Optimising implementation of block ciphers

Tan, Quan Quan

2019

Tan, Q. Q. (2019). Optimising implementation of block ciphers. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/83155>

<https://doi.org/10.32657/10220/49160>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**OPTIMISING IMPLEMENTATION OF
BLOCK CIPHERS**

TAN QUAN QUAN

**SCHOOL OF PHYSICAL AND
MATHEMATICAL SCIENCES**

2019

OPTIMISING IMPLEMENTATION OF BLOCK CIPHERS

TAN QUAN QUAN

SCHOOL OF PHYSICAL AND MATHEMATICAL
SCIENCES

A thesis submitted to the Nanyang Technological
University in partial fulfilment of the requirement for
the degree Master of Science (Research)

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research done by me except where otherwise stated in this thesis. The thesis work has not been submitted for a degree or professional qualification to any other university or institution. I declare that this thesis is written by myself and is free of plagiarism and of sufficient grammatical clarity to be examined. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

04/06/2019

Date

Tan Quan Quan

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it of sufficient grammatical clarity to be examined. To the best of my knowledge, the thesis is free of plagiarism and the research and writing are those of the candidate's except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

04/06/2019

Date

Associate Professor
Thomas Peyrin

Authorship Attribution Statement

This thesis does not contain any materials from papers published in peer-reviewed journals or from papers accepted at conferences in which I am listed as an author.

04/06/2019

Date

Tan Quan Quan

Acknowledgements

I would like to express my sincere gratitude to my Supervisor, Associate Professor Thomas Peyrin of School of Physical and Mathematical Sciences at Nanyang Technological University for the continuous support during the course of my Masters programme. Without his guidance throughout this journey, the results attained in this thesis will not have been possible.

-Tan Quan Quan

Abstract

With the increasing need to protect information digitally, the study of cryptography has become of paramount importance. The digital world requires strong and efficient encryption algorithms in order to ensure private information are not compromised. As we strive towards making our devices smaller, the cost of the cryptographic implementation becomes more significant. In this thesis, we will look at new optimising algorithms to reduce the number of exclusive or (XOR) boolean operations in the workhorses algorithms for encryption. The algorithms presented in this thesis are new and based on an improved version of the one described in [BP10]. It was motivated by a graph search approach to find a path towards the various targets. We will be drawing comparisons to the results obtained by various algorithms out there such as Paar's algorithm [Paa97], Boyar and Peralta's algorithm [BP10]. Overall, the results obtained from the new heuristic showed that this algorithm fares better than those mentioned above.

Contents

1	Introduction	2
1.1	History of Cryptography	2
1.2	Modern Cryptography	3
1.3	Block Ciphers	4
1.4	Motivation	5
1.5	Organisation of the paper	5
2	Preliminaries	7
2.1	Mathematical Definitions and Operations	7
2.2	Example: Advanced Encryption Standard . . .	10
2.3	Counting XORs	12
2.4	Straight Line programs	13
2.5	AES MixColumn matrix	13
2.6	Patterns in M_{AES}	15
3	Past Works	17
3.1	General	17

3.2	Paar's algorithms	17
3.2.1	Cancellation-Free programs	22
3.3	BP Algorithm	23
3.3.1	Complexity of the BP algorithm	25
3.4	Recent Developments	27
3.4.1	Minimizing the XOR count for the AES matrix	27
3.4.2	Constructing new matrices	27
4	The Algorithm	30
4.1	An analysis on the BP algorithm	30
4.2	A small tweak	31
4.3	Graph based search	31
4.4	An intuition and algorithm	35
5	Results	40
5.1	Randomized 12×12 matrices	40
5.2	On random matrices from 16×16 to 20×20 . .	41
5.3	On MDS matrices constructed in [DL18]	42
5.4	On AES Matrix, M_{AES}	44
6	Conclusion	45
6.1	Limitations	45
6.2	Further Research	46

A	Proofs for inequalities from Section 3.3.1	51
B	Matrices from [DL18]	53
C	95 XOR implementation of AES matrix	55
D	94 XOR implementation of AES matrix	56

Chapter 1

Introduction

1.1 History of Cryptography

Cryptography is the study of encryption and decryption. While the term is often associated with electronic devices in the modern world, the usage of cryptography can be dated back to the ancient Greeks. The ciphers from then till the pre-computer era are also known as the classical ciphers or traditional ciphers. One example of these ciphers is the Caesar Shift Cipher, which is the first . In the English alphabet, the key is a number K , with $1 \leq K \leq 25$. The original message is termed as the plaintext and the resulting encrypted message is termed as the ciphertext. To encrypt a message, each letter in the message is substituted with the K^{th} alphabet after the plaintext letter in the alphabetical order. To decrypt the message, we simply take the ciphertext and substitute each letter by the K^{th} alphabet preceding the ciphertext letter. The motivation for the shift cipher was to protect messages that are of military significance at that time.

However, this can be easily broken as we can see that the number of possible keys, or the key space is only 25. That is, we can simply exhaust all possible keys until the resulting possible plaintext makes sense, especially with modern day computers,

the plaintext can be found in less than a second.

1.2 Modern Cryptography

Fast forward to the twentieth century, with the rise of technology, there is a need to improve on the security of ciphers. One notable cipher is the Enigma ciphering system. This series of machines were employed by the Germans during World War II to transmit military intelligence in the battlefield. The Enigma machines can produce a polyalphabetic substitution cipher with a key space size of approximately 10^{16} depending on the specific model of the machine. While the size may be overwhelming at first, it has some weakness in the cipher which eventually led to its downfall [Li].

In modern cryptography, there are two main categories of cryptography. They are public key cryptography and symmetric key cryptography.

Public key cryptography, which is also known as asymmetric key cryptography, is usually set up with a public key and a secret key pair. The public key is used for encryption purposes and the secret key is used for decryption. Most of the public key cryptography relies on the hardness of unsolved mathematical problems. For example, the RSA (Rivest-Shamir-Adleman) cryptosystem uses the hardness of computing integer factorization and the ElGamal encryption uses the hardness of computing discrete logarithms.

Symmetric key cryptography consists of only a single key, which is used for both encryption and decryption purpose. While symmetric key cryptography is often more efficient than asymmetric key cryptography, it requires the parties involved to have a key preshared. Furthermore, when there are n different channels involved, $O(n^2)$ keys are involved as compared to $O(n)$ key

pairs in the context of asymmetric key cryptography. Thus, key exchange protocols often use asymmetric key cryptography to establish the a temporary symmetric key. The temporary key will then be used to secure the communication between two parties, A and B. The two main types of symmetric key ciphers are stream ciphers and block ciphers. In this thesis, we will be focusing on the symmetric key cryptography, specifically, on block ciphers. In the next section, we will describe block ciphers more in detail.

1.3 Block Ciphers

Block cipher is a type of symmetric key algorithm that encrypts data block by block and each block is of a fixed size. This means that before the data is fit into the encryption algorithm, the plaintext has to be first broken up into multiple blocks. Some examples are the Data Encryption Standard (DES) [SB88] and the Advanced Encryption Standard (AES) [DR02]. DES was first published as a federal standard by the National Institute of Standards and Technology (NIST) in the United States in 1977. However, due to various cryptanalysis techniques introduced and the small key space of the algorithm, DES was eventually broken in the 1990s. For instance, the Electronic Frontier Foundation’s Deep Cracker decrypted a DES key in 56 hours in 1998 [Fou98]. A year later, Deep Crack, with the help of distributed computing effort by Distributed.net, a DES key can be recovered in less than 23 hours [Kel06]. To replace DES, NIST started an AES competition. The Rijndael cipher, designed by Joan Daemen and Vincent Rijmen, won the competition and became the AES [DR02]. The AES became effective in FIPS 197 in the year 2001. Currently, the AES is still functioning as the latest standard.

When using the block ciphers, there are also several block cipher

modes that could be used. Some examples include Cipher Block Chain (CBC) and the Counter Mode (CTR) [DH79] [BDJR97]. These different modes have their own advantages and disadvantages. Hence, they are used in various implementations depending on their suitability.

1.4 Motivation

When designing or implementing a cipher, we have to assess its suitability. While security is of utmost importance, we have to consider the cost of implementing the cipher as well. This is usually motivated by space or time constraints and the cost can be measured in several ways. For instance, in hardware implementation, the area cost required; that is, the number of gate operations required for the functions in block ciphers. Block ciphers are usually built by iterating a round function several times (iterative block ciphers). According to Shannon, confusion and diffusion are two required properties for the round function [Sha49]. Confusion refers to the masking of the relationship between the ciphertext and the key while diffusion refers to the dissipation of any statistical structure of the plaintext over the ciphertext. In a round function, the confusion layer is often implemented by nonlinear functions while the diffusion layer is implemented by linear functions. In this thesis, we will be focusing on the optimization of the linear functions in block ciphers. We will first investigate some of the existing algorithms available to optimise the functions and subsequently, we will present new heuristics that optimize the functions further.

1.5 Organisation of the paper

In Chapter 2, we will be going through some mathematical notations and preliminaries required to understand the thesis. Sub-

sequently, the example of the AES will be given to understand the general operations in a block cipher. We will also give a brief introduction in how cost saving works. In Chapter 3, we will be going through some of the state-of-the-art algorithms. This includes some algorithms used in reducing the number of gate operations and the results obtained using different techniques. In Chapter 4, we explain our new search heuristic and the results of our heuristic can be found in Chapter 5. Further research and conclusion can be found in Chapter 6.

Chapter 2

Preliminaries

In this chapter, we will be reviewing some basic mathematical concepts that are relevant to the understanding of this thesis.

2.1 Mathematical Definitions and Operations

Definition 2.1.1. *A field, often denoted as $\mathbb{F}$ is a set of elements obeying two operations, $+$ and $\cdot$ (commonly referred to as addition and multiplication respectively), satisfying the following axioms:*

1. $\mathbb{F}$ is closed under $+$ and $\cdot$
2. $\mathbb{F}$ is commutative under both $+$ and $\cdot$
3. $\mathbb{F}$ is associative under both $+$ and $\cdot$
4. $\cdot$ is distributive over $+$ in $\mathbb{F}$
5. $\mathbb{F}$ is distributive over $+$ in $\cdot$
6. Existence of additive and multiplicative identity
7. Existence of additive inverse element $\forall a \in \mathbb{F}$
8. Existence of multiplicative inverse element $\forall a \neq 0 \in \mathbb{F}$

If a field has a finite number of elements, i.e. $|\mathbb{F}| < \infty$ then it will be called a finite field. $\mathbb{F}_p$ will be used to denote the finite field with p elements.

We will go through some basic operations in the finite field. We will be focusing on the field with two elements, $\mathbb{F}_2$ or also commonly known as $GF(2)$, representing the Galois Field with two elements as well as the field extensions of $GF(2)$. In $GF(2)$, the set of elements is $\{0, 1\}$. Thus, a table can be constructed for the operations.

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Definition 2.1.2. A polynomial $p(x) \in \mathbb{F}_k$ is said to be irreducible over $\mathbb{F}_k$ if

1. $p(x)$ has a positive degree
2. $p(x) = bc$ s.t. $b, c \in \mathbb{F}_k \implies b$ or c is a constant polynomial

Now, we can consider the field extensions of $GF(2)$ that are of the form with 2^k elements. We will use $GF(2^k)$ to denote it. Every field with 2^k elements is isomorphic to some polynomial ring over $\mathbb{F}_2$ modulo an irreducible polynomial p of degree k . In other words, for a field with 2^k elements, we can use the set $\{0, 1, \dots, 2^k - 1\}$ to represent the elements. Each element can also be represented in terms of the polynomial basis. To simplify the notation, the polynomials are often represented by their coefficients only. For example, the field used in AES is $GF(2^8)$ and the irreducible polynomial, $x^8 + x^4 + x^3 + x + 1$, is written as **0b100011011**. Alternatively, we can write it in hexadecimal, **0x11B** for a more compact representation.

We will discuss the case where $k = 8$. Yet, this can easily be generalised to any value of $k > 1$. In $GF(2^8)$, all the elements can be represented by a polynomial of degree at most 7, $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ where $b_7b_6b_5b_4b_3b_2b_1b_0$ is the element in binary representation. To do additions or multiplications of two elements in $GF(2^8)$, we sim-

ply add or multiply the polynomials together modulo an irreducible polynomial. Note that the coefficients are in $GF(2)$. In Table 2.1, we give an illustration of the addition and multiplication operations.

Table 2.1: Illustration of operations in $GF(2^8)$

Operation	Inputs	Inputs in polynomial form	Result in polynomial form	Result
Addition	$3 + 5$	$(x + 1) + (x^2 + 1)$	$x^2 + x$	6
Multiplication	$100 \cdot 4$	$(x^6 + x^5 + x^2) * (x^2)$ mod $x^8 + x^4 + x^3 + x + 1$	$x^7 + x^3 + x + 1$	138

Definition 2.1.3. *The EXCLUSIVE-OR (XOR) gate is an operation with the following truth table.*

Inputs		Output
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Remark 2.1.1. *A function that only involves the XOR operation is linear. This can be seen from the truth table in Definition 2.1.3 as the XOR operation is equivalent to addition modulo 2.*

$$\begin{aligned}
 T(u \oplus v) &= T(u) + T(v) \mod 2 \\
 &= T(u) \oplus T(v)
 \end{aligned}$$

Since doing additions in $GF(2)$ is equivalent to the XOR operation, we will be using the notations “ $\oplus$ ” and “ $+$ ” interchangeably when the context is in $GF(2)$.

Hamming Weight of a matrix M , denoted as $HW(M)$ is defined as the number of non-zero entries in M . Similarly, for a vector V , $HW(V)$ represents the number of non-zero entries in V . We will be using this notation often for the Hamming Weight of the

rows in a matrix.

Given two vectors u and v , the Hamming Distance of u and v is defined as the Hamming Weight of the XOR difference of u and v , i.e. $HD(u, v) = HW(u - v) = HW(u \oplus v)$ where the last equality holds in $GF(2)$.

Definition 2.1.4. *The branch number of a matrix M is defined as $bn(M) := \min\{HW(v) + HW(Mv) : v \neq 0\}$.*

Definition 2.1.5. *A matrix M is said to be Maximum Distance Separable (MDS) if and only if $bn(M) = n + 1$.*

In designing block ciphers, MDS matrices have been widely used due to its diffusion capabilities in ensuring the security of block ciphers. This was suggested by Serge Vaudenay in [Vau95]. However, there are also some notable examples of ciphers that do not adopt MDS matrices such as PRESENT and PRINCE in order to reduce the cost of implementation [BKL⁺07] [BCG⁺12]. But, in general, the matrices still exhibit a high diffusion property.

2.2 Example: Advanced Encryption Standard

In this section, we will briefly go through the details of the AES to understand the inner workings of it. A more detailed specification can be found in [DR02]. In the 1990s, the National Institute of Standards and Technology (NIST) organised a competition in effort to find a new standard cryptosystem to replace DES. Many ciphers were submitted and they have been analysed by the cryptography community. Many aspects were considered, including the security of the cipher and the cost of implementation on various architectures. Eventually, the cipher Rijndael, won the competition and became the AES. The AES uses a substitution-permutation network (SPN) structure with various (sub)functions. The functions are: SubBytes,

ShiftRows, MixColumn, AddRoundKey and the Key Scheduling Function (KSF). In Figure 2.1, a flowchart of the AES encryption process is shown with the various functions (the KSF is omitted).

Figure 2.1: Flowchart of AES Encryption

The AES has a block size of 128 bits and three different key lengths (128, 192 and 256 bits). The number of rounds for each encryption differ depending on the key length. For instance, AES-128, that is, the AES implemented with 128 bit key length, requires 10 rounds. We will briefly go into the details of the functions. For the SubBytes function, it is a 8-bit to 8-bit substitution box (S-Box). Given an input a that is 8 bits, this function computes the inverse over $GF(2^8)$ with an irreducible polynomial of $x^8 + x^4 + x^3 + x + 1$ and applies an

affine transformation with fixed coefficients. The affine transformation is given as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

where $b = b_7b_6b_5b_4b_3b_2b_1b_0$ is the inverse element of a over $GF(2^8)$.

Note that this is also the only non-linear function in AES. The KSF is a function that generates the various round keys. The AddRoundKey is a function that XORs the key with the *state*¹. The ShiftRows is a function that shifts the position of the bytes in the *state*. In terms of a round-based hardware implementation, this can be realised by switching the wires. Thus, this does not lead to any additional hardware cost in this scenario. The MixColumn function is what we are interested here. We will give a brief introduction here and a section dedicated to analysing the matrix below. The MixColumn function is a function that left-multiplies an input vector $v \in \mathbb{F}_{2^8}^4$ by a MDS matrix M_{AES} ². In [KLSW17], they remarked such a matrix as a straight line program (SLP) as it is linear (only involves the XOR operation). Note that, the MixColumn is part of the diffusion layer in the workings of AES.

¹state refers to the intermediate value between the plaintext and ciphertext

² M_{AES} is a 4×4 matrix with elements from $GF(2^8)$, the matrix is given in section 2.5

2.3 Counting XORs

As M_{AES} is a matrix with fixed elements independent of the key and the input *state*, the cost of the function can be measured and evaluated free of context. One way to calculate the cost of implementing a matrix M with size $n \times n$, is to measure the number of bitwise XOR required. As matrices are usually represented such that the entries in M are in $GF(2^k)$ with $k > 1$, a trick is to view each element of M as a matrix $T \in \mathbb{F}^{k \times k}$. Thus, we can express M as M' , a matrix of size $kn \times kn$ with elements in $GF(2)$. The simplest way to count the number of XORs is the naive implementation; it simply means take the $HW(M') - (k \times n)$ as the cost of implementation. However, in many cases, there are many repeated calculations involved. The idea is that if a repeated result was stored after it was first calculated, it can be used for the subsequent ones without being evaluated again. This is the basis of how to reduce the number of XOR gate operations in a linear circuit.

2.4 Straight Line programs

Like many optimization problems, gate optimization of linear circuits has been widely studied. This problem has been shown to be NP-Hard [BMP13]. Thus, unless $P=NP$, an efficient ϵ -approximation algorithm will not be possible. Generally, a linear straight line program over a field F is a program that only contains the XOR gate [KLSW17]. This means that the program only contains lines that are in the form $y = av + bw$ for some $a, b \in F$ and v, w are some variables. In the context of $GF(2)$, a, b will be equal to 1 (as $a, b \in \{0, 1\}$ and if a or b equals to 0, the line in the program does not do anything other than replacing a variable by another variable).

2.5 AES MixColumn matrix

In this section, we will look at the AES MixColumn matrix in detail. The AES matrix is a 4×4 matrix with elements from $GF(2^8)$:

$$M_{AES} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

The finite field is built upon the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. To calculate the cost of implementing M_{AES} , we measure it in terms of the number of bitwise XORs required. Thus, we must first, express M_{AES} as M'_{AES} such that the elements of M'_{AES} are in $GF(2)$. To do this, we have to consider every element in $GF(2^8)$ as $T \in \mathbb{F}_2^{8 \times 8}$. We use the notations as employed in [DL18]. 1 refers to the identity. Let $\alpha = 2$ and $\alpha + 1 = 3$. The element α has a matrix given by

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

To calculate a naive implementation of the number of XORs required, we can start by looking at the number of XORs for each submatrix $I_8, M_\alpha, M_{\alpha+1}$. They require 0, 3 and 11 XORs respectively. Since each row of M has exactly two '1's, one '2' and one '3', we will need $(0 + 0 + 3 + 11 + 3 \times 8) \times 4 = 152$ XORs for the naive implementation of M_{AES} .

2.6 Patterns in M_{AES}

One simple way to reduce the number of XORs is to analyse the structure of the matrix; most MDS are constructed using a particular structure that usually exhibits some form of symmetry. The MDS matrix in AES is no exception too; M_{AES} is a circulant matrix.

Definition 2.6.1. *A $n \times n$ matrix is a (right) circulant matrix if it has the following structure:*

$$M = \begin{pmatrix} c_1 & c_2 & \dots & c_n \\ c_n & c_1 & \dots & c_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_2 & c_3 & \dots & c_1 \end{pmatrix}$$

We can see that a (right) circulant matrix is defined by the elements of the first row. Subsequent rows are just a right circular shift of the rows above them.

One way of evaluating M_{AES} is to split the matrix up into 2 parts:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 2 & 2 & 0 & 0 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 2 \end{pmatrix}$$

We let M_1 to be the matrix containing the ‘1’s and M_2 to be the matrix containing the ‘2’s. To evaluate M_1 :

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \oplus y \oplus z \\ w \oplus y \oplus z \\ w \oplus x \oplus z \\ w \oplus x \oplus y \end{pmatrix}$$

Observe that $y \oplus z$ and $w \oplus x$ are common sub-expressions in the first two and the last two entries respectively. We can

reuse those values after evaluating them for the first time. This allows us to save $2 \times 8 = 16$ XORs - since each word is 8 bits, each term saves 8 XORs. In M_2 , the savings can occur if we view the elements in $GF(2)$. If we observe M_α , there will be repeated calculations within M_α as long as α is located in the same column in M . For instance, the first column has ‘2’ in the first and fourth row which causes the calculation of α to be repeated. This allow us to save 3 XORs for the first column. Repeating the process for the second, third and fourth columns allows a total of $3 \times 4 = 12$ XORs to be saved. In all, this helps to reduce the cost down to 124 XORs. In [DL18], a more efficient way of implementing with 108 XORs is described.

Chapter 3

Past Works

3.1 General

Over the years, many papers have been published in trying to reduce the overall cost of linear circuits. Some focused on reducing the number of XOR operations required in an existing matrix such as the M_{AES} . Some went on to design new MDS matrices with small circuits. Some common metrics used to compare are the number of gates and the depth¹ of the circuit. In this paper, we will be focusing only of the number of XOR gates. In this chapter, we will review the past research that are related to the reduction of the linear circuits, in terms of the number of XOR gates. We will cover two main methods, namely, Paar’s algorithms and Joan Boyar and Rene Peralta’s algorithm. In the algorithm descriptions, the index of the variables will start from 0. The t_i s will refer to the temporary variables, indicating an intermediate step.

3.2 Paar’s algorithms

Encoders/decoders and encryption/decryption require computations in $GF(2^m)$. Thus, the research on optimising encoders

¹The depth of a circuit refers to the minimum number of clock cycles required to evaluate it

may also be used to evaluate matrices for cryptographic purposes. Before Christof Paar proposed his algorithms in [Paa97], the optimisation research had mostly focused only on finding efficient general finite field multipliers using different architectures and representations [WTS⁺85] [Mas91]. The motivation of the algorithms was to be used in Reed-Solomon encoders where multiplication with constant elements in $GF(2)$ are used.

There are two algorithms described in [Paa97]. In this paper, we will term the algorithms as Paar1 and Paar2. The gist of the algorithms is to combine pairs of variables (in each row) that have the highest number of occurrences in the matrix. A new column representing the sum of the pair is appended to the matrix. Suppose the pair of variables with the highest number of occurrences are v_i and v_j (i.e. columns i and j), then we can append a new column, column $n + 1 = \text{column } i \oplus \text{column } j$ to the matrix. Then, we update column i and column j accordingly to ensure that the appended matrix is logically equivalent to the original matrix. Algorithm 1 shows the pseudocode for the Paar1. In general, Paar1 is very efficient and manages to reduce a significant number of operations required even for large and dense matrices.

The following is an example of how the Paar's algorithm works in practice. For better understanding, we convert the matrix into one with variables corresponding to the columns. Also, we can see that what we are trying to evaluate here are the individual rows of M . We will be terming the various rows as *targets*.

Algorithm 1: Paar1 Algorithm

```

1 input: a binary matrix M
2 output: number of XORs operations required for evaluating M
3 function Paar1(M);
4  $r \leftarrow$  number of rows in M
5  $counter \leftarrow 0$ 
6 do
7    $c \leftarrow$  number of columns in M
8   Initialize  $A$  to be a zero matrix of size  $r \times c$ 
9   let  $M_i$  be the column vectors of  $M$ ,
10   $A[i][j] \leftarrow M_i \cdot M_j \quad i, j \in \{0 \dots n-1\}, \forall j > i$ 
11   $hmax \leftarrow \max(A[i][j])$ 
12   $i_{max}, j_{max} \leftarrow \text{argmax}(A[i][j])$ 
13  if  $hmax > 1$  then
14     $counter \leftarrow counter + 1$ 
15    append  $M_c = AND(M_{i_{max}}, M_{j_{max}})$  as a new column to M
16     $M_{i_{max}} \leftarrow M_{i_{max}} - M_c$ 
17     $M_{j_{max}} \leftarrow M_{j_{max}} - M_c$ 
18  end
19 while  $hmax > 1$ ;
20 return  $counter + \sum_{i \in \text{rows}(M)} (\sum_{j \in \text{col}(M)} M[i][j] - 1)$ 

```

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_7 \end{bmatrix}$$

$$= \begin{bmatrix} x_0 + x_1 & & + x_3 & & & & & & \\ x_0 + x_1 & & & & + x_4 & & & & \\ & x_1 + x_2 & & & & + x_5 & & & \\ & x_1 + x_2 + x_3 & & & & & + x_6 & & \\ & x_1 + x_2 + x_3 & & & & & & + x_7 & \\ & x_1 + x_2 & & & & & & & + x_8 \\ x_0 & & + x_2 + x_3 & & & & & & \end{bmatrix}$$

In subsequent steps, we will be analysing the matrix on the right hand side. For simplicity, we will also be omitting the $+$ in the matrices. Note that it is easy to reconstruct the original matrix from the matrix on the right hand side. The naive implementation for this matrix requires 16 XORs. From the matrix with variables on the right, we can start comparing the number of times each pair of variables occurs. The top few are listed below:

Pair	Number of Occurrences
$\{x_1, x_2\}$	4
$\{x_1, x_3\}$	3
$\{x_2, x_3\}$	3
$\{x_0, x_1\}$	2
$\vdots$	$\vdots$

We will select the one at the top and let

$$t_0 = x_1 + x_2$$

and append it to the matrix as a new variable, while removing

the pair from the row that we append to.

$$\xrightarrow{t_0=x_1+x_2} \begin{bmatrix} x_0 & x_1 & & x_3 & & & & & \\ & x_0 & x_1 & & x_4 & & & & \\ & & & & & x_5 & & & t_0 \\ & & & x_3 & & x_6 & & & t_0 \\ & & & x_3 & & & x_7 & & t_0 \\ & & & & & & & x_8 & t_0 \\ x_0 & & x_2 & x_3 & & & & & \end{bmatrix} \quad (3.1)$$

We will continue the process until the number of occurrences at the top of the table is 1, i.e. when there are no more repeated pairs. The series of steps for this example are shown below:

$$\xrightarrow{t_1=x_0+x_1} \begin{bmatrix} & & x_3 & & & & & & t_1 \\ & & & x_4 & & & & & t_1 \\ & & & & x_5 & & & & t_0 \\ & & x_3 & & & x_6 & & & t_0 \\ & & x_3 & & & & x_7 & & t_0 \\ & & & & & & & x_8 & t_0 \\ x_0 & x_2 & x_3 & & & & & & \end{bmatrix} \quad (3.2)$$

$$\xrightarrow{t_2=x_3+t_0} \begin{bmatrix} & & x_3 & & & & & & t_1 \\ & & & x_4 & & & & & t_1 \\ & & & & x_5 & & & & t_0 \\ & & & & & x_6 & & & t_2 \\ & & & & & & x_7 & & t_2 \\ & & & & & & & x_8 & t_0 \\ x_0 & x_2 & x_3 & & & & & & \end{bmatrix} \quad (3.3)$$

To calculate the number of XORs required after using the optimization algorithm, we can use the following formula (assuming

no all-zero rows)

$$\text{Number of XORs} = N + \sum_{i=1}^m (HW(i) - 1)$$

where N is number of iterations, $HW(i)$ is the Hamming Weight of row i of the appended matrix after the algorithm, m is the number of rows of the matrix. In this example, we have to use a total of $3 + (1 + 1 + 1 + 1 + 1 + 1 + 2) = 11$ XOR gates to build this circuit.

Remark 3.2.1. *In (3.2), we can see that the pairs $\{x_0, x_1\}$ and $\{x_3, t_0\}$ were tied as the highest number of occurrences in that iteration of the algorithm. Paar’s second algorithm, Paar2, is one that considers all possible choices at any step; searching all possible branches of the decision tree.*

Remark 3.2.2. *For a $m \times n$ matrix M , for every iteration, Paar1 check a total of $O(mn^2)$ pairs of variables. After each iteration, the Hamming Weight of M strictly decreases. Thus, Paar1 algorithm will terminate after at most mn steps. Hence, the complexity of Paar1 is $O(m^2n^3)$.*

3.2.1 Cancellation-Free programs

One prominent feature of Paar1 and Paar2 is that both algorithms explore only the “cancellation-free” programs. Cancellation-free program is a term coined by Boyar and Peralta in [BMP13]. It refers to programs that perform arithmetic addition operations, $u = v + w$, such that if $x_i \in v$, then $x_i \notin w$. Many other papers such as [Can05] and [SMTM01] have only focused on non-cancellation free programs but they are often restrictive in terms of gate reduction. This restriction prevented the exploration of some paths. In the example above, the last row will be evaluated as

$$x_0 + x_2 + x_3$$

for which 2 XORs are required. However, a more efficient way would only require one operation. After evaluating the first two rows, y_1 and y_2 , y_7 can be evaluated as follows:

$$\begin{aligned} y_1 + y_2 &= t_1 + t_2 \\ &= (x_0 + x_1) + (x_1 + x_2 + x_3) \\ &= x_0 + x_2 + x_3 \end{aligned}$$

3.3 BP Algorithm

In 2010, Boyar and Peralta published a paper on an algorithm which produces linear paths that are not cancellation free [BP10]. The algorithm fared much better on most of the matrices as compared to Paar1 and Paar2 [KLSW17]. From here on, we will term this algorithm as the BP algorithm in this paper.

The BP algorithm, using [KLSW17] implementation, is described in Algorithm 2. The idea is to use a set of base elements and track the distance to each *target*. By choosing a suitable element in each iteration to add into the base, the distance to the *targets* will eventually decrease to 0. If there is a tie, the Euclidean norm is used to resolve the tie. Note that the function *SUM* refers to the sum of the values in the array.

Using the same example in the previous section, an illustration of the BP algorithm is given below:

Dist	Matrix							
2	x_0	x_1	x_3					
2	x_0	x_1		x_4				
2		x_1	x_2		x_5			
3		x_1	x_2	x_3		x_6		
3		x_1	x_2	x_3			x_7	
2		x_1	x_2					x_8
2	x_0		x_2	x_3				

Let $X = \{x_0, x_1, \dots, x_8\}$

Algorithm 2: BP Algorithm

```
1 input: a binary matrix  $M$ 
2 output: number of XORs operations required for evaluating  $M$ 
3 function BP( $M$ );
4 Initialize a set of variables  $\{x_0, x_1, \dots, x_{n-1}\}$  as the base
5 Set each row  $i$  in  $M$  as  $Target[i]$  with a distance  $Dist[i] = HW(i) - 1$ 
6  $counter \leftarrow 0$ 
7 while  $Dist[i] > 0$  for at least one  $i$  do
8    $counter \leftarrow counter + 1$ 
9    $bestDist = \infty$ 
10   $bestNorm = -\infty$ 
11  Initialise  $bestBase$ 
12  for every possible pair of base elements,  $x_i$  and  $x_j$  where  $i \neq j$  do
13     $tryBase \leftarrow x_i + x_j$ 
14    Update  $Dist$  vector with  $tryBase$ 
15     $tryDist \leftarrow SUM(Dist)$ 
16     $tryNorm \leftarrow \|Dist\|$ 
17    if  $newDist < bestDist$  then
18       $bestDist \leftarrow tryDist$ 
19       $bestNorm \leftarrow tryNorm$ 
20       $bestBase \leftarrow tryBase$ 
21    else if  $tryDist = bestDist$  and  $tryNorm > bestNorm$  then
22       $bestDist \leftarrow tryDist$ 
23       $bestNorm \leftarrow tryNorm$ 
24       $bestBase \leftarrow tryBase$ 
25    end
26  end
27  Add  $bestBase$  into the set of base elements
28  Update  $Dist$  vector accordingly
29 end
30 return  $counter$ 
```

Step	Base Elements	New Base	Dist
0	X	$t_0 = x_1 + x_2$	$[2, 2, 1, 2, 2, 1, 2]$
1	$X \cup \{t_0\}$	$y_2 = x_5 + t_0$	$[2, 2, 0, 2, 2, 1, 2]$
2	$X \cup \{t_0, y_2\}$	$y_5 = x_8 + t_0$	$[2, 2, 0, 2, 2, 0, 2]$
3	$X \cup \{t_0, y_2, y_5\}$	$t_3 = x_0 + x_1$	$[1, 1, 0, 2, 2, 0, 2]$
4	$X \cup \{t_0, y_2, y_5, t_3\}$	$y_0 = x_3 + t_3$	$[0, 1, 0, 2, 2, 0, 1]$
5	$X \cup \{t_0, y_2, y_5, t_3, y_0\}$	$y_1 = x_4 + t_3$	$[0, 0, 0, 2, 2, 0, 1]$
6	.	$y_6 = t_0 + y_0$	$[0, 0, 0, 2, 2, 0, 0]$
7		$t_7 = x_3 + t_0$	$[0, 0, 0, 1, 1, 0, 0]$
8		$y_3 = x_6 + t_7$	$[0, 0, 0, 0, 1, 0, 0]$
9		$y_4 = x_7 + t_7$	$[0, 0, 0, 0, 0, 0, 0]$

Note that a cancellation can be observed in step 6. This allows the BP algorithm to be more effective. This example matrix, under the BP algorithm, requires 10 XORs in the circuit. To save some time in searching, whenever there is a *target* that is distance 1 away, the new base that reduces it will be chosen instead regardless of the sum of distances.

In [BP10] and [BMP13], they have acknowledged the importance of tie resolution in the BP algorithm and experimented with various tie breakers. The ideas were revolving around different combinations of the Euclidean norm and the distances.

3.3.1 Complexity of the BP algorithm

The BP algorithm has a high complexity. This is mostly due to the exhaustive search nature for the distance vector. Suppose that there are B base elements. Then, in order to search if the *target* can be reached by $d-1$ elements, we have to try a total of $\binom{B}{d-1}$ number of different sets of base elements in the worst case scenario. In other words, the time required for the algorithm increases exponentially as the size of base elements increases or as the distances of the targets increase. This explains why matrices with a large size or high density work slower.

For a $m \times n$ matrix M , the complexity can be derived as follows:

To check if a potential base can reach $target[i]$ in $d_i - 1$ steps, where d_i is the distance of $target[i]$, we need $O(\binom{n+j}{d_i})$ where $(n+j)$ refers to the number of base elements at the j^{th} iteration. Thus, for all m targets, we would need

$$O\left(\sum_{i=0}^{m-1} \binom{n+j}{d_i}\right)$$

. Since there are $O((n+i)^2)$ potential new bases, we have

$$O\left(\sum_{i=0}^{m-1} \binom{n+j}{d_i} (n+j)^2\right)$$

for each round. For every step, at least one of the d_i has to decrease by 1, the algorithm will terminate with at most $\sum_{i=0}^{m-1} m(d_i - 1) < m(n-1)$ iterations.

The complexity of this algorithm is

$$O\left(\sum_{j=0}^{m(n-1)} \sum_{i=0}^{m-1} \binom{n+j}{d_i} (n+j)^2\right) \quad (3.4)$$

Since $\binom{n+j}{d_i} \leq \left(\frac{(n+j)\exp(1)}{d_i}\right)^{d_i} \leq \left(\frac{(n+j)\exp(1)}{n}\right)^n = \left(1 + \frac{j\exp(1)}{n}\right)^n$ ², we can approximate (3.4) to

$$\begin{aligned} & O\left(\sum_{j=0}^{m(n-1)} m\left(1 + \frac{j\exp(1)}{n}\right)^n (n+j)^2\right) \\ &= O\left(mn\left(1 + \frac{mn\exp(1)}{n}\right)^n (n+mn)^2\right) \\ &= O\left(m^2 n^3 (1 + m\exp(1))^n (m+1)^2\right) \\ &= O\left(m^4 n^3 (1 + m\exp(1))^n\right) \\ &= O(m^n) \end{aligned}$$

Remark 3.3.1. In [VSP18], it was suggested that the BP is not optimal for matrices with high density and a variation of

²The proofs of the inequalities can be found in Appendix A

the BP algorithm that is aimed at dense matrices was proposed. The idea of the algorithm is rather similar to the one mentioned above but with one difference: it measures the distance towards the complement of rows as well. Intuitively, since the complement of the rows in a dense matrix may have a Hamming Weight that is less than the row, thus, it may be easier to achieve the complement and then add to the all-one vector to get the original row vector.

3.4 Recent Developments

3.4.1 Minimizing the XOR count for the AES matrix

The cost of the AES matrix has always been a subject of discussion for many papers. Over the years, many papers have also produced a program that led to a better XOR cost for the AES matrix. The naive cost of implementing the AES matrix is 152. With the basic analysis on the AES matrix described in section 2.5, we will be able to achieve a cost of 124. Using the both Paar1 and Paar2, it has been shown to have a cost of 108 [KLSW17]. While this may be an impressive reduction as compared to the naive method, many other ways have been analyzed to achieve better results. In [BBR16], a similar XOR count was achieved. In [JMPS17], the XOR count is reduced down to 103 XOR using a search tool. Finally, in [KLSW17], BP algorithm showed that this can be further improved to a count of 97. As far as we know, this is the best XOR count thus far.

3.4.2 Constructing new matrices

In [DL18], Duval and Leurent have constructed several MDS matrices. Starting with some known cheap matrices, they add an operation at a time until they obtain MDS matrices. They

also compared the circuits using the various optimization techniques such as BP, Paar1, Paar2. For the convenience of the reader, we have attached it in here. Table 3.1 shows the results of the 16×16 matrices and 32×32 they have obtained. The columns, “Matrix” and “Instantiation”, correspond to the matrix that they used in their paper can be found in [DL18]. We have attached the matrices in Appendix B. The column “Const.” refers to their search results, “Ours” and the column of “Yosys” refers to the results after using the synthesis tool Yosys [Wol] on their own search results (“Const.”). The rest of the columns refer to the various global optimization algorithms that we have mentioned above along with the naive method.

Table 3.1: Note. comparison of circuit cost with several optimization tools from [DL18]

Matrix	Instantiation	Const.	Yosys	BP	Paar1	Paar2	Naive
$M_{4,5}^{9,3}$	$\alpha = A_4$	39/5	35/5	38/7	46/4	45/5	87/4
$M_{4,5}^{9,3}$	$\alpha = A_4^{-1}$	39/5	36/5	40/4	46/4	46/4	77/3
$M_{4,6}^{8,3}$	$\alpha = A_4$	35/5	35/5	38/7	46/4	45/5	87/4
$M_{4,6}^{8,3}$	$\alpha = A_4^{-1}$	35/6	35/5	40/4	46/4	46/4	77/3
$M_{4,5}^{8,3}$	$\alpha = A_4^{-1}, \beta = A_4, \gamma = A_4^{-2}$	36/6	36/6	40/6	48/4	47/4	99/4
$M_{4,4}^{9,4}$	$\alpha = A_4$	40/4	39/4	41/9	49/5	47/5	94/4
$M_{4,4}^{9,3}$	$\alpha = A_4^{-1}, \beta = A_4, \gamma = A_4^2$	40/4	40/4	40/7	45/4	43/4	93/4
$M_{4,4}^{8,4}$	$\alpha = A_4$	38/4	38/4	40/7	43/5	43/5	102/4
$M_{4,4}^{8,4'}$	$\alpha = A_4$	38/4	38/4	43/6	41/4	41/4	100/4
$M_{4,4}^{8,4''}$	$\alpha = A_4$	37/4	37/4	40/5	44/4	43/5	84/4
$M_{4,3}^{9,5}$	$\alpha = A_4$	41/3	41/3	40/4	44/4	43/4	73/3
$M_{4,3}^{9,5}$	$\alpha = A_4^{-1}$	41/3	41/3	43/5	44/3	44/3	77/3
$M_{4,5}^{9,3}$	$\alpha = A_8$	75/5	67/5	74/5	88/4	88/4	161/4
$M_{4,5}^{9,3}$	$\alpha = A_8^{-1}$	75/5	67/5	71/6	89/5	89/5	161/4
$M_{4,6}^{8,3}$	$\alpha = A_8$	67/5	67/5	74/5	88/4	88/4	161/4
$M_{4,6}^{8,3}$	$\alpha = A_8^{-1}$	67/5	67/5	71/6	89/5	89/5	161/4
$M_{4,5}^{8,3}$	$\alpha = A_8^{-1}, \beta = A_8, \gamma = A_8^{-2}$	68/5	68/5	75/6	81/4	77/4	186/4
$M_{4,4}^{9,4}$	$\alpha = A_8$	76/4	76/4	77/6	92/4	92/4	174/4
$M_{4,4}^{9,3}$	$\alpha = A_8^{-1}, \beta = A_8, \gamma = A_8^2$	76/4	76/4	76/6	83/6	83/6	171/4
$M_{4,4}^{8,4}$	$\alpha = A_8$	70/4	70/4	72/5	76/4	76/4	198/4
$M_{4,4}^{8,4'}$	$\alpha = A_8$	70/4	70/4	81/7	79/5	79/5	196/4
$M_{4,4}^{8,4''}$	$\alpha = A_8$	69/4	69/4	72/6	85/5	85/5	160/4
$M_{4,3}^{9,5}$	$\alpha = A_8$	77/3	77/3	76/7	86/4	86/4	145/3
$M_{4,3}^{9,5}$	$\alpha = A_8^{-1}$	77/3	77/3	79/5	86/4	86/4	145/3

While Duval and Leurent's work were not based on global optimization, the matrices they used will serve as the basis of comparison for this paper.

Chapter 4

The Algorithm

4.1 An analysis on the BP algorithm

In view that the BP algorithm produces a record of 97 XORs for M_{AES} , we spent some time analysing the algorithm. After a close examination on how the algorithm works on this particular circuit, we noticed a couple of properties. To simplify the description, we will term the initial criteria of BP algorithm, that is, the sum of the distances to the *targets* as the first criteria. The tie breaker, sum of the Euclidean norm of the distances to the *target* as the second criteria.

1. The second criteria used in this algorithm did not work well. When there is a tie between the first criteria, most of the time, there is a tie of the second criteria as well.
2. The algorithm did not provide a process for cases when there is a tie for both first and second criteria. In [KLSW17], the C++ code implemented in their GitHub repository describes an algorithm for which if there are two potential bases B_x and B_y that are “equally good”, the one that comes first in the loop will be the chosen one.

4.2 A small tweak

We tweak the program such that all “equally good” bases will be taken into consideration and a random choice among them will be chosen. After a few rounds of the program, a new record of 95 XORs for the AES linear circuit can be found. We have provided the linear circuit in Appendix C. In this paper, we will refer this altered algorithm as *BP random*.

4.3 Graph based search

This section explains a technique we employed to visualise the distance vector. Eventually, this idea provided the intuition to the new formulation of the criteria and algorithm.

In an attempt to do further analysis to understand the BP algorithm and to formulate better criteria, we designed a graph based search technique. We plot a tree $T = (V, E)$ where V is the set of nodes and E is the set of edges. Each node n can be viewed as an integer N^1 where $1 \leq N \leq 2^c - 1$, where c is the number of columns in the matrix M . Next, we partition the nodes using their Hamming Weight (which we will refer it as the node’s “level”).

An edge $\{u, w\}$ with $u, w \in V$ exists if $HD(u, w) = k$ where k is a node at level 1 and u and w are from different levels. In the beginning, it can be observed that two nodes u and w have an edge if and only if $HD(u, w) = 1$. We say that w is a child of u if $\{u, w\}$ exists and level of u is less than level of w . We also say that u is a parent of w . It is easy to see that if w is at level p then all the child nodes of w will be at level $p + 1$. A node z is a descendant of w if z is a child of w or it is recursively the descendant of w . When a node shifts from level q to $q - 1$, we

¹It will be easier to visualise if we view the numbers in binary.

Figure 4.1: An example of the graph with 4 bits

will term it as a “promotion”.

Now, suppose we have a matrix M , for each row of M , we convert it from a binary vector to a value. For instance, the vector $[1, 0, 1, 1]$ is converted to $0b1011$. Then, for the node that correspond to this number, we indicate it as $targets[i]$ according to its row index i .

The following information are associated with each node:

1. Its own unique number
2. A tuple with number of unique ways to the $targets[i]$ for $i = 0, 1, \dots, c - 1$
3. A list of its parent nodes
4. A list of its child nodes
5. The level of the node
6. A boolean flag to indicate if the node is a *target*

We can do a comparison with the BP algorithm here. The initial base in BP algorithm corresponds to the nodes at level 1. In each iteration, we have to promote a node, n , from level 2 to level 1. Thus adding the value n into the base. To maintain the structure of the graph described above, there are several changes required for a promotion. Suppose k is the selected node to be promoted from level 2 to level 1, then

1. Every descendant of k will be promoted.
2. Additional edges will be added to some nodes.
3. Some existing edges will be removed.

The reasons are given as follows: (1) is rather straightforward here. (2): a new node being added to level 1, thus, previously disconnected nodes can form new edges if the Hamming Distance of the 2 nodes equals k . (3): Suppose v is a child of u , and v got promoted but not u (via some other parent nodes). Then u and v are at the same level.

In Figure 4.1, an example of the graph representation of the matrix

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

is shown. The red nodes, 1011 and 1110, are the target nodes ($targets[0]$ and $targets[1]$ respectively). The i^{th} element in the tuple associated with each node is the number of unique ways to the $targets[i]$. For instance, the node 0010 has two unique paths to the targets 1011 and 1110:

$$0010 \rightarrow 0011 \rightarrow 1011$$

$$0010 \rightarrow 1001 \rightarrow 1011$$

$$0010 \rightarrow 0110 \rightarrow 1110$$

$$0010 \rightarrow 1010 \rightarrow 1110$$

In this graph based search technique, distance to a target is defined as the minimum number of promotions required to promote a *target* to level 1. The aim of this approach is to find the shortest way to promote all *target* nodes to level 1. Viewing it as a graph provides several advantages including opening up to more criterion as compared to the BP algorithm. Each node is able store information such as the number of child nodes it has, the number of targets it can reach and the number of unique paths to the targets a node has. Another advantage of this technique is that it can potentially allow the removal of nodes that are deemed to be redundant or to simply reduce the search space for the circuit. For instance, impose a maximum Hamming Weight of a node, reducing the number of nodes, thereby reducing the complexity of the algorithm. Also, we can visualise the algorithm and knowing what exactly happens to each node in each iteration.

We have explored the following criterion that can be used to choose which node to be promoted.

- C1: Minimum sum of distance
- C2: Maximum Euclidean norm of the distance vector
- C3: Minimum Euclidean norm of the distance vector
- C4: Minimum sum of unique paths to *targets*
- C5: Maximum sum of unique paths to *targets*

We have tried various combinations of the above criterion, using one as a main criteria and another one as a tie breaker. Note that if *C1* is used as the main criteria and *C2* is used as the tie breaker, then it will be similar to the BP algorithm. Two particular combination stood out: using *C1* as the main criteria and using *C4* and *C5* as the tie breaker in each combination. Randomness in choosing “equally good” nodes in the case that the tie breaker was not able to distinguish between potential nodes is implemented too. Intuitively, using a tie breaker of

$C5$ should be relatively better than $C4$ as more unique paths implies a stronger connectivity between the nodes and may suggest more nodes are promoted. However, a preliminary test of both options show that $C4$ is a better tie breaker. We generated 20 random matrices of size 12×12 with density $\rho = 0.5$. Table 4.1 shows that comparison between the various algorithms after 5000 iterations. The second value in each entry is the average value. In table 4.1, we can see that $C4$ produces better results in 16 of the matrices and only 2 matrices have more XOR gates in $C4$ than in $C5$. This gives the motivation for our heuristic.

Table 4.1: A comparison of new tie breakers to BP random, BP and Paar1

Matrices	$C5$	$C4$	BP random	BP	Paar1
randmat0	29/30.0	28/29.4	28/29.1	30	32
randmat1	31/32.2	29/29.3	29/29.3	29	34
randmat2	29/31.6	31/31.4	28/29.9	30	36
randmat3	30/31.3	28/29.5	29/29.8	29	33
randmat4	32/32.9	30/31.4	30/31.5	32	35
randmat5	29/30.2	27/28.4	27/28.5	28	33
randmat6	31/32.4	29/29.3	28/29.2	29	34
randmat7	28/31.0	29/29.2	28/28.5	28	31
randmat8	30/30	29/29	29/29	29	34
randmat9	30/32.3	28/29.0	28/29.2	30	34
randmat10	29/30.8	29/29	28/29.0	29	33
randmat11	27/28.2	26/26.6	26/26.8	26	32
randmat12	27/28.5	26/26.5	26/27.0	27	30
randmat13	30/31.2	29/29	28/29.0	30	32
randmat14	29/30.0	27/27.7	27/27.6	27	31
randmat15	28/28	28/29.2	28/29.2	29	30
randmat16	32/33.0	29/30.0	29/30.1	30	35
randmat17	31/33.0	30/30	29/30.0	30	34
randmat18	29/29.5	27/27	27/27	27	30
randmat19	28/28.5	27/27.5	27/27.6	27	33

4.4 An intuition and algorithm

A view of the graph search method shows that new edges are constantly formed and some edges are removed from the graph as a result of the promotion. This prompted us to review the

criteria in the BP algorithm. The purpose of choosing the one that minimizes the sum of distances is an attempt to find the longest common path. Intuitively this looks like a good strategy, as it aims to find a new base such that the sum of distances is reduced to the minimum. However, in the graph perspective, the graph is constantly being updated in every iteration, which may cause what seems to be a wise choice now to become not that good later on. More importantly, it diverts the attention away from achieving the targets itself which are the ones that we are actually aiming for. Thus, we decided to add a filter prior to the BP criteria to always go for the nearest target(s) first. The pseudocode is given in Algorithm 3 shows the filter followed by the BP random criterion. As mentioned previously, the original tie breaker did not actually help much in resolving the tie. Thus, we have also made a version to do away with the tie breaker which can be seen in Algorithm 4.

We illustrate Algorithm 3 using the same example in Chapter 3. Let $X = \{x_0, x_1, \dots, x_8\}$

Step	Base Elements	New Base	Dist
0	X	$t_0 = x_1 + x_2$	$[2, 2, 1, 2, 2, 1, 2]$
1	$X \cup \{t_0\}$	$y_2 = x_5 + t_0$	$[2, 2, 0, 2, 2, 1, 2]$
2	$X \cup \{t_0, y_2\}$	$y_5 = x_8 + t_0$	$[2, 2, 0, 2, 2, 0, 2]$
3	$X \cup \{t_0, y_2, y_5\}$	$t_3 = x_3 + t_0$	$[2, 2, 0, 1, 1, 0, 2]$
4	$X \cup \{t_0, y_2, y_5, t_3\}$	$y_3 = x_6 + t_3$	$[2, 2, 0, 0, 1, 0, 2]$
5	$X \cup \{t_0, y_2, y_5, t_3, y_3\}$	$y_4 = x_7 + t_3$	$[2, 2, 0, 0, 0, 0, 2]$
6	.	$t_6 = x_0 + x_1$	$[1, 1, 0, 0, 0, 0, 1]$
7		$y_0 = x_3 + t_6$	$[0, 1, 0, 0, 0, 0, 1]$
8		$y_1 = x_4 + t_6$	$[0, 0, 0, 0, 0, 0, 1]$
9		$y_6 = t_3 + t_6$	$[0, 0, 0, 0, 0, 0, 0]$

The main difference for Algorithm 3 and 4 from BP algorithm is that it favours a new base that reduces the distance to the nearest target first instead of the one that minimizes the sum of distances the most. While this may sound rather counter-intuitive, the results showed that this algorithm works better for

Algorithm 3: A1 Algorithm

```
1 input: a binary matrix  $M$ 
2 output: number of XORs operations required for evaluating  $M$ 
3 function  $A1(M)$ ;
4 Initialize a set of variables  $\{x_0, x_1, \dots, x_{n-1}\}$  as the base
5 Set each row  $i$  in  $M$  as  $Target[i]$  with a distance  $Dist[i] = HW(i) - 1$ 
6  $counter \leftarrow 0$ 
7 while  $Dist[i] > 0$  for at least one  $i$  do
8    $counter \leftarrow counter + 1$ 
9    $bestDist = \infty$ 
10   $bestNorm = -\infty$ 
11  Initialise  $bestBase\_list$  as a list
12  for every possible pair of base elements,  $x_i$  and  $x_j$  where  $i \neq j$  do
13     $tryBase \leftarrow x_i + x_j$ 
14    Update  $Dist$  vector with  $tryBase$ 
15     $tryDist \leftarrow SUM(Dist)$ 
16     $tryNorm \leftarrow \|Dist\|$ 
17    if  $tryBase$  does not reduce the  $Dist[i]$  where  $i$  is the nearest target
18      then
19        | continue
20    end
21    if  $newDist < bestDist$  then
22      | Clear  $bestBase\_list$ 
23      |  $bestDist \leftarrow tryDist$ 
24      |  $bestNorm \leftarrow tryNorm$ 
25      |  $bestBase\_list \leftarrow tryBase$ 
26    else if  $tryDist = bestDist$  and  $tryNorm > bestNorm$  then
27      | Clear  $bestBase\_list$ 
28      |  $bestDist \leftarrow tryDist$ 
29      |  $bestNorm \leftarrow tryNorm$ 
30      |  $bestBase\_list \leftarrow tryBase$ 
31    else if  $tryDist = bestDist$  and  $tryNorm == bestNorm$  then
32      |  $bestBase\_list \leftarrow tryBase$ 
33    end
34  end
35  Randomly pick one from  $bestBase\_list$  and add into the set of base
    elements
36  Update  $Dist$  vector accordingly
37 end
38 return counter
```

Algorithm 4: A2 Algorithm

```
1 input: a binary matrix  $M$ 
2 output: number of XORs operations required for evaluating  $M$ 
3 function A2( $M$ );
4 Initialize a set of variables  $\{x_0, x_1, \dots, x_{n-1}\}$  as the base
5 Set each row  $i$  in  $M$  as  $Target[i]$  with a distance  $Dist[i] = HW(i) - 1$ 
6  $counter \leftarrow 0$ 
7 while  $Dist[i] > 0$  for at least one  $i$  do
8    $counter \leftarrow counter + 1$ 
9    $bestDist = \infty$ 
10   $bestNorm = -\infty$ 
11  Initialise  $bestBase\_list$  as a list
12  for every possible pair of base elements,  $x_i$  and  $x_j$  where  $i \neq j$  do
13     $tryBase \leftarrow x_i + x_j$ 
14    Update  $Dist$  vector with  $tryBase$ 
15     $tryDist \leftarrow SUM(Dist)$ 
16     $tryNorm \leftarrow ||Dist||$ 
17    if  $tryBase$  does not reduce the  $Dist[i]$  where  $i$  is the nearest target
18      then
19        | continue
20    end
21    if  $newDist < bestDist$  then
22      | Clear  $bestBase\_list$ 
23      |  $bestDist \leftarrow tryDist$ 
24      |  $bestNorm \leftarrow tryNorm$ 
25      |  $bestBase\_list \leftarrow tryBase$ 
26    else if  $tryDist = bestDist$  then
27      |  $bestBase\_list \leftarrow tryBase$ 
28    end
29  Randomly pick one from  $bestBase\_list$  and add into the set of base
30  elements
31  Update  $Dist$  vector accordingly
32 end
33 return counter
```

some matrices. For the rest of the paper, we will refer Algorithm 3 and 4 as A1 and A2 respectively.

Chapter 5

Results

In this section, the results of the new heuristics, A1 and A2, are shown. In the following, we will be presenting the results based on their sizes. Although the matrices we have tested on are square matrices, i.e. $n \times n$, the algorithms can be extended to non-square matrices. As A1, A2 and BP random are not deterministic, a number of rounds with the program have been tested and the best result is obtained. The number of iterations depends on the size and density of the matrix and will be made clear in each section.

5.1 Randomized 12×12 matrices

For preliminary testing, we have generated 20 random 12×12 matrices (over $GF(2)$) with a density, $\rho = 0.5$ to test on. We have compared the results across various algorithms as shown in Table 5.1. The number of rounds is 5000. In each entry, the best circuit with the average count of all 5000 rounds are shown. BP random shows a slight improvement than BP in 12 of the matrices. A1 and A2 showed improvement in 18 and 17 of the matrices respectively when compared to the circuit generated by the BP algorithm.

Table 5.1: A comparison of A1, A2 to BP random, BP and Paar1

Matrices	A1	A2	BP random	BP	Paar1
randmat0	27/28.7	27/28.1	28/29.1	30	32
randmat1	28/29.0	28/28.7	29/29.3	29	34
randmat2	28/30.5	28/28.6	28/29.9	30	36
randmat3	28/29.6	29/29	29/29.8	29	33
randmat4	29/31.0	29/29.9	30/31.5	32	35
randmat5	27/28.5	27/28.2	27/28.5	28	33
randmat6	28/28.4	28/28.1	28/29.2	29	34
randmat7	27/28.5	27/27.7	28/28.5	28	31
randmat8	28/28.8	27/27.9	29/29	29	34
randmat9	27/27.5	27/27.8	28/29.2	30	34
randmat10	26/28.5	28/28.5	28/29.0	29	33
randmat11	27/27.9	27/27.6	26/26.8	26	32
randmat12	25/26.0	26/26	26/27.0	27	30
randmat13	28/28.8	28/28.8	28/29.0	30	32
randmat14	27/28.6	28/28.3	27/27.6	27	31
randmat15	27/29.1	28/29.0	28/29.2	29	30
randmat16	29/29.7	29/29.6	29/30.1	30	35
randmat17	27/28.6	28/28.4	29/30.0	30	34
randmat18	25/27.5	26/26.7	27/27	27	30
randmat19	26/26.5	26/26.4	27/27.6	27	33

5.2 On random matrices from 16×16 to 20×20

We have also implemented the new algorithm on a benchmark set [Vis] that was mentioned in [VSP18]. Due to time constraints, we only tested matrices of size ranging from 15×15 to 20×20 , with density $\rho = 0.1, 0.2, \dots, 0.9$. For each matrix size and density, there are a total of 150 matrices. Thus, a total of $6 \times 9 \times 150 = 8100$ matrices were evaluated. We ran the algorithm for BP, BP random, A1 and A2. We ran 100 rounds for the BP random, A1 and A2 algorithms and the best XOR count for each matrix was recorded. The results of the matrices using BP algorithm served as a basis. The difference between the best gate count obtained with BP random, A1 and A2 with BP were computed. For each matrix size and density, we average out the difference among the 150 matrices. In Figure 5.1, a

graph representing the differences is shown. *BPdiff*, *A1diff* and *A2diff* refer to the XOR count difference obtained by the best implementation obtained by BP random with BP, A1 with BP and A2 with BP respectively. We can see that A1 and A2 algorithms seem to be performing better when the density $\rho \approx 0.6$ with A2 performing better than A1 in general.

Figure 5.1: A comparison of BP random, A1 and A2 with respect to BP

5.3 On MDS matrices constructed in [DL18]

We choose the matrices that are of size 4×4 with elements in $GF(2^4)$ and $GF(2^8)$ from [DL18]. These matrices translate to matrices of size 16×16 and 32×32 with elements in $GF(2)$. The results are shown in Table 5.2. We omitted the columns on *Paar1*, *Paar2* and *Naive* as the values from those columns can be found in Table 3.1. The first 12 matrices are matrices of size 16×16 and the number of rounds computed was 5000.

As compared to BP and BP random, A1 and A2 have improvements in the XOR count, especially in the case of A1. The next 12 matrices are matrices of size 32×32 . As the running time for A1 and A2 are large, we compare the results generated in a span of 48 hours on a machine with AMD Opteron(TM) Processor 6272 and 130 GB of RAM. Each matrix runs on a single core. In some matrices such as $M_{4,4}^{8,4'}$ and $M_{4,5}^{8,3}$, significant improvements are observed, whereas in matrices such as $M_{4,5}^{9,3}$ and $M_{4,6}^{8,3}$, A1 seems to be performing below-par to the rest of the algorithms¹. It is good to note again that the ‘Yosys’ column refers to the results obtained after using Yosys synthesis tool on the circuits obtained from ‘Const.’

¹We do not include Yosys synthesis tool in the comparison list as it is used after a relatively good circuit has been formed. Experimental results showed that using the tool directly on the naive implementation of the circuits do not yield any good results.

Table 5.2: Comparison of circuit cost of A1 and A2 with several optimization tools from [DL18]

Matrix	Instantiation	A1	A2	Const.	Yosys	BP	BP rand
$M_{4,5}^{9,3}$	$\alpha = A_4$	39	38	39/5	35/5	38/7	38
$M_{4,5}^{9,3}$	$\alpha = A_4^{-1}$	38	36	39/5	36/5	40/4	39
$M_{4,6}^{8,3}$	$\alpha = A_4$	39	38	35/5	35/5	38/7	38
$M_{4,6}^{8,3}$	$\alpha = A_4^{-1}$	38	36	35/6	35/5	40/4	39
$M_{4,5}^{8,3}$	$\alpha = A_4^{-1}, \beta = A_4, \gamma = A_4^{-2}$	39	39	36/6	36/6	40/6	39
$M_{4,4}^{9,4}$	$\alpha = A_4$	39	39	40/4	39/4	41/9	40
$M_{4,4}^{9,3}$	$\alpha = A_4^{-1}, \beta = A_4, \gamma = A_4^{-2}$	42	40	40/4	40/4	40/7	40
$M_{4,4}^{8,4}$	$\alpha = A_4$	40	39	38/4	38/4	40/7	39
$M_{4,4}^{8,4'}$	$\alpha = A_4$	40	40	38/4	38/4	43/6	42
$M_{4,4}^{8,4''}$	$\alpha = A_4$	40	37	37/4	37/4	40/5	40
$M_{4,3}^{9,5}$	$\alpha = A_4$	41	40	41/3	41/3	40/4	40
$M_{4,3}^{9,5}$	$\alpha = A_4^{-1}$	41	40	41/3	41/3	43/5	41
$M_{4,5}^{9,3}$	$\alpha = A_8$	79	71	75/5	67/5	74/5	69
$M_{4,5}^{9,3}$	$\alpha = A_8^{-1}$	80	68	75/5	67/5	71/6	69
$M_{4,6}^{8,3}$	$\alpha = A_8$	79	71	67/5	67/5	74/5	69
$M_{4,6}^{8,3}$	$\alpha = A_8^{-1}$	80	68	67/5	67/5	71/6	69
$M_{4,5}^{8,3}$	$\alpha = A_8^{-1}, \beta = A_8, \gamma = A_8^{-2}$	74	69	68/5	68/5	75/6	75
$M_{4,4}^{9,4}$	$\alpha = A_8$	76	77	76/4	76/4	77/6	76
$M_{4,4}^{9,3}$	$\alpha = A_8^{-1}, \beta = A_8, \gamma = A_8^2$	78	77	76/4	76/4	76/6	75
$M_{4,4}^{8,4}$	$\alpha = A_8$	71	71	70/4	70/4	72/5	70
$M_{4,4}^{8,4'}$	$\alpha = A_8$	75	74	70/4	70/4	81/7	80
$M_{4,4}^{8,4''}$	$\alpha = A_8$	77	71	69/4	69/4	72/6	70
$M_{4,3}^{9,5}$	$\alpha = A_8$	78	76	77/3	77/3	76/7	76
$M_{4,3}^{9,5}$	$\alpha = A_8^{-1}$	78	78	77/3	77/3	79/5	77

5.4 On AES Matrix, M_{AES}

When implemented on M_{AES} , A2 produced a circuit that requires 94 XOR gate operations. Due to the indeterministic nature of this algorithm, the circuit can be found in Appendix D.

Chapter 6

Conclusion

In this thesis, we presented a total of 3 different algorithms. They are BP random, A1 and A2. While BP random is just a tweak of the BP algorithm which was used in many studies, the potential of BP random in reducing the circuit size should not be ignored. Also, we have described two new heuristics, A1 and A2, which performed better than BP and BP random in general, especially on matrices with density $\rho \approx 0.5, 0.6$.

While construction-based approaches [DL18] will naturally lead to diffusion matrices with lower number of XORs, our new algorithms, BP random, A1 and A2 can be applied to evaluate the number of XOR operations for any matrices in general.

6.1 Limitations

While A1 and A2 seem to be similar to the BP and BP random algorithm, the running time of the algorithm could potentially be slower than BP and BP random. In section 3.3.1, we see that the complexity of the BP algorithm depends on the size of the base elements and the distance of the targets. In particular, we are more concerned about the targets that are far away. In BP, the criteria is to choose a new base such that it reduces the sum

of distances the most as compared to other bases. In A1 and A2, the criterion are in a way such that they focus on reducing the nearer *targets* first. This causes *targets* that are far away to remain far with the size of the base elements increasing in A1 and A2. Because of the high distance and large base size, the running time eventually increases. Hence, A1 and A2 may be restricted to small size matrices with low density. Ways to improve the calculation of the distance can be explored in order to reduce the amount of time required.

6.2 Further Research

As seen in the minor difference in the results of A1 and A2, the tie breaker in BP did not work as good for the filter applied in A1 and A2. Thus, there may be other tie breakers that are more suitable for this case. It may be of interest to explore more on the graph search technique by trying out different criterion and observe how those criterion interact with one another.

The improvement that BP random has over BP algorithm shows that randomization is important for searching a circuit with the lowest number of gates. Hence, it may be interesting to develop more randomized algorithms for similar optimization problems.

While in Figure 5.1, it is shown that A1 and A2 perform better than BP random and BP in random matrices of size 15×15 to 20×20 , further analysis can be done to review the algorithm's potential on larger matrices or matrices that follow a certain structure.

Bibliography

- [BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-aes: A compact implementation of the aes encryption/decryption core. volume 10095, pages 173–190, 12 2016. doi:10.1007/978-3-319-49890-4_10.
- [BCG⁺12] J. Borghoff, Anne Michele Canteaut, T. Guneysu, E.B. Kavun, M. Knezevic, Lars Ramkilde Knudsen, Gregor Leander, V. Nikov, C. Paar, and Christian Rechberger. Prince - a low-latency block cipher for pervasive computing applications: Extended abstract. In *Advances in Cryptology – ASIACRYPT 2012*, pages 208–225. Springer, 2012. doi:10.1007/978-3-642-34961-4_14.
- [BDJR97] Mihir Bellare, Anand Desai, E. Jorjipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption: Analysis of the des modes of operation. In *FOCS 1997*, 1997.
- [BKL⁺07] Andrey Bogdanov, Lars Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C Vikkelse. Present: an ultra-lightweight block cipher. volume 4727, pages 450–466, 09 2007. doi:10.1007/978-3-540-74735-2_31.
- [BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to

- cryptology. *Journal of Cryptology*, 26(2):280–312, Apr 2013. doi:10.1007/s00145-012-9124-7.
- [BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms*, pages 178–189, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [Can05] D. Canright. A very compact s-box for aes. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 441–455, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [DH79] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, March 1979. doi:10.1109/PROC.1979.11256.
- [DL18] Sébastien Duval and Gaëtan Leurent. Mds matrices with lightweight circuits. *IACR Transactions on Symmetric Cryptology*, 2018(2):48–78, Jun. 2018. doi:10.13154/tosc.v2018.i2.48-78.
- [DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag Berlin Heidelberg, 2002. doi:10.1007/978-3-662-04722-4.
- [Fou98] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. O’Reilly & Associates Inc., 1998.
- [JMPS17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-sliding: A generic technique for bit-serial implementations of spn-based primitives. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems –*

- CHES 2017*, pages 687–707, Cham, 2017. Springer International Publishing.
- [Kel06] S Kelly. Security implications of using the data encryption standard (des). 01 2006.
 - [KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for mds matrices. *IACR Transactions on Symmetric Cryptology*, 2017(4):188–211, Dec. 2017. doi:10.13154/tosc.v2017.i4.188–211.
 - [Li] Ninghui Li. Introduction to cryptography. URL: https://www.cs.purdue.edu/homes/ninghui/courses/Fall105/lectures/355_Fall105_lect05.pdf.
 - [Mas91] Edoardo D. Mastrovito. Vlsi architectures for computation in galois fields. 1991.
 - [Paa97] C. Paar. Optimized arithmetic for reed-solomon encoders. In *Proceedings of IEEE International Symposium on Information Theory*, pages 250–, June 1997. doi:10.1109/ISIT.1997.613165.
 - [SB88] M. E. Smid and D. K. Branstad. Data encryption standard: past and future. *Proceedings of the IEEE*, 76(5):550–559, May 1988. doi:10.1109/5.4441.
 - [Sha49] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, Oct 1949. doi:10.1002/j.1538-7305.1949.tb00928.x.
 - [SMTM01] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact rijndael hardware architecture with s-box optimization. pages 239–254, 11 2001. doi:10.1007/3-540-45682-1_15.

- [Vau95] Serge Vaudenay. On the need for multipermutations: Cryptanalysis of md4 and safer. In Bart Preneel, editor, *Fast Software Encryption*, pages 286–297, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- [Vis] Andrea Visconti. A benchmark set. URL: <http://homes.di.unimi.it/visconti/RESEARCH/BenchmarkSet.zip>.
- [VSP18] Andrea Visconti, Chiara Valentina Schiavo, and René Peralta. Improved upper bounds for the expected circuit complexity of dense systems of linear equations over $gf(2)$. *Information Processing Letters*, 137:1 – 5, 2018. doi:<https://doi.org/10.1016/j.ipl.2018.04.010>.
- [Wol] Clifford Wolf. Yosys open synthesis suite. <http://www.clifford.at/yosys/>.
- [WTS⁺85] Charles C. Wang, Trieu-Kien Truong, Howard M. Shao, Leslie J. Deutsch, Jim K. Omura, and Irving S. Reed. VLSI architectures for computing multiplications and inverses in $GF(2^m)$. *IEEE Trans. Computers*, 34(8):709–717, 1985. URL: <https://doi.org/10.1109/TC.1985.1676616>, doi:10.1109/TC.1985.1676616.

Appendix A

Proofs for inequalities from Section 3.3.1

$$\binom{n+j}{d_i} \leq \left(\frac{(n+j)\exp(1)}{d_i}\right)^{d_i} \leq \left(\frac{(n+j)\exp(1)}{n}\right)^n$$

First inequality: $\binom{n+j}{d_i} \leq \left(\frac{(n+j)\exp(1)}{d_i}\right)^{d_i}$

Proof.

$$\begin{aligned} \binom{n+j}{d_i} &= \frac{(n+j)!}{(d_i)!(n+j-d_i)!} \\ &= \frac{\prod_{k=n+j-d_i+1}^{n+j} k}{d_i!} \\ &\leq \frac{(n+j)^{d_i}}{d_i!} \\ &\leq \left(\frac{(n+j)\exp(1)}{d_i}\right)^{d_i} \end{aligned}$$

where the last inequality is obtained by applying the Stirling's formula, $n! \geq \left(\frac{n}{\exp(1)}\right)^n$. □

Second Inequality: $\left(\frac{(n+j)\exp(1)}{d_i}\right)^{d_i} \leq \left(\frac{(n+j)\exp(1)}{n}\right)^n$

Proof. Since $n > d_i \forall i$, if we show that $\left(\frac{(n+j)\exp(1)}{k}\right)^k \leq \left(\frac{(n+j)\exp(1)}{k+1}\right)^{k+1}$

for all $k < n$ then we are done. Note that

$$(n+j)exp(1) \geq nexp(1) \geq (k+1)exp(1) \geq (k+1)(1+\frac{1}{k})^k = (\frac{k+1}{k})^k(k+1)$$

Thus,

$$\begin{aligned} ((n+j)exp(1))^{k+1} &\geq \frac{(k+1)^{k+1}}{k^k} ((n+j)exp(1))^k \\ \implies (\frac{(n+j)exp(1)}{k+1})^{k+1} &\geq (\frac{(n+j)exp(1)}{k})^k \end{aligned}$$

□

Appendix B

Matrices from [DL18]

The following matrices are obtained from [DL18] with the instantiation matrices:

$$A_4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad A_4^{-1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$A_8 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad A_8^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$M_{4,6}^{8,3}$	$\begin{pmatrix} 3 & 1 & 4 & 4 \\ 1 & 3 & 6 & 4 \\ 2 & 2 & 3 & 1 \\ 3 & 2 & 1 & 3 \end{pmatrix}$	$M_{4,4}^{8,4}$	$\begin{pmatrix} 5 & 7 & 1 & 3 \\ 4 & 6 & 1 & 1 \\ 1 & 3 & 5 & 7 \\ 1 & 1 & 4 & 6 \end{pmatrix}$
$M_{4,4}^{8,4'}$	$\begin{pmatrix} 2 & 2 & 3 & 1 \\ 1 & 3 & 6 & 4 \\ 3 & 1 & 4 & 4 \\ 3 & 2 & 1 & 3 \end{pmatrix}$	$M_{4,4}^{8,4''}$	$\begin{pmatrix} 6 & 7 & 1 & 5 \\ 2 & 3 & 1 & 1 \\ 1 & 5 & 6 & 7 \\ 1 & 1 & 2 & 3 \end{pmatrix}$
$M_{4,5}^{8,3}$	$\begin{pmatrix} \alpha & \alpha + \gamma & \gamma & \gamma \\ 1 & \gamma + 1 & \gamma + \beta & \gamma + \beta + 1 \\ 1 & 1 & \beta & \beta + 1 \\ \alpha + 1 & \alpha + \gamma + 1 & \gamma & \gamma + 1 \end{pmatrix}$	$M_{4,4}^{8,4''}$	$\begin{pmatrix} 3 & 2 & 1 & 3 \\ 2 & 3 & 1 & 1 \\ 1 & 3 & 6 & 4 \\ 1 & 1 & 4 & 6 \end{pmatrix}$
$M_{4,4}^{9,4}$	$\begin{pmatrix} 1 & 2 & 4 & 3 \\ 2 & 3 & 2 & 3 \\ 3 & 3 & 5 & 1 \\ 3 & 1 & 1 & 3 \end{pmatrix}$	$M_{4,3}^{9,5}$	$\begin{pmatrix} \alpha + \alpha^{-1} & \alpha & 1 & 1 \\ 1 & \alpha + 1 & \alpha & \alpha^{-1} \\ 1 + \alpha^{-1} & 1 & 1 & 1 + \alpha^{-1} \\ \alpha^{-1} & \alpha^{-1} & 1 + \alpha^{-1} & 1 \end{pmatrix}$
$M_{4,4}^{9,3}$	$\begin{pmatrix} \alpha + 1 & \alpha & \gamma + 1 & \gamma + 1 \\ \beta & \beta & 1 & \beta \\ 1 & 1 & \gamma & \gamma + 1 \\ \alpha & \alpha + 1 & \gamma + 1 & \gamma \end{pmatrix}$		

Appendix C

95 XOR implementation of AES matrix

Note that $x_1, x_2, \dots, x_n$ are the 32 input bits and $y_0, y_1, \dots, y_n$ are the 32 output bits.

$t0 = x8 + x16$	$t32 = t3 + t28$	$t64 = t26 + t28$
$t1 = x0 + x24$	$y24 = x25 + t32$	$y8 = t1 + t64$
$t2 = x16 + x24$	$t34 = t0 + t8$	$t66 = x22 + y5$
$t3 = x0 + x8$	$y0 = x24 + t34$	$y13 = t23 + t66$
$t4 = x7 + x31$	$t36 = x18 + t29$	$t68 = y31 + t49$
$t5 = x17 + x25$	$y17 = y25 + t36$	$y6 = y14 + t68$
$t6 = x10 + x18$	$t38 = x10 + t9$	$t70 = t5 + t6$
$t7 = x15 + x23$	$y18 = t15 + t38$	$y9 = x1 + t70$
$t8 = x1 + x9$	$t40 = x27 + t6$	$t72 = t10 + t17$
$t9 = x2 + x26$	$t41 = x2 + x3$	$t73 = x28 + t72$
$t10 = x5 + x13$	$y26 = t40 + t41$	$y4 = t3 + t73$
$t11 = x22 + x30$	$t43 = x15 + t2$	$t75 = t1 + t16$
$t12 = x21 + x29$	$y23 = t4 + t43$	$t76 = x19 + t75$
$t13 = x6 + x14$	$t45 = x31 + t7$	$y27 = t14 + t76$
$t14 = x3 + x11$	$y7 = t3 + t45$	$t78 = x13 + t75$
$t15 = x19 + x27$	$t47 = t1 + t4$	$t79 = x20 + t2$
$t16 = x4 + x28$	$y31 = t45 + t47$	$t80 = x29 + t73$
$t17 = x12 + x20$	$t49 = t13 + t43$	$y28 = t78 + t80$
$t18 = x23 + t0$	$t50 = x21 + x30$	$t82 = t12 + t79$
$y15 = t4 + t18$	$y29 = t23 + t50$	$t83 = t17 + t82$
$t20 = x15 + t11$	$t52 = t45 + t49$	$y20 = t16 + t83$
$t21 = x13 + t12$	$y22 = x30 + t52$	$t85 = y4 + t82$
$y5 = t13 + t21$	$t54 = t6 + t36$	$y12 = y28 + t85$
$t23 = x6 + t10$	$y1 = x9 + t54$	$t87 = x3 + t17$
$t24 = x6 + t20$	$t56 = x8 + t5$	$t88 = t15 + t87$
$y14 = t18 + t24$	$y16 = t1 + t56$	$y11 = t0 + t88$
$t26 = x17 + t8$	$t58 = x26 + t6$	$t90 = x28 + t79$
$y25 = t9 + t26$	$y2 = t14 + t58$	$t91 = t14 + t90$
$t28 = x1 + x16$	$t60 = x22 + t13$	$y19 = x27 + t91$
$t29 = x2 + t5$	$y30 = t47 + t60$	$t93 = y11 + t91$
$t30 = x29 + t11$	$t62 = x11 + t40$	$y3 = t75 + t93$
$y21 = t10 + t30$	$y10 = y18 + 62$	

Appendix D

94 XOR implementation of AES matrix

Note that $x_1, x_2, \dots, x_n$ are the 32 input bits and $y_0, y_1, \dots, y_n$ are the 32 output bits.

$t0 = x0 + x24$	$y9 = y17 + t31$	$t64 = t46 + t62$
$t1 = x9 + x17$	$t33 = y1 + t13$	$y28 = t63 + t64$
$t2 = t0 + t1$	$y25 = t30 + t33$	$t66 = x28 + t63$
$y8 = x16 + t2$	$t35 = x11 + t30$	$t67 = x3 + t19$
$t4 = x2 + x10$	$t36 = x27 + t35$	$y27 = t66 + t67$
$t5 = x25 + t1$	$y2 = t23 + t36$	$t69 = t22 + t39$
$y1 = t4 + t5$	$y18 = t20 + t36$	$t70 = t62 + t69$
$t7 = x15 + x23$	$t39 = x8 + x16$	$y11 = x19 + t70$
$t8 = x7 + t0$	$t40 = x23 + t39$	$t72 = x12 + t10$
$y31 = t7 + t8$	$t41 = t11 + t40$	$t73 = x4 + t19$
$t10 = x0 + x8$	$y23 = y31 + t41$	$t74 = t72 + t73$
$t11 = x31 + t10$	$t43 = x7 + t10$	$y3 = x27 + t74$
$y7 = t7 + t11$	$y15 = t41 + t43$	$t76 = y11 + t74$
$t13 = x1 + x25$	$t45 = x22 + x30$	$y19 = t66 + t76$
$t14 = x16 + t13$	$t46 = x5 + x29$	$t78 = t0 + t50$
$y24 = t10 + t14$	$t47 = x13 + t45$	$t79 = y6 + t78$
$t16 = x18 + x26$	$y21 = t46 + t47$	$t80 = t11 + t79$
$t17 = x9 + t13$	$t49 = x15 + t45$	$y30 = x22 + t80$
$y17 = t16 + t17$	$t50 = x6 + t49$	$t82 = x30 + t79$
$t19 = x11 + x19$	$y14 = t40 + t50$	$y22 = y15 + t82$
$t20 = x2 + t19$	$t52 = t43 + t49$	$t84 = x29 + t66$
$y10 = t16 + t20$	$y6 = x14 + t52$	$t85 = t39 + t84$
$t22 = x3 + x27$	$t54 = x21 + x22$	$t86 = t72 + t85$
$t23 = x18 + t22$	$t55 = t46 + t54$	$y20 = x21 + t86$
$y26 = t4 + t23$	$y13 = x14 + t55$	$t88 = x13 + t10$
$t25 = x17 + t2$	$t57 = x5 + t47$	$t89 = y28 + t88$
$t26 = x8 + t25$	$t58 = x6 + t57$	$y4 = t84 + t89$
$y16 = t5 + t26$	$y29 = t54 + t58$	$t91 = x5 + y20$
$t28 = x25 + t25$	$t60 = x30 + y13$	$t92 = x4 + t91$
$y0 = y24 + t28$	$y5 = t58 + t60$	$y12 = t89 + t92$
$t30 = x10 + x26$	$t62 = x12 + x20$	
$t31 = t1 + t30$	$t63 = x4 + t0$	