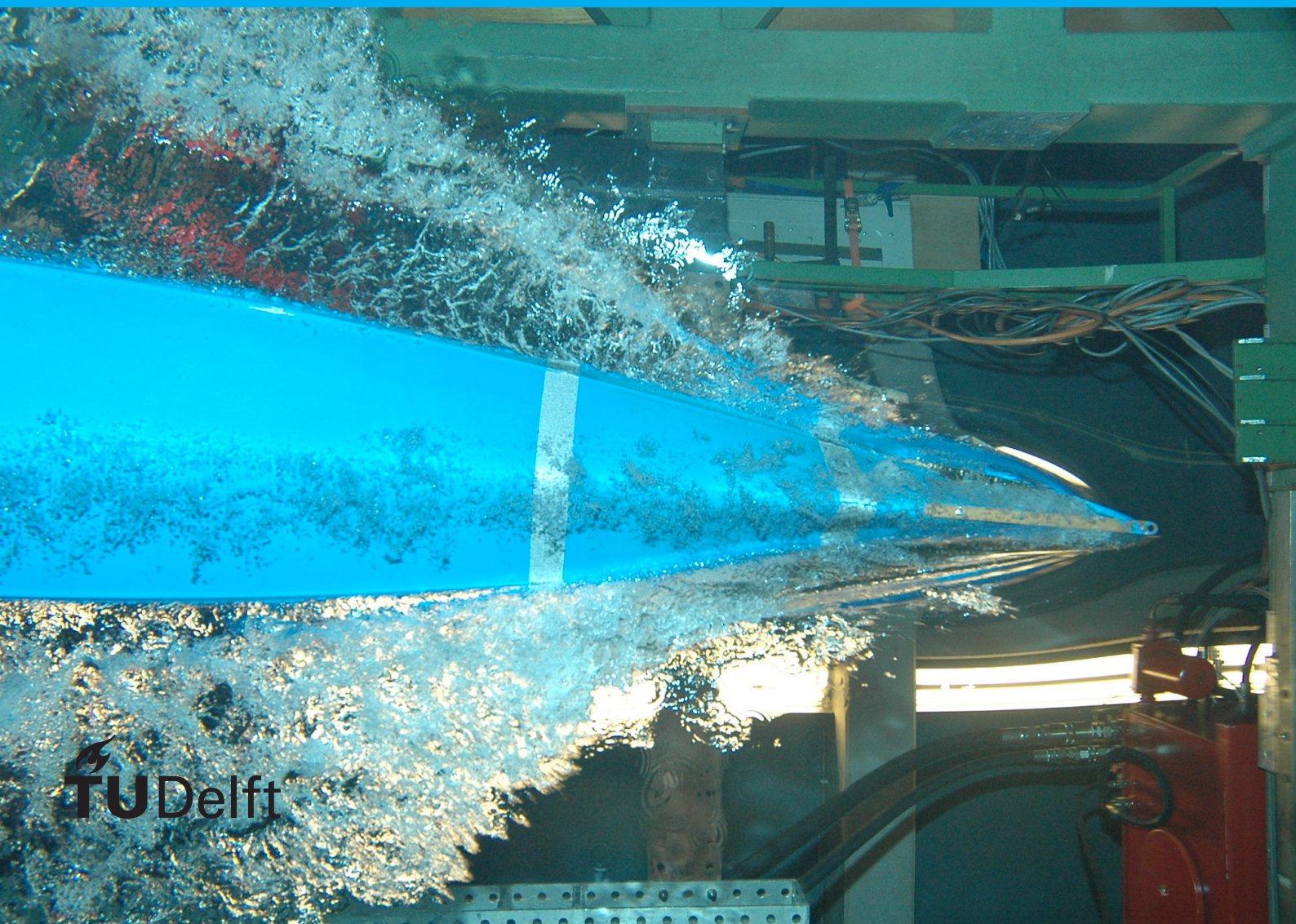
Securing an Efficient Lightweight AES Accelerator

Ruoyu Huang



Securing an Efficient Lightweight AES Accelerator

by

Ruoyu Huang

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday September 26th, 2023 at 16:00 PM.

Student number:	5529085	
Project duration:	October 1, 2022 – September 26, 2023	
Thesis committee:	Prof. Georgi Gaydadjiev,	TU Delft, responsible supervisor
	Dr. RangaRao Venkatesha Prasad,	TU Delft
	Dr.ir. Mottaqiallah Taouil,	TU Delft, daily supervisor
	Kezheng Ma,	Silicon Integrated, company supervisor

This thesis is confidential and cannot be made public until December 31, 2023.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

The thesis project started in October, 2022 and ended in September, 2023. During this year, I have been fortunate to receive substantial assistance and support. I would like to express my deepest gratitude to all those who have supported and guided me throughout the journey of completing this thesis.

First and foremost, I am immensely thankful to my thesis supervisors, Prof. Georgi Gaydadji, Dr.ir. Mottaqiallah Taouil and Kezheng Ma, for their consistent guidance, patience, and invaluable insights. Their mentorship has steered me into the realm of security and digital IC design, shaping my path in these domains.

I extend my heartfelt appreciation to Abdullah Aljuffri for his valuable guidance and constructive feedback that greatly enhanced the quality of this work.

Next, I would say thanks to people in Silicon Integrated B.V. Tianli Song, Quanhao Yu, Junwei Guo, Junyu Yang for sharing experiences and joy with me during my study.

Furthermore, I am deeply grateful to my parents Haidong Huang and Qing li for their unwavering support in enabling me to pursue my education abroad and shaping the person have become. I also want to express my heartfelt gratitude to my girlfriend, Jiaqi Wang, for making me feel beloved.

Lastly, I am thankful to all those whose names may not appear here, but whose contributions, whether small or large, have played a role in this achievement. Thank you.

Ruoyu Huang
Delft, The Netherlands
September 2023

Contents

List of Figures

List of Tables

Abbreviations

AES Advanced Encryption Standard 4

IoT Internet of Things 4

NIST National Institute of Standards and Technology 4

SBOX Substitution Box 4

DOM Domain-Oriented Masking 5

SCAs Side Channel Attacks 5

LUT Look-up Table 9

I-SBOX Inverse Substitution Box 9

SPA Simple Power Analysis 17

DPA Differential Power Analysis 17

CPA Correlation Power Analysis 17

HW Hamming Weight 19

HD Hamming Distance 19

PCC Pearson correlation coefficient 19

TBA Template Based Attack 20

TI Threshold Implementation 22

MQTT Message Queue Telemetry Transport 43

ADC Analogue-to-Digital Converter 43

TSMC Taiwan Semiconductor Manufacturing Company 43

Abstract

Internet of Things (IoT) devices regularly process sensitive data, including personal information. Therefore, ensuring their security is crucial to avoid damage and prevent data breaches. The Advanced Encryption Standard (AES) is generally regarded as one of the most popular cryptographic algorithms for ensuring data security. Typical lightweight implementations of the algorithm published in the literature focus on area and power optimization, while neglecting the performance. This paper presents a novel lightweight approach for the AES algorithm and considers both encryption and decryption. In terms of performance per unit area and performance per unit power, our 32-bit design outperforms the state-of-the-art by 1.69x and 1.27x, respectively. These improvements become even larger when implementing higher data-path designs, such as 64-bit or 128-bit designs. Our non-DOM AES design is secure against Correlation Power Analysis (CPA) but vulnerable to Template Based Attack (TBA) when more than 1500 traces are considered. To enhance its resilience against side channel attacks (SCAs), we modified our design by adopting and further improving on the most recent countermeasure, i.e., Domain-Oriented Masking (DOM). The results demonstrate that incorporating DOM into our design enables it to withstand against both CPA and TBA. Besides, our simplified eight-stage and five-stage 1^{st}-order DOM SBOX designs achieve a reduction in area of 9.9% and 6.9% compared to the original proposed designs, respectively.

1

Introduction

This chapter provides a brief introduction to the topics addressed in this thesis, highlights their significance, discusses the current state-of-the-art, outlines the main contributions, and presents the organization of the thesis. Section 1.1 discussed the importance of protecting IOT devices and current problem of implementing AES on these devices. Section 1.2 reviews the current state-of-the-art in lightweight AES designs and discusses the limitations of these designs. Section 1.3 elaborates on the contributions made by this thesis. Section 1.4 outlines the organization of the thesis.

1.1. Problem Statement

According to a study that was recently released in the World Economic Forum [1], malicious attacks were launched against 1.5 billion Internet of Things devices during the first half of 2021. The number of instances of data breaches increased by 15.1% as compared to the previous year. Fig. 1.2 [2] depicts the average weekly IoT cyber attacks per organization by sector, comparing the data from January to February in 2023 with that of 2022.

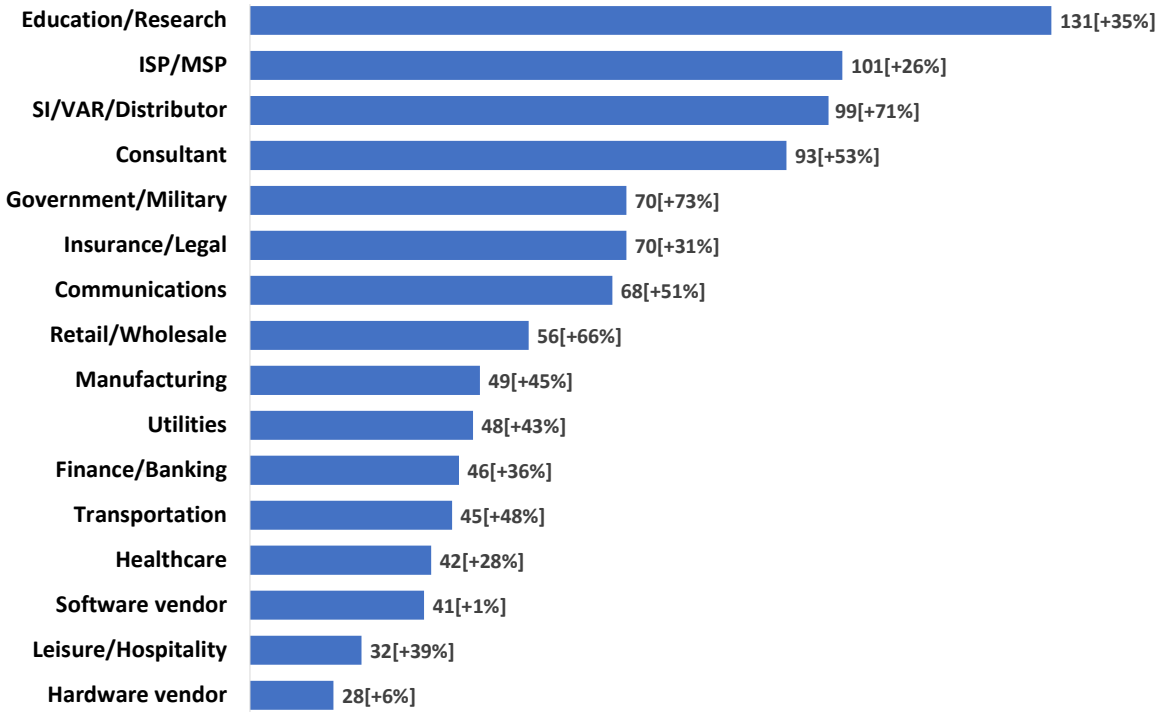


Figure 1.1: Average weekly IoT cyber attacks per organization by sector - Jan-Feb 2023 vs. 2022 [2]

This figure illustrates that the education and research sector is currently grappling with an unprecedented surge in attacks targeting IoT devices (131 weekly attacks per organization). Furthermore, other sectors are also witnessing a significant uptick in attacks, with the majority of them reporting double-digit growth when compared to the statistics from the year 2022.

As a consequence of this, the security of the Internet of Things (IoT) has become of the utmost importance and can no longer be treated as an afterthought. This is particularly true when considering the anticipated yearly increase in adoption of those devices, which is expected to reach 43 billion in the year 2023 [3]. According to the Fig. 1.2 [4], the IoT security market is positioned for substantial expansion, with expectations of an increase from $20.9 billion in 2023 to a substantial $59.2 billion by the year 2028.

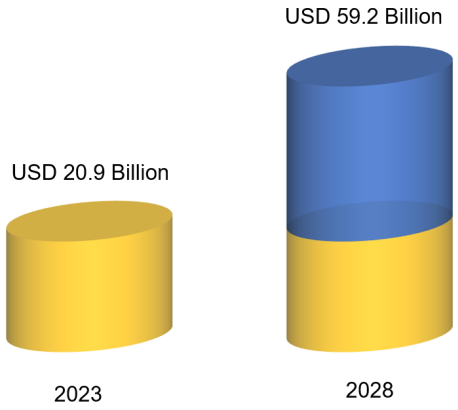


Figure 1.2: Global IoT Security Market Growth (2023-2028) [4]

With respect to data protection, the Advanced Encryption Standard (AES) [5] is one of the algorithms that is most generally recognized and utilized today. It was first presented to the public by the National Institute of Standards and Technology (NIST) in the year 2001. AES is currently the primary encryption method for many applications, including cloud computing [6] and health care [7]. Traditional implementations of the AES use pipelining techniques in order to achieve a high throughput [8–10]. However, as these implementations require a significant amount of memory and power, it is unfeasible to use them in IoT devices that are limited by area and battery capacity [11]. Therefore, implementations of AES should be area and power efficient, while simultaneously minimizing the impact on throughput to the greatest degree possible.

1.2. State-of-the-art

To overcome these challenges, several lightweight AES accelerators have been proposed, which reduce area and power consumption by shortening the data-path from the conventional 128-bit to 8-bit [12–15] i.e., reducing the number of Substitution Box (SBOX) from 16 to 1. Several researchers [16–18] further pursued the reduction of area requirements at the expense of additional cycles. Moradi et al. (2011) [16] improved the area by 23% using the Canright implementation technique at the performance cost of 216 cycles. Three years later, Mathew et al. [17] came up with a novel approach using just one SBOX copy, which allowed them to decrease the area, but it increased the amount of time required for the execution to 336 cycles. In 2019, Yu et al. [18] significantly improved the performance of their design by skillfully incorporating the most effective elements from all the previously mentioned designs. Their approach resulted in a design that maintains a comparable area to that of [17], with a lower cost of performance (i.e. 216 cycles). In general, 8-bit data-path designs significantly effect the throughput as at least 160 cycles are required per encryption. More recently, 32-bit designs have been proposed. Such designs achieve a better trade-off between performance and energy efficiency [19]. Most of the above articles focused or reported only on the encryption module. Davis and John [20] considered actually both modules and proposed optimizations across them. However, as will be shown in this paper later, the shared modules have not been fully optimized. Additionally, their reported area measurements only consider the encryption module. To fairly evaluate the designs and realize the best power/latency/area trade-off, it is important to provide the overhead of both the encryption and

decryption modules.

1.3. Main thesis contributions

This paper presents a novel low-area, low-power and low-latency AES hardware accelerator. Our method takes advantage of the fact that the key remains unchanged throughout a communication session, eliminating the need for repeated execution of the key expansion module. Additionally, for applications that demand a high-level of security, we integrate an improved version of the Domain-Oriented Masking (DOM) which is one of the most advanced countermeasures against Side Channel Attacks (SCAs). Our DOM-based AES design is more area-efficient in comparison to the original DOM design. In summary, the contributions of this paper are:

- **A proposal of a novel low-area, low-power, and low-latency design of the AES algorithm which is suitable for IoT applications.** In contrast to traditional lightweight AES designs, our approach considers both encryption and decryption, and further simplifies shared modules. Additionally, we optimize the performance by conducting key expansion initially and storing the generated key. When the key remains unchanged, there is no need to re-execute the key expansion module during subsequent encryption or decryption operations.
- **A proposal of a side channel resilient version using an improved DOM implementation.** To enhance the resilience of our AES designs against SCAs, we integrate DOM techniques into our designs. Furthermore, we simplify the DOM SBOX module to enhance its area efficiency.
- **Evaluation of the energy consumption, area overhead and performance of the proposed designs.** We synthesize our designs using TSMC 180 nm technology and conduct a thorough comparison with state-of-the-art implementations in terms of performance, power efficiency, and area utilization. The results demonstrate that our designs outperform the current state-of-the-art approaches.
- **Validation of the protected implementation using several side channel attacks.** We capture power traces for both AES and DOM designs and subject them to Correlation Power Analysis (CPA) and Template Based Attack (TBA). The results indicate that our proposed AES designs exhibit resilience against CPA but are vulnerable to TBA. However, the DOM-version AES designs demonstrate resistance to both CPA and TBA.

The design of AES and its DOM extension have been documented in the conference paper and have been accepted for presentation at the 2023 IEEE 22nd International Conference on Trust, Security, and Privacy in Computing and Communications (TrustCom-2023). Detailed information pertaining to the conference paper is available in Appendix A. Additionally, our work on extending the research for a journal publication, particularly concerning the security analysis, is currently underway.

1.4. Thesis Organization

This paper is organized as follows. Section 2 introduces the background on Galois Field, AES algorithm, Side Channel Attacks, and countermeasures of SCAs. Section 3 introduces our proposed AES designs and its DOM security extension. A comparison of the implementation results and security analysis are provided in Section 4. In particular, we first compare our unprotected AES designs with state-of-the-art designs, followed by a comparison of our proposed DOM multipliers with the original DOM multipliers. We then implement the proposed DOM multipliers on our unprotected AES designs and conduct a comparison. At last, we evaluate the security of our AES and DOM designs. Finally, Section 5 concludes this paper and offers insights into potential directions for future research.

2

Background

This chapter provides a brief background on Galois Field, AES algorithm, Side Channel Attacks, and countermeasures against SCAs. Section 2.1 provides a basic introduction to the Galois Field. In Section 2.2, the AES algorithm and its hardware implementation are presented in detail. Section 2.3 explores the concept of Side Channel Attacks (SCAs) and their application to AES designs. Lastly, Section 2.4 outlines various countermeasures against SCAs, including conventional masking, Threshold Implementation (TI), and Domain-Oriented Masking (DOM)

2.1. Galois Field (GF)

This section provides a brief introduction to the concept of the prime field, also known as a Galois Field.

2.1.1. Field

In mathematics, a field *F* is a set, along with two operations (addition $'+'$ and multiplication $'*'$) defined on that set [21]. These operations are required to satisfy the following properties:

- **Associativity**
  Addition: $a + (b + c) = (a + b) + c$;
  Multiplication: $a * (b * c) = (a * b) * c$.
- **Commutativity**
  Addition: $a + b = b + a$;
  Multiplication: $a * b = b * a$.
- **Inverse**
  Addition: For each element 'a' within the field 'F', there exists an element '-a' within the same field, satisfying $a + (-a) = 0$. Then '-a' denotes the additive inverse of 'a'.
  Multiplication: For each element 'a' within the field 'F', there exists an element a^{-1} or $1/a$ in F, satisfying $a * a^{-1} = 1$. Then a^{-1} or $1/a$ denotes the multiplicative inverse of 'a'.
- **Distributivity**
  $a * (b + c) = (a * b) + (a * c)$.

2.1.2. Prime Field

Prime Field or Galois Field (GF), a field F_p that contains a finite number of elements, is highly beneficial for translating computer data represented in binary form [22]. This field in binary form is comprised of the set of remainders modulo 2. Ahmad proposed that AES algorithm can be operated over the Galois Field GF(2^8) [23]. In GF(2^8), all bytes values are presented as the polynomial notation with 0 or 1. Assume p_7,p_6,...,and p_0 are binary values, then we can represent number N as $N = \sum_{i=0}^{7} p_i 2^i$ by selecting suitable values for b. There are two main operations in GF(2^8): addition and multiplication.

- **GF(2^8) Addition:**

In GF(2^8), addition is performed by applying a module-2 addition between the corresponding bits in the byte. In module-2 addition, the rules are: $0+0=0$, $0+1=1$, $1+0=1$, and $1+1=0$, which can be translated as XOR operation $\oplus$. For two GF(2^8) elements A $=\{a_7, a_6, ..., a_0\}$ and B $=\{b_7, b_6, ..., b_0\}$, their sum C $=\{c_7, c_6, ..., c_0\}$ can be calculated as $C(x) = A(x) + B(x) = \sum_{i=0}^{7}(a_i + b_i)x_i$.

- **GF(2^8) Multiplication**

  In GF(2^8) multiplication, the resulting polynomial may have a degree greater than 7. Therefore, it is necessary to perform modulo operation using an irreducible polynomial $m(x) = 1 + x + x^3 + x^4 + x^8$ to maintain the field characteristics. For two GF(2^8) elements A $=\{a_7, a_6, ..., a_0\}$ and B $=\{b_7, b_6, ..., b_0\}$, their multiplication D $=\{d_7, d_6, ..., d_0\}$ can be calculated as $D(x) = (\sum_{i=0}^{7} a_i x_i) \cdot (\sum_{i=0}^{7} b_i x_i)\ mod\ (1 + x + x^3 + x^4 + x^8)$.

2.2. Advanced Encryption Standard (AES)

AES is a symmetric cryptographic algorithm that is used in the cyber world for encrypting and decrypting data to protect them from cyberattacks. It has a fixed data block size of 128 bits, and a key length of 128, 192, or 256 bits. The key length determines the number of rounds required: 10, 12, or 14 rounds for 128-bit, 192-bit, or 256-bit key lengths, respectively. The 128-bit data block is divided into 16 bytes, which are mapped to a 4 × 4 array referred to as the State array. The diagram of AES encryption and decryption flow is presented in Fig. 2.1. Each round of encryption includes four primary modules: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*, except for Round 0 and Round N (see Fig. 2.1).

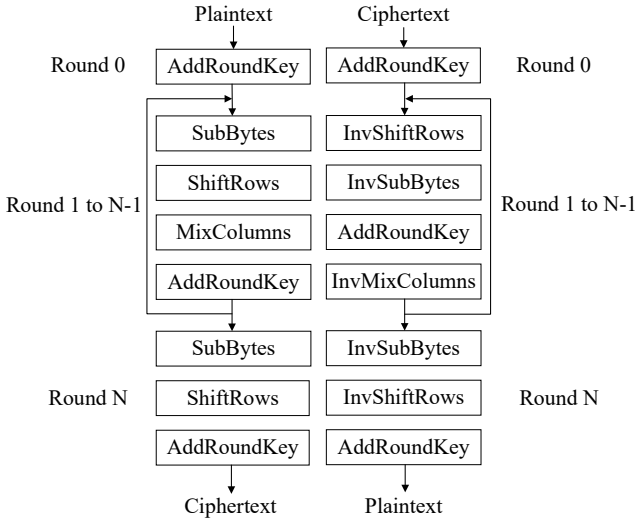


Figure 2.1: AES Encryption & Decryption Flow Diagram

2.2.1. AddRoundkey

AddRoundKey module involves bit-wise XOR operations of the round key and State array. It introduces the key into the data, guaranteeing that each encryption round employs a distinct, uniquely derived key originating from the original encryption key. In decryption, the ”AddRoundKey” module functions similarly to encryption, with the only difference being the input data. Equation (2.1) shows an example of AddRoundkey.

$$\begin{array}{|c|c|c|c|} \hline 20 & 01 & 02 & 03 \\ \hline 04 & 05 & 06 & 07 \\ \hline 08 & 09 & 0A & 0B \\ \hline 0C & 0D & 0E & 1F \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline B0 & B4 & B8 & BC \\ \hline B1 & B5 & 29 & BD \\ \hline B2 & B2 & BA & B1 \\ \hline B3 & C7 & BB & BF \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline D0 & B5 & BA & BF \\ \hline B5 & B0 & 2F & BA \\ \hline BA & BB & B0 & BC \\ \hline BF & CA & B4 & C0 \\ \hline \end{array} \tag{2.1}$$

2.2.2. SubBytes and InvSubBytes

SubBytes module is the only nonlinear module in the AES and plays a crucial role in defending against linear crypt-analysis [24]. When performing *SubBytes* module, each byte in the State array is substituted with another byte using Substitution Box (SBOX). *InvSubBytes* module reverses the *SubBytes* by substituting each byte of data in the State array using Inverse Substitution Box (I-SBOX) to facilitate the recovery of the original plaintext during the decryption process.

- SBOX
  The SBOX can be calculated using Look-up Table (LUT) in the Rijndael cipher [25], which is shown in Table 2.2. For instance, assuming the input of SBOX is ”8f6d”. According to the Table 2.2, ”8f” maps to ”73” and ”6d” converts to ”3c”. Consequently, the output of SBOX is ”733c”.

Table 2.1: Rijndael SBOX [25]

	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xa**	**xb**	**xc**	**xd**	**xe**	**xf**
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	fo	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	do	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	oe	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	od	bf	e6	42	68	41	99	2d	of	b0	54	bb	16

The SBOX can also be generated using a combination of a multiplicative inverse in GF(2^8) and an affine transformation [26], which is shown in Fig. 2.2. The input and output of the SBOX in Fig. 2.2 are $\{i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7\}$ and $\{o_0, o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$, respectively.

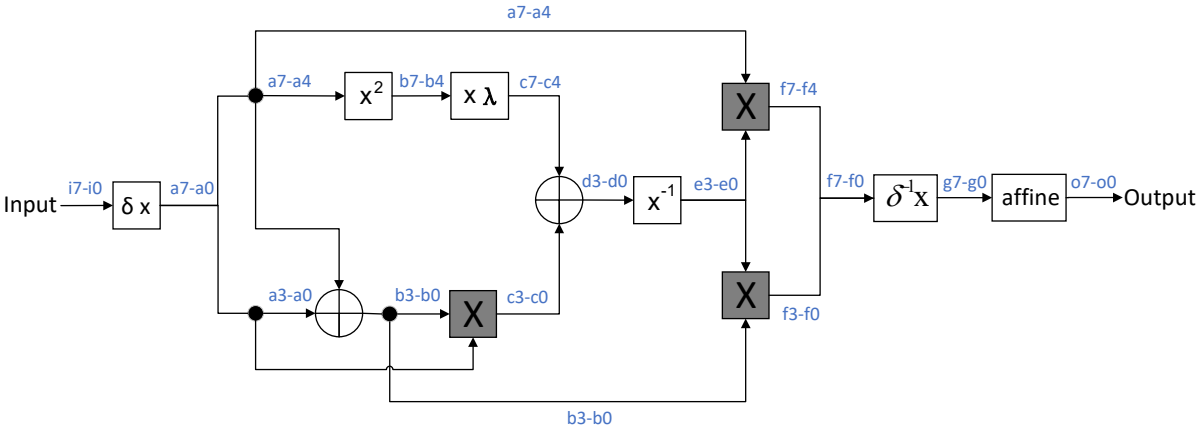


Figure 2.2: AES SBOX in GF(2^8) [26]

The SBOX includes δx, x^2, $x\lambda$, $\oplus$, X, x^{-1}, $\delta^{-1}x$, and ”affine” modules, which will be described in detail as follows:

1) δx module represents the isomorphic mapping transformation, which can be calculated as (2.2). $i_7 - i_0$ and $a_7 - a_0$ denote the 8-bit input and output of this module, respectively.

$$[ISO(a)] = \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{bmatrix} \tag{2.2}$$

$$= \begin{bmatrix} i_7 \oplus i_5 \\ i_7 \oplus i_6 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_5 \oplus i_3 \oplus i_2 \\ i_7 \oplus i_5 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_6 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_6 \oplus i_4 \oplus i_1 \\ i_6 \oplus i_1 \oplus i_0 \end{bmatrix}$$

2) x^2 module represents the GF(2^4) square calculation. In Fig. 2.2, we denote A=$\{a_7a_6a_5a_4\}$ and B=$\{b_7b_6b_5b_4\}$ as the 4-bit input and output of this module, respectively. These values can be expressed as extensions over the GF(2^2) field, such that $A = \{a_7a_6\}x + \{a_5a_4\} = A_Hx + A_L$, where $A_H = \{a_7a_6\}$, and $A_L = \{a_5a_4\}$. Similarly, $B = \{b_7b_6\}x + \{b_5b_4\} = B_Hx + B_L$, where $B_H = \{b_7b_6\}$ and $B_L = \{b_5b_4\}$. Note that $x^2 = x + \varphi$ in GF(2^2) and $x^2 = x + 1$ in GF(2), where $\varphi = \{1, 0\}$. Equations (2.3), (2.4), and (2.5) illustrate that $b_7 = a_7, b_6 = a_7 + a_6, b_5 = a_6 + a_5$, and $b_4 = a_7 + a_5 + a_4$.

$$\begin{aligned} B_H &= {A_H}^2 \\ &= (a_7x + a_6)^2 \\ &= a_7x^2 + a_6 \\ &= a_7(x + 1) + a_6 \\ &= a_7x + (a_7 + a_6) \end{aligned} \tag{2.3}$$

$$\begin{aligned} B_L &= {A_H}^2\varphi + {A_L}^2 \\ &= (a_7x + a_6)^2(1 \cdot x + 0) + (a_5x + a_4)^2 \\ &= (a_7x + a_7 + a_6)x + (a_5x + a_5 + a_4) \\ &= a_7x^2 + (a_7 + a_6 + a_5)x + a_5 + a_4 \\ &= a_7(x + 1) + (a_7 + a_6 + a_5)x + a_5 + a_4 \\ &= (a_6 + a_5)x + (a_7 + a_5 + a_4) \end{aligned} \tag{2.4}$$

$$\begin{aligned} B &= (A_Hx + A_L)^2 \\ &= {A_H}^2x^2 + {A_L}^2 \\ &= {A_H}^2(x + \varphi) + {A_L}^2 \\ &= {A_H}^2x + ({A_H}^2\varphi + {A_L}^2) \\ &= B_Hx + B_L \end{aligned} \tag{2.5}$$

3) $x\lambda$ module represents multiplication with a constant number $\lambda = \{1, 1, 0, 0\}$. B=$\{b_7b_6b_5b_4\}$ and C=$\{c_7c_6c_5c_4\}$ are the 4-bit input and output of this module, respectively (also see Fig. 2.2). In a similar manner as described in 2), $B = \{b_7b_6\}x + \{b_5b_4\} = B_Hx + B_L$, where $B_H = \{b_7b_6\}$, and $B_L = \{b_5b_4\}$. $C = \{c_7c_6\}x + \{c_5c_4\} = C_Hx + C_L$, where $C_H = \{c_7c_6\}$, and $C_L = \{c_5c_4\}$.

Equations (2.6), (2.7), and (2.8) illustrate that $c_7 = b_6 + b_4, c_6 = b_7 + b_6 + b_5 + b_4, c_5 = b_7$, and $c_4 = b_6$.

$$\begin{aligned} C_H &= B_H\lambda_H + B_L\lambda_H \\ &= B_H\lambda_H x^2 + (b_7 + b_6 + b_5 + b_4)x + (b_6 + b_4) \\ &= (b_7 + b_5)(x + 1) + (b_7 + b_6 + b_5 + b_4)x + (b_6 + b_4) \\ &= (b_6 + b_4)x + (b_7 + b_6 + b_5 + b_4) \end{aligned} \tag{2.6}$$

$$\begin{aligned} C_L = B_H\lambda_H\varphi &= (b_7x + b_6)\,(1 * x + 1)(1 * x + 0) \\ &= \left(b_7x^2 + b_7x + b_6x + b_6\right)x \\ &= (b_7 + b_6x + b_6)\,x \\ &= (b_7 + b_6)\,x + b_6x^2 \\ &= (b_7 + b_6)\,x + b_6(x + 1) \\ &= b_7x + b_6 \end{aligned} \tag{2.7}$$

$$\begin{aligned} C &= (B_Hx + B_L)(\lambda_Hx + \lambda_L) \\ &= B_H\lambda_Hx^2 + B_H\lambda_Lx + B_L\lambda_Hx + B_L\lambda_L \\ &= B_H\lambda_Hx^2 + B_L\lambda_Hx \\ &= B_H\lambda_H(x + \varphi) + B_L\lambda_Hx \\ &= (B_H\lambda_H + B_L\lambda_H)x + B_H\lambda_H\varphi \\ &= C_Hx + C_L \end{aligned} \tag{2.8}$$

4) $\oplus$ represents XOR operations. The $\{b_3b_2b_1b_0\}$ and $\{d_3d_2d_1d_0\}$ can be expressed as (2.9).

$$\begin{aligned} b_3 &= a_7 \oplus a_3; \\ b_2 &= a_6 \oplus a_2; \\ b_1 &= a_5 \oplus a_1; \\ b_0 &= a_4 \oplus a_0. \\ d_3 &= c_7 \oplus c_3; \\ d_2 &= c_6 \oplus c_2; \\ d_1 &= c_5 \oplus c_1; \\ d_0 &= c_4 \oplus c_0. \end{aligned} \tag{2.9}$$

5) X module in grey represents GF(2^4) multiplication. Take the left X module in Fig. 2.2 as an example. The inputs of X module are $A' = \{a_3a_2a_1a_0\}$ and $B' = \{b_3b_2b_1b_0\}$, respectively. The output of X module is $C' = \{c_3c_2c_1c_0\}$. Similarly in 2), $A' = \{a_3a_2\}x + \{a_1a_0\} = A'_Hx + A'_L$, where $A'_H = \{a_3a_2\}$, and $A'_L = \{a_1a_0\}$. $B' = \{b_3b_2\}x + \{b_1b_0\} = B'_Hx + B'_L$, where $B'_H = \{b_3b_2\}$ and $B'_L = \{b_1b_0\}$. $C' = \{c_3c_2\}x + \{c_1c_0\} = C'_Hx + C'_L$, where $C'_H = \{c_3c_2\}$ and $C'_L = \{c_1c_0\}$. Equation (2.10) illustrates that $C'_H = A'_HB'_H + A'_HB'_L + A'_LB'_H$ and $C'_L = A'_HB'_H\varphi + A'_LB'_L$.

$$\begin{aligned} C' &= (A'_Hx + B_L)(B'_Hx + B'_L) \\ &= A'_HB'_Hx^2 + A'_HB'_Lx + A'_LB'_Hx + A'_LB'_L \\ &= (A'_HB'_H + A'_HB'_L + A'_LB'_H)x + A'_HB'_H\varphi + A'_LB'_L \\ &= C'_Hx + C'_L \end{aligned} \tag{2.10}$$

According to the irreducible polynomial $x^2 + x + 1$ in GF(2^2), we can further simplify equa-

tion (2.10) by equation (2.11).

$$\begin{aligned}
A'_H B'_H &= (a_3x + a_2)(b_3x + b_2) \\
&= (a_3b_3 + a_3b_2 + a_2b_3)x + a_3b_3 + a_2b_2 \\
A'_H B'_L &= (a_3x + a_2)(b_1x + b_0) \\
&= (a_3b_1 + a_3b_0 + a_2b_1)x + a_3b_1 + a_2b_0 \\
A'_L B'_H &= (a_1x + a_0)(b_3x + b_2) \\
&= (a_1b_3 + a_1b_2 + a_0b_3)x + a_1b_3 + a_0b_2 \\
A'_L B'_L &= (a_1x + a_0)(b_1x + b_0) \\
&= (a_1b_1 + a_1b_0 + a_0b_1)x + a_1b_1 + a_0b_0 \\
A'_H B'_H \varphi &= (a_3x + a_2)(b_3x + b_2)(1 \cdot x + 0) \\
&= (a_3b_3 + a_3b_2 + a_2b_3)x^2 + (a_3b_3 + a_2b_2)x \\
&= (a_3b_2 + a_2b_3 + a_2b_2)x + a_3b_3 + a_3b_2 + a_2b_3.
\end{aligned} \tag{2.11}$$

Combining (2.10) and (2.11), we can achieve the (2.12).

$$\begin{aligned}
c_3 &= (a_3 + a_1)(b_3 + b_1) + (a_3 + a_1)(b_2 + b_0) + (a_2 + a_0)(b_3 + b_1) + \\
&\quad a_1b_1 + a_1b_0 + a_0b_1 \\
c_2 &= (a_3 + a_1)(b_3 + b_1) + (a_2 + a_0)(b_2 + b_0) + a_1b_1 + a_0b_0 \\
c_1 &= a_3b_3 + a_3b_2 + a_2b_3 + a_3b_3 + a_2b_2 + a_1b_1 + a_1b_0 + a_0b_1 \\
c_0 &= a_3b_3 + a_3b_2 + a_2b_3 + a_1b_1 + a_0b_0
\end{aligned} \tag{2.12}$$

6) x^{-1} represents the GF(2^4) inverter. According to [26], the inverter can be calculated as (2.13), where $\{d_3d_2d_1d_0\}$ and $\{e_3e_2e_1e_0\}$ are the inputs and the outputs of this module.

$$\begin{aligned}
e_3 &= d_3 + d_3d_2d_1 + d_3d_0 + d_2 \\
e_2 &= d_3d_2d_1 + d_3d_2d_0 + d_3d_0 + d_2 + d_2d_1 \\
e_1 &= d_3 + d_3d_2d_1 + d_3d_1d_0 + d_2 + d_2d_0 + d_1 \\
e_0 &= d_3d_2d_1 + d_3d_2d_0 + d_3d_1 + d_3d_1d_0 + \\
&\quad d_3d_0 + d_2 + d_2d_1 + d_2d_1d_0 + d_1 + d_0
\end{aligned} \tag{2.13}$$

Zhang in [26] also proposed the decomposition of GF(2^4) inverter, which is shown in Fig. 2.3. Here, all modules are $GF(2^2)$ and we will provide a comprehensive breakdown of the calculation process.

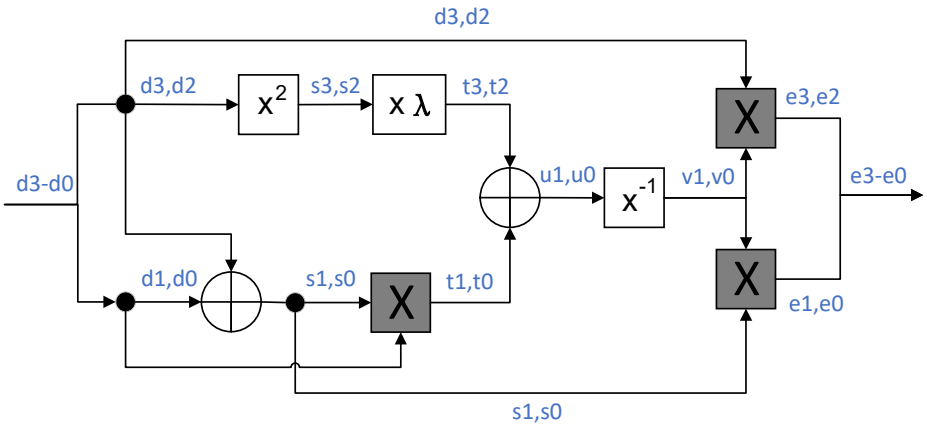


Figure 2.3: Decomposition of GF(2^4) inverter [26]

◦ $GF(2^2)\ x^2$ module
The inputs and outputs of this module are $D = \{d_3, d_2\}$ and $S = \{s_3, s_2\}$, respectively. Similarly in previous definitions, $D = d_3x + d_2$ and $S = s_3x + s_2$. Then the $GF(2^2)\ x^2$ can be calculated as equation (2.14), where $s_3 = d_3$ and $s_2 = d_3 + d_2$.

$$\begin{aligned} S &= (d_3x + d_2)^2 \\ &= {d_3}^2x^2 + {d_2}^2 \\ &= d_3(x + 1) + d_2 \\ &= d_3x + (d_3 + d_2) \\ &= s_3x + s_2 \end{aligned} \tag{2.14}$$

◦ $GF(2^2)\ x\lambda$ module
$GF(2^2)\ x\lambda$ represents multiplication with a constant number λ = {1, 0}. The inputs and outputs of this module are $S = \{s_3, s_2\}$ and $T = \{t_3, t_2\}$, respectively. Similarly in previous definitions, $S = s_3x + s_2$ and $T = t_3x + t_2$. Then the $GF(2^2)\ x\lambda$ can be calculated as equation (2.15), where $t_3 = s_3 + s_2$ and $t_2 = s_3$.

$$\begin{aligned} T &= (s_3x + s_2)(1 * x + 0)) \\ &= s_3x^2 + s_2x \\ &= s_3(x + 1) + s_2x \\ &= s_3 + s_2x + s_3 \\ &= t_3x + t_2 \end{aligned} \tag{2.15}$$

◦ $\oplus$ represents XOR operations, shown in equation (2.16).

$$\begin{aligned} s_1 &= d_3 \oplus d_1; \\ s_0 &= d_2 \oplus d_0; \\ u_1 &= t_3 \oplus t_1; \\ u_0 &= t_2 \oplus t_0. \end{aligned} \tag{2.16}$$

◦ $GF(2^2)$ "X" module
Take the top "X" module as an example. The inputs are $D = \{d_3, d_2\}$ and $V = \{v_1, v_0\}$, and the outputs are $E = \{e_3, e_2\}$. The calculation of "X" module is illustrated in equation (2.17), where $e_3 = d_3v_1 + d_2v_1 + d_3v_0$ and $e_2 = d_3v_1 + d_2v_0$.

$$\begin{aligned} E &= (d_3x + d_2)(v_1x + v_0)) \\ &= d_3v_1x^2 + (d_2v_1 + d_3v_0)x + d_2v_0 \\ &= d_3v_1(x + 1) + (d_2v_1 + d_3v_0)x + d_2v_0 \\ &= (d_3v_1 + d_2v_1 + d_3v_0)x + (d_3v_1 + d_2v_0) \\ &= e_3x + e_2 \end{aligned} \tag{2.17}$$

Similarly, we achieve the results of other two "X" modules, illustrated in equation (2.18).

$$\begin{aligned} t_1 &= d_1s_1 + d_0s_1 + d_1s_0; \\ t_0 &= d_1s_1 + d_0s_0; \\ e_1 &= v_1s_1 + v_0s_1 + v_1s_0; \\ e_0 &= v_1s_1 + v_0s_0. \end{aligned} \tag{2.18}$$

◦ $GF(2^2)\ x^{-1}$ module
In $GF(2^2)$, $x^{-1} = x^2$. Assuming that the inputs and outputs of this module are repre-

sented as u_1, u_0 and v_1, v_0, respectively, we can express the achieved results as equation (2.19).

$$\begin{aligned} v_1 &= u_1; \\ v_0 &= u_1 + u_0. \end{aligned} \tag{2.19}$$

7) The inverse isomorphic transformation is denoted as $\delta^{-1}x$. Equation (2.20) demonstrates the computation of the isomorphic transformation, where the input is represented as $f = \{f_7 f_6 f_5 f_4 f_3 f_2 f_1 f_0\}$ and the output as $g = \{g_7 g_6 g_5 g_4 g_3 g_2 g_1 g_0\}$.

$$[ISO^{-1}(g)] = \begin{bmatrix} g_7 \\ g_6 \\ g_5 \\ g_4 \\ g_3 \\ g_2 \\ g_1 \\ g_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} f_7 \\ f_6 \\ f_5 \\ f_4 \\ f_3 \\ f_2 \\ f_1 \\ f_0 \end{bmatrix} \tag{2.20}$$

$$= \begin{bmatrix} f_7 \oplus f_6 \oplus f_5 \oplus f_1 \\ f_6 \oplus f_2 \\ f_6 \oplus f_5 \oplus f_1 \\ f_6 \oplus f_5 \oplus f_4 \oplus f_2 \oplus f_1 \\ f_5 \oplus f_4 \oplus f_3 \oplus f_2 \oplus f_1 \\ f_7 \oplus f_4 \oplus f_3 \oplus f_2 \oplus f_1 \\ f_5 \oplus f_4 \\ f_6 \oplus f_5 \oplus f_4 \oplus f_2 \oplus f_0 \end{bmatrix}$$

8) Affine transformation module is a linear mapping, which is shown in (2.21). The input and output of this module are $g = \{g_7 g_6 g_5 g_4 g_3 g_2 g_1 g_0\}$ and the output as $o = \{o_7 o_6 o_5 o_4 o_3 o_2 o_1 o_0\}$.

$$[AT(o)] = \begin{bmatrix} o_7 \\ o_6 \\ o_5 \\ o_4 \\ o_3 \\ o_2 \\ o_1 \\ o_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} g_7 \\ g_6 \\ g_5 \\ g_4 \\ g_3 \\ g_2 \\ g_1 \\ g_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{2.21}$$

$$= \begin{bmatrix} g_7 \oplus g_6 \oplus g_5 \oplus g_4 \oplus g_3 \\ g_6 \oplus g_5 \oplus g_4 \oplus g_3 \oplus g_2 \oplus 1 \\ g_5 \oplus g_4 \oplus g_3 \oplus g_2 \oplus g_1 \oplus 1 \\ g_4 \oplus g_3 \oplus g_2 \oplus g_1 \oplus g_0 \\ g_7 \oplus g_3 \oplus g_2 \oplus g_1 \oplus g_0 \\ g_7 \oplus g_6 \oplus g_2 \oplus g_1 \oplus g_0 \\ g_7 \oplus g_6 \oplus g_5 \oplus g_1 \oplus g_0 \oplus 1 \\ g_7 \oplus g_6 \oplus g_5 \oplus g_4 \oplus g_0 \oplus 1 \end{bmatrix}$$

- I-SBOX

  The I-SBOX is essentially the SBOX operated in reverse [25]. The LUT of I-SBOX is shown in Table 2.1. I-SBOX can also be implemented in in Galois Field GF(2^8). The structure of I-SBOX is illustrated in Fig. 2.4. In comparison to Figure 2.2, the I-SBOX incorporates an initial step of utilizing the inverse affine transformation. Supposing that the inputs and outputs of inverse affine transformation are $\{i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7\}$ and $\{j_0, j_1, j_2, j_3, j_4, j_5, j_6, j_7\}$, respectively, we can express the calculation in equation (2.22). However, the remaining blocks within both the SBOX and I-SBOX modules remain identical.

Table 2.2: Rijndael I-SBOX [25]

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	Ob	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	Oa	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	of	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	fo	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	Od	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

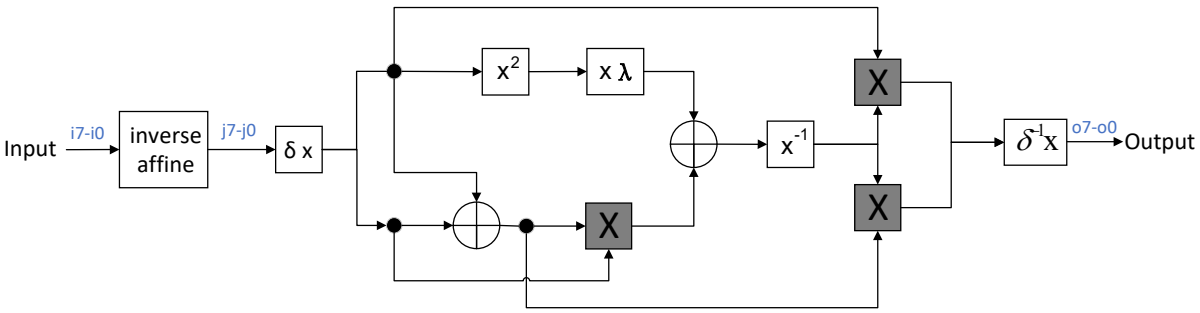


Figure 2.4: AES SBOX in GF(2^8) [26]

$$[AT(j)^{-1}] = \begin{bmatrix} j_7 \\ j_6 \\ j_5 \\ j_4 \\ j_3 \\ j_2 \\ j_1 \\ j_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} i_6 \oplus i_4 \oplus i_1 \\ i_5 \oplus i_3 \oplus i_0 \\ i_7 \oplus i_4 \oplus i_2 \\ i_6 \oplus i_3 \oplus i_1 \\ i_5 \oplus i_2 \oplus i_0 \\ i_7 \oplus i_4 \oplus i_1 \oplus 1 \\ i_6 \oplus i_3 \oplus i_0 \\ i_7 \oplus i_5 \oplus i_2 \oplus 1 \end{bmatrix} \tag{2.22}$$

2.2.3. ShiftRows and InvShiftRows

ShiftRows module is a transformation that cyclically shifts the second, third, and fourth rows of the State array by one, two, and three bytes to the left, respectively, while leaving the first row unchanged. The calculation of *ShiftRows* module is shown in equation (2.23). The *InvShiftRows* module is computed by performing the corresponding rotations to the right, which is calculated in equation (2.24).

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{bmatrix} \tag{2.23}$$

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,3} & S_{1,0} & S_{1,1} & S_{1,2} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,0} \end{bmatrix} \tag{2.24}$$

2.2.4. MixColumns and InvMixColumns

MixColumns and *InvMixColumns* modules perform a modular polynomial multiplication in Galois Field GF(2^8) on each column of the State array, shown in equation (2.25) and (2.26).

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \tag{2.25}$$

$$\begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} \tag{2.26}$$

To generate the round key, we need to perform key expansion. The key expansion process in AES, as described by Standards2001 [5], results in the generation of $4(N+1)$ 4-byte words, denoted as $w[i]$, where i ranges from 0 to $4(N+1)-1$. Here, N represents the round number in AES. The initial key can be divided into 4 words: key = (key[0], key[1], key[2], key[3]). Subsequently, each round key can be represented as: roundkey[i] = (w[4i], w[4i+1], w[4i+2], w[4i+3]). The pseudocode of 128-bit key expansion can be expressed below [5]. In the AES key expansion process, the **SubWord** operation applies *SubBytes* to each of the 4-byte input words. Additionally, the **RotWord** operation cyclically shifts each byte in a word one position to the left. The **Rcon** is a constant word array expressed as Rcon[i] = [x^{i-1}, 00, 00, 00], where x^{i-1} represents the powers of x (with x denoted as 02) in $GF(2^8)$.

2.3. Side channel attacks (SCAs)

Side channel attacks (SCAs) [27] take advantage of vulnerabilities by analyzing unintentional physical information that is disclosed during the device's normal execution. The focus of these attacks is on the physical attributes of a system, as opposed to a direct exploitation of software vulnerabilities. SCAs are capable of extracting confidential data, including cryptographic keys, from a system by analyzing the information that is unintentionally leaked. There are various physical information of a system, such as its power consumption, electromagnetic radiation, timing, and acoustic emissions, which can offer insights into its internal functioning [28,29]. Among them, power consumption is one of the most widely used information, primarily due to its high success rate and straightforward execution. The power consumption of a device is influenced by two main factors. The first factor is dynamic power, which is generated by the switching activities of transistors within the device. The second factor is leakage power, which occurs when a transistor exhibits an undesired behavior by generating leakage current in its off state [30]. In general, adversaries are typically interested in capturing the dynamic power

Algorithm 1 Pseudocode of 128-bit key expansion

```
1: begin
2: i=0;
3: while i<4 do
4:    w[i]=key[i];
5:    i=i+1;
6: end while
7: i=4;
8: while i<4(N+1) do
9:    temp=w[i-1];
10:   if i mod 4 = 0 then
11:      temp=SubWord(RotWord(w[i-1])) XOR Rcon(i/4);
12:   else
13:      w[i] = w[i-4] XOR temp
14:   end if
15: end while
16: i=i+1;
17: end
```

signals because they provide direct insight into the functional behavior of the device, revealing specific operations being performed. For example, the dynamic power of an inverter is influenced by the switching activities of both its input and output signals. When the input signal undergoes a transition from low to high or high to low, the internal transistors of the inverter switch states, resulting in a temporary increase in current flow and power consumption. Additionally, when the output signal transitions, the charging or discharging of the output load capacitance leads to further power consumption. The switching activities of both the input and output signals of an inverter contribute to the overall dynamic power consumption. By analyzing the power consumption patterns during different input combinations and output transitions, an adversary may attempt to extract sensitive information or gain insights into the behavior of the circuit.

In SCAs of AES, the adversary performs a statistical analysis on power consumption measurements obtained at an intermediate target, such as the SBOX operation. These measurements will then be connected to a leakage model [31], to obtain the secret information. Leakage models rely on assumptions in which confidential information is computed, taking into account the operations and switching activities involved. SCAs can be categorized into two types: non-profiled attacks and profiled attacks. A brief description of each class is provided below.

2.3.1. Non-profile power attacks

In these types of attacks, the adversary does not have access to any pre-measured power records of the device that is being targeted. Instead, he or she takes power measurements while the system is running and examines those measurements without any sort of reference. Examples of such attacks are Simple Power Analysis (SPA) [32], Differential Power Analysis (DPA) [32], and Correlation Power Analysis (CPA) [31]. SPA attack observes the differences in power consumption trace of a cryptographic device or system without sophisticated statistical analysis. Although sole SPA might not be sufficient to directly deduce the secret key using this method, it does possess the capability to discern the cryptographic algorithm being executed on a device, and enable more powerful attacks which specifically exploit any weakness of an algorithm to take place [33].

DPA and CPA are more powerful attacks. DPA relies on the statistical analysis of power traces obtained from multiple iterations of the cryptographic operation [32], while CPA analyzes the correlation between power consumption traces and known information [31]. The following outlines the steps involved in a DPA and CPA attack on the AES algorithm.

- **DPA**

  1) Select the point of attack: $S = SBOX(P \oplus K)$, with P as the plaintext, K as the key, and SBOX as the SubByte calculation.

2) Measure power traces $t_1, ..., t_d$ of plaintext $P_1, ..., P_d$. ”d” number of power traces ”T” with ”N” points can be expressed as (2.27).

$$\mathrm{T} = \begin{bmatrix} t_{1,1} & \cdots & t_{1,N} \\ \vdots & \ddots & \vdots \\ t_{d,1} & \cdots & t_{d,N} \end{bmatrix} \tag{2.27}$$

3) Compute hypothetical SBOX value for a single byte of the plaintext P_i, which is shown in (2.28).

$$\mathrm{S} = \begin{bmatrix} S_{1,0} & \cdots & S_{1,255} \\ \vdots & \ddots & \vdots \\ S_{d,0} & \cdots & S_{d,255} \end{bmatrix} S_{i,j} \tag{2.28}$$

$$= SBOX(P_i \oplus K_j)$$

4) For each subkey, classify the traces into two sets based on a target bit b of S_j. If bit is 0, the power trace will be added to set S_0; otherwise S_1.

5) Compute the average power signal for each set S. The different trace containing the max value among all traces indicates the most probable key.

6) Repeat steps 3 to 6 for all possible bytes of the key.

Fig. 2.5 shows an example of the DPA results (1000 power traces) [32]. Positioned at the uppermost part is a reference trace representing power consumption. The trace in the second line illustrates the standard deviation within the measurements of power consumption. The third and fourth trace show incorrect guesses for subkey.

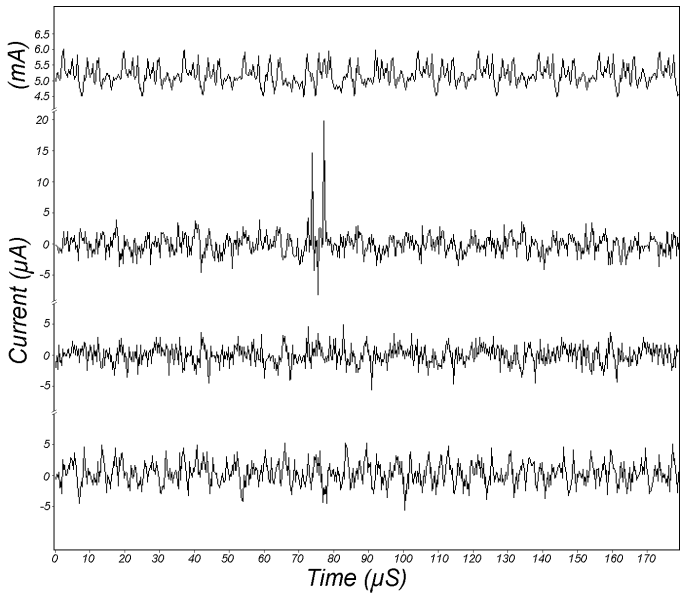


Figure 2.5: Differential Power Analysis (DPA) traces showing power references, a valid guess, and two incorrect guesses (from [32])

- **CPA**

1) Select the point of attack: $S = SBOX(P \oplus K)$, with P as the plaintext, K as the key, and SBOX as the SubByte calculation.

2) Measure power traces $t_1, ..., t_d$ of plaintext $P_1, ..., P_d$. "d" number of power traces "T" with "N" points can be expressed as (2.29).

$$\mathrm{T} = \begin{bmatrix} t_{1,1} & \cdots & t_{1,N} \\ \vdots & \ddots & \vdots \\ t_{d,1} & \cdots & t_{d,N} \end{bmatrix} \tag{2.29}$$

3) Compute hypothetical SBOX value for all possible values of the target subkey K_j, which is shown in (2.30).

$$\begin{aligned} \mathrm{S} &= \begin{bmatrix} S_{1,1} & \cdots & S_{1,255} \\ \vdots & \ddots & \vdots \\ S_{d,1} & \cdots & S_{d,255} \end{bmatrix} S_{i,j} \\ &= SBOX(P_i \oplus K_j) \end{aligned} \tag{2.30}$$

4) Compute hypothetical power H by using a power leakage model (2.31), where H is calculated with Hamming Weight or Hamming Distance [31].

   ◦ **Hamming Weight (HW) Model**
   The HW model is particularly useful in various applications, including error detection and correction codes, cryptography, signal processing, and data compression [34]. Mathematically, the HW of a binary sequence is calculated by counting the number of '1' bits in that sequence. For example, the HW of the binary sequence "11011101" is 6, as it contains six '1' bits.

   ◦ **Hamming Distance (HD) Model**
   The HD is a concept used in information theory and computer science to measure the dissimilarity between two strings of equal length [35]. To calculate the HD, one compares each symbol in the first string with the corresponding symbol in the second string. If the symbols at a particular position are different, the HD is incremented. The process continues until all positions are compared, providing the final count of differences. For example, the HD of the binary sequences "11011101" and "11010000" is 3, as there are three different bits between these two sequences.

$$\mathrm{H} = \begin{bmatrix} h_{1,0} & \cdots & h_{1,255} \\ \vdots & \ddots & \vdots \\ h_{d,0} & \cdots & h_{d,255} \end{bmatrix} \tag{2.31}$$

5) Guess sub-key value from the max value obtained in Pearson Correlation [31] between Hamming Weight/Distance H and power trace T.

   ◦ **Pearson correlation coefficient (PCC) Model**
   PCC is a correlation coefficient that quantifies the linear correlation existing between two sets of data [31]. Its resultant value invariably falls within the range of -1 to 1. This correlation coefficient between two variables, X and Y, can be defined as (2.32).

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}} \tag{2.32}$$

   Cov is the covariance, which can be expressed in equation (2.33), where $\mathbb{E}$ denotes the expectation of variables. Var (σ^2) denotes the variance, then σ_X^2 and σ_Y^2 can be defined as (2.34).

$$\begin{aligned} \mathrm{cov}(X,Y) &= \mathbb{E}\left[(X - \mu_X)\,(Y - \mu_Y)\right] \\ &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \end{aligned} \tag{2.33}$$

$$\begin{aligned}\sigma_X^2 &= \mathbb{E}\left[(X-\mathbb{E}[X])^2\right] \\ &= \mathbb{E}\left[X^2\right]-(\mathbb{E}[X])^2; \\ \sigma_Y^2 &= \mathbb{E}\left[(Y-\mathbb{E}[Y])^2\right] \\ &= \mathbb{E}\left[Y^2\right]-(\mathbb{E}[Y])^2.\end{aligned} \tag{2.34}$$

When considering the n samples, the PCC equation can be defined as (2.35)

$$r_{xy}=\frac{\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i-\mu_x)^2}\sqrt{\sum_{i=1}^{n}(y_i-\mu_y)^2}} \tag{2.35}$$

If we define the $g_{i,j}$ as the Pearson Correlation between H and T, the $g_{i,j}$ can be expressed as (2.36), where d is the number of power traces. The array of $g_{i,j}$ is shown in (2.37).

$$\mathrm{g_{i,j}}=\frac{\sum_{k=1}^{d}\left(h_{k,i}-\mu_{H,i}\right)\left(t_{k,j}-\mu_{t,j}\right)}{\sqrt{\sum_{k=1}^{d}\left(h_{k,i}-\mu_{H,i}\right)^2\sum_{k=1}^{d}\left(t_{k,j}-\mu_{t,j}\right)^2}} \tag{2.36}$$

$$\mathrm{G}=\begin{bmatrix} g_{1,1} & \cdots & g_{1,N} \\ \vdots & \ddots & \vdots \\ g_{d,1} & \cdots & g_{d,N} \end{bmatrix} \tag{2.37}$$

6) Repeat steps 3 to 6 for all possible bytes of the key.

2.3.2. Profile power attacks

Unlike non-profiled attacks, profiled power attacks require the attacker to have control of a device that is similar to the target device in order to make a leaking reference referred to as template. Then, he or she uses this template to take advantage of the target device's leak and get the device's key. Profiled power attacks are usually more effective than non-profiled attacks because the attacker has access to a reference for comparison, allowing for precise analysis and information retrieval. Template Based Attack (TBA) is the well-known examples of this category of attacks. TBA requires more setup compared to CPA [36]. Although the attacker needs control of an additional device copy and performs extensive pre-processing with thousands of power traces to create the template, he/she can complete template attacks using only a few victim traces. With sufficient pre-processing, the key could potentially be recovered from a single trace.

- **TBA**
  There are two phases in TBA: Profiling and Extraction [36].
  - **Profiling**
    1) Get access to a similar device to create profiles.
    2) Select the point of attack: $S = SBOX(P \oplus K)$, with P as the plaintext, K as the key, and SBOX as the SubByte calculation.
    3) Compute hypothetical SBOX value for all possible values of the target subkey K_j, which is shown in (2.38).

$$\begin{aligned}\mathrm{S}&=\begin{bmatrix} S_{1,1} & \cdots & S_{1,255} \\ \vdots & \ddots & \vdots \\ S_{d,1} & \cdots & S_{d,255} \end{bmatrix} S_{i,j} \\ &= SBOX(P_i \oplus K_j)\end{aligned} \tag{2.38}$$

    4) Compute the Hamming Weight h_i of the target operation using P_i and K_i.
    5) Separate the traces into sets HW_0, HW_1,..., and HW_8 (Hamming Weight creates 9 sets).

6) Calculate the average trace m_i of each group.
7) Find the Points-of-Interest (POI). For example, we can apply the sum of differences among the average traces as $D_s = \sum_{j=1}^{8} \sum_{z=0}^{j-1} |m_{j,s} - m_{z,s}|$.
8) Create the templates: Calculate the mean μ of all POI inside each HW group and the covariance C among all POIs inside the group, which can be calculated as (2.39).

$$\mu_i = \frac{1}{d^*} \sum_{j=0}^{d^*} t_{j,POI},$$

$$c_{ij} = \frac{1}{d^*} \sum_{k=1}^{d^*} \left(t_{k,i} - \mu_i\right) * \left(t_{k,j} - \mu_j\right), \tag{2.39}$$

$$\mathrm{C} = \begin{bmatrix} v_1 & c_{1,2} & c_{1,3} & \cdots & c_{1,n} \\ c_{2,1} & v_2 & c_{2,3} & \cdots & c_{2,n} \\ c_{3,1} & v_3 & c_{3,3} & \cdots & c_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{d^*,1} & c_{d^*,2} & \cdots & a_{d^*,n-1} & a_{d^*,n} \end{bmatrix},$$

where d^* is the amount of traces in the HW group, n is the amount of points in each trace, c_{ij} denotes the covariance of power trace, and v_j represents the variance of power trace.

– **Extraction**

1) Identifies the target operation in the target device.
2) Measure power traces $t_1, \ldots, t_d$ of plaintext $P_1, \ldots, P_d$.
3) Create a matrix of power traces A containing only the POIs, as shown in (2.40).

$$\mathrm{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{d,1} & \cdots & a_{d,n} \end{bmatrix}, \tag{2.40}$$

where d is the number of traces.

4) Compute leakage model (e.g. Hamming Weight) of SBOX output using hypothetical keys, as shown in (2.41).

$$\begin{aligned} \mathrm{H} &= \begin{bmatrix} HW_{1,0} & \cdots & HW_{1,255} \\ \vdots & \ddots & \vdots \\ HW_{d,0} & \cdots & HW_{d,255} \end{bmatrix} \\ &= \begin{bmatrix} HW(SBOX(P_1 \oplus K_0)) & \cdots & HW(SBOX(P_1 \oplus K_{255})) \\ \vdots & \ddots & \vdots \\ HW(SBOX(P_d \oplus K_0)) & \cdots & HW(SBOX(P_d \oplus K_{255})) \end{bmatrix}, \end{aligned} \tag{2.41}$$

5) Compute the probability density function (PDF) of the power trace A and its Hamming Weight H, which is calculated in equation (2.42). All the results are stored in matrix P of dimension d (number of traces) by 256 (possible key values), shown in (2.43).

$$PDF(a_i, HW(i,j)) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(C)}} \cdot e^{-((a_i - HW(i,j))^T \cdot C^{-1} \cdot (a_i - HW(i,j)))/2} \tag{2.42}$$

$$\mathrm{P} = \begin{bmatrix} PDF(a_1, HW_{1,0}) & \cdots & PDF(a_1, HW_{1,255}) \\ \vdots & \ddots & \vdots \\ PDF(a_d, HW_{d,0}) & \cdots & PDF(a_d, HW_{d,255}) \end{bmatrix} \tag{2.43}$$

6) Accumulate the results associated with the same guessed key (e.g. calculating the product of each column in matrix P). The column index with the highest product indicates the most probable key value.

2.4. Countermeasures against SCAs

In this section, the conventional masking countermeasures, Threshold Implementation (TI) and Domain-Oriented Masking (DOM) are introduced in detail.

2.4.1. Conventional Masking Countermeasures

To protect implementations of AES against SCAs, a widely used technique involves randomizing all intermediate results during the algorithm's computation, known as masking schemes. The fundamental concept underlying masking is to establish computational independence from the processed data. Mehdi and Christophe [37] proposed a XOR masking countermeasure to realize that independence, shown in Fig. 2.6. X denotes the applied mask,and $X_1 = f_1(X)$, where f_1 is the linear part of the affine transformation of *SubBytes*. $X_2 = ShiftRow(X_1)$, $X_3 = MixColumn(X_2)$, and K_i represents the round key i.

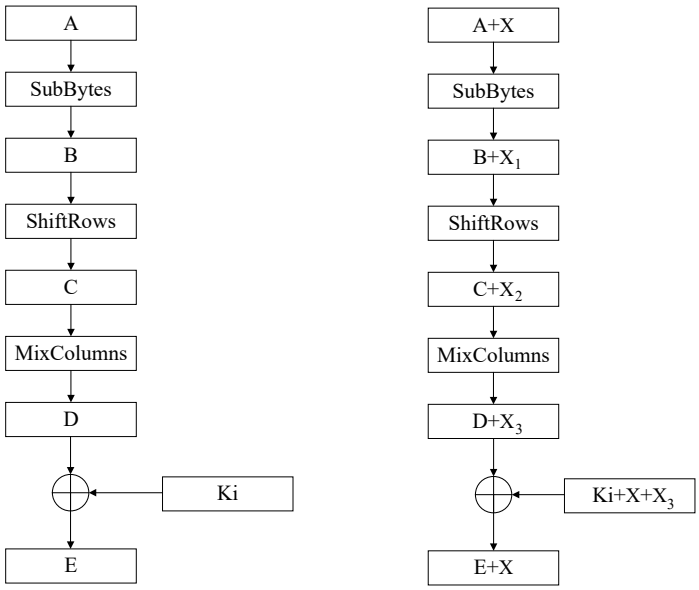


Figure 2.6: One round of the AES with (right) and without (left) masking countermeasure [37]

Several masking schemes have been proposed against SPA, CPA, DPA, etc. [38–40]. However, in 2005, Stefan and his team [41] discovered that circuits of masked gates are vulnerable to classical first-order DPA attacks due to glitches that occur within the circuits. They further contended that glitches occur in every CMOS circuit, leading them to conclude that the currently known masking schemes for CMOS gates are ineffective against DPA attacks.

2.4.2. Threshold Implementation (TI)

To overcome issues of masking countermeasures, Threshold Implementation (TI) was proposed by Nikova et al. [42] to counter SCAs and glitches. TI focus on both component functions, which is proposed based on secret sharing [43], threshold cryptography [44] and multi-party computation protocols [45]. The (k, n) security sharing is to divide the data into n pieces, enabling convenient reconstruction from any k fragments. However, possessing complete knowledge of $k-1$ fragments should impart no information about the data [43]. The approach of (n,n) sharing is employed in [42], so that all n shares are needed in order to determine the data $x = A_x \oplus B_x \oplus C_x \oplus$ ..., where $A_x, B_x, C_x,$ ... denote shares of x, and $\oplus$ denotes the XOR operation. The chosen sharing technique can also impact the

function applied to the data. The function ”F” is decomposed into multiple sub-functions, as illustrated by $F = F_A \oplus F_B \oplus F_C \oplus ....$

Functions can be divide into two parts: linear and non-linear functions. Mathematically, a linear function z=LF(x) can be expressed as (2.44), where input $x = A_x \oplus B_x \oplus C_x \oplus ...$ and $z = A_z \oplus B_z \oplus C_z \oplus ....$ In linear function, each output ($A_z, B_z, C_z, ...$) is independent of ($A_x, B_x, C_x, ...$). Consequently, conventional masking techniques can be applied in equation (2.44).

$$\begin{aligned} z &= A_z \oplus B_z \oplus C_z \oplus ... \\ &= LF(A_x) \oplus LF(B_x) \oplus LF(C_x) \oplus ... \\ &= LF(A_x \oplus B_x \oplus C_x \oplus ...) \\ &= LF(x) \end{aligned} \tag{2.44}$$

However, the implementation of non-linear functions is much more challenging. According to [42], correctness, non-completeness, and uniformity of non-linear functions are required to guarantee the data independence even in the presence of glitches.

- **Correctness**
  The correctness requires that the sum of the sub-functions yield identical results to those of the unshared variables.
- **Non-completeness**
  Each function demonstrates independence from at least one share of every input variable.
- **Uniformity**
  Each intermediate value generated during the processing of a share conforms to a uniform distribution. Additionally, it is imperative that all shared inputs and outputs of functions exhibit uniform distribution characteristics.

Since then, extensive protected hardware implementations based on TI have been proposed [46–49]. Bilgin et al. [49] stated that three shares per variable and one fresh random share are enough to realize a first-order secure TI, shown in (2.45). F represents a non-linear function, specifically characterized by the equation F = x * y. Three shares per variable in this function satisfy that $F = F_A \oplus F_B \oplus F_C$, $x = A_x \oplus B_x \oplus C_x$, and $y = A_y \oplus B_y \oplus C_y$. Z_0 is the random value.

$$\begin{aligned} \mathrm{F_A} &= B_x B_y + B_x C_y + C_x B_y + Z_0 \\ \mathrm{F_B} &= C_x C_y + A_x C_y + C_x A_y + A_x Z_0 + A_y Z_0 \\ \mathrm{F_C} &= A_x A_y + A_x B_y + B_x A_y + A_x Z_0 + A_y Z_0 + Z_0 \end{aligned} \tag{2.45}$$

Then Bilgin [50] extended 1st-order TI to higher orders. Nonetheless, the first extension of TI to safeguard circuits against higher-order attack is proven to be susceptible to glitches by Reparaz [51]. Thomas De Cnudde et al. [52] proposed a solution to address this issue. However, implementing their design requires a high number of shares, which can be costly. Besides, redesigning the non-linear components of the circuit is needed to meet the requirements for higher-order TI, leading to a significant increase in design effort.

2.4.3. Domain-Oriented Masking (DOM)

To provide a lower cost of design, the DOM technique has been proposed [24]. Compared with TI, the number of required shares in DOM is reduced, while maintaining the same level of security. DOM implementation in hardware has demonstrated its resilience to SCAs up to at least 15^{th} order [24].

In DOM implementations, each variable is represented by d + 1 shares to protect the circuit from d^{th}-order SCAs. The fundamental principle of the DOM approach is to maintain the independence of shares within each domain and those of other domains. In linear modules of AES (*MixColumns*, *AddRoundKey*, and *ShiftRows*), there is no cross-domain calculation among different shares, making it simple to maintain their independence in the DOM implementation. However, during the *SubBytes* operation, which is the only non-linear module in AES, there is a substantial occurrence of multiplication calculations. When performing these multiplications, shares may cross domain borders, making it necessary to use fresh random numbers to maintain their independence. Therefore, the development of a new multiplier is imperative to safeguard the security of the *SubBytes* module.

Hannes and his team [24] proposed two multipliers named DOM-indep and DOM-dep. DOM-indep multiplier requires that the inputs are shared independently. Take the 1^{st}-order DOM-indep multiplier as an example, which is shown in Fig. 2.7.

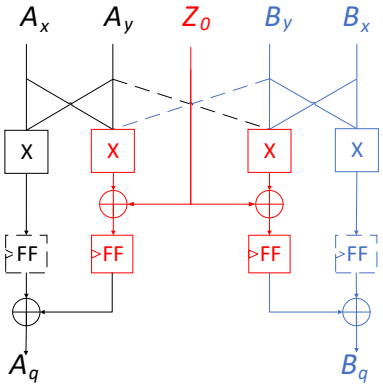


Figure 2.7: DOM_indep multiplier [24]

Assume two independent inputs of DOM-indep multiplier are x and y, where x = $A_x \oplus B_x$, y = $A_y \oplus B_y$ ("$\oplus$" represents the xor operation). Z_0 denotes the fresh random number. Then the output q (q = $A_q \oplus B_q$) can be expressed as (2.46).

$$\begin{aligned} q &= x \cdot y \\ &= (A_x \oplus B_x)\left(A_y \oplus B_y\right) \\ &= A_xA_y \oplus A_xB_y \oplus B_xA_y \oplus B_xB_y \\ &= [A_xA_y \oplus (A_xB_y \oplus Z_0)] \oplus [(B_xA_y \oplus Z_0) \oplus B_xB_y] \\ &= A_q \oplus B_q \end{aligned} \tag{2.46}$$

Hannes [24] categorized the product terms into two parts: inner-domain terms (A_xB_x and A_yB_y) and cross-domain (A_xB_y and A_yB_x). The inner-domain product terms do not reveal critical information since they involve inputs from a single domain. In contrast, cross-domain calculations can only be performed on independent inputs to prevent leakage of information for either x or y. A fresh random Z_0 value is used to remain the statistical independence of the DOM-indep multiplier outputs from other values. Additionally, flip-flops are employed to prevent glitches from propagating through this block. Furthermore, Hannes et al. [24] proposed the DOM-dep multiplier based on the structure of the DOM-indep multiplier, which does not require the independence of inputs (shown in Fig. 2.8). However, it requires more fresh randomness values and area space.

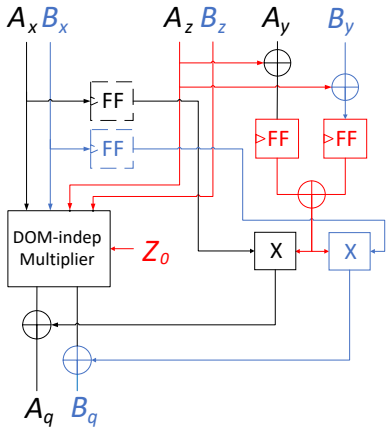


Figure 2.8: DOM_dep multiplier [24]

Two inputs of DOM-dep multiplier are x and y, where x = $A_x \oplus B_x$, y = $A_y \oplus B_y$. Z_0 and z are random values, where z = $A_z \oplus B_z$. Then the output q (q = $A_q \oplus B_q$) can be expressed as (2.47). The inclusion

of a random variable, z, prevents the direct multiplication of x and y, thereby relaxing the necessity for these inputs to maintain independence.

$$\begin{aligned} q &= x \cdot y \\ &= x \cdot (y \oplus z) \oplus x \cdot z \\ &= (A_x \oplus B_x) \cdot (A_y \oplus B_y \oplus A_z \oplus B_z) \oplus (A_x \oplus B_x) \cdot (A_z \oplus B_z) \\ &= A_x[A_y \oplus A_z \oplus B_y \oplus B_z] \oplus A_x A_z \oplus (A_x B_z \oplus Z_0) \oplus \\ &\quad B_x[A_y \oplus A_z \oplus B_y \oplus B_z] \oplus (B_x A_z \oplus Z_0) \oplus B_x B_z \\ &= A_q \oplus B_q \end{aligned} \tag{2.47}$$

The whole structure of the 1^{st}-order DOM design of the SBOX with five or eight pipeline stages is shown in Fig. 2.9 [24]. The SBOX in Fig. 2.9 is divided into 5 stages by the red dotted line, while the grey dotted line offers an optional division for 8 stages. 8-bit A_x, B_x and A_y, B_y are the shared inputs and outputs of DOM SBOX module, respectively. The inputs of $GF(2^4)$ multiplier in Stage 1 (see Fig. 2.9) are derived from the outputs of the linear mapping at the SBOX input. However, due to variations in signal transition times and gate delays, the output of the linear mapping may contain bits with related sharing. Consequently, we have two viable options: 1. Employing DOM-dep multiplier for 5 stages SBOX; 2. Inserting registers after the linear mapping and utilizing a DOM-indep multiplier for 8 stages SBOX.

The situation is similar at Stage 2 and Stage 3. Glitches can occur when combining the outputs of Scalar Square and multipliers. To mitigate these glitches, it is essential to utilize DOM-dep multipliers or insert pipelining registers. However, the inputs of the multipliers in Stage 4 consist of SBOX inputs (higher 4 bits of both A_x and B_x) and the outputs of $GF(2^2)$ multipliers in Stage 3. As a result, the inputs of the multipliers in Stage 4 are independent of each other, allowing us to utilize DOM-indep multipliers without inserting registers. In Stage 5, the operations are solely comprised of linear transformations.

The 1^{st}-order DOM-dep and DOM-indep multipliers can be easily extended to higher order by using more shares and fresh random values [24].

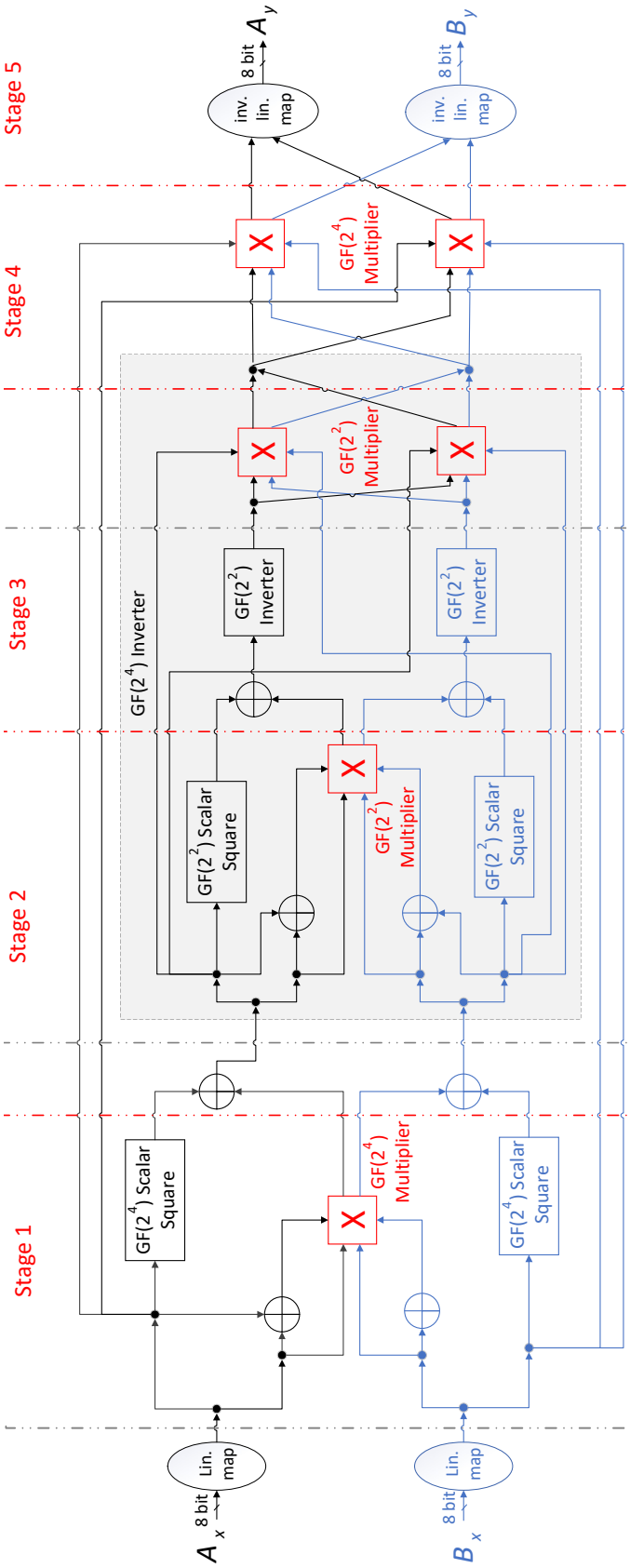


Figure 2.9: Structure of the 1^{st}-order five/eight-stage DOM SBOX Module [24]

3

Lightweight AES and its DOM extension

This chapter presents our proposed implementation approach for AES and its protected version using DOM. Section 3.1 addresses the motivation behind our approach. Section 3.2 presents the structure of the proposed AES design and provides a detailed introduction to the shared modules within AES. Section 3.3 outlines the structure of the simplified DOM-version AES design and its implementation.

3.1. Motivation

Previous research primarily focused on implementing AES on either an 8-bit [12–18] or 32-bit [19, 20] data-path to reduce area and energy consumption. However, these studies only reported the area of the encryption module and neglect the decryption part. In reality, the eleven 128-bit registers required for the key expansion in the decryption module contribute significantly to the overall core area. In the decryption module, all key rounds must be computed first before the decryption can start. Shortening the data-path from 128-bit to a lower-bit width has a much lower improvement on the area when the decryption module is not ignored. Therefore, in our design, we focus on different data-paths in the presence of the decryption unit and compare their performance in terms of throughput, area, and power. Secondly, in actual applications, keys do not change frequently. Hence, we perform the key expansion once and store the results in the registers. As long as the key remains the same, we can skip the key expansion step, resulting in a significant power and latency reduction. In addition, we reorder the sequences of *Addroundkey* and *MixColumns* in the round function which results in further area and performance improvements.

3.2. Design and Implementation of the Proposed Lightweight AES

In this section, we present the high-level structure of the AES design, followed by a comprehensive introduction to each module.

3.2.1. High-level Structure of AES design

Our proposed AES designs verify whether the key changes at the start of every encryption/decryption execution. In case a key change is detected, we perform the *key expansion* module and leave the keys inside the key registers. Otherwise, we directly execute AES encryption or decryption. This reduces the execution time of the decryption part by eleven cycles. To further optimize the design area, resource sharing is employed. Initially, we limit the number of registers to store the state to a single 128-bit register that can be reused among all the AES modules. Next, we combine the encryption and decryption modules to decrease the overall area. The pseudocode for the proposed AES Encryption and Decryption is presented in Algorithm 2. The inputs encompass the 128-bit Key, 128-bit Datain, and 1-bit Enc/Dec signal, while the outputs are 128-bit Dataout and 1-bit Dataout_valid signal. Note that Dataout is reused to save area. When the dataout_valid signal is set to 1, the Dataout aligns with the intended result. In this pseudocode, 'cnt' signifies the specific round of AES, and 'Key_reg' designates the register responsible for retaining the eleven 128-bit keys. When key changes, the *key expansion* module is performed first. Conversely, AES encryption or decryption is executed. During

round 0, the operation carried out is *AddRoundKey*. Subsequently, for rounds 1 to 9, the AES Round Function, encompassing *AddRoundKey*, *Shared SBOX*, *Shared MixColumns*, and *Shared ShiftRows*, is performed. In the final round, *Shared MixColumns* is skipped, and the remaining three modules are executed sequentially. Elaborate descriptions of the AES round function and shared modules are provided subsequently.

Algorithm 2 Pseudocode of proposed AES Encryption and Decryption

Input: Key, Datain, Enc/Dec signal;
Output: Dataout, Dataout_valid;

```
1: begin
2: cnt ←0, Key_reg ←0;
3: if (key changes) then
4:    key_reg ←Key Expansion(Key, Datain, Enc/Dec signal);
5: else
6:    if (cnt == 0) then
7:       Dataout ←Addroundkey(Key_reg, Dataout, Enc/Dec signal);
8:       cnt ←cnt + 1;
9:    else if (cnt > 0) && (cnt < 10) then
10:      Dataout←AES Round Function(Key_reg, Dataout, Enc/Dec signal);
11:      cnt ←cnt + 1;
12:   else
13:      Dataout←Shared SBOX(Dataout, Enc/Dec signal);
14:      Dataout←Shared ShiftRows(Dataout, Enc/Dec signal);
15:      Dataout←AddRoundKey(Key_reg, Dataout, Enc/Dec signal);
16:      cnt ←0;
17:      Dataout_valid ←1.
18:   end if
19: end if
20: end
```

The proposed AES round function is depicted in Fig. 3.1, where *cnt* represents the round index and *key[cnt]* denotes the 128-bit key that needs to be XORed with the State array in the current round. ”Enc/Dec” is the signal that indicates whether the operation involves encryption or decryption. The boxes with shared denote shared functionality between the encryption and decryption operations. Note that no additional registers are required for storing and comparing the key because the original design and our design both employ eleven 128-bit registers for storing the key. Meanwhile, *key expansion* module and *encryption/decryption* module share the SBOX module.

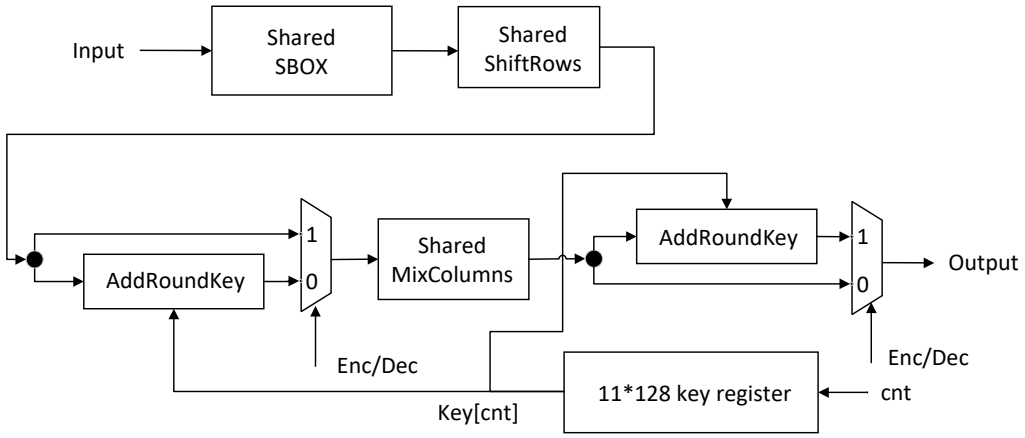
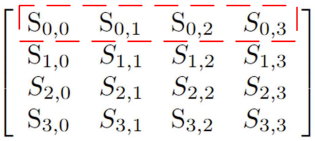
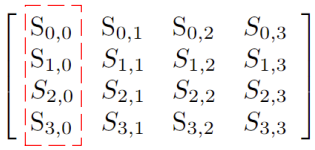


Figure 3.1: Proposed Round Function AES Encryption and Decryption

In addition, we optimized the *MixColumns* by carefully selecting the input of the round function to enable sharing and reduce complexity. When implementing a data-path design lower than 128-bit, it is



(a) row selection



(b) column selection

Figure 3.2: Input Selection Example in 32-bit Design

necessary to select the column numbers as the input of the round function in order to save one cycle in each round and reduce the number of required *MixColumns* modules. Take the 32-bit data-path design (see Fig. 3.2) as an example to illustrate the impact of the input selection on the round function. The figure shows that 32-bits can be selected in a row and column fashion. We notice that the *MixColumns* module performs a modular polynomial multiplication on each column of the State array. If we use a row as the input for round function, only one row value is available after *SubBytes* and *ShiftRows*, making it impossible to perform *MixColumns* as it requires a complete column. Therefore, it is necessary to wait for 4 cycles to process all rows, which leads to an additional cycle and an increase in the number of required *MixColumns* modules from 1 to 4. Instead, if we select the columns of the State array as input to the round function, we can directly perform *MixColumns* using a single *MixColumns* module and save one cycle in each round. An in-depth explanation of the shared modules will be provided next, including *Shared SBOX, Shared ShiftRows*, and *Shared MixColumns*.

3.2.2. Shared SBOX

Akashi et al. [53] proposed a new composite gelois field to optimize the structure of the SBOX, resulting in a significant reduction of the area compared to using a Look-up table (LUT). Thereafter, several researchers [26, 54, 55] optimized the SBOX based on this structure. These papers used an SBOX shared by both the encryption and decryption modules to reduce area. To the best of our knowledge, the SBOX design described in [55] has the lowest area. Compared with previous designs, they shared resources in three modules: preprocess, postprocess, and scalar square. The preprocess module performs isomorphic mapping and inverse affine transformation for the decryption and isomorphic mapping only for the encryption. The postprocess module executes affine transformation and inverse isomorphic mapping for the encryption and inverse isomorphic mapping only for the decryption. The scalar square performs square and multiplication with constant $\lambda = \{1, 1, 0, 0\}$, which leads to three XOR reductions. Fig. 3.3 illustrates the proposed Shared SBOX, which utilizes optimized multiplier, modifies the inverter, and shares the resources of last two multipliers. Each optimization is described next into more details.

- $GF(2^4)$ **Optimized multiplier:** Our optimized $GF(2^4)$ multiplier is based on the work in [55]. That multiplier consists of 18 XOR and 12 AND gates and its critical path consists of 4 XOR and 1 AND gate. We simplified the $GF(2^4)$ multiplier based on Equation (3.2), where $\{a_3, a_2, a_1, a_0\}$ and $\{b_3, b_2, b_1, b_0\}$ denote the two 4-bit inputs (see also left bottom of Fig. 3.3), $\{c_3, c_2, c_1, c_0\}$ denote the 4-bit output, and $\{m_4, m_3, m_2, m_1, m_0\}$ are intermediate variables that can be defined as equation (3.1).

$$\begin{aligned} m_4 &= m_0 \oplus m_1; \\ m_3 &= a_0 \oplus a_1; \\ m_2 &= a_3 \oplus a_2; \\ m_1 &= a_2 \oplus a_0; \\ m_0 &= a_3 \oplus a_1. \end{aligned} \tag{3.1}$$

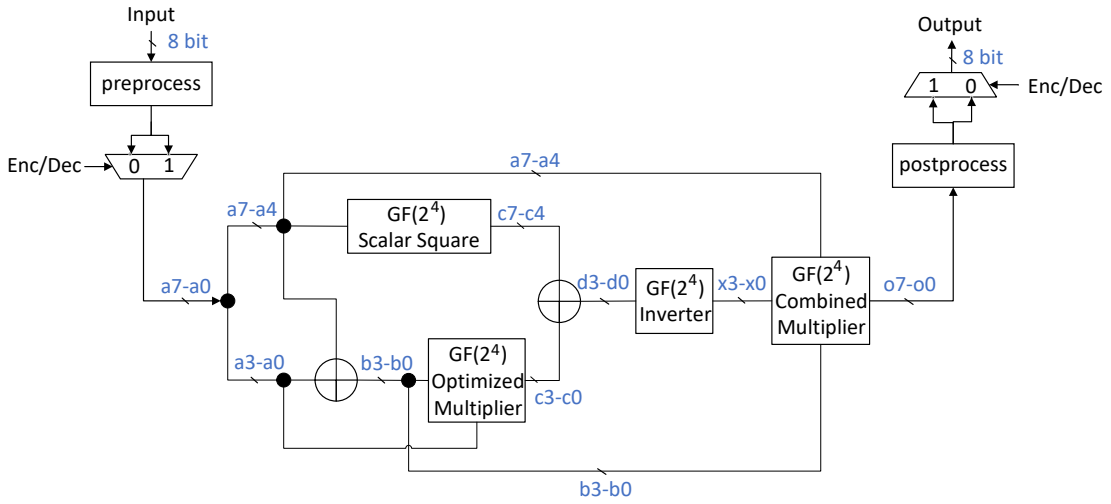


Figure 3.3: Proposed Shared SBOX

Although our $GF(2^4)$ optimized multiplier utilizes 4 more AND gates compared to [55], it requires 1 XOR gate less and more importantly has a shorter critical path (1 AND gate and 3 XOR gates). Surprisingly, after synthesis it turns out that the area of this implementation is also better after synthesis. We believe that compiler is able to extract more common resources with this implementation.

$$
\begin{aligned}
c_3 &= [(a_3 \oplus a_1)\&(b_3 \oplus b_1) \oplus (a_3 \oplus a_1)\&(b_2 \oplus b_0) \oplus (a_2 \oplus \\
&\quad a_0)\&(b_3 \oplus b_1)] \oplus [(a_1\&b_1) \oplus (a_1\&b_0) \oplus (a_0\&b_1)] \\
&= (b_0\&a_3) \oplus (b_1\&(a_2 \oplus a_3)) \oplus (b_2\&(a_1 \oplus a_3)) \oplus (b_3\&(a_0 \oplus a_1 \oplus a_2 \oplus a_3)) \\
&= (b_0\&a_3) \oplus (b_1\&m_2) \oplus (b_2\&m_0) \oplus (b_3\&m_4);
\end{aligned}
$$

$$
\begin{aligned}
c_2 &= [(a_3 \oplus a_1)\&(b_3 \oplus b_1) \oplus (a_2 \oplus a_0)\&(b_2 \oplus b_0)] \oplus (a_1\&b_1) \oplus (a_0\&b_0) \\
&= (b_0\&a_2) \oplus (b_1\&a_3) \oplus (b_2\&(a_0 \oplus a_2)) \oplus (b_3\&(a_1 \oplus a_3)) \\
&= (b_0\&a_2) \oplus (b_1\&a_3) \oplus (b_2\&m_1) \oplus (b_3\&m_0);
\end{aligned}
\tag{3.2}
$$

$$
\begin{aligned}
c_1 &= [(a_3\&b_3) \oplus (a_3\&b_2) \oplus (a_2\&b_3)] \oplus [(a_3\&b_3) \oplus (a_2\&b_2)] \\
&\quad \oplus [(a_1\&b_1) \oplus (a_1\&b_0) \oplus (a_0\&b_1)] \\
&= (b_0\&a_1) \oplus (b_1\&(a_0 \oplus a_1)) \oplus (b_2\&(a_2 \oplus a_3)) \oplus (b_3\&a_2) \\
&= (b_0\&a_1) \oplus (b_1\&m_3) \oplus (b_2\&m_2) \oplus (b_3\&a_2);
\end{aligned}
$$

$$
\begin{aligned}
c_0 &= [(a_3\&b_3) \oplus (a_3\&b_2) \oplus (a_2\&b_3)] \oplus [(a_1\&b_1) \oplus (a_0\&b_0)] \\
&= (b_0\&a_0) \oplus (b_1\&a_1) \oplus (b_2\&a_3) \oplus (b_3\&(a_2 \oplus a_3)) \\
&= (b_0\&a_0) \oplus (b_1\&a_1) \oplus (b_2\&a_3) \oplus (b_3\&m_2).
\end{aligned}
$$

- $GF(2^4)$ **Inverter:** To make the inverter more area-efficient and suitable for security extension (less non-linear calculations), we optimized the inverter based on the structure that proposed in [26], which is shown on Fig. 3.4. The $GF(2^2)$ *Scalar Square* combines $GF(2^2)$ square and multiplication with a constant $\varphi = \{1, 0\}$ (scalar), leading to 2 XOR gate reduction. Assume that d_3, d_2 is the input of $GF(2^2)$ square module, e_3, e_2 denotes the ouput of $GF(2^2)$ square module and input of $GF(2^2)$ multiplication with a constant module, and f_3, f_2 represents the output of $GF(2^2)$ multiplication with a constant module. In a similar manner as described in Section 2.2, $D = d_3x \oplus d_2$, $E = e_3x \oplus e_2$, and $F = f_3x \oplus f_2$. Equation (3.3) and (3.4) illustrate that the

calculation of $GF(2^2)$ square and scalar.

$$\begin{aligned} E &= (d_3x \oplus d_2)^2 \\ &= {d_3}^2x^2 \oplus {d_2}^2 \\ &= d_3(x \oplus 1) \oplus d_2 \\ &= d_3x \oplus (d_3 \oplus d_2) \\ &= e_3x \oplus e_2 \end{aligned} \tag{3.3}$$

$$\begin{aligned} F &= (e_3x \oplus e_2)(1 \cdot x \oplus 0) \\ &= e_3x^2 \oplus e_2x \\ &= e_3(x \oplus 1) \oplus e_2x \\ &= (e_3 \oplus e_2)x \oplus e_3 \\ &= d_2x \oplus d_3 \\ &= f_3x \oplus f_2 \end{aligned} \tag{3.4}$$

Therefore, we can achieve that $f_3 = d_2$ and $f_2 = d_3$. Equation (3.5) illustrates the $GF(2^2)$ *Scalar Square* calculation of [26] (left) and our combined design (right).

$$[24]\begin{cases} e_3 = d_3 \\ e_2 = d_3 \oplus d_2 \\ f_3 = e_3 \oplus e_2 \\ f_2 = e_3 \end{cases} \qquad Combined\begin{cases} f_3 = d_2 \\ f_2 = d_3 \end{cases} \tag{3.5}$$

The last step of the inverter consist of two $GF(2^2)$ multipliers. Equation (3.6) shows the design proposed in [26]. Our optimizations are provided after the second "=" sign by factoring out $X = h_1 \oplus h_0$ commonly between both multiplications. Our combined $GF(2^2)$ multiplier achieves a reduction of 1 XOR gate and 2 AND gates, compared with the design in [26].

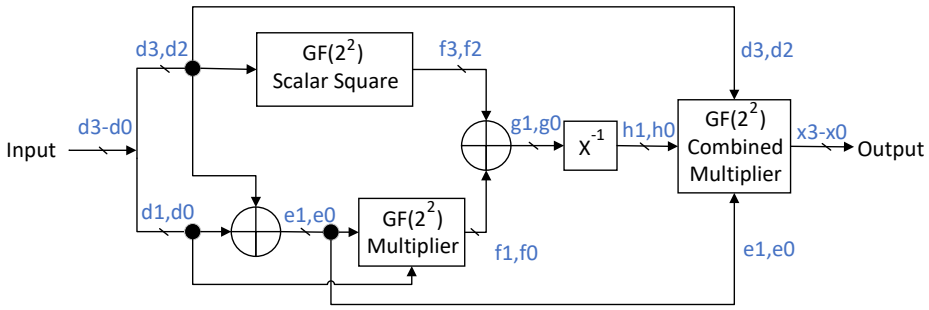


Figure 3.4: $GF(2^4)$ Inverter

$$\begin{aligned} &x_3 = (d_3 \& h_1) \oplus (d_3 \& h_0) \oplus (d_2 \& h_1) = (d_3 \& X) \oplus (d_2 \& h_1); \\ &x_2 = (d_3 \& h_1) \oplus (d_2 \& h_0); \\ &x_1 = (e_1 \& h_1) \oplus (e_1 \& h_0) \oplus (f_0 \& h_1) = (e_1 \& X) \oplus (f_0 \& h_1); \\ &x_0 = (e_1 \& h_1) \oplus (e_0 \& h_0). \end{aligned} \tag{3.6}$$

- $GF(2^4)$ **Combined Multiplier:** The last two multipliers utilized in [55] were treated as two seperate multipliers. However, Ahmad [56] proposed that these two multipliers can be merged together, resulting a significant reduction in area. However, it is not clear from the paper how this shared multiplier works. For clarity, we provided a detailed structure of the shared multiplier. Fig. 3.5 shows the structure of the $GF(2^4)$ Shared Multiplier. The intermediate value A, B, C, D, E can be

calculated as equation (3.7).

$$\begin{aligned} A &= x3 \oplus x1; \\ B &= x2 \oplus x0; \\ C &= x3 \oplus x2; \\ D &= A \oplus B; \\ E &= x1 \oplus x0. \end{aligned} \tag{3.7}$$

In AND_XOR block (see Fig. 3.5), the output $O_0 = (i_0 \& j_0) \oplus (i_1 \& j_1) \oplus (i_2 \& j_2) \oplus (i_3 \& j_3)$, where $\{i_0, i_1, i_2, i_3\}$ and $\{j_0, j_1, j_2, j_3\}$ are two 4-bit inputs.

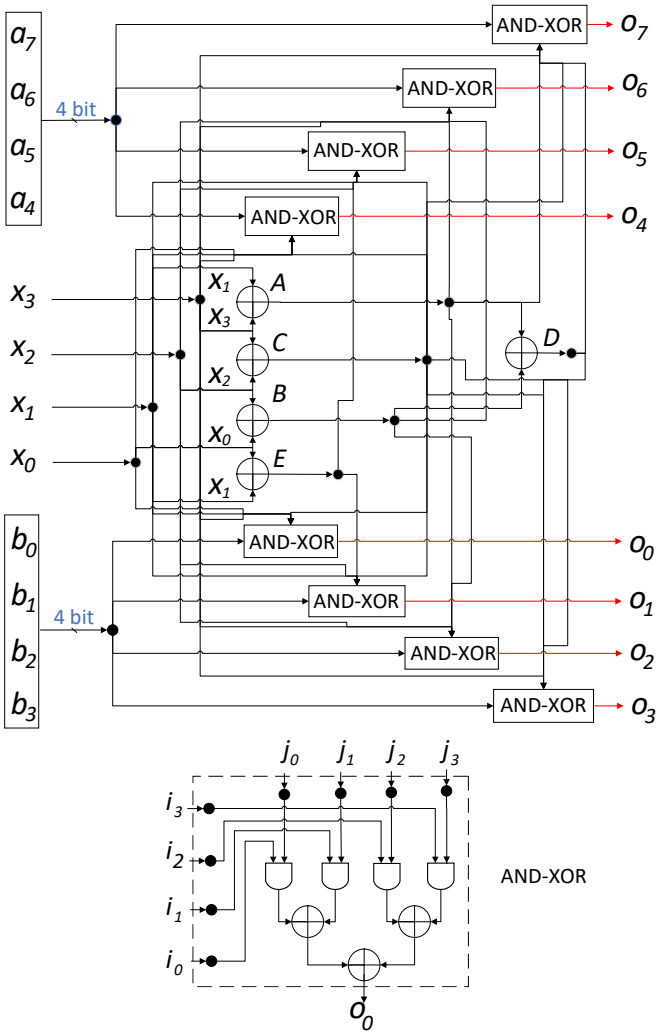


Figure 3.5: $GF(2^4)$ Combined Multiplier

3.2.3. Shared ShiftRows

Davis and John proposed that the first and third shift operations in *ShiftRows* and *InvShiftRows* can be shared [20], as both produce the same results for the decryption and encryption. This can be seen in Fig. 3.6. However, the other two rows (i.e., row two and four) have different behavior and multiplexers are needed to select between them. Assuming the inputs and outputs of four rows are $\{sr0\_in, sr1\_in, sr2\_in, sr3\_in\}$ and $\{sr0\_out, sr1\_out, sr2\_out, sr3\_out\}$, then we can calculate the *Shared ShiftRows* by equation (3.8) and (3.9), where "Enc/Dec" signal indicates whether encryption ("Enc/Dec"=1) or decryption ("Enc/Dec"=0) is performed. Note that the $S_{i,j}$ represents the value stored in the state array, which holds the outputs of the preceding shared SBOX module.

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{bmatrix}$$

(a) ShiftRows

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,3} & S_{1,0} & S_{1,1} & S_{1,2} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,0} \end{bmatrix}$$

(b) InvShiftRows

Figure 3.6: Shift Transformation of ShiftRows and InvShiftRows

$$\begin{bmatrix} sr0\_in \\ sr1\_in \\ sr2\_in \\ sr3\_in \end{bmatrix} = \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \tag{3.8}$$

$$\begin{aligned} sr0\_out &= [S_{0,0}, S_{0,1}, S_{0,2}, S_{0,3}]; \\ sr1\_out &= (Enc/Dec == 1)\ ?\ [S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3}] : [S_{1,3}, S_{1,0}, S_{1,1}, S_{1,2}]; \\ sr2\_out &= [S_{2,0}, S_{2,1}, S_{2,2}, S_{2,3}]; \\ sr3\_out &= (Enc/Dec == 1)\ ?\ [S_{3,0}, S_{3,1}, S_{3,2}, S_{3,3}] : [S_{3,1}, S_{3,2}, S_{3,3}, S_{3,0}]. \end{aligned} \tag{3.9}$$

3.2.4. Shared MixColumns:

We further optimize the Shared *MixColumns* based on the proposed design in paper [26], which shares resources between *MixColumns* and *InvMixColumns* (see Fig. 3.7). Zhang [26] mentioned that the Part I of *InvMixColumns* implement the *MixColumns* transformation and an additional *InvMixColumns* calculation is required to rectify the roundkey and achieve the mixroundkey (see Fig. 3.7). Fig. 3.8 shows that our proposed Shared *MixColumns* module combines *MixColumns* and *InvMixColumns*, and reorganize the sequence of *Addroundkey* and *Shared MixColumns* to avoid performing another *InvMixColumns* calculation in Fig. 3.7, leading to area reduction.

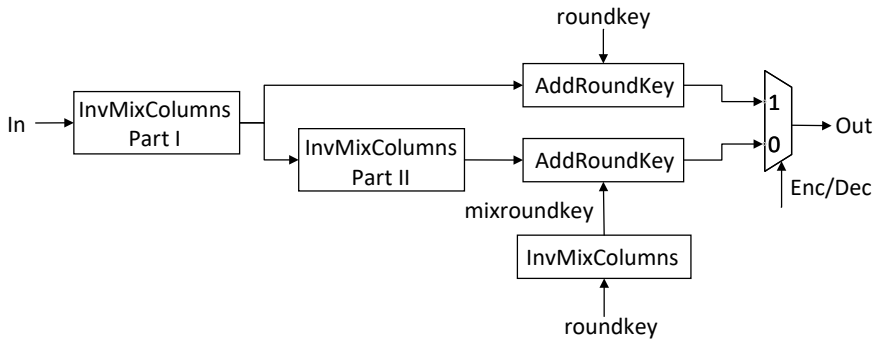


Figure 3.7: Diagram of MixColumns and InvMixColumns [26]

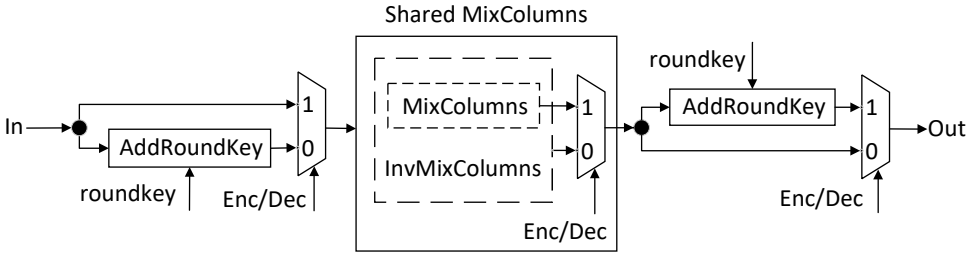


Figure 3.8: Diagram of Proposed Shared MixColumns

The detailed structure of $GF(2^4)$ Shared MixColumns is shown in Fig. 3.9, where $\{In0, In1, In2, In3\}$ and $\{Out0, Out1, Out2, Out3\}$ are four inputs and outputs of Shared MixColumns, respectively. "The block enclosed by red dashed lines represents the MixColumns operation, while the block enclosed by blue dashed lines represents the Inverse MixColumns operation. "Enc/Dec" signal indicates whether encryption ("Enc/Dec"=1) or decryption ("Enc/Dec"=0) is performed. The intermediate value "Sum" is

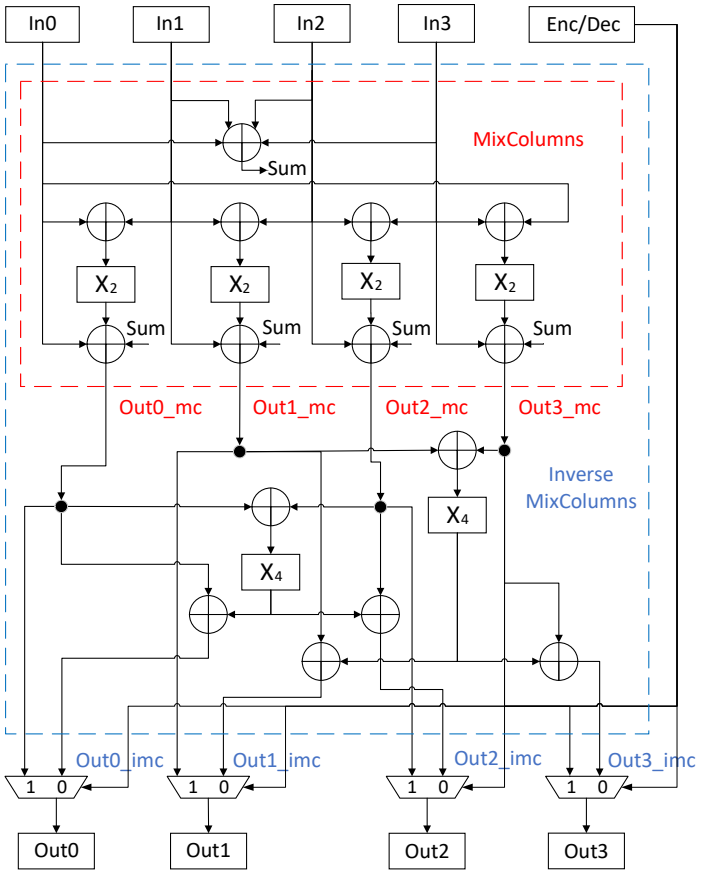


Figure 3.9: $GF(2^4)$ Shared MixColumns

calculated as $Sum = In0 \oplus In1 \oplus In2 \oplus In3$. "$X_2$" function is to compute constant multiplication by $\{02\}$ within the field $GF(2^8)$. The detailed calculation of this function is illustrated in equations (3.10), where $\{i_2[0], i_2[1], i_2[2], i_2[3], i_2[4], i_2[5], i_2[6], i_2[7]\}$ represents the 8-bit input to the "X_2" function, while $\{x_2[0], x_2[1], x_2[2], x_2[3], x_2[4], x_2[5], x_2[6], x_2[7]\}$ denotes the 8-bit output produced by the "X_2" func-

tion.

$$\begin{aligned} x_2[0] &= i_2[7]; \\ x_2[1] &= i_2[0] \oplus i_2[7]; \\ x_2[2] &= i_2[1]; \\ x_2[3] &= i_2[2] \oplus i_2[7]; \\ x_2[4] &= i_2[3] \oplus i_2[7]; \\ x_2[5] &= i_2[4]; \\ x_2[6] &= i_2[5]; \\ x_2[7] &= i_2[6]. \end{aligned} \tag{3.10}$$

"X_4" function computes constant multiplication by $\{04\}$ within the field $GF(2^8)$. The detailed calculation of this function is illustrated in equations (3.11), where $\{i_4[0], i_4[1], i_4[2], i_4[3], i_4[4], i_4[5], i_4[6], i_4[7]\}$ represents the 8-bit input to the "X_4" funrction, while $\{x_4[0], x_4[1], x_4[2], x_4[3], x_4[4], x_4[5], x_4[6], x_4[7]\}$ denotes the 8-bit output produced by the "X_4" function.

$$\begin{aligned} x_4[0] &= i_4[6]; \\ x_4[1] &= i_4[6] \oplus i_4[7]; \\ x_4[2] &= i_4[0] \oplus i_4[7]; \\ x_4[3] &= i_4[1] \oplus i_4[6]; \\ x_4[4] &= i_4[2] \oplus i_4[6] \oplus i_4[7]; \\ x_4[5] &= i_4[3] \oplus i_4[7]; \\ x_4[6] &= i_4[4]; \\ x_4[7] &= i_4[5]. \end{aligned} \tag{3.11}$$

Assuming that $\{Out0\_mc, Out1\_mc, Out2\_mc, Out3\_mc\}$ represent the four outputs of the Mix-Columns block as depicted in Fig. 3.9, these values can be determined using equation (3.12).

$$\begin{aligned} Out0\_mc &= (X_2(In0 \oplus In1) \oplus In0 \oplus Sum); \\ Out1\_mc &= (X_2(In1 \oplus In2) \oplus In1 \oplus Sum); \\ Out2\_mc &= (X_2(In2 \oplus In3) \oplus In2 \oplus Sum); \\ Out3\_mc &= (X_2(In3 \oplus In0) \oplus In3 \oplus Sum). \end{aligned} \tag{3.12}$$

Likewise, suppose that $\{Out0\_imc, Out1\_imc, Out2\_imc, Out3\_imc\}$ are four outputs of Inverse Mix-Columns block (see Fig. 3.9), these values can be determined using equation (3.13).

$$\begin{aligned} Out0\_imc &= X_4[Out0\_mc \oplus Out2\_mc)] \oplus Out0\_mc; \\ Out1\_imc &= X_4[Out1\_mc \oplus Out3\_mc)] \oplus Out1\_mc; \\ Out2\_imc &= X_4[Out0\_mc \oplus Out2\_mc)] \oplus Out2\_mc; \\ Out3\_imc &= X_4[Out1\_mc \oplus Out3\_mc)] \oplus Out3\_mc. \end{aligned} \tag{3.13}$$

Then, the outputs of Shared MixColumns can be expressed as equation (3.14).

$$\begin{aligned} Out0 &= (Enc/Dec == 1)\ ?\ Out0\_mc : Out0\_imc; \\ Out1 &= (Enc/Dec == 1)\ ?\ Out1\_mc : Out1\_imc; \\ Out2 &= (Enc/Dec == 1)\ ?\ Out2\_mc : Out2\_imc; \\ Out3 &= (Enc/Dec == 1)\ ?\ Out3\_mc : Out3\_imc. \end{aligned} \tag{3.14}$$

3.3. Design and Implementation of Proposed Lightweight DOM

DOM was proposed in [24] to protect AES implementations against SCAs. The authors introduced two types of SBOXes: a five-stage SBOX and an eight-stage SBOX. The five-stage SBOX represents an optimized version of the eight-stage SBOX, resulting in a savings of three cycles per round. Hence, overall it is 33 cycles (3 cycles $\times$ 11 rounds) faster, which is a significant performance improvement. This improvement comes with only a minor increase in the overall area, from 2.6k Gates to 2.8k Gates, as documented in [24]. Therefore, we chose to start from the five-stage SBOX and integrate it into our optimized design. Note that a 1^{st}-order DOM can be easily scaled into a higher order DOM without redesigning components [24]. Therefore, without loss of generality, we focus on the optimization of the 1^{st}-order DOM.

3.3.1. High-level structure of DOM-version AES design

As our proposed low-area design considers both encryption and decryption with shared resources (see Fig. 3.1), the lightweight DOM AES was implemented in the same manner, diverging from the original design that concentrated only on encryption [24]. Fig. 3.10 illustrates the flow diagram for DOM-version AES. The input for the DOM-version AES is designated as 'x'. Two shares of 'x' satisfy the equation $A_x \oplus B_x = x$. Correspondingly, the output of the DOM-version AES is denoted as 'y', and similarly, two shares of 'y' satisfy the equation $A_y \oplus B_y = y$. In Round 0, exclusively the *AddRoundKey* module is executed. From Rounds 1 to N-1, the DOM Round Function is implemented, as detailed in the forthcoming explanation. In the final round, Round N, a sequence of operations including the *DOM SBOX* (which will be elaborated upon), *Shared ShiftRows*, and *AddRoundKey* modules are carried out consecutively.

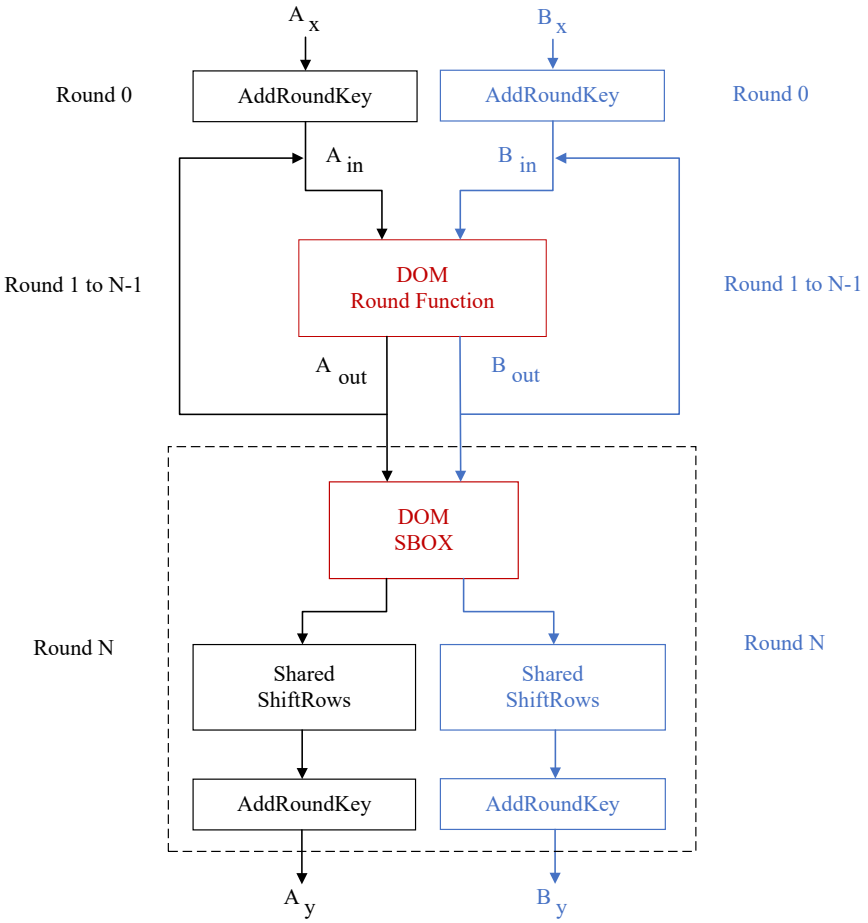


Figure 3.10: Proposed Design Flow Diagram for DOM-version AES

Fig. 3.11 shows the main part of our lightweight DOM design, where A_{in}, B_{in} represent the input

shares, and A_{out}, B_{out} correspond to the output shares. Except *DOM SBOX* module, other modules in Fig. 3.11 is similar to the modules in Fig. 3.1. To maintain data independence, we employ two independent *Shared ShiftRows* and *Shared MixColumns* operations for processing two shares. The DOM SBOX is much more complex than other modules, which will be introduces as follows.

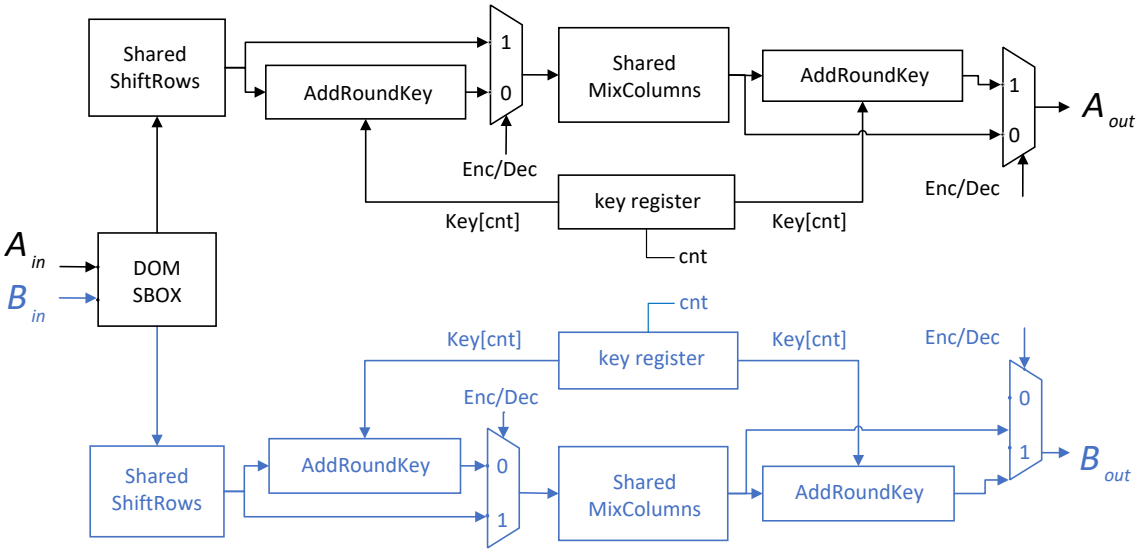


Figure 3.11: Round Function of DOM design

3.3.2. DOM SBOX

As discussed in the background, the independence of d+1 shares within linear modules can be ensured by employing d+1 identical modules. However, the non-linear SBOX module requires to be carefully designed. Our design is based on the lightweight DOM SBOX proposed in [24]. In comparison to that design, our approach shares the resources between encryption and decryption parts using the pre-process and postprocess functions. In addition, we optimize the DOM-indep and DOM-dep multipliers based on our simplified and shared multipliers to further reduce the area. Fig. 3.12 depicts our design of the 1^{st}-order five stages lightweight DOM SBOX, where A_{sin} and B_{sin} denote the input shares, and A_{sout} and B_{sout} denote the output shares. In the figure, $Z_0, Z_1, ..., Z_6$ and $A_{z0}, B_{z0}, ..., A_{z3}, B_{z3}$ are fresh random values of the simplified DOM-indep and DOM-dep multipliers, respectively. The flip-flops with dotted boxes are optional registers that are only necessary in pipelining scenarios. For example, when the data-path is less than 128 bits, the SBOX needs to be reused multiple times within one round, causing the input to change before the round is completed. In this case, the dotted flip-flops are necessary to ensure the design's functional correctness.

Fig. 3.12 also highlights our updated parts in red, i.e., the DOM multipliers. Compared to the original DOM multipliers (see [24]), we replace the normal multipliers with our simplified multipliers (see (3.2)) and shared multipliers (see Fig. 3.5), resulting in a reduction in power and area. In addition, multiplexers are used to select between inputs for the encryption and decryption units. Then we will elaborate on our simplified DOM-indep/dep multipliers in detail.

3.3.3. Simplified DOM-indep Multiplier

Figure 3.13 illustrates the configuration of the simplified DOM-indep multiplier. In contrast to the DOM-indep multiplier depicted in Figure 2.7, our simplified DOM-indep multiplier employs simplified multipliers (refer to Equation (3.2)) instead of conventional multipliers. The inputs of first simplified DOM-indep are A_{ai}, B_{ai}, A_{bi}, and B_{bi}, while its outputs are A_{qi} and B_{qi}. Z_0 denotes the random number. $\{M_1, M_2, M_3, M_4\}$ are the results of multiplication, where $M_3 = A_{ai} * A_{bi}$, $M_2 = A_{ai} * B_{bi}$, $M_1 = A_{bi} * B_{ai}$, and $M_0 = B_{bi} * B_{ai}$. When performing multiplication operations, the optimized multiplier (see (3.2)) is utilized to simplify the calculation.

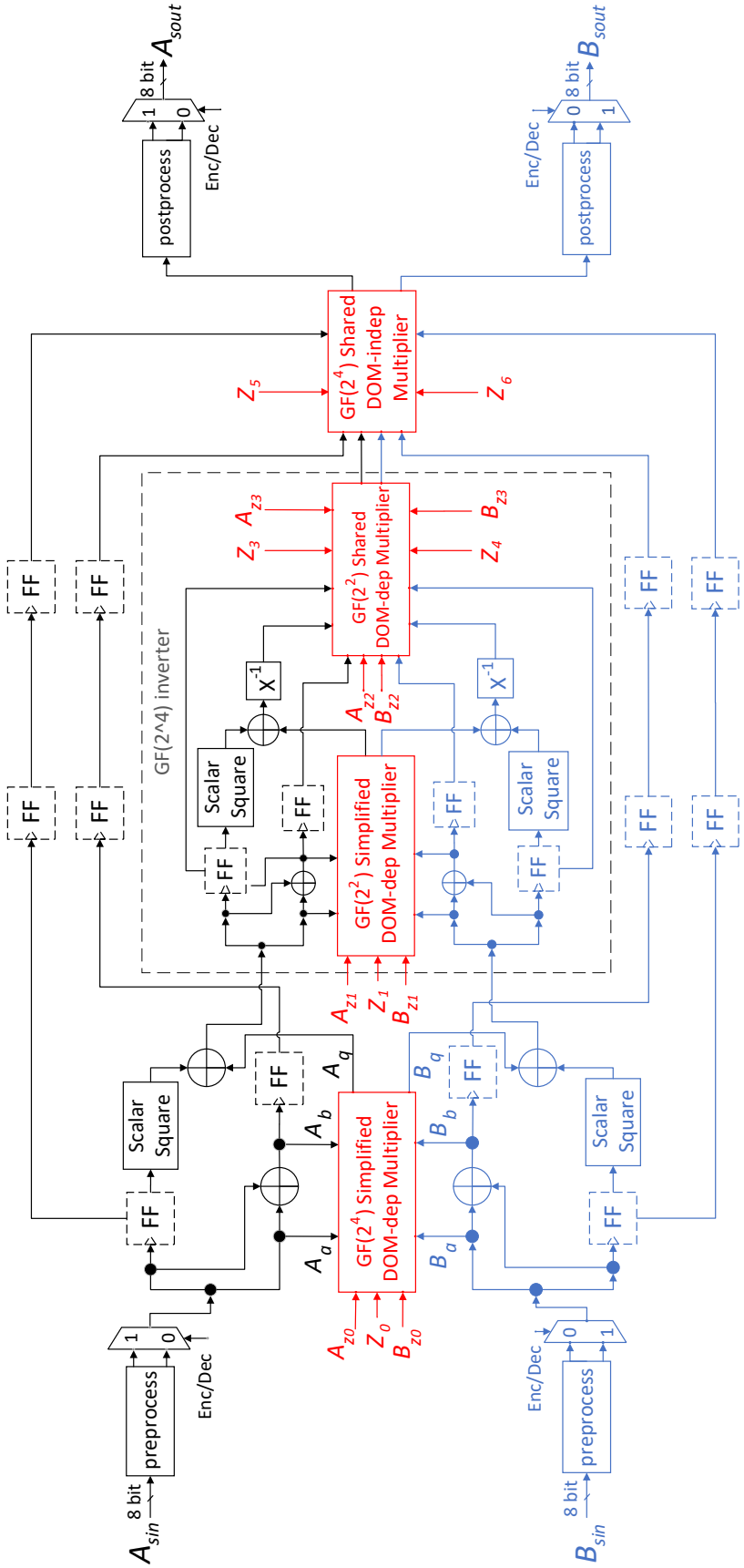


Figure 3.12: Structure of the 1^{st}-order five-stage DOM SBOX Module (modified based on [24])

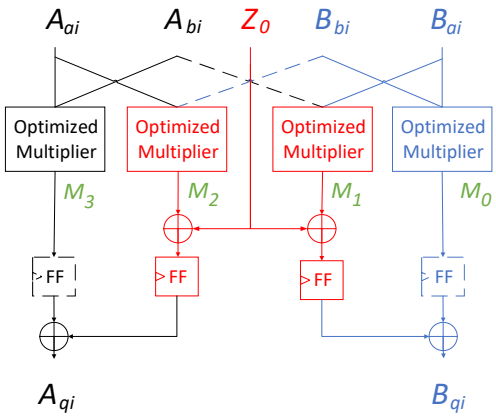


Figure 3.13: Proposed 1^{st}-order Simplified DOM-indep Multipliers based on [24]

Take GF(2^4) simplified DOM-indep multiplier as an example. Assume the inputs of the DOM-indep multiplier are $A_{ai} = \{A_{ai}[3], A_{ai}[2], A_{ai}[1], A_{ai}[0]\}$ and $A_{bi} = \{A_{bi}[3], A_{bi}[2], A_{bi}[1], A_{bi}[0]\}$. The intermediate results $A_{mi} = \{A_{mi}[4], A_{mi}[3], A_{mi}[2], A_{mi}[1], A_{mi}[0]\}$, which can be calculated as (3.15).

$$\begin{aligned} A_{mi}[4] &= A_{mi}[0] \oplus A_{mi}[1]; \\ A_{mi}[3] &= A_{ai}[0] \oplus A_{ai}[1]; \\ A_{mi}[2] &= A_{ai}[2] \oplus A_{ai}[3]; \\ A_{mi}[1] &= A_{ai}[0] \oplus A_{ai}[2]; \\ A_{mi}[0] &= A_{ai}[1] \oplus A_{ai}[3]. \end{aligned} \tag{3.15}$$

Assume 4-bit output of optimized multiplier $M_3 = \{M_3[3], M_3[2], M_3[1], M_3[0]\}$ (see Fig. 3.13), we can calculate its value using the following equation (3.16), which is similar to (3.2). Using the same approach, we can calculate the values of M_2, M_1, and M_0.

$$\begin{aligned} M_3[3] &= (A_{bi}[0]\&A_{ai}[3]) \oplus (A_{bi}[1]\&A_{mi}[2]) \oplus (A_{bi}[2]\&A_{mi}[0]) \oplus (A_{bi}[3]\&A_{mi}[4]); \\ M_3[2] &= (A_{bi}[0]\&A_{ai}[2]) \oplus (A_{bi}[1]\&A_{ai}[3]) \oplus (A_{bi}[2]\&A_{mi}[1]) \oplus (A_{bi}[3]\&A_{mi}[0]); \\ M_3[1] &= (A_{bi}[0]\&A_{ai}[1]) \oplus (A_{bi}[1]\&A_{mi}[3]) \oplus (A_{bi}[2]\&A_{mi}[2]) \oplus (A_{bi}[3]\&A_{ai}[2]); \\ M_3[0] &= (A_{bi}[0]\&A_{ai}[0]) \oplus (A_{bi}[1]\&A_{mi}[1]) \oplus (A_{bi}[2]\&A_{ai}[3]) \oplus (A_{bi}[3]\&A_{mi}[2]). \end{aligned} \tag{3.16}$$

After obtaining the values of M_3, M_2, M_1, and M_0, we can use them to calculate the outputs A_{qi} and B_{qi} as follows (3.17):

$$\begin{aligned} A_{qi} &= M_3 \oplus (M_2 \oplus Z_0); \\ B_{qi} &= (M_1 \oplus Z_0) \oplus M_0. \end{aligned} \tag{3.17}$$

3.3.4. Simpilified DOM-dep Multiplier

The simplified DOM-dep multiplier is designed based on the simplified DOM-indep multiplier, as shown in Fig. 3.14. In contrast to the DOM-dep multiplier depicted in Fig. 2.8, our simplified DOM-dep multiplier shared the resources between two right multipliers (also see Fig. 2.8). Besides, simplified DOM-indep multiplier (see Fig. 3.13 is employed instead of conventional DOM-indep multiplier (see Fig. 2.7).

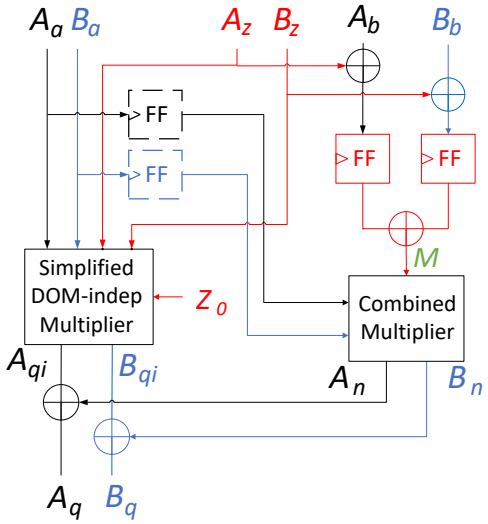


Figure 3.14: Proposed 1^{st}-order Simplified DOM-dep Multipliers based on [24]

A_a, B_a, A_b, B_b are the inputs of the first simplified DOM-indep, while A_q and B_q are the outputs. A_z, B_z, and Z_0 are random numbers that used to ensure the independence of shares. Equation (3.18) illustrates the expression of DOM-dep multiplier, where intermediate values M=$(A_b \oplus B_b) \oplus (A_z \oplus B_z)$, $A_n = A_a * M$, and $B_n = B_a * M$. We denote that a= $A_a \oplus B_a$, b= $A_b \oplus B_b$, z= $A_z \oplus B_z$, and q= $A_q \oplus B_q$.

$$
\begin{aligned}
& a * b \\
& = a * (b \oplus z) + a * z \\
& = (A_a \oplus B_a)(A_b \oplus B_b \oplus A_z \oplus B_z) \oplus (A_a \oplus B_a)(A_z \oplus B_z) \\
& = (A_a * M \oplus B_a * M) \oplus (A_{qi} \oplus B_{qi}) \\
& = (A_n \oplus B_n) \oplus (A_{qi} \oplus B_{qi}) \\
& = (A_n \oplus A_{qi}) \oplus (B_n \oplus B_{qi}) \\
& = A_q \oplus B_q = q
\end{aligned}
\tag{3.18}
$$

The combined multiplier (see Fig. 3.5) is utilized for the calculation of $(A_a * M \oplus B_a * M)$ to reduce area. The detailed computation procedure is illustrated as follows. Assume $A_a = \{A_a[3], A_a[2], A_a[1], A_a[0]\}$, $B_a = \{B_a[3], B_a[2], B_a[1], B_a[0]\}$, $M = \{M[4], M[3], M[2], M[1], M[0]\}$, $A_n = \{A_n[3], A_n[2], A_n[1], A_n[0]\}$, and $B_n = \{B_n[3], B_n[2], B_n[1], B_n[0]\}$. The intermediate results $A_m = \{A_m[4], A_m[3], A_m[2], A_m[1], A_m[0]\}$, which can be calculated as (3.19).

$$
\begin{aligned}
& A_m[4] = A_m[0] \oplus A_m[1]; \\
& A_m[3] = M[0] \oplus M[1]; \\
& A_m[2] = M[2] \oplus M[3]; \\
& A_m[1] = M[0] \oplus M[2]; \\
& A_m[0] = M[1] \oplus M[3].
\end{aligned}
\tag{3.19}
$$

Then the outputs of the combined multiplier in Fig. 3.14 can be calculated as (3.20).

$$
\begin{aligned}
A_n[3] &= (A_a[0]\&M[3]) \oplus (A_a[1]\&A_m[2]) \oplus (A_a[2]\&A_m[0]) \oplus (A_a[3]\&A_m[4]);\\
A_n[2] &= (A_a[0]\&M[2]) \oplus (A_a[1]\&M[3]) \oplus (A_a[2]\&A_m[1]) \oplus (A_a[3]\&A_m[0]);\\
A_n[1] &= (A_a[0]\&M[1]) \oplus (A_a[1]\&A_m[3]) \oplus (A_a[2]\&A_m[2]) \oplus (A_a[3]\&M[2]);\\
A_n[0] &= (A_a[0]\&M[0]) \oplus (A_a[1]\&A_m[1]) \oplus (A_a[2]\&M[3]) \oplus (A_a[3]\&A_m[2]).
\end{aligned}
\tag{3.20}
$$

$$
\begin{aligned}
B_n[3] &= (B_a[0]\&M[3]) \oplus (B_a[1]\&A_m[2]) \oplus (B_a[2]\&A_m[0]) \oplus (B_a[3]\&A_m[4]);\\
B_n[2] &= (B_a[0]\&M[2]) \oplus (B_a[1]\&M[3]) \oplus (B_a[2]\&A_m[1]) \oplus (B_a[3]\&A_m[0]);\\
B_n[1] &= (B_a[0]\&M[1]) \oplus (B_a[1]\&A_m[3]) \oplus (B_a[2]\&A_m[2]) \oplus (B_a[3]\&M[2]);\\
B_n[0] &= (B_a[0]\&M[0]) \oplus (B_a[1]\&A_m[1]) \oplus (B_a[2]\&M[3]) \oplus (B_a[3]\&A_m[2]).
\end{aligned}
$$

The shared DOM-indep/DOM-dep multiplier modules in Fig. 3.12 are the combination of two simpilified DOM-indep/DOM-dep multipliers (see Fig. 3.13). We share the common resources between these two multipliers, resulting a further area reduction.

4

Experimental Results

This chapter presents the experiments setup, simulation validation, performed experiments, evaluations of the obtained results, and security analysis. Section 4.1 details the prerequisites and tools of experiments. Section 4.2 presents the pre-synthesis and post-synthesis simulation results for both AES and DOM designs. Section 4.3 offers a comparison of the area, power, and performance of the proposed AES designs with state-of-the-art. Section 4.4 initially compares the area of the simplified DOM SBOX with that of state-of-the-art solutions. It then proceeds to compare the area, power, and performance of various proposed DOM-version AES designs with different bit configurations. Section 4.5 analyzes the security of proposed AES and DOM designs. Section 4.6 provides a concise discussion of the results presented in this chapter.

4.1. Setup

We re-implemented the state-of-the-art AES design proposed by Davis and Jones [20] and compared it with our proposed AES designs. All designs are synthesized using TSMC CMOS 180 nm technology. In the majority of IoT applications, the maximum payload size for each packet is between 1600 bytes (e.g., NarrowBand-IoT [57]) and 256 megabytes (e.g., Message Queue Telemetry Transport (MQTT) [58]). Taking the lower limit into consideration, we make the assumption that the key will stay the same during the communication session of at least one hundred encrypting/decrypting operations. For that reason, we assumed a fixed key for 100 encryption and decryption operations. Initially, we conducted simulations using NC-Sim to validate the functional correctness of the designs. Then, using Synopsys Design and Power Compiler, we computed the total area and power consumption of each design for these operations. Subsequently, we implemented an updated version of DOM using our proposed AES approach, and then compared the area of each implementation. Finally, we conducted security analysis of the non-DOM AES design and DOM-based AES design to evaluate the security of all designs. We emulated both the proposed lightweight AES and proposed lightweight DOM in an FPGA and collected actual power traces from it. To realize this, we used Chipwhisperer CW305 equipment [59], which contains a Xilinx FPGA and an Analogue-to-Digital Converter (ADC) with a maximum sampling rate of 105 MS/s. The attacks were implemented using python software language and attack libraries such as SCALib [60]. The detailed introduction of the tools used in the development of this project will be presented as follows.

4.1.1. TSMC 0.18-micron Technology

Taiwan Semiconductor Manufacturing Company (TSMC) offered the world's first 0.18-micron (µm) low power process technology in 1998 [61]. The 0.18-micron technology allowed for the integration of various components on a single chip, including logic circuits, memory, and other functional blocks. This integration contributed to the miniaturization of electronic devices and reduced manufacturing costs.

4.1.2. NC-Sim

NC-SIM is a widely used Electronic Design Automation (EDA) tool for simulating and verifying digital hardware designs [62]. Developed by Cadence Design Systems, it is part of the Cadence Verification

Suite and is commonly utilized in the design and verification process of complex integrated circuits and digital systems.

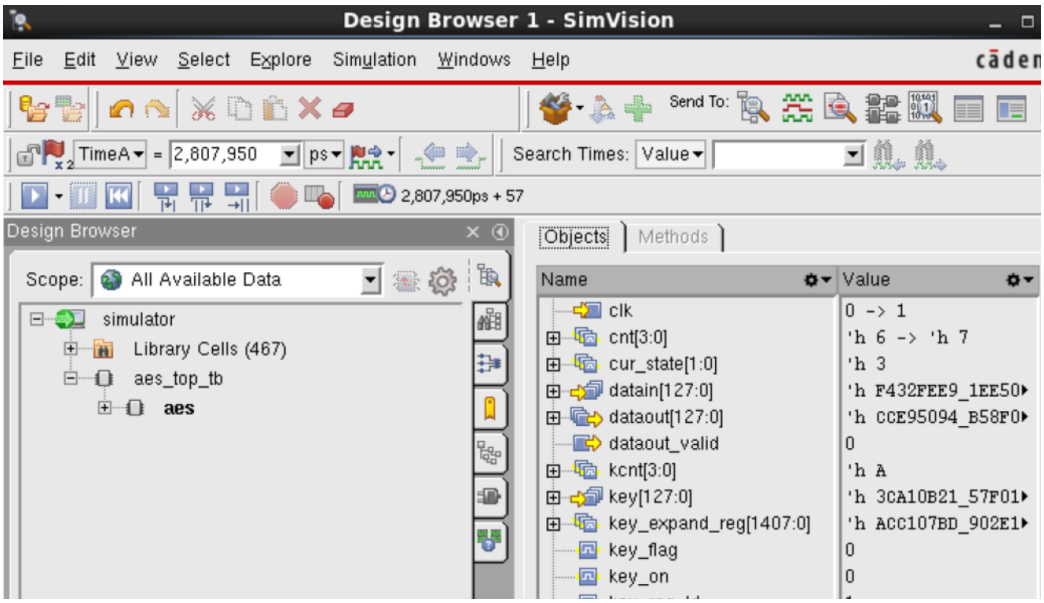


Figure 4.1: NC-Sim User Interface

4.1.3. Synopsys Design Compiler and Spyglass Power

Synopsys Design Compiler (DC) [63] is a widely used EDA tool that plays a critical role in digital synthesis during the development of integrated circuits. DC supports a wide range of Hardware Description Languages (HDLs) and allows designers to write Register Transfer Level (RTL) code using languages like VHDL or Verilog. It can handle complex designs and optimize them for various performance metrics. Besides, it offers gate-to-gate optimization for smaller area on new or legacy designs while maintaining timing Quality of Results. It also provides advanced power optimization techniques to reduce the power consumption of the synthesized design. It includes features such as clock gating, power shutoff, and power-aware optimizations to achieve low-power targets.

```
log

                        Design Compiler Graphical
                              DC Ultra (TM)
                               DFTMAX (TM)
                           Power Compiler (TM)
                              DesignWare (R)
                              DC Expert (TM)
                            Design Vision (TM)
                            HDL Compiler (TM)
                            VHDL Compiler (TM)
                               DFT Compiler
                            Design Compiler(R)

                    Version L-2016.03-SP3 for linux64 - Jul 18, 2016

                         Copyright (c) 1988 - 2016 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
            or distribution of this software is strictly prohibited.
Initializing...
Starting shell in Topographical mode...
        proc_machine_info
        proc_time
Initializing gui preferences from file  /home/huangry/.synopsys_dv_prefs.tcl
proc_time START_DC
Starting Timer for the First Time ....
```

Figure 4.2: Synopsys DC Command-line Interface

Synopsys SpyGlass Power [64] enables accurate power estimation at the RTL and gate levels, eliminating the time wasted in waiting for the final netlist for precise power numbers. This enables designers to assess power consumption early in the design process, helping them make informed decisions and trade-offs to meet power targets.

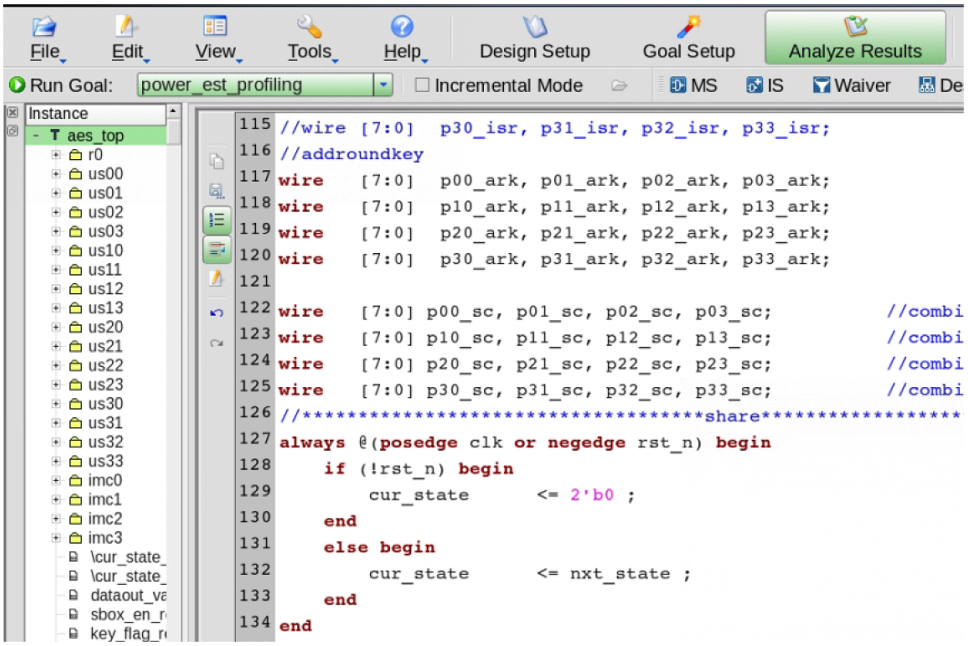


Figure 4.3: Spyglass Power User Interface

4.1.4. CW305 Artix-7 FGPA target board and CW1173 ChipWhisperer-Lite

The CW305 is a standalone FPGA target board [65], which is designed by NewAE Technology. It enables embedded security analysis testing on the Xilinx Artix-7 Field Programmable Gate Array (FPGA) devices. This board is designed to interface to hardware such as the ChipWhisperer Pro (CW1200) or the ChipWhisperer-Lite (CW1173) Capture Tools.

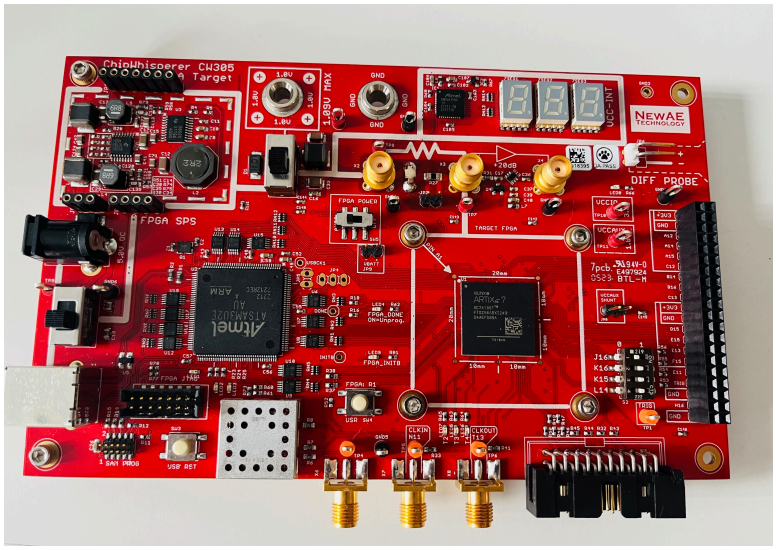


Figure 4.4: CW305 Artix-7 FGPA target board

The NewAE Technology ChipWhisperer-Lite (CW1173) [66] is a cost-effective device for design engineers and students to perform DPA and investigate glitch vulnerabilities in their hardware projects.

It is an integral component of an open-source tool chain dedicated to for perform side-channel power analysis measurement, device programming, and glitching. Besides, it includes a 10-bit ADC with a sampling rate of 105 MS/s, combined with up to +55 dB gain amplifier, facilitating easier measurements of small signals.

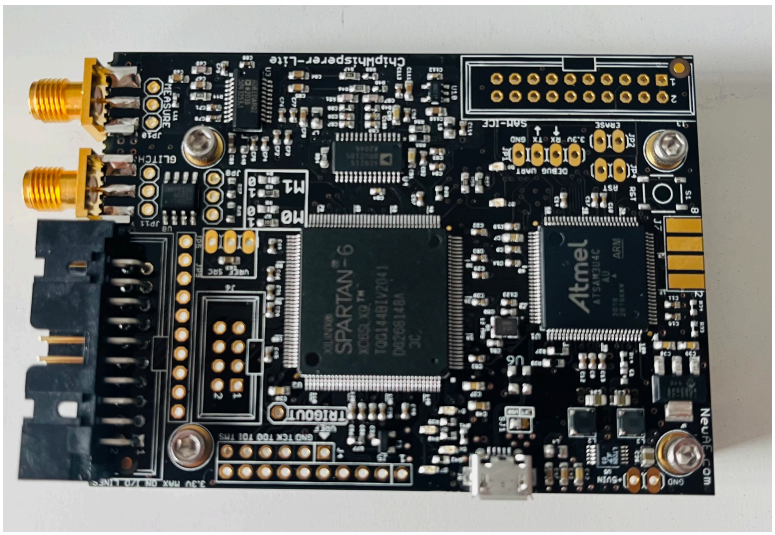


Figure 4.5: CW1173 ChipWhisperer-Lite

4.1.5. Vivado 2019.1

Vivado 2019.1 is a version of Xilinx's Vivado Design Suite, which is a comprehensive electronic design automation (EDA) tool used for designing, synthesizing, and implementing digital systems on Xilinx FPGAs and System-on-Chips (SoCs).

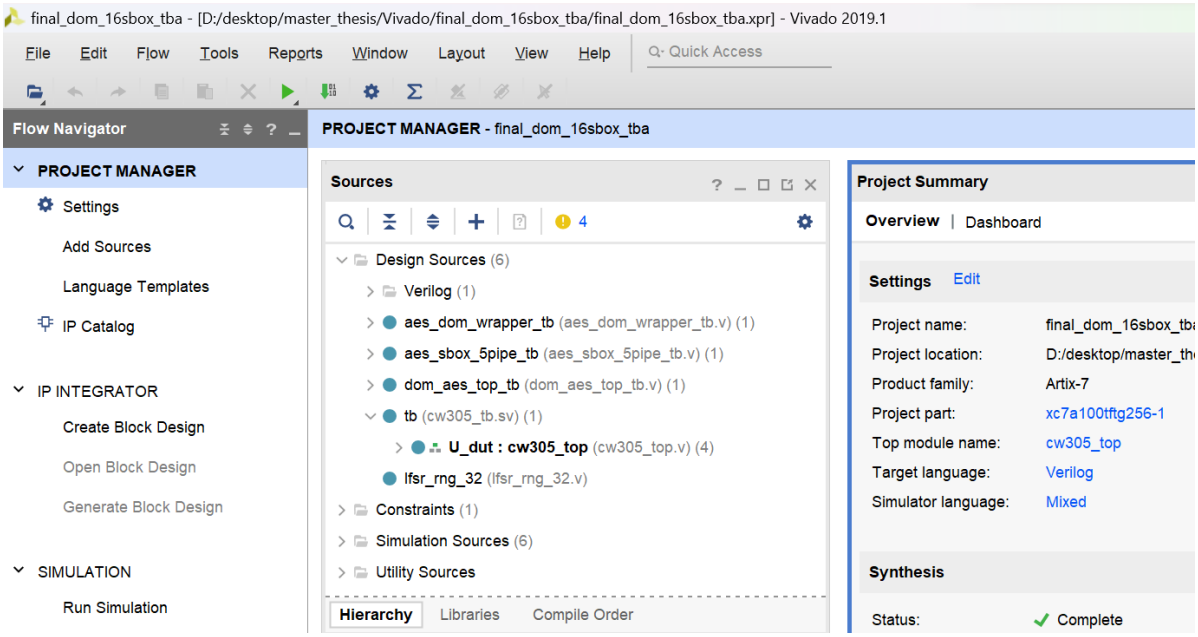


Figure 4.6: Vivado 2019.1 User Interface

4.1.6. Jupyter Notebook

Jupyter Notebook, previously known as IPython Notebook, constitutes a web-based interactive computational environment designed for crafting notebook documents [67]. Within a Jupyter Notebook, individuals have the ability to compose and run code across a range of programming languages like Python, R, and Julia, all within the confines of the notebook interface. This platform has gained widespread recognition as an Integrated Development Environment (IDE) of choice for researchers in data science and machine learning.

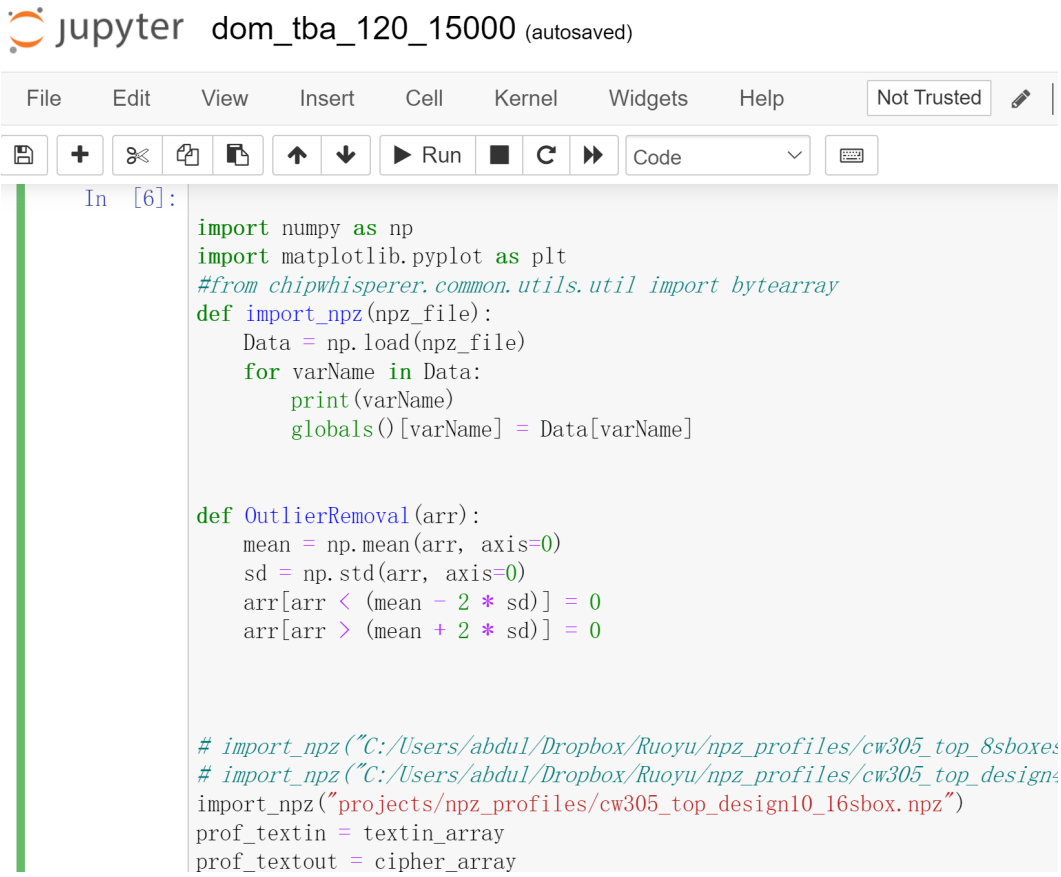


Figure 4.7: Jupyter Notebook User Interface

4.2. Simulation of AES and DOM designs

To validate the functional correctness of our designs, we utilized NC-Sim for simulation. Initially, we conducted the pre-synthesis simulation of the AES design, as depicted in Fig. 4.8. After synthesizing the RTL code into gate-level representations, like netlists, we proceeded with post-synthesis simulation to ensure the functionality of this gate-level implementation. For the post-synthesis simulation of the AES design, we utilized the gate-level netlist, the top file (output of synthesis), and the corresponding testbench. The simulation results are depicted in Fig. 4.9.

Likewise, we also performed pre-synthesis and post-synthesis simulations for DOM designs, as shown in shown in Fig. 4.10 and Fig. 4.11, respectively.

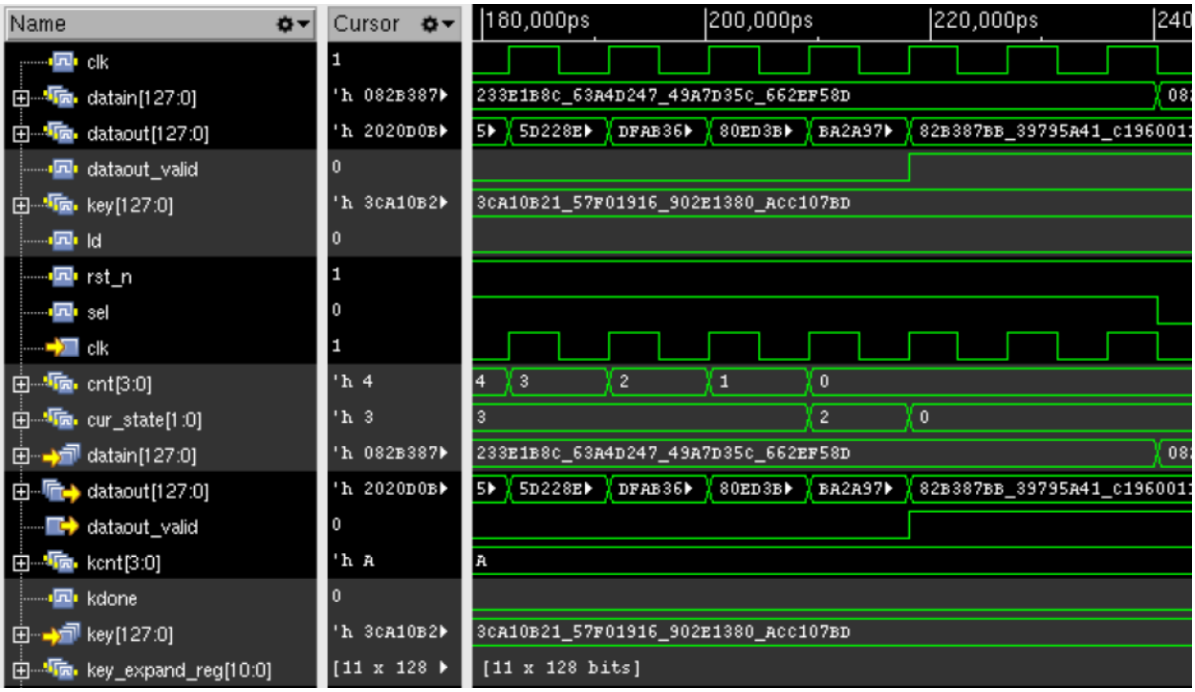


Figure 4.8: Partial results of AES pre-synthesis simulation

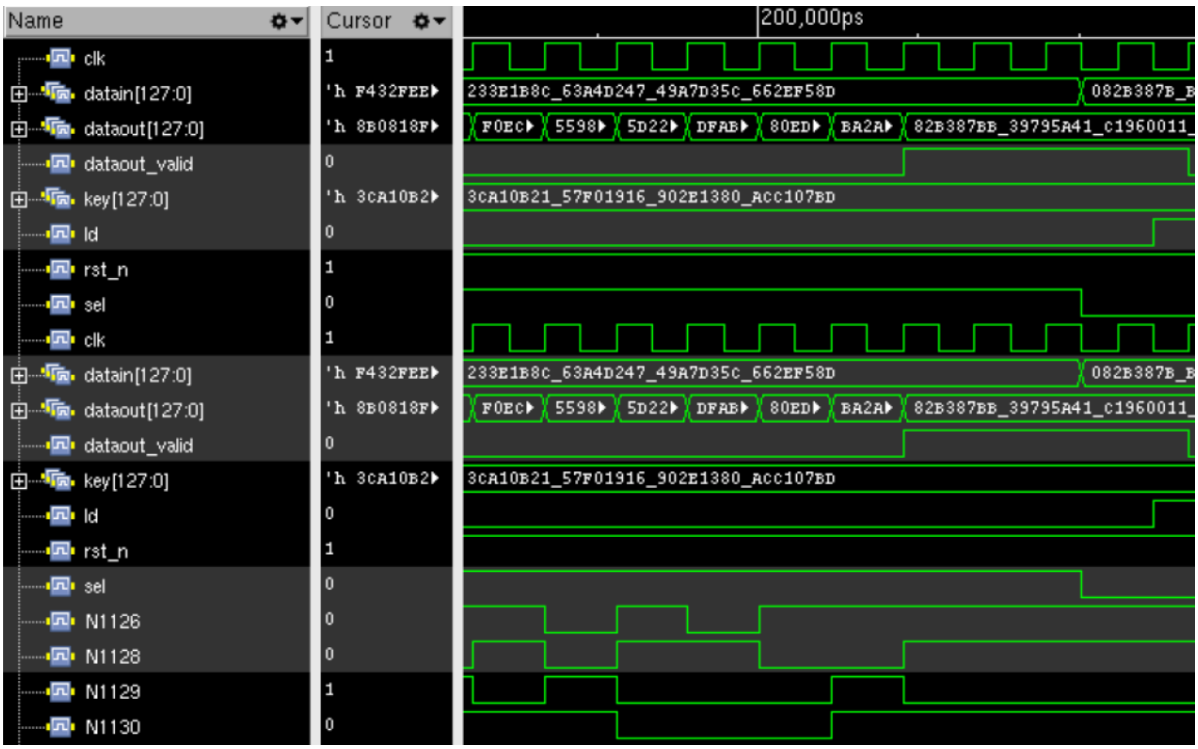


Figure 4.9: Partial results of AES post-synthesis simulation

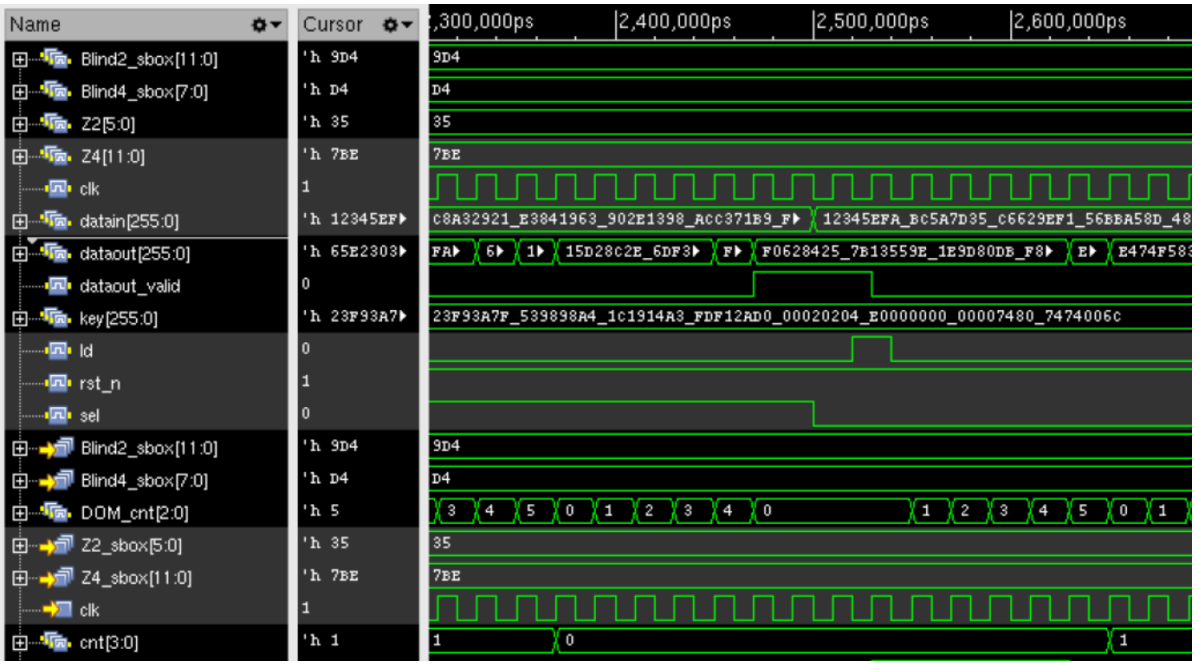


Figure 4.10: Partial results of DOM pre-synthesis simulation

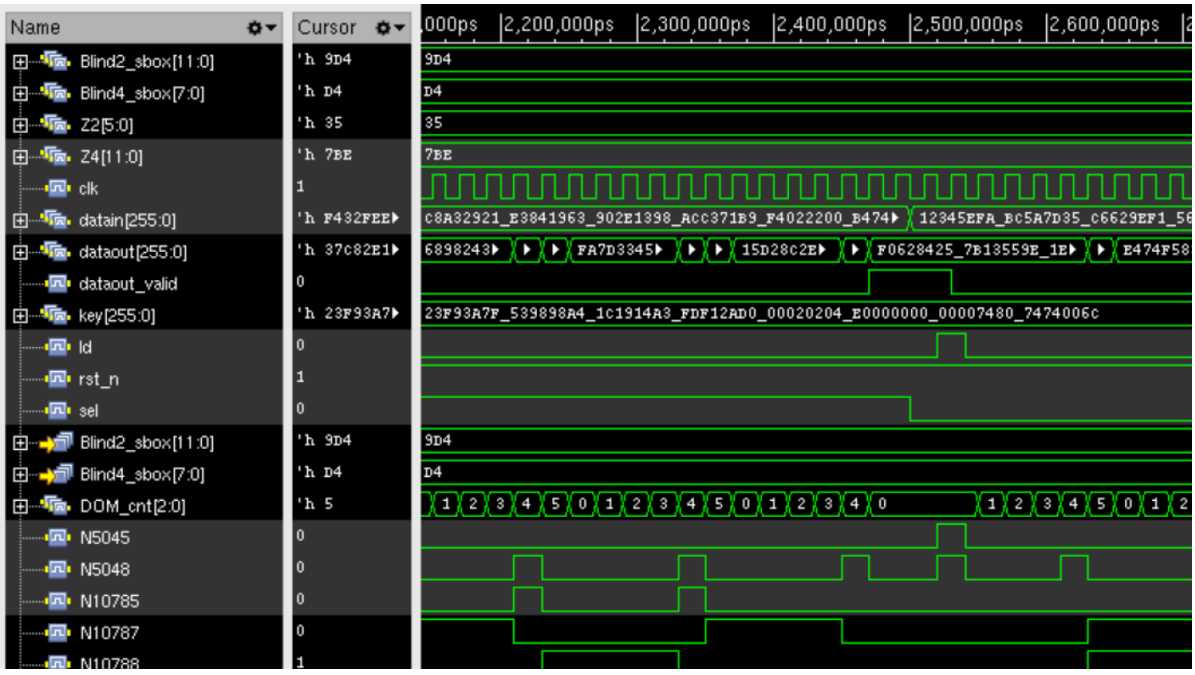


Figure 4.11: Partial results of DOM post-synthesis simulation

4.3. AES Performance Evaluation

In our design, we chose to implement the SBOX using methods that are similar to those described in [55], which we believe to be the most efficient approach. Then, we further utilized several optimization techniques, which are described in Subsection 3.2. After synthesising SBOXes using TSMC CMOS 180 nm technology, we obtained that the area of our proposed SBOX and state-of-the-art SBOX is 2558 and 2788 μm^2, respectively (see Fig. 4.12). The results show that our proposed SBOX achieved an area reduction of 8.2% when compared with SBOX in [55].

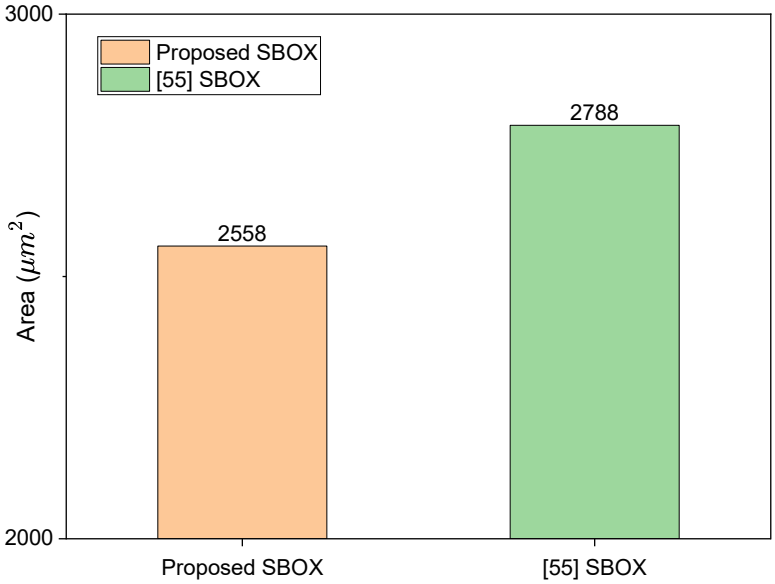


Figure 4.12: Area Analysis of Proposed and state-of-the-art [55] SBOX

For comparing the entire AES, we chose the paper [20] as it represents a better overall implementation in terms of area, power, and speed. Table 4.1 and Table 4.2 compare the area and total cycle number of our proposed AES designs with state-of-the-art design [20] at 50MHz and maximum clock frequency, respectively. Table 4.1 shows that our design achieves approximately 13% and 20% reduction in area for the 64-bit and 32-bit designs, respectively, compared to the state-of-the-art. Table 4.2 illustrates that our 32-bit design achieves an 18% increase in frequency while a 14% reduction in area compared to the design in [20]. Note that we evaluate the design using 100 encryption/decryption operations to demonstrate the key expansion optimization technique. In this evaluation, the first 11 cycles are dedicated to key expansion. Subsequently, in the 128-bit, 64-bit, and 32-bit data-path designs, encryption/decryption is performed over 11, 21, and 41 cycles, respectively. As a result, it takes 2211 (11*2*100+11), 4211 (21*2*100+11), and 8211 (41*2*100+11) cycles to perform 100 encryption/decryption operations.

Table 4.1: AES Performance Analysis at 50MHz

Design	Data-path	Frequency (MHz)	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
[20]	32-bit	50	174156	1	11100	1
Proposed AES	32-bit	50	139415	0.8	8211	0.74
	64-bit	50	151677	0.87	4211	0.38
	128-bit	50	178674	1.03	2211	0.2

Table 4.2: AES Performance Analysis at the Maximum Frequency

Design	Data-path	Frequency (MHz)	Frequency Ratio	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
[20]	32-bit	103.1	1	196857	1	11100	1
Proposed AES	32-bit	121.9	1.18	169794	0.86	8211	0.74
	64-bit	117.6	1.14	195355	0.99	4211	0.38
	128-bit	112.4	1.09	236553	1.2	2211	0.2

Fig. 4.13 depicts the performance per area of our proposed AES design and the state-of-the-art design [20]. Fig. 4.14 illustrates the performance per power of these designs. Since the performance and power of the design increase proportionally with frequency, running the design at a 50MHz fre-

quency yields similar results to running it at the maximum frequency. The analysis shows that among our proposed designs, the 128-bit design achieves the highest score in terms of performance per area and performance per power. Our proposed 128-bit, 64-bit, and 32-bit designs surpass the state-of-the-art [20] in terms of performance per area by a factor of 4.90x, 3.02x, and 1.69x, respectively, when operating at a 50MHz frequency and by a factor of 4.55x, 3.03x, and 1.85x, respectively, when operating at the maximum frequency. They also outperform the state-of-the-art design in terms of performance per power by a factor of 2.68x, 1.70x, and 1.27x, respectively. Note that in Fig. 4.14, only the performance per power for the 50 MHz implementation is displayed, as there were minimal differences observed when compared to the designs operating at their maximum frequencies.

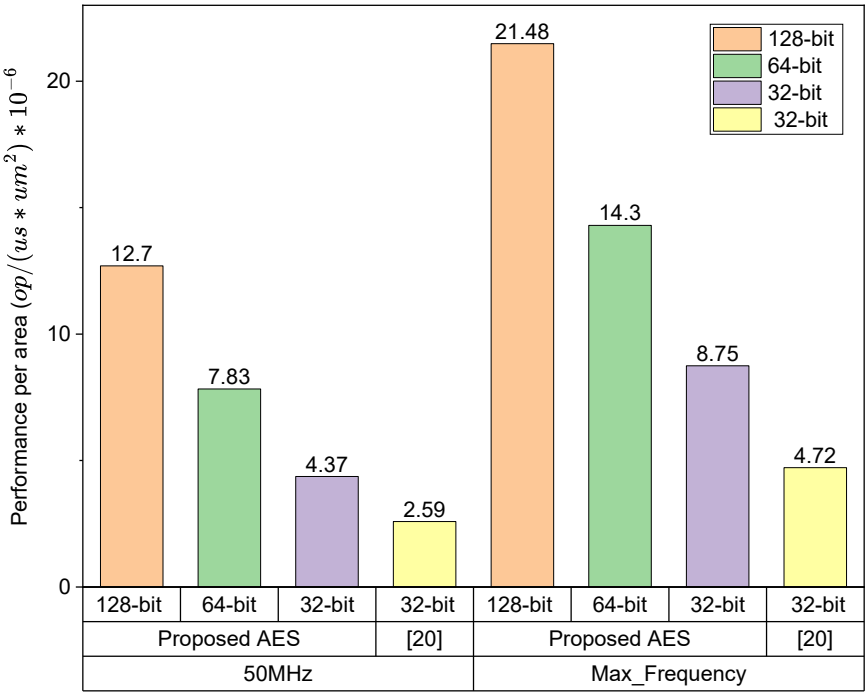


Figure 4.13: Performance per Area Comparison vs AES Design in [20]

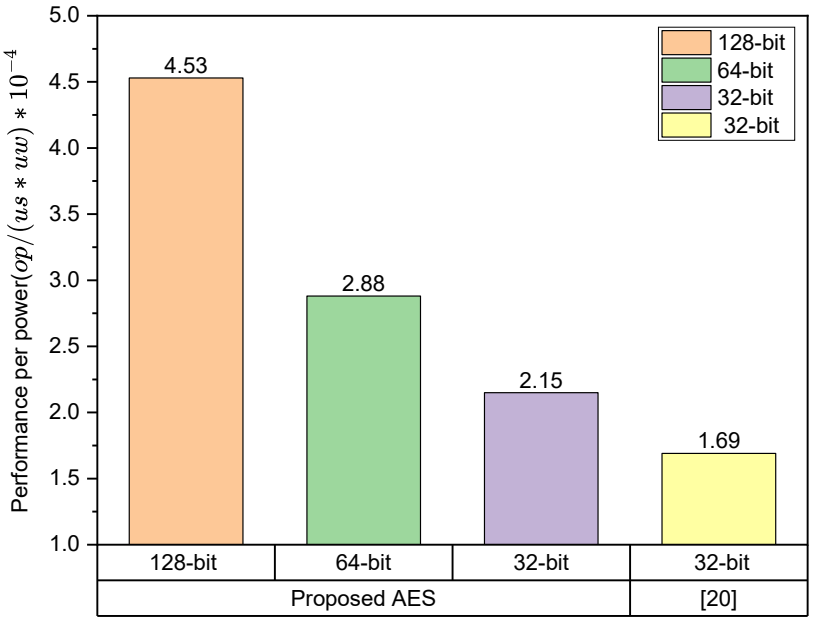


Figure 4.14: Performance per Power Comparison vs AES Design in [20]

4.4. DOM Performance Evalution

Table 4.3 shows a comparison of the area between our proposed 1^{st}-order DOM SBOX and the original 1^{st}-order SBOX [24]. Compared with the original design, our eight-stage and five-stage 1^{st}-order DOM SBOX designs achieve area reductions of 9.9% and 6.9%, respectively (See Table 4.3).

Table 4.3: Area comparison of DOM SBOX

Design	SBOX Type	Area (μm^2)	Area Ratio
[24]	eight-stage	19682	1
DOM SBOX	five-stage	21196	1.077
Proposed	eight-stage	17735	0.901
DOM SBOX	five-stage	19741	1.003

Despite the optimal balance achieved by the 128-bit AES design, the proposed DOM SBOX is implemented across all AES designs, including the 128-bit, 64-bit, and 32-bit versions, in order to comprehensively evaluate the impact of these designs on overall performance. Table 4.4 and table 4.5 present a comparison of the area and total cycles of the 1^{st}-order DOM AES designs, operating at a frequency of 50 MHz and maximum frequency, respectively. As we mentioned in the Section 3.3, we chose the five-stage SBOX in our designs because it offers a substantial performance improvement with only minor sacrifices in terms of area when compared to the eight-stage SBOX. Consequently, key expansion process necessitates 51 (5*10+1) cycles, while the encryption and decryption operations in the 128-bit, 64-bit, and 32-bit data-path designs require 51 (5*10+1), 61 (6*10+1), and 81 (8*10+1) cycles, respectively. As a result, it takes 10251 (51*2*100+51), 12251 (61*2*100+51), and 16251 (81*2*100+51) cycles for these designs to perform 100 encryption/decryption operations. Tables 4.4 and 4.5 demonstrate that the 32-bit design exhibits a better area, whereas the 128-bit design has a higher performance.

Table 4.4: DOM Performance Analysis at 50MHz

Data-path	Frequency (MHz)	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
128-bit	50	615172	1	10251	1
64-bit	50	445269	0.72	12251	1.2
32-bit	50	361582	0.59	16251	1.59

Table 4.5: DOM Performance Analysis at the Maximum Frequency

Data-path	Frequency (MHz)	Frequency ratio	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
128-bit	188.7	1.79	698780	1	10251	1
64-bit	190.8	1.81	522844	0.75	12251	1.2
32-bit	192.3	1.83	446528	0.64	16251	1.59

The DOM SBOX includes a great number of registers, resulting in increased power consumption and area. Therefore, lower data-path designs can significantly reduce area and power consumption by utilizing fewer DOM SBOXes. Fig. 4.15 shows the performance per area comparison of our proposed DOM designs. According to this figure, the 64-bit design performs better at both 50 MHz and the maximum frequency. Fig. 4.16 illustrates a comparison of the performance per power of our proposed DOM designs, where the 32-bit design outperforms the others.

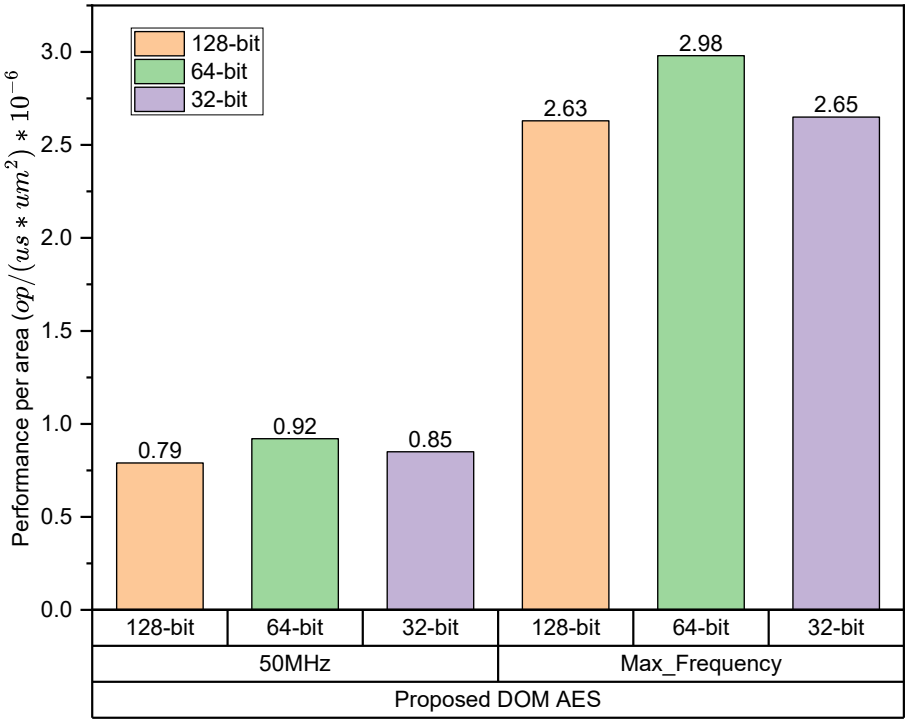


Figure 4.15: Performance per area comparison of our proposed 1^{st}-order DOM AES designs

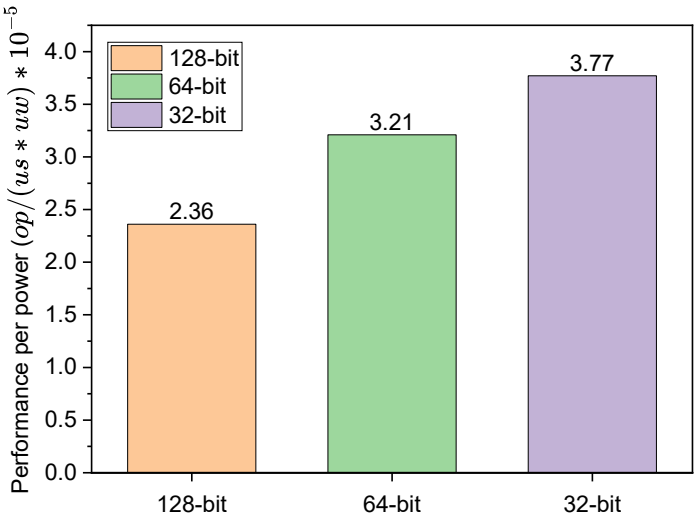


Figure 4.16: Performance per power comparison of our proposed 1^{st}-order DOM AES designs

4.5. Security Analysis

In this section, we capture the power traces of AES and DOM designs and subsequently conduct Correlation Power Analysis (CPA) and Template Based Attack (TBA) on both designs to assess their security level.

4.5.1. Power Traces Record

We connected our non-DOM and DOM AES core to the CW305 Artix-7 FPGA, and generated the corresponding bitstream using *Vivado 2019.1*. Subsequently, we connected the cw305 Artix-7 FPGA and CW1173 ChipWhisperer-Lite devices with the laptop, and initialized the devices. We verify the encryption results' correctness and recorded the power trace of designs for CPA and TBA using Jupyter

Notebook, illustrated in Fig. 4.17. If there is no incorrect encryption result in the last round of power trace, we conclude that all the recorded results are accurate and reliable. Fig. 4.17 confirms the correctness of all results and the successful recording of power traces.

```
for i in trange(N, desc='Capturing traces'):
    # run aux stuff that should come before trace here
    key, text = ktp.next()  # manual creation of a key, text pair can be substituted here
    textin.append(text)
    keys.append(key)
    ret = cw.capture_trace(scope, target, text, key)
    if not ret:
        print("Failed capture")
        continue
    assert (list(ret.textout) == list(cipher.encrypt(bytes(text)))),"Incorrect encryption result!
    #trace += scope.getLastTrace()
    if(i==N-1):
        print("PASS! Results are correct!")
        continue
    traces.append(ret.wave)
    project.traces.append(ret)
```

```
Capturing traces: 100%|████████████████████████████████████████████████
████████████████████| 5000/5000 [03:54<00:00, 21.32it/s]
```

```
PASS! Results are correct!
```

Figure 4.17: Verification of encryption results and Power tracing

Both the TBA and CPA are used in our security analysis to access the resistance of the proposed designs against side channel attacks. To evaluate the attack's performance, reliable metrics are essential.The Guessing Entropy (GE), which represents the average number of successive guesses required when employing an optimal strategy to ascertain the true value of a random variable X, is widely accepted as a suitable metric for SCA [68]. A rank of zero indicates successful retrieval of the correct sub-key by the attacker, whereas a rank of 255 represents the lowest confidence in guessing the right sub-key. Here, we rank the possible values of the subkey from most to least possibility based on the value of the PCC (for CPA) or PDF (for TBA). A higher PCC or PDF corresponds to a higher likelihood of the subkey being accurate. The Partial Guessing Entropy (PGE) indicates the mean value of GE of each subkey [69].

4.5.2. CPA Attacks on AES and DOM Designs

We initiated the Correlation Power Analysis (CPA) with a dataset of 5000 power traces, selecting the output of the SBOX as the target for attack, and employing the Hamming Weight as the chosen leakage model. In the CPA attack scenario (see Fig. 4.18), it is difficult to estimate the correct key values. In our For a comprehensive security evaluation, we expanded our efforts by collecting an extensive dataset of 15000 power traces. Fig. 4.19 illustrates that the designs still remain resilient, which highlights the effectiveness of optimization techniques in enhancing security. To delve deeper into the analysis, we conducted a detailed examination of the rank distribution for each subkey. The outcomes of this examination are presented in Fig. 4.20 and Fig. 4.21. Regarding the AES design, the rank of only one subkey approaches a value close to 0, and the ranks of two additional subkeys remain below 50. For DOM design, none of the subkeys display a rank nears 0, and only a single subkey's rank is below 50. While both designs demonstrate notable resistance against CPA, the DOM design exhibits a higher level of resilience.

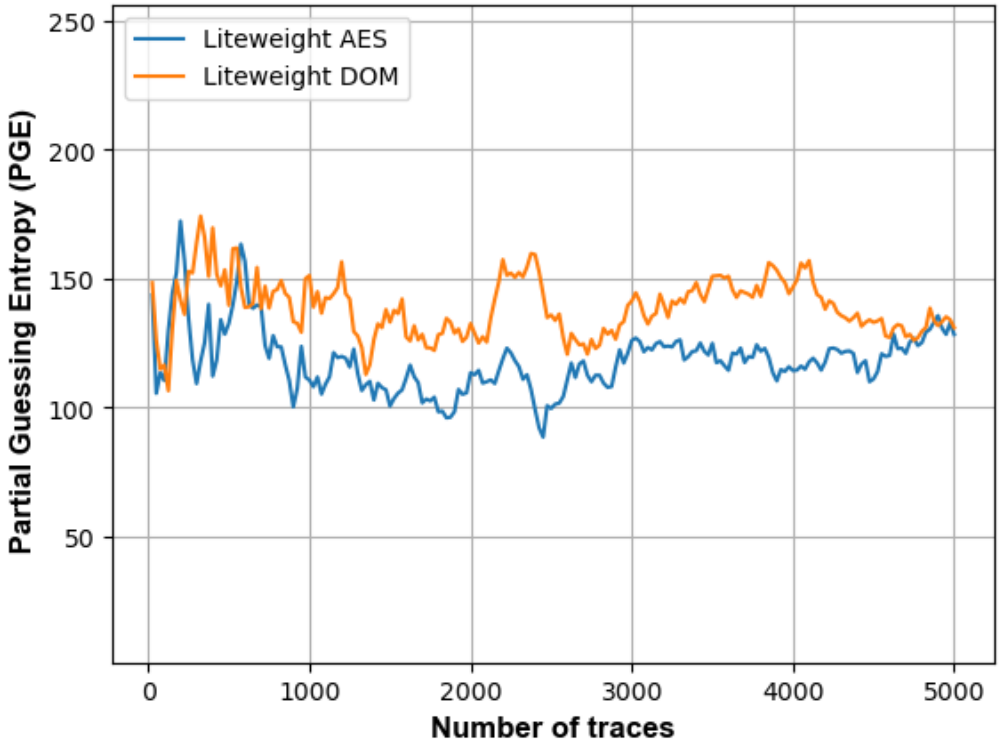


Figure 4.18: PGE of the proposed designs using CPA (5000 power traces)

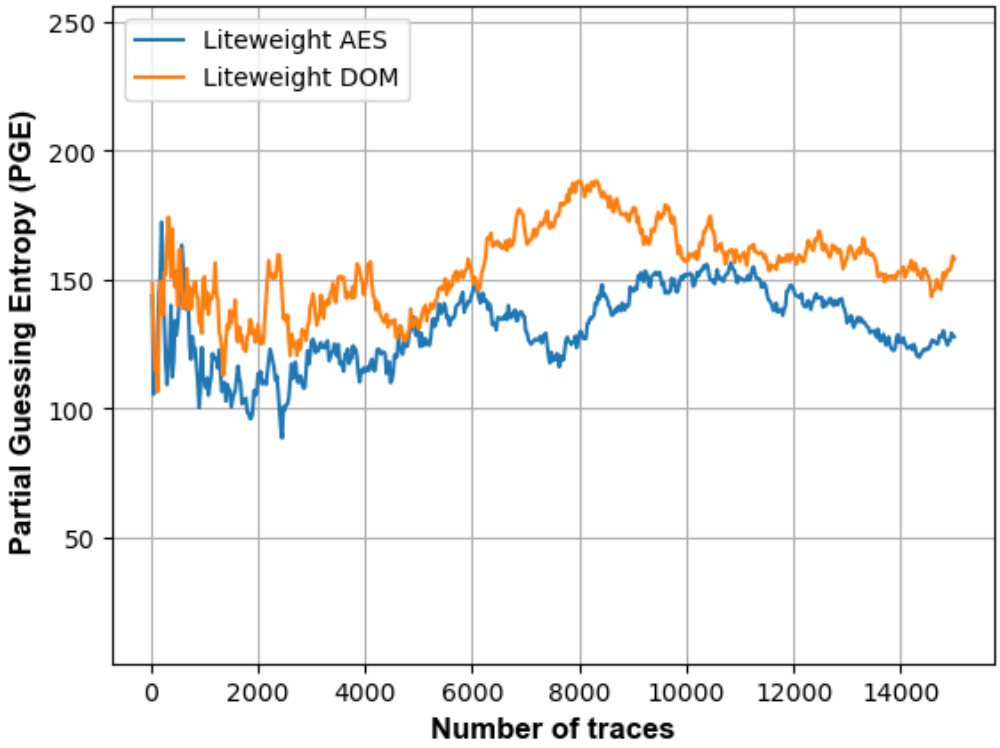


Figure 4.19: PGE of the proposed designs using on CPA (15000 power traces)

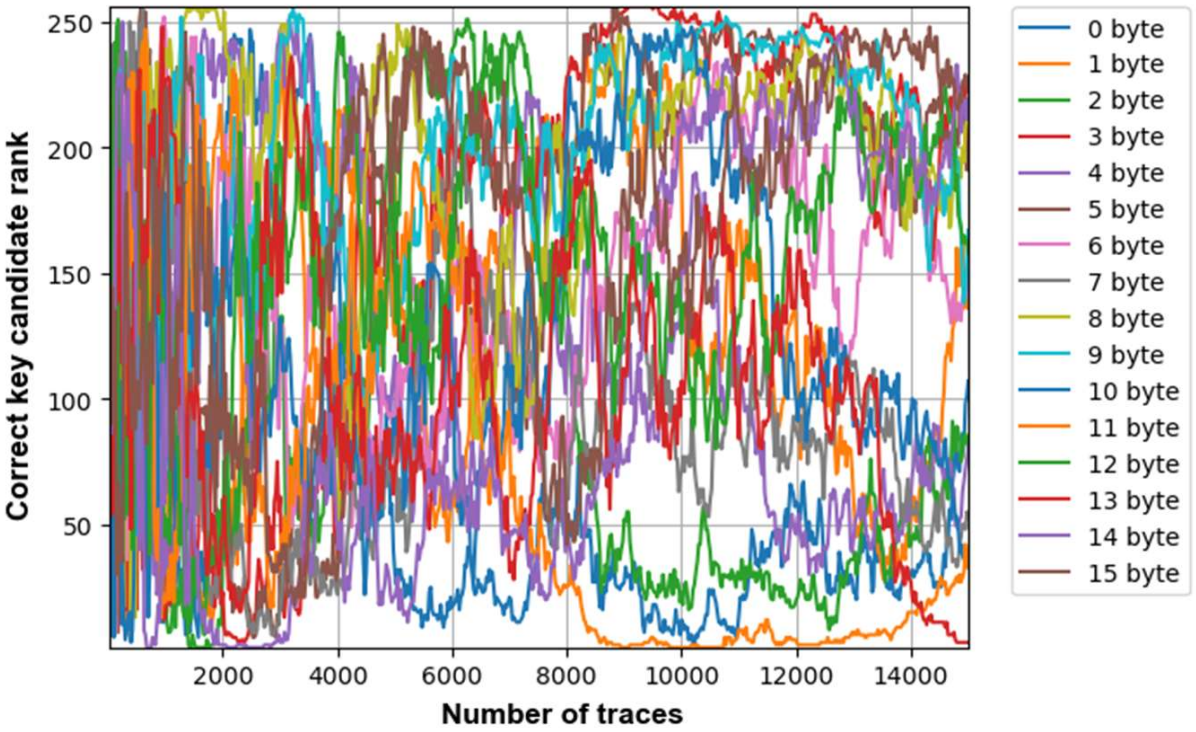


Figure 4.20: Rank analysis of the AES implementations based on CPA (15000 power traces)

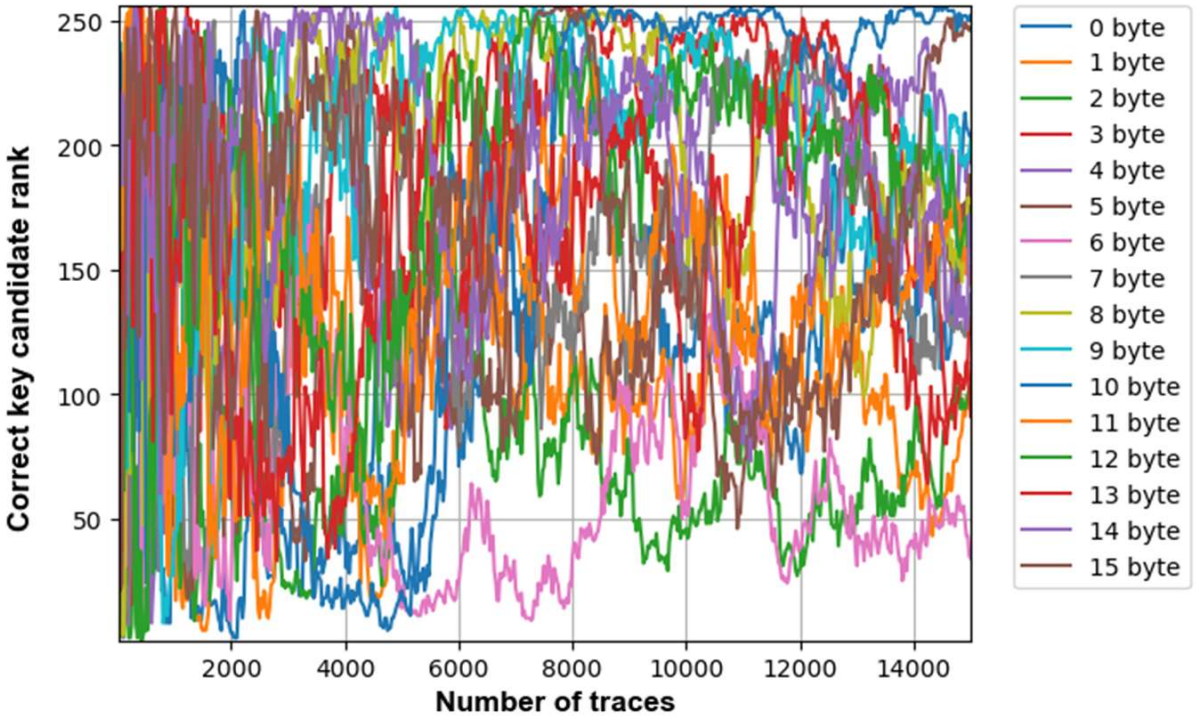


Figure 4.21: Rank analysis of the DOM implementations based on CPA (15000 power traces)

4.5.3. TBA on AES and DOM designs

In the TBA scenario, we recorded 15000 power traces and performed attack to both designs. Fig. 4.22 illustrates that the declining trend of PGE with the growing number of power traces, indicating the vulnerability of non-DOM AES design to TBA. Note that after approximately 1500 power traces, the PGE of the AES design begins to decrease, and as the number of power traces reaches 12000, the PGE falls below 90. In contrast, the PGE of the DOM design remains relatively stable at around 140.

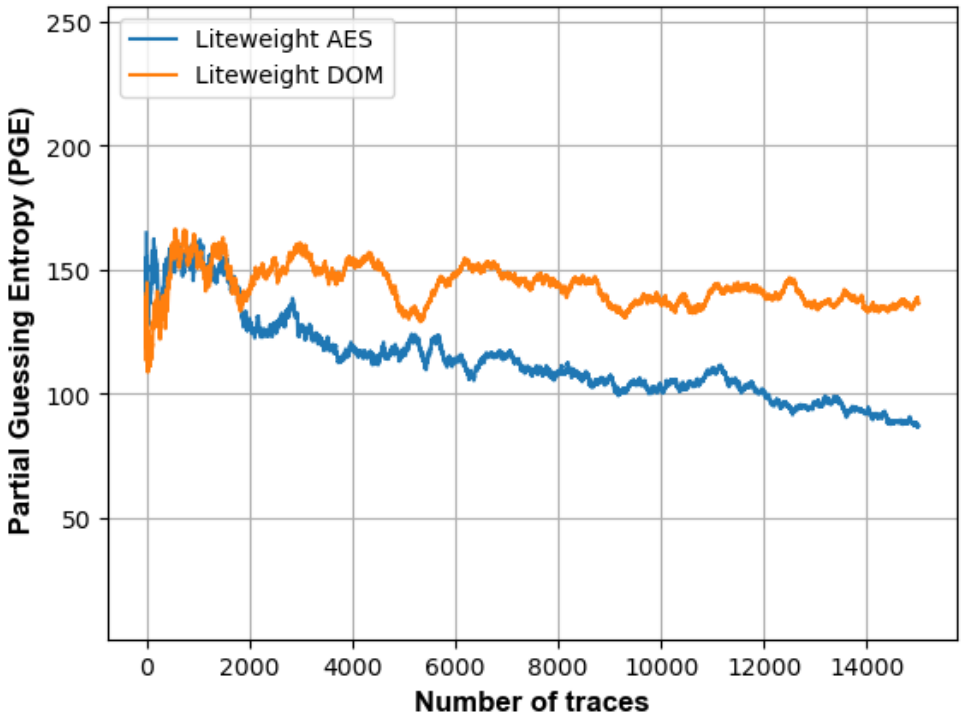


Figure 4.22: PGE of the proposed designs using TBA (15000 power traces)

A comprehensive investigation and rank analysis of each sub-key were carried out. In the AES design, the results indicate significant progress in identifying multiple subkeys, as evidenced by three subkeys having ranks near zero and nearly half of the subkeys having ranks below 50 (see Fig. 4.23). On the basis of this information, we made the assumption that an adversary with more advanced equipment and a high sampling rate measurement tool could be able to accurately guess the key. Conversely, in the DOM design, no subkey ranks approach 0, and only two subkeys rank fall below 50, shown in Fig. 4.24. Therefore, the DOM-version AES design is resilient to TBA.

We can conclude that non-DOM AES design could be safe enough for applications with a limited data size and minimum security requirement, as it requires thousands of traces for the attack to be carried out successfully. For applications that demand a high-level of security, DOM design is required.

4.6. Discussion

In our AES designs, we integrate both encryption and decryption processes rather than treating them as separate entities to efficiently allocate shared resources. Furthermore, we simply the SBOX module and sidestep redundant key expansion procedures during communication sessions. Our proposed 32-bit AES design outperforms the state-of-the-art by 1.69x and 1.27x in terms of performance per unit area and performance per unit power, respectively. These improvements can extend to 4.90x and 2.68x when considering the 128-bit design. Among our proposed AES designs, the 128-bit design achieves superior score in terms of both performance per unit area and performance per unit power.

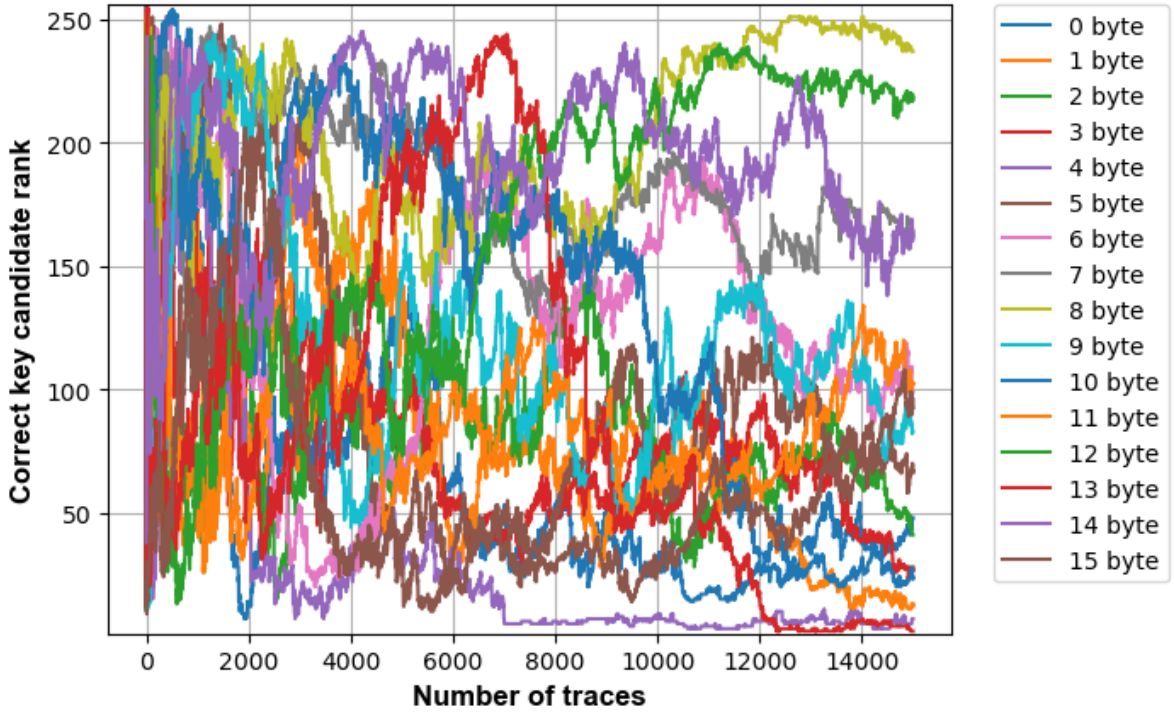


Figure 4.23: Rank analysis of the AES implementations based on TBA (15000 power traces)

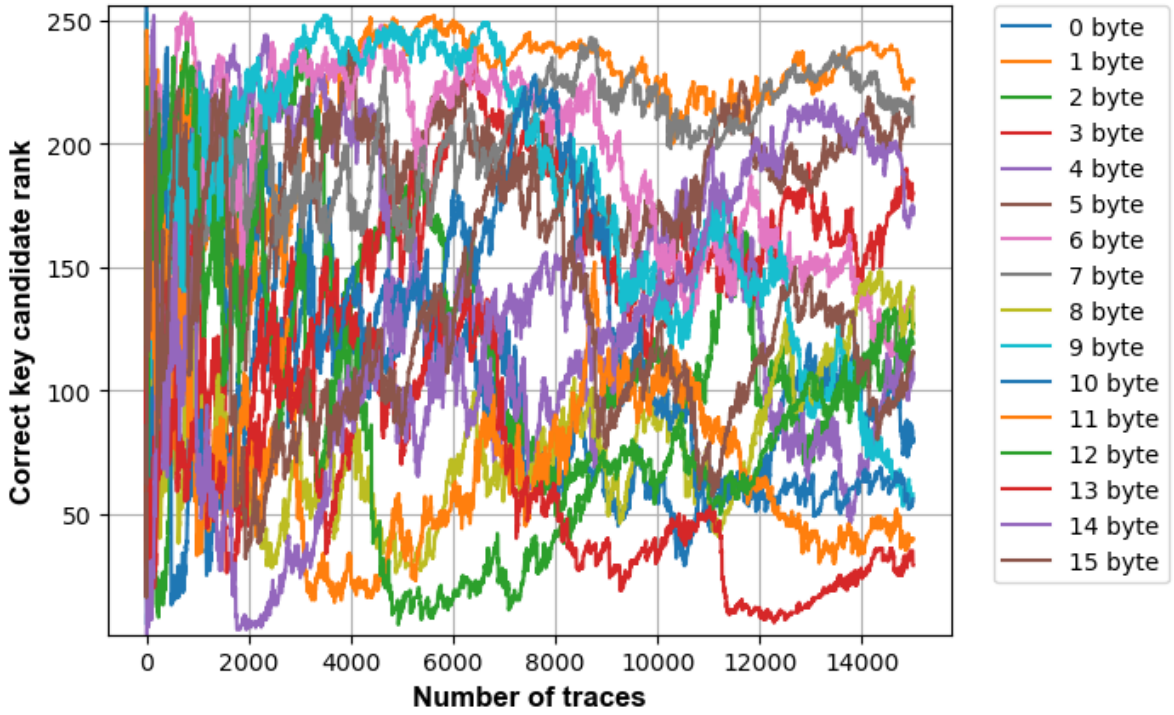


Figure 4.24: Rank analysis of the DOM implementations based on TBA (15000 power traces)

To enhance the resistance of our AES designs against side-channel attacks, we incorporate DOM technology into our designs. In the DOM design, we simplify the SBOX by utilizing ”simplified multipliers” and ”combined multipliers” proposed in our AES designs. Our experimental results showcase that our eight-stage and five-stage 1^{st}-order DOM SBOX designs yield area reductions of 9.9% and

6.9%, respectively. After integrating a simplified DOM architecture into our AES design, the DOM-version AES design demonstrates robust resilience against both CPA and TBA, even under 15000 power traces. Conversely, non-DOM AES design is resilient to CPA not to TBA. Although the area of DOM design is larger than original AES design, it is still worthwhile, particularly when catering to applications of heightened significance and criticality. Furthermore, it is important to highlight that the 64-bit DOM-version AES design outperforms in terms of performance per unit area, whereas the 32-bit DOM-version AES design excels in terms of performance per unit power.

5

Summary and Future Work

This chapter offers a comprehensive summary of all the achievements from the conducted research and outlines potential future research directions. Section 5.1 presents a summary of this thesis, organized by chapter. Section 5.2 discusses future research directions.

5.1. Summary

In this study, we presented a novel low-power and low-area AES accelerator, while maintaining high performance. A number of different optimization techniques, such as key expansion bypassing, resource sharing, and careful modules optimizations, were used to realize this. In Chapter 1, the introduction information and the structure of this paper were stated.

In Chapter 2, the background of the paper was introduced. Initially, the concept of Galois Field was presented, followed by a detailed explanation of how it can be effectively implemented in the AES SBOX. Besides, the process of AES encryption and decryption was stated, providing a clear understanding of how the algorithm operates to secure data. Next, the chapter discussed two types of Side-Channel Attacks (SCAs): Non-profile power attacks (e.g., DPA, CPA) and Profile power attacks (e.g., TBA). These attack types were explained in detail to underscore potential security risks in AES systems. Furthermore, we explore the development of countermeasures for SCAs. By comparing with previous techniques threshold implementation (TI), DOM offered the advantage of achieving same security level while occupying a smaller area, making it a promising choice for safeguarding AES implementations.

In Chapter 3, we presented detailed designs of both the lightweight AES and the DOM extension. In the AES design, we adopted an efficient approach to key expansion and store the key in registers only when it is changed. Otherwise, we bypassed the key expansion and executed the round functions to save cycles. To further optimize the design, we shared the resources (Shared SBOX, Shared ShiftRows, and Shared MixColumns) between encryption and decryption, and simplified some modules ($GF(2^4)$ Optimized Multiplier, Inverter, and Combined Multiplier) in Shared SBOX. To enhance the resilience of our AES designs against side-channel attacks, we incorporated DOM into our AES designs based on the work presented in [24]. We simplified the multipliers in DOM SBOX to make the design more area-efficient. Besides, our DOM design was developed to accommodate both encryption and decryption processes.

In Chapter 4, we initially presented the experimental results obtained from conducting pre-synthesis and post-synthesis simulations using NC-Sim. For the evaluations, the AES and DOM designs were implemented using TSMC 180nm technology and compared against state-of-the-art designs. According to the results, our proposed AES designs outperform the current state-of-the-art in terms of area, power, and performance. Subsequently, we developed an efficient version of a side channel countermeasure known as DOM using the same optimization techniques. According to the synthesis results, the DOM SBOX implementation achieves a smaller area compared to the original design [24]. We initiated the analysis using a dataset comprising 5000 power traces, followed by an extended dataset of 15000 power traces. The results indicated that the non-DOM AES is resilient against CPA but susceptible to TBA. However, the DOM-based AES design exhibited robust security against both CPA and TBA.

5.2. Future Work

In this section, we provide several recommendations aimed at further improving the topics discussed in the thesis.

5.2.1. AES design

When considering both encryption and decryption in AES design, it is worthwhile to initiate the process with key expansion. This approach not only significantly enhances performance but also retains optimal area utilization and power efficiency. This methodology can be readily extended to other AES designs, enabling the realization of a faster lightweight AES accelerator.

Furthermore, leveraging more advanced CMOS technology, particularly below 180nm, can yield significant benefits in terms of both area and power efficiency.

Last but not least, the design can be subjected to the tapeout process, which provides an opportunity to identify and rectify any potential issues or discrepancies that may have emerged during the design phase. This crucial step ensures that the final product is of high quality and meets performance specifications.

5.2.2. Security Extension

While implementing DOM on our proposed AES designs can protect them from SCA attacks, it results in a significant increase in the area and power consumption of our designs. In the future, we have the potential to further simplify DOM and integrate it into our AES designs. Furthermore, we can explore the synergies of combining DOM with other small-area countermeasures, creating a hybrid protection scheme that maintains a robust security level while minimizing the required area resources.

For security analysis, a more advanced equipment and a higher sampling rate measurement tool can be utilized to perform a more accurate attack to the designs. Furthermore, there exists room for further refinement of POI within the context of TBA, aiming to optimize its effectiveness. Additionally, the application of deep-learning techniques in the attack process can be utilized for achieving even more potent and precise outcomes.

A

Accepted conference paper

This appendix contains my accepted conference paper from 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-2023). The link to access the acceptance list for TrustCom-2023 is `https://hpcn.exeter.ac.uk/trustcom2023/accepted-paper-list.php`. Our paper is identified by the number TrustCom-468. In this paper, a new AES design is optimized for IoT in terms of area, power, and latency. Besides, an improved implementation of the DOM for superior protection against side channel attacks. The primary advantage of this work is the balanced approach to encryption and decryption, offering a design that's both efficient and secure, specifically tailored for the constraints of IoT devices.

Securing an Efficient Lightweight AES Accelerator

*Ruoyu Huang[†‡], *Abdullah Aljuffri[†], Said Hamdioui[†], Kezheng Ma[‡], Mottaqiallah Taouil[†]
[†] *Department of Computer Engineering, Delft University of Technology*, Delft, The Netherlands,
R.Huang-4@student.tudelft.nl, {A.A.M.Aljuffri, S.Hamdioui, M.Taouil}@tudelft.nl
[‡] *Silicon Integrated*, Eindhoven, The Netherlands, kezheng.ma@si-in.com
[*] *These authors contributed equally to this paper.*

***Abstract*—The Advanced Encryption Standard (AES) is generally regarded as one of the most popular cryptographic algorithms for ensuring data security. Typical lightweight implementations of the algorithm published in the literature focus on area and power optimization, while neglecting the performance. This paper presents a novel lightweight approach for the AES algorithm and considers both encryption and decryption. In terms of performance per unit area and performance per unit power, our 32-bit design outperforms the state-of-the-art by 1.69x and 1.27x, respectively. These improvements become even larger when implementing higher data-path designs, such as 64-bit or 128-bit designs. To enhance its resilience against side-channel attacks, we modified our design by adopting and further improving on the most recent countermeasure, i.e., Domain-Oriented Masking (DOM). The results demonstrate that our five-stage and eight-stage 1^{st}-order DOM SBOX designs achieve a reduction in area of 9.9% and 6.9% compared to the original proposed design, respectively.**

***Index Terms*—Advanced Encryption Standard, Domain Oriented Masking, Lightweight Accelerator, Internet of Things, Side Channel Attacks**

I. Introduction

According to a study that was recently released in the World Economic Forum [1], malicious attacks were launched against 1.5 billion Internet of Things devices during the first half of 2021. The number of instances of data breaches increased by 15.1% as compared to the previous year. As a consequence of this, the security of the Internet of Things (IoT) has become of the utmost importance and can no longer be treated as an afterthought. This is particularly true when considering the anticipated yearly increase in adoption of those devices, which is expected to reach 43 billion in the year 2023 [2]. With respect to data protection, the Advanced Encryption Standard (AES) [3] is one of the algorithms that is most generally recognized and utilized today. It was first presented to the public by the National Institute of Standards and Technology (NIST) in the year 2001. AES is currently the primary encryption method for many applications, including cloud computing [4] and health care [5]. Traditional implementations of the AES use pipelining techniques in order to achieve a high throughput [6–8]. However, as these implementations consume a significant amount of memory and power, it is unfeasible to use them in IoT devices that are limited by area and battery capacity [9]. Therefore, implementations of AES should be area and power efficient, while simultaneously minimizing the impact on throughput to the greatest degree possible.

To overcome these challenges, several lightweight AES accelerators have been proposed, which reduce area and power consumption by shortening the data-path from the conventional 128-bit to 8-bit [10–13] i.e., reducing the number of SBOXes from 16 to 1. Several researchers [14–16] further pursued the reduction of area requirements at the expense of additional cycles. In general, 8-bit data-path designs significantly effect the throughput as at least 160 cycles are required per encryption. More recently, 32-bit designs have been proposed. Such designs achieve a better trade-off between performance and energy efficiency [17]. Most of the above articles focused or reported only on the encryption module. Davis and John [18] considered actually both modules and proposed optimizations across them. However, as will be shown in this paper later, the shared modules have not been fully optimized. Additionally, their reported area measurements only consider the encryption module. To fairly evaluate the designs and realize the best power/latency/area trade-off, it is important to provide the overhead of both the encryption and decryption modules.

This paper presents a novel low-area, low-power and low-latency AES hardware accelerator. Our method takes advantage of the fact that the key remains unchanged throughout a communication session, eliminating the need for repeated execution of the key expansion module. Additionally, we integrate an improved version of the Domain-Oriented Masking (DOM) which is one of the most advanced countermeasures against side channel attacks (SCAs). Our DOM-based AES design is more area-efficient in comparison to the original DOM design. In summary, the contributions of this paper are:

- A proposal of an novel low-area, low-power, and low-latency design of the AES algorithm which is suitable for IoT applications.
- A proposal of a side channel resilient version using an improved DOM implementation.
- Evaluation of the energy consumption, area overhead and performance of the proposed design.

This paper is organized as follows. Section II introduces the background on AES, SCAs and DOM. Section III introduces our proposed AES designs and its DOM security extension. A comparison of the implementation results are provided in Section IV. Finally, Section V concludes this paper.

II. Background

This section provides a brief background on the AES algorithm, SCAs, and DOM countermeasure.

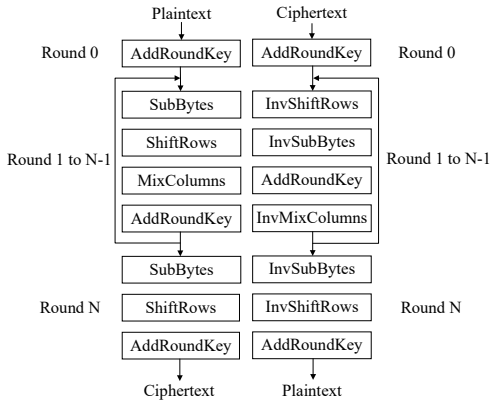


Fig. 1. AES Encryption & Decryption Flow Diagram

A. Advanced Encryption Standard (AES)

AES is a symmetric cryptographic algorithm that is used in the cyber world for encrypting and decrypting data to protect them from cyberattacks. It has a fixed data block size of 128 bits, and a key length of 128, 192, or 256 bits. The key length determines the number of rounds required: 10, 12, or 14 rounds for 128-bit, 192-bit, or 256-bit key lengths, respectively. The 128-bit data block is divided into 16 bytes, which are mapped to a 4×4 array referred to as the State array. The diagram of AES encryption and decryption flow is presented in Fig. 1. Each round of encryption includes four primary modules: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*, except for Round 0 and the last round (see Fig. 1).

AddRoundKey module involves bit-wise XOR operations of the round key and State array. *SubBytes* module is the only nonlinear module in the AES and plays a crucial role in defending against linear crypt-analysis [19]. When performing *SubBytes* module, each byte in the state array is substituted with another byte using SBOX. The SBOX is generated using a combination of a multiplicative inverse in Galois Field GF(2^8) and an affine transformation [20]. *ShiftRows* module is a transformation that cyclically shifts the second, third, and fourth rows of the State array by one, two, and three bytes to the left, respectively, while leaving the first row unchanged. The *InvShiftRows* module is computed by performing the corresponding rotations to the right. The *MixColumns* and *InvMixColumns* module perform a modular polynomial multiplication in Galois Field GF(2^8) on each column of the State array.

B. Side channel attacks (SCAs)

Side channel attacks take advantage of vulnerabilities by analyzing unintentional physical information that is disclosed during the device's normal execution [21]. Side channel attacks are capable of extracting confidential data, including cryptographic keys, from a system by analyzing the information that is unintentionally leaked. There are various physical information of a system, such as its power consumption, electromagnetic radiation, timing, and acoustic emissions, which can offer insights into its internal functioning [22, 23]. Among them, power consumption is one of the most widely abused, primarily due to its high success rate and straightforward execution. In side channel power attacks, the adversary performs a statistical analysis on power consumption measurements obtained at an intermediate target, such as an SBOX operation. These measurements will then be connected to a leakage model [24], to obtain the secret information.

C. Domain Oriented Masking (DOM)

To protect implementations of AES against SCA attacks, a widely used technique involves randomizing all intermediate results, known as masking schemes. However, in 2005, Stefan et al. [25] discovered that circuits of masked gates are vulnerable to classical DPA attacks due to glitches that occur within the circuits. To overcome this issue, the threshold implementation (TI) was proposed by Nikova et al. [26] to counter SCA attacks and glitches. However, implementing TI requires a high number of shares, which can be costly. Besides, redesigning the non-linear components of the circuit is needed to meet the requirements for higher-order TI, leading to a significant increase in design effort. To provide a lower cost of design, the DOM technique was proposed [19]. Compared with TI, the number of required shares in DOM is reduced without sacrificing the security level. DOM implementation in hardware has demonstrated its resilience to SCAs up to at least 15^{th} order attacks [19].

In DOM, each variable is represented by d + 1 shares to protect the circuit from d^{th}-order SCAs. The fundamental principle of the DOM approach is to maintain the independence of shares within each domain. This is simple for the linear modules of AES, i.e., *MixColumns*, *AddRoundKey*, and *ShiftRows*, as there are no cross-domain calculation required among the different shares. However, the *SubBytes* operation, which is the only non-linear module in AES requires a substantial amount of multiplications. When performing these multiplications, shares may cross domain borders, making it necessary to use fresh random numbers to maintain their statistical independence.

Hannes and his team [19] proposed two multipliers named DOM-indep and DOM-dep. First, the DOM-indep multiplier requires that the inputs from different domains are uniformly random and independent from each other. Take the 1^{st}-order DOM-indep multiplier as an example which consists of two share domains. Assume that the two independent inputs of multiplier are x and y, where x = $A_x \oplus B_x$, y = $A_y \oplus B_y$ ("$\oplus$" represents the xor operation). Furthermore, Z_0 denotes the fresh random number. The output q can be expressed as (1).

$$\begin{aligned} q &= x * y \\ &= (A_x \oplus B_x)(A_y \oplus B_y) \\ &= A_x A_y \oplus A_x B_y \oplus B_x A_y \oplus B_x B_y \\ &= A_x A_y \oplus (A_x B_y \oplus Z_0) \oplus (B_x A_y \oplus Z_0) \oplus B_x B_y \end{aligned} \tag{1}$$

Hannes et al. [19] categorized the product terms into two parts: inner-domain terms ($A_x B_x$ and $A_y B_y$) and cross-domain ($A_x B_y$ and $A_y B_x$). The inner-domain product terms do not reveal critical information since they originate from the same domain. In contrast, cross-domain calculations can

only be performed on independent inputs to prevent leakage of information for either x or y. A fresh random Z_0 value is used to remain the statistical independence of the DOM-indep multiplier outputs from other values. Additionally, flip-flops are employed to prevent glitches from propagating through this block. Secondly, Hannes et al. [19] proposed the DOM-dep multiplier based on the structure of the DOM-indep multiplier, which does not require the independence of inputs. However, it requires more fresh random values and additional area. The 1^{st}-order DOM-dep and DOM-indep multipliers can be easily extended to higher order by using more shares and fresh random values.

III. Lightweight AES and DOM Extension

This section presents our proposed implementation approach for AES and its protected version using DOM. We start by motivating our approach, followed by a detailed explanation of its design and implementation.

A. Motivation

Previous research primarily focused on implementing AES on either an 8-bit [10–16] or 32-bit [17, 18] data-path to reduce area and energy consumption. However, these studies only report the area of the encryption module and neglect the decryption part. In reality, the eleven 128-bit registers required for the key expansion in the decryption module contribute significantly to the overall core area. In the decryption module, all round keys must be computed first before the decryption can start. Shortening the data-path from 128-bit to a lower-bit width has a much lower improvement on the area when the decryption module is not ignored. Therefore, in our design we focus on different data-paths in the presence of the decryption unit and compare their performance in terms of throughput, area, and power. Secondly, in actual applications, keys do not change frequently. Hence, we perform the key expansion once and store the results in the registers. As long as the key remains the same, we can skip the key expansion step, resulting in a significant power and latency reduction. In addition, we reorder the sequences of *Addroundkey* and *MixColumns* in the round function which results in further area and performance improvements.

B. Design and Implementation of Proposed Lightweight AES

Our proposed AES designs verify whether the key changes at the start of every encryption/decryption execution. In case a key change is detected, we perform the key expansion module and leave the keys inside the key registers. Otherwise, we directly execute the round modules (i.e., *AddRoundKey*, *SubBytes*, *MixColumns*, and *ShiftRows*). This reduces the execution time of the decryption part by eleven cycles. To further optimize the design area, resource sharing is employed. Initially, we limit the number of registers to store the state to a single 128-bit register that is shared in all the round modules of both encryption and decryption. Next, we combine the encryption and decryption modules to decrease the overall area. The proposed scheme is depicted in Fig. 2, where *cnt* represents the round index and *key[cnt]* denotes the key that needs to be XORed with the State array. The boxes containing the word "shared" represent blocks that are shared between the encryption and decryption. An in-depth explanations of the shared modules will be provided next, including *Shared SBOX, Shared ShiftRows*, and *Shared MixColumns*.

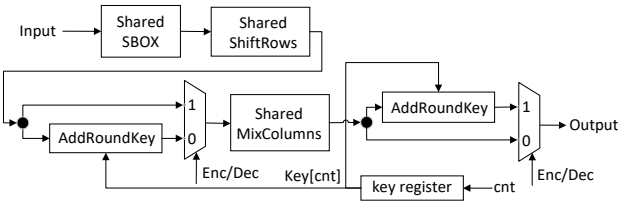


Fig. 2. Proposed Round Function for AES Encryption and Decryption

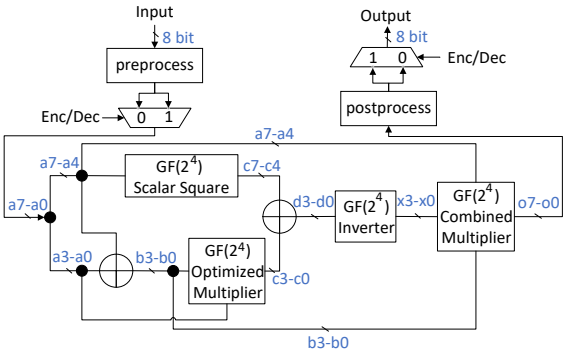


Fig. 3. Proposed Shared SBOX

*1) **Shared SBOX**:* Akashi et al. [27] proposed a new composite field to optimize the structure of the SBOX, resulting in a significant reduction of the area compared to using a Look-up table (LUT). Thereafter, several researchers [20, 28, 29] optimized the SBOX based on the structure provided in [27]. These papers used an SBOX which is shared by both the encryption and decryption modules to reduce area. To the best of our knowledge, the SBOX design described in [29] has the lowest area. Compared with previous designs, they shared resources in three modules: preprocess, postprocess, and scalar square. The preprocess module performs the isomorphic mapping and inverse affine transformation for the decryption and the isomorphic mapping only for the encryption. The postprocess module executes the affine transformation and the inverse isomorphic mapping for the encryption and inverse isomorphic mapping only for the decryption. The scalar square performs a square and multiplication with constant $\lambda = \{1, 1, 0, 0\}$, which leads to three XOR reductions [20]. Our new SBOX is based on the SBOX proposed in [29]; it is shown in Fig. 3. It contains an optimized multiplier and a modified inverter. In addition, it combines the operations of the last two multipliers proposed in [27]. Each optimization is described next into more details.

- $GF(2^4)$ **Optimized multiplier:** Our optimized $GF(2^4)$ multiplier is based on the work in [29]. That multiplier consists of 18 XOR and 12 AND gates and its critical path consists of 4 XOR and 1 AND gate. We simplified

the $GF(2^4)$ multiplier based on Equation (2), where $\{a_3, a_2, a_1, a_0\}$ and $\{b_3, b_2, b_1, b_0\}$ denote the two 4-bit inputs (see also left bottom of Fig. 3), $\{c_3, c_2, c_1, c_0\}$ denote the 4-bit output, and $\{m_4, m_3, m_2, m_1, m_0\}$ are intermediate variables defined as: $m_4 = m_0 \oplus m_1$, $m_3 = a_0 \oplus a_1$, $m_2 = a_3 \oplus a_2$, $m_1 = a_2 \oplus a_0$, and $m_0 = a_3 \oplus a_1$. Although our $GF(2^4)$ optimized multiplier utilizes 4 more AND gates compared to [29], it requires 1 XOR gate less and more importantly has a shorter critical path (1 AND gate and 3 XOR gates). Surprisingly, after synthesis it turns out that the area of this implementation is also better after synthesis. We believe that compiler is able to extract more common resources with this implementation.

$$
\begin{aligned}
c_3 &= [(a_3 \oplus a_1)\&(b_3 \oplus b_1) \oplus (a_3 \oplus a_1)\&(b_2 \oplus b_0) \oplus (a_2 \oplus a_0)\&(b_3 \oplus b_1)] \oplus [(a_1\&b_1) \oplus (a_1\&b_0) \oplus (a_0\&b_1)] \\
&= (b_0\&a_3) \oplus (b_1\&(a_2 \oplus a_3)) \oplus (b_2\&(a_1 \oplus a_3)) \oplus (b_3\&(a_0 \oplus a_1 \oplus a_2 \oplus a_3)) \\
&= (b_0\&a_3) \oplus (b_1\&m_2) \oplus (b_2\&m_0) \oplus (b_3\&m_4); \\
c_2 &= [(a_3 \oplus a_1)\&(b_3 \oplus b_1) \oplus (a_2 \oplus a_0)\&(b_2 \oplus b_0)] \oplus (a_1\&b_1) \oplus (a_0\&b_0) \\
&= (b_0\&a_2) \oplus (b_1\&a_3) \oplus (b_2\&(a_0 \oplus a_2)) \oplus (b_3\&(a_1 \oplus a_3)) \\
&= (b_0\&a_2) \oplus (b_1\&a_3) \oplus (b_2\&m_1) \oplus (b_3\&m_0); \\
c_1 &= [(a_3\&b_3) \oplus (a_3\&b_2) \oplus (a_2\&b_3)] \oplus [(a_3\&b_3) \oplus (a_2\&b_2)] \oplus [(a_1\&b_1) \oplus (a_1\&b_0) \oplus (a_0\&b_1)] \\
&= (b_0\&a_1) \oplus (b_1\&(a_0 \oplus a_1)) \oplus (b_2\&(a_2 \oplus a_3)) \oplus (b_3\&a_2) \\
&= (b_0\&a_1) \oplus (b_1\&m_3) \oplus (b_2\&m_2) \oplus (b_3\&a_2); \\
c_0 &= [(a_3\&b_3) \oplus (a_3\&b_2) \oplus (a_2\&b_3)] \oplus [(a_1\&b_1) \oplus (a_0\&b_0)] \\
&= (b_0\&a_0) \oplus (b_1\&a_1) \oplus (b_2\&a_3) \oplus (b_3\&(a_2 \oplus a_3)) \\
&= (b_0\&a_0) \oplus (b_1\&a_1) \oplus (b_2\&a_3) \oplus (b_3\&m_2).
\end{aligned}
\tag{2}
$$

- $GF(2^4)$ **Inverter:** To decrease the area of the inverter and make the design easier to secure (by performing less non-linear operations), we further optimized the inverter based on the structure that proposed in [20]. The improved design is shown in Fig. 4. The $GF(2^2)$ *Scalar Square* performs a $GF(2^2)$ square operation and a scalar multiplication with the constant $\varphi = \{1, 0\}$. Equation (3) illustrates the *Scalar Square* calculation of [20] (left) and our combined design (right), where d_3, d_2 are the inputs of *Square* module, e_3, e_2 denote intermediate results between *Square* and *Scalar* module, and f_3, f_2 represent the outputs of *Scalar* module (see Fig. 4). As can be seen, our design requires 2 XOR operations less.

$$
[20]\begin{cases} e_3 = d_3 \\ e_2 = d_3 \oplus d_2 \\ f_3 = e_3 \oplus e_2 \\ f_2 = e_3 \end{cases}
\qquad
Combined\begin{cases} f_3 = d_2 \\ f_2 = d_3 \end{cases}
\tag{3}
$$

The last step of the inverter consist of two $GF(2^2)$ multipliers. Equation (4) shows the design proposed in [20]. Our optimizations are provided after the second "=" sign by factoring out $X = h_1 \oplus h_0$ commonly between

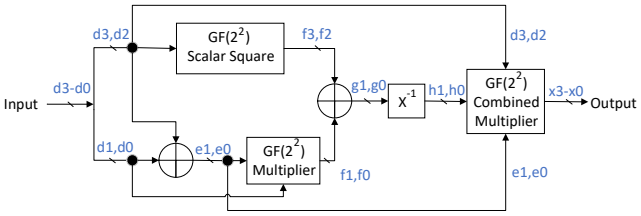


Fig. 4. Proposed $GF(2^4)$ Inverter

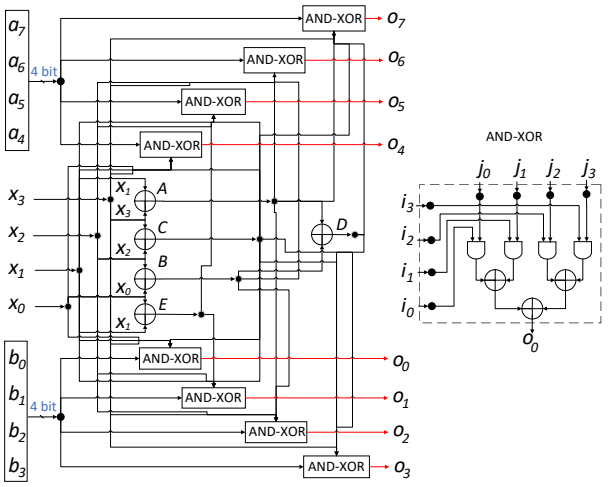


Fig. 5. $GF(2^4)$ Combined Multiplier

both multiplications. Our combined $GF(2^2)$ multiplier achieves a reduction of 1 XOR gate and 2 AND gates, compared with the design in [20].

$$
\begin{aligned}
x_3 &= (d_3\&h_1) \oplus (d_3\&h_0) \oplus (d_2\&h_1) = (d_3\&X) \oplus (d_2\&h_1); \\
x_2 &= (d_3\&h_1) \oplus (d_2\&h_0); \\
x_1 &= (e_1\&h_1) \oplus (e_1\&h_0) \oplus (f_0\&h_1) = (e_1\&X) \oplus (f_0\&h_1); \\
x_0 &= (e_1\&h_1) \oplus (e_0\&h_0).
\end{aligned}
\tag{4}
$$

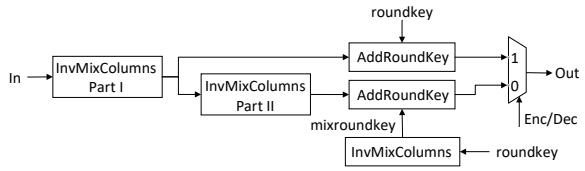
- $GF(2^4)$ **Combined Multiplier:** The last two multipliers used in the SBOX presented in [29] were treated as two separate multipliers. However, Ahmad [30] proposed that these two multipliers can be merged together, resulting a significant reduction in area as can be seen at the most right part in Fig. 3. However, it is not clear from the paper how this shared multiplier works. For clarity, we combined the multipliers ourselves and provided a detailed structure of it in Fig. 5.

*2) **Shared ShiftRows**:* Davis and John observed that the first and third shift operations in *ShiftRows* and *InvShiftRows* can be shared [18], as both produce the same results for the decryption and encryption. This can be seen in Fig. 6. However, the other two rows (i.e., row two and four) have different behavior and multiplexers are needed to select between them.

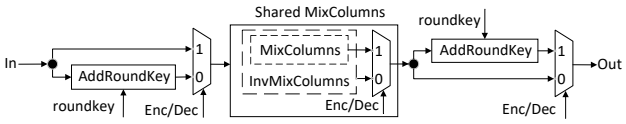
$$
\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{bmatrix}
\qquad
\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,3} & S_{1,0} & S_{1,1} & S_{1,2} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,0} \end{bmatrix}
$$

(a) ShiftRows (b) InvShiftRows

Fig. 6. Shift Transformation of ShiftRows and InvShiftRows



(a) MixColumns/InvMixColums [20]



(b) Proposed Shared MixColumns

Fig. 7. Diagram of MixColumns [20] and Proposed Shared Mixcolumns

3) ***Shared MixColumns****:* We optimize the *Shared MixColumns* based on the proposed design in paper [20], which shares resources between *MixColumns* and *InvMixColumns*. The design in [20] however requires an additional *InvMixColumns* calculation to rectify the roundkey (see Fig. 7(a)). In contrast, Fig. 7(b) shows that our proposed *Shared MixColumns* combines *MixColumns* and *InvMixColumns*, and reorganizes the sequence of *Addroundkey* and *Shared MixColumns* to avoid performing this additional *InvMixColumns* calculation. This lead to further area improvements.

C. Design and Implementation of Proposed Lightweight DOM

DOM was proposed in [19] to protect AES implementations against SCAs. The authors introduced two types of SBOXes: a five-stage SBOX and an eight-stage SBOX. The five-stage SBOX represents an optimized version of the eight-stage SBOX, resulting in a savings of three cycles per round. Hence, overall it is 33 cycles (3 cycles $\times$ 11 rounds) faster, which is a significant performance improvement. This improvement comes with only a minor increase in the overall area, from 2.6k Gates to 2.8k Gates, as documented in [19]. Therefore, we chose to start from the five-stage SBOX and integrate it into our optimized design. Note that a 1^{st}-order DOM can be easily scaled into a higher order DOM without redesigning components [19]. Therefore, without loss of generality, we focus on the optimization of the 1^{st}-order DOM. As our proposed low-area design considers both encryption and decryption with shared resources (see Fig. 2), the lightweight DOM AES was implemented in the same manner, diverging from the original design that concentrated only on encryption [19]. Fig. 8 shows the main part of our lightweight DOM design, where A_{in}, B_{in} represent the input shares, and A_{out}, B_{out} correspond to the output shares.

As discussed in the background, the independence of d+1 shares within linear modules can be ensured by employing d+1 identical modules. However, the non-linear SBOX module requires to be carefully designed. Our design is based on the lightweight DOM SBOX proposed in [19]. In comparison to that design, our approach shares the resources between encryption and decryption parts using the preprocess and postprocess functions. In addition, we optimize the DOM-

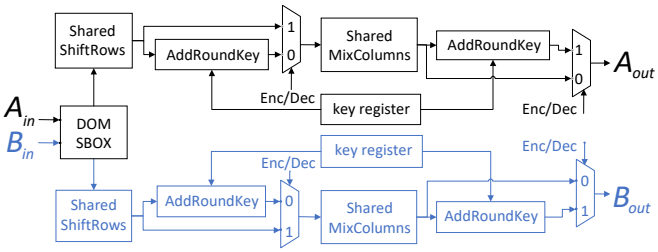


Fig. 8. Proposed Design for DOM Encryption and Decryption

indep and DOM-dep multipliers based on our simplified and shared multipliers to further reduce the area. Fig. 9 depicts our design of the 1^{st}-order five stages lightweight DOM SBOX, where A_{sin} and B_{sin} denote the input shares, and A_{sout} and B_{sout} denote the output shares. In the figure, $Z_0, Z_1, ..., Z_6$ and $A_{z0}, B_{z0}, ..., A_{z3}, B_{z3}$ are fresh random values of the simplified DOM-indep and DOM-dep multipliers, respectively. The flip-flops with dotted boxes are optional registers that are only necessary in pipelining scenarios. For example, when the data-path is less than 128 bits, the SBOX needs to be reused multiple times within one round, causing the input to change before the round is completed. In this case, the dotted flip-flops are necessary to ensure the design's functional correctness.

Fig. 9 also highlights the parts that we improved in red, i.e., the DOM multipliers. Compared to the original DOM multipliers (see [19]), we replaced the normal multipliers with our simplified multipliers (see (2)) and shared multipliers (see Fig. 5), resulting in a reduction in power and area. In addition, multiplexers are used to select between inputs for the encryption and decryption units.

Fig. 10 illustrates our changes made to the 1^{st}-order Dom-dep multiplier [19]; we refer to it as simplified DOM-dep multiplier. In the figure, A_a, B_a, A_b, B_b are the inputs, while A_q and B_q correspond to the outputs. A_z, B_z, and Z_0 are random numbers that used to ensure the independence of shares. In contrast to the design proposed in [19], our proposed simplified DOM-dep multiplier utilizes a simplified version of the DOM-indep multiplier and merges the right two multipliers, resulting in significant area reduction. The simplified DOM-indep multiplier is based on the simplified multiplier shown in Equation (2). Equation (5) illustrates the expression of our DOM-dep multiplier, where M=$(A_b \oplus B_b) \oplus (A_z \oplus B_z)$. We denote that a= $A_a \oplus B_a$, b= $A_b \oplus B_b$, z= $A_z \oplus B_z$, and q= $A_q \oplus B_q$. In the equation, A_{qi} and B_{qi} are the outputs of simplified DOM-indep multipliers. The combined multiplier (see Fig. 5) is utilized for the calculation of $(A_a * M \oplus B_a * M)$ to further reduce area.

$$
\begin{aligned}
& a * b \\
&= a * (b \oplus z) \oplus a * z \\
&= (A_a \oplus B_a)\,(A_b \oplus B_b \oplus A_z \oplus B_z) \oplus (A_a \oplus B_a)\,(A_z \oplus B_z) \\
&= (A_a * M \oplus B_a * M) \oplus (A_{qi} \oplus B_{qi}) \\
&= (A_a * M \oplus A_{qi}) \oplus (B_a * M \oplus B_{qi}) \\
&= A_q \oplus B_q = q
\end{aligned}
\tag{5}
$$

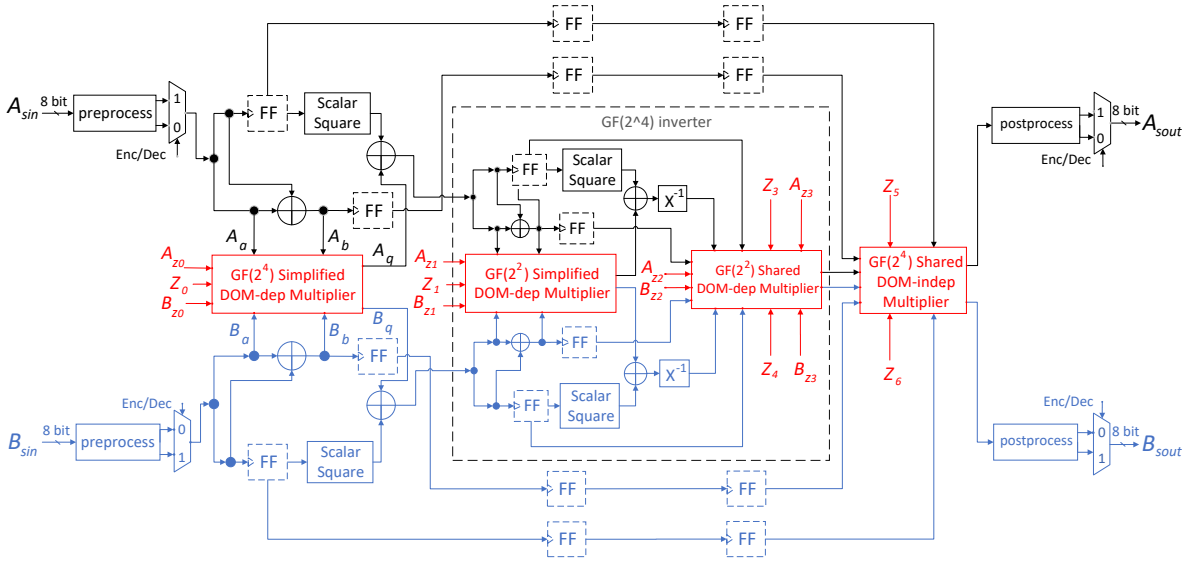
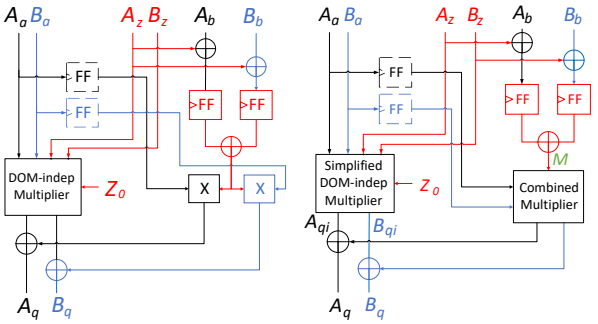


Fig. 9. Structure of the 1^{st}-order five-stage DOM SBOX Module (modified based on [19])



(a) DOM-dep [19] (b) Simplified DOM-dep

Fig. 10. DOM-dep Multiplier [19] and Our Simplified DOM-dep Multiplier

The shared DOM-indep/DOM-dep multiplier modules in Fig. 9 are implemented with the simpilified DOM-indep/DOM-dep multipliers (see Fig. 10(b)). We share the common resources between these two multipliers to further reduce the area.

IV. Experimental Results

This section presents the experiments setup and performed experiments, and evaluates the results.

A. Setup

For the majority of IoT applications, the maximum payload size for each packet is between 1600 bytes (e.g., NarrowBand-IoT [31]) and 256 megabytes (e.g., Message Queue Telemetry Transport(MQTT) [32]). Taking the lower limit into consideration, we assume that the key will stay the same during the communication session of at least one hundred encrypting/decrypting operations. For that reason, we assumed a fixed key for 100 encryption and decryption operations.

To compare with the start-of-the-art, we reimplemented the state-of-the-art AES design proposed by Davis and Jones [18] and compared it with ours. All designs are synthesized using TSMC CMOS 180 nm technology. The total area and power consumption of each design are evaluated Using Synopsys Design and Power Compiler. We took the same approach with respect to the DOM design originally proposed in [19].

TABLE I
AES Performance Analysis at 50MHz

Design	Data-path	Freq. (MHz)	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
[18]	32	50	174156	1	11100	1
Proposed AES	32	50	139415	0.8	8211	0.74
	64	50	151677	0.87	4211	0.38
	128	50	178674	1.03	2211	0.2

B. AES Performance Evaluation

In this section we compare our AES SBOX design proposed in Section III.B with the SBOX proposed in [29]. Note that this paper limited itself to only an SBOX implementation. The synthesis results show that the area of our proposed SBOX design is 8.2% lower than the design in [29]. The actual area numbers are 2558 vs 2788 μm^2, respectively.

We compare our complete non-DOM AES design proposed in Section III.B with the design proposed in [18].Tables I and II show the results for both designs synthesized at 50 MHz and maximum frequency, respectively. . From the first table, we can see that our 32-bit data-path implementation needs 20% less area than the design proposed in in [18]. Actually, our 128-bit data-path design is comparable in size to their 32-bit data-path design, while being 5x faster. When we look at Table II we see similar trends. Comparing both 32-bit data-path designs, our design has 14% less area, needs 26% less cycles and can run at 18% higher frequency.

Figs. 11 and 12 depict the performance per area and performance per power of our proposed AES design and the state-of-the-art design in [18]. The figures show that among

TABLE II
AES PERFORMANCE ANALYSIS AT THE MAXIMUM FREQUENCY

Design	Data-path	Freq. (MHz)	Freq. Ratio	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
[18]	32	103.1	1	196857	1	11100	1
Proposed AES	32	121.9	1.18	169794	0.86	8211	0.74
	64	117.6	1.14	195355	0.99	4211	0.38
	128	112.4	1.09	236553	1.2	2211	0.2

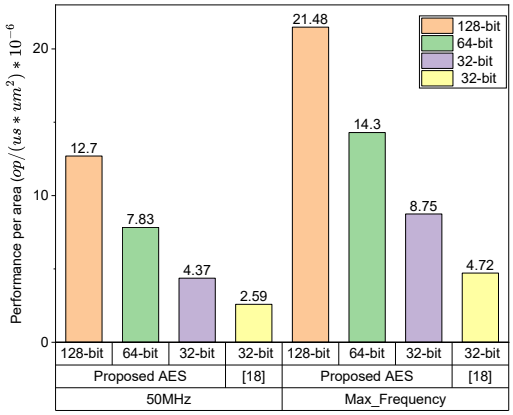


Fig. 11. Performance per Area Comparison vs AES Design in [18]

our proposed designs, the 128-bit design achieves the highest score in terms of performance per area and performance per power. Our proposed 128-bit, 64-bit, and 32-bit designs surpass the state-of-the-art [18] in terms of performance per area by a factor of 4.90x, 3.02x, and 1.69x, respectively, when operating at 50MHz and by a factor of 4.55x, 3.03x, and 1.85x, respectively, when operating at the maximum frequency. They also outperform the state-of-the-art design in terms of performance per power by a factor of 2.68x, 1.70x, and 1.27x, respectively. Note that in Fig. 12, only the performance per power for the 50 MHz implementation is displayed, as there were minimal differences observed when compared to the designs operating at their maximum frequencies.

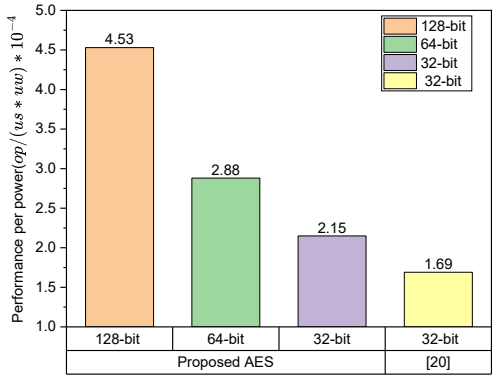


Fig. 12. Performance per Power Comparison vs AES Design in [18]

TABLE III
AREA COMPARISON OF DOM SBOX

Design	SBOX Type	Area (μm^2)	Area Ratio
[19] DOM SBOX	eight-stage	19682	1
	five-stage	21196	1.077
Proposed DOM SBOX	eight-stage	17735	0.901
	five-stage	19741	1.003

TABLE IV
DOM PERFORMANCE ANALYSIS AT 50MHZ

Data-path	Frequency (MHz)	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
128-bit	50	615172	1	10251	1
64-bit	50	445269	0.72	12251	1.2
32-bit	50	361582	0.59	16251	1.59

C. DOM Performance Evaluation

Table III shows a comparison of the area between our proposed 1^{st}-order DOM SBOX and the original 1^{st}-order SBOX proposed in [19]. Compared to their design, our eight-stage and five-stage 1^{st}-order DOM SBOX designs achieve an area reduction of 9.9% and 6.9%, respectively.

Although the 128-bit data-path design has the best performance, we have implemented also 64-bit and 32-bit data-path versions. Unfortunately, the authors in [19] focused only on the SBOX and have not evaluated the complete AES design. Nevertheless, to comprehensively assess the influence of these designs on overall performance, our proposed DOM SBOX has been incorporated into all AES configurations, encompassing the 128-bit, 64-bit, and 32-bit versions. Tables IV and V present their area and latency results for an operating frequency of 50 MHz and their maximum operating frequency, respectively. As we mentioned in the Section III-C, we chose the five-stage SBOX in our designs because it offers a substantial performance improvement with only minor sacrifices in terms of area when compared to the eight-stage SBOX. Consequently, the key expansion process takes 51 (5*10+1) cycles, while the encryption and decryption operations in the 128-bit, 64-bit, and 32-bit data-path designs require 51 (5*10+1), 61 (6*10+1), and 81 (8*10+1) cycles, respectively. As a result, it takes 10251 (51*2*100+51), 12251 (61*2*100+51), and 16251 (81*2*100+51) cycles for these designs to perform 100 encryption/decryption operations. Tables IV and V demonstrate that the 32-bit design exhibits a better area, whereas the 128-bit design has a higher performance.

The DOM SBOX contains a great number of registers and operation, resulting in a high power consumption and area. However, lower data-path designs can significantly reduce area and power consumption by utilizing fewer DOM SBOXes. Fig. 13 shows the performance per area comparison of our

TABLE V
DOM PERFORMANCE ANALYSIS AT THE MAXIMUM FREQUENCY

Data-path	Frequency (MHz)	Frequency ratio	Area (μm^2)	Area Ratio	Cycle	Cycle Ratio
128-bit	188.7	1.79	698780	1	10251	1
64-bit	190.8	1.81	522844	0.75	12251	1.2
32-bit	192.3	1.83	446528	0.64	16251	1.59

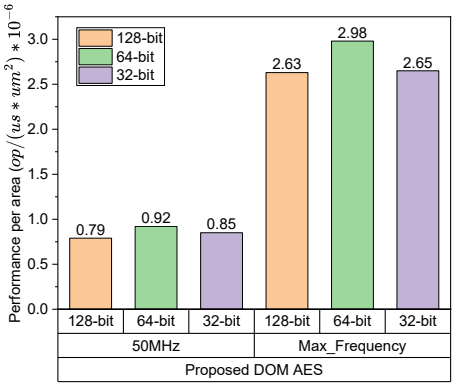


Fig. 13. Performance per Area Analysis of our 1^{st}-Order DOM AES designs

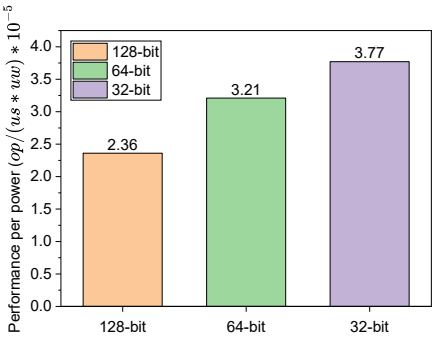


Fig. 14. Performance per Power Analysis of our 1^{st}-Order DOM AES Designs

proposed DOM designs. According to the figure, the 64-bit design performs better at both 50 MHz and the maximum frequency. Fig. 14 illustrates a comparison of the performance per power for our proposed DOM designs, where the 32-bit design outperforms the others.

V. Conclusion

In this paper we presented a novel low-power and low-area AES accelerator with high performance. A number of different optimization techniques, such as key expansion bypassing, resource sharing, and careful modules optimizations have been proposed and implemented. According to the results, our designs outperform the current state-of-the-art in terms of area, power, and performance. Subsequently, we developed an efficient version of the DOM side channel countermeasure using the same optimization techniques. The results demonstrated that also our DOM SBOX achieves a lower area than the originally proposed design.

References

[1] W. E. Forum, "Connected devices need better governance: Here's how to achieve it," https://www.weforum.org/agenda/2023/01/connected-devices-need-better-governance/, 2023.

[2] Mackinsey *et al.*, "Connected devices need better governance: Here's how to achieve it," https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things, 2023, accessed: 2023-04-15.

[3] M. Dworkin *et al.*, "Advanced encryption standard(aes)," *Federal Inf. Process. Stds. (NIST FIPS)*, 2001.

[4] P. Sivakumar *et al.*, "Securing data and reducing the time traffic using aes encryption with dual cloud," in *IEEE ICSCAN*, 2019, pp. 1–5.

[5] M. Khader *et al.*, "Simplified aes algorithm for healthcare applications on internet of thing," in *8th ICIT*, 2017.

[6] S. Mathew *et al.*, "53 gbps native $gf(2^4)^2$ composite-field aes-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors," *IEEE J. Solid State Circuits*, vol. 46, pp. 767–776, 2011.

[7] R. V. Kshirsagar *et al.*, "FPGA implementation of high speed VLSI architectures for AES algorithm," in *Fifth ICEST, Himeji, Japan*, November 5-7, 2012.

[8] S. Chellappa *et al.*, "Advanced encryption system with dynamic pipeline reconfiguration for minimum energy operation," in *Sixteenth ISQED Santa Clara, CA, USA*, March 2-4, 2015.

[9] E. Kwarteng *et al.*, "A survey on security issues in modern implantable devices: Solutions and future issues," *CoRR*, 2022.

[10] M. Lu *et al.*, "A compact, lightweight and low-cost 8-bit datapath AES circuit for iot applications in 28nm CMOS," in *TrustCom/BigDataSE 2018, New York, USA*, August 1-3, 2018.

[11] S. N. Dhanuskodi *et al.*, "Efficient register renaming architectures for 8-bit AES datapath at 0.55 pj/bit in 16-nm finfet," *IEEE Trans. Very Large Scale Integr. Syst.*, 2020.

[12] M. S. Wamser *et al.*, "Pushing the limits further: Sub-atomic AES," in *VLSI-SoC, Abu Dhabi, United Arab Emirates*, October 23-25, 2017.

[13] S. Banik *et al.*, "Atomic-aes: A compact implementation of the AES encryption/decryption core," in *Progress in Cryptology -17th INDOCRYPT in Kolkata, India*, December 11-14, 2016.

[14] A. Moradi *et al.*, "Pushing the limits: A very compact and a threshold implementation of AES," in *-30th Eurocrypt, Tallinn, Estonia*, 2011.

[15] S. Mathew *et al.*, "340 mv–1.1 v, 289 gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt $(gf(2^4))^2$ polynomials in 22 nm tri-gate cmos," in *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, 2015.

[16] J. Yu *et al.*, "Benchmarking and optimizing aes for lightweight cryptography on asics,," in *Proceedings of the Lightweight Cryptography Workshop, Gaithersburg, MD, USA.*, 4–6 November, 2019.

[17] M.-H. Dao *et al.*, "An energy efficient aes encryption core for hardware security implementation in iot systems," in *2018 International Conference on ATC*, 2018, pp. 301–304.

[18] C. Davis *et al.*, "Shared round core architecture: A novel AES implementation for implantable cardiac devices," in *65th IEEE MWSCAS, Fukuoka, Japan*, August 7-10, 2022.

[19] H. Groß *et al.*, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," *IACR Cryptol. ePrint Arch.*, p. 486, 2016.

[20] X. Zhang *et al.*, "High-speed VLSI architectures for the AES algorithm," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, pp. 957–967, 2004.

[21] Y. Zhou *et al.*, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing," *IACR Cryptol. ePrint Arch.*, p. 388, 2005.

[22] G. Joy Persial *et al.*, "Side channel attack-survey," *Int. J. Adv. Sci. Res. Rev*, vol. 1, pp. 54–57, 2011.

[23] T.-H. Le *et al.*, "How can signal processing benefit side channel attacks ?" in *2007 IEEE Workshop on SAFE*, 2007, pp. 1–7.

[24] E. Brier *et al.*, "Correlation power analysis with a leakage model," in *CHES: MA, USA.*, August 11-13, 2004.

[25] S. Mangard *et al.*, "Side-channel leakage of masked CMOS gates," in *Topics in Cryptology - CT-RSA*, vol. 3376. Springer, 2005, pp. 351–365.

[26] S. Nikova *et al.*, "Threshold implementations against side-channel attacks and glitches," in *ICICS, Raleigh, NC, USA.*, December 2006.

[27] A. Satoh *et al.*, "A compact rijndael hardware architecture with s-box optimization," in *Advances in Cryptology - ASIACRYPT 2001, Gold Coast, Australia.*, C. Boyd, Ed., December 9-13, 2001.

[28] N. Ahmad *et al.*, "Low-power compact composite field AES s-box/inv s-box design in 65 nm CMOS using novel XOR gate," *Integr.*, vol. 46, pp. 333–344, 2013.

[29] Y. Teng *et al.*, "VLSI architecture of s-box with high area efficiency based on composite field arithmetic," *IEEE Access*, vol. 10, 2022.

[30] N. Ahmad, "New architecture of low area aes s-box/ inv s-box using vlsi implementation," *Jurnal Teknologi*, vol. 78, May 2016.

[31] K. Mekki *et al.*, "A comparative study of LPWAN technologies for large-scale iot deployment," *ICT Express*, vol. 5, pp. 1–7, 2019.

[32] D. Thavamani, "Mqtt messages-an overview," *International Journal of Mathematics and Computer Research*, vol. 09, 04 2021.

Bibliography

[1] World Economic Forum. Connected devices need better governance: Here's how to achieve it. https://www.weforum.org/agenda/2023/01/connected-devices-need-better-governance/, 2023. Accessed: 2023-04-15.

[2] https://blog.checkpoint.com/security/the-tipping-point-exploring-the-surge-in-iot-cyberattacks-plaguing-the-education-sector.

[3] Mackinsey and Company. Connected devices need better governance: Here's how to achieve it. https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things, 2023. Accessed: 2023-04-15.

[4] https://www.globenewswire.com/en/news-release/2023/08/11/2723472/28124/en/Global-IoT-Security-Market-Set-for-Robust-Growth-into-2028-Driven-by-Rising-Cybersecurity-Concerns-and-Proliferation-of-Internet-Connected-Devices.html.

[5] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard(aes). *Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD*, 2001.

[6] P. Sivakumar, M. NandhaKumar, R. Jayaraj, and A.Sakthi Kumaran. Securing data and reducing the time traffic using aes encryption with dual cloud. In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, pages 1–5, 2019.

[7] Mariam Khader, Marwah Alian, Raghda Hraiz, and Sufyan Almajali. Simplified aes algorithm for healthcare applications on internet of thing. In *2017 8th International Conference on Information Technology (ICIT)*, pages 543–547, 2017.

[8] Sanu Mathew, Farhana Sheikh, Michael E. Kounavis, Shay Gueron, Amit Agarwal, Steven Hsu, Himanshu Kaul, Mark A. Anders, and Ram Krishnamurthy. 53 gbps native GF(2^4)^2 composite-field aes-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors. *IEEE J. Solid State Circuits*, 46(4):767–776, 2011.

[9] R. V. Kshirsagar and M. V. Vyawahare. FPGA implementation of high speed VLSI architectures for AES algorithm. In *2012 Fifth International Conference on Emerging Trends in Engineering and Technology, Himeji, Japan, November 5-7, 2012*, pages 239–242. IEEE, 2012.

[10] Srivatsan Chellappa, Chandarasekaran Ramamurthy, Vinay Vashishtha, and Lawrence T. Clark. Advanced encryption system with dynamic pipeline reconfiguration for minimum energy operation. In *Sixteenth International Symposium on Quality Electronic Design, ISQED 2015, Santa Clara, CA, USA, March 2-4, 2015*, pages 201–206. IEEE, 2015.

[11] Emmanuel Kwarteng and Mumin Cebe. A survey on security issues in modern implantable devices: Solutions and future issues. *CoRR*, abs/2205.00893, 2022.

[12] Minyi Lu, Ao Fan, Jiaming Xu, and Weiwei Shan. A compact, lightweight and low-cost 8-bit datapath AES circuit for iot applications in 28nm CMOS. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, pages 1464–1469. IEEE, 2018.

[13] Siva Nishok Dhanuskodi, Samuel Allen, and Daniel E. Holcomb. Efficient register renaming architectures for 8-bit AES datapath at 0.55 pj/bit in 16-nm finfet. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(8):1807–1820, 2020.

[14] Markus Stefan Wamser and Georg Sigl. Pushing the limits further: Sub-atomic AES. In *2017 IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2017, Abu Dhabi, United Arab Emirates, October 23-25, 2017*, pages 1–6. IEEE, 2017.

[15] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-aes: A compact implementation of the AES encryption/decryption core. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, volume 10095 of *Lecture Notes in Computer Science*, pages 173–190, 2016.

[16] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[17] Sanu Mathew, Sudhir Satpathy, Vikram B. Suresh, Himanshu Kaul, Mark A. Anders, Gregory K. Chen, Amit Agarwal, Steven Hsu, and Ram Krishnamurthy. 340mv-1.1v, 289gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt gf$(2^4)^2$ polynomials in 22nm tri-gate CMOS. In *Symposium on VLSI Circuits, VLSIC 2014, Digest of Technical Papers, Honolulu, HI, USA, June 10-13, 2014*, pages 1–2. IEEE, 2014.

[18] Jenny Yu and Mark Aagaard. Benchmarking and optimizing aes for lightweight cryptography on asics,. 2019.

[19] Manh-Hiep Dao, Van-Phuc Hoang, Van-Lan Dao, and Xuan-Tu Tran. An energy efficient aes encryption core for hardware security implementation in iot systems. In *2018 International Conference on Advanced Technologies for Communications (ATC)*, pages 301–304, 2018.

[20] Cory Davis and Eugene John. Shared round core architecture: A novel AES implementation for implantable cardiac devices. In *65th IEEE International Midwest Symposium on Circuits and Systems, MWSCAS 2022, Fukuoka, Japan, August 7-10, 2022*, pages 1–4. IEEE, 2022.

[21] Wikipedia. Field (mathematics). `https://en.wikipedia.org/wiki/Field_(mathematics)`, 2023. "[Online; accessed 19-August-2023]".

[22] Christoforus Juan Benvenuto. Galois field in cryptography. *University of Washington*, 1(1):1–11, 2012.

[23] Nabihah Nornabihah Ahmad. *Novel digital VLSI implementation of data encryption algorithm using nano-metric CMOS technology: a thesis presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering at Massey University, Auckland, New Zealand*. PhD thesis, Massey University, 2013.

[24] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptol. ePrint Arch.*, page 486, 2016.

[25] Wikipedia. Rijndael_S-box. `https://en.wikipedia.org/wiki/Rijndael_S-box`, 2023. "[Online; accessed 06-September-2023]".

[26] Xinmiao Zhang and Keshab K. Parhi. High-speed VLSI architectures for the AES algorithm. *IEEE Trans. Very Large Scale Integr. Syst.*, 12(9):957–967, 2004.

[27] Yongbin Zhou and Dengguo Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptol. ePrint Arch.*, page 388, 2005.

[28] G Joy Persial, M Prabhu, and R Shanmugalakshmi. Side channel attack-survey. *Int. J. Adv. Sci. Res. Rev*, 1(4):54–57, 2011.

[29] Thanh-Ha Le, Jessy Clediere, Christine Serviere, and Jean-Louis Lacoume. How can signal processing benefit side channel attacks ? In *2007 IEEE Workshop on SAFE*, pages 1–7, 2007.

[30] Swarup Bhunia and Mark Tehranipoor. Chapter 8 - side-channel attacks. In Swarup Bhunia and Mark Tehranipoor, editors, *Hardware Security*, pages 193–218. Morgan Kaufmann, 2019.

[31] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[32] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[33] Owen Lo, William J. Buchanan, and Douglas Carson. Power analysis attacks on the aes-128 s-box using differential power analysis (dpa) and correlation power analysis (cpa). *Journal of Cyber Security Technology*, 1(2):88–107, 2017.

[34] Wikipedia. Hamming weight. `https://en.wikipedia.org/wiki/Hamming_weight`, 2023. "[Online; accessed 8-August-2023]".

[35] Wikipedia. Hamming distance. `https://en.wikipedia.org/wiki/Hamming_distance`, 2023. "[Online; accessed 8-August-2023]".

[36] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[37] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and aes, secure against some attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[38] Elena Trichina, Domenico De Seta, and Lucia Germani. Simplified adaptive multiplicative masking for AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2002.

[39] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the AES s-box. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005.

[40] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably secure masking of AES. *IACR Cryptol. ePrint Arch.*, page 101, 2004.

[41] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[42] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[43] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[44] Yvo Desmedt. Some recent research aspects of threshold cryptography. In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *Information Security, First International Workshop, ISW '97, Tatsunokuchi, Japan, September 17-19, 1997, Proceedings*, volume 1396 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 1997.

[45] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[46] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.

[47] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3x3 and 4x4 s-boxes. *IACR Cryptol. ePrint Arch.*, page 300, 2012.

[48] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient AES threshold implementation. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer, 2014.

[49] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold implementations of small s-boxes. *Cryptogr. Commun.*, 7(1):3–33, 2015.

[50] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[51] Oscar Reparaz. A note on the security of higher-order threshold implementations. *IACR Cryptol. ePrint Arch.*, page 1, 2015.

[52] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.

[53] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact rijndael hardware architecture with s-box optimization. In Colin Boyd, 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, 2001.

[54] Nabihah Ahmad and S. M. Rezaul Hasan. Low-power compact composite field AES s-box/inv s-box design in 65 nm CMOS using novel XOR gate. *Integr.*, 46(4):333–344, 2013.

[55] You-Tun Teng, Wen-Long Chin, Deng-Kai Chang, Pei-Yin Chen, and Pin-Wei Chen. VLSI architecture of s-box with high area efficiency based on composite field arithmetic. *IEEE Access*, 10:2721–2728, 2022.

[56] Nabihah Ahmad. New architecture of low area aes s-box/ inv s-box using vlsi implementation. *Jurnal Teknologi*, 78(5-9), May 2016.

[57] Kais Mekki, Eddy Bajic, Frédéric Chaxel, and Fernand Meyer. A comparative study of LPWAN technologies for large-scale iot deployment. *ICT Express*, 5(1):1–7, 2019.

[58] Dr Thavamani. Mqtt messages-an overview. *INTERNATIONAL JOURNAL OF MATHEMATICS AND COMPUTER RESEARCH*, 09, 04 2021.

[59] NewAE Technology. CW305 Artix FPGA Target Board. `http://store.newae.com/cw305-artix-fpga-target-board/`. Accessed: 2023-04-15.

[60] Olivier Bronchain. The side-channel analysis library (scalib). `https://github.com/simple-crypto/SCALib`, 2021. Accessed: 2023-04-15.

[61] TSMC. 0.18-micron Technology. `https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_018micron`, 1998. "[Online; accessed 13-July-2023]".

[62] Wikipedia. NCSim. `https://en.wikipedia.org/wiki/NCSim`, 2023. "[Online; accessed 24-July-2023]".

[63] Synopsys. Synopsys Design Compiler. `https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html`, 2023. "[Online; accessed 13-July-2023]".

[64] Synopsys. Synopsys Spyglass Power. `https://www.synopsys.com/verification/static-and-formal-verification/spyglass/spyglass-power.html`, 2023. "[Online; accessed 13-July-2023]".

[65] NewAE Technology. CW1173 ChipWhisperer-Lite. `https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA/`, 2023. "[Online; accessed 17-July-2023]".

[66] NewAE Technology. CW1173 ChipWhisperer-Lite. `https://rtfm.newae.com/Capture/ChipWhisperer-Lite/`, 2023. "[Online; accessed 17-July-2023]".

[67] Wikipedia. Project Jupyter. `https://en.wikipedia.org/wiki/Project_Jupyter`, 2023. "[Online; accessed 30-August-2023]".

[68] J.L. Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, pages 204–, 1994.

[69] Colin O'Flynn and Zhizhang Chen. Side channel power analysis of an aes-256 bootloader. *Canadian Conference on Electrical and Computer Engineering*, 2015:750–755, 06 2015.