NTNU – Trondheim
Norwegian University of
Science and Technology

DPA-Resistant ASIC implementation of AES

Henrik Fegran

Master of Science in Electronics
Submission date: June 2015
Supervisor: Bjørn B. Larsen, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Side channel attacks such as Differential Power Analysis (DPA) is a hot topic in cryptographic circuit design. This master thesis aims to design, implement and verify a version of Advanced Encryption Standard (AES) that is resistant to DPA attacks. The design should be synthesized in a nanoscale technology and characterized with regards to power, performance, area and evaluated with regards to security measures.

Supervisor: Bjørn B. Larsen, IET.

Sammendrag

Med den økte utbredelsen av små innvevde systemer tilkoblet internet, og internet-of-things blir sikkerhetsforanstaltninger stadig viktigere. Kryptering, og beskyttelse av krypterte kretser blir stadig viktigere. Med denne masteroppgaven er målet å designe en krypteringsbrikke som er i stand til å fungere uten å lekke sensitiv informasjon selv under angrep, mer spesifikt til å være i stand til å motstå differensiell effektanalyseangrep.

En maskert AES-krypterings- og dekrypteringsalgoritme med 128-bits datapath blir foreslått, den støtter AES-128, 192 og 256 med cipher-block chaining modus. Systemet har blitt syntetisert til 65nm og oppnå en ytelse på 0.99-1.32 Gb/s ved 400MHz avhengig av nøkkelmodus, med et gjennomsnittlig effektforbruk på 167.9mW. Den maskerte tilnærmingen benyttet skal være i stand til å motstå selv andreordens DPA-angrep med en arealkostnad på 486% av den umaskerte ekvivalente kretsen.

Abstract

With the increased proliferation of small embedded systems connected to the internet and the internet-of-things, the security concerns becomes increasingly important. Encryption, and the protection of encrypted circuits can be of great importance. With this thesis the aim was to design an encryption chip that was able to operate without leaking sensitive information even in the presence of a malicious adversary, specifically to be able to withstand differential power analysis attacks.

A masked 128-bit data-path AES encryption and decryption architecture is proposed, supporting AES-128, 192 and 256 using cipher-block chaining mode of operation. Synthesized to 65nm technology, the system achieves a keymode-dependent throughput of 0.99-1.32 Gb/s operating at 400MHz with an average power consumption of 167.9mW. Our masking approach should withstand second order DPA-attacks at an area cost of 486% compared to the unmasked equivalent circuit.

Preface

This master thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in electronics engineering at the Norwegian University of Science and Technology (NTNU). The thesis is a result of a self-defined research project which was performed over the course of five months, from January to June 2015 at the institute for electronics and telecommunications at NTNU.

I would like to thank my supervisor, Bjørn B. Larsen, as well as my family and friends for their guidance, feedback and support throughout my studies.

Trondheim, June 17th. 2015
Henrik Fegran

Contents

1 Introduction **1**
1.1 Historical Perspective and Motivation . . . 2
1.2 Thesis Organization . . . 3

2 Advanced Encryption Standard **4**
2.1 Background . . . 5
2.2 Overview . . . 6
2.3 Preliminaries . . . 7
2.3.1 Standard Conventions . . . 7
2.3.2 Finite Fields . . . 8
2.3.3 Addition in GF(2^8) . . . 9
2.3.4 Multiplication in GF(2^8) . . . 9
2.4 Components and Their Function . . . 10
2.4.1 Key Schedule . . . 10
2.4.2 AddRoundKey . . . 11
2.4.3 SubBytes, S-Box and its Derivation . . . 12
2.4.4 Multiplicative Inversion via Isomorphic Mapping . . . 13
2.4.5 ShiftRows . . . 16
2.4.6 MixColumns . . . 17
2.5 Modes of Operation . . . 19

3 Differential Power-Analysis **21**
3.1 Background . . . 22
3.2 Why Does it Work? . . . 22
3.2.1 Security of Glitchy Circuits . . . 23
3.3 Method Explained . . . 25
3.4 Prevention of DPA . . . 27
3.4.1 Protocol Level Countermeasures . . . 27
3.4.2 Masking . . . 28

4 Implementation **32**
4.1 Overview of Proposed Unmasked Architecture 33
4.2 Control Logic Modules 33
4.2.1 Clock Divider 34
4.2.2 Clock Synchronizer 34
4.2.3 Main Control FSM 35
4.2.4 Round Counter 35
4.3 Key Expansion 36
4.4 Key Registers 39
4.5 Core Interconnect 39
4.6 S-Box 39
4.6.1 Affine Transformation and Isomorphism 40
4.6.2 Notation in Normal and Polynomial Basis Calculations . . 40
4.6.3 GF(2^4) Square-Scaler 41
4.6.4 GF(2^4) Multiplier 43
4.6.5 GF(2^4) Inverter 44
4.7 ShiftRows 46
4.8 MixColumns 46
4.9 Masking 49
4.10 Cipher Block Chaining 51
4.11 The Complete Masked Core Architecture 51

5 Test Methodology & Results **53**
5.1 Software Setup 54
5.2 Simulation 54
5.3 Synthesis 55
5.4 Power Analysis 56
5.5 Performance 56
5.6 Security Considerations 57
5.7 Possible Improvements/Changes 57

6 Conclusion **58**
6.1 Summary of Thesis and Results 59
6.2 Further Work 59

A A **64**
A.1 Isomorphisms 64
A.2 Sharing Schemes 65
A.3 Inversion in GF(2^8) 66
A.4 GF(2^4) Inverter - Masked Realization in Polynomial Basis 67
A.5 GF(2^4) Inverter - Masked Realization in Normal Basis 72
A.6 GF(2^4) Multiplier - Masked Realization in Polynomial Basis 75

A.7 GF(2^4) Multiplier - Masked Realization in Normal Basis 77

List of Figures

2.1 Overview of the AES Algorithm 6
2.2 Representation of Data in AES 7
2.3 AddRoundKey transformation 12
2.4 SubByte transformation 12
2.5 Polynomial Basis GF(2^8) Inverter 14
2.6 Normal Basis GF(2^8) Inverter 16
2.7 The ShiftRows transformation 16
2.8 The MixColumns Transformation 17
2.9 Unencrypted 19
2.10 ECB 19
2.11 CBC 19
2.12 CBC mode of operation, encryption and decryption respectively. . 20

3.1 Circuit of function 3.1 23
3.2 Power Trace collected from unprotected AES[20]. 25
3.3 Register remasking as suggested by Moradi[24] for a 3-share case. . 31

4.1 Basic structure of unmasked AES implementation. 33
4.2 1:3 Clock Divider Circuit 34
4.3 Finite State Machine - S_1: Wait for key, S_2: Wait for data, S_3: Do rounds, Signals (A/B/C): A: key stable, B: Input Data Ready, C: Rounds Executing 35
4.4 Key expansion 38
4.5 Structure of suggested non-masked polynomial basis S-box implementation 40
4.6 Structure of suggested non-masked normal basis S-box implementation 40
4.7 Polynomial Basis Square-Scaler 42
4.8 Normal Basis Square-Scaler 43

4.9 Modified (Inv)MixColums implementation based on the architecture proposed by Zhang[36]. 48
4.10 Masking overview of linear components 49
4.11 Structure of suggested masked polynomial basis S-box implementation 50
4.12 Structure of suggested masked normal basis S-box implementation 50
4.13 AES-core implementation 52

5.1 Simulated Power Trace from PrimeTime 56

List of Tables

2.1 Round-block-key Combinations 8

3.1 AND/XOR-transitions based on late arrival of X_2 24

4.1 Key Expansion Rounds, P: Passthrough, R: Rotate-SubByte-Rcon (RSR), X: XOR, S: Subbyte 37

5.1 Synthesis Results 55
5.2 Primetime Results 56
5.3 Performance @400MHz 57

Acronyms

AES Advanced Encryption Standard

ANF Algebraic Normal Form

CBC Cipher-Block Chaining

CEPACA Correlation-Enhanced Power Analysis Collision Attack

CMOS Complementary Metal-Oxide-Semiconductor

CPA Correlation Power Analysis

DES Data Encryption Standard

DPA Differential Power Analysis

ECB Electronic Codebook

FF Flip-Flop

FSM Finite-State Machine

GF Galois fields

HO-DPA Higher Order Differential Power Analysis

IoT Internet of Things

IV Initialization Vector

NIST National Institute for Standards and Technology

ROM Read-Only Memory

S-box Substitution-box

SCA Side-Channel Attack

SNR Signal-to-Noise Ratio

SPA Simple Power Analysis

SPN Substitution-Permutation Network

XOR Exclusive-OR

Chapter 1

Introduction

1.1 Historical Perspective and Motivation

Cryptography, the art of secret writing, has roots back to ancient times. However it is not until recent decades with the advent of semiconductor technology and the internet age that cryptosystems have become widespread outside of government and military areas. Today the proliferation of embedded systems is ever increasing, including such applications as smart cards, cellular phones, key-less entry systems and the recent popular buzz word, Internet of Things (IoT).

The widespread adoptance of embedded systems, and especially internet connected embedded systems bring with them an increased need for security. Researchers have spent considerable effort in designing secure cryptographic ciphers, the Advanced Encryption Standard (AES) notably being among the most widespread after being adopted by the US National Institute of Standards and Technology in 2001.

However, the security of a cipher not only lies in its inherent mathematical properties, also the equipment on which it is implemented can be targeted to attain knowledge of the secret. This class of attacks is known as side-channel attacks and includes attacks such as timing-, power- and electromagnetic radiation analysis.

Differential Power Analysis (DPA) is one such attack that has been shown to be a powerful tool in breaking previously assumed secure cryptosystems. A famous example is the attack on the KeeLoq-system [16] used in many cars for keyless entry. Not only was DPA able to retrieve the secret key, but also the manufacturer key was retrieved such that new valid remote controls could be easily made in a short amount of time.

Since DPA was suggested by Kocher[19], much research has been performed in this field, and our aim is to build on this research and propose a complete secure cryptosystem utilizing AES that is not vulnerable to DPA and implemented on standard CMOS.

1.2 Thesis Organization

- **Chapter 1: Introduction** gives an overview of the topic, a historical perspective and motivation behind this thesis.
- **Chapter 2: Advanced Encryption Standard** explains the fundamental principles and mathematical preliminaries of the advanced encryption standard algorithm. Furthermore, previous research considered for this thesis is presented.
- **Chapter 3: Differential Power-Analysis** explains the principles behind the side-channel attack known as differential power-analysis (DPA) and presents relevant research about DPA and protection thereof.
- **Chapter 4: Implementation** gives a detailed overview of the implementation steps taken to create an DPA-resistant AES-architecture in SystemVerilog.
- **Chapter 5: Test Methodology & Results** shows the step taken to verify and analyze the proposed architecture and the results obtained.
- **Chapter 6: Conclusion and Further Work** Presents the result of the thesis and possible future research.
- **Appendix A:** Contains some additional formulas and materials.

Chapter 2

Advanced Encryption Standard

2.1 Background

Since the end of the 1970s, the Data Encryption Standard (DES) had been the predominant symmetric-key algorithm for encryption of electronic data. By the 1990s, due to advances in cryptanalysis, computing power and its limited key length of only 56 bits was no longer considered secure. Consequently, in 1997 the National Institute for Standards and Technology (NIST) reached out for proposals of algorithms that that could replace DES and become the Advanced Encryption Standard. It was intended that the AES would specify an unclassified, publicly disclosed encryption algorithm that were to be available royalty free world wide and capable of protecting sensitive government information well into the next century[27]. After three years, in October 2000, an algorithm coined Rijndael after its creators, Vincent Rijmen and Joan Daemen was chosen to become the new advanced encryption standard.

One of the primary features of AES is its good performance in both hardware and software implementations. On smartcards, AES can be implemented with less than 1kb of code and using only 36 bytes of memory. With high-end processors AES may exploit the use of caches and parallelism to achieve significant speedups[35]. The main cryptographic design goal of AES was to provide adequate and provable protection against linear and differential cryptanalysis.

The following design principles were used to achieve the effectiveness and security of the cipher[35]:

- Keep it simple - No unneeded complexity, everything is there for a reason. The cipher should be secure against known attacks but should also avoid introducing new vulnerabilities.

- Modularity - AES is built from several distinct modules, each selected according to quantitative selection criteria.

- Symmetry and Parallelism - All steps can be parallelized and and operate on the data in a symmetrical way to allow great freedom in designing efficient hardware implementations.

- Choice of operations - All steps are defined in GF(2^8) and can be implemented with Exclusive-OR (XOR) gates and table lookups only. No arithmetic operations are needed thus reducing the area consumption on hardware platforms.

2.2 Overview

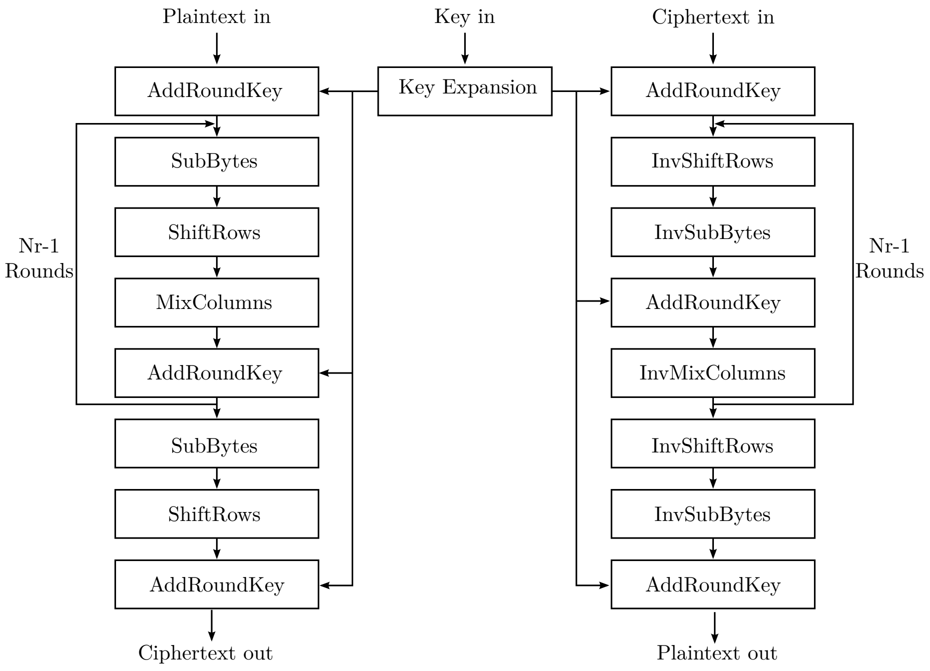


Figure 2.1: Overview of the AES Algorithm

Figure 2.1 gives an overview of the various parts and dataflow of the AES encryption and decryption algorithms. The left part of the figure describes the encryption process, while the right side describes the decryption process. After initialization of the AES algorithm. a cipher key is asserted to the key expansion module, which expands this key into a number of round keys, one for each round. When the first round key is ready, the algorithm is ready to accept input data; plaintext in the case of encryption or ciphertext in the case of decryption. In the initial round, the input data is combined with the first round key, then passed on to the next step, the rounds of AES, which are the following four modules whose operation is repeated Nr times (values of Nr is given in table 2.1). After being processed Nr times, the data is passed on to be processed by the remaining three modules before it is asserted on the output as ciphertext or plaintext for encryption and decryption respectively. In section 2.4, a more detailed description of the algorithm and its constituents will be presented.

2.3 Preliminaries

2.3.1 Standard Conventions

The AES algorithm is a symmetric block cipher, as such the key for encryption and decryption remains the same. To provide the confusion and diffusion properties as described by Shannon [34] a Substitution-Permutation Network (SPN) is used. Input and output data, as well as the internal data representation, called the state, are all represented by blocks of 128 bits divided into 8 bit subfields in a 4x4 row column fashion as seen in figure 2.2

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Figure 2.2: Representation of Data in AES

Key length is defined to be 128, 192 or 256 bits, no other key-lengths nor input/output sizes are allowed according to the standard. The state undergoes four transformations, namely AddRoundKey, SubByte, ShiftRow and MixColumns, of which the SubByte transformation is the most computationally heavy. Together these four transformations form what is known as a round of AES which is repeated a specific number of times as given in table 2.1. Modules used in the decryption process are denoted as the inverse of the original function, and performs the operation in reverse compared to the regular variant.

This table gives the combinations of allowed key-lengths, data block sizes and number of rounds as specified by AES. No other combinations are approved for use.

Table 2.1: Round-block-key Combinations

	Key Length	Block Size	Number of Rounds
	Nk words	*Nb words*	*Nr rounds*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

2.3.2 Finite Fields

Finite fields, or Galois fields (GF) form the foundation for the mathematical operations in the AES algorithm. A field is a set of F elements with two binary operations, $\oplus$ and $\otimes$, addition and multiplication respectively.

For all operations, the result must be confined within this field, and thus these operations must satisfy the following properties [12]:

1. The set is *closed* with respect to both operations:
   (a) $a \oplus b \in F$
   (b) $a \otimes b \in F$
2. Both operations are *associative*:
   (a) $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
   (b) $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
3. Both operations are *commutative*:
   (a) $a \oplus b = b \oplus a$
   (b) $a \otimes b = b \otimes a$
4. The operations are *distributive*: $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
5. For both operations, an *identity* exists:
   (a) $a \oplus 0 = a$
   (b) $a \otimes 1 = a$
6. Each element in the field has an *additive inverse*: If q is the additive inverse of a then $a \oplus q = 0$ and as such the additive inverse defines subtraction. Commonly written as $-a$.
7. Each element in the field has a *multiplicative inverse*: If r defines the multiplicative inverse of a then $a \otimes r = 1$. Notation for the multiplicative inverse is a^{-1}. See section on multiplication in GF(2^8) for more details.

The basic unit of operation within each block is one byte, consisting of 8 bits presented in the following order $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1\}$. These bytes are interpreted as finite field elements in GF(2^8) using polynomial representation:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 = \sum_{i=0}^{7} b_i x^i \tag{2.1}$$

Thus it is convenient to refer to individual bytes with the following equivalent representations:

$$\begin{aligned} &\text{Polynomial :} & x^7 + x^6 + x^3 + x^2 + 1 \quad &(2.2)\\ &\text{Binary :} & \{11001101\} \quad &(2.3)\\ &\text{Hexadecimal :} & \{\text{CD}\} \quad &(2.4) \end{aligned}$$

2.3.3 Addition in GF(2^8)

Addition in GF(2^8) is performed by adding the polynomial coefficients together modulo 2. In other words, to bytes can be added together by exclusive-OR (denoted by $\oplus$) of the values in its respective bit positions. Consequently, addition and subtraction are equivalent operations.

$$(x^5 + x^4 + x + 1) + (x^7 + x + 1) = x^7 + x^5 + x^4 \tag{2.5}$$

$$\{00110011\} + \{10000011\} = \{10110000\} \tag{2.6}$$

$$\{33\} + \{83\} = \{\text{B0}\} \tag{2.7}$$

2.3.4 Multiplication in GF(2^8)

Multiplication of polynomials in GF(2^8) (denoted by $\otimes$) is accomplished by a polynomial multiplication modulo an irreducible polynomial of degree 8. For AES this polynomial is

$$m(x) = x^8 + x^4 + x^3 + x + 1 = \{01\}\{1\text{D}\} \tag{2.8}$$

This polynomial ensures that the resulting polynomial will be of degree less than 8 and as such can be represented by the finite field GF(2^8).

Multiplication by a polynomial

$\{57\} \otimes \{83\} = \{C1\}$ because

$$\begin{aligned} (x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) \mod m(x) &= \\ x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \mod m(x) &= \\ x^7 + x^6 + 1 & \end{aligned} \tag{2.9}$$

Note that the two factors of equal powers cancel each other out due to the additive inverse property. The modulo result can be easily obtained by either polynomial long division or simply by recursively replacing all instances of x^8 with the root of $m(x)$, $x^4 + x^3 + x + 1$.

Multiplication by a constant

The method of multiplication of a polynomial and a constant are identical if the constant is represented as a polynomial in GF(2^8). For example, the constant 3 in binary is $\{11\}_2$ which can also be represented as the polynomial $x + 1$ in the finite field and thus the multiplication is performed as above.

2.4 Components and Their Function

2.4.1 Key Schedule

The key schedule of AES consists of two parts, key expansion and a round key selection function. Once per round, the algorithm requires the addition of 128 bits of key data. To supply the necessary key bits, the cipher key, which is 128, 192 or 256 bits long needs to be expanded so that there is 128 unique bits necessary for each round. AES specifies a lightweight algorithm, as given in algorithm 1, to perform the key expansion. A lightweight key expansion routine is necessary for a general purpose encryption algorithm, particularly for applications which require frequent change of secret keys [35].

The key expansion of the AES algorithm takes an input key and generates a total of $Nb(Nr + 1)$ round keys. This forms the basis for a key scheduling routine. Upon the initialization of the AES algorithm, one round key of Nb words (128 bit) is required, followed by one round key for each successive round. Round keys are generated as a function of previous round keys; conveniently enabling generation of round keys for encryption on the fly. For decryption, the round keys are utilized in reverse, and thus not conveniently generated on the fly. The key expansion function itself requires two further functions, RotWord and SubWord in addition to a special register rcon. The function SubWord() applies the SubByte s-box substitution on each byte in the four words and RotWord()

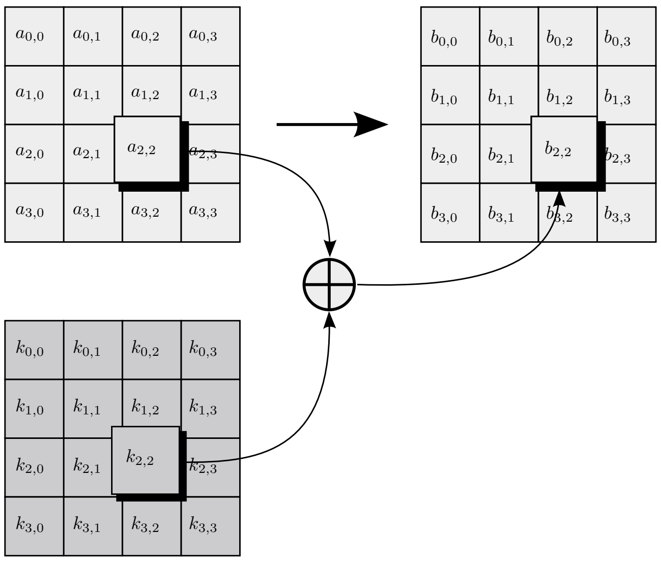
performs a cyclic left shift by one byte. The round constant word array Rcon[i] contains the values $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ where x^i being powers of $x = \{02\}$.

Algorithm 1 Key expansion algorithm

```
1:  function KeyExpansion(byte key[$4 * Nk$], word w[$Nb * (Nr + 1)$], Nk)
2:      word temp
3:      i = 0
4:      while ($i < Nk$) do
5:          $w[i]$ =word(key[$4 \cdot i$], key[$4 \cdot i + 1$], key[$4 \cdot i + 2$], key[$4 \cdot i + 3$])
6:          $i = i + 1$
7:      end while
8:      i = Nk
9:      while ($i < Nb \cdot (Nr + 1)$) do
10:         temp = $w[i - 1]$
11:         if ($i \mod Nk = 0$) then
12:             temp = SubWord(RotWord(temp)) $\oplus$ Rcon[$i/Nk$]
13:         else if ($Nk > 6$ and $i \mod Nk = 4$) then
14:             temp = SubWord(temp)
15:         end if
16:         $w[i] = w[i - Nk] \oplus$ temp
17:         $i = i + 1$
18:     end while
19: end functionKeyExpansion
```

2.4.2 AddRoundKey

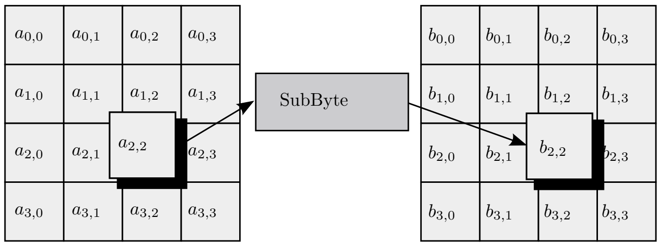
The AddRoundKey transformation is where the state is combined with the round key for the current round. Each byte in the state is XORed with its corresponding bytes in the round key. Expansion of the round key from the cipher key is explained in detail in section 2.4.1



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.3: AddRoundKey transformation

2.4.3 SubBytes, S-Box and its Derivation



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubByte
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.4: SubByte transformation

SubBytes is the function that substitutes a byte from the state with its corresponding match in the substitution box. The Substitution-box (S-box) is the part of a SP-network that provides non-linearity to the cipher. Several older ciphers used modifiable S-boxes, e.g. GOST 28147-89, where it was rumored that the governing authorities would provide secure S-boxes for entities trusted, and insecure or backdoored s-boxes for others [33]. AES on the other hand was specifically designed to avoid such problems by using a fixed, standardized S-box based on a mathematical derivation known to provide good degree of non-linearity and resistance to known cryptanalyisis-attacks.[13] By standardizing the S-box based on known mathematical properties, there are two main approaches to its implementation. Firstly the traditional look-up table based method, where the precomputed S-box values would be stored in a ROM and fetched as needed. Using a ROM based method would incur some performance penalties with each access to the ROM. For encryption or decryption alone, a 256 byte look-up table would be needed and 512 bytes if both operations are to be supported. The alternative is to implement the S-box inversion as logic based on its mathematical properties. Implementation of a secure S-box in AES is one of the largest challenges in creating efficient hardware and software.

The S-box consists of two parts, a multiplicative inverse in GF(2^8) followed by an affine transformation and the addition of a constant. The inverse transformation used in decryption is accomplished by adding the inverse affine transform with a constant before inverting the input. The affine transformation is given in equation 2.10. Input The multiplicative inverse was chosen to provide necessary non-linearity to the cipher as required by the wide trail strategy[14] to protect against linear and differential cryptanalysis, while the affine mapping seeks to complicate the algebraic expressions without affecting the nonlinearity properties to combat cryptanalysis methods such as interpolation attacks[35].

Decryption uses the same inversion module from the S-box, the difference lies in adding an inverse affine transformation and constant to the byte prior to inversion.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.10}$$

2.4.4 Multiplicative Inversion via Isomorphic Mapping

To create a more area efficient implementation of the S-box, based on research by Rijmen [30], Canright [12] suggested using a Tower- or composite field approach to implement the S-box. The basic idea is that the data is isomorphically mapped to a smaller subfield in which the operations become less complicated and technically challenging to implement. In the following section this will be further elaborated.

Polynomial Basis

Finite fields such as the Galois field used in AES are unique and finite. However, multiple methods for representing these fields and its elements exist. The first considered in this thesis is the polynomial basis. The foundation for the polynomial basis is that each element in the field-array, $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$ is represented by a polynomial $a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$, where a_i is either 0 or 1 depending on the value of the element, and the powers of x defines the position. Furthermore, the field itself is constructed using an *irreducible primitive polynomial*; such that the root of this polynomial function as a generator for the field. To give an example for GF(2^8), the smallest such polynomial is $f(x) = x^8 + x^4 + x^3 + x^2 + 1$, with one *primitive root*, $\gamma = x^4 + x^3 + x^2 + 1$.

The primitive root forms the basis of the field, by taking powers of $\gamma \mod f(x)$ all the elements in the field can be uniquely defined. As such, any possible polynomial that represents the field can be expressed by γ to some distinct power. For example, $\gamma^2 = x^6 + x^3 + x^2 + 1 = \{01001101\}$. Note that two equal factors cancel, due to the additive inverse property previously described.

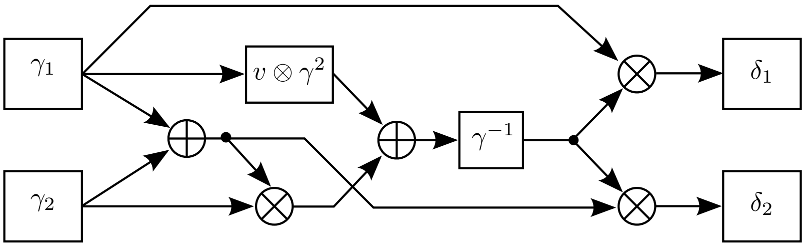


Figure 2.5: Polynomial Basis GF(2^8) Inverter

Calculating the multiplicative inverse in GF(2^8) is quite complicated. Canright [12] showed that it is possible to represent the inverse in GF(2^8) by its coefficients in GF(2^4), with an irreducible *primitive* polynomial $y^2 + y\tau + \nu$ where τ is the trace and ν is the norm of the function, as follows (For the complete derivation, see appendix):

$$\begin{aligned} g =& \gamma_1 y + \gamma_0 \\ d =& g^{-1} = \delta_1 y + \delta_0 \\ \text{s.t. } gd =& 1 \end{aligned} \tag{2.11}$$

$$\begin{aligned}\delta_1 =& (\gamma_1^2 + \gamma_1\gamma_0\tau + \gamma_0^2)^{-1}\gamma_1 \\ \delta_0 =& (\gamma_1^2 + \gamma_1\gamma_0\tau + \gamma_0^2)^{-1}(\gamma_0 + \gamma_1\tau)\end{aligned} \tag{2.12}$$

Here the coefficients, γ_1 and γ_2 are represented in GF(2^4) together forming the GF(2^8) field. From the above formula, it is apparent that the inversion in GF(2^8) can be carried out by a combination of multiplications, additions, squarings and inversion in GF(2^4). Further decomposition down to GF(2^2) and GF(2) is possible for further simplification. [12]. A possible realization of this in hardware is shown in figure 2.5. For the full realization here we may either use an inversion in GF(2^4 directly, or continue decomposing the GF(2^4) inverter down to GF(2^2) where inversion amounts to a simple bit swap.

The AES field polynomial however is irreducible but not primitive, so the above representation does not automatically apply. First, a linear mapping, or isomorphism, is necessary to represent the values in a properly constructed field. The basic idea here is to map a primitive element of GF($(2^n)^m$) to GF(2^k) such that the group homomorphism holds. For further details the reader may confer [32][28].

Normal Basis

Polynomial basis is not the only possible representation of a finite field. Several other representations exist, of which normal basis is one of the more commonly used.

Consider a string of bits $\{b_m, b_{m-1}, \ldots, b_1, b_0\}$ in GF(2^m). To represent the individual bits with normal basis, we first find an element β, such that the m elements $\{\beta^{2^{m-1}}, \beta^{2^{m-2}}, \ldots, \beta^2, \beta\}$ are linearly independent. As such, $\beta^{2^{m-1}}$ represents the most significant bit, $\beta^{2^{m-2}}$ the second most significant bit and so on.

Normal basis often leads to much more efficient[1] implementation of bit arithmetic in hardware, particularly if squaring is involved as squaring in the normal basis is simply a cyclic shift of bits. [28], although not in all cases. Canright [12] throughly examined the use of normal basis and AES, and concluded that the most efficient implementation uses normal basis for the s-box, but not for the rest of the affine transformations.

[1]Efficient with regards to low usage of logic gates in the implementation

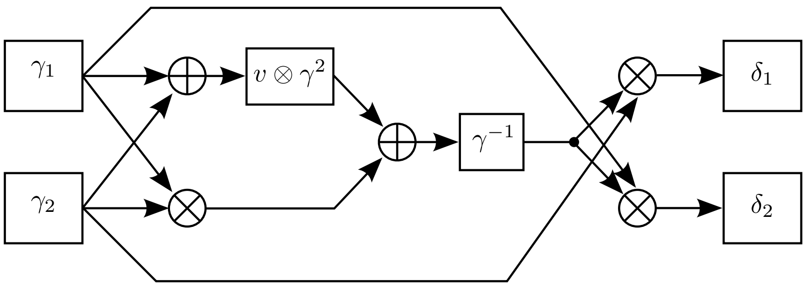


Figure 2.6: Normal Basis GF(2^8) Inverter

Similarly to polynomial basis, the inverse in normal basis over GF(2^8) can be derived, in [12] it was shown to be as follows:

$$\begin{aligned} g =& \gamma_1 \mathbf{Y}^{16} + \gamma_0 \mathbf{Y} \\ g^{-1} =& d = \delta_1 \mathbf{Y}^{16} + \delta_0 \mathbf{Y} \\ \text{s.t. } gd =& 1 \end{aligned} \tag{2.13}$$

$$\begin{aligned} \delta_1 =& \gamma_0 \left[\gamma_1 \gamma_0 \tau^2 + (\gamma_0^2 + \gamma_1^2)\nu\right]^{-1} \\ \delta_0 =& \gamma_1 \left[\gamma_1 \gamma_0 \tau^2 + (\gamma_0^2 + \gamma_1^2)\nu\right]^{-1} \end{aligned} \tag{2.14}$$

Selecting τ, the trace, to be one leads to a structure similar to that of the polynomial basis as shown in figure 2.6, however the internal functions are all performed with the normal representation, so that the underlying logic will be substantially different. Canright found this representation to give the smallest S-box implementation so far, although critical path was not taken into consideration.

2.4.5 ShiftRows

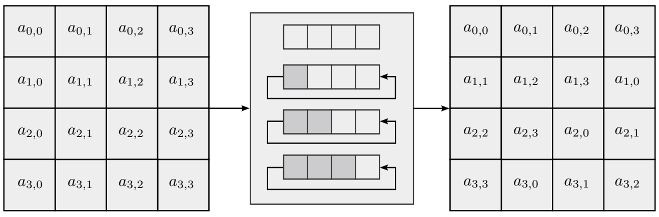

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

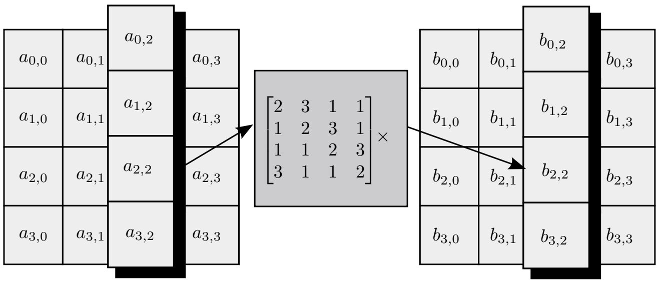
Figure 2.7: The ShiftRows transformation

The ShiftRows transformation operates on the state in a row-based manner. 0-indexing the rows from top to bottom, rows are cyclically shifted left the number of bytes denoted by their index. Mathematically this can be described as follows:

$$S'_{r,c} = S_{r,(c+shift(r,Nb)) \mathrm{mod} Nb} \text{ for } 0 < r < 4 \text{ and } 0 \le c < \mathbf{Nb} \tag{2.15}$$

Without the ShiftRows transformation, the columns of the AES cipher would be linearily independent, and thus be equivalent to four distinct ciphers running on each column. The inverse ShiftRows transformation performs the same number of cyclical shifts as ShiftRows but in the right direction.

2.4.6 MixColumns



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$\begin{bmatrix}2&3&1&1\\1&2&3&1\\1&1&2&3\\3&1&1&2\end{bmatrix}\times$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.8: The MixColumns Transformation

The MixColumns transformation operates on the state in a column-by-column fashion, treating each term as a four term polynomial in GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$ given as

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{2.16}$$

Writing this as matrix multiplication we have

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{2.17}$$

Thus equating to the following transformed state bytes:

$$\begin{aligned} s'_{0,c} &= (\{02\} \cdot s_{0,c}) \oplus (\{03\} \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \cdot s_{1,c}) \oplus (\{03\} \cdot s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \cdot s_{2,c}) \oplus (\{03\} \cdot s_{3,c}) \\ s'_{3,c} &= (\{03\} \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \cdot s_{3,c}) \end{aligned} \tag{2.18}$$

Similarily for the inverse case, the operation differs by the coefficients of the four term polynomial, that is:

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x^1 + \{0e\}x \tag{2.19}$$

2.5 Modes of Operation

When a block cipher is used for confidentiality protection, the goal is to prevent an adversary with limited computational power to gain any knowledge of the plaintext. The basic AES algorithm operates on each block of input data completely independent of past or future inputs, only the key and the input data defines the output. This is known as Electronic Codebook (ECB) mode. The inherent weakness by this approach is that each and every identical plaintext will produce identical output[35].

For example, bitmap images normally contains a header followed by uncompressed data specifying the color of the individual pixels. If an image contains large areas of the same color it forms a clear pattern, both visually and in the data. Normally bitmap images are much larger than the block size of the encryption algorithm thus these patterns are repeatedly given as input to the cipher. As a consequence, each block containing the same color information gets encrypted identically leaving the original patterns of the image largely intact. Figure 2.10 shows an example of encrypting the NTNU-logo from figure 2.9 in ECB mode. From these images it is apparent that if the algorithm is used to encrypt bitmap images it does not provide confidentiality as information about the structure of the image is leaked.

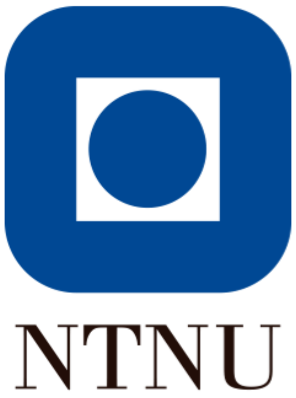


Figure 2.9: Unencrypted

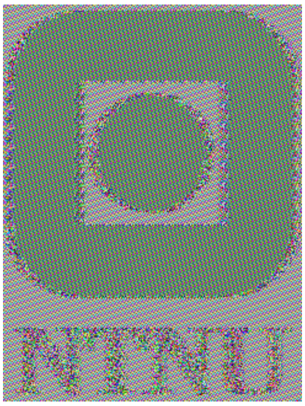


Figure 2.10: ECB

Figure 2.11: CBC

To remedy this problem and provide confidentiality for block ciphers, several modes of operation have been introduced. The most common mode is known as Cipher-Block Chaining (CBC)[35].

Cipher block chaining

Figure 2.11 shows the output of the AES encryption of the NTNU-logo in CBC-mode. Visually inspecting the figure does not reveal any information of the original image in contrast to the ECB-mode encrypted variant.

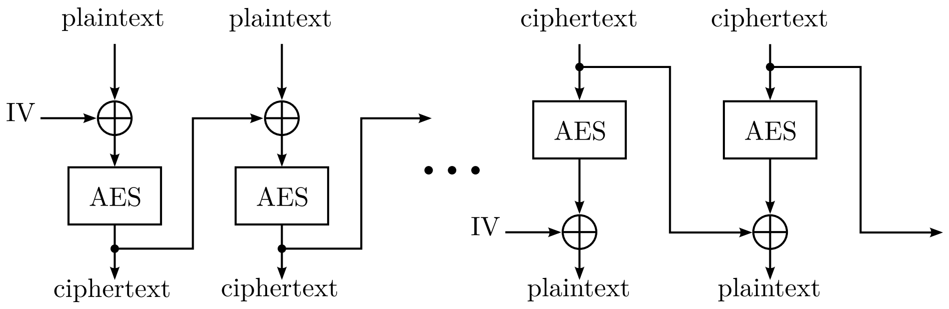


Figure 2.12: CBC mode of operation, encryption and decryption respectively.

Figure 2.12 shows how this is performed in practice. For the first plaintext after a reset, an additional Initialization Vector (IV) is required as no previous ciphertext is available. The IV is then combined with the plaintext, together forming the input to the algorithm. Subsequent blocks combine the previous ciphertext with the current plaintext instead of the IV. This ensures that each block of input data to the cipher is not identical. As the output of the previous block is required for encryption, a disadvantage of this mode is that it is not possible to parallelize encryption. Decryption follows a similar principle, the ciphertext (input) from the previous round is combined with the output of the present round. Decryption does not depend on the output of the previous round, only on its input, which makes it possible to implement the mode in parallel.

Chapter 3

Differential Power-Analysis

3.1 Background

Historically, attacks on cryptosystems have largely been targeted towards mathematical weaknesses in the encryption scheme. Several methods, such as linear cryptanalysis [22] and differential cryptanalysis [4] have been devised to display weaknesses in cryptographic algorithms. Modern cryptographic algorithms are designed to withstand such attacks, and in the case of AES there is still no method available that can break the cipher faster than an exhaustive key-search approach. With mathematical attacks on AES out of the question, researchers have turned to side-channel attacks. Side-channel attacks are a group of attacks that do not target the mathematical structure of the algorithm itself, but rather the equipment on which it is implemented. Thus, for a system to be secure it is not enough for a cryptographic algorithm to be able to withstand a cryptanalysis attack. Attacks using defective computations [10], timing information [18] and power consumption [19] have been demonstrated in practice. US government also made a large effort with their TEMPEST-program to secure cryptographic hardware from leaking information through electromagnetic radiation[2].

In this thesis the focus will largely be on a class of side channels attacks based on instantaneous power consumption, known as simple and differential power analysis (SPA and DPA) and its related methods. In the following chapter this will be thoroughly explained.

3.2 Why Does it Work?

Modern cryptosystems, as with most other modern microelectronics is usually implemented in CMOS technology. The CMOS logic gates are composed of transistors, that controls the electron flow across the substrate. This flow is dependent on a charge being applied or removed from the transitor gate, which in turn affects the power consumption and emission of electromagnetic radiation. Furthermore, the activity of the circuit is directly correlated to its power consumption through the aggregate activities of its individual components.

Activity of the components is in turn affected by the operations performed by the circuit. For example, adding the hexadecimal numbers {4A} and {7F} may cause more transistors to switch compared to adding {00} and {01}.

Because the power consumption of a circuit is dependent on the data processed and calculations performed, power measurements will contain information about the processed data. Even the effect of a single transistor in a large system can show a weak correlation to the processed data. As such it is paramount in a secure cryptosystem to avoid leaking secrets in a way that an adversary can easily exploit. DPA and its related attacks of side channels are very powerful methods that may be employed even in systems where large amounts of noise is present.

3.2.1 Security of Glitchy Circuits

The ideal CMOS logic gate exhibits at most one transition per clock cycle, in reality however this is not the case. Propagation delays can cause gates to go through more than one transition during a clock cycle. This is called a glitch, and is very common in unbalanced CMOS circuits. Consider the following example from [5] for a masked sharing of the function $f = Z \oplus XY$ with order of operations shown by parantheses.

$$\begin{aligned} f_1(X_1, Y_1) &= Z_1 \oplus X_1 Y_1 \\ f_2(X_1, X_2, Y_1, Y_2) &= ((Z_2 \oplus X_1 Y_2) \oplus X_2 Y_1) \oplus X_2 Y_2 \end{aligned} \tag{3.1}$$

The function in 3.1 is securely shared as long as the order of operations is maintained and the circuit is free of glitches.

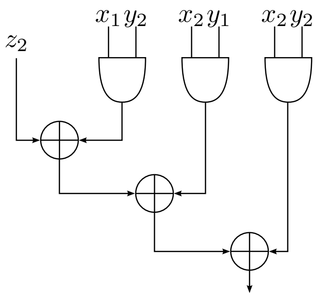


Figure 3.1: Circuit of function 3.1

Assume a constant zero initial state of the inputs followed by a change to a random value with x_2 delayed due to propagation delay. The last two columns of table 3.1 shows the amount of AND and XOR transitions in the circuit with the given input stimuli. Plus signs denote the transitions before and after arrival of x_2.

Comparing the leftmost column, the unmasked (secret) value, and the columns showing the transitions it is apparent that the average number of transitions is not independent of the input value y. Hence the power consumption with different values of y differs and thus reveals information that can be exploited to attack the circuit. Glitches at other points in the circuit may or may not leak secret information, but verifying the safety of each transition in a larger circuit is not feasible.

Table 3.1: AND/XOR-transitions based on late arrival of X_2

y	y_1	y_2	x_2	$z_2 \oplus x_1 y_2$	AND	XOR
0	0	0	0	0	0+0	0+0
0	1	1	0	0	0+0	0+0
0	0	0	1	0	0+0	0+0
0	1	1	1	0	0+2	0+1
0	0	0	0	1	0+0	2+0
0	1	1	0	1	0+0	2+0
0	0	0	1	1	0+0	2+0
0	1	1	1	1	0+2	2+1
1	0	1	0	0	0+0	0+0
1	1	0	0	0	0+0	0+0
1	0	1	1	0	0+1	0+1
1	1	0	1	0	0+1	0+2
1	0	1	0	1	0+0	2+0
1	1	0	0	1	0+0	2+0
1	0	1	1	1	0+1	2+1
1	1	0	1	1	0+1	2+2

3.3 Method Explained

Simple Power Analysis Attack (SPA)

The SPA-attack can be considered a precursor of the differential power analysis. This method involves collecting power traces from the cryptographic hardware, and visually inspecting these for determining data-dependent power variation.

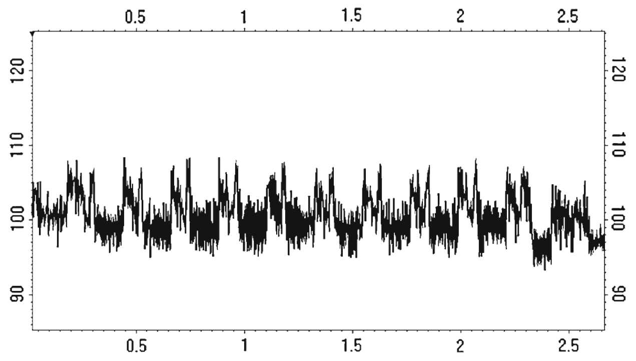


Figure 3.2: Power Trace collected from unprotected AES[20].

From figure 3.2, the various stages of AES is clearly visible - the rounds of AES forms a clear pattern. Visually inspecting and analyzing the power plots to attain knowledge of secret data is very difficult, and the difficulty increases with added noise. This has led to the development of DPA, differential power analysis.

Difference of Means - Basic DPA Attack

The difference of means-attack is DPA in its most basic form, and was the method that was proposed by Kocher [19]. DPA is a statistical method, thus to be enable to employ this attack, the first thing needed is to collect data. Furthermore, it requires knowledge of the employed algorithm. The necessary data consists of power traces from the system during normal operation, where the key is static, and different plaintexts are applied. (in the case of encryption). Next, it is necessary to identify an intermediate value that depends only on a small part of the cipher key, the subkey and then proceed to guess the subkey and generate a list of hypothetical values for the intermediate (commonly hamming weight or hamming distance[20]). These values form the basis of a selection function, which is used to assign traces to subsets.

Kocher[20] summarized the process of a DPA attack as follows: Let T denote the set of traces collected, with T_i being the ith trace. $T_i[j]$ denotes the power measurement at the jth time offset. C corresponds to the set of known inputs or outputs for the traces, while C_i denotes the ith trace. $D(C_i, K_n)$ denotes the selection function with inputs C_i and the partial key guess K_n. Then each point j in the differential trace Δ_D is computed as follows:

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(C_i, K_n) T_i[j]}{\sum_{i=1}^{m} D(C_i, K_n)} - \frac{\sum_{i=1}^{m}(1 - D(C_i, K_n)) T_i[j]}{\sum_{i=1}^{m}(1 - D(C_i, K_n))} \tag{3.2}$$

Typically the guesses of K_n that yields the largest spikes in Δ_D are the most likely candidates for a correct guess.

Correlation Power Analysis

Correlation Power Analysis (CPA) was proposed by Brier et al. in 2004[11] and involves the evaluation of variations in internal key-dependent variables compared to a device leakage model. This leakage model is based on one or more intermediates in the cryptographic calculations, such as correlating power consumption to Hamming weight of a multi-bit register, or the Hamming distance between a value and the next value stored in the same register.

As this method requires a leakage model for the system it is most effective in a white-box attack, where the leakage of the device is fully known. It can also be utilized in a black-box setting if there is some correlation between the actual leakage of the device and the leakage model used by CPA[20].

- Generate $p \times t$ matrix T corresponding to p power traces with length t with values from repeated encryptions with varying plaintexts and the same key.
- Generate $p \times k$ matrix L corresponding to p hypothetical leakage values of the target with all plaintexts and a different key hypothesis.

Using the aforementioned matrices the attacker constructs a matrix R from the values $r_{j,l}$ in equation 3.3, $\bar{T}$ and $\bar{L}$ represents the mean values.

$$r_{j,l} = \frac{\sum_{i=1}^{p}(T_{i,j} - \bar{T}_j)(L_{i,l} - \bar{L}_l)}{\sqrt{\sum_{i=1}^{p}(T_{i,j} - \bar{T}_j)^2}\sqrt{\sum_{i=1}^{p}(L_{i,l} - \bar{L}_l)^2}} \tag{3.3}$$

Each column of R corresponds to the correlation coefficients of a key hypothesis on all the time samples. Indices corresponding to the absolute maxima of elements in R reveals the specific samples where the hypothetical leakage matches a particular key hypothesis indicating a correct guess.

Correlation-Enhanced Power Analysis Collision Attack

Moradi et al.[25] proposed the Correlation-Enhanced Power Analysis Collision Attack (CEPACA) in 2010 that enables a significant reduction in the possible key space in the cryptosytem. The main principle behind CEPACA is to exploit collisions of two different intermediate values. Section 2.2 shows that the input to the s-box is formed by a linear combination of the state and the round key. For two different inputs, $state_i \oplus key_i$ and $state_j \oplus key_j$ with $i \neq j$. When these two inputs to the s-box are equal, it implies that $state_i \oplus state_j = key_i \oplus key_j$. This further implies that the difference between the partial inputs are equal to the difference between the subkeys.

Performing this analysis on several collisions representing different subkeys, CEPACA can significantly reduce the key space, although other methods is needed to reveal the actual cipher key.

Higher-Order DPA Attacks

Higher Order Differential Power Analysis (HO-DPA) attacks were introduced by Messerges[23], and seek to undermine the security in circuits using 1st. order masking. Typically, HO-DPA consists of probing multiple points in a circuit. Multiple samples within a power trace lets an adversary analyze the joint distribution between these measurements and target a known or hypothesized relationship between the parameters. dth.-order DPA can be successfully used to attack an implementation masked with d number of shares, however the complexity increases exponentially with increasing number of shares[7].

3.4 Prevention of DPA

Kocher [20] gives an overview of several countermeasuresemployed against differential power attacks, of which the ones relevant to AES will be summarized in the following section. Furthermore a more in depth coverage of the secret sharing scheme will be presented.

3.4.1 Protocol Level Countermeasures

The most straightforward way of securing AES, or any other block cipher against DPA attacks is through protocol level countermeasures. In other words, to restrict how many times a given key may be used before it is invalidated. This effectively thwarths an adversary's ability to gather enough samples to determine the secret key. As convenient as it may seem, several new problems arise such as secure key exchange and generation of secure cryptographic keys.

Introduction of Noise

DPA depends on statistical information of the power consumption in a circuit at specific points in time. This means that if additional noise into the system the collection of power samples usable becomes more difficult. Added noise can be in the form of amplitude noise, e.g. adding noisy power consuming devices on the chip, or it can be in the form of temporal noise. Conversely, one may seek to decrease the amount of leakage from the circuit to reduce the signal-to-noise ratio of the circuit. Temporal noise on the other hand seeks to disrupt an adversary's ability to perform measurements at predictable intervals, for example by randomizing order of operations or adding delays. Adding any noise makes the attack more difficult, but not impossible, given enough time and resources the adversary is still able to obtain the secret.

Logic Styles, CMOS Countermeasures

Last, it is possible to implement countermeasures on a physical level, by using glitch free logic and balanced functions. However, making sure that each module in a larger system has a balanced power consumption soon becomes an infeasible task. Adding physical shielding to the device may also be used in extremely security sensitive devices.

3.4.2 Masking

In this section the underlying theory behind secret sharing, requirements and proofs will be presented. Presently to our knowledge there are only two known AES-implementation methods assumed to be secure even in the presence of glitches, the methods by Nikova[26] and Prouff[29]. In this thesis we will mainly focus on the method by Nikova, which is based on the tower field approach[12], as it has lower hardware complexity. Masking, or secret sharing, attempts to conceal the intermediate data processed by the algorithm by modifying the data such that the complete secret is never processed simultaneously, and such that the power consumption is not directly correlated to the intermediate values.

Requirements

Nikova et al[26] postulated 3 requirements necessary for secure implementation of CMOS logic in the presence of glitches.

Property 1 (Correctness):

$$(Z_1, \ldots, Z_q) = f(X^1, \ldots, X^p) = \sum_{i=1}^{s} f_i(\bar{X}^1, \ldots, \bar{X}^p) \tag{3.4}$$

for all vectors of input shares $(\bar{X}^1, \ldots, \bar{X}^p)$ satisfying $\sum_{i=1}^{s} X_i^j = X^j$ with $1 \leq j \leq p$. In other words, the unshared vector function operating on the original data must be equivalent to a linear recombination of the shared vector function, thus preserving the intended functionality.

Property 2 (Non-completeness): Every function is independent of at least one share of each component. Denoting the reduced vector $(x_1^j, \ldots, x_{i-1}^j, x_{i+1}^j, \ldots, x_s^j)$ by $\bar{x}_i^j$. Requrire z_i to be independent of $x_i^j, \forall j$

$$\begin{aligned} z_1 &= f_1(\bar{x}_1^1, \bar{x}_1^2, \ldots, \bar{x}_1^p) \\ z_2 &= f_2(\bar{x}_2^1, \bar{x}_2^2, \ldots, \bar{x}_2^p) \\ &\ldots \\ z_s &= f_s(\bar{x}_s^1, \bar{x}_s^2, \ldots, \bar{x}_s^p) \end{aligned} \tag{3.5}$$

Property 3 (Uniformity): A realization of $(z^1, \ldots, z^q) = f(x^1, \ldots, x^p)$ is uniform if the distribution of the shares of the output satisfies:

$$P(\bar{z}^1 = \bar{Z}^1, \ldots, \bar{z}^p = \bar{Z}^p) = Q^{1-s} P(z^1 = \sum_{i=1}^{s} Z_i^1, \ldots, z^q = \sum_{i=1}^{s} Z_i^q) \tag{3.6}$$

In other words, consider the function f(x) = X, if for each x, each of the masked output X occur with the same probability.

Nikova further proposed the following theorems to show that circuits fulfilling the aforementioned properties are secure (The formal proofs are available in [26], and are omitted here for brevity):

- *Theorem 1*: If the masking **X** is uniform and the shared function **f** is non-complete, then any single component of **f** does not leak any information about X. That is, any single component in the system does not contain the information necessary to determine X. Intuitively, if a function does not know X, then it cannot leak X.
- *Theorem 2*: If the masking **X** is uniform and the circuit of **f** is non-complete, the expected value of the system leakage is constant.

Implementing Linear Transformations

Let $\mathscr{L}$ be a linear vector function where $(z^1, \ldots, z^q) = \mathscr{L}(x^1, \ldots, x^p)$. The easiest way to securely implement linear functions is by processing each share independently. and as such, each output share variable depends only on one input share of each variable. this yields an implementation that does not leak information usable by a Side-Channel Attack (SCA), even in the presence of glitches [21][26]. More formally, splitting each variable into n shares such that

$$\begin{aligned}(z_i^1,\ldots,z_i^q) =& \mathscr{L}(x_{i+1}^1,\ldots,x_{i+1}^p), 1 \leq i < n \\ (z_s^1,\ldots,z_s^q) =& \mathscr{L}(x_1^1,\ldots,x_1^p)\end{aligned} \tag{3.7}$$

By definition of the linearity property it follows that

$$\begin{aligned}(z^1,\ldots,z^q) =& \sum_{i=1}^{s}(z_i^1,\ldots,z_i^q) = \sum_{i=1}^{s}\mathscr{L}(x_i^1,\ldots,x_i^p) \\ =& \mathscr{L}\left(\sum_{i=1}^{s}(x_i^1,\ldots,x_i^p)\right) = \mathscr{L}(x^1,\ldots,x^p)\end{aligned} \tag{3.8}$$

Processing each share independently is relatively trivial to implement in hardware by either by duplicating the modules involved or serially processing the data. From [26] we have that the minimum number of shares required to implement a function that is a product of s variables is:

$$n \geq 1 + s \tag{3.9}$$

Thus the minimum number of shares necessary to implement the linear functions of AES is two. However, more shares may add to the protection of the circuit and/or reduce necessary amount of randomness.

Implementing Nonlinear Transformations

For the linear parts of the algorithm, it suffices to simply add a random data mask to the information to be concealed (i.e. by use of a simple XOR operation). Most operations in the AES-algorithm are linear - with the exception of operations in the S-box. Thus, implementation of the S-box is where the difficulty in efficiently implementing such a scheme lies.

Initially, several schemes were introduced, some also provably secure[31][9][17]. The main issue however is that they all base their formal proofs on an idealized hardware model and thus do not consider issues such as glitches. Glitches have been shown to introduce vulnerabilities into circuits, and cryptographic algorithms can be compromised with the aid of DPA and glitch-analysis.

Bilgin[6] proposed one solution adhering to the properties proposed by Nikova for the AES S-box. The solution uses a 3-share tower field implementation of the S-box in normal basis, with additional re-masking bits to ensure uniformity.

Implementing Cascaded and Parallel Functions

Implementing one function uniformly does not necessarily imply that the outputs of said function are uniform when taken as an input to another cascaded function. To ensure the uniformity in the joint distribution, Moradi [24] suggested using fresh random bits to mask the registers between the functions in the S-box.

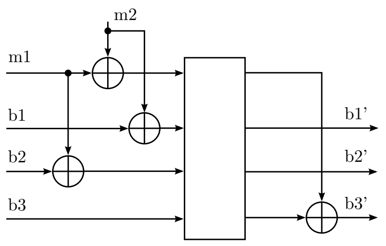


Figure 3.3: Register remasking as suggested by Moradi[24] for a 3-share case.

3.3 shows an example on how this may be done in practice. By adding the random masks m1 and m2 to the register as shown, uniformity can be ensured in the system. Similarly for parallel functions, the same can be applied to the output of one of the functions to ensure uniformity[5].

Chapter 4

Implementation

4.1 Overview of Proposed Unmasked Architecture

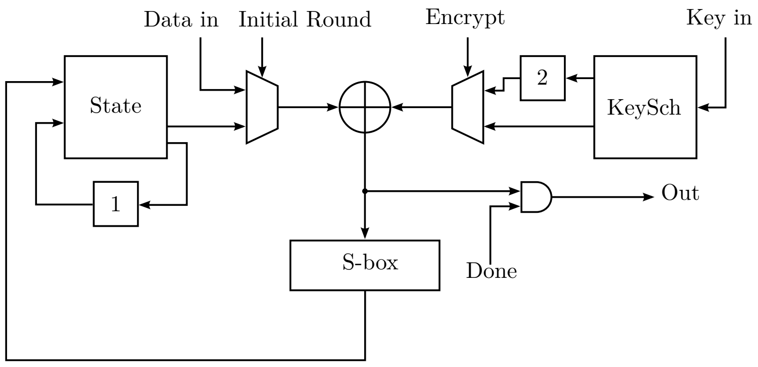


Figure 4.1: Basic structure of unmasked AES implementation.

Figure 4.1 shows a basic overview of the proposed AES core architecture. KeySch represents the key scheduling routine, state is the intermediate state-register keeping the state between rounds. Module 1 is the combined ShiftRows and MixColumns modules while 2 is an inverse MixColumns module for use during decryption. In the following sections the various parts of the architecture will be introduced and at the end of the chapter the masked version will be discussed. We chose to implement a 128 bit data-path, with a two register pipelined s-box such that one AES-round is performed per three cycles.

4.2 Control Logic Modules

This section describes the individual modules necessary to control the data flow in the proposed implementation. The control modules are identical for the masked and the unmasked implementation as they do not operate on secret data directly.

4.2.1 Clock Divider

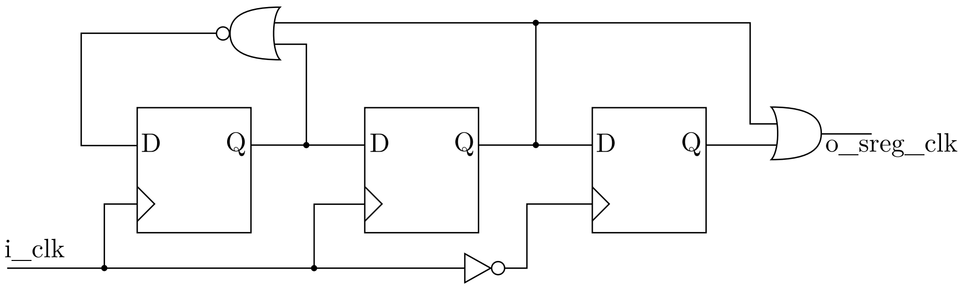


Figure 4.2: 1:3 Clock Divider Circuit

As the proposed S-box implementations uses two pipeline registers, and our state-logic only consists of one register, there is a minimum delay through one round of three clock cycles. To avoid complicating the Finite-State Machine (FSM) and adding additional multiplexers to the system, an approach with a 1:3 clock divider was chosen, where the state logic (MixColumns, ShiftRows, AddRoundKey) operates on a clock (i_sreg_clk) that runs slower than the system clock (i_clk) and thus process one round in one cycle. The S-box however still runs on the fast system clock. The clock divider is designed by placing two Flip-Flop (FF) in series with an NOR-gate, creating a 1:3 divider with 33.3% duty cycle, followed by another negative edge triggered FF ORed with the output of the second FF. This gives a compact 50% duty cycle, 1:3 clock divider.

4.2.2 Clock Synchronizer

When the system operates on two different clock frequencies, it is essential that the signals that operates across clock domains are synchronized. In this case, it is essential that the stable-signal from the key expansion is matched to that of the slow system clock. This is implemented with a FF triggered on the rising edge of the state-reg clock signal.

4.2.3 Main Control FSM

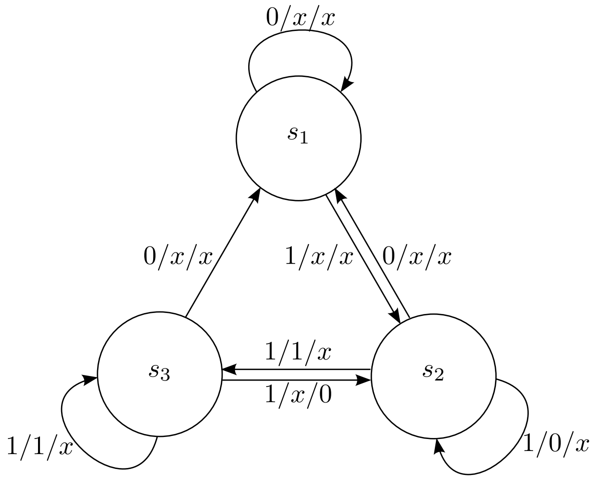


Figure 4.3: Finite State Machine - S_1: Wait for key, S_2: Wait for data, S_3: Do rounds, Signals (A/B/C): A: key stable, B: Input Data Ready, C: Rounds Executing

The main control FSM of our system consists of three states as shown in figure 4.3. The figure shows the main control signals, however the fully implemented system requires some additional synchronization signals that are not included here for brevity. The included signals however do represent the necessary conditions for state transition.

The FSM operates with two different clocks, for the wait_key state we operate with the regular system clock, and for the remaining two states the i_sreg_clk is used.

4.2.4 Round Counter

The round consists of a counter counting from 0 to either 10, 12 or 14 depending on the current operating mode. Additionally it outputs a signal, **flf**, representing first, last or intermediate round to control the part of the algorithm that differs in the first and last rounds in addition to controlling the exit from the do rounds part of the FSM. Initial synthesis of the counter module showed some problems with glitches during AES-256 mode, this was solved by converting the counter architecture to a state machine methodology, using synchronous outputs.

4.3 Key Expansion

AES suggests two main methods of implementing the key expansion algorithm; on-the fly generation of round keys or precomputing the round keys prior to executing the algorithm[1]. For the encryption process the round keys are required in the ascending order, which caters well to the iterative key-generation process. Decryption on the other hand faces the problem of requiring the keys in the reverse order. Thus if an on the fly key-generator is used, we would either need to store the keys in advance, or first generate the keys in the correct order to obtain the last roundkey, whereafter the last round-key forms the initial key for a modified expansion routine.

Second, expanding a round key from the original cipher key during the execution of every single round adds another pontential probing point for a side channel attack. This is due to the key expansion itself leaking secret information, and thus would also need to be masked. This would also call for a the need of a preshared key such that the complete secret is never repeatedly processed by the system. Based on the aformentioned, pre-computational approach with masked storage was selected for this paper. In this chapter the key expansion module itself will be explained, while the storage part and masking is considered in section 4.4 and 4.9 respectively.

Note from algorithm 1 that each round key depends on the previous and the Nk previous key, specifically

$$w[i] = f(w[i - Nk], w[i - 1]) \tag{4.1}$$

where $f(a, b)$ can take four possible forms, where a and b are 32-bit words.

- Pass-through mode, a is passed on to the output. This happens during the initial round (and second round in the case of AES-192/256). With some modifications to the key expansion, we may bypass this stage entirely, thus removing the need to implement this into the module.
- Rotate-SubByte-Rcon mode, involving a circular shift, an s-box substitution and an addition of a round constant to b followed by an XOR of a.
- Subbytes mode, involving an s-box substitution of b during AES-256 key expansion followed by an XOR of a.
- XOR-mode, performing an XOR of the two input words.

Considering that operations are performed modulo Nk, it is convenient to represent the operations for the three modes of operation as follows:

- AES-128: RXXX

- AES-192: RXXXXX
- AES-256: RXXXSXXX

AES defines the round-key size to be equivalent to the state-matrix, 128-bits (4×32-bit words), mapping the aformentioned representation to each round of 128-bits gives the following table:

Table 4.1: Key Expansion Rounds, P: Passthrough, R: Rotate-SubByte-Rcon (RSR), X: XOR, S: Subbyte

Round #	**AES-128**	**AES-192**	**AES-256**
0	P-P-P-P	P-P-P-P	P-P-P-P
1	R-X-X-X	P-P-R-X	P-P-P-P
2	R-X-X-X	X-X-X-X	R-X-X-X
3	R-X-X-X	R-X-X-X	S-X-X-X
4	R-X-X-X	X-X-R-X	R-X-X-X
5	R-X-X-X	X-X-X-X	S-X-X-X
6	R-X-X-X	R-X-X-X	R-X-X-X
7	R-X-X-X	X-X-R-X	S-X-X-X
8	R-X-X-X	X-X-X-X	R-X-X-X
9	R-X-X-X	R-X-X-X	S-X-X-X
10	R-X-X-X	X-X-R-X	R-X-X-X
11		X-X-X-X	S-X-X-X
12		R-X-X-X	R-X-X-X
13			S-X-X-X
14			R-X-X-X

From table 4.1 it is apparent that each round key contains one and only one SubByte or Rotate-SubByte-XOR operation, the rest of the operations are XORs. By directing the correct input to the correct module each round, the current roundkey can be properly generated.

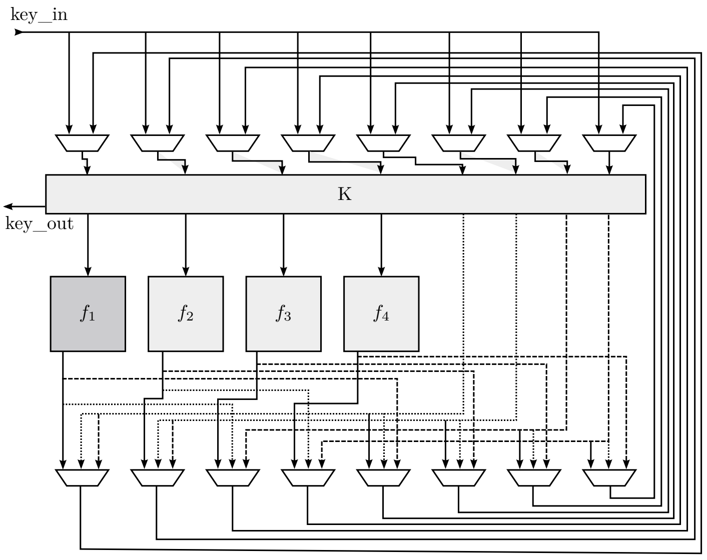


Figure 4.4: Key expansion

It is apparent from table 4.1 that it possible to use four functional units $f(a, b)$ to map this representation to hardware, as shown in figure 4.4. The leftmost unit needs to be capable of rotate-sub-rcon-mode, while the three others only needs to support XOR and pass-through. With an iterative 4-cycle generation of keys it is also possible to make do with only one common module, but to keep the possibility of modifying the structure to fit a one key per cycle we opted to keep using four modules. To further simplify the logic, the initial round does not need to pass through the functional units, and can be passed directly to the output. The register K in figure 4.4 contains the current round key, and through a cycle of four rounds calculates the consecutive-words, iteratively overwriting the previous word. Whenever the last word is written to the input register, the data in the input register is sent to the output together with the round counter.

Three counters are needed for the control logic, to represent the current round, the position in the Nk-word array and to represent the number of Nk-word strings processed. Furthermore the control logic must set the correct inputs and outputs to the functional modules each cycle. For AES-128 this corresponds to a direct 1-to-1 mapping between the outputs and inputs. AES-192 requires some more modification, by mapping inputs and outputs as shown in figure 4.4 by the dotted lines. In 256 bit mode the key expansion proceeds similarly as AES-128, but the the key is split in two 128-bit chunks. The first chunk is processed with the same logic as AES-128, while the second chunk replaces the RSR-operation with an S-box substitution instead. In figure 4.4 the AES-256 data-path is represented by the dashed lines.

4.4 Key Registers

The basic key register module covers the roundkey selection function and consists of a 128x15 bit register, one for each round key. According to the AES standard [1] it is possible to modify the decryption algorithm to perform the necessary operations in the same order as for encryption. This is accomplished by modifying the key scheduling routine with the addition of a conditional inverse MixColumns operation on the given roundkey.

In the AES standard, the decryption and encryption operation the order of operations differ, however it is mentioned that by modifying the key scheduling routine To be able to use the same hardware structure for decryption and encryption operation as mentioned by [1] a conditional inverse MixColumns module is connected to each of the outputs.

4.5 Core Interconnect

To cover the functions involved in input and output to the system, in addition to the AddRoundKey function, the proposed system uses a module which is named core_interconnect. This module covers everything between the state-module, the key registers and the s-box.

4.6 S-Box

Figures 4.5 and 4.6 show the basic structure for the unmasked s-box implementations. These were created to serve as a reference for performance and area compared to the implementations using secret sharing. The A^{-1} and A blocks represent the inverse affine and affine transformations as specified by the AES standard. I and I^{-1} are the isomorphisms converting the standard binary representation of AES to and from the polynomial or normal basis representations. The modules located between the I modules form the GF(2^8) inverter, which consists of a GF(2^4) square-scaler module ($N \otimes x^2$), two pipeline registers (gray bars), a GF(2^4) inverter (x^{-1}) and three multipliers ($\otimes$). Addition ($\oplus$) is simply a bitwise xor of the incoming signals. Thin arrows represent 4 bit signals while bold arrows represent 8 bits. The following subsections will elaborate on these modules in more detail.

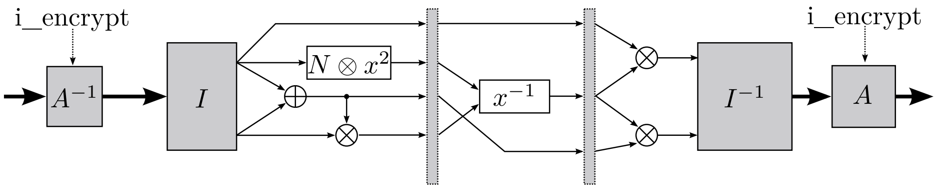


Figure 4.5: Structure of suggested non-masked polynomial basis S-box implementation

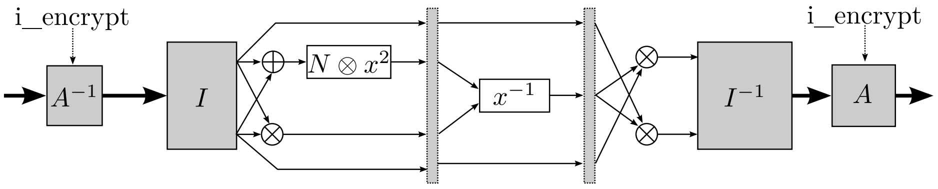


Figure 4.6: Structure of suggested non-masked normal basis S-box implementation

4.6.1 Affine Transformation and Isomorphism

For implementation of the affine transformation and the isomorphism two possible approaches were considered. One would be to combine the two transforms - in an encryption- or decryption only system this would be the most efficient, as it saves area as only two transforms are needed; one at the input and one at the output of the multiplicative inverse module. For the dual-mode system however, the choice of representation is not as clear. A combination of the two modules on for example the input, would require a mux and another transform module only including the isomorphism for encryption mode as well. The isomorphisms used are given in appendix, normal and polynomial basis representations require different isomorphisms.

4.6.2 Notation in Normal and Polynomial Basis Calculations

In the following sections, the mathematical derivation on some of the modules will be shown, and thus some explanation of the notation is necessary. Greek lowercase letters, γ, δ represents numbers in GF(2^4), Greek uppercase letters, Γ and Δ represent numbers in GF(2^2) and Latin lowercase letters represents individual bits in GF(2). z represents the position of the upper two bits in GF(2^4) for polynomial basis, equivalently in normal basis this is represented by Z^4. The lower two bits in polynomial basis are without such notation, while in normal basis Z is used. ω represents the position of the upper bit in GF(2^2).

4.6.3 GF(2^4) Square-Scaler

The square scaler modules perform scaling by a factor ν in GF(2^4) a squaring of the result, and is optimized by creating a dedicated fixed multiplier module combined with a squarer.

Polynomial Basis

In general, for squaring and multiplication by a constant in a polynomial extension field GF(2^8)/GF(2^4) represented by $r(y) = y^2 + \tau y + \nu$ with $\tau = 1$, we have the following:

$$\begin{aligned}\nu \otimes \gamma^2 =& \nu \otimes (\Gamma_1 z + \Gamma_0) \otimes (\Gamma_1 z + \Gamma_0)\\ =&\nu \otimes (\Gamma_1^2 z + \Gamma_0^2 + N\Gamma_1^2)\\ =&(\Delta_1 z + \Delta_0) \otimes (\Gamma_1^2 z + \Gamma_0^2 + N\Gamma_1^2)\end{aligned} \tag{4.2}$$

Equation 4.2 becomes less complex if Δ_0 of ν is set equal to zero (It is not possible to set Δ_1 to zero as this would lead to a reducible polynomial). Furthermore, $N \neq 0, 1$ since this would lead to an irreducible polynomial over GF(2^2) Thus we have the option of selecting $N = \omega$ or $N^2 = \omega$. Note here that this implies $N^2 = N + 1$. Selecting $N = \omega$ and replacing $\Delta_1 z$ in ν by $N^2 z$:

$$\begin{aligned}\nu \otimes \gamma^2 =& N^2 z \otimes (\Gamma_1^2 z + \Gamma_0^2 + N \otimes \Gamma_1^2)\\ =&(N^2 \otimes (\Gamma_1^2 + \Gamma_0^2 + N^2\Gamma_0^2))z + \Gamma_1^2\\ =&((N^2 + 1) \otimes \Gamma_1^2 + N^2 \otimes \Gamma_0^2)z + \gamma_1^2\\ =&(N\Gamma_1^2 + N^2\Gamma_0^2)z + \Gamma_1^2\end{aligned} \tag{4.3}$$

To realize the structure in equation 4.3 it is necessary to calculate multiplications by N and N^2 respectively in addition to the squaring of the higher and lower bits.

$$\begin{aligned}\Gamma^2 =&(a\omega + b)^2 = a\omega + (a + b)\\ N\Gamma^2 =&\omega(a\omega + (a + b)) = a(\omega + 1) + a\omega + b\omega = b\omega + a\\ N^2\Gamma^2 =&\omega^2(a\omega + (a + b)) = (a + b)\omega + b\end{aligned} \tag{4.4}$$

Inserting 4.4 into 4.3, the Algebraic Normal Form (ANF) for the polynomial square scaler becomes

$$\begin{aligned} b'3 =& b_2 \oplus b_1 \oplus b_0 \\ b'2 =& b_3 \oplus b_0 \\ b'1 =& b_3 \\ b'0 =& b_3 \oplus b_2 \end{aligned} \tag{4.5}$$

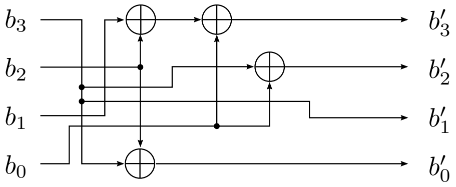


Figure 4.7: Polynomial Basis Square-Scaler

Normal Basis

Note that in polynomial basis $(Z^4)Z = N$ and $T = Z^4 + Z$.

$$\begin{aligned} \nu \otimes \gamma^2 =& \nu \otimes \{(\Gamma_1 Z^4 + \Gamma_0 Z) \otimes (\Gamma_1 Z^4 + \Gamma_0 Z)\} \\ =& \nu \otimes \{\Gamma_1^2 (Z^4)^2 + \Gamma_0^2 (Z)^2\} \\ =& \nu \otimes \{\Gamma_1^2 Z + \Gamma_1^2 N + \Gamma_0^2 Z + \Gamma_0^2 N\} \\ =& (\Delta_1 Z^4 + \Delta_0 Z) \otimes \{(\Gamma_1^2 + N \otimes (\Gamma_1^2 + \Gamma_0^2))Z^4 + \\ & (\Gamma_0^2 + N \otimes (\Gamma_1^2 + \Gamma_0^2))Z\} \\ =& \Delta_1 (Z^4 + N)(\Gamma_1^2 + N \otimes (\Gamma_1^2 + \Gamma_0^2)) + \Delta_1 N(\Gamma_0^2 + N \otimes (\Gamma_1 + \Gamma_0^2)) \\ & + \Delta_0 (Z + N)(\Gamma_0^2 + N \otimes (\Gamma_1^2 + \Gamma_0^2)) + \Delta_0 N(\Gamma_1^2 + N \otimes (\Gamma_1 + \Gamma_0^2)) \\ =& \{(\Delta_1 + N\Delta_0)\Gamma_1^2 + N\Delta_0 \otimes \Gamma_0^2\} Z^4 + \\ & \{(\Delta_0 + N\Delta_1)\Gamma_0^2 + N\Delta_1 \otimes \Gamma_1^2\} Z \end{aligned} \tag{4.6}$$

Assuming[1] that Bilgin et al.[8] used $N = \omega^2$ for the given formulas 4.17 and 4.13, and suggested as a convenient N by Canright [12] the higher bits of ν, Δ_1 is equal to $\{00\}$ and disappears, thus the above simplifies to

$$\begin{aligned} & (N\Delta_0\Gamma_1^2 + N\Delta_0\Gamma_0^2)Z^4 + \Delta_0\Gamma_0^2 Z = \\ & (N^3\Gamma_1^2 + N^3\Gamma_0^2)Z^4 + N^2\Gamma_0^2 = \\ & (\Gamma_1 + \Gamma_0)^2 Z^4 + N^2\Gamma_0^2 Z \end{aligned} \tag{4.7}$$

[1]Simulation verified this assumption to be correct.

Equation 4.7 shows that the square scaling module can be represented by addition of the upper and lower bits followed by a squaring module for the high output bits. The low output bits are equal to the low input bits multiplied by N and squared. From section 2.4.4 we have that squaring in Normal basis equals a rotation of the elements, thus for the two bits in equation 4.7 it equals a bit swap and can be implemented for free in the circuit. From 4.7 we have that the lower two bits equal $N^2\Gamma_0^2$, where $N = \omega^2$. Name the two bits a and b and note that $(a\omega^2 + b\omega)^2 = b\omega^2 + a\omega$, the expression then simplifies to

$$\begin{aligned} N^2\Gamma_0^2 =& \omega(b\omega^2 + a\omega) = b\omega^3 + a\omega^2 \\ =& (a+b)\omega^2 + b\omega \end{aligned} \tag{4.8}$$

The ANF for the square scaler thus equals the formula given in 4.9 and as shown in figure 4.8.

$$\begin{aligned} b'_3 =& b_2 \oplus b_0 \\ b'_2 =& b_3 \oplus b_1 \\ b'_1 =& b_1 \oplus b_0 \\ b'_0 =& b_0 \end{aligned} \tag{4.9}$$

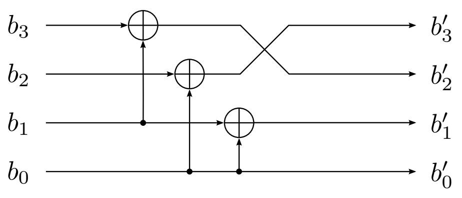


Figure 4.8: Normal Basis Square-Scaler

4.6.4 GF(2^4) Multiplier

Polynomial Basis

The following shows how to find the representation for the product of two numbers over GF(2^4) in polynomial basis:

$$\begin{aligned} \gamma\delta =& (\Delta_1 z + \Delta_0) \otimes (\Gamma_1 z + \Gamma_0) \\ =& \Gamma_1\Delta_1 z^2 + \Gamma_1\Delta_0 z + \Gamma_0\Delta_1 z + \Gamma_0\Delta_0 \\ =& (\Gamma_1\Delta_1 + \Gamma_1\Delta_0 + \Gamma_0\Delta_1)z + (\Gamma_0\Delta_0 + N\Gamma_1\Delta_1) \end{aligned} \tag{4.10}$$

Then for the operations in GF(2^2) we have

$$\begin{aligned}
\Gamma\Delta &= (a\omega + b) \otimes (c\omega + d) = ac(\omega + 1) + ad\omega + bc\omega + bd \\
&= (ac + ad + bc)\omega + (ac + bd) \\
N\Gamma\Delta &= (ac + ad + bc)(\omega + 1) + (ac + bd)\omega \\
&= (ad + bc + bd)\omega + (ac + ad + bc)
\end{aligned} \tag{4.11}$$

Inserting 4.11 into 4.10 and solving for the individual bits we get the ANF for the multiplier as follows:

$$\begin{aligned}
(Z_3, Z_2, Z_1, Z_0) =&(X_3, X_2, X_1, X_0) \otimes (Y_3, Y_2, Y_1, Y_0) \\
Z_3 =&X_3Y_3 \oplus X_3Y_2 \oplus X_3Y_1 \oplus X_3Y_0 \oplus X_2Y_3 \oplus X_2Y_1 \\
&\oplus X_1Y_3 \oplus X_1Y_2 \oplus X_0Y_3 \\
Z_2 =&X_3Y_3 \oplus X_3Y_1 \oplus X_2Y_2 \oplus X_2Y_0 \oplus X_1Y_3 \oplus X_0Y_2 \\
Z_1 =&X_3Y_2 \oplus X_2Y_3 \oplus X_2Y_2 \oplus X_1Y_1 \oplus X_1Y_0 \oplus X_0Y_1 \\
Z_0 =&X_3Y_3 \oplus X_3Y_2 \oplus X_2Y_3 \oplus X_1Y_1 \oplus X_0Y_0
\end{aligned} \tag{4.12}$$

Normal Basis

For the normal basis, the structure suggested by Bilgin[6] was used, its ANF is given in equation 4.13

$$\begin{aligned}
Z_3 =&X_3Y_3 \oplus X_1Y_3 \oplus X_0Y_3 \oplus X_2Y_2 \oplus X_1Y_2 \oplus X_3Y_1 \oplus X_2Y_1 \oplus X_1Y_1 \oplus X_0Y_1 \oplus \\
&X_3Y_0 \oplus X_1Y_0 \\
Z_2 =&X_2Y_3 \oplus X_1Y_3 \oplus X_3Y_2 \oplus X_2Y_2 \oplus X_0Y_2 \oplus X_3Y_1 \oplus X_1Y_1 \oplus X_2Y_0 \oplus X_0Y_0 \\
Z_1 =&X_3Y_3 \oplus X_2Y_3 \oplus X_1Y_3 \oplus X_0Y_3 \oplus X_3Y_2 \oplus X_1Y_2 \oplus X_3Y_1 \oplus X_2Y_1 \oplus X_1Y_1 \oplus \\
&X_3Y_0 \oplus X_0Y_0 \\
Z_0 =&X_3Y_3 \oplus X_1Y_3 \oplus X_2Y_2 \oplus X_0Y_2 \oplus X_3Y_1 \oplus X_0Y_1 \oplus X_2Y_0 \oplus X_1Y_0 \oplus X_0Y_0
\end{aligned} \tag{4.13}$$

4.6.5 GF(2^4) Inverter

The two first implementations of the GF(2^4) inverters presented here serves as reference for the following masked implementations.

Polynomial Basis

The method for deducing the GF(2^4) inverter proceeds similarly as to the polynomial GF(2^8) inverter described in section 2.4.4, with the trace over GF(2^4)/GF(2^2) $= T = 1$ and the norm, $N = \omega$.

$$\begin{aligned}\gamma\delta =&(\Gamma_1 z+\Gamma_0)(\Delta_1 z+\Delta_0) = 0z+1\\ &\Rightarrow\\ \Delta_1 =&(\Gamma_1^2 N+\Gamma_1\Gamma_0+\Gamma_0^2)^{-1}\Gamma_1\\ \Delta_0 =&(\Gamma_1^2 N+\Gamma_1\Gamma_0+\Gamma_0^2)^{-1}(\Gamma_0+\Gamma_1)\end{aligned}\tag{4.14}$$

We then proceed by finding the inverse in GF(2^2)

$$\begin{aligned}\Gamma\Delta =&(g_1\omega+g_0)(d_1\omega+d_0) = 0\omega+1\\ &\Rightarrow\\ d_1 =&(g_1^2+g_1g_0+g_0^2)^{-1}g_1\\ d_0 =&(g_1^2+g_1g_0+g_0^2)^{-1}(g_1+g_0)\end{aligned}\tag{4.15}$$

in GF(2), an inversion is simply a flip of bit, which again is the same as squaring. Considering this and combine the equations above we get the following ANF for the GF(2^4) inverter:

$$\begin{aligned}(Y_3,Y_2,Y_1,Y_0) =&Inv(X_3,X_2,X_1,X_0)\\ Y_3 =&X_3\oplus X_2\oplus X_3X_0\oplus X_3X_2X_1\\ Y_2 =&X_2\oplus X_3X_0\oplus X_2X_1\oplus X_3X_2X_0\oplus X_3X_2X_1\\ Y_1 =&X_3\oplus X_2\oplus X_1\oplus X_2X_0\oplus X_3X_1X_0\oplus X_3X_2X_1\\ Y_0 =&X_2\oplus X_1\oplus X_0\oplus X_2X_1\oplus X_3X_0\oplus X_3X_1\\ &\oplus X_2X_1X_0\oplus X_3X_1X_0\oplus X_3X_2X_0\oplus X_3X_2X_1\end{aligned}\tag{4.16}$$

Normal Basis

For the normal basis implementation, the proposed structure by Bilgin et al. [6] was used. its functionality in ANF given in equation 4.17.

$$\begin{aligned}(Y_3,Y_2,Y_1,Y_0) =&Inv(X_3,X_2,X_1,X_0)\\ Y_3 =&X_1\oplus X_0\oplus X_3X_1\oplus X_2X_1\oplus X_2X_1X_0\\ Y_2 =&X_0\oplus X_3X_1\oplus X_2X_1\oplus X_2X_0\oplus X_3X_1X_0\\ Y_1 =&X_3\oplus X_2\oplus X_3X_1\oplus X_3X_0\oplus X_3X_2X_0\\ Y_0 =&X_2\oplus X_3X_1\oplus X_3X_0\oplus X_2X_0\oplus X_3X_2X_1\end{aligned}\tag{4.17}$$

According to [7] both the aforementioned implementations belong to the same class of substitution boxes and can be shared without decomposition using a 5-input 5-output sharing scheme.

4.7 ShiftRows

The operations in ShiftRows and inverse ShiftRows only involves a transposition of elements in the state and it can be implemented by wiring, i.e. without any cost penalty. However, combining these into one module requires the addition of a multiplexer to select which wires are used in the respective operations.

4.8 MixColumns

Operation in the MixColumns module is based on polynomial multiplication on columns of the state. As shown earlier, the multiplication in the MixColumns module only involves factors 2 and 3. A multiplication by two in the Galois field is equivalent to a left shift of one bit and a conditional addition of the root of the characteristic field-polynomial. The inverse MixColumns module however requires multiplications by 9, 11, 13 and 15 complicating matters substantially. Initially, the inverse module was implemented using a look-up table for the multiplication operations, however this gave very little control over the actual arcitechture deduced by the synthesis tool. Zhang [36] proposed a more efficient VLSI-architecture that enabled a combination of MixColumns and inverse MixColumns into one module and by utilizing substructure sharing only the multiplications by 2 and 4 are needed.

For MixColumns we have:

$$\begin{aligned}
S'_0 &= \{02\} \otimes (S_0 \oplus S_1) \oplus (S_2 \oplus S_3) \oplus S_1 \\
S'_1 &= \{02\} \otimes (S_1 \oplus S_2) \oplus (S_3 \oplus S_0) \oplus S_2 \\
S'_2 &= \{02\} \otimes (S_2 \oplus S_3) \oplus (S_0 \oplus S_1) \oplus S_3 \\
S'_3 &= \{02\} \otimes (S_3 \oplus S_0) \oplus (S_1 \oplus S_2) \oplus S_0
\end{aligned} \tag{4.18}$$

and for inverse MixColumns:

$$\begin{aligned}
S'_0 =&(\{02\} \otimes (S_0 \oplus S_1) \oplus (S_2 \oplus S_3) \oplus S_1)\\
&\oplus(\{02\} \otimes (\{04\} \otimes (S_0 \oplus S_2) \oplus \{04\} \otimes (S_1 \oplus S_3))\\
&\oplus\{04\} \otimes (S_0 \oplus S_2))\\
S'_1 =&(\{02\} \otimes (S_1 \oplus S_2) \oplus (S_3 \oplus S_0) \oplus S_2)\\
&\oplus(\{02\} \otimes (\{04\} \otimes (S_0 \oplus S_2) \oplus \{04\} \otimes (S_1 \oplus S_3))\\
&\oplus\{04\} \otimes (S_1 \oplus S_3))\\
S'_2 =&(\{02\} \otimes (S_2 \oplus S_3) \oplus (S_0 \oplus S_1) \oplus S_3)\\
&\oplus(\{02\} \otimes (\{04\} \otimes (S_0 \oplus S_2) \oplus \{04\} \otimes (S_1 \oplus S_3))\\
&\oplus\{04\} \otimes (S_0 \oplus S_2))\\
S'_3 =&(\{02\} \otimes (S_3 \oplus S_0) \oplus (S_1 \oplus S_2) \oplus S_0)\\
&\oplus(\{02\} \otimes (\{04\} \otimes (S_0 \oplus S_2) \oplus \{04\} \otimes (S_1 \oplus S_3))\\
&\oplus\{04\} \otimes (S_1 \oplus S_3))
\end{aligned} \tag{4.19}$$

As can be seen from the formulas above, MixColumns constitutes the first part of the inverse MixColumns module. During AES operation these two modules are never processing data simultaneously and can be combined into one module with a conditional output.

Multiplication by two in the GF(2^8) field is given by

$$\begin{aligned}
\{02\} \otimes S = xS =& s_7x^8 + s_6x^7 + s_5x^6 + s_4x^5 + s_3x^4\\
&+ s_2x^3 + s_1x^2 + s_0x \mod p(x)
\end{aligned} \tag{4.20}$$

$p(x)$ is the root of the polynomial $x^8 + x^4 + x^3 + x + 1$, i.e. $p(x) = x^4 + x^3 + x + 1$ and thus every instance of x^8 in the above formula should be replaced by $p(x)$ yielding

$$\begin{aligned}
\{02\} \otimes S = & s_6x^7 + s_5x^6 + s_4x^5 + (s_3 + s_7)x^4\\
&+(s_2 + s_7)x^3 + s_1x^2 + (s_0 + s_7)x + s_7
\end{aligned} \tag{4.21}$$

Similarily for multiplication by 4:

$$\begin{aligned}
\{04\} \otimes S =& x^2S = s_7x^9 + s_6x^8 + s_5x^7 + s_4x^6\\
&+s_3x^5 + s_2x^4 + s_1x^3 + s_0x^2 \mod p(x)\\
&=s_5x^7 + s_4x^6 + (s_3 + s_7)x^5 + (s_2 + (s_6 + s_7))x^4\\
&+(s_1 + s_6)x^3 + (s_0 + s_7)x^2 + (s_6 + s_7)x + s_6
\end{aligned} \tag{4.22}$$

Combining the above calculations leads to the implementation as shown in figure 4.9.

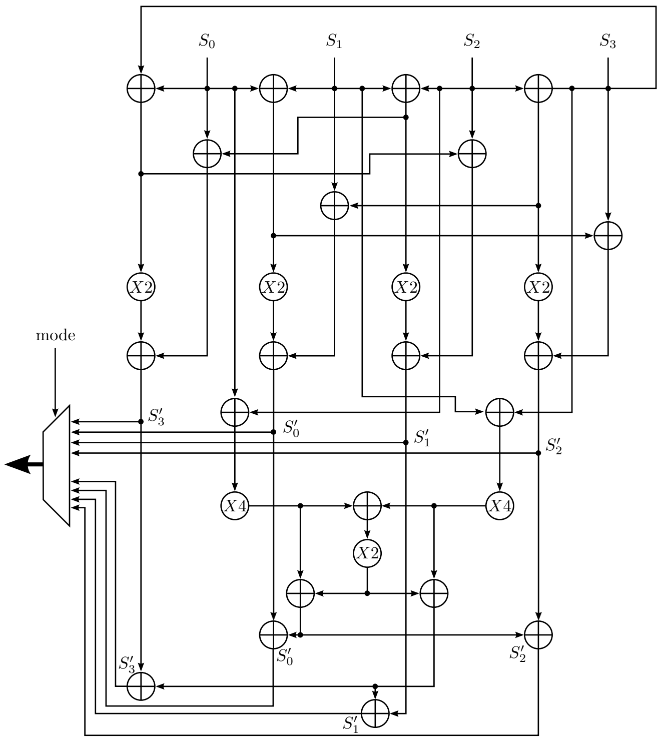


Figure 4.9: Modified (Inv)MixColums implementation based on the architecture proposed by Zhang[36].

4.9 Masking

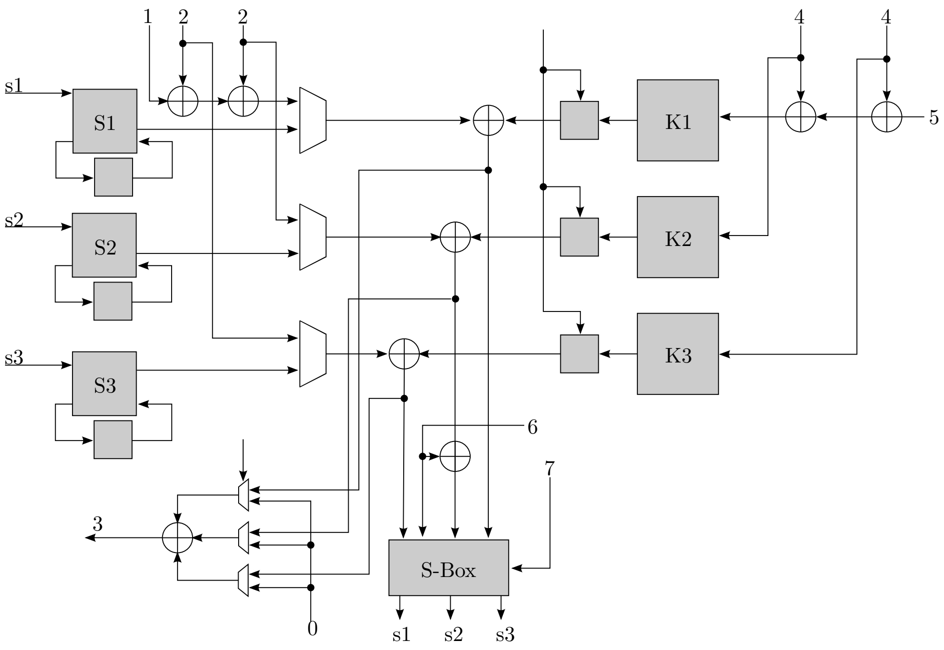


Figure 4.10: Masking overview of linear components

As stated by formula 3.9, a minimum of two shares is necessary to implement linear functions securely to prevent a first order DPA attack. Increasing the number of shares increases the difficulty of a DPA attack exponentially, but also incurs a large increase in area consumption. Masking schemes require the addition of a certain amount of cryptographically secure random data. The cost of implementing such circuitry will also be proportional to the amount of random bits needed. Therefore, not only the direct cost of our implementation must be taken into consideration, but a balance between area and need of randomness must also be considered. In this thesis a 3-share method have been used on all components in the data-path. This should create resistance against 2nd. order DPA. Figure 4.10 gives an overview of the linear component masking. Each linear component in the data-path is duplicated 3 times. S1, S2 and S3 represent the state registers, each with their own combined mixcolumn-shiftrows module. K1, K2 and K3 are the shared 15x128 bit round key registers. Round keys arrive at input 5 and are masked by two random masks at input 4. Similarly, the plaintext arrives at input 1, and is masked upon input by two new random masks arriving at input 2. When encryption is complete, the three masks are combined to form the correct ciphertext output at output 3.

The S-box takes a four-share input, and outputs three shares, thus an additional 128 bit block of random data is necessary at input 6 during each round. The additional bits needed for re-masking are added at input 7.

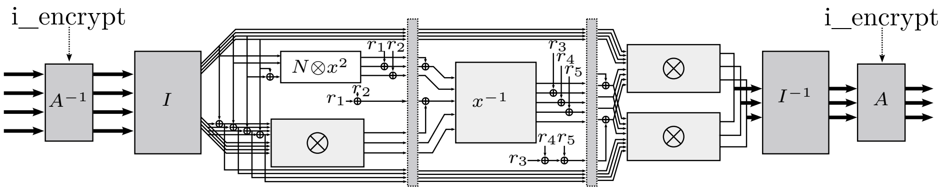


Figure 4.11: Structure of suggested masked polynomial basis S-box implementation

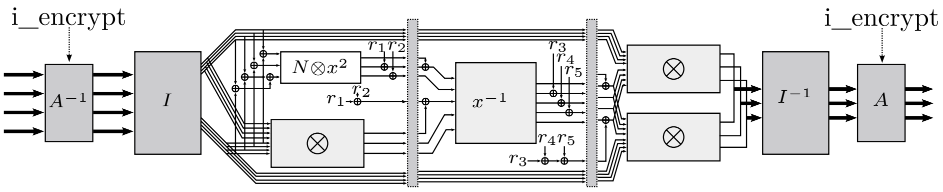


Figure 4.12: Structure of suggested masked normal basis S-box implementation

Figures 4.12 and 4.11 shows the masked implementations of the normal and polynomial basis S-boxes respectively. The linear parts, i.e. the affine transformations and isomorphisms on each end of the inverter and the square scaler, are implemented by duplicating the non-shared structures with one slight alteration. The affine transformation modules contains a constant that should only be added once, thus it is only added to one of the shares.

Sharing of the non-linear parts, the inverted and the multipliers, uses the uniform sharing functions proposed by Bilgin[6]. For the inverter, a 5-input, 5-output sharing is used, which is the smallest possible uniform sharing for the inverter without decomposition. Similarly, the multiplier uses a 4-input, 3-output sharing, again the smallest possible uniform sharing implementation. ANF for the sharing and their realizations are given in appendix.

Additionally, random bits r^1-r^5 are applied at the pipeline register in each stage to ensure that the inputs to the next stage is uniformly distributed.

The single byte version of the normal basis S-box used in this thesis was tested by Bilgin et al. and with 10 million power traces they were still unable to attain knowledge of the key using DPA-attacks. Our proposed implementation uses 16 of these single byte S-boxes in parallel, which should increase the amount of noise in the system and further complicating attacks.

4.10 Cipher Block Chaining

The AES-algorithm on its own operates in ECB mode, as mentioned in section 2.5 this is not a secure solution. Cipher-Block Chaining (CBC) is one of the modes of operation recommended by FIPS[15], and its lightweight structure lend itself well to our suggested implementation.

Figure 4.13 shows an overview of the entire suggested architecture including CBC. Note first that we assume that the ciphertext is not secret, and is known to a possible adversary. To avoid any direct correlation with the plaintext data, the suggested encryption mode implementation adds the ciphertext to one of the shares that holds one of the random state-masks, and not the one containing the original masked data. Similarily, the decryption mode adds the ciphertext together with the shares prior to outputting the decrypted data. Encryption uses one 128 bit register to hold the previous ciphertext until a new encryption round is started. Decryption on the other hand needs an additional register as both the previous ciphertext c_{n-1} and the present ciphertext c_n needs to be stored in registers 1 and 2 respectively. Upon completion of one round, c_n is shifted into register 1 and the next ciphertext c_{n+1} is placed in register 2.

4.11 The Complete Masked Core Architecture

Figure 4.13 gives a complete overview of the proposed masked core architecture including CBC. s_reg_1 through 3 represent the state registers, mcsr represents the combined ShiftRows and MixColumn modules and imc represents the InverseMixColumn used in decryption. Key registers are denoted by k_reg_1 through 3. Control logic and key expansion modules are not shown.

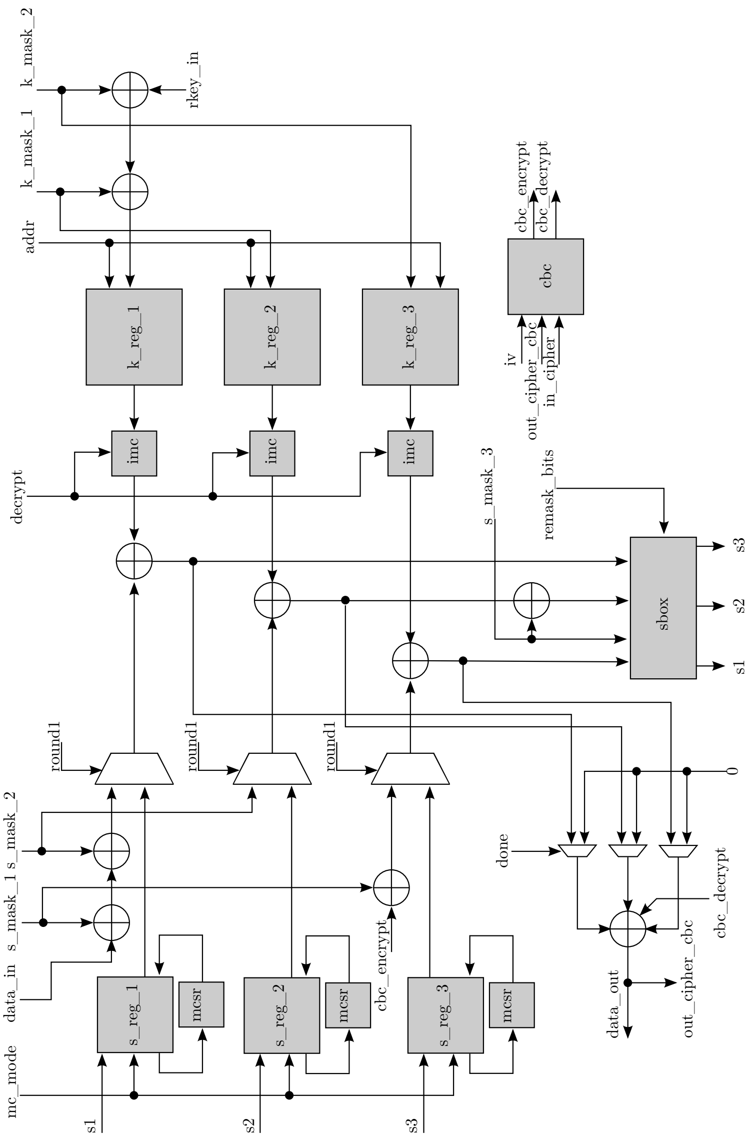


Figure 4.13: AES-core implementation

Chapter 5

Test Methodology & Results

5.1 Software Setup

The various implementations were implemented using SystemVerilog and simulated with Mentor Graphics Modelsim SE 10.4. For logic synthesis, Synopsys DesignCompiler I-2013.12-SP3 for RHEL64 was used in conjunction with CORE65GPSVT v5.3.6 65nm standard cell library from STMicroelectronics. Preliminary power results were collected using the `report_power` feature in Design Compiler, while the chosen architecture was evaluated using vector based, cycle accurate power analysis using Synopsys PrimeTimePX.

5.2 Simulation

While designing the cryptosystem, the modules have been individually verified, and the S-box have been exhaustively tested.

During the first phase of testing the complete implementation, the CBC circuit was disabled letting AES operate in ECB-mode for easier verification. The large key- and data block size in AES makes exhaustive testing very time consuming. As such, known input-response vectors provided by NIST[3] were used. These vectors are constructed in the following way:

- Constant zero key, sequentially flipping each bit in the input data for 128, 192 and 256 bit key-lengths.
- Constant zero input data, sequentially flipping each bit in the key for 128, 192 and 256 bit key-lengths.

One common criteria for implementing secure cryptographic algorithms is the avalanche effect. Thus a change in one bit of input data or key, should incur a completely different ciphertext. The random bits for the masks and S-box were generated using the random number generator in SystemVerilog, regenerating a new value every cycle. Due to this the supplied vectors should give a good coverage of the functionality of the algorithm, although corner cases might not be completely covered.

To improve the coverage further, individual module verification was used for the submodules. Additionally the step-by-step encryption/decryption from the AES standard was used to verify correct output at each point during operation. As large parts of the architecture operates on masked data, it is necessary to combine the masks to verify the correct output at any given module. This is accomplished by simply XORing the masked values.

The second phase involved creating a test bench to test correct CBC-operation. Similarly to the previous method, known input-response vectors from NIST [3] was used.

Both the aforementioned test benches uses SystemVerilog assertion to perform a check where the output of the circuit is verified towards the vector given in the input-response file.

No errors were found during simulation indicating correct functionality of our implementation and that the random bits asserted to mask the internal data does not affect the end result.

5.3 Synthesis

The circuits were synthesized once with a target frequency of 400MHz[1], using the `compile_ultra` command in together with `no_autoungroup`. To prevent DC from optimizing across hierarchical boundaries the `no_autoungroup` command is essential with our proposed masking scheme. The reason for this is that our masking depends on *independent* processing of the data shares. Optimizing between modules could possible cause correlations between data in different shares.

The synthesis results clearly show why recent lightweight implementations of AES largely use the normal basis representations for calculating the multiplicative inverse in GF(2^8). The equivalent polynomial basis implementation uses 16% larger area, consumes 20% more power and operates at a speed 20MHz slower than the normal basis implementation. Further testing showed that the normal basis implementation successfully synthesized and verified for speeds in excess of 500MHz, still using less area than the polynomial basis implementation. A summary of these results is shown in table 5.1. Based on these results we decided to proceed with the normal basis implementation of the system.

Table 5.1: Synthesis Results

Target f	**Version**	**Area (GE)**	**Dyn. Pwr.**	**Stat. Pwr.**
400Mhz	Normal (S)	258046	46.56 mW	2.83 mW
400Mhz	Normal (N)	53072	12.69 mW	0.92 mW
380Mhz	Polynomial (S)	301192	56.13 mW	4.40 mW
400Mhz	Polynomial (N)	54369	16.38 mW	1.03 mW
500Mhz	Normal (S)	282443	67.22 mW	3.88 mW

The linear modules of our implementation scale linearly with the number of shares, occupying about 300% of the area of the non-shared version. The S-box on the other hand is 511% the size of the non-shared version. Considering the whole architecture, the increased area over the unmasked variant of the circuit is about 486%.

[1]Polynomial shared version was re-synthesized with a slightly slower target frequency to avoid negative slack

5.4 Power Analysis

To attain a more accurate power figure during normal operation for the selected implementation it is necessary to supply the power analysis tool, Synopsys Primetime PX, with switching activity for the circuit. Figure 5.1 shows the power trace for the proposed architecture while encrypting 100 random plaintexts. Note the low power consumption initially, this is during key expansion. Key expansion is performed unmasked, thus there is no masking overhead as opposed to the rest of the algorithm.

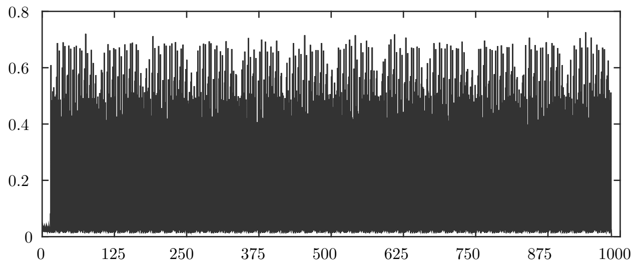


Figure 5.1: Simulated Power Trace from PrimeTime

Primetime also estimated the power consumption of the circuit to be substantially larger than what was reported by Design Compiler, with an average dynamic power of 167.9mW and an instantaneous peak power of 725.9 mW.

Table 5.2: Primetime Results

Target f	**Version**	**Dyn. Pwr.**	**Stat. Pwr.**
400Mhz	Normal (S)	167.9 mW	3.09 mW

This is likely due to constantly encrypting plaintexts. The avalanche effect in cryptographic algorithms cause most of the bytes in the state to change between two rounds, and thus also switching a large percentage of the transistors in the circuit.

5.5 Performance

Table 5.3 shows an overview of the performance of our proposed circuit at 400 MHz. Due to how clock synchronization is implemented in the proposed implementation there is quite a large overhead of 12 cycles from the theoretical minimum. With a 2-stage pipelined s-box, using 3 cycles per round, AES-128 can theoretically be performed in 33 cycles, with a similar reduction in cycles for the other modes. Considering this there is still room for optimization of our control circuitry. On the other hand, a throughput of approximately 1Gb/s should be sufficient for many applications, considering that this is a relatively lightweight implementation of AES.

Table 5.3: Performance @400MHz

Mode	**Cycles/Encryption**	**Throughput**
AES-128	42	1.32 Gb/s
AES-192	51	1.13 Gb/s
AES-256	57	0.99 Gb/s

5.6 Security Considerations

The S-box of our implementation is a 16 byte version based on the 1-byte S-box by Bilgin et al.[6] which has been attacked using DPA and CEPACA-attacks with more than 10 million power traces. Bilgin et al. did not find any weakness in the sharing as implemented here. By implementing the S-box as a 16-byte parallel module should increase the security as the Signal-to-Noise Ratio (SNR) of the circuit decreases. However we can not rule out that other weaknesses has been created in the process, this would be a good candidate for further work and analysis. By using 3 shares in all data path modules throughout the system, the system should be secure against second order DPA attacks.

5.7 Possible Improvements/Changes

During post synthesis simulations it became apparent that some further work may be needed on the control logic to improve the robustness of the architecture, as the clock synchronization causes a 27-30% time-increase for each encryption compared to the theoretical minimum. Furthermore, due to debugging reasons, the affine transformation and linear mapping modules of the S-box were kept separate. Combining these modules could also potentially yield a more area efficient structure, for example utilizing the exhaustive method proposed by Canright[12].

Chapter 6

Conclusion

6.1 Summary of Thesis and Results

This thesis propose a masked AES architecture with a 128-bit datapath supporting AES-128, AES-192 and AES-256 with CBC-support. The system has been synthesized to 65nm technology and verified pre- and post synthesis. Our proposed implementation should theoretically be able to withstand second order DPA-attacks with no assumptions on the underlying hardware. Two different basis representations for the S-boxes were evaluated and as for the unshared case, the normal basis appears to be the optimal basis resulting in a circuit 20% smaller, with 20% lower power consumption and more headroom for increasing the frequency. In preliminary synthesis tests, the proposed circuit is able to achieve frequencies in excess of 500MHz, in comparison to the polynomial basis version which is experiencing issues even at 380MHz. Operating at 400 MHz our implementation achieves a keymode-dependent throughput of 0.99-1.32Gb/s with an average power consumption of 167.9mW. Compared to the size of the unmasked version, the suggested implementation occupies 4.86 times the area of the unmasked version.

6.2 Further Work

Complete verification of our circuit would not be possible without loading it onto actual hardware, this will be left for future research. In addition some performance issues were identified that have room for optimization, both low level datapath logic and control logic.

Bibliography

[1] Federal information processing standards publication (FIPS 197). Advanced Encryption Standard (AES). `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, 2001.

[2] National Security Agency. Nacsim 5000 tempest fundamentals. `http://cryptome.org/jya/nacsim-5000/nacsim-5000.htm`, 1982.

[3] L.E. Bassham III. The advanced encryption standard algorithm validation suite (AESAVS). National Instistute of Standards and Technology, 2002. `http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf`.

[4] E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[5] B. Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, Enschede, May 2015.

[6] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Trade-offs for threshold implementations illustrated on aes. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, PP(99):1–1, 2015.

[7] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold implementations of all 3×3 and 4×4 s-boxes. In *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin Heidelberg, 2012.

[8] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup. Threshold implementations of small s-boxes. *Cryptography and Communications*, 7(1):3–33, 2015.

[9] J. Blömer, J. Guajardo, and V. Krummel. Provably secure masking of aes. In *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer Berlin Heidelberg, 2005.

[10] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer Berlin Heidelberg, 1997.

[11] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer Berlin Heidelberg, 2004.

[12] D. Canright. A very compact s-box for aes. In *in Proceedings of CHES 2005, ser. LNCS*, pages 441–455. Springer-Verlag, 2005.

[13] J. Daemen and V. Rijmen. Aes proposal: Rijndael. `http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf`, 1998.

[14] J. Daemen and V. Rijmen. Aes and the wide trail design strategy. In *Advances in Cryptology—EUROCRYPT 2002*, pages 108–109. Springer, 2002.

[15] M. Dworkin, R.M. Blank, P.D. Gallagher, et al. Recommendation for block cipher modes of operation: Methods and techniques. In *NIST Special Publication.* National Instistute of Standards and Technology, 2001. `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`.

[16] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A practical attack on keeloq. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2008.

[17] H. Kim, S. Hong, and J. Lim. A fast and provably secure higher-order masking of aes s-box. In *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer Berlin Heidelberg, 2011.

[18] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin Heidelberg, 1996.

[19] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99*, pages 388–397. Springer, 1999.

[20] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[21] S. Mangard, N. Pramstaller, and E. Oswald. Successfully attacking masked aes hardware implementations. In *in Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES), Aug. 2005, LNCS 3659*, pages 157–171. Springer, 2005.

[22] M. Matsui and A. Yamagishi. A new method for known plaintext attack of feal cipher. In *Advances in Cryptology — EUROCRYPT' 92*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer Berlin Heidelberg, 1993.

[23] T.S. Messerges. Using second-order power analysis to attack dpa resistant software. In *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer Berlin Heidelberg, 2000.

[24] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer Berlin Heidelberg, 2011.

[25] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139. Springer Berlin Heidelberg, 2010.

[26] S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *Journal of Cryptology*, 24(2):292–321, 2011.

[27] US Department of Commerce, National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for the advanced encryption standard (aes). [Docket No. 970725180-7180-01] RIN No. 0693-ZA16, 1997. `http://csrc.nist.gov/archive/aes/pre-round1/aes_9709.htm`.

[28] C. Paar. *Efficient VLSI architectures for bit parallel computation in Galois fields*. VDI-Verlag, 1994.

[29] E. Prouff and T. Roche. Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer Berlin Heidelberg, 2011.

[30] V. Rijmen. Efficient implementation of the rijndael s-box. *Katholieke Universiteit Leuven, Dept. ESAT. Belgium*, 2000.

[31] M. Rivain and E. Prouff. Provably secure higher-order masking of aes. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer Berlin Heidelberg, 2010.

[32] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi. Efficient rijndael encryption implementation with composite field arithmetic. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '01, pages 171–184, London, UK, UK, 2001. Springer-Verlag.

[33] B. Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., New York, NY, USA, 1995.

[34] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[35] H.C.A. van Tilborg. *Encyclopedia of Cryptography and Security.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[36] X.M. Zhang and K.K. Parhi. High-speed vlsi architectures for the aes algorithm. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(9):957–967, Sept 2004.

Appendix A

A

A.1 Isomorphisms

Equation A.1 and A.2 gives the isomorphism and inverse isomorphism for conversion from the AES-field to the composite normal field.

$$\begin{bmatrix}1&1&1&0&0&1&1&1\\0&1&1&1&0&0&0&1\\0&1&1&0&0&0&1&1\\1&1&1&0&0&0&0&1\\1&0&0&1&1&0&1&1\\0&0&0&0&0&0&0&1\\0&1&1&0&0&0&0&1\\0&1&0&0&1&1&1&1\end{bmatrix}\begin{bmatrix}b_7\\b_6\\b_5\\b_4\\b_3\\b_2\\b_1\\b_0\end{bmatrix} \tag{A.1}$$

$$\begin{bmatrix}0&0&0&1&0&0&1&0\\1&1&1&0&1&0&1&1\\1&1&1&0&1&1&0&1\\0&1&0&0&0&0&1&0\\0&1&1&1&1&1&1&0\\1&0&1&1&0&0&1&0\\0&0&1&0&0&0&1&0\\0&0&0&0&0&1&0&0\end{bmatrix}\begin{bmatrix}b_7\\b_6\\b_5\\b_4\\b_3\\b_2\\b_1\\b_0\end{bmatrix} \tag{A.2}$$

Equation A.3 and A.4 gives the isomorphism and inverse isomorphism for conversion from the AES-field to the composite polynomial field.

$$\begin{bmatrix}1&0&1&0&0&0&0&0\\1&1&0&1&1&1&1&0\\1&0&1&0&1&1&0&0\\1&0&1&0&1&1&1&0\\1&1&0&0&0&1&1&0\\1&0&0&1&1&1&1&0\\0&1&0&1&0&0&1&0\\0&1&0&0&0&0&1&1\end{bmatrix}\begin{bmatrix}b_7\\b_6\\b_5\\b_4\\b_3\\b_2\\b_1\\b_0\end{bmatrix} \tag{A.3}$$

$$\begin{bmatrix}1&1&1&0&0&0&1&0\\0&1&0&0&0&1&0&0\\0&1&1&0&0&0&1&0\\0&1&1&1&0&1&1&0\\0&0&1&1&1&1&1&0\\1&0&0&1&1&1&1&0\\0&0&1&1&0&0&0&0\\0&1&1&1&0&1&0&1\end{bmatrix}\begin{bmatrix}b_7\\b_6\\b_5\\b_4\\b_3\\b_2\\b_1\\b_0\end{bmatrix} \tag{A.4}$$

A.2 Sharing Schemes

$$\begin{aligned}F &= XY \text{ where}\\F &= F_1 \oplus F_2 \oplus F_3\\X &= X_1 \oplus X_2 \oplus X_3 \oplus X_4\\Y &= Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4\\F_1 &= (X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3) \oplus Y_4\\F_2 &= ((X_1 \oplus X_3)(Y_1 \oplus Y_4)) \oplus X_1Y_3 \oplus X_4\\F_3 &= ((X_2 \oplus X_4)(Y_1 \oplus Y_4)) \oplus X_1Y_2 \oplus X_4 \oplus Y_4\end{aligned} \tag{A.5}$$

$$
\begin{aligned}
F =& XYZ \oplus XY \oplus Z \\
F =& F_1 \oplus F_2 \oplus F_3 \oplus F_4 \oplus F_5 \\
X =& X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5 \\
Y =& Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5 \\
Z =& Z_1 \oplus Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5 \\
F_1 =& ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)(Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5)) \oplus \\
& ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)) \oplus Z_2 \\
F_2 =& (X_1(Y_3 \oplus Y_4 \oplus Y_5)(Z_3 \oplus Z_4 \oplus Z_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5)(Z_3 \oplus Z_4 \oplus Z_5) \oplus \\
& Z_1(X_3 \oplus X_4 \oplus X_5)(Y_3 \oplus Y_4 \oplus Y_5) \oplus X_1Y_1(Z_3 \oplus Z_4 \oplus Z_5) \oplus \\
& X_1Z_1(Y_3 \oplus Y_4 \oplus Y_5) \oplus Y_1Z_1(X_3 \oplus X_4 \oplus X_5) \oplus X_1Y_1Z_1) \oplus \\
& (X_1(Y_3 \oplus Y_4 \oplus Y_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5) \oplus X_1Y_1) \oplus Z_3 \\
F_3 =& (X_1Y_1Z_2 \oplus X_1Y_2Z_1 \oplus X_2Y_1Z_1 \oplus X_1Y_2Z_2 \oplus X_2Y_1Z_2 \oplus X_2Y_2Z_1 \oplus \\
& X_1Y_2Z_4 \oplus X_2Y_1Z_4 \oplus X_1Y_4Z_2 \oplus X_2Y_4Z_1 \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1 \oplus \\
& X1Y_2Z_5 \oplus X_2Y_1Z_5 \oplus X1Y_5Z_2 \oplus X_2Y_5Z_1 \oplus X_5Y_1Z_2 \oplus X_5Y_2Z_1) \oplus \\
& (X_1Y_2 \oplus Y_1X_2) \oplus Z_4 \\
F_4 =& X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_2Y_1Z_3 \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_3Y_2Z_1) \oplus Z_5 \\
F_5 =& Z_1
\end{aligned}
\tag{A.6}
$$

A.3 Inversion in GF(2^8)

$$
\begin{aligned}
g =& \gamma_1 y + \gamma_0 \\
d =& \delta_1 y + \delta_0 \\
gd =& 1 = (\gamma_1 y + \gamma_0)(\delta_1 y + \delta_0) \mod (y^2 + \tau y + \nu) \\
=& \gamma_1\delta_1 y^2 + \gamma_1\delta_0 y + \gamma_0\delta_1 y + \gamma_0\delta_0 \mod (y^2 + \tau y + \nu) \\
=& \gamma_1\delta_1(\tau y + \nu) + y(\gamma_1\delta_0 + \gamma_0\delta_1) + \gamma_0\delta_0 \\
=& (\gamma_1\delta_0 + \gamma_0\delta_1 + \tau\gamma_1\delta_1)y + \gamma_0\delta_0 + \gamma_1\delta_1\nu \\
=& 0y + 1
\end{aligned}
\tag{A.7}
$$

$$
\begin{aligned}
0 =& \gamma_1\delta_0 + \gamma_0\delta_1 + \gamma_1\delta_1\tau \\
1 =& \gamma_0\delta_0 + \gamma_1\delta_1\nu
\end{aligned}
\tag{A.8}
$$

$$\begin{aligned}
0 =&\gamma_1\gamma_0\delta_0 + (\gamma_0^2 + \gamma_0\gamma_1\tau)\delta_1 \\
\gamma_1 =&\gamma_1\gamma_0\delta_0 + (\gamma_1^2\nu)\delta_1 \\
\gamma_1 =&(\gamma_0^2 + \gamma_0\gamma_1\tau)\delta_1 + (\gamma_1^2\nu)\delta_1 \\
\gamma_1 =&(\gamma_1^2\nu + \gamma_1\gamma_0\tau + \gamma_0^2)\delta_1 \\
\gamma_1\delta_0 =&(\gamma_0 + \gamma_1\tau)\delta_1
\end{aligned} \tag{A.9}$$

$$\begin{aligned}
\Rightarrow \delta_1 =&(\gamma_1^2 + \gamma_1\gamma_0\tau + \gamma_0^2)^{-1}\gamma_1 \\
\delta_0 =&(\gamma_1^2 + \gamma_1\gamma_0\tau + \gamma_0^2)^{-1}(\gamma_0 + \gamma_1\tau)
\end{aligned} \tag{A.10}$$

A.4 GF(2^4) Inverter - Masked Realization in Polynomial Basis

$$\begin{aligned}
S_{0,0} =&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus \\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4)) \oplus x_1 \oplus y_1 \\
S_{0,1} =&((x_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(z_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0y_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(x_0z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0y_0z_0))\oplus \\
&(x_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0v_0) \oplus x_2 \oplus y_2 \\
S_{0,2} =&(x_0y_0z_1) \oplus (x_0y_1z_0) \oplus (x_1y_0z_0) \oplus (x_0y_1z_1) \oplus (x_1y_0z_1)\oplus \\
&(x_1y_1z_0) \oplus (x_0y_1z_3) \oplus (x_1y_0z_3) \oplus (x_0y_3z_1) \oplus (x_1y_3z_0)\oplus \\
&(x_3y_0z_1) \oplus (x_3y_1z_0) \oplus (x_0y_1z_4) \oplus (x_1y_0z_4) \oplus (x_0y_4z_1)\oplus \\
&(x_1y_4z_0) \oplus (x_4y_0z_1) \oplus (x_4y_1z_0) \oplus ((x_0v_1) \oplus (x_1v_0)) \oplus x_3 \oplus y_3 \\
S_{0,3} =&(x_0y_1z_2) \oplus (x_0y_2z_1) \oplus (x_1y_0z_2) \oplus (x_1y_2z_0) \oplus (x_2y_0z_1) \oplus (x_2y_1z_0)\oplus \\
&x_4 \oplus y_4 \\
S_{0,4} =&x_0 \oplus y_0
\end{aligned} \tag{A.11}$$

$$
\begin{aligned}
S_{1,0} =& ((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus \\
& ((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus \\
& ((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus \\
& ((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4)) \oplus y_1 \\
S_{1,1} =& ((x_0(y_2 \oplus y_3 \oplus y_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(v_2 \oplus v_3 \oplus v_4))\oplus \\
& (v_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0y_0(v_2 \oplus v_3 \oplus v_4))\oplus \\
& (x_0v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0y_0v_0))\oplus \\
& ((x_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4))\oplus \\
& (z_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0y_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
& (x_0z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0y_0z_0))\oplus \\
& (y_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0) \oplus (x_0(v_2 \oplus v_3 \oplus v_4))\oplus \\
& (v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0v_0) \oplus y_2 \\
S_{1,2} =& (x_0y_0v_1) \oplus (x_0y_1v_0) \oplus (x_1y_0v_0) \oplus (x_0y_1v_1) \oplus (x_1y_0v_1) \oplus (x_1y_1v_0)\oplus \\
& (x_0y_1v_3) \oplus (x_1y_0v_3) \oplus (x_0y_3v_1) \oplus (x_1y_3v_0) \oplus (x_3y_0v_1) \oplus (x_3y_1v_0)\oplus \\
& (x_0y_1v_4) \oplus (x_1y_0v_4) \oplus (x_0y_4v_1) \oplus (x_1y_4v_0) \oplus (x_4y_0v_1) \oplus (x_4y_1v_0)\oplus \\
& (x_0y_0z_1) \oplus (x_0y_1z_0) \oplus (x_1y_0z_0) \oplus (x_0y_1z_1) \oplus (x_1y_0z_1) \oplus (x_1y_1z_0)\oplus \\
& (x_0y_1z_3) \oplus (x_1y_0z_3) \oplus (x_0y_3z_1) \oplus (x_1y_3z_0) \oplus (x_3y_0z_1) \oplus (x_3y_1z_0)\oplus \\
& (x_0y_1z_4) \oplus (x_1y_0z_4) \oplus (x_0y_4z_1) \oplus (x_1y_4z_0) \oplus (x_4y_0z_1) \oplus (x_4y_1z_0)\oplus \\
& ((x_0v_1) \oplus (x_1v_0)) \oplus ((y_0z_1) \oplus (y_1z_0)) \oplus y_3 \\
S_{1,3} =& (x_0y_1v_2) \oplus (x_0y_2v_1) \oplus (x_1y_0v_2) \oplus (x_1y_2v_0) \oplus (x_2y_0v_1) \oplus (x_2y_1v_0)\oplus \\
& (x_0y_1z_2) \oplus (x_0y_2z_1) \oplus (x_1y_0z_2) \oplus (x_1y_2z_0) \oplus (x_2y_0z_1) \oplus (x_2y_1z_0) \oplus y_4 \\
S_{1,4} =& y_0
\end{aligned}
\tag{A.12}
$$

$$
\begin{aligned}
S_{2,0} =&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus \\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus \\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4)) \oplus x_1 \oplus y_1 \oplus z_1 \\
S_{2,1} =&((x_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(z_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0 y_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(x_0 z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0 z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 y_0 z_0))\oplus \\
&((x_0(z_2 \oplus z_3 \oplus z_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)(v_2 \oplus v_3 \oplus v_4))\oplus \\
&(v_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (x_0 z_0(v_2 \oplus v_3 \oplus v_4))\oplus \\
&(x_0 v_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0 v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 z_0 v_0))\oplus \\
&(y_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0 v_0) \oplus x_2 \oplus y_2 \oplus z_2 \\
S_{2,2} =&(x_0 y_0 z_1) \oplus (x_0 y_1 z_0) \oplus (x_1 y_0 z_0) \oplus (x_0 y_1 z_1) \oplus (x_1 y_0 z_1) \oplus (x_1 y_1 z_0)\oplus \\
&(x_0 y_1 z_3) \oplus (x_1 y_0 z_3) \oplus (x_0 y_3 z_1) \oplus (x_1 y_3 z_0) \oplus (x_3 y_0 z_1) \oplus (x_3 y_1 z_0)\oplus \\
&(x_0 y_1 z_4) \oplus (x_1 y_0 z_4) \oplus (x_0 y_4 z_1) \oplus (x_1 y_4 z_0) \oplus (x_4 y_0 z_1) \oplus (x_4 y_1 z_0)\oplus \\
&(x_0 z_0 v_1) \oplus (x_0 z_1 v_0) \oplus (x_1 z_0 v_0) \oplus (x_0 z_1 v_1) \oplus (x_1 z_0 v_1) \oplus (x_1 z_1 v_0)\oplus \\
&(x_0 z_1 v_3) \oplus (x_1 z_0 v_3) \oplus (x_0 z_3 v_1) \oplus (x_1 z_3 v_0) \oplus (x_3 z_0 v_1) \oplus (x_3 z_1 v_0)\oplus \\
&(x_0 z_1 v_4) \oplus (x_1 z_0 v_4) \oplus (x_0 z_4 v_1) \oplus (x_1 z_4 v_0) \oplus (x_4 z_0 v_1) \oplus (x_4 z_1 v_0)\oplus \\
&((y_0 v_1) \oplus (y_1 v_0)) \oplus x_3 \oplus y_3 \oplus z_3 \\
S_{2,3} =&(x_0 y_1 z_2) \oplus (x_0 y_2 z_1) \oplus (x_1 y_0 z_2) \oplus (x_1 y_2 z_0) \oplus (x_2 y_0 z_1) \oplus (x_2 y_1 z_0)\oplus \\
&(x_0 z_1 v_2) \oplus (x_0 z_2 v_1) \oplus (x_1 z_0 v_2) \oplus (x_1 z_2 v_0) \oplus (x_2 z_0 v_1) \oplus (x_2 z_1 v_0)\oplus \\
&x_4 \oplus y_4 \oplus z_4 \\
S_{2,4} =&x_0 \oplus y_0 \oplus z_0
\end{aligned}
\tag{A.13}
$$

$$
\begin{aligned}
S_{3,0} =&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)) \oplus y_1 \oplus z_1 \oplus v_1\\
S_{3,1} =&((x_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4))\oplus\\
&(z_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0y_0(z_2 \oplus z_3 \oplus z_4))\oplus\\
&(x_0z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0y_0z_0))\oplus\\
&((x_0(y_2 \oplus y_3 \oplus y_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(v_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0y_0(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(x_0v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0y_0v_0))\oplus\\
&((x_0(z_2 \oplus z_3 \oplus z_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(v_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (x_0z_0(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(x_0v_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0z_0v_0))\oplus\\
&((y_0(z_2 \oplus z_3 \oplus z_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (z_0(y_2 \oplus y_3 \oplus y_4)(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(v_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (y_0z_0(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(y_0v_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0v_0))\oplus\\
&(x_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0z_0)\oplus\\
&(x_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0v_0)\oplus\\
&(y_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0) \oplus y_2 \oplus z_2 \oplus v_2
\end{aligned}
$$

$$
\begin{aligned}
S_{3,2} =&(x_0y_0z_1) \oplus (x_0y_1z_0) \oplus (x_1y_0z_0) \oplus (x_0y_1z_1) \oplus (x_1y_0z_1) \oplus (x_1y_1z_0)\oplus\\
&(x_0y_1z_3) \oplus (x_1y_0z_3) \oplus (x_0y_3z_1) \oplus (x_1y_3z_0) \oplus (x_3y_0z_1) \oplus (x_3y_1z_0)\oplus\\
&(x_0y_1z_4) \oplus (x_1y_0z_4) \oplus (x_0y_4z_1) \oplus (x_1y_4z_0) \oplus (x_4y_0z_1) \oplus (x_4y_1z_0)\oplus\\
&(x_0y_0v_1) \oplus (x_0y_1v_0) \oplus (x_1y_0v_0) \oplus (x_0y_1v_1) \oplus (x_1y_0v_1) \oplus (x_1y_1v_0)\oplus\\
&(x_0y_1v_3) \oplus (x_1y_0v_3) \oplus (x_0y_3v_1) \oplus (x_1y_3v_0) \oplus (x_3y_0v_1) \oplus (x_3y_1v_0)\oplus\\
&(x_0y_1v_4) \oplus (x_1y_0v_4) \oplus (x_0y_4v_1) \oplus (x_1y_4v_0) \oplus (x_4y_0v_1) \oplus (x_4y_1v_0)\oplus\\
&(x_0z_0v_1) \oplus (x_0z_1v_0) \oplus (x_1z_0v_0) \oplus (x_0z_1v_1) \oplus (x_1z_0v_1) \oplus (x_1z_1v_0)\oplus\\
&(x_0z_1v_3) \oplus (x_1z_0v_3) \oplus (x_0z_3v_1) \oplus (x_1z_3v_0) \oplus (x_3z_0v_1) \oplus (x_3z_1v_0)\oplus\\
&(x_0z_1v_4) \oplus (x_1z_0v_4) \oplus (x_0z_4v_1) \oplus (x_1z_4v_0) \oplus (x_4z_0v_1) \oplus (x_4z_1v_0)\oplus\\
&(y_0z_0v_1) \oplus (y_0z_1v_0) \oplus (y_1z_0v_0) \oplus (y_0z_1v_1) \oplus (y_1z_0v_1) \oplus (y_1z_1v_0)\oplus\\
&(y_0z_1v_3) \oplus (y_1z_0v_3) \oplus (y_0z_3v_1) \oplus (y_1z_3v_0) \oplus (y_3z_0v_1) \oplus (y_3z_1v_0)\oplus\\
&(y_0z_1v_4) \oplus (y_1z_0v_4) \oplus (y_0z_4v_1) \oplus (y_1z_4v_0) \oplus (y_4z_0v_1) \oplus (y_4z_1v_0)\oplus\\
&((x_0z_1) \oplus (x_1z_0)) \oplus ((x_0v_1) \oplus (x_1v_0)) \oplus ((y_0z_1) \oplus (y_1z_0)) \oplus y_3 \oplus z_3 \oplus v_3\\
S_{3,3} =&(x_0y_1z_2) \oplus (x_0y_2z_1) \oplus (x_1y_0z_2) \oplus (x_1y_2z_0) \oplus (x_2y_0z_1) \oplus (x_2y_1z_0)\oplus\\
&(x_0y_1v_2) \oplus (x_0y_2v_1) \oplus (x_1y_0v_2) \oplus (x_1y_2v_0) \oplus (x_2y_0v_1) \oplus (x_2y_1v_0)\oplus\\
&(x_0z_1v_2) \oplus (x_0z_2v_1) \oplus (x_1z_0v_2) \oplus (x_1z_2v_0) \oplus (x_2z_0v_1) \oplus (x_2z_1v_0)\oplus\\
&(y_0z_1v_2) \oplus (y_0z_2v_1) \oplus (y_1z_0v_2) \oplus (y_1z_2v_0) \oplus (y_2z_0v_1) \oplus (y_2z_1v_0)\oplus\\
&y_4 \oplus z_4 \oplus v_4\\
S_{3,4} =&y_0 \oplus z_0 \oplus v_0
\end{aligned}
\tag{A.14}
$$

A.5 GF(2^4) Inverter - Masked Realization in Normal Basis

$$
\begin{aligned}
S_{0,0} =&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus \\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus \\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)) \oplus z_1 \oplus v_1 \\
S_{0,1} =&((y_0(z_2 \oplus z_3 \oplus z_4)(v_2 \oplus v_3 \oplus v_4))\oplus \\
&(z_0(y_2 \oplus y_3 \oplus y_4)(v_2 \oplus v_3 \oplus v_4))\oplus \\
&(v_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(y_0z_0(v_2 \oplus v_3 \oplus v_4)) \oplus (y_0v_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(z_0v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0v_0)) \oplus ((y_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0)) \oplus ((x_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0z_0)) \oplus z_2 \oplus v_2 \\
S_{0,2} =&((y_0z_0v_1) \oplus (y_0z_1v_0) \oplus (y_1z_0v_0) \oplus (y_0z_1v_1) \oplus (y_1z_0v_1) \oplus (y_1z_1v_0)\oplus \\
&(y_0z_1v_3) \oplus (y_1z_0v_3) \oplus (y_0z_3v_1) \oplus (y_1z_3v_0) \oplus (y_3z_0v_1) \oplus (y_3z_1v_0)\oplus \\
&(y_0z_1v_4) \oplus (y_1z_0v_4) \oplus (y_0z_4v_1) \oplus (y_1z_4v_0) \oplus (y_4z_0v_1) \oplus (y_4z_1v_0))\oplus \\
&((y_0z_1) \oplus (y_1z_0)) \oplus ((x_0z_1) \oplus (x_1z_0)) \oplus z_3 \oplus v_3 \\
S_{0,3} =&(y_0z_1v_2) \oplus (y_0z_2v_1) \oplus (y_1z_0v_2) \oplus (y_1z_2v_0) \oplus (y_2z_0v_1) \oplus (y_2z_1v_0)\oplus \\
&z_4 \oplus v_4 \\
S_{0,4} =&z_0 \oplus v_0
\end{aligned}
\tag{A.15}
$$

$$\begin{aligned}
S_{1,0} =&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)) \oplus v_1\\
S_{1,1} =&((x_0(z_2 \oplus z_3 \oplus z_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(v_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (x_0 z_0(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(x_0 v_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0 v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 z_0 v_0))\oplus\\
&((y_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (v_0 y_0))\oplus\\
&((y_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0 z_0))\oplus\\
&((x_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 z_0)) \oplus v_2\\
S_{1,2} =&((x_0 z_0 v_1) \oplus (x_0 z_1 v_0) \oplus (x_1 z_0 v_0) \oplus (x_0 z_1 v_1) \oplus (x_1 z_0 v_1) \oplus (x_1 z_1 v_0)\oplus\\
&(x_0 z_1 v_3) \oplus (x_1 z_0 v_3) \oplus (x_0 z_3 v_1) \oplus (x_1 z_3 v_0) \oplus (x_3 z_0 v_1) \oplus (x_3 z_1 v_0)\oplus\\
&(x_0 z_1 v_4) \oplus (x_1 z_0 v_4) \oplus (x_0 z_4 v_1) \oplus (x_1 z_4 v_0) \oplus (x_4 z_0 v_1) \oplus (x_4 z_1 v_0))\oplus\\
&((y_0 v_1) \oplus (y_1 v_0)) \oplus ((y_0 z_1) \oplus (y_1 z_0)) \oplus ((x_0 z_1) \oplus (x_1 z_0)) \oplus v_3\\
S_{1,3} =&(x_0 z_1 v_2) \oplus (x_0 z_2 v_1) \oplus (x_1 z_0 v_2) \oplus (x_1 z_2 v_0) \oplus (x_2 z_0 v_1) \oplus (x_2 z_1 v_0) \oplus v_4\\
S_{1,4} =&v_0
\end{aligned} \tag{A.16}$$

$$\begin{aligned}
S_{2,0} =&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus\\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)) \oplus x_1 \oplus y_1\\
S_{2,1} =&((x_0(y_2 \oplus y_3 \oplus y_4)(v_2 \oplus v_3 \oplus v_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(v_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0 y_0(v_2 \oplus v_3 \oplus v_4))\oplus\\
&(x_0 v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0 v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 y_0 v_0))\oplus\\
&((x_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 v_0))\oplus\\
&((x_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0 z_0)) \oplus x_2 \oplus y_2\\
S_{2,2} =&((x_0 y_0 v_1) \oplus (x_0 y_1 v_0) \oplus (x_1 y_0 v_0) \oplus (x_0 y_1 v_1) \oplus (x_1 y_0 v_1) \oplus (x_1 y_1 v_0)\oplus\\
&(x_0 y_1 v_3) \oplus (x_1 y_0 v_3) \oplus (x_0 y_3 v_1) \oplus (x_1 y_3 v_0) \oplus (x_3 y_0 v_1) \oplus (x_3 y_1 v_0)\oplus\\
&(x_0 y_1 v_4) \oplus (x_1 y_0 v_4) \oplus (x_0 y_4 v_1) \oplus (x_1 y_4 v_0) \oplus (x_4 y_0 v_1) \oplus (x_4 y_1 v_0))\oplus\\
&((x_0 v_1) \oplus (x_1 v_0)) \oplus ((x_0 z_1) \oplus (x_1 z_0)) \oplus x_3 \oplus y_3\\
S_{2,3} =&(x_0 y_1 v_2) \oplus (x_0 y_2 v_1) \oplus (x_1 y_0 v_2) \oplus (x_1 y_2 v_0) \oplus (x_2 y_0 v_1) \oplus (x_2 y_1 v_0) \oplus x_4 \oplus y_4\\
S_{2,4} =&x_0 \oplus y_0
\end{aligned} \tag{A.17}$$

$$
\begin{aligned}
S_{3,0} =&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(y_1 \oplus y_2 \oplus y_3 \oplus y_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4))\oplus \\
&((y_1 \oplus y_2 \oplus y_3 \oplus y_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus \\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(v_1 \oplus v_2 \oplus v_3 \oplus v_4))\oplus \\
&((x_1 \oplus x_2 \oplus x_3 \oplus x_4)(z_1 \oplus z_2 \oplus z_3 \oplus z_4)) \oplus y_1 \\
S_{3,1} =&((x_0(y_2 \oplus y_3 \oplus y_4)(z_2 \oplus z_3 \oplus z_4)) \oplus (y_0(x_2 \oplus x_3 \oplus x_4)(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(z_0(x_2 \oplus x_3 \oplus x_4)(y_2 \oplus y_3 \oplus y_4)) \oplus (x_0y_0(z_2 \oplus z_3 \oplus z_4))\oplus \\
&(x_0z_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0y_0z_0))\oplus \\
&((y_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(y_2 \oplus y_3 \oplus y_4)) \oplus (y_0v_0))\oplus \\
&((x_0(v_2 \oplus v_3 \oplus v_4)) \oplus (v_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0v_0))\oplus \\
&((x_0(z_2 \oplus z_3 \oplus z_4)) \oplus (z_0(x_2 \oplus x_3 \oplus x_4)) \oplus (x_0z_0)) \oplus y_2 \\
S_{3,2} =&((x_0y_0z_1) \oplus (x_0y_1z_0) \oplus (x_1y_0z_0) \oplus (x_0y_1z_1) \oplus (x_1y_0z_1) \oplus (x_1y_1z_0)\oplus \\
&(x_0y_1z_3) \oplus (x_1y_0z_3) \oplus (x_0y_3z_1) \oplus (x_1y_3z_0) \oplus (x_3y_0z_1) \oplus (x_3y_1z_0)\oplus \\
&(x_0y_1z_4) \oplus (x_1y_0z_4) \oplus (x_0y_4z_1) \oplus (x_1y_4z_0) \oplus (x_4y_0z_1) \oplus (x_4y_1z_0))\oplus \\
&((y_0v_1) \oplus (y_1v_0)) \oplus ((x_0v_1) \oplus (x_1v_0)) \oplus ((x_0z_1) \oplus (x_1z_0)) \oplus y_3 \\
S_{3,3} =&(x_0y_1z_2) \oplus (x_0y_2z_1) \oplus (x_1y_0z_2) \oplus (x_1y_2z_0) \oplus (x_2y_0z_1) \oplus (x_2y_1z_0) \oplus y_4 \\
S_{3,4} =&y_0
\end{aligned}
\tag{A.18}
$$

A.6 GF(2^4) Multiplier - Masked Realization in Polynomial Basis

$$
\begin{aligned}
S_{0,0} =&((a_1 \oplus a_2 \oplus a_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((a_1 \oplus a_2 \oplus a_3)(y_1 \oplus y_2)) \oplus y_3 \oplus \\
&((a_1 \oplus a_2 \oplus a_3)(z_1 \oplus z_2)) \oplus z_3 \oplus ((a_1 \oplus a_2 \oplus a_3)(v_1 \oplus v_2)) \oplus v_3 \oplus \\
&((b_1 \oplus b_2 \oplus b_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((b_1 \oplus b_2 \oplus b_3)(z_1 \oplus z_2)) \oplus z_3 \oplus \\
&((c_1 \oplus c_2 \oplus c_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((c_1 \oplus c_2 \oplus c_3)(y_1 \oplus y_2)) \oplus y_3 \oplus \\
&((d_1 \oplus d_2 \oplus d_3)(x_1 \oplus x_2)) \oplus x_3 \\
S_{0,1} =&((a_0 \oplus a_2)(x_0 \oplus x_3)) \oplus (a_0 x_2) \oplus a_3 \oplus ((a_0 \oplus a_2)(y_0 \oplus y_3)) \oplus \\
&(a_0 y_2) \oplus a_3 \oplus ((a_0 \oplus a_2)(z_0 \oplus z_3)) \oplus (a_0 z_2) \oplus a_3 \oplus ((a_0 \oplus a_2)(v_0 \oplus v_3)) \oplus \\
&(a_0 v_2) \oplus a_3 \oplus ((b_0 \oplus b_2)(x_0 \oplus x_3)) \oplus (b_0 x_2) \oplus b_3 \oplus ((b_0 \oplus b_2)(z_0 \oplus z_3)) \oplus \\
&(b_0 z_2) \oplus b_3 \oplus ((c_0 \oplus c_2)(x_0 \oplus x_3)) \oplus (c_0 x_2) \oplus c_3 \oplus ((c_0 \oplus c_2)(y_0 \oplus y_3)) \oplus \\
&(c_0 y_2) \oplus c_3 \oplus ((d_0 \oplus d_2)(x_0 \oplus x_3)) \oplus (d_0 x_2) \oplus d_3 \\
S_{0,2} =&((a_1 \oplus a_3)(x_0 \oplus x_3)) \oplus (a_0 x_1) \oplus a_3 \oplus x_3 \oplus ((a_1 \oplus a_3)(y_0 \oplus y_3)) \oplus \\
&(a_0 y_1) \oplus a_3 \oplus y_3 \oplus ((a_1 \oplus a_3)(z_0 \oplus z_3)) \oplus (a_0 z_1) \oplus a_3 \oplus z_3 \oplus \\
&((a_1 \oplus a_3)(v_0 \oplus v_3)) \oplus (a_0 v_1) \oplus a_3 \oplus v_3 \oplus ((b_1 \oplus b_3)(x_0 \oplus x_3)) \oplus \\
&(b_0 x_1) \oplus b_3 \oplus x_3 \oplus ((b_1 \oplus b_3)(z_0 \oplus z_3)) \oplus (b_0 z_1) \oplus b_3 \oplus z_3 \oplus \\
&((c_1 \oplus c_3)(x_0 \oplus x_3)) \oplus (c_0 x_1) \oplus c_3 \oplus x_3 \oplus ((c_1 \oplus c_3)(y_0 \oplus y_3)) \oplus \\
&(c_0 y_1) \oplus c_3 \oplus y_3 \oplus ((d_1 \oplus d_3)(x_0 \oplus x_3)) \oplus (d_0 x_1) \oplus d_3 \oplus x_3
\end{aligned}
\tag{A.19}
$$

$$
\begin{aligned}
S_{1,0} =&((a_1 \oplus a_2 \oplus a_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((a_1 \oplus a_2 \oplus a_3)(z_1 \oplus z_2)) \oplus z_3 \oplus \\
&((b_1 \oplus b_2 \oplus b_3)(y_1 \oplus y_2)) \oplus y_3 \oplus ((b_1 \oplus b_2 \oplus b_3)(v_1 \oplus v_2)) \oplus v_3 \oplus \\
&((c_1 \oplus c_2 \oplus c_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((d_1 \oplus d_2 \oplus d_3)(y_1 \oplus y_2)) \oplus y_3 \\
S_{1,1} =&((a_0 \oplus a_2)(x_0 \oplus x_3)) \oplus (a_0 x_2) \oplus a_3 \oplus ((a_0 \oplus a_2)(z_0 \oplus z_3)) \oplus (a_0 z_2) \oplus a_3 \oplus \\
&((b_0 \oplus b_2)(y_0 \oplus y_3)) \oplus (b_0 y_2) \oplus b_3 \oplus ((b_0 \oplus b_2)(v_0 \oplus v_3)) \oplus (b_0 v_2) \oplus b_3 \oplus \\
&((c_0 \oplus c_2)(x_0 \oplus x_3)) \oplus (c_0 x_2) \oplus c_3 \oplus ((d_0 \oplus d_2)(y_0 \oplus y_3)) \oplus (d_0 y_2) \oplus d_3 \\
S_{1,2} =&((a_1 \oplus a_3)(x_0 \oplus x_3)) \oplus (a_0 x_1) \oplus a_3 \oplus x_3 \oplus ((a_1 \oplus a_3)(z_0 \oplus z_3)) \oplus (a_0 z_1) \oplus a_3 \oplus z_3 \oplus \\
&((b_1 \oplus b_3)(y_0 \oplus y_3)) \oplus (b_0 y_1) \oplus b_3 \oplus y_3 \oplus ((b_1 \oplus b_3)(v_0 \oplus v_3)) \oplus (b_0 v_1) \oplus b_3 \oplus v_3 \oplus \\
&((c_1 \oplus c_3)(x_0 \oplus x_3)) \oplus (c_0 x_1) \oplus c_3 \oplus x_3 \oplus ((d_1 \oplus d_3)(y_0 \oplus y_3)) \oplus (d_0 y_1) \oplus d_3 \oplus y_3
\end{aligned}
\tag{A.20}
$$

$$\begin{aligned}
S_{2,0} =&((a_1 \oplus a_2 \oplus a_3)(y_1 \oplus y_2)) \oplus y_3 \oplus ((b_1 \oplus b_2 \oplus b_3)(x_1 \oplus x_2)) \oplus x_3 \oplus \\
&((b_1 \oplus b_2 \oplus b_3)(y_1 \oplus y_2)) \oplus y_3 \oplus ((c_1 \oplus c_2 \oplus c_3)(z_1 \oplus z_2)) \oplus z_3 \oplus \\
&((c_1 \oplus c_2 \oplus c_3)(v_1 \oplus v_2)) \oplus v_3 \oplus ((d_1 \oplus d_2 \oplus d_3)(z_1 \oplus z_2)) \oplus z_3 \\
S_{2,1} =&((a_0 \oplus a_2)(y_0 \oplus y_3)) \oplus (a_0 y_2) \oplus a_3 \oplus ((b_0 \oplus b_2)(x_0 \oplus x_3)) \oplus (b_0 x_2) \oplus b_3 \oplus \\
&((b_0 \oplus b_2)(y_0 \oplus y_3)) \oplus (b_0 y_2) \oplus b_3 \oplus ((c_0 \oplus c_2)(z_0 \oplus z_3)) \oplus (c_0 z_2) \oplus c_3 \oplus \\
&((c_0 \oplus c_2)(v_0 \oplus v_3)) \oplus (c_0 v_2) \oplus c_3 \oplus ((d_0 \oplus d_2)(z_0 \oplus z_3)) \oplus (d_0 z_2) \oplus d_3 \\
S_{2,2} =&((a_1 \oplus a_3)(y_0 \oplus y_3)) \oplus (a_0 y_1) \oplus a_3 \oplus y_3 \oplus ((b_1 \oplus b_3)(x_0 \oplus x_3)) \oplus (b_0 x_1) \oplus b_3 \oplus x_3 \oplus \\
&((b_1 \oplus b_3)(y_0 \oplus y_3)) \oplus (b_0 y_1) \oplus b_3 \oplus y_3 \oplus ((c_1 \oplus c_3)(z_0 \oplus z_3)) \oplus (c_0 z_1) \oplus c_3 \oplus z_3 \oplus \\
&((c_1 \oplus c_3)(v_0 \oplus v_3)) \oplus (c_0 v_1) \oplus c_3 \oplus v_3 \oplus ((d_1 \oplus d_3)(z_0 \oplus z_3)) \oplus (d_0 z_1) \oplus d_3 \oplus z_3
\end{aligned} \tag{A.21}$$

$$\begin{aligned}
S_{3,0} =&((a_1 \oplus a_2 \oplus a_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((a_1 \oplus a_2 \oplus a_3)(y_1 \oplus y_2)) \oplus y_3 \oplus \\
&((b_1 \oplus b_2 \oplus b_3)(x_1 \oplus x_2)) \oplus x_3 \oplus ((c_1 \oplus c_2 \oplus c_3)(z_1 \oplus z_2)) \oplus z_3 \oplus \\
&((d_1 \oplus d_2 \oplus d_3)(v_1 \oplus v_2)) \oplus v_3 \\
S_{3,1} =&((a_0 \oplus a_2)(x_0 \oplus x_3)) \oplus (a_0 x_2) \oplus a_3 \oplus ((a_0 \oplus a_2)(y_0 \oplus y_3)) \oplus (a_0 y_2) \oplus a_3 \oplus \\
&((b_0 \oplus b_2)(x_0 \oplus x_3)) \oplus (b_0 x_2) \oplus b_3 \oplus ((c_0 \oplus c_2)(z_0 \oplus z_3)) \oplus (c_0 z_2) \oplus c_3 \oplus \\
&((d_0 \oplus d_2)(v_0 \oplus v_3)) \oplus (d_0 v_2) \oplus d_3 \\
S_{3,2} =&((a_1 \oplus a_3)(x_0 \oplus x_3)) \oplus (a_0 x_1) \oplus a_3 \oplus x_3 \oplus ((a_1 \oplus a_3)(y_0 \oplus y_3)) \oplus (a_0 y_1) \oplus a_3 \oplus y_3 \oplus \\
&((b_1 \oplus b_3)(x_0 \oplus x_3)) \oplus (b_0 x_1) \oplus b_3 \oplus x_3 \oplus ((c_1 \oplus c_3)(z_0 \oplus z_3)) \oplus (c_0 z_1) \oplus c_3 \oplus z_3 \oplus \\
&((d_1 \oplus d_3)(v_0 \oplus v_3)) \oplus (d_0 v_1) \oplus d_3 \oplus v_3
\end{aligned} \tag{A.22}$$

A.7 GF(2^4) Multiplier - Masked Realization in Normal Basis

$$
\begin{aligned}
S_{0,0} =&(((a_1 \oplus a_2 \oplus a_3)(e_1 \oplus e_2)) \oplus e_3) \oplus (((c_1 \oplus c_2 \oplus c_3)(e_1 \oplus e_2)) \oplus e_3)\oplus \\
&(((d_1 \oplus d_2 \oplus d_3)(e_1 \oplus e_2)) \oplus e_3) \oplus (((b_1 \oplus b_2 \oplus b_3)(f_1 \oplus f_2)) \oplus f_3)\oplus \\
&(((c_1 \oplus c_2 \oplus c_3)(f_1 \oplus f_2)) \oplus f_3) \oplus (((a_1 \oplus a_2 \oplus a_3)(g_1 \oplus g_2)) \oplus g_3)\oplus \\
&(((b_1 \oplus b_2 \oplus b_3)(g_1 \oplus g_2)) \oplus g_3) \oplus (((c_1 \oplus c_2 \oplus c_3)(g_1 \oplus g_2)) \oplus g_3)\oplus \\
&(((d_1 \oplus d_2 \oplus d_3)(g_1 \oplus g_2)) \oplus g_3) \oplus (((a_1 \oplus a_2 \oplus a_3)(h_1 \oplus h_2)) \oplus h_3)\oplus \\
&(((c_1 \oplus c_2 \oplus c_3)(h_1 \oplus h_2)) \oplus h_3) \\
S_{0,1} =&(((a_0 \oplus a_2)(e_0 \oplus e_3)) \oplus (a_0e_2) \oplus a_3) \oplus (((c_0 \oplus c_2)(e_0 \oplus e_3)) \oplus (c_0e_2) \oplus c_3)\oplus \\
&(((d_0 \oplus d_2)(e_0 \oplus e_3)) \oplus (d_0e_2) \oplus d_3) \oplus (((b_0 \oplus b_2)(f_0 \oplus f_3)) \oplus (b_0f_2) \oplus b_3)\oplus \\
&(((c_0 \oplus c_2)(f_0 \oplus f_3)) \oplus (c_0f_2) \oplus c_3) \oplus (((a_0 \oplus a_2)(g_0 \oplus g_3)) \oplus (a_0g_2) \oplus a_3)\oplus \\
&(((b_0 \oplus b_2)(g_0 \oplus g_3)) \oplus (b_0g_2) \oplus b_3) \oplus (((c_0 \oplus c_2)(g_0 \oplus g_3)) \oplus (c_0g_2) \oplus c_3)\oplus \\
&(((d_0 \oplus d_2)(g_0 \oplus g_3)) \oplus (d_0g_2) \oplus d_3) \oplus (((a_0 \oplus a_2)(h_0 \oplus h_3)) \oplus (a_0h_2) \oplus a_3)\oplus \\
&(((c_0 \oplus c_2)(h_0 \oplus h_3)) \oplus (c_0h_2) \oplus c_3) \\
S_{0,2} =&(((a_1 \oplus a_3)(e_0 \oplus e_3)) \oplus (a_0e_1) \oplus a_3 \oplus e_3)\oplus \\
&(((c_1 \oplus c_3)(e_0 \oplus e_3)) \oplus (c_0e_1) \oplus c_3 \oplus e_3)\oplus \\
&(((d_1 \oplus d_3)(e_0 \oplus e_3)) \oplus (d_0e_1) \oplus d_3 \oplus e_3)\oplus \\
&(((b_1 \oplus b_3)(f_0 \oplus f_3)) \oplus (b_0f_1) \oplus b_3 \oplus f_3)\oplus \\
&(((c_1 \oplus c_3)(f_0 \oplus f_3)) \oplus (c_0f_1) \oplus c_3 \oplus f_3)\oplus \\
&(((a_1 \oplus a_3)(g_0 \oplus g_3)) \oplus (a_0g_1) \oplus a_3 \oplus g_3)\oplus \\
&(((b_1 \oplus b_3)(g_0 \oplus g_3)) \oplus (b_0g_1) \oplus b_3 \oplus g_3)\oplus \\
&(((c_1 \oplus c_3)(g_0 \oplus g_3)) \oplus (c_0g_1) \oplus c_3 \oplus g_3)\oplus \\
&(((d_1 \oplus d_3)(g_0 \oplus g_3)) \oplus (d_0g_1) \oplus d_3 \oplus g_3)\oplus \\
&(((a_1 \oplus a_3)(h_0 \oplus h_3)) \oplus (a_0h_1) \oplus a_3 \oplus h_3)\oplus \\
&(((c_1 \oplus c_3)(h_0 \oplus h_3)) \oplus (c_0h_1) \oplus c_3 \oplus h_3)
\end{aligned}
\tag{A.23}
$$

$$
\begin{aligned}
S_{1,0} =&(((b_1 \oplus b_2 \oplus b_3)(e_1 \oplus e_2)) \oplus e_3) \oplus (((c_1 \oplus c_2 \oplus c_3)(e_1 \oplus e_2)) \oplus e_3)\oplus \\
&(((a_1 \oplus a_2 \oplus a_3)(f_1 \oplus f_2)) \oplus f_3) \oplus (((b_1 \oplus b_2 \oplus b_3)(f_1 \oplus f_2)) \oplus f_3)\oplus \\
&(((d_1 \oplus d_2 \oplus d_3)(f_1 \oplus f_2)) \oplus f_3) \oplus (((a_1 \oplus a_2 \oplus a_3)(g_1 \oplus g_2)) \oplus g_3)\oplus \\
&(((c_1 \oplus c_2 \oplus c_3)(g_1 \oplus g_2)) \oplus g_3) \oplus (((b_1 \oplus b_2 \oplus b_3)(h_1 \oplus h_2)) \oplus h_3)\oplus \\
&(((d_1 \oplus d_2 \oplus d_3)(h_1 \oplus h_2)) \oplus h_3) \\
S_{1,1} =&(((b_0 \oplus b_2)(e_0 \oplus e_3)) \oplus (b_0e_2) \oplus b_3) \oplus (((c_0 \oplus c_2)(e_0 \oplus e_3)) \oplus (c_0e_2) \oplus c_3)\oplus \\
&(((a_0 \oplus a_2)(f_0 \oplus f_3)) \oplus (a_0f_2) \oplus a_3) \oplus (((b_0 \oplus b_2)(f_0 \oplus f_3)) \oplus (b_0f_2) \oplus b_3)\oplus \\
&(((d_0 \oplus d_2)(f_0 \oplus f_3)) \oplus (d_0f_2) \oplus d_3) \oplus (((a_0 \oplus a_2)(g_0 \oplus g_3)) \oplus (a_0g_2) \oplus a_3)\oplus \\
&(((c_0 \oplus c_2)(g_0 \oplus g_3)) \oplus (c_0g_2) \oplus c_3) \oplus (((b_0 \oplus b_2)(h_0 \oplus h_3)) \oplus (b_0h_2) \oplus b_3)\oplus \\
&(((d_0 \oplus d_2)(h_0 \oplus h_3)) \oplus (d_0h_2) \oplus d_3) \\
S_{1,2} =&(((b_1 \oplus b_3)(e_0 \oplus e_3)) \oplus (b_0e_1) \oplus b_3 \oplus e_3)\oplus \\
&(((c_1 \oplus c_3)(e_0 \oplus e_3)) \oplus (c_0e_1) \oplus c_3 \oplus e_3)\oplus \\
&(((a_1 \oplus a_3)(f_0 \oplus f_3)) \oplus (a_0f_1) \oplus a_3 \oplus f_3)\oplus \\
&(((b_1 \oplus b_3)(f_0 \oplus f_3)) \oplus (b_0f_1) \oplus b_3 \oplus f_3)\oplus \\
&(((d_1 \oplus d_3)(f_0 \oplus f_3)) \oplus (d_0f_1) \oplus d_3 \oplus f_3)\oplus \\
&(((a_1 \oplus a_3)(g_0 \oplus g_3)) \oplus (a_0g_1) \oplus a_3 \oplus g_3)\oplus \\
&(((c_1 \oplus c_3)(g_0 \oplus g_3)) \oplus (c_0g_1) \oplus c_3 \oplus g_3)\oplus \\
&(((b_1 \oplus b_3)(h_0 \oplus h_3)) \oplus (b_0h_1) \oplus b_3 \oplus h_3)\oplus \\
&(((d_1 \oplus d_3)(h_0 \oplus h_3)) \oplus (d_0h_1) \oplus d_3 \oplus h_3)
\end{aligned}
\tag{A.24}
$$

$$
\begin{aligned}
S_{2,0} =& (((a_1 \oplus a_2 \oplus a_3)(e_1 \oplus e_2)) \oplus e_3) \oplus (((b_1 \oplus b_2 \oplus b_3)(e_1 \oplus e_2)) \oplus e_3)\oplus \\
& (((c_1 \oplus c_2 \oplus c_3)(e_1 \oplus e_2)) \oplus e_3) \oplus (((d_1 \oplus d_2 \oplus d_3)(e_1 \oplus e_2)) \oplus e_3)\oplus \\
& (((a_1 \oplus a_2 \oplus a_3)(f_1 \oplus f_2)) \oplus f_3) \oplus (((c_1 \oplus c_2 \oplus c_3)(f_1 \oplus f_2)) \oplus f_3)\oplus \\
& (((a_1 \oplus a_2 \oplus a_3)(g_1 \oplus g_2)) \oplus g_3) \oplus (((b_1 \oplus b_2 \oplus b_3)(g_1 \oplus g_2)) \oplus g_3)\oplus \\
& (((c_1 \oplus c_2 \oplus c_3)(g_1 \oplus g_2)) \oplus g_3) \oplus (((a_1 \oplus a_2 \oplus a_3)(h_1 \oplus h_2)) \oplus h_3)\oplus \\
& (((d_1 \oplus d_2 \oplus d_3)(h_1 \oplus h_2)) \oplus h_3) \\
S_{2,1} =& (((a_0 \oplus a_2)(e_0 \oplus e_3)) \oplus (a_0 e_2) \oplus a_3) \oplus (((b_0 \oplus b_2)(e_0 \oplus e_3)) \oplus (b_0 e_2) \oplus b_3)\oplus \\
& (((c_0 \oplus c_2)(e_0 \oplus e_3)) \oplus (c_0 e_2) \oplus c_3) \oplus (((d_0 \oplus d_2)(e_0 \oplus e_3)) \oplus (d_0 e_2) \oplus d_3)\oplus \\
& (((a_0 \oplus a_2)(f_0 \oplus f_3)) \oplus (a_0 f_2) \oplus a_3) \oplus (((c_0 \oplus c_2)(f_0 \oplus f_3)) \oplus (c_0 f_2) \oplus c_3)\oplus \\
& (((a_0 \oplus a_2)(g_0 \oplus g_3)) \oplus (a_0 g_2) \oplus a_3) \oplus (((b_0 \oplus b_2)(g_0 \oplus g_3)) \oplus (b_0 g_2) \oplus b_3)\oplus \\
& (((c_0 \oplus c_2)(g_0 \oplus g_3)) \oplus (c_0 g_2) \oplus c_3) \oplus (((a_0 \oplus a_2)(h_0 \oplus h_3)) \oplus (a_0 h_2) \oplus a_3)\oplus \\
& (((d_0 \oplus d_2)(h_0 \oplus h_3)) \oplus (d_0 h_2) \oplus d_3) \\
S_{2,2} =& (((a_1 \oplus a_3)(e_0 \oplus e_3)) \oplus (a_0 e_1) \oplus a_3 \oplus e_3)\oplus \\
& (((b_1 \oplus b_3)(e_0 \oplus e_3)) \oplus (b_0 e_1) \oplus b_3 \oplus e_3)\oplus \\
& (((c_1 \oplus c_3)(e_0 \oplus e_3)) \oplus (c_0 e_1) \oplus c_3 \oplus e_3)\oplus \\
& (((d_1 \oplus d_3)(e_0 \oplus e_3)) \oplus (d_0 e_1) \oplus d_3 \oplus e_3)\oplus \\
& (((a_1 \oplus a_3)(f_0 \oplus f_3)) \oplus (a_0 f_1) \oplus a_3 \oplus f_3)\oplus \\
& (((c_1 \oplus c_3)(f_0 \oplus f_3)) \oplus (c_0 f_1) \oplus c_3 \oplus f_3)\oplus \\
& (((a_1 \oplus a_3)(g_0 \oplus g_3)) \oplus (a_0 g_1) \oplus a_3 \oplus g_3)\oplus \\
& (((b_1 \oplus b_3)(g_0 \oplus g_3)) \oplus (b_0 g_1) \oplus b_3 \oplus g_3)\oplus \\
& (((c_1 \oplus c_3)(g_0 \oplus g_3)) \oplus (c_0 g_1) \oplus c_3 \oplus g_3)\oplus \\
& (((a_1 \oplus a_3)(h_0 \oplus h_3)) \oplus (a_0 h_1) \oplus a_3 \oplus h_3)\oplus \\
& (((d_1 \oplus d_3)(h_0 \oplus h_3)) \oplus (d_0 h_1) \oplus d_3 \oplus h_3)
\end{aligned}
\tag{A.25}
$$

$$\begin{aligned}
S_{3,0} =&(((a_1 \oplus a_2 \oplus a_3)(e_1 \oplus e_2)) \oplus e_3) \oplus (((c_1 \oplus c_2 \oplus c_3)(e_1 \oplus e_2)) \oplus e_3)\oplus\\
&(((b_1 \oplus b_2 \oplus b_3)(f_1 \oplus f_2)) \oplus f_3) \oplus (((d_1 \oplus d_2 \oplus d_3)(f_1 \oplus f_2)) \oplus f_3)\oplus\\
&(((a_1 \oplus a_2 \oplus a_3)(g_1 \oplus g_2)) \oplus g_3) \oplus (((d_1 \oplus d_2 \oplus d_3)(g_1 \oplus g_2)) \oplus g_3)\oplus\\
&(((b_1 \oplus b_2 \oplus b_3)(h_1 \oplus h_2)) \oplus h_3) \oplus (((c_1 \oplus c_2 \oplus c_3)(h_1 \oplus h_2)) \oplus h_3)\oplus\\
&(((d_1 \oplus d_2 \oplus d_3)(h_1 \oplus h_2)) \oplus h_3)\\
S_{3,1} =&(((a_0 \oplus a_2)(e_0 \oplus e_3)) \oplus (a_0e_2) \oplus a_3) \oplus (((c_0 \oplus c_2)(e_0 \oplus e_3)) \oplus (c_0e_2) \oplus c_3)\oplus\\
&(((b_0 \oplus b_2)(f_0 \oplus f_3)) \oplus (b_0f_2) \oplus b_3) \oplus (((d_0 \oplus d_2)(f_0 \oplus f_3)) \oplus (d_0f_2) \oplus d_3)\oplus\\
&(((a_0 \oplus a_2)(g_0 \oplus g_3)) \oplus (a_0g_2) \oplus a_3) \oplus (((d_0 \oplus d_2)(g_0 \oplus g_3)) \oplus (d_0g_2) \oplus d_3)\oplus\\
&(((b_0 \oplus b_2)(h_0 \oplus h_3)) \oplus (b_0h_2) \oplus b_3) \oplus (((c_0 \oplus c_2)(h_0 \oplus h_3)) \oplus (c_0h_2) \oplus c_3)\oplus\\
&(((d_0 \oplus d_2)(h_0 \oplus h_3)) \oplus (d_0h_2) \oplus d_3)\\
S_{3,2} =&(((a_1 \oplus a_3)(e_0 \oplus e_3)) \oplus (a_0e_1) \oplus a_3 \oplus e_3)\oplus\\
&(((c_1 \oplus c_3)(e_0 \oplus e_3)) \oplus (c_0e_1) \oplus c_3 \oplus e_3)\oplus\\
&(((b_1 \oplus b_3)(f_0 \oplus f_3)) \oplus (b_0f_1) \oplus b_3 \oplus f_3)\oplus\\
&(((d_1 \oplus d_3)(f_0 \oplus f_3)) \oplus (d_0f_1) \oplus d_3 \oplus f_3)\oplus\\
&(((a_1 \oplus a_3)(g_0 \oplus g_3)) \oplus (a_0g_1) \oplus a_3 \oplus g_3)\oplus\\
&(((d_1 \oplus d_3)(g_0 \oplus g_3)) \oplus (d_0g_1) \oplus d_3 \oplus g_3)\oplus\\
&(((b_1 \oplus b_3)(h_0 \oplus h_3)) \oplus (b_0h_1) \oplus b_3 \oplus h_3)\oplus\\
&(((c_1 \oplus c_3)(h_0 \oplus h_3)) \oplus (c_0h_1) \oplus c_3 \oplus h_3)\oplus\\
&(((d_1 \oplus d_3)(h_0 \oplus h_3)) \oplus (d_0h_1) \oplus d_3 \oplus h_3)
\end{aligned} \tag{A.26}$$