# FPGA-FRIENDLY COMPACT AND EFFICIENT AES-LIKE 8X8 S-BOX

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

AHMET MALAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF CYBER SECURITY

JUNE 2023

**FPGA-Friendly Compact and Efficient AES-like 8x8 S-Box**

submitted by **AHMET MALAL** in partial fulfillment of the requirements for the degree of **Master of Science in Cyber Security Department, Middle East Technical University** by,

Prof. Dr. Banu GÜNEL KILIÇ _____
Dean, **Graduate School of Informatics**

Assoc. Prof. Dr. Cihangir TEZCAN _____
Head of Department, **Cyber Security**

Assoc. Prof. Dr. Cihangir TEZCAN _____
Supervisor, **Cyber Security, METU**

**Examining Committee Members:**

Assist. Prof. Dr. Aybar Can ACAR
Health Informatics, METU _____

Prof. Dr. Ali Aydın SELÇUK
Computer Engineering, TOBB _____

Assoc. Prof. Dr. Cihangir TEZCAN
Cyber Security, METU _____

**Date: 22.06.2023**

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

**Name, Surname: AHMET MALAL**

**Signature :**

## ABSTRACT

# FPGA-FRIENDLY COMPACT AND EFFICIENT AES-LIKE 8X8 S-BOX

MALAL, AHMET
M.S., Department of Cyber Security
Supervisor: Assoc. Prof. Dr. Cihangir TEZCAN

June 2023, 65 pages

One of the main layers in the Advanced Encryption Standard (AES) is the substitution layer, where an $8 \times 8$ S-Box is used $16$ times. The substitution layer provides confusion and makes the algorithm resistant to cryptanalysis techniques. Therefore, the security of the algorithm is also highly dependent on this layer. However, the cost of implementing $8 \times 8$ S-Box on FPGA platforms is considerably higher than other layers of the algorithm. In 2005, Canright used different extension fields to represent AES S-Box to get FPGA-friendly compact designs.

We use the same optimization methods that Canright used to optimize AES S-Box on hardware platforms. Our purpose is not to optimize AES S-Box; we aim to create another an $8 \times 8$ S-Box which is strong and compact enough for FPGA platforms. We create an 8x8 S-Box using the inverse field operation as in the case of AES S-Box. We use another primitive polynomial to represent the finite field and get an FPGA-friendly compact and efficient an $8 \times 8$ S-Box. The finite field we propose provides the same level of security against cryptanalysis techniques with a $3.125\%$ less gate-area on Virtex-7 and Artix-7 FPGAs compared to Canright's results. Moreover, our proposed S-Box requires $11.76\%$ less gate on Virtex-4 FPGAs. The enhancements made to the gate area offer advantages to IoT devices with limited resources, enabling increased duplication of the S-Box for improved algorithm parallelism. Therefore, we claim that our proposed S-Box is more compact and efficient than AES S-Box.

Keywords: AES, Rijndael S-Box, Compact S-Box, Finite Field, Group Isomorphisms

# ÖZ

## FPGA DOSTU KOMPAKT VE VERİMLİ AES BENZERİ 8X8 S-KUTUSU

MALAL, AHMET
Yüksek Lisans, Siber Güvenlik Bölümü
Tez Yöneticisi: Doç. Dr. Cihangir TEZCAN

Haziran 2023, 65 sayfa

Gelişmiş Şifreleme Standardındaki (AES) ana katmanlardan biri, $8 \times 8$ S-Kutusunun $16$ kez kullanıldığı BaytDeğiştir katmanıdır. BaytDeğiştir katmanı karışıklık sağlar ve algoritmayı kriptanaliz tekniklerine dirençli hale getirir. Bu nedenle, algoritmanın güvenliği de büyük ölçüde bu katmana bağlıdır. Ancak, $8 \times 8$ S-Kutusunun FPGA platformlarında uygulamanın maliyeti, algoritmanın diğer katmanlarına göre oldukça yüksektir. Algoritmada S-Kutuları tekrar tekrar kullanıldığından, algoritmanın maliyeti büyük ölçüde bu katmandan gelmektedir. 2005 yılında Canright, FPGA dostu kompakt tasarımlar elde etmek için AES S-Kutusunu farklı matematiksel alanlarda ifade etti.

Bu çalışmada, Canright'ın AES S-Kutusunu donanım platformlarında optimize etmek için kullandığı matematiksel yöntemlerinin aynısını kullandık. Amacımız AES S-Kutusunu daha da optimize etmek değil; FPGA platformları için yeterince güçlü ve kompakt olan başka bir $8 \times 8$ S-Kutusu oluşturmaktı. AES S-Kutusunun yapısında olduğu gibi ters alan işlemini kullanarak $8 \times 8$ S-Kutusu oluşturduk. Sonlu alanı temsil etmek için başka bir indirgenemez polinom kullanıp FPGA dostu kompakt ve verimli bir $8 \times 8$ S-Box elde ettik. Önerdiğimiz sonlu alan, Canright'ın sonuçlarına kıyasla Virtex-7 ve Artix-7 FPGA'larda $3.125\%$ daha az kapı alanıyla kriptanaliz tekniklerine karşı aynı düzeyde güvenlik sağladığını tespit ettik. Ayrıca, önerdiğimiz S-Kutusu, Virtex-4 FPGA'larda $11.76\%$ daha az kapı ile gerçeklenebiliyor. Bu kapladığı alan iyileştirmeleri, kaynak kısıtlaması olan IoT cihazları için oldukça önemlidir ve algoritma paralelliği için S-Kutusunun daha fazla kopyalanarak kullanılmasına olanak

sağlar. Bu nedenle, önerdiğimiz S-Kutusunun AES S-Kutusundan daha kompakt ve verimli olduğunu iddia ediyoruz.

Anahtar Kelimeler: AES, Rijndael S-Kutusu, Kompakt S-Kutusu, Sonlu Alan, Grup İzomorfizmalar

***to my lovely wife***

## ACKNOWLEDGMENTS

I want to extend my heartfelt appreciation to my thesis supervisor, Assoc. Prof. Cihangir TEZCAN. His invaluable guidance and support during my research deserve sincere gratitude. He has offered expert advice, constructive criticism, and constant encouragement to push my limits. Writing this thesis under his supervision has been an absolute pleasure.

I would like to express my gratitude to Asst. Prof. Dr. Aybar Can ACAR and Prof. Dr. Ali Aydın SELÇUK for their valuable participation and contributions as members of the thesis defense jury.

I express my gratitude to Jérémy Jean, Carl Schneider, Diana Maimut, and Florian Delporte for generously sharing their Feistel, SPN, DES, and Mode of Operations figures through *TikZ for Cryptographers.*

I am immensely grateful to Prof. Debdeep MUKHOPADHYAY for his invaluable YouTube guidance, which significantly influenced the completion of my master's thesis.

ASELSAN Inc. provided me with the flexibility to balance my work and academic responsibilities, and for that, I am truly grateful. The knowledge and skills I have gained from my job have been invaluable in shaping my research and analysis. I am grateful for the constant support and encouragement from my colleagues and friends.

I am deeply grateful to my family for their love and support. Their motivation and positivity helped me navigate through the ups and downs of my academic life.

Lastly, my deepest gratitude goes to my wife, Şeyma MURATOĞLU MALAL, for her unwavering support throughout my master's thesis journey. Her unconditional love and constant encouragement have been instrumental in my achievements. I am forever grateful for her selfless sacrifices, dedicating her time, effort, and energy to support me throughout my academic journey. This thesis would not have been possible without her.

Thank you all for being a part of my journey and contributing to completing this thesis successfully.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# LIST OF ALGORITHMS

ALGORITHMS

## LIST OF ABBREVIATIONS

| | |
|---|---|
| AES | Advanced Encryption Standard |
| ASIC | Application Specific Integrated Circuit |
| BCT | Boomerang Connectivity Table |
| BRAM | Block Random Access Memory |
| CBC | Cipher Block Chaining |
| CFB | Cipher Feedback |
| CTR | Counter Mode |
| DDT | Difference Distribution Table |
| DES | Data Encryption System |
| ECB | Electronic Code Block |
| FPGA | Field Programmable Gate Array |
| GCM | Galois Counter Mode |
| IOT | Internet of Things |
| LAT | Linear Approximation Table |
| LUT | Look-up Table |
| MAC | Message Authentication Code |
| OFB | Output Feedback |
| RAM | Random Access Memory |
| SHA | Secure Hash Algorithm |

# CHAPTER 1

# INTRODUCTION

## 1.1 Cryptography

Cryptography, or cryptology, is the study of securing communication in the existence of third parties. The study aims to protect the transaction of data from others. In more detail, cryptography constructs algorithms and protocols to protect sensitive data from adversaries. The study of cryptography or cryptology is born thousands of years ago. Secure communication has always been crucial for people and states throughout history. Sometimes it changed the course of wars and caused the collapse of great states. Until the 1970s, many cryptographic systems were developed, most of which were pen-and-paper methods. These systems existed as works of classical cryptography. They evolved into insecure systems with the advancement of computer science. After the 1970s, modern cryptography was born. Modern cryptographic algorithms and protocols provide security against computers. Cryptographers have designed many cryptographic algorithms until today. However, it was realized in the late 1990s that modern cryptography is vulnerable to quantum computers, just like classical cryptography is vulnerable to classical computers. Quantum computers are machines that consist of quantum bits or simply qubits. The working mechanisms of these computers are quite different from classical computers.

A quantum computer with enough qubits can break many modern cryptographic algorithms. Such quantum computers have still not been produced. However, rapid quantum physics and mechanics developments are expected to produce such computers soon. Therefore, the National Institute of Standard Technology (NIST) announced a competition to develop post-quantum algorithms in 2015. Our study is one of the topics of modern cryptography. Therefore we did not mention classical cryptography and post-quantum cryptography in this thesis.

Many security concepts are solved by cryptography, but three of them are considered the most important. These three concepts are also called the CIA triad.

- **Confidentiality:** In this scenario, the adversary is able to intercept the communication channel but lacks the ability to comprehend or decipher the content of the communication.

- **Integrity:** In this scenario, the adversary has the capability to eavesdrop on the communication channel but lacks the ability to tamper with or manipulate the content of the communication.
- **Availability:** Sensitive data remains inaccessible to unauthorized parties, ensuring its protection and confidentiality.

It is good to know the following terms before going into detail.
*Plaintext:* The clear data or message.
*Ciphertext:* The secret message is in a form that humans cannot read.
*Encryption:* The process in which plaintext is turned into ciphertext.
*Decryption:* The process in which ciphertext is turned into plaintext.
*Key:* The secret asset used for encryption and decryption.

Modern cryptography is commonly categorized into two subsections based on the key mechanisms employed. The first is symmetric cryptography, which involves the use of a single key for both the encryption and decryption processes. The second is asymmetric cryptography, which utilizes distinct keys for encryption and decryption operations.

### 1.1.1 Symmetric Cryptography

In order to securely send a message from Alice to Bob, they must first agree on a shared secret key using symmetric cryptography. They must hold the same key before communication. Alice uses one of the symmetric algorithms and encrypts the message with a secret key which is already shared with Bob. Then Alice produces the ciphertext and sends it to Bob. Bob uses the secret key, decrypts the ciphertext, and gets the plaintext. Figure 1 demonstrates the encryption and decryption process of a symmetric cipher.

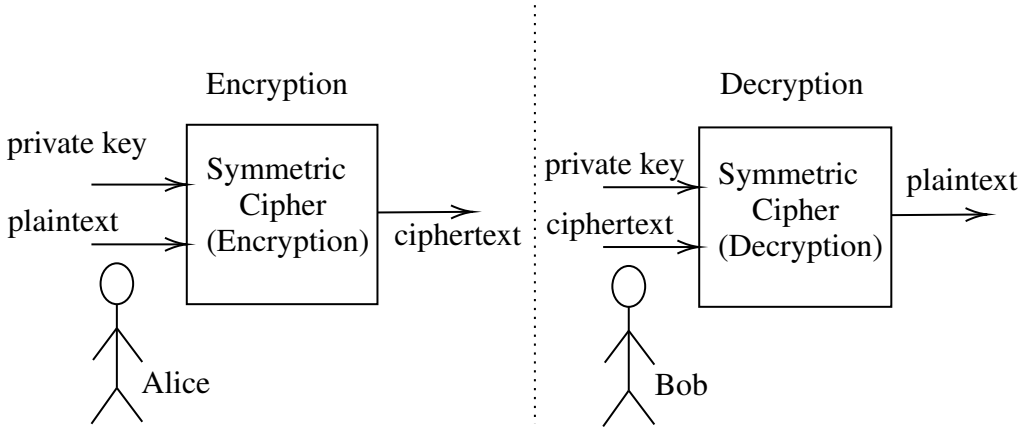


Figure 1: Enryption and Decryption Scheme of Symmetric Ciphers

**Some of well-known symmetric ciphers:**

- DES
- TripleDES (3DES)
- RC4
- AES
- Blowfish
- IDEA
- PRESENT
- ASCON

Advantageous and disadvantages of the symmetric schemes are listed below:

**Advantageous:**

- They need just one key to encrypt and decrypt data.
- They provide confidentiality.
- They are fast compared to asymmetric algorithms.
- They have small key sizes.
- They are hardware oriented, easy to implement in hardware platforms.

**Disadvantageous:**

- It is difficult to share private keys between parties.
- They have key-storage problem.
- They do not provide authenticity and non-repudiation.

### 1.1.2 Asymmetric Cryptography

Asymmetric cryptographic algorithms, or public key cryptography, employ two distinct keys for encryption and decryption. In a public key cryptography system, Alice generates a pair of keys: a private key and a public key. She shares her public key with Bob, who uses it to encrypt a message. Bob sends the encrypted message to Alice. Alice, being the only one with access to the corresponding private key, can decrypt the message and read its contents securely. The encryption and decryption scheme of an asymmetric cipher is shown in Figure 2.

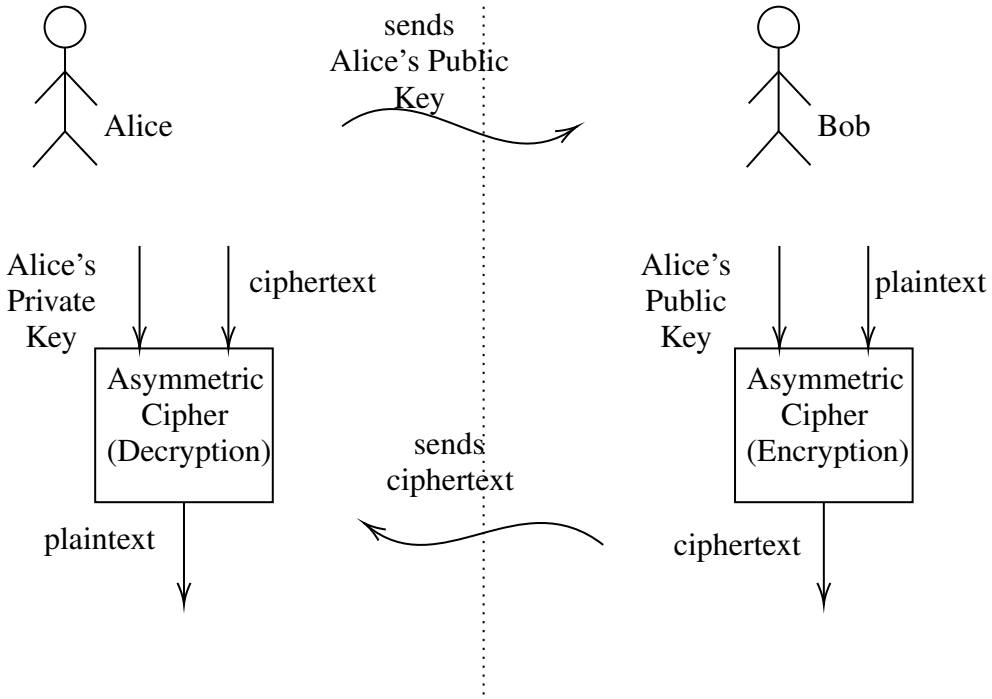


Figure 2: Encryption and Decryption Scheme of Asymmetric Ciphers

**Some of well-known asymmetric ciphers:**

- Diffie-Hellman
- Elliptic Curve Cryptography
- El-Gamal
- DSA
- RSA
- ECDSA

Advantageous and disadvantages of the asymmetric schemes are listed below:

**Advantageous:**

- They provide confidentiality, authenticity and non-repudiation altogether.
- They do not have problem with sharing keys.

**Disadvantageous:**

- They require two different keys, one for encryption and one for decryption.
- They are slow compared to symmetric algorithms.
- They have large key sizes.
- It is quite difficult to implement them in hardware platforms.

## 1.2 Block Ciphers

In the study of symmetric cryptography, block ciphers are examined in detail. Block ciphers are a type of symmetric cryptography where the same key is used for encryption and decryption. Messages are divided into fixed-length blocks, typically 64-bit or 128-bit in size. With a block size of 128-bit, the block cipher algorithm takes a 128-bit input along with the private key and generates a 128-bit output. It is a permutation function indexed by the secret key. The block size determines the number of outputs the algorithm might generate. Therefore, the block size directly affects the security of the algorithm. Having a large block size is better in terms of security. Block ciphers use two common structures: Feistel network and Substitution-Permutation Network (SPN).

### 1.2.1 Feistel Ciphers

Horst Feistel and Don Coppersmith proposed the Feistel structure in 1973. Lucifer cipher was the first cryptographic algorithm based on the Feistel structure. The structure of the block cipher is designed to create robust pseudorandom permutations by utilizing a permutation function F and the XOR operation. The F function varies depending on the specific algorithm. The plaintext is divided into two equal blocks, and in each round, the F function is applied to one block while the other block is XORed with the output of the F function. These operations are repeated a predetermined number of times, known as the round number of the algorithm. As function F, the round number also differs in each algorithm. Inverse of the operations is the most crucial advantage of Feistel networks compared to other networks. Even if the function F is not invertible, the entire Feistel network is invertible. The Feistel structure of the DES cipher is a design that ensures similarity between the encryption and decryption processes. Figure 3 illustrates the Feistel structure of the DES cipher.

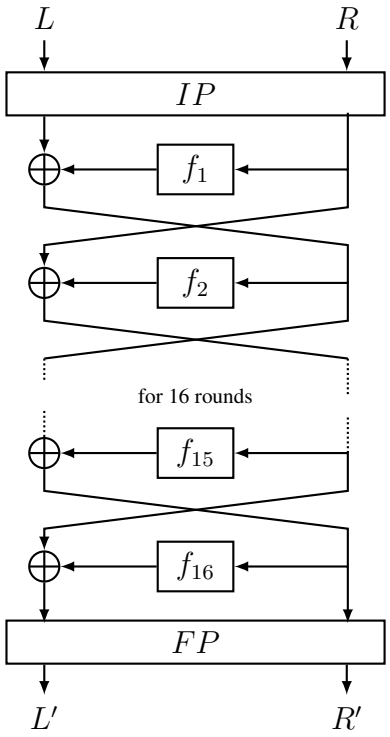


Figure 3: The Feistel structure figure, created by Jeremy Jean, can be found in [Jean, 2015] and is publicly accessible.

After 1973, this structure has been used in many cryptographic algorithms. The most common algorithms that have Feistel structure are listed below:

- DES
- Triple DES
- Blowfish
- Twofish
- CLEFIA
- Camellia
- Lucifer
- MARS
- MISTY1
- RC2
- RC5
- RC6

### 1.2.2 Substitution–Permutation Networks

This structure takes data and a key and performs operations on them. SPN structure consists of three layers: Substitution, Permutation, and Add Round Key. The substitution layer adds confusion in the encryption process, while the permutation layer contributes to diffusion. Through the substitution layer, each bit in the ciphertext output relies on every bit of the plaintext input. The positions of bits are replaced in the permutation layer to provide diffusion. In order to be the network invertible, the substitution layer must be invertible. The entire network becomes non-invertible if a non-invertible function is used in the substitution layer. Round key is used in Add Round Key layer. The decryption of algorithm is done by performing layers in reverse order with the inverse functions. Figure 4 depicts the structure of substitution-permutation networks.

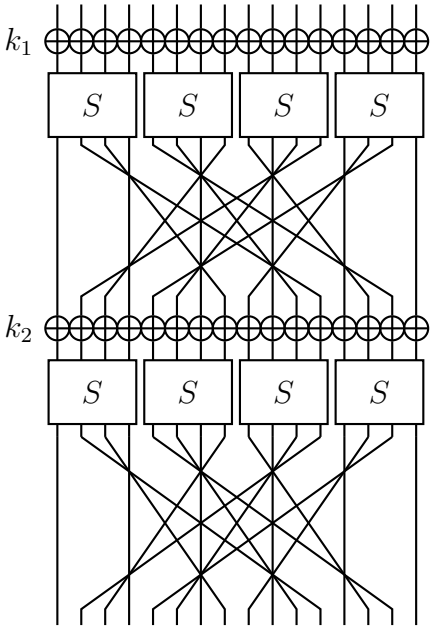


Figure 4: SPN Structure, created by Florian Delporte, can be found in [Delporte, 2016] and is publicly accessible.

The most common algorithms that have SPN structure are listed below:

- AES (The Rijndael)
- 3-Way
- PRESENT
- SAFER
- SHARK
- Square
- Kalyna
- Square
- Kuznyechik

### 1.3 Modes of Operation

The block ciphers encrypt or decrypt only a single data block. Modes of operations are used for encrypting or securely decrypting large amounts of data using block ciphers. According to requirements, different modes of operation are designed. Electronic Code Block (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), and Cipher Feedback (CFB) were specified with FIBS PUB 81 in 1981. Counter mode (CTR) and Galois Counter Mode (GCM) were published respectively in 2001 and 2007 by the National Institute of Standards (NIST) with SP800-38A and SP800-38D. A more detailed explanation of the modes of operation is mentioned below.

- **Electronic Code Block (ECB)**

The ECB (Electronic Codebook) mode is a straightforward encryption mode where a large message is divided into small blocks, and each block is encrypted individually. Parallel encryption and decryption are possible since each block is separated. The

disadvantage of this mode is that ECB maps identical plaintexts to identical ciphertexts. Therefore, the method does not provide diffusion very well. It is not capable of hiding patterns. This method is not recommended for cryptographic purposes and is no longer secure. Encryption scheme of electronic code block is shown in Figure 5.

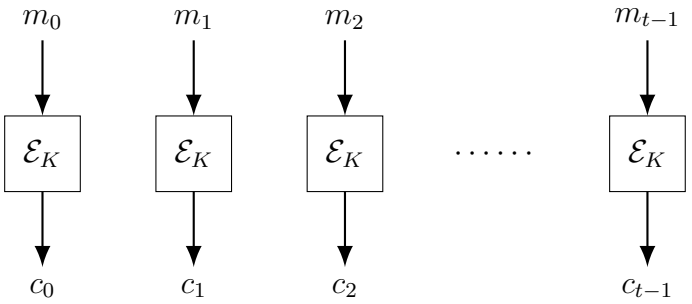


Figure 5: Encryption Scheme of Electronic Code Block, created by Jeremy Jean, can be found in [Jean, 2015] and is publicly accessible.

- **Cipher Block Chaining (CBC)**

This mode was designed in 1976 by four researchers [William et al., 1978]. Before encryption, the previous ciphertext is XORed with the block of data. The initialization vector (IV) is used in encryption or decryption to process the initial block. Because of the chaining structure, parallel encryption is not possible. However, parallel decryption is possible since XORed values are already in the ciphertext. Having a single bit of error propagates the rest of the operations. Therefore, a single point of failure causes invalid encryption. Encryption scheme of cipher block chaining is shown in Figure 6.

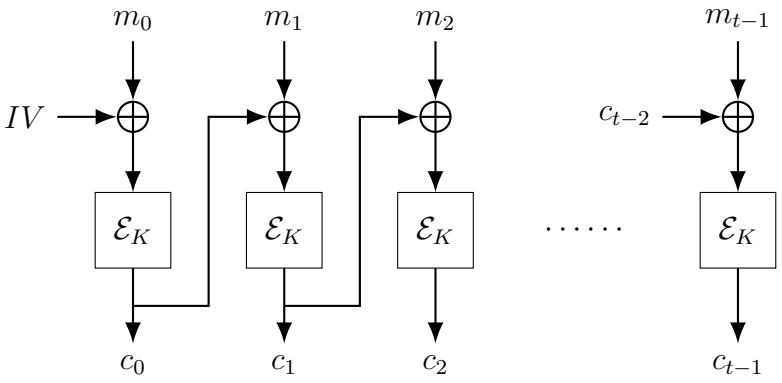


Figure 6: Encryption Scheme of Cipher Block Chaining, created by Jeremy Jean, can be found in [Jean, 2015] and is publicly accessible.

- **Output Feedback (OFB)**

In this mode, a message-sized keystream is produced. The initialization vector is encrypted with the secret key in the first step, and output is produced. This output is

then used as input for the subsequent block cipher operation. Then the produced output is used as the input of the following block cipher. By continuing this process, the keystream is generated. The message is XORed with the produced keystream synchronously. Unfortunately, this mode does not provide either parallel encryption or parallel decryption. Encryption scheme of output feedback mode is shown in Figure 7.

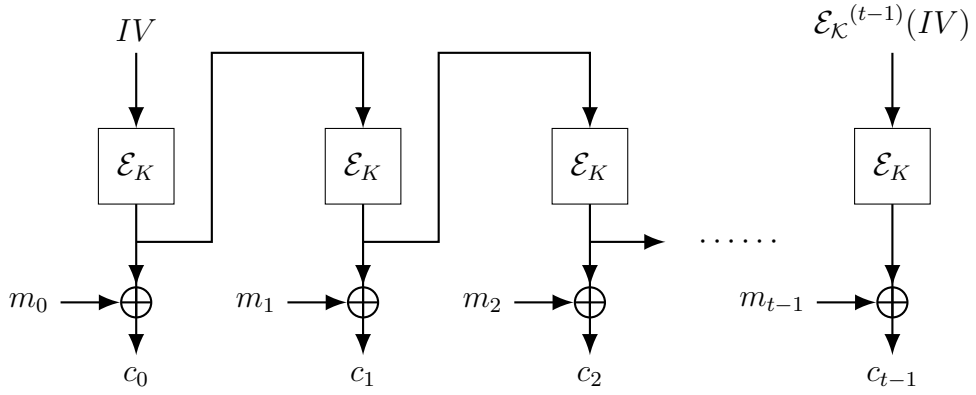


Figure 7: Encryption Scheme of Output Feedback Mode, created by Theodor Schneider, can be found in [Schneider, 2016] and is publicly accessible.

- **Cipher Feedback (CFB)**

In this mode, the initialization vector is encrypted using the secret key. The resulting ciphertext is then XORed with the first block of the plaintext to generate the first block of ciphertext. For subsequent blocks, the previously generated ciphertext is used as input to the block cipher. The output of the block cipher is XORed with the corresponding block of the plaintext to produce the next block of ciphertext. This process repeats until the final block of the data is processed. The decryption process is the same as the encryption process. CFB does not provide parallel encryption; however, it provides parallel decryption. The single-bit error will cause invalid encryption or decryption to the rest of the blocks. Encryption scheme of cipher feedback mode is shown in Figure 8.

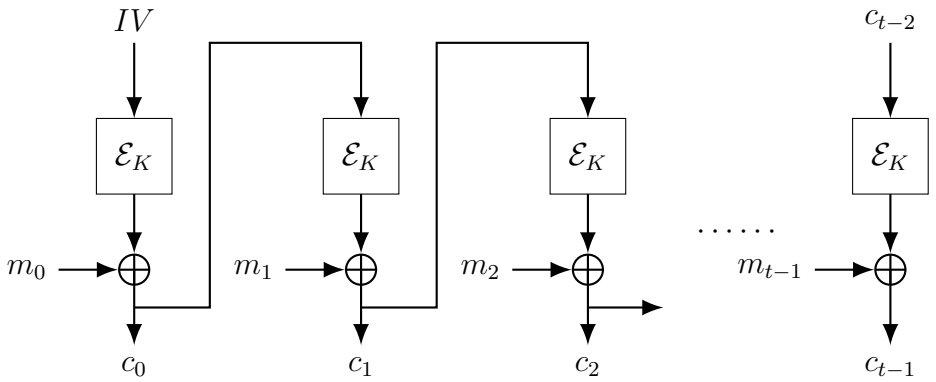


Figure 8: Encryption Scheme of Cipher Feedback Mode, created by Theodor Schneider, can be found in [Schneider, 2016] and is publicly accessible.

- **Counter Mode (CTR)**

The counter mode was proposed in 1979 [Lipmaa et al., 2000]. This mode is similar to OFB in creating keystream while encryption or decryption processes. Rather than OFB, CTR uses incremental initialization vector while generating keystream. Every block is XORed with the block cipher output, which uses incremental IV. Parallel encryption and decryption are possible. The error does not propagate on different blocks since no chaining mechanism exists. Encryption scheme of counter mode is shown in Figure 9.

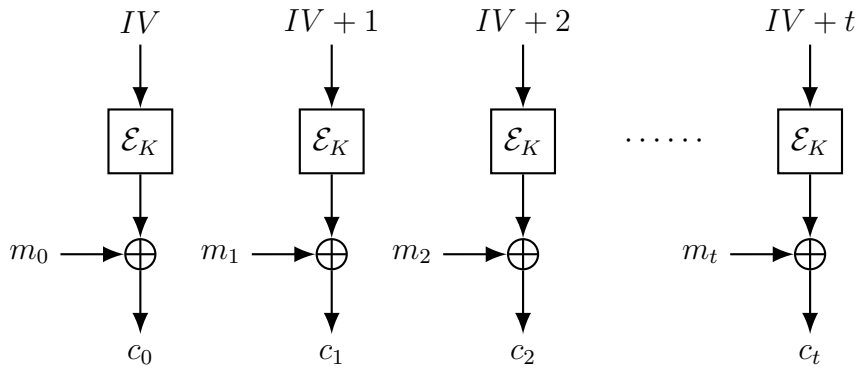


Figure 9: Encryption Scheme of Counter Mode, created by Jeremy Jean, can be found in [Jean, 2015] and is publicly accessible.

- **Galois Counter Mode (GCM)**

McGrew and Viega proposed the Galois counter mode in 2004. It is the first mode of operation that provides integrity and confidentiality. GCM takes the parallel mechanism of CTR mode and the chaining structure of CBC at the same time. A 128-bit authentication tag is produced with ciphertext at the end of the encryption processes. The message is encrypted with CTR mode, and an authentication tag is produced with a chaining structure. The 128-bit Galois multiplication is used for generating the tag.

Instead of plaintext, the ciphertext is used in decryption processes. At the end of the decryption, plaintext and another tag are produced. If the produced tag is the same as the taken tag, then the integrity and confidentiality of the encryption and decryption processes are guaranteed [McGrew and Viega, 2004]. Encryption scheme of Galois counter mode is shown in Figure 10.

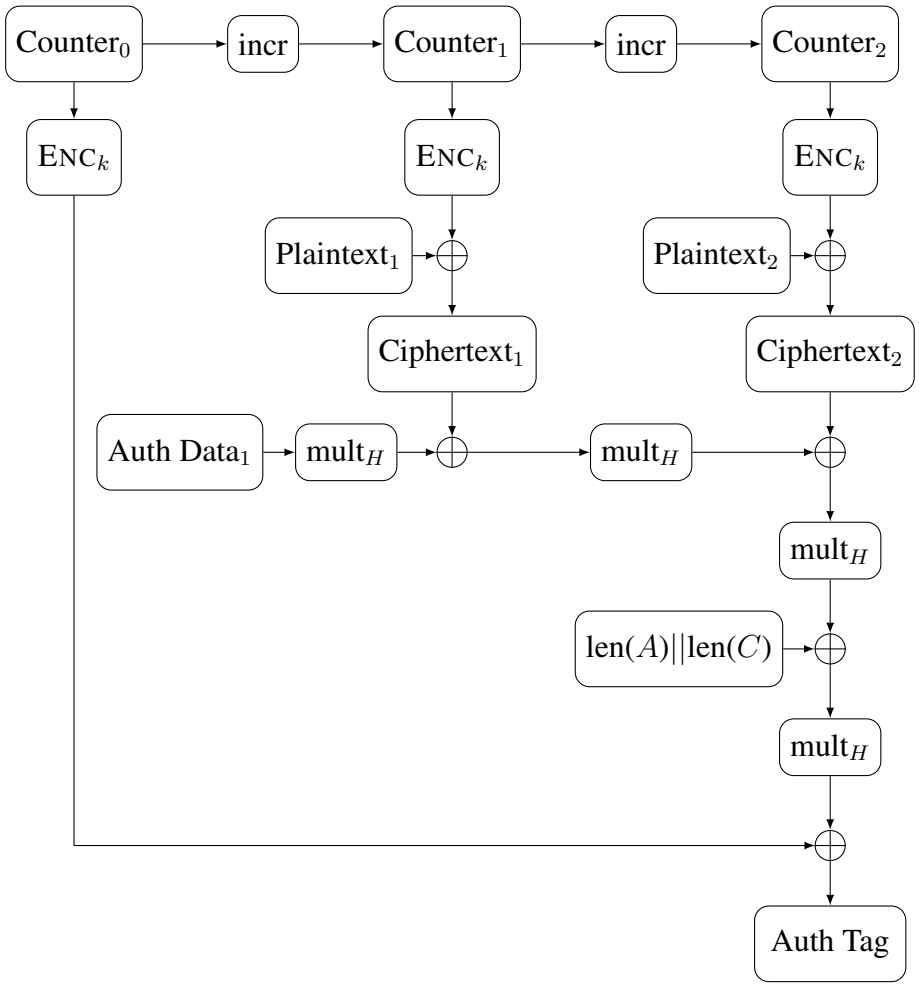


Figure 10: Encryption Scheme of Galois Counter Mode, created by Diana Maimut, can be found in [Maimut, 2017] and is publicly accessible.

### 1.4 Advanced Encryption Standard Competition

NIST announced a competition to choose a block cipher algorithm to be the successor of the Data Encryption Standard (DES) on January 2, 1997. NIST decided the specification of the algorithm. The block size of the algorithm was chosen to be 128-bit. The algorithm was supposed to have three modes that have different key sizes. The key sizes are chosen as 128, 196, and 256 bits. The competition initially began with fifteen candidates submitted from several countries.

The list of the submitted candidate algorithms (The finalists are highlighted.) :

- CAST-256
- CRYPTON
- **Rijndael**
- DEAL
- DFC
- **Serpent**
- E2
- FROG
- **RC6**
- HPC
- LOKI97
- MAGENTA
- **MARS**
- SAFER+
- **Twofish**

Vulnerabilities were detected in some candidates, and some were eliminated due to poor performance. NIST eliminated ten of fifteen candidates and announced five finalist algorithms. Rijndael, Serpent, Twofish, RC6, and MARS are announced as the finalists of the competition. After three rounds of elimination stage, the Rijndael algorithm was selected as the winner in April 2000, called later the Advanced Encryption Standard.

### 1.5 The AES Winner: Rijndael

The Rijndael algorithm was designed by Joan Daemon and Vincent Rijmen in 1997. The Rijndael algorithm is chosen thanks to its strong performance on different platforms and security resistance against cryptographic attacks. Since many cryptographers and researchers worked and analyzed the algorithm in the competition process, several papers and articles have been published about the Rijndael algorithm. So many designs and implementation strategies have been proposed with different metrics until today.

Unlike DES, AES does not based on the Feistel network. The structure of AES is designed with an SPN cipher. The algorithm consists of substitution, permutation, mixColumn, and addRoundKey layers [Daemen and Rijmen, 2002]. More detailed information about the Rijndael algorithm is given in the next chapter. The substitution layer, which uses S-box, is the only non-linear layer of the algorithm. Therefore the chosen method for the substitution layer must satisfy many criteria to be strong against attacks. The algorithm's security depends highly on the choice of S-box [Knudsen et al., 2004]. The S-box used by Rijndael proved to be strong against cryptographic attacks. According to Knudsen, there is no solid attack against the Rijndael algorithm [Knudsen et al., 2004].

Many researchers studied to implement efficient versions of the Rijndael S-box. Different approaches are proposed with different metrics. The gate area of Rijndael S-box is optimized for hardware platforms [Canright, 2005]. Since the S-box of Rijndael is strong and optimized by many researchers for different platforms, its popularity is increased. Many cryptographic algorithms used the Rijndael S-box in the following years.

The algorithms that use the Rijndael S-box:

- AES
- ARIA
- Grand Cru
- Panama

## 1.6 Our Contribution

We focused on the compact implementation of the substitution layer of the Rijndael algorithm. As we mentioned above, there are many studies on optimizing the Rijndael S-box for hardware platforms. In this thesis, we proposed another S-box that is strong as the Rijndael S-box and can be implemented with fewer gates on Field Programmable Gate Arrays (FPGAs). Cryptographers who use the Rijndael S-box in their algorithms can use our S-box since we proposed a more compact new S-box with the same level of security that the Rijndael provides.

In this chapter, basic cryptographic information was given. A detailed explanation of the Rijndael cipher was mentioned in Chapter 2. The implementation details of the compact S-box were explained in Chapter 3. In Chapter 4, our studies and the proposed S-box were discussed. In the last chapter, we concluded our thesis.

# CHAPTER 2

# ADVANCED ENCRYPTION STANDARD (AES)

## 2.1 Mathematical Preliminaries

AES uses a byte as a basic unit for processing. **A byte** consists of eight bits. The AES algorithm's input and output are 128-bits, equal to 16 bytes. AES supports three different key sizes, which are 16 bytes, 24 bytes and 32 bytes. The bytes are represented as the concatenation of bit values from the most significant to the least significant bits. Let *a* be a byte, then *a* is represented as $\{a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0\}$. With a polynomial representation, each byte is represented as finite field elements.

$$\sum_{i=0}^{7} a_i x^i = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

The hexadecimal notation is also widely used for representing bytes. **A nibble** consists of four bits. Two nibbles can be used to represent a byte. *0x* is used before hexadecimal notation of bytes. For example 10100011 identifies the element $x^7 + x^5 + x + 1$. The first nibble is "***a***" and the last nibble is "***3***". 10100011 is represented as *0xa3* in hexadecimal notation. Table 1 shows the hexadecimal representations of nibbles.

Table 1: Hexadecimal Representation of Nibbles

| Bit Patterns | Hex | Bit Patterns | Hex | Bit Patterns | Hex | Bit Patterns | Hex |
|---|---|---|---|---|---|---|---|
| 0b0001 | 0x1 | 0b0101 | 0x5 | 0b1001 | 0x9 | 0b1101 | 0xd |
| 0b0000 | 0x0 | 0b0100 | 0x4 | 0b1000 | 0x8 | 0b1100 | 0xc |
| 0b0010 | 0x2 | 0b0110 | 0x6 | 0b1010 | 0xa | 0b1110 | 0xe |
| 0b0011 | 0x3 | 0b0111 | 0x7 | 0b1011 | 0xb | 0b1111 | 0xf |

In AES (Advanced Encryption Standard), each round operates on a 16-byte block of data. This block is organized as a two-dimensional array of bytes known as **the state**. The state is composed of four columns and four rows, with each column containing four bytes.State array of AES is represented in Table 2.

Table 2: State Array of AES

| | | | |
|---|---|---|---|
| $s_0$ | $s_4$ | $s_8$ | $s_{12}$ |
| $s_1$ | $s_5$ | $s_9$ | $s_{13}$ |
| $s_2$ | $s_6$ | $s_{10}$ | $s_{14}$ |
| $s_3$ | $s_7$ | $s_{11}$ | $s_{15}$ |

The columns and rows of state is represented as follows ( $c_i$'s denotes columns, $r_i$'s denotes rows of the states.):

$$c_0 = s_0 s_1 s_2 s_3 \qquad r_0 = s_0 s_4 s_8 s_{12}$$
$$c_1 = s_4 s_5 s_6 s_7 \qquad r_1 = s_1 s_5 s_9 s_{13}$$
$$c_2 = s_8 s_9 s_{10} s_{11} \qquad r_2 = s_2 s_6 s_{10} s_{14}$$
$$c_3 = s_{12} s_{13} s_{14} s_{15} \qquad r_3 = s_3 s_7 s_{11} s_{15}$$

- XOR operation

The operation XOR is denoted with $\bigoplus$. This operation involves adding each bit in the bytes correspondingly in modulo 2 arithmetic, resulting in a new byte where each bit represents the XOR of the corresponding bits in the original bytes.

For example, $0x2a \oplus 0x32 = 0x18$. The truth table of XOR operation is shown in Table 3.

Table 3: The XOR Funciton

| a | b | a $\oplus$ b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

The addition of two bytes is performed with addition of each bit correspondingly. For example, $0x2a \oplus 0x32 = 0x18$.

- Multiplication

Multiplication of two elements is performed in Galois Field ($2^8$), and it is denoted as $\bullet$. After a standard polynomial multiplication, the modulo operation in the irreducible polynomial is applied to the result. The irreducible polynomial is defined as there are no any factors of the polynomial except 1 and itself. The designer of the AES chose the irreducible polynomial to be $x^8+x^4+x^3+x+1$, which is the smallest irreducible polynomial degree of 8.

For example, $0x82 \bullet 0x21$:

$$\begin{aligned}(x^7+x) \bullet (x^5+1) &= x^{12}+x^7+x^6+x \quad mod(x^8+x^4+x^3+x+1) \\ &= x^4x^8+x^7+x^6+x \\ &= x^4(x^4+x^3+x+1)+x^7+x^6+x \\ &= x^8+x^7+x^5+x^4+x^7+x^6+x \\ &= x^4+x^3+x+1+x^5+x^4+x^6+x \\ &= x^6+x^5+x^3+1\end{aligned}$$

$x^8$ is replaced by $x^4+x^3+x+1$ in each time. $x^6+x^5+x^3+1$ is equal to 01101001 in binary system, which is *0x69*.

## 2.2 Algorithm Specification

The AES algorithm takes 128-bits as input, performs a series of operations, and generates 128-bit ciphertext. The round function of AES consists of four different layers; each has a different property. The number of rounds in AES varies based on the key size. Each round is generally similar to one another, except for the last round which excludes the MixColumns operation. The specification of AES algorithm is given in Table 4.

Table 4: AES Specifications

| | keySize | blockSize | numberOfRounds |
|---|---|---|---|
| AES-128 | 128 | 128 | 10 |
| AES-196 | 196 | 128 | 12 |
| AES-256 | 256 | 128 | 14 |

### 2.2.1 Data Encryption / Data Decryption

#### 2.2.1.1 Sub-Bytes / Inverse Sub-Bytes

Each state byte is substituted with another byte by using the substitution table, which is called S-Box. Transformation of bytes provides non-linearity, making the algorithm strong against differential and linear attacks. Each byte of the state goes through to the S-Box operation. Figure 11 shows this operation. The S-Box of the algorithm is generated with an affine transform. The affine transformation formula of AES S-Box is given below.

$$S(x) = M * x^{-1} \oplus c, \tag{1}$$

is the S-Box affine transformation formula, where *M* is the $8 \times 8$ circulant binary matrix and *c* is a constant byte.

***Step 1:*** Compute the multiplicative inverse of $a$ in $\mathbb{GF}(2^8)$

$$b = a^{-1}, \tag{2}$$

(if a = 0, then b = 0).

***Step 2:*** Calculate the following affine transformation.

$$\begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$s_i$'s and $b_i$'s are the bits of $s$ and $b$ correspondingly and $0 \leq i < 7$.

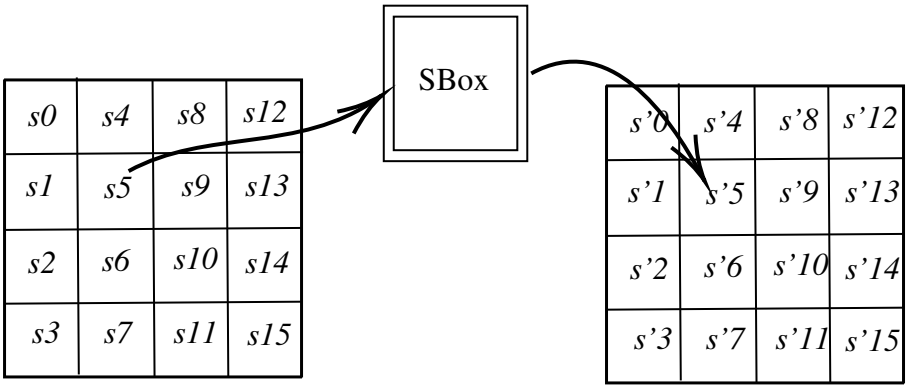


Figure 11: S-Box Operation for a byte

The Inverse Sub-Bytes operation in AES is the inverse of the Sub-Bytes operation. It applies an inverse multiplication operation on the result of the affine transformation for each byte of the state. The forward and inverse S-boxes of the AES are given in Table 5 and Table 6.

Table 5: The S-box Table of AES

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

Table 6: The Inverse S-box Table of AES

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 52 | 09 | 6a | d5 | 30 | 36 | a5 | 38 | bf | 40 | a3 | 9e | 81 | f3 | d7 | fb |
| 1 | 7c | e3 | 39 | 82 | 9b | 2f | ff | 87 | 34 | 8e | 43 | 44 | c4 | de | e9 | cb |
| 2 | 54 | 7b | 94 | 32 | a6 | c2 | 23 | 3d | ee | 4c | 95 | 0b | 42 | fa | c3 | 4e |
| 3 | 08 | 2e | a1 | 66 | 28 | d9 | 24 | b2 | 76 | 5b | a2 | 49 | 6d | 8b | d1 | 25 |
| 4 | 72 | f8 | f6 | 64 | 86 | 68 | 98 | 16 | d4 | a4 | 5c | cc | 5d | 65 | b6 | 92 |
| 5 | 6c | 70 | 48 | 50 | fd | ed | b9 | da | 5e | 15 | 46 | 57 | a7 | 8d | 9d | 84 |
| 6 | 90 | d8 | ab | 00 | 8c | bc | d3 | 0a | f7 | e4 | 58 | 05 | b8 | b3 | 45 | 06 |
| 7 | d0 | 2c | 1e | 8f | ca | 3f | 0f | 02 | c1 | af | bd | 03 | 01 | 13 | 8a | 6b |
| 8 | 3a | 91 | 11 | 41 | 4f | 67 | dc | ea | 97 | f2 | cf | ce | f0 | b4 | e6 | 73 |
| 9 | 96 | ac | 74 | 22 | e7 | ad | 35 | 85 | e2 | f9 | 37 | e8 | 1c | 75 | df | 6e |
| a | 47 | f1 | 1a | 71 | 1d | 29 | c5 | 89 | 6f | b7 | 62 | 0e | aa | 18 | be | 1b |
| b | fc | 56 | 3e | 4b | c6 | d2 | 79 | 20 | 9a | db | c0 | fe | 78 | cd | 5a | f4 |
| c | 1f | dd | a8 | 33 | 88 | 07 | c7 | 31 | b1 | 12 | 10 | 59 | 27 | 80 | ec | 5f |
| d | 60 | 51 | 7f | a9 | 19 | b5 | 4a | 0d | 2d | e5 | 7a | 9f | 93 | c9 | 9c | ef |
| e | a0 | e0 | 3b | 4d | ae | 2a | f5 | b0 | c8 | eb | bb | 3c | 83 | 53 | 99 | 61 |
| f | 17 | 2b | 04 | 7e | ba | 77 | d6 | 26 | e1 | 69 | 14 | 63 | 55 | 21 | 0c | 7d |

#### 2.2.1.2 Shift-Rows / Inverse Shift-Rows

The state rows are cyclically shifted to the left with offset bytes. The first row remains unchanged, while the bytes in the second row are cyclically shifted to the left. Similarly, the bytes in the third row are shifted to the left by two bytes. This shifting

operation ensures diffusion and adds complexity to the encryption process. Bytes in the last row are cyclically moved three bytes to the left. Input and output state representations are shown in Figure 12. The inverse of the Shift-Rows is the same as the Shift-Rows, except for shifting direction. The cyclic right shift is applied on the rows.

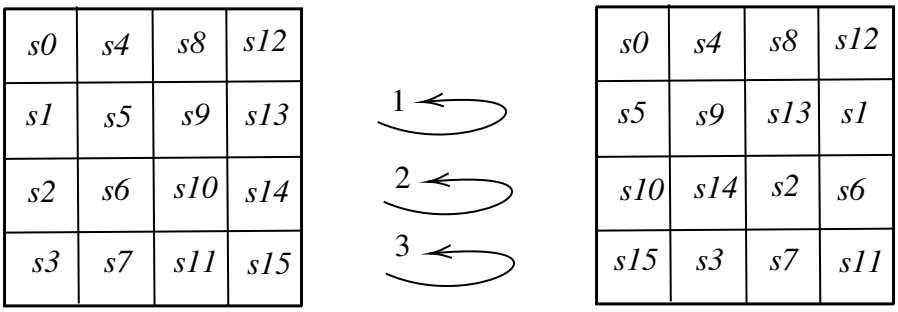


Figure 12: Shift-Rows Operation of AES State

#### 2.2.1.3 MixColumns / Inverse MixColumns

Constant matrix multiplication is applied on the state. Each column is separately multiplied by the rows of the constant matrix. The inverse of the Mix-Columns is also a constant matrix multiplication. For Inverse Mix-Columns, the inverse of the matrix is used.

- Mix-Columns

$$\begin{bmatrix} s'_0 & s'_4 & s'_8 & s'_{12} \\ s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \bullet \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

- Inverse Mix-Columns

$$\begin{bmatrix} s'_0 & s'_4 & s'_8 & s'_{12} \\ s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \end{bmatrix} = \begin{bmatrix} e & b & d & 9 \\ 9 & e & b & d \\ d & 9 & e & b \\ b & d & 9 & e \end{bmatrix} \bullet \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

#### 2.2.1.4 AddRoundKey / Inverse AddRoundKey

In this layer, the state is XORed with the 128-bit round key, and both the AddRound-Key and Inverse AddRoundKey operations are identical because the inverse of the XOR operation is the XOR operation itself. Algorithm 1 and Algorithm 2 show the pseudo-codes for AES encryption and decryption.

**Algorithm 1:** Encryption Algorithm of AES

**Inputs** byte $in[16]$, word $key[4 \cdot (Nr + 1)]$
**Output** byte $out[16]$]
/* A word is 4-bytes. */
/* Nr is the number of rounds. */
$state = in$
$AddRoundKey(state, w[0, 3])$
$round = 1$
**while** $round < Nr$ **do**
  $SubBytes(state)$
  $ShiftRows(state)$
  $MixColumns(state)$
  $AddRoundKey(state, w[4 \cdot round, 4 \cdot (round + 1) - 1])$
  $round = round + 1$
**end**
$SubBytes(state)$
$ShiftRows(state)$
$AddRoundKey(state, w[4 \cdot Nr, 4 \cdot (Nr + 1) - 1])$
**return** $state$

**Algorithm 2:** Decryption Algorithm of AES

**Inputs** byte $in[16]$, word $key[4 * (Nr + 1)]$
**Output** byte $out[16]$]
/* A word is 4-bytes. */
/* Nr is the number of rounds. */
$state = in$
$AddRoundKey(state, w[4 \cdot Nr, 4 \cdot (Nr + 1) - 1])$
$round = Nr - 1$
**while** $round > 0$ **do**
  $InverseShiftRows(state)$
  $InverseSubBytes(state)$
  $InverseAddRoundKey(state, w[4 \cdot round, 4 \cdot (round + 1) - 1])$
  $InverseMixColumns(state)$
  $round = round - 1$
**end**
$InverseSubBytes(state)$
$InverseShiftRows(state)$
$InverseAddRoundKey(state, w[0, 3])$
**return** $state$

### 2.2.2 Key Generation

The AES algorithm has three versions, each supporting a different key size: 128 bits, 196 bits, and 256 bits. The Key Expansion function takes an initial key and expands it to generate round keys for the desired number of rounds. For a key size of 128 bits,

the algorithm executes ten rounds. The round keys for these ten rounds are generated using the Key Expansion algorithm, as shown in Algorithm 3.

**roundCons** is the array of round constant bytes, which is defined as:

$$roundCons = \{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36\}$$

#### 2.2.2.1 Sub-Word

The Sub-Word function is the sub-version of Sub-Bytes. The function takes four bytes, calculates the S-box function of each bytes, and returns them.

#### 2.2.2.2 Rot-Word

The Rot-Word function is similar with Shift-Rows. It takes four bytes and performs cyclic shift to left.

**Algorithm 3:** Generating Round Keys of AES

**Inputs** byte $key[4 \cdot Nk], Nk$
**Output** word $w[4 \cdot (Nr + 1)]$]
/* A word is 4-bytes. */
/* Nk is the wordSize of the AES Key. */
/* Nr is the number of rounds. */
$word = 0x00, 0x00, 0x00, 0x00$
$temp = 0x00, 0x00, 0x00, 0x00$
$i = 0$
**while** $i < Nk$ **do**
  $w[i] = word(key[4 \cdot i], key[4 \cdot i + 1], key[4 \cdot i + 2], key[4 \cdot i + 3])$
  $i = i + 1$
**end**
$i = Nk$
**while** $i < 4 * (Nr + 1)$ **do**
  $temp = w[i - 1]$
  **if** $i$ *mod* $Nk = 0$ **then**
    $temp = SubWord(RotWord(temp))$ xor $Rcon[i/Nk]$
  **end**
  **if** $(Nk > 6)$ *and* $(i$ *mod* $Nk) = 4$ **then**
    $temp = SubWord(temp)$
  **end**
  $w[i] = w[i - Nk]$ xor $temp$
  $i = i + 1$
**end**
**return** $w[4 \cdot (Nr + 1)]$

### 2.3 Performance Evaluation of AES on Different Platforms

After the Rijndael Algorithm was chosen as a standard, it is used in various systems that require security. As the AES algorithm has become increasingly popular, the

number of researchers trying to optimize it has increased. Numerous implementation techniques have been published by considering various metrics. Throughput, power consumption, and gate-area are only three of the metrics. We summarized recent software and hardware performance analysis results on the AES algorithm in this subsection. Table 7 shows the recent AES implementation result on CUDA devices. The table was sorted according to Gbps per Watt. Many researchers also proposed optimized implementation of AES for Intel CPUs. The best result is 2.072 Gbps/Watt, which is obtained by [Tezcan, 2021]. The designs which focused on the throughput per slice on FPGA are listed in Table 8. These implementations are suitable for resource constrained devices. Table 9 shows the fastest AES implementations on FPGA. The table is arranged in ascending order based on the measure of gigabits per second (Gbps).

Table 7: AES-128 encryption performance on different CUDA device

| Device | Architecture | Launch Year | Gbps/W | Gbps | Design |
|---|---|---|---|---|---|
| Nvidia RTX 2070 Super | Turing | 2019 | 0.236 | 50.8 | [An and Seo, 2020] |
| Nvidia GTX 1070 | Pascal | 2016 | 1.427 | 214.0 | [An and SEO, 2020] |
| Nvidia GTX 1080 | Pascal | 2016 | 1.555 | 279.9 | [Abdelrahman et al., 2017] |
| Nvidia RTX 2070 | Turing | 2018 | 1.771 | 310.0 | [An and SEO, 2020] |
| Nvidia GTX 970 | Maxwell | 2014 | 2.174 | 315.2 | [Tezcan, 2021] |
| Nvidia Tesla P100 | Pascal | 2016 | 2.423 | 605.9 | [Nishikawa et al., 2017] |
| Nvidia RTX 2070 Super | Turing | 2019 | 4.087 | 878.6 | [Tezcan, 2021] |

Table 8: Throughput comparison of AES per slice (LUT) on different FPGA devices

| Device | BRAM | Slice(LUT) | Throughput(GBit/s) | TPS(Mbps/Slice) | Design |
|---|---|---|---|---|---|
| Virtex-5 | 2 | 459 | 4.262 | 9.29 | [Kundi et al., 2016] |
| Virtex-5 | 0 | 798 | 4.34 | 5.43 | [Rais and Qasim, 2010] |
| Virtex-5 | 0 | 950 | 4.1 | 4.315 | [Bulens et al., 2008] |
| Virtex-5 | 0 | 1223 | 3.7 | 2.76 | [Bouhraoua, 2010] |
| Virtex-7 | 0 | 2444 | 5.306 | 2.17 | [Hussain and Jamal, 2012] |
| Virtex-6 | 16 | 4926 | 1.815 | 0.368 | [Criado et al., 2014] |

Table 9: Throughput comparison of AES per seconds on FPGA devices

| Device | BRAM | Slice(LUT) | Throughput(GBit/s) | TPS(Mbps/Slice) | Design |
|---|---|---|---|---|---|
| Virtex-5 | 0 | 8896 | 25.89 | 2.91 | [Reddy et al., 2011] |
| Virtex-5 | 20 | 4491 | 42.62 | 9.49 | [Kundi et al., 2016] |
| Virtex-6 | 0 | 18854 | 44.074 | 3.71 | [Wang and Ha, 2013] |
| Virtex-2 Pro | 80 | 11398 | 57.28 s | 5.026 | [Iyer et al., 2011] |

# CHAPTER 3

# COMPACT AES S-BOX IMPLEMENTATION FOR HARDWARE PLATFORMS

## 3.1 Mathematical Background

**Definition 3.1.1** (Binary Number)**.** A binary number is a number represented in the $modulo\ 2$. Only two symbols are used, typically: $0$ and $1$.

The binary system is the system that represents every number with binary numbers ($0$ or $1$). Almost all modern computers and computer-based devices prefer to use the binary system. This is due to its straightforward implementation in computer architecture. The behaviors of logic gates can be represented in binary systems.

**Definition 3.1.2** (Binary Operation)**.** Let $S$ be set and let $S \times S$ denote the set of all ordered pairs $(x, y)$ where $x, y \in S$. Then mapping from $S \times S$ into $S$ will be called a binary operation. This requires that the image of $(x, y) \in S \times S$ be in $S$. This is called **closure property**.

**Definition 3.1.3** (Group)**.** A group is a set $G$ together with a binary operation $\cdot$, and satisfies the following properties:

**1) Associative:** For any $x, y, z \in G$, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

**2) Identity(or Unity:** There is an element $e \in G$, such that for all $x \in G$, $x \cdot e = e \cdot x = x$.

**3) Inverse Element:** For all $x \in G$, there is an element $x^{-1} \in G$ such that $x \cdot x^{-1} = e$, where $e$ is the identity element.

**Definition 3.1.4** (Abelian Group)**.** Abelian group $G$ is a group with **commutative property**.

**Commutative:** For all $x, y \in G$, $x \cdot y = y \cdot x$.

**Example:** $(\mathbb{Z}, +)$ is a group.

For all $x, y, z \in \mathbb{Z}$, $(x + y) + z = x + (y + z)$ holds, so it is associative .

There is $0 \in \mathbb{Z}$, for all $x \in \mathbb{Z}$, $x + 0 = x$ holds, so it has identity element.

For all $x \in \mathbb{Z}$, there is an element $-x \in \mathbb{Z}$, so every element has its inverse.

Therefore, $(\mathbb{Z}, +)$ is a group. $(\mathbb{Z}, +)$ is also abelian group because for every $x, y \in \mathbb{Z}$, $x + y = y + x$ holds.

However; $(\mathbb{Z}, -)$ is a **not** group, because it is not associative. For example $(2-4)-6 \neq 2-(4-6)$.

**Definition 3.1.5** (Field)**.** A field is a set $F$ together with two binary operations $(+, \cdot)$, and satisfies the following properties:

**1) Associative:** Both multiplication and addition operations satisfy the following equations:For any $x, y, z \in F$,

$(x \cdot y) \cdot z = x \cdot (y \cdot z)$ and $(x + y) + z = x + (y + z)$.

**2) Commutative:** Both multiplication and addition operations satisfy the following equations:For any $x, y \in F$,
$(x \cdot y) = x \cdot y$ and $(x \ + y) = x + y$.

**3) Identity Element:** Both multiplication and addition operations have identity element. There is $0, 1 \in F$ such that for all $x \in F, 1 \cdot x = x$ and $0 + x = x$.

**4) Additive Inverse:** For all $x \in F$, there is an element $-x \in F$ such that $x + (-x) = 0$, where $0$ is the identity element of addition.

**5) Multiplicative Inverse:** For all $x \in F$, there is an element $x^{-1} \in F$ such that $x \cdot x^{-1} = 1$, where $1$ is the identity element of multiplication.

**6) Distributive:** There holds distributive property of multiplication over addition. For all $x, y, z \in F$, $x \cdot (y + z) = x \cdot y + x \cdot z$.

**Example:** $\big(\mathbb{R}, (+, \cdot)\big)$ is a field.

**Example:** $\big(\mathbb{C}, (+, \cdot)\big)$ is a field, where $\mathbb{C}$ is the complex numbers: $\mathbb{C} = \{x+y\cdot i | x, y \in \mathbb{R}, i^2 = -1\}$ and $0 = 0 + 0 \cdot i, 1 = 1 + 0 \cdot i$.

**Example:** $\big(\mathbb{Z}_7, (+, \cdot)\big)$ is a field.

In $\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$, the addition and multiplication operations are closed under modulo arithmetic. The associative, commutative, and distributive properties hold trivially. The identity element for addition is $0$, and the identity element for multiplication is $1$. To verify the inverse properties, we find that each element has an additive inverse and a multiplicative inverse. Now, we need to check inverse properties.

For additive inverse;

$0 + 0 \equiv 0 \bmod 7$
$1 + 6 \equiv 0 \bmod 7$. (1 and 6) are the inverse of each other.
$2 + 5 \equiv 0 \bmod 7$. (2 and 5) are the inverse of each other.
$3 + 4 \equiv 0 \bmod 7$. (3 and 4) are the inverse of each other.

For multiplicative inverse;

$1 \cdot 1 \equiv 1 \bmod 7$
$2 \cdot 4 \equiv 1 \bmod 7$. (2 and 4) are the inverse of each other.
$3 \cdot 5 \equiv 1 \bmod 7$. (3 and 5) are the inverse of each other.
$6 \cdot 6 \equiv 1 \bmod 7$.

**Lemma 3.1.1.** if $p$ is prime, then $\big(\mathbb{Z}_p, (+, \cdot)\big)$ is a field.

**Definition 3.1.6** (Extension Field)**.** Assume $F$ is a field and $K$ is the subset of $F$. If $K$ itself is a field (under operations of $F$), it is called a *subfield* of $F$. Then $F$ is called an **extension(field)** of $K$.

- **Polynomials over Fields**

We have introduced the basic algebraic structures to build polynomials over fields. Assume that $F$ is field with $(+, \cdot)$ operations. A polynomial $f$ over $F$ (also denoted as $f \in F[x]$ ) is defined in the form:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0,$$

where the coefficients $a_0, a_1, a_2, .., a_n \in f$. The set of all polynomials over a field is represented as $F[x]$. The degree of $f$ is denoted *deg(f) = n*, where $a_n \neq 0$. $f$ is also denoted as:

$$f(x) = (a_n, a_{n-1}, ..., a_2, a_1, a_0).$$

If the leading coefficient (nonzero coefficient of highest degree) $a_n = 1$, then $f$ is called *monic polynomial*.

**Example:** Consider the field $\big(\mathbb{Z}_5, (+, \cdot)\big)$. An example of a polynomial over $\mathbb{Z}_5$ is
*f(x)* = $3x^2 + 2x + 4$.
All the coefficients of $f$ are in $\mathbb{Z}_5$, and *deg(f) = 2*. It is not a *monic* polynomial since the leading coefficient is 3.

**Multiplications of Polynomials:**

Let $f = (c_n, c_{n-1}, ..., c_2, c_1, c_0)$ and $g = (d_m, d_{m-1}, ..., d_2, d_1, d_0)$, then the $f \cdot g = (e_{n+m}, e_{n+m-1}, ..., e_2, e_1, e_0)$ is defined as:

$$\sum_{i=0}^{n} c_i x_i \cdot \sum_{j=0}^{m} d_j x_j = \sum_{k=0}^{n+m} \sum_{l=0}^{k} c_l d_{k-l} x^k$$

**Division of Polynomials:**

Let $f$ and $g$ be polynomials over $F[x]$, If there is a polynomial $h$ over $F[x]$ such that:

$$g = f \cdot h$$

Then we say that $f$ divides $g$ and, it denoted as "$f|g$". "$f \nmid g$" denotes that $f$ does not divide $g$.

**Example:** Let $f(x) = 2x^2 + x + 1$ and $g(x) = x^3 + x + 1$. Assume $h = f \cdot g$, then

$$\begin{aligned} h(x) &= (2x^2 + x + 1) \cdot (x^3 + x + 1) \\ &= 2x^5 + 2x^3 + 2x^2 + x^4 + x^2 + x + x^3 + x + 1 \\ &= 2x^5 + x^4 + 3x^3 + 3x^2 + 2x + 1 \end{aligned}$$

It is also represented as:

$$(2, 1, 1) \cdot (1, 0, 1, 1) = (2, 1, 3, 3, 2, 1).$$

**Definition 3.1.7** (Irreducible Polynomials)**.** A polynomial $f$ over $F[x]$ is *irreducible* if the only factorization of it are the trivial ones ( *one* and the polynomial itself).

**Example:** Let $f(x) = x^2 + x + 1$ be a polynomial over $\big(\mathbb{Z}_2, (+, \cdot)\big)$.
For $x = 0$, $f(0) = 1$, which is not 0.
For $x = 1$, $f(1) = 1 + 1 + 1 = 3 \equiv 1 \bmod 2$, which is not 0.
$f$ is also not divisible by any polynomials of degree 1. Therefore, $f$ is irreducible polynomial over $\big(\mathbb{Z}_2, (+, \cdot)\big)$.

### 3.2 Field Isomorphism

The optimization studies of [Canright, 2005] is mainly constructed on isomorphism of finite fields. Thanks to irreducible polynomial of AES S-box, Canright proposed different way to construct AES S-box by using field isomorphism.

**Definition 3.2.1** (Isomorphism)**.** Assume there are two fields $\big(\mathbb{F}_1, (+, \times)\big)$ and $\big(\mathbb{F}_2, (\oplus, \cdot)\big)$ with the same number of elements. A one-to-one map $f : \mathbb{F}_1 \mapsto \mathbb{F}_2$ is called an *isomorphism* from $\big(\mathbb{F}_1, (+, \times)\big)$ onto $\big(\mathbb{F}_2, (\oplus, \cdot)\big)$, if for any $a, b \in \mathbb{F}_1$

$$\begin{aligned} f(a + b) &= f(a) \oplus f(b), \\ f(a \times b) &= f(b) \cdot f(b) \end{aligned}$$

Then $\big(\mathbb{F}_1, (+, \times)\big)$ and $\big(\mathbb{F}_2, (\oplus, \cdot)\big)$ are called isomorphic fields to each other.

**Example:** Consider the field $\big(\mathbb{Q}(\sqrt{2}), (+, \cdot)\big)$ whose elements are in the form $a + b\sqrt{2}$, and the field $\big(\mathbb{Q}(\sqrt{5}), (+, \cdot)\big)$ whose elements are in the form $c + d\sqrt{5}$, where $a, b, c, d \in \mathbb{Q}$.

The function $f : \mathbb{Q}(\sqrt{2}) \to \mathbb{Q}(\sqrt{5})$ defined by:

$$f(a + b\sqrt{2}) = a + b\sqrt{10}$$

is claimed to be a field isomorphism. In order to be a field isomorphism, the following three properties must be satisfied by the function $f$:

$f$ **is a one-to-one:** Since $f$ maps every element in $\mathbb{Q}(\sqrt{2})$ to a distinct element in $\mathbb{Q}(\sqrt{5})$, and vice versa, $f$ is a one-to-one.

$f$ **satisfies addition:** For any $x, y \in \mathbb{Q}(\sqrt{2})$ assume $x = a + b\sqrt{2}$ and $y = c + d\sqrt{2}$, the following equation is verified:

$$\begin{aligned} f(x+y) &= f((a+b\sqrt{2}) + (c+d\sqrt{2})) \\ &= f((a+c) + (b+d)\sqrt{2}) \\ &= (a+c) + (b+d)\sqrt{10} \\ &= (a+b\sqrt{10}) + (c+d\sqrt{10}) \\ &= f(x) + f(y) \end{aligned}$$

Thus, $f$ satisfies addition.

$f$ **satisfies multiplication:** For any $x, y \in \mathbb{Q}(\sqrt{2})$ assume $x = a + b\sqrt{2}$ and $y = c + d\sqrt{2}$, the following equation is verified:

$$\begin{aligned} f(xy) &= f((a+b\sqrt{2})(c+d\sqrt{2})) \\ &= f((ac+2bd) + (ad+bc)\sqrt{2}) \\ &= (ac+2bd) + (ad+bc)\sqrt{10} \\ &= (a+b\sqrt{10})(c+d\sqrt{10}) \\ &= f(x)f(y) \end{aligned}$$

Thus, $f$ satisfies multiplication. Since $f$ is a one-to-one that preserves addition and multiplication, it is a field isomorphism.

**Example:** Consider the fields $\big(\mathbb{Q}, (+, \cdot)\big)$ and $\big(\mathbb{Q}(\sqrt{7}), (+, \cdot)\big)$ , where $\mathbb{Q}$ is the field of rational numbers and $\mathbb{Q}(\sqrt{7})$ is the field obtained by adjoining the square root of 7 to $\mathbb{Q}$.

The function $f : \mathbb{Q} \to \mathbb{Q}(\sqrt{7})$ defined by:

$$f(a + b\sqrt{7}) = a - b\sqrt{7}$$

is a field isomorphism. To see why this is the case, we need to verify that $f$ satisfies the following properties:

$f$ **is a one-to-one**: It's easy to see that $f$ is one-to-one and onto.
$f$ **satisfies addition**: Let $x, y \in \mathbb{Q}(\sqrt{7})$. Then, we have:

$$\begin{aligned} f(x+y) &= f((a+b\sqrt{7}) + (c+d\sqrt{7})) \\ &= f((a+c) + (b+d)\sqrt{7}) \\ &= (a+c) - (b+d)\sqrt{7} \\ &= (a-b\sqrt{7}) + (c-d\sqrt{7}) \\ &= f(x) + f(y) \end{aligned}$$

$f$ **satisfies multiplication**: Let $x, y \in \mathbb{Q}(\sqrt{7})$. Then, we have:

$$\begin{aligned} f(x \cdot y) &= f((a+b\sqrt{7}) \cdot (c+d\sqrt{7})) \\ &= f((ac+2bd) + (ad+bc)\sqrt{7}) \\ &= (ac+2bd) - (ad+bc)\sqrt{7} \\ &= (a-b\sqrt{7}) \cdot (c-d\sqrt{7}) \\ &= f(x) \cdot f(y) \end{aligned}$$

Since $f$ satisfies all the required properties, it is a field isomorphism between $\mathbb{Q}$ and $\mathbb{Q}(\sqrt{7})$.

### 3.3 Galois Field

**Definition 3.3.1** (Galois Field)**.** A Galois field, denoted as $\mathbb{GF}$ or $\mathbb{F}_q$, is a finite field with a finite number of elements. The cardinality of a Galois field, which represents the number of elements in the field, is always a prime number or a power of a prime.

The elements of Galois field $\mathbb{GF}_q$ is defined as

$$\mathbb{GF}(q) = (0, 1, 2, ..., q-2, q-1)$$

The elements of $\mathbb{GF}(q^n)$ can be represented as the field of equivalence classes of polynomials whose coefficients belong to $\mathbb{GF}(q)$. Any irreducible polynomial of degree $n$ creates the same field elements. For example, $x^3 + x + 1$ and $x^3 + x^2 + 1$ yields the same field elements, because both are $3^{rd}$ degree irreducible polynomials.

**Example**: List the elements of $\mathbb{GF}(3^2)$ in polynomial notation.

$$\mathbb{GF}(3^2) = \{0, 1, 2, x, x+1, x+2, 2x, 2x+1, 2x+2\}$$

There are $3^2 = 9$ elements, where each of the element is polynomial of degree at most $1$ and coefficients are in $\mathbb{Z}_3$.

**Example**: List the elements of $\mathbb{GF}(2^4)$ in polynomial notation.

$$\begin{aligned}\mathbb{GF}(2^4) =&\{0,1\} \quad \cup \\ &\{x, x+1\} \quad \cup \\ &\{x^2, x^2+1, x^2+x, x^2+x+1\} \quad \cup \\ &\{x^3, x^3+1, x^3+x, x^3+x+1, x^3+x^2, \\ &\qquad x^3+x^2+1, x^3+x^2+x, x^3+x^2+x+1\}\end{aligned}$$

There are $2^4 = 16$ elements, where each of the element is polynomial of degree at most 3 and coefficients are in $\mathbb{Z}_2$.

**Example**: List the elements of $\mathbb{GF}(4^2)$ in polynomial notation.

$$\begin{aligned}\mathbb{GF}(4^2) =&\{0,1,2,3\} \quad \cup \\ &\{x, x+1, x+2, x+3, 2x, 2x+1, 2x+2, 2x+3 \\ &\qquad 3x, 3x+1, 3x+2, 3x+3\}\end{aligned}$$

There are $4^2 = 16$ elements, where each of the element is polynomial of degree at most 1 and coefficients are in $\mathbb{Z}_4$.

**Definition 3.3.2** (Primitive Element)**.** A group *G* is called *cyclic* if there is an element $e \in G$ such that for any $x \in G$ there is some integer $i$ with $x = e^i$. The element $e$ is called *primitive element* (or *generator*), and it is written as $G =< e >$.

**Definition 3.3.3** (Primitive Polynomials)**.** A polynomial that produces all the elements of an extension field is referred to as a *primitive polynomial*.

**Example:** Show that $x^4 + x + 1$ is a primitive polynomial over $\mathbb{GF}(2)$.

We need to prove that there is an element $e$ such that $e$ can generate all the elements of $\mathbb{GF}(2^4)$. We have shown the elements of $\mathbb{GF}(2^4)$ above.
Let generator be $x$.

$$\begin{aligned} x^1 &= x & x^9 &= x^3 + x \\ x^2 &= x^2 & x^{10} &= x^2 + x + 1 \\ x^3 &= x^3 & x^{11} &= x^3 + x^2 + x \\ x^4 &= x + 1 & x^{12} &= x^3 + x^2 + x + 1 \\ x^5 &= x^2 + x & x^{13} &= x^3 + x^2 + 1 \\ x^6 &= x^3 + x^2 & x^{14} &= x^3 + 1 \\ x^7 &= x^3 + x + 1 & x^{15} &= 1 \\ x^8 &= x^2 + 1 \end{aligned}$$

We replaced $x^4$ with $x+1$ since they are equal in the given field. As we can see that all the elements are generated. To sum up, $x$ is the primitive element and $x^4+x+1$ is a primitive polynomial over $\mathbb{GF}(2)$.

**Definition 3.3.4** (Irreducible Primitive Polynomials)**.** If a polynomial $f$ is both irreducible and primitive, then it is called *irreducible primitive polynomial*.

For AES S-box, the field $\mathbb{GF}(2^8)$ is used. Now we know that the field has exactly $256$ elements and each element of the field can be represented by a polynomial degree at most 7. The coefficient of polynomials are in $\mathbb{Z}_2$, which means *binary system* is used for representation.

## 3.4 Construction of S-Box

**Definition 3.4.1** (Circulant Matrix)**.** An $n \times n$ matrix called *circulant* if it is in the form:

$$M = \begin{pmatrix} c_{n-1} & c_0 & c_1 & . & . & . & c_{n-3} & c_{n-2} \\ c_{n-2} & c_{n-1} & c_0 & . & . & . & . & c_{n-3} \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ c_2 & . & . & . & . & . & c_0 & . \\ c_1 & c_2 & . & . & . & . & c_{n-1} & c_0 \\ c_0 & c_1 & c_2 & . & . & . & c_{n-2} & c_{n-1} \end{pmatrix}$$

The transpose of this form also constructs *circulant* matrix. The cyclic permutation of the first row creates other rows of the matrix. The circulant matrix that contains only $0$s and $1$s is called *circulant binary matrix*.

The polynomial representation of the matrix is as follow:

$$f(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + ... + c_1x + c_0$$

The Rijndael S-Box use the following circulant binary matrix.

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The polynomial representation of the Rijndael S-Box is

$$f(x) = x^7 + x^3 + x^2 + x + 1$$

The polynomial is also denoted in hexadecimal form $0x8f$. In this thesis, the hexadecimal notation of circulant matrices is used for simplicity.

### 3.4.1 Affine Transform

The S-Box function of AES is generated with an affine transform. The affine transformation formula is given below.

$$S(x) = M \times x^{-1} \oplus c,$$

where *M* is the $8 \times 8$ circulant binary matrix and *c* is a constant byte.

***Step 1:*** Find the multiplicative inverse of $a$ in $\mathbb{GF}(2^8)$

$$b = a^{-1}, \tag{3}$$

(if a = 0, then b = 0).

***Step 2:*** Calculate the following affine transformation.

$$\begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$s_i$'s and $b_i$'s are the bits of $s$ and $b$ correspondingly and $0 \leq i < 7$.

## 3.5 Implementation Methods of S-boxes for FPGA Platforms

### 3.5.1 RAM-Based Implementation

There are 256 different input values for an S-Box function. The output of the S-box function is a byte. Therefore, a 256-byte table is enough to keep an S-Box. Block Rams on FPGA platforms can be used to store the S-Boxes. RAMs are the memory spaces in FPGA, which can be written with data or retrieved as data. The type of memory which is not writable is called ROM (read-only memory). We do not need to write anything to memory for S-Box tables; we just read the data. For convention, we use RAM instead of ROM for explanations. Several designers prefer to use RAM-based implementation [Kundi et al., 2016],[Aziz and Ikram, 2007]. Although keeping S-boxes in a block ram increases the throughput performance of the algorithm [Saggese et al., 2003], using RAMs for S-Boxes also has many drawbacks.

Depending on FPGA, a RAM can store a block of 18 Kbits or 36 Kbits data. For every 8-bit of data, one parity bit is used. Therefore, 18 Kbits RAMs can store 16 Kbits, and 36 Kbits RAMs can store 32 Kbits [AMD Xilinx Company, 2022]. Since

only a single data can be retrieved from RAM in one cycle, one RAM can be used for only one byte. AES state has 16 bytes. Therefore 16 Block RAMs are needed to implement the S-Box layer in one cycle. Only 256 bytes of the RAM is used to keep an S-Box; the rest of the ram will be useless.

- Singe Port Block-RAM

The Singe Port Block-RAM has only one interface. These types of RAMs can read or write only a single data in each cycle. This is the simplest RAM configuration for FPGA platforms to retrieve data. According to *write enable(we)* signal, a data is retrieved or written. If the *write enable* signal is $high(\text{"1"})$, *data input (dina)* is written into *address(addra)* index of RAM. If the *write enable* signal is $low(\text{"0"})$, the data in the *data input (dina)* index is retrieved from RAM with *data out (dout)* signal. *Enable(en)* signal enables and disables the RAM. The input and output ports of Single Port Block-RAM is shown in Figure 13.

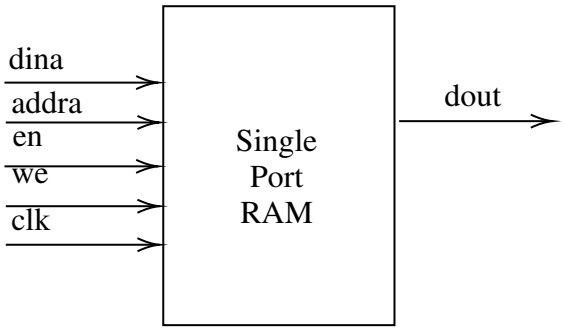


Figure 13: The input and output ports of Single Port Block-RAM

- Dual Port Block-RAM

The Dual Port Block-RAM has two interfaces. These types of RAMs can read or write two data in each cycle. According to write enable signals, a data is retrieved or written from relevant ports. The input and output ports of Dual Port Block-RAM is shown in Figure 14.

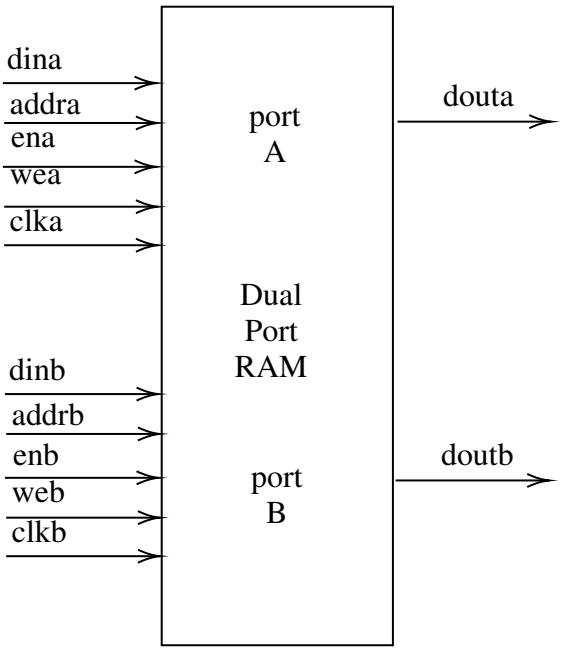


Figure 14: The input and output ports of Dual Port Block-RAM

### 3.5.2 Logic-Based Implementation

Instead of storing S-box in RAMs, the Look-up Tables(LUT) can be also used for this purpose. There are different types of LUTs; in this thesis, we focused on 4-LUT and 6-LUT. Since the FPGAs that we worked on use 4-LUT and 6-LUT types of LUTs. According to each input combination, one-bit output is kept in the truth table. If we write the S-Box as a byte array in the code, the synthesizer converts this S-Box into LUTs. According to our studies, the FPGAs that use 4-LUTs as a primitive need 64 Look-up tables to represent 256-byte S-Box table. For the FPGAs whose primitives are 6-LUTs need 40 Look-up tables to implement the S-Box. The Naive Look-up table implementation refers to this implementation style. The detailed results for specific FPGAs are given in the sub-chapter 4.4.3.

- 4-LUT

In 4-LUT, there are four different bits as an input. According to each combination of inputs, the output bit is stored in the Look-up table. In this case, the size of the table is $2^4 = 16$. The structure of 4-LUT is given in Figure 15. For interested readers, check [AMD Xilinx Company, 2022].

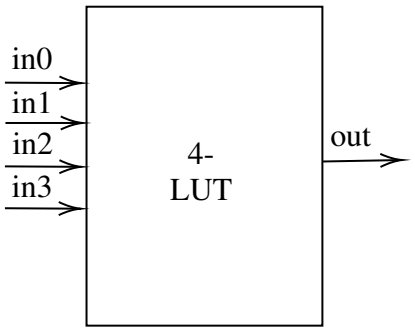


Figure 15: 4-Input Look-up Table with General Output

- 6-LUT

In 6-LUT, there are six different bits as an input. According to each combination of inputs, the output bit is stored in the Look-up table. In this case, the size of the table is $2^6 = 64$. The structure of 6-LUT is given in Figure 16. It consists of two 5-LUTs and one multiplexer. For interested readers, check [AMD Xilinx Company, 2022].

### 3.5.3 Arithmetic-Based Implementation

The last approach that we mentioned in this thesis is arithmetic-based implementation. The S-box function is performed directly by implementing the multiplicative inverse and affine transform. The affine transform is straightforward and costs less gate-area on hardware platforms. However, taking the multiplicative inverse in $\mathbb{GF}(2^8)$ is expensive. Several designs use arithmetic-based implementation. Optimizing the operations on the Galois field reduces the cost of implementing S-Box. [Satoh et al., 2001] used field isomorphism properties and proposed a new method to calculate the multiplicative inverse in $\mathbb{GF}(((2^2)^2)^2)$ instead of $\mathbb{GF}(2^8)$. Later, [Canright, 2005] optimized [Satoh et al., 2001] results and proposed new isomorphism matrices to represent S-Box.

In this thesis, we used a composite field approach to find a new S-Box that costs less gate-area than Canright results. Our purpose was not to optimize AES S-Box in this thesis. We aimed to find another S-Box that could be implemented with fewer gates on the hardware platform. The security that S-Box provides against cryptanalysis techniques is also concerned. At the end of our studies, we proposed a new S-box that can be implemented with fewer gates and provides at least the same security level of AES's S-Box. The details of our study is explained in the next chapter.

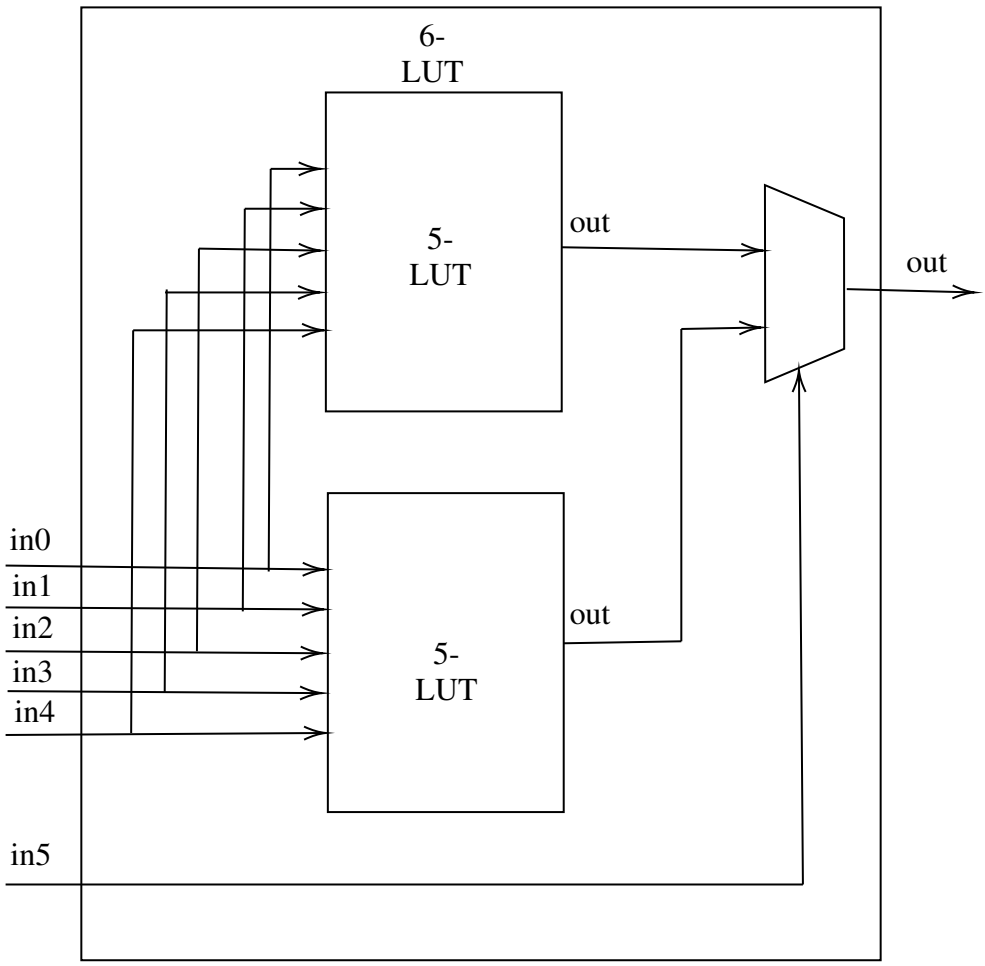


Figure 16: 6-Input Look-up Table with General Output

# CHAPTER 4

# THE PROPOSED COMPACT S-BOX FOR HARDWARE PLATFORMS

AES can be implemented in a variety of ways on hardware platforms. The implementation details are highly dependent on a target platform and specific FPGA. Pipelining, unrolling, datapath width, and frequency requirements are the methods/aspects that can affect the architecture of the implementation.

The approach of implementing S-Box is the most critical part of these design methods [[Bulens et al., 2008]]. This is because the S-Box layer is the most expensive to implement among the other levels of AES on hardware. There are three approaches to implement an S-Box on hardware platforms: logic-based, arithmetic-based, and RAM-based implementations. The details of these approaches are explained in the previous section 3.5.

The S-Box of AES is already implemented in a very compact way on hardware by using composite field approach of Galois field $\mathbb{GF}(2^8)$ [Canright, 2005], which is one of the approaches of arithmetic-based implementation. In this thesis, we aim to look for different primitive polynomial of finite field $\mathbb{GF}(2^8)$, which provides at least the same security level of AES's S-Box against linear and differential cryptanalysis and costs less gate area on hardware platforms.

## 4.1 Implementation Details

Irreducible polynomials are the polynomials which can not be expressed by the multiple of two non-constant polynomials. Primitive polynomials have at least one generator, as we defined in the previous section in detail. A polynomial which is both irreducible and primitive is called irreducible primitive polynomial.

By Gauss's formula, there are $30$ different irreducible primitive polynomials of degree 8 over $\mathbb{GF}(2^8)$ [Sunil and Jan, 2011]. The list of the irreducible primitive polynomials is given in the Table 10. The polynomial of AES S-Box is the first polynomial of the table, which is the smallest $8^{th}$ degree primitive.

$$S(x) = M * x^{-1} \oplus c,$$

is the S-Box affine transformation formula, where *M* is the $8 \times 8$ circulant binary matrix and *c* is a constant byte.

Table 10: List of all $8^{th}$ degree irreducible primitive polynomials in $\mathbb{GF}(2^8)$

| | Hexadecimal Form | Polynomial |
|---|---|---|
| 1 | $0x11b$ | $x^8+x^4+x^3+x^1+1$ |
| 2 | $0x11d$ | $x^8+x^4+x^3+x^2+1$ |
| 3 | $0x12b$ | $x^8+x^5+x^3+x^1+1$ |
| 4 | $0x12d$ | $x^8+x^5+x^3+x^2+1$ |
| 5 | $0x139$ | $x^8+x^5+x^4+x^3+1$ |
| 6 | $0x13f$ | $x^8+x^5+x^4+x^3+x^2+x^1+1$ |
| 7 | $0x14d$ | $x^8+x^6+x^3+x^2+1$ |
| 8 | $0x15f$ | $x^8+x^6+x^4+x^3+x^2+x^1+1$ |
| 9 | $0x163$ | $x^8+x^6+x^5+x^1+1$ |
| 10 | $0x165$ | $x^8+x^6+x^5+x^2+1$ |
| 11 | $0x169$ | $x^8+x^6+x^5+x^3+1$ |
| 12 | $0x171$ | $x^8+x^6+x^5+x^4+1$ |
| 13 | $0x177$ | $x^8+x^6+x^5+x^4+x^2+x^1+1$ |
| 14 | $0x17b$ | $x^8+x^6+x^5+x^4+x^3+x^1+1$ |
| 15 | $0x187$ | $x^8+x^7+x^2+x^1+1$ |
| 16 | $0x18b$ | $x^8+x^7+x^3+x^1+1$ |
| 17 | $0x18d$ | $x^8+x^7+x^3+x^2+1$ |
| 18 | $0x19f$ | $x^8+x^7+x^4+x^3+x^2+x^1+1$ |
| 19 | $0x1a3$ | $x^8+x^7+x^5+x^1+1$ |
| 20 | $0x1a9$ | $x^8+x^7+x^5+x^3+1$ |
| 21 | $0x1b1$ | $x^8+x^7+x^5+x^4+1$ |
| 22 | $0x1bd$ | $x^8+x^7+x^5+x^4+x^3+x^2+1$ |
| 23 | $0x1c3$ | $x^8+x^7+x^6+x^1+1$ |
| 24 | $0x1cf$ | $x^8+x^7+x^6+x^3+x^2+x^1+1$ |
| 25 | $0x1d7$ | $x^8+x^7+x^6+x^4+x^2+x^1+1$ |
| 26 | $0x1dd$ | $x^8+x^7+x^6+x^4+x^3+x^2+1$ |
| 27 | $0x1e7$ | $x^8+x^7+x^6+x^5+x^2+x^1+1$ |
| 28 | $0x1f3$ | $x^8+x^7+x^6+x^5+x^4+x^1+1$ |
| 29 | $0x1f5$ | $x^8+x^7+x^6+x^5+x^4+x^2+1$ |
| 30 | $0x1f9$ | $x^8+x^7+x^6+x^5+x^4+x^3+1$ |

Using composite field approach, Canright optimized the taking inverse of $x$ over $\mathbb{GF}(2^8)$, $M$ and $c$ are not considered in the optimization part. There are $256$ different $8 \times 8$ circulant binary matrices and $256$ different constant values. As we explained above, there are $30$ different irreducible primitive polynomials. Therefore, $30 \times 256 \times 256$ different look-up tables can be created in the considered affine transformation form. The look-up tables, which do not have one-to-one property, are ignored in this thesis.

- **Composite Field Approach**

The notation of [Canright, 2005] is used for compatibility. Let $g$ be an element of $\mathbb{GF}(2^8)$ and represented as g = $g_7g_6g_5g_4g_3g_2g_1g_0$, where $\forall\ g_i \in\ \mathbb{GF}(2)$. $g$ is also represented as

$$g(x) = g_7x^7 + g_6x^6 + g_5x^5 + g_4x^4 + g_3x^3 + g_2x^2 + g_1x^1 + g_0.$$

We need to convert an element of $g \in \mathbb{GF}(2^8)$, into new basis form. $g$ can be expressed by $\gamma_1 y^{16} + \gamma_0 y \in \mathbb{GF}(2^8)/\mathbb{GF}(2^4)$, where $\gamma_1$ and $\gamma_0 \in \mathbb{GF}(2^4)/\mathbb{GF}(2^2)$. Each $\gamma$ can be expressed by $\Gamma_1 z^4 + \Gamma_0 z$, where each $\Gamma \in \mathbb{GF}(2^2)/\mathbb{GF}(2)$. $\Gamma_1$ and $\Gamma_0$ are considered as a pair of bits $\beta_1$ and $\beta_0$, $\Gamma = \beta_1 w^2 + \beta_0 w$, where $\beta_0$ and $\beta_1 \in \{0, 1\}$. These conversions are possible thanks to field isomorphism between $\mathbb{GF}(2^8) \rightarrow \mathbb{GF}((2^4)^2) \rightarrow \mathbb{GF}(((2^2)^2)^2)$.

The purpose is to find equivalence of $g \in \mathbb{GF}(2^8)$ in $\mathbb{GF}(((2^2)^2)^2)$.

$$\begin{aligned} g &= \gamma_1 y^{16} + \gamma_0 y \\ &= (\Gamma_3 z^4 + \Gamma_2 z) y^{16} + (\Gamma_1 z^4 + \Gamma_0 z) y \\ &= ((\beta_7 w^2 + \beta_6 w) z^4 + (\beta_5 w^2 + \beta_4 w) z) y^{16} \\ &\quad + ((\beta_3 w^2 + \beta_2 w) z^4 + (\beta_1 w^2 + \beta_0 w) z) y \end{aligned}$$

After putting the g(x) into the equation,

$$\begin{aligned} g(x) &= g_7x^7 + g_6x^6 + g_5x^5 + g_4x^4 + g_3x^3 + g_2x^2 + g_1x + g_0 \\ &= (g_7x^7 + g_6x^6 + g_5x^5 + g_4x^4) y^{16} + (g_3x^3 + g_2x^2 + g_1x + g_0) y \\ &= ((g_7x^7 + g_6x^6) z^4 + (g_5x^5 + g_4x^4) z) y^{16} \\ &\quad + ((g_3x^3 + g_2x^2) z^4 + (g_1x + g_0) z) y \\ \\ &= ((g_7x^7w^2 + g_6x^6w) z^4 + (g_5x^5w^2 + g_4x^4w) z) y^{16} \\ &\quad + ((g_3x^3w^2 + g_2x^2w) z^4 + (g_1xw^2 + g_0w) z) y \end{aligned}$$

We will map $g(x) \rightarrow b(x)$, where $g \in \mathbb{GF}(2^8)$ and $b \in \mathbb{GF}(((2^2)^2)^2)$.

$$
\begin{aligned}
b(x) &= ((b_7w^2 + b_6w)z^4 + (b_5w^2 + b_4w)z)y^{16} \\
&\quad + ((b_3w^2 + b_2w)z^4 + (b_1w^2 + b_0w)z)y \\
&= (b_7w^2z^4 + b_6wz^4 + b_5w^2z + b_4wz)y^{16} \\
&\quad + ((b_3w^2z^4 + b_2wz^4 + b_1w^2z + b_0wz)y \\
&= b_7\boldsymbol{w^2z^4y^{16}} + b_6\boldsymbol{wz^4y^{16}} + b_5\boldsymbol{w^2zy^{16}} + b_4\boldsymbol{wzy^{16}} \\
&\quad + b_3\boldsymbol{w^2z^4y} + b_2\boldsymbol{wz^4y} + b_1\boldsymbol{w^2zy} + b_0\boldsymbol{wzy}
\end{aligned}
$$

The constant $w^i z^j y^k$ values will be the columns of the mapping matrix, where *i,j,k* $\in \mathbb{N}$.

$$
X = \begin{pmatrix}
\chi_{0,0} & \chi_{1,0} & \chi_{2,0} & \chi_{3,0} & \chi_{4,0} & \chi_{5,0} & \chi_{6,0} & \chi_{7,0} \\
\chi_{0,1} & \chi_{1,1} & \chi_{2,1} & \chi_{3,1} & \chi_{4,1} & \chi_{5,1} & \chi_{6,1} & \chi_{7,1} \\
\chi_{0,2} & \chi_{1,2} & \chi_{2,2} & \chi_{3,2} & \chi_{4,2} & \chi_{5,2} & \chi_{6,2} & \chi_{7,2} \\
\chi_{0,3} & \chi_{1,3} & \chi_{2,3} & \chi_{3,3} & \chi_{4,3} & \chi_{5,3} & \chi_{6,3} & \chi_{7,3} \\
\chi_{0,4} & \chi_{1,4} & \chi_{2,4} & \chi_{3,4} & \chi_{4,4} & \chi_{5,4} & \chi_{6,4} & \chi_{7,4} \\
\chi_{0,5} & \chi_{1,5} & \chi_{2,5} & \chi_{3,5} & \chi_{4,5} & \chi_{5,5} & \chi_{6,5} & \chi_{7,5} \\
\chi_{0,6} & \chi_{1,6} & \chi_{2,6} & \chi_{3,6} & \chi_{4,6} & \chi_{5,6} & \chi_{6,6} & \chi_{7,6} \\
\chi_{0,7} & \chi_{1,7} & \chi_{2,7} & \chi_{3,7} & \chi_{4,7} & \chi_{5,7} & \chi_{6,7} & \chi_{7,7}
\end{pmatrix} = \begin{pmatrix}
 & & & & & & & \\
 & & & & & & & \\
\chi_0 & \chi_1 & \cdot & \cdot & \cdot & & & \chi_7 \\
 & & & & & & & \\
 & & & & & & &
\end{pmatrix}
$$

where each $\chi_i$'s are defined as follow:

$$
\begin{aligned}
\chi_0 &= w^2z^4y^{16}, & \chi_4 &= w^2z^4y^1 \\
\chi_1 &= w^1z^4y^{16}, & \chi_5 &= w^1z^4y^1 \\
\chi_2 &= w^2z^1y^{16}, & \chi_6 &= w^2z^1y^1 \\
\chi_3 &= w^1z^1y^{16}, & \chi_7 &= w^1z^1y^1
\end{aligned}
$$

The matrix $X$ converts an element $g \in \mathbb{GF}(2^8)$ into new basis form, which is in the field $\mathbb{GF}(((2^2)^2)^2)$. As we can see, the mapping is decided for a choice of the basis denoted as bytes of $(y, z, w)$ values. $X$ denotes the isomorphism bit matrix throughout the chapter.

### 4.1.1 Generating Affine-based S-Box Table

S-Box table is the byte array whose length is 256. Each index of the byte array has its own corresponding S-Box value. In order to perform S-Box value of a given index *x*, we need to calculate $Mx^{-1} \oplus c$, where *c* is the constant byte and *M* is the circulant bit matrix. The algorithm of generating affine-based S-Box table is given in Algorithm 4. Flowchart of calculating S-box function with naive-way is shown in Figure 17.

**Algorithm 4:** Algorithm of Generating S-Box from poly, M and c

**Inputs:** $poly, M, c$

; /* poly is the $8^{th}$ degree irr-primitive poly */

; /* M is the circulant bit matrix. */

; /* c is the constant byte. */

**Output:** $sboxTable[256]$

; /* Output is the byte array of sized 256. */

$sboxTable[256] = [0, 0, \ldots, 0]$

**for** $x = 0, 1, 2, \ldots, 255$ **do**

; /* $x^{-1}$ is the multiplicative inverse of x in GF. */

$sboxTable[x] = M \times x^{-1} \bigoplus c$

**end**

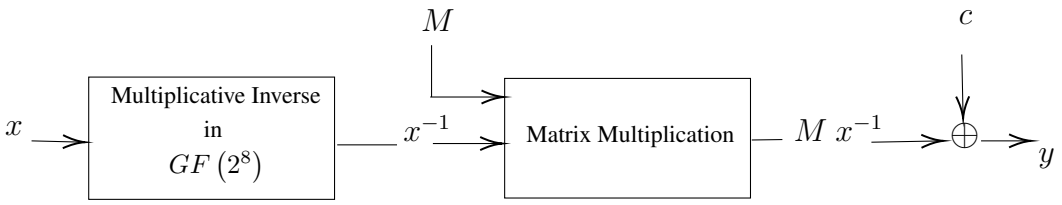


Figure 17: Flowchart of Calculating S-Box Function with Naive-way

### 4.1.2 Generating Isomorphism Bit Matrices of Given S-Box

Composite field is used to compute the multiplicative inverse efficiently. An element in the field $\mathbb{GF}(2^8)$ is mapped into an element in the $\mathbb{GF}(((2^2)^2)^2)$. The multiplicative inverse is taken in that field. Then the result is converted back to the initial field, which is $\mathbb{GF}(2^8)$. Flowchart of calculating S-box function with composite field approach is shown in Figure 18

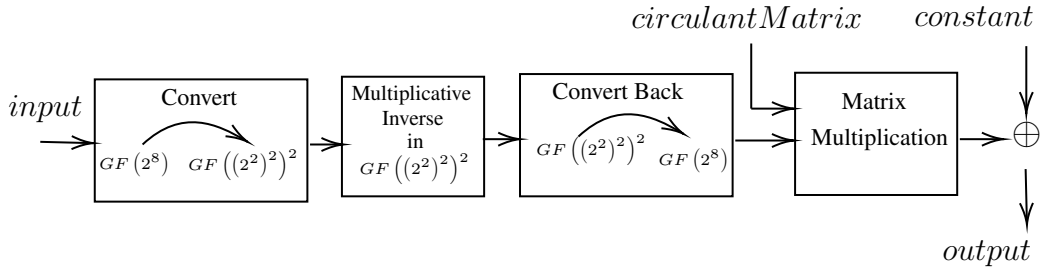


Figure 18: Flowchart of Calculating S-Box Function with Composite Field Approach

The conversion map *X* converts an element of the field g $\in \mathbb{GF}(2^8)$ to the element of the field b $\in \mathbb{GF}(((2^2)^2)^2)$. The inverse of the input is calculated in the domain

of the field $\mathbb{GF}(((2^2)^2)^2)$. The S-Box formula is $S(x) = M * x^{-1} \oplus c$, where $M$ is the circulant matrix and $c$ is the constant byte. We first change the form of the given input then calculate its inverse. This operation is exactly the same with multiplying calculated $X^{-1}$, where $X^{-1}$ is the inverse of $X$ over $\mathbb{GF}(2)$. Then, we multiply by $M \times X$ to change the basis back again. Having $X^{-1}$ and $M \times X$, it is enough to construct an S-Box function.

Instead of finding isomorphism functions between the fields one by one, we searched for basis values (*y*, *z* and, *w*), which will build *X* bit matrix at the end. Even though we used a brute-force approach for finding isomorphism bit matrices, we quickly generated all the possible isomorphism bit matrices. The algorithm of finding isomorphism bit matrices is given in Algorithm 5.

**Algorithm 5:** Algorithm of Generating Isomorphism Bit Matrices from S-Box Table

**Inputs:** $sboxTable, poly, M, c$
; /* sboxTable is the byte array of sized 256. */
; /* poly is the $8^{th}$ degree irr-primitive poly. */
; /* M is the circulant bit matrix. */
; /* c is the constant byte. */
**Output:** $isoList$
; /* The list of isomorphism bit matices. */

$isoList = [\ ]$
$sboxTable'[256] = [0, 0, \ldots, 0]$
**for** $y = 0, 1, 2, \ldots, 255$ **do**
  **for** $z = 0, 1, 2, \ldots, 255$ **do**
    **for** $w = 0, 1, 2, \ldots, 255$ **do**
      $X = generateX(y, z, w)$
      $sboxTable' = generateTable(M, X, c)$
      ; /* Compare both S-Boxes. */
      **if** $sboxTable' = sboxTable$ **then**
        $isoList.add(X)$
      **end**
    **end**
  **end**
**end**
**return** isoList

The search space is vast enough, so optimizing all the $8 \times 8$ S-Boxes was impossible. We have chosen one of the irreducible primitive polynomials from the Table 10. After choosing the polynomial, we chose one $8 \times 8$ binary circulant matrix and a constant byte value. We generated the S-Box with chosen parameters. Then we found isomorphism bit matrices of generated S-Box. There are more than one isomorphism bit matrix for each generated S-Box. Canright proposed $432$ different isomorphism bit matrices for AES S-Box. We used Python programming language to generate all the S-Boxes and their corresponding isomorphism bit matrices.

The time-consuming part was to synthesize matrices on hardware. On the hardware side, we have used VHDL language on Xilinx Vivado. Our purpose was to try all the possible isomorphism bit matrices and choose the smallest S-Box in terms of gate area with good algebraic properties. We have used Xilinx Vivado 21.1 and Xilinx ISE Suite 14.3 tools to synthesize S-Boxes. Synthesizing each isomorphism bit matrix takes approximately one minute in our computer, the specifications of our setup are as follow:

**Processor:** Intel(R) Xeon(R) Gold 62226R CPU 2.90 GHz
**Installed Memory(RAM):** 128 GB
**System Type:** 64-bit Operating System, Windows 10

There are $256 \times 256$ different S-Boxes for only one irreducible polynomial. Moreover, each S-Box has many different isomorphism bit matrices. There are $30$ different $8^{th}$ degree irreducible primitive polynomials by Gauss's formula. Therefore the time required to find the smallest affine-based S-Box for the hardware platform is approximately

$$30 \times 256 \times 256 \times 432 \times 1 \quad minutes \approx \ 1615,95 \quad years.$$

It requires a considerable amount of computational power to search all the space. Due to a license issue, we were unable to search our space across many cores. One license provides only one synthesis at a time. Therefore, we could only search some of the space but found what we sought. We came up with an S-Box that requires $3.125\%$ less gate-are and has the same algebraic properties as AES S-Box. Although the composite field approach finds more efficient way to implement an S-Box, some conversion matrices use more hardware gates than naive look-up table implementation. Therefore, we have to be careful while choosing the hardware-oriented bit matrices. The pseudo-algorithm of the finding the most compact $8 \times 8$ S-Box is given in Algorithm 6.

### 4.2 Construction of the Proposed S-Box

According to our search method, we have generated many S-Boxes. Some of them were weak in terms of algebraic properties. Some of them were costly. We focused on the S-Boxes, which have good algebraic properties and are efficiently implementable, among the generated S-Boxes. We desired to select the smallest and the most secure S-Box. We found many equivalent S-Boxes according to the gate area and security concerns. One of them was chosen to explain in detail.

**The parameters of the proposed S-Box as follows:**

**Algorithm 6:** Algorithm of Searching Compact S-Box

```
Inputs: polyList ;     /* 8^th degree irr-primitive poly list
 */
Output: X, M, c ;     /* X is the isomorphism bit matrix.
 */
;                /* M is the circulant bit matrix.  */
;                      /* c is the constant byte.  */
for poly ∈ polyList do
    for M = 0, 1, 2, ..., 255 do
        for c = 0, 1, 2, ..., 255 do
            sbox ← generateSbox(poly, M, c)
             isoList ← generateIsomorphismMatrices(sbox, poly, M, c)
            for X ∈ isoList do
                ;       /* Check algebraic properties.  */
                ;    /* Synthesize and find gate-area.  */
                ;      /* Keep the most compact S-Box.  */
            end
        end
    end
end
```

The irreducible primitive polynomial is

$$g(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1 \in \mathbb{GF}(2^8) \tag{4}$$

or $0x1f5$ in hexadecimal notation.

The circulant matrix is $M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$,

which is $0x45$ in hexadecimal notation. The constant value is $c = 0x09$.

The proposed S-Box is constructed in two steps. Let $a$ be the input and $s$ be the output of the S-Box function.

***Step 1:*** Find the multiplicative inverse of $a$ in $\mathbb{GF}(2^8)$

$$b = a^{-1}, \tag{5}$$

(if a = 0, then b = 0).

***Step 2:*** Calculate the following affine transformation.

$$\begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} \oplus \begin{pmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix}$$

Multiplication of $M$ and $b$ can be also expressed as:

$$b'_i = b_{(i+1)mod8} \oplus b_{(i+3)mod8} \oplus b_{(i+7)mod8} \oplus c_i \tag{6}$$

$$s = s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0$$
$$c = c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0$$
$$b = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

$s_i$'s, $c_i$'s and $b_i$'s are the bits of $s$, $c$, $b$ correspondingly and $0 \leq i < 7$.

All the calculations are computed in $\mathbb{GF}(2^8)$. Multiplication and addition of binary matrices do not increase the implementation cost. The difficulty of constructing an S-Box on hardware platforms come from taking the inverse of an element in the Galois field. Therefore, we focused on finding the multiplicative inverse of an element in the Galois field. The complete $8 \times 8$ proposed S-Box and inverse S-box are given in Table 11 and Table 12.

Table 11: The S-Box Table, generated by the following parameters: Irreducible primitive polynomial is $0x1f5$, the constant is $0x09$ and the circulant matrix is $0x45$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 09 | ab | fc | a5 | e2 | 3e | 50 | 6c | 6e | 4f | 67 | d3 | bb | 45 | 73 | 43 |
| 1 | c7 | 15 | 35 | 51 | d5 | 7e | bc | ee | dc | 8b | 91 | 1d | fd | 0a | 2d | 34 |
| 2 | 94 | b0 | 31 | e4 | c1 | af | b9 | 0b | 00 | f9 | e7 | f5 | 62 | 11 | 76 | b6 |
| 3 | 12 | 1c | 0c | 71 | 88 | f6 | 21 | 20 | e0 | b3 | bf | 26 | f1 | 7f | c3 | ac |
| 4 | 82 | cf | 7a | 9f | c9 | 3a | d2 | 3d | 28 | cb | f4 | b5 | 68 | 53 | bd | 37 |
| 5 | 1b | 47 | e8 | 80 | 64 | a6 | f0 | ce | df | a8 | 39 | 61 | f7 | 55 | c6 | a3 |
| 6 | 8f | 2c | 23 | b7 | 03 | 40 | 49 | 99 | ba | 9a | 46 | 4d | e9 | b8 | eb | cd |
| 7 | da | 4c | cc | ff | d4 | 13 | 57 | 19 | f8 | 22 | e5 | a9 | 9c | a1 | 42 | c0 |
| 8 | 1e | 78 | 84 | 33 | ef | 93 | 24 | 56 | 38 | c2 | 6f | 3b | be | 36 | d1 | 06 |
| 9 | fb | 3f | 8c | 1f | f2 | 52 | 70 | d8 | 7b | 79 | 0d | b4 | 60 | 18 | 75 | 8a |
| a | 9d | 66 | 25 | a0 | ca | dd | aa | 87 | 63 | 16 | e6 | 85 | fa | 5e | 86 | 17 |
| b | a4 | e1 | 4a | 5a | d9 | 90 | 69 | d7 | 44 | fe | b1 | 14 | 96 | 8e | ec | 10 |
| c | 04 | 01 | f3 | 72 | 5d | ed | c4 | 59 | ad | 41 | 9b | 0e | 89 | 6b | 98 | 2b |
| d | de | b2 | 2e | 95 | 27 | 5c | 81 | 65 | c8 | 4b | 6a | 1a | 7c | 4e | 30 | 0f |
| e | ae | c5 | 83 | 6d | 32 | db | 54 | 9e | 02 | 08 | 8d | a7 | 05 | d6 | 29 | 77 |
| f | ea | a2 | 5f | 7d | d0 | 97 | 48 | 5b | 92 | e3 | 58 | 07 | 2f | 74 | 2a | 3c |

Table 12: The Inverse S-Box Table, generated by the following parameters: Irreducible primitive polynomial is $0x1f5$, the constant is $0x09$ and the circulant matrix is $0x45$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 28 | c1 | e8 | 64 | c0 | ec | 8f | fb | e9 | 00 | 1d | 27 | 32 | 9a | cb | df |
| 1 | bf | 2d | 30 | 75 | bb | 11 | a9 | af | 9d | 77 | db | 50 | 31 | 1b | 80 | 93 |
| 2 | 37 | 36 | 79 | 62 | 86 | a2 | 3b | d4 | 48 | ee | fe | cf | 61 | 1e | d2 | fc |
| 3 | de | 22 | e4 | 83 | 1f | 12 | 8d | 4f | 88 | 5a | 45 | 8b | ff | 47 | 05 | 91 |
| 4 | 65 | c9 | 7e | 0f | b8 | 0d | 6a | 51 | f6 | 66 | b2 | d9 | 71 | 6b | dd | 09 |
| 5 | 06 | 13 | 95 | 4d | e6 | 5d | 87 | 76 | fa | c7 | b3 | f7 | d5 | c4 | ad | f2 |
| 6 | 9c | 5b | 2c | a8 | 54 | d7 | a1 | 0a | 4c | b6 | da | cd | 07 | e3 | 08 | 8a |
| 7 | 96 | 33 | c3 | 0e | fd | 9e | 2e | ef | 81 | 99 | 42 | 98 | dc | f3 | 15 | 3d |
| 8 | 53 | d6 | 40 | e2 | 82 | ab | ae | a7 | 34 | cc | 9f | 19 | 92 | ea | bd | 60 |
| 9 | b5 | 1a | f8 | 85 | 20 | d3 | bc | f5 | ce | 67 | 69 | ca | 7c | a0 | e7 | 43 |
| a | a3 | 7d | f1 | 5f | b0 | 03 | 55 | eb | 59 | 7b | a6 | 01 | 3f | c8 | e0 | 25 |
| b | 21 | ba | d1 | 39 | 9b | 4b | 2f | 63 | 6d | 26 | 68 | 0c | 16 | 4e | 8c | 3a |
| c | 7f | 24 | 89 | 3e | c6 | e1 | 5e | 10 | d8 | 44 | a4 | 49 | 72 | 6f | 57 | 41 |
| d | f4 | 8e | 46 | 0b | 74 | 14 | ed | b7 | 97 | b4 | 70 | e5 | 18 | a5 | d0 | 58 |
| e | 38 | b1 | 04 | f9 | 23 | 7a | aa | 2a | 52 | 6c | f0 | 6e | be | c5 | 17 | 84 |
| f | 56 | 3c | 94 | c2 | 4a | 2b | 35 | 5c | 78 | 29 | ac | 90 | 02 | 1c | b9 | 73 |

The input value's initial four bits are represented in the table's first column, and its final four bits are shown in the table's first row. The output of the S-Box function is determined by where the row and column intersect. For example, if input value is $0x7a$, then the intersection of the row with index $'7'$ and the column with index $'a'$ in Table 11 is $0xe5$.

### 4.3 Group Isomorphism of the Proposed S-Box

According to chosen irreducible primitive polynomial $0x1f5$, one of the conversion parameters is $(y, z, w) = (0x13, 0x7a, 0x5d)$. Therefore, the isomorphism bit matrix of the proposed S-Box is constructed as follow:

$$\begin{aligned} \chi_0 &= 0xf4, & \chi_4 &= 0xd2 \\ \chi_1 &= 0xec, & \chi_5 &= 0xc7 \\ \chi_2 &= 0x54, & \chi_6 &= 0x2e \\ \chi_3 &= 0xa2, & \chi_7 &= 0xd4 \end{aligned}$$

And the final conversion map is $X$ = $\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$.

This map $X$ converts an element in the field $\mathbb{GF}(((2^2)^2)^2)$ into an element of $\mathbb{GF}(2^8)$. The inverse of $X$ in $\mathbb{GF}(2)$ computes its reverse mapping.

The inverse conversion map is $X^{-1}$ = $\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$.

While looking for S-Box function of a given input, first we need to multiply by $X^{-1}$ to convert into the basis for $\mathbb{GF}(((2^2)^2)^2)$ and calculate its inverse in $\mathbb{GF}(2)$. After that, we need to change the basis back again and perform the affine transformation. Therefore, we multiply by $M \times X$ for simplicity.

$$M \times X = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

### 4.4 Results

We compared the results of our study to those of other researchers who had already worked on this similar topic, using the same setup and platforms. We downloaded the Xilinx ISE Design Suite 14.4 for Virex-4, Virex-5, and Virex-6 FPGAs and the Vivado 2021.1 for Artix-7 and Virtex-7 FPGAs. During synthesis, we used the identical FPGA components in the same configuration. Since implementing only S-Box occupies very small amount of space on an FPGA, the synthesis frequencies of the designs do not make sense. Therefore, we did not discuss synthesis frequency or synthesis period in our research.

The naive look-up table design is the complete 8x8 S-Box table, a byte array of size 256. Canright's matrices design implements AES S-Box in composite field arithmetic. The isomorphism bit matrices are taken from [Canright, 2005].

#### 4.4.1 Naive Look-up Table Implementation

This approach makes use of a full-byte array of size 256. This byte array can be distributed to LUTs or mapped into a BRAM. Some FPGAs only map to BRAM, which is both inefficient and undesirable. Since one BRAM, which is 18-Kbits or 32-Kbits, will be used for only the 256-bytes S-Box.

As a result, it is preferable to design S-Boxes in a distributed manner. Therefore, the distribution cost of the S-Box is taken into account in our comparison tables.

#### 4.4.2 Canright's Matrices for AES S-Box

[Canright, 2005] found 432 different isomorphism bit matrices for AES S-Box and proposed the smallest matrix in terms of gate-area in early 2005. The AES S-Box's irreducible primitive polynomial is $0x11b$, and the proposed conversion parameters is $(y, z, w) = (0xff, 0x5c, 0xbd)$. The most compact isomorphism bit matrix of the AES S-Box is proposed by [Canright, 2005]as follow:

$$\begin{aligned} \chi_0 &= 0x64, & \chi_4 &= 0x68 \\ \chi_1 &= 0x78, & \chi_5 &= 0x29 \\ \chi_2 &= 0x6e, & \chi_6 &= 0xde \\ \chi_3 &= 0x8c, & \chi_7 &= 0x60 \end{aligned}$$

The conversion map is $X = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$.

This map *X* converts an element from the field $\mathbb{GF}(((2^2)^2)^2)$ into an element of $\mathbb{GF}(2^8)$. The inverse of *X* in $\mathbb{GF}(2)$ computes its reverse mapping. *M* is the circulant matrix of AES S-Box.

The inverse conversion map of X is $X^{-1}$ = $\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$.

$$M \times X = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

The synthesis results of the given $X^{-1}$ and $M \times X$ matrices are used for comparison tables.

### 4.4.3 Comparisons

The synthesis result of the proposed S-Box and previous AES S-Box implementation results are compared in the Table 13 for the platform Virtex-4. The composite field approach of implementing the proposed S-Box is executed on Xilinx ISE Design Suite 14.4. Virtex-4 FPGA LUTs, as explained in the previous section, consist of 4-independent inputs and returns 1-bit output.The complete circuit is optimized by the Xilinx synthesis tool at the end. We observed that the isomorphism bit matrices of the proposed S-Box are optimized more than AES S-Box in terms of the number of used LUTs.

The AES S-Box implementation on Virtex-4 FPGAs was optimized by the researchers using a variety of techniques. Table 13 shows that for Virtex-4 xc4vlx25-10ff668, our synthesis results are 21% more compact than [Murugan et al., 2020]. For Virtex-4

Table 13: Synthesis result comparison of the proposed S-Box and the best optimized AES S-Box designs on Virtex-4 FPGAs for Only One Byte of Input

| Year | Used Resource | | Design | Platform |
|---|---|---|---|---|
| 2020 | 35 Slices | 69 LUTs | [Murugan et al., 2020] | Virtex4 - xc4vlx25-10ff668 (Xilinx ISE Design Suite 14.4 System Edition) |
| | 64 Slices | 128 LUTs | Naive Look-up Table 4.4.1 | |
| | 34 Slices | 60 LUTs | Canright's Matrices 4.4.2 | |
| 2023 | 30 Slices | 57 LUTs | Our work, Table 11 | |
| 2017 | 37 Slices | 71 LUTs | [Priya et al., 2017] | Virtex4 - xc4vlx200-11ff1513 (Xilinx ISE Design Suite 14.4 System Edition) |
| | 64 Slices | 128 LUTs | Naive Look-up Table 4.4.1 | |
| | 34 Slices | 60 LUTs | Canright's Matrices 4.4.2 | |
| 2023 | 32 Slices | 57 LUTs | Our work, Table 11 | |

Table 14: Synthesis result comparison of the proposed S-Box and the best optimized AES S-Box designs on Virtex-5 and Virtex-6 FPGAs for Only One Byte of Input

| Year | Used Resource | Design | Platform |
|---|---|---|---|
| 2008 | 32 LUTs | [Bulens et al., 2008] | Virtex5 - xc5vlx50t-3ff36 (Xilinx ISE Design Suite 14.4 System Edition) |
| 2015 | 32 LUTs | [Nadjia and Mohamed, 2015] | |
| 2019 | 172 LUTs | [Priya and Karthigaikumar, 2019] | |
| | 32 LUTs | Naive Look-up Table 4.4.1 | |
| | 32 LUTs | Canright's Matrices 4.4.2 | |
| 2023 | 32 LUTs | Our work, Table 11 | |
| 2015 | 32 LUTs | [Savalam and Korapati, 2015] | Virtex6 - xc6vlx240t-3ff784 (Xilinx ISE Design Suite 14.4 System Edition) |
| | 32 LUTs | Naive Look-up Table 4.4.1 | |
| | 32 LUTs | Canright's Matrices 4.4.2 | |
| 2023 | 32 LUTs | Our work, Table 11 | |

xc4vlx200-11ff1513, our circuit costs 24% less than Priya's results. [Priya et al., 2017] used mux-based S-Box implementation in their design. The multiplication in $\mathbb{GF}((2^2)^2)$ is computed with multiplexers instead of XOR gates. The study reduced the number of multipliers and decreased the critical path [Priya et al., 2017]. [Murugan et al., 2020] proposed a design which computes the multiplicative inverse in composite fields to some point. The multiplications in $\mathbb{GF}((2^2)^2)$ is stored in pre-calculated LUTs to decrease the number of LUTs in total design. According to [Canright, 2005]'s matrices, our S-Box is 11.76% and 5.9% smaller respectively on Virtex-4 xc4vlx25-10ff668 and Virtex xc4vlx200-11ff1513 FPGAs.

Virtex-5 and Virtex-6 FPGAs use the more recent iteration of LUTs. The Virtex-5 and Virtex-6 LUTs can use 6-independent inputs. With this improvement, FPGAs can produce more compact results; as seen in Table 14, the number of LUTs significantly dropped according to Virtex-4 FPGAs. The proposed S-Box implementation results are identical to those of other researchers. The only design that costs more than 32 LUT is [Priya and Karthigaikumar, 2019] design. Multiplication block is implemented with five-stage pipelining in the design of [Priya and Karthigaikumar, 2019]. Others researchers in Table 14 used the composite field approach and found the same results.

Table 15: Synthesis result comparison of the proposed S-Box and the best optimized AES S-Box designs on Virtex-7 and Artix-7 FPGAs for Only One Byte of Input

| Year | Used Resource | Design | Platform |
|---|---|---|---|
| 2019 | 39 LUTs | [Pradeep et al., 2019] | Artix7 - xc7a200tfbg676-2 (Xilinx Vivado 2021.1 ML Edition) |
| | 40 LUTs | Naive Look-up Table 4.4.1 | |
| | 32 LUTs | Canright's Matrices 4.4.2 | |
| 2023 | 31 LUTs | Our work, Table 11 | |
| 2019 | 61 LUTs | [Huy et al., 2019] | Virtex7 - xc7vlx980t-2ffg1926 (Xilinx Vivado 2021.1 ML Edition) |
| | 40 LUTs | Naive Look-up Table 4.4.1 | |
| | 32 LUTs | Canright's Matrices 4.4.2 | |
| 2023 | 31 LUTs | Our work, Table 11 | |

The proposed S-Box results differ from those of the other researchers for the new FPGA models Artix-7 and Virtex-7. This difference comes from the synthesis tool's ability to optimize the circuit. Compared to Pradeep's and Huy's designs for Artix-7 and Virtex-7, our result is 21% and 96% better, respectively. [Pradeep et al., 2019] preferred to use half-way composite field. The inverse operation is taken in the field $\mathbb{GF}((4^2)^2)$ instead of $\mathbb{GF}(((2^2)^2)^2)$. Canright's matrices cost at least same or more than the suggested S-Box on all platforms. The number of used LUTs for Virtex-7 and Artix-7 are given in Table 15 in detail.

These tables only show the required number of LUTs for a single byte. For the latest FPGA models, each byte requires around $32$ LUTs. AES uses blocks of a $16$-byte size. In order to implement the AES substitution layer, hardware platforms need $16$ pipelined S-Box tables for round function and $4$ pipelined S-Box tables for key schedule, which means that values in the table will be multiplied by $20$. The AES core is also used more than once for high-throughput devices. Some systems use even $256$ copies of AES core. In this case, the values in the table will be multiplied by $5120$. Therefore, even minor LUT enhancement will significantly impact high-throughput systems.

The implementation of [Hussain and Jamal, 2012] uses total $2444$ LUTs for AES core. He used naive-way implementation of S-Box in his design. If he use our S-Box instead of Rijndael S-Box, he would save $180$ LUTs more. Since implementing one S-Box in a naive-way costs $40$ LUTs on Virtex-7 FPGAs. Our S-Box costs only $31$ LUTs. Each round use $20$ S-Box tables, therefore he would saved $20 \times 9 = 180$ LUTs.

### 4.5 Security of the Proposed S-Box

The S-Box layer has a significant impact on security of the algorithm. Since this layer is the only non-linear layer of the algorithm. The S-Box provides security against linear and differential cryptanalysis. In addition to the hardware cost, the security of the proposed S-Box is also taken into account. The quality of an S-box requires a

comprehensive evaluation based on various criteria and security properties. Here are some factors to consider when assessing the quality of an S-box:

**Definition 4.5.1** (Difference Distribution Table, [Biham and Shamir, 1990])**.** Difference Distribution table of $8 \times 8$ S-Box *S* is defined as follow:

*DDT* is an $2^8 \times 2^8$ matrix with row $i \in \mathbb{F}_2^8$ and column $j \in \mathbb{F}_2^8$ equal to

$$\#\{x \in \{0,1\}^8 \mid S(x) \oplus S(x \oplus i) = j\}, \tag{7}$$

for any pair (*i, j*). The highest value in the difference distribution table, without considering the first entry, is called **differential uniformity of the S-Box**. Having small differential uniformity makes the S-Box table strong against differential cryptanalysis. The differential cryptanalysis is proposed by Biham and Shamir in 1990. For more information about difference distribution table and differential attack, see [Biham and Shamir, 1990].

**Definition 4.5.2** (Linear Approximation Table, [Matsui, 1993])**.** Linear approximation table of $8 \times 8$ S-Box *S* is defined as follow:

*LAT* is a $2^8 \times 2^8$ matrix with row $i \in \mathbb{F}_2^8$ and column $j \in \mathbb{F}_2^8$ equal to

$$\Big[\sum_{x=0}^{255}(i \cdot x) \oplus \big(j \cdot S(x)\big)\Big] - 128. \tag{8}$$

The maximum absolute value of linear approximation table, except the first entry, is called **linear uniformity of the S-Box**. The S-Boxes whose linear uniformity is small have strong resistance against linear cryptanalysis. The linear cryptanalysis is discovered by Mitsuru Matsui in 1993. For more information about linear approximation table and linear cryptanalysis, see [Matsui, 1993].

**Definition 4.5.3** (Boomerang Connectivity Table, [Cid et al., 2018])**.** Boomerang connectivity table of $8 \times 8$ S-Box *S* is defined as follow:

*BCT* is a $2^8 \times 2^8$ matrix with row $i \in \mathbb{F}_2^8$ and column $j \in \mathbb{F}_2^8$ equal to

$$\Big|\Big\{x \in \mathbb{F}_2^8 \mid S^{-1}\big(S(x) \oplus j\big) \oplus S'\big(S(x \oplus i) \oplus j\big) = i\Big\}\Big|. \tag{9}$$

The highest value in the boomerang connectivity table, ignoring the elements of the first row and column, is called **boomerang uniformity of the S-Box**. Lower boomerang uniformity provides higher security against the boomerang attacks. For detailed explanation of boomerang attacks and connectivity table, see [Wagner, 1999] and [Boura and Canteaut, 2018].

**Definition 4.5.4** (Differential Branch Number, [Sarkar and Syed, 2018])**.** The differential branch number of an S-Box S is defined as follow:

$$\min_{\substack{x \neq x' \\ x,x' \in \mathbb{F}_2^8}} \Big\{ wt(x \oplus x') + wt\big(S(x) \oplus S(x')\big) \Big\}, \tag{10}$$

where *wt(x)* is the Hamming weight of the vector *x*.

**Definition 4.5.5** (Linear Branch Number, [Sarkar and Syed, 2018])**.** The linear branch number of an S-Box S is defined as follow:

$$\min_{\substack{\alpha,\beta \in \mathbb{F}_2^8 \\ LAT(\alpha,\beta) \neq 0}} \Big\{ wt(\alpha) + wt(\beta) \Big\}, \tag{11}$$

where *wt(x)* is the Hamming weight of the vector *x* and *LAT($\alpha, \beta$)* denotes the entry at row $\alpha$ and column $\beta$ of linear approximation table of the S-Box S.

**Definition 4.5.6** (Almost Bent Function)**.** An $m \times m$ S-Box *S*, for *m* odd, is called almost bent function if its non-linearity is equal to

$$2^{m-1} - 2^{(m-1)/2}. \tag{12}$$

**Definition 4.5.7** (Involution Function)**.** An S-Box *S* is called involution function when

$$S(x) = S^{-1}(x), \tag{13}$$

for all $x \in \mathbb{F}_2^8$ where $S^{-1}(x)$ is the inverse of the S-Box *S*.

**Definition 4.5.8** (Bent Function)**.** An $m \times m$ S-Box *S* is called bent function if its non-linearity is equal to

$$2^{m-1} - 2^{m/2-1}. \tag{14}$$

**Definition 4.5.9** (Linear Structure)**.** An *n*-variable Boolean function *S* has a linear structure if there exists a nonzero $\alpha \in \mathbb{F}_2^n$ such that

$$f(x \oplus \alpha) \oplus f(x) \tag{15}$$

is a constant function.

**Definition 4.5.10** (Almost Perfect Nonlinear Function)**.** An $n \times n$ S-Box *S* is called almost perfect nonlinear function if the differential uniformity of the S-Box *S* is equal to 2.

**Definition 4.5.11** (Fixed Points)**.** If $x = S(x)$ then *x* is called fixed point of the S-Box *S*.

**Definition 4.5.12** (Monomial Function)**.** An S-Box *S* is called monomial function when *S* is a power function.

**Definition 4.5.13** (Permutation Function)**.** An S-Box *S* is called permutation function if all the entries are reordered.

**Definition 4.5.14** (Non-Linearity)**.** The minimum non-linearity of all component functions of S-Box *S* is called the non-linearity of S-Box *S*.

**Definition 4.5.15** (Monomial Function)**.** An S-Box *S* is called monomial function when *S* is a power function.

**Definition 4.5.16** (Balance)**.** An S-Box *S* is called balanced if all its component functions are balanced.

**Definition 4.5.17** (Maximal Degree)**.** Maximal degree is the maximal algebraic degree of all its component functions.

**Definition 4.5.18** (Minimal Degree)**.** Minimal degree is the minimal algebraic degree of all its component functions.

Table 16: As it is seen, the security properties of the proposed S-Box and AES S-Box are the same.

| Algebraic Property | AES S-Box | Our S-Box | AES Inverse S-Box | Our Inverse S-Box |
|---|---|---|---|---|
| Boomerang Uniformity | 6 | 6 | 6 | 6 |
| Differential Branch Number | 2 | 2 | 2 | 2 |
| Differential Uniformity | 4 | 4 | 4 | 4 |
| Has Fixed Points | False | False | False | False |
| Has Linear Structure | False | False | False | False |
| Is Almost Bent | False | False | False | False |
| Is Almost Perfect Nonlinear | False | False | False | False |
| Is Balanced | True | True | True | True |
| Is Bent | False | False | False | False |
| Is Involution | False | False | False | False |
| Is Monomial Function | False | False | False | False |
| Is Permutation | True | True | True | True |
| Linear Branch Number | 2 | 2 | 2 | 2 |
| Linearity | 32 | 32 | 32 | 32 |
| Minimal Degree | 7 | 7 | 7 | 7 |
| Max. Degree | 7 | 7 | 7 | 7 |
| Max. Diff. Prob. | 0.015625 | 0.015625 | 0.015625 | 0.015625 |
| Max. Diff. Prob. Absolute | 4 | 4 | 4 | 4 |
| Max. Linear Bias Absolute | 16 | 16 | 16 | 16 |
| Max. Linear Bias Relative | 0.625 | 0.625 | 0.625 | 0.625 |
| Non-Linearity | 112 | 112 | 112 | 112 |

The security of a block cipher lies in its ability to exhibit strong diffusion properties, ensuring that changes in a single input bit propagate through multiple output bits in a complex and unpredictable manner. This non-linearity and confusion introduced by the substitution layer enhance the resistance of the algorithm against differential and linear attacks, where attackers analyze patterns and correlations within the cipher.

In our comparison of the proposed S-box with the AES S-box, we evaluated them based on commonly used criteria for assessing S-box quality. The algebraic properties of both the proposed S-box and the AES S-box were analyzed and summarized in Table 16. It was found that the proposed S-box exhibits the same algebraic properties as the AES S-box.

# CHAPTER 5

# CONCLUSION

In this work, we have used the same optimization methods that Canright used to optimize the S-Box layer of AES. We aimed to find a more compact $8 \times 8$ S-Box that provides at least the same security level as the AES S-Box. We proposed another S-Box which provides the same level of security against cryptanalysis techniques with an 11.76% less gate-area on Virtex-4 FPGAs. Our S-Box costs 3.125% fewer gates on Artix-7 and Virtex-7 FPGAs and uses the equal resource on Virtex-5 and Virtex-6 FPGAs when it is compared to AES S-Box. The algebraic properties of the proposed S-Box confirm that it provides the same level of security as the AES S-Box. Thus, the proposed S-Box is a better alternative to the AES S-Box as it offers improved efficiency on Virtex-4, Virtex-7, and Artix-7 FPGAs.

Our proposed S-Box offers significant gate-area improvements, making it well-suited for resource-constrained IoT devices and enabling algorithm parallelism through increased S-Box replication. As a result, we assert that our S-Box is more compact and efficient compared to the AES S-Box. Cryptographers seeking an 8x8 S-Box can confidently incorporate our proposed S-Box into their designs, achieving the same level of security with improved efficiency over the AES S-Box.

Due to the vast search space, conducting a comprehensive analysis of the isomorphism bit matrices for all possible $8 \times 8$ S-Boxes on FPGA proved computationally infeasible. Although we were able to generate all the $8 \times 8$ S-Boxes, synthesizing each of them on FPGA was hindered by the limitation of multiple licenses. As a potential route for future research, optimizing the synthesis process and identifying the smallest S-Box would be valuable, as this could contribute significant insights into the field.

# REFERENCES


[Abdelrahman et al., 2017] Abdelrahman, A. A., Fouad, M. M., Dahshan, H., and Mousa, A. M. (2017). High performance cuda aes implementation: A quantitative performance analysis approach. In *2017 Computing Conference*, pages 1077–1085.

[AMD Xilinx Company, 2022] AMD Xilinx Company (2022). Vivado design suite 7 series fpga and zynq-7000 soc libraries guide (ug953). https://docs.xilinx.com/r/en-US/ug953-vivado-7series-libraries.

[An and SEO, 2020] An, S. W. and SEO, S. (2020). Highly efficient implementation of block ciphers on graphic processing units for massively large data. *Applied Sciences*, 10:3711.

[An and Seo, 2020] An, S. W. and Seo, S. C. (2020). Study on optimizing block ciphers (aes, cham) on graphic processing units. In *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pages 1–4.

[Aziz and Ikram, 2007] Aziz, A. and Ikram, N. (2007). Memory efficient implementation of AES s-boxes on FPGA. *J. Circuits Syst. Comput.*, 16(4):603–611.

[Biham and Shamir, 1990] Biham, E. and Shamir, A. (1990). Differential cryptanalysis of des-like cryptosystems. In Menezes, A. and Vanstone, S. A., editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer.

[Bouhraoua, 2010] Bouhraoua, A. (2010). Design feasibility study for a 500 gbit-s/s advanced encryption standard cipher/decipher engine. *Computers and Digital Techniques, IET*, 4:334 – 348.

[Boura and Canteaut, 2018] Boura, C. and Canteaut, A. (2018). On the boomerang uniformity of cryptographic sboxes. *IACR Trans. Symmetric Cryptol.*, 2018(3):290–310.

[Bulens et al., 2008] Bulens, P., Standaert, F., Quisquater, J., Pellegrin, P., and Rouvroy, G. (2008). Implementation of the AES-128 on virtex-5 fpgas. In Vaudenay, S., editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 16–26. Springer.

[Canright, 2005] Canright, D. (2005). A very compact s-box for AES. In Rao, J. and Sunar, B., editors, *Cryptographic Hardware and Embedded Systems - CHES*

*2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer.

[Cid et al., 2018] Cid, C., Huang, T., Peyrin, T., Sasaki, Y., and Song, L. (2018). Boomerang connectivity table: A new cryptanalysis tool. In Nielsen, J. B. and Rijmen, V., editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 683–714. Springer.

[Criado et al., 2014] Criado, J. M. G., Vega-Rodríguez, M. A., Sánchez-Pérez, J. M., and Pulido, J. A. G. (2014). Hardware security platform for multicast communications. *J. Syst. Archit.*, 60(1):11–21.

[Daemen and Rijmen, 2002] Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer.

[Delporte, 2016] Delporte, F. (2016). Spn toy cipher. https://www.iacr.org/authors/tikz/.

[Hussain and Jamal, 2012] Hussain, U. and Jamal, H. (2012). An efficient high throughput fpga implementation of aes for multi-gigabit protocols. In *2012 10th International Conference on Frontiers of Information Technology*, pages 215–218.

[Huy et al., 2019] Huy, D. Q., Duc, N. M., Khai, L. D., and Lung, V. D. (2019). Hardware implementation of AES with s-box using composite-field for WLAN systems. In *2019 IEEE-RIVF International Conference on Computing and Communication Technologies, RIVF 2019, Danang, Vietnam, March 20-22, 2019*, pages 1–6. IEEE.

[Iyer et al., 2011] Iyer, N., Anandmohan, P., Poornaiah, D., and Kulkarni, V. (2011). *Computational Intelligence and Information Technology*, volume 250, chapter Efficient hardware architectures for AES on FPGA, pages 249–257. Springer.

[Jean, 2015] Jean, J. (2015). Mode of operations. https://www.iacr.org/authors/tikz/.

[Knudsen et al., 2004] Knudsen, L. R., Dobbertin, H., and Robshaw, M. J. B. (2004). The cryptanalysis of the AES - A brief survey. In Dobbertin, H., Rijmen, V., and Sowa, A., editors, *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *Lecture Notes in Computer Science*, pages 1–10. Springer.

[Kundi et al., 2016] Kundi, D., Aziz, A., and Ikram, N. (2016). A high performance st-box based unified AES encryption/decryption architecture on FPGA. *Microprocess. Microsystems*, 41:37–46.

[Lipmaa et al., 2000] Lipmaa, H., Rogaway, P., and Wagner, D. A. (2000). Comments to nist concerning aes modes of operations: Ctr-mode encryption. In *Symmetric Key Block Cipher Modes of Operation Workshop, 18 Oct 2000, Baltimore, Maryland, USA*, pages 1–4.

[Maimut, 2017] Maimut, D. (2017). Mode of operations. https://www.iacr.org/authors/tikz/.

[Matsui, 1993] Matsui, M. (1993). Linear cryptanalysis method for DES cipher. In Helleseth, T., editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer.

[McGrew and Viega, 2004] McGrew, D. and Viega, J. (2004). The security and performance of the galois/counter mode (GCM) of operation. In Canteaut, A. and Viswanathan, K., editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer.

[Murugan et al., 2020] Murugan, A., Karthigaikumar, and Priya, S. S. (2020). Fpga implementation of hardware architecture with aes encryptor using sub-pipelined s-box techniques for compact applications. In *Automatika*, volume 61, pages 682–693. Taylor and Francis.

[Nadjia and Mohamed, 2015] Nadjia, A. and Mohamed, A. (2015). Efficient implementation of aes s-box in lut-6 fpgas. In *2015 4th International Conference on Electrical Engineering (ICEE)*, pages 1–4. IEEE.

[Nishikawa et al., 2017] Nishikawa, N., Amano, H., and Iwai, K. (2017). Implementation of bitsliced aes encryption on cuda-enabled gpu. In *Lecture Notes in Computer Science*, pages 273–287.

[Pradeep et al., 2019] Pradeep, A., Mohanty, V., Subramaniam, A. M., and Rebeiro, C. (2019). Revisiting AES sbox composite field implementations for fpgas. *IEEE Embed. Syst. Lett.*, 11(3):85–88.

[Priya and Karthigaikumar, 2019] Priya, S. S. and Karthigaikumar (2019). Fpga implementation of high speed compact s-box. In *International Journal of Pure and Applied Mathematics*, volume 119, pages 1703–1711. IEEE.

[Priya et al., 2017] Priya, S. S., Karthigaikumar, Mangai, S., and Das, K. G. (2017). An efficient hardware architecture for high throughput AES encryptor using MUX based sub pipelined s-box. *Wirel. Pers. Commun.*, 94(4):2259–2273.

[Rais and Qasim, 2010] Rais, M. and Qasim, S. M. (2010). Virtex-5 fpga implementation of advanced encryption standard algorithm. In *AIP Conference Proceedings*, volume 1239, pages 201–205.

[Reddy et al., 2011] Reddy, S. K., Sakthivel, R., and Praneeth, P. (2011). Vlsi implementation of aes crypto processor for high throughput. *International journal of advanced engineering sciences and technologies*, 6(1):022–026.

[Saggese et al., 2003] Saggese, G. P., Mazzeo, A., Mazzocca, N., and Strollo, A. G. M. (2003). An fpga-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In Cheung, P. Y. K., Constantinides, G. A., and de Sousa, J. T., editors, *Field Programmable Logic and Application, 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003, Proceedings*, volume 2778 of *Lecture Notes in Computer Science*, pages 292–302. Springer.

[Sarkar and Syed, 2018] Sarkar, S. and Syed, H. (2018). Bounds on differential and linear branch number of permutations. In Susilo, W. and Yang, G., editors, *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*, volume 10946 of *Lecture Notes in Computer Science*, pages 207–224. Springer.

[Satoh et al., 2001] Satoh, A., Morioka, S., Takano, K., and Munetoh, S. (2001). A compact rijndael hardware architecture with s-box optimization. In Boyd, C., editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer.

[Savalam and Korapati, 2015] Savalam, C. and Korapati, P. (2015). Implementation and design of aes s-box on fpga. *International Journal of Research in Engineering and Science*, 3(1):9–14.

[Schneider, 2016] Schneider, C. R. T. (2016). Mode of operations. https://www.iacr.org/authors/tikz/.

[Sunil and Jan, 2011] Sunil, C. and Jan, M. (2011). Counting irreducible polynomials over finite fields using the inclusion-exclusion principle. In *Mathematics Magazine*, volume 84, pages 369–371. arXiv.

[Tezcan, 2021] Tezcan, C. (2021). Optimization of advanced encryption standard on graphics processing units. *IEEE Access*, 9:67315–67326.

[Wagner, 1999] Wagner, D. A. (1999). The boomerang attack. In Knudsen, L. R., editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer.

[Wang and Ha, 2013] Wang, Y. and Ha, Y. (2013). Fpga-based 40.9-gbits/s masked AES with area optimization for storage area network. *IEEE Trans. Circuits Syst. II Express Briefs*, 60-II(1):36–40.

[William et al., 1978] William, E., Carl, M., John, S., and and, T. W. (1978). Message verification and transmission error detection by block chaining. *International Business Machines Corporation*.