EFFICIENT IMPLEMENTATION OF SIMPLIFIED AES AND ITS SIDE-CHANNEL ANALYSIS

BASİTLEŞTİRİLMİŞ İLERİ ŞİFRELEME STANDARDI'NIN VERİMLİ UYGULAMASI VE YAN KANAL ANALİZİ

BARIŞ BERK ZORBA

PROF. DR. ALİ ZİYA ALKAR

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Electrical and Electronics Engineering.

2022

ABSTRACT

EFFICIENT IMPLEMENTATION OF SIMPLIFIED AES

AND ITS SIDE-CHANNEL ANALYSIS

Barış Berk ZORBA

Master of Science, Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Ali Ziya ALKAR

January 2022, 58 pages

Lightweight cryptography is gaining momentum as the number of connected Internet of Things (IoT) devices in the world is expected to reach beyond 50 billion by 2030. With all that massive amount of connectivity, the demand for communication security will be higher as well as the edge processing speed. In this thesis, the Simplified AES (S-AES) algorithm, a popular, fast and feasible solution for the security of embedded devices concerning other ultra-lightweight block ciphers is studied. S-AES has been considered as one of the better options in terms of memory usage in microcontrollers, where memory is already in high demand. In this thesis, the S-AES encryption performance was improved by simple, yet efficient techniques applied on the MixColumns layer. STM32F4-DISCOVERY board having an ARM Cortex-M4 embedded processor is used to demonstrate these techniques while relatively comparing them concerning each other to show the performance improvements. Additionally, Altera Cyclone-IV and Xilinx

BASYS-3 FPGA demo boards are used to demonstrate the high performance and low-area implementation of this algorithm compared to other lightweight block ciphers. Side-channel security of the implementation techniques was analyzed on STM32F4-DISCOVERY board to prove the implementation is still secure even though the performance is improved. As a conclusion, the embedded software and hardware implementations of S-AES showed that this encryption algorithm can be used in performance-critical areas such as Internet of Things (IoT).

Keywords: Simplified AES, Block Cipher, Embedded Software Implementation, FPGA Implementation, Ultra-Lightweight Cryptography

ÖZET

BASİTLEŞTİRİLMİŞ İLERİ ŞİFRELEME STANDARDI'NIN VERİMLİ UYGULAMASI VE YAN-KANAL ANALİZİ

Barış Berk ZORBA

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği

Tez Danışmanı: Prof. Dr. Ali Ziya ALKAR

Ocak 2022, 58 sayfa

Hafif Blok Tipi Şifreleme yöntemleri, Nesnelerin İnterneti cihazlarının sayısının 2030 yılına kadar 50 milyarı aşması beklendiğinden ötürü ivme kazanmaktadır. Bütün bu büyük miktarda bağlantı ile, iletişim güvenliğine olan talep ve uçtan uca işlem hızındaki beklenti daha yüksek olacaktır. Bu tezde, diğer ultra hafif blok tipi şifrelemelere göre gömülü cihazların güvenliği için popüler, hızlı ve uygulanabilir bir çözüm olan Basitleştirilmiş AES (S-AES) algoritması incelenmiştir. Bu tezde S-AES, belleğin zaten yüksek talep gördüğü mikrodenetleyicilerde bellek kullanımı açısından iyi bir seçenek olarak gösterilmiştir. Bununla birlikte, S-AES şifreleme performansı, MixColumns katmanında uygulanan basit ama etkili tekniklerle geliştirilebilmiştir. Bu teknikleri doğrulamak için ARM Cortex-M4 gömülü işlemcisine sahip bir STM32F4-DISCOVERY bordu kullanılırken, performans iyileştirmelerini göstermek için bu katmanda uygulanan teknikler birbirleriyle karşılaştırılmıştır. Ek olarak, diğer Hafif Blok Şifrelere kıyasla bu algoritmanın donanımda da yüksek performansta çalıştığını ve düşük

alan kapladığını göstermek için Altera Cyclone-IV ve Xilinx BASYS-3 demo kartlarında gerçeklenmiştir. Uygulamalarda performans iyileştirmesi olsa bile uygulamanın hala güvenli olduğunu kanıtlamak için STM32F4-DISCOVERY bordu üzerinde çığ etkisi ve yan kanal analizi incelenmiştir. Sonuç olarak, bu tez kapsamında S-AES'in gömülü yazılım ve donanım uygulamaları, bu şifreleme algoritmasının Nesnelerin İnterneti (IoT) gibi performans açısından kritik alanlarda kullanılabileceği gösterilmiştir.

Anahtar Kelimeler: Basitleştirilmiş İleri Şifreleme Standardı, Blok Tipi Şifreleme, Gömülü Yazılım Gerçeklemesi, FPGA Gerçeklemesi, Ultra-Hafif Kriptografi

THANKS

I would like to extend my sincere thanks to:

My esteemed professors, Prof. Dr. Ali Ziya ALKAR and Assoc. Prof. Murat AYDOS, who guided my life with their vast knowledge and experience throughout my graduate education,

Saim Buğrahan ÖZTÜRK and Erkan USLU for sharing their valuable ideas with me while I was writing this thesis,

My father, my mother, my brother, and my wife who always supported me and showed patience.

Barış Berk ZORBA

January 2022, Ankara

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

Symbols

GF(2n)	Finite Field Extensions
Sn,n	Nibble Representation of a Hex
wn,n	Byte Representation of a Hex
$\oplus$	XOR Operation

Abbreviations

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
CLB	Configurable Logic Block
CPA	Correlation Power Analysis
CPU	Central Processing Unit
CYCCNT	Clock Cycle Counter
DES	Data Encryption Standard
DPA	Differential Power Analysis
DWT	Data Watchpoint and Trace Unit
FF	Flip-Flop
FPGA	Field Programmable Gate Array
GB	GigaBytes
Gbps	Gigabits per Second
GCC	GNU Compiler Collection
GF	Galois Field
HEX	Hexadecimal

IDE	Integrated Development Environment
IoT	Internet of Things
ISA	Instruction Set Architecture
KB	KiloBytes
LE	Logic Element
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look-up Table
MB	MegaBytes
Mbps	Megabits per Second
MCU	Microcontroller Unit
MDS	Maximum Distance Separable
MHZ	MegaHertz
ms	Milliseconds
μs	Microseconds
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology
ns	Nanoseconds
P2P	Peer to Peer
RAM	Random Access Memory
RCON	Round Constant
RotNibble	Rotate Nibble
RTL	Register-Transfer Level
S-AES	Simplified Advanced Encryption Standard
SCA	Side-Channel Attacks
SPN	Substitution–Permutation Network

us	Microseconds
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit
WNS	Worst Negative Slack
WSN	Wireless Sensor Network

1. INTRODUCTION

1.1. Introduction to Applied Cryptography

There have been important developments in the field of information security in recent years. As the technology constantly develops, security and performance requirements in communication systems increase too. Encryption has a great role in latest communication devices for keeping the information private and authenticated. New attacking techniques are being applied by cryptanalysts each and every day. While the purpose is not only to "break the code" but also to find many ways to pass the security barrier. Cryptographers search for various ways to develop a secure and fast cryptosystem, where these encryption algorithms rely on hard mathematical problems. When the mathematical structure of the algorithm is built this way, it is nearly impossible to solve the plaintext or the key from ciphertext using reverse engineering.

Cryptology is the science concerned with the security of data in communication systems, which consists of cryptography and cryptanalysis. Moreover, Public-Key Cryptography (PKC) and Symmetric-Key Cryptography are branched under the cryptography subject. Even though PKC techniques provide confidentiality using public & private key pairs and authentication using digital signatures, symmetric-key cryptography is considered as the main subject to provide the same level of confidentiality with lower key size [1].

Symmetric-key encryption algorithms are divided into two: stream ciphers and block ciphers, where block ciphers are the main concentration of this thesis. The data is encrypted bit-by-bit in stream ciphers, leading the usage in Linear Feedback Shift Registers (LFSR). The blocks are encrypted as a whole in these encryption algorithms and then integrated using modes of operation techniques. Block ciphers are used in many data communication environments and specific applications such as deterministic pseudo-random number generation. Data Encryption Standard (DES) was used as the national encryption standard by National Institute of Standards and Technology (NIST) until it was broken with differential cryptanalysis technique [2].

After DES is broken, there was a need for a secure and fast block cipher, which led to a competition started by the NIST. Rijndael was chosen as the most suitable algorithm for the Advanced Encryption Standard (AES) competition by NIST due to its secure, efficient, fast and feasible structure [3]. This block cipher has been used in different kinds of applications ever since it was first introduced. Even though there has been plenty of attacks applied on it, AES is still considered as secure considering algebraic security according to NIST.

The most common attacks applied on block ciphers are differential and linear cryptanalyses which are applied on the mathematical structure of the algorithm. Physical attacks such as side-channel attacks and their countermeasures are some of the greatest concerns of Cryptographic Engineering. Side-Channel Analysis (SCA) mainly consists of timing attacks, power analysis, electromagnetic analysis, and cache attacks.

AES-256 is still utilized in security-critical applications of 5G communications [4], whereas AES-128 is used as a lightweight block cipher for Internet of Things (IoT) applications [5]. Even though AES is a comparably fast symmetric-key encryption algorithm, most of the block ciphers are open to performance upgrades in both software and hardware since faster instructions are developed in newer microarchitectures and the demand on faster encryption algorithms increases as the nanometer technology grows. Since the block ciphers are classified according to their structures, a performance upgrade in a cipher might be used to speed up another that uses the same structure.

1.2. Motivation of the Thesis

Even though the fast peer to peer (P2P) transportation of data is a need, it is a matter of fact that security plays an important role in latest communication systems. To maintain security in IoT applications, integrity, authentication, and confidentiality must be provided by the channel. While designing secure cryptosystems, preserving the performance of the communication system is important in IoT. Performance evaluations

of encryption algorithms in the IoT applications are organized mainly based on memory, timing and energy requirements [6], where the power consumption, computational performance and utilized memory are some of the most important resource constraints. Lightweight block ciphers are utilized in this process to fulfill the need for a fast and reliably secure algorithm in IoT.

Even though modifying the key size, number of rounds or block size might be useful for increasing the performance of a block cipher, it is not a great solution since the algebraic security of the algorithm decreases. To standardize performance throughputs of block ciphers, alternative structures or implementation techniques should be provided for performance-critical applications.

The weakness of some block ciphers such as AES in SCA attacks revealed that implementation methods resistant to SCA should be developed. It is a fact that there is a need for a performance acceleration in the MixColumns layer of AES-like algorithms, where Finite Field multiplications are utilized. At this point, it is feasible to apply the performance upgrade on Simplified Advanced Encryption Standard (S-AES) algorithm which is an AES-like algorithm that can be used to demonstrate the performance upgrade on MixColumns layer. To do so, it is aimed to accelerate this layer using proposed methods and fast implementation methods that already exist in the literature. Besides, proposed method should provide durability to some SCA attacks for security improvements. Increase in the throughput and total clock cycles should be verified using various hardware and software platforms. The rationale of this research depends on the weakness of block ciphers in embedded software implementation performances and excessive area consumption in hardware.

Purpose of the study is to come up with a solution to solve the need for a high-performance encryption algorithm in IoT applications. Moreover, the throughput of S-AES was increased in this thesis study, while presenting the mathematical analysis, comparing the implementations on target platforms and adapt to application domains such as IoT or Ultra-Lightweight Cryptography. It is aimed to implement S-AES as a low-area

implementation on hardware, while the purpose in software is to apply Switch-Case method as a fast and low memory consuming method in the MixColumns layer. It is desired to prove that the method applied by comparing it with other efficient implementation methods is successful in terms of performance and security parameters, which are the building blocks of applied cryptography studies.

1.3. Thesis Work

S-AES, an AES-like symmetric-key encryption algorithm with lower rounds, block size and key size, can been considered as one of the best options for high-performance applications in Embedded Systems or IoT applications [7] and can be adapted to Ultra-Lightweight Cryptography.

MixColumns is the layer of S-AES where the diffusion occurs by multiplying the bits of plaintext. The mathematical structure of this layer is not modified in this research, though its implementation is modified. The implementations on STM32, Altera Cyclone-IV and Xilinx BASYS-3 demo boards are given as proof of concepts. Pre-calculated results of the Galois Field (GF) multiplications in 24 are written into switch-case statements and are called as a function. Another applicable model that relies on Look-up Table (LUT) is also implemented. Lastly, the commonly known method of implementing MixColumns layer is implemented where Maximum Distance Separable (MDS) matrix is used. Then, the results of these three implementation methods are compared with each other. It is also necessary not to waste resources in Embedded software implementations, while ensuring high-performance encryption. Throughput and memory utilization are examined in Atollic TrueSTUDIO for STM32 implementation, in order to scale the encrypt time and total area that S-AES covers respectively.

It was stated by NIST that LUT implementations are robust against timing attacks [8]. SCA methods were applied on all of these three implementations and results are compared between each other in the MixColumns stage and whole encryption process. To prove the security needs of a block cipher, the Avalanche Effect of S-AES is analyzed, and an average result is obtained compared to some other block ciphers.

This thesis consists of the background in Section 2 that includes a background on symmetric-key cryptography, block ciphers, Advanced Encryption Standard, the structure of S-AES, performance vs. security and related works, whereas its software, embedded software and hardware implementations are explained in third section with security analyses including Avalanche Effect and SCA resistance. The results and comparison of this thesis work with other implementations are revealed in Section 4 as Results and Discussion.

2. BACKGROUND WORK

2.1. Symmetric-Key Cryptography

Symmetric-key cryptography is one of the most significant fields of mathematics and engineering. It is classified under cryptography and differs from PKC with the number of keys that are used during the encryption process. Symmetric-key encryption algorithms are most frequently used in data communications to hide the data from intruders using a single pre-shared key.

Symmetric-key cryptography is divided into two domains as block ciphers and stream ciphers according to the data size of the encryption algorithm [9]. While the plaintext enters the cryptosystem as bits in Stream Ciphers, it is encrypted as blocks in block ciphers.

2.2. Block Ciphers

Operation modes in block ciphers are used for plaintext to be given as multiple blocks to the input of the algorithm. While the mathematical design is not considered as the main concentration of this thesis, the performance-security trade-off is examined the most. The classification of block ciphers with respect to their structures are given in five categories as shown below [10].

- Iterated block ciphers
- Substitution–permutation networks
- Feistel ciphers
- Lai–Massey ciphers
- Add-Rotate-XOR (ARX)

One of the most utilized Balanced Feistel network algorithms, DES, is used in the form of two equally divided blocks during the encryption [11]. DES is given as a standard example to demonstrate the structure of a Feistel cipher as shown in Figure 2.1.

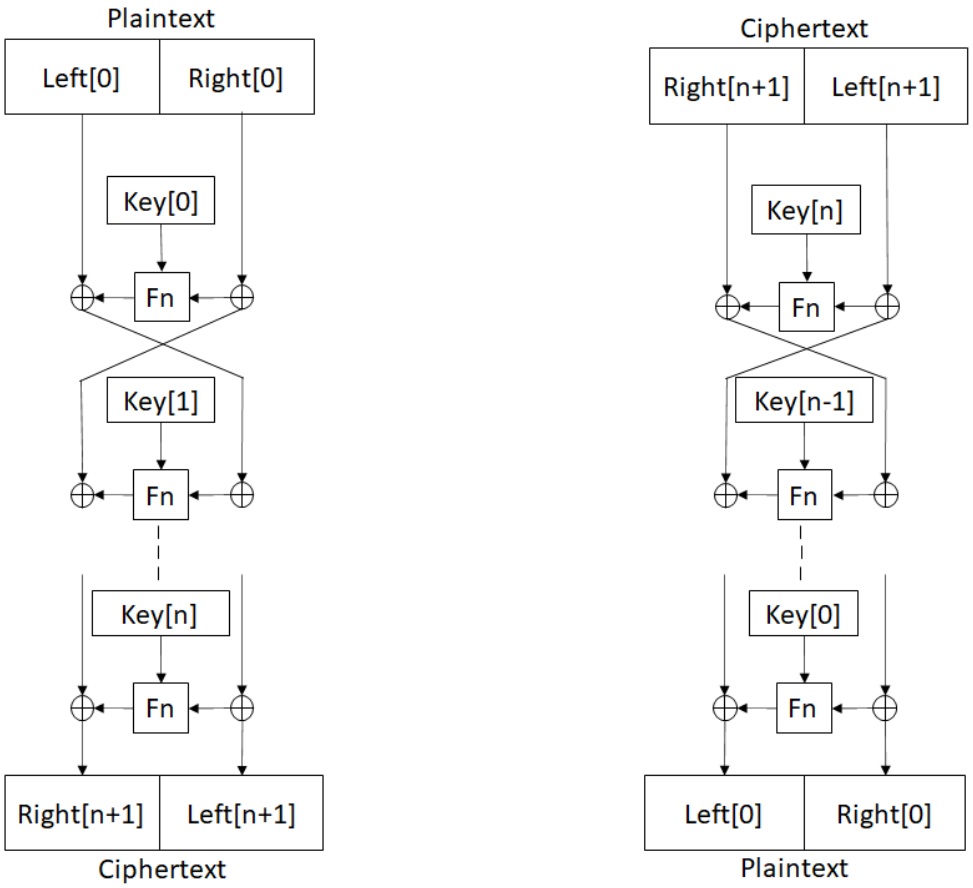


Figure 2.1 - Encryption and Decryption of a Feistel Structured Block Cipher [12]

The data is encrypted using substitution and permutation operations in SPN structure. Substitution and Permutation layers can also be considered as confusion and diffusion respectively according to their mathematical structures. A general SPN structure of a block cipher is demonstrated in Figure 2.2, in which the plaintext is XORed with the round key and divided into blocks for substitution in S-boxes and permutation in P-layers. The finalists of the AES competition, Blowfish and Rijndael (also known as AES) can be categorized as remarkable block ciphers with SPN structure [13].

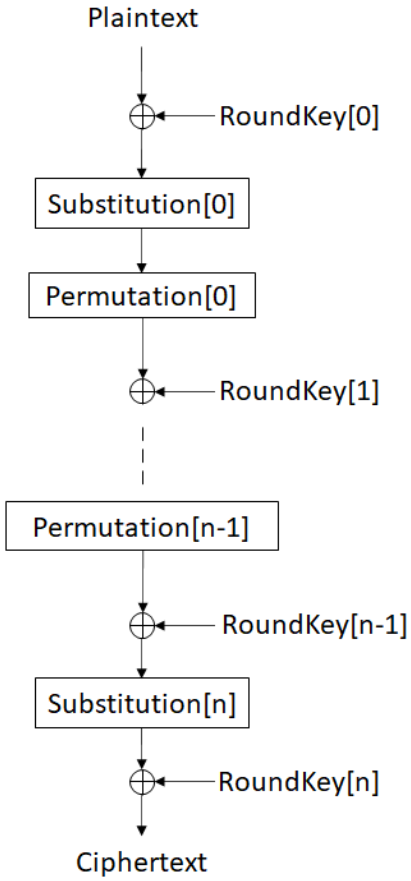


Figure 2.2 – Encryption of an SPN Structured Block Cipher [14]

2.2.1. Modes of Operation

When the input is longer than the defined block length, there are operation modes to encrypt the message while using a single key [15]. Initialization Vector (IV) is the randomly generated number used to start the encryption block and hence to produce distinct ciphertext. The most common Modes of Operations used in block ciphers are described below.

Electronic Codebook Mode (ECB): Each block is encrypted independently and substituted with a value similar to codebook. Due to the identical plaintexts being encrypted similarly, there is no diffusion in the block, which makes this mode insecure from a point of view [16]. On the other hand, this method is simple, fast in implementation and the damage or loss can be tolerated. However, because of using the same patterns in the plaintext, this method is not suitable for encrypting documents and images.

Cipher Block Chaining (CBC): Each ciphertext block is chained as the input of current plaintext block while the initial vector is used as the first input. Since the encryption depends on the blocks before it, this method brings in diffusion between blocks [17]. IV integrity should be protected but not necessarily needed to be secret. A single bit error may flip the next block, so this mode is not suitable for packet-critical applications.

Cipher Feedback Mode (CFB): While encrypting the IV with the key in the first block, this method encrypts the ciphertext of previous block and XORs with the plaintext of the current round. Because of a corrupted ciphertext block affecting every block after itself, this mode is error-intolerant [18].

Output Feedback Mode (OFB): The only difference between OFB and CFB is that, OFB uses the encrypted block of previous round as input to the current round, not the XORed ciphertext where feedback is independent from the message. Therefore, the error propagation is avoided in this mode [18]. However, when a plaintext is known, the output of the forward cipher function will be known, which makes this mode insecure in some ways.

Counter Mode (CTR): In CTR mode, current counter value is encrypted and XORed with plaintext, where only the low-order bits of the input are altered with the encryption of each block [19]. Furthermore, this method does not use any feedback value, which is an advantage against algebraic cryptanalysis. The data blocks can be randomly accessed, and the encryption is simple and fast.

2.2.2. Advanced Encryption Standard (AES)

In the competition of AES organized by NIST the idea was to replace DES with a faster, feasible and most importantly more secure encryption standard. The finalists of the competition were MARS, Twofish, RC6, Rijndael and Serpent. In most of the software implementations on different processors, Serpent was found out as the slowest algorithm out of all [20]. While the encryption rates were close at higher key sizes, Rijndael resulted

in the highest-performing block cipher among all the finalists [20]. While this algorithm had low memory consumption and a structure suitable for instruction-level parallelism, it was mathematically easy to understand and implement on both hardware and software [21]. Considering all the requirements of the competition, Rijndael was shown to have the best design to be a standard.

Rijndael Algorithm was called as AES after the invention of Vincent Rijmen and Joan Daemen [21]. Unlike DES, this algorithm does not have a Feistel structure. It is a symmetric-key block cipher that has SPN structure. Its key size is modifiable as 128/192/256. Even though there are three standard key sizes, it may be modified by changing the number of rounds and the length of the round key [22].

AES encryption algorithm consists of 4 layers that provide confusion and diffusion as demonstrated in the instance of SPN structure in Figure 2.2. In MixColumn and SubBytes layers, every byte is treated as an element of GF (2^8) and the arithmetics are computed in this Finite Field. Since the block length is 128 bits, data is arranged in a 4x4 matrix, whereas the key is arranged as:

- 4x4 matrix (128-bit key)
- 4x6 matrix (192-bit key)
- 4x8 matrix (256-bit key)

“SubBytes” layer is a non-linear substitution of bytes with the data in the lookup table so-called S-box. The contents of the S-box are pre-computed inverses of Finite Field elements [22]. In Rijndael, this inversion is always done in modulo $P(x) = x^8 + x^4 + x^3 + x + 1$.

“ShiftRows” layer is the step where a simple transposition occurs. Bytes in the second, third and fourth row of state matrix are circularly left-shifted by 1, 2, and 3 times respectively while the first row remains the same.

"MixColumns" operation is the multiplication of the State Matrix by a fixed matrix called MDS matrix. Additions and multiplications are performed in GF(2^8) in this state. Diffusion in this state occurs by multiplying a byte of the state matrix with the corresponding element of MDS matrix and then XORing with other elements of the state matrix. Finally, each byte of a column in the state matrix is mapped into a new value after the mixing operation [23]. Since multiplication with 1 in GF(2^8) is equal to the value itself, it is only needed to compute the multiplication of the element of the state matrix with 2 and 3.

"AddRoundKey" is the bitwise XOR operation of the round key with State Matrix. The round key is defined as the key that is added to the current round, which was created from the symmetric-key using the "Key Expansion Algorithm". For instance, a 4-word (16-byte) input symmetric-key is used to create 44-word (176-byte) round keys using this algorithm. Moreover, the number of rounds for 128/192/256-bit key length is 10/12/14 respectively. There is an extra AddRoundKey layer in the first round and there is no MixColumns layer in the last round according to Rijndael's design.

AES decryption algorithm is very similar to encryption due to the fact that these layers are easily reversible. AddRoundKey and ShiftRows layers are identical to their inverses. There is a fixed Inverse S-Box to be replaced in InverseSubBytes layer and a fixed matrix to be multiplied in InverseMixColumn layer.

2.2.2. Simplified AES (S-AES)

S-AES, an easy-to-understand educational algorithm, was founded by E. Schaefer and introduced in 2003 [24]. Arithmetics in this block cipher are computed in GF(2^4) using the polynomial $P(x) = x^4 + x + 1$. The structure is the same as the original Rijndael, whereas the number of rounds is simply two. Additionally, the significant difference is that the block size and key size are both decreased to 16-bits for faster encryption and implementation [3]. Just like Rijndael, the last round does not include MixColumns layer and there is an extra AddRoundKey before the first round. The encryption and decryption process of S-AES is given in Figure 2.3.

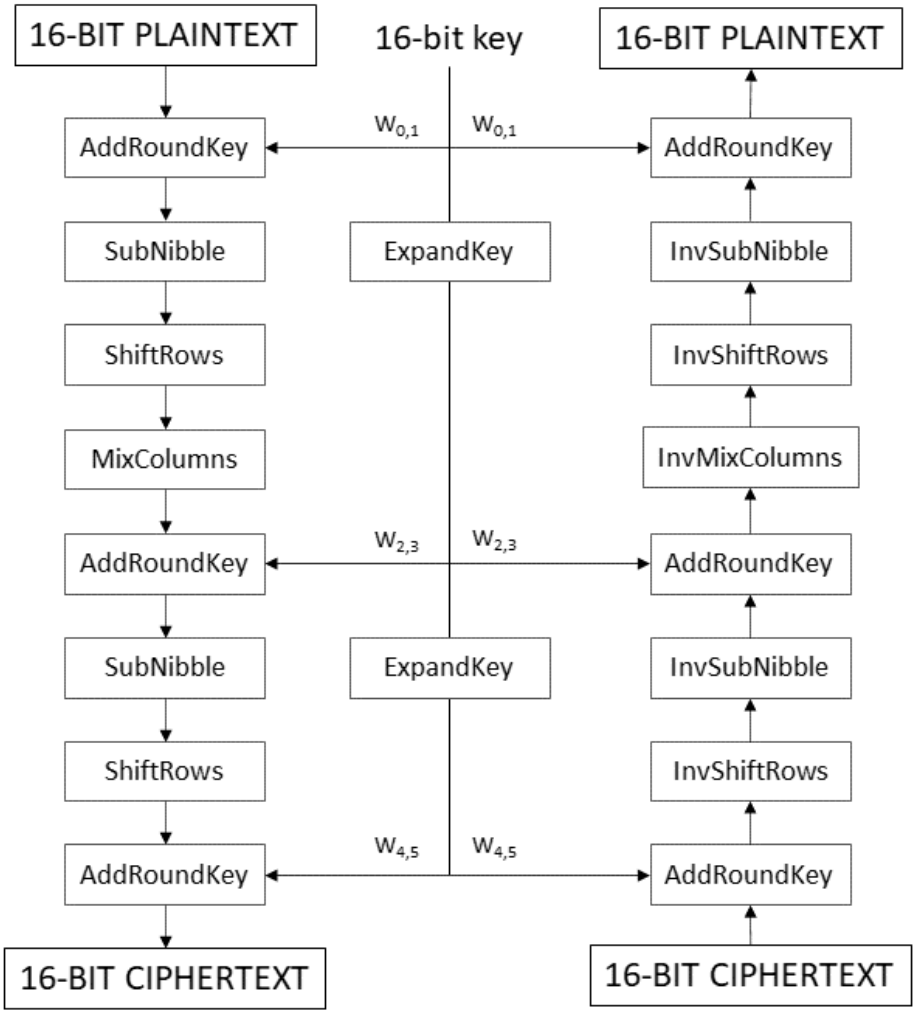


Figure 2.3 - Structure of S-AES Encryption and Decryption [7]

At first, to process the input data block, the given ciphertext is converted to a state matrix. Key Expansion Algorithm creates round keys 1 and 2 for the first and second rounds. The original symmetric-key ($w_0 w_1$) is used in the first AddRoundKey operation. The same key is XORed with State Matrix in the decryption process. Byte representation of Round Key is $[w_0\ w_1\ w_2\ w_3\ w_4\ w_5]$ whereas nibble representation of the State Matrix is $\begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix}$.

Bytes of RoundKey are computed using Eq. 2.1-2.6, while Eq. 2.7 is used to find Round Constant. Since there are only 2 rounds in S-AES, there will be 2 Round Constant values: RCON(1) & RCON(2).

$$w_0 = w_0 \tag{2.1}$$

$$w_1 = w_1 \tag{2.2}$$

$$w_2 = w_0 \oplus \text{RCON}(1) \oplus \text{SubNibble}(\text{RotNibble}(w_1)) \tag{2.3}$$

$$w_3 = w_1 \oplus w_2 \tag{2.4}$$

$$w_4 = w_2 \oplus \text{RCON}(2) \oplus \text{SubNibble}(\text{RotNibble}(w_3)) \tag{2.5}$$

$$w_5 = w_3 \oplus w_4 \tag{2.6}$$

$$\text{RC[i]} = x_i + 2 \tag{2.7}$$

Using Eq. 2.7, RC[1] and RC[2] values are found as follows.

- RC[1] = $x^3 = 1000_H$
- RC[2] = $x^4 \bmod (x^4 + x + 1) = 0011_H$

Since RCON(1) and RCON(2) constant values are used as hexadecimal (HEX) inputs, they are represented as 0x80 and 0x30 respectively. Nibbles are substituted with State Matrix instead of bytes in S-AES. Instead of SubBytes, this layer is called SubNibbles. The contents of the S-box are precomputed inverses of Finite Field elements in GF(2^4). Contents of S-Box and Inverse S-Box are given in Table 2.1 and Table 2.2 as hexadecimal where the first row and columns are binary. SubNibbles layer involves four independent nibble-to-nibble transformations [25].

Table 2.1 - Content of S-Box in SubNibbles layer

	00	01	10	11
00	9	4	A	B
01	D	1	8	5
10	6	2	0	3

11	C	E	F	7

Table 2.2- Content of Inverse S-Box in InvSubNibbles layer

	00	01	10	11
00	A	5	9	B
01	1	7	8	F
10	6	0	2	3
11	C	4	D	E

ShiftRows is a simple layer since the State Matrix consists of 2 rows and 2 columns. In encryption, the first row remains the same, while the second row is circularly shifted left. Due to having 2 columns, InvShiftRows is the identical operation of ShiftRows.

MixColumns layer includes Finite Field multiplication of states with MDS matrix which is $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$. The output nibbles of the MixColumns layer are given in Eq. 2.8-2.11 where the multiplication is computed in GF(2^4).

$$S'_{0,0} = S_{0,0} \oplus (4 \times S_{1,0}) \tag{2.8}$$

$$S'_{1,0} = (4 \times S_{0,0}) \oplus S_{1,0} \tag{2.9}$$

$$S'_{0,1} = S_{0,1} \oplus (4 \times S_{1,1}) \tag{2.10}$$

$$S'_{1,1} = (4 \times S_{0,1}) \oplus S_{1,1} \tag{2.11}$$

In order to understand the multiplication in MixColumns layer, it is necessary to learn Finite Field arithmetic. First of all, while computing the elements of the binary extension field, GF(2^4), x+1 is replaced with x^4 to arrange the coefficients of the polynomial into a binary string [26]. Using this method, hexadecimal equivalences of these elements are computed as shown in Table 2.3.

Table 2.3 - Powers of Every Element in GF(2^4)

Element of GF(2^4)	**Calculation**	**Hex Representation**
x^0	$x^0 = 1$	1
x^1	$x^1 = x$	2
x^2	$x^2 = x^2$	4
x^3	$x^3 = x^3$	8
x^4	$x^4 = x + 1$	3
x^5	$x^5 = x.(x + 1) = x^2 + x$	6
x^6	$x^6 = x^2.(x+1) = = x^3 + x^2$	C
x^7	$x^7 = x^3.(x+1)=x^4+x^3=x^3+x+1$	B
x^8	$x^8=x^4+x^2+x=x^2+2x+1=x^2+1$	5
x^9	$x^9 = x.(x^2+1)$	A
x^{10}	$x^{10}=x^4+x^2=x^2+x+1$	7
x^{11}	$x^{11}=x^3+x^2+x$	E
x^{12}	$x^{12}=x^4+x^3+x^2+x=x^3+x^2+x+1$	F
x^{13}	$x^{13}=x^4+x^3+x^2=x^3+x^2+1$	D
x^{14}	$x^{14}=x^4+x^3+x=x^3+1$	9

After computing the elements of the binary extension field, it is important to understand the multiplication of every element with 4 since MDS matrix in S-AES includes multiplication with only 1 and 4. From the fact that multiplying the input with one is equal to itself, there are only 15 different possible values that a nibble can get. Multiplications of every element in GF(2^4) with 4 are given in Table 2.4.

Table 2.4 - Multiplication of 4 with the Elements of GF(2^4)

Multiplication	Polynomial Representation	Result in HEX
4 × 0	$x^2 . 0$	0
4 × 1	$x^2.x^0 = x^2$	4
4 × 2	$x^2.x^1 = x^3$	8
4 × 3	$x^2.(x^1 + 1) = x^3 + x^2$	C
4 × 4	$x^2.x^2 \equiv x + 1$	3
4 × 5	$x^2.(x^2 + 1) \equiv x^2 + x + 1$	7
4 × 6	$x^2.(x^2 + x) \equiv x^3 + x + 1$	B
4 × 7	$x^2.(x^2 + x + 1) \equiv x^3 + x^2 + x + 1$	F
4 × 8	$x^2.x^3 \equiv x^2 + x$	6
4 × 9	$x^2.(x^3 + 1) \equiv x$	2
4 × 10	$x^2.(x^3 + x) \equiv x^3 + x^2 + x$	E
4 × 11	$x^2.(x^3 + x + 1) \equiv x^3 + x$	A
4 × 12	$x^2.(x^3 + x^2) \equiv x^2 + 1$	5
4 × 13	$x^2.(x^3 + x^2 + 1) \equiv 2x^2 + 2x + 1$	1
4 × 14	$x^2.(x^3 + x^2 + x) \equiv x^3 + x^2 + 1$	D
4 × 15	$x^2.(x^3 + x^2 + x + 1) \equiv x^3 + 1$	9

InvMixColumn is computed using the same operation while changing the MDS matrix to $\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix}$. Due to multiplying a state matrix element with both 9 and 2, InvMixColumn layer is slower than MixColumn layer. The output state matrix elements from the InvMixColumn layer are given in Eq. 2.12-2.15.

$$S'_{0,0} = (9 \times S_{0,0}) \oplus (2 \times S_{1,0}) \quad (2.12)$$

$$S'_{1,0} = (2 \times S_{0,0}) \oplus (9 \times S_{1,0}) \quad (2.13)$$

$$S'_{0,1} = (9 \times S_{0,1}) \oplus (2 \times S_{1,1}) \quad (2.14)$$

$$S'_{1,1} = (2 \times S_{0,1}) \oplus (9 \times S_{1,1}) \quad (2.15)$$

2.3. Implementations of Block Ciphers

There are different implementation techniques in the literature for implementing each layer of block ciphers. SPN and Feistel network ciphers are implemented frequently for performance and security purposes. As an example, ARX ciphers are easy to implement both in hardware and software due to using logical operations [27].

Hardware implementations of block ciphers appeared to be mostly faster than software implementations in the literature due to the availability of multiprocessing in hardware [28]. Processors of the computer or Microcontroller Units (MCUs) are generally used to implement algorithms on bare-metal due to the unnecessity of a complex structure in mathematical operations. In software, high throughput can be achieved using Assembly language because of using fewer instructions in each layer.

Encryption time in terms of clock cycles and utilized area in terms of memory usage are some of the important parameters when the implementation performance of a block cipher is measured [29]. Since the operations depend on the target platform's capabilities, the implementation efficiency depends on the algorithm's structure being whether hardware-oriented or software-oriented.

Some block ciphers such as AES are adjustable for the area (Lightweight Cryptography, IoT, security-critical applications, etc.) that the algorithm will be used in relation to the structure of the algorithm [30]. As the key size increases, encryption time increases, which leads to a more secure system for brute-force attacks [31]. For high-performance applications, AES-128 or a similar block cipher having less rounds is preferred.

2.3.1. Performance vs. Security

The critical points in the design of a cryptosystem are the implementation performance and security. The security consists of the algorithms' resistance to cryptanalysis

techniques such as SCA, linear, differential, or algebraic cryptanalysis. Even though the algebraic cryptanalysis is a big threat to symmetric-key block ciphers, AES is still algebraically secure [32]. It is difficult to design a secure and fast algorithm, yet it is possible to rearrange the key size of AES to adjust the algorithm for different applications that require various security demands. The time it takes to process every layer of AES in software can be seen in Figure 2.4.

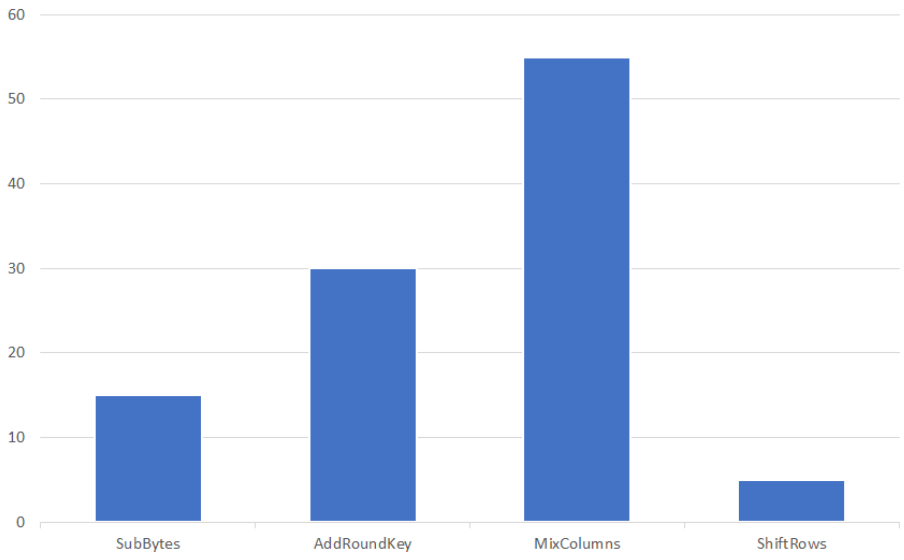


Figure 2.4 - Processing Time Percentage of Every Layer in AES [33]

According to the Institute for Defense Analyses Science and Technology Division, security metrics of a cryptographic algorithm are Type, Functions, Key Size, Rounds, Complexity, Attacks, and Strength [34]. There is another significant parameter called the Avalanche Effect. It refers to a small change in the input, affecting greater change in the output. In the design of block ciphers, Avalanche Effect is highly desired.

SCA is a cryptanalysis method where the information can be gathered via the board of the hardware implementations of cryptographic algorithms. Implementation characteristics are used to expose the details of the encryption [35]. There are many SCA techniques such as Timing Attacks, Power Analysis attacks, Electromagnetic Analysis attacks, Fault Induction Attacks, optical SCA, traffic analysis attacks, etc. Timing attacks are applied by sending random data and collecting the period of running time for different

inputs to obtain cryptographic parameters [36]. Power analysis consists of the analysis of the power that is consumed during the transitions between 1's and 0's. The transition from 0 to 1 consumes more power compared to the transition from 1 to 0 [37]. The electromagnetic attack is collecting meaningful data that leaks from the board which occurs by the changes in the electric current magnitude [38]. Electromagnetic coverings are used to weaken this type of SCA.

From the researches of Quantum Computers', it is a known fact that the processing speed will increase with the increasing qubit size [39]. It is assumed that many encryption algorithms will become vulnerable since the quantum computers are capable of decreasing the number of steps to N/2 where N is the key length. The security level of the algorithm also depends on key length, which should be at least 80-bits according to today's processors' frequencies [15]. With the help of Quantum processors, symmetric-key encryption algorithms might be vulnerable against Brute Force Attacks including AES-128 due to the key length halving to 64 bits [40]. However, AES-192/256 might still be strong enough to resist quantum computers for more years [40]. The emergence of Quantum Computers leads mathematicians to search for a more secure algorithm or to hide the details of already existing algorithms. Quantum Grover Attack was applied on S-AES and a new method is proposed to recover the private key when the Grover attack is applied [41].

On the other hand, there are many areas where high-performance algorithms are needed such as bus tickets, health-care devices, or school ID cards. Lightweight Cryptography finds applications in such areas, with emphasis on wireless communication or IoT applications. In these kinds of applications, encryption is still an ultimate priority where performance is also a must.

Mathematical operations are used in all of the layers of AES. In a survey on symmetric ciphers, multiplication takes 2881 clock cycles where arithmetic operations take clock cycles between 100-200 [42]. Therefore, there are steps to be taken to speed up multiplication during encryption.

2.4. Related Works

S-AES algorithm was first introduced and mathematically examined [24], though it was not implemented and performance or security analyses were not performed. The idea of implementing MixColumns as LUT was first proposed in [43] though the throughput was not enhanced compared to an MDS matrix implementation. In this thesis, the shift and add algorithm with respect to Basic and Fast LUT techniques implemented on any binary Finite Field are compared. Since the LUT is computed once, it is efficient to call the function many times. In software implementations, this approach is simple and easily implementable but may waste memory.

In [44], another approach to compute MixColumns layer in AES was presented and implemented in hardware. As a result of comparison with AES that implements MixColumns layer using MDS matrix, the proposed method is shown to have less number of gates.

In the research of [7], the performance of a Wireless Sensor Network (WSN) is improved while the memory usage is reduced using S-AES. The algorithm was implemented in a Field Programmable Gate Array (FPGA) and this implementation showed that S-AES can be used in a variety of areas to improve the performance of cryptosystems. The results proved that the throughput of S-AES is close to Hummingbird but less than AES, while the occupied memory was much lower than both AES and Hummingbird.

In [45], the researchers applied a cryptanalysis technique to break the key used in S-AES based on Known Plaintext Attack. Genetic Algorithms are used for cryptanalysis on S-AES for the first time. Consequently, this thesis provided reduced search space mostly for known plaintexts.

Another application that S-AES is used is on IP address management in [46]. This implementation provides different IP addresses generated by S-AES to users trying to access the network. This research revealed that S-AES is an option to provide security in

different domains including computer networks, cyber-physical systems, etc. and can be used to ensure confidentiality.

In [47], a modified version of AES is used for encrypting the health parameters of patients to keep privacy. The implementation also provides low power consumption which is important for IoT applications. This research showed that mathematical or implementational modifications on AES are applicable, although the result may not always be promising.

In [48], AES was again modified to improve processing speed and minimize the memory usage in embedded system applications. The proposed algorithm is called "LMEP-S-AES" and functions similar to AES. The application area of this research is quite similar to ours which is an implementation of a block cipher in an embedded processor, although the implementation process, used algorithm and the target MCUs are different. The round key remains the same for all rounds in LMEP-S-AES, although it differs in every round for S-AES.

Finally, even though the researches on modifying AES may be found useful and provide speed-ups, they are not standardized and cannot be used for security-critical or performance-critical applications. Because of this reason, a standardized algorithm such as S-AES should be used for encryption in IoT applications or domains where Ultra-Lightweight Cryptography is utilized.

3. EXPERIMENTAL WORK

3.1. Software Implementation of S-AES

S-AES was implemented in C programming language, using Dev-C as Integrated Development Environment (IDE), where the high-level programming language codes were compiled using GNU Compiler Collection (GCC) version 4.9.2. ShiftRows and SubNibbles are the easiest layers of all two versions due to their linear algebraic property. In this implementation, 3 different codes are written that included 3 different implementations of S-AES MixColumns: LUT, MDS and Switch-Case. Source codes of S-AES written for software and hardware platforms can be found in [49].

In the MixColumns layer of the encryption, every input is to be multiplied by 4 which corresponds to x^3 in GF(2^4). Therefore, the multiplication could be expressed using Eq. 3.1.

$$S_{out} = ((S_{in}.x).x).x \tag{3.1}$$

While the "multiplication by 4" operation is calculated bitwise, the output of the multiplication by 4 operation results as shown in Eq. 3.2-3.5.

$$out_3 = in_1 \tag{3.2}$$

$$out_2 = in_0 \oplus in_3 \tag{3.3}$$

$$out_1 = in_2 \oplus in_3 \tag{3.4}$$

$$out_0 = in_2 \tag{3.5}$$

Since we are working with nibbles in Finite Field, multiplication in the implementation will be equal to shifting and XORing operations. Because of having the polynomial P(x) = $x^4 + x + 1$, every number in GF(2^4) that are higher than x^4 will be replaced by x + 1 as

explained in Section 2.2.2. Since x + 1 in the binary field corresponds to 3, there will be an additional XOR operation to encounter the overflow that can happen due to x^4. Therefore, multiplication with 4 is computed by left shifting the input 3 times if the MSB of the input is 0. When the MSB is equal to 1, the input is left-shifted and XORed with hexadecimal 3 for 3 turns.

After computing the hexadecimal values of Finite Field elements, the multiplication of every element of $GF(2^4)$ with 4 has to be calculated for LUT implementation. Then the results of these calculations are written down in a LUT and called for every different input state. Moreover, this pre-calculation does not create a vulnerability in the implementation since the diffusion is provided by XORing the states as told in Eq. 2.8-2.11.

The third and last implementation, Switch-Case, was still based on the pre-calculation of these Finite Field elements with 4. This time the algorithm does not search for the value inside the LUT. Instead, it outputs the chosen "case". This fast selection mechanism is practical for small block-sized cryptosystems. Although, it might cover too much memory when used in long plaintexts.

LUT, MDS, and Switch-Case implementations of this diffusion layer are shown in Figure 3.1. In order to verify the correctness of the encryption, we entered 2 test vectors as input from [7, 45, 50], as given in Table 3.1 and the results are verified.

Table 3.1 - Test Vectors in Hex

Plaintext	Key	Ciphertext
6F6B	A73B	0738
616E	A73B	5547
6279	A73B	C36A

```
Function LUT(value)
    Array ResultofMult [16]:
        0x00,0x04,0x08,0x0c,
        0x03,0x07,0x0b,0x0f,
        0x06,0x02,0x0e,0x0a,
        0x05,0x01,0x0d,0x09
    return ResultofMult[value];
Function MDSmult(value)
    for i ← 0 to 2 do
        if LSB of value is 1 then
            (value << 1) ⊕ 0x03
        else
            (value << 1)
        return value;
Function SwitchMult(value)
    switch value do
        case 0 do
            value = 0x00
        case 1 do
            value = 0x04
        case 2 do
            value = 0x08
        .
        .
        .
    return value;
```

Figure 3.1 - Pseudo Codes of the Implementations of GF (2^4) Multiplications

3.2. Embedded Software Implementation of S-AES

The S-AES code implemented in software is the same as in embedded software except the execution time measurement part. The MCU that is used to implement this algorithm is STM32F4-DISCOVERY which is a high-performance board that included an ARM Cortex-M4 32-bit core. This controller works with the maximum frequency of 168 MegaHertz (MHz), and it included 1 Megabyte (MB) of Flash memory and 128 Kilobytes (KB) of Random-Access Memory (RAM) which were enough to fit the memory expectations of S-AES. Since Cortex-M4 has a pipelined structure, it fetches, decodes, and executes the disassembled instructions in a short period.

In order to implement an algorithm on an embedded processor, the critical point is to run the correct timer for the purpose. Data Watchpoint and Trace Unit (DWT) of STM32F4-DISCOVERY board is used in our application to count the cycles that pass-through S-AES encryption process. Clock Cycle Counter (CYCCNT) register of DWT counts until the end of the operation which is connected to the 168 MHz system clock. The pseudo code of the algorithm to measure the execution time is given in Figure 3.2.

```
#define ”startTimer” -> enable CYCCNT of DWT
#define ”stopTimer” -> disable CYCCNT of DWT
#define ”getValue” -> get value from CYCCNT
.
.
int main ()
    startTimer()
    beginCycles = getTimer()
    Encrpytion Process
    totalCycles = getTimer() - beginCycles
    stopTimer()
```

Figure 3.2 - Pseudo Codes of Measurement of the Execution Time in STM32

In cryptosystems, throughput is considered as the rate of data that is successfully encrypted over unit time. In other words, it is the size of the plaintext over execution time which is shown in Eq. 3.6 [51]. To calculate the execution time of the encryption process, the cycles that pass between starting and ending times are measured as shown in Eq. 3.7, while setting a breakpoint after measuring the ending time.

$$\text{Throughput} = \frac{\text{Block Length}}{\text{Execution Time}} \tag{3.6}$$

$$\text{Execution Time} = \frac{\text{Total Clock Cycles}}{\text{CPU Clock Frequency}} \tag{3.7}$$

In this implementation, the total clock cycle of the key scheduling is counted as 1785 whereas the encryption took 1172 clock cycles in an ARM Cortex-M4 processor. Since the block length is 16 bits, the execution time and throughput of S-AES encryption can be found using the following calculations.

- $\text{Execution Time} = \frac{1172 \text{ clock cycles}}{168 \text{ MHz}} = 6.97 \text{ µs}$
- $\text{Throughput} = \frac{16\ bits}{6.97 \text{ µs}} = 2.29 \text{ Mbps}$

The encryption times in terms of clock cycles of three implementation techniques in MixColumns layer are given below.

- MDS: 394 clock cycles
- LUT: 409 clock cycles
- Switch-Case: 240 clock cycles

From these results, the processing time of MixColumns layer over whole encryption is 240/1172 in Switch-Case method and 394/1279 in MDS method. In percentage, it is 20.47% and 12.86% when Switch-Case and MDS matrix are used respectively. Therefore, it can be said that the performance of MixColumns is improved using the proposed method.

Memory utilization of the encryption algorithm is also an important factor alongside the clock cycles. Total memory and utilized memory can be viewed using the Build Analyzer tool of Atollic TrueStudio. When all of the files in the project are included, the program takes up 2.19 KB out of 1 MB in Flash memory and 1.09 KB in RAM. In total, 3.28 KB of memory is used in this application which can be seen in Figure 3.3.

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08000000	0x08100000	1024 KB	1021,81 KB	2,19 KB	0,21%
RAM	0x20000000	0x20020000	128 KB	126,91 KB	1,09 KB	0,85%
MEMORY_B1	0x60000000	0x60000000	0 B	0 B	0 B	
CCMRAM	0x10000000	0x10010000	64 KB	64 KB	0 B	0,00%

Figure 3.3 - Memory Utilization of S-AES Encryption

Change in memory usage is not significant between MixColumns implementation techniques, although the fastest method is found as the most memory consuming method in terms of Flash Memory where RAM utilization stays unchanged as 1.09 KB. The program includes startup and configuration files, which also consume half of the Flash memory and 94% of RAM. The total memory utilizations of encryption in each method after other files are excluded from the project is given below.

- S-AES encryption (using Switch-Case): 1.17 KB
- S-AES encryption (using LUT): 1.1 KB
- S-AES encryption (using MDS): 1.05 KB

Since the algorithm searches the output in LUT, the performance of this method is slower than using the MDS matrix. The compiled and built codes are stored in the Flash memory

and RAM, not the Cache Memory. Even though using LUT takes lower clock cycles than multiplication as stated in the literature [43], the results are observed to be just the opposite. This is due to the Central Processing Unit (CPU) memory search where the desired output is stored in the LUT method. On the other hand, only logical operations are computed using the Arithmetic Logic Unit (ALU) while implementing this layer with an MDS matrix. The Switch-Case method resulted to be the fastest compared to both LUT and the MDS methods which are used in the literature.

To examine the time consumption of every method, it is easier to compare the clock cycles of instruction sets using Assembly language codes. Compiler, Assembler, and Linker are used in the routine of software development in a high-level programming language, whereas only Assembler and Linker are utilized in an Assembly code [52]. The software codes of each method that were written in C programming language were converted to Assembly codes after compilation. Consequently, an object file was created in the same folder as the project workspace.

While working on Dev-C/C++ IDE, "IDA Freeware" tool is used to disassemble the C codes into Assembly codes since it was a free and easy tool. Although, the free tool does not support object files compiled for ARM and needs an "ELF" dynamic library for ARM. To disassemble the embedded software codes, GNU ARM Embedded Toolchain was utilized which is a Pre-built GNU toolchain for ARM Cortex-M processors. With the help of "arm-none-eabi-objdump" command, the object file was disassembled into Assembly instructions. This command was applied to all of the three MixColumn implementation methods and the total clock cycles that each instruction takes are compared with each other[53]. The total number of instructions that are contained in the encryption process are found as follows.

- S-AES Encryption (using MDS): 76 instructions
- S-AES Encryption (using LUT): 80 instructions
- S-AES Encryption (using Switch-Case): 146 instructions

Although there are 146 instructions in Switch-Case method, a few instructions are computed while calling the output from cases. Instruction cycles to compute one nibble in MixColumns are presented in Table 3.2.

As it can also be seen from the Assembly code, the procedure of a LUT is to copy a block of memory by store and load operations which takes 1 + N clock cycles each. The letter "N" in LUT represents the number of registers to be loaded or stored. For instance, in "ldmia" instruction, N is equal to 5 due to using r0, r1, r2, and r3 registers for loading and r5 register for storing. Consequently, the clock cycles to compute "ldmia" and "stmia" becomes 6 each. Since the execution time of one nibble becomes 12 clock cycles, the reason for low-speed encryption in LUT method was proved.

On the other hand, the procedure of MDS was divided into 2 depending on the entering Most Significant Bit (MSB) of the nibble. The corresponding nibble is branched, loaded into a register, and shifted towards left three times. Additionally, logical AND operation and logical XOR operation are computed if the corresponding bit is equal to one. Finally, in Switch-Case method, it only takes 3 or 4 clock cycles to read the nibble that was written inside the "case" because of only writing, storing, and branching operations.

Table 3.2 - Disassembled ARM Instructions of MixColumns Implementation Methods

Method	**ARM Instruction**	**Description**	**Clock Cycles**
MDS_1	b.n 36<multiplyBy4+0x36>	Branch	1
	ldr r3, [r7, #4]	Load Word	1 or 2
	lsrs r3, r3, #3	Logical Shift Right	1
MDS_2	bne.n 2a<multiplyBy4+0x2a>	Branch Not Equal	1
	ldr r3, [r7, #4]	Load Word	1 or 2
	lsls r3, r3, #1	Logical Shift Left	1

	and.w r3, r3, #15	Logical AND	1
	eor.w r3, r3, #3	Logical Exclusive OR	1
LUT	ldmia r5!,[r0, r1, r2, r3]	Load Multiple, Increment After	1+N
	stmia r4!,[r0, r1, r2, r3]	Store Multiple, Increment After	1+N
Switch-Case	movs r3, #0	Move (Segment)	1
	str r3, [r7, #4]	Store Word	1 or 2
	b.n b4<multSwitch+0xb4>	Branch	1

Theoretically, the execution time of InvMixColumns in the decryption phase will take much time due to the multiplication of every element in GF(2^4) with 2 and 9 instead of just 4. All of the other steps in decryption will approximately have similar encryption times compared to encryption because of performing the inverse of the same operations.

Single execution times of the multiplication in MixColumns layer in terms of clock cycles can be stated as given below:

- MDS: 4 or 6 (depending on the state matrix)
- LUT: 10
- Switch-Case: 4

3.3. RISC-V and ARM Implementations of S-AES on QEMU

The purpose of this experiment is to compare two architectures and verify whether ARM is the best possible option to implement S-AES on or not. Most of the newly designed block ciphers are verified on RISC-V and ARM cores. As RISC-V is another architecture used for high-performance computing applications, it was a requirement to at least demonstrate our cryptosystem on this core.

S-AES is implemented on QEMU which is a simulator running on a Linux distribution (Ubuntu 18.04) for both RISC-V and ARM architectures. The simulated RISC-V board is selected as “RV64”. The execution time of RISC-V implementation is measured as 0.004998 seconds, whereas it took 0.003995 seconds to execute the algorithm on ARM architecture. Even though the execution time difference shows the high-performance implementation of ARM compared to RISC-V in this experiment, it does not indicate the encryption time since it includes more instructions during the fetch and execute steps of the processor. Commands while compiling, running, and disassembling on both platforms are mentioned in Appendix 1.

Consequently, it was shown that the implementation of S-AES on ARM took less time due to having a less complex ISA (Instruction Set Architecture) and including more compatible instructions for encryption operations which lead to the feasibility of block ciphers on ARM architectures.

In this experiment, we used the object files created by the C implementation of S-AES as introduced in Section 3.1 to compare the execution times between ARM and RISC-V architectures. Eventually, it is found out that ARM is, yet the best architecture that S-AES can be implemented on due to its fast instructions as explained in Section 3.2.

3.4. FPGA Implementation of S-AES

Since many of the block ciphers are implemented on hardware to provide high throughput compared to software implementations, the implementations are put in FPGA Benchmarking competitions arranged by NIST.

It was also important to demonstrate the high performance and low-area implementation of S-AES in FPGA to obtain alternatives to other ultra-lightweight block ciphers. To do so, the algorithm was implemented in VHSIC Hardware Description Language (VHDL), and all of the modules are behaviorally tested in simulation with test benches. The VHDL implementation was performed in Quartus II and Xilinx Vivado 2021.1 tools to ensure

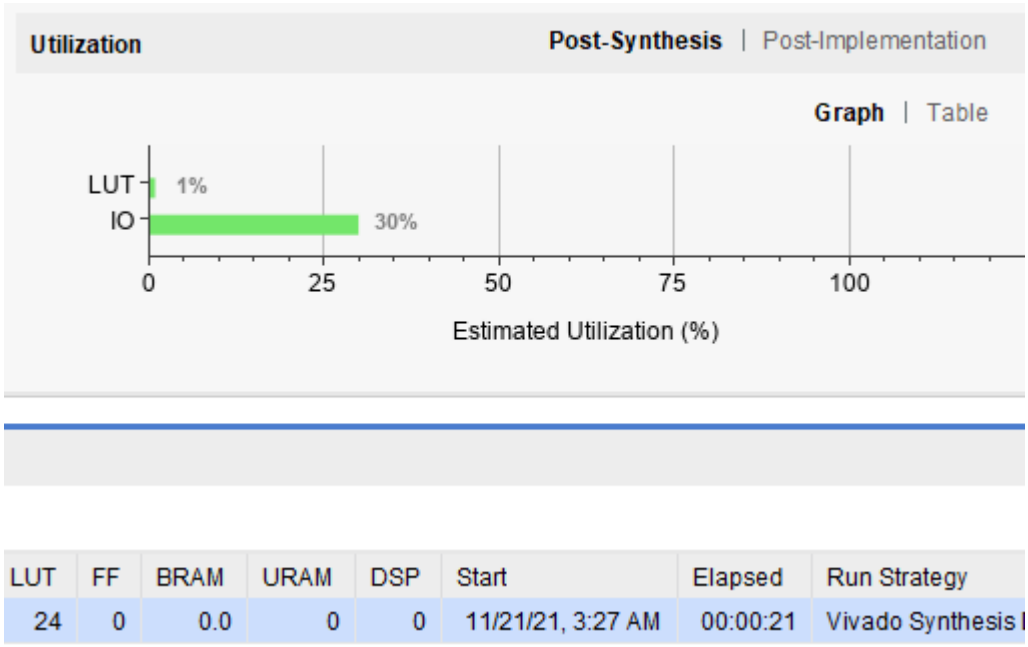
validity. Compact and fast VHDL implementation of S-AES can be found in [49] with their corresponding test benches.

The synthesized code was embedded to Altera Cyclone IV FPGA that appears to be on EP4CE6E22C8 demo board to capture the area of the implementation on hardware. According to its datasheet, the device has 6,272 logic elements, 144 pins, and a speed grade of 8 out of 9. This IC is categorized under Enhanced Thin Quad Flat Pack and operates up to 85° C maximum temperature. Moreover, compact S-AES design was also verified using Xilinx BASYS-3 demo board.

From the results of the S-AES VHDL implementation as shown in Figure 3.4 and Figure 3.5, it can be stated that all of the logic elements are implemented using combinational circuits. Additionally, there are no memory elements used in the core algorithm since S-AES only relies on mathematical operations. Because of having 16-bit key, plaintext, and ciphertext; the module uses only 50 pins on the device which is a compact and feasible solution for FPGAs with fewer pins such as Cyclone-IV and Artix-7. It is also advantageous that there will be no need for a register file or RAM to read the inputs and write the outputs, where the inputs and outputs can directly be assigned to pins for FPGA benchmarkings. Moreover, most of the lightweight cryptography algorithms are not compatible with this board even though they have fast and low-area structure.

Flow Status	Successful - Sat Oct 16 12:01:32 2021
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	S_AES_enc
Top-level Entity Name	S_AES_enc
Family	Cyclone IV E
Device	EP4CE6E22C8
Timing Models	Final
Total logic elements	48 / 6,272 (< 1 %)
Total combinational functions	48 / 6,272 (< 1 %)
Dedicated logic registers	0 / 6,272 (0 %)
Total registers	0
Total pins	50 / 92 (54 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)

Figure 3.4 - Synthesis Results of S-AES in Quartus II



LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
24	0	0.0	0	0	11/21/21, 3:27 AM	00:00:21	Vivado Synthesis

Figure 3.5 - Synthesis Results of S-AES in Vivado

The top module is simulated using ModelSim-Altera tool for Cyclone-IV and ISim tool for BASYS-3. Simulation results of S-AES VHDL implementation in Vivado are shown in Figure 3.6. Test vectors used in software implementations are also verified on this platform. The encryption process starts at the rising edge of the second clock period since the plaintext and key enter the encryption at this point. As a result of the simulation, the encryption process took only 1 clock cycle on hardware.

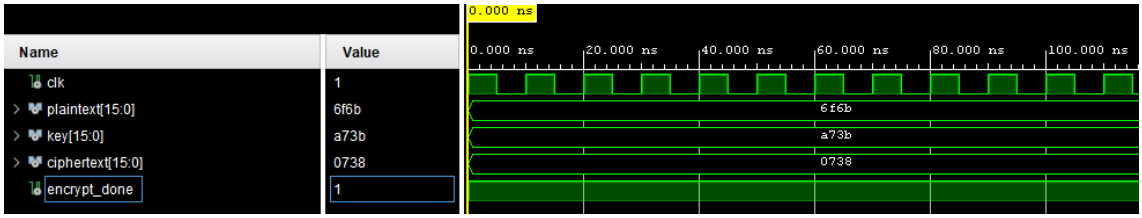


Figure 3.6 - Simulation Results of S-AES in Vivado

During the calculation of throughput, the same method in software implementations is utilized as given in Eq. 4.5 and 4.6, where FPGA clock frequency is used instead of CPU clock frequency. To measure the throughput of the encryption, it is essential to perform Static Timing Analysis on the target FPGA which is an important step of FPGA Benchmarking [54]. To do so, the top-level encryption block is connected to two 16-bit registers from the input and output that are designed using a total of 32 Flip-Flops (FF). Addition of FF's do not cause an effective decrease in throughput since FF's are only capable of creating hold times and the delay is caused by the setup time of the combinational circuit [55]. The Schematic of the core algorithm and register-driven circuit are given in Figure 3.7 and Figure 3.8.

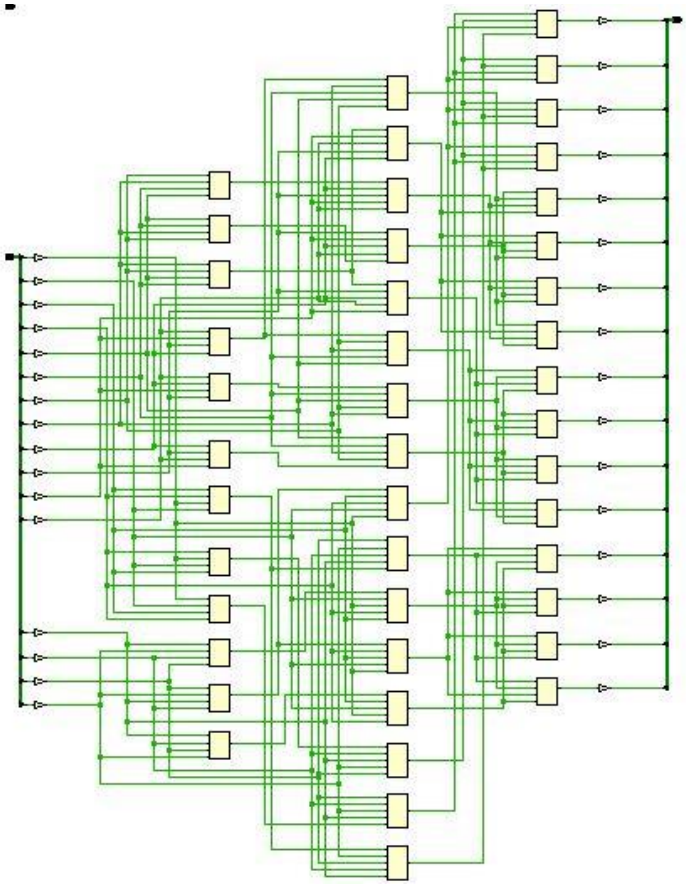


Figure 3.7 - Schematic of the IP Core of S-AES Encryption Algorithm

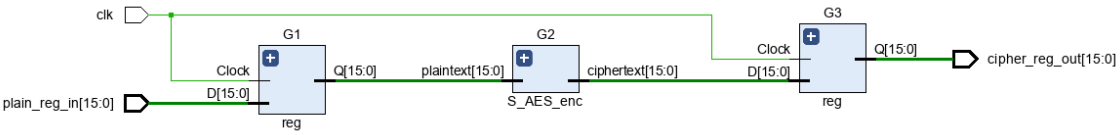


Figure 3.8 - Schematic of S-AES Encryption with 2x 16-bit Registers

After decreasing the pre-defined clock period to 4 ns, the WNS value decreases to 0.025 when the period is 4 ns, as it can be seen from Figure 3.9. From the results of the timing analysis, the execution time of the S-AES encryption is found as 2.85 ns. Accordingly, the working clock frequency of the FGPA in Xilinx BASYS-3 is found as 329 MHz, where the throughput becomes 5.61 Gigabits per second (Gbps) as calculated in the equation below.

$$\text{Throughput} = \frac{\text{Block Length}}{\text{Execution Time}} = \frac{16\ bits}{2.85\ ns} = 5.61\ Gb/s$$

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.025 ns	Worst Hold Slack (WHS):	0.173 ns	Worst Pulse Width Slack (WPWS):	0.705 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	32	Total Number of Endpoints:	32	Total Number of Endpoints:	33

All user specified timing constraints are met.

Figure 3.9 - Results of Static Timing Analysis

3.5. Security Analysis

S-AES key size and number of rounds are pretty small for performance, but the algorithm complexity is high enough in the manner of diffusion and confusion. Differential and Linear cryptanalysis of the S-AES cipher was introduced in [24], stating that the algorithm cannot be broken by an algebraic attack. Avalanche Effect is measured to analyze the security of the cipher, while the security of the key and encryption operations are measured in SCA.

3.5.1. Avalanche Effect

An important parameter while measuring the security of a block cipher is the Avalanche Effect. The same test vector in Table 3.1 is used to measure the Avalanche Effect as shown in Table 3.4. Firstly, 1 bit of the plaintext is flipped, and it is observed that 6 bits of the text are changed. Then 1 bit of the key is flipped, and it is observed that 10 bits of the text are changed. The average rate of change in ciphertext compared to plaintext is 8 bits out of 16 bits. Therefore, the Avalanche Effect of S-AES is equal to 50%. Avalanche Effects of some block ciphers are shown in Figure 3.10. According to this graph, AES is the most desired block cipher for Avalanche Effect while S-AES has an average value and ended up with a promising result despite its low text-key sizes.

Table 3.4 - Results of Avalanche Effect

Process	Plaintext	Key	Ciphertext
Test Vector	6F6B	A73B	0738
Changing Plaintext	6E7B	A73B	9737
Changing Key	6F6B	A72B	D284

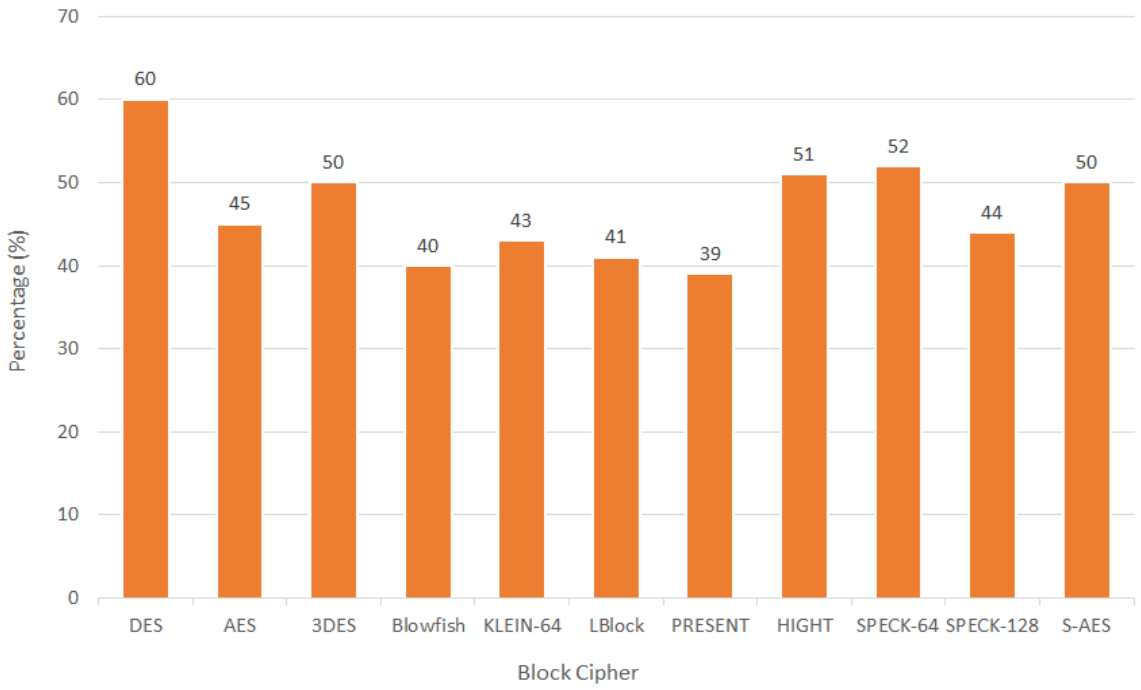


Figure 3.10 - Avalanche Effect of Some Lightweight Block Ciphers [13, 56]

3.5.2. Side-Channel Analysis

It is a fact that SCA has a big role in embedded system implementations of algorithms including FPGAs and MCUs. Since the main concentration of this thesis is ARM Cortex-M4, it is necessary to examine the strength of this implementation against at least the most common and basic SCA such as Timing Attack and Simple Power Analysis.

3.5.2.1. Timing Attack

From the fact that the encryption process should not be data-dependent, it is important to analyze the encryption times of the three implementations with altering plaintexts [57]. The results of obtained ciphertexts, clock cycles of the encryption process, and clock cycles of the MixColumns layer are given in Table 3.5 based on the implementation method.

Table 3.5 - Results of Timing Attack

Method	Plaintext	Key	Ciphertext	Clock Cycles of MixColumns	Clock Cycles of Encryption
MDS	1624	A73B	9F44	393	1281
	6028	A73B	221B	396	1284
	2F94	A73B	1278	393	1281
LUT	1624	A73B	9F44	409	1327
	6028	A73B	221B	409	1327
	2F94	A73B	1278	409	1327
Switch - case	1624	A73B	9F44	229	1121
	6028	A73B	221B	232	1124
	2F94	A73B	1278	234	1126

Each mathematical operation takes a different time to compute, which leads MDS method to return in different encryption times. From the controlled experiment in Timing Attack, it is found out that it takes 393 and 396 clock cycles to compute the MixColumns layer when the key is fixed to "A73B". The reason for this output is due to the extra "AND" and "XOR" operations when the MSB of the entering nibble is equal to 1 as shown in Figure 3.1. When the plaintext is 1624, the value of the state matrix is 3744 before entering the MixColumns layer. It takes 393 clock cycles to compute this plaintext since neither of the MSB of nibbles in 3744 is equal to one. On the other hand, the state matrix is equal to CB45 before computing the MixColumns layer when the plaintext is 6028. Since the MSB of "0x0C" and "0x0B" are equal to one, it takes 3 more clock cycles to

encrypt this plaintext. Since the operation in the encryption process should not alter the encryption time or current flowing through the processor to counter SCA [57], MDS matrix method can be vulnerable against Timing Attacks.

As explained in Section 2.3.1, LUT method is robust against timing attacks [8] because it gives output in the same periods instead of computing the input. As it can be seen from Table 3.5, the encryption process took 1327 clock cycles independent from the plaintext. This is due to loading or storing the data in identical time slots independent from the input as explained in Table 3.2. Therefore, it can be clearly said that LUT method implementation is invulnerable against Timing Attacks.

Execution times in Switch-Case for different inputs resulted in similar execution times. Since no mathematical operation occurs in this method, it can be stated that the execution time depends on reading and branching as explained in Section 3.2. From the results of Table 3.5, multiplication using MDS matrix might be vulnerable against SCA, whereas Switch-Case and LUT methods provided a secure implementation in terms of Timing Attack since the execution time does not depend on the input or operation.

3.5.2.2. Simple Power Analysis

In order to perform Simple Power Analysis, datasheet of STM32F4-DISCOVERY board was searched to measure the power consumption of the MCU from the correct unit. While examining the power consumption of an embedded processor, the current that passes through the supply voltage of the processor was measured since the critical data such as key can be recovered from this pin. Accordingly, measuring the power using the current probe of a highly sensitive digital oscilloscope was the best method to obtain a noise-free signal. During this process, Tektronix TCP0030 current probe is used with DPO7254 Digital Phosphor Oscilloscope.

From the datasheet of the MCU, V_{cap1} and V_{cap2} were found as the nodes that connect two grounding capacitors, known as C33 and C36 respectively, to the processor in STM32F407-DISCOVERY board. In order to prevent the noise, the current measurement

was obtained from the ground leg of the corresponding capacitor. Generally, the regulator that feeds the processor is shut down and the processor is fed externally with a power supply to decrease the noise in this type of SCA. Although, the bypass regulator that reduces the core voltage in STM32F4-DISCOVERY board is embedded inside the core. Because of this reason, the current measurement was obtained from the ground leg of C33 using the current probe. Block diagram representation of the Power Analysis experimental setup is given in Figure 3.11.

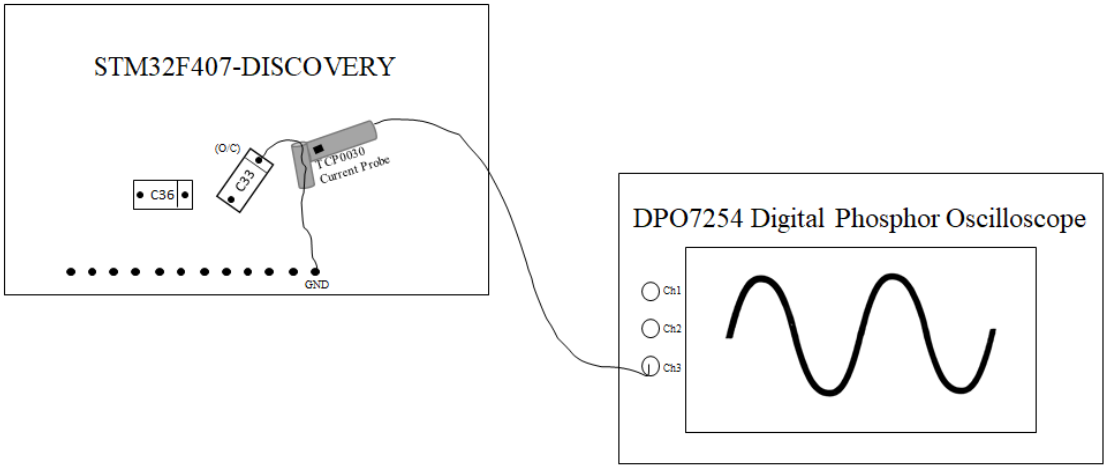


Figure 3.11 - Block Diagram of the Power Analysis Experimental Setup

Waveforms of the power consumption of the core during the execution of S-AES were shown in Figure 3.12 and Figure 3.13. The x-axis shows time domain, where y-axis shows the drain current of ARM Cortex-M4. Since the execution time of S-AES is found as 6.97 µs in Section 3.2, whole process is included in the waveforms as the total passing time is 10 µs in Figure 3.12 and 20 µs in Figure 3.13.

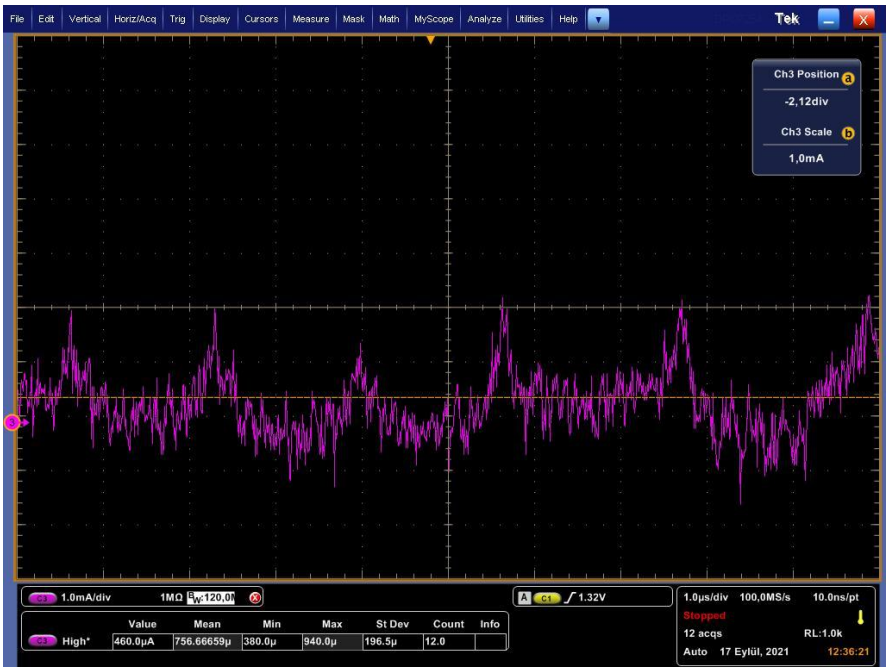


Figure 3.12 - Power Consumption of the Core for 1 us/div

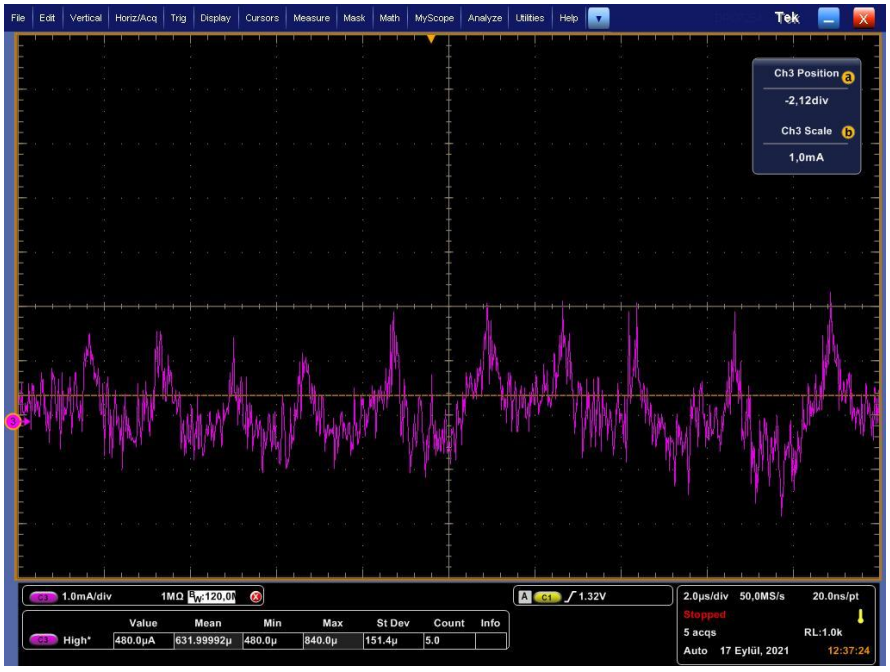


Figure 3.13 - Power Consumption of the Core for 2 us/div

The horizontal orange line in the waveforms shows the average current value. As introduced in Section 2.3.1, the drain current that feeds the core increases when a logic operation occurs on ALU, which means that the MCU core consumes more power when there is a state change. It can be inferred that a total of 5 increasing points shown on

Figure 3.12 is due to occurrence of a bit flip from 0 to 1, since it consumes more power than the transition from 1 to 0 [37].

Differential Power Analysis (DPA) and Correlation Power Analysis (CPA) are more methodologic techniques for power consumption in SCA for revealing the key, whereas SPA can be useful for finding out the operation performed in the core [58]. Since the layers of block ciphers having SPN structures consist of substitution and permutation operations, it is possible to determine the performed layer by the voltage or current level. As introduced in Section 2.2.2, S-AES performs 4 layers in 2 rounds, where MixColumns is performed only once. Due to performing the highest computation in MixColumns layer as shown in Figure 2.4, it can be inferred that the current fluctuates on the average current value for more than 1.5 μs after the 4th peak of the waveform. Since most of the nibbles are substituted before XORing in Switch-Case implementation of MixColumns, this fluctuation and peak after that can be estimated as the MixColumns layer of S-AES as shown in Figure 3.14.

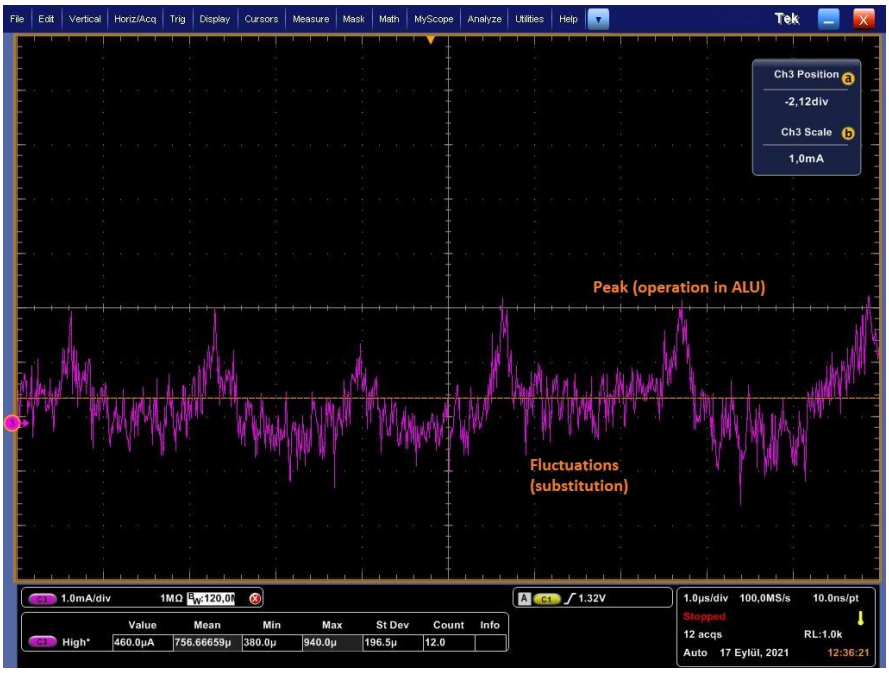


Figure 3.14 – Analysis of the Power Consumption Waveform

Eventually, no critical data such as a block of the state matrix or a part of the key was gathered using SPA, while it was shown that the processed layer can be approximated if

the structure of the block cipher is well-known. Even though the acquired information can be useless for the attacker, it is required to create confusion in every operation to prevent the encryption system safe against SCA.

4. RESULTS AND DISCUSSION

4.1. Results

In this thesis, S-AES implementation performance has been improved using the proposed Switch-Case method regarding the standard MDS matrix implementation. Three methods for implementing the MixColumns layer of S-AES were applied on an STM32F4-DISCOVERY and compared with each other. With the proposed Switch-Case method, a faster implementation was obtained because of executing simpler instructions with a little trade-off from memory. On the other hand, implementing MixColumns layer using MDS matrix was shown as a slower but memory-saving method in ARM architecture. Finally, LUT was presented as the slowest but the simplest one to implement with an average memory consumption. The structure of the block cipher was analyzed, and as a contribution, the proposed Switch-Case method was shown as the best option due to its simple and fast structure which resulted in a throughput of 2.29 Megabits per Second (Mbps) on ARM Cortex-M4.

The memory analyses on the MCU and FPGA boards showed that S-AES encryption process does not waste too much memory neither in software nor hardware. While validating the area utilization of S-AES on hardware, Altera Cyclone-IV and Xilinx BASYS-3 boards were used. 48 LUTs in Cyclone-IV and 30 LUTS in BASYS-3 are utilized which are highly considered as low-area implementations. Regarding to the static timing analysis, the implemented Register-Transfer Level (RTL) works on 329 MHz in BASYS-3, which lead to a throughput of 5.61 Gbps. Lowest-area and highest-throughput implementation of S-AES on FPGA was performed in this study.

While investigating the security of the implementation, the Avalanche effect of the cipher was examined and an average of 50 percent change in the plaintext occurred in the demonstrations. Timing attack was applied on S-AES implementation methods and the resulting execution times were compared with each other. Switch-Case and LUT methods were proven to be invulnerable against the attack, while the output obtained from MDS method was found out to be dependent to the MSB of the input. Therefore, proposed

Switch-Case method can be used to replace the MDS matrix implementation in AES-like block ciphers when Side-Channel security is considered as critical. It was also experienced that MixColumns is the only layer that causes a weakness against Timing Attack since execution time of the core in other layers stay unchanged.

Another non-invasive SCA was performed in the implementation, which is known as SPA, where none of the parts of the key, state matrix or ciphertext are recovered. Although, the time intervals where specific operations are performed are revealed by knowing the structure of the block cipher. Using this method, power consumption during the MixColumns layer that was implemented using Switch-Case method was approximated on the waveform.

4.2. Discussion

4.2.1. Discussion on Embedded Software Implementations

Implementation in software showed that this algorithm can be used in some embedded system applications. The target areas are ultra-lightweight cryptography and IoT applications, where cryptographic algorithms or protocols are implemented and used mostly in real-time environments [59]. Its standards are introduced in ISO/IEC 29192 which is the standard for Information Technology/ Security Techniques. According to this document, there are block ciphers that are suitable for only hardware and only software such as SEA, IDEA, TEA, and AES. Properties of some block ciphers implemented in ARM Cortex-M4 that are compared with S-AES are presented in Table 4.1.

Table 4.1- Implementation Results of Lightweight Block Ciphers on ARM Cortex-M4

Algorithm	**Block Size**	**Key Size**	**Clock Cycles**	**Memory (KB)**
AES [60]	128	128	943	4.39
3DES [61]	64	56	5620	-
Blowfish [62]	64	32–448	440	4

PRESENT [63]	64	80	1618	4.5
CLEFIA [51]	128	128	3349	1.3
TWINE [51]	64	80	2463	1.3
HUMMINGBIRD-2 [64]	16	128	332	2.3
S-AES	16	16	1172	1.17

PRESENT is a state-of-art ultra-lightweight block cipher and is fast in hardware. There is another ultra-lightweight block cipher called HUMMINGBIRD which has 4.7 times faster throughput than PRESENT. Furthermore, this Ultra-Lightweight Cipher is resistant to linear and differential cryptanalyses [25]. Another Lightweight block cipher called KLEIN is especially for the usage of resource-constrained devices [65], whereas TWINE algorithm introduces simple embedded software implementation with its Feistel structure [66]. S-AES was shown as the lowest area consumption encryption algorithm out of all Lightweight Cryptography algorithms implemented in this work. The common point of these algorithms is that they yield slow throughput when implemented in software.

Switch-Case implementation of MixColumns is also shown as applicable to AES, but this causes slightly more of a memory waste due to high length block size and three times more multiplications in GF(2^4). On the other hand, there is only one Finite Field multiplication in the encryption phase and there are two multiplications in the decryption phase that are based on shifting and XORing operations when the implementation perspective is considered.

Finally, it can be inferred that S-AES is a great option for low-power microcontroller applications such as IoT or Ultra-Lightweight Cryptography where moderate security is required. On the other hand, for security-critical applications, this algorithm might not be a good option due to its small key size. In this process, it is found out that fast implementation or algebraic design of a block cipher does not only require performance improvement but also requires secure, feasible, and accessible implementation.

4.2.1. Discussion on Hardware Implementations

Implementations of some lightweight block ciphers in Altera Cyclone-IV and Xilinx BASYS-3 FPGAs are given in Table 4.2. The source codes of the verified block cipher IP cores are taken from Open Cores [67], except S-AES, which is performed during the experimental work of this thesis study. From the table, it can be stated that AES, DES and Noekeon are fast and low-area implementations, whereas AES appeared to waste too much memory on Cyclone-IV. On the other hand, S-AES is exceptionally fast and memory efficient compared to other lightweight block ciphers.

Table 4.2 – Implementation Results of Lightweight Block Ciphers on FPGAs

Block Cipher Structure			**Performance**	**Utilized Area on Devices**			
				Altera Cyclone-IV (EP4CE6E22C8)		**Xilinx BASYS-3 (XC7A35T)**	
Block Cipher	*Block Size (bits)*	*Key Size (bits)*	*Clock Cycles*	*Area (LUT)*	*Area (FF)*	*Area (LUT)*	*Area (FF)*
AES [68]	128	128	10	4773	264	1069	264
Camellia [69]	128	128	26	4486	6994	7437	8223
DES [70]	64	56	18	740	142	367	142
Noekeon [71]	128	128	15	980	264	760	264
PRESENT [72]	64	80	33	224	152	180	152
TEA [73]	64	128	65	301	231	200	231
XTEA [74]	64	128	268	389	162	319	162
S-AES	16	16	1	48	0	24	0

Since the working frequency of the FPGA depends on the designed RTL, it is not possible to obtain the working frequency of the algorithm unless the static timing analysis is

applied. Therefore, comparing the total clock cycles and number of LUTs are better scales for hardware benchmarking since the delay on the circuit depends on the combinational circuits, and not registers [75]. Another variable to scale the efficiency of the hardware implementation, Throughput/Area is not presented on the table due to its incorrectness while the memory elements are considered in some of the block cipher implementations.

5. CONCLUSION

In this thesis, S-AES was mathematically analyzed and implemented in software and hardware, while the resistance against side-channel attacks was examined. Switch-Case method was proposed to implement MixColumns layer, while comparing the encryption times with LUT and MDS methods. Accordingly, it was shown that the MixColumns layer in AES-like algorithms can be accelerated with a small trade-off from memory using the proposed method. With the QEMU implementation, execution time of S-AES was shown to be faster on ARM architecture rather than RISC-V. A fast and low area hardware implementation showed that block ciphers like S-AES can be implemented without any registers with the lowest possible number of LUTs. Furthermore, proposed method brought invulnerability of MixColumns layer against Timing Attack. SPA was applied against the implementation on STM32 board and the time duration of the MixColumns operation was approximated using the fluctuations and peaks on the power consumption waveform. The Avalanche Effect of this cipher was found out to be 50 %, which is a promising result for such a small key size compared to other lightweight block ciphers.

To sum up, we showed that performance in a block cipher is an important parameter that needs to be improved while maintaining the implementation security since performance critical applications are requested as the technology evolves. Based on this thesis, it can be said that S-AES algorithm can be utilized in IoT applications where ultra-lightweight cryptography applications are needed for fast and comparably secure encryption.

In addition to this study, DPA or CPA analysis of this implementation can be studied for security enhancements. Since the MCU we are using is a low-power microcontroller, a µA scalable probe is needed for precise power measurement from the core which leads to a more accurate key correlation. Therefore, CPA can be applied on STM32 implementation of S-AES using a µA scalable probe as a future work for security improvements.

In terms of performance improvements, Switch-Case technique can also be applied to AES-128/192/256 to provide an efficient and high-throughput software implementations. An IoT application can be performed as well to verify the software or hardware encryption performance in IoT devices.

6. REFERENCES

[1] H. Wang, B. Sheng, C. C. Tan and Q. Li, Comparing symmetric-key and public-key based security schemes in sensor networks: A case study of user access control, 2008 The 28th International Conference on Distributed Computing Systems, IEEE, **2008**, pp. 11-18.

[2] E. Biham and A. Shamir, Differential cryptanalysis of the data encryption standard, Springer Science & Business Media, **2012**.

[3] W. Stallings, Cryptography and network security, 4/E, Pearson Education India, **2006**.

[4] A. H. Awlla and S. M. Aziz, Secure Device to Device Communication for 5G Network Based on improved AES, The Scientific Journal of Cihan University–Sulaimaniya PP, 57 (**2021**) 67.

[5] W. Yu and S. Köse, A lightweight masked AES implementation for securing IoT against CPA attacks, IEEE Transactions on Circuits and Systems I: Regular Papers, 64 (**2017**) 2934-2944.

[6] S. Maitra, D. Richards, A. Abdelgawad and K. Yelamarthi, Performance Evaluation of IoT Encryption Algorithms: Memory, Timing, and Energy, 2019 IEEE Sensors Applications Symposium (SAS), **2019**, pp. 1-6.

[7] N. Kavana and S. PremanandaB., Implementation of Simplified AES algorithm for Wireless Sensor Nodes on FPGA, **2013**.

[8] J. Bonneau and I. Mironov, Cache-Collision Timing Attacks Against AES, Springer Berlin Heidelberg, Berlin, Heidelberg, **2006**, pp. 201-215.

[9] A. Biryukov, Block Ciphers and Stream Ciphers: The State of the Art, IACR Cryptol. ePrint Arch., 2004 (**2004**) 94.

[10] A. Sevin and A. A. O. Mohammed, A survey on software implementation of lightweight block ciphers for IoT devices, Journal of Ambient Intelligence and Humanized Computing, (**2021**) 1-15.

[11] V. Nachef, J. Patarin and E. Volte, Feistel ciphers, Cham: Springer International Publishing, (**2017**).

[12] S. S. Ali and D. Mukhopadhyay, Differential fault analysis of Twofish, International Conference on Information Security and Cryptology, Springer, **2012**, pp. 10-28.

[13] P. Patil, P. Narayankar, D. G. Narayan and S. M. Meena, A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish, Procedia Computer Science, 78 (**2016**) 617-624.

[14] B.-T. Liu, L. Li, R.-X. Wu, M.-M. Xie and Q. P. Li, Loong: a family of involutional lightweight block cipher based on SPN structure, IEEE Access, 7 (**2019**) 136023-136035.

[15] P. B. Ghewari, J. Patil and A. Chougule, Efficient hardware design and implementation of AES cryptosystem, International journal of engineering science and technology, 2 (**2010**) 213-219.

[16] I. F. Elashry, O. S. Faragallah, A. M. Abbas, S. El-Rabaie and F. E. Abd El-Samie, A new method for encrypting images with few details using Rijndael and RC6 block ciphers in the electronic code book mode, Information security journal: A global perspective, 21 (**2012**) 193-205.

[17] R. Awangga, Peuyeum: A Geospatial URL Encrypted Web Framework Using Advance Encryption Standard-Cipher Block Chaining Mode, IOP Conference Series: Earth and Environmental Science, IOP Publishing, **2018**, pp. 012055.
[18] H. M. Heys, Analysis of the statistical cipher feedback mode of block ciphers, IEEE Transactions on Computers, 52 (**2003**) 77-92.
[19] J. Jaffe, A first-order DPA attack against AES in counter mode with unknown initial counter, International Workshop on Cryptographic Hardware and Embedded Systems, Springer, **2007**, pp. 1-13.
[20] B. Schneier and D. Whiting, A Performance Comparison of the Five AES Finalists, AES Candidate Conference, **2000**, pp. 123-135.
[21] J. Daemen and V. Rijmen, The design of Rijndael, Springer, **2002**.
[22] C. Paar and J. Pelzl, Understanding cryptography: a textbook for students and practitioners, Springer Science & Business Media, **2009**.
[23] E. G. Ahmed, E. Shaaban and M. Hashem, Lightweight mix columns implementation for AES, Proceedings of the 9th WSEAS international conference on Applied informatics and communications (AIC'09), **2009**, pp. 253-258.
[24] M. A. Musa, E. F. Schaefer and S. Wedig, A Simplified AES Algorithm and its Linear and Differential Cryptanalyses, Cryptologia, 27 (**2003**) 148-177.
[25] D. Engels, X. Fan, G. Gong, H. Hu and E. M. Smith, Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices, Springer Berlin Heidelberg, Berlin, Heidelberg, **2010**, pp. 3-18.
[26] E. Savaş and Ç. K. Koç, Finite field arithmetic for cryptography, IEEE Circuits and Systems Magazine, 10 (**2010**) 40-56.
[27] 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, Springer International Publishing, Cham, **2014**, pp. 3-27.
[28] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks and L. Wingers, The SIMON and SPECK lightweight block ciphers, Proceedings of the 52nd Annual Design Automation Conference, Association for Computing Machinery, San Francisco, California, **2015**, pp. Article 175.
[29] M. Umair, Comparison of Symmetric Block Encryption Algorithms, ResearchGate, April, (**2017**).
[30] C. Manifavas, G. Hatzivasilis, K. Fysarakis and Y. Papaefstathiou, A survey of lightweight stream ciphers for embedded systems, Security and Communication Networks, 9 (**2016**) 1226-1246.
[31] J. Thakur and N. Kumar, DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis, International journal of emerging technology and advanced engineering, 1 (**2011**) 6-12.
[32] S. Simmons, Algebraic Cryptanalysis of Simplified AES∗, Cryptologia, 33 (**2009**) 305-314.
[33] A. Kak, AES: The Advanced Encryption Standard lecture notes on Computer and Network Security, Lecture8, **2017**.
[34] M. N. A. Wahid, A. Ali, B. Esparham and M. Marwan, A comparison of cryptographic algorithms: DES, 3DES, AES, RSA and blowfish for guessing attacks prevention, Journal Computer Science Applications and Information Technology, 3 (**2018**) 1-7.
[35] T.-H. Le, C. Canovas and J. Clédiere, An overview of side channel analysis attacks, Proceedings of the 2008 ACM symposium on Information, computer and communications security, **2008**, pp. 33-43.

[36] C. Rebeiro, D. Mukhopadhyay and S. Bhattacharya, An Introduction to Timing Attacks, Timing Channels in Cryptography: A Micro-Architectural Perspective, Springer International Publishing, Cham, **2015**, pp. 1-11.
[37] M. N. I. Khan, S. Bhasin, A. Yuan, A. Chattopadhyay and S. Ghosh, Side-Channel Attack on STTRAM Based Cache for Cryptographic Application, 2017 IEEE International Conference on Computer Design (ICCD), **2017**, pp. 33-40.
[38] J. Longo, E. De Mulder, D. Page and M. Tunstall, SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip, Springer Berlin Heidelberg, Berlin, Heidelberg, **2015**, pp. 620-640.
[39] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner and D. Smith-Tone, Report on post-quantum cryptography, US Department of Commerce, National Institute of Standards and Technology, **2016**.
[40] V. Mavroeidis, K. Vishi, M. D. Zych and A. Jøsang, The impact of quantum computing on present cryptography, arXiv preprint arXiv:1804.00200, (**2018**).
[41] M. Almazrooie, R. Abdullah, A. Samsudin and K. N. Mutter, Quantum Grover Attack on the Simplified-AES, Proceedings of the 2018 7th International Conference on Software and Computer Applications, Association for Computing Machinery, Kuantan, Malaysia, **2018**, pp. 204–211.
[42] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann and L. Uhsadel, A Survey of Lightweight-Cryptography Implementations, IEEE Design & Test of Computers, 24 (**2007**) 522-533.
[43] A. Mahboob and N. Ikram, Lookup table based multiplication technique for GF (2m) with cryptographic significance, IEE Proceedings-Communications, 152 (**2005**) 965-974.
[44] L. Hua and Z. Friggstad, An efficient architecture for the AES mix columns operation, 2005 IEEE International Symposium on Circuits and Systems, **2005**, pp. 4637-4640 Vol. 4635.
[45] R. Vimalathithan, D. Rossi, M. Omana, C. Metra and M. Valarmathi, Cryptanalysis of Simplified-AES Encrypted Communication, International Journal of Computer Science and Information Security, 13 (**2015**) 142.
[46] N. Hakiem, A. U. Priantoro, M. U. Siddiqi and T. H. Hasan, Generation of cryptographic one-to-many mapping IPv6 address using S-AES, Proceeding of the 3rd International Conference on Information and Communication Technology for the Moslem World (ICT4M) 2010, **2010**, pp. E13-E18.
[47] M. Khader, M. Alian, R. Hraiz and S. Almajali, Simplified AES algorithm for healthcare applications on Internet of Thing, 2017 8th International Conference on Information Technology (ICIT), **2017**, pp. 543-547.
[48] S. J. Manangi, P. Chaurasia and M. P. Singh, Simplified AES for Low Memory Embedded Processors, Global Journal of Computer Science and Technology, (**2010**).
[49] Simplified AES Source Codes, https://github.com/bariszorba/simplifiedAES (Date of Access: 2021-10-19)
[50] Ü. Çavuşoğlu, S. Kaçar, A. Zengin and I. Pehlivan, A novel hybrid encryption algorithm based on chaos and S-AES algorithm, Nonlinear Dynamics, 92 (**2018**) 1745-1759.
[51] L. Ertaul and S. K. Rajegowda, Performance analysis of CLEFIA, PICCOLO, TWINE Lightweight block ciphers in IoT environment, Proceedings of the International Conference on Security and Management (SAM), The Steering Committee of The World Congress in Computer Science, Computer …, **2017**, pp. 25-31.
[52] D. Page, Linkers and Assemblers, Practical Introduction to Computer Architecture, Springer London, London, **2009**, pp. 397-450.

[53] J. Yiu, Chapter 3 - Technical Overview, in: J. Yiu (Ed.) The Definitive Guide to ARM® CORTEX®-M3 and CORTEX®-M4 Processors (Third Edition), Newnes, Oxford, **2014**, pp. 57-73.
[54] K. E. Murray and V. Betz, Tatum: Parallel Timing Analysis for Faster Design Cycles and Improved Optimization, 2018 International Conference on Field-Programmable Technology (FPT), **2018**, pp. 110-117.
[55] V. Manohararajah, S. D. Brown and Z. G. Vranesic, Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25 (**2006**) 2331-2340.
[56] C. Pei, Y. Xiao, W. Liang and X. Han, Trade-off of security and performance of lightweight block ciphers in Industrial Wireless Sensor Networks, EURASIP Journal on Wireless Communications and Networking, 2018 (**2018**) 117.
[57] Q. Ge, Y. Yarom, D. Cock and G. Heiser, A survey of microarchitectural timing attacks and countermeasures on contemporary hardware, Journal of Cryptographic Engineering, 8 (**2018**) 1-27.
[58] O. Lo, W. J. Buchanan and D. Carson, Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA), Journal of Cyber Security Technology, 1 (**2017**) 88-107.
[59] M. Katagi and S. Moriai, Lightweight cryptography for the internet of things, Sony Corporation, 2008 (**2008**) 7-10.
[60] K. Atasu, L. Breveglieri and M. Macchetti, Efficient AES implementations for ARM based platforms, Proceedings of the 2004 ACM symposium on Applied computing, Association for Computing Machinery, Nicosia, Cyprus, **2004**, pp. 841–845.
[61] G. Bansod, N. Raval and N. Pisharoty, Implementation of a New Lightweight Encryption Design for Embedded Security, IEEE Transactions on Information Forensics and Security, 10 (**2015**) 142-151.
[62] M. P. H. Dixit, U. L. Bombale and M. V. B. Patil, Comparative Implementation of Cryptographic Algorithms on ARM Platform.
[63] T. B. S. Reis, D. F. Aranha and J. López, PRESENT Runs Fast, Springer International Publishing, Cham, **2017**, pp. 644-664.
[64] D. Engels, M.-J. O. Saarinen, P. Schweitzer and E. M. Smith, The Hummingbird-2 lightweight authenticated encryption algorithm, International Workshop on Radio Frequency Identification: Security and Privacy Issues, Springer, **2011**, pp. 19-31.
[65] Z. Gong, S. Nikova and Y. W. Law, KLEIN: A New Family of Lightweight Block Ciphers, Springer Berlin Heidelberg, Berlin, Heidelberg, **2012**, pp. 1-18.
[66] T. Suzaki, K. Minematsu, S. Morioka and E. Kobayashi, Twine: A lightweight, versatile block cipher, ECRYPT Workshop on Lightweight Cryptography, **2011**.
[67] Open Cores, https://opencores.org/ (Date of Access:
[68] AES VHDL Core, https://github.com/hadipourh/AES-VHDL (Date of Access: 2021-10-19)
[69] Camellia VHDL Core, https://opencores.org/projects/camellia-vhdl (Date of Access: 2021-10-19)
[70] DES VHDL Core, https://github.com/mitkof6/DataEncryptionStandard (Date of Access: 2021-10-19)
[71] Noekeon VHDL Core, https://opencores.org/projects/noekeoncore (Date of Access: 2021-10-19)
[72] PRESENT VHDL Core, https://opencores.org/projects/present (Date of Access: 2021-10-19)

[73] Tiny Encryption Algorithm (TEA) VHDL Core, https://opencores.org/projects/tiny_encryption_algorithm (Date of Access: 2021-10-19)
[74] XTEA VHDL Core, https://opencores.org/projects/xteacore (Date of Access: 2021-10-19)
[75] W. Feng, J. Greene and A. Mishchenko, Improving FPGA Performance with a S44 LUT Structure, Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Association for Computing Machinery, Monterey, CALIFORNIA, USA, **2018**, pp. 61–66.

APPENDICES

Appendix 1 – Commands While Executing S-AES on QEMU

Commands	Platform	
	RISC-V	**ARM**
Compiling the algorithm with 64-bit architectures	riscv64-linux-gnu-gcc -static -o s_aes_riscv s_aes.c	aarch64-linux-gnu-gcc -static -o saes64 s_aes.c
Running the object file on rv64 core of qemu-riscv64 library	qemu-riscv64 -cpu rv64 s_aes_riscv	qemu-aarch64 -L /usr/aarch64-linux-gnu ./s_aes_arm
Disassembling the object file	qemu-riscv64 -d in_asm -D log_riscv -cpu rv64 s_aes_riscv	aarch64-linux-gnu-objdump -D s_aes_arm