technology (SVNIT), Surat, India*

ARTICLE INFO

Keywords:
Internet of Things (IoT)
Radio Frequency Identification (RFID)
Lightweight Cryptography (LWC)
Substitution-box (S-box)
Chaotic mapping
Cryptanalysis
ASIC platform

ABSTRACT

Cryptography is one of the techniques to secure communication and data transfer over the network. It performs well on resource-rich devices (PC, servers, smartphones, etc.). However, it may not fit or, if forcefully fitted, perform poorly on the resource-constrained Internet of Things (IoT) devices (e.g., Radio Frequency Identification (RFID) tags, sensors). For these reasons, there is a need for a lightweight version of cryptography, called lightweight cryptography (LWC). While designing any cryptography algorithm, a substitution box (S-box) is a core and the only component that offers a nonlinear functionality between inputs and outputs. Various researchers propose various S-box designs for different applications. Still, very few of them maintain the trade-offs among cost, performance

Paper 3: AUM: A Lightweight Cryptography Algorithm for resource constrained IoT devices (Journal of Information Security and Applications (JISA), Elsevier)-Ready

Abstract

IoT is becoming more common and popular due to its wide range of applications in various domains. They collect data from the real environment and transfer it over the networks. While deploying IoT in the real world, there are many challenges, varying from tiny sensors to servers. Since most IoT devices are physically accessible in the real world, security is considered the number one challenge in IoT deployments. Moreover, many of them are limited in resources such as energy, memory, processing power and even physical space, where implanting traditional security options is challenging.

This research focuses on resource-constrained IoT devices such as RFID tags, sensors, smart cards, etc., as it is challenging to secure them in a real environment. The communication from such devices can be secured using lightweight cryptography, a lighter version of cryptography. To give a holistic view of lightweight cryptography, existing lightweight cryptography algorithms are studied and compared in terms of implementation cost, hardware and software performances, and finally, in terms of cryptanalysis to identify the research gaps. A trade-off between cost, performance and security is missing in the existing list of lightweight cryptography algorithms. This creates a demand for new research to carry on and further inspires to development of a new LWC algorithm.

The research study proposes a novel LWC algorithm, named AUM, specially designed for small messages in resource-constrained IoT devices. It offers unique features such as a robust 5-Bit S-Box structure using chaotic mapping theory and lightweight permutation through the 2D array transpose technique. Moreover, the research proposes a lightweight approach to generate distinct subkeys from the original key provided by the user. The resulting cost and performance values are evaluated on popular ASIC platforms and compared with 4-bit and 5-bit competitors. Also, the attack resistance property of the proposed model is analysed over various measured such as bijective property, nonlinearity, linearity (LP), differential probability (cryptanalysis), degree of avalanche effect, bit Independence criteria (BIC) and algebraic attacks and compared with its competitors (4-bit and 5-bit) S-boxes. The test results prove the efficiency of the proposed LWC algorithm, AUM, and show how it maintains the trade-off between cost, performance and security.

Table of contents

List of figures xv

List of tables xvii

Nomenclature xix

1 Introduction 1

1.1 IoT Overview 1

1.1.1 Classification of IoT devices 3

1.1.2 Security Concerns of Resource-Constrained IoT devices 3

1.2 Cryptography Overview 4

1.2.1 Classification of Cryptography 6

1.3 Research Motivation 10

1.3.1 Research Aim 11

1.3.2 Research Objectives 11

1.4 Thesis Outlines 11

2 Research Background 13

2.1 Key requirements of Lightweight Cryptography 13

2.1.1 Lightweight Cryptography Features 15

2.1.2 Standardization of Cryptography Algorithms 17

2.2 Existing Lightweight Cryptography Algorithms 18

2.2.1 Structure wise Classification of LWC Algorithms 18

2.3 Performance Comparison of existing LWC algorithms 26

2.4 Cryptanalysis of existing LWC algorithms 31

2.5 Real-time Applications and their Lightweight demands 34

2.6 Research Gaps 35

3 Research Methodology **37**
3.1 Proposed LWC algorithm: An Overview 37
3.2 Proposed LWC algorithm: Detailed Structure 38
3.2.1 Permutation through Transpose 39
3.2.2 AddRoundKey 42
3.2.3 Substitution using 5-bit S-box 42
3.2.4 Subkey Generation 52
3.3 Inspiration for naming the proposed LWC algorithm 53
3.4 Proposed LWC algorithm (AUM): Pseudo Code 54

4 Result Analysis **59**
4.1 Experiment Setup 59
4.2 Performance and Cost Analysis 62
4.3 Security Analysis 68
4.3.1 Bijective Property 68
4.3.2 Nonlinearity 68
4.3.3 Linear Approximation Probability (LP) 70
4.3.4 High Resistance to Differential Cryptanalysis 71
4.3.5 Boomerang Connectivity Table (BCT) 72
4.3.6 Feistel counterpart of BCT (FBCT) 73
4.3.7 High Degree of Avalanche Effect 74
4.3.8 Bit Independence Criterion (BIC) 77
4.3.9 Algebraic Attacks 78
4.4 Execution results of the full algorithm, AUM 78

5 Research Contribution **83**

6 Conclusion **85**

References **87**

Appendix A AUM: C Code **103**

Appendix B AUM: Verilog Implementation **113**

List of figures

1.1 Simple definition of IoT 2
1.2 IoT Applications 2
1.3 Classification of IoT devices 3
1.4 IoT Security requirements 4
1.5 Cryptography: Encryption & Decryption 5
1.6 Two main types of Cryptography 6
1.7 Block Cipher 7
1.8 (a) Stream Cipher (b) Hash Function 7
1.9 Structure wise Classification of Cryptography 8
1.10 (a) Substitution-Permutation Network (SPN) (b) Feistel Network (FN) . . . 9
1.11 (a) General Feistel Network (GFN) (b) Add-Rotate-Xor (ARX) 9
1.12 Nonlinear feedback shift register (NLFSR) 10

2.1 Key Challenges in resource-constrained IoT devices 14
2.2 Key Challenges addressed by LW Cryptography 15
2.3 Software Efficient LWC algorithms (Top 10) 28
2.4 Memory Efficient LWC algorithms (Top 10) 28
2.5 Latency Efficient LWC algorithms (Top 10) 29
2.6 Hardware Efficient LWC algorithms (Top 10) 29
2.7 Block & Key size wise Hardware Efficient LWC algorithms (Top 10) 30
2.8 Physical Area wise Hardware Efficient LWC algorithms (Top 10) 30
2.9 Energy Efficient Hardware Efficient LWC algorithms (Top 10) 31

3.1 Brief of the proposed LWC algorithm (AUM) 37
3.2 General structure of AUM 38
3.3 Full functioning of the proposed algorithm (AUM) 39
3.4 Filling plaintext bits column wise into 2-D transpose 40
3.5 Reading the bits from 2-D array row wise 40

3.6 Permutation through Transpose 41
3.7 First two steps of substitution process 48
3.8 Mapping each 5-bit block into an S-box for its replacement bits (example) . 49
3.9 Subkey generation process through an example 53

4.1 Experiment Setup 60
4.2 The datapath of an area-optimized version of AUM 60
4.3 Behavioural simulation of AUM (32 clock cycle) 61
4.4 Performance and cost comparison of AUM with other competitors 63
4.5 Datapath of 5-bit S-box 65
4.6 Area, Power & Energy requirement by various S-boxes @ 100KHz frequencies 66
4.7 Various S-box Area (GE) Comparison 67
4.8 Nonlinearity (Hamming distance (H_d)) 70
4.9 Linear Approximation Probability (LP) comparison 71
4.10 Differential Approximate Probability Comparison 72
4.11 Avalanche effect/Strict Avalanche Criterion (SAC) 76
4.12 Bit Independence Criterion - SAC 77
4.13 Cryptanalysis of various 5-bit S-boxes 79
4.14 AUM execution (Encryption) 79
4.15 AUM subkey generation from the original key 80

List of tables

2.1 LWC Features 16
2.2 Structure wise LWC algorithms 19
2.3 Hardware and Software performances of LWC algorithms 27
2.4 Security Analysis of LWC Algorithms 33
2.5 Real-time IoT Applications and their lightweight demands 34

3.1 3-bit S-box design 43
3.2 4-bit S-Box Design 43
3.3 5-bit S-Box Design 44
3.4 6-bit S-Box Design 45
3.5 8-bit S-box Design 45
3.6 S_0: 4x4 S-box 46
3.7 S_1: 4x4 S-box 46
3.8 Existing S-boxes and related concerns 47
3.9 Implementation Flexibility of 5-bit S-box with different block size 51

4.1 AUM implementation @100KHz frequencies 62
4.2 GE requirements by individual modules in AUM 65
4.3 Various S-box implementation @ 100KHz frequencies 66
4.4 Various S-box Area Comparison 67
4.5 Nonlinearity measure through Hamming distance (H_d) 69
4.6 The DAP matrix of the proposed 5-bit S-box 72
4.7 Difference Distribution Table of the proposed 5-bit S-box 73
4.8 The occurrence of each element in BCT and DDT of the proposed S-box . . 74
4.9 Feistel counterpart of BCT (FBCT) of the proposed 5-bit S-box 75
4.10 Cryptanalysis of various 5-bit S-boxes 78
4.11 Plaintext and their' corresponding cipher of AUM (Avalanche effect) 81

Nomenclature

Acronyms / Abbreviations

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

ASIC Application Specific Integrated Circuit

BCT Boomerang Connectivity Table

BIC Bit Independence Criterion

CAGR Compound Annual Growth Rate

Cryptrec Cryptography Research and Evaluation Committees (Japan)

DAP Differential Approximate Probability

DDT Differential Distribution Table

DES Data Encryption Standard

Ecrypt European Network of Excellence in Cryptology

FBCT Feistel counterpart of BCT

FPGA Field Programmable Gate Arrays

GE Gate Equivalents

GPRS General Packet Radio Service

GSM Global System for Mobile Communications

HDL Hardware Description Language

IETF Internet Engineering Task Force

IoT Internet of Things

ISO/IEC International Organization of Standardization and the International Electrotechnical Commission

LWC Lightweight Cryptography

MITM Man-in-the-Middle Attack

NESSIE New European Schemes for Signatures, Integrity and Encryption

NIST National Institute of Standards and Technology (USA)

NSA National Security Agency (USA)

RFID Radio Frequency IDentification

RTL Register Transfer Level

S-box Substitution box

SAC Strict Avalanche Criterion

UMTS Universal Mobile Telecommunications System

Chapter 1

Introduction

This chapter gives an overview of the Internet of Things devices and cryptography. The chapter focuses on a specific category of IoT devices, resource-constrained IoT devices such as RFID tags, smartcards, sensors, etc., and showcase their security concerns, primarily when implemented in constrained circumstances. Also, how the lightweight version of cryptography meets the above requirements is discussed in this chapter. Further, a brief on standardizing bodies for cryptography algorithms are discussed, followed by motivation for this research with its aim and objectives.

1.1 IoT Overview

The first "thing" on the Internet appeared in the early ’80s and then surpassed the world’s human population in 2008/09 [1]. Since then, the number of connected devices to the Internet has grown expeditiously worldwide by touching every field. According to Gartner, IoT will capture the **$ 58bn markets by 2025** with rising of **34% CAGR** compare to 2020 [2].

IBM [3] defines "IoT as a giant network of connected things and humans that collect and share data with the surrounding environment". "IoT is the collective network of connected devices, and the technology that facilitates communication between devices and the cloud, as well as between the devices themselves" by Amazon [4]. Microsoft Azure [5] says, "IoT could be any equipment, machines, products, and devices with the capacity to collect and transmit data securely to the cloud".

Many researchers and industry giants have given various definitions of IoT depending on their applications and implementation area [6–10]. Still, in simple words, IoT is a network of connected things, each with a unique identification, able to collect and exchange data over the Internet with or without human interaction (Figure 1.1).

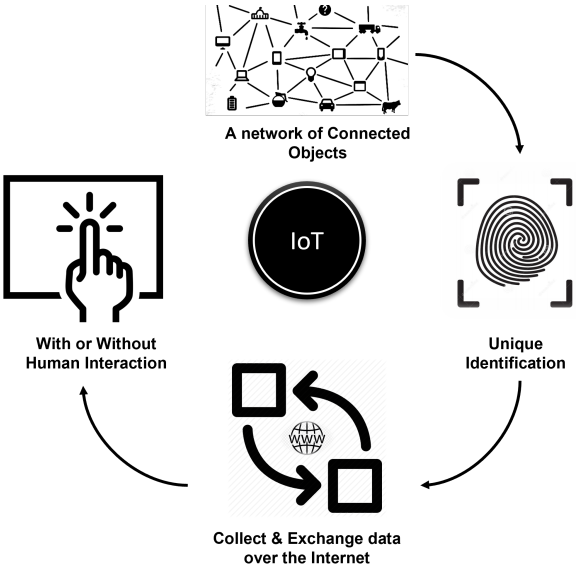


Fig. 1.1 Simple definition of IoT

Internet of Things (IoT) has become a dominant research area because of its applications in various domains such as smart transport & logistics, smart healthcare, smart environment, smart infrastructure (that includes smart cities, smart homes, smart offices, smart malls, Industry 4.0), smart agriculture and many more (Figure 1.2).

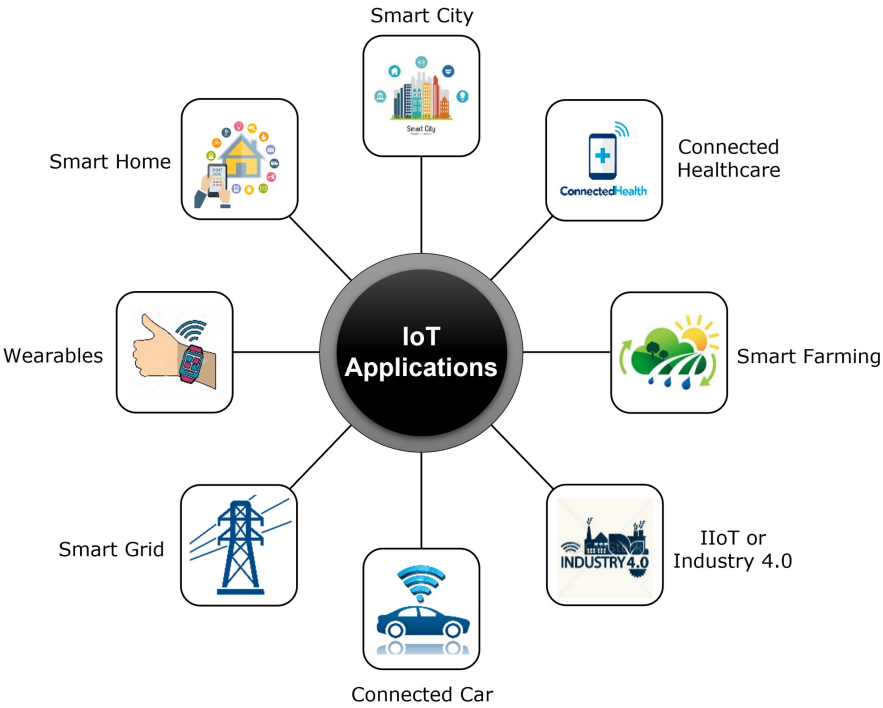


Fig. 1.2 IoT Applications

1.1.1 Classification of IoT devices

For any IoT solution or application, IoT devices are the key elements. These IoT devices could be divided into two main categories [11]: rich in resources such as servers, personal computers, tablets and smartphones, etc., and limited in resources (resource-constrained) such as industrial sensors or sensor nodes, RFID tags, actuators, etc., (Figure 1.3).

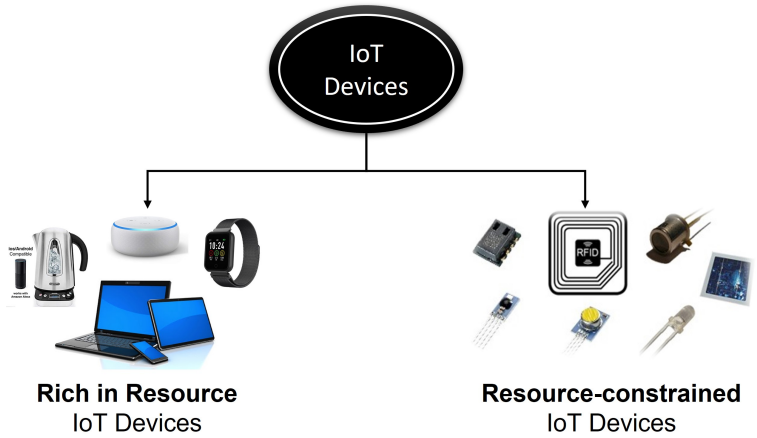


Fig. 1.3 Classification of IoT devices

These connected devices are becoming more popular due to their use in various applications. With the rise of IoT, they will flood the market [11], leading to an enormous data exchange rate amongst [12]. This research focuses on the second category of IoT devices, i.e., resource-constrained IoT devices.

1.1.2 Security Concerns of Resource-Constrained IoT devices

When billions of smart devices (connected devices) work in a diverse set of platforms, especially when shifting from server to sensors, it gives birth to various unprecedented challenges to their owners or users [11]. It could be security & privacy, interoperability, longevity & support, technologies, and many more [13]. Also, IoT devices are easily accessible and exposed to many security attacks [14] as they interact directly with the physical world to collect confidential data or to control physical environment variables, which makes them attractive targets for attackers [15]. All these circumstances make cybersecurity a major challenge in IoT devices with demands of confidentiality, data integrity, authentication & authorization, availability, privacy & regulation standards and regular system updates [13]. Figure 1.4 depicts IoT security challenges and its security requirements.

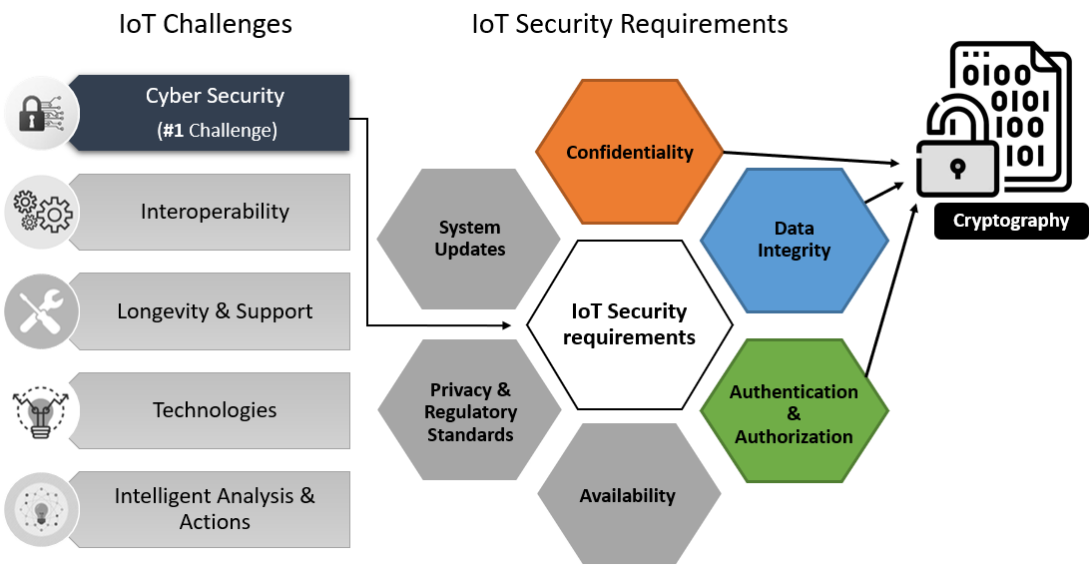


Fig. 1.4 IoT Security requirements

In this scenario, cryptography could be one of the effective measures to guarantee confidentiality, integrity and authentication & authorization of the traversing data through IoT devices [12].

1.2 Cryptography Overview

Cryptography is an ancient technique used by humans to hide secret messages. Some 4000 years old cryptic text, hieroglyphs, found on a tomb from the old kingdom of Egypt [16]. A craftsman's formula for pottery glazing was discovered on clay tablets from Mesopotamia (about 1500 BC) [17]. Around 600 to 500 BC, Hebrew intellectuals used the Atbash cipher, a basic monoalphabetic substitution ciphers [18][19]. "Kama Sutra", an ancient Hindu Sanskrit text (400 BC to 200 AD), documents "Mlecchita vikalpa" (the art of understanding writing in cypher, and the writing of words in a peculiar way) for secret communication between lovers [20][17]. Sparta, an ancient Greek city, used a scytale transposition cipher (transposition cylinder) for military communication [18][21]. Around 100 BC, Julius Caesar, a Roman politician and soldier, introduced the Caesar cipher, one of the most simple and well-known encryption systems, for his personal conversations [22]. In the 16th century, Vigenere built the first cipher with an encryption key, and multiple spanning of the message [23]. Also, cryptography is documented as an important technique of communication during world war-I [24], and world war-II [25].

Cryptography is originally from the Greek words, "kryptÓs (i.e., hidden/secret) and graphein (i.e., to write)", means “secret writing” [26]. It is a technique that converts readable text into an unreadable form, called encryption [27]. Then the reverse procedure restores it to its original form only for the intended user(s), called decryption [27]. Figure 1.5 explains the encryption and decryption process on a message between sender and receiver.

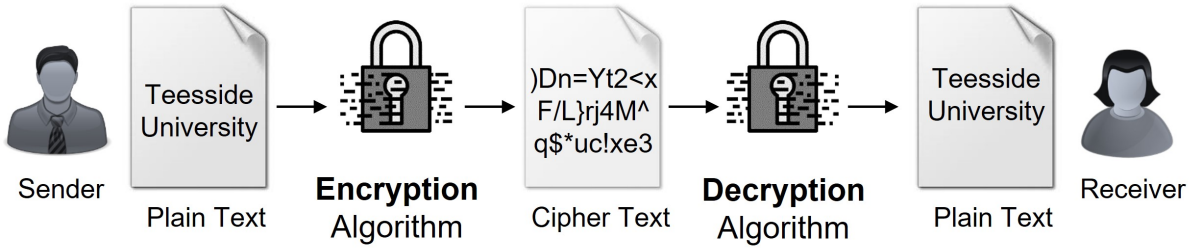


Fig. 1.5 Cryptography: Encryption & Decryption

Some of the key terminologies in cryptography are as follows [28]:

Plain text: An original message, generally readable by humans.

Cipher text: The coded/encrypted message which unreadable by the humans.

Encryption Algorithm: An algorithm that converts plaintext into ciphertext.

Decryption Algorithm: An algorithm that restores the plaintext from the ciphertext.

Data Confidentiality: It guarantees the protection of private or confidential information from unauthorised viewing and access.

Data Integrity: It assures the reliability and correctness of the data, i.e., it is protected from unauthorised modifications while transferring.

Authentication: It is the process of verifying whether the user is who he/she claims to be (i.e., who you are). It could be confirmed through passwords, OTP, biometrics, etc.

Authorization: It defines what resources a user can access. It is determined by setting up the rules and enforcing the policies at different levels.

Confusion: It makes the relationship between the ciphertext and the key as complex as possible using the substitution technique.

Diffusion: It dissipates the statistical structure of plaintext over the bulk of ciphertext using the permutation technique.

Cryptanalysis: It is the technique used for deciphering a message without any knowledge of the enciphering details, i.e. “breaking the code”.

In addition, **Lightweight Cryptography (LWC)** is a lighter version of cryptography that ensures communication security in a resource-constrained environment, especially for resource-constraint IoT devices, with very few resource requirements such as small memory, low computing power, small physical area and low battery power.

1.2.1 Classification of Cryptography

Two main types of cryptography algorithms are symmetric keys cryptography and asymmetric keys cryptography [28]. Symmetric key uses a single key for both encryption and decryption of the data, whereas asymmetric cipher uses two different keys to encrypt and decrypt the data (Figure 1.6).

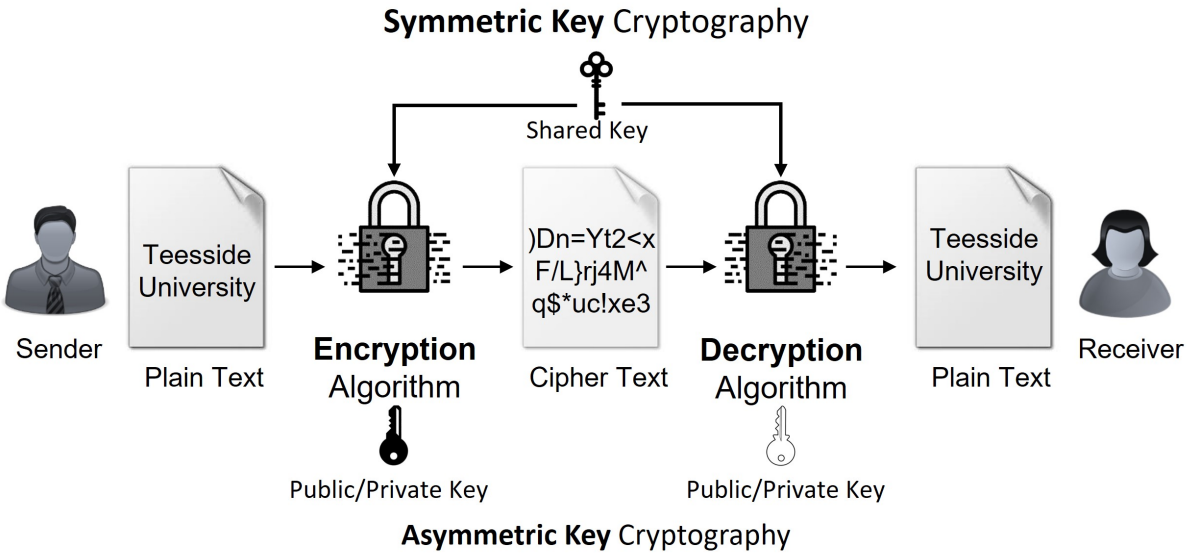


Fig. 1.6 Two main types of Cryptography

Symmetric key cryptography is safe and comparatively fast; the only downside of symmetric key encryption is the sharing of the key between the communicating parties without compromising it [29]. But this could be overcome by pre-sharing the key through a trusted third party. Also, it ensures confidentiality, data integrity and authentication (using authentication encryption mode (AEAD)) of the data. Asymmetric cryptography uses a pair of keys, namely public and private keys. It ensures confidentiality and integrity by using the public key of the receiver. Further, it ensures authentication by encrypting the data by using the sender's private key (as a digital signature). At the other end, the receiver decrypts it by using the sender's public key first and then using his/her private key [28]. The only disadvantage of asymmetric encryption is its large key which increases the complexity and slows down the process [29]. For the reasons mentioned above, symmetric cryptography best fits IoT devices in the resource-constrained environment (section 2.1); this research focuses on symmetric cryptography algorithms.

Symmetric key cryptography could be further classified into three categories depending on how it works: 1) block cipher, 2) stream cipher, and 3) hash functions. Figure 1.7 shows the functioning of symmetric block ciphers as follows.

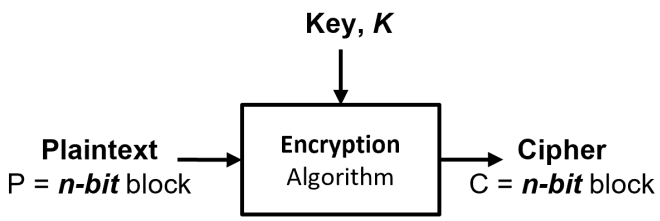


Fig. 1.7 Block Cipher

In block cipher, both encryption and decryption take place on a fixed size block (64 bits or more) at a time, whereas stream cipher continuously processes the input elements bit by bit (or byte by byte) [28] as shown in the Figure 1.8 (a). There are two fundamental properties of any cryptography, confusion and diffusion [30][31], introduced by Claude Shannon to strengthen the cipher. The stream cipher uses only the confusion property, whereas the block cipher uses both confusion and diffusion with a simple design compared to the stream one. Furthermore, following the reverse encryption process to extract the original text is challenging in a block cipher. A stream cipher performs XOR function(s) to encrypt the data, which could be easily reverted to its original form.

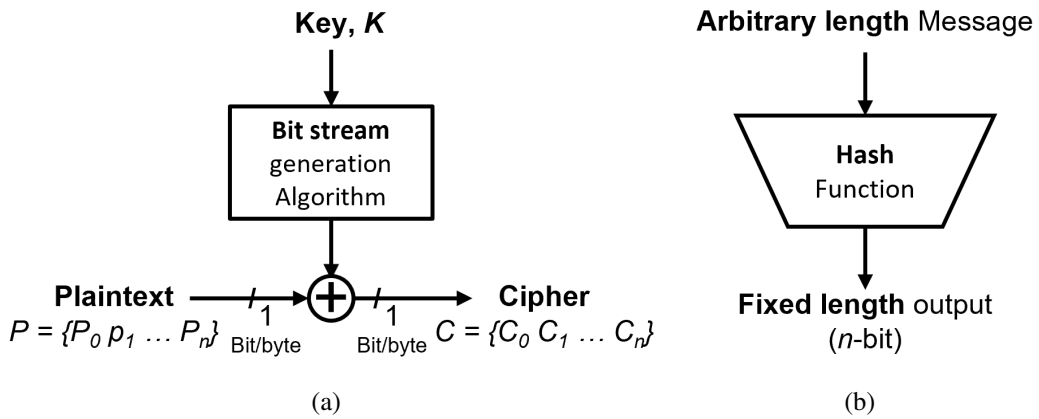


Fig. 1.8 (a) Stream Cipher (b) Hash Function

On the contrary, Hash is a one-way mathematical function that transforms arbitrary length data into a specified-length bit string (short string). Usually, it compresses the input data to produce the output, which is impossible to retrieve. Nevertheless, the hash functions are widely popular among password storage (oneway/non-retrieval) and data integrity verification applications (Figure 1.8 (b)).

From the above discussion, it is evident that the block cipher provides better security over the stream cipher by offering both crucial properties, confusion and diffusion, to strengthen the cipher. Further, Hash is a one-way function, where retrial of the original message is

impossible. Since retrieval of the original message is essential in most IoT applications, the Hash is unsuitable for such scenarios (even in resource-constrained environments). For these reasons, a block cipher is preferred over a stream cipher and hash functions in resource-constrained IoT devices.

Depending on the internal structure (functions) used to create the cipher, the block cipher is classified into six following types. The Figure 1.9 portrays the full classification of cryptography.

- Substitution-Permutation Network (SPN)
- Feistel Network (FN)
- General Feistel Network (GFN)
- Add-Rotate-XOR (ARX)
- NonLinear-Feedback Shift Register (NLFSR)
- Hybrid

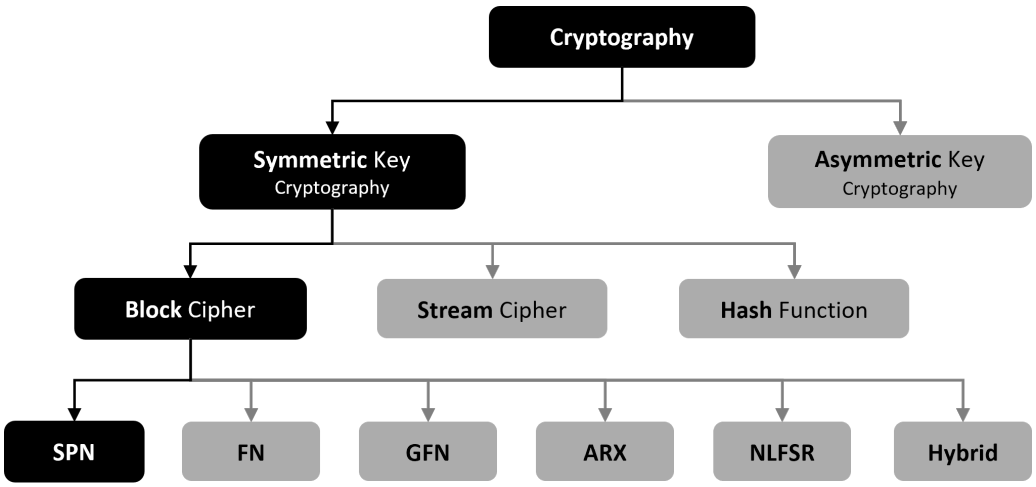


Fig. 1.9 Structure wise Classification of Cryptography

Substitution-Permutation network (SPN) tweaks the data through a set of substitution boxes and permutation tables and formulates them for the following round. A **Feistel network (FN)** breaks the input block into equal halves and applies diffusion in each round to just one half. In addition, swapping of two halves happens at the beginning of each round. The **generalized Feistel network (GFN)** is an extrapolated version of the classic Feistel network. It splits the input block into many sub-blocks and applies the Feistel functions to every pair

of sub-blocks, followed by a cyclic shift proportional to the number of sub-blocks [32]. **ARX** performs encryption-decryption using addition, rotation and XOR functions without making use of S-box. Implementation of ARX is fast and compact but limited in security properties compared to SPN and Feistel ciphers. **Nonlinear feedback shift register (NLFSR)**, applies to both stream and block ciphers, utilizes the building blocks of stream ciphers whose current state is derived from its last state, which is a nonlinear feedback value [33]. **Hybrid** cipher combines any three types (SPN, FN, GFN, ARX, NLFSR) or even mixes block and stream property to improve specific characteristics (for example, throughput, energy, GE, etc.) based on its application requirements. Figure 1.10, Figure 1.11 and Figure 1.12 depict the internal structure of the subcategories of the block cipher.

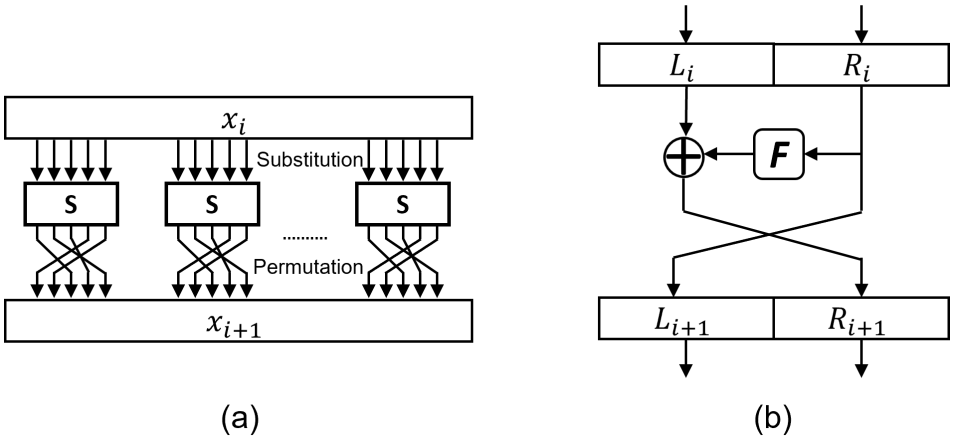


Fig. 1.10 (a) Substitution-Permutation Network (SPN) (b) Feistel Network (FN)

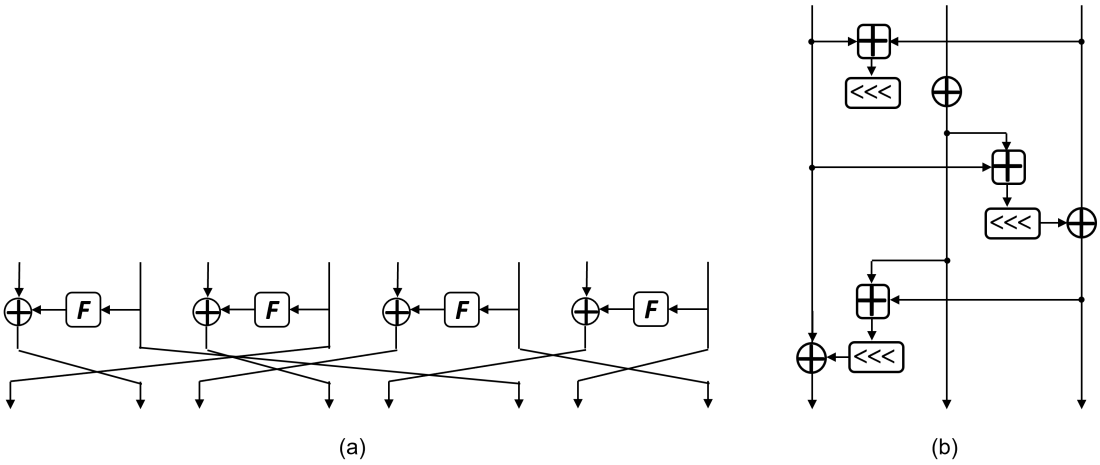


Fig. 1.11 (a) General Feistel Network (GFN) (b) Add-Rotate-Xor (ARX)

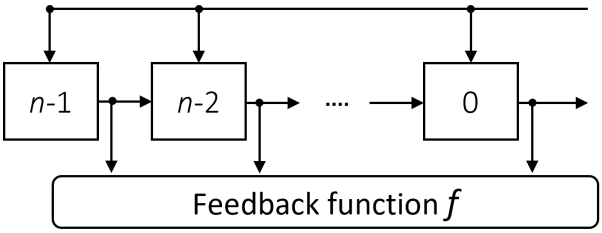


Fig. 1.12 Nonlinear feedback shift register (NLFSR)

Out of these structures, SPN and FN are the most popular choice due to their flexibility to implement, based on application requirements [31]. Although Feistel structures are incorporated easily into low-average power hardware (due to the absence of round function in one-half of the states), it usually requires more round function compared to SPN structures for safety reasons [34]. Therefore, when there is a choice between fewer SPN function rounds and higher Feistel function rounds, the SPN function could be better. SPN provides the same level of security with similar or low energy costs (in most cases) over the Feistel network that shows the higher need for other resources such as computing power, GE etc. [34]. Due to these reasons, this research concentrates on SPN structure to develop a new symmetric LWC algorithm.

1.3 Research Motivation

A strong motivation is always the source of good research. There are many factors that motivated research in cybersecurity, especially IoT security through cryptography. With the popularity and enormous growth of IoT devices at the consumer, commercial and industrial levels, where they interact directly with the physical world to exchange confidential data, it has become the centre of attraction for attackers [14][15]. The security concerns such as confidentiality, integrity and authentication & authorization of the data could be maintained for these IoT devices by the mean of Cryptography [12]. However, since many IoT devices that interact in the real-world environment (such as RFID tags, smart cards, sensors, actuators, etc.) are very limited in resources, traditional cryptography could not be an effective solution in such a resource-constrained environment. Thus, a lighter version, LWC could be one of the effective solutions to protect the stored or communicating data in such circumstances (see section 2.1 and 2.1.1).

In the last decade, many algorithms have been proposed for LWC. Besides, many works have revealed the security attacks on particular LWC algorithm(s) [33, 35–50]. Also, most of these algorithms are designed to work in certain domains or suitable for certain applications

only. After a thorough study of existing LWC algorithms, it was understood that none of the existing algorithms meets all the lightweight criteria in terms of cost, performance and strong security. Also, non of the existing LWC algorithms with SPN structure offers small blocks and keys (i.e., < 64 bits). Small block size is essential for processing small messages efficiently, typically in IoT devices such as sensors, RFID tags and smart cards, where the message size is less than $2Kb$. **Thus, an efficient algorithm to work on small messages (message size $< 2Kb$) of resource-constrained IoT devices is still missing.** In addition, a competition call from NIST [51] in 2018 to create new LWC algorithms for easy and efficient implementation on resource-constrained circuitry, really motivated me to explore further research on lightweight cryptography algorithms and it's demand.

1.3.1 Research Aim

This research aims to develop a novel lightweight cryptography algorithm with a trade-off amongst cost, performance and security.

1.3.2 Research Objectives

In the direction to achieve the above aim, the following objectives are focused on and performed during this research study:

- How to design lighter but secure Permutation?
- How to design simple but fast and robust S-Box (Substitution box) for confusion properties with the reduced number of rounds?
- How to make key scheduling lighter with a small key size but adequate strength?

1.4 Thesis Outlines

Considering the significance of IoT security, this research takes an inclusive view of symmetric key lightweight cryptography. Chapter 1 (Introduction) talks fundamentals of IoT and cryptography by highlighting essential requirements for lightweight cryptography for resource-constrained IoT devices.

A comprehensive study of existing LWC algorithms and their performances, cryptanalysis and real-time use cases are documented in Chapter 2 (Research Background). A vital element of any research, i.e., research gap, is also identified at the chapter's end of this literature review.

Based on these research challenges, a solution is proposed in the form of an algorithm in Chapter 3 (Research Methodology). It describes the proposed LWC algorithm's operational structure and its pseudo code.

The efficiency of the proposed model is analysed by conducting experiments. The process of evaluating the algorithm and its results in the form of cost, performance and security are discussed in Chapter 4 (Result Analysis).

The next chapter, Chapter 5, focuses on the contribution of this research by flashbacking on research gaps (research questions) and showing how the proposed LWC algorithm addresses these challenges.

Finally, Chapter 6 (Conclusion) concludes the proposed research work.

Also, the code developed for the proposed algorithm, AUM, in C and Verilog, is attached in Appendix A and Appendix B.

Chapter 2

Research Background

As lightweight cryptography is one of the robust security solutions for resource-constrained IoT devices, this chapter comprehensively views existing LWC algorithms. The chapter begins by highlighting the key challenges of resource-constrained IoT devices while using traditional cryptography. Then it shows how lightweight cryptography addresses those challenges. Further, it briefs hardware and software metrics used to evaluate key features offered by LWC. The study compares the performance of more than forty existing symmetric key lightweight cryptography algorithms over seven metrics (Block/Key size, Memory, Gate Area, Latency, Throughput, Power & Energy requirements along with hardware and software efficiency) carried out by different researchers. Another essential characteristic of these LWC algorithms, cryptanalysis, is evaluated in a grid form to identify the security strength and common attacks on these LWC algorithms. In addition, the chapter demonstrates real-world IoT applications with their best suite LWC options. Finally, the section outlines open research challenges to design and develop a new lightweight cryptography algorithm for further research.

2.1 Key requirements of Lightweight Cryptography

Cryptography could be a solution to secure the data at rest or move over the network. However, the conventional algorithms could be implemented easily into rich resource IoT devices (such as PC, Servers, Smartphones, etc.), they do not fit into resource-constrained IoT devices. A lighter version of these solutions, lightweight cryptography, can address these challenges to secure communication in resource-constrained IoT devices. Figure 2.1 shows the key challenges while implementation of conventional cryptography in resource-constrained IoT devices as follows [52]:

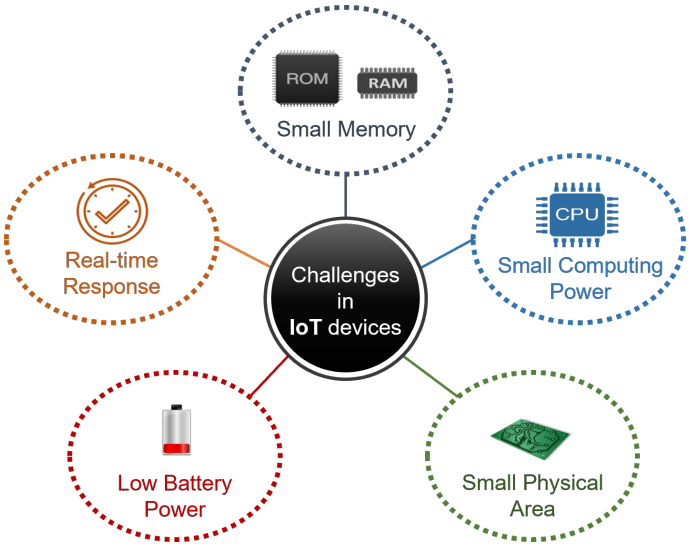


Fig. 2.1 Key Challenges in resource-constrained IoT devices

- Limited memory (registers, RAM, ROM)
- Reduced computing power
- Small physical area to implement the assembly
- Low battery power (or no battery)
- Real-time response

Most IoT devices (such as RFIDs and sensors) are small in size. They are equipped with limited resources such as small memory (RAM, ROM) to store and run an application, low computing power to process the data, little battery power (or no battery in case of passive RFID tags) [11], small physical area to fit-in the assembly [11][52]. Moreover, most of the IoT devices deal with real-time applications where quick and accurate response with essential security using available resources is a challenging task [53][54]. IoT device designers face several risks and challenges, including energy capacity [55], and data security [14]. In these context, if conventional cryptography standards are applied to IoT devices (mainly RFIDs and sensors), their performance may not be acceptable [11].

The issues with conventional cryptography are very well addressed by its sub-discipline, lightweight cryptography, by introducing lightweight features such as small memory, small processing power, low power consumption, real-time response, etc. (Figure 2.2).

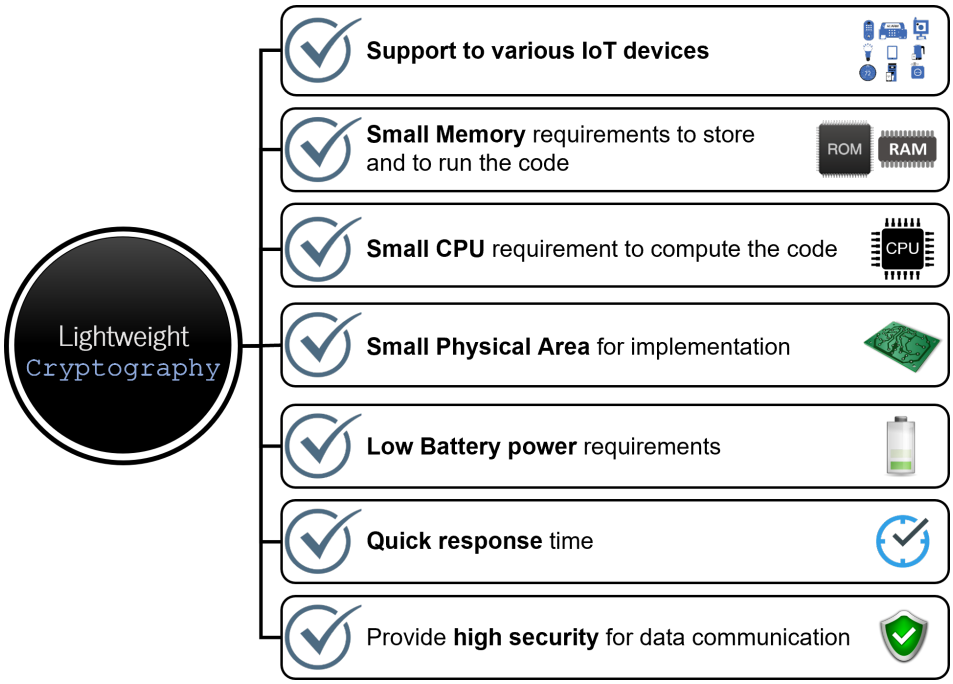


Fig. 2.2 Key Challenges addressed by LW Cryptography

Another important aspect of lightweight cryptography is that it is not just applicable to resource-constrained devices (RFID tags, sensors, etc.) but readily applicable to other devices that are rich in resources (such as servers, PCs, tablets, smartphones, etc.) where it directly or indirectly interacts with [11].

2.1.1 Lightweight Cryptography Features

The features offered Lightweight Cryptography algorithms are grouped into three categories: cost, performance and security as shown in Table 2.1 [14][52]:

Before implementing any Cryptographic algorithm to a resource-constrained IoT device, the cost, performance and security must be considered to obtain effective outcomes. Each of these features is further observed where physical space occupied, memory demand and energy consumption as a cost to implement, processing power in terms of latency and through as performance (speed) and block/key length and various security analysis such as differential cryptanalysis, algebraic attack, an avalanche of change, bit Independence criteria, linearity and nonlinearity property of the algorithm as security measures. The first two features are satisfied by LWC algorithms by offering simple round functions on the small block ($\leq$ 64bit) using a small key ($\leq$ 80bit) with simple key scheduling. The last but essential aspect, security, is fulfilled by adopting one of the six internal structures (SPN, FN, GFN, ARX, NLFSR, Hybrid) to immune against the security attacks.

Table 2.1 LWC Features

Features		What LWC can offer?
Physical (Cost)	Physical Area (GEs, logic blocks)	• Tiny key & block size • Simple rounds with simple computation • Simple key generation
	Memory (registers, RAM, ROM)	
	Battery power (energy consumption)	
Performance	Computing power (latency, throughput)	
Security	Minimum security strength (bits)	• Strong internal structure
	Various cryptanalysis such as Linearity, Nonlinearity, Differential cryptanalysis, Avalanche Effect, Bit Independence criteria and Algebraic attacks etc.	

Hardware and Software performance metrics

Based on the first two characteristics (physical and performance) offered by any LWC algorithms, hardware and software specific resource requirement could be measured in terms of memory requirements, gate area, latency, throughput, and power and energy consumption as follows:

Memory requirements: Generally, it is measured in KB [31]. RAM is required to store intermediate values that can be used in computations, and ROM is required to store the program/algorithm, and static data, such as algorithm key, S-box (in some cases), etc. [11].

Gate Area: It is the physical area required to implement/run the algorithm on a board/circuit, measured in μm^2. This space can be specified using logical blocks for FPGA or using GE for ASIC (1GE = 2 input-NAND Gate) [11]. Typically, 200 to 2000 GE (out of 1000 to 10,000 GE of total available) are allocated for security reasons in an economic RFID tag [56].

Latency: It is the time to produce the cipher from the original text in terms of hardware performance [11] whereas the amount of clock cycles per block (during encryption) defines the software latency.

Throughput: Throughput, in hardware, can be measured in terms of plain text processed per time unit (bits per second) at 100 KHz frequency, whereas in software, it is the average amount of plaintext processed per CPU clock cycle at 4 MHz frequency [57].

Power requirements: The amount of power required by the circuit to process the algorithm can be measured in μW.

Energy consumption: Energy consumption per bit can be calculated as follows [31]:

$$Energy\ [\mu J] = (Latency\ [cycles/block] * Power\ [\mu W])/block\ size\ [bits] \tag{2.1}$$

Here, latency is in terms of software implementation

Efficiency: Gives performance over resource requirements. For hardware, it can be calculated as follows [31]:

$$Hardware\ Efficiency = Throughput[Kbps]/Complexity[KGE] \tag{2.2}$$

Here, complexity means physical space.

Similarly, **software efficiency** can be determined as follows [31]:

$$Software\ Efficiency = Throughput[Kbps]/Code\ Size[KB] \tag{2.3}$$

Here, code size is the algorithm size.

2.1.2 Standardization of Cryptography Algorithms

The organizations that are actively contributing in the field of cryptography to improve the lightweight standards for the resource-constrained devices are:

- **NIST** - National Institute of Standards and Technology, USA [58]
- **ISO/IEC** - International Organization of Standardization and the International Electrotechnical Commission [59]
- **Cryptrec** - Cryptography Research and Evaluation Committees, a set up by Japanese Government [60]
- **Ecrypt** - European Network of Excellence in Cryptology [61]
- **NSA** - National Security Agency, USA [62]
- **CryptoLUX** - University of Luxembourg, Luxembourg [63]

The above organizations conduct various activities such as expert talks, seminars, workshops, conferences. Furthermore, they invite researchers from all around the world to participate and share their innovative ideas in the various security competitions.

2.2 Existing Lightweight Cryptography Algorithms

More than fifty symmetric LWC algorithms are proposed by various academia, proprietaries and government bodies with a focus on reducing cost (memory, processing power, physical area (GE), energy consumption) and enhanced hardware and software performance (latency, throughput). Most of the symmetric block cipher cryptography algorithms are derived either from DES or AES, which are the milestones for cryptography algorithms. From the literature review (subsection 2.2.1), it could be said that PRESENT, inspired by AES, is the milestone of lightweight cryptography and is at the root of many newly proposed lightweight cryptography algorithms. However, most of them do not concentrate on all three fundamental characteristics (cost, performance & security) all at a time and focus on one or any two of these [54]. PRESENT [64], and CLEFIA [65] are the only two algorithms approved by the ISO/IEC 29192 standard, whereas AES, CLEFIA, TDES, Camellia, PRESENT, PRINCE, Piccolo, LED, TWINE, SIMON & SPECK and Midori are targeted by Cryptrec.

Most of these algorithms are designed to work in certain environments and for specific applications such as telecommunication, automobiles, finance, military, etc. A wide variation of resource requirements (section 2.3) with different security levels (section 2.4) is observed in this study. It creates demand for some standardization of lightweight cryptography algorithms. NIST invited researchers from all over the world to participate in a competition with their LWC algorithm ideas. The competition aimed to set up lightweight cryptography standards to fit into constrained environments, and 57 proposals were submitted to it, out of which only ten were finalized, in March 2021 [66].

2.2.1 Structure wise Classification of LWC Algorithms

Depending on the operations they carried on, the lightweight cryptography algorithms (except NIST competition) could be categorized structure-wise as follows (Table 2.2):

Table 2.2 Structure wise LWC algorithms

Structure Types	Algorithms
SPN	AES, Present, GIFT, SKINNY, Rectangle, Midori, mCrypton, Noekeon, Iceberg, Puffin-2, Prince, Pride, Print, Klein, Led, Picaro, Zorro, EPCBC, I-Present, ASCON, SHAMASH, PRIMATE, ICEPOLE
FN	DESL/DESXL, TEA/XTEA/XXTEA, Camellia, Simon, SEA, KASUMI, MIBS, LBlock, ITUbee, FeW, GOST, Robin, Fantomas
GFN	CLEFIA, Piccolo, Twis, Twine, HISEC
ARX	Speck, IDEA, HIGHT, BEST-1, LEA
NLFSR	KeeLoq, KATAN/KTANTAN, Halka
Hybrid	Hummingbird, Hummingbird-2, Present-GRP

The following subsections unfold these LWC algorithms category wise.

Substitution Permutation Network (SPN)

AES [67] is a classic example of SPN based algorithm, standardized by NIST, performs on 128-bit block with 128, 192 and 256-bit key variants [68]. The minimum GE requirement recorded for AES is around 2400 GEs (23% smaller than the usual one) [68], which is still heavy for some small scale real-time applications [30]. It shows the comparatively efficient performance when supplied with additional resources [69].

Another, most hardware and software efficient and ISO/IEC(29192-2P:2012) approved algorithm is **PRESENT**. It is Substitution-Permutation network-based uses 64-bit block on two key variants: 80-bit and 128-bit keys with the GE requirements of 1570 and 1886, respectively [64]. The minimum GE requirement noted for a version of PRESENT is approx. 1000 GE (encryption only) [70], where it takes 2520-3010 GE to provide an adequate level of security [30]. It is a hardware efficient algorithm and uses 4-bit S-boxes (substitution layer - replaces eight S-boxes with single S-box), whereas it takes large cycles in software (permutation layer) which demands an improved version of this [29][30][31][64][71].

GIFT[72], an improved version of PRESENT, was presented in CHES-2017. It offers a lighter S-Box with smaller physical space. Also, the number of rounds is less and gives high throughput along with a simpler and faster key schedule. There are two versions of GIFT: GIFT-64, 28-round with 64-bit block size and GIFT-128, 40-round with 128-bit block size.

Both use a 128-bit key. Also, lighter version, GIFT-64 found more vulnerable than GIFT-128[73][74]. Minimal documents have been found with the micro-controller implementation of GIFT[75][76].

SKINNY[77] has two versions: SKINNY-64 and SKINNY-128. SKINNY-64 uses a 64-bit block with 64/128/192-bit key variants to perform 32/36/40 rounds, whereas SKINNY-128 uses a 128-bit block with 128/256/384-bit key variants to perform 40/48/56 rounds.

RECTANGLE is an ultra-lightweight block cipher that can be used with various applications. With minor changes in SPN structure, the rounds are reduced to 25 (compared to 31 rounds in PRESENT) to meet with the competitive environment [71].

TWINE achieves good overall status as PRESENT and also overcomes many of its implementation issues. It operates 64-bit input with two key variants, 80-bit and 128-bit [78]. It requires around 2000 GE and a larger circuit size per throughput compared to AES [53]. In speed comparison, when 1KB or more ROM is available, AES is faster than TWINE, but when only 512bytes of ROM is available, AES can't be implemented and works 250% faster than PRESENT [53].

Midori was designed with a focus on low/tight energy budget, for instance, medical implants. It comes in two different versions, Midori64 and Midori128. Both of these use a 128-bit key on two different block sizes, 64-bit and 128-bit through 16 and 20 iterations, respectively [34][79].

mCrypton (miniature of Crypton) [80] is a cost and energy-efficient, lightweight edition of Crypton [81], suitable for both hardware and software deployments. It performs 13 iterations on the 64-bit block using a variety of keys (64-bit, 96-bit and 128-bit).

NOEKEON [82] works on the same block and key size, 128 bit, via 16 iterations. The NESSIE project rejected the cipher due to its less resistance against the attacks [83].

ICEBERG [84] is optimized for re-configurable hardware deployment with a property of modifying the key at each clock cycle without compromising quality. Here, the round keys are derived on the fly. It performs on 64-bit input with 128-bit key via 16 iterations with a demand of 5800 GE at a throughput of 400 Kb/s [85].

PUFFIN-2 [86] is a compact edition of PUFFIN (2303GE) [87]. It uses 80-bit key to perform 34 iterations on 64-bit data using serialized SPN structure. It requires only 1083 GEs for both encryption and decryption.

PRINCE is both hardware and software efficient lightweight algorithm [88] which performs on 64-bit input using a 128-bit key for 12 times [89]. The smallest hardware implementation demands 2953GE at a throughput of 533.3 Kb/s. It shows the low energy consumption of 5.53 $\mu J/bit$ [90].

PRIDE [88] exhibits low latency and low energy demand with a 128-bit key to perform 20 iterations on 64-bit input.

PRINT [91] is a domain-specific cipher designed for two applications: PRINT-48 for IC-printing applications which make use of an 80-bit key to perform 48 iterations on 48-bit input (402GE) and PRINT-96 for EPC encryption which uses a 160-bit key to perform 96 iterations on 96-bit input (726GE). Although it uses 3-bit operations where an odd number of bit operations is not feasible, the actual deployment of the algorithm is not ready yet.

Klein [92] works on 64-bit input using 64-bit, 80-bit and 96-bit keys through 12 (1220 GE), 16 (1478 GE), and 20 (1528 GE) iterations, respectively. It was designed with a focus on software implementation, mainly for sensors.

To obtain efficient hardware and software footprints, **LED** [93] borrows features from PRESENT (S-box), Lighter version of AES (row-wise data processing) [68] and PHOTON (mix column approach) [94]. There is an absence of key scheduling in LED, which is a unique feature. This approach reduces the chip area but increases the security risk like related key attacks [95]. It processes 64-bit input using various keys such as 64-bit (966 GE), 80-bit (1040 GE), 96-bit (1116 GE) and 128-bit (1265 GE) keys for either 32 or 48 times [93].

PICARO [96] is a novel cipher with a good balance between performance and security (by a good choice of S-box). It has four different masking levels with faster hardware performance compared to AES. In addition, it uses a 128-bit key through 12 rounds and shows high resistance to side-channel attacks.

Zorro [97] is based on AES, suitable for embedded systems and more efficient than PICARO. It takes a similar block and key size (128-bit) through 24 rounds.

EPCBC (Electronic Product Code Block Cipher) [98] is a lightweight cipher, inspired by PRESENT, supports a 96-bit key with the input of 48-bit and 96-bit block to perform 32 iterations. The most compact version needs 1008GE. In addition, the optimized sub-key generation technique of EPCBC enhances its immunity against related-key differential attacks.

I-PRESENT [99] is an involutive version of PRESENT inspired by PRINCE and NOEKEON. It takes a similar block and key size to perform 30 rounds with two additional 4x4 S-boxes (16 times). The most compact hardware implementation requires about 2769 GE (encryption and decryption).

ASCON [100] [101] is a sponge-based family of authenticated encryption and hashing algorithms, with two versions, ASCON-128 and ASCON-128a. Both versions use the 128-bit key on 64-bit and 128-bit data blocks. It performs 12 rounds for initialization and finalization permutation, whereas performing 6 and 8 rounds for the intermediate permutation. Moreover, they use 5-bit S-box in parallel over the state of 320-bits in a bit-slice manner.

Similar to ASCON, **SHAMASH** [102] is also a sponge based authenticated algorithm. It performs 12 rounds for initialization and finalization, whereas nine rounds for intermediate processing on 128-bit data using a 128-bit key and 128-bit nonce. In addition, it uses 5-bit S-box with little linearity and bit distribution difference compared to ASCON's S-box.

PRIMATE [103] is a sponge-based family of three modes of operation named APE, HANUMAN and GIBBON. It uses substitution-permutation networks (SPN). Two versions of PRIMATE, PRIMATE-80 and PRIMATE-120, work multiple times on 5x8 and 7x8 states of 5-bit elements, respectively.

ICEPOLE [104][105] is an authenticated encryption algorithm. It relies on the length of the key, a secret message number and nonce. There are two key lengths available, 128 bits and 256 bits. Both secret message number and nonce length vary from 0 and 128 bits. ICEPOLE operates a 5-bit S-box on 256 rows of 1280 states of plaintext.

Feistel Network (FN)

The lightweight DES (Data Encryption Standard) is known as **DESL**. It works on a similar block size (64-bit), key (56-bit) and a similar number of rounds as DES. The reduced number of S-box (eight to only one [106]) and multiplexer [107] used in DESL distinguishes it from DES. It demands 1850 GE which is 20% compact compare to DES (2310 GE) [107]. DESL also discards the initial and final permutation of DES to make it lighter [108]. **DESXL** is another lighter edition of DES with a key whitening feature to strengthen the cipher and with 2170 GE demands[107]. It performs the same number of cycles and uses the same block size as DESL, but a larger key, 184-bit (k=56, k1=64, k2=64) [108].

Tiny Encryption Algorithm (TEA) is suitable for very small, computationally weak and low-cost hardware [109]. It operates 128-bit key on 64-bit input to perform 32 rounds [110] with GE requirements of 3872 [111]. Its simple key scheduling is vulnerable to brute force attack [112][113]. Another limitation of the TEA structure is its three equivalent keys for decryption which makes it vulnerable to the attackers [112]. The improved version of TEA is **(XTEA)** which uses the same size of key and block but with more iterations (64 rounds), demanding 3490 GE [114]. It offers more complex key scheduling with little change in Shift, XOR and addition functions [115]. XTEA was further modified with **XXTEA** [116] to immune against related-key rectangle attack (on 36 rounds) [115].

Camellia [117] is an ISO/IEC, IETF, NESSIE and CRYPTREC recognised cipher. It was designed by Nippon Telegraph and Telephone Corporation, and Mitsubishi Electric Corporation. Camellia offers a similar level of security by processing the same size of key and block as AES with two round variants, 18 and 24. It is known for its fast software implementations [118] whereas the hardware implementation requires 6511 GE.

NSA designed **SIMON** [119], which is known for its small footprint in hardware. It offers various keys of size (64-bit, 72-bit, 96-bit, 128-bit, 144-bit, 192-bit, 256-bit) over the block of 32-bit, 48-bit, 64-bit, 96-bit, 128-bit through 32, 36, 42, 44, 52, 54, 68, 69, 72 rounds [119]. The most compact version requires 763GE for execution [119].

SEA [120] is designed for tiny IoT devices, especially for memory-constrained devices [121], with the concept of on-the-fly key generation [120]. It uses 96-bit key on two recommended block size 96-bit and 8-bit with the requirement of 3758GE [121] for the most lightweight hardware version. The optimised software execution demands 426 bytes with encryption cycle of 41604 on 8-bit micro-controllers [122].

KASUMI [123] takes 64-bit input to performs 8 iterations using a 128-bit key. It demands 3437GE for deployment on hardware [124]. It is mainly designed for GSM, UMTS and GPRS systems.

MIBS [125] takes 64-bit input to perform 32 iterations using two variants of keys, 64-bit (1396 GE) and 80-bit (1530 GE). It is Feistel based structure, makes use of S-box from mCrypton [80] and uses PRESENT's keys extraction technique to derive the sub-keys.

LBlock [126] is an ultra-lightweight cipher, performs 32 iterations on 64-bit input along with 80-bit keys. The smallest hardware deployment needs 1320 GE for a throughput of 200 Kb/s, whereas the most efficient software implementation takes 3955 clock cycles to encrypt a single block (on the 8-bit microcontroller).

Designed and developed by the government of the Soviet Union (1989), the lightweight version of **GOST** executes 32 times on 64-bit input with a 256-bit key. The S-Box in this version is adopted from PRESENT [127] with the demands of 651 GE.

ITUbee [128] is a software efficient cipher with a code size of 586 bytes and 2937 cycles (the most compact version of encryption). It takes the exact size of the key and block (80-bit). Here, key scheduling is replaced by round-dependent constants to reduce software overload.

FeW [129] processes 64-bit input with two varieties of the key, 80-bit and 128-bit for 32 times. It makes use of the S-box of Humminbird-2 and follows the key expansion process from PRESENT. There no cryptanalytic attack found on FeW [129].

Generalised Feistel Network (GFN)

Introduced by SONY corporation and approved by NIST, **CLEFIA** offers 128-bit block with choice of 128, 192, 256 bit key through 18, 22, 26 round, respectively [65][130]. It shows high performance and strong immunity against various attacks [31][65][131][132] with comparative high cost as the most compact version requires 2488 GE (encryption only) for 128-bit key [130]. The strong immunity of CLEFIA against security attacks is grateful to

its dual confusion and diffusion properties. On the contrary, this demands higher memory and limits its use in ultra-small applications [30].

Piccolo [133] is another ultra-lightweight cryptography algorithm suitable for extremely restricted environmental devices (RFID, sensors, etc.). It processes 64-bit input to perform two iterations, 25 and 31, using two key sets, 80-bit and 128-bit, respectively. The smallest hardware deployment (80-bit key) requires 432 GE and additional 60 GE to perform decryption.

TWIS [134], derived from CLEFIA, takes equal size block and key (128-bit) to perform 10 iterations. It is a victim of differential distinguisher with probability one [135].

TWINE [78], derived from LBlock, performs 36 iterations on a 64-bit state along with two key options, 80-bit and 128-bit. The most compact hardware implementation requires 1866 GE. TWINE uses nibble permutation instead of bit permutation (for sub-key generation) of LBlock. Also, it uses a single S-box instead of ten S-Boxes of LBlock.

HISEC [136] performs 15 iterations on 64-bit input along with an 80-bit key, demanding 1695 GE. It shows good resistance against different attacks, and the characteristics are more like to PRESENT except bit-permutation.

Add-Rotate-XOR (ARX)

SPECK [119], sibling of SIMON and designed by NSA, is a software-oriented cipher. It supports the similar size of blocks and keys as SIMON to perform 22, 23, 26, 27, 28, 29, 32, 33 and 34 iterations. The most compact hardware implementation recorded uses a 48-bit block with a 96-bit key with requirements of 884 GE, whereas the most efficient software implementation requires 599 cycles with 186-byte of ROM for a 64-bit block with 128-bit key [119].

IDEA [137], designed by Lai and Massey, makes use of a 128-bit key on 64-bit input to perform 8.5 iterations, mainly used for high-speed networks [138]. It uses 16-bit unsigned integer and performs data operations such as XOR, addition and modular multiplication without using S-box or P-box. It is known for its best performance on embedded systems (such as PGP v2.0.) with memory needs of 596 bytes at a throughput of 94.8 Kb/s (the smallest software version) [139].

HIGHT [140], an ultra-lightweight algorithm, processes 64-bit data using a 128-bit key for 32 times. It performs compact round function (no S-boxes) using simple computational operations. The most compact version acquires 2608 GE for 188 Kbps throughput [141].

BEST-1 [142], an ultra-lightweight cipher, targets Wireless Sensor Networks and RFID tags. It takes 64-bit input with a 128-bit key through 12 rounds on 8-bit processors, demanding

2200 GE. The core functions of BEST-1 are mod 2^8 addition and subtraction, bitwise shift and XOR.

LEA [143] is a software-oriented cipher and was introduced by the ETRIK for 32-bit common processor. It processes 128-bit input to perform 24, 28, and 32 iterations using 128-bit, 192-bit and 256-bit keys. On the ARM platform, LEA performs 326.94 cycles/byte with a storage demand of 590 bytes (code) and 32 bytes for execution. The most compact version requires 3826 GE for 76.19 Mbps throughput [144].

NonLinear-Feedback Shift Register (NLFSR)

With focus on automobile industry, **KeeLoq** [41] is designed with an aim to keyless authentication (remote access) in cars [145] by Gideon Kuhn. It takes 32-bit input with a 64-bit key to perform 528 rounds. Even though developed in the '80s, the cryptanalysis report of KeeLoq was issued in February 2007 for the first time by Bogdanov [146].

KATAN/KTANTAN [147], inspired by KeeLoq, cipher family applies 80-bit key on various block size (32-bit, 48-bit and 64-bit) through 254 iterations. They could be executed on small-scale hardware (KATAN 802 GE and KTANTAN 462 GE), mainly designed for RFID tags and sensor networks. They follow a linear structure (LFSR) instead of the NLFSR of KeeLoq. KATAN has straightforward key scheduling compared to KeeLoq, whereas KTANTAN exhibits no key generation operations (reduce GE requirement). As the key remains unchanged once initialized, the applications of KTANTAN is limited. KTANTAN-48 (588 GE) is more appropriate for RFID tags. In software, both shows poor performance (low throughput and high energy consumption) due to overuse of bit manipulation [122].

Halka [148] performs well on both hardware and software. It takes 64-bit input with an 80-bit key to perform 24 iterations. The multiplicative inverse based S-boxes (8-bit) with LFSR makes Halka more secure than PRESENT. It demands 138 GE (7% less GE than PRESENT) [148]. Also, the software performance is three times more efficient than PRESENT [148].

Hybrid

Hummingbird [149] is an ultra-lightweight algorithm, introduces a hybrid structure (block and stream). It takes 16-bit input with a 256-bit key to perform 20 iterations. It was vulnerable to several attacks [150].

Hummingbird-2 [151], designed for low-end microcontrollers, takes 64-bit input (initial vector) with a 128-bit key. It performs well on both platforms (hardware/software). It also satisfies the ISO 18000-6C protocol. It gives better performance compared to PRESENT (on

4-bit micro-controllers). Still, it has a few drawbacks: 1) Initialization is necessary before encryption (or decryption) due to its stream property 2) Different encryption and decryption functions and due to that full version is 70% heavier than only encryption. Moreover, its performance degrades while processing small messages.

PRESENT-GRP [30] works on 64-bit input with a 128-bit key to perform 31 iterations. It makes use of the substitution-permutation technique from PRESENT along with a group(GRP) operation for additional confusion properties (in replacement of permutation table). The hardware implementation of PRESENT (1884 GE) is slightly better than PRESENT-GRP (2125 GE). Similarly, PRESENT is more efficient than PRESENT-GRP in software implementation too.

2.3 Performance Comparison of existing LWC algorithms

Various experiments have been carried out by many researchers using different platforms such as NXP [30], AVR [152], ARM [30] micro-controllers to evaluate the performance of the popular lightweight cryptography algorithms [30][31][54][68][69][107][152][153]. During these experiments, various characteristics such as area (GE), logic process (μm), power consumption (μW), throughput, RAM/ROM (*bytes*) requirements, latency (*cycle/block*), etc. have been compared for different lightweight cryptography algorithms in different circumstances (file types (C/C++, Java, Python), message size, etc.). Table 2.3 summarizes the hardware and software performance of the listed LWC algorithms evaluated on 0.09/0.13/0.18/0.35 μm technologies (hardware implementation) and on 8/16/32 bit micro-controllers (software implementation) platforms.

Table 2.3 reveals that KTANTAN requires a minimal area to implement, only 462 GE, followed by PRINT, SIMON, KATAN and SPECK. Also, LED, PICCOLO, RECTANGLE, MIDORI and PRESENT show competition in this hardware performance race. Overall, this study observes the ups and downs in the area, power and throughput of these LWC algorithms. However, SPECK and SIMON are the best software performers, documenting the highest software efficiency, 3511.19 Kbps/KB and 1900 Kbps/KB, respectively. Not only this but they are also known for their lowest latencies, 408 cycles/ block and 594 cycles/ block, respectively.

Table 2.3 Hardware and Software performances of LWC algorithms

LWC Algorithm	Hardware Implementation								Software Implementation							
	Key Size	Block Size	Tech (μm)	Area (GE)	Power (μW)	Energy ($\mu J/bit$)	Throughput @100KHz (Kbps)	Hardware Efficiency (Kbps/KGE)	Key Size	Block Size	ROM (byte)	RAM (byte)	Latency (Cycles/ block)	Energy ($\mu J/bit$)	Throughput @4MHz (Kbps)	Software Efficiency (Kbps/KB)
AES*	128	128	0.13	2400	2.4	42.38	56.64	23.6	128	128	918	0	4192	16.7	122	132.9
PRESENT*	80	64	0.18	1570	2.35	11.77	200	127.38	128	64	660	0	10792	43.1	23.7	35.91
RECTANGLE	80	64	0.13	1467	1.46	5.96	246	167.68	-	-	-	-	-	-	-	-
MIDORI	128	64	0.09	1542	60.6	1.61	400	259.4	-	-	-	-	-	-	-	-
mCrypton*	128	64	0.35	2594	4.66	138.61	33.51	12.91	96	64	1076	28	16457	68	15.5	14.41
NOEKEON*	128	128	0.35	2604	4.68	1362.21	3.44	1.32	128	128	364	32	23517	95.9	21.7	59.62
ICEBERG	128	64	0.18	5817	8.72	21.81	400	68.76	-	-	-	-	-	-	-	-
PUFFIN-2	80	64	0.18	1083	1.62	314.74	5.2	4.8	-	-	-	-	-	-	-	-
PRINCE*	128	64	0.13	2953	2.95	5.53	533.3	180.59	128	64	1108	0	3614	14.4	70.8	63.9
PRIDE*	-	-	-	-	-	-	-	-	128	64	266	0	1514	6	169	635.34
PRINT^	80	48	0.18	503	0.75	7.54	100	198.8	80	48	6210	48	87272	117.8	2.2	0.35
Klein*	64	64	0.18	1220	1.83	59.18	30.9	25.32	64	64	2980	50	7901	10.6	32.4	10.87
LED#	64	64	0.18	966	1.45	282.55	5.1	5.27	80	64	2164	368	35161	-	7.28	3.36
I-PRESENT	80	64	0.18	2467	370	-	-	-	-	-	-	-	-	-	-	-
EPCBC	96	48	0.18	1008	1.51	124.74	12.12	12.02	-	-	-	-	-	-	-	-
DESL*	56	64	0.18	1848	2.77	62.37	44.4	24.02	56	64	3098	0	8365	33.4	30.6	9.88
TEA*	128	64	0.18	2355	3.53	35.32	100	42.46	128	64	648	24	7408	30.3	34.5	53.24
XTEA*	-	-	-	-	-	-	-	-	128	64	504	0	17514	70	14.6	28.97
Camellia*	128	128	0.18	6511	9.76	33.57	290.1	44.55	128	128	1262	12	64000	256	8	6.34
SIMON*	96	48	0.13	763	0.76	48.32	15.8	20.7	96	48	170	0	594	2.3	323	1900
SEA*	96	8	0.13	2562	2.56	1117.67	2.29	0.89	96	96	426	24	41604	173.7	9.2	21.6
KASUMI*	128	64	0.13	3437	3.44	29.9	115.14	33.5	128	64	1264	24	11939	47.6	21.4	16.93
MIBS^	64	64	0.18	1396	2.09	10.47	200	143.26	64	64	3184	29	49056	66.2	5.2	1.63
LBlock^	80	64	0.18	1320	2	9.9	200	151.51	80	64	976	58	18988	25.6	13.48	13.81
ITUbee*	-	-	-	-	-	-	-	-	80	80	716	0	2607	10.4	122.7	171.37
GOST^	256	64	0.18	1000	1.5	7.5	200	200	256	64	4748	190	10240	13.8	25	5.27
Robin^	-	-	-	-	-	-	-	-	128	128	1942	80	4935	6.6	103.74	53.42
Fantomas^	-	-	-	-	-	-	-	-	128	128	1920	78	3646	4.9	140.42	73.14
CLEFIA*	128	128	0.13	2678	2.67	36.82	76	28.37	128	128	3046	0	28648	114.5	17.8	5.84
PICCOLO^	80	64	0.13	1136	1.13	4.8	237.04	208.66	80	64	966	70	21448	28.9	11.93	12.35
TWINE#	80	64	0.09	1503	1.05	5.91	178	118.42	80	64	1180	140	20505	-	12.48	10.58
SPECK*	96	48	0.13	884	0.88	73.67	12	13.57	96	48	134	0	408	1.6	470.5	3511.19
IDEA*	-	-	-	-	-	-	-	-	128	64	596	0	2700	10.8	94.8	159.06
HIGHT*	128	64	0.35	2608	4.7	24.93	188	72.08	128	64	5718	47	6377	25.5	40.14	7.02
LEA#	128	128	0.13	3826	3.82	50.22	76.19	19.91	128	128	590	32	5231	-	97.8	165.76
KATAN*	80	32	0.13	802	0.8	64.16	12.5	15.58	80	64	338	18	72063	289.2	3.5	10.35
KTANTAN^	80	32	0.13	462	0.46	36.96	12.5	27.05	80	32	10516	614	10233211	13814.8	0.012	0
Hummingbird^	-	-	-	-	-	-	-	-	128	16	1822	82	4637	6.2	13.8	7.57
Hummingbird-2^	128	16	0.18	2159	3.23	40.48	80	37.05	128	16	770	50	1520	2	42.1	54.68

* 8-bit microcontroller, ^16-bit microcontroller, # 32-bit microcontroller

According to the graph (Figure 2.3), software efficiency competition is won by SPECK, followed by SIMON and then PRIDE. Also, ITUbee, LEA, IDEA, and AES show better software efficiency than the other LWC algorithms.

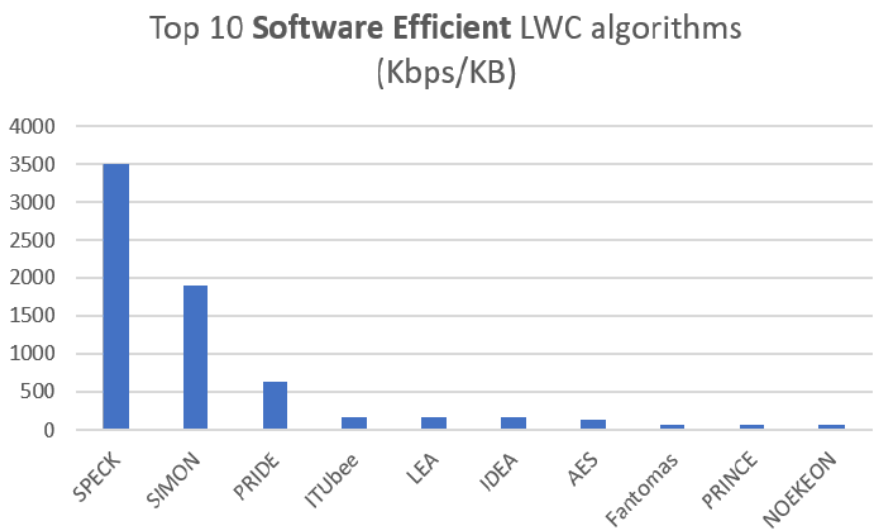


Fig. 2.3 Software Efficient LWC algorithms (Top 10)

Memory (RAM and ROM) requirements by various LWC algorithms can be studied from the above graph (Figure 2.4), which reveals the top ten memory-efficient LWC algorithms. The competition is again won by SPECK and SIMON with less than 200 bytes of ROM and zero bytes of RAM requirement, closely followed by PRIDE.

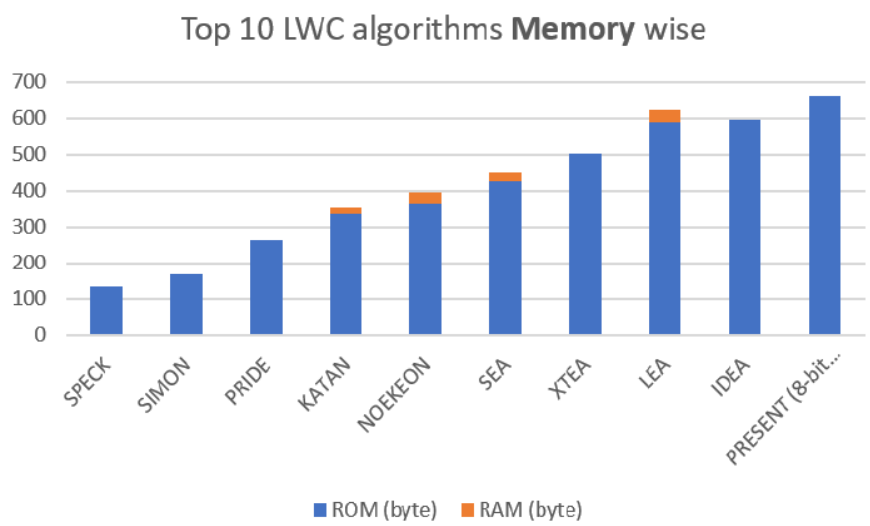


Fig. 2.4 Memory Efficient LWC algorithms (Top 10)

Another two crucial software metrics, latency and throughput, led by again SPECK and SIMON with the lowest latency rate, 408 cycles/block and 594 cycles/block, respectively, and highest throughput, 470.5 Kb/s and 323 Kb/s, respectively, unceasingly followed by PRIDE. Also, ITUbee and IDEA secure their places in the list of the top ten performers with almost 7-times high latency compared to SPECK (Figure 2.5).

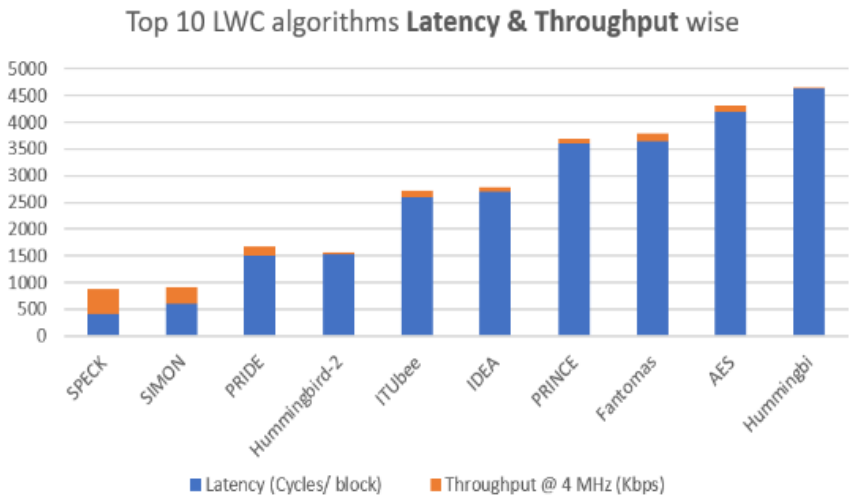


Fig. 2.5 Latency Efficient LWC algorithms (Top 10)

Midori is on the top of the list in the race of hardware efficiency, 259.4 Kbps/KGE, followed by PICCOLO, as runners-up with a minor difference of 8.66 Kbps/KGE with GOST. One of the standard LWC algorithms, PRESENT, documents half efficiency compared to the top one in this race. Figure 2.6 visualizes the first ten hardware efficient LWC algorithms.

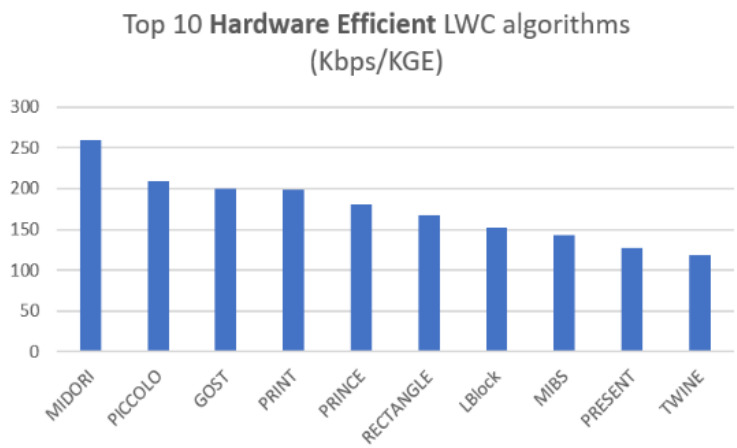


Fig. 2.6 Hardware Efficient LWC algorithms (Top 10)

From the graph (Figure 2.7), SEA leads the key and block-wise (hardware efficiency) competition with tiny block size (only 8-bit), followed by Hummingbird-2 with a double-sized block (and the largest key in this top-10 list). Further, KATAN/KTANTAN has four times bigger blocks than the leader. List accommodates PRINT, EPCBC, SIMON/SPECK, PRESENT and RECTANGLE with either 48-bit or 64-bit block along with 80-bit or 96-bit key, respectively.

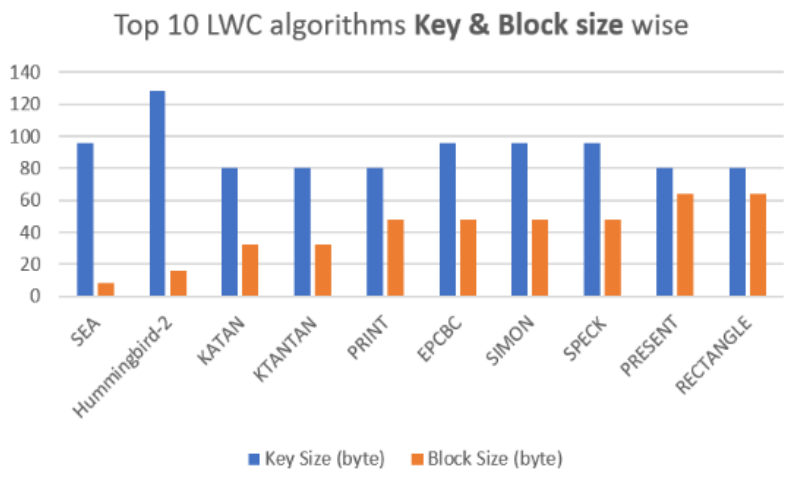


Fig. 2.7 Block & Key size wise Hardware Efficient LWC algorithms (Top 10)

From the graph (Figure 2.8), we can say that KTANTAN demands the smallest area (462 GE) to implement, with a minor difference from PRINT (41GE more). SPECK/SIMON shows their presence in the top 5 lists with less than 900GE needs. All of these performances are noticed either on 0.13 μm or 0.18 μm technologies.

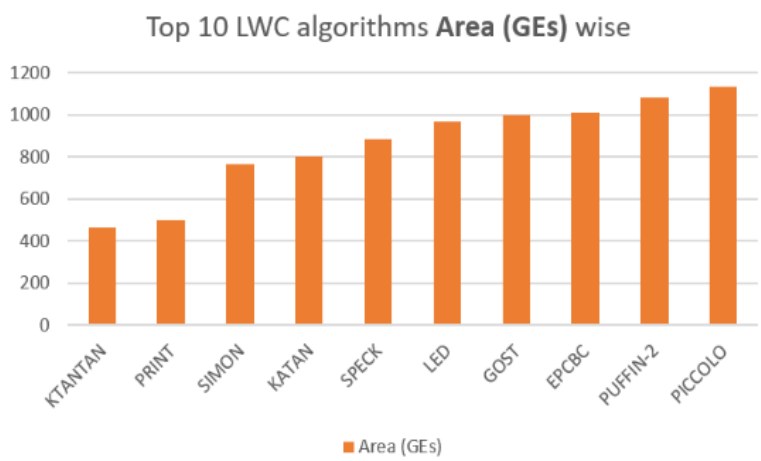


Fig. 2.8 Physical Area wise Hardware Efficient LWC algorithms (Top 10)

In terms of energy consumption, Midori shows the lowest energy requirement (only 1.61 $\mu J/bit$), followed by Piccolo, PRINCE, TWINE and RECTANGLE with more than double energy requirements. Here, PRESENT, placed in the tenth position, demands almost six times more energy than the top performer. Figure 2.9 depicts the top 10 hardware efficient LWC performers by their energy demand.

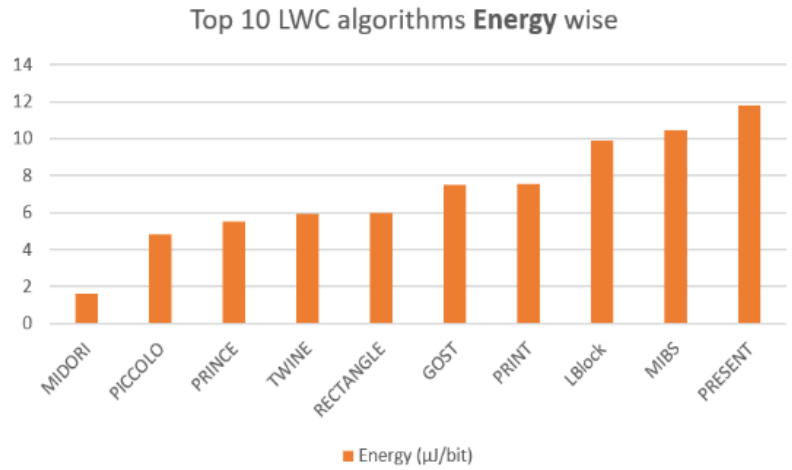


Fig. 2.9 Energy Efficient Hardware Efficient LWC algorithms (Top 10)

In summary, SIMON and SPECK shine by their most efficient software implementation but disappears from the top-10 list of hardware efficient LWC algorithms. Also, derived versions of AES such as PRESENT and derived lighter versions of DES such as DESL/DESLX and CLEFIA are widely recognised algorithms (by the standardising bodies) due to high-security reasons. However, overall, none of the LWC algorithms meets all the efficiency metrics of the hardware and software requirements and shows distinct performances in different circumstances.

2.4 Cryptanalysis of existing LWC algorithms

Security is an essential measure for any lightweight cryptography algorithm, along with performance and cost. The attack resistance property of any lightweight cryptography algorithm can be measured through cryptanalysis. Cryptanalysis aims at detecting algorithm vulnerabilities by attempting various attacks, and decryption techniques [69]. The main types of cryptanalysis on block cipher are nonlinearity, linear approximation probability (LP), differential approximate probability (DAP), degree of avalanche effect (SAC), bit independence criteria (BIC) and algebraic attacks. Differential approximate probability

(DAP) or simply differential cryptanalysis [154] is an analysis of outputs against various inputs. The particular types are higher-order, truncated, impossible and boomerang attack (BCT/FBCT). Linear cryptanalysis (Linear approximation probability (LP)) postulates a linear approximation based on the piling-up lemma principle (introduced by Mitsuru Matsui [155]) between plaintext, ciphertext and key by characters or individual bits. Nonlinearity is a measure of any correlations between the input value and the corresponding output value. It can be measured either using Hamming distance or using the Walsh matrix. The higher the value (nonlinearity property), the higher the security. Algebraic cryptanalysis is based on equation-solving algorithms and has been proven effective on lightweight versions due to its simple structure (fewer rounds with less algebraic complexity).

These cryptanalyses are based on Ciphertext only, Known plaintext, Chosen plaintext and Chosen ciphertext along with MITM, Brute force and side channel. Differential Fault Attacks, a type of side-channel attack, analyzes the internal structure and finds exploitable places to attack the algorithm [156][157].

Table 2.4 demonstrates the security analysis of various existing LWC algorithms in a grid form. The study shows that almost all existing lightweight block cipher solutions suffer from multiple attacks, especially related-key attacks, followed by various differential and MITM attacks. Moreover, the lighter versions (with reduced rounds) are more vulnerable to numerous attacks than standard ones. Contrarily, most algorithms show good immunity against linearity and algebraic properties. The study observed that AES, PRESENT, GIFT, MIDORI, SIMON, SPECK, LBlock, KeeLoq and HIGHT are the most analyzed LWC algorithms against the various cryptanalysis by the researchers. On the contrary, mCrypton, NOEKEON, ICEBERG, PUFFIN-2, PRINCE, PRINT, TEA, XTEA, XXTEA, ITUbee, IDEA, KATAN, KTANTAN, Hummingbird-2 are the least evaluated LWC algorithms.

Table 2.4 Security Analysis of LWC Algorithms

LWC Algorithm	Differential Cryptanalysis*	Linear Crypt-analysis	Integral/Square/ Saturation Cryptanalysis	Algebraic/Cube Cryptanalysis	MITM/ Biclique	Related Key attack	Side-Channel/ Differential fault attacks
AES	✓ [158]	-	-	-	✓ [68]	✓ [68]	✓ [159][160][161]
PRESENT	✓ [162][163]	-	-	-	✓ [47]	✓ [164]	✓ [159][161][165][166]
GIFT	✓ [167][74]	-	✓ [168]	-	✓ [72][168][75]	✓ [169][74][170]	✓ [171]
SKINNY	✓ [172]	-	-	-	-	✓ [170]	✓ [161][173][174]
RECTANGLE	-	-	✓ [71]	-	-	✓ [71]	✓ [71]
MIDORI	✓ [34]	✓ [34]	-	-	✓ [34]	-	-
mCrypton	-	-	-	-	-	✓ [175]	-
NOEKEON	-	-	-	-	-	✓ [83]	-
ICEBERG	✓ [176]	-	-	-	-	-	-
PUFFIN-2	✓ [177]	-	-	-	-	-	-
PRINCE	✓ [178]	-	-	-	-	-	-
PRINT	-	-	-	-	-	✓ [179]	-
Klein	-	-	-	-	✓ [180]	✓ [181]	✓ [182]
LED	✓ [183]	-	-	-	✓ [47]	✓ [95]	-
EPCBC	-	-	-	✓ [42]	-	✓ [42]	-
TEA	-	-	-	-	-	✓ [112][113]	-
XTEA	-	-	-	-	-	✓ [115]	-
XXTEA	✓ [184]	-	-	-	-	-	-
Camellia	✓ [118]	-	-	-	-	-	✓ [43][160]
SIMON	✓ [48][185]	-	-	✓ [49]	-	✓ [49]	-
KASUMI	✓ [37]	-	-	-	-	✓ [38]	-
MIBS	✓ [186]	✓ [186]	-	-	-	-	-
LBlock	✓ [44][45]	-	✓ [78]	-	✓ [187]	-	-
ITUbee	-	-	-	-	-	✓ [188]	-
GOST	-	-	-	-	✓ [189]	✓ [190]	-
CLEFIA	-	-	✓ [130]	-	-	✓ [130]	✓ [160]
PICCOLO	✓ [191]	-	-	-	✓ [47][192]	-	-
TWIS	✓ [135]	-	-	-	-	-	-
TWINE	-	-	✓ [108]	-	✓ [193]	-	-
SPECK	✓ [185][194]	-	-	-	-	✓ [194]	-
IDEA	-	-	-	-	✓ [36]	-	-
HIGHT	✓ [164]	✓ [35]	-	-	✓ [192]	✓ [195]	-
LEA	-	-	-	-	-	-	✓ [46]
KeeLoq	-	✓ [146]	-	✓ [145]	✓ [196]	-	✓ [146]
KATAN	-	-	-	-	✓ [197]	-	-
KTANTAN	-	-	-	-	-	✓ [39]	-
Hummingbird-2	-	-	-	-	-	✓ [198]	-
Hummingbird	Vulnerable to several attacks [150]						

*It includes Truncated/Higher-order/Impossible/Boomerang Differencial Cryptanalysis

2.5 Real-time Applications and their Lightweight demands

The wide range of IoT applications in various fields creates the demand for lightweight cryptography algorithms with different requirements [199]. Table 2.5 gives a brief on real-time IoT application along with its lightweight requirements and the best suit LWC algorithms as follows:

Table 2.5 Real-time IoT Applications and their lightweight demands

IoT Application	Lightweight requirements	Best suit LWC algorithms
Smart Home Appliances	Less CPU Time, Smaller ROM	SIMON, SPECK, Piccolo, TWINE
Smart Health Care	Low power consumption, low CPU cost, Small circuit size	SIMON, SPECK, Picollo, PRESENT, Midori
Smart Industries	Real-time processing	AES, Midori, PRINCE
Smart Logistics	Small circuit size, low power consumption	SIMON, SPECK, Piccolo, PRINCE
Smart Automobiles	Small circuit size, lower latency	Keeloq, Midori, PRINCE, PRESENT, SIMON
Smart Agriculture	Compact implementation, less processing cycles, low power consumption	SIMON, SPECK, PRESENT, TWINE

Smart home appliances such as smart TV, smart fridge, smart kettle, smart bulbs, etc., demands small memory and small processing. The best-suited algorithms in this scenario are SIMON, SPECK, PICCOLO and TWINE.

A person under medical treatment in a hospital or residence could be monitored for pulse count, pressure level, sugar and oxygen in the blood using IoT sensors. Here, security and privacy of the transmitting data are crucial, along with tiny circuitry, little processing power, limited batteries (in case of an implanted device), and quick response time. In this constrained environment, SIMON, SPECK, PICCOLO, PRESENT and Midori are the best suit solutions to secure the communication in **health care applications** due to their overall compact hardware and software implementation to match with real-time response while in-body and/or out-body (wearable) implantation.

For **industrial systems (Industry 4.0)**, sensors could be attached to equipment at various places (not easily accessed by the operators) to transmit the data wirelessly for specific distances. In this state, real-time processing is the critical element with adequate security (without bothering about energy consumption). As a result, Midori and PRINCE show the best performance in a demanding scenario.

Due to tiny physical space and a little or no power backup in RFID tags, SIMON, SPECK, Piccolo and PRINCE are the best options for **logistics applications**.

In an era of 5G technology, **automobile industry** demands not only in-vehicle communication but also communication with infrastructures such as traffic signals and road signs (V2X). This communication requires a prompt response (low latency) on a tiny circuitry with high security. Midori, PRINCE, PRESENT, and SIMON are the right choices for auto industries. Keeloq is another powerful LWC algorithm for secure remote keyless entry in cars and buildings [196].

Nowadays, **smart agriculture** is an emerging field that demands compact implementation, fewer processing cycles, little power consumption with plenty of sensors in a remote location. SIMON, SPECK, PRESENT and TWINE fulfil the requirements of smart agriculture.

2.6 Research Gaps

An ideal algorithm should show the trade-off between cost, performance and security. Any two of these three can be easily optimized, whereas achieving all of these together is challenging [69]. For example, an increasing number of rounds [194] or key size results in degradation of algorithm performance. These could be achieved by design focus on less memory and less computing power requirement, leading to less Gate Equivalent (physical area) requirements along with low power (energy) consumption without compromising strong security [30]. Since S-box is a fundamental and the only component that offers confusion property and further ensures nonlinear in any SPN-based cryptography algorithm, it takes the primary focus while designing any cryptography algorithms.

Based on the above study, we have identified the following research issues, which require further attention to make the LWCs algorithms effective in IoT security:

1. One of the two fundamental properties of cryptography, confusion, could be achieved by choosing an efficient and adequate number of S-boxes to demonstrate a proper balance between performance and security [96]. So designing simple and fast but strong confusion (Substitution, S-box) and diffusion (Bit Permutation) properties with the right balance amongst cost, performance, and security is of practical interest. For example, How to reduce the number of S-boxes as they increase the demands for

memory (to store) and computing power (to produce) while maintaining the same security level? (motivation: PRESENT is designed from AES and replaces eight S-boxes with just one. Similarly, many researchers have derived the lighter versions from the standard cryptography algorithms with a few modifications by reducing substitution-permutation (counter-effect on security level)). But how to replace S-boxes with some other confusion techniques with the same level of security and less overhead of memory and processing cost is still an open problem.

2. Making key scheduling lighter with smaller key size and adequate strength, i.e., How to generate random sub-keys from the provided initial key for all n rounds?

3. An increase in the number of rounds adversely affects the performance and cost, i.e., How to decrease (or increase) the number of rounds without compromising performance as well as security level?

A concern from the above-discussed challenges is how to fulfil all three lightweight characteristics (cost, performance, and security) into a single LWC algorithm. Therefore, it encourages rethinking and redesigning the substitution-permutation network (i.e., S-Box and permutation technique).

Chapter 3

Research Methodology

This experiment-based research aims to develop a novel lightweight cryptography algorithm to trade-off cost, performance, and security characteristics. This chapter discusses the detailed structure of the proposed LWC algorithm, AUM, with its primary modules: permutation through transpose, addRoundKey to apply one of the subkeys, substitution using 5-bit S-box and subkeys generation from the original key provided by the user. The research questions about new S-box design, key scheduling and number of rounds, raised in an earlier chapter, section 2.6, are answered in this chapter. Finally, the chapter ends with the pseudo-code of the proposed algorithm.

3.1 Proposed LWC algorithm: An Overview

The proposed algorithm, AUM, targets IoT devices such as RFID tags, smart cards, sensors, actuators where message sizes are generally small (message size < 2Kb), it uses a 32-bit key over the same bit size block. It performs 16 rounds to produce the cipher (Figure 3.1).

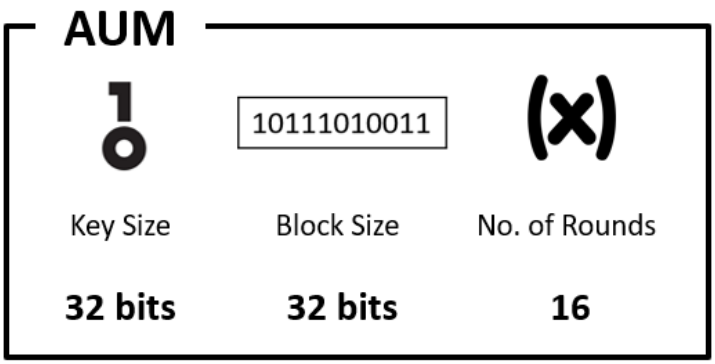


Fig. 3.1 Brief of the proposed LWC algorithm (AUM)

The algorithm is designed carefully with the novelty of features such as an eccentric permutation technique (transpose) to create diffusion [27] as well as a novel 5-bit substitution box to develop the confusion property [27]. In addition, it offers a unique way to create 16 subkeys from the original one. Here, the number of rounds is reduced to 16 compared to 32 in [64] to balance the performance and cost ratio, consequently affecting the subkey generation operation in a lightweight manner. On the contrary, security is maintained by offering more bit-size S-box (5-bit instead of 4-bit).

Three main functions perform in each round are as follows:

1. Permutation (Transpose)
2. AddRoundkey
3. Substitution using 5-bit S-box

Figure 3.2 illustrates the structure of AUM, in brief, using the functions mentioned above.

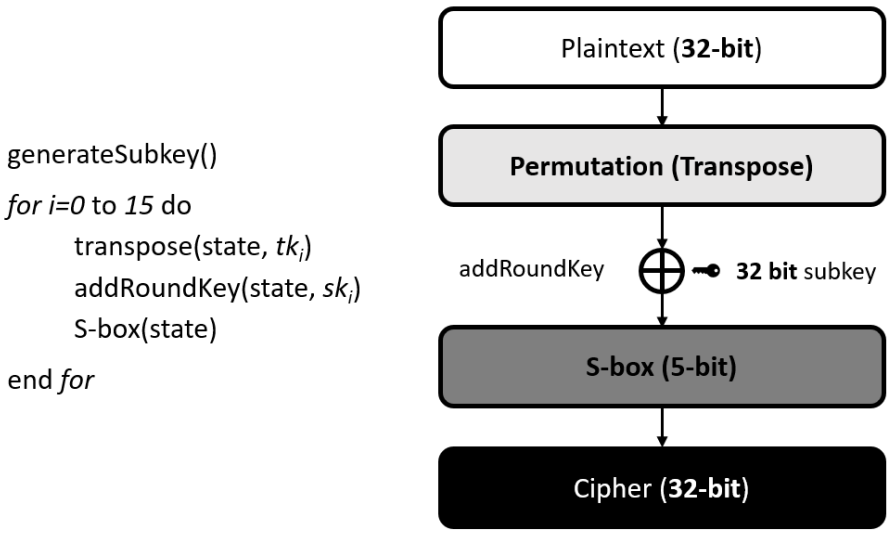


Fig. 3.2 General structure of AUM

3.2 Proposed LWC algorithm: Detailed Structure

Figure 3.3 depicts the full functioning of the proposed algorithm, AUM. The three core functions performed in every round are discussed in detail in the following subsections, subsection 3.2.1, subsection 3.2.2, subsection 3.2.3, along with a unique technique to create 16 subkeys in subsection 3.2.4.

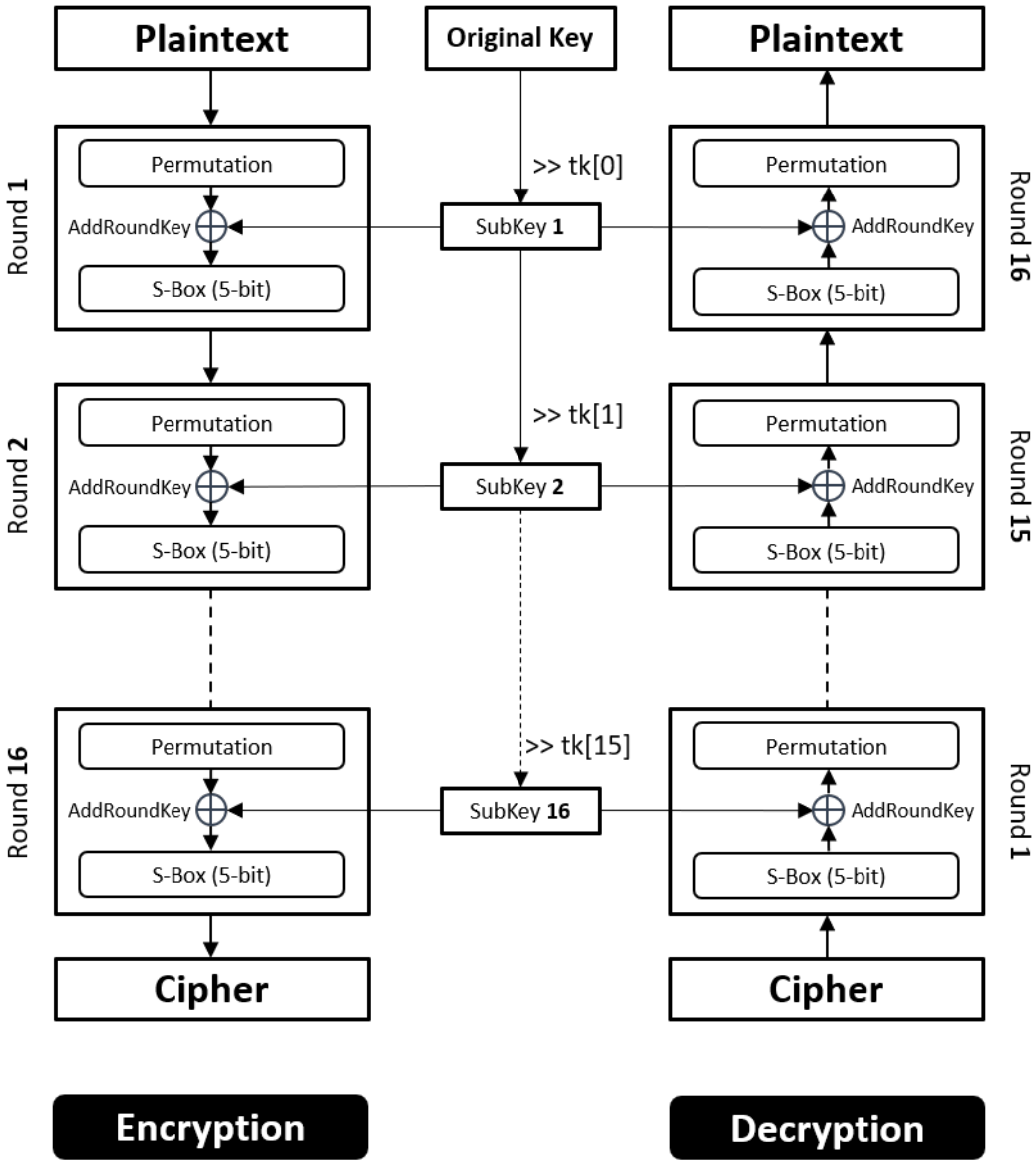


Fig. 3.3 Full functioning of the proposed algorithm (AUM)

3.2.1 Permutation through Transpose

For enhanced security, Claude Shannon proposed two essential features of any cryptography algorithm: confusion and diffusion [27]. One of the key aspects, diffusion (permutation), which disseminates the structure of the plaintext over the ciphertext, is effectively achieved through the transpose technique in AUM. It uses a 2-D array to disperse the 32-bit of the plaintext using a random transpose key. The 32-bit plaintext input is spread over two rows and sixteen columns according to the transpose key value, tk_i.

Let's understand the transpose in detail by taking the following example:

$$Transpose\ Key: 7\ 12\ 9\ 14\ 3\ 5\ 11\ 8\ 2\ 13\ 4\ 10\ 15\ 0\ 1\ 6$$

Now, place each bit of plaintext according to the transpose key value, tk_i, in a column wise manner (figure 3.4), where $0 \leq tk_i \leq 15$.

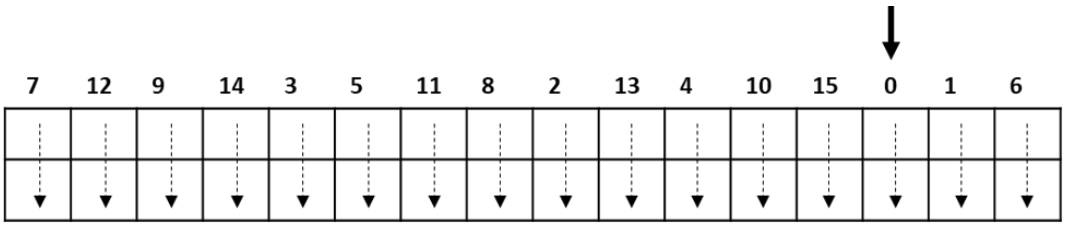


Fig. 3.4 Filling plaintext bits column wise into 2-D transpose

The above 2-D array will be now read in a row-wise manner to increase the diffusion level (figure 3.5). Thus the output from the transpose will be, P_{14}, P_{24}, P_{18}, P_{28}, P_6, ... P_{31}, P_1, P_3, P_{13}.

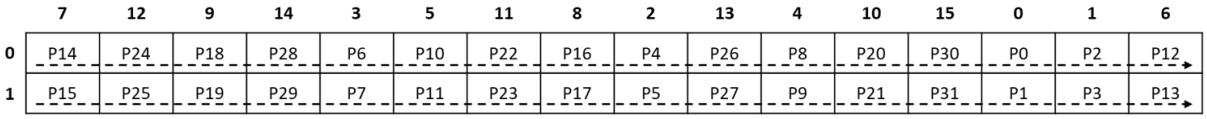


Fig. 3.5 Reading the bits from 2-D array row wise

The process could be presented in the form of an equation, T_i, which is based on two independent variables j and n as follows:

$$T_i = P_{tk_j * 2 + n} \tag{3.1}$$

$$\text{where } i = 0, 1, 2, ...31, \quad j = \begin{cases} i & for\ i < 16 \\ i - 16 & for\ i \geq 16 \end{cases} \quad and$$

$$n = \begin{cases} 0 & for\ i < 16 \\ 1 & for\ i \geq 16\ and\ i < 31 \end{cases}$$

Here, the transpose key value, tk, is random and ranges from 0 to 15. Instead of having the same number of transpose/permutation values (like in traditional permutation) to rearrange the inputs, the 2-D array transpose design of AUM reduces the same to half $(32/2)$. It reduces the resource requirement [64][68], only 16 values instead of 32, and makes the permutation lighter by providing almost the same level of security. Figure 3.6 depicts the permutation process as a whole.

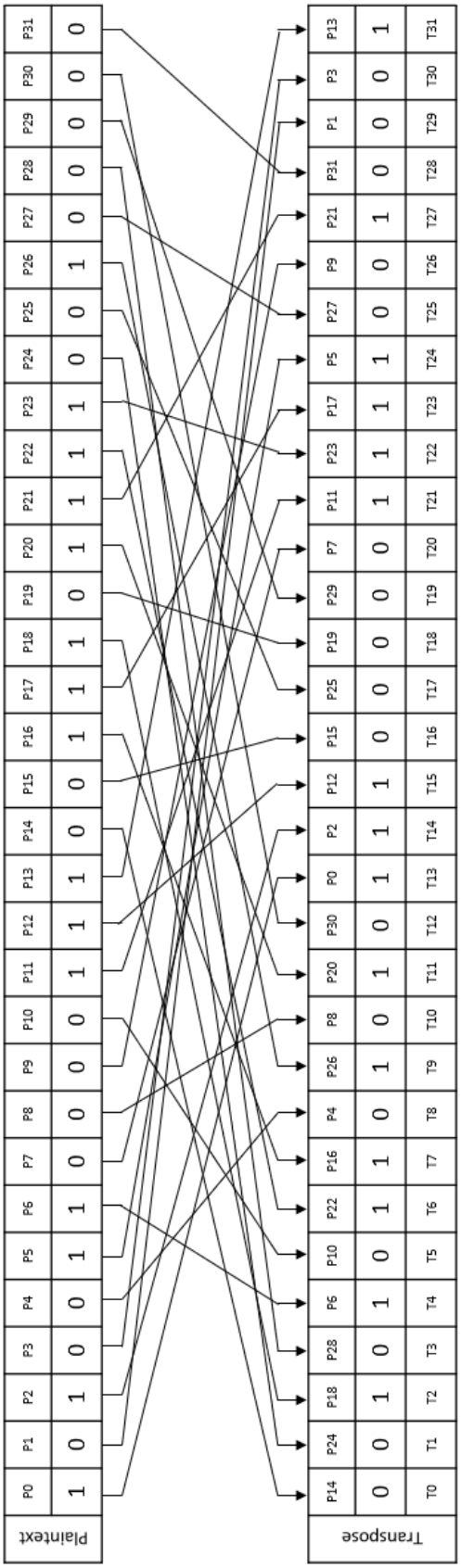


Fig. 3.6 Permutation through Transpose

3.2.2 AddRoundKey

The number of rounds is reduced to 16 to boost the performance along with robust security through 3 levels of random inputs from the user: transpose (permutation), AddRoundKey and finally, 5-bit S-box. Here, at AddRoundKey, the input from transpose is altered by performing an Exclusive-OR with a subkey, uniquely generated for that round.

For any given round, the subkey, $sk_j = \{sk_{31}^j, sk_{30}^j, ...sk_0^j\}$ for $0 \leq j \leq 15$ and the transpose value, $T_i = \{t_{31}, t_{30}, ...t_0\}$, addRoundKey consists of the operation for $0 \leq i \leq 31$ as follows:

$$adRK_i \Rightarrow t_i \oplus sk_i^j \tag{3.2}$$

The output of the addRoundKey function is served as an input to the next level for substitution using the 5-bit S-box.

3.2.3 Substitution using 5-bit S-box

Recently, many researchers have proposed various S-boxes [200–203] based on some chaotic theory that shows good resistance against cryptanalysis. However, most of them are 8-bit in size. The comparison of cryptanalysis for these 8-bit S-boxes is showcased, but the performance and cost are not compared with other bit-size S-boxes. Due to their large size (8x8 bit), these S-boxes are not suitable for resource-constrained IoT devices or, in other words, the design is not ideal for lightweight cryptography. Also, very few algorithm designs suit the short messages due to these reasons. Since S-box is a fundamental and the only component that offers confusion property and further ensures nonlinear in any SPN-based cryptography algorithm, it encourages redesigning the S-box to address the above-discussed challenges while developing a new cryptography algorithms.

This subsection takes an inclusive view of the design criteria of S-box in lightweight cryptography algorithms by considering the significance of the substitution technique to trade-off among performance, cost and security. Further, it discusses existing S-boxes along with their advantages and limitations. Also, the subsection describes the detailed structure of the proposed 5-bit S-box by elaborating its design criteria and various schemes to derive over different block sizes.

Popular S-boxes

Many researchers and scientists proposed a variety of S-box concepts in the past. Some show high resistance against various attacks and high resource demand, whereas some demonstrate better performance but a weak stand against the security attacks. Most of these S-boxes take

3-bit, 4-bit, 5-bit, 6-bit or 8-bit input and produce either the same or compressed bit output [204]. Among these, 4-bit S-boxes are popular among lightweight cryptography algorithms due to their compact [205][206] but simple implementation [207]. This section presents an overview of S-boxes used by popular lightweight cryptography algorithms such as PRINT, PRESENT, RECTANGLE, EPCBC, TWINE, LED (Light Encryption Device), SKINNY, Piccolo, KLEIN, Puffin, LBlock, SPONGENT, DESL/DESXL, ASCON, PRIMATE, ICE-POLE, and SHAMASH.

3-bit S-box: PRINT [91], dedicated designed for integrated circuit (IC) printing, offers the smallest 3x3-bit S-box, $S : \{0,1\}^3 \rightarrow \{0,1\}^3$. A single S-box (Table 3.1) in an octal numeral system is used parallelly for $\frac{b}{3}$ times, where $b \in \{48, 96\}$, the size of input block. It is both hardware and software efficient due to its cost-effective implementation on extremely low-cost RFID tags. At the same time, it is vulnerable to attackers due to its small number of possibilities to create different S-boxes.

Table 3.1 3-bit S-box design

x	0	1	2	3	4	5	6	7
$S(x)$	0	1	3	6	7	4	5	2

4-bit S-box: PRESENT [64] uses 4-bit S-box, $S : \{0,1\}^4 \rightarrow \{0,1\}^4$, (Table 3.2). It relies on of Hexadecimal system and forms a state as sixteen 4-bit words in each sBoxLayer. The S-box design criteria allows 8064 possible S-boxes schemes (maximum) [205]. It is victime of differential cryptanalysis [163][44].

Table 3.2 4-bit S-Box Design

x	0	1	2	3	.	.	.	.	.	.	D	E	F
$S(x)$	C	5	6	A	.	.	.	.	.	.	7	2	9

RECTANGLE [71] adopts a 4x4 S-box from PRESENT with a reduced number of rounds (25 compared to 31) to offer software efficiency. It has AES like structure with the removal of a few functionalities (a slight change in SP Network) and the introduction of the bit-slice

technique to improve performance and cost. Unfortunately, like PRESENT, it also suffers from various cyber-attacks [71].

EPCBC (Electronic Product Code Block Cipher) [98] uses the same 4x4 S-box as used in PRESENT. It just varies in key scheduling from PRESENT.

Like the above algorithms, TWINE [78] also use the ready-made 4x4 S-box from PRESENT. With other structural changes in the algorithm, it gives faster performance than PRESENT [53].

The trend of using 4x4 S-box from PRESENT continues with LED [93], SKINNY [77], Piccolo (four bijective S-boxes) [133], KLEIN [92], Puffin [87], LBlock (such 8 different 4x4 bit S-boxes) [126] and SPONGENT (uses it for $\frac{b}{4}$ times parallelly, where b is the fixed number of bits of a state) [208].

5-bit S-box: ASCON [100][101] uses 5-bit S-box, $S : \{0,1\}^5 \rightarrow \{0,1\}^5$, in parallel over 320 bits in bit-slice manner (Table 3.3). SHAMASH [102], similar to ASCON, uses 5-bit S-box with minor linearity and bit distribution difference compared to ASCON's S-box. PRIMATE [103] works on 5x8 and 7x8 state of 5-bit elements for multiple times on different variance. ICEPOLE [104][105] operates 5-bit S-box on 256 rows of 1280 state of the plaintext. The structure of all of these S-boxes is remarkably similar. Due to their odd size (not the multiple of two, i.e., $size \neq 2^n$), they are not as popular as 4-bit S-boxes and have limited history.

Table 3.3 5-bit S-Box Design

x	0	1	2	3	.	.	.	.	.	.	29	30	31
$S(x)$	4	11	31	20	.	.	.	.	.	.	10	15	23

6-bit S-box: DESL is the lightweight version of DES (Data Encryption Standard), where it is further updated as DESXL with a key whitening feature to improve the security [106]. DESL/DESXL, uses 6-bit S-box that takes 6-bit input and produces compressed 4-bit output [204][106]. Both replaces 8 different 6x4 bits S-box of DES with a single 6x4 bits S-box, $S : \{0,1\}^6 \rightarrow \{0,1\}^4$. The first and last bits of the input form a 2-bit binary to select one of four rows, and the middle 4-bit selects one of the sixteen columns (Table 3.4). For instance, 6-bit input 011001, the row is 01 (row 1), and the column is 1100 (column 12) will be selected to produce the output 13 (1101). The possible number of different S-boxes with this design criteria is 2^{56} [106].

Table 3.4 6-bit S-Box Design

x		0	1	2	3	.	.	11	12	13	14	15
$S(x)$	(00)	14	5	7	.	.	.	.	0	6	13	3
	(01)	5	0	8	.	.	.	.	13	4	1	10
	(10)	4	9	2	.	.	.	.	5	11	3	6
	(11)	9	6	15	.	.	.	.	0	14	10	13

8-bit S-box: ICEBERG [84] uses an 8×8 S-box, $S : \{0,1\}^8 \rightarrow \{0,1\}^8$ (Table 3.5) (inspired from AES [209]), spread over 3 stages (S_0, S_1, S_0) in the form of 4x4 S-boxes (Table 3.6, 3.7) in parallel to achieve the substitution. It is quite expensive in terms of cost and performance while implementing on resource-constrained IoT devices.

Table 3.5 8-bit S-box Design

	00	01	02	03	04	05	.	.	0C	0D	0E	0F
00	24	c1	38	e7	.	.	.	.	.	d6	52	fd
10	40	6c	d3	3d	.	.	.	.	.	fb	fc	f1
20	07	f5	93	cd	.	.	.	.	.	5f	92	6b
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
d0	10	49	25	fa	.	.	.	.	.	b9	c4	12
e0	a0	95	65	bf	.	.	.	.	.	9b	a4	d1
f0	cb	1f	8f	8e	.	.	.	.	.	1e	0f	79

Table 3.6 S_0: 4x4 S-box

0	1	2	3	4	.	.	.	A	B	C	D	E	F
d	7	3	9	.	.	.	.	.	5	e	6	0	8

Table 3.7 S_1: 4x4 S-box

0	1	2	3	4	.	.	.	A	B	C	D	E	F
4	a	f	c	.	.	.	.	.	6	1	7	3	2

S-box Designs and Facts:

The facts about S-box observed from the study are as follows:

- 3-bit S-box is the cheapest in terms of memory, energy and computing power along with high performance but can be easily victimised of an attack due to only 2^3 different S-box possibilities.
- 4-bit S-box is more efficient than 8-bit S-box in terms of energy consumption but provides low security (this could be resolved by increasing the number of rounds).
- 5-bit S-box is not widely used due to its odd nature but could be an alternative to 4-bit S-box in terms of improved security.
- 6-bit and 8-bit S-boxes are comparatively more secure than 4-bit S-box but expensive in terms of resources.

Table 3.8 exhibits the existing n-bit S-boxes and their related concerns.

Table 3.8 Existing S-boxes and related concerns

S-Box Type	Concerns
3-bit [91]	• Implementation cost is very low but could be easily breakable (only 8 possible values) • Even the increase in the no. of rounds could not help to bring an adequate security level
4-bit [44, 64, 71, 163]	• Low resource requirements, Low security (only 16 possible values) • An increase in the no. of rounds could resolve this issue but affects adversely on execution time
5-bit [100–105]	• A very minor increase in resource requirements compared to 4-bit S-box • Moderate/adequate security level (32 possible values)
6-bit [106]	• Demands little more memory (to store 64 possible values) and little high processing power (to derive/process 64 possible values) compare to 4-bit S-box • The above demand leads to high energy consumption compared to 4-bit S-box • The above parameters could increase the demand for the physical area (GE)
8-bit [84, 209]	• Demands huge memory (to store 256 possible values)and very high processing power (to derive/process 64 possible values) • The above demand leads to very high energy consumption compared to 4-bit and 6-bit S-box • The above parameters could dramatically increase the demand for the physical area (GE)

Proposed S-box Design

One of the two key aspects of any cryptography scheme: confusion and diffusion [27], confusion can be procured using substitution to build a complex relationship between the ciphertext and the key. Various academics have proposed a variety of S-boxes to accommodate a variety of cryptography applications. The majority of lightweight cryptography algorithms use a 4-bit or 6-bit S-box [53, 64, 71, 78, 98]. The proposed LWC algorithm, AUM, offers a new 5-bit S-box derived using a dynamic chaotic system called enhanced logistic mapping.

AUM performs the substitution in three steps: 1) Swapping the first and last bit of 32-bit input from addRoundKey, and 2) Divide the remaining 30-bits into six blocks, each block made up of 5-bit and 3) Mapping each 5-bit block into an S-box for its replacement bits (Figure 3.7, Figure 3.8).

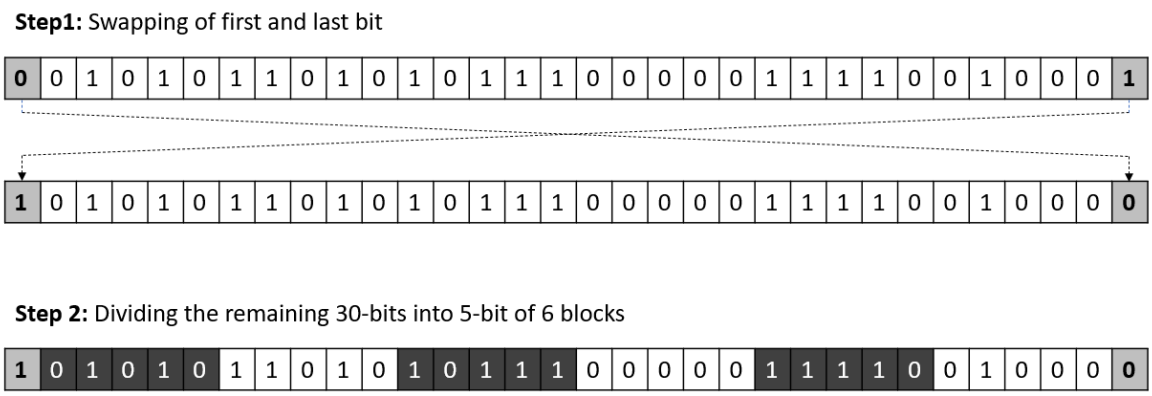
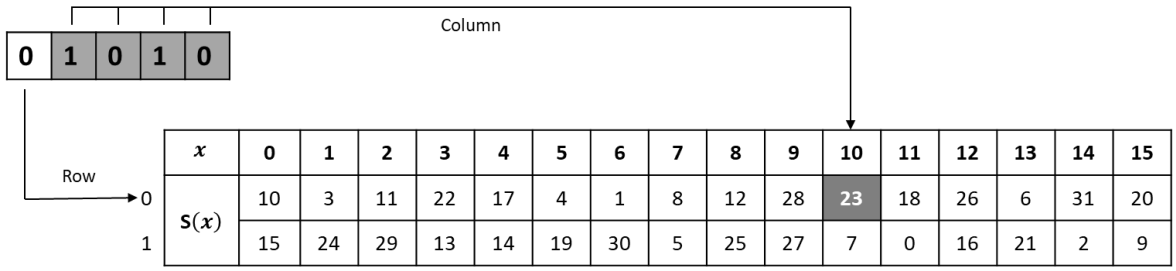


Fig. 3.7 First two steps of substitution process

Now, each 5-bit input block is transformed into a distinctive 5-bit output, $S : \{0,1\}^5 \rightarrow \{0,1\}^5 : x \rightarrow S(x)$. The 5-bit S-box consists of 2 rows and 16 columns. The first bit of 5-bit input (i.e., 0 or 1) selects the row, and the remaining four bits decide the column number. The number of columns is equal to half the number of distinct output values in the S-box (2^m, where $m = 5$ is the number of output bits). The 5-bit input creates 2^5 possible input values, and these 2^5 (i.e., 32) values can be easily accommodated into this S-box table.

Figure 3.8 demonstrates an example of the proposed 5-bit S-box with randomly placed 32 values (using Enhanced Logistic mapping theory) into a 2x16 table. Further, it demonstrates a unique mapping of 5-bit input into an S-box for matching output.

Step 3: Mapping each 5-bit block into an S-box for its replacement bits



x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	10	3	11	22	17	4	1	8	12	28	23	18	26	6	31	20
	15	24	29	13	14	19	30	5	25	27	7	0	16	21	2	9

5-bit Input	Row	Column	Mapping Value	5-bit Output
0 1 0 1 0	(0) 0	(1 0 1 0) 10	23 (1 0 1 1 1)	**1 0 1 1 1**

Fig. 3.8 Mapping each 5-bit block into an S-box for its replacement bits (example)

The pseudocode to generate the dynamic chaotic sequence is as follows:

Step 1. Declare decimal constant p

Step 2. Assign p, where $p \geq 2.0$ (In our case, $p = 4.0$)

Step 3. Declare variable v_i

Step 4. Initialized v_i, where $v_i < 1$ (In our case, $v_i = 0.972$)

Step 5. Calculate v_{i+1} using $\sin(\pi p v_i(1 - v_i))$

Step 6. Repeat step 5 for n-times to generate dynamic chaotic sequence (In our case, n = 32 times)

Step 7. Finally, arrange the elements $(1, 2, \ldots n)$ in ascending order of the sequence generated

Design Criteria of 5-bit S-box

To build a simple but robust 5-bit S-box, $S : F_2^5 \rightarrow F_2^5$, that could be easily implemented on resource-constrained IoT devices, the following simple but security efficient rules need to apply:

1. S-box, S, must have distinct 32 elements (0 - 31) spread over 16 columns and 2 rows that satisfies bijective property (section 4.3.1).

2. Generate the complex chaotic sequence of the elements in 5-bit S-box using enhanced logistic map equation [210] as defined follows:

$$v_{i+1} = E(L(v_i)) = \sin(\pi p v_i(1 - v_i)) \tag{3.3}$$

where p is a control parameter, and $p \in (2, +\infty)$. Even though p could have initiated with 0, the suggested initial value of p is 2 for a complex dynamic chaotic behavior reasons. This is because all the fixed points of the enhanced logistic map are unstable when $p \geq 2$ [200].

3. Any value, V_i, in S, must be different from its column index, C_i, to avoid a fixed point, i.e.,

$$V_i \neq \begin{cases} C_i & \text{if } V_i \in R_0 \\ C_{(15+i)} & \text{if } V_i \in R_1 \end{cases}$$

where, R_0 is the 0th row and R_1 is the 1st row, $R_0 \subset S$ and $R_1 \subset S$

4. An input value, In_i, and its corresponding value, V_i, in S must have bit variation of n bit(s), $0 < n \leq 5$, to meet overall Strict Avalanche Criteria (SAC), i.e.,

$$Bit_{var}(V_i, In_i) \geq n, \ \forall V_i \in S \tag{3.4}$$

where In_i: $f(R_i, C_i)$, $R_i \rightarrow \{0,1\}$, $C_i \rightarrow \{0,1\}^4$ and $V_i \rightarrow \{0,1\}^5$

Here, in design criteria 2), the sequence of elements in the 5-bit S-box could be generated using any chaotic mapping methods such as logistic map, sine map, tent map and quadratic map to improve its dynamic behaviour. But, we adopt an enhanced logistic map technique for our 5-bit S-box as it eliminates fixed point [200] weakness of an S-box design.

By implementing the above-defined set of rules, the total number of possible random 5-bit S-box could be $31! \approx 8.22 * 10^{33}$ which is enormous compared to a number of 4-bit S-box that is $15! \approx 1.3 * 10^{12}$. Thus, the proposed technic takes care of all security aspects (see section 4.3 for Security Analysis) with high performance and low cost (see section 4.2 for Performance and Cost Analysis) with the flexibility to go with various block sizes (following section 3.2.3).

Implementation Flexibility of 5-bit S-box with different block size

Usually, the input block size are even and of 2^n ($n = 5, 6, 7, ...$) [211][212][205]. Also, they are multiple of either 4 or 6, in general. Odd size S-boxes are avoided with the flexibility to split the input block over the S-box size, and always 4, 6 or 8 bits are considered.

Although the proposed S-box is 5-bit, an odd size S-box, it easily fits over the various input block sizes such as 32, 48, 64, 128 and 256.

Let's consider a 32-bit input block where the middle 30 bits (out of 32) can be split into six 5-bit inputs (to the 5-bit S-box). Then remaining first and last bits can be swapped. Similarly, a n-bit input block can be divided into m 5-bit input (to the 5-bit S-box). Then remaining $x\,(i.e.,\ n-5m)$ bits, $x \in \{1, 2, 3, 4\}$, can be interchanged. Table 3.9 gives the brief of how 5-bit S-box can be implemented with popular input block sizes.

Table 3.9 Implementation Flexibility of 5-bit S-box with different block size

Block Size	Implementation on 5-bit S-box	Remaining bits
32-bit	The middle 30 bits (out of 32) can be split into six 5-bit inputs to the 5-bit S-box	The remaining first and last bits can be swapped.
48-bit	The middle 45 bits (out of 48) can be split into nine 5-bit inputs to the 5-bit S-box	The remaining three bits, either 'first and last two' or 'first two and last' bits, can be interchanged.
64-bit	The middle 60 bits (out of 64) can be split into twelve 5-bit inputs to the 5-bit S-box	The remaining first two and last two bits can be interchanged.
128-bit	The middle 125 bits (out of 128) can be split into twenty-five 5-bit inputs to the 5-bit S-box	The remaining three bits can be interchanged, either 'first and last two' or 'first two and last' bits.
256-bit	The middle 255 bits (out of 256) can be split into fifty-one 5-bit inputs to the 5-bit S-box	The remaining one bit (either first or last) can be inverted (Ones' complement).

3.2.4 Subkey Generation

One of the critical challenges while designing an LWC algorithm for resource-constrained IoT devices is the key generation technic. It increases the load on the device by demanding more resources. Due to this reason, some researchers have proposed algorithms which derive the keys on the fly to reduce the memory requirements [120][84], or there is a complete absence of key scheduling to minimise processing costs [93]. It helps reduce the cost and improve performance but adversely affects security features. This challenge encourages designers to reconsider and modify key generation techniques.

This proposal generates sixteen distinctive 32-bit subkeys for each round from a 32-bit original key. These subkeys are similar to the input block size and are produced by rotating right/left (round rotation) randomly. For example, they rotate right/left by transpose key values, tk, or by s-box values (by first 16 values or by last 16 values) or by any random numbers for 16 times. This technique increases the security level while reducing the key generation load of the algorithm at the same time.

For example, each subkey, sk_i, to go with the ith round, a round rotation (right) by transpose key value, tk_i, could be performed on the current state of the subkey (on the original key in case of the first round), Figure 3.9. The number of rotations for every subkey is quite random depending on the transpose key value as follows:

$$sk_i \rightarrow sk_{i-1} \gg tk_i \quad (round\ rotation) \tag{3.5}$$

Let's take an example to understand the subkey generation process,

$$Transpose\ Key: 7\ 12\ 9\ 14\ 3\ 5\ 11\ 8\ 2\ 13\ 4\ 10\ 15\ 0\ 1\ 6$$

Now rotate the original key right seven times to produce the first subkey (for round 1), rotate the first subkey right for 12 times to produce the second subkey, rotate the second subkey right for nine times to produce the third subkey and so on till the last subkey for the round 16. If the transpose key value, tk_i, is 0 (in 14th key in this example), rotate it for 16 times.

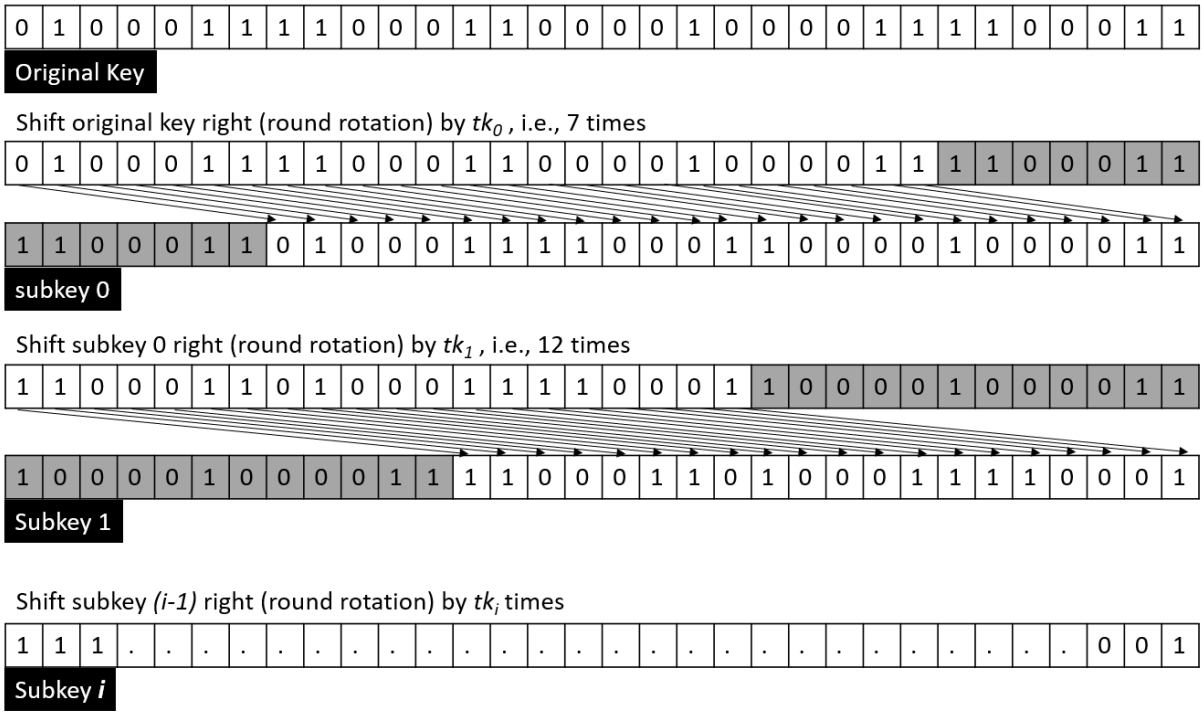


Fig. 3.9 Subkey generation process through an example

3.3 Inspiration for naming the proposed LWC algorithm

The number of IoT devices used by various applications is increasing numerously; or in other words, many IoT devices are present around us in the form of various IoT applications such as smart homes, smart offices, industries, public areas, etc. Also, the cyber risk increases in parallel with the number of these devices. This need to be addressed this issue, mainly when devices are small, by a cryptography technic that could easily fit most of these omnipresent devices. The proposed algorithm is named **"AUM"**, which represents three aspects of the Almighty (A-U-M), creation, manifestation/preservation, and destruction. AUM is omnipresent and an ultimate power that controls the whole universe. Inspired by this holy power, the proposed algorithm is named “AUM” to create a shield and protect every IoT communication happening around the world from destruction. Also, it is designed to fit into every IoT device ranging from small to large size.

3.4 Proposed LWC algorithm (AUM): Pseudo Code

Global Variable Declaration

Step 1: Declare global variables

```
unsigned integer array p[32];
unsigned integer array tk[16];
unsigned integer array Org_key[8];
unsigned integer array Org_key_bits[32];
unsigned integer array in[32];
unsigned integer array trans[32];
unsigned integer array addKey_out[32];
unsigned integer array s_out[32];
unsigned integer bits_size = 1 and size = 0;
Two dimensional unsigned integer array sBox[2][16];
Two dimensional unsigned integer array sub_key[16][32];
```

Subkey generation() from the original key

Step 1: Rotate the original key (binary form) by transpose key (tk[0])
Step 2: Repeat the process for another 15 times using the distinct transpose key tk[i], i.e., (tk[1]...tk[15])

main() - with subkey generation()

Step 1: Declare unsigned integer variables i and round.
Step 2: Assign 0/1 to round variable.
Step 3: Call subkey_Gen() to generate 16 subkeys from the original key
Step 4: Copy each bit of plaintext (32-bits) into an array in[]
Step 5: Call transpose(), addRoundKey(round-1), s_Box() and feed_input() functions
Step 6: Increment the round variable by 1
Step 7: Repeat step 5 and 6 for 16 times

[OR] main() - without subkey generation()

Step 1: Declare unsigned integer variables i and round.
Step 2: Assign 0/1 to round variable.
Step 3: Copy each bit of plaintext (32-bits) into an array in[]
Step 4: Call transpose(), addRoundKey(round-1), s_Box() and feed_input() functions
Step 5: Increment the round variable by 1
Step 6: Repeat step 4 and 5 for 16 times

Converting Key into binary from decimal

Step 1: Declare unsigned integer variables i, k, m, binaryNum[4], four_bit[4]
Step 2: Assign i to 0
Step 3: Declare integer variables j, q
Step 4: Perform n modulo 2 and store the remainder into binaryNum[i]
Step 5: Divide n by 2 and store the result again into n again
Step 6: Increment variable i by 1
Step 7: Repeat step 4, 5 and 6 until n > 0
Step 8: Perform left shift by q on 0 and store the result into m, (m = 0 « q;)
Step 9: Perform OR operation between binaryNum[q] and m and store the result into some variable k
Step 10: Assign four_bit[q] with 1 if k = 1 and with 0 otherwise
Step 11: Repeat step 8, 9 and 10 for q = 3 until q >= 0 by decrementing q by 1
Step 12: Storing each bits of four_bit into Org_key_bits[] by repeating the step, starting with j = 3 until j >= 0, i.e., (Org_key_bits[size++] = four_bit[j];)

Permutation through Transpose function

Step 1: Declare unsigned integer variables i, j
Step 2: Check transpose key, tk[i] for all 16 values, and if the tk[i] = 16, change it to 0
Step 3: Assign j to 0
Step 4: For first 16 input values, derive the transpose, trans[i] using the formula "in[tk[j] * 2 + 0]" and for rest 16 values, use the formula in[tk[j] * 2 + 1]
Step 5: Increment j by 1
Step 6: If j reaches to 16, reset it 0 again

Step 7: Repeat the step 4, 5 and 6 for 32 times to produce 32 bit trans[] bits

Exclusive OR (XOR) between the output from transpose and the subkey (addRoundKey)

Step 1: Declare unsigned integer variables i
Step 2: Perform XOR between each bit of trans[i] and sub_key[r][i] and store the result into some variable addKey_out[i]
Step 3: Repeat the step 2 for 32 times

Substitution using 5-bit S-box

Step 1: Declare unsigned integer variables i, j, k, n, r, c[4], dec, s_val
Step 2: Declare integer variables l
Step 3: Assign first bit of s_out (i.e., s_out[0]) with the last bit of addKey_out (i.e., addKey_out[31]) and the last bit of s_out (i.e., s_out[31]) with the first bit of addKey_out (i.e., addKey_out[0])
Step 4: Assign n by 1
Step 5: Store each bit of addKey_out to some array, c[n]
Step 6: Increment n by 1
Step 7: Repeat the step 5 and 6 for 4 times starting with i+1 and incrementing by 1 until <= i+4 (i.e., for(j = i+1; j <= i+4; j++))
Step 8: Store i^{th} bit of addKey_out (i.e., addKey_out[i]) to some variable r as row number
Step 9: Assign dec with 0
Step 10: Compute/Assign dec = dec + (int)pow(2, l) * c[3 - l + 1]; i.e., dec = dec + 2l * c[3 - l + 1]
Step 11: Repeat step 10 for 4 times; starting from 3 and decrementing by 1 until it reaches to 0 (0 included)
Step 12: Assign sBox[r][dec] to some variable, s_val
Step 13: Call Dec_to_Binary(s_val)
Step 14: Repeat the steps 4 to 13 for 6 times (for(i=1; i<31; i=i+5))
Step 15: Reset/Assign the bits_size to 1 again for the next round

Converting Decimal to Binary

Step 1: Declare unsigned integer variables i, k, m, binaryNum[5], five_bit[5]
Step 2: Assign i to 0
Step 3: Declare integer variables j, q
Step 4: Perform n modulo 2 and store the remainder into binaryNum[i]
Step 5: Divide n by 2 and store the result into n again
Step 6: Increment variable i by 1
Step 7: Repeat step 4, 5 and 6 until n > 0
Step 8: Perform left shift by q on 0 and store the result into m, (m = 0 « q;)
Step 9: Perform OR operation between binaryNum[q] and m, store the result into some variable, k
Step 10: Assign five_bit[q] with 1 if k = 1 and with 0 otherwise
Step 11: Repeat step 8, 9 and 10 for q=4 until q>=0 by decrementing q by 1
Step 12: Storing each bits of five_bit into Org_key_bits[] by repeating the step, starting with j=4 until j>=0 (s_out[bits_size++] = five_bit[j];)

Feeding the 32-bit output of a round to the next round

Step 1: Declare unsigned integer variables i
Step 2: Copy each bit of s-box, s_out (32-bits) into an array in[]
Step 3: Repeat step 2 for 32 times

Chapter 4

Result Analysis

This chapter discusses three criteria defined by NIST to evaluate any lightweight cryptography algorithm. It talks about the cost and performance of the proposed model and then gives a complete view of security aspects in the second part. First, it reveals the performance and cost efficiency of the proposed algorithm by implementing it on different ASIC platforms and comparing it with its competitor LWC algorithms. It also discusses the resource requirements by their individual elements. Then, a broad view of its core element, the S-box, is given by comparing it with other n-bit size S-boxes. Further, the section gives a clear picture of the security strength of the proposed algorithm by comparing it with the same bit-size S-boxes. Finally, the security measure is carried out by considering all essential and popular security metrics such as bijective property, nonlinearity, linearity (LP), differential probability (cryptanalysis), degree of avalanche effect (SAC), bit Independence criteria (BIC) and algebraic attacks.

4.1 Experiment Setup

To evaluate the performance and cost of the proposed algorithm, AUM, first, the code is generated in *C language* [Appendix A] from the pseudo code to define its structure and flow and to check the correct functioning of the algorithm. After satisfactory results from *C* executions of the algorithm modules, the code is developed in *Verilog* [Appendix B] which is a hardware description language (HDL) [213]. The performance and cost of the proposed algorithm, AUM, is evaluated on two ASIC platforms: *Cadence (Genus) RTL synthesis tool* (RTL compiler) using *180nm* SCL180 library and *Synopsys Design Compiler* tool (version *K2015.06*) using *180nm* Faraday cell library *fsa0m_a_generic_core_bc* to achieve and to compare the results of gate level synthesis (Table 4.1). Figure 4.1 gives an overview of the experiment setup.

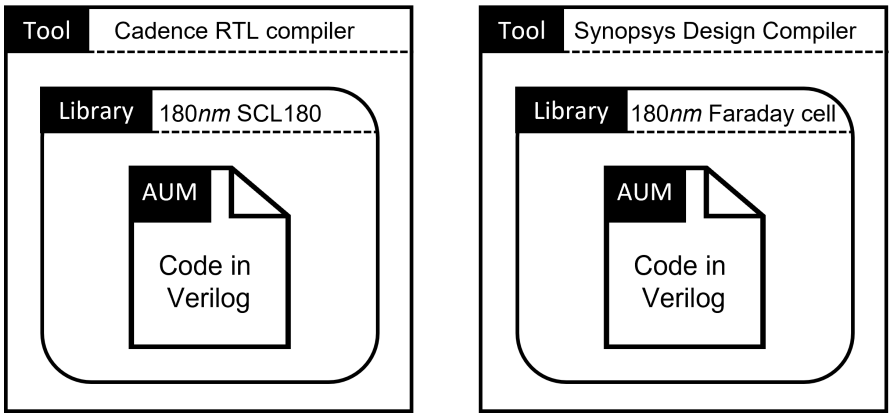


Fig. 4.1 Experiment Setup

The experiment is conducted at various frequencies such as 10KHz, 100KHz, 10MHz, and 100MHz, but the results obtained at 100KHz frequency are discussed in this research due to its high popularity and wide use by the researchers [31]. Also, the performances of 4-bit, 5-bit and 6-bit S-boxes are tested and compared separately along with the implementation of the whole algorithm.

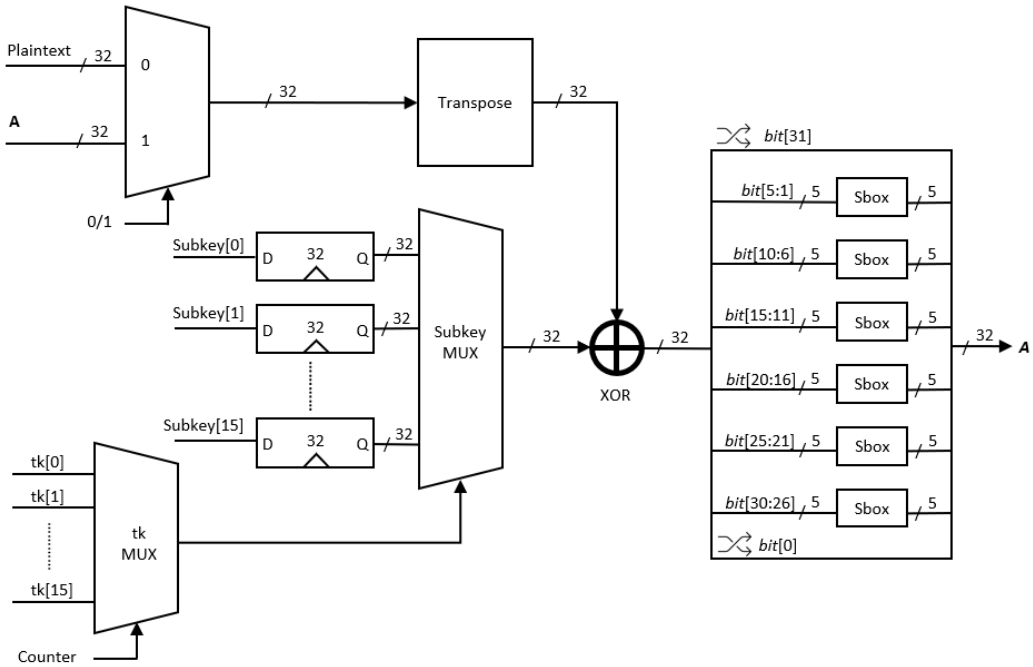


Fig. 4.2 The datapath of an area-optimized version of AUM

Figure 4.2 shows the datapath of an area-optimized version of the proposed algorithm (AUM), which performs one round in 32 clock cycle to encrypt 32-bit plaintext using a 32-bit key. Also, Figure 4.3 shows the behavioural simulation of the AUM implementation.

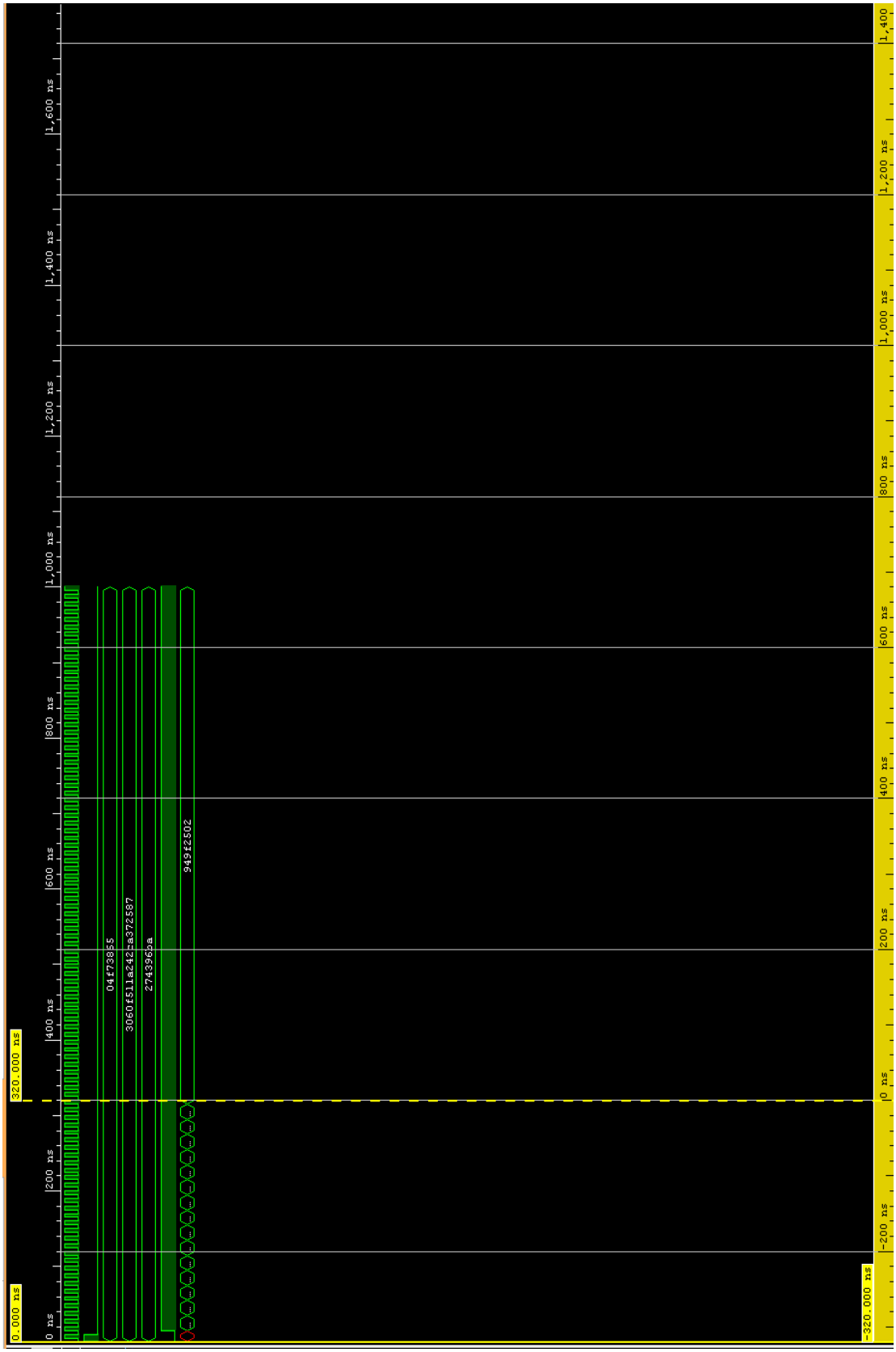


Fig. 4.3 Behavioural simulation of AUM (32 clock cycle)

4.2 Performance and Cost Analysis

The resource requirements of the proposed LWC algorithm, AUM, including and excluding the key generation module, on different compilers are recorded and documented in Table 4.1. However, the experiment results on both the platforms (compilers) are almost the same for physical implementation (GE) and throughput (Kbps); the best results for power and energy efficiency are observed on *Cadence* Compiler. In addition, a detailed comparison of the cost and performance of the proposed algorithm with its 4-bit and 5-bit competitors (evaluated on one of these compilers) is listed in the table and further portrayed in Figure 4.4. The primary criteria for selecting the algorithm in Table 4.1 is the similar size of the S-box (5-bit). Also, one of the milestone LWC algorithms, PRESENT, is included in this comparison as a benchmark to prove the proposed algorithm's efficiency. The compared parameters of LWC algorithms in this competition are the standard features referred to by most researchers and also suggested by NIST. Moreover, the parameter values for the referred algorithms are taken from the respective original author's articles.

Table 4.1 AUM implementation @100KHz frequencies

Algorithm	**Simulator**	**Frequency**	**Tech** (μm)	**Area** (GE)	**Power** (μW)	**Energy** (μJ/bit)	**Throughput** ($Kbps$)
AUM (excluding KG)	Cadence RTL Compiler	$100KHz$	0.18	1056	0.13	0.13	100
AUM (including KG)	Cadence RTL Compiler	$100KHz$	0.18	1289	0.16	0.16	100
AUM#	Synopsys Design Compiler (v-K2015.06)	$100KHz$	0.18	1319	0.95	0.92	103
PRESENT [64]	Synopsys Design Compiler (v-Y2006.06)	$100KHz$	0.18	1570	5	2.5	200
ASCON [214, 215]	Cadence RTL Compiler (v08.10-s28_1)	$1MHz$	0.09	2570	15	713.25	14000
ASCON-round based [214, 215]	Cadence RTL Compiler (v08.10-s28_1)	$1MHz$	0.09	7970	45	2154.25	15000
PRIMATE-80A [216]	Synopsys Design Compiler (v2015.06)	$100KHz$	0.09	3680	2.32	0.12	20000
PRIMATE-80B [216]	Synopsys Design Compiler (v2015.06)	$100KHz$	0.09	1200	0.68	0.57	1250

where, *Energy [μJ] = (Latency [cycles/block] * Power [μW])/block size [bits]*

with Key Generation

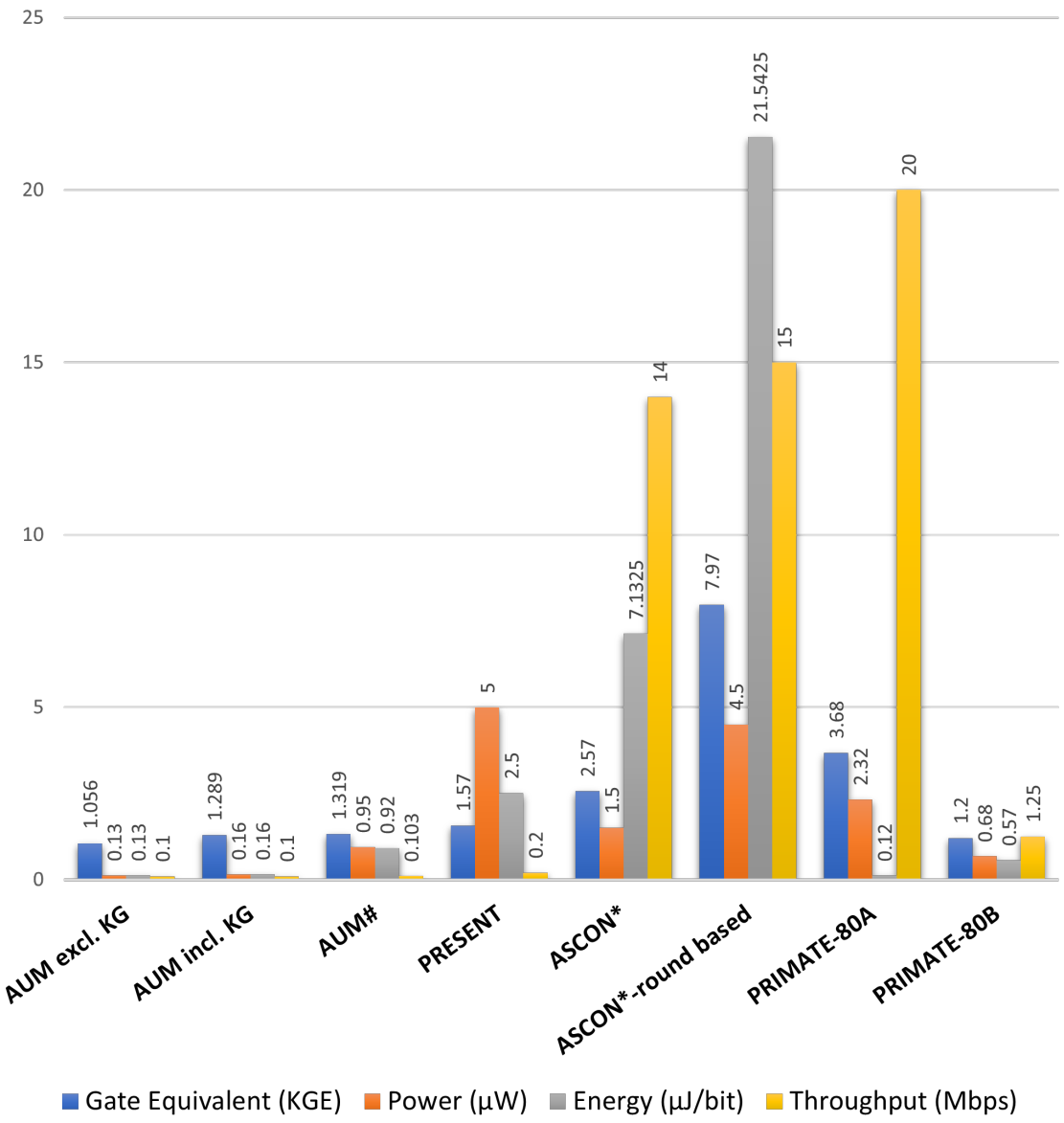


Fig. 4.4 Performance and cost comparison of AUM with other competitors

The proposed LWC algorithm, AUM, takes very little power, 0.13 μW and 0.16 μW, on *Cadence RTL Compiler* for without subkey generation and with subkey generation modules, respectively. On the other hand, it takes 0.95 μW of power on *Synopsys Design Compiler* for the full implementation of the AUM. One of the competitors (4-bit S-box) and milestone of the LWC algorithms, PRESENT, demands 5 μW of power which is very high compared to the proposed one. Even the power demands by both of the version of ASCON is drastic (up to 45 μW) at 1 MHz frequency on *Cadence* platform. Two variants of PRIMATE, PRIMATE-80A

and PRIMATE-80B, demand 2.32 μW and 0.68 μW of power, respectively. The second variant gives close competition to the proposed algorithm for this ASIC implementation.

Also, the proposed algorithm, AUM, without subkey generation and with subkey generation module requires only 1056 GE, 1289 GE (on *Cadence*) and 1319 GE (on *Synopsys*) of area to execute on the listed 0.18 μm technology platforms. All the competitors, including PRESENT, demand a higher implementation area on either platform except PRIMATE-80B. PRIMATE-80A shows high throughput (20000 $Kbps$) but requires almost 3-time more GE than AUM$^{\#}$. While PRIMATE-80B is a strong contender in performance and cost, our proposed algorithm, AUM, exhibits better immunity against various cryptanalysis (Table 4.10) with a minor affordable resource increase. Thus, selecting the AUM over PRIMATE-80B is preferable for safety reasons. All ASCON and PRIMATE versions are evaluated on 0.09 μm technology. The lighter version of ASCON requires almost double physical area (GE) to implement, while round base ASCON demands 6-times more GE than the key generation version of AUM. Thus, AUM beats all of these competitors and proves the best choice.

An implementation that takes up to 3000 logic gates is described as a lightweight implementation, an implementation that takes around 2000 logic gates is called a low-cost, and a further compact implementation with about 1000 logic gates is known as an ultra-lightweight implementation [31]. Our proposed algorithm requires only 1056 GE (without subkey generation) and 1289 GE (with subkey generation), and thus it earns the ultra-lightweight title. Figure 4.4 outlines performance and cost comparison between AUM and its' competitors.

The proposed algorithm targets RFID tags, smart cards and sensors with small messages, usually, message size $< 2Kb$. However, AUM achieves a throughput of 100 $Kbps$, and other competitors show higher values; it doesn't really affect real-world processing while message size is small. Also, many real-time applications, such as RFID tags, smart cards, etc., use the same subkeys (already created) for every transaction. In addition, the key remains the same for its lifetime (mostly); there is no need for subkey generation again. So, it is possible to execute the subkey generation module separately, and the algorithm excluding the subkey generation module could be embedded on the device. These subkeys could be updated periodically or on-demand separately.

The individual GE requirements by each component of the proposed algorithm, AUM, is exhibited in Table 4.2.

Table 4.2 GE requirements by individual modules in AUM

Module Name	Cadence RTL Compiler				Synopsys Design Compiler	
	Without subkey Gen		With subkey Gen		Full Algorithm	
	GE requirements	Percentage (%)	GE requirements	Percentage (%)	GE requirements	Percentage (%)
Top module	194	18.37 %	194	15.05 %	-	-
Control Unit	54	5.11 %	55	4.27 %	118.71	9%
Key Generation	-	-	581	45.07 %	276.33	20.95%
Transpose	50	4.74 %	50	3.88 %	225.55	17.1%
Key Select	349	33.05 %	-	-	-	-
AddRoundKey	63	5.96 %	63	4.89 %	184	13.95%
sBox	346	32.77 %	346	26.84 %	514.41	39%
Total	**1056**	**100%**	**1289**	**100%**	**1319**	**100%**

As S-box is the key element of this research work, the following sections discusses more on 5-bit S-box by comparing it with the 4-bit and 6-bit S-boxes. Figure 4.5 shows the datapath of an area-optimized by 5-bit S-box, which performs one round in one clock cycle i.e. a 32-bit width datapath.

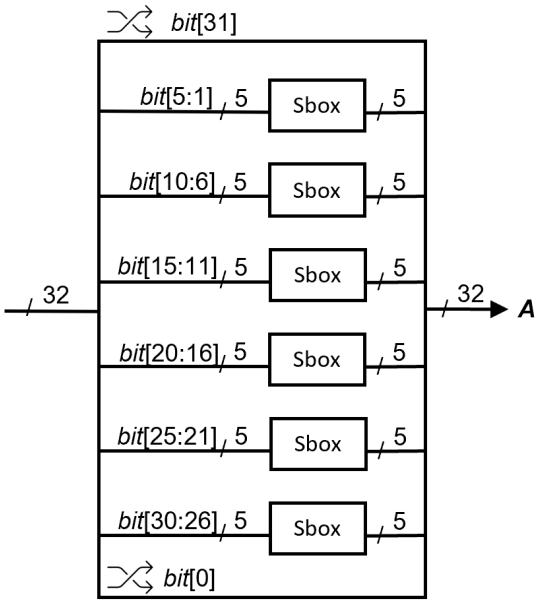


Fig. 4.5 Datapath of 5-bit S-box

Table 4.3 gives the comparison of area (KGE), power (μW) and energy ($\mu J/bit$) requirements by 4-bit, 5-bit (the proposed one) and 6-bit S-boxes while implementing on *Synopsys Design Compiler* (v-K2015.06). It is illustrated in Figure 4.6. From the graph, it can be seen that all three parameters increase more or less as the number of input bit(s) to the S-box increases. Broadly, there is a little rise in power (μW) as well as in energy ($\mu J/bit$) while shifting from 4-bit input to 5-bit input to the S-box. On contrary, there is an observable difference in all three dimensions while adding 1 or 2 bit(s) to the S-box input. Overall, the cost of a 4-bit or 5-bit S-box is almost similar and affordable compare to 6-bit S-box.

Table 4.3 Various S-box implementation @ 100KHz frequencies

S-box Type	**Input Size** (*bit*)	**Tech** (μm)	**Area** (GE)	**Power** (μW)	**Energy** (μJ/bit)
4-bit	32	0.18	355	0.248	0.010
5-bit	32	0.18	515	0.347	0.011
6-bit	32	0.18	803	0.535	0.017

*Energy [µJ] = (Latency [cycles/block] * Power [µW])/block size [bits]*

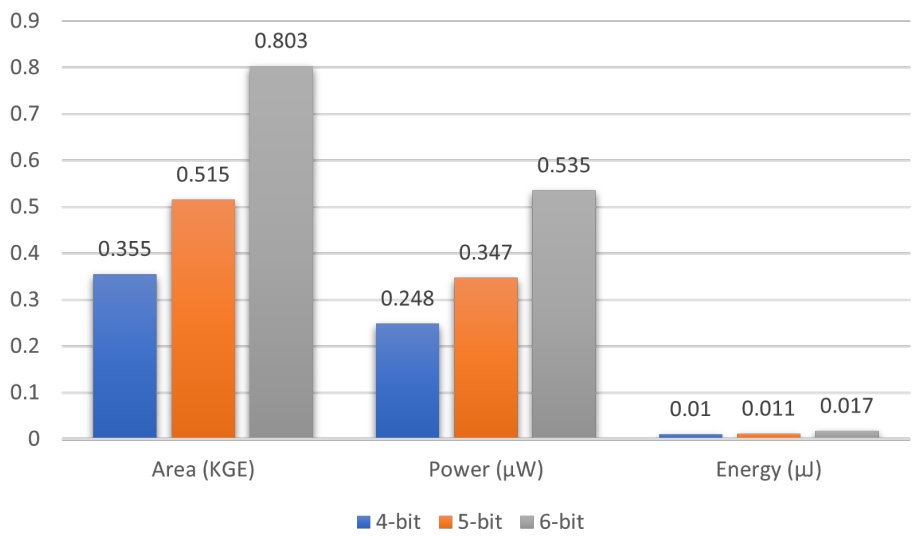


Fig. 4.6 Area, Power & Energy requirement by various S-boxes @ 100KHz frequencies

Further, Table 4.4 compares area requirements (in GE) of the various S-boxes (4-bit/5-bit) used in the popular LWC algorithms. Our proposed 5-bit S-box requires only 12.54 GE on *Cadence* (Genus) RTL synthesis tool (compiler) and beats out 4-bit/5-bit competitors. This can be visualized in Figure 4.7. Our S-box consists of eight NAND gates (seven 2-input NAND gates and one 3-input NAND gate) and seven NOR gates (six 2-input NOR gates and one 3-input NOR gate). Here, one NAND/NOR gate costs 0.78 GE for all 2-input logic gates, whereas a 3-input NAND gate costs 1.37 GE and a 3-input NOR gate costs 0.98 GE. It takes around 0.042 μW of power to implement this logic.

Table 4.4 Various S-box Area Comparison

Algorithm	Ref	S-box bits	Area (GE)
PRESENT	[64]	4-bit	28.03
	[217]	4-bit	22.67
SKINNY	[77]	4-bit	12 - 14.68
LED	[93]	4-bit	22.33
Piccolo	[133]	4-bit	24
	[217]	4-bit	12
PRIMATE	[216]	5-bit	30 - 40
Keccak	[217]	5-bit	17
Proposed	-	**5-bit**	**12.54**

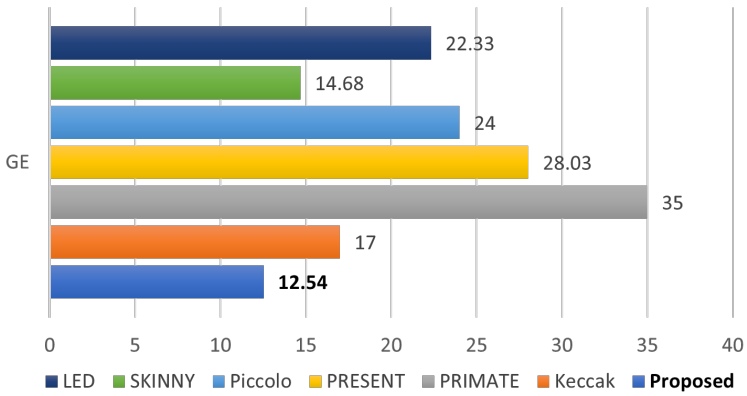


Fig. 4.7 Various S-box Area (GE) Comparison

4.3 Security Analysis

This section demonstrates the security strength of the proposed 5-bit S-box, measured over bijective property, nonlinearity, linearity (LP), differential probability (DAP), differential style boomerang attack (BCT/FBCT), degree of avalanche effect, bit Independence criteria (BIC) and algebraic attacks. It also gives comparison of cryptanalysis of the proposed 5-bit S-box with other existing 5-bit S-boxes from ASCON [100, 101], PRIMATE [103], ICEPOLE [104, 105], and SHAMASH [102]. The superiority of the proposed S-box could be analysed from Table 4.10 and Figure 4.13.

4.3.1 Bijective Property

The bijective property of the proposed n x n S-box (where $n = 5$) is derived using Hamming weight $H_{wt}()$, which is the number of symbols (1) that are different from the zero-symbol (0) of the alphabet $(0,1)$ used in a string [218], i.e., the number of 1's in the string of bits. It is defined as follows:

$$H_{wt}\left(\sum_{i=1}^{n} b_i f_i\right) = 2^{n-1} \tag{4.1}$$

where $b_i \in \{0,1\}$ and $(b_1, b_2, ...b_n) \neq (0,0,...0)$ for each Boolean function, $f_i(1 \leq i \leq n)$. Here, f_i fulfils the bijective property by balancing 0 and 1. Also, the proposed 5-bit S-box has all distinct values from 0 to 31, and thus manifest the bijective property.

4.3.2 Nonlinearity

S-box operations are designed to obtain nonlinearity to the algorithm. It should not be possible to break the algorithm by solving a set of equations using some set of unknown values. Since the proposed S-box selects its elements at random using a dynamic chaotic mapping system (under a defined set of rules), it is almost impossible to derive an equation that solves any correlations between the input value and the corresponding substitution value. Section 3.2.3 details how 5-bit input is arbitrary replaced with another 5-bit output. The nonlinearity can be measured either using Hamming distance or the Walsh matrix. Here, the Walsh matrix is not possible to apply as it works on multiple of two, and our proposed S-box makes use of 5 bits. We measure the nonlinearity using Hamming distance (H_d). It is the distance between any corresponding input-output pairs (x_i, y_i), where $H_d(x_i, y_i) = \#(x_i \neq y_i)$. The minimum value of Hamming distance (H_d) for the proposed 5-bit S-box is 1, whereas maximum is 5 (Table 4.5). The average Hamming distance (H_d) calculated for the proposed

5-bit S-box is 2.625. Figure 4.8 depicts the nonlinearity comparison via Hamming distance (H_d) for the proposed S-box and its 5-bit competitors. The higher the Hamming distance (H_d), the higher the nonlinearity property. Thus our proposed 5-bit S-box satisfies the nonlinearity characteristics.

Table 4.5 Nonlinearity measure through Hamming distance (H_d)

Input	**Output**	**Hamming distance (H_d)**	**Input**	**Output**	**Hamming distance (H_d)**
0 (00000)	10 (01010)	2	16 (10000)	15 (01111)	5
1 (00001)	3 (00011)	1	17 (10001)	24 (11000)	2
2 (00010)	11 (01011)	2	18 (10010)	29 (11101)	4
3 (00011)	22 (10110)	3	19 (10011)	13 (01101)	4
4 (00100)	17 (10001)	3	20 (10100)	14 (01110)	3
5 (00101)	4 (00100)	1	21 (10101)	19 (10011)	2
6 (00110)	1 (00001)	3	22 (10110)	30 (11110)	1
7 (00111)	8 (01000)	4	23 (10111)	5 (00101)	2
8 (01000)	12 (01100)	1	24 (11000)	25 (11001)	1
9 (01001)	28 (11100)	3	25 (11001)	27 (11011)	1
10 (01010)	23 (10111)	4	26 (11010)	7 (00111)	4
11 (01011)	18 (10010)	3	27 (11011)	0 (00000)	4
12 (01100)	26 (11010)	3	28 (11100)	16 (10000)	2
13 (01101)	6 (00110)	3	29 (11101)	21 (10101)	1
14 (01110)	31 (11111)	2	30 (11110)	2 (00010)	3
15 (01111)	20 (10100)	4	31 (11111)	9 (01001)	3

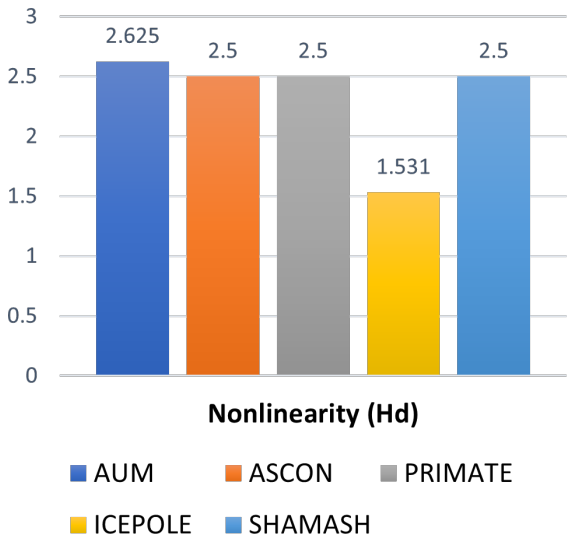


Fig. 4.8 Nonlinearity (Hamming distance (H_d))

4.3.3 Linear Approximation Probability (LP)

Introduced by Matsui's [155], linear approximation probability finds out the maximum value of imbalance in the input-output elements. Let Δx and Δy be the input and output differentials, respectively, and x is a set of all possible inputs with cardinality 2^n. The linear approximation probability for a given S-box is defined as

$$LP = \max_{\Delta x, \Delta y \neq 0} \left| \frac{\#\{x \in X | x.\Delta x = S(x).\Delta y\}}{2^n} - \frac{1}{2} \right| \tag{4.2}$$

The highest matching event of $x.\Delta x = S(x).\Delta y$, for all $x \in X$, found in the proposed 5-bit S-box is eight, and thus maximum linear approximation probability according to the above equation is 0.25. The value of LP close to zero means better security property. Figure 4.9 reveals that the proposed 5-bit S-box has either better or similar linearity property compared to its 5-bit competitors.

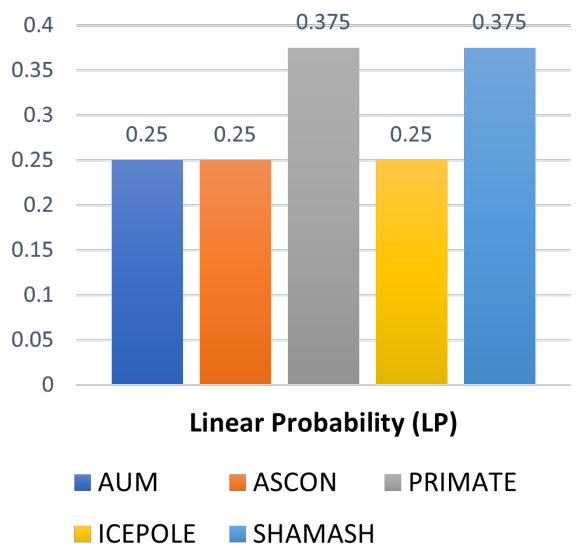


Fig. 4.9 Linear Approximation Probability (LP) comparison

4.3.4 High Resistance to Differential Cryptanalysis

Differential Cryptanalysis [154] is a statistical attack using an S-box's Differential Distribution Table (DDT) characteristic (Table 4.7). It signifies how the output of an S-box varies as the input is changed. There must be undefined changes in output to protect against Differential Cryptanalysis. It is measured as differential approximation probability (DAP), the differential uniformity of the S-box input-output. It is defined as

$$DAP = \max_{\Delta x \neq 0, \Delta y} \left(\frac{\#\{x \in X | S(x) \oplus (S(x \oplus \Delta x) = \Delta y)\}}{2^n} \right) \tag{4.3}$$

Here, X is the set of all possible input values and n is the number of input bits. DAP is the maximum probability of output difference Δy when the input difference is Δx. For each input value x, $(\Delta x, \Delta y) \in [0, 31]$, the maximum differential approximate probability of the proposed 5-bit S-box is 8, i.e., DAP value is 0.25 $(8/2^5)$ (Table 4.6). Figure 4.10 gives the comparison of the differential approximate probability of the various 5-bit S-boxes. For an ideal S-box, the DAP should be $1/2^n$, which is practically not possible (i.e., it reveals no differential information about the input-output). In other words, the lower the occurrence (DAP), the higher the nonlinearity property. Thus, the proposed S-box shows a good resistance against the differential cryptanalysis even when the size is small (2^5).

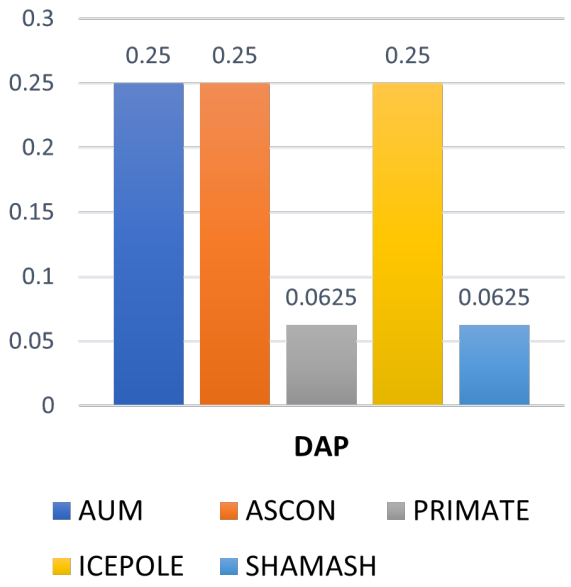


Fig. 4.10 Differential Approximate Probability Comparison

Table 4.6 The DAP matrix of the proposed 5-bit S-box

6	6	6	6	8	6	8	8	8	8	6	4	4	8	8	8
6	6	8	8	8	4	8	6	6	6	6	8	8	4	8	8

4.3.5 Boomerang Connectivity Table (BCT)

The Boomerang attack [219], proposed by David Wagner, is a differential style attack on block ciphers used to analyze the security of a block cipher. The Boomerang Connectivity Table (BCT) [220] is a systematic approach for calculating the connection probability for a Boomerang attack. Let $S : \{0,1\}^n \rightarrow \{0,1\}^n$ be an invert function, than for a given input difference Δx and output difference Δy for all values of input x, the the probability of boomerang of Δx, i.e., BCT of S is given by a 2^n x 2^n table T for all pairs of $(\Delta x, \Delta y)$ as follows:

$$\#\{x \in \{0,1\}^n | S^{-1}(S(x) \oplus \Delta y) \oplus S^{-1}(S(x \oplus \Delta x) \oplus \Delta y) = \Delta x\} \tag{4.4}$$

Here, S^{-1} is the inverse function of S-box. The values in the boomerang connectivity table are usually greater than or equal to that in the differential distribution table values in terms of strength. This relationship is described in [221]. Table 4.8 summarizes the occurrence of each element in BCT and DDT of the proposed S-box.

Table 4.7 Difference Distribution Table of the proposed 5-bit S-box

32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	0	0	4	0	2	0	4	0	4	0	0	0	0	4	0	0	0	0	2	0	2	0	0	0	2	2	4	0	0
0	2	0	0	0	2	0	0	0	0	0	0	2	0	2	0	4	0	6	0	0	4	2	0	0	0	0	4	2	0	2	0
0	0	2	0	0	4	0	0	2	0	0	4	0	2	2	0	0	0	0	0	0	0	0	2	0	8	0	0	4	0	2	0
0	2	0	2	0	2	2	2	4	4	2	2	0	0	2	0	0	0	0	0	0	0	2	0	0	0	2	2	0	0	2	0
0	0	2	4	0	0	2	0	0	0	2	2	2	2	4	0	0	0	2	2	0	0	2	2	2	0	0	0	2	0	0	0
0	0	0	0	0	0	0	0	2	0	0	4	0	2	0	0	0	2	4	4	2	2	0	2	0	0	2	2	0	2	2	0
0	0	4	4	0	0	2	2	2	0	2	0	0	0	2	2	4	2	2	0	0	0	0	0	2	2	0	0	0	0	0	0
0	0	2	2	2	0	4	0	0	0	0	2	2	2	0	0	0	0	0	0	0	0	2	0	0	0	2	0	6	0	4	2
0	4	0	2	0	0	0	2	0	0	2	0	0	0	0	2	0	0	0	0	2	2	2	6	0	2	0	2	0	2	2	0
0	0	0	0	2	0	0	2	2	0	2	0	2	0	6	0	4	2	0	0	0	0	2	0	2	0	2	2	0	2	0	0
0	0	0	0	0	2	2	4	0	0	0	2	0	0	0	2	0	2	2	0	4	2	0	2	2	0	2	2	0	0	0	2
0	0	2	0	2	4	0	0	2	0	0	0	0	2	0	0	2	0	0	0	2	0	2	2	2	2	2	0	0	2	0	4
0	0	2	0	0	0	0	0	4	2	2	0	2	2	0	2	0	0	0	2	2	2	0	0	0	2	2	0	0	0	2	4
0	0	0	0	0	0	2	2	0	2	0	0	0	6	0	0	2	4	0	2	2	2	2	2	2	0	0	0	0	0	2	0
0	0	0	2	2	2	2	0	2	0	0	0	2	2	2	0	0	0	0	2	2	0	0	0	0	0	2	0	4	4	2	0
0	0	0	0	0	2	0	2	0	0	2	0	0	2	0	0	2	0	2	2	0	2	2	2	0	0	0	4	0	4	0	4
0	0	2	0	2	2	2	0	0	0	2	2	2	0	0	2	0	0	2	0	0	2	8	4	0	0	0	0	0	0	0	0
0	4	0	0	2	0	0	0	0	2	0	2	0	0	6	8	0	0	0	0	0	0	0	2	2	0	0	2	2	0	0	0
0	0	0	0	4	0	2	2	0	0	2	2	4	0	0	0	0	0	2	4	2	0	0	0	0	2	2	2	0	0	2	0
0	0	0	2	2	2	0	0	0	2	0	0	0	0	0	0	2	0	0	2	2	4	0	0	2	0	0	2	6	2	2	0
0	2	0	0	0	0	0	0	2	2	0	2	4	2	2	0	2	0	0	2	0	2	0	0	0	4	0	0	0	0	2	4
0	0	0	0	0	4	6	4	0	2	0	0	2	0	4	2	2	0	0	0	2	2	0	0	0	0	0	0	0	2	0	0
0	2	4	0	2	2	0	2	0	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	4	4	2	0	2	2	2	0
0	6	0	4	2	0	0	0	0	0	2	0	2	0	0	0	0	4	0	2	2	2	2	0	2	0	0	0	0	0	0	2
0	0	0	0	2	0	0	0	4	2	4	2	0	0	0	2	0	4	0	2	4	0	0	0	0	0	6	0	0	0	0	0
0	0	0	4	2	0	0	2	0	0	2	0	0	6	0	0	0	8	2	2	0	0	0	0	2	0	0	0	0	2	0	0
0	4	0	0	2	0	2	0	0	0	2	0	2	0	0	4	2	0	0	0	2	0	0	0	6	0	2	0	0	2	0	2
0	0	4	0	0	0	2	0	4	4	0	0	0	0	0	2	0	0	0	0	0	2	2	2	0	2	2	0	0	0	2	4
0	2	4	0	0	0	0	0	0	4	2	0	2	0	0	2	0	0	6	2	2	0	0	0	0	0	0	0	0	2	0	4
0	2	0	2	2	0	0	2	2	0	2	2	2	0	0	0	2	0	2	2	0	0	2	0	2	4	0	2	0	0	0	0
0	2	2	4	2	0	2	2	0	2	0	0	0	0	0	0	0	4	0	0	0	0	0	2	0	0	2	4	2	0	2	0

4.3.6 Feistel counterpart of BCT (FBCT)

The Boomerang Connectivity Table (BCT) is only valid for an S-box that is part of an S-layer in an SPN cipher but not for the S-box that is part of a Feistel cipher. In [222], an extension of BCT is proposed to address the counterpart for a Feistel cipher. Like BCT, Feistel counterpart of BCT (FBCT) for the pairs of $(\Delta x, \Delta y)$ can be given as follows:

$$\#\{x \in \{0,1\}^n | S(x) \oplus S(x \oplus \Delta x) \oplus S(x \oplus \Delta y) \oplus S(x \oplus \Delta x \oplus \Delta y) = 0\} \tag{4.5}$$

Table 4.9 exhibits the FBCT of the proposed 5-bit S-box. Here, the values 32, 8, 4 and 0 appear 94, 42, 186 and 702 times, respectively, in the FBCT of the proposed S-box. The highest value in FBCT, known as Feistel boomerang uniformity (β^F), is 8. Here, the FBCT values at the first row, first column, and diagonal is 2^n (i.e., 32) which are neglected. The first row and first column with the values 2^n are known as ladder switch, whereas the diagonal

Table 4.8 The occurrence of each element in BCT and DDT of the proposed S-box

	32	**16**	**14**	**12**	**10**	**8**	**6**	**4**	**2**	**0**
BCT	63	2	1	8	9	30	72	178	228	433
DDT	1	-	-	-	-	4	13	72	297	637

with the values 2^n is known as Feistel switch. Some common properties of any FBCT are as follows [222]:

1. Symmetry: for all $0 \leq \Delta x, \Delta y \leq 2^n - 1$,
   $\text{FBCT}(\Delta x, \Delta y) = \text{FBCT}(\Delta y, \Delta x)$
2. Fixed values:
   (a) First row: for all $0 \leq \Delta y \leq 2^n - 1$,
   $\text{FBCT}(0, \Delta y) = 2^n$
   (b) First column: for all $0 \leq \Delta x \leq 2^n - 1$,
   $\text{FBCT}(\Delta x, 0) = 2^n$
   (c) Diagonal: for all $0 \leq \Delta x \leq 2^n - 1$,
   $\text{FBCT}(\Delta x, \Delta x) = 2^n$
3. Multiplicity: for all $0 \leq \Delta x, \Delta y \leq 2^n - 1$,
   $\text{FBCT}(\Delta x, \Delta y) \equiv 0 \bmod 4$
4. Equalities: for all $0 \leq \Delta x, \Delta y \leq 2^n - 1$,
   $\text{FBCT}(\Delta x, \Delta y) = \text{FBCT}(\Delta x, \Delta x \oplus \Delta y)$

4.3.7 High Degree of Avalanche Effect

A slight change in input bits that significantly change output bits is known as an avalanche effect. When a change in one input bit results in a change in at least half of the output bits, it is called strict avalanche criterion (SAC)), i.e., for any *n* bits input, at least *n/2* bits in output must differ [223]. For any block cipher, an avalanche of change is an essential property and could be boosted by an efficient S-box design that offers high resistance to differential attacks.

Table 4.9 Feistel counterpart of BCT (FBCT) of the proposed 5-bit S-box

32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
32	32	0	0	0	0	4	4	0	0	0	0	0	0	0	0	4	4	0	0	0	0	8	8	0	0	4	4	0	0	0	0
32	0	32	0	0	0	0	0	0	0	0	0	4	0	4	0	8	8	8	8	0	0	0	0	0	0	0	0	0	4	0	4
32	0	0	32	4	0	0	4	0	4	4	0	0	0	0	0	0	4	4	0	4	4	4	4	4	4	4	4	4	4	4	4
32	0	0	4	32	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	4	0	0
32	0	0	0	0	32	0	0	4	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	4
32	4	0	0	0	0	32	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	8	0	0	0
32	4	0	4	4	0	4	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	4	0	0
32	0	0	0	0	4	0	0	32	0	0	0	0	4	0	0	0	4	4	0	4	4	0	0	0	4	4	0	4	4	0	0
32	0	0	4	0	0	0	0	0	32	4	0	0	0	0	0	0	4	8	0	0	0	0	0	4	0	0	8	0	0	0	0
32	0	0	4	0	0	0	0	0	4	32	0	0	0	0	0	0	4	8	0	0	0	0	0	8	0	0	4	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	8	0	0	0	0	0	0	8	0	0	0	0	0	0
32	0	4	0	0	0	0	0	0	0	0	0	32	0	4	0	0	0	0	0	0	0	4	0	0	0	4	0	0	0	0	0
32	0	0	0	0	4	0	0	4	0	0	0	0	32	0	0	0	0	0	0	4	0	0	0	0	4	0	0	0	0	0	0
32	0	4	0	0	0	0	0	0	0	0	0	4	0	32	0	0	0	0	4	4	0	4	0	4	0	4	0	0	4	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	4	4	0	0	4	4	0	0	0	0	0
32	4	8	0	0	0	0	0	0	0	0	0	0	0	0	0	32	4	8	0	0	0	0	0	0	0	0	0	0	0	0	0
32	4	8	4	0	0	0	0	4	4	4	0	0	0	0	0	4	32	4	8	0	0	0	0	4	4	0	4	0	0	0	0
32	0	8	4	0	0	0	0	4	8	8	8	0	0	0	0	8	4	32	0	0	0	0	0	8	8	4	8	0	0	0	0
32	0	8	0	0	0	0	0	0	0	0	0	0	0	4	0	0	8	0	32	0	0	0	0	0	0	0	0	0	4	0	0
32	0	0	4	0	0	0	0	4	0	0	0	0	4	4	0	0	0	0	0	32	0	0	4	0	4	4	0	4	0	0	0
32	0	0	4	0	0	0	0	4	0	0	0	0	0	0	4	0	0	0	0	0	32	4	0	0	0	4	0	0	4	0	0
32	8	0	4	0	0	0	0	0	0	0	0	4	0	4	4	0	0	0	0	0	4	32	8	4	4	4	0	0	0	0	0
32	8	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	8	32	0	0	0	0	0	0	0	0
32	0	0	4	0	0	0	0	0	4	8	0	0	0	4	0	0	4	8	0	0	0	4	0	32	0	0	4	0	0	0	0
32	0	0	4	4	0	0	0	4	0	0	8	0	4	0	4	0	4	8	0	4	0	4	0	0	32	4	0	0	4	0	0
32	4	0	4	0	4	8	4	4	0	0	0	4	0	4	4	0	0	4	0	4	4	4	0	0	4	32	4	8	4	0	4
32	4	0	4	0	0	0	0	0	8	4	0	0	0	0	0	0	4	8	0	0	0	0	0	4	0	4	32	0	0	0	0
32	0	0	4	0	0	8	0	4	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	8	0	32	0	0	4
32	0	4	4	4	0	0	4	4	0	0	0	0	0	4	0	0	0	0	4	0	4	0	0	0	4	4	0	0	32	4	4
32	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	32	0
32	0	4	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	4	4	0	32

As introduced by Webster and Tavares [224], we can confirm whether an S-box fulfil the SAC property or not by considering a 5-bit input X and a set of input vectors, $X_1, X_2, ..., X_5$, derived by changing j^{th} bit only. Its corresponding 5-bit output vectors, $Y_1, Y_2, ..., Y_5$, can be assigned using a substitution function, $Y_j = S(X_j)$. An avalanche vector, V_j, can be computed by XORing output vector Y and Y_j, i.e., $V_j = Y \oplus Y_j$. A 5x5 dependency matrix, A, can be calculated by adding i^{th} bit of V_j to $a_{i,j}$, where $a_{i,j}$ is the ith element of the matrix A. Now, depending on the vector X, repeat the above steps multiple times and then divide each element of matrix A by 2^n (n is the number of input/output bits) to compute SAC matrix.

The average avalanche of change effect of the proposed algorithm is 0.51 (51.00%) (Table 4.10). If the average of each element and matrix is close the ideal value, 0.5 (50%), then the S-box is considered to satisfy SAC. Thus, the proposed S-box fulfils the SAC property. Figure 4.11 compares the avalanche of change effect (SAC) of the proposed S-box

and other existing 5-bit S-boxes. It can be observed that the average SAC value of the proposed S-box is closest to the ideal value (0.5), and thus it beats the competition.

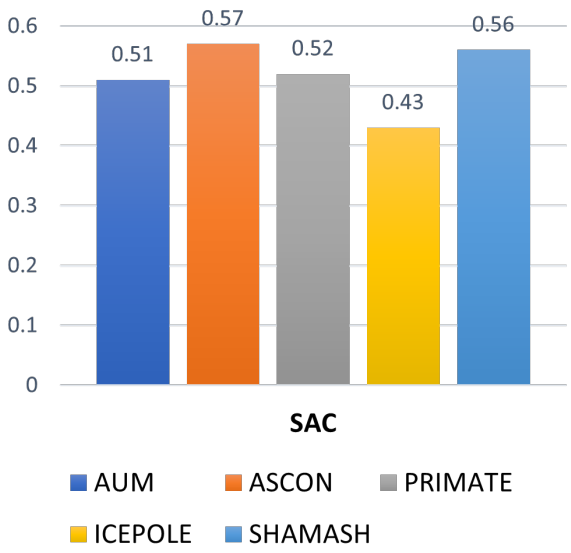


Fig. 4.11 Avalanche effect/Strict Avalanche Criterion (SAC)

Since SAC works on the concepts of completeness along with the avalanche effect, it can be proven below. Let's consider a multi-output function $S : F_2^n \rightarrow F_2^m$ which generates m-bit output in responds to n-bit input. Let X be an input where $X = (x_1, x_2, ...x_n)$, $x_p \in \{0,1\}$ and X^i be another input with change in i^{th} bit(s) where $i = 1,2,...n$. Let $F(X)$ and $F(X^i)$ be the corresponding outputs, $F \in T$ where $T \subset S_2^5$.

Definition 1: For multi-output function, $S : F_2^n \rightarrow F_2^m$, each output bit relies on every input bit, called *Completeness*, as follows:

$$C = 1 - \frac{1}{nm}\#\{(i,j)|b_{ij} = 0\} \tag{4.6}$$

where $b_{ij} = \#\{X \in T|(f(X))_j \neq (f(X^i))_j\}$, is the total of resulting output bits when two inputs with i^{th} bit difference is passed, also $i = 1,2,...n$ and $j = 1,2,...m$. The value of completeness, C, closer to 1 offers strong non-linearity. Our S-box satisfies this property ($C = 1$) as it generates unique pairs of input-output using 5-bits.

Definition 2: For any function, F, if a change in an input reflects the change in half of the output, then it presents a strict avalanche property (SAC) as follows:

$$Avl\_eft_{(strict)} = 1 - \frac{2}{\#T * nm}\sum_{i=1}^{n}\sum_{j=1}^{m}|b_{ij} - \frac{1}{2}\#T \tag{4.7}$$

Here, $\#T$ is the number of inputs. Similar to *completeness*, the $Avl\_eft_{(strict)}$ value close to 1 shows high avalanche effect. Our S-box shows strict avalanche criterion (SAC) value 0.51 and thus $Avl\_eft_{(strict)} \approx 1$.

4.3.8 Bit Independence Criterion (BIC)

Another essential property, *bit independence criterion (BIC)* introduced by Webster and Tavares [224], where each input bit affects/changes every output bit, i.e., a change in i^{th} bit reflects an independent change of output bits j and k, where $i, j, k \in (1, 2, .., n)$ and $j \neq k$. According to this, two output bits of an S-box, f_j and f_k, where $j \neq k$, if $f_j \oplus f_k$ shows high nonlinearity and fulfil the SAC, the S-box has has good BIC property.

BIC-SAC property can be computed by determining output vectors $Y_1, Y_2, ..., Y_5$ for each input vector X as defined in the previous section (4.3.7). An avalanche vector, $V_{i,j,k}$, can be computed by XORing $P_{i,j}$ and $Q_{i,j}$, i.e., $V_{i,j,k} = P_{i,j} \oplus Q_{i,j}$. Here, $P_{i,j}$ is the XORed value of ith and j^{th} bit of Y and $Q_{i,j}$ is the XORed value of ith and j^{th} bit of Y_k, where $i, j, k \in \{1, 2, ..., 5\}$. Now, depending on the vector X, repeat the above steps multiple times and then divide each element of matrix A by 2^n (n is the number of input/output bits) to obtain BIC-SAC matrix.

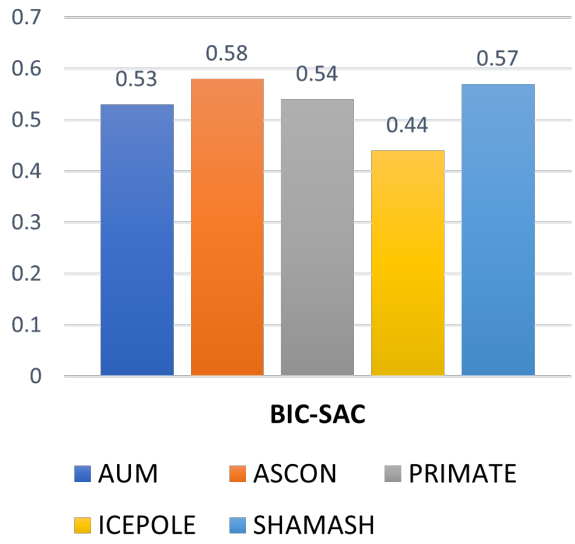


Fig. 4.12 Bit Independence Criterion - SAC

To better the resistance property, an S-box must show the BIC-SAC value close to 0.5. The BIC-SAC value for the proposed S-box is 0.53, and thus it satisfies the BIC-SAC property. Figure 4.12 illustrates a comparison of BIC-SAC values of the proposed S-box and other 5-bit S-boxes competitors where the average BIC-SAC value of the proposed S-box is closest to the ideal value (0.5), and thus it wins the race.

4.3.9 Algebraic Attacks

The proposed S-box has a simple but robust structure. Based on the design criteria we proposed for the 5-bit S-box, as discussed in Section 3.2.3, the possible S-boxes are $31! \approx 8.22 * 10^{33}$ which is huge and noticeably more than that of the 4-bit S-box, i.e., $15! \approx 1.3 * 10^{12}$. Also, the proposed 5-bit S-box make use of a complex dynamic chaotic system to create randomness of the element in the S-box, and it is tough to breakthrough. Moreover, the results achieved through the S-box Evaluation Toolbox (SET) [225], the algebraic immunity of the proposed S-box is 2, which is excellent and similar to its 5-bit S-box competitors.

Table 4.10 Cryptanalysis of various 5-bit S-boxes

S-box (5-bit)	**Linear Probability (LP)**	**Nonlinearity (H_d)**	**DAP**	**SAC**	**BIC-SAC**
Proposed	**0.25**	**2.625**	**0.25**	**0.51**	**0.53**
ASCON	0.25	2.5	0.25	0.57	0.58
PRIMATE	0.375	2.5	0.0625	0.52	0.54
ICEPOLE	0.25	1.531	0.25	0.43	0.44
SHAMASH	0.375	2.5	0.0625	0.56	0.57

Table 4.10 exhibits that the proposed algorithm shows high resistance to linearity by achieving a value close to zero, the highest value for nonlinearity (higher the better), close to value 0.5 for strict avalanche criterion and BIC-SAC count. Thus, the proposed algorithm ensures the best security resistance against various cryptanalysis. Figure 4.13 visualize this comparison and proves the security strength of the proposed model. The compared version of PRIMATE is PRIMATE-80B.

4.4 Execution results of the full algorithm, AUM

When all the modules, permutation (through transpose), addRoundkey and 5-bit S-box are executed together for the considered 32-bit input plaintext, transpose key and 32-bit original key for sixteen times, following results are observed (Figure 4.14).

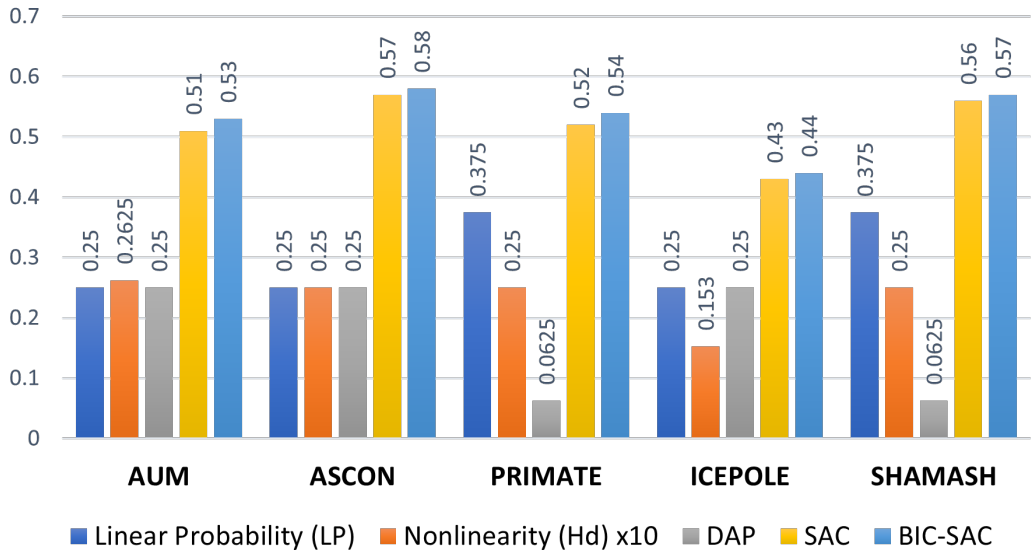


Fig. 4.13 Cryptanalysis of various 5-bit S-boxes

Transpose Key: 7 12 9 14 3 5 11 8 2 13 4 10 15 0 1 6

Original Key: 6 1 13 5 3 2 12 15

S-box (5-bit): {{10 3 11 22 17 4 1 8 12 28 23 18 26 6 31 20},
{15 24 29 13 14 19 30 5 25 27 7 0 16 21 2 9}}

Plaintext (32-bit): 1 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0

Cipher (32-bit): 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 1 1

```
*****************************************
         Full execution of AUM
*****************************************

Transpose Key:
--------------
7 12 9 14 3 5 11 8 2 13 4 10 15 0 1 6

Original Key:
-------------
6 1 13 5 3 2 12 15

S-box (5-bit):
--------------
10 3 11 22 17 4 1 8 12 28 23 18 26 6 31 20
15 24 29 13 14 19 30 5 25 27 7 0 16 21 2 9


Plaintext (32-bit):
-------------------
1 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0

Cipher (32-bit):
----------------
1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 1 1 _
```

The following Figure 4.15 shows how sixteen distinct subkeys are generated from the user-provided original key.

```
**********************************
         Key Generation
**********************************
8-digit Original key:
 6  1  13  5  3  2  12  15
 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1
 Temporary Subkey[0]:  0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1
16-Subkeys:
 Subkey[0]: 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1
 Subkey[1]: 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0
 Subkey[2]: 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0
 Subkey[3]: 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0
 Subkey[4]: 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1
 Subkey[5]: 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1
 Subkey[6]: 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1
 Subkey[7]: 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0
 Subkey[8]: 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1
 Subkey[9]: 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1
 Subkey[10]: 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1
 Subkey[11]: 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1
 Subkey[12]: 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0
 Subkey[13]: 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1
 Subkey[14]: 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1
 Subkey[15]: 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0
```

Fig. 4.15 AUM subkey generation from the original key

The following Table 4.11 shows twenty 32-bit input plaintext (random) and their' corresponding 32-bit ciphers while full execution (16 rounds) of AUM. Here, one bit is changed in the input plaintext to examine the avalanche of change (strict avalanche effect (SAC)) in the output ciphers. The obtained results show that the minimum bits changes in the cipher are 12, whereas the maximum bit changes in the cipher are 20 while making a 1-bit change in the plaintext input. The average number of bit changes in the output (cipher) is 15.53 (around 49%), which is closed to the desired value (50%). Thus, AUM shows a strict avalanche effect and proves high immunity.

The achieved experimental results are available in form of soft reports (hardware implementation) or software programs developed to evaluate them (mainly, cryptanalysis) for validation purpose. Some the examples included are Figure 4.3 Behavioural simulation of AUM (32 clock cycle), page 61 and Figure 4.15 AUM subkey generation from the original key, page 80.

Table 4.11 Plaintext and their' corresponding cipher of AUM (Avalanche effect)

Plaintext input with 1 bit change (32-bit)	**Cipher (32-bit)**	**Bit Difference**
10100110000111001110111100100000	10010100111100011101101010010011	-
10100110000111001110111100100001	10100010010100100010011000000100	19
10100110000111001110111100100010	11000011110011100001100110101001	19
10100110000111001110111100100100	10111010001001100110110111111111	20
10100110000111001110111100101000	00010000001100100011001000011010	13
10100110000111001110111100110000	10101101100100110100110100010001	14
10100110000111001110111100000000	10000100100000100000101010111111	12
10100110000111001110111101100000	01110000000100111001100100101100	18
10100110000111001110111110100000	01000010001001001101110010000001	14
10100110000111001110111000100000	10000111010001111111111111000000	15
10100110000111001110110100100000	01100100011101111111111101111001	15
10100110000111001110011100100000	01111110011010111101101100110000	14
10100110000111001111111100100000	11010101011001000110001010110111	12
10100110000111001100111100100000	01010111011100110000001011110101	14
10100110000111001010111100100000	10111001101010000001000101011000	18
10100110000111000110111100100000	11011100001101011111001110100101	12
10100110000111011110111100100000	00010100000110001100000100110010	13
10100110000111101110111100100000	10001101011010100001001001000101	16
10100110000110001110111100100000	00001010110111110110111000000110	17
10100110000101001110111100100000	01111100001111010110011100001100	20
	Average	**15.53 (49%)**

Chapter 5

Research Contribution

The research was started with an aim to develop a novel lightweight cryptography algorithm for resource-constrained IoT devices to trade-off amongst cost, performance and security. It targets IoT devices such as RFID cards, smart cards, sensors, actuators where the message size are tiny ($< 2Kb$). With an objective to design a lighter but secure permutation, simple but fast and robust S-box and more lightweight key scheduling, all the research challenges (section 2.6) are addressed by the proposed algorithm, AUM, as follows:

- Small block size (32-bit) and small key size (32-bit) compared to existing algorithms (section 3.1)
- 32-bit permutation through Transpose technique (section 3.2.1)
- A novel S-Box design, 5-bit S-Box (section 3.2.3)
- Lightweight and random subkey generation (section 3.2.4)
- Highly Secure due to (section 4.3)
  - Use of three random/anonymous inputs in each round of execution
  - 5-bit S-Box (32 possible values) instead of 4-bit S-Box (16 possible values)
  - Random subkey generation technique
- Small memory, low processing, low energy and small physical area (GE) requirements (section 4.2)

Thus, this research contributes to the field of Cryptography, exceptionally Lightweight Cryptography for resource-constrained IoT devices, by proposing a novel algorithm, AUM, with the above mentioned unique features.

Chapter 6

Conclusion

Due to the exponential growth in the number of IoT devices in various domains, IoT security is one of the main concerns. For resource-constrained IoT devices, lightweight cryptography is an effective way to secure communication by transforming the data. The well-defined LWC characteristics (cost, performance and security) by NIST are studied and compared for various lightweight cryptography algorithms, and further research gaps and open research challenges are highlighted in this research. The literature review reveals that none of the existing LWC algorithms fulfils all the hardware and software cost and performance criteria but performs at their best in the specified environment. Also, most of these algorithms are victims of various cyber attacks (cryptanalysis). Consequently, there is a need for a lightweight algorithm(s) with trade-offs amongst cost, performance and security.

This research proposes a lightweight cryptography algorithm, AUM. It targets IoT devices such as RFID tags, smart cards, sensors, and actuators where message sizes are usually small (message size < 2Kb). AUM performs 16 rounds using a small key (32-bit) over the same bit size block. Each round consists of three main functions: permutation through transpose, addRoundKey (apply one of the 16 subkeys), and substitution using a 5-bit S-box. First, the research proposes a lightweight permutation using a 2-D array to disperse the 32-bit plaintext through a random transpose key. Then, one of the 16 subkeys is applied to offer nonlinearity to the algorithm. Finally, a robust cipher is achieved by passing 32-bit data through a novel and a compact 5-bit S-box. Here, transpose offers diffusion property, whereas addRoundKey and S-box offer confusion property to straighten the algorithm. A unique lightweight technique to extract sixteen distinct subkeys from an original key is also introduced to straighten the algorithm. Since S-box is a fundamental and the only component that embeds a nonlinear functionality in any SPN based cryptography algorithm, the primary focus of the proposed lightweight cryptography algorithm, AUM, is on S-box design.

Many S-boxes with various bit lengths (e.g., 4-bit, 5-bit, 6-bit, and 8-bit) are proposed by various researchers. Out of these proposed S-boxes, 4-bit S-box from PRESENT is widely used due to its low resource requirements in constrained environments. Also, the 8-bit S-box attracts designers due to its promising security structure but witnesses high implementation costs. Due to its odd size (not the multiple of two, i.e., $size \neq 2^n$), 5-bit S-box is not a popular choice. Since the input block size is usually even (usually, a multiple of 4), an odd-size S-box is avoided to provide the flexibility to split the input block over the S-box size (generally, 4, 6 or 8 bits). However, our proposed 5-bit S-box resolves this issue by introducing various schemes to fit different popular input block sizes. The other S-boxes, 3-bit and 6-bit, are far from the competition because of either low-security support or expensive implementation reasons.

This research proposes a new 5-bit S-box design based on the latest chaotic mapping theory for lightweight cryptography algorithms, particularly for small/tiny messages in IoT devices like RFID tags, sensors, and smart cards. The proposed design (including chaotic mapping) is flexible to fit various block sizes, ranging from 32-bit to 256-bit blocks, without any challenge. Also, the experiment results on popular ASIC platforms and its comparison with other n-bit S-boxes competitors clearly demonstrate the significance of the 5-bit S-box in terms of performance and execution cost. Furthermore, it proves that a small number of resources inclusion (approximately 150GE and even less than $0.1\mu W$ of Power) can increase a significant level of security (more than double) and encourage to use of 5-bit S-box over 4-bit S-box. Further, the cryptanalysis of the proposed 5-bit S-box and its comparison with the same bit-size S-boxes over the essential security measures such as bijective property, nonlinearity, linearity (LP), differential probability (cryptanalysis), degree of avalanche effect, bit Independence criteria (BIC) and algebraic attacks, exhibits the superiority of the proposed S-box.

In summary, the implementation of the proposed algorithm, AUM, on popular ASIC platforms and a comparison with its competitors over different cost and performance metrics demonstrate its compactness and efficiency. However, this could extend by implementing proposed algorithms on popular FPGA platforms and comparing results over more metrics in future. Also, the proposed algorithm shows strong resistance against various cryptanalysis and stands ahead of all its competitors. Thus, the proposed algorithm, AUM, and its core component, 5-bit S-box, prove the trade-off between cost, performance and security.

References

[1] C. Buck and C. Winkler, "The iot story," Jan 2020. [Online]. Available: https://new.siemens.com/global/en/company/stories/research-technologies/digitaltwin/iot-story.html

[2] GartnerInc, "Forecast: It services for iot, worldwide, 2019-2025," Aug 2021. [Online]. Available: https://www.gartner.com/en/documents/4004741/forecast-it-services-for-iot-worldwide-2019-2025

[3] J. Clark, "What is the internet of things, and how does it work?" Aug 2020. [Online]. Available: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/

[4] "What is ..." [Online]. Available: https://aws.amazon.com/what-is/iot/

[5] "What is iot (internet of things)?: Microsoft azure." [Online]. Available: https://azure.microsoft.com/en-us/overview/internet-of-things-iot/what-is-the-internet-of-things/#overview

[6] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the internet of things," *IEEE Internet Computing*, vol. 14, no. 1, pp. 44–51, 2010.

[7] N. Pérez Moldón, "Security in iot ecosystems."

[8] E. Brown, "21 open source projects for iot," *Linux. com. Retrieved*, vol. 23, 2016.

[9] S. Charmonman and P. Mongkhonvanit, "Internet of things in e-business," in *Proceeding of the 10th International Conference on e-Business King Mongkut's University of Technology Thonburi*, 2015, pp. 1–9.

[10] "The trouble with the internet of things," Aug 2015. [Online]. Available: https://data.london.gov.uk/blog/the-trouble-with-the-internet-of-things/

[11] K. McKay, L. Bassham, M. S. Turan, and N. Mouha, "Report on lightweight cryptography (nistir8114)," *National Institute of Standards and Technology (NIST)*, 2017.

[12] B. J. Mohd and T. Hayajneh, "Lightweight block ciphers for iot: Energy optimization and survivability techniques," *IEEE Access*, vol. 6, pp. 35 966–35 978, 2018.

[13] A. Banafa, "Three major challenges facing iot," *IEEE IoT Newsletter*, 2017.

[14] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–18, 2017.

[15] W. Feng, Y. Qin, S. Zhao, and D. Feng, "Aaot: Lightweight attestation and authentication of low-resource things in iot and cps," *Computer Networks*, vol. 134, pp. 167–182, 2018.

[16] S. Singh, "A brief history of cryptography from caesar to bletchley park," in *Colossus*. Oxford University Press, 2006.

[17] D. Kahn, *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. Simon and Schuster, 1996.

[18] F. Cohen, "A short history of cryptography," *http://www. all. net/books/ip/Chap2-1. html*, 1990.

[19] D. Davies, "A brief history of cryptography," *Information Security Technical Report*, vol. 2, no. 2, pp. 14–17, 1997.

[20] R. F. Burton, F. Arbuthnot *et al.*, *The Kama Sutra of Vatsyayana: Translated from the Sanscrit. In seven parts, with preface, introduction, and concluding remarks.* Kama Shastra Society of London and Benares, 1883.

[21] M. Jackob, "Sans info sec reading room," *History of Encryption*, 2001.

[22] T. M. Damico, "A brief history of cryptography," *Inquiries Journal*, vol. 1, no. 11, 2009.

[23] L. D. Smith, *Cryptography: The science of secret writing*. Courier Corporation, 1955.

[24] "World war i cryptography," Dec 2021. [Online]. Available: https://en.wikipedia.org/wiki/World_War_I_cryptography

[25] "World war ii cryptography," Jan 2022. [Online]. Available: https://en.wikipedia.org/wiki/World_War_II_cryptography

[26] R. L. Rivest, "Cryptography," in *Algorithms and complexity*. Elsevier, 1990, pp. 717–755.

[27] W. Stallings and M. P. Tahiliani, "Cryptography and network security: principles and practice, vol. 6," 2014.

[28] W. Stallings, *Cryptography and Network Security: Principles and Practice*. [Online]. Available: https://www.pearson.com/us/higher-education/product/Stallings-Cryptography-and-Network-Security-Principles-and-Practice-6th-Edition/9780133354690.html

[29] I. Bhardwaj, A. Kumar, and M. Bansal, "A review on lightweight cryptography algorithms for data security and authentication in iots," in *2017 4th International Conference on Signal Processing, Computing and Control (ISPCC)*. IEEE, 2017, pp. 504–509.

[30] G. Bansod, N. Raval, and N. Pisharoty, "Implementation of a new lightweight encryption design for embedded security," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 1, pp. 142–151, 2014.

[31] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," *Journal of Cryptographic Engineering*, vol. 8, no. 2, pp. 141–184, 2018.

[32] T. Suzaki and K. Minematsu, "Improving the generalized feistel," in *International Workshop on Fast Software Encryption*. Springer, 2010, pp. 19–39.

[33] A. Bogdanov, "Cryptanalysis of the keeloq block cipher." *IACR Cryptology ePrint Archive*, vol. 2007, p. 55, 2007.

[34] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A block cipher for low energy," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2015, pp. 411–436.

[35] L. Wen, M. Wang, A. Bogdanov, and H. Chen, "Multidimensional zero-correlation attacks on lightweight block cipher hight: improved cryptanalysis of an iso standard," *Information Processing Letters*, vol. 114, no. 6, pp. 322–330, 2014.

[36] D. Khovratovich, G. Leurent, and C. Rechberger, "Narrow-bicliques: cryptanalysis of full idea," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 392–410.

[37] E. Biham, O. Dunkelman, and N. Keller, "A related-key rectangle attack on the full kasumi," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2005, pp. 443–461.

[38] T. Saito, "A single-key attack on 6-round kasumi." *IACR Cryptology ePrint Archive*, vol. 2011, p. 584, 2011.

[39] M. Ågren, "Some instant-and practical-time related-key attacks on ktantan32/48/64," in *International Workshop on Selected Areas in Cryptography*. Springer, 2011, pp. 213–229.

[40] N. T. Courtois, G. V. Bard, and D. Wagner, "Algebraic and slide attacks on keeloq," in *International Workshop on Fast Software Encryption*. Springer, 2008, pp. 97–115.

[41] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel, "A practical attack on keeloq," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 1–18.

[42] M. Walter, S. Bulygin, and J. Buchmann, "Optimizing guessing strategies for algebraic cryptanalysis with applications to epcbc," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 175–197.

[43] X.-j. Zhao, T. Wang, and Y. Zheng, "Cache timing attacks on camellia block cipher." *IACR Cryptology ePrint Archive*, vol. 2009, p. 354, 2009.

[44] K. Jeong, C. Lee, and J. I. Lim, "Improved differential fault analysis on lightweight block cipher lblock for wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, p. 151, 2013.

[45] H. Yoshikawa, M. Kaminaga, A. Shikoda, and T. Suzuki, "Secret key reconstruction method using round addition dfa on lightweight block cipher lblock," in *2014 International Symposium on Information Theory and its Applications*. IEEE, 2014, pp. 493–496.

[46] Y. Kim and H. Yoon, "First experimental result of power analysis attacks on a fpga implementation of lea." *IACR Cryptology ePrint Archive*, vol. 2014, p. 999, 2014.

[47] K. Jeong, H. Kang, C. Lee, J. Sung, and S. Hong, "Biclique cryptanalysis of lightweight block ciphers present, piccolo and led." *IACR Cryptology ePrint Archive*, vol. 2012, p. 621, 2012.

[48] H. AlKhzaimi and M. M. Lauridsen, "Cryptanalysis of the simon family of block ciphers." *IACR Cryptology ePrint Archive*, vol. 2013, p. 543, 2013.

[49] R. Rabbaninejad, Z. Ahmadian, M. Salmasizadeh, and M. R. Aref, "Cube and dynamic cube attacks on simon32/64," in *2014 11th International ISC Conference on Information Security and Cryptology*. IEEE, 2014, pp. 98–103.

[50] F. Abed, E. List, S. Lucks, and J. Wenzel, "Differential and linear cryptanalysis of reduced-round simon," *Cryptology ePrint Archive, Report 2013/526*, 2013.

[51] CSRC, "Lightweight cryptography | csrc," https://csrc.nist.gov/projects/lightweight-cryptography, Jan 2017, (Accessed on 10/30/2020).

[52] B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, "A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues," *Journal of Network and Computer Applications*, vol. 58, pp. 73–93, 2015.

[53] O. Toshihiko, "Lightweight cryptography applicable to various iot devices," *NEC Technical Journal*, vol. 12, no. 1, pp. 67–71, 2017.

[54] A. Biryukov and L. P. Perrin, "State of the art in lightweight symmetric cryptography," 2017.

[55] Z. Sheng, S. Yang, Y. Yu, A. V. Vasilakos, J. A. McCann, and K. K. Leung, "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities," *IEEE Wireless Communications*, vol. 20, no. 6, pp. 91–98, 2013.

[56] A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in *Annual international cryptology conference*. Springer, 2005, pp. 293–308.

[57] W. J. Okello, Q. Liu, F. A. Siddiqui, and C. Zhang, "A survey of the current state of lightweight cryptography for the internet of things," in *2017 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2017, pp. 292–296.

[58] "Cybersecurity | nist," https://www.nist.gov/cybersecurity, (Accessed on 02/07/2022).

[59] "Iso - standards," https://www.iso.org/standards.html, (Accessed on 02/07/2022).

[60] "Cryptrec | about cryptrec," https://www.cryptrec.go.jp/en/about.html, (Accessed on 02/07/2022).

[61] "Ecrypt - european network of excellence for cryptology," https://www.iacr.org/newsletter/v21n2/ecrypt.html, (Accessed on 02/07/2022).

[62] "National security agency/central security service > home," https://www.nsa.gov/, (Accessed on 02/07/2022).

[63] "Cryptolux," https://cryptolux.org/index.php/Home, (Accessed on 02/07/2022).

[64] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2007, pp. 450–466.

[65] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher clefia (extended abstract). fse 2007. lncs, vol. 4593," 2007.

[66] I. T. L. Computer Security Division, "Lightweight cryptography: Csrc," Nov 2021. [Online]. Available: https://csrc.nist.gov/Projects/lightweight-cryptography

[67] N. F. Pub, "197: Advanced encryption standard (aes)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[68] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: a very compact and a threshold implementation of aes," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 69–88.

[69] S. Sallam and B. D. Beheshti, "A survey on lightweight cryptographic algorithms," in *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE, 2018, pp. 1784–1789.

[70] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2008, pp. 89–103.

[71] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, "Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms," *Science China Information Sciences*, vol. 58, no. 12, pp. 1–15, 2015.

[72] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. Sim, and Y. Todo, "Gift: A small present towards reaching the limit of lightweight encryption (full version)," Tech. Rep., 2017.[Online]. Available: https://infoscience. epfl. ch/record . . . , Tech. Rep.

[73] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "Gift-cofb," *Submission to Round*, vol. 1, 2019.

[74] Y. Liu and Y. Sasaki, "Related-key boomerang attacks on gift with automated trail search including bct effect," in *Australasian Conference on Information Security and Privacy*. Springer, 2019, pp. 555–572.

[75] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Fpga-based assessment of midori and gift lightweight block ciphers," in *International Conference on Information and Communications Security*. Springer, 2018, pp. 745–755.

[76] A. Adomnicai, Z. Najm, and T. Peyrin, "Fixslicing: A new gift representation." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 412, 2020.

[77] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The skinny family of block ciphers and its low-latency variant mantis," in *Annual International Cryptology Conference*. Springer, 2016, pp. 123–153.

[78] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "Twine: A lightweight, versatile block cipher," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011.

[79] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, and T. Akishita, "Regaz zoni, f.: Midori: A block cipher for low energy (extended version)," Cryptology ePrint Archive, Report 2015/1142, Tech. Rep., 2015.

[80] C. H. Lim and T. Korkishko, "mcrypton–a lightweight block cipher for security of low-cost rfid tags and sensors," in *International Workshop on Information Security Applications*. Springer, 2005, pp. 243–258.

[81] C. H. Lim, "A revised version of crypton: Crypton v1. 0," in *International Workshop on Fast Software Encryption*. Springer, 1999, pp. 31–45.

[82] J. Daemen, M. Peeters, G. Assche, and V. Rijmen, "The noekeon block cipher," in *First Open NESSIE workshop*, 2000.

[83] L. Knudsen and H. Raddum, "On noekeon. public reports of the nessie project. report: Nes," DOC/UIB/WP3/009/1, Tech. Rep., 2001.

[84] F.-X. Standaert, G. Piret, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "Iceberg: An involutional cipher efficient for block encryption in reconfigurable hardware," in *International Workshop on Fast Software Encryption*. Springer, 2004, pp. 279–298.

[85] H. Cheng and H. M. Heys, "Compact asic implementation of the iceberg block cipher with concurrent error detection," in *2008 IEEE International Symposium on Circuits and Systems*. IEEE, 2008, pp. 2921–2924.

[86] C. Wang and H. M. Heys, "An ultra compact block cipher for serialized architecture implementations," in *2009 Canadian Conference on Electrical and Computer Engineering*. IEEE, 2009, pp. 1085–1090.

[87] H. Cheng, H. M. Heys, and C. Wang, "Puffin: A novel compact block cipher targeted to embedded digital systems," in *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*. IEEE, 2008, pp. 383–390.

[88] M. R. Albrecht, B. Driessen, E. B. Kavun, G. Leander, C. Paar, and T. Yalçın, "Block ciphers–focus on the linear layer (feat. pride)," in *Annual Cryptology Conference*. Springer, 2014, pp. 57–76.

[89] J. Borgho, A. Canteaut, T. Guneysu, E. Kavun, M. Knezevic, L. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger *et al.*, "Prince-a low-latency block cipher for pervasive computing applications-proc. of advances in cryptology," 2012.

[90] L. Batina, A. Das, B. Ege, E. B. Kavun, N. Mentens, C. Paar, I. Verbauwhede, and T. Yalçın, "Dietary recommendations for lightweight block ciphers: power, energy and area analysis of recently developed architectures," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2013, pp. 103–112.

[91] L. Knudsen, G. Leander, A. Poschmann, and M. Robshaw, "Printcipher: A block cipher for ic-printing. cryptographic hardware and embedded systems ches 2010, lncs, vol. 6225/2010," 2010.

[92] Z. Gong, S. Nikova, and Y. W. Law, "Klein: a new family of lightweight block ciphers," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2011, pp. 1–18.

[93] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The led block cipher," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2011, pp. 326–341.

[94] J. Guo, T. Peyrin, and A. Poschmann, "The photon family of lightweight hash functions," in *Annual Cryptology Conference*. Springer, 2011, pp. 222–239.

[95] E. Biham, "New types of cryptanalytic attacks using related keys," *Journal of Cryptology*, vol. 7, no. 4, pp. 229–246, 1994.

[96] G. Piret, T. Roche, and C. Carlet, "Picaro–a block cipher allowing efficient higher-order side-channel resistance," in *International Conference on Applied Cryptography and Network Security*. Springer, 2012, pp. 311–328.

[97] B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert, "Block ciphers that are easier to mask: How far can we go?" in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 383–399.

[98] H. Yap, K. Khoo, A. Poschmann, and M. Henricksen, "Epcbc-a block cipher suitable for electronic product code encryption," in *International Conference on Cryptology and Network Security*. Springer, 2011, pp. 76–97.

[99] M. R. Z'aba, N. Jamil, M. E. Rusli, M. Z. Jamaludin, and A. A. M. Yasir, "I-present tm: An involutive lightweight block cipher," *Journal of Information Security*, vol. 2014, 2014.

[100] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2. submission to the caesar competition," *Institute for Applied Information Processing and Communications, Graz*, 2016.

[101] ——, "Ascon v1. 2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, no. 3, pp. 1–42, 2021.

[102] D. Penazzi and M. Montes, "Shamash (and shamashash)(version 1)," *Lightweight Cryptography Standardization Process round*, vol. 1, 2019.

[103] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, F. Mendel, B. Mennink, N. Mouha, Q. Wang, and K. Yasuda, "Primates v1," *Submission to CAESAR*, 2014.

[104] P. Morawiecki, K. Gaj, E. Homsirikamol, K. Matusiewicz, J. Pieprzyk, M. Rogawski, M. Srebrny, and M. Wójcik, "Icepole: high-speed, hardware-oriented authenticated encryption," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 392–413.

[105] P. l Morawiecki, K. Gaj, E. Homsirikamol, K. Matusiewicz, J. Pieprzyk, M. Rogawski, M. Srebrny, and M. Wójcik, "Icepole v2," *CAESAR submission: http://competitions. cr. yp. to/round2/icepolev2. pdf*, 2015.

[106] A. Poschmann, G. Leander, K. Schramm, and C. Paar, "New light-weight crypto algorithms for rfid," in *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 2007, pp. 1843–1846.

[107] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[108] P. K. Kushwaha, M. Singh, and P. Kumar, "A survey on lightweight block ciphers," *International Journal of Computer Applications*, vol. 96, no. 17, 2014.

[109] M. Appel, A. Bossert, S. Cooper, T. Kußmaul, J. Löffler, C. Pauer, and A. Wiesmaier, "Block ciphers for the iot–simon, speck, katan, led, tea, present, and sea compared," 2016.

[110] B. Andrews, S. Chapman, and S. Dearstyne, "Tiny encryption algorithm (tea) cryptography 4005.705. 01 graduate team acd final report."

[111] P. Israsena and S. Wongnamkum, "Hardware implementation of a tea-based lightweight encryption for rfid security," in *RFID Security*. Springer, 2008, pp. 417–433.

[112] D. Williams, "The tiny encryption algorithm (tea)," *Network Security*, pp. 1–14, 2008.

[113] G. Sekar, N. Mouha, V. Velichkov, and B. Preneel, "Meet-in-the-middle attacks on reduced-round xtea," in *Cryptographers' Track at the RSA Conference*. Springer, 2011, pp. 250–267.

[114] J.-P. Kaps, "Chai-tea, cryptographic hardware implementations of xtea," in *International Conference on Cryptology in India*. Springer, 2008, pp. 363–375.

[115] J. Lu, "Related-key rectangle attack on 36 rounds of the xtea block cipher," *International Journal of Information Security*, vol. 8, no. 1, pp. 1–11, 2009.

[116] D. J. Wheeler and R. M. Needham, "Correction to xtea," *Unpublished manuscript, Computer Laboratory, Cambridge University, England*, 1998.

[117] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-bit block cipher suitable for multiple platforms—design andanalysis," in *International Workshop on Selected Areas in Cryptography*. Springer, 2000, pp. 39–56.

[118] A. Satoh and S. Morioka, "Hardware-focused performance comparison for the standard block ciphers aes, camellia, and triple-des," in *International Conference on Information Security*. Springer, 2003, pp. 252–266.

[119] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers." *IACR Cryptology ePrint Archive*, vol. 2013, no. 1, pp. 404–449, 2013.

[120] F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater, "Sea: A scalable encryption algorithm for small embedded applications," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2006, pp. 222–236.

[121] F. Mace, F.-X. Standaert, J.-J. Quisquater *et al.*, "Asic implementations of the block cipher sea for constrained applications," in *Proceedings of the Third International Conference on RFID Security-RFIDSec*, vol. 2007, 2007, pp. 103–114.

[122] T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni *et al.*, "Compact implementation and performance evaluation of block ciphers in attiny devices," in *International Conference on Cryptology in Africa*. Springer, 2012, pp. 172–187.

[123] C. Rizzo and C. Brookson, "Security for ict-the work of etsi," *ETSI White Paper*, no. 1, 2009.

[124] A. Satoh and S. Morioka, "Small and high-speed hardware architectures for the 3gpp standard cipher kasumi," in *International Conference on Information Security*. Springer, 2002, pp. 48–62.

[125] M. Izadi, B. Sadeghiyan, S. S. Sadeghian, and H. A. Khanooki, "Mibs: a new lightweight block cipher," in *International Conference on Cryptology and Network Security*. Springer, 2009, pp. 334–348.

[126] W. Wu and L. Zhang, "Lblock: a lightweight block cipher," in *International Conference on Applied Cryptography and Network Security*. Springer, 2011, pp. 327–344.

[127] A. Poschmann, S. Ling, and H. Wang, "256 bit standardized crypto for 650 ge–gost revisited," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 219–233.

[128] F. Karakoç, H. Demirci, and A. E. Harmancı, "Itubee: a software oriented lightweight block cipher," in *International Workshop on Lightweight Cryptography for Security and Privacy*. Springer, 2013, pp. 16–27.

[129] M. Kumar, S. K. Pal, and A. Panigrahi, "Few: a lightweight block cipher," *Turkish Journal of Mathematics and Computer Science*, vol. 11, no. 2, pp. 58–73, 2014.

[130] T. Akishita and H. Hiwatari, "Very compact hardware implementations of the blockcipher clefia," in *International Workshop on Selected Areas in Cryptography*. Springer, 2011, pp. 278–292.

[131] C. Tezcan, "The improbable differential attack: Cryptanalysis of reduced round clefia," in *International Conference on Cryptology in India*. Springer, 2010, pp. 197–209.

[132] J. Hosseinzadeh and M. Hosseinzadeh, "A comprehensive survey on evaluation of lightweight symmetric ciphers: hardware and software implementation," *Advances in Computer Science: an International Journal*, vol. 5, no. 4, pp. 31–41, 2016.

[133] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: an ultra-lightweight blockcipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 342–357.

[134] S. K. Ojha, N. Kumar, K. Jain *et al.*, "Twis–a lightweight block cipher," in *International Conference on Information Systems Security*. Springer, 2009, pp. 280–291.

[135] B. Su, W. Wu, L. Zhang, and Y. Li, "Full-round differential attack on twis block cipher," in *International Workshop on Information Security Applications*. Springer, 2010, pp. 234–242.

[136] S. S. M. AlDabbagh, I. F. T. Al Shaikhli, and M. A. Alahmad, "Hisec: A new lightweight block cipher algorithm," in *Proceedings of the 7th International Conference on Security of Information and Networks*, 2014, pp. 151–156.

[137] X. Lai and J. Massey, "A proposal for a new block encryption standard. advantages in cryptology-eurocrypt90 proceedings," 1991.

[138] O. Tigli, "Area efficient asic implementation of idea (international data encryption standard)," *Best design for ASIC implementation of IDEA, GMU*, 2003.

[139] S. Mukherjee and B. Sahoo, "A survey on hardware implementation of idea cryptosystem," *Information Security Journal: A Global Perspective*, vol. 20, no. 4-5, pp. 210–218, 2011.

[140] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong *et al.*, "Hight: A new block cipher suitable for low-resource device," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 46–59.

[141] Y.-I. Lim, J.-H. Lee, Y. You, and K.-R. Cho, "Implementation of hight cryptic circuit for rfid tag," *IEICE Electronics Express*, vol. 6, no. 4, pp. 180–186, 2009.

[142] J. John, "Best-1: a light weight block cipher," *IOSR Journal of Computer Engineering (IOSRJCE)*, vol. 16, no. 2, pp. 91–95, 2014.

[143] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee, "Lea: A 128-bit block cipher for fast encryption on common processors," in *International Workshop on Information Security Applications*. Springer, 2013, pp. 3–27.

[144] D. Lee, D.-C. Kim, D. Kwon, and H. Kim, "Efficient hardware implementation of the lightweight block encryption algorithm lea," *Sensors*, vol. 14, no. 1, pp. 975–994, 2014.

[145] N. Courtois, G. V. Bard, and D. A. Wagner, "Algebraic and slide attacks on keeloq." *IACR Cryptology ePrint Archive*, vol. 2007, pp. 62–62, 2007.

[146] A. Bogdanov, "Linear slide attacks on the keeloq block cipher," in *International Conference on Information Security and Cryptology*. Springer, 2007, pp. 66–80.

[147] C. De Canniere, O. Dunkelman, and M. Knežević, "Katan and ktantan—a family of small and efficient hardware-oriented block ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 272–288.

[148] S. Das, "Halka: A lightweight, software friendly block cipher using ultra-lightweight 8-bit s-box." *IACR Cryptology ePrint Archive*, vol. 2014, p. 110, 2014.

[149] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, "Hummingbird: ultra-lightweight cryptography for resource-constrained devices," in *International Conference on Financial Cryptography and Data Security*. Springer, 2010, pp. 3–18.

[150] M.-J. O. Saarinen, "Cryptanalysis of hummingbird-1," in *International Workshop on Fast Software Encryption*. Springer, 2011, pp. 328–341.

[151] D. Engels, M.-J. O. Saarinen, P. Schweitzer, and E. M. Smith, "The hummingbird-2 lightweight authenticated encryption algorithm," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2011, pp. 19–31.

[152] D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. Le Corre, and L. Perrin, "Felics–fair evaluation of lightweight cryptographic systems," in *NIST Workshop on Lightweight Cryptography*, vol. 128, 2015.

[153] D. Dinu, Y. Le Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the internet of things," *Journal of Cryptographic Engineering*, vol. 9, no. 3, pp. 283–302, 2019.

[154] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," *Journal of CRYPTOLOGY*, vol. 4, no. 1, pp. 3–72, 1991.

[155] M. Matsui, "Linear cryptanalysis method for des cipher," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1993, pp. 386–397.

[156] "Differential fault analysis - wikipedia," https://en.wikipedia.org/wiki/Differential_fault_analysis, (Accessed on 10/31/2020).

[157] J. Breier, X. Hou, and Y. Liu, "Fault attacks made easy: differential fault analysis automation on assembly code," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 96–122, 2018.

[158] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2011, pp. 344–371.

[159] F. Zhang, Y. Zhang, H. Jiang, X. Zhu, S. Bhasin, X. Zhao, Z. Liu, D. Gu, and K. Ren, "Persistent fault attack in practice," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 172–195, 2020.

[160] K. Keerthi, I. Roy, C. Rebeiro, A. Hazra, and S. Bhunia, "Feds: Comprehensive fault attack exploitability detection for software implementations of block ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 272–299, 2020.

[161] S. Bhasin, J. Breier, X. Hou, D. Jap, R. Poussier, and S. M. Sim, "Sitm: See-in-the-middle side-channel assisted middle round differential cryptanalysis on spn block ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 95–122, 2020.

[162] K. Jeong, Y. Lee, J. Sung, and S. Hong, "Improved differential fault analysis on present-80/128," *International Journal of Computer Mathematics*, vol. 90, no. 12, pp. 2553–2563, 2013.

[163] C. Blondeau and K. Nyberg, "Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 165–182.

[164] O. Özen, K. Varıcı, C. Tezcan, and Ç. Kocair, "Lightweight block ciphers revisited: Cryptanalysis of reduced round present and hight," in *Australasian Conference on Information Security and Privacy*. Springer, 2009, pp. 90–107.

[165] M. Renauld and F.-X. Standaert, "Algebraic side-channel attacks," in *International Conference on Information Security and Cryptology*. Springer, 2009, pp. 393–410.

[166] L. Yang, M. Wang, and S. Qiao, "Side channel cube attack on present," in *International Conference on Cryptology and Network Security*. Springer, 2009, pp. 379–391.

[167] B. Zhu, X. Dong, and H. Yu, "Milp-based differential attack on round-reduced gift," in *Cryptographers' Track at the RSA Conference*. Springer, 2019, pp. 372–390.

[168] Y. Sasaki, "Integer linear programming for three-subset meet-in-the-middle attacks: Application to gift," in *International Workshop on Security*. Springer, 2018, pp. 227–243.

[169] M. Cao and W. Zhang, "Related-key differential cryptanalysis of the reduced-round block cipher gift," *IEEE Access*, vol. 7, pp. 175 769–175 778, 2019.

[170] B. Zhao, X. Dong, W. Meier, K. Jia, and G. Wang, "Generalized related-key rectangle attacks on block ciphers with linear key schedule: applications to skinny and gift," *Designs, Codes and Cryptography*, pp. 1–24, 2020.

[171] L. Dalmasso, F. Bruguier, P. Benoit, and L. Torres, "Evaluation of spn-based lightweight crypto-ciphers," *IEEE Access*, vol. 7, pp. 10 559–10 567, 2019.

[172] P. Zhang and W. Zhang, "Differential cryptanalysis on block cipher skinny with milp program," *Security and Communication Networks*, vol. 2018, 2018.

[173] J. Ge, Y. Xu, R. Liu, E. Si, N. Shang, and A. Wang, "Power attack and protected implementation on lightweight block cipher skinny," in *2018 13th Asia Joint Conference on Information Security (AsiaJCIS)*. IEEE, 2018, pp. 69–74.

[174] B. Nallathambi and K. Palanivel, "Fault diagnosis architecture for skinny family of block ciphers," *Microprocessors and Microsystems*, vol. 77, p. 103202, 2020.

[175] J. H. Park, "Security analysis of mcrypton proper to low-cost ubiquitous computing devices and applications," *International Journal of Communication Systems*, vol. 22, no. 8, pp. 959–969, 2009.

[176] Y. Sun, M. Wang, S. Jiang, and Q. Sun, "Differential cryptanalysis of reduced-round iceberg," in *International Conference on Cryptology in Africa*. Springer, 2012, pp. 155–171.

[177] C. Blondeau and B. Gérard, "Differential cryptanalysis of puffin and puffin2," in *ECRYPT Workshop on Lightweight Cryptography*. Citeseer, 2011, p. 1.

[178] G. Zhao, B. Sun, C. Li, and J. Su, "Truncated differential cryptanalysis of prince," *Security and Communication Networks*, vol. 8, no. 16, pp. 2875–2887, 2015.

[179] Y. Lee, K. Jeong, C. Lee, J. Sung, and S. Hong, "Related-key cryptanalysis on the full printcipher suitable for ic-printing," *International Journal of Distributed Sensor Networks*, vol. 10, no. 1, p. 389476, 2014.

[180] Z. Ahmadian, M. Salmasizadeh, and M. R. Aref, "Biclique cryptanalysis of the full-round klein block cipher," *IET Information Security*, vol. 9, no. 5, pp. 294–301, 2015.

[181] J.-P. Aumasson, M. Naya-Plasencia, and M.-J. O. Saarinen, "Practical attack on 8 rounds of the lightweight block cipher klein," in *International Conference on Cryptology in India*. Springer, 2011, pp. 134–145.

[182] M. Gruber and B. Selmke, "Differential fault attacks on klein," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2019, pp. 80–95.

[183] G. Zhao, R. Li, L. Cheng, C. Li, and B. Sun, "Differential fault analysis on led using super-sbox," *IET Information Security*, vol. 9, no. 4, pp. 209–218, 2014.

[184] E. Yarrkov, "Cryptanalysis of xxtea." *IACR Cryptology ePrint Archive*, vol. 2010, p. 254, 2010.

[185] H. Tupsamudre, S. Bisht, and D. Mukhopadhyay, "Differential fault analysis on the families of simon and speck ciphers," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2014, pp. 40–48.

[186] A. Bay, J. Nakahara, and S. Vaudenay, "Cryptanalysis of reduced-round mibs block cipher," in *International Conference on Cryptology and Network Security*. Springer, 2010, pp. 1–19.

[187] Y. Wang, W. Wu, X. Yu, and L. Zhang, "Security on lblock against biclique cryptanalysis," in *International Workshop on Information Security Applications*. Springer, 2012, pp. 1–14.

[188] H. Soleimany, "Self-similarity cryptanalysis of the block cipher itubee," *IET Information Security*, vol. 9, no. 3, pp. 179–184, 2014.

[189] T. Isobe, "A single-key attack on the full gost block cipher," in *International Workshop on Fast Software Encryption*. Springer, 2011, pp. 290–305.

[190] N. T. Courtois, "An improved differential attack on full gost," in *The new codebreakers*. Springer, 2016, pp. 282–303.

[191] S. A. Azimi, Z. Ahmadian, J. Mohajeri, and M. R. Aref, "Impossible differential cryptanalysis of piccolo lightweight block cipher," in *2014 11th International ISC Conference on Information Security and Cryptology*. IEEE, 2014, pp. 89–94.

[192] J. Song, K. Lee, and H. Lee, "Biclique cryptanalysis on lightweight block cipher: Hight and piccolo," *International Journal of Computer Mathematics*, vol. 90, no. 12, pp. 2564–2580, 2013.

[193] J. Huang, S. Vaudenay, and X. Lai, "On the key schedule of lightweight block ciphers," in *International Conference on Cryptology in India*. Springer, 2014, pp. 124–142.

[194] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The simon and speck lightweight block ciphers," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[195] B. Koo, D. Hong, and D. Kwon, "Related-key attack on the full hight," in *International Conference on Information Security and Cryptology*. Springer, 2010, pp. 49–67.

[196] A. Bogdanov, "Attacks on the keeloq block cipher and authentication systems," in *3rd Conference on RFID Security*, vol. 2007. Citeseer, 2007.

[197] B. Zhu and G. Gong, "Multidimensional meet-in-the-middle attack and its applications to katan32/48/64," *Cryptography and Communications*, vol. 6, no. 4, pp. 313–333, 2014.

[198] M.-J. O. Saarinen, "Related-key attacks against full hummingbird-2," in *International Workshop on Fast Software Encryption*. Springer, 2013, pp. 467–482.

[199] *Cryptographic Technology Guideline (Lightweight Cryptography)*, Mar 2017. [Online]. Available: https://www.cryptrec.go.jp/report/cryptrec-gl-2003-2016en.pdf

[200] Z. Hua, J. Li, Y. Chen, and S. Yi, "Design and application of an s-box using complete latin square," *Nonlinear Dynamics*, vol. 104, no. 1, pp. 807–825, 2021.

[201] L. Yi, X. Tong, Z. Wang, M. Zhang, H. Zhu, and J. Liu, "A novel block encryption algorithm based on chaotic s-box for wireless sensor network," *IEEE Access*, vol. 7, pp. 53 079–53 090, 2019.

[202] Z. Hua, Z. Zhu, Y. Chen, and Y. Li, "Color image encryption using orthogonal latin squares and a new 2d chaotic system," *Nonlinear Dynamics*, vol. 104, no. 4, pp. 4505–4522, 2021.

[203]

[204] L. De Meyer and S. Vaudenay, "Des s-box generator," *Cryptologia*, vol. 41, no. 2, pp. 153–171, 2017.

[205] W. Zhang, Z. Bao, V. Rijmen, and M. Liu, "A new classification of 4-bit optimal s-boxes and its application to present, rectangle and spongent," in *International Workshop on Fast Software Encryption*. Springer, 2015, pp. 494–515.

[206] M.-J. O. Saarinen, "Cryptographic analysis of all 4× 4-bit s-boxes," in *International Workshop on Selected Areas in Cryptography*. Springer, 2011, pp. 118–133.

[207] S. Dey and R. Ghosh, "4, 8, 32, 64 bit substitution box generation using irreducible or reducible polynomials over galois field gf (p q) for smart applications," in *Security in Smart Cities: Models, Applications, and Challenges*. Springer, 2019, pp. 279–295.

[208] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede, "Spongent: A lightweight hash function," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2011, pp. 312–325.

[209] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[210] Z. Hua, B. Zhou, and Y. Zhou, "Sine chaotification model for enhancing chaos and its hardware implementation," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 2, pp. 1273–1284, 2018.

[211] B. M. Alshammari, R. Guesmi, T. Guesmi, H. Alsaif, and A. Alzamil, "Implementing a symmetric lightweight cryptosystem in highly constrained iot devices by using a chaotic s-box," *Symmetry*, vol. 13, no. 1, p. 129, 2021.

[212] A. Prathiba and V. Bhaaskaran, "Lightweight s-box architecture for secure internet of things," *Information*, vol. 9, no. 1, p. 13, 2018.

[213] "Verilog," Dec 2021. [Online]. Available: https://en.wikipedia.org/wiki/Verilog

[214] H. Gross, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Suit up!–made-to-measure hardware implementations of ascon," in *2015 Euromicro Conference on Digital System Design*. IEEE, 2015, pp. 645–652.

[215] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2 submission to nist," *NIST Round*, vol. 2, 2019.

[216] D. Šijačić, A. B. Kidmose, B. Yang, S. Banik, B. Bilgin, A. Bogdanov, and I. Verbauwhede, "Hold your breath, primates are lightweight," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 197–216.

[217] S. Picek, L. Mariot, B. Yang, D. Jakobovic, and N. Mentens, "Design of s-boxes defined with cellular automata rules," in *Proceedings of the Computing Frontiers Conference*, 2017, pp. 409–414.

[218] V. K. Wei and K. Yang, "On the generalized hamming weights of product codes," *IEEE transactions on information theory*, vol. 39, no. 5, pp. 1709–1713, 1993.

[219] D. Wagner, "The boomerang attack," in *International Workshop on Fast Software Encryption*. Springer, 1999, pp. 156–170.

[220] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song, "Boomerang connectivity table: a new cryptanalysis tool," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 683–714.

[221] L. Song, X. Qin, and L. Hu, "Boomerang connectivity table revisited. application to skinny and aes," *IACR Transactions on Symmetric Cryptology*, pp. 118–141, 2019.

[222] H. Boukerrou, P. Huynh, V. Lallemand, B. Mandal, and M. Minier, "On the feistel counterpart of the boomerang connectivity table," *IACR Transactions on Symmetric Cryptology*, vol. 2020, no. 1, pp. 331–362, 2020.

[223] H. Feistel, "Cryptography and computer privacy," *Scientific american*, vol. 228, no. 5, pp. 15–23, 1973.

[224] H. Williams, A. Webster, and S. Tavares, "On the design of s-boxes," in *Advances in Cryptology—CRYPTO'85 Proceedings*, vol. 218, 1986, pp. 523–534.

[225] S. Picek, L. Batina, D. Jakobović, B. Ege, and M. Golub, "S-box, SET, Match: A Toolbox for S-box Analysis," in *8th IFIP International Workshop on Information Security Theory and Practice (WISTP)*, ser. Information Security Theory and Practice. Securing the Internet of Things, D. Naccache and D. Sauveron, Eds., vol. LNCS-8501. Heraklion, Crete, Greece: Springer, Jun. 2014, pp. 140–149, part 5: Short Papers. [Online]. Available: https://hal.inria.fr/hal-01400936

Appendix A

AUM: C Code

aum.c

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include "Key_Gen.c"
#include "transpose.c"
#include "addRoundKey.c"
#include "s_Box.c"
#include "feed_input.c"

unsigned int p[32] = { 1 , 0 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0 , 1 , 1 ,
1 , 0 , 0 , 1 , 1 , 1 ,0 , 1 , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 };
unsigned int tk[16] = { 7 , 12 , 9 , 14 , 3 , 5 , 11 , 8 , 2 , 13 , 4 ,
10 , 15 , 0 , 1 , 6 };
unsigned int sBox[2][16] = { { 10 , 3 , 11 , 22 , 17 , 4 , 1 , 8 , 12 , 28 ,
23 , 18 , 26 , 6 , 31 , 20 } , { 15 , 24 , 29 , 13 , 14 , 19 , 30 , 5 , 25 ,
27 , 7 , 0 , 16 , 21 , 2 , 9 } };

unsigned int Org_key[8], Org_key_bits[32], sub_key[16][32], in[32],
trans[32], addKey_out[32], s_out[32], bits_size = 1, size = 0;


void Key_Gen();
```

```
void transpose();
void addRoundKey(int);
void s_Box();
void feed_input();

int main()
{
unsigned int i, round = 1;

printf("\n*********************************");
printf("\n         Key Generation          ");
printf("\n*********************************");
Key_Gen();

for(i = 0; i < 32; i++)
{
in[i] = p[i];
}

while(round<=16)
{
printf("\n\nRound: %d", round);
printf("\n*********************************");
printf("\n         Permutation             ");
printf("\n*********************************");
transpose();

printf("\n*********************************");
printf("\n         addRoundKey             ");
printf("\n*********************************");
addRoundKey(round-1);

printf("\n*********************************");
printf("\n         Substitution            ");
printf("\n*********************************");
s_Box();
```

```
feed_input();

round++;
}

printf("\n\n Input (32-bit):\n");
    for(i = 0; i < 32; i++)
    {
 printf("%d ", p[i]);
    }

    printf("\n\n Cipher (32-bit):\n");
    for(i = 0; i < 32; i++)
    {
 printf("%d ", s_out[i]);
    }

getch();
    return 0;
}
```

Dec_to_Binary.c

```
extern unsigned int s_out[32], bits_size;

void Dec_to_Binary(unsigned int n)
{
    unsigned int i = 0, k, m, five_bit[5];
    unsigned int binaryNum[5] = { 0 , 0 , 0 , 0 , 0 };
    int j, q;

    while (n > 0)
    {
```

```
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }

for (q = 4; q >= 0; q--)
{
m = 0 << q;
k = binaryNum[q] | m;

if (k == 1)
five_bit[q] = 1;
else
five_bit[q] = 0;
}

    for (j = 4; j >= 0; j--)
    {
s_out[bits_size++] = five_bit[j];
}
}
```

Key_Dec_to_Binary.c

```
extern unsigned int Org_key_bits[32], size;

void Key_Dec_to_Binary(unsigned int n)
{
    unsigned int i = 0, k, m, binaryNum[4], four_bit[4];
    int j, q;

    while (n > 0)
    {
        binaryNum[i] = n % 2;
```

```
        n = n / 2;
        i++;
    }

for (q = 3; q >= 0; q--)
{
m = 0 << q;
k = binaryNum[q] | m;
if (k == 1)
four_bit[q] = 1;
else
four_bit[q] = 0;
}

    for (j = 3; j >= 0; j--)
    {
Org_key_bits[size++] = four_bit[j];
//printf(" %d", four_bit[j]);
}
}
```

Key_Gen.c

```
#include "Key_Dec_to_Binary.c"

extern unsigned int Org_key_bits[32], Org_key[8], sub_key[16][32];
extern unsigned int tk[16];
extern unsigned int sBox[2][16];

//Key Generation - Start

void Key_Dec_to_Binary ( unsigned int );

void Key_Gen()
{
```

```
    unsigned int i, j, k, l, dec = 0;
    int minus;

    k = 0;

    for(i = 0; i < 16; i++)
    {
           for(j = 0; j < 2; j++)
           {
                   minus = 16-sBox[j][i];
                   if(minus > 0 && minus < 16)
                   {
                     Org_key[k] = minus;
                     k++;
                   }
                   if(k >= 8)
                   break;
           }
           if(k >= 8)
           break;
    }


   for (i = 0; i < 8; i++)
    {
     printf(" %d ", Org_key[i]);
     Key_Dec_to_Binary(Org_key[i]);
    }

    for (i = 0; i < 32; i++)
{
sub_key[0][i] = Org_key_bits[i];
}

for (k = 0; k < 16; k++)
{
```

```
//if tk[k] = 0, make it to 16 rotation
if(tk[k] == 0)
tk[k] = 16;

 //Rotate the given array by n times toward right
    for (i = 0; i < tk[k]; i++)
{
        int j;
unsigned int last;
        //Stores the last element of the array
        last = sub_key[k][31];

        for (j = 31; j > 0; j--)
{
            //Shift element of array by one
            sub_key[k][j] = sub_key[k][j-1];
        }
        //Last element of array will be added to the start of array.
        sub_key[k][0] = last;
    }
    //temporary coping k subkey to k+1 subkey
    for (i=0; i<32; i++)
{
sub_key[k+1][i] = sub_key[k][i];
}
}
}//Key Generation - End
```

transpose.c

```
extern unsigned int tk[16], trans[32], in[32];

//32-bit Permutation/Transpose - Start
void transpose()
{
```

```
unsigned int i, j;

for (i = 0; i < 16; i++)
{
//if tk[k] = 16, make it to 0 as changed during key generation
if(tk[i] == 16)
tk[i] = 0;
}
j = 0;
for (i = 0; i < 32; i++)
{
if (i < 16)
trans[i] = in[tk[j] * 2 + 0];
else
trans[i] = in[tk[j] * 2 + 1];
j++;

if (j == 16)
   j = 0;
}
}//32-bit Permutation/Transpose - End
```

addRoundKey.c

```
extern unsigned int trans[32], addKey_out[32], sub_key[16][32];

//AddRoundKey - Start

void addRoundKey( int r )
{
unsigned int i;

for (i = 0; i < 32; i++)
    {
```

```
     addKey_out[i] = trans[i]^sub_key[r][i];
}
}
//AddRoundKey - End
```

s_Box.c

```
#include "Dec_to_Binary.c"

extern unsigned int sBox[2][16];
extern unsigned int addKey_out[32], s_out[32], bits_size;


//Substitution (S-Box) - Start
void Dec_to_Binary(unsigned int);

void s_Box()
{
unsigned int i, j, k, n, r, c[4], dec, s_val;
int l;
    s_out[0] = addKey_out[31];
    s_out[31] = addKey_out[0];

    for(i = 1; i < 31; i = i+5)
    {
     n = 1;
     for(j = i+1; j <= i+4; j++)
        {
        c[n] = addKey_out[j];
        n++;
     }

        r = addKey_out[i];
```

```
     dec = 0;
        for (l = 3; l >= 0; l--)
            dec += (int)pow(2, l) * c[3 - l + 1];

        s_val = sBox[r][dec];
        printf("\n S-Box output: %d ", s_val);
        Dec_to_Binary(s_val);
    }

    bits_size = 1;
}//Substitution (S-Box) - End
```

feed_input.c

```
extern unsigned int in[32], s_out[32];

//Feed input to next round

void feed_input()
{
unsigned int i;

for(i = 0; i < 32; i++)
in[i] = s_out[i];
}
```

Appendix B

AUM: Verilog Implementation

input_all.v (Top Module - with subkey gen)

```
‘timescale 1ns / 1ps

module input_all(
input clk,
input rst,
input  [31:0] p,
input [79:0] tk,
input  [31:0] orgKey,
input readIn,
output [31:0] out
    );
wire [5:0] keyNumber;
wire [31:0] subKey;
wire [31:0] subkey;
wire rotationDone ;
wire rotationDone1;
wire rotationDone2;
wire [31:0] trans;
reg [31:0] trans1;
wire [31:0]transExor;


transpose trans_p (.p(p),.subkey(subkey),.rotationDone2(rotationDone2),
```

```
.trans(trans));

sBox sBox_1(.clk(clk),.rst(rst),.rotationDone(rotationDone),
.transExor(transExor),.subkey(subkey));

addround addround_1 (.trans1(trans1),.subKey(subKey),.transExor(transExor));

keygen keygen_1(.clk(clk),.rst(rst),.orgKey(orgKey),.tk(tk),
.readIn(readIn),.keyNumber(keyNumber),.rotationDone(rotationDone),.subKey(subKey));

controlunit cu_1(.clk(clk),.rst(rst),.readIn(readIn),.rotationDone(rotationDone),
.keyNumber(keyNumber),.rotationDone1(rotationDone1),.rotationDone2(rotationDone2));

always @(posedge rotationDone)begin
    trans1<=trans;
end
assign out=rotationDone?subkey:out;

endmodule
```

input_all_2.v (Top Module - without subkey gen)

```
`timescale 1ns / 1ps

module input_all_2(
input clk,
input rst,
input  [31:0] p,
input [79:0] tk,
input  [31:0] orgKey,
input readIn,
output [31:0] out
    );

wire [31:0] subKey;
```

```
wire [31:0] subkey;
wire rotationDone ;
wire rotationDone1;
wire rotationDone2;
wire [31:0] trans;
reg [31:0] trans1;
wire [31:0]transExor;
wire [5:0] keyNumber;
//wire [4:0]a1,a2,a3,a4,a5,a6;

tranpose trans_p(.p(p),.subkey(subkey),.rotationDone2(rotationDone2),
.trans(trans));

sBox sBox_1(.clk(clk),.rst(rst),.rotationDone(rotationDone),
.transExor(transExor),.subkey(subkey));

addround ad1(.trans1(trans1), .subKey(subKey), .transExor(transExor));

keySelect key_1 (.clk(clk),.rst(rst),.rotationDone(rotationDone),
.readIn(readIn),.subKey(subKey),.tk(tk),.keyNumber(keyNumber));

controlunit cu_1(.clk(clk),.rst(rst),.readIn(readIn),
.rotationDone(rotationDone),.rotationDone1(rotationDone1),
.rotationDone2(rotationDone2),.keyNumber(keyNumber));

always @(posedge rotationDone)begin
    trans1<=trans;
end

assign out=rotationDone?subkey:out;

endmodule
```

controlunit.v - (with subkey gen)

```
`timescale 1ns / 1ps

module controlunit(clk,rst,readIn,rotationDone,keyNumber,rotationDone1,
rotationDone2);
input clk,rst,readIn;
output reg rotationDone;
output reg rotationDone1;

output reg rotationDone2;

output reg [5:0]keyNumber;
always @(posedge rst, posedge clk) begin
    if (rst == 1'b1) begin
        rotationDone <= 1'b0;
    end
    else begin
        if ( keyNumber == 16 ) begin
            rotationDone <= 1'b1;
        end
        else if (readIn == 1'b1 )begin

            rotationDone <= ~rotationDone;
        end

    end
end
always @(posedge rst, posedge clk) begin
    if (rst == 1'b1) begin
        keyNumber <= 6'd0;
    end
    else begin
    if(readIn == 1'b1 && rotationDone == 1'b0) begin
       if (keyNumber <= 15) begin
        keyNumber <= keyNumber +1;
       end
```

```
    end
    end
end
always @(posedge clk)begin
    rotationDone1<=rotationDone;
end

always @(posedge clk,posedge rst) begin
        if (rst == 1'b1) begin
            rotationDone2<= 1'b0;
        end
       else begin
            if (rotationDone1==1'b1)begin
                rotationDone2<=rotationDone1;
            end
        end
end

endmodule
```

controlunit.v - (without subkey gen)

```
`timescale 1ns / 1ps
module controlunit(clk,rst,readIn,rotationDone,keyNumber,rotationDone1,
rotationDone2);
input clk,rst,readIn;
output reg rotationDone;
output reg rotationDone1;

output reg rotationDone2;

output reg [5:0]keyNumber;
always @(posedge rst, posedge clk) begin
    if (rst == 1'b1) begin
        rotationDone <= 1'b0;
```

```
    end
    else begin
        if ( keyNumber == 16 ) begin
            rotationDone <= 1'b1;
        end
        else if (readIn == 1'b1 )begin

            rotationDone <= ~rotationDone;
        end

    end
end
always @(posedge rst, posedge clk) begin
    if (rst == 1'b1) begin
        keyNumber <= 6'd0;
    end
    else begin
    if(readIn == 1'b1 && rotationDone == 1'b0) begin
       if (keyNumber <= 15) begin
        keyNumber <= keyNumber +1;
       end
    end
    end
end
always @(posedge clk)begin
    rotationDone1<=rotationDone;
end

always @(posedge clk,posedge rst) begin
        if (rst == 1'b1) begin
            rotationDone2<= 1'b0;
        end
       else begin
            if (rotationDone1==1'b1)begin
                rotationDone2<=rotationDone1;
            end
```

```
        end
end

endmodule
```

keygen.v

```
`timescale 1ns / 1ps

module keygen(clk,rst,orgKey,tk,readIn,subKey,keyNumber,rotationDone);
input clk,rst;
input [31:0] orgKey;
input [79:0]tk;
input readIn;
input [5:0] keyNumber;
output reg [31:0] subKey;
input rotationDone;

//reg [4:0] keyNumber;

wire [4:0]tkReg[15:0];
assign tkReg[0]=tk[4:0];
assign tkReg[1]=tk[9:5];
assign tkReg[2]=tk[14:10];
assign tkReg[3]=tk[19:15];
assign tkReg[4]=tk[24:20];
assign tkReg[5]=tk[29:25];
assign tkReg[6]=tk[34:30];
assign tkReg[7]=tk[39:35];
assign tkReg[8]=tk[44:40];
assign tkReg[9]=tk[49:45];
assign tkReg[10]=tk[54:50];
assign tkReg[11]=tk[59:55];
assign tkReg[12]=tk[64:60];
assign tkReg[13]=tk[69:65];
```

```
assign tkReg[14]=tk[74:70];
assign tkReg[15]=tk[79:75];
always @(posedge clk, posedge rst) begin
    if(rst == 1'b1) begin
       // keyNumber <= 5'd0;
       // rotationDone <= 1'b0;
        subKey <= orgKey;

    end
 /*   else if(rotationDone == 1'b1 && keyNumber <= 15) begin
        rotationDone <= 1'b0;
    end*/
    else begin
        if(readIn == 1'b1 && rotationDone == 1'b0) begin
               // rotationDone<=1'b1;
                case(tkReg[keyNumber])
                    5'd1:begin subKey<={subKey[30:0], subKey[31]}; end
                    5'd2:begin subKey<={subKey[29:0], subKey[31:30]}; end
                    5'd3:begin subKey<={subKey[28:0], subKey[31:29]}; end
                    5'd4:begin subKey<={subKey[27:0], subKey[31:28]}; end
                    5'd5:begin subKey<={subKey[26:0], subKey[31:27]}; end
                    5'd6:begin subKey<={subKey[25:0], subKey[31:26]}; end
                    5'd7:begin subKey<={subKey[24:0], subKey[31:25]}; end
                    5'd8:begin subKey<={subKey[23:0], subKey[31:24]}; end
                    5'd9:begin subKey<={subKey[22:0], subKey[31:23]}; end
                    5'd10:begin subKey<={subKey[21:0], subKey[31:22]}; end
                    5'd11:begin subKey<={subKey[20:0], subKey[31:21]}; end
                    5'd12:begin subKey<={subKey[19:0], subKey[31:20]}; end
                    5'd13:begin subKey<={subKey[18:0], subKey[31:19]}; end
                    5'd14:begin subKey<={subKey[17:0], subKey[31:18]}; end
                    5'd15:begin subKey<={subKey[16:0], subKey[31:17]}; end
                    //5'd16:begin subKey<={subKey[15:0], subKey[31:16]}; end
                    default:begin subKey<={subKey[15:0], subKey[31:16]}; end
                   /* 5'd7:begin subKey<=32'd2714459411; end
                    5'd12:begin subKey<=32'd3050387996; end
                    5'd9:begin subKey<=32'd2725525867; end
```

```
                    5'd14:begin subKey<=32'd240838813; end
                    5'd3:begin subKey<=32'd1926710504; end
                    5'd5:begin subKey<=32'd1525193998; end
                    5'd11:begin subKey<=32'd1156084439; end
                    5'd8:begin subKey<=32'd3899840324; end
                    5'd2:begin subKey<=32'd2714459411; end
                    5'd13:begin subKey<=32'd1805808697; end
                    5'd4:begin subKey<=32'd3123135382; end
                    5'd10:begin subKey<=32'd2634963688; end
                    5'd15:begin subKey<=32'd762596999; end
                    5'd16:begin subKey<=32'd1317481844; end
                    5'd1:begin subKey<=32'd2634963688; end
                    5'd6:begin subKey<=32'd1133951527; end*/
                endcase
               /* if (keyNumber <= 15) begin
                    keyNumber <= keyNumber +1;
                    rotationDone <= 1'b1;
                end*/
        end
    end
end
endmodule
```

transpose.v

```
`timescale 1ns / 1ps
module tranpose(p,subkey,rotationDone2, trans);
input [31:0] subkey, p;
input rotationDone2;
output [31:0] trans;
assign trans=rotationDone2?{subkey[13],subkey[3],subkey[1],subkey[31],
                            subkey[21],subkey[9],subkey[27],subkey[5],
                            subkey[17],subkey[23],subkey[11],subkey[7],
                            subkey[29],subkey[19],subkey[25],subkey[15],
                            subkey[12],subkey[2],subkey[0],subkey[30],
```

```
                                subkey[20],subkey[8],subkey[26],subkey[4],
                                subkey[16],subkey[22],subkey[10],subkey[6],
                                subkey[28],subkey[18],subkey[24],subkey[14]}:
                               {p[13],p[3],p[1],p[31],
                                p[21],p[9],p[27],p[5],
                                p[17],p[23],p[11],p[7],
                                p[29],p[19],p[25],p[15],
                                p[12],p[2],p[0],p[30],
                                p[20],p[8],p[26],p[4],
                                p[16],p[22],p[10],p[6],
                                p[28],p[18],p[24],p[14]};
endmodule
```

keyselect.v

```
`timescale 1ns / 1ps

module keySelect(clk,rst,rotationDone,readIn,subKey,tk,keyNumber); //
input clk,rst,readIn;
input [79:0] tk;
input rotationDone;
//output reg rotationDone;
output reg [31:0] subKey;
input  [5:0] keyNumber;
wire [4:0]tkReg[15:0];
assign tkReg[0]=tk[4:0];
assign tkReg[1]=tk[9:5];
assign tkReg[2]=tk[14:10];
assign tkReg[3]=tk[19:15];
assign tkReg[4]=tk[24:20];
assign tkReg[5]=tk[29:25];
assign tkReg[6]=tk[34:30];
assign tkReg[7]=tk[39:35];
assign tkReg[8]=tk[44:40];
assign tkReg[9]=tk[49:45];
```

```
assign tkReg[10]=tk[54:50];
assign tkReg[11]=tk[59:55];
assign tkReg[12]=tk[64:60];
assign tkReg[13]=tk[69:65];
assign tkReg[14]=tk[74:70];
assign tkReg[15]=tk[79:75];
always @(posedge clk, posedge rst) begin
    if(rst == 1'b1) begin
  //      keyNumber <= 5'd0;
//        rotationDone <= 1'b0;
        subKey <= 32'd0;

    end
/*    else if(rotationDone == 1'b1 && keyNumber <= 15) begin
        rotationDone <= 1'b0;
    end*/
    else begin
        if(readIn == 1'b1 && rotationDone == 1'b0) begin
//                rotationDone<=1'b1;
                case(tkReg[keyNumber])
                    5'd7:begin subKey<=32'd2714459411; end
                    5'd12:begin subKey<=32'd3050387996; end
                    5'd9:begin subKey<=32'd2725525867; end
                    5'd14:begin subKey<=32'd240838813; end
                    5'd3:begin subKey<=32'd1926710504; end
                    5'd5:begin subKey<=32'd1525193998; end
                    5'd11:begin subKey<=32'd1156084439; end
                    5'd8:begin subKey<=32'd3899840324; end
                    5'd2:begin subKey<=32'd2714459411; end
                    5'd13:begin subKey<=32'd1805808697; end
                    5'd4:begin subKey<=32'd3123135382; end
                    5'd10:begin subKey<=32'd2634963688; end
                    5'd15:begin subKey<=32'd762596999; end
                    5'd16:begin subKey<=32'd1317481844; end
                    5'd1:begin subKey<=32'd2634963688; end
                    5'd6:begin subKey<=32'd1133951527; end
```

```
                endcase
               /* if (keyNumber <= 15) begin
                    keyNumber <= keyNumber +1;
//                   rotationDone <= 1'b1;
                end*/
        end
    end
end
endmodule
```

addRound.v

```
`timescale 1ns / 1ps
module addround(trans1, subKey, transExor);
input [31:0] trans1, subKey;
output [31:0] transExor;
assign transExor =  (trans1 ^ subKey) ;
endmodule
```

sBox.v

```
`timescale 1ns / 1ps
module sBox(clk,rst,rotationDone,transExor,subkey );
input clk,rst,rotationDone;
input [31:0] transExor;
output [31:0] subkey;
wire [4:0] a1,a2,a3,a4,a5,a6;

case_st m1(.clk(clk),.rst(rst),.exorDone(rotationDone),
.in(transExor[30:26]),.out(a1));

case_st m2(.clk(clk),.rst(rst),.exorDone(rotationDone),
.in(transExor[25:21]),.out(a2));
```

```
case_st m3(.clk(clk),.rst(rst),.exorDone(rotationDone),
.in(transExor[20:16]),.out(a3));

case_st m4(.clk(clk),.rst(rst),.exorDone(rotationDone),
.in(transExor[15:11]),.out(a4));

case_st m5(.clk(clk),.rst(rst),.exorDone(rotationDone),
.in(transExor[10:6]),.out(a5));

case_st m6(.clk(clk),.rst(rst),.exorDone(rotationDone),
.in(transExor[5:1]),.out(a6));

assign subkey=
rotationDone?{transExor[0],a1,a2,a3,a4,a5,a6,transExor[31]}:subkey;

endmodule

module case_st(
input clk,rst,exorDone,
input [4:0]in,
output  [4:0]out);
wire [4:0] inSet;
reg [4:0] out1;
assign inSet ={in[0],in[1],in[2],in[3],in[4]};
assign out = {out1[0],out1[1],out1[2],out1[3],out1[4]};
always @(rst,in,inSet)begin
    if (rst==1'b1)begin
     out1 <= 5'd0;
    end
    else begin
            case(inSet)
                5'd0:begin out1<=5'd10; end
                5'd1:begin out1<=5'd3; end
                5'd2:begin out1<=5'd11; end
                5'd3:begin out1<=5'd22; end
                5'd4:begin out1<=5'd17; end
```

```
                5'd5:begin out1<=5'd4; end
                5'd6:begin out1<=5'd1; end
                5'd7:begin out1<=5'd8; end
                5'd8:begin out1<=5'd12; end
                5'd9:begin out1<=5'd28; end
                5'd10:begin out1<=5'd23; end
                5'd11:begin out1<=5'd18; end
                5'd12:begin out1<=5'd26; end
                5'd13:begin out1<=5'd6; end
                5'd14:begin out1<=5'd31; end
                5'd15:begin out1<=5'd20; end
                5'd16:begin out1<=5'd15; end
                5'd17:begin out1<=5'd24; end
                5'd18:begin out1<=5'd29; end
                5'd19:begin out1<=5'd13; end
                5'd20:begin out1<=5'd14; end
                5'd21:begin out1<=5'd19; end
                5'd22:begin out1<=5'd30; end
                5'd23:begin out1<=5'd5; end
                5'd24:begin out1<=5'd25; end
                5'd25:begin out1<=5'd27; end
                5'd26:begin out1<=5'd7; end
                5'd27:begin out1<=5'd0; end
                5'd28:begin out1<=5'd16; end
                5'd29:begin out1<=5'd21; end
                5'd30:begin out1<=5'd2; end
                5'd31:begin out1<=5'd9; end
                default: out1<=5'd0;
            endcase
    end
end
endmodule
```