5-2021

Evaluating Performance and Efficiency of a 16-bit Substitution Box on an FPGA

Daniel F. Stafford
dfs1501@rit.edu

Evaluating Performance and Efficiency of a 16-bit Substitution Box on an FPGA

by

Daniel F. Stafford

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science
in Computer Engineering

Supervised by

Dr. Marcin Łukowiak
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
May 2021

Approved by:

Dr. Marcin Łukowiak
Thesis Advisor, Department of Computer Engineering

Dr. Stanisław Radziszowski
Committee Member, Department of Computer Science

Dr. Sonia Lopez Alarcon
Committee Member, Department of Computer Engineering

Dr. Michael Kurdziel
Committee Member, L3Harris

Abstract

Evaluating Performance and Efficiency of a 16-bit Substitution Box on an FPGA

Daniel F. Stafford

Supervising Professor: Dr. Marcin Łukowiak

A Substitution Box (S-Box) is an integral component of modern block ciphers that provides confusion. The term ”confusion” was introduced by Shannon in 1949 and it refers to the complexity of the relationship between the key and the ciphertext. Most S-Boxes are non-linear in order to promote confusion. Due to this, the S-Box is usually the most complex component of a block cipher. The Advanced Encryption Standard (AES) features an 8-bit S-Box where the output depends on the Galois field multiplicative inverse of the input.

MK-3 is a sponge based Authenticated Encryption (AE) algorithm which provides both authenticity and confidentiality. It was developed through a joint effort between the Rochester Institute of Technology and the former Harris Corporation, now L3Harris. The MK-3 algorithm has a state that is 512 bits wide and it uses 32 instances of a 16-bit S-Box to cover the entire state. These 16-bit S-Boxes are similar to what is seen in the AES, however, they are notably larger and more complex.

Binary Galois field arithmetic is well suited to hardware implementations where addition and multiplication are mapped to a combination of basic XOR and AND operations. A simple method to calculate Galois field multiplicative inversion is through the extended Euclidean algorithm. This is, however, very expensive to implement in hardware. A possible solution is to use a composite field representation, where the original operation is broken down to a series of simpler operations in the base field. This lends itself very well to implementations that consume less area and power with better performance.

Given the size and number of the S-Boxes in MK-3, these units contribute to the majority of the implementation resources. Several composite field structures are explored in this work which provide different area utilization and clock frequency characteristics. This thesis evaluates the composite field structures and recommends several candidates for high performing MK-3 Field Programmable Gate Array (FPGA) applications.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The vast majority of research on hardware implementations of cryptographic algorithms uses Application Specific Integrated Circuits (ASICs) as the platform. There is a good reason for this; by implementing the algorithm on an ASIC, the designer has control of the implementation down to the logic gate or even the transistors if necessary. Each logical operation of the design is synthesized into a set of logic gates on an ASIC. This allows the researchers to get the most efficient designs and also provides an easy way to compare against others just by the gate count. Comparison on an equivalent design on a Field Programmable Gate Array (FPGA) is more difficult since the atomic computation unit is a Look Up Table (LUT). Where one operation on an ASIC would be mapped to a series of logic gates, one or more operations in an FPGA can be mapped to a single LUT. This obfuscates the impact of each optimization.

However, there are occasions where an FPGA is desired for the implementation. An FPGA is reconfigurable; the design can be changed at a moment's notice. Whereas an implementation on an ASIC has one predetermined purpose, it cannot change. Likewise, the development time is shorter on an FPGA and does not require as large of an investment to get started.

1.2 Overview of Symmetric Key Cryptography

Symmetric key cryptography involves three components: the unencrypted message (plaintext), the key for the encryption, and the encrypted message (ciphertext). The classic example shown in Fig. 1.1 features Alice sending a message to Bob over an insecure channel. Since the contents of messages can be observed through the insecure channel, Alice encrypts the plaintext with the key in order to produce a ciphertext. While it may be observed while in transit, without the key, the ciphertext will not involve any details of the plaintext. When Bob receives the cipher text, he uses the same key to decrypt it as Alice used to encrypt it.

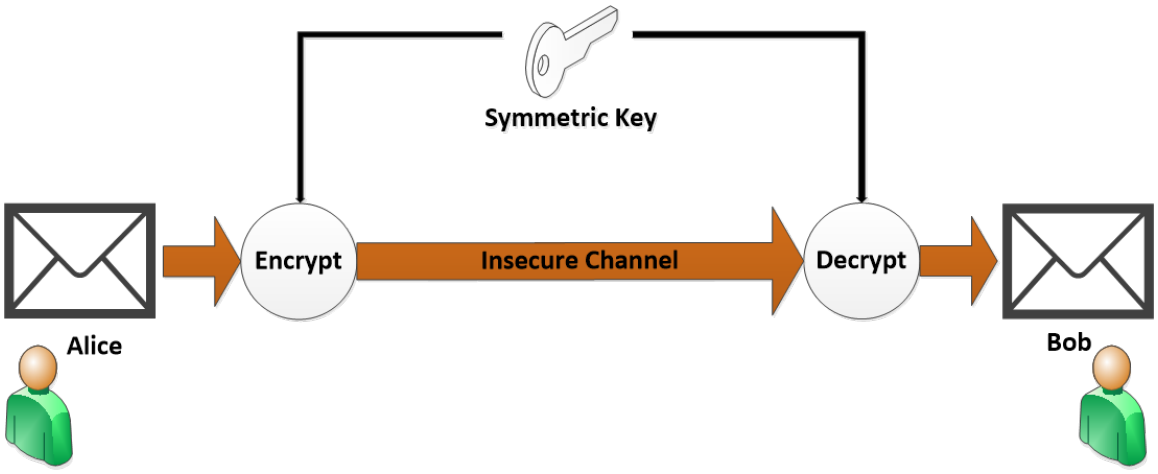

Figure 1.1: Symmetric key cryptography

1.2.1 Confusion and Diffusion

At the core of any cryptography algorithm are the properties confusion and diffusion. They come from Shannon's Communication theory of secrecy systems [1].

Diffusion is a linear transformation spread across the entire state. The purpose of it is to hide the relationship between the ciphertext and plaintext. If a single bit of plaintext changes, statistically half of the ciphertext should change.

Confusion is a non-linear transformation of small pieces of data. The purpose of it is to hide the relationship between the ciphertext and the key. In other words, each bit of the ciphertext should depend on multiple bits of the key.

1.2.2 Block and Stream Ciphers

There are two types of ciphers. The first type, a block cipher, encrypts fixed-sized blocks of plaintext symbols into ciphertext [2].

A stream cipher operates by encrypting one symbol of plaintext into ciphertext at a time. This is accomplished by adding a symbol from a key stream onto the plaintext symbol [2].

Compared to a stream cipher, a block cipher has higher diffusion since one symbol of plaintext is diffused across the entire block of ciphertext. However, it may be slower than a stream cipher since an entire block of plaintext must be accumulated before it can be encrypted.

1.2.3 Substitution Box

A Substitution Box (S-Box) is a component seen in many encryption algorithms and is often the main source of non-linear confusion. An S-box simply works by mapping an

input vector of length l to an output vector of length m [3]. The length of the input and the output do not have to be equal. S-Boxes are often implemented as tables.

Perhaps the most famous one is the S-Box seen in the Advanced Encryption standard (AES) [4]. The AES S-Box features Galois Field multiplicative inversion followed by an affine transformation.

The Advanced Encryption Standard was announced by the US National Institute of Standards and Technology (NIST) on Nov. 26 2001 [5]. NIST selected the candidate Rijndael designed by Vincent Rijmen and Joan Daemen out of 15 others to become the standard after a 5 year long selection process. It features a block size of 128 bits and three different key sizes: 128, 192, and 256 bits.

There has been a large amount of research focused on efficiently implementing the AES S-Box on hardware. Before AES and the candidate Rijndael became the symmetric key encryption standard, Paar [6] used tower composite fields for efficient Galois field arithmetic in 1994. Later, when AES became the standard, Satoh et al [7] in 2001 used tower composite fields to implement the AES S-Box. The change of basis matrices were further optimized by Canright between 2004 and 2005 [8, 9]. Later on in 2013, Wood provided further logic optimizations to the change of basis matrix and the composite fields [10].

1.2.4 Authenticated Encryption

The flaw to symmetric key cryptography used on it's own is that there is no guarantee regarding the authenticity of the message. In Fig. 1.2, Bob has no way of knowing if the received message truly arrived from Alice or if Oscar intercepted it and modified it.

To combat this, Message Authentication Code (MAC) can be embedded inside of the ciphertext. The MAC is a product of the key and either the plaintext or the ciphertext [3]. By doing so, any modification by Oscar to the ciphertext will produce a different MAC since he does not have the key. When Bob receives the message and checks the MAC, Bob will be aware that the authenticity of the message can be compromised.

1.3 Contribution

The main contribution of this thesis is an investigation into the area and performance trade offs of design choices on the S-Box of the Authenticated Encryption algorithm MK-3. The S-Box featured inside of MK-3 is 16-bits wide which is twice the width of most other S-Boxes used in cryptography. Due to the size of this, it stands to benefit from a hardware design intended for an FPGA implementation. This work found two candidates: one that has an optimal clock frequency of 125 MHz and another with the smallest area of 271 Look Up Tables. Both of these exceed the frequency of the ASIC optimized implementation

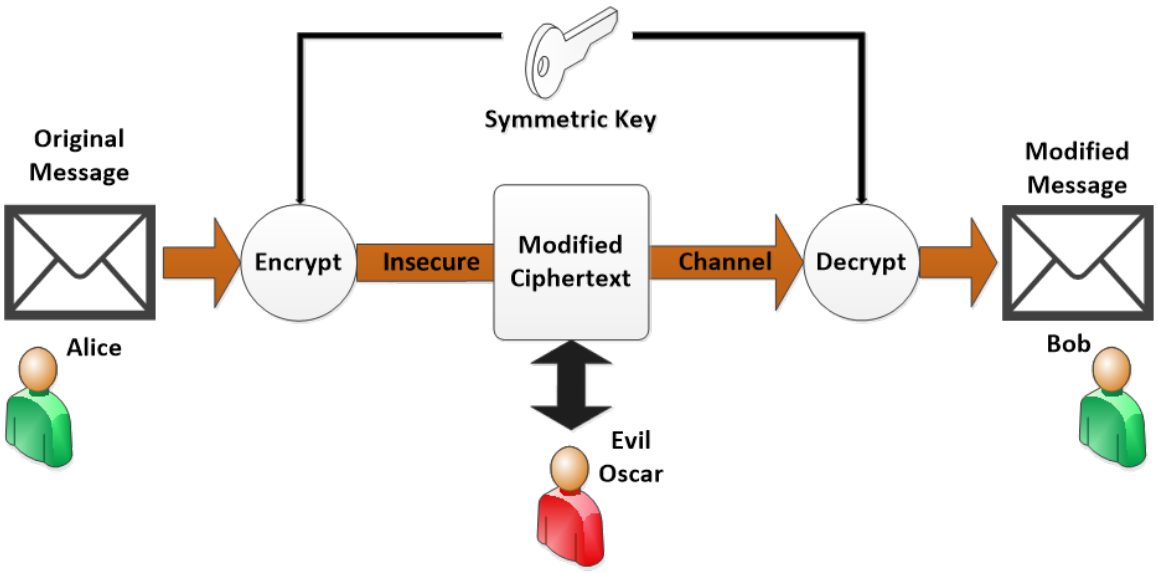

Figure 1.2: The need for authenticated encryption

chosen by Wood [10] and implemented on an FPGA by Werner [11]. The latter one also improves upon the area utilization. The list of specific contributions are listed below:

- Analysis of the effect of a normal and polynomial basis extension on the MK-3 S-Box
- Analysis of the effect of the base field used for the composite MK-3 S-Box
- Analysis of the effect of a hierarchical or flattened implementation of the sub field operations
- Consolidated overview of composite Galois field arithmetic pertaining to MK-3 S-Boxes
- Python scripts for finding optimal composite candidates
- Python scripts for generating composite, Galois field inversion and multiplication

The thesis is organized as follows. Chapter 3 provides a background on Galois field arithmetic that is the basis for the substitution box studied in this thesis. Next, Chapter 2 introduces the Authenticated Encryption Algorithm MK-3. Chapter 4 discusses how MK-3 can be implemented on an FPGA. The scripting infrastructure used to find optimal implementation candidates and generate the RTL is shown in Chapter 5. Chapter 6 analyzes the results of the FPGA implementations. Finally, a conclusion is offered in Chapter 7 as well as suggestions for future work.

Chapter 2

MK-3

MK-3 is an authenticated encryption algorithm developed through a joint effort between the Rochester Institute of Technology and the former Harris Corporation, now known as L3Harris [12, 13, 14, 15].

This AE cipher uses a duplex sponge to simultaneously absorb input and squeeze output shown in Figure 2.1.This is a modification to the original sponge that allows the algorithm to both absorb input and squeeze output simultaneously [12]. The MK-3 core is largely composed of binary XOR, AND, and shift operations which yields itself very well to hardware implementations [11].

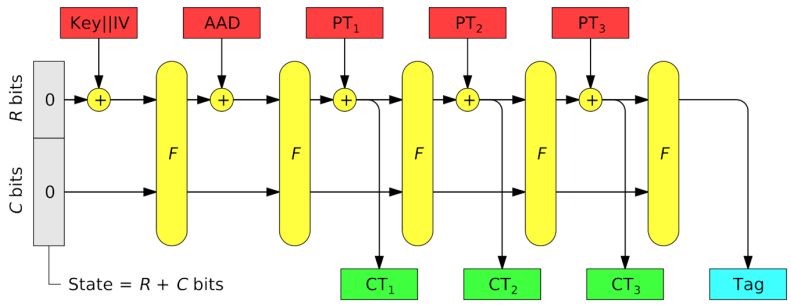

Figure 2.1: MK-3 duplex construction utilizing the MK-3 bijective function (F) [12]

The bijective function (F) shown in Figure 2.2 used in MK-3 is composed of 4 stages: Substitution (S), Bit Permutation (π), Mixing (M), and Round Key ($\bigoplus$). The bijective function operations across 512 bits as a whole.

Substitution	Operates across 32 groups of 16 bits. Galois field multiplicative inversion is performed on each 16 bit group.
Permutation	Spans the full 512 bits, the position of each bit is swapped.
Mixing	Operates across 16 groups of 32 bits. Galois field multiplication is performed between the left and right halves of each group.
Round Key	A round key is XORed to create the output of the function.

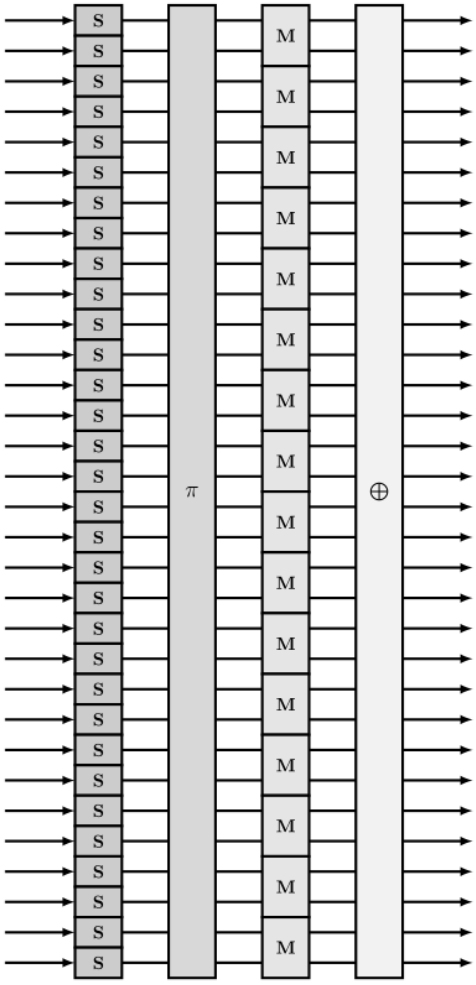

Figure 2.2: MK-3 bijective function [16]

The substitution stage in the MK-3 bijective function show as 'S' in Fig 2.2 performs Galois field multiplicative inversion followed by an affine transformation. There are 32 S-boxes which each operate on 16-bits of the state at a time. The multiplicative inversion is non-linear and is the primary source of confusion in the algorithm. The Galois field featured in MK-3 is from Christopher Wood's thesis [10]. It was chosen due to it's efficient hardware implementation on ASICs, requiring only 1238 XOR gates and 144 AND gates [10]. The irreducible polynomial, $p(x)$ is

$$p(x) = x^{16} + x^5 + x^3 + x + 1$$

The forward S-Box function is given as

$$S(x) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{15} \\ x_{14} \\ x_{13} \\ x_{12} \\ x_{11} \\ x_{10} \\ x_9 \\ x_8 \\ x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix}^{-1} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

And the inverse S-Box function is

$$S^{-1} = \left(\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{15} \\ x_{14} + 1 \\ x_{13} \\ x_{12} \\ x_{11} \\ x_{10} + 1 \\ x_9 \\ x_8 + 1 \\ x_7 + 1 \\ x_6 \\ x_5 + 1 \\ x_4 + 1 \\ x_3 \\ x_2 + 1 \\ x_1 + 1 \\ x_0 + 1 \end{bmatrix} \right)^{-1}$$

The second stage of the MK-3 bijective function is the permutation, shown as 'π' in Fig

2.2. It reorders the state according to the function

$$y = 31x + 15 \mod (512)$$

This is the primary source of diffusion as it shuffles the bit order across the entire state. The output of each S-box will be even spread across every mixer in the following stage.

The mixers are in the third stage, shown as 'M' in Fig 2.2. They are responsible for short range diffusion across two adjacent 16-bit blocks (A, B) by multiplying with the following matrix [16].

$$\begin{bmatrix} A' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & x \\ x & x+1 \end{bmatrix} \times \begin{bmatrix} A \\ B \end{bmatrix}$$

Resulting in

$$A' = A + Bx$$
$$B' = Ax + Bx + B$$

Since the only multiplication in the mixing stage is by x, the entire operation only requires shifting the coefficients and binary XORs for hardware implementations.

The final stage is the round key, shown as $\bigoplus$ in Fig 2.2. A 512 bit round key constant is added to the state to inject asymmetry. The round key is different for each round. The Keccak hash of the round number in ASCII is used.

Chapter 3

Fields

A field $\mathbb{F} = (F, +, \cdot, 0, 1)$ is an algebra on a set of elements F, where addition (+) and multiplication ($\cdot$) with neutral elements 0 and 1, and domains F and $F \setminus \{0\}$, respectively, are groups . The elements must also be mutually exclusive ($0 \neq 1$). Inside the field $\mathbb{F}$, the addition and multiplication operations satisfy the following [17]:

Identity:	For all a in $\mathbb{F}$
	$a + 0 = a$ and $a \cdot 1 = a$
Commutativity:	For all a, b in $\mathbb{F}$, $a + b = b + a$ and $a \cdot b = b \cdot a$
Associativity:	For all a, b, c in $\mathbb{F}$,
	$a + (b + c) = (a + b) + c$
	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Distributivity:	For all a, b, c in $\mathbb{F}$,
	$a \cdot (b + c) = a \cdot b + a \cdot c$
	$(b + c) \cdot a = b \cdot a + c \cdot a$
Inverses:	For all a in $\mathbb{F}$, $-a$ exists and $a + (-a) = 0$
	For all nonzero a in $\mathbb{F}$, a^{-1} exists and $a \cdot a^{-1} = 1$

The classical examples of fields are real numbers, complex numbers, or rational numbers, with the standard operations $+, \cdot$. These are the typically mentioned fields with an infinite number of elements. The remainder of this thesis will only use finite fields. One of the most remarkable results in abstract algebra is that the structure and properties of all finite fields can be summarized in an elegant and compact way, and then used in effective ways in many areas of computing. This is what finite Galois fields are, as described in the following sections.

3.1 Finite Galois Fields

Theorem: For every prime number p and positive integer m, there exists a finite field with p^m elements for every isomorphism, denoted $GF(p^m)$. Furthermore no other finite fields exist [17].

For $m = 1$ the above reduces prime order Galois fields, which in this case are essentially equivalent to modular arithmetic modulo p. In cryptography, telecommunication, and other computing applications the most common case is that of so called binary fields, where $p = 2$. The remainder of this document will only deal with binary Galois fields, i.e. with $GF(2^m)$. From this perspective, $GF(2^m)$ will be considered degree m extension of the binary $GF(2)$ field. However, if $m = st$ is composite, then it can be seen that $GF(2^{st}) = GF((2^s)^t)$ as degree t extension of the base field $GF(2^s)$. These two perspectives lead to isomorphic (logically equivalent) representations of the same field, but as will be seen in detail in the following, the algorithms implementing basic arithmetic operations within them have quite different characteristics.

The two most common ways to represent the fields $GF(2^m)$, and design and implement algorithms operating within them, are by using *polynomial bases* or by using *normal bases*. These are summarized in the following.

Consider any polynomial $P(x)$ of degree m over the field $F_2 = GF(2)$ of the form

$$P(x) = a_m x^m + a_{m-1} x^{m-1} + \cdot + a_1 x + a_0, \tag{3.1}$$

where $a_i \in F_2$, and x is a formal variable. The set of such polynomials is denoted by $F_2[x]$. The polynomial $P(x) \in F_2[x]$ is called an $irreducible$ over F_2 if and only if it cannot be factored into two polynomials in $F_2[x]$ both of degree less than m. If $P(x)$ is irreducible then it can play a role similar to prime numbers in integer modular arithmetic. Remainders of the division of any polynomial in $F_2[x]$ by $P(x)$ are considered, which must be polynomials of degree at most $m - 1$. The set of polynomials in $F_2[x]$ taken modulo $P(x)$, together with arithmetic defined on them, is isomorphic to the unique Galois field $GF(2^m)$. These remarks make a foundation of the most common representation of $GF(2^m)$ in *polynomial bases*. If in addition some algebraical properties of special elements of the field are a starting point, then a representation of $GF(2^m)$ in the *normal bases* can be built. The details of both are presented in the next subsections.

The following sections incorporate the notation Pm and Nm to describe $GF(2^m)$ fields using polynomial and normal basis representations respectively.

3.2 Polynomial Basis for Binary Galois Fields

In the basic field F_2 addition is the same as the standard XOR operation and multiplication is a binary AND. Both operations equivalently can be seen as as addition and multiplication modulo 2.

F_2 can be extended to $GF(2^m)$ by using polynomials in $F_2[x]$, taken modulo a degree m irreducible polynomial $P(x)$. This means that each element of $GF(2^m)$ is represented as a polynomial in $F_2[x]$ of degree less than m. Suppose there are two such polynomials $A(x)$ and $B(x)$, then their sum (addition) $S(x)$ and product (multiplication) $C(x)$ are expressed

as follows:

$$\begin{aligned} A(x) &= a_{m-1}x^{m-1} + ... + a_1x + a_0, \\ B(x) &= b_{m-1}x^{m-1} + ... + b_1x + b_0, \\ S(x) &= A(x) + B(x) \\ &= (a_{m-1} + b_{m-1})x^{m-1} + ... + (a_1 + b_1)x + (a_0 + b_0) \end{aligned} \tag{3.2}$$

$$C(x) = A(x) \times B(x) = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j x^{i+j} \mod P(x) \tag{3.3}$$

Addition simply involves adding the corresponding coefficients modulo 2 to obtain the sum. Multiplication requires reduction modulo $P(x)$ after the standard summation expressions. Note that in the special case shown below when $B(x) = x$, which is important in some applications:

$$\begin{aligned} C(x) &= A(x) \times x \mod P(x) \\ &= (a_{m-1}x^{m-1} + ... + a_1x + a_0) \times x \mod P(x) \\ &= \begin{cases} a_{m-2}x^{m-1} + ... + a_1x^2 + a_0x, & \textit{if } a_{m-1} = 0, \\ (x^m + P(x)) + a_{m-2}x^{m-1} + ... + a_1x^2 + a_0x, & \textit{if } a_{m-1} = 1. \end{cases} \end{aligned} \tag{3.4}$$

Observe that since $P(x)$ is a polynomial of degree m, then its highest term cancels with the term x^m, and thus the sum $x^m + P(x)$ is a polynomial of degree at most $m - 1$.

3.3 Normal Basis for Binary Galois Fields

The element Y of the field $GF(2^m)$ is called *normal* if and only if the set

$$\mathcal{B} = \{Y, Y^2, Y^{2^2}, \cdots, Y^{2^{m-1}}\}$$

is linearly independent. In other words, for the polynomial

$$A(Y) = a_{m-1}Y^{2^{m-1}} + a_{m-2}Y^{2^{m-2}} + ... + a_1Y^2 + a_0Y$$

it holds that $A(Y) = 0$ if and only if $a_i = 0$ for all $0 \leq i < m$. If $\mathcal{B}$ is linearly independent, then it can be used as a basis in which every element $\alpha \in GF(2^m)$ is uniquely represented by the vector $(a_{m-1}, \cdots, a_0)$ so that $A(Y) = \alpha$. This is called normal basis representation of the field. Every nonzero element α, and in particular every normal element Y, must have

Y	010	011	100	101	110	111
Y^2	100	101	110	111	010	011
Y^4	110	111	010	011	100	101
normal		✓		✓		✓

Table 3.1: Verification of the candidates for normal elements.

the order (the smallest $k > 0$ for which $\alpha^k = 1$) which is a divisor of $2^m - 1$. Thus

$$YY^2Y^{2^2} \cdots Y^{2^{m-1}} = 1.$$

Next, observe that each next element of the basis $\mathcal{B}$ is the square of the previous one $(Y^{2^i})^2 = Y^{2^{i+1}}$, with wrapping at the lowest power $Y = (Y^{2^{m-1}})^2$.

Ex 3.3.1. Consider $GF(2^3)$ using the irreducible polynomial $x^3 + x + 1$. There are only 6 normal element candidates since 0 and 1 are ineligible: x, $x + 1$, x^2, $x^2 + 1$, $x^2 + x$, and $x^2 + x + 1$. The coefficients of the set $\mathcal{B}$ are shown in Table 3.1 for each of the candidates.

In order to be a normal element, the set $\mathcal{B}$ must be linearly independent. This disqualifies x, x^2, and $x^2 + x$. The remaining three candidates for normal elements actually lead to the same elements in $\mathcal{B}$, only a different order. In this case, using $x + 1$ for Y, the normal basis can be constructed as shown below

$$\begin{aligned} Y &= x + 1 \\ Y^2 &= x^2 + 1 \\ Y^4 &= x^2 + x + 1 \end{aligned}$$

which in matrix form is

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{3.5}$$

These equations and the matrix can be inverted to

$$\begin{aligned} 1 &= Y^4 + Y^2 + Y \\ x &= Y^4 + Y^2 \\ x^2 &= Y^4 + Y \end{aligned}$$

and

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Addition using normal basis $\mathcal{B}$ is essentially the same as polynomial basis addition. Here, it involves coordinate-wise addition (in F_2, which is the same as XOR) of vectors $(a_{m-1}, \cdots, a_0)$ and $(b_{m-1}, \cdots, b_0)$, which represent $\alpha = A(Y)$ and $\beta = B(Y)$ in the normal basis $\mathcal{B}$:

$$\begin{aligned} A(Y) &= a_{m-1}Y^{2^{m-1}} + ... + a_1Y^2 + a_0Y \\ B(Y) &= b_{m-1}Y^{2^{m-1}} + ... + b_1Y^2 + b_0Y \\ S(Y) &= A(Y) + B(Y) = \alpha + \beta = (s_{m-1}, \cdots, s_1, s_0) \\ S(Y) &= (a_{m-1} + b_{m-1})Y^{2^{m-1}} + ... + (a_1 + b_1)Y^2 + (a_0 + b_0)Y \end{aligned} \tag{3.7}$$

For multiplication, by elementary algebra, the product $\gamma = C(Y) = A(Y) \times B(Y)$ can be expressed as follows

$$C(Y) = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j Y^{2^i + 2^j}.$$

The next step is to represent the result of the last expression when evaluated in $GF(2^m)$ using the normal basis $\mathcal{B}$. For this, the terms $Y^{2^i+2^j}$ need to be expressed in basis $\mathcal{B}$, then just use normal basis addition as above.

Ex 3.3.2. Using $GF(2^4)$ represented by an irreducible polynomial $P(x) = x^4 + x + 1$ and a normal element $Y = x^3$. Next, the elements of the normal basis, Y^{2^i} are expressed using polynomial arithmetic modulo $P(x)$:

$$\begin{aligned} Y^8 &= x^3 + x \\ Y^4 &= x^3 + x^2 + x + 1 \\ Y^2 &= x^3 + x^2 \\ Y &= x^3 \end{aligned} \tag{3.8}$$

The cumulative effect of equations (3.8) can be expressed by a matrix M, which produces polynomials in x equivalent to Y^{2^i}, for all $i < m$.

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.9}$$

The matrix M can be inverted to:

$$M^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.10}$$

$$\begin{aligned} x^3 &= Y \\ x^2 &= Y^2 + Y \\ x &= Y^8 + Y \\ 1 &= Y^8 + Y^4 + Y^2 + Y \end{aligned} \tag{3.11}$$

which can be used to express polynomial basis elements using normal basis. Note that all powers Y^k can be represented this way. For example, for $k = 3$

$$Y^3 = Y^2 \times Y = (x^3 + x^2)x^3 = x^6 + x^5 = x^3 + x \mod P(x).$$

The normal representation of all Y^k terms can be found by converting the polynomial representation to the normal basis using the entries shown in Table 3.2. Observe that the sequence $Y, Y^2, Y^8, Y^4, Y^8 + Y^4 + Y^2 + Y$ repeats across the powers of Y.

Next, going back to the multiplication problem, (3.9) can be used to express the result in normal basis. The actual multiplication of two polynomials

$$\begin{aligned} A(Y) &= a_3Y^8 + a_2Y^4 + a_1Y^2 + a_0Y, \\ B(Y) &= b_3Y^8 + b_2Y^4 + b_1Y^2 + b_0Y, \text{ gives} \\ C(Y) &= A \times B \\ &= a_3b_3Y^{16} + a_3b_2Y^{12} + a_3b_1Y^{10} + a_3b_0Y^9 + \\ &\quad a_2b_3Y^{12} + a_2b_2Y^8 + a_2b_1Y^6 + a_2b_0Y^5 + \\ &\quad a_1b_3Y^{10} + a_1b_2Y^6 + a_1b_1Y^4 + a_1b_0Y^3 + \\ &\quad a_0b_3Y^9 + a_0b_2Y^5 + a_0b_1Y^3 + a_0b_0Y^2. \end{aligned} \tag{3.12}$$

Now, $C(Y)$ in this form is not represented using the normal basis. However, this is where

	Polynomial	**Normal**
$\boldsymbol{Y}$	x^3	Y
$\boldsymbol{Y^2}$	$x^3 + x^2$	Y^2
$\boldsymbol{Y^3}$	$x^3 + x$	Y^8
$\boldsymbol{Y^4}$	$x^3 + x^2 + x + 1$	Y^4
$\boldsymbol{Y^5}$	1	$Y^8 + Y^4 + Y^2 + Y$
$\boldsymbol{Y^6}$	x^3	Y
$\boldsymbol{Y^7}$	$x^3 + x^2$	Y^2
$\boldsymbol{Y^8}$	$x^3 + x$	Y^8
$\boldsymbol{Y^9}$	$x^3 + x^2 + x + 1$	Y^4
$\boldsymbol{Y^{10}}$	1	$Y^8 + Y^4 + Y^2 + Y$
$\boldsymbol{Y^{11}}$	x^3	Y
$\boldsymbol{Y^{12}}$	$x^3 + x^2$	Y^2
$\boldsymbol{Y^{13}}$	$x^3 + x$	Y^8
$\boldsymbol{Y^{14}}$	$x^3 + x^2 + x + 1$	Y^4
$\boldsymbol{Y^{15}}$	1	$Y^8 + Y^4 + Y^2 + Y$

Table 3.2: Powers of normal element Y represented in polynomial and normal bases.

the reduction can be performed using Table 3.2, which expresses each of the terms Y^k as a combination of normal basis elements Y^{2^i}. Recall that $Y^{2^m} = Y$, which means that exponents of Y can be reduced modulo $2^m - 1$ (in this example, modulo 15). Thus, $C(Y)$ can be further rewritten as follows

$$\begin{aligned} C(Y) =& a_3b_3Y + a_3b_2Y^2 + a_3b_1(Y^8 + Y^4 + Y^2 + Y) + a_3b_0Y^4 + \\ & a_2b_3Y^2 + a_2b_2Y^8 + a_2b_1Y + a_2b_0(Y^8 + Y^4 + Y^2 + Y) + \\ & a_1b_3(Y^8 + Y^4 + Y^2 + Y) + a_1b_2Y + a_1b_1Y^4 + a_1b_0Y^8 + \\ & a_0b_3Y^4 + a_0b_2(Y^8 + Y^4 + Y^2 + Y) + a_0b_1Y^8 + a_0b_0Y^2, \end{aligned} \tag{3.13}$$

and reorganized according to powers of Y as

$$\begin{aligned} C(Y) =& (a_0b_1 + a_0b_2 + a_1b_0 + a_1b_3 + a_2b_0 + a_2b_2 + a_3b_1)Y^8 \\ & (a_0b_2 + a_0b_3 + a_1b_1 + a_1b_3 + a_2b_0 + a_3b_0 + a_3b_1)Y^4 \\ & (a_0b_0 + a_0b_2 + a_1b_3 + a_2b_0 + a_2b_3 + a_3b_1 + a_3b_2)Y^2 \\ & (a_0b_2 + a_1b_2 + a_1b_3 + a_2b_0 + a_2b_1 + a_3b_1 + a_3b_3)Y. \end{aligned} \tag{3.14}$$

In general, the conversion between normal and polynomial basis representations occurs through a linear isomorphism defined by matrix M, as shown in examples (3.9) and (3.10). The elements of M are from the base field, which so far was binary $F_2 = GF(2)$, though this will be generalized to $GF(2^s)$ here and in the next section, where the composite

representations of binary Galois fields are discussed.

An important property of normal basis representation (in the basic case $F_2, s = 1$, but also in composite case with $s > 1$) is that raising $A(Y) = a_{m-1}Y^{n^{m-1}} + ... + a_1Y^n + a_0Y$ to power $n = 2^s$ (which is squaring in the basic case $n = 2$, and raising to powers of the form 2^s in general) can be reduced to a circular left shift. Observe that the scalar coefficients a_i of $A(Y)$ are in $GF(n) = GF(2^s)$, and thus they satisfy the Fermat-type property $a_i^n = a_i$ and the parity cancellation rule $a_i + a_i = 0$. Note also that mixed power terms in the expansion of $A(Y)^n$ will vanish because of the latter. This gives:

$$\begin{aligned} A(Y)^n &= a_{m-1}^n Y^{n^m} + a_{m-2}^n Y^{n^{m-1}} + ... + a_1^n Y^{n^2} + a_0^n Y^n \\ &= a_{m-1}Y + a_{m-2}Y^{n^{m-1}} + ... + a_1 Y^{n^2} + a_0 Y^n \\ &= a_{m-2}Y^{n^{m-1}} + ... + a_1 Y^{n^2} + a_0 Y^n + a_{m-1}Y. \end{aligned} \tag{3.15}$$

3.4 Composite Fields

The Galois fields of the form $GF(p^m)$, where p is a prime and $m = st$ is a composite number, can be represented by a special two-stage extension process. First, the field $GF(p^s)$ is built on an irreducible polynomial of degree s over $GF(p)$, then it is extended to $GF(p^m)$ by using degree-t extension of $GF(p^s)$. For this second step an irreducible polynomial of degree t over $GF(p^s)$ is needed. Such constructions are leading to what is called *composite field representations*. If m has $k > 2$ factors, then it can be represented by the final $GF(p^m)$ using k consecutive extensions starting from the base field $GF(p)$.

By theorem 1 in section 3.1, if the number of elements in the field is fixed, then all constructions must lead to the same field up to isomorphism. This is what is exploited in this section: for fixed p^m, computations in $GF(p^m)$ performed in different constructions of the field, yet under properly defined isomorphisms they must describe the same computations. This is utilized to find more efficient ways to evaluate the standard operators in $GF(p^m)$. An isomorphism between the two fields can be used to interchange between the two. Arithmetic performed in the standard polynomial representation of the field is equivalent to converting to the composite field, performing the operation in it, and converting back.

Ex 3.4.1. For a standard field $GF(2^4)$ with $R(z) = z^4 + z + 1$ and a composite field $GF((2^2)^2)$ with $Q(x) = x^2 + x + 1$ and $P(y) = 01y^2 + 01y + 10$. The isomorphism c can be $01y + 10$.

$$\begin{aligned} R(c) &= (01y + 10)^4 + (01y + 10) + 01 \\ &= (01y^2 + 100)^2 + 01y + 11 \end{aligned}$$

The 100 term is outside of the bounds of $Q(x)$, it can be reduced by XORing with the irreducible, yielding $x + 1$ or 11.

$$= (01y^2 + 11)^2 + 01y + 11$$

The y^2 term also needs to be reduced by XORing with $P(y)$ to yield $01y + 10$.

$$\begin{aligned} &= ((01y + 10) + 11)^2 + 01y + 11 \\ &= (01y + 01)^2 + 01y + 11 \\ &= 01y^2 + 01 + 01y + 11 \\ &= (01y + 10) + 01y + 10 \\ &= 0 \end{aligned}$$

Therefore, $c = 01y + 10$ is a valid isomorphism between the two representations. Next, the powers of c are computed as below and placed into a the matrix in 3.16.

$$\begin{aligned} c^0 &= 00y + 01 \\ c^1 &= 01y + 10 \\ c^2 &= 01y + 01 \\ c^3 &= 10y + 00 \end{aligned}$$

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.16}$$

This M_1 contains the corresponding composite term of each individual scalar coefficient. This can be inverted as in 3.17 to covert from composite to scalar representation.

$$M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.17}$$

As a test of M_1 and M_1^{-1}, z of P4 can be converted to P2P2

$$M_1 z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = 01y + 10$$

squared,

$$(01y + 10)^2 = 01y^2 + 10y + 10y + 100 = (01y + 10) + 11 = 01y + 01$$

And then converted back to P4.

$$M_1^{-1} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = z^2$$

3.5 Mixing Bases in a Composite Field

The above example was exclusively in the polynomial basis. The conversion matrix in (3.17) can be combined with a normal basis transformation so that the composite representation uses the normal basis.

Ex 3.5.1. Using the same fields as in the previous example, a normal element of $GF((2^2)^2)$, $Y = 01y + 01$ can be used to transform the extension field to the normal basis, $P2N2$.

$$Y^1 = 01y + 01$$
$$Y^2 = 01y + 11$$
$$Y^4 = 01y + 00$$

Since Y belongs to $GF((2^2)^2)$, the conversion matrix M_2 will be a 2×2 matrix consisting of $GF(2^2)$ elements.

$$M_2 = \begin{bmatrix} 01 & 01 \\ 00 & 01 \end{bmatrix} \tag{3.18}$$

The inverse of M_2, used to convert back to the polynomial composite representation happens to be equal to M_2.

As a test $01y + 10$ can be converted to P2N2,

$$M_2(01y + 10) = \begin{bmatrix} 01 & 01 \\ 00 & 01 \end{bmatrix} \begin{bmatrix} 01 \\ 10 \end{bmatrix} = \begin{bmatrix} 11 \\ 10 \end{bmatrix} = 11^4 + 10Y$$

squared,

$$\begin{aligned} (11^4 + 10Y)^2 &= 101Y^8 + 110Y^5 + 110Y^5 + 100Y^2 \\ &= 10Y^8 + 11Y^2 = 01(11Y^4 + 10Y) + 11(10Y^4 + 11Y) \\ &= 110Y^4 + 100Y + 110Y^4 + 101Y = 10Y^4 + 11Y + 01Y^4 + 10Y \\ &= 01Y \end{aligned}$$

And then converted back to P2P2.

$$M_2^{-1} \begin{bmatrix} 00 \\ 01 \end{bmatrix} = \begin{bmatrix} 01 & 01 \\ 00 & 01 \end{bmatrix} \begin{bmatrix} 01 \\ 01 \end{bmatrix} = \begin{bmatrix} 01 \\ 01 \end{bmatrix} = 01y + 01$$

The composite representation of an element in the normal extension field now uses the form

$$\delta_1 Y^4 + \delta_0 Y \tag{3.19}$$

where $\delta \in GF(2^2)$ This can be can be reconfigured using $\gamma \in GF(2)$ such that

$$\begin{aligned} &= (\gamma_3 x + \gamma_2)Y^4 + (\gamma_1 x + \gamma_0)Y \\ &= \gamma_3 x Y^4 + \gamma_2 Y^4 + \gamma_1 x Y + \gamma_0 Y \\ &= \gamma_3(10Y^4) + \gamma_2(01Y^4) + \gamma_1(10Y) + \gamma_0(01Y) \end{aligned} \tag{3.20}$$

The Y terms can then be transformed using M_2^{-1} so that the outer field uses the polynomial basis. Now that the representation uses the polynomial basis for both fields, it can be

brought back to the original $GF(2^4)$ representation by applying M_1^{-1}.

$$\begin{aligned} 10Y^4 =& 10y + 00 = z^3 \\ 01Y^4 =& 01y + 00 = z^2 + 1 \\ 10Y =& 10y + 10 = z^3 + z^2 + z + 1 \\ 01Y =& 01y + 01 = z^2 \end{aligned} \tag{3.21}$$

Replacing the Y terms in (3.20) with (3.21) yields

$$\gamma_3(z^3) + \gamma_2(z^2 + 1) + \gamma_1(z^3 + z^2 + z + 1) + \gamma_0(z^2) \tag{3.22}$$

M_3^{-1} can be constructed using (3.22) to transform from the mixed basis composite form back to the polynomial basis $GF(2^4)$ form.

$$M_3^{-1} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.23}$$

$$\begin{bmatrix} z_3 \\ z_2 \\ z_1 \\ z_0 \end{bmatrix} = M_3^{-1} \begin{bmatrix} 10Y^4 \\ 10Y \\ Y^4 \\ Y \end{bmatrix} \tag{3.24}$$

Inverting this yields the final part used to change the representation to the mixed basis composite form

$$M_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.25}$$

$$\begin{bmatrix} 10Y^4 \\ 10Y \\ Y^4 \\ Y \end{bmatrix} = M_3 \begin{bmatrix} z_3 \\ z_2 \\ z_1 \\ z_0 \end{bmatrix} \tag{3.26}$$

As a test of M_3 and M_3^{-1}, z of P4 can be converted to P2N2

$$M_3 z = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = 11Y^4 + 10$$

squared,

$$(11Y^4 + 10Y)^2 = 01Y$$

And then converted back to P4.

$$M_1^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = z^2$$

Ex 3.5.2. For a standard field $GF(2^8)$ with $R(z) = z^8 + z^4 + z^3 + z + 1$ and a composite field $GF((2^4)^2)$ with $Q(x) = x^4 + x + 1$ and $P(y) = 0001y^2 + 0001y + 1000$ a combined set of change of basis matrices can be constructed to go directly from P8 to P4N2. The isomorphism $c = 0010y + 0000$ and norm $Y = 0001y + 0000$ will be used.

M_1 can be constructed with c and will convert from P8 to P4P2

$$M_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

Inverting M_1 will yield the transformation back to P8.

$$M_1^{-1} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

As a test of M_1 and M_1^{-1}, z of P8 can be converted to P4P2

$$M_1 z = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = 0010y + 0000$$

squared,

$$(0010y + 0000)^2 = 0100y^2 = 0100(y + 1000) = 0100y + 0110$$

And then converted back to P8.

$$M_1^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = z^2$$

Y and Y^{16} are used to construct M_2 and M_2^{-1} which will provide the conversion between P4P2 and P4N2.

$$\begin{aligned} Y &= 0001y + 0000 \\ Y^2 &= 0001y + 1000 \\ Y^4 &= 0001y + 0100 \\ Y^8 &= 0001y + 1011 \\ Y^{16} &= 0001y + 0001 \end{aligned}$$

M_2^{-1} transforms P4N2 to P4P2.

$$M_2^{-1} = \begin{bmatrix} 0001 & 0001 \\ 0001 & 0000 \end{bmatrix} \tag{3.29}$$

M_2 transforms P4P2 to P4N2.

$$M_2 = \begin{bmatrix} 0000 & 0001 \\ 0001 & 0001 \end{bmatrix} \tag{3.30}$$

As a test, $0010y + 0000$ can be converted to P4N2,

$$M_2(0010y + 0000) = \begin{bmatrix} 0000 & 0001 \\ 0001 & 0001 \end{bmatrix} \begin{bmatrix} 0010 \\ 0000 \end{bmatrix} = \begin{bmatrix} 0000 \\ 0010 \end{bmatrix} = 0000Y^{16} + 0010Y$$

squared,

$$\begin{aligned} (0010Y)^2 = 0100Y^2 &= 0100(1000Y^{16} + 1001Y) \\ &= 0110Y^{16} + 0010Y \end{aligned}$$

And then converted back to P4P2.

$$M_2^{-1} \begin{bmatrix} 0110 \\ 0010 \end{bmatrix} = \begin{bmatrix} 0001 & 0001 \\ 0001 & 0000 \end{bmatrix} \begin{bmatrix} 0110 \\ 0010 \end{bmatrix} = \begin{bmatrix} 0100 \\ 0110 \end{bmatrix} = 0100y + 0110$$

Using the same procedure as the previous example, M_3^{-1} can be constructed by converting each P4N2 coefficient to P4P2 and then to P8.

$$
\begin{aligned}
0000Y^{16} + 0001Y &= 0001y + 0000 = z^7 + z^5 + z \\
0000Y^{16} + 0010Y &= 0010y + 0000 = z \\
0000Y^{16} + 0100Y &= 0100y + 0000 = z^7 + z^5 + z^4 + z^3 \\
0000Y^{16} + 1000Y &= 1000y + 0000 = z^7 + z^6 + z^4 + z^3 + z + 1 \\
0001Y^{16} + 0000Y &= 0001y + 0001 = z^7 + z^5 + z + 1 \\
0010Y^{16} + 0000Y &= 0010y + 0010 = z^6 + z^4 + z^3 + z^2 + z \\
0100Y^{16} + 0000Y &= 0100y + 0100 = z^6 + z^4 + z^3 \\
1000Y^{16} + 0000Y &= 1000y + 1000 = z^7 + z^3 + z + 1
\end{aligned}
$$

Assembling the above results into a matrix yields the transformation from P4N2 to P8.

$$
M_3^{-1} = \begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0
\end{bmatrix} \tag{3.31}
$$

By inverting M_3^{-1} the transformation from P8 to P4N2 can be found.

$$
M_3 = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1
\end{bmatrix} \tag{3.32}
$$

As a test of M_3 and M_3^{-1}, z of P8 can be converted to P4N2

$$M_3 z = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = 0000Y^{16} + 0010Y$$

squared,

$$(0010Y)^2 = 0110Y^{16} + 0010Y$$

And then converted back to P8.

$$M_3^{-1} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = z^2$$

3.6 Composite Field Hardware Derivation

The composite fields used in this thesis feature extensions of degree 2. The irreducible polynomials used will follow the form shown in (3.33) where there is a trace component τ and a norm component ν.

$$Q(y) = y^2 + \tau y + \nu \tag{3.33}$$

The derivation of the composite field operations in the following sections was derived from [8, 10]. The term ϵ will be used for the top field and δ will be used for the base field.

3.6.1 Polynomial Basis Multiplication

Given two elements ϵ_1 and ϵ_2, the multiplication begins by first separating the components into their composite representations and distributing them.

$$\begin{aligned}\epsilon_3 &= \epsilon_1 \times \epsilon_2 \\ \delta_5 x + \delta_6 &= (\delta_1 x + \delta_1)(\delta_3 x + \delta_4) \\ &= \delta_1\delta_3 x^2 + \delta_1\delta_4 x + \delta_2\delta_3 x + \delta_2\delta_4\end{aligned}$$

The x^2 term can be reduced by XORing with the irreducible $P(x)$.

$$\begin{aligned}&= \delta_1\delta_3(x + \nu) + \delta_1\delta_4 x + \delta_2\delta_3 x + \delta_2\delta_4 \\ &= (\delta_1\delta_3 + \delta_1\delta_4 + \delta_2\delta_3)x + (\delta_1\delta_3\nu + \delta_2\delta_4)\end{aligned}$$

This can then be rewritten by using the relation $(\delta_1 + \delta_2)(\delta_3 + \delta_4) = \delta_1\delta_3 + \delta_1\delta_4 + \delta_2\delta_3 + \delta_4$ on the x coefficients.

$$= (\delta_2\delta_4 + (\delta_1 + \delta_2)(\delta_3 + \delta_4))x + (\delta_1\delta_3\nu + \delta_2\delta_4)$$

Therefore, δ_5 and δ_6 can be solved for in order to obtain ϵ_3

$$\delta_5 = \delta_2\delta_4 + (\delta_1 + \delta_2)(\delta_3 + \delta_4) \tag{3.34}$$

$$\delta_6 = \delta_1\delta_3\nu + \delta_2\delta_4 \tag{3.35}$$

At this point, δ_5 and δ_6 share the term $\delta_2\delta_4$.

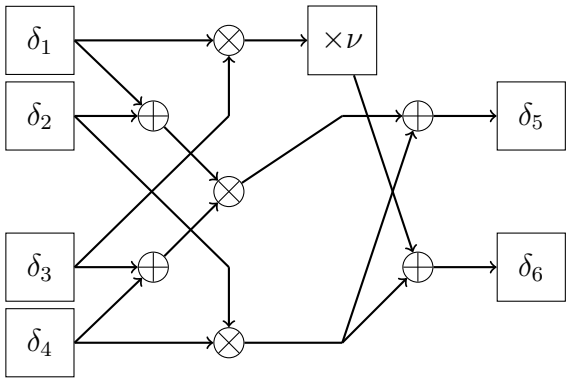

Figure 3.1: Polynomial basis composite field multiplication

3.6.2 Polynomial Basis Inversion

For the inversion derivation, the start is similar with the exception that ϵ_3 is known to be equal to 1 and the goal is to solve for ϵ_2 which is ϵ_1^{-1} in terms of ϵ_1.

$$\begin{aligned} 1 &= \epsilon_1 \times \epsilon_2 \\ &= (\delta_1 x + \delta_2)(\delta_3 x + \delta_4) \\ &= \delta_1\delta_3 x^2 + \delta_1\delta_4 x + \delta_2\delta_3 x + \delta_3\delta_4 \end{aligned} \tag{3.36}$$

In order for the inverse to exist, the right side of the equation must follow the form $k \times P(x) + 1 = k(x^2 + x + \nu) + 1 = 1$. Knowing this, the terms of the right side can be separated such that

$$k = \delta_1\delta_3 \tag{3.37}$$

$$k = (\delta_1\delta_4 + \delta_2\delta_3) \tag{3.38}$$

$$k\nu + 1 = \delta_2\delta_4 \tag{3.39}$$

By substituting 3.37 into 3.39

$$\delta_2\delta_4 = \nu\delta_1\delta_3 + 1 \tag{3.40}$$

And setting 3.37 equal to 3.38

$$\begin{aligned} \delta_1\delta_3 &= (\delta_1\delta_4 + \delta_2\delta_3) \\ \delta_1\delta_3 + \delta_2\delta_3 &= \delta_1\delta_4 \\ \delta_3(\delta_1\delta_2) &= \delta_1\delta_4 \\ \delta_3 &= \delta_1\delta_4(\delta_1 + \delta_2)^{-1} \end{aligned} \tag{3.41}$$

δ_3 has been solved for, but it still contains a δ_4 component that will need to be removed. δ_4 can be solved for by substituting 3.41 into 3.40.

$$\begin{aligned} \delta_2\delta_4 &= \nu\delta_1(\delta_1\delta_4(\delta_1 + \delta_2)^{-1}) + 1 \\ \delta_2\delta_4(\delta_1 + \delta_2) &= \nu\delta_1 + (\delta_1 + \delta_2) \\ \delta_2\delta_4 + \delta_1\delta_2\delta_4 + \delta_1^2\nu\delta_4 &= (\delta_1 + \delta_2) \\ \delta_4(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\nu) &= (\delta_1 + \delta_2) \\ \delta_4 &= (\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\nu)^{-1} \end{aligned} \tag{3.42}$$

This can then be substituted into 3.40 to obtain δ_3 in terms of δ_1 and δ_2 only.

$$\begin{aligned} \delta_3 &= \delta_1(\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\nu)^{-1}(\delta_1 + \delta_2)^{-1} \\ \delta_3 &= \delta_1(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\nu)^{-1} \end{aligned} \tag{3.43}$$

Therefore, the inverse of ϵ_1 has been solved for.

$$\begin{aligned}
\epsilon_1 &= \delta_3 x + \delta_4 \\
\delta_3 &= \delta_1(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\nu)^{-1} \\
\delta_4 &= (\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\nu)^{-1}
\end{aligned}$$

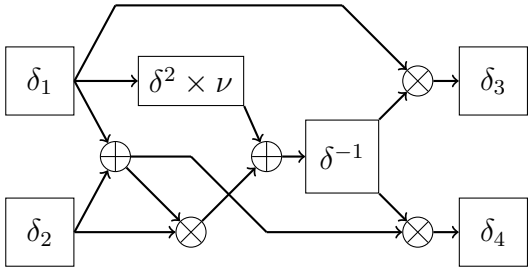

Figure 3.2: Polynomial basis composite field inversion

3.6.3 Polynomial Basis Square Scaling

The P4P2P2 and P4P2N2 inversion circuits perform P4P2 square-scaling inside. The scaling component (ϵ_2) is constant which leaves the squared component (ϵ_1^2) as the only variable.

$$\begin{aligned}
\epsilon_1 &= \epsilon_1^2 \times \epsilon_2 \\
&= (\delta_1 x + \delta_2)^2(\delta_3 + \delta_4) \\
&= (\delta_1^2 x^2 + \delta_2^2)(\delta_3 x + \delta_4) \\
&= (\delta_1^2 x + \delta_1^2\nu + \delta_2^2)(\delta_3 x + \delta_4) \\
&= \delta_1^2\delta_3 x^2 + \delta_1^2\delta_4 x + \delta_1^2\delta_3\nu x + \delta_1^2\delta_4\nu + \delta_2^2\delta_3 x + \delta_2\delta_4 \\
&= \delta_1^2\delta_3 x + \delta_1^2\delta_3\nu + \delta_1^2\delta_4 x + \delta_1^2\delta_3\nu x + \delta_1^2\delta_4\nu + \delta_2^2\delta_3 x + \delta_2\delta_4 \\
&= (\delta_1^2\delta_3 + \delta_1^2\delta_4 + \delta_1^2\delta_3\nu + \delta_2^2\delta_3)x + (\delta_1^2\delta_3\nu + \delta_1^2\delta_4\nu + \delta_2\delta_4) \qquad (3.44)
\end{aligned}$$

Since the scalar $(\delta_3 x + \delta_4)$ is known, the composite operations can be reduced if the scalar has certain properties. The first optimization that can be made is if δ_4 is equal to 0. This will allow for several terms to be removed.

$$= (\delta_1^2\delta_3 + \delta_1^2\delta_3\nu + \delta_2^2\delta_3)x + (\delta_1^2\delta_3\nu) \qquad (3.45)$$

The composite hardware can be further reduced if δ_3 is the inverse of nu. By doing so, several multiplications will cancel out.

$$= (\delta_1^2\nu^{-1} + \delta_1^2 + \delta_2^2\nu^{-1})x + (\delta_1^2) \tag{3.46}$$

This leaves two square-scaling operations, one squaring, and two additions.

3.6.4 Normal Basis Multiplication

If the irreducible polynomial $Q(y)$ was to have roots, they would be Y and Y^q. If the two roots are distributed, the irreducible can be rewritten as (3.47).

$$\begin{aligned} Q(y) &= y^2 + \tau y + \nu \\ &= (y + Y^q)(y + Y) \\ &= y^2 + (Y^q + Y)y + YY^q \end{aligned} \tag{3.47}$$

Upon comparison of eqs. (3.33) and (3.47), it can be said that the trace component τ is the sum of the two roots and the norm component ν is the product.

$$\tau = (Y^q + Y) \tag{3.48}$$
$$\nu = YY^q \tag{3.49}$$
$$1 = (Y^q + Y)\tau^{-1} \tag{3.50}$$

The trace and norm components belong to the sub field $GF(2^s)$ however Y and Y^q belong to the field formed by $GF((2^s)^t)$. Since $Y^q = Y^{2^s}$ the leading coefficient of Y and Y^q will be equal. Therefore the leading coefficients in YY^q and $Y + Y^q$ cancel each other out and equal zero. The norm and trace components can be expressed inside of the sub field.

The multiplication of ϵ_1 by ϵ_2 to obtain ϵ_3 begins by first separating the components into their composite representation and distributing the terms.

$$\begin{aligned} \epsilon_3 &= \epsilon_1 \times \epsilon_2 \\ (\delta_5 Y^q + \delta_6 Y) &= (\delta_1 Y^x + \delta_2 Y)(\delta_3 Y^q + \delta_4 Y) \\ &= \delta_1\delta_3(Y^q)^2 + (\delta_1\delta_4 + \delta_2\delta_3)Y^qY + \delta_2\delta_4 Y^2 \end{aligned}$$

The distribution has produced the terms $(Y^q)^2$, and Y^2 which are outside of the normal basis representation. These can be rewritten so that they exist in the normal basis using the

following derived from eqs. (3.48) and (3.49).

$$(Y^q)^2 = Y^q\tau + \nu \tag{3.51}$$
$$Y^2 = Y\tau + \nu \tag{3.52}$$
$$Y^qY = \nu \tag{3.53}$$

$$\begin{aligned}
&(\delta_5 Y^q + \delta_6 Y) \\
&\quad = \delta_1\delta_3(Y^q\tau + \nu) + (\delta_1\delta_4 + \delta_2\delta_3)\nu + \delta_2\delta_4(Y\tau + \nu) \\
&\quad = \delta_1\delta_3\tau Y^q + \delta_2\delta_4\tau Y + (\delta_1\delta_3\nu + (\delta_1\delta_4 + \delta_2\delta_4)\nu + \delta_2\delta_4\nu) \\
&\quad = \delta_1\delta_3\tau Y^q + \delta_2\delta_4\tau Y + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1}(Y^q + Y) \\
&\quad = (\delta_1\delta_3\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1})Y^q \\
&\qquad + (\delta_2\delta_4\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1})Y
\end{aligned} \tag{3.54}$$

Therefore

$$\delta_5 = \delta_1\delta_3\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1} \tag{3.55}$$
$$\delta_6 = \delta_2\delta_4\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1} \tag{3.56}$$

Equations (3.55) and (3.56) provide δ_5 and δ_6. One optimization to this is to remove the multiplication involving the trace component (τ). Since $\tau = Y^q + Y$, the norm Y can be selected such that the trace is equal to 1. By doing so, the multiplications by the trace have no effect on δ_5 and δ_6. Figure 3.3 shows the block diagram of the multiplier.

$$\delta_5 = \delta_1\delta_3 + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu] \tag{3.57}$$
$$\delta_6 = \delta_2\delta_4 + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu] \tag{3.58}$$

3.6.5 Normal Basis Inversion

Composite field inversion begins much in the same way as the multiplication with the exception that only ϵ_1 is known and the goal is to obtain ϵ_2 such that it is the inverse ϵ_1.

$$\begin{aligned}
1 &= \epsilon_1 \times \epsilon_2 \\
&= (\delta_1 Y^q + \delta_2 Y)(\delta_3 Y^x + \delta_4 Y) \\
&= \delta_1\delta_3(Y^q)^2 + (\delta_1\delta_4 + \delta_2\delta_3)Y^xY + \delta_2\delta_4 Y^2
\end{aligned}$$

Simplify using the relations in eqs. (3.51) to (3.53)

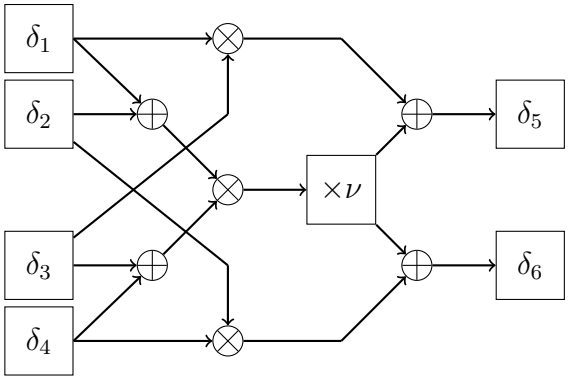

Figure 3.3: Normal basis composite field multiplication

$$\begin{aligned}
1 &= \delta_1\delta_3(Y^q\tau + \nu) + (\delta_1\delta_4 + \delta_2\delta_3)\nu + \delta_2\delta_4(Y\tau + \nu) \\
&= \delta_1\delta_3\tau Y^x + \delta_2\delta_4\tau Y + (\delta_1\delta_3\nu + (\delta_1\delta_4 + \delta_2\delta_4)\nu + \delta_2\delta_4\nu) \\
&= \delta_1\delta_3\tau Y^q + \delta_2\delta_4\tau Y + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1}(Y^q + Y) \\
&= (\delta_1\delta_3\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1})Y^x \\
&\quad + (\delta_2\delta_4\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1})Y
\end{aligned} \tag{3.59}$$

However, now that the coefficients are separated into Y and Y^q components, the derivation can proceed further than the multiplication. Using (3.50), it is known that both the Y and Y^q coefficients must both be equal to τ^{-1}. Therefore both terms can now be separately evaluated.

$$\tau^{-1} = \delta_1\delta_3\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1} \tag{3.60}$$
$$\tau^{-1} = \delta_2\delta_4\tau + [(\delta_1 + \delta_2)(\delta_3 + \delta_4)\nu]\tau^{-1} \tag{3.61}$$

Since both equations are equal to τ^{-1}, they can be added together to arrive at (3.63).

$$0 = \delta_1\delta_3\tau + \delta_2\delta_4\tau \tag{3.62}$$
$$\therefore \delta_1\delta_3 = \delta_2\delta_4 \tag{3.63}$$

Using (3.60), the τ^{-1} can be removed by multiplying by τ.

$$\begin{aligned}
1 &= \delta_1\delta_3\tau^2 + (\delta_1\delta_4 + \delta_2\delta_3)\nu \\
\delta_2 &= \delta_1\delta_2\delta_3\tau^2 + (\delta_1\delta_2\delta_4 + \delta_2^2\delta_3)\nu
\end{aligned} \tag{3.64}$$

The $\delta_1\delta_2\delta_4$ term can then be replaced with $\delta_1^2\delta_3$ using (3.63), therefore

$$\begin{aligned}\delta_2 &= \delta_1\delta_2\delta_3\tau^2 + (\delta_1^2\delta_3 + \delta_2^2\delta_3)\nu \\ &= [\delta_1\delta_2\tau^2 + (\delta_1^2 + \delta_2^2)\nu]\delta_3 \end{aligned} \tag{3.65}$$

Now, the only unknown variable in the equation is δ_3. It can now be solved for; the same process applies to (3.61) in order to obtain δ_4.

$$\delta_3 = \delta_2[\delta_1\delta_2\tau^2 + (\delta_1^2 + \delta_2^2)\nu]^{-1} \tag{3.66}$$

$$\delta_4 = \delta_1[\delta_1\delta_2\tau^2 + (\delta_1^2 + \delta_2^2)\nu]^{-1} \tag{3.67}$$

The equations for δ_3 and δ_4 share the same expression $[\delta_1\delta_2\tau^2 + (\delta_1^2 + \delta_2^2)\nu]^{-1}$, meaning that the same components can be used in hardware besides the multiplication with δ_2 and δ_1, respectively. Furthermore, the trace can again be set to 1 as shown in the multiplication derivation. Figure 3.4, shows the block diagram for the composite field inversion using the normal basis.

$$\delta_3 = \delta_2[\delta_1\delta_2 + (\delta_1^2 + \delta_2^2)\nu]^{-1} \tag{3.68}$$

$$\delta_4 = \delta_1[\delta_1\delta_2 + (\delta_1^2 + \delta_2^2)\nu]^{-1} \tag{3.69}$$

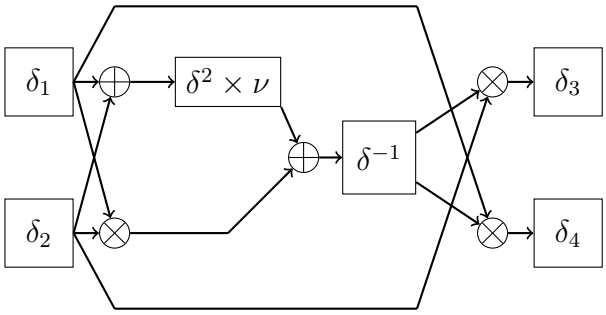

Figure 3.4: Normal basis composite field inversion

Chapter 4

MK-3 S-Box Implementation

The majority of the complexity in MK-3 is the Substitution stage which is the focus of this thesis. If left alone, The $GF(2^{16})$ multiplicative inversion will require a table with 65,536 16-bit elements. The Galois field can be represented compositely in order to break down the inversion into a sequence of smaller operations that are more efficient when implemented in hardware.

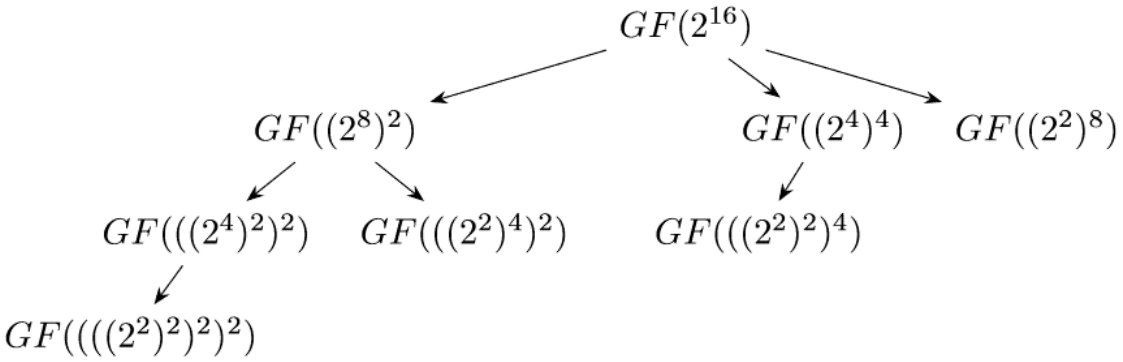

Figure 4.1: Composite Tower Field Representations

Fig 4.1 shows the possible composite representations of a 16-bit irreducible polynomial. Prior work by Wood [10] implemented the $GF(2^{16})$ inversion used in MK-3 compositely using $GF((((2^2)^2)^2)^2)$ representations. Later on, this was adapted by Werner for an FPGA [11] which also used towers of binary extensions. This thesis expands upon this by examining $GF(((2^4)^2)^2)$ and $GF((2^8)^2)$ composite representations for an FPGA.

4.1 FPGA Architecture

The atomic computation unit of an FPGA is a Look-Up-Table (LUT). As shown in Fig 4.2, a LUT typically has 4-6 inputs which index a table containing the output bit for every combination of inputs. The LUTs inside of an FPGA are of a fixed size; using a portion of the inputs for computation will leave some of the LUT utilized.

The target FPGA for this thesis is a Xilinx Kintex 7 (xc7k160tfbg676-1) which features logic slices containing 4 6-input LUTs. The LUTs feature two outputs, O5 and O6 [18]. When all 6 inputs are used in a LUT, only O6 is used. When 5 or fewer inputs are used, both O5 and O6 can be used.

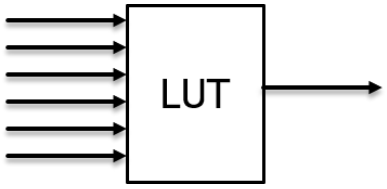

Figure 4.2: A single LUT

The LUTs inside of an FPGA are arranged into units called logic slices. Logic slices mainly contain LUTs, multiplexers, and registers. The LUTs and a multiplexer inside of a logic slice are often configured to function as a single, larger LUT. Fig 4.3 shows 4 6-input LUTs feeding into two stages of multiplexers that uses the last two bits of the input to function as a single 8-input LUT.

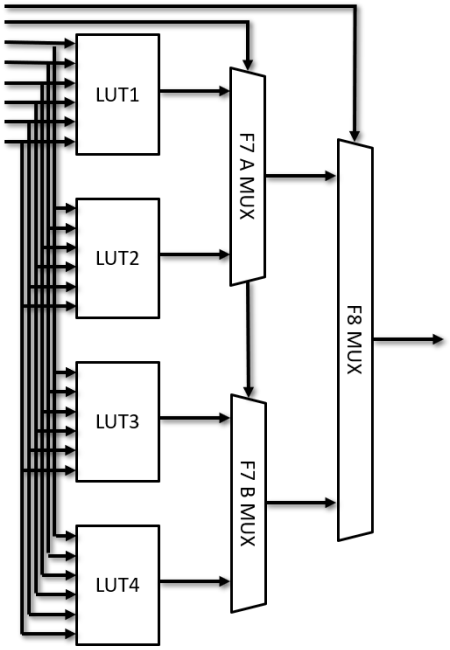

Figure 4.3: Four 6-input LUTs configured to function as an 8-input LUT

The logic slices inside of the Xilinx Kintex 7 Feature two multiplexer stages: F7 and F8 [18]. The F7 stage has two multiplexers, each one takes the output of two LUTs and selects between them using one more input bit. This allows each F7 Mux to function as a 7

bit LUT. The F8 Mux operates in the same fashion, but combines the output of the two F7 MUXs to function as 8 inputs.

4.2 Implementation Candidates

Of the implementations studied in this thesis:

- $GF((2^8)^2)$
- $GF(((2^4)^2)^2)$
- $GF((((2^2)^2)^2)^2)$

the $GF((((2^2)^2)^2)^2)$ option is well suited to ASICs where optimizations can remove individual logic gates. Both Canright and Wood used towers of binary extension fields over $GF(2^2)$ for their AES and MK-3 implementations [8, 10]. $GF(2^2)$ operations use at most 4 inputs during multiplication of two elements. Addition, squaring, and scaling all need at most 2 inputs. If the same designs are placed on an FPGA, this will leave the some of the LUTs inside of a design underutilized. Instead, the S-box can be altered to use $GF(2^4)$ or a $GF(2^8)$ as the base of the composite representation. This will help the design better fit the LUTs inside of an FPGA and may result in a better area efficiency and clock frequency. This thesis seeks to investigate the trade offs between area and frequency of the three candidates on an FPGA.

In addition to the base field of the composite representation, the basis of the outer extension field will be implemented using the normal and polynomial basis in an effort to understand the effect on the FGPA design.

MK-3 features four stages: Substitution, Permutation, Mixing, and the Round Key. Since the permutation stage mixes the bits across the entire state, it is not feasible to remain in the composite representation for the duration of the encryption. Therefore the composite representation will only be used inside of the Substitution stage. Change of basis matrices will be on the input and output of the composite inversion in order to translate into and out of the composite representation. It will resemble Figure 4.4.

4.2.1 P8P2 and P8N2 Inversion

$GF((2^8)^2)$ inversion breaks the inversion into a sequence of polynomial basis $GF(2^8)$ operations (P8). The addition being a binary XOR will easily fit inside of a single LUT. Inversion and square-scaling operate on one P8 element and produce a P8 result. Since each output bit depends on all 8 input bits, this can fit into 4 LUTs and a MUX as in Fig. 4.3. P8 multiplication requires two P8 inputs to produce one P8 result. This will not easily synthesize onto an FPGA.

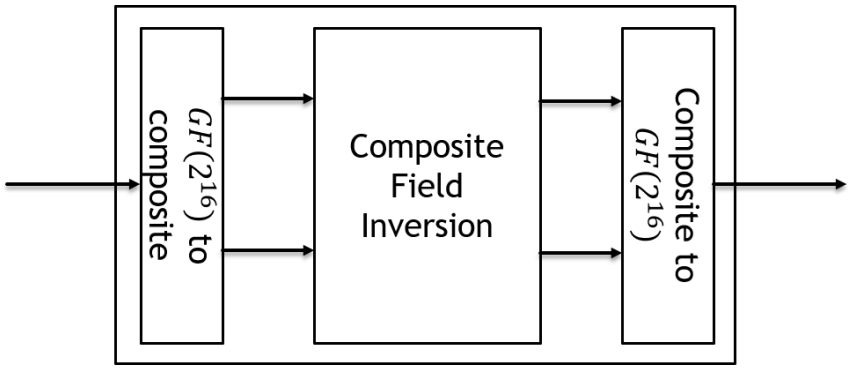

Figure 4.4: MK-3 Composite S-Box

To get around this, a second basis transformation can be used to perform the multiplication compositely using a P4P2 representation. Besides P4 multiplication, all other operations only need one LUT per output bit to compute the result. As in the case of P8 inversion and square-scaling, P4 multiplication will require 8 input bits to output 1 output bit (2 4-bit inputs) and will fit inside of 4 LUTs and a MUX as shown in Fig. 4.3.

4.2.2 P4P2P2 and P4P2N2 Inversion

P4P2P2 and P4P2N2 inversion circuits each contain the following P4P2 operations: addition, multiplication, inversion, and square-scaling. Addition is always an XOR between the two inputs and is simple. The multiplication of P4P2 elements is best implemented by a composite set of P4 operations. Since the composite square-scaling and inversion operations only have one P4P2 input, it can either be implemented through a series of hierarchical composite operations as in Sections 3.6.3 and 3.6.2 or through a flattened LUT containing all 256 possible P4P2 results.

The LUT or 'flattened' approach is a fixed cost for both inversion and square-scaling. 4 LUTs and the multiplexers in a logic slice will function as a single 8-input LUT inside of a logic slice as in Figure 4.3. This means that the P4P2 operation will require 8 logic slices, consisting of 32 LUTs, 16 F7 MUXs, and 8 F8 MUXs in total.

If implemented hierarchically, LUTs will be used for each logical operation. The sub field operations for P4P2 inversion and square-scaling are shown in Table 4.1.

Operation	**Addition**	**Mult.**	**Inversion**	**Squaring**	**Sq-Scaling**
Inversion	2	3	1	0	1
Sq-Scaling	2	0	0	1	2

Table 4.1: Sub field operations inside of a composite operation

The equation for polynomial basis square-scaling is $(\delta_1^2\nu^{-1} + \delta_1^2 + \delta_2^2\nu^{-1})x + (\delta_1^2)$, from (3.46). The LUT cost for hierarchical P4P2 composite square-scaling is as follows:

Square-Scaling and Square Operations

- $\delta_1^2\nu^{-1}$ and δ_1^2 share the same input, δ_1. Due to this, they can share the same dual output LUT. 4 LUTs will be needed.
- $\delta_2^2\nu^{-1}$ does not share any common inputs and it will need it's own LUTs. Therefore, 4 LUTs will be necessary.

Addition There are two additions in the square-scale circuit. Since they sum together the square-scale and square operations, they can both be combined into one addition of 3 inputs. With 4 output bits, 4 LUTs are required in total.

In total, this requires 12 LUTs.

Unlike the P4P2 square-scaling, the implementation of the P4P2 inversion does not feature any functions that share common inputs. The LUT cost for hierarchical inversion is as follows:

Addition and Square-Scaling Each of these operations require 4 input functions. They do not share any common inputs. As such, 1 LUT per output bit will be required, with 2 additions, 1 square-scale, and 1 inversion, that is 16 LUTs total.

Multiplication Multiplication of two 4-bit inputs will require 4 LUTs per output bit. There are 3 multipliers in the circuit, which together, require 48 LUTs.

In total, this requires 64 LUTs.

The P4P2P2 and P4P2P2 inversion circuits will require one P4P2 inversion and one P4P2 square-scaling. Therefore, a hierarchical implementation of both operations will require 76 LUTs compared to 64 LUTs using the flattened implementation. The flattened implementation will also have a latency advantage since each output bit can be computed inside of a single logic slice. However, the hierarchical implementation may provide more opportunity for the synthesis tool to optimize the design. Due to this, both design choices will be implemented.

This leaves three main variables controlling the S-Box implementation: the base field, the outer extension basis, and for $GF(((2^4)^2)^2)$, the implementation of the P4P2 inversion and square-scaling.

Chapter 5

Python Scripting Infrastructure

The composite S-Box implementations of MK-3 were generated through a series of Python scripts. Wood [19] wrote a Python library for performing Galois field arithmetic. This library has been extended in this thesis to handle multiple levels of extension fields. The Python scripts provide two functions: finding optimal field candidates and generating the RTL for the composite multiplicative inverse module.

5.1 Finding Optimal Field Candidates

The Python code to find all of the composite field candidates for MK-3 was divided into three main scripts

1. Finding all possible composite representations and isomorphisms
2. Finding all normal elements in each composite representation
3. Finding optimal candidates

5.1.1 General Construction

The Python scripts written to find all suitable composite field representations of MK-3 feature several places where parallelism can be exposed. The Multiprocessing package was chosen to handle running all of the concurrent blocks. It uses process-based parallelism instead of thread-based parallelism which allows it to run across multiple CPU cores. This enables it to work in the Research Computing environment [20] available at RIT where jobs are scheduled through SLURM.

The Multiprocessing package [21] arranges the available CPUs into a 'pool.' Once jobs are assigned to the pool, the pool will allocate available CPUs to the jobs until all of the jobs have completed. The jobs are simply Python function calls. The Multiprocessing package also includes a Queue class that can handle asynchronous communication from the parallel jobs; this was used to return job results through so that they may be processed while the rest of the jobs are executing.

Results from the scripts were saved to simple plain text output files. Galois field elements were written to the file by printing out the coefficients of the field encased in square brackets ([]). A rudimentary regex parser was built to read back the Galois field elements in the output files and rebuild the components for the later scripts. If this was to be done again, a YAML file or the pickle package in Python would be better solutions.

The file structure was organized as follows

Directory	Description
core	Contains core classes for arithmetic and general utilities
gen_hdl	Classes responsible for generating VHDL components of MK-3
output	Generated output files for composite field candidates
output_hdl	Generated VHDL source code
.	Main scripts

5.1.2 Finding Isomorphisms

The first script is responsible for finding all of the possible composite fields and the isomorphisms that link it to the MK-3 polynomial. In order to simplify the isomorphism search, constraints were placed on the composite fields that were used. Binary extension fields of the form $y^2 + y + \lambda$ were used. In addition the base field was chosen to be the AES polynomial, $x^8 + x^4 + x^3 + x + 1$ for $GF((2^8)^2)$ and $x^4 + x + 1$ for $GF(((2^4)^2)^2)$. The AES polynomial was chosen since it has already undergone rigorous research as part of becoming a standard of encryption; $x^4 + x + 1$ was chosen since it is one of only two choices.

The next step in the script is to eliminate composite field candidates that are not irreducible. Since the extension fields used in this thesis are binary, the roots can be found by solving

$$y^2 + y = \lambda \tag{5.1}$$

All possible values of y can be evaluated in (5.1). The values of λ that equal $y^2 + y$ can be eliminated. Those that remain form an irreducible composite representation.

Now that all of the suitable composite fields are known, all that remains is to find the isomorphisms to the MK-3 polynomial. For each composite field F, every possible value is substituted into the MK-3 polynomial. Those that equal 0 form an isomorphism. This step lends itself very well to parallelism, it needs to be evaluated for every field and for every possible value in the field.

Listing 5.1: Snippet searching for the isomorphisms in $GF((2^8)^2)$

```
1 for j in range(1,2**8):
2     x_val = GFElem(bitfield(j))
3     for k in range(0,2**8):
4         one_val = GFElem(bitfield(k))
5         val = GFExtensionElem([x_val, one_val])
```

```
6
7         # Check for isomorphism
8         # res should be 0 if that is the case
9         res = F.g_add(F.g_add(F.g_add(F.g_add(fast_exp_gf(F, val, 16),
10              fast_exp_gf(F, val, 5)), fast_exp_gf(F, val, 3)), val), GFExtensionElem([
                    ↪ GFElem([1])]))
```

For each isomorphism found, both the isomorphism and the associated field are printed out to an output file. This will be used in the following steps. There are four different implementations of this script. There is a single-threaded version primarily for debugging and a version using the Multiprocessing module for both the $GF(((2^4)^2)^2)$ and $GF((2^8)^2)$ implementations.

5.1.3 Finding Normal Elements

For an element inside of a composite field to be normal, it must be linearly independent. The scripts to find the normal elements are responsible for evaluating every element inside each composite field candidate. This is easily parallelizeable across each composite field. A function was written, *find_norms*, that evaluates every element in a composite field and returns the list of elements that are normal.

Listing 5.2: Snippet finding all normal elements inside of a composite field

```
1  # Function to find a normal element inside a field
2  # Accepts either a GF or GFExtension field
3  # Returns a list of normal elements
4  def find_norms(F):
5      norms = []
6      m = len(F.ip) - 1
7      mat = []
8
9      if F.__class__ == GF:
10         p = F.base
11         # Exhaustively search for a normal element
12         for i in range(2, 2**m):
13             mat = []
14             elem = GFElem(bitfield(i, p))
15             for i in range(m):
16                 mat.append(gfElem2Lst(F, elem))
17                 elem = F.power(elem, p)
18             if is_linear_ind(p, mat):
19                 norms.append(elem)
20     elif F.__class__ == GFExtension:
21         coeffCount = get_coeffCount(F)
22         p = get_baseFieldPrim(F)
23         n = get_coeffCount(F.baseField)
24
25         for i in range(1, p**coeffCount):
26             mat = []
27
28             # Create one large GFExtension element
29             elem = createElem(F, bitfield(i, p, coeffCount))
30             for j in range(F.extension):
31                 row = F.power(elem, (p**n)**j)
32                 mat.append(gfElem2Lst(F, row, flatten=False))
```

```
33            if is_linear_ind(F.baseField, mat):
34                norms.append(elem)
35    return norms
```

The script begins by reading in the list of composite fields produced by the function. A function call to *find_norms* for each composite field is submitted to the pool of parallel tasks to be run. Upon completion, all of the normal elements and their corresponding composite field are written to a second output file for further processing in the next step.

5.1.4 Optimization

The main choices that affect the size of the multiplicative inverse circuit is the composite field type ($GF((2^8)^2)$ or $GF(((2^4)^2)^2)$), the basis (normal or polynomial), the coefficients of the field, and for P4P2X2 fields, the implementation of single input P4P2 operations (square-scaling and inversion). The type of the field will determine the overall construction of the circuit. Both the type and the coefficients together determine the change of basis matrices on either end of the circuit that are responsible for converting to and from the $GF(2^{16})$ representation used in the rest of the the MK-3 algorithm.

In order to determine the impact that the basis and type of composite field have on the circuit, a separate script was written for each combination: $GF((2^8)^2)$ polynomial and normal; $GF(((2^4)^2)^2))$ polynomial and normal. The scripts use the list of composite field candidates and isomorphims to construct the change of basis matrices. For the normal basis versions, the script also reads in the list of normal basis elements for each composite field and constructs the matrices for each normal element. For P4P2P2 and P4P2N2 implementations that feature a hierarchical P4P2 inversion and square-scaling, the candidates that do not meet the criteria for the scaling component are filtered out. The change of basis matrices are then ranked according to the number of LUTs needed.

The construction of the change of basis matrices occurs in two steps where the second step is only used for the normal basis composite fields. The first step, shown in Listing 5.3 simply involves constructing a change of basis matrix to the polynomial, composite representation. This is accomplished using the provided isomorphism *self.c* and composite field *self.F*.

Listing 5.3: GFCompFieldBasis class definition and genChangeBasisMat member function constructing a polynomial basis composite field

```
1  # Class to contain c, F, and both change of basis matrices
2  class GFCompFieldBasis(object):
3      def __init__(self, c, F):
4          self.c = c
5          self.F = F
6          self.p = get_baseFieldPrim(F)
7          self.basis = None
8          self.inv_basis = None
9          self.direct_count = None
10         self.inv_count = None
11         self.direct_luts = None
```

```
12          self.direct_latency = None
13          self.inv_luts = None
14          self.inv_latency = None
15          self.total_luts = None
16          self.total_latency = None
17          self._genChangeBasisMat()
18          self._countOnesInBasis()
19          self._LUTsInBasis()
20
21      def _genChangeBasisMat(self):
22          c_lst = []
23          c_lst.append(createOneElem(self.F))
24          c_lst.append(self.c)
25          for i in range(2,get_coeffCount(self.F)):
26              new_c = self.F.g_mult(c_lst[-1],self.c)
27              c_lst.append(new_c)
28
29          basis_mat = []
30          for c in c_lst:
31              # Create a list of coefficients
32              basis_mat.append(gfElem2Lst(self.F, c))
33
34          basis_tuple = list(zip(*basis_mat[::-1]))
35          # Convert the list of tuples to a list of lists
36          self.basis = []
37          for row in basis_tuple:
38              self.basis.append(list(row))
39          self.inv_basis = gf_mat_mod_inv(self.p, self.basis)
```

Due to the limitations of the Galois field library used, the normal basis change of basis matrices cannot be computed in the same fashion as the polynomial basis versions since the library is not capable of normal basis arithmetic. Instead the class containing the normal basis member function *_genChangeBasisMat* inherits from the polynomial basis class. It uses the parent class implementation to construct the change of basis matrices transforming between the original representation and the composite polynomial representation. The matrices for conversion between composite polynomial and composite normal are also constructed. Using these two sets of matrices, a combined matrix to directly transfer between the original and composite normal representation can be implemented. Each normal basis coefficient can be first converted to composite polynomial, and then to the original representation. Once done for each coefficient, the new combined inverse matrix is formed. The direct matrix can then be found through inversion.

Listing 5.4: GFNormCompFieldBasis derived class definition and genChangeBasisMat member function constructing a normal basis composite field, this inherits from the polynomial basis version

```
1   class GFNormCompFieldBasis(GFCompFieldBasis):
2       def __init__(self, c, norm, F):
3           self.norm = norm
4           super(GFNormCompFieldBasis, self).__init__(c, F)
5
6       def _genChangeBasisMat(self):
7           super(GFNormCompFieldBasis, self)._genChangeBasisMat()
8           # Get the normal basis conversion matrices
9           toNormLst, toPolyLst = genBasisTables(self.F, self.norm)
```

```
10
11          num_coeff = get_coeffCount(self.F)
12
13          # Initialize combined_inv_basis
14          combined_inv_basis = []
15          for i in range(0, num_coeff):
16              combined_inv_basis.append(num_coeff*[0])
17
18          # For each normal coefficient
19          # 1. Change basis to polynomial
20          # 2. Then apply original inverse basis to go back to original representation
21          # 3. Update combined_inv_basis with new row
22          for i in range(0, num_coeff):
23              # Create a normal element with one coefficient equal to 1
24              norm_elem = createElem(self.F, bitfield(self.p**i, min_len=num_coeff))
25
26              # Change basis to polynomial
27              poly_coeff = changeBasis(self.F.baseField, toPolyLst, gfElem2Lst(self.F,
                    ↪ norm_elem, flatten=False))
28              poly_elem = self.F.ip.__class__(poly_coeff)
29
30              # Then apply original inverse basis to go back to original representation
31              orig_coeff = changeBasis(self.p, self.inv_basis, gfElem2Lst(self.F, poly_elem))
32              for j in range(num_coeff):
33                  combined_inv_basis[j][num_coeff-i-1] = orig_coeff[j]
34
35          # Reassign to inv_basis and basis
36          self.inv_basis = combined_inv_basis
37          # Using new basis, compute the inverse
38          self.basis = gf_mat_mod_inv(self.p, self.inv_basis)
```

5.2 RTL Generation

The RTL for the composite multiplicative inversion implementations were generated using Python scripts. The scripts were separated into three components.

1. A wrapper script that contains the chosen field, isomorphism, and normal element (for normal representations only) that calls functions to generate the RTL.

2. Functions to generate the structural composite modules.

3. Functions to generate the behavioral sub field operations.

5.2.1 Main Wrapper Scripts

The main script for P4P2P2 RTL generation is shown in Appendix A.1. Each of the candidates have a script organized in a similar fashion. It is responsible for declaring the field used for the composite inversion and calling each of the functions generating the RTL.

5.2.2 Composite RTL Generation Scripts

Since the structure of the composite operations are the same for all PXP2 or PXN2 implementations, the core functionality was made to be generic. A wrapper function provides the field, the suffixes for the extension and sub field, and the width of the operands. The main function then consumes the arguments and writes the RTL to an output file. An example of this for the composite polynomial basis inversion is shown in Appendix A.2.

5.2.3 Base Field RTL Generation Scripts

Behavioral implementation functions were made for the base field operations as well as the P4P2 inversion and square-scaling operations. These functions take an argument for the field and then generates a module with a sum-of-product assignment for each bit of the output. An example of the script is shown in Listing 5.5 and the auto-generated RTL in Listing 5.6.

Listing 5.5: Script generating $GF(2^4)$ inversion RTL

```
#!/usr/bin/env python

from core.galois import *
from core.utilities import *
from gen_hdl.gen_assignment import gen_1d_sop_assignment

def gen_2_4_inv(F, outfile):
    fh = open(outfile, 'w')
    fh.write("""
library ieee;
use ieee.std_logic_1164.all;

entity inv_4 is
port(
    x : in  std_logic_vector(3 downto 0);
    y : out std_logic_vector(3 downto 0)
);
end inv_4;

architecture behavioral of inv_4 is
begin

    process(x)
    begin\n""")
    # Create table of inverses
    res = []
    for i in range(2**4):
        elem = GFElem(bitfield(i))
        inv  = F.power(elem, 14)
        res.append(gfElem2Lst(F, inv))
    # Create transpose table
    # Organized Idx0(msb) IdxN(lsb)
    res_t = []
    for i in range(3,-1,-1):
        row_t = []
        for j in range(2**4):
            row_t.append(res[j][i])
```

```
38          res_t.append(row_t)
39
40      # Generate assignment string and print line
41      for i in range(3, -1, -1):
42          assign_str =
43            gen_1d_sop_assignment("x", 4, "y(" + str(i) + ")",
44            res_t[i])
45          fh.write("      " + assign_str + "\n")
46
47      fh.write("""\
48      end process;
49  end behavioral;""")
```

Listing 5.6: $GF(2^4)$ inversion autogenerated RTL

```
1
2   library ieee;
3   use ieee.std_logic_1164.all;
4
5   entity inv_4 is
6   port(
7       x : in  std_logic_vector(3 downto 0);
8       y : out std_logic_vector(3 downto 0)
9   );
10  end inv_4;
11
12  architecture behavioral of inv_4 is
13  begin
14
15      process(x)
16      begin
17          y(3) <= ((not x(3)) and (not x(2)) and (    x(1)) and (not x(0))) or ((not x(3)) and
                ↪  (not x(2)) and (    x(1)) and (    x(0))) or ((not x(3)) and (    x(2)) and
                ↪ (not x(1)) and (not x(0))) or ((not x(3)) and (    x(2)) and (not x(1)) and (
                ↪     x(0))) or ((    x(3)) and (not x(2)) and (not x(1)) and (not x(0))) or ((
                ↪     x(3)) and (not x(2)) and (    x(1)) and (not x(0))) or ((    x(3)) and (
                ↪    x(2)) and (not x(1)) and (not x(0))) or ((    x(3)) and (    x(2)) and (
                ↪    x(1)) and (    x(0)));
18          y(2) <= ((not x(3)) and (not x(2)) and (    x(1)) and (    x(0))) or ((not x(3)) and
                ↪  (    x(2)) and (not x(1)) and (not x(0))) or ((not x(3)) and (    x(2)) and
                ↪ (    x(1)) and (not x(0))) or ((not x(3)) and (    x(2)) and (    x(1)) and (
                ↪     x(0))) or ((    x(3)) and (not x(2)) and (not x(1)) and (not x(0))) or ((
                ↪     x(3)) and (not x(2)) and (    x(1)) and (not x(0))) or ((    x(3)) and (
                ↪ not x(2)) and (    x(1)) and (    x(0))) or ((    x(3)) and (    x(2)) and (
                ↪ not x(1)) and (    x(0)));
19          y(1) <= ((not x(3)) and (not x(2)) and (    x(1)) and (    x(0))) or ((not x(3)) and
                ↪  (    x(2)) and (not x(1)) and (    x(0))) or ((not x(3)) and (    x(2)) and
                ↪ (    x(1)) and (not x(0))) or ((not x(3)) and (    x(2)) and (    x(1)) and (
                ↪     x(0))) or ((    x(3)) and (not x(2)) and (not x(1)) and (not x(0))) or ((
                ↪     x(3)) and (not x(2)) and (    x(1)) and (not x(0))) or ((    x(3)) and (
                ↪    x(2)) and (not x(1)) and (not x(0))) or ((    x(3)) and (    x(2)) and (
                ↪    x(1)) and (not x(0)));
20          y(0) <= ((not x(3)) and (not x(2)) and (not x(1)) and (    x(0))) or ((not x(3)) and
                ↪  (not x(2)) and (    x(1)) and (not x(0))) or ((not x(3)) and (    x(2)) and
                ↪ (not x(1)) and (not x(0))) or ((not x(3)) and (    x(2)) and (not x(1)) and (
                ↪     x(0))) or ((not x(3)) and (    x(2)) and (    x(1)) and (not x(0))) or ((
                ↪     x(3)) and (not x(2)) and (not x(1)) and (not x(0))) or ((    x(3)) and (
                ↪ not x(2)) and (    x(1)) and (    x(0))) or ((    x(3)) and (    x(2)) and (
                ↪    x(1)) and (not x(0)));
21      end process;
22  end behavioral;
```

Chapter 6

Results

6.1 Composite Candidates

The fields for each of the candidates are listed in Table 6.1. The change of basis matrices can be found in the Appendix. The $GF((2^8)^2)$ and $GF(((2^4)^2)^2)$ candidates were selected by the Python scripts. The $GF(((((2^2)^2)^2)^2)^2)$ candidate which serves as a benchmark is from Werner's MK-3 FPGA implementation [11] which uses a composite representation similar to what is in Wood's thesis [10].

The field notation uses Q as the uppermost extension field and the subordinate fields use progressively lower letters. $GF((2^8)^2)$ representations use P as the base field $GF(2^8)$, and Q as the extension field. $GF(((2^4)^2)^2)$ representations use O as the base field $GF(2^4)$, P as the first extension $GF((2^4)^2)$ and Q as the upper extension. Likewise, $GF((((2^2)^2)^2)^2)$ uses the letters N, O, P, Q.

The $GF(((2^4)^2)^2)$ implementations use $w^4 + w + 1$ for the base field. This is one of the two possible irreducible polynomials possible to use inside of $GF(2^4)$.

The $GF((2^8)^2)$ implementations each use the AES irreducible polynomial for the base field. This was chosen due to the extensive research on it as a part of Rijndael becoming the advanced encryption standard.

Additionally, the P8P2 and P8N2 use a secondary basis transformation to convert P8 multiplication to P4P2 multiplication. The base field uses the same irreducible polynomial as seen in the $GF(((2^4)^2)^2)$ implementations, $x^4 + x + 1$.

$$P(x) = x^4 + x + 1$$
$$Q(y) = y^2 + y + \{x^3\}$$

6.2 Implementation Results

Table 6.2 shows the implementation metrics for each of the S-Box candidates. Each S-Box candidate was synthesized with a register stage placed before and after the S-Box

Candidate	**P4P2 Operation Implementation**	**Polynomials**
P8P2	-	$P(x) = x^8 + x^4 + x^3 + x + 1$ $Q(y) = y^2 + y + \{x^5\}$
P8N2	-	$P(x) = x^8 + x^4 + x^3 + x + 1$ $Q(y) = y^2 + y + \{x^5 + x\}$ $norm = y + \{x^7 + x^6 + x^5 + x^2 + x\}$
P4P2P2	Hierarchical	$O(w) = w^4 + w + 1$ $P(x) = x^2 + x + \{w^3 + w^2 + w + 1\}$ $Q(y) = y^2 + y + \{\{w^3\}x\}$
P4P2N2	Hierarchical	$O(w) = w^4 + w + 1$ $P(x) = x^2 + x + \{w^3 + w\}$ $Q(y) = y^2 + y + \{\{w^3 + 1\}x + \{w^3\}\}$ $norm = y + \{\{w^3 + w^2 + w + 1\}x + \{1\}\}$
P4P2P2	Flattened	$O(w) = w^4 + w + 1$ $P(x) = x^2 + x + \{w^3 + w^2 + w + 1\}$ $Q(y) = y^2 + y + \{\{w^3 + w^2\}x + \{w\}\}$
P4P2N2	Flattened	$O(w) = w^4 + w + 1$ $P(x) = x^2 + x + \{w^3 + w\}$ $Q(y) = y^2 + y + \{\{w^3 + 1\}x + \{w^3\}\}$ $norm = y + \{x + \{w\}\}$
N2N2N2N2 [11]	-	NA[a]

Table 6.1: Composite S-Box candidates

[a]The RTL used for this implementation was inherited from Werner's FPGA implementation [11]. The composite representation beyond being N2N2N2N2 is unknown since the RTL does not contain relevant comments. Note that this differs from the candidate suggested by Wood which uses a P2P2P2N2 representation [10] but is functionally identical.

in order to get timing metrics. The $GF((2^8)^2)$ and $GF(((2^4)^2)^2)$ implementations use the RTL generated from the scripts. The N2N2N2N2 S-Box was implemented using the original RTL from Werner's MK-3 FPGA implementation [11]. Each of the designs were implemented using Xilinx Vivado 2016.4 with the default synthesis and place-and-route heuristics.

Candidate	P4P2 Operation Impl.	Slices	Slice LUTs	Slice Reg.	F7 MUXs	F8 MUXs	LUTs using O5 and O6	LUTs using O5 only	LUTs using O6 Only	Freq. (MHz)
P8N2	-	93	334	32	60	20	22	1	311	90.9
P8P2	-	96	344	32	60	20	28	1	315	90.9
P4P2P2	Hierarchical	81	295	32	39	11	19	1	275	111.1
P4P2P2	Flattened	78	271	32	32	4	19	1	251	111.1
P4P2N2	Hierarchical	86	313	32	63	26	12	0	201	111.1
P4P2N2	Flattened	85	312	32	66	27	0	1	311	125.0
N2N2N2N2 [11]	-	82	278	32	0	0	44	2	232	90.9

Table 6.2: S-Box implementation results

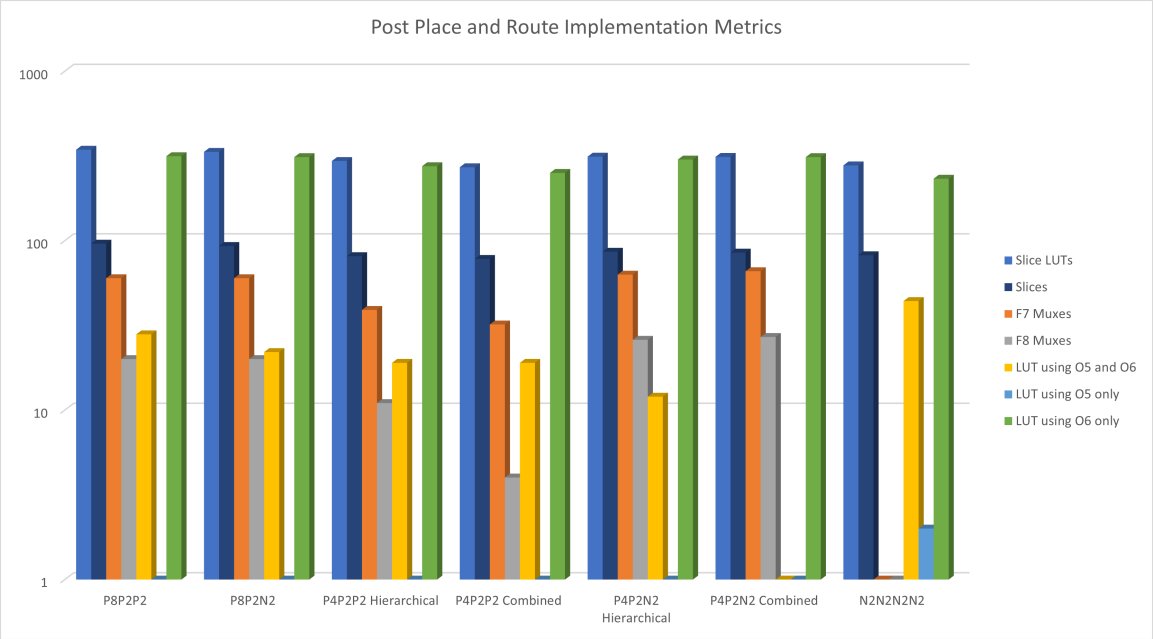

Figure 6.1: Post Place and Route Implementation Metrics

Overall, the two best candidates are the P4P2P2 flattened implementation which has the smallest area footprint of 271 LUTs and the P4P2N2 flattened implementation which has the highest clock frequency of 125 MHz. The notable outcome of this is that the flattened P4P2 inversion and square-scaling implementation had opposite effects on these two candidates. For P4P2P2, it lowered the number of LUTs used in the design and had no impact on the clock frequency. For P4P2N2, it only lowered the number of LUTs, but it increased the clock frequency. The increase in the clock frequency is likely a result of the P4P2N2 flattened approach having one fewer logic slice.

Some of this can be explained by looking at the change in the number of F7 and F8 MUXs across the hierarchical and flattened implementations of the $GF(((2^4)^2)^2)$ candidates. For P4P2P2, the number of F7 multiplexers dropped from 39 to 32 and F8 multiplexers from 11 to 4. This indicates that 7 signals originally requiring 8 inputs in the hierarchical approach now only need 7 or fewer inputs. Furthermore, since the number of F7 MUXs decreased, there are 7 signals that now need 6 or fewer inputs.

For P4P2N2, the flattened approach actually increased the number of F7 and F8 MUXs. This indicates that the synthesis or place-and-route algorithm found a way to combine more of the logic inside of a slice. While using more F7 and F8 MUXs can increase the area utilization, it can increase the locality of the operation; it is now performed inside of a logic slice. This resulted in a design with a shorter critical path and a faster clock frequency.

The $GF((2^8)^2)$ candidates are less optimal, and offer no benefit over the N2N2N2N2 baseline implementation. While the P8 inversion and square-scaling is optimal, the cost of

the extra change of basis matrices to break up the P8 multiplication into P4P2 negatively affected the resource utilization and the clock frequency was still 90.91 MHz.

The choice of the base field does not directly affect the area utilization. While the $GF((2^8)^2)$ implementations have the worst area utilization of all of them, this is a result of needing the extra change of basis matrices to break up the P8 multiplication into P4P2. The $GF(((2^4)^2)^2)$ candidates are within 13% of the baseline area utilization and in one case actually better. The main impact the base field has is in the clock frequency. By increasing the base field to $GF(2^4)$, the synthesis tool was able to better pack the logic into slices and minimize the critical path. The $GF((((2^2)^2)^2)^2)$ does not use any F7 or F8 multiplexers, and nor should it since the RTL was written to perform operations on no more than 4 bits at a time. Due to this, the implementation heuristics were not able to optimize the design for clock frequency as well as the $GF(((2^4)^2)^2)$ designs. If the $GF((2^8)^2)$ designs could have functioned without the extra change of basis matrices, it may have been competitive with the $GF(((2^4)^2)^2)$ designs.

The effect of the outer extension basis had little to no effect on the outcome of the implementations. While there is some variation seen in the $GF(((2^4)^2)^2)$, most of that seems to be a result of the heuristics used to synthesize the design. This work could be extended by examining the designs when implementing with an area or frequency optimized synthesis and place and route heuristics.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The goal of this thesis was to investigate the effect of ASIC S-Box hardware optimizations on an FPGA. The most dramatic affect on the area and performance on the implementation was the choice of the base field size. $GF(2^4)$ had the best clock frequency of the three options. This was due to the base field operations each being able to fit inside of a single Logic Slice which minimized the critical path.

The choice of the basis of the outer extension field did not have a meaningful affect on the implementation results. This is a result of composite inversion requiring the same number of sub field inversions and multiplication units; changing the basis simply rearranges the order of the operations.

Finally while the P4P2 flattened approach increased the clock frequency of the P4P2N2 design but lowered the area utilization of the P4P2P2 design when compared to the hierarchical approach, this is likely due to the heuristics of the synthesis and place and route algorithms. Further work can be done in the future to explore the implementation strategies on an FGPA.

7.2 Future Work

There are several areas that this thesis could be expanded upon. Some of these topics arose as questions during the work of this thesis and others were initially planned for but there was not sufficient time left to complete.

7.2.1 Implementation Strategies

The first subject would be the implementation strategies used for each of the candidates. The flattened approach for the $GF(((2^4)^2)^2)$ implementations saw considerable variation in the area utilization and clock frequency. A better understanding could be reached by using a range of synthesis and place and route heuristics. By doing so, the individual features of

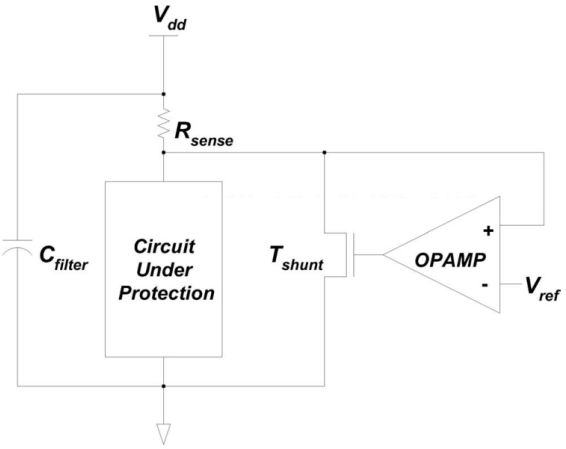

Figure 7.1: High Level Compensation block diagram [24]

each implementation could be better understood since the heuristics would focus on area or frequency.

7.2.2 MK-3 Resistance to Power Attacks

This area of research was originally part of the proposal but it was trimmed due to time constraints. Differential Power Analysis and Correlation Power Analysis are two techniques that can be used to discover the key and the plaintext used in encryption by examining variations in the power consumption of the device and the ciphertext that it produces [22, 23].

There are several approaches that can mitigate this. The first solution called High Level Compensation seeks to make the power consumption of the encryption device constant. This is accomplished by sensing the instantaneous power consumption consumed by the device and consuming the unused portion [24] using an operational amplifier as shown in Fig. 7.1.

Another approach that yields itself very well to ASICs is Dynamic Differential Logic (DDL). The principal behind this is to make the power consumption of the device uniform, but not necessarily constant [25]. The original Single Ended (SE) design is duplicated into direct and complementary. The output of the complementary component is the inverse of the original direct component. Therefore, a logic transition in one of the elements is counteracted by the opposite. Additionally the clock period is split into two phases. The first phase, pre-charge forces each net to 0. The second phase, evaluation performs the operation. By combining the differential logic and the split clock phase, every clock cycle has one logical transition between the differential pairs. In doing so, the power trace is rendered uniform such that an adversary can not glean any information from it.

DDL can be introduced into an ASIC design where there is a high degree of control over the individual gates that are rendered for an operation. Special care must be taken

to implement this on an FPGA where the logic is mapped to LUTs and the DDL results in a dramatic increase in area utilization. Yu et al [26] found that some DDL techniques increased the area utilization in an FPGA as much as 11.6 times the SE design.

Much of the prior research on DDL has focused on encryption algorithms such as AES [25, 27, 28, 29] and hashes such as SHA-3 and the candidate for SHA-3, Keccak [30, 31]. DDL and High Level Compensation could be implemented on MK-3 in order to investigate the impact in area on a 16-bit S-Box.

Bibliography

[1] C. E. Shannon, “Communication theory of secrecy systems,” *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949, conference Name: The Bell System Technical Journal.

[2] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[3] D. R. Stinson, *Cryptography: Theory and Practice*, 3rd ed. USA: CRC Press, Inc., 1995.

[4] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[5] N. I. of Standards and Technology”, “Advanced Encryption Standard (AES),” U.S. Department of Commerce, Tech. Rep. Federal Information Processing Standard (FIPS) 197, Nov. 2001. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/197/final

[6] C. Paar, “Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields,” Dissertation, Institute for Experimental Mathematics, Universität Essen, Germany, Jun. 1994. [Online]. Available: https://www.emsec.ruhr-uni-bochum.de/media/crypto/attachments/files/2010/04/paar_php_diss.pdf

[7] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A Compact Rijndael Hardware Architecture with S-Box Optimization,” in *Advances in Cryptology — ASIACRYPT 2001*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Dec. 2001, pp. 239–254. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45682-1_15

[8] D. Canright, “A very compact Rijndael S-box,” Monterey, California. Naval Postgraduate School, Report, 2004. [Online]. Available: https://calhoun.nps.edu/handle/10945/791

[9] ——, "A Very Compact S-Box for AES," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Aug. 2005, pp. 441–455. [Online]. Available: https://link.springer.com/chapter/10.1007/11545262_32

[10] C. Wood, "Large substitution boxes with efficient combinational implementations," Master's thesis, Computer Science Rochester Institute of Technology, Aug. 2013. [Online]. Available: http://scholarworks.rit.edu/theses/5527

[11] G. Werner, S. Farris, A. Kaminsky, M. Kurdziel, M. Lukowiak, and S. Radziszowski, "Implementing authenticated encryption algorithm MK-3 on FPGA," in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, Nov. 2016, pp. 1225–1230. [Online]. Available: http://ieeexplore.ieee.org/document/7795498/

[12] M. Kelly, A. Kaminsky, M. Kurdziel, M. Lukowiak, and S. Radziszowski, "Customizable sponge-based authenticated encryption using 16-bit S-boxes," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*. IEEE, Oct. 2015, pp. 43–48. [Online]. Available: http://ieeexplore.ieee.org/document/7357416/

[13] P. Bajorski, A. Kaminsky, M. Kurdziel, M. Łukowiak, and S. Radziszowski, "Array-Based Statistical Analysis of the MK-3 Authenticated Encryption Scheme," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, Oct. 2018, pp. 1–9, iSSN: 2155-7586.

[14] M. T. Kurdziel, M. Kelly, A. Kaminsky, M. Lukowiak, and S. Radziszowski, "United States Patent: 9438416 - Customizable encryption algorithm based on a sponge construction with authenticated and non-authenticated modes of operation," US Patent 9 438 416, Sep., 2016. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=9438416.PN.&OS=PN/9438416&RS=PN/9438416

[15] M. T. Kurdziel, S. M. Farris, A. R. Kaminsky, S. P. Radziszowski, M. X. Lukowiak, S. Soldavini, and D. F. Stafford, "United States Patent: 10666437 - Customizable encryption/decryption algorithm," US Patent 10 666 437, May, 2020. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2Fsearch-adv.htm&r=1&f=G&l=50&d=PALL&S1=9438416.UREF.&OS=ref/9438416&RS=REF/9438416

[16] M. Kelly, “Design and Cryptanalysis of a Customizable Authenticated Encryption Algorithm,” Master’s thesis, Computer Engineering Rochester Institute of Technology, Aug. 2014. [Online]. Available: http://scholarworks.rit.edu/theses/8325

[17] R. Lidl and H. Niederreiter, *Finite fields / Rudolf Lidl, Harald Niederreiter ; foreword by P.M. Cohn*, 2nd ed. Cambridge University Press Cambridge ; New York, 1997. [Online]. Available: http://www.loc.gov/catdir/toc/cam029/96031467.html

[18] “7 Series FPGAs CLB User Guide,” Sep. 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[19] C. Wood, “Sboxes: S-box research code,” Jun. 2015, original-date: 2013-10-16T19:46:58Z. [Online]. Available: https://github.com/chris-wood/Sboxes

[20] Rochester Institute of Technology, *Research Computing Services*. Rochester Institute of Technology, 2019. [Online]. Available: https://www.rit.edu/researchcomputing/

[21] “Process-based “threading” interface — Python 2.7.18 documentation.” [Online]. Available: https://docs.python.org/2.7/library/multiprocessing.html

[22] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” in *Advances in Cryptology — CRYPTO’ 99*. Springer, Berlin, Heidelberg, Aug. 1999, pp. 388–397. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-48405-1_25

[23] E. Brier, C. Clavier, and F. Olivier, “Correlation Power Analysis with a Leakage Model,” in *Cryptographic Hardware and Embedded Systems - CHES 2004*. Springer, Berlin, Heidelberg, Aug. 2004, pp. 16–29. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-28632-5_2

[24] G. B. Ratanpal, R. D. Williams, and T. N. Blalock, “An on-chip signal suppression countermeasure to power analysis attacks,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 179–189, Jul. 2004.

[25] K. Tiri, M. Akmal, and I. Verbauwhede, “A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards,” in *Proceedings of the 28th European Solid-State Circuits Conference*, Sep. 2002, pp. 403–406.

[26] P. Yu and P. Schaumont, “Secure FPGA Circuits Using Controlled Placement and Routing,” in *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS ’07. New York, NY, USA: ACM, 2007, pp. 45–50. [Online]. Available: http://doi.acm.org/10.1145/1289816.1289831

[27] K. Tiri and I. Verbauwhede, “A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation,” in *Automation and Test in Europe Conference and Exhibition Proceedings Design*, vol. 1. IEEE, Feb. 2004, pp. 246–251 Vol.1.

[28] R. Velegalati and J.-P. Kaps, “DPA resistance for light-weight implementations of cryptographic algorithms on FPGAs,” in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, Aug. 2009, pp. 385–390. [Online]. Available: http://ieeexplore.ieee.org/document/5272260/

[29] J. P. Kaps and R. Velegalati, “DPA Resistant AES on FPGA Using Partial DDL,” in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2010, pp. 273–280.

[30] X. Tran, “Power Analysis Attacks on Keccak,” Master’s thesis, Computer Engineering Rochester Institute of Technology, 2015. [Online]. Available: http://scholarworks.rit.edu/theses/8802

[31] X. D. Tran, M. Łukowiak, and S. P. Radziszowski, “Effectiveness of variable bit-length power analysis attacks on SHA-3 based MAC,” in *MILCOM 2016 - 2016 IEEE Military Communications Conference*. IEEE, Nov. 2016, pp. 794–799. [Online]. Available: http://ieeexplore.ieee.org/document/7795426/

Appendix A

RTL Generation Script Examples

Additional scripts are provided in this chapter to show the structure of the RTL generation.

A.1 P4P2P2 Main Script

The script below is responsible for generating all of the RTL for P4P2P2 inversion. The script imports all of the necessary modules, declares the field, and then calls the functions from the imported modules to generate the RTL. The RTL for the other candidates is generated using the same format.

Listing A.1: Script generating P4P2P2 inversion RTL

```
1  #!/usr/bin/env python
2
3  from core.galois import *
4  from core.utilities import *
5  from core.comp_field_basis import *
6
7  from gen_hdl.gen_2_4_mul import *
8  from gen_hdl.gen_2_4_inv import *
9  from gen_hdl.gen_2_4_sq import *
10 from gen_hdl.gen_2_4_sc import *
11 from gen_hdl.gen_2_4_ss import *
12 from gen_hdl.gen_poly_comp_mul import *
13 from gen_hdl.gen_poly_comp_inv import *
14 from gen_hdl.gen_poly_comp_sc import *
15 from gen_hdl.gen_poly_comp_sq import *
16 from gen_hdl.gen_poly_comp_ss import *
17 from gen_hdl.gen_cb import *
18
19 if __name__ == "__main__":
20     if len(sys.argv) == 2:
21         path = sys.argv[1]
22     else:
23         path = 'output_hdl/2_4'
24
25     W = GFElem([1,0,0,1,1]) # w^4+w+1
26     X = GFExtensionElem([GFElem([1]), GFElem([1]), GFElem([1,1,1,1])]) # x^2+x+{w^3+w^2+w
       ↪ +1}
27     Y = GFExtensionElem([GFExtensionElem([GFElem([1])]), GFExtensionElem([GFElem([1])]),
28         GFExtensionElem([GFElem([1,0,0,0]), GFElem()])]) # y^+y+{[w^3]x}
29     F_gnd = GF(2, 4, W)
30     F_mid = GFExtension(F_gnd, 2, X)
```

```
31      F_top = GFExtension(F_mid, 2, Y)
32
33      nu_x = X[0]
34      nu_y = Y[0]
35      nu_x_inv = F_gnd.power(X[0], 14)
36      nu_y_inv = F_mid.power(Y[0], 254)
37
38      # Conditions for the composite square-scaling to be optimized
39      assert Y[0][0].isZero()
40      assert F_mid.g_add(nu_x_inv,Y[0][1]).isZero()
41
42      c = GFExtensionElem([GFExtensionElem([GFElem([1,0,0]), GFElem()]),
43          GFExtensionElem([GFElem([1,0,0,1]), GFElem([1,1,0,1])])])
44      # C:  [[(1w^2)x^1]y^1 + [(1w^3 + 1w^0)x^1 + (1w^3 + 1w^2 + 1w^0)x^0]y^0]
45
46      basis = GFCompFieldBasis(c, F_top)
47
48      # Base field operations
49      gen_2_4_mul(F_gnd, path + "/mul_4.vhd")
50      gen_2_4_inv(F_gnd, path + "/inv_4.vhd")
51      gen_2_4_sq(F_gnd, path + "/sq_4.vhd")
52      gen_2_4_sc(F_gnd, nu_x, path + "/sc_4.vhd")
53      gen_2_4_sc(F_gnd, nu_x_inv, path + "/sc_nu_inv_4.vhd", "sc_nu_inv_4")
54      gen_2_4_ss(F_gnd, nu_x, path + "/ss_4.vhd")
55
56      # GF((2^4)^2) composite operations
57      gen_2_4_2_mul(F_mid, path + "/mul_poly_4_2.vhd")
58      gen_2_4_2_inv(F_mid, path + "/inv_poly_4_2.vhd")
59      gen_2_4_2_sc(F_mid, path + "/sc_poly_4_2.vhd")
60      gen_2_4_2_sq(F_mid, path + "/sq_poly_4_2.vhd")
61      gen_2_4_2_ss(F_mid, path + "/ss_poly_4_2.vhd")
62
63      # GF(((2^4)^2)^2) composite operations
64      gen_2_4_2_2_poly_mul(F_top, path + "/mul_poly_4_2_2.vhd")
65      gen_2_4_2_2_poly_inv(F_top, path + "/inv_poly_4_2_2.vhd")
66
67      # Change of basis
68      gen_plain_to_comp(basis, path + "/p2c_16.vhd")
69      gen_comp_to_plain(basis, path + "/c2p_16.vhd")
```

A.2 Polynomial, Composite Wrapper Generation

The main function of this script *gen_poly_comp_inv* is written generically to handle any size binary extension field. The end of the script features helper functions that supply the arguments necessary to perform P4P2, P4P2P2, and P8P2 inversion. The other composite wrappers are written in the same fashion.

Listing A.2: Script generating structural composite inversion RTL

```
1  #!/usr/bin/env python
2
3  from core.galois import *
4  from core.utilities import *
5
6  def gen_poly_comp_inv(F, ent_str, sub_str, width, outfile):
7      fh = open(outfile, 'w')
```

```
8
9       sub_width = width / 2
10
11      fh.write("""
12  library ieee;
13  use ieee.std_logic_1164.all;
14
15  entity inv_%s is
16  port(
17      x : in  std_logic_vector(%d downto 0);
18      y : out std_logic_vector(%d downto 0)
19  );
20  end inv_%s;
21
22  architecture behavioral of inv_%s is
23  signal delta_1 : std_logic_vector(%d downto 0);  -- Upper component of direct
24  signal delta_2 : std_logic_vector(%d downto 0);  -- Lower component of direct
25  signal delta_3 : std_logic_vector(%d downto 0);  -- Upper component of inverse
26  signal delta_4 : std_logic_vector(%d downto 0);  -- Outer component of inverse
27  signal ss      : std_logic_vector(%d downto 0);  -- delta_1 squared * nu
28  signal add_1_2 : std_logic_vector(%d downto 0);  -- delta_1 + delta_2
29  signal mul_1_2 : std_logic_vector(%d downto 0);  -- add_1_2 * delta_2
30  signal inv_in  : std_logic_vector(%d downto 0);  -- mul_1_2 + ss (input to the inverter)
31  signal inv_out : std_logic_vector(%d downto 0);  -- output of GF(2^4) inverter
32
33  """ % (ent_str, width-1, width-1, ent_str, ent_str, sub_width-1, sub_width-1,
34      sub_width-1, sub_width-1, sub_width-1, sub_width-1, sub_width-1,
35      sub_width-1, sub_width-1))
36
37      fh.write("""
38  component mul_%s is
39  port(
40      x : in  std_logic_vector(%d downto 0);
41      y : in  std_logic_vector(%d downto 0);
42      z : out std_logic_vector(%d downto 0)
43  );
44  end component;
45
46  component inv_%s is
47  port(
48      x : in  std_logic_vector(%d downto 0);
49      y : out std_logic_vector(%d downto 0)
50  );
51  end component;
52
53  component ss_%s is
54  port(
55      x : in  std_logic_vector(%d downto 0);
56      y : out std_logic_vector(%d downto 0)
57  );
58  end component;
59
60  begin
61
62  """ % (sub_str, sub_width-1, sub_width-1, sub_width-1,
63      sub_str, sub_width-1, sub_width-1,
64      sub_str, sub_width-1, sub_width-1))
65
66      fh.write("""
67  delta_1 <= x(%d downto %d);
68  delta_2 <= x(%d downto 0);
```

```
69
70  add_1_2 <= delta_1 xor delta_2;
71
72  mul_1_2_comp : mul_%s
73  port map(
74      x => add_1_2,
75      y => delta_2,
76      z => mul_1_2
77  );
78
79  ss_comp : ss_%s
80  port map(
81      x => delta_1,
82      y => ss
83  );
84
85  inv_in <= mul_1_2 xor ss;
86
87  inv_out_comp : inv_%s
88  port map(
89      x => inv_in,
90      y => inv_out
91  );
92
93  delta_3_comp : mul_%s
94  port map(
95      x => delta_1,
96      y => inv_out,
97      z => delta_3
98  );
99
100 delta_4_comp : mul_%s
101 port map(
102     x => inv_out,
103     y => add_1_2,
104     z => delta_4
105 );
106
107 y <= delta_3 & delta_4;
108
109 end behavioral;""" % (width-1, sub_width, sub_width-1, sub_str, sub_str,
110     sub_str, sub_str, sub_str))
111
112 def gen_2_4_2_inv(F, outfile):
113     gen_poly_comp_inv(F, "4_2", "4", 8, outfile)
114
115 def gen_2_4_2_2_poly_inv(F, outfile):
116     gen_poly_comp_inv(F, "4_2_2", "4_2", 16, outfile)
117
118 def gen_2_8_2_poly_inv(F, outfile):
119     gen_poly_comp_inv(F, "8_2", "8", 16, outfile)
```

Appendix B

Composite Field Candidates

The fields for the candidates are listed below:

B.1 P8P2

$$P(x) = x^8 + x^4 + x^3 + x + 1$$
$$Q(y) = y^2 + y + \{x^5\}$$

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

$$M^{-1} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

B.2 P8N2

$$\begin{aligned} P(x) &= x^8 + x^4 + x^3 + x + 1 \\ Q(y) &= y^2 + y + \{x^5 + x\} \\ normal &= y^1 + \{x^7 + x^6 + x^5 + x^2 + x^1\} \end{aligned}$$

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

$$M^{-1} = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}$$

B.3 P4P2 for composite P8 multiplication

$$P(x) = x^4 + x + 1$$
$$Q(y) = y^2 + y + \{x^3\}$$

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

$$M^{-1} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

B.4 P4P2P2 with hierarchical P4P2 operations

$$\begin{aligned} O(w) &= w^4 + w + 1 \\ P(x) &= x^2 + x + \{w^3 + w^2 + w + 1\} \\ Q(y) &= y^2 + y + \{\{w^3\}x\} \end{aligned}$$

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

$$M^{-1} = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1
\end{bmatrix}$$

B.5 P4P2N2 with hierarchical P4P2 operations

$$
\begin{aligned}
O(w) &= w^4 + w + 1 \\
P(x) &= x^2 + x + \{w^3 + w\} \\
Q(y) &= y^2 + y + \{\{w^3 + 1\}x + \{w^3\}\} \\
normal &= y^1 + \{\{w^3 + w^2 + w + 1\}x + 1\}
\end{aligned}
$$

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

$$M^{-1} = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}$$

B.6 P4P2P2 with combined P4P2 operations

$$\begin{aligned}
O(w) &= w^4 + w + 1 \\
P(x) &= x^2 + x + \{w^3 + w^2 + w + 1\} \\
Q(y) &= y^2 + y + \{\{w^3 + w^2\}x + \{w\}\}
\end{aligned}$$

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

$$M^{-1} = \begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1
\end{bmatrix}$$

B.7 P4P2N2 with combined P4P2 operations

$$
\begin{aligned}
O(w) &= w^4 + w + 1 \\
P(x) &= x^2 + x + \{w^3 + w\} \\
Q(y) &= y^2 + y + \{\{w^3 + 1\}x + \{w^3\}\} \\
normal &= y^1 + \{x + \{w\}\}
\end{aligned}
$$

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

$$M^{-1} = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}$$