VLSI Implementation and Its Optimisation for Digital Cryptosystems

Wong Ming Ming

School of Engineering, Computing and Science (SoECS)
Swinburne University of Technology
Sarawak Campus, Malaysia

Submitted for the degree of Doctor of Philosophy

2012
To my loving family and to all my friends.
Abstract

The advent of the modern computing and the widespread of digital communications fuel the need for secure and reliable methods in information and communication technologies. Professional bodies around the globe have therefore emerged with a common goal, that is to provide the highest level of information security in data communication protocols and standards. Information security requires the use of cryptography to provide authentication, confidentiality, integrity and non-repudiation as a means to protect information from unauthorised access or accidental disclosure. Subsequently, it is essential to ensure the effectiveness and efficiency of the selected cryptographic scheme.

In this doctoral work, two cryptographic schemes are investigated, which are namely the Advanced Encryption Standard (AES) and the elliptic curve cryptography (ECC). Naturally, this PhD thesis is organised into two parts according to our works in AES and ECC respectively. In Part I, we attempt to optimise the non-linear S-box defined over Galois field of $2^8$, $GF(2^8)$. The S-box is the major bottleneck to achieving small area, high throughput and lower power consumption in AES hardware implementation. More specifically, we take advantage of isomorphism to map the operation from $GF(2^8)$ to $GF(((2^2)^2)^2)$ using CFA.

After a sequence of algorithmic and architectural optimisations, we manage to derive an optimal construction for the S-box. The optimality that we seek for is one with minimum gate counts and the shortest critical path. Furthermore, in each composite field construction, there exists eight possible isomorphic mappings. Therefore, we design a new common subexpression elimination (CSE) algorithm to choose the isomorphic mapping that results in the lowest implementation cost. In the final stage, we exploit; one Algebraic Normal Form (ANF) compliant with a custom fine-grained pipelining scheme to achieve performance speed up and power reduction in our CFA AES S-box.

In the second part of this thesis, we focus on the optimisation approaches for elliptic curve (EC) hardware cryptosystem. Scalar multiplication, $kP$, the fundamental to all of the EC based cryptographic schemes, requires multiplicative inversions in affine coordinate system. Therefore, compact and efficient finite field multiplicative inverter
design has become more important than ever before. In this work, we present a new composite field composite multiplicative inverter of the form $GF(q^l)$, with $q = 2^{n.m}$ that is suitable for hardware realisation. Considering both the security aspect and the hardware cost required, we choose the composite field $GF(((2^2)^2)_41)$ for realising our EC cryptosystem. We employ a Fermat’s Little Theorem (FLT)-based inversion algorithm, the Itoh and Tsuji inversion (ITI) algorithm over optimal normal type II (ONB II) basis in our multiplicative inverter. Without the use of look-up tables (LUTs), the arithmetic in the subfield, $GF(2^4)$ will be performed in its isomorphic composite field, $GF((2^2)^2)$, which leads to combinatorial implementation.

To validate all the theoretical work presented in this thesis, we implement all our designs on field programmable gate arrays (FPGA) devices using the Altera Designer Suite. The experimental results are presented in each part along with in-depth discussions on the results obtained.
Acknowledgments

First of all, I praise God for the wisdom and the perseverance that He has bestowed upon me during this research project. This thesis appears in its current form due to the continual support and guidance from several people. I would therefore like to offer my sincere thanks to all of them.

I would like to express my gratitude to my supervisor Associate Professor Dennis Wong for his enthusiastic supervision, support and encouragement throughout my post graduate study. I really appreciate and admire his deep insight and patience in guiding my research in VLSI circuitry designs, specifically in the cryptography field. I would also like to thank my doctoral supervisory team, Professor Anatoli Vakhguelt and Dr. Patrick Then.

In addition to that, I gratefully acknowledge Drs. David Canright and Michael Rosing for sharing their extensive knowledge and prompting me the possible directions in the study of AES and ECC respectively. I have indeed learnt a lot from them. Many thanks go in particular to Professor Asoke Nandi (from University of Liverpool, UK) and Ismat Hijazin (from FEIS, Swinburne) for their valuable advices and discussion on the AES hardware implementation. I am also very much indebted to Jenny Choo for her constructive comments on this thesis. I am thankful that she took time out in the midst of her busy schedule to proofread this thesis.

Furthermore, I wish to express my gratitude to Swinburne University of Technology (Sarawak Campus) for my PhD studentship and the financial support in this research. Finally, many thanks and appreciation to my parents and my fellow friends for their support and encouragement when it was most required.
Declaration

I declare that this thesis contains no material that has been accepted for the award of any other degree or diploma and to the best of my knowledge contains no material previously published or written by another person except where due reference is made in the text of this thesis.

____________________
WONG MING MING
July, 2012
Publications Arising from this Doctoral Study


Other Publication

## Commonly Used Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>ANF</td>
<td>Algebraic Normal Form</td>
</tr>
<tr>
<td>CFA</td>
<td>Composite Field Arithmetic</td>
</tr>
<tr>
<td>CSE</td>
<td>Common Subexpression Elimination</td>
</tr>
<tr>
<td>ECC</td>
<td>Elliptic Curve Cryptography</td>
</tr>
<tr>
<td>ECDH</td>
<td>Elliptic Curve Diffie-Hellman</td>
</tr>
<tr>
<td>ECDLP</td>
<td>Elliptic Curve Discrete Logarithm Problem</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic Curve Digital Signal Algorithm</td>
</tr>
<tr>
<td>ECIES</td>
<td>Elliptic Curve Integrated Encryption Scheme</td>
</tr>
<tr>
<td>EEA</td>
<td>Extended Euclidean Algorithm</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DLP</td>
<td>Discrete Logarithm Problem</td>
</tr>
<tr>
<td>FLT</td>
<td>Fermat’s Little Theorem</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field</td>
</tr>
<tr>
<td>ITI</td>
<td>Itoh and Tsuji Inversion</td>
</tr>
<tr>
<td>KOA</td>
<td>Karatsuba-Ofman Algorithm</td>
</tr>
<tr>
<td>LE</td>
<td>Logic Element</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-up Table</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>ONB</td>
<td>Optimal Normal Basis</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very-Large-Scale Integration</td>
</tr>
</tbody>
</table>
Contents

1 Introduction ........................................... 1
   1.1 Information Security and Cryptography .......... 1
   1.2 Secret-Key (Symmetric) Cryptography .......... 2
   1.3 Public-Key (Asymmetric) Cryptography .......... 3
   1.4 Research Objectives and Contributions .......... 6
   1.5 Thesis Outline .................................. 7

2 Preliminaries ........................................ 10
   2.1 Preliminary in Galois Field ..................... 10
      2.1.1 Composite Field ........................... 11
      2.1.2 Optimal Normal Basis Representation ....... 11
      2.1.3 Addition .................................. 13
      2.1.4 Multiplication ............................ 13
      2.1.5 Multiplicative Inversion ................... 18
   2.2 Preliminary in VLSI Architectural Optimisation .... 22
      2.2.1 Pipelining ................................ 22
      2.2.2 Sub-Pipelining ............................ 23
      2.2.3 Loop Unrolling ............................ 24
      2.2.4 Tiling ................................... 25

Part I .................................................. 26

3 Advanced Encryption Standard ....................... 27
   3.1 Introduction to Advanced Encryption Standard .... 27
CONTENTS

3.2 Substitute Bytes / Inverse Substitute Bytes Transformation (S-box function) ........................................ 28
  3.2.1 Multiplicative Inversion ........................................ 30
  3.2.2 Affine Transformation ........................................ 32
3.3 Shift Row / Inverse Shift Row Transformation ..................... 32
3.4 Mix Columns / Inverse Mix Columns Transformation ............. 34
3.5 Add Round Key Transformation ................................... 36
3.6 AES Key Expansion .................................................. 36
3.7 Discussion ............................................................ 37

4 Related Works in Hardware Implementation of AES Cryptosystem 39
  4.1 Previous Works in AES cryptosystems .............................. 39
  4.2 Hardware Area Cost Reduction in AES S-box using Composite Field Arithmetic .............................. 40
  4.3 Very High Speed Design for AES S-box ............................ 42
  4.4 Low Power Consumption AES Design .............................. 43
  4.5 Summary Notes ...................................................... 45

5 AES S-box Reduction: Compact Multiplicative Inversion over $GF((2^2)^2)$ 47
  5.1 Compact AES S-box Derivation ....................................... 47
    5.1.1 Order of Field ................................................. 48
    5.1.2 Basis Representation .......................................... 48
    5.1.3 Irreducible Polynomial ....................................... 49
    5.1.4 Isomorphic Mapping .......................................... 49
  5.2 Multiplicative Inverse in S-Box Algorithm Using CFA ........... 50
    5.2.1 Multiplicative Inverse in Polynomial Basis .................. 51
    5.2.2 Multiplicative Inverse in Normal Basis ..................... 52
  5.3 Impact of Irreducible Polynomials in Multiplicative Inversion Architecture 54
  5.4 Optimisation of CFA Architectures ................................ 55
    5.4.1 Algorithmic Strength Reduction .............................. 55
    5.4.2 Substructure Sharing at Architectural Level ................. 61
  5.5 Discussion .......................................................... 62
6 AES S-box Reduction: Common Subexpression Elimination Algorithm 65
   6.1 Common Subexpression Elimination Algorithm .......................... 65
   6.2 Related Works in CSE ................................................. 67
   6.3 Proposed Common Subexpression Elimination Algorithm ............. 68
   6.4 Algorithm’s Performance and Evaluation .............................. 70
   6.5 Derivation of Optimum Isomorphism Mapping for Compact AES S-box 74
   6.6 Discussion .................................................................. 75

7 Speed Enhancement and Power Reduction in Hardware CFA AES S-box 79
   7.1 Hardware Implementation of Optimum CFA AES S-box ............... 79
   7.2 CFA AES S-box in Algebraic Normal Form ............................ 80
      7.2.1 ANF Representation in $GF((2^2)^2)$ Multiplicative Inversion for AES S-box .................................................. 81
   7.3 Fully Fine-grained Pipelined CFA AES S-box ........................... 83
   7.4 Implementation and Results .............................................. 83
   7.5 Discussion .................................................................. 85

8 Composite Field AES S-box Construction Using Fermat’s Little Theorem 89
   8.1 Fermat’s Little Theorem in Composite Field AES S-box .......... 89
   8.2 The Proposed AES S-boxes Using Fermat’s Little Theorem ....... 89
      8.2.1 Multiplicative Inversion in Field $GF((2^4)^2)$ .................. 90
      8.2.2 Multiplicative Inversion in Field $GF((2^2)^4)$ ................. 92
   8.3 Implementation and Results .............................................. 95
   8.4 Discussion .................................................................. 96

Part II 99

9 Elliptic Curve Cryptography 100
   9.1 Introduction to Elliptic Curve Cryptography ......................... 100
   9.2 Elliptic Curve ............................................................. 102
   9.3 Group Law .................................................................. 103
      9.3.1 Point Addition ....................................................... 105
      9.3.2 The Point at Infinity $O$ ......................................... 106
CONTENTS

9.4 Scalar Point Multiplication ............................................. 106
9.5 Elliptic Curve in Finite Fields ........................................... 108
  9.5.1 Binary Field .......................................................... 108
  9.5.2 Composite Binary Field ............................................. 108
  9.5.3 Prime Field .......................................................... 109
  9.5.4 Optimal Extension Field ............................................ 109
9.6 Affine and Projective Coordinate Systems ................................ 109
9.7 Elliptic Curve Discrete Logarithm Problem .............................. 112
9.8 Elliptic Curve Based Protocol ........................................... 113
9.9 Criteria in Elliptic Curve Cryptosystem Implementation ................. 114

10 Related Works in Elliptic Curve Cryptosystems .......................... 116
  10.1 Previous Works in Elliptic Curve Cryptosystem ...................... 116
  10.2 Software Elliptic Curve Cryptosystem Implementations .............. 116
    10.2.1 Binary Finite Field .............................................. 117
    10.2.2 Composite Field ................................................. 118
  10.3 Hardware Elliptic Curve Cryptosystem Implementations ............. 118
    10.3.1 Reconfigurable FPGA and Dedicated ASIC ....................... 119
    10.3.2 Generators of Elliptic Curve Processor ......................... 119
    10.3.3 Versatile Scalar Multiplier ...................................... 120
    10.3.4 General Purpose Processor ...................................... 120
  10.4 Finite Field Arithmetic in Hardware Cryptosystem .................. 121
    10.4.1 Binary Finite Field Operations .................................. 122
    10.4.2 Basis Representations .......................................... 123
    10.4.3 Serial and Parallel architecture .................................. 123
  10.5 Summaries Notes ...................................................... 124

11 Composite Binary Field for Elliptic Curve Hardware Cryptosystem ....... 126
  11.1 Optimisation in Elliptic Curve Hardware Cryptosystem ............... 126
  11.2 Field level Optimisation in Elliptic Curve Cryptography ............. 126
  11.3 Elliptic Curve Discrete Logarithm Problem in Composite Field ....... 127
  11.4 Complexity of the Composite Field .................................. 128

xii
11.5 Area minimisation in Scalar Multiplication ........................................ 129
11.6 Discussion ......................................................................................... 131

12 Efficient Composite Fields Multiplicative Inversion over \( GF((2^n)^m) \) .................. 132
   12.1 Itoh and Tsujii Inversion Algorithm for Multiplicative Inversion over \( GF((2^2)^2^{41}) \) ................................................................. 132
      12.1.1 Step 1: Exponentiation of \( A^{r^{-1}} \in GF(q') \) ...................... 133
      12.1.2 Step 2: Multiplication of \( A \) and \( A^{r^{-1}} \) that yield \( A' \in GF(q) \) . . . . 134
      12.1.3 Step 3: Multiplicative Inversion in \( GF(q) \) yields \( (A')^{-1} \) ....... 135
      12.1.4 Step 4: Multiplication of \( (A')^{-1} \cdot A^{r^{-1}} \) .......................... 135
   12.2 Parallel Sunar-Koc Multiplication in Composite Field ...................... 136
   12.3 Implementation and Results ............................................................ 139
   12.4 Discussion ..................................................................................... 144

13 Conclusion and Future Works ................................................................. 145
   13.1 Summary ....................................................................................... 145
      13.1.1 Part I: Advanced Encryption Standard ....................................... 145
      13.1.2 Part II: Elliptic Curve Hardware Cryptosystem ....................... 146
   13.2 Directions for Future Works ............................................................. 148
   13.3 Concluding Remarks ..................................................................... 149

Bibliography ............................................................................................ 166

Appendices .............................................................................................. 167

A List of possible coefficients in \( GF((2^2)^2^2) \) irreducible polynomials .......... 167

B CFA AES S-box over \( GF((2^2)^2^2) \) of other formations ......................... 170

C Gujardo and Paar Inversion Algorithm Over \( GF(2^{4})^{41} \) ....................... 176
## List of Figures

1.1 Secret-key (symmetric) cryptography .......................... 3
1.2 Encryption in public-key cryptography .......................... 4
1.3 Authentication in public-key cryptography .......................... 4

2.1 Pipelined architecture .............................................. 23
2.2 Sub-pipelining architecture ......................................... 24
2.3 Loop unrolling architecture ......................................... 25

3.1 AES encryption and decryption processes ................................. 29
3.2 Generation of AES S-box and inverse S-box .............................. 33
3.3 Shift Row step ....................................................... 33
3.4 Inverse Shift Row step ............................................... 33
3.5 Mix Columns and inverse Mix Columns .................................. 34
3.6 Add Round Key Transformation ...................................... 36
3.7 Key Expansion Transformation ....................................... 37

5.1 Case I: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in polynomial basis with irreducible polynomials’ norms equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$ ................................. 56

5.2 Case II: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in normal basis with irreducible polynomials’ norms equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$ ................................. 56
5.3 Case III: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in normal basis with combination of trace and norm equal unity in different irreducible polynomials. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$ ................................. 57

6.1 CSE in FIR filter design ........................................ 66
6.2 CSE for substructure sharing .................................. 67
6.3 Performance comparison between our CSE and previous work as summarised from Table 6.2. ................................. 73
6.4 Total gate count (XOR) for isomorphism mapping (isomorphic and inverse isomorphic + affine) for three cases, whereby $\beta_1$ to $\beta_8$ are the root and its conjugates. The minimum isomorphism mapping attainable for Case I requires 28 XORs, 31 XORs for Case II and 29 XORs for Case III. The critical path for each is 6 XOR gates. ................................. 75

7.1 ANF-CFA AES S-box with 7-stage fine-grained pipelining for (a) Case I, (b) Case II and (c) Case III ................................. 84

8.1 AES S-box over $GF((2^4)^2)$ and using generalised ITI algorithm ................................. 96
8.2 AES S-box over $GF((2^2)^4)$ and using generalised ITI algorithm ................................. 97

9.1 Elliptic curves .................................................... 103
9.2 Point addition in elliptic curve ................................ 105
9.3 Point doubling in elliptic curve ................................. 105
9.4 Addition of a point to its negative ............................ 107

12.1 Implementation of multiplicative inversion over $GF(2^4)$ using CFA in normal basis. (a) Multiplicative inversion over $GF((2^2)^2)$, (b) Multiplicative inversion over $GF(2^2)$, (c) Multiplication in $GF(2^2)$ ................................. 135
12.2 Multiplication in $GF((2^2)^2)$ ................................. 136
12.3 Hybrid multiplier for $GF(((2^2)^2)^{41})$ ....................... 140
12.4 Overview of our proposed compact and efficient multiplicative inversion over $GF(((2^2)^2)^{41})$. The $GF(((2^2)^2)^{41})$ multiplier in Step 1 is illustrated in Figure 12.3. ................................. 140
12.5 Multiplicative inversion and multiplication in $GF(2^4)$ using log and antilog conversions ........................................ 143

B.1 Case i: Implementation of multiplicative inversion of $GF(2^8)$ using CFA in polynomial basis with field polynomials’ traces equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$ ......................................................... 171

B.2 Case ii: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in normal basis with field polynomials’ traces equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$ ......................................................... 171
# List of Tables

1.1 Secret-Key (symmetric) cryptography and public-key (asymmetric) cryptography ........................................ 5

2.1 Optimal normal bases for $GF(2^m)$ .................................................. 12

2.2 Construction of $C_1$ ................................................................. 17

2.3 Construction of $D_1$ ................................................................. 17

2.4 Construction of $D_2$ ................................................................. 18

3.1 AES S-box ............................................................... 30

3.2 Inverse AES S-box ............................................................... 31

5.1 Multiplicative inverse for $GF(2^8)$ in polynomial basis .................................. 52

5.2 Multiplicative inverse for $GF(2^8)$ in normal basis .................................. 54

5.3 Optimisation in multiplicative inversion for Case I (polynomial basis and norm unity), with $\gamma = \Gamma_1z + \Gamma_0$ and $\tau = \Delta_1z + \Delta_0$. The pairs of $T$ and $\tau$ that result in minimal complexity in $\gamma\tau$ are determined. ........ 59

5.4 Optimisation in multiplicative inversion Case II (normal basis and norm unity), with $\gamma = \Gamma_1z + \Gamma_0$ and $\tau^2 = \Delta_1z + \Delta_0$. The pairs of $T$ and $\tau$ that result in minimal complexity in $\gamma\tau$ are determined. ........ 60

5.5 Optimisation in multiplicative inversion Case III (normal basis and combination of trace and norm unity), with $\gamma = \Gamma_1z + \Gamma_0$ and $\nu^2 = \Delta_1z + \Delta_0$. Pairs of $T$ and $\nu$ that result in minimal complexity in $\nu\gamma^2$ are determined. 61

5.6 Detailed description of CFA AES S-box for Case I, II and III with Case I using polynomial basis and Case II and III using normal basis ........... 64

6.1 Case study of optimisation in binary linear transformations using our CSE algorithm ........................................ 71
6.2 Performance analysis of the complexity (total logic operator and logic depth) reduction achievable by using our CSE algorithm and the one reported in [1]. Optimisations are performed in isomorphism function and its inverse together with affine transformation for 8 possible roots $\beta$ for a composite field AES S-box. ................................................. 72

6.3 Performance in terms of time elapsed (sec) deduced from our hybrid CSE technique and our CSE with sole greedy algorithm in pattern selection. The optimisation algorithms are performed in binary linear transformations in composite field AES S-box. The simulation is repeated for 10 round runtime. ................................................................. 72

6.4 Gate count and critical path of 3 novel (Cases I-III) CFA AES S-boxes. The best case of each constructions are chosen after algorithmic optimisation and their respective detail complexities are tabulated under column ‘Optimised Algorithm’. This is followed by architectural optimisation and the logic gates required are precisely analyzed and noted under column ‘Optimised Architecture’. The most optimum isomorphic mapping and inverse isomorphic mapping + affine transformation are deduced. Overall complexity after taking in optimisations and isomorphism into accounts is summarised in final column ......................... 77

6.5 Comparison of hardware complexities between previous works and our architectures ................................................................. 78

7.1 Multiplicative inversion for field element of $GF((2^2)^2)$ in ANF representation. Taking an element of four bits as $a \in GF((2^2)^2)$ as \{\textit{a}\textsubscript{3}, \textit{a}\textsubscript{2}, \textit{a}\textsubscript{1}, \textit{a}\textsubscript{0}\}, its multiplicative inversion is $a^{-1} = \{\textit{a}\textsubscript{3}^{-1}, \textit{a}\textsubscript{2}^{-1}, \textit{a}\textsubscript{1}^{-1}, \textit{a}\textsubscript{0}^{-1}\}$. ............................................. 87

7.2 Complexity in terms of total gate count and critical path for $GF((2^2)^2)$ multiplicative inversion in our works and the prior works. .................. 88

7.3 Area requirement, Timing analysis and Power consumption of FPGA implementation on Cyclone II EP2C5T144C6 device for our original CFA AES S-boxes and our seven stages pipelined ANF-CFA AES S-boxes. .. 88

8.1 Power-element relation for the primitive element $w$ in the field $GF(2^4)$ . 91

8.2 Comparison of hardware complexity (total gate count) between our Case III and FLT-based inversion approach. ................................. 96

8.3 Area requirement and Timing analysis of FPGA implementation of AES S-box in (i) $GF((2^4)^2)$ and (ii) $GF((2^2)^4)$ (iii) Case III ................................. 96
9.1 The simplified Weierstrass equations .......................... 104
9.2 Cost of point addition and doubling for different types of coordinates in elliptic curve $E : y^2 = x^3 + ax^2 + b$ (M-Multiplication, S-Squaring, I-Inversion) ......................................................... 111
9.3 Cost of point addition and doubling for different types of coordinates in elliptic curve $E : y^2 + xy = x^3 + ax^2 + b$ (M-Multiplication, V-Divisions) .............................. 112
9.4 Security analysis of public key cryptosystems .......................... 113
9.5 Criteria in EC cryptosystem implementation .......................... 115
12.1 Multiplicative inversion over $GF((2^2)^2)$ in ANF representation. Taking an element of four bits as $a \in GF((2^2)^2)$ as $\{a_3, a_2, a_1, a_0\}$ thus its multiplicative inversion is $a^{-1} = \{a_3^{-1}, a_2^{-1}, a_1^{-1}, a_0^{-1}\}$ ......................................................... 135
12.2 Permutation mapping of element in the basis $M$ to the basis $N$ (shifted canonical). Given value in the second column is $i$ of element $A_i$ (in $M$ basis) and element $A'_i$ (in $N$ basis) ......................................................... 137
12.3 Constructions of $C_1, D_1$ and $D_2$ ......................................................... 138
12.4 Hardware cost and clock rate performance for three hybrid serial-parallel multiplications ......................................................... 139
12.5 Complexity of $GF((2^4)^{41})$ multiplicative inversion using our proposed scheme and the work by Guajardo and Paar in [2] ......................................................... 141
12.6 Computational cost comparison of various multiplicative inverter architectures in hardware platform (CLB = Configurable Logic Block, LE = Logic Element) ......................................................... 142
12.7 Hardware analysis of FPGA implementation for $GF(((2^2)^2)^{41})$ multiplicative inverter with the $GF(2^4)/GF((2^2)^2)$ multiplier implemented using (i) combinatorial circuitry as proposed in our work (ii) log and antilog conversions and (iii) multiplication LUT. Device used for all of the implementations is EP3C120F780I7 Cyclone III ......................................................... 143
A.1 Choices of norm value ($\nu$) for $N = \{10\}_2$ and $N = \{11\}_2$ in polynomial basis representations ......................................................... 168
A.2 Choices of norm value ($\nu$) for $N = \{01\}_2$ and $N = \{10\}_2$ in normal basis representations ......................................................... 168
A.3 Choices of trace value $\tau$ for $T = \{10\}_2$ and $T = \{11\}_2$ in polynomial basis representations ......................................................... 169
LIST OF TABLES

A.4 Choices of trace value for $T = \{01\}_2$ and $T = \{10\}_2$ in normal basis representations .................................................. 169

B.1 Optimisation in multiplicative inversion using polynomial basis (trace unity), with $\gamma = \Gamma_1z + \Gamma_0$ and $\nu = \Delta_1z + \Delta_0$. Using Case i and polynomial basis representation, the possible pairs of $N$ and $\nu$ that result in minimal complexity in the $\nu\gamma^2$ operation are determined. ........................................ 173

B.2 Optimisation in multiplicative inversion using normal basis (trace unity), with $\gamma = \Gamma_1z + \Gamma_0$ and $\nu = \Delta_1z + \Delta_0$. Using Case ii and normal basis representation, the possible pairs of $N$ and $\nu$ that result in minimal complexity in operation $\nu\gamma^2$ are determined. .................................................. 175

B.3 The best cases of CFA from Case i and Case ii ........................................ 175
Chapter 1

Introduction

1.1 Information Security and Cryptography

Ever since the Internet has emerged as an essential platform of mass communication, the security aspect of digital communication has become a tremendously important issue to deal with. As a result, cryptography has become an essential component in the modern information systems.

Cryptography is the science of information security. It started as an ancient art, where the information was transformed in a way that it was unintelligible to anyone, except for those who possessed the special knowledge, which is the key. For instance, the Egyptian scribe used the non-standard hieroglyphs in the inscription back in around 1900 B.C. Some also claimed that cryptography appeared sometime after writing was invented, with its applications ranging from the diplomatic missives to war-time battle plans.

Following the advent of the computers and the widespread of digital communications, cryptography has arisen as an important agent in providing a secure channel for exchanging information. Unlike the ancient cryptography, the modern cryptography is complex in nature and it is based on solid scientific approaches. The cryptographic algorithms are designed with the adoption of the computational hardness assumptions. As a result, the algorithms are hard to break by an adversary. Though such systems are not completely unbreakable in theory, it is difficult and physically infeasible to do so by any current means. Therefore, such cryptographic schemes are classified as computationally secure.

From another point of view, cryptography is also a study of mathematical techniques
that provides the accountability, accuracy and confidentiality in information exchange [3]. Cryptography itself is not a means of information security, but it provides a set of techniques that is capable of fulfilling the security services listed in the following [4],

- **Authentication**, is a service related to identification.
- **Privacy/Confidentiality**, is a service that keeps the information content from the unauthorised party.
- **Integrity**, is a service that ensures the received message has not been altered in any way from the original.
- **Non-repudiation**, is a service which prevents an entity from denying previous commitments or actions.

Therefore, the fundamental goal of cryptography is to effectively address these four areas in practical usage. In general, there are three types of cryptographic schemes typically used to accomplish the above mentioned services. These are namely the **secret-key (symmetric)** cryptography, **public-key (asymmetric)** cryptography and **hash functions**. In this research, we are specifically interested in the secret-key and public-key cryptography, which will be introduced in Section 1.2 and Section 1.3 respectively. In both cases, the initial data (original text) is denoted as the plaintext and the encrypted data is referred as the ciphertext.

1.2 Secret-Key (Symmetric) Cryptography

Secret-key cryptography uses a single key for both encryption and decryption [5]. The sender uses the key to encrypt the plaintext (following some set of rules) and sends the ciphertext to the receiver. The receiver applies the same key (and some set of rules) to retrieve the original plaintext. This operation is as depicted in Figure 1.1. As only one single key is used for both functions, secret-key cryptography is also termed as the symmetric-key cryptography. In this form of cryptography, the reveal of the key is restricted to the sender and the receiver only, which is in fact a secret between them [6].

The study of secret-key cryptography relates mainly to the study of **stream cipher** or **block cipher** and to their applications. Stream ciphers operate on a single bit, byte or computer word at a time. It requires a feedback mechanism such that the key produced
is constantly changing. On the other hand, block cipher encrypts one block of data at a time using the same key on each block. Furthermore, in block cipher, the same plaintext block will always be encrypted into the same ciphertext if the same key were used. On the contrary, the same plaintext with the same key will be encrypted into different ciphertext in the stream cipher.

In December 2001, the Advanced Encryption Standard (AES) is chosen by the NIST (National Institute of Standards and Technology) to replace the Data Encryption Standard (DES) as the new cryptography standard in US government. AES adopted a secret-key block cipher scheme called the Rijndael, which is designed by Joan Daemen and Vincent Rijmen. The algorithm works on the combinations of 128, 192 and 256 bits key length and blocks of 128 bits. More information about AES and its efficient implementation in hardware platform is presented in Part I of this thesis.

1.3 Public-Key (Asymmetric) Cryptography

Unlike all the other cryptographic systems which are based on the elementary tools using substitution and permutations, public-key cryptography is based on mathematical functions. The modern public-key cryptography was first introduced by Martin Hellman and Whitfield Diffie from Stanford University in 1976 [6]. They had successfully described a two-key cryptosystem in which two parties could engage in a secure communications over a non-secure communication channel without distributing the secret key. In this system, two different but mathematically related keys are used, i.e. a public-key and private-key. The sender uses a public-key to encrypt the plaintext and the receiver will use his private key to decrypt the ciphertext. While the public key maybe advertised widely as the owner wants, the private key must be kept secret (not
revealed to any other party). It is important to note that it does not matter which key is applied first, but both keys are required for the process to work.

Note that, cryptography is mainly dependent upon the existence of the one-way function of which the mathematical functions are easy to compute, but their inverse function is relatively much more difficult to compute [6]. These are the significant features in public-key cryptography that make it computationally secure. Broadly, we can classify the use of public-key cryptography into three categories [5]:

- **Encryption/Decryption**: The sender encrypts a plaintext using the recipient’s public key (see Figure 1.2).

- **Digital Signature**: The sender digitally “signs” (using a cryptographic algorithms) a message with its private key (see Figure 1.3).

- **Key Exchange**: Two sides cooperate to exchange a session key which involves private key(s) from one or both parties.

Here, we would like to clarify two common misconceptions concerning the public-key cryptography [5]. First, the public-key cryptography was claimed to be more secure
Table 1.1: Secret-Key (symmetric) cryptography and public-key (asymmetric) cryptography

<table>
<thead>
<tr>
<th>Work Needed</th>
<th>Secret-Key Encryption</th>
<th>Public-key Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>The same algorithm and the same key are used for encryption and decryption.</td>
<td>One algorithm is used for encryption and decryption with a pair of key is used separately.</td>
<td></td>
</tr>
<tr>
<td>The sender and receiver must share the algorithm and the key.</td>
<td>The sender and receiver must each have one of the matched pairs of keys.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security Needed</th>
<th>Secret-Key Encryption</th>
<th>Public-key Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>The key must be kept secret.</td>
<td>One of the two keys must be kept secret.</td>
<td></td>
</tr>
<tr>
<td>It must be impossible or at least impractical to decipher a ciphertext if no other information is available.</td>
<td>It must be impossible or at least impractical to decipher a ciphertext if no other information is available.</td>
<td></td>
</tr>
<tr>
<td>Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.</td>
<td>Knowledge of the algorithm plus the public key and samples of ciphertext must be insufficient to determine the other key.</td>
<td></td>
</tr>
</tbody>
</table>

compared to the secret-key cryptography. In fact, the security of any cryptography scheme depends on the length of the key and the computational work involved in breaking the cipher. Therefore, there is no justification whether the secret-key or the public-key that is more superior from the view of cryptanalysis. Second, public-key cryptography is a general-purpose technique, thus making the secret-key cryptography obsolete. There is no foreseeable likelihood that secret-key cryptography will be replaced or abandoned as the current public-key cryptography tends to impose higher computational overhead. Table 1.1 summarises the important aspects of the secret-key (symmetric) cryptography and the public-key (asymmetric) cryptography.

Soon after Diffie and Hellman revolutionised the field of cryptography with the invention of public-key cryptography, RSA, was introduced as the first usable public-key cryptosystem. The cryptosystem is based on the difficulty of factoring very large numbers and remains as the widely used public-key cryptosystems in the world. However, in the later date, research works in the field of number theory have worked out efficient integer factorisation and this leads to the rise of the new types of public-key cryptosystems. The most important competitors to RSA are the cryptography schemes that are based on the Discrete Logarithm (DL) problem. In 1985, a variant of DL problem was proposed by Miller and Koblitz independently, based on the group of points of an elliptic curve over a finite field.

In this research study, we are interested in optimising the hardware implementation of the public-key cryptography developed using the group of points defined over elliptic curve, which is presented in Part II of this thesis.
1.4 Research Objectives and Contributions

Cryptography plays a vital role in the wide variety of applications that require secure communication mechanisms. As such, several cryptographic algorithms have been proposed to secure the transactions between the communicating parties in different protocols. Apart from the security requirements, implementation of the cryptographic systems presents specific challenges. The challenges are: First, as many applications require (near) real-time interaction communication, the overheads needed for encrypting and decrypting the data must be minimal. Second, many cryptographic applications are realised on embedded platforms, which are inherently constrained in terms of area and power consumption. Therefore, the cryptographic operation ought to be designed in a way that is compliant with the resource-limited platform. However, in most hardware implementations, there exists a definite trade-off between hardware performance and its requirement. As a result, meeting these two challenges at the same time is a difficult task.

In this study, we investigate ways to meet these challenges through designing specific hardware architectures to perform critical cryptographic operations. We are particularly interested in hardware cryptosystem implementation of AES encryption and elliptic curve cryptography (ECC). Both schemes have become the most commonly deployed cryptographic algorithms in the field of secret-key and public-key cryptosystems respectively. AES is considered as one of the strongest and the most secure cryptographic algorithm as it was designated as a cryptography standard in US government. Meanwhile, ECC has outperformed other public-key cryptography for it can be built in a way that it requires a shorter operand length while achieving the same security level.

The main objective of this work is to investigate new optimisation methodologies to reduce the hardware resources (area and power consumption) and to enhance the performance of these cryptosystems in hardware. The requirement of these hardware cryptosystems varies according to the nature of the applications. While some applications require high throughputs such as in e-commerce servers, others may require medium throughput range with a smaller area of implementation as in mobile phones and other electronic devices. Several others require very low power consumption and low hardware area cost such as in active RFID realms.

In the first part of this thesis, we present a detailed study on composite field construction over the finite field $GF(((2^2)^2)^2)$ for the S-box function, the most resource consum-
ing operation in AES. We derive a pure combinatorial construction that is optimally balanced in both of the area of implementation and critical path. In order to achieve the optimal construction for AES S-box, we will employ a series of optimisations. These include the algorithmic strength reduction, substructure sharing in architectural level, common subexpression elimination (CSE), conversion of the complicated circuitry into several logical expressions and strategic fine-grained pipelining.

In the second part of this thesis, we shift our focus to the ECC. We perform area reduction optimisation specifically in the scalar multiplication operation, \( kP \), which is the most crucial operation in EC hardware cryptosystem. Taking both the security aspect and the hardware cost required into account, we present a new efficient composite field multiplicative inverter of the form \( GF(q) \) with \( q = (2^n)^m \). As opposed to previous works, we propose to use a three-level composite field, which leads to combinatorial implementation without resorting to the conventional look-up table (LUT) approach. For verification and validation purposes, both our works in AES and ECC will be digitally synthesized and validated in Altera's Field Programmable Gates Arrays (FPGA) devices.

1.5 Thesis Outline

This thesis consists of two major parts. Part I reports the optimisation in hardware AES cipher, whereas Part II contains the work done in EC hardware cryptosystem. Before diving into the details, we first present the required preliminary knowledge in Chapter 2. We will cover some of the introductory mathematical theory behind the composite field arithmetic (CFA) and the basic rules that govern the basic arithmetic such as the addition, multiplication and multiplicative inversion. In addition to that, we also explain the commonly deployed speed optimisation schemes in hardware circuitries, namely the pipelining, sub-pipelining, loop unrolling and tilling.

Part I

Chapter 3 describes the overall structure and the working principles of AES. Detailed discussion on the four main transformations in AES are presented. Chapter 4 summarises the previous works on AES implementation. The reported studies are classified into three broad categories; the compact hardware using CFA, the high speed AES
Chapter 1: Introduction

S-box realisation and low power consumption AES S-box.

Chapter 5 reviews the derivation of compact multiplicative inversion over $\text{GF}((2^2)^2)$. In this chapter, we present series of algorithmic optimisation to simplify the multiplicative inversion circuitry and thus lead to the area cost reduction. The previously reported CFA AES S-box, of which will be used for benchmarking purpose can be found in Appendix B.

The area reduction methodologies for the required isomorphism function and its inverse, followed by the affine transformation are explained in Chapter 6. We propose a new CSE algorithm that combines greedy algorithm and exhaustive search to perform substructure sharing in the binary linear transformations in AES S-box.

Next, the optimisation in speed enhancement and power reduction are reported in Chapter 7. Two major strategies are employed here, the Algebraic Normal Form (ANF) and fine-grained pipelining. The best construction of our AES S-box along with its implementation in hardware is reviewed in this chapter.

To further prove the efficacy of our AES S-box in hardware implementation, our best architecture is benchmarked with the AES S-box constructed using the FLT-based inversion in Chapter 8. Two FLT-based AES S-boxes over the field $\text{GF}((2^4)^2)$ and $\text{GF}((2^2)^4)$ are derived and the comparison in terms of area requirement and performance are analyzed and justified.

Part II

Chapter 9 provides an elementary introduction to the ECC. It covers some of the related mathematical theory and the methods to perform the point addition and doubling of elliptic curve points. Next, brief introduction to the elliptic curve in different common finite fields is presented as well. We also include the Elliptic Curve Discrete Logarithm Problem (ECDLP) which determines the security level of an EC cryptosystem. Finally, we summarise the important criteria in developing an EC cryptosystem towards the end of the chapter.

Chapter 10 summarises the previous works on EC cryptosystems and other related studies which contribute to the ECC research developments. These include the studies of the software and the hardware implementations of EC cryptosystems, along with the development in finite field architectures.
Chapter 11 discusses the feasibility of field level optimisation to promote area cost reduction in EC hardware cryptosystems in general. Here, we discuss the study on CFA in ECC, considering both the resultant complexity and the security provided. As an end result, we have chosen to work on composite field of $\mathbb{GF}((2^2)^{41})$ in ONBII basis for EC hardware cryptosystem.

Next, we attempt to reduce the most resource consuming operation in scalar multiplication, the multiplicative inversion by proposing a small area combinatorial architecture which will be explained in Chapter 12. The proposed approach which employs FLT in the multiplicative inversion and a hybrid parallel Sunar-Koc multiplication is realised in hardware implementation. For verification and validation purposes, our architecture is benchmarked with the previous studies and the results are discussed towards the end of the chapter. The advantages of using combinatorial circuitry over the LUTs, in terms of hardware cost, performance and power consumption is also analyzed.

Finally, Chapter 13 concludes this work with a short summary and suggestions for future research in this area.
Chapter 2

Preliminaries

2.1 Preliminary in Galois Field

A set of integers modulo $q$, where $q$ is a prime, forms a field which is denoted as $\mathbb{Z}_q$ [7]. Meanwhile, a set of all the polynomials with their coefficients from $\mathbb{Z}_q$ in the indeterminate $x$, are defined as $\mathbb{Z}_q[x]$. Therefore, the ring of polynomials modulo $q$ can be constructed by combining the set of $\mathbb{Z}_q[x]$ with additions and multiplications, followed by the coefficients reduction modulo $q$.

In the case where $f(x) \in \mathbb{Z}_q[x]$ and $\text{deg}(f(x)) = m \geq 1$, we can construct the ring of polynomials by reduction modulo $f(x)$ [8]. This ring is denoted as $\mathbb{Z}_q[x]/f(x)$. All the elements in the ring are polynomials in $\mathbb{Z}_q[x]$ with the degree less or equal to $m - 1$. The field operations such as the additions and multiplications can be defined in a similar manner, followed by reduction modulo $f(x)$.

If $f(x)$ happens to be an irreducible polynomials, $\mathbb{Z}_q[x]/f(x)$ is a finite field, which can be written as $GF(q^m) \cong \mathbb{Z}_q[x]/f(x)$. In addition to that, it is stated that an irreducible polynomial of degree $m$ over $GF(q)$ exists for any finite field $GF(q)$. Therefore, we will be using the notation $GF(q^m)$ to indicate the finite fields hereafter.

It is worth a noting that $GF(q^m)$ is also known as an extension field of $GF(q)$, of order $q^m$. With this, the field $GF(q)$ is referred as the subfield of $GF(q^m)$. The elements in $GF(q^m)$ can be represented as a polynomial $A(x) = a_{m-1}x^{m-1} + \cdots + a_0$, with the coefficients $a_i \in GF(q)$ and $i = 0, 1, 2, \ldots, m - 1$. Thus, $A(x)$ is said to be “a polynomial over $GF(q)$”. These $q^m$ polynomials in $GF(q^m)$ form the residue classes modulo $f(x)$ of all the polynomials over $GF(q)$.

In this chapter, we present brief discussion of the binary composite field and the im-
portant finite field operations such as the addition, multiplication and multiplicative inversion.

2.1.1 Composite Field

As stated in Section 2.1, the order of the finite field, \( q \) may be a prime number. Apart from that, with \( q \) is either power of prime or \( q = p^m \) such that \( p \) is prime would also form a valid finite field. In addition to that, it is possible to extend a finite field \( GF(q^n) \) to form a composite field, denoted as \( GF((q^n)^m) \). Composite field is highly favoured and has been widely deployed in several applications as it promotes simple computations and allows faster arithmetic architectures. Furthermore, binary finite field, the field of the order 2 \( (q = 2) \), is useful in representing the information for the practical applications in both hardware and software.

Therefore, we have a special type of finite fields, the binary composite field. As described in [9], any two pairs \( \{GF(2^n), Q(y) = y^n + \sum_{i=0}^{n-1} q_i y^i\} \) and \( \{GF((2^n)^m), P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i\} \) form a composite field if \( GF(2^n) \) is constructed as an extension field of \( GF(2) \) by \( Q(y) \) and \( GF((2^n)^m) \) is constructed as extension field of \( GF(2^n) \) by \( P(x) \), where \( Q(y) \) and \( P(x) \) are irreducible polynomials over \( GF(2) \). Take note that the composite field \( GF((2^n)^m) \) and \( GF(2^k) \) are said to be isomorphic to each other if \( k = n \cdot m \) [9]. For any two isomorphic fields, the algorithmic complexity with respect to the field operation (such as addition and multiplication) may still differ. Complexity of a composite field is dependent on the field construction factors of which will be discussed in Section 5.1, Chapter 5.

2.1.2 Optimal Normal Basis Representation

The optimal normal bases (ONBI and ONBII) are the special type of normal basis representation (refer Section 5.1.2 in Chapter 5) which are introduced following the development of the Massey-Omura multiplier in [10]. There are 117 and 319 \( m \) values in the range \( m \in [2, 2001] \), such that the field \( GF(2^m) \) has ONBI and ONBII respectively [11]. This also means that the ONBII is three times more likely to occur within the range and therefore it is a useful choice for implementation.

In terms of hardware implementation, these bases manage to further reduce the complexity of the complicated normal basis multipliers. Elements in the composite field \( GF((2^n)^m) \) can be represented using optimal normal bases when there exist ONBI or
ONBII for $GF(2^m)$ [12]. Therefore, a linearly independent set that forms ONB for binary field $GF(2^m)$ also forms an optimal basis for composite field $GF((2^n)^m)$ when $gcd(n,m) = 1$. Thus, $A \in GF((2^n)^m)$ can be written as $A = \sum_{i=0}^{m-1} A_i \beta^i$, with $A_i \in GF(2^n)$.

It turns out that an ONBI over field $GF(2^m)$ can be constructed if the following two conditions holds [13]:

- $p = m + 1$ must be a prime.
- 2 is a primitive root modulo $p$.

The second rule means that 2 raised to any power in the range of $[0, 1, \ldots, m - 1]$ modulo $p$ must result in a unique integer in the range of $[1, 2, \ldots, m]$. On the other hand, an ONBII over $GF(2^m)$ can be created if $p = 2m + 1$ is a prime and if either of the following two conditions also holds [13]:

- 2 is a primitive root modulo $p$.
- $p \equiv 3 \mod 4$ and 2 generates the quadratic residues in $Z_{2m+1}$.

The first condition means that when we take $2^k \mod p$ for $k \in [0, 1, 2, \ldots, 2m - 1]$, we will get every value in the range $[1, \ldots, 2m]$ back. The second condition has two parts. The first part simply means that the last two bits are set in the binary representation of the prime $p$. The second part means that if $2^k \mod p$ does not generate every element in the range $[0, \ldots, 2m]$, we can at least take the square root mod $p$ of $2^k$.

Table 2.1 shows a few values which there exists an ONB in $GF(2^m)$ (taken from Table 5.1 in [11]). The first column is the field size $m$ and the second column shows the type of ONB.

<table>
<thead>
<tr>
<th>m</th>
<th>Type</th>
<th>m</th>
<th>Type</th>
<th>m</th>
<th>Type</th>
<th>m</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>I</td>
<td>23</td>
<td>II</td>
<td>33</td>
<td>II</td>
<td>50</td>
<td>II</td>
</tr>
<tr>
<td>11</td>
<td>II</td>
<td>26</td>
<td>II</td>
<td>35</td>
<td>II</td>
<td>51</td>
<td>II</td>
</tr>
<tr>
<td>12</td>
<td>I</td>
<td>28</td>
<td>I</td>
<td>36</td>
<td>I</td>
<td>52</td>
<td>I</td>
</tr>
<tr>
<td>14</td>
<td>II</td>
<td>29</td>
<td>II</td>
<td>39</td>
<td>II</td>
<td>53</td>
<td>II</td>
</tr>
<tr>
<td>18</td>
<td>I&amp;II</td>
<td>30</td>
<td>II</td>
<td>41</td>
<td>II</td>
<td>58</td>
<td>I</td>
</tr>
<tr>
<td>74</td>
<td>II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Optimal normal bases for $GF(2^m)$
2.1.3 Addition

Addition is a simple operation in finite field and it is performed by adding the polynomial’s coefficients of the corresponding powers. In the field $GF(2^n)$, the resulting coefficients are reduced modulo 2 of which can be accomplished with a bitwise XOR function. Furthermore, addition of the polynomial-based composite field can be easily done by XOR from each of the subfield elements. Therefore, an addition in $GF((2^n)^m)$ requires $n.m$ number of XOR gates. Note that, subtraction of the polynomials is performed exactly the same as addition in finite fields.

2.1.4 Multiplication

Multiplication in the finite field is performed by multiplying both the operand polynomials and reduced by the field’s irreducible polynomial. This is to ensure that the resultant product of multiplication has a degree less than the order of the field. Unlike addition, the finite field multiplication cannot be accomplished in a single operation. The following shows an example of multiplication in the field $GF(2^8)$ over the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$.

\[
\begin{align*}
(x^6 + x^5 + x^4 + x)(x^4 + x^3) &= (x^{10} + x^9 + x^8 + x^5) + (x^9 + x^8 + x^7 + x^4) \\
&= (x^{10} + x^7 + x^5 + x^4) \text{ modulo } (x^8 + x^4 + x^3 + x + 1) \\
&= x^7 + x^6 + x^4 + x^3 + x^2
\end{align*}
\]

Finite field multiplication is the second most complicated operation in the finite field arithmetic after the multiplicative inversion. Therefore, it is essential to choose the efficient multiplication algorithm to promote performance speed up in the cryptographic application. We have surveyed the available work on finite field multiplier designs. The two most efficient multiplication algorithms for both the polynomial basis and the ONBII representation respectively are reported in the following subsections.
Karatsuba-Ofman Polynomial Multiplication

Field multiplication of \( C(x) = A(x) \cdot B(x) \), where \( A, B, C \in GF(2^m) \) and are using polynomial basis representation can be performed by,

\[
C(x) = A(x) \cdot B(x) \mod P(x) \quad (2.1.1)
\]

Operation in (2.1.1) can be computed in two steps:

1. Ordinary polynomial multiplication.
2. Reduction modulo the irreducible polynomial.

In this case, Karatsuba-Ofman’s algorithm (KOA) [14] can be applied to the first step in (2.1.1), the polynomial multiplication. The algorithm performs multiplications with a reduced amount of coefficient multiplication but at the cost of an increased number of coefficient additions [15]. KOA for polynomial in \( GF(2^m) \) is based on a divide-and-conquer strategy [16]. It replaces a multiplication by three multiplications of a half-length operands. A straightforward application of the KOA requires \( \log_2(m+1) \) iteration steps for polynomials in \( GF(2^m) \).

Let the two multiplication operands denoted as \( A = \{a_0, a_1, \ldots, a_{m-1}\} \) and \( B = \{b_0, b_1, \ldots, b_{m-1}\} \). Both the operands can be decomposed into two equal-sized parts denoted as \( A_1 \) and \( A_0 \), and \( B_1 \) and \( B_0 \). Each part will represent the \( m/2 \) higher and lower order bits of \( A \) and \( B \) such as described in the following [16]:

\[
A(x) = \sum_{i=0}^{m-1} a_i x^i \\
= \sum_{i=0}^{m/2-1} a_i x^i + \sum_{i=m/2}^{m-1} a_i x^i \\
= x^{m/2} \sum_{i=0}^{m/2-1} a_{i+m/2} x^i + \sum_{i=0}^{m/2-1} a_i x^i \\
= x^{m/2} A_1 + A_0 \quad (2.1.2)
\]
\[ B(x) = \sum_{i=0}^{m-1} b_i x^i = x^{\frac{m}{2}} B_1 + B_0 \] (2.1.3)

Therefore, the product \( C(x) = A(x) \cdot B(x) \) can be computed as follows,

\[ A(x) \cdot B(x) = A_0 B_0 + A_1 B_1 x^m + (A_1 B_0 + A_0 B_1) x^{m/2} = A_0 B_0 + A_1 B_1 x^m + (A_0 B_0 + A_1 B_1 + (A_1 + A_0)(B_1 + B_0)) x^{m/2} \] (2.1.4)

which results in,

\[ C'(x) = A(x) \cdot B(x) \]

\[ A(x) \cdot B(x) = C_0 + C_1 x^{m/2} + C_2 x^m \] (2.1.5)

The first step in field multiplication in (2.1.1) can be accomplished through (2.1.5). As a result, an element \( C'(x) \) of \( 2m - 1 \) bits will be produced. Therefore, the second step in (2.1.1) is performed such that \( C'(x) \) is reduced modulo the irreducible polynomial \( C(x) \equiv C'(x) \mod P(x) \). This reduction operation can be simplified when \( P(x) \) of the AOP form and trinomial form are used. For instance, the modulo reduction would require \((m + k - 1)/2\) XOR gates only for irreducible trinomial of the form \( P(x) = x^m + x^k + 1 \).

**Sunar-Koc Multiplication**

The parallel Sunar-Koc multipliers designed specifically for ONB representation was introduced in [17], which requires 25% less XOR gates than the conventional Massey-Omura multipliers [10]. In a nutshell, the algorithm for deriving a Sunar-Koc multiplier involves the following procedures:

1. Convert the elements represented in the basis \( M \) to the basis \( N \) (shifted canonical)
using permutation.

2. Multiply the elements in basis $N$.

3. Convert the result back to the basis $M$ through inverse permutation.

The first and the third steps can be easily implemented through simple routing, without using any gates. The second step is where the multiplication is performed in basis $N$.

Let $A, B \in GF(2^m)$, we need to first convert the operands to basis $N$;

$$A = \sum_{i=1}^{m} a_i \beta_i = \sum_{i=1}^{m} a_i (\gamma^i + \gamma^{-i}),$$

$$B = \sum_{i=1}^{m} b_i \beta_i = \sum_{i=1}^{m} b_i (\gamma^i + \gamma^{-i})$$  

(2.1.6)

Next, the product $C = A \cdot B$ is computed as follows:

$$C = \left( \sum_{i=1}^{m} a_i (\gamma^i + \gamma^{-i}) \right) \cdot \left( \sum_{j=1}^{m} b_j (\gamma^j + \gamma^{-j}) \right)$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)}) + \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)})$$

$$= C_1 + C_2$$  

(2.1.7)

For $C_1$, the exponent $(i - j)$ of $\gamma$ is already within the proper range, i.e. means $m \leq (i - j) \leq m$ for all $i, j \in [1, m]$. When $i = j$, we obtain $\gamma^{i-j} + \gamma^{-(i-j)} = \gamma^0 + \gamma^0 = 0$.

Therefore, we write $C_1$ as follows:

$$C_1 = \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)})$$

$$= \sum_{1 \leq i,j \leq m, i \neq j} a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)})$$  

(2.1.8)

Table 2.2 shows the elements of summation in $C_1$ being arranged in terms of the order of the basis elements. Furthermore, the term $C_2$ can be rewritten into the following:
The term $D$ terms in the appropriate order will result in the product $C$. 

Rewrite them into the required range, $D$ elements of proper range. Table 2.3 shows the construction of summation $D$.

The multiplication algorithm in basis $N$ constructs $C_1$, $D_1$ and $D_2$. The sums of the terms in the appropriate order will result in the product $C = C_1 + D_1 + D_2$. 

\[ C_2 = \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{i+j} + \gamma^{-i+j}) \]

\[ = \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{i+j} + \gamma^{-i+j}) + \sum_{i=1}^{m} \sum_{j=m+1}^{m} a_i b_j (\gamma^{i+j} + \gamma^{-i+j}) \]

\[ = D_1 + D_2 \hspace{1cm} (2.1.9) \]

The term $D_1$ in (2.1.9) has the exponents of the basis elements $\gamma^{i+j} + \gamma^{-i+j}$ in the proper range. Table 2.3 shows the construction of summation $D_1$. However, the basis elements of $D_2$ are totally out of range. Thus, we can use the identity $\gamma^{2m+1} = 1$ and rewrite them into the required range,

\[ D_2 = \sum_{i=1}^{m} \sum_{j=m+1}^{m} a_i b_j (\gamma^{i+j} + \gamma^{-i+j}) \]

\[ = \sum_{i=1}^{m} \sum_{j=m+1}^{m} a_i b_j (\gamma^{2m+1-i+j} + \gamma^{-(2m+1-i+j)}) \hspace{1cm} (2.1.10) \]

with the construction of $D_2$ as listed in Table 2.4.

The multiplication algorithm in basis $N$ constructs $C_1$, $D_1$ and $D_2$. The sums of the terms in the appropriate order will result in the product $C = C_1 + D_1 + D_2$. 

17
### 2.1.5 Multiplicative Inversion

Among all of the customary field arithmetic, multiplicative inversion (inversion over finite field) is the most complicated and time consuming arithmetic in the finite field $GF(2^m)$. There are two families of algorithms that are commonly adopted in the multiplicative inversion computation. These are namely the Extended Euclidean Algorithm (EEA) and the Fermat’s Little Theorem (FLT). We shall now present these algorithms in the following subsections.

#### Extended Euclidean Algorithm-based Inversion

Let $a$ and $b$ the the non-zero binary polynomials. The greatest common divisor ($gcd$) of $a$ and $b$ is the polynomial $d$ of the highest degree that divides both of them [18].

When the numbers involved are small, the $gcd$ can be easily calculated by factoring both numbers and then the highest common factor is determined. However, for large numbers (which frequently occur in cryptography schemes), the Euclidean algorithm appears to be a more efficient algorithm for $gcd$ computation.

The algorithm for computing $gcd(a, b)$ is based on the following polynomial analogue in Theorem 2.1.1 [19].

**Theorem 2.1.1.** Let $a$ and $b$ be binary polynomials. Then $gcd(a, b) = gcd(b - ca, a)$ for all binary polynomials $c$.

The $gcd$ computation of binary polynomials $a$ and $b$ where $deg(b) \leq deg(a)$, is by dividing $b$ with $a$ to obtain a quotient $q$ and a remainder $r$ that satisfies $b = qa + r$ and $deg(r) < deg(a)$ [19]. Based on Theorem 2.1.1, we obtain $gcd(a, b) = gcd(r, a)$. The problem of finding $gcd$ of two given numbers is now reduced to finding the $gcd$ of two
Chapter 2: Preliminaries

smaller numbers. The process is applied recursively until one of the argument turns zero, and the result is then immediately obtained since \( \gcd(0,d) = d \). As each iteration preserves the \( \gcd \) of the previous step, it turns out that the final \( \gcd \) is actually finding the \( \gcd \) of the original numbers such as the following:

\[
\gcd(a, b) = \cdots = \gcd(0, d) = d
\]  
(2.1.11)

However, the main application of the Euclidean algorithm is not finding the \( \gcd \) only. The extension of the algorithm allows the computation of modular multiplicative inverse \cite{18}, which is a major contribution to the cryptography algorithms. Apart from computing the \( \gcd \), the EEA computes a linear combination of the form,

\[
\gcd(a, b) = s \cdot a + t \cdot b
\]  
(2.1.12)

where \( s \) and \( t \) are integer coefficients. The main idea of the algorithm is similar to the process in (2.1.11). Having the current remainder been expressed as \( d_i = s_i a + t_i b \), we would get the last iteration as,

\[
d_i = \gcd(a, b) = s_i \cdot a + t_i \cdot b = s \cdot a + t \cdot b
\]  
(2.1.13)

Note that the inverse exists when \( \gcd(a, b) = 1 \). Through EEA, we can perform the inversion as follows:

\[
\gcd(a, b) = s \cdot a + t \cdot b \\
= 1 \\
s \cdot 0 + t \cdot b \equiv 1 \mod a \\
t \cdot b \equiv 1 \mod a
\]  
(2.1.14)

The (2.1.14) is the exact definition of the inverse of \( b \). In other word, \( t \) is the inverse of \( b; t = b^{-1} \mod a \).
EEA can also be used for computing the multiplicative inversion in Galois field. The algorithm can be used analogously with polynomials instead of integers [18]. Assume that we want to determine the multiplicative inverse of $A \in GF(2^m)$ and given the irreducible polynomial is $P(x)$. The auxiliary polynomials $s(x)$ and $t(x)$, as well as the $gcd(P(x), A(x))$ resultant from EEA computation, are shown in the following.

$$s(x)P(x) + t(x)A(x) = gcd(P(x), A(x))$$

$$= 1$$

$$s(x)0 + t(x)A(x) \equiv 1 \mod P(x)$$

$$t(x) \equiv A^{-1}(x) \mod P(x) \quad (2.1.15)$$

where the polynomial $t(x)$ is equal to the multiplicative inverse of $A(x)$. Algorithm 1 summarised the inversion algorithm using EEA [20].

**Algorithm 1 GF($2^m$) Multiplicative inversion using EEA**

**INPUT:** Non-zero $a \in GF(2^m)$

**OUTPUT:** $c = a^{-1} \mod p(x)$

1. $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow p$
2. WHILE $\text{deg}(u) \neq 0$ DO
3. $j \leftarrow \text{deg}(u) - \text{deg}(v)$
4. IF $j < 0$ THEN
5. $u \leftrightarrow v, b \leftrightarrow c, j \leftarrow -j$
6. ENDIF
7. $u \leftarrow u + x^jv, b \leftarrow b + x^jc$
8. END WHILE
9. RETURN $c$

**Fermat’s Little Theorem-based Inversion**

According to the FLT, with $a$ being an integer and $p$ a prime number, we can obtain $a^p \equiv a \pmod{p}$. The same is applicable to the finite field as well. Let $A \in GF(2^m)$ and $P(x)$ is an irreducible polynomial defined over $GF(2^m)$, the theorem can be stated in the form $a^{2^m-1} \equiv 1 \mod P(x)$. The equation can be redefined as $A.A^{2^m-2} \equiv 1 \mod P(x)$, which is an exact definition for inversion [18]. Consequently, we can invert a finite field element modulo an irreducible polynomial easily:
where, the arithmetic involved exponentiation operations. Therefore, the FLT-based inversion is also known as the exponentiation-based inversion. Although performing the exponentiation in (2.1.16) would be slower than using the EEA, there exists some situations where there would be advantageous to use the FLT-based inversions. One example would be the applications like smart cards or other devices which have a hardware accelerator for fast exponentiation [18]. Generally, in cases where very high speed multiplications and squarings can be performed without much effort required, FLT-based inversion may result in faster multiplicative inversion and with lesser effort compared to the EEA based techniques [20].

Furthermore, Itoh and Tsujii [21] proposed an addition chain which allows this particular exponentiation in (2.1.16) to be computed out in at most \(2\left\lfloor \log_2 (m - 1) \right\rfloor\) multiplications and \(m - 1\) squarings. When the arithmetic is performed in normal basis representation, the squaring operation is of negligible cost for it is just a cyclic rotation of the vector representing a field elements. At the same time, they also proposed an exponentiation-based inversion algorithm, namely the Itoh-Tsujii inversion (ITI) algorithm.

The ITI algorithm was presented in three different algorithms [21]. The first two algorithms describe addition chains for exponentiation-based inversion on the binary finite field of \(GF(2^m)\). As our work is based on composite field arithmetic (CFA), we are only interested in the third algorithm which describes a method that is based on subfield multiplicative inversion. More specifically, the third ITI algorithm reduces the multiplicative inversion in the composite field of \(GF(q^m)\) to the multiplicative inversion in \(GF(q)\), where \(q = 2^n\) as described in Theorem 2.1.2.

**Theorem 2.1.2.** [21] Let \(A \in GF(q^m)\) with \(q = (2^n), A \neq 0\) and \(r = (q^m - 1)/(q - 1)\). The inverse of an element \(A\) can be computed as,

\[
A^{-1} = (A^r)^{-1}.A^{r-1}
\]

with \(A^r \in GF(q)\) and \(A^{-1}, A^{r-1} \in GF((q^m)).\)
Computing the multiplicative inverse through Theorem 2.1.2 requires four steps such as listed in Algorithm 2.

**Algorithm 2** Itoh-Tsujii Inversion Algorithm in $GF(q^m)$ [22]

**INPUT:** Non-zero $A \in GF(2^m)$

**OUTPUT:** $C = A^{-1} \mod P(x)$

1. $B \leftarrow A'^{-1}$ (using an addition chain)
2. $b \leftarrow BA = A'^{-1}.A = A' \in GF(q)$
3. $b \leftarrow b^{-1} = (A')^{-1}$
4. $C \leftarrow bB = (A')^{-1}.A'^{-1} = A^{-1}$
5. RETURN $C$

The second and the fourth steps are rather trivial since both $A'$ and $(A')^{-1}$ are elements in the subfield $GF(q)$. Both of the steps can be compared with a complexity that is very much less than the single extension field multiplication. The complexity of third step, subfield multiplicative inversion, depends on the type and order of the subfield $GF(q)$ [22]. Thus, the first step which involved exponentiation to the $(r - 1)th$ power in the extension field $GF(q^m)$ is the most complicated step of all.

### 2.2 Preliminary in VLSI Architectural Optimisation

Optimisation in the architectural level is commonly deployed to achieve clock rate improvement in VLSI circuitries. These optimisations are namely the **pipelining**, **sub-pipelining**, **loop unrolling** (also known as **unfolding**) and **tiling**. They are categorised as the architectural optimisation as the modification are performed at the architectural level. Here, the total number of round unit involved to perform one complete operation is denoted as $N_r$. In an ordinary case, a complete process would take up a total number of $N_r$ clock cycles.

#### 2.2.1 Pipelining

Pipelined architecture [23, 24], such as depicted in Figure 2.1, increases the speed of operations by processing multiple blocks of data simultaneously. This is realised by inserting rows of registers in between two combinatorial logic blocks. Thus, parts of the logic between two consecutive registers form a pipeline stage. Each pipeline stage will be considered as a round unit. After every clock cycle, the partially processed data will proceed to the next stage and at the same time, its place is taken by the subsequent data block.
The number of round units in each loop, \( k \), is usually chosen as a divisor of \( N_r \). When \( k \) is at its maximum value, \( N_r \), it implies that the architecture is fully pipelined. In fully pipelined architecture, the partially processed block will be fed to the first round when the process reaches to the \( kth \) round. This procedure will repeat until all the \( N_r \) rounds are performed on the particular block. The pipeline reaches its full depth when the first block has reached to the \( kth \) stage. At this stage, \( k \) block of data will be processed simultaneously for every \( N_r \) cycles. Therefore, the hardware area being imposed is proportional to the number of pipeline stages, \( k \).

![Figure 2.1: Pipelined architecture](image)

### 2.2.2 Sub-Pipelining

Sub-pipelining [23], also known as fine-grained pipelining, is as depicted in Figure 2.2. It is similar to pipelining except that the rows of registers are not only inserted between the each round unit, but also inside the round unit. For instance, each round can be divided into \( r \) stages, thus with \( k \) round fully sub-pipelining, the architecture is able to achieve an approximately \( r \) times the speed of a \( k \) round fully pipelined architecture, but with the cost of a slight additional area (due to extra registers and control logic).

Note that, the minimum attainable clock period is determined by the indivisible com-
binatorial element with the longest delay. Therefore, the idea of dividing each round into stages may not induce effective speed improvement. When more data blocks are processed, the required time to process one block will increase proportionally. However, the increase in hardware cost is proportional with $k$, and it is not much affected by $r$. Therefore, by increasing the number of inner round stages, this will introduce additional registers but it is small in comparison to overall implementation.

![Figure 2.2: Sub-pipelining architecture](image)

### 2.2.3 Loop Unrolling

A loop unrolled (unfolded) architecture [23, 25, 26] (refer Figure 2.3) processes one data block at a time, but at the same time multiple rounds are performed in each clock cycle. The unrolling factor, $k$, is chosen as a divisor of $N_r$ and the number of cycles to process one block of data is $\frac{N_r}{r}$.

The area required is proportional to number of rounds in each loop, $k$. Compared to $k$-stage pipelined architecture, the resultant speedup is much lower but with the same area imposed. Therefore, this approach does not necessarily increase the throughput but it allows further pipelining to be performed [27].
2.2.4 Tiling

Besides pipelining (inner and outer round) and loop unrolling, tiling [28] is yet another technique which exploits intrinsic parallelism and is found mostly in the technical literature. Tiling transformation replicates the instances of a block in order to promote parallelism. In the past, researchers believed that tiling a serial block was better than unrolling in terms of throughput achievement. They justified that,

1. Unrolled architecture is faster than the iterative one but the difference is often negligible.
2. The computational time saved through unrolling comes with an increase in implementation area.

In recent years, researchers agree with this analysis but it has been emphasised that tiling has different flaws with respect to unrolling [27],

1. Unrolling enables further pipelining, thus this is justified with the extra area being imposed.
2. When a design is tiled, additional logic with corresponding to area and time is usually involved. However, unrolled and pipelined architecture does not require any other additional logic other than pipeline register.

Thus, tiling is rarely exploited as an approach to promote speed optimisations.
Part I

Advanced Encryption Standard (AES)

This part of thesis addresses the studies on the optimisation methods for VLSI AES cipher
Chapter 3

Advanced Encryption Standard

3.1 Introduction to Advanced Encryption Standard

In 1997, the National Institute of Standards and Technology (NIST) of USA announced a formal Call for Algorithms to search for a replacement to the old Data Encryption Standard (DES). Fifteen candidate algorithms were shortlisted in the first round (August 1998) and among them, five finalists were chosen in the second round (April 1999). These finalist were namely the: MARS (multiplication, addition, rotation and substitution) from IBM; RC6 designed by Ronald Rivest; Rijndael from a Belgian team, John Daemen and Vincent Rijmen; Serpent developed jointly by a team from England, Israel and Norway; and Twofish, developed by Bruce Schneier [3]. At the end, Rijndael was selected as the new Advanced Encryption Standard (AES) in October 2000.

The Rijndael AES is a symmetric block cipher with a constant block size of 128 bits (16 bytes) that supports the key lengths of 128 bits, 192 bits and 256 bits. It performs encryption and decryption processes on an iterative basis where each iterative step is known as an operation round. The number of the operation round used is determined by the key length (10, 12 and 14 rounds for key lengths 128, 192 and 256 bits respectively). The AES algorithm operates on a changeable square array of dimension $4 \times 4$, known as the state array [3]. The encryption process begins with setting the original input data (plaintext) in the state array. Along the process, the state transforms and produces the encrypted data (ciphertext). Meanwhile in the decryption process, the state starts with the ciphertext and performs changes until the original plaintext is recovered.

One of the noteworthy features of this AES structure is that it is not a Feistel structure
In the classical Feistel structure, half of the data block is used to modify the other half of the data block and then both are swapped. On the other hand, AES cipher processes the entire block in parallel. There are four major operation stages in AES; one permutation and three substitutions in both the encryption and decryption processes. Each encryption processes SubByte transformation (S-box), Shift Rows, Mix Columns and Add Round Key. The decryption which performs the reverse, consists of SubByte transformation using the inverse S-box, inverse Shift Rows, inverse Mix Columns and Add Round Key. Among all the transformations, only the Add Round Key makes use of the round key generated from the key expansion unit. This particular unit produces 176, 208 or 240 bytes of round keys based on the key length used. For both the encryption and the decryption processes with 128-bit key, the cipher begins with the Add Round Key stage, followed by nine rounds of operations (which includes all the four stages) and ends with the tenth round (final round) that consists of three stages only (without the Mix Columns / inverse Mix Columns transformation). To summarise, the structure for the encryption and the decryption processes are depicted in Figure 3.1.

All of the four stages used in the cipher will be explained in the following subsections (cf. Section 3.2 to Section 3.6). For each stage, we will discuss the forward (encryption) algorithm, the inverse (decryption) algorithm and the rationale of the operation involved.

### 3.2 Substitute Bytes / Inverse Substitute Bytes Transformation (S-box function)

The SubByte (Substitution Byte) or better known as the S-box function, is a transformation that maps each byte in the state to a new byte. This transformation was designed to have a low correlation between the input and output bits and the output is described using a non-linear function of the input. Therefore, it is resistant towards the existing cryptanalytic attacks.

The transformation can be visualised as a substitution table (S-box), which contains a permutation of all the 256 possible 8-bit values where the mapping strategy can be performed using Table 3.1. The leftmost 4-bit of the byte is used as a row index, while the rightmost 4-bit is used as a column value [3]. These row and column values will serve as the indexes in the S-box in order to select the unique 8-bit output.
Chapter 3: Advanced Encryption Standard

**Figure 3.1:** AES encryption and decryption processes

(a) Encryption

- Plaintext
  - Add Round Key
  - Substitute Byte
  - Shift Rows
  - Mix Columns
  - Add Round Key
  - ... (repeated 9 times)
  - Substitute Byte
  - Shift Rows
  - Mix Columns
  - Add Round Key
- Ciphertext

(b) Decryption

- Ciphertext
  - Add Round Key
  - Inv Substitute Byte
  - Inv Shift Rows
  - Inv Mix Columns
  - Add Round Key
  - ... (repeated 9 times)
  - Inv Shift Rows
  - Inv Substitute Byte
  - Add Round Key
- Plaintext
The creation of the S-box is based on the composition of two major transformations over the Galois field $GF(2^8)$ [29]. First, every byte is placed by its multiplicative inverse (the reciprocal), which will be discussed in Section 3.2.1. This is the most complicated operation in finite field arithmetics, thus also the most resource consuming operation in AES cipher. Second, an affine transformation which is relatively easier, is performed onto the resultant from the prior step such as described in Section 3.2.2.

On the other hand, the inverse of the SubByte transformation also requires a similar mapping strategy. The difference is that a different substitution table is used and it is based on the inverse property of the SubByte’s S-box. The inverse of the AES S-box is tabulated in Table 3.2.

### 3.2.1 Multiplicative Inversion

The first step in deriving the S-box is finding the multiplicative inverse over Galois field of $GF(2^8)$ defined over the irreducible polynomials $p(x)$, (3.2.1) [30].

$$p(x) = x^8 + x^4 + x^3 + x + 1 \quad (3.2.1)$$
Table 3.2: Inverse AES S-box

<table>
<thead>
<tr>
<th></th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
<th>x8</th>
<th>x9</th>
<th>xA</th>
<th>xB</th>
<th>xC</th>
<th>xD</th>
<th>xE</th>
<th>xF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>52</td>
<td>09</td>
<td>6A</td>
<td>D5</td>
<td>30</td>
<td>36</td>
<td>A5</td>
<td>38</td>
<td>BF</td>
<td>40</td>
<td>A3</td>
<td>9E</td>
<td>81</td>
<td>F3</td>
<td>D7</td>
<td>FB</td>
</tr>
<tr>
<td>1x</td>
<td>7C</td>
<td>E3</td>
<td>39</td>
<td>82</td>
<td>9B</td>
<td>2F</td>
<td>FF</td>
<td>87</td>
<td>34</td>
<td>8E</td>
<td>43</td>
<td>44</td>
<td>C4</td>
<td>DE</td>
<td>E9</td>
<td>CB</td>
</tr>
<tr>
<td>2x</td>
<td>54</td>
<td>7B</td>
<td>94</td>
<td>32</td>
<td>A6</td>
<td>C2</td>
<td>23</td>
<td>3D</td>
<td>EE</td>
<td>4C</td>
<td>95</td>
<td>0B</td>
<td>42</td>
<td>FA</td>
<td>C3</td>
<td>4E</td>
</tr>
<tr>
<td>3x</td>
<td>08</td>
<td>2E</td>
<td>A1</td>
<td>66</td>
<td>28</td>
<td>D9</td>
<td>24</td>
<td>B2</td>
<td>76</td>
<td>5B</td>
<td>A2</td>
<td>49</td>
<td>6D</td>
<td>8B</td>
<td>D1</td>
<td>25</td>
</tr>
<tr>
<td>4x</td>
<td>72</td>
<td>F8</td>
<td>F6</td>
<td>64</td>
<td>86</td>
<td>68</td>
<td>98</td>
<td>16</td>
<td>D4</td>
<td>A4</td>
<td>5C</td>
<td>CC</td>
<td>5D</td>
<td>65</td>
<td>B6</td>
<td>92</td>
</tr>
<tr>
<td>5x</td>
<td>6C</td>
<td>70</td>
<td>48</td>
<td>50</td>
<td>FD</td>
<td>ED</td>
<td>B9</td>
<td>DA</td>
<td>5E</td>
<td>15</td>
<td>46</td>
<td>57</td>
<td>A7</td>
<td>BD</td>
<td>9D</td>
<td>84</td>
</tr>
<tr>
<td>6x</td>
<td>90</td>
<td>D8</td>
<td>AB</td>
<td>00</td>
<td>8C</td>
<td>BC</td>
<td>D3</td>
<td>0A</td>
<td>F7</td>
<td>E4</td>
<td>58</td>
<td>05</td>
<td>B8</td>
<td>B3</td>
<td>45</td>
<td>06</td>
</tr>
<tr>
<td>7x</td>
<td>D0</td>
<td>2C</td>
<td>1E</td>
<td>8F</td>
<td>CA</td>
<td>3F</td>
<td>0F</td>
<td>02</td>
<td>C1</td>
<td>AF</td>
<td>BD</td>
<td>03</td>
<td>01</td>
<td>13</td>
<td>8A</td>
<td>6B</td>
</tr>
<tr>
<td>8x</td>
<td>3A</td>
<td>91</td>
<td>11</td>
<td>41</td>
<td>4F</td>
<td>67</td>
<td>DC</td>
<td>EA</td>
<td>97</td>
<td>F2</td>
<td>CF</td>
<td>CE</td>
<td>F0</td>
<td>B4</td>
<td>E6</td>
<td>73</td>
</tr>
<tr>
<td>9x</td>
<td>96</td>
<td>AC</td>
<td>74</td>
<td>22</td>
<td>E7</td>
<td>AD</td>
<td>35</td>
<td>85</td>
<td>E2</td>
<td>F9</td>
<td>37</td>
<td>E8</td>
<td>1C</td>
<td>75</td>
<td>DF</td>
<td>6E</td>
</tr>
<tr>
<td>Ax</td>
<td>47</td>
<td>F1</td>
<td>1A</td>
<td>71</td>
<td>1D</td>
<td>29</td>
<td>C5</td>
<td>89</td>
<td>6F</td>
<td>B7</td>
<td>62</td>
<td>0E</td>
<td>AA</td>
<td>18</td>
<td>BE</td>
<td>1B</td>
</tr>
<tr>
<td>Bx</td>
<td>FC</td>
<td>56</td>
<td>3E</td>
<td>4B</td>
<td>C6</td>
<td>D2</td>
<td>79</td>
<td>20</td>
<td>9A</td>
<td>DB</td>
<td>C0</td>
<td>FE</td>
<td>78</td>
<td>CD</td>
<td>5A</td>
<td>F4</td>
</tr>
<tr>
<td>Cx</td>
<td>1F</td>
<td>DD</td>
<td>A8</td>
<td>33</td>
<td>88</td>
<td>07</td>
<td>C7</td>
<td>31</td>
<td>B1</td>
<td>12</td>
<td>10</td>
<td>59</td>
<td>27</td>
<td>80</td>
<td>EC</td>
<td>5F</td>
</tr>
<tr>
<td>Dx</td>
<td>60</td>
<td>51</td>
<td>7F</td>
<td>A9</td>
<td>19</td>
<td>B5</td>
<td>4A</td>
<td>0D</td>
<td>2D</td>
<td>E5</td>
<td>7A</td>
<td>9F</td>
<td>93</td>
<td>C9</td>
<td>9C</td>
<td>EF</td>
</tr>
<tr>
<td>Ex</td>
<td>A0</td>
<td>E0</td>
<td>3B</td>
<td>4D</td>
<td>AE</td>
<td>2A</td>
<td>F5</td>
<td>B0</td>
<td>C8</td>
<td>EB</td>
<td>BB</td>
<td>3C</td>
<td>83</td>
<td>53</td>
<td>99</td>
<td>61</td>
</tr>
<tr>
<td>Fx</td>
<td>17</td>
<td>2B</td>
<td>04</td>
<td>7E</td>
<td>BA</td>
<td>77</td>
<td>D6</td>
<td>26</td>
<td>E1</td>
<td>69</td>
<td>14</td>
<td>63</td>
<td>55</td>
<td>21</td>
<td>0C</td>
<td>7D</td>
</tr>
</tbody>
</table>

One of the possible ways in deriving the multiplicative inversion over Galois field is through the use of EEA algorithm (refer Section 2.1.5 in Chapter 2). EEA stated that for every polynomial $a(x)$, there exists two polynomials $b(x)$ and $c(x)$ such that:

$$a(x) \cdot b(x) + p(x) \cdot c(x) = 1 \tag{3.2.2}$$

and given that,

$$a(x) \cdot b(x) \mod p(x) = 1 \tag{3.2.3}$$

Therefore, the multiplicative inversion of $a(x) \in GF(2^8)$, $a(x)^{-1}$, can be derived as shown in (3.2.4).

$$a(x)^{-1} = b(x) \mod p(x) \tag{3.2.4}$$
3.2.2 Affine Transformation

The second step in deriving the S-box is performing the affine transformation. Both affine transformation and its inverse consist of two parts, multiplication part and addition part. The 8-bit data (output from the multiplicative inversion) will be first multiplied with a constant matrix and followed by the addition with a constant polynomial vector. The affine transformations for both the encryption and the decryption processes are listed in (3.2.5) and (3.2.6) respectively. In the encryption process, affine transformation is applied after the completion of the multiplicative inverse calculation in SubBytes transformation. On the contrary, it would be applied first in the inverse SubBytes transformation for the decryption process. Both the S-box and and inverse S-box transformations are summarised in Figure 3.2.

\[
AT(q) = A_1 \times q + C_1 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1
\end{bmatrix} \times \begin{bmatrix}
q_7 \\
q_6 \\
q_5 \\
q_4 \\
q_3 \\
q_2 \\
q_1 \\
q_0
\end{bmatrix} + \begin{bmatrix}
0 \\
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1
\end{bmatrix} \tag{3.2.5}
\]

\[
AT(q) = A_2 \times q + C_2 = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{bmatrix} \times \begin{bmatrix}
q_7 \\
q_6 \\
q_5 \\
q_4 \\
q_3 \\
q_2 \\
q_1 \\
q_0
\end{bmatrix} + \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
0 \\
1
\end{bmatrix} \tag{3.2.6}
\]

3.3 Shift Row / Inverse Shift Row Transformation

Shift Row transformation is changing the sequence of the element bytes within the state’s row. The transformation is made by shifting the state’s row with a certain offset.
The first row is shifted with offset zero, in other words it remains intact. The second row is cyclically left shifted with one offset. Likewise, the elements in the third row and fourth row (last row) of the state are cyclically left shifted with two offsets and three offsets respectively. This Shift Row transformation is as illustrated in Figure 3.3.

The inverse Shift Row transformation in the decryption process is the reverse of the Shift Row operation where the cyclical shift is performed in the opposite direction. The first row is remained unaltered. The offsets used during the operation are three, two and one for the respective second, third and fourth row. The inverse Shift Row is shown in Figure 3.4.
3.4 Mix Columns / Inverse Mix Columns Transformation

Mix Columns transformation performs multiplication on each of the state’s columns. Each column (four bytes) is represented in the polynomial form over the field $GF(2^8)$ (see (3.4.1)). The polynomial consists of four operands representing the column elements from the previous state and will be used to derive the new state elements through the transformation as depicted in Figure 3.5.

![Figure 3.5: Mix Columns and inverse Mix Columns](image)

The transformation involves multiplication of $b(x)$, (3.4.1) and a fixed polynomial $c(x)$, (3.4.2) modulo $k(x)$, (3.4.3) which can be defined as (3.4.4). The operation in (3.4.4) can be written in polynomials as shown in (3.4.5) [31].

$$b(x) = S_3 x^3 + S_2 x^2 + S_1 x + S_0$$  
(3.4.1)

$$c(x) = \{03\} x^3 + \{01\} x^2 + \{01\} x + \{02\}$$  
(3.4.2)

$$k(x) = x^4 + 1$$  
(3.4.3)
Chapter 3: Advanced Encryption Standard

\[
\begin{bmatrix}
S'_{0,C} \\
S'_{1,C} \\
S'_{2,C} \\
S'_{3,C}
\end{bmatrix} =
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 02 & 01 & 01
\end{bmatrix}
\begin{bmatrix}
S_{0,C} \\
S_{1,C} \\
S_{2,C} \\
S_{3,C}
\end{bmatrix}
\]

\[(3.4.4)\]

\[
S'_{0,C} = \{02\} S_{0,C} + \{03\} S_{1,C} + S_{2,C} + S_{3,C}
\]

\[
S'_{1,C} = \{02\} S_{1,C} + \{03\} S_{2,C} + S_{3,C} + S_{0,C}
\]

\[
S'_{2,C} = \{02\} S_{2,C} + \{03\} S_{3,C} + S_{0,C} + S_{1,C}
\]

\[
S'_{3,C} = \{02\} S_{3,C} + \{03\} S_{0,C} + S_{1,C} + S_{2,C}
\]

\[(3.4.5)\]

On the other hand, the inverse of Mix Column step is obtained by multiplying the \(b(x)\) and the \(c(x)^{-1}\) in (3.4.6) modulo \(k(x)\). This is represented in the matrix multiplication as stated in (3.4.7) or can be written in the polynomial form as stated (3.4.8).

\[
c(x)^{-1} = \{0B\} x^3 + \{0D\} x^2 + \{09\} x + \{0E\}
\]

\[(3.4.6)\]

\[
\begin{bmatrix}
S'_{0,C} \\
S'_{1,C} \\
S'_{2,C} \\
S'_{3,C}
\end{bmatrix} =
\begin{bmatrix}
0E & 0B & 0D & 09 \\
09 & 0E & 0B & 0D \\
0D & 09 & 0E & 0B \\
0B & 0D & 09 & 0E
\end{bmatrix}
\begin{bmatrix}
S_{0,C} \\
S_{1,C} \\
S_{2,C} \\
S_{3,C}
\end{bmatrix}
\]

\[(3.4.7)\]

\[
S_{0,C} = \{0E\} S_{0,C} + \{0B\} S_{1,C} + \{0D\} S_{2,C} + \{09\} S_{3,C}
\]

\[
S_{1,C} = \{0E\} S_{1,C} + \{0B\} S_{2,C} + \{0D\} S_{3,C} + \{09\} S_{0,C}
\]

\[
S_{2,C} = \{0E\} S_{2,C} + \{0B\} S_{3,C} + \{0D\} S_{0,C} + \{09\} S_{1,C}
\]

\[
S_{3,C} = \{0E\} S_{3,C} + \{0B\} S_{0,C} + \{0D\} S_{1,C} + \{09\} S_{2,C}
\]

\[(3.4.8)\]

Unlike the previous stages (see Section 3.2 and 3.3), both the Mix Columns and its inverse steps will not be performed in the last round of encryption or decryption processes.
3.5 Add Round Key Transformation

Add Round Key transformation can be viewed as a columnwise operation where the 128-bits (four bytes) of state are bitwise XORed with the 128-bits (4 bytes) round key [29]. The inverse Add Round Key transformation is identical to the forward Add Round Key because the XOR operation is same as its own inverse. The Add Round Key (same as its inverse transformation) is as depicted in Figure 3.6. These round keys are generated using key expansion unit which will be explained in the following subsection.

![Figure 3.6: Add Round Key Transformation](image)

3.6 AES Key Expansion

The key expansion unit generates the required round keys in the AES cipher from the original input key. The original input key is used as the initial round key in the encryption process and is used in the last group in the key expansion unit for decryption process. By using 128-bits key size, 10 groups of 16 bytes round key will be generated. The routine of generating a new round key is described in the following steps:

1. The last four bytes of the previous round key (or the initial key) are cyclically left shifted with one offset.

2. SubBytes transformation is applied on the new byte sequence.

3. The first byte of the sequence is bitwise XORed with a variable called $Rcon$, which represents the round number.

4. The result of the previous step is bitwise XORed with the first byte of the previous round key and this results in the first byte of the new round key.

5. The second, third and fourth bytes of the previous round key are now bitwise...
XORed with the first byte of the new round key. Therefore, the new second, third and fourth byte of the new round key are produced.

![Key Expansion Transformation](image)

**Figure 3.7: Key Expansion Transformation**

This routine will be repeated until all of the 10 round keys with a total of 176 bytes are obtained. The summary of the Key Expansion is shown in Figure 3.7.

### 3.7 Discussion

Since the announcement of the AES algorithm in 2001, Rijndael AES block cipher is widely available in various cryptographic applications. AES with 128 bits key size is deployed mostly in commercial applications. Meanwhile, AES of 192-bits and 256-bits, which are able to provide higher security level, are mainly used in military applications. However, the implementations of the AES of higher key size would consume larger hardware area and longer processing time.

In addition to that, AES algorithm has broad applications on various architecture platforms with different design constraints to fulfill different implementation requirements such as small area, high speed and low power consumption. In general, the study of optimisation in AES can be classified into two major groups, namely the algorithmic optimisation and architectural optimisation.

Algorithmic optimisation exploits the algorithmic strength inside the round unit to simplify the arithmetic within the AES algorithm. This will lead to area cost reduct-
tion in hardware implementation. Architectural optimisation can be utilised to perform enhancement in the architectural level which will in turn promote performance speedup in the cryptosystem. On the other hand, both algorithmic and architectural optimisations are required to deduce a low power consumption system.

After the detailed discussion of the operation in the AES cipher, we are ready to proceed to the previous studies in relation to the performance and implementation cost for AES implementation in the next chapter. Based on the studies, no further optimisations is required for ShiftRows/invShiftRows and AddRoundKey transformations. For the former, the transformation performs only shifting which does not require any logic gate. While for the latter, only one step of XOR operation is needed. Therefore, major attention is paid to study the SubBytes/invSubBytes transformation, the only non-linear transformation in AES.

Note that each stage in the cipher is designed to be easily reversible. Thus, the inverse function of the SubByte, Shift Row and Mix Column stages, which are used in the decryption algorithm, share the similar computational features with their original counterpart. Therefore, our work are explained in consideration of the encryption process only.
CHAPTER 4

Related Works in Hardware Implementation of AES Cryptosystem

4.1 Previous Works in AES cryptosystems

AES S-box (the SubBytes transformation) is the most costly and performance critical module block in AES hardware circuitry. It also takes up the most power consumption in AES circuit, owing to the excessive occurrence of dynamic hazards. Thus, the non-linear S-box is the major bottleneck in achieving small area, high speed and low power AES hardware implementations [32]. AES S-box hardware implementation can be categorised into three major paradigms.

The first paradigm, which is highly favoured in the software based AES implementation, is by pre-storing all the S-box elements in memory form such as ROM and BRAMs (FPGA block of RAM). As a result, the S-box function is implemented as a form of LUT, which is relatively easy and straightforward. However, the use of the memory blocks in hardware realisation will impose a large silicon area of consumptions. On top of that, it also introduces an unbreakable timing delay which will predominate the minimum clock rate attainable by the final cryptosystem. Note that a BRAM is devised as one memory block, therefore pipelining is not applicable within the block. Subsequently, this will limit the number of sub-pipelining stages in the design.

In order to avoid this delay while saving more silicon areas, one can opt for the second paradigm. This is a pure combinatorial approach where only the combinatorial logic
components are used for the final computation. In this approach, the multiplicative inverse of elements in $GF(2^8)$ is computed prior to using the CFA with respect to the polynomial $p(x)$ (refer (3.2.1)), followed by affine transformation. The detailed description of this approach will be presented in Chapter 5. Through this approach, designer can further improve the circuit’s clock rate by inserting standard pipeline registers at the appropriate points of the system.

The third paradigm is to construct single circuit where the input output relation is equivalent to a S-box. For instance, Betroni proposed Decoder-Switch-Encoder (DSE) S-box to achieve low power solution in AES S-box [33]. This method involves a permutation block (the switch unit) which takes 256 one-hot coded decoder outputs and having them connected to the inputs of encoder. The connection is designed in a way that it performs the functionality of a S-box. This method is less commonly practiced in comparison to the prior approaches.

In general, an ideal AES S-box ought to have a high throughput and requires minimal amount of hardware resources. Unfortunately, speed and area requirement are two aims that are often contradictory in most VLSI designs. Subsequently, the trends in designing AES S-box is to either focusing solely on high speed implementation; or ensuring minimal circuitries required during the implementation. Lately, AES circuits have been made available in some of the resource constrained and battery-powered embedded devices. Thus, the demands for a lower power consumption have increased accordingly.

In this chapter we discuss the various hardware designs for AES S-box algorithm which were reported in the previous studies. The presented implementation can be classified into three main categories; the compact hardware AES using CFA (Section 4.2), the high speed AES S-box realisation (Section 4.3) and low power consumption AES S-box (Section 4.4).

### 4.2 Hardware Area Cost Reduction in AES S-box using Composite Field Arithmetic

To date, there are several composite field construction schemes proposed for AES S-box implementation [34, 35, 36, 37, 38, 39, 40, 41, 42]. Rijmen [34] was the first to introduce the use of CFA in all of the transformations in AES algorithm. This work was then
reaffirmed by Rudra et al. [35] as they claimed that there were substantial gains in performance through CFA, where mapping is performed from $GF(2^8)$ to $GF((2^4)^2)$. Through this mapping, the multiplicative inversion in $GF(2^8)$ was reduced to finding the multiplicative inversion in $GF(2^4)$. Subsequently, the multiplicative inversion of the elements in field $GF(2^4)$ can be stored in a smaller LUT. Wolkerstofer et al. [36] and Satoh et al. [37] further optimised their S-boxes by introducing a new composite field mapping, from $GF(2^8)$ to $GF(((2^2)^2)^2)$ via the “tower field” approach proposed by Paar [9]. This approach was proven to result in fewer $GF(2)$ operators compared to any prior works.

In a separate development, the effects of different irreducible polynomials towards the complexity of CFA were studied by Mentes et al. [38]. In which, they explored all of the possible irreducible polynomials to search for the most optimum and compact solution for CFA AES S-Box over $GF(((2^2)^2)^2)$. At the top level of design hierarchy, Mentes’ architecture was similar to that of Satoh’s architecture. The main difference between the two architectures can be seen in the next lower level i.e. the implementation of constant multiplication with different irreducible polynomials’ coefficients.

CFA AES S-Box area reduction was further improved by Canright as reported in [39]. In his work, Canright presented the use $GF(((2^2)^2)^2)$ of normal basis representation which was not commonly adopted in the previous work as compared to polynomial basis representation. From which, Canright concluded that both polynomial and normal basis give comparable amount of operations, and the key to compact S-box implement is dependent on which irreducible polynomials of minimal coefficient were chosen.

In addition to that, Mehran et al. have presented an improved formulations for the normal basis multiplicative inversion in the subfields within the CFA AES S-box over $GF(((2^2)^2)^2)$ to achieve further area reduction [41]. They had shown that their scheme has the lowest area and power consumption compared to its counterparts in ASIC implementation using $0.81\mu$ CMOS technology.

To the best of our knowledge, the smallest AES S-box to date was reported in Boyar and Peralta’s work [42]. Boyar and Peralta proposed a new technique for combinatorial logic optimisation over $GF(((2^2)^2)^2)$ which involved two major steps. In the first step, an ad-hoc heuristics method was used to construct a circuit with compact multiplicative inversion over $GF(2^4)$, the non-linear subcircuit in AES. The second step involved
the finding of maximal linear component of the circuits and therefore the number of
XORs gate within was reduced. As a direct result, a compact S-box of 32 AND gates
and 83 XOR gates were derived.

Apart from the area reduction, another critical issue arisen during designing an opti-
mal S-box is the critical path of the resultant combinatorial circuit. Zhang and Parhi
were the first to stress on the importance of critical path in AES S-box [40]. They had
contributed a slightly different CFA AES S-box over $GF((2^2)^2)$, where direct com-
putation was used to compute multiplicative inverse over $GF(2^4)$. Therefore, further
decomposition to the subfield $GF(2^2)$ was no longer needed, as it would be required
in the previous works. Summarising the above, the approach by Zhang and Parhi con-
tributed an AES S-box with the shortest critical path but with a much larger area of
implementation compared to Boyar and Peralta’s work.

### 4.3 Very High Speed Design for AES S-box

Pipelining, sub-pipelining and loop unrolling (cf. Section 2.2 in Chapter 2) have been
widely employed for performance enhancement in hardware AES implementations
[23]. The highest attainable speed of a circuitry is determined by the longest path in
the circuitry. Therefore, pipeline cuts can be placed to shorten the delay path between
latches. For instance, in Xilinx FPGA, the minimum cycle period is determined by the
number of the cascaded 4-input LUTs in the critical path together with routing delays
[43].

By pipelining the registers between the LUTs, this would lead to an increase in speed
but it comes with a compromise in terms of amount of fan-in required by the logic
expressions in the design and the acceptable latency in the routing. The design by
Good et al. [43], yielded an optimal cut with a throughput of 12 Gbit/sec using Xilinx
Virtex-E XCV812E-8BG560. This architecture had become the major reference to many
other researches and further optimisations were proposed.

The fastest pipelined non-feedback mode AES (without adapting any algorithmic opti-
misation) was capable of achieving 25 Gbit/sec using Xilinx Spartan-III FPGA XC3S2000
[44]. This work by McLoone et al., utilised 16-stage fully pipelined and LUT-based sys-
tem. Meanwhile, the fastest implementation of the feed-back mode AES using loop
unrolling was capable in achieving throughput of 1.95 Gbit/sec based on Mitsubishi
Electric’s 0.35 micron CMOS technology [45].

However, the achieved speedup is traded off with a high increase in resource consumption, thus inefficient. In fact, subpipelining is better exploited with the adoption of CFA in AES S-box in order to achieve efficient speedup without imposing high increase in area. For instance, Hodjat and Verbauwhede employed a fast and area efficient composite field implementation for the AES S-box and followed by pipelining with an optimum amount of stages [46]. Their work managed to achieve a throughput of 21.54 Gbit/sec using VirtexII-Pro FPGA. In addition to that, Zhang and Parhi presented a loop unrolled architecture on the composite field S-box in [47]. In their work, a new design of $GF(2^4)$ multiplicative inversion block was designed and followed by seven-stages subpipelining in the loop unrolled designs. As a result, the proposed AES cipher was able to achieve an efficiency of 21.56 Gbps on a Xilinx XCV1000 e-8bg560 device in non-feedback modes.

The critical path in both the architectures in [46, 47] (which determines the highest speed achievable) was bounded to the complexities of the $GF(2^4)$ multiplier. Therefore, Liu and Parhi improved these conventional architectures by proposing a pre-computation technique over the respective multiplier [48]. This had successfully reduced the critical path delay but at the expense of area increase. In another development, Good and Bernaissa managed to outperform the prior works through further pipelining the sub-operations, specifically the multipliers, isomorphism and inverse isomorphism module [49].

### 4.4 Low Power Consumption AES Design

The requirement for low power consumption AES ciphers continue to increase significantly as many applications have become battery-powered and smaller in size. Therefore, it is essential to constrain the power consumption for the design in these low-power applications efficiently. There are three main sources of power consumption which are namely the inrush, current static and dynamic power. Inrush current is associated with the power-up sequence of a device and therefore it is device specific. The static power or sometime known as the standby power is the power consumed by a device when the power lines are active and without switching activities on the I/Os. The dynamic power, also known as the switching power, is the power associated with a device during normal operation.
The dynamic power can easily be several times greater than the standby power and thus, contributes to the major power consumption in most applications. It is proportional to the frequency of charging and discharging of the internal capacitances of a component such as the registers and combinatorial logic. This is the main reason that major efforts in power reduction schemes are taken to reduce switching activities within the circuitry.

Apart from the work by Bertoni in [33], one of the early works in deriving low power AES S-box using a refined DSE architecture was reported by Xing et al in [50]. The architecture used an optimised balanced architectures of 3-stage decoder and 4-stage encoder. The switch unit in DSE architecture performed only mapping which consumed zero power. Thus, the power reduction effort was focused in the decoder-encoder design. Zhang et al. later proposed a Sequence-Switch Coding (SSC) to reduce the inputs’ transitions in the S-box circuit to achieve further power reduction in AES[51]. SSC aimed at the data sequences rearrangement and decreasing the switching activities, which in turn will reduce the dynamic power consumption in the circuits.

In addition to that, there are several studies which worked on low power consumption CFA AES S-box modules [52, 53, 54]. Morioka and Satoh had proposed a multi-stage PPRM architecture over composite field S-box in [52]. In their work, Morioka and Satoh expressed every suboperations in their logical expressions and the resultant expressions are pipelined to avoid dynamic hazards. By reducing the occurrence of dynamic hazards, the total power consumption will be reduced accordingly. The 3-stage pipelining strategy used in their work was devised solely to reduce the power consumption in the circuit. To date, this is the lowest power composite field AES S-box reported.

On the other hand, a full-custom hardware implementation of composite field S-box using pass transmission gate (PTG) was reported by Zeng et al. in [53, 54]. PTG was used to build a 3-input XOR gates, which were later used to realise the logic function of a S-box. Furthermore, the PTG-based latches, controlled by asynchronous latch controllers, were inserted in the datapath to block the propagation of the dynamic hazards [54]. This approach managed to further reduce the power consumption in the S-box in comparison to the work in [52]. However, this method is only useful for ASIC implementation. It is not applicable in FPGA where the combinatorial functions are implemented as block of LUTs (also known as Logic Element (LE)).
The CFA AES S-box is found to have the smallest area coverage, but the estimated power consumption is larger [52, 55] than the DSE AES S-box. The reason is that composite field S-box involves many crossing and branched signal paths which results in differences in the signal arrival time of the internal gates. When this happened, the internal gates tend to switch many times per single transition and leads to the occurrence of dynamic hazards [52]. As the multiple gates are connected serially, the hazards generated by the internal gates will propagate into the circuit path and extra power will be consumed.

On the other hand, the signal arrival times in DSE AES S-box is more consistent and thus, requires far less power consumption. However, this saving is traded off with large area of consumption. Therefore, power reduction in AES S-box is a typical space-power trade-off dilemma in circuit design.

4.5 Summary Notes

AES algorithm has been implemented in various cryptographic applications of wide variety platforms. This ranges from deeply pipelined loop unrolled implementations which have performance in the order of Gbit/sec, to iterative single-rounded implementations through resource-shared designs which are both power and area saving.

Applications in optical network require secure data transmission at a very high speed, up to the rates of 30 Gbits/s. On the other hand, applications like smart cards, for instance, require minimal area of implementation as the computational and memory resources are limited. Other applications such as digital video recorder, require optimisation in terms of speed/area ratio. Apart from that, there exists high demand on applications which seek for even lower resource design with limited power supply. These include the RF identification (RFID), wireless sensor networks (WSN) and smart cards.

From the list of possible implementation approaches reported in the literature, we choose to design a pure combinatorial circuit AES S-box using CFA. The reason being that the resultant circuitry is relatively smaller compared to other alternative methods. Therefore, we will perform a series of algorithmic optimisation to further reducing the area-speed-power trade-offs in the design. In addition to that, we intend to employ pipelining in both inner and outer rounds for speed and power optimisation.
Studies show that the impact of architectural optimisation on AES, followed by the power reduction techniques, is found to be pretty useful for the overall power and energy savings [55]. Adding registers at the inputs of the large combinatorial logic can suppress the unwanted switching activities as the input will be latched only when the outputs are supposed to be updated. This can effectively reduce the amount of irrelevant switching. However, the major handicap with this technique is that it comes with a large overhead on area, complexity and cost. This is the major reason we need to adopt algorithmic optimisations for better optimisation outcome.

Overall, we aim to design an optimal AES S-box that strikes a balance in area, speed and power consumption. To summarise, our area reduction optimisation in CFA AES S-box over will be presented in two parts. First, the derivation of compact multiplicative inversion over $GF((2^2)^2)$ is discussed in Chapter 5. Second, the area reduction in the required isomorphism and its inverse, followed by affine transformation will be explained in Chapter 6. Next, the optimisation in speed and power reduction will be listed in Chapter 7. This study was also reported in our published work [56].
AES S-box Reduction: Compact

Multiplicative Inversion over $GF(((2^2)^2)^2)$

5.1 Compact AES S-box Derivation

Direct computation for the multiplicative inversion over seventh degree polynomial (modulo an eighth-degree polynomial) for AES S-box involves tedious computation. On the other hand, calculating an multiplicative inverse of a first-degree polynomial (modulo second degree polynomial) will be rather easy. This is performed through CFA where the field elements are mapped to the subfields before computing their multiplicative inverse and are transformed back to its original field [57].

CFA, also known as subfield arithmetic, is a frequent tool used in Galois field arithmetic (refer Chapter 2). This is due to the fact the technique could be used to obtain a relatively efficient implementation for specific operation such as multiplication, multiplicative inversion and exponentiation [58]. Composite field is built iteratively from lower order fields; therefore the actual mathematical computation can be done in lower field rather than in its original higher order field [59, 60, 61, 58]. Therefore, many researchers took the advantage to employ CFA in AES to achieve lower-gate complexity. The following summarises the three main steps required in performing CFA AES S-box:

1. Map all elements of field $A$ to a composite field $B$ using isomorphism function;
   
   $b = f(a) = \delta \times a$. 


2. Compute the multiplicative inverse over the field $B$; $x = b^{-1}$ (except if $b = 0$, then $x = 0$).

3. Remap the computation results to $A$, using the inverse isomorphism function; $a = f'(x) = \delta^{-1} \times x$.

The complexity of finite field arithmetic is heavily dependent on several factors: the order of the field, the representations of the field elements, (i.e. the irreducible polynomials and basis representations used) as well as the isomorphic mapping chosen for the representation. Therefore, one can conveniently take the advantage of the isomorphism to map a computation from one field to another to search for the most efficient implementation. These construction factors will be explained in the following.

### 5.1.1 Order of Field

For AES implementation, one can map the original field, $GF(2^8)$ to composite field $GF((2^4)^2)$ or $GF(((2^2)^2)^2)$ of which both are isomorphic to the original field. Since one of our goals is to seek for compact AES S-box construction, we choose the isomorphic composite field of $GF(((2^2)^2)^2)$. The reason is that in its lowest field $GF(2)$, each element is its own additive inverse; thus no further multiplicative inversion is required. With this property, the need of LUT can be eliminated totally. However, the arithmetic involved tend to get more complicated as the number of isomorphisms levels are larger. In other words, the computation in $GF(((2^2)^2)^2)$ will be more complex compared to the $GF((2^4)^2)$, but the effort is justified by the significant reduction in the number of logic gates.

### 5.1.2 Basis Representation

Elements in Galois field can be represented in different basis such as polynomial basis, normal basis and dual basis. In Galois field of $GF(q^m)$, each element can be represented as a linear combination of $m$ elements of the base. Galois field’s elements can be written as linear combinations of \{1, $a$, $a^2$, ..., $a^{m-1}$\} for polynomial basis representation and also as linear combinations of \{$a$, $a^q$, $a^{q^2}$, ..., $a^{q^{m-1}}$\} for normal basis representation. So far, only polynomial basis and normal basis are proven to be feasible and efficient for cryptographic applications.

Isomorphism from $GF(2^8)$ to $GF(((2^2)^2)^2)$ involves three stages, of which it is possible
to have each stage to be mapped to a different basis representation of choice. However, in this study a single basis representation is chosen for all three stages. The architectures to perform multiplicative inversion over $\mathbb{GF}(((2^2)^2)^2)$ will be reviewed in Section 5.2.

### 5.1.3 Irreducible Polynomial

In CFA, an irreducible polynomial (also known as field polynomial), $p(x)$ of degree $m$ over $\mathbb{GF}(q)$ can be used to construct an extension field of $\mathbb{GF}(q)$. Thus, the extension field of order $m$ is denoted by $\mathbb{GF}(q^m)$. The field $\mathbb{GF}(q)$ is then referred as the subfield of $\mathbb{GF}(q^m)$. All of the $q^m$ elements of the extension field can be represented as polynomials with the maximum degree $(m - 1)$ over $\mathbb{GF}(q)$. Therefore, these polynomials are also known as the residue class modulo of $p(x)$ over $\mathbb{GF}(q)$. The irreducible polynomial $p(x)$ will determine the algorithm for finite field arithmetic.

Once the isomorphic subfield has been determined, one would need to identify the irreducible polynomials that are used as the generator of the fields. In this work, the field has been chosen to be $\mathbb{GF}(((2^2)^2)^2)$, for which three irreducible polynomials are needed (for 3 stages of isomorphism). These being: $r(y)$ for the extension of $\mathbb{GF}(2^8)/\mathbb{GF}(2^4)$, $s(z)$ for the extension of $\mathbb{GF}(2^4)/\mathbb{GF}(2^2)$ and $t(w)$ for the extension of $\mathbb{GF}(2^2)/\mathbb{GF}(2)$. For each of the irreducible polynomials, there are several possible choices of coefficients (both norm and trace). These will be the dominant factors that determine the complexity of the arithmetic itself. Further discussion on the impact of irreducible polynomials in the CFA AES S-box can be found in Section 5.3.

### 5.1.4 Isomorphic Mapping

The mapping of a field element to the respective composite field is done by means of multiplication with an $8 \times 8$ isomorphic matrix (binary matrix). In order to perform CFA AES S-box function of a given byte, it is first multiplied with the isomorphic matrix, and followed by multiplicative inversion using CFA. The resultant from CFA is changed back to its original field by multiplication with the inverse isomorphic matrix followed by affine transformation. The isomorphic matrix, $\delta$, is derived from the field polynomials of $\mathbb{GF}(2^8)$ and its subfields. Finding the modulo 2 inversion of $\delta$ gives the inverse isomorphic matrix, $\delta^{-1}$. Affine transformation involves another bit matrix multiply and XOR with a 8-bit constant. For further simplicity, the inverse isomorphic
matrix is often combined with affine transformation to reduce one matrix multiplication operation.

In order to construct the isomorphism mappings from $GF(2^8)$ to $GF(((2^2)^2)^2)$, we need to determine the root $\beta$ that satisfies $p(\beta) = 0$ such that $p(x)$ is the irreducible polynomial defined for AES and that $\beta$ is an element in $GF(((2^2)^2)^2)$. For each isomorphic mapping, there exists eight possible isomorphic matrices for the same field representations. This is due to the fact that for each root $\beta$, there exists seven other conjugates which can be utilised. The complexities resulting from each matrix varies, thus it is crucial that the most optimum one is selected to promote further area reduction. The optimisation in both the isomorphic mapping and its inverse will be presented in Chapter 6.

5.2 Multiplicative Inverse in S-Box Algorithm Using CFA

Galois field mapping from $GF(2^8)$ to $GF(((2^2)^2)^2)$ involves three stages of isomorphism and irreducible polynomials which are stated (in their general forms) as follows:

\[
\begin{align*}
    r(y) &= y^2 + \tau y + \nu \\
    &\text{(extension of } GF(2^8)/GF(2^4)) \\
    s(z) &= z^2 + Tz + N \\
    &\text{(extension of } GF(2^4)/GF(2^2)) \\
    t(w) &= w^2 + w + 1 \\
    &\text{(extension of } GF(2^2)/GF(2))
\end{align*}
\]

(5.2.1) (5.2.2) (5.2.3)

In this chapter, we study the multiplicative inversion (specifically for S-Box algorithm) over composite field $GF(((2^2)^2)^2)$ with respect to polynomial and normal bases. Each architecture is described with reference to their respective irreducible polynomials.

First, for the isomorphism between $GF(2^8)$ and $GF(2^8)/GF(2^4)$, the element of field $GF(2^8)$, $g$, can be expressed as $\gamma_1 y + \gamma_0$, where $\gamma_1, \gamma_0 \in GF(2^4)$ and multiplication modulo $r(y)$ in (5.2.1). The pair $\gamma_1, \gamma_0$ represents element $g$ in terms of polynomial basis $[Y, 1]$ where $Y$ is the root of $r(y)$. Alternatively, the same element can also be expressed in normal basis $[Y^{16}, Y]$ using both roots of $r(y)$ as $r(y) = (y + Y)(y + Y^{16})$. 

50
Second, for the isomorphism between \( GF(2^4) \) and \( GF(2^4)/GF(2^2) \), the element of field \( GF(2^4) \), \( \gamma \), can be expressed as \( \Gamma_1 z + \Gamma_0 \), where \( \Gamma_1, \Gamma_0 \in GF(2^2) \) and multiplication modulo \( s(z) \) in (5.2.2). The pair \( \Gamma_1, \Gamma_0 \) represents element \( \gamma \) in terms of polynomial basis \([Z, 1]\) where \( Z \) is the root of \( s(z) \) or in normal basis \([Z^4, Z] \) where \( s(z) = (z + Z)(z + Z^4) \).

Last, using \( GF(2^2)/GF(2) \), the element of field \( GF(2^2) \), \( \Gamma \), can be expressed as \( g_1 w + g_0 \), where \( g_1, g_0 \in GF(2) \) with multiplication modulo \( t(w) \) in (5.2.3). The pair \( g_1, g_0 \) represents element \( \Gamma \) in terms of polynomial basis \([W, 1]\) where \( W \) is the root of \( t(w) \) and in terms of normal basis \([W^2, W] \) with \( t(w) = (w + W)(w + W^2) \).

With these isomorphisms, the operations in \( GF(2^8) \) can be expressed using simpler operations in \( GF(2^4) \) and likewise in \( GF(2^2) \). Note that subtraction is the same as addition for all the field \( GF(2^k) \) of characteristic 2. Multiplicative inversion over composite field \( GF(((2^2)^2)^2) \) using polynomial and normal bases representations are described in Section 5.2.1 and Section 5.2.2 respectively.

### 5.2.1 Multiplicative Inverse in Polynomial Basis

For multiplicative inversion over \( GF(2^8) \); we denote the multiplicative inverse of \( g = (\gamma_1 y + \gamma_0) \) as \( d = (\delta_1 y + \delta_0) \) and therefore,

\[
gd = 1 = (\gamma_1 y + \gamma_0)(\delta_1 y + \delta_0) \mod (y^2 + \tau y + \nu) \\
0 \cdot y + 1 = (\gamma_1 \delta_0 + \gamma_0 \delta_1 + \gamma_1 \delta_1 \tau)y + (\gamma_0 \delta_0 + \gamma_1 \delta_1 \nu) \quad (5.2.4)
\]

Comparing and solving both LHS and RHS of (5.2.4) gives:

\[
\begin{align*}
\gamma_1 &= \gamma_0 (\gamma_0 + \gamma_1 \tau) \delta_1 + (\gamma_1^2 \nu) \delta_1 \\
\gamma_1 &= (\gamma_0^2 + \gamma_0 \gamma_1 \tau + \gamma_1 \nu^2) \delta_1
\end{align*}
\]

and,

\[
\begin{align*}
\delta_1 &= (\gamma_0^2 + \gamma_1 \gamma_0 \tau + \gamma_1^2 \nu)^{-1} \gamma_1 \\
\delta_0 &= (\gamma_0^2 + \gamma_1 \gamma_0 \tau + \gamma_1^2 \nu)^{-1} (\gamma_0 + \gamma_1 \tau) \quad (5.2.5)
\end{align*}
\]
Equation \( \Delta d \) \( \delta \)

From (5.2.5) and (5.2.6), we learn that multiplicative inversion over \( \text{GF}(2^8) \) and eventually been reduced to a multiplicative inversion and some multiplications in \( \text{GF}(2^4) \). Multiplicative inversion in \( \text{GF}(2^4) \) can be derived using the similar steps and eventually been reduced to a multiplicative inversion and some multiplications in \( \text{GF}(2^2) \). Multiplication in \( \text{GF}(2^4) \) is performed as follows:

\[
(G_1z + G_0)(\Delta_1z + \Delta_0) \mod (z^2 + Tz + N)
\]

\[
= [(G_1\Delta_1)z^2 + (G_1\Delta_0 + G_0\Delta_1)z + (G_0\Delta_0)] + (G_1\Delta_1)(z^2 + Tz + N)
\]

\[
= (G_1\Delta_0 + G_0\Delta_1 + G_1\Delta_1 T)z + (G_0\Delta_0 + G_1\Delta_1 N)
\]

Similar treatment can be applied for the multiplication in \( \text{GF}(2^2) \). All the operations required in performing multiplicative inverse of the field \( \text{GF}(((2^2)^2)^2) \) using polynomial basis representation are summarised and tabulated in Table 5.1.

### 5.2.2 Multiplicative Inverse in Normal Basis

In the case of normal basis, the treatment differs slightly. The same set of irreducible polynomials will be used here, i.e. the equations in (5.2.1), (5.2.2) and (5.2.3). For element \( g \) in \( \text{GF}(2^8) \), the multiplicative inverse of \( g = (\gamma_1 Y^{16} + \gamma_0 Y) \) is now denoted as \( d = (\delta_1 Y^{16} + \delta_0 Y) \). Note that both \( Y^{16} \) and \( Y \) satisfy the irreducible polynomial \( y^2 + \tau y + \nu \), where \( Y^{16} + Y = \tau \) and \( (Y^{16})Y = \nu \). Setting \( (Y^{16} + Y)\tau^{-1} = 1 \) gives,

\[
gd = 1
\]

\[
= (\gamma_1 Y^{16} + \gamma_0 Y)(\delta_1 Y^{16} + \delta_0 Y)
\]

\[
= (\gamma_1\delta_1)(Y^{16})^2 + (\gamma_1\delta_0 + \gamma_0\delta_1)(Y^{16}Y) + (\gamma_0\delta_0)Y^2 \quad (5.2.7)
\]

### Table 5.1: Multiplicative inverse for \( \text{GF}(2^8) \) in polynomial basis

<table>
<thead>
<tr>
<th>Operation</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicative inversion in ( \text{GF}(2^8) )</td>
<td>( d_1 = (\gamma_1 + \gamma_0 z)(\gamma_1 + \gamma_0 \Delta_1)z + (\gamma_1\Delta_0 + \gamma_0\Delta_1) + (\gamma_1\Delta_1)z^2 + (\gamma_1\Delta_0 + \gamma_0\Delta_1)z + (\gamma_1\Delta_1) \mod (z^2 + Tz + N) )</td>
</tr>
<tr>
<td>Multiplicative inversion in ( \text{GF}(2^4) )</td>
<td>( d_0 = \gamma_0 + \gamma_0 \Delta_1 + \gamma_0 \Delta_1 Tz + \gamma_0\Delta_0 + \gamma_0\Delta_1 N )</td>
</tr>
<tr>
<td>Multiplication in ( \text{GF}(2^8) )</td>
<td>( (G_1\Delta_0 + G_0\Delta_1 + G_1\Delta_1 T)z + (G_0\Delta_0 + G_1\Delta_1 N) )</td>
</tr>
<tr>
<td>Multiplication in ( \text{GF}(2^4) )</td>
<td>( (G_1 + G_0)(d_1 + d_0) + (G_0\Delta_0 + G_0\Delta_1) \nu + (G_1\Delta_0 + G_1\Delta_1) \nu )</td>
</tr>
</tbody>
</table>
By substituting $Y^{16} + Y = \tau$ to $(Y^{16})Y = \nu$, we will obtain $Y^2 = \tau Y + \nu$ and $(Y^{16})^2 = \tau Y^{16} + \nu$. Subsequently (5.2.7) becomes,

\[
gd = (\gamma_1 \delta_1)(\tau Y^{16} + \nu) + (\gamma_1 \delta_0 + \gamma_0 \delta_1)\nu + (\gamma_0 \delta_0)(\tau Y + \nu)
\]

\[
1 = (\gamma_1 \delta_1 \tau Y^{16} + (\gamma_0 \delta_0 \tau)Y + (\gamma_1 \delta_1 + \gamma_1 \delta_0 + \gamma_0 \delta_1 + \gamma_0 \delta_0)\nu
\]

\[
\tau^{-1}Y^{16} + \tau^{-1}Y = [\gamma_1 \delta_1 \tau + (\gamma_1 + \gamma_0)(\delta_1 + \delta_0)\nu \tau^{-1}]Y^{16} + [\gamma_0 \delta_0 \tau + (\gamma_1 + \gamma_0)(\delta_1 + \delta_0)\nu \tau^{-1}]Y
\]

Comparing and solving both LHS and RHS of (5.2.8), we have:

\[
\tau^{-1} = \gamma_1 \delta_1 \tau^2 + (\gamma_1 + \gamma_0)(\delta_1 + \delta_0)\nu
\]

\[
1 = \gamma_1 \delta_1 \tau^2 + (\gamma_1 + \gamma_0)(\delta_1 + \delta_0)\nu
\]

\[
\gamma_0 = [\gamma_1 \gamma_0 \tau^2 + (\gamma_1^2 + \gamma_0^2)\nu] \delta_1
\]

and therefore,

\[
\delta_1 = [\gamma_1 \gamma_0 \tau^2 + (\gamma_1^2 + \gamma_0^2)\nu]^{-1} \gamma_0
\]

\[
\delta_0 = [\gamma_1 \gamma_0 \tau^2 + (\gamma_1^2 + \gamma_0^2)\nu]^{-1} \gamma_1
\]

Multiplicative inversion in $GF(2^8)$ is reduced to a multiplicative inversion and several multiplications (include scaling and squaring) in $GF(2^4)$ as shown in (5.2.9) and (5.2.10). Similar approach is followed in finding multiplicative inverse of field $GF(2^4)$, which is then reduced to a multiplicative inverse and several multiplications in $GF(2^2)$. Take note that the multiplicative inversion and squaring are identical operations in the field $GF(2^2)$ and they require only swapping operation without requiring any logic gate.

Multiplication in $GF(2^4)$ in general form can be deduced as the following.

\[
(\Gamma_1 Z + \Gamma_0 Z)(\Delta_1 Z + \Delta_0 Z)
\]

\[
= (\Gamma_1 \Delta_1)(Z^4)^2 + (\Gamma_1 \Delta_0 + \Gamma_0 \Delta_1)Z^4 Z + (\Gamma_0 \Delta_0)Z^2
\]
(5.2.11) can be further simplified as $Z^4 + Z = T$ and $Z^4Z = N$. Similarly, multiplication in $GF(2^2)$ is derived using the same way. All the operations that are essential for performing multiplicative inverse of element in the field $GF(2^8)$ using normal basis representation are summarised in Table 5.2.

## 5.3 Impact of Irreducible Polynomials in Multiplicative Inversion Architecture

Coefficients in the irreducible polynomials have a direct impact over the complexities in finite field arithmetic. Consequently, the operations required for multiplicative inverse over composite field $GF(((2^2)^2)^2)$ (as summarized in Table 5.1 and Table 5.2) can be simplified depending on the chosen coefficients of their respective irreducible polynomials; $r(y)$, $s(z)$ and $t(w)$ (cf. (5.2.1), (5.2.2) and (5.2.3)).

Only a few coefficients that ensure the irreducibility of the polynomials can be used. A polynomial over $GF(2^k)$ is irreducible if and only if it cannot be factored into nontrivial polynomials other than itself over $GF(2^k)$. As $w^2 + w + 1 = 0$ is the only irreducible polynomial of degree 2 over $GF(2)$, there is no other candidate coefficient for (5.2.3). Meanwhile, for (5.2.1) and (5.2.2), we need to determine all of the valid coefficients of $\nu$, $\tau$, $N$ and $T$ in both normal and polynomial bases. In order to promote simplicity in CFA, we can either have the trace or the norm of $r(y)$ and $s(z)$ equal to unity but not both. Using this approach, there are two possible $T$ values for $s(z)$ to be irreducible over $GF(2^2)$. For $r(y)$ to be irreducible over $GF(((2^2)^2)^2)$, there are eight choices for $\tau$ with respect to each of the $T$ values. Based on the irreducibility test, the possible values of $\tau$, $\nu$, $T$ and $N$ for the simplicity approach and considering both polynomial and normal bases are tabulated in Tables A.1, A.2, A.3 and A.4 in Appendix A.

To our best knowledge, previous studies attempted optimisation with traces of irre-
ductible polynomials equal to unity only. We denoted these architectures as Case $i$ and Case $ii$ for polynomial basis and normal basis representation respectively. Further details of both cases can be found in Appendix B. Therefore, we extend these studies by choosing the norms of the irreducible polynomials to be unity for both polynomial and normal bases. These two new constructions (hereafter, referred to as Case I and Case II) are then compared with the ones in Case $i$ and Case $ii$. Eventually, based on the thorough reviews of these architectures, we derived yet another new normal basis composite field AES S-box that uses a combination of norm and trace unities in different irreducible polynomials, Case III. As a result, we propose three new constructions in total, which are as listed below. These constructions were reported in our publication [56].

1. Case I (see Figure 5.1): Using polynomial basis representation with irreducible polynomials’ norms equal to unity (both $\nu$ and $N$ in (5.2.1) and (5.2.2) equal to unity).

2. Case II (see Figure 5.2): Using normal basis representation with irreducible polynomials’ norms equal to unity (both $\nu$ and $N$ in (5.2.1) and (5.2.2) equal to unity).

3. Case III (see Figure 5.3): Using normal basis representation with $\tau$ in (5.2.1) and $N$ in (5.2.2) equal to unity.

5.4 Optimisation of CFA Architectures

To further enhance the performance of the proposed constructions, we now perform some optimisation steps from both the algorithmic and the architectural viewpoint. In algorithmic optimisation, careful selection of the irreducible polynomials’ coefficients that result in minimal arithmetic complexity is conducted (refer Section 5.4.1). In architectural level, elimination of the redundant common factor in the multiplicative inverter is performed, followed by a merger of certain multipliers with its respective sub-operations (refer Section 5.4.2).

5.4.1 Algorithmic Strength Reduction

As aforementioned, one has a certain level of freedom in the choice of coefficients in the irreducible polynomials $r(y)$ and $s(z)$. Each of the coefficient combinations will lead
CHAPTER 5: AES S-box Reduction: Compact Multiplicative Inversion over $GF(((2^2)^2)^2)$

Figure 5.1: Case I: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in polynomial basis with irreducible polynomials’ norms equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$

Figure 5.2: Case II: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in normal basis with irreducible polynomials’ norms equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$
Figure 5.3: Case III: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in normal basis with combination of trace and norm equal unity in different irreducible polynomials. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$

to an individual level of complexities in CFA. The following presents the algorithmic optimisation available in each architectures, namely the Case I, Case II and Case III. For reference purposes, the similar optimisations which are applicable to Case $i$ and Case $ii$ can be found in Appendix B.

### Optimisations for Case I

In the course of investigating the architecture with its irreducible polynomials’ norms equal to unity, a new composite field construction was deduced (refer to (5.4.1) and (5.4.2)). In $GF(2^8)/GF(2^4)$, the multiplicative inversion over this new composite field construction is as follows,

\[ \delta_1 = (\gamma_1^2 + \gamma_1 \gamma_0 \tau + \gamma_0^2)^{-1} \gamma_1 \]  
\[ \delta_0 = (\gamma_1^2 + \gamma_1 \gamma_0 \tau + \gamma_0^2)^{-1}(\gamma_0 + \gamma_1 \tau) \]  

which is reduced to multiplicative inversion and multiplications in the field of $GF(2^4)$. Multiplication by a known constant, $\tau$, which is known as scaling, ought to have a
separate circuit rather than using a generic $GF(2^4)$ multiplier. The same is true for squaring, $\gamma^2$ as well. The scaling of $\gamma = \Gamma_1 z + \Gamma_0 (GF(2^4))$ by $\tau = \Delta_1 z + \Delta_0$ is deduced as the following:

\[
\gamma \tau = (\Gamma_1 z + \Gamma_0)(\Delta_1 z + \Delta_0) \mod (z^2 + Tz + 1)
\]
\[
= (\Gamma_1 \Delta_0 + \Gamma_0 \Delta_1 + \Gamma_1 \Delta_1 T) z + (\Gamma_0 \Delta_0 + \Gamma_1 \Delta_0) \quad (5.4.3)
\]

Meanwhile, squaring of $\gamma = \Gamma_1 z + \Gamma_0 (GF(2^4))$ is:

\[
\gamma^2 = (\Gamma_1 z + \Gamma_0)(\Gamma_1 z + \Gamma_0) \mod (z^2 + Tz + 1)
\]
\[
= (\Gamma_1^2 T) z + (\Gamma_0^2 + \Gamma_1^2) \quad (5.4.4)
\]

Both (5.4.3) and (5.4.4) require sub-operations in the field of $GF(2^2)$. The elements in $GF(2^2)$ are $\{0, 1, T, T + 1\}$, but only $T$ and $T + 1$ are valid for $s(z) = z^2 + Tz + 1$ to be irreducible. In (5.4.3) and (5.4.4), we need both scaler and squarer of $GF(2^2)$ as defined in the following:

\[
(w) \otimes (g_1 w + g_0) = (g_1 \oplus g_0) w + g_1 \quad (5.4.5)
\]
\[
(w^2) \otimes (g_1 w + g_0) = (g_0) w + (g_0 \oplus g_1) \quad (5.4.6)
\]

and

\[
\Gamma^2 = (g_1 w + g_0)(g_1 w + g_0) \mod (w^2 + w + 1)
\]
\[
= (g_1) w + (g_0 + g_1) \quad (5.4.7)
\]

Both scaling ((5.4.5), (5.4.6)) and squaring, (5.4.7) in $GF(2^2)$ require only one addition each. In addition to that, as every nonzero element in $GF(2^2)$, $\Gamma$, satisfies $\Gamma^3 = 1$, thus $T^{-1} = T^2 = T + 1$, i.e. the multiplicative inverter is same as the squarer. It is worth noticing that a combination of the (5.4.7) and (5.4.5) in the field of $GF(2^2)$ produces,
Optimisation in multiplicative inversion for Case I (polynomial basis and norm unity), with $\gamma = \Gamma_1 z + \Gamma_0$ and $\tau = \Delta_1 z + \Delta_0$. The pairs of $T$ and $\tau$ that result in minimal complexity in $\gamma \tau$ are determined.

<table>
<thead>
<tr>
<th>$\Delta_1$</th>
<th>$\Delta_0$</th>
<th>$T$</th>
<th>$\tau$</th>
<th>$\gamma \tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>${10}_2$</td>
<td>${0100}_2$</td>
<td>$\gamma \tau = (\Gamma_0 + \Gamma_1) z + (\Gamma_1)$</td>
</tr>
<tr>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>${11}_2$</td>
<td>${1010}_2$</td>
<td>$\gamma \tau = [\Gamma_1 (\Gamma_1 + \Gamma_0) + \Gamma_1 z + (\Gamma_2^2 (\Gamma_1 + \Gamma_0))]$</td>
</tr>
</tbody>
</table>

\[
(w) \otimes \Gamma^2 = (w) \otimes (g_1 w + g_0)^2 = (w) \otimes (g_1 w + (g_0 + g_1)) = (g_0) w + g_1
\] (5.4.8)

which is a free operation using swapping operation and no logic gate is required. Specific circuits for (5.4.3) can be minimised with any pairs of $\Delta_1$ and $\Delta_0$ as noted in Table 5.3.

**Optimisations for Case II**

Similar optimisations are available in normal basis as well, though some details have changed. In $GF(2^8)/GF(2^4)$, the multiplicative inversion involved is as follows,

\[
\delta_1 = [(\gamma_1 \gamma_0 \tau^2 + (\gamma_1^2 + \gamma_0^2))^{-1} \gamma_0]
\] (5.4.9)

\[
\delta_0 = [(\gamma_1 \gamma_0 \tau^2 + (\gamma_1^2 + \gamma_0^2))^{-1} \gamma_1]
\] (5.4.10)

which is reduced to multiplicative inversion and multiplications in the field of $GF(2^4)$. Same as the previous case, two specific circuits for scaling and squaring in $GF(2^4)$ are required (based on (5.4.9) and (5.4.10)). Squaring of $\gamma = (\Gamma_1 Z^4 + \Gamma_0 Z)$ is deduced as:

\[
\gamma^2 = [(\Gamma_1^2 + \Gamma_0^2 T^2)] Z^4 + [(\Gamma_1^2 + \Gamma_0^2 T^2)] Z
\] (5.4.11)

Meanwhile, the scaling $\gamma = \Gamma_1 Z^4 + \Gamma_0 Z \ (GF(2^4))$ by $\tau^2 = \Delta_1 Z^4 + \Delta_0 Z$, with $\tau$ values that contribute to minimal complexity, are tabulated in Table 5.4. Both of the specific
Table 5.4: Optimisation in multiplicative inversion Case II (normal basis and norm unity), with \( \gamma = \Gamma_1 z + \Gamma_0 \) and \( \tau^2 = \Delta_1 z + \Delta_0 \). The pairs of \( T \) and \( \tau \) that result in minimal complexity in \( \gamma \tau \) are determined.

<table>
<thead>
<tr>
<th>( \Delta_1 )</th>
<th>( \Delta_0 )</th>
<th>( T )</th>
<th>( \tau )</th>
<th>( \gamma \tau^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>{01}_2</td>
<td>{1100}_2</td>
<td>( \gamma \tau^2 = (\Gamma_1 T^2 + \Gamma_0)Z^4 + (\Gamma_1)Z )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>{10}_2</td>
<td>{1100}_2</td>
<td>( \gamma \tau^2 = (\Gamma_1 T^2 + \Gamma_0)Z^4 + (\Gamma_1)Z )</td>
</tr>
</tbody>
</table>

Circuits require sub-operations in the field of \( GF(2^2) \). Scaling of both \( T \) and \( T + 1 \) are derived:

\[
(W) \otimes (g_1 W^2 + g_0 W) = (g_0 \oplus g_1) W^2 + (g_1) W \quad (5.4.12)
\]

\[
(W^2) \otimes (g_1 W^2 + g_0 W) = (g_0) W^2 + (g_0 \oplus g_1) W \quad (5.4.13)
\]

And squaring of \( \Gamma = g_1 W^2 + g_0 W \) in \( GF(2^2) \) is given by

\[
\Gamma^2 = (g_1 W^2 + g_0 W)(g_1 W^2 + g_0 W) = (g_0 W^2 + g_1 W) \quad (5.4.14)
\]

On the other hand, a combination of scaling, (5.4.12) and squaring, (5.4.14) gives

\[
WT^2 = W(g_0 W^2 + g_1 W) = (g_0 + g_1) W^2 + g_0 W
\]

Note that (5.4.12) and (5.4.13) require only one addition each. For normal basis, squaring in field \( GF(2^2) \), (5.4.14), is a free operation, which is performed through swapping.

Optimisations for Case III

Architecture in Case III is very similar to the one in Case II. The only difference is having \( \tau \), rather than \( \nu \), in (5.2.1) equal to unity. Subsequently, both architectures will share the same operation in the field of \( GF(2^4) \) and \( GF(2^2) \), while the multiplicative inversion in \( GF(2^8)/GF(2^4) \), is as follows:

\[
\delta_1 = [(\gamma_1 \gamma_0 + (\gamma_1^2 + \gamma_0^2)\nu)^{-1}\gamma_0] \quad (5.4.15)
\]

\[
\delta_0 = [(\gamma_1 \gamma_0 + (\gamma_1^2 + \gamma_0^2)\nu)^{-1}\gamma_1] \quad (5.4.16)
\]
Table 5.5: Optimisation in multiplicative inversion Case III (normal basis and combination of trace and norm unity), with $\gamma = \Gamma_1 z + \Gamma_0$ and $\nu^2 = \Delta_1 z + \Delta_0$. Pairs of $T$ and $\nu$ that result in minimal complexity in $\nu\gamma^2$ are determined.

<table>
<thead>
<tr>
<th>$\Delta_1$</th>
<th>$\Delta_0$</th>
<th>$T$</th>
<th>$\tau$</th>
<th>$\nu\gamma^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>{01}_2</td>
<td>{0001}_2</td>
<td>$\nu\gamma^2 = [T(\Gamma_1 + \Gamma_0)^2]Z^4 + [\Gamma_0^\prime]Z$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>{10}_2</td>
<td>{0010}_2</td>
<td>$\nu\gamma^2 = [T(\Gamma_1 + \Gamma_0)^2]Z^4 + [\Gamma_0^\prime]Z$</td>
</tr>
</tbody>
</table>

which is reduced to multiplicative inversion and multiplications in the field of $GF(2^4)$. In this case (refer to (5.4.15) and (5.4.16)), we can combine scaling of constant $\nu = \Delta_1 z + \Delta_0$ with squaring of $\gamma = \Gamma_1 z + \Gamma_0$. As such, only one additional specific circuit for $\nu\gamma^2$ is needed. Squaring in $GF(2^4)$ is defined in (5.4.11) and therefore $\nu\gamma^2$ would be deduced as follows:

$$\nu\gamma^2 = [(\Delta_1 \Gamma_1^2)T^2 + \Delta_0 (\Gamma_1 + \Gamma_0)^2]Z^4 + [(\Delta_0 \Gamma_0^2)T^2 + \Delta_1 (\Gamma_1 + \Gamma_0)^2]Z$$  \hspace{1cm} (5.4.17)$$

The $\nu$ values that contribute to minimal complexity in this square-scaler are tabulated in Table 5.5.

5.4.2 Substructure Sharing at Architectural Level

Up to this point, the best irreducible polynomials to construct the smallest multiplicative inverse over composite field $GF(((2^2)^2)^2)$ have been determined for each case. Substructure sharing at the architectural level is then deployed to remove the redundancy factor in CFA, as well as merging certain multipliers with sub-operations, in order to further deduce a smaller architecture [39].

First, a potential sharing is readily available in the subfield multipliers. The sum of the higher and lower halves of each factor can be shared between two or more subfield multipliers which have the same input factor. Note that a 2-bit factor shared by two $GF(2^2)$ multipliers saves one XOR addition, while a 4-bit factor shared by two $GF(2^4)$ multipliers saves five XORs. Polynomial basis multiplicative inversion (Case I) shared common input in two field multipliers (of $GF(2^8)$ and $GF(2^4)$ multiplicative inverters) and this saves a total of $2 \times 5 + 2 = 12$ XORs. On the other hand, normal basis multiplicative inversion (Case II and III) shared common factor in three field multipliers and therefore manage to save a total of $3 \times 5 + 3 = 18$ XORs. Though sub-sharing of com-
mon factors reduces a significant amount of XOR gates in the area of implementation, this does not contribute to any reduction in critical path.

Second, combining sub-operations in $GF(2^4)$ is capable of producing further area savings. To begin with, we combine a $GF(2^2)$ multiplier with a scaler in a $GF(2^4)$ multiplier. Such approach results in saving three XORs in total gates and one XOR in the critical path for the $GF(2^4)$ multiplier in Case II and Case III. Then, another potential multiplier-scaler combination is available in $GF(2^4)$ multiplicative inverter. Such merging gives one additional XOR reduction in both total gates and critical path for both cases as well.

Last, Case III architecture gets to have additional subtle saving in its $GF(2^8)$ multiplicative inverter. Combining the sum of higher and lower halves of the inputs (common factors with $GF(2^4)$ multiplier) and the following square-scaler saves two XORs.

## 5.5 Discussion

Multiplicative inversion using CFA can be optimised in both algorithmic and architectural wise. For the prior (refer to Subsection 5.4.1), the resulting hardware requirements in terms of gate count and critical path of the best cases in each architecture (Case I, II and III) are noted in Table 6.4 (see Chapter 6) under column ‘Optimised Algorithm’.

In the substructure sharing, we observed that normal basis architecture gain more saving in substructure sharing than that of polynomial basis. Both total logic count and critical path for the three CFA AES S-box architectures are analyzed and tabulated in Table 6.4 (see Chapter 6) under column ‘Optimised Architecture’.

So far, the most compact circuitry with the shortest critical path is found in Case III. It consumes a total of 36 ANDs and 97 XORs and having the critical path of 4 ANDs and 17 XORs after algorithmic optimisation. The architecture is further reduced to 36 ANDs and 67 XORs with critical path of 4 ANDs and 14 XORs after architectural optimisations.

Up to this point, we only showed the circuitries for efficient multiplicative inversion over $GF(((2^2)^2)^2)$, as summarized in Table 5.6. Not to forget that, S-box function over composite field involves isomorphic mapping and its inverse, followed by affine transformations. These operations contribute a significant amount of computation that will determine the area coverage of AES S-box in hardware. Therefore, it is essential to per-
form area reduction for these operation, of which will be discussed in the next chapter.
Table 5.6: Detailed description of CFA AES S-box for Case I, II and III with Case I using polynomial basis and Case II and III using normal basis

<table>
<thead>
<tr>
<th>Computation</th>
<th>Case I (Norm Unity)</th>
<th>Case II (Norm Unity)</th>
<th>Case III (Trace + Norm Unity)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiplicative inversion in (GF(2^4))</strong></td>
<td>(d_1 = (\gamma_1^3 + \gamma_1^2\gamma_0 + \gamma_0^2)^{-1}\gamma_1)</td>
<td>(d_1 = [\gamma_1^3 + (\gamma_1^3 + \gamma_0^2)^{-1}\gamma_1\gamma_2])</td>
<td>(d_1 = [\gamma_1^3 + (\gamma_1^3 + \gamma_0^2)^{-1}\gamma_1\gamma_2])</td>
</tr>
<tr>
<td>(d_0 = (\gamma_0^3 + \gamma_1\gamma_0 + \gamma_1^2)^{-1}(\gamma_0 + \gamma_1))</td>
<td>(d_0 = [\gamma_0^3 + (\gamma_1^3 + \gamma_0^2)^{-1}\gamma_1\gamma_2])</td>
<td>(d_0 = [\gamma_0^3 + (\gamma_1^3 + \gamma_0^2)^{-1}\gamma_1\gamma_2])</td>
<td></td>
</tr>
<tr>
<td><strong>Multiplicative inversion in (GF(2^4))</strong></td>
<td>(\gamma^2 = (\Gamma_1 T^2 + \Gamma_1)\gamma_1)</td>
<td>(\gamma^2 = ((\Gamma_1 T^2 + \Gamma_2) \gamma_1 + (\Gamma_3 T)^2 + \Gamma_3))</td>
<td>(\gamma^2 = ((\Gamma_1 T^2 + \Gamma_2) \gamma_1 + (\Gamma_3 T)^2 + \Gamma_3))</td>
</tr>
<tr>
<td><strong>Multiplicative inversion in (GF(2^4))</strong></td>
<td>(\Delta_1 = (\Gamma_1^{\prime} + \Gamma_2 T + \Gamma_3)^{-1}\Delta_1)</td>
<td>(\Delta_1 = (\Gamma_1^{\prime} + \Gamma_2 T + \Gamma_3)^{-1}\Delta_1)</td>
<td>(\Delta_1 = (\Gamma_1^{\prime} + \Gamma_2 T + \Gamma_3)^{-1}\Delta_1)</td>
</tr>
<tr>
<td>(\Delta_0 = (\Gamma_1^{\prime} + \Gamma_2 T + \Gamma_3)^{-1}(\Gamma_0 + \Gamma_1 T))</td>
<td>(\Delta_0 = (\Gamma_1^{\prime} + \Gamma_2 T + \Gamma_3)^{-1}(\Gamma_0 + \Gamma_1 T))</td>
<td>(\Delta_0 = (\Gamma_1^{\prime} + \Gamma_2 T + \Gamma_3)^{-1}(\Gamma_0 + \Gamma_1 T))</td>
<td></td>
</tr>
<tr>
<td>(\Gamma^2 = \gamma_0^3 \gamma_1 T + \gamma_1 + \gamma_1)</td>
<td>(\Gamma^2 = \gamma_0^3 \gamma_1 T + \gamma_1 + \gamma_1)</td>
<td>(\Gamma^2 = \gamma_0^3 \gamma_1 T + \gamma_1 + \gamma_1)</td>
<td></td>
</tr>
<tr>
<td>(\Gamma T = \gamma_0^3 \gamma_1 T + \gamma_1 + \gamma_1)</td>
<td>(\Gamma T = \gamma_0^3 \gamma_1 T + \gamma_1 + \gamma_1)</td>
<td>(\Gamma T = \gamma_0^3 \gamma_1 T + \gamma_1 + \gamma_1)</td>
<td></td>
</tr>
<tr>
<td><strong>Multiplication in (GF(2^4))</strong></td>
<td>((\Gamma_1^{\prime} + \Gamma_2 T)(\Delta_0 + \Delta_1) - \Gamma_0 \Delta_0 - \Gamma_1 \Delta_1 + \Gamma_1 T^2)</td>
<td>((\Gamma_1^{\prime} + \Gamma_2 T)(\Delta_0 + \Delta_1) - \Gamma_0 \Delta_0 - \Gamma_1 \Delta_1 + \Gamma_1 T^2)</td>
<td>((\Gamma_1^{\prime} + \Gamma_2 T)(\Delta_0 + \Delta_1) - \Gamma_0 \Delta_0 - \Gamma_1 \Delta_1 + \Gamma_1 T^2)</td>
</tr>
<tr>
<td>((\Gamma_0 + \Gamma_1 T))</td>
<td>((\Gamma_0 + \Gamma_1 T))</td>
<td>((\Gamma_0 + \Gamma_1 T))</td>
<td></td>
</tr>
<tr>
<td><strong>Multiplication in (GF(2^4))</strong></td>
<td>((\gamma_1 + \gamma_0)\Delta_0 + \gamma_0 + (\gamma_1 \gamma_0))</td>
<td>((\gamma_1 + \gamma_0)\Delta_0 + \gamma_0 + (\gamma_1 \gamma_0))</td>
<td>((\gamma_1 + \gamma_0)\Delta_0 + \gamma_0 + (\gamma_1 \gamma_0))</td>
</tr>
<tr>
<td>((\gamma_0 \gamma_1 + \gamma_0 \gamma_0)\gamma_1 T + (\gamma_0 \gamma_1 + \gamma_0 \gamma_0))</td>
<td>((\gamma_0 \gamma_1 + \gamma_0 \gamma_0)\gamma_1 T + (\gamma_0 \gamma_1 + \gamma_0 \gamma_0))</td>
<td>((\gamma_0 \gamma_1 + \gamma_0 \gamma_0)\gamma_1 T + (\gamma_0 \gamma_1 + \gamma_0 \gamma_0))</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 6

AES S-box Reduction: Common Subexpression Elimination Algorithm

6.1 Common Subexpression Elimination Algorithm

Common subexpression elimination (CSE) is a critical procedure in many multiplier-less implementation of DSP algorithms. The aim of CSE is dual-pronged: 1) to reduce the number of logic operators used and 2) to minimise the logic depth (critical path) of the DSP algorithm implemented in VLSI. It is an optimisation procedure that searches for instances of identical expressions and replacing them with a single variable that holds the computed value. With this, the identical expressions will be computed only once and leading to a savings in terms of the hardware required.

CSE is often used in solving the multiple constant multiplication (MCM) problem. MCM is optimisation in generation of a minimal multiplier block from the set of constants. Avoiding costly multipliers is an essential design criterion in VLSI implementation. Therefore, MCM problems are often related to digital finite impulse response (FIR) filter implementation as well as DSP linear transformation. A simple idea of CSE demonstration on FIR filter is illustrated in Figure 6.1.

Apart from that, CSE is also commonly used in tackling substructure sharing optimisation in combinatorial logic implementation, which is similar to the MCM problem. This combinatorial logic is expressed in bit-level equations, which CSE can be exploited to extract the common factors in all the equations to reduce the area cost. An example of
substructure sharing optimisation using CSE is noted in Figure 6.2.

\[ X \rightarrow X \rightarrow X \]
\[ h_1 = 0.0110 \quad h_2 = 0.10110 \]
\[ D + D + D \rightarrow Y \]

(a)

Pattern

\[ X \rightarrow + \rightarrow >>1 \rightarrow Y_1 \]
\[ X \rightarrow + \rightarrow >>2 \rightarrow Y_2 \]
\[ X \rightarrow + \rightarrow >>3 \rightarrow Y_3 \]

(b)

\[ X \rightarrow + \rightarrow >>2 \rightarrow Y_1 \]
\[ + \rightarrow >>1 \rightarrow Y_2 \]
\[ + \rightarrow >>3 \rightarrow Y_3 \]

(c)

**Figure 6.1:** CSE in FIR filter design

In general, any CSE algorithm involves the following steps:

1. Identify patterns (common factors) present in the transformation.
2. Select a pattern for elimination.
3. Remove all the occurrences of the selected pattern.
4. Compute the eliminated pattern only once.
5. Repeat Step 1 - 4 until none multiple patterns is present.

CSE algorithm can be employed in promoting area optimisation in linear transformations using binary matrix multiplication. To be precise, we extend the use of CSE in the composite field implementation of AES. In doing so, we propose a novel CSE algorithm that results in optimum substructure sharing scheme for isomorphism function and affine transformation in composite field AES S-box.
One of the important issues in CSE is the optimality of the algorithm. In particular, the optimality referred here emphasises on the elimination of logic operators. However, by eliminating one pattern, we are very likely to lose other possible patterns owing to the sharing of the nonzero bits [62]. Therefore, the key issue here is to select the right pattern and eliminate over the others that will result in maximal reduction. However, this pattern selection (Step 2) problem is claimed to be NP-complete [62]. Since there is no exact algorithm to solve this problem, the previous studies, as well as our work, attempted on the heuristic approach to achieve solution that is as optimal as possible. Apart from maximum pattern reduction, our algorithm also emphasises on another important feature - short critical path.

### 6.2 Related Works in CSE

To date, there are several studies that have worked on CSE for constant multiplications optimisation in linear systems, particularly digital filters [63, 64, 62, 65, 66, 67, 68]. Whereas, CSE specifically for binary linear transformation (Galois field isomorphic mapping) was introduced by Hsiao et al. in [1]. The technique utilised greedy algorithm along with three other criteria in pattern selection, which differs with the algorithm proposed herein. Besides, their iterative two-term pattern selection does not guarantee minimal critical path.

On the other hand, the work reported by Pasko et al. [62] employed both exhaustive search and greedy algorithm in different stages. The prior is performed to extract all
the possible patterns multiplying a single variable (pattern identification), while only
the latter is used in pattern selection. In the work by Hosangadi et al. in [68], a polyno-
mial transformation of the constant multiplications is employed to enable the detection
of multi-variable pattern using a rectangle covering an intersection matrix. Following
which, they used heuristic ping-pong algorithm in order to obtain a good prime rect-
gle that result in inferior results in certain cases.

As most linear DSP systems involved in signed integers matrix multiplication, the ma-
jor effort in designing CSE algorithms is to emphasise on pattern identification either
from a single variable [63, 62, 67] or multiple variables [64, 68] multiplications. As a
result, the elimination strategy will be very much specific to the structure of their pat-
terns. Consequently, most of the reported CSE algorithms are incapable of giving direct
optimisation in binary linear transformation.

In this chapter, we present a new CSE algorithm that combines both greedy algorithm
and exhaustive search in an iterative two-term pattern selection, which is also reported
in one of our conference paper [69].

### 6.3 Proposed Common Subexpression Elimination Algorithm

First, an exhaustive search for all of the possible 2-term patterns in the expressions is
performed. Next, a complete statistic of the pattern frequencies are deduced. Using the
greedy algorithm, the pattern which has the highest frequency will then be selected. If
there are more than one pattern with the same highest occurrence, exhaustive search
will be applied in addition to greedy algorithm. As such, all of the patterns that bear the
highest occurrence and their consequents pairings will be derived. Every occurrence
of the selected pattern are then renamed as the new 4-term pattern. The process will
iterate until there is no 2-term pattern is found.

Next, we move on to the second stage with the same algorithm, the newly named 4-
term pattern is paired in order of priority. Again the paired factors are renamed as
a new 8-term pattern. This process is repeated until there is only one pattern or one
nonzero term in each expression. The constructed tree is searched and the path (com-
mon sub-expressions elimination) that results in the least gate count is determined.

The occurrence of renaming the pattern is equivalent to the total number of adders
(XOR gates) needed. Meanwhile, the number of iteration in second stage plus one
would be the total XOR gates required in the critical path. The shortest achievable logic depth of a $Q$-term equation, $P$, satisfies the following,

$$2^{P-1} < Q \leq 2^P \text{ where, } P = 1, 2, 3 \ldots \quad (6.3.1)$$

where, $P$ can be obtained if and only if the subexpressions (pattern) additions of the equations are fully arranged in a tree structure as opposed to the serial structure. As some of the terms are combination of two or more variables, the number of AND gates required has to be taken into account in both total gate count and critical path too. The CSE algorithm proposed is summarised in Algorithm 3.

**Algorithm 3 CSE Algorithm**

```plaintext
N ← 2
repeat
    repeat
        Perform exhaustive search for all possible N-term patterns in all expressions and the frequency, $f$, of each pattern is deduced.
        Determine the largest $f$ value, $f_{\text{max}}$.
        All pattern with $f_{\text{max}}$ are served as tree nodes.
        Every N-term pattern with $f_{\text{max}}$ is eliminated and renamed as a new $2N$-term pattern separately.
    until no N-term pattern is found.
    $N_{i+1} ← 2N_i$.
until only a single pattern or one nonzero term exists in each bit-level equation.
Iterate through the constructed tree and the path with the shortest depth (minimal gate count) is determined.
```

For a better understanding, we show an example of our CSE algorithm in a case study of binary linear transformation (see 6.3.2).

$$
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6 \\
y_7 \\
y_8 \\
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix} \times
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
\end{bmatrix}
\quad (6.3.2)
$$
For clarity, the output byte resultant from the binary constant multiplication is represented in bit-level equations as in the following:

\[
\begin{bmatrix}
y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8
\end{bmatrix} =
\begin{bmatrix}
x_2 + x_3 + x_5 + x_6 + x_8 \\ x_3 + x_4 + x_7 + x_8 \\ x_2 + x_6 + x_7 + x_8 \\ x_5 + x_6 + x_7 + x_8 \\ x_2 + x_3 + x_5 + x_6 + x_7 + x_8 \\ x_1 + x_4 + x_5 + x_6 + x_8 \\ x_3 + x_4 + x_5 + x_7 + x_8 \\ x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8
\end{bmatrix}
\]

Using our CSE algorithm, the first iteration chose 2 bit pattern \( x_7 + x_8 \) with the highest occurrence of \( f(x_7, x_8) = 6 \). This factor is replaced with new variable, \( A_1 \) throughout the entire bit-level equations. This is followed with \( x_5 + x_6 \) with \( f(x_5, x_6) = 4 \) and then \( x_3 + x_4 \) with \( f(x_3, x_4) = 3 \). The second stage started off with selecting 4-bit pattern of \( A_1 + A_2 \) that holds the highest occurrence of \( f(A_1, A_2) = 3 \) and is replaced with new variable \( A_7 \). The final outcome is as tabulated in Table 6.1. The performance of our algorithm, in terms of effectiveness and efficiency will be discussed next.

### 6.4 Algorithm’s Performance and Evaluation

Using MATLAB, we employ our CSE on all the eight possible binary linear transformations for a single CFA AES S-box architecture. Optimisation in binary linear transformations achieved by our proposed algorithm is as tabulated in Table 6.2. Based on the experimental results, our approach gave an average area saving of 44.09\% and an average logic depth reduction of 47.55\%, which is nearly half of the original complexity.

Furthermore, our results are benchmarked against the performance reported by an earlier CSE method proposed in [1]. Their algorithm gave a slightly larger average area saving (45.96\% on average) compared to ours. However, the critical path reduction achievable by the latter was a mere 16.35\%. In general, their algorithm was effective in promoting maximal area minimisation but the performance was traded off with a longer critical path. This will limit the highest achievable clocking frequency in the circuitry. The reason for the long critical path was attributed to the fact that the algo-
Table 6.1: Case study of optimisation in binary linear transformations using our CSE algorithm

<table>
<thead>
<tr>
<th>Our proposed CSE algorithm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = x_7 + x_8$</td>
<td>$A_7 = A_1 + A_2$</td>
</tr>
<tr>
<td>$A_2 = x_5 + x_6$</td>
<td>$A_8 = A_1 + A_3$</td>
</tr>
<tr>
<td>$A_3 = x_3 + x_4$</td>
<td>$A_9 = A_2 + x_8$</td>
</tr>
<tr>
<td>$A_4 = x_2 + x_3$</td>
<td>$A_{10} = A_3 + x_2$</td>
</tr>
<tr>
<td>$A_5 = x_1 + x_4$</td>
<td>$A_{11} = A_1 + A_6$</td>
</tr>
<tr>
<td>$A_6 = x_2 + x_6$</td>
<td>$A_{12} = A_4 + A_7$</td>
</tr>
<tr>
<td></td>
<td>$A_{13} = A_8 + x_5$</td>
</tr>
<tr>
<td>$f(x_7, x_8) = 6$</td>
<td>$A_{14} = A_4 + A_9$</td>
</tr>
<tr>
<td>$f(x_5, x_6) = 4$</td>
<td>$A_{15} = A_5 + A_9$</td>
</tr>
<tr>
<td>$f(x_3, x_4) = 3$</td>
<td>$A_{16} = A_7 + A_{10}$</td>
</tr>
</tbody>
</table>

$y_1 = A_{14} = A_4 + A_9 = (x_2 + x_3) + [(x_5 + x_6) + x_8]$  
$y_2 = A_8 = A_1 + A_3 = (x_7 + x_8) + (x_3 + x_4)$  
$y_3 = A_{11} = A_1 + A_6 = (x_7 + x_8) + (x_2 + x_6)$  
$y_4 = A_7 = A_1 + A_2 = (x_7 + x_8) + (x_5 + x_6)$  
$y_5 = A_{12} = A_4 + A_7 = (x_2 + x_3) + [(x_7 + x_8) + (x_5 + x_6)]$  
$y_6 = A_{15} = A_5 + A_9 = (x_1 + x_4) + [(x_5 + x_6) + x_8]$  
$y_7 = A_{13} = A_8 + A_4 = [(x_7 + x_8) + (x_3 + x_4) + x_5]$  
$y_8 = A_{16} = A_7 + A_{10} = [(x_7 + x_8) + (x_5 + x_6)] + [(x_3 + x_4) + x_2]$  

rithm had the pattern additions arranged in both serial and tree architectures. A clear performance comparison for both algorithms are as summarised in Figure 6.3.

In the second experiment, we investigated the advantage of having exhaustive search along with the greedy algorithm in the pattern selection process. From the results, we have shown that the performance of this hybrid approach is always better than or at the very least the same as that of a sole greedy algorithm. Using MATLAB, we implemented two versions of the proposed algorithm (i.e. with and without exhaustive search) to optimise the complexities of the binary linear transformations in a composite field AES S-box. The performances of these approaches in terms of their effectiveness (total operator elimination) and their efficiency (total time elapsed) were recorded. Their respective minimum and maximum elapsed times (in seconds), as well as the corresponding average and standard deviation of the elapsed times for both algorithms, were as tabulated in Table 6.3.
Table 6.2: Performance analysis of the complexity (total logic operator and logic depth) reduction achievable by using our CSE algorithm and the one reported in [1]. Optimisations are performed in isomorphism function and its inverse together with affine transformation for 8 possible roots $\beta$ for a composite field AES S-box.

<table>
<thead>
<tr>
<th>Logic Operator</th>
<th>Achieved XOR Gate</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta_1$</td>
<td>$\beta_2$</td>
</tr>
<tr>
<td>Without CSE</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>Our CSE</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Reduction (%)</td>
<td>40.68</td>
<td>40.68</td>
</tr>
<tr>
<td>CSE in [1]</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>Reduction (%)</td>
<td>42.37</td>
<td>42.37</td>
</tr>
</tbody>
</table>

Table 6.3: Performance in terms of time elapsed (sec) deduced from our hybrid CSE technique and our CSE with sole greedy algorithm in pattern selection. The optimisation algorithms are performed in binary linear transformations in composite field AES S-box. The simulation is repeated for 10 round runtime.

<table>
<thead>
<tr>
<th>Our Work</th>
<th>Time elapsed (sec)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta_1$</td>
<td>$\beta_2$</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.58</td>
<td>0.60</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.66</td>
<td>0.90</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.03</td>
<td>0.09</td>
</tr>
<tr>
<td>Average</td>
<td>0.62</td>
<td>0.64</td>
</tr>
<tr>
<td>Achieved XOR gate</td>
<td>35</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Greedy Algorithm</th>
<th>Time elapsed (sec)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta_1$</td>
<td>$\beta_2$</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Average</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Achieved XOR gate</td>
<td>38</td>
<td>38</td>
</tr>
</tbody>
</table>

Although the hybrid approach (the one with exhaustive search) is of a higher complexity, the exhaustive search is still a worthwhile approach given that the additional logic operator reduction gained and that the runtime achievable is within satisfactory time frame. From Table 6.3, it is shown that the hybrid approach took a longer time for completion compared to the sole “Greedy Algorithm” approach. This was expected as the
hybrid algorithm was based on the greedy algorithm and the extra time elapsed was contributed by the exhaustive search mechanism. Nevertheless, this result is justified by the additional operator reduction gained.

It may be worth noting that the same complexity was deduced for the case root equals $\beta_5$. This phenomenon occurs under three possible circumstances. First, there is only one pattern found with the highest occurrence at every stage. Second, the most optimal path derived from exhaustive search lies on the most left hand side of the tree (the first node from every branch), of which will be the same as the solution derived using sole greedy algorithm. Last, all the paths derived using exhaustive tree lead to substructure sharing of same complexity.

In fact, the true optimal result is achievable through a pure exhaustive search as proposed by Canright [39]. However, the biggest drawback of Canright’s method is that the search time is not deterministic due to the combinatorial complexity of the algorithm [39]. As development time is a critical requirement in the VLSI design cycle, it is therefore undesirable to consume excessive design time in exchange for a mere reduction of a few operators, or sometimes no additional reduction at all. As such, we attempt to maximise both efficiency and effectiveness of the algorithm by introducing CSE that combines both greedy algorithm and exhaustive search.

In the previous Chapter 5, we have derived three CFA AES S-boxes namely the Case I, Case II and Case III. Now we are ready to apply substructure sharing using our CSE

Figure 6.3: Performance comparison between our CSE and previous work as summarised from Table 6.2.
algorithm to promote further area saving in those S-boxes.

6.5 Derivation of Optimum Isomorphism Mapping for Compact AES S-box

Isomorphism function and its inverse are required for mapping the elements in \( GF(2^8) \) to its equivalents in the subfield and vice versa. These transformations are essential steps which one performed prior and post to any CFA manipulation. Both isomorphism function and its inverse, as well as affine transformation, are the applications of the binary linear transformations. The manipulations are done by the mean of vector product by an \( 8 \times 8 \) binary matrix (refer (6.5.1)),

\[
\begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  \vdots \\
  y_8
\end{bmatrix} = \begin{bmatrix}
  a_{11} & a_{12} & \ldots & \ldots & a_{18} \\
  a_{21} & a_{22} & \ldots & \ldots & a_{28} \\
  a_{31} & a_{32} & \ddots & \ddots & a_{38} \\
  \vdots & \vdots & \ddots & \ddots & \vdots \\
  a_{81} & \ldots & a_{82} & \ldots & a_{88}
\end{bmatrix} \times \begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  \vdots \\
  x_8
\end{bmatrix}
\]

(6.5.1)

with all the vector elements \( x, y \) and constant matrix \( a \in \{0, 1\} \). A potential optimisation here is to first express the multiplication in bit-level equations. Then, substructure sharing is performed to save up the common factors from the equations effectively. Take note that, in CFA, addition is referring to the bitwise XOR operation.

In this work, we adopted our proposed CSE algorithm as a solution to achieve minimal complexity (minimal additions with minimal critical path) of isomorphism and its inverse together with affine transformation. Most of the previous works only emphasised on the derivation of the optimal composite field that result in the minimal multiplicative inversion circuit. Here, we would like to point out that the optimisation of binary linear transformation in CFA AES S-box will contribute a significant amount of saving in hardware as well. Therefore, it is crucial to determine the optimum isomorphism function and affine transformation, to ensure compact construction of the CFA AES S-box as a whole. For that result, we exploit our CSE on all the eight possible binary linear transformations for a single composite field AES S-box architecture.

Using MATLAB, we had determined the complexities of the isomorphic and inverse isomorphic mapping with affine transformation. Our experiments took into account of
all the eight possible roots for Case I, Case II and Case III. From our results, the critical path for both isomorphism and inverse isomorphism are three XORs each. Meanwhile, the total gates for both isomorphism and inverse isomorphism are as depicted in Figure 6.4.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.4.png}
\caption{Total gate count (XOR) for isomorphism mapping (isomorphic and inverse isomorphic + affine) for three cases, whereby \( \beta_1 \) to \( \beta_8 \) are the root and its conjugates. The minimum isomorphism mapping attainable for Case I requires 28 XORs, 31 XORS for Case II and 29 XORs for Case III. The critical path for each is 6 XOR gates.}
\end{figure}

6.6 Discussion

In this chapter, we presented a novel CSE algorithm with the application for the binary linear transformation in CFA AES S-box. These involved the isomorphism and its inverse followed by affine transformation. The resultant total gate counts and the respective critical path for these binary linear transformations in Case I, Case II and Case III are tabulated in Table 6.4 under column ‘Isomorphism’ and ‘Inverse Isomorphism and Affine’. Our best construction Case III (refer Chapter 5), requires 13 XORs and 16 XORs for both isomorphism and its inverse with affine transformation and a critical path of 3 XORs each.

To summarize, Case I is constructed in the field of \( GF(((2^2)^2)^2) \) using polynomial basis representation with the root \( \beta = \{01011001\}_2 \) and the respective field polynomials: \( r(y) = y^2 + \{0100\}_2y + 1 \), \( s(z) = z^2 + \{10\}_2z + 1 \) and \( t(w) = w^2 + w + 1 \). Meanwhile Case II is constructed in the field of \( GF(((2^2)^2)^2) \) using normal basis representation, with the root \( \beta = \{10100000\}_2 \) and the respective field polynomials: \( r(y) = y^2 + \{1100\}_2y + 1 \), \( s(z) = z^2 + \{10\}_2z + 1 \) and \( t(w) = w^2 + w + 1 \). Overall, the best
architecture is found in Case III, which requires a total of 36 ANDs and 96 XORs and the critical path of 4 ANDs and 20 XORs. This composite field S-box is constructed in the field of $GF((2^2)^2)$ using normal basis representation. To be precise, the field is constructed using root $\beta = \{00101100\}_2$ and the respective field polynomials: $r(y) = y^2 + y + \{0010\}_2$, $s(z) = z^2 + \{10\}_2 z + 1$ and $t(w) = w^2 + w + 1$.

Analytical comparison in terms of hardware complexities between our work and several previous works are summarised in Table 6.5. From the table, it can be seen that our best construction, Case III, strikes a balance between the total gate count and the critical path gate count. To our best knowledge, this is only the second study that addresses the issue of critical path in CFA AES S-box (apart from the work by Zhang and Parhi in [40]). Our work offers minimal critical path that is close to construction reported in [40] (with difference of 1 XOR only) but with reduction of 20% in total area occupancy, which is nearly as small as the work by Canright [39], the second smallest CFA AES S-box architecture. The smallest CFA AES S-box was attributed to the work by Boyar and Peralta in [42], which consists only 32 ANDs and 83 XORs, with critical path of 4 ANDs, 21 XORs and 1 XNOR.

Note that, our CFA AES S-boxes will obtain further gate reduction as a result from the power reduction and speed enhancement schemes, which will be presented in the next chapter.
Table 6.4: Gate count and critical path of 3 novel (Cases I-III) CFA AES S-boxes. The best case of each construction are chosen after algorithmic optimisation and their respective detail complexities are tabulated under column ‘Optimised Algorithm’. This is followed by architectural optimisation and the logic gates required are precisely analyzed and noted under column ‘Optimised Architecture’. The most optimum isomorphic mapping and inverse isomorphic mapping + affine transformation are deduced. Overall complexity after taking in optimisations and isomorphism into accounts is summarised in final column.

<table>
<thead>
<tr>
<th>Case I</th>
<th>Optimised Algorithm</th>
<th>Total Gate</th>
<th>Critical Path</th>
<th>Isomorphism</th>
<th>Total Gate</th>
<th>Critical Path</th>
<th>Inverse Isomorphism and Affine</th>
<th>Total Gate</th>
<th>Critical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>14</td>
<td>3</td>
<td>14</td>
<td>3</td>
<td>36</td>
<td>118</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>Normal Basis</td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>9</td>
<td>17</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>(Norm Unity)</td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Multiplication in $GF(2^{4})$</td>
<td>9</td>
<td>23</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>23</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Multiplication in $GF(2^{2})$</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case II</th>
<th>Optimised Algorithm</th>
<th>Total Gate</th>
<th>Critical Path</th>
<th>Isomorphism</th>
<th>Total Gate</th>
<th>Critical Path</th>
<th>Inverse Isomorphism and Affine</th>
<th>Total Gate</th>
<th>Critical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Basis</td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>14</td>
<td>3</td>
<td>17</td>
<td>3</td>
<td>36</td>
<td>106</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>(Norm Unity)</td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>9</td>
<td>17</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Multiplication in $GF(2^{4})$</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Multiplication in $GF(2^{2})$</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case III</th>
<th>Optimised Algorithm</th>
<th>Total Gate</th>
<th>Critical Path</th>
<th>Isomorphism</th>
<th>Total Gate</th>
<th>Critical Path</th>
<th>Inverse Isomorphism and Affine</th>
<th>Total Gate</th>
<th>Critical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Basis</td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>13</td>
<td>3</td>
<td>16</td>
<td>3</td>
<td>36</td>
<td>96</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>(Trace + Norm Unity)</td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>9</td>
<td>17</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Multiplicative inversion in $GF(2^{7})$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Multiplication in $GF(2^{4})$</td>
<td>9</td>
<td>23</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>23</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Multiplication in $GF(2^{2})$</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The values in bold denote the complexity of the most optimum composite field AES S-Box, which correspond to Case III.
### Table 6.5: Comparison of hardware complexities between previous works and our architectures

<table>
<thead>
<tr>
<th>Work</th>
<th>Total Gates</th>
<th>Critical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AND</td>
<td>XOR</td>
</tr>
<tr>
<td>Satoh et al. [37]</td>
<td>36</td>
<td>126</td>
</tr>
<tr>
<td>Mentens et al. [38]</td>
<td>36</td>
<td>123</td>
</tr>
<tr>
<td>Canright [39]</td>
<td>36</td>
<td>91</td>
</tr>
<tr>
<td>Zhang and Parhi [40]</td>
<td>35</td>
<td>120</td>
</tr>
<tr>
<td>Boyar and Peralta [42]</td>
<td>32</td>
<td>83</td>
</tr>
<tr>
<td>Case I</td>
<td>36</td>
<td>118</td>
</tr>
<tr>
<td>Case II</td>
<td>36</td>
<td>106</td>
</tr>
<tr>
<td>Case III</td>
<td>36</td>
<td>96</td>
</tr>
</tbody>
</table>
Chapter 7

Speed Enhancement and Power Reduction in Hardware CFA AES S-box

7.1 Hardware Implementation of Optimum CFA AES S-box

In the preceding chapters, we presented compact CFA AES S-boxes available in three architectures, Case I, Case II and Case III. After a series of algorithmic optimisations and substructure sharing in the architectural level along with the CSE exploitation, the optimum architecture is found in Case III. Based on the analytical analysis, the architecture is having minimal gate counts and shortest critical path in comparison to the other cases.

In this chapter, we validate the optimality of our best case in terms of hardware implementations. Here, we develop the actual hardware implementation of the proposed CFA AES S-boxes (Cases I, II and III). The implementations can be viewed as a two-stage procedure. First, we represent all of the three proposed CFA AES S-boxes using Algebraic Normal Form (ANF). Next, we adopt a strategic fine-grained pipelining, to validate the feasibility of the proposed compact CFA AES S-boxes in achieving high throughput and low power consumption in actual hardwares.

These hardware development along with the proposed speed enhancement and power reduction optimisations will be discussed in Section 7.2 and Section 7.3 respectively. The optimisations proposed here are also reported in our research publications in [56, 70, 71, 72].
CHAPTER 7: SPEED ENHANCEMENT AND POWER REDUCTION IN HARDWARE CFA
AES S-box

7.2 CFA AES S-box in Algebraic Normal Form

Despite the fact that CFA has been well recognised as a compact solution for AES S-box design, it often results in architectures that require high dynamic power consumption. Several studies have concluded that power consumption in the composite field S-box is highly correlated with the number of dynamic hazards present in the circuit [52]. The reason being is that CFA involves several complicated sub-operations modules with different complexities. This is especially true for a S-box that works in the field $GF(((2^2)^2)^2)$. In which, the mathematical manipulations are performed through three isomorphism levels. Therefore, the respective circuit has several crossings and branched signal paths. Consequently, this will result in the differences of signal arrival times and causing several gates to switch many times per single input transition before at rest at the final result.

In order to reduce the dynamic hazards propagation in the composite field circuits, we need to keep a consistent amount of signal transitions within every clock cycle. This can be done by placing delay elements (pipeline) appropriately in the circuit such that the signal arrival times from each parallel path is consistent. However, due to the complicated crossing and branching in the composite field S-box, pipelining alone is insufficient to ensure complete consistency within the circuit.

In order to make this fine-grained pipelining efficient, we perform an additional procedure i.e. converting the complicated circuit into several logical expressions without violating the functionality of the circuit’s algorithm. In other words, the sub-operations over different isomorphism stages are replaced with direct computation modules which are expressed in ANF, consisting of only AND gates and XOR gates. With this additional procedure, pipelining can then be performed effectively, as described in the next section.

Converting the entire $GF(2^8)$ multiplicative inversion circuit into logical expressions is a tedious effort and it is definitely not an area efficient approach. We need to divide the circuit into several ANF modules in order to enable consistent pipelining without inducing excessive area increase. In doing so, we first translate all the sub-operations in $GF(2^8)$ multiplicative inversion into logical expressions separately. These include 4-bit adder, square-scaler, $GF(2^4)$ multiplier and $GF(2^4)$ multiplicative inverter.

Next, we merge some of the sub-operations and form three main parts in the $GF(2^8)$
multiplicative inversion circuit. Furthermore, we have another two more parts for isomorphism and inverse isomorphism and affine transformation respectively. All of the five ANF modules that form the proposed AES S-box are listed below as depicted in Figure 7.1:

1. Isomorphism

2. Two 4-bit adders, a square-scaler/ a squarer and a scaler, and a $GF(2^4)$ multiplier.

3. $GF(2^4)$ multiplicative inverter

4. Two $GF(2^4)$ multipliers

5. Inverse isomorphism and affine transformation

Here we would like to emphasise on the ANF formation in the $GF(2^4)$ multiplicative inverter which will be discussed in the next subsection.

### 7.2.1 ANF Representation in $GF((2^2)^2)$ Multiplicative Inversion for AES S-box

The exploitation of the ANF representation in the subfield multiplicative inversion over $GF((2^2)^2)$ is capable to enhance both area reduction and performance improvement in CFA AES S-box. This is performed through individual deduction of each of the bit’s inversion in the field element. For instance, taking element $a \in GF((2^2)^2)$ as \{a_3,a_2,a_1,a_0\}, the multiplicative inversion, $a^{-1}$, is equivalent to deducing the multiplicative inversion of its subfield’s elements individually such as, \{a_3^{-1},a_2^{-1},a_1^{-1},a_0^{-1}\}. Rather than having several circuits (sub-operations) to define the multiplicative inversion of an element in $GF(2^n)$, the algorithm is deduced into $n$ irreducible logical expressions (consisting of only AND and XOR operations) that define the multiplicative inversion of all the $n$ bits in the element. With this technique, the need for further decomposition at lower levels (multiplicative inversion and multiplication in field of $GF(2^2)$ is no longer required.

As aforementioned, the complexity of the finite field arithmetic is dependent on the coefficients of their field polynomials (refer Chapter 5). In order to promote simplicity in the arithmetic, we can choose to have either the trace or the norm of in (5.2.2) set to unity but not together. Since there are two possible values ($W$ and $W^2$) for the trace or
the norm, two detailed implementations for Case I, Case II and Case III are offered. As Case II and Case III share the same multiplicative inversion arithmetic, a total of four different ANF $GF((2^2)^2)$ multiplicative inversions are derived as tabulated in Table 7.1. The one with the less complexity from each cases will be chosen.

For benchmarking purpose, the resultant complexity will be compared to the selected previous works [47, 40, 42, 41] in Table 7.2. The smallest $GF((2^2)^2)$ multiplicative inverter is found in the work by Boyar and Peralta [42], with 5 AND gates and 11 XOR gates. However, this advantage is traded off with a relatively longer critical path of 4 AND gates and 5 XOR gates. Meanwhile, the works by Zhang and Parhi reported in both [47, 40] emphasised on achieving short critical path architecture and they considered only CFA in polynomial basis representation. In addition to that, the work by Mozaffari and Reyhani [41] focused on implementation in ASIC hardware. Therefore, the usage of AND gates and XOR gates are reduced and replaced by other logic gates that require less transistors. Their $GF((2^2)^2)$ multiplicative inverter requires 104 transistors and with a critical path of 20 transistors. On the other hand, our Case II and Case III strike a balance in both total gate counts and critical path. Note that, both $W$ and $W^2$ result in the multiplicative inversion architecture of equivalent complexity. Thus, we will just keep the original construction by using $W$ for Case I and $W^2$ for Case II and Case III (as summarized in Section 6.6 in Chapter 6).

Based on the Table 6.4 in Chapter 6, our best case, Case III is having a total gate counts of 9 ANDs and 13 XORs with critical path of 2 ANDs and 5 XORs. It is further reduced to 8 ANDs and 12 XORs with critical path of 1 ANDs and 3 XORs as shown in Table 7.2. Overall, the total gate count of our S-box in Case III is 35 ANDs and 95 XORs with critical path of 3 ANDs and 18 XORs. Our work has therefore become the AES S-box with the shortest critical path and with a relatively small area of implementation (c.f Table 6.5).

Furthermore, we can observe that the normal basis representation offers an important figure of merit ANF represented $GF((2^2)^2)$ multiplicative inversion. As seen in Table 7.1, the number of operations required for each bit inversion for Cases III and IV are more consistent compared to the other cases. Such consistency will reduce the efforts required to further pipeline the resultant architecture. As a result, it is easier to achieve a high speed low power architecture using the normal basis representation as compared to the conventional polynomial basis representation.
7.3 Fully Fine-grained Pipelined CFA AES S-box

After the ANF conversion, we employ a fine-grained pipelining methodology (Section 2.2.2, Chapter 2) to increase the CFA AES S-box’s performance in clock frequency. From Table 6.4 (see Chapter 5), we can justify that multiplication over $GF(2^4)$ is the most complex (consuming the highest amount of logic gates) subfield operation in a CFA AES S-box. In previous studies, high speed implementation of CFA AES S-box often employed pipelining scheme of which row of registers are placed in between sub-operations such as reported in [47]. Therefore, the best performance achievable through this approach is bounded to the complexity of the multiplication in $GF(2^4)$. In order to overcome this performance constraint, we propose further pipelining within the multiplication modules. This can be easily done since we have initially converted the complicated operations over several isomorphism stages into ANF.

Furthermore, the implementation platform is another important consideration factor in our pipelining scheme. Here, we implement our proposed AES S-boxes in Altera FPGA. In Altera FPGA, any abstract 4-input combinations will be fitted in a single LE. Consequently, the number of 4-input LE in the critical paths together with the routing delays will determine the minimum attainable cycle period. Therefore, the optimal placement of pipeline cut will be justified through synthesis estimates of the design. From there, we decided to place pipeline cuts after every two LEs and thus, a total of 7-stage fine-grained pipelining for all Cases I, II and III. Detailed illustration of the hardware architectures for each of our ANF-CFA AES S-box is summarised in Figure 7.1.

7.4 Implementation and Results

All of the proposed CFA AES S-boxes (Case I, Case II and Case III) had been implemented in Cyclone II EP2C5T144C6 and were synthesised using Quartus II 7.2sp3. Having the architectures clocked at 100 MHz, the timing analysis of the architectures was deduced using TimeQuest Timing Analyzer. In this work, two groups of hardware implementations were performed; our original CFA AES S-boxes and our seven stages pipelined ANF-CFA AES S-boxes.

The summary of the hardware requirements (total resource and power consumption) and their respective performances (speed and throughput) of all the three designs (for
Figure 7.1: ANF-CFA AES S-box with 7-stage fine-grained pipelining for (a) Case I, (b) Case II and (c) Case III
both groups of hardware implementations) are as tabulated in Table 7.3.

7.5 Discussion

Based on Table 7.3, our original best case (Case III) architecture requires a total of 83 LEs, with the highest achievable frequency of 122.28 MHz, consumes total power of 33.45mW and having a throughput of 0.98 Gbps. As a direct result from our proposed speed optimisation methodologies, the architecture now requires 96 LEs, with the highest achievable frequency of 436.3 MHz, consumes a total power of 34.80mW and having a throughput of 3.49 Gbps.

This additional measure imposed a considerably little amount of hardware and power increment in exchange for an increase of almost four times of its original performance. Further on, it is observed that the pipelined architecture in Case III gives the best performance improvement and having the minimum hardware requirement, compared to the hardware architecture in Case I and Case II. This is primarily owing to the optimum construction of its composite field such that its subfield arithmetic requires minimal amount of LEs and that the resulting circuitry allows optimal pipelining at the same time.

In this work, we used a specific pipelining strategy to improve the speed and the power requirements in the proposed CFA AES S-boxes. Generally, pipelining strategies are used to increase a circuit’s performance in terms of its clock frequency. Besides, pipelining strategies can also effectively reduce the dynamic hazards’ presence in a circuit, provided the circuit does not have complicated crossing and branching. This latter condition was met through converting our proposed CFA AES S-boxes into ANF modules prior to the pipelining procedures. Otherwise, the pipelining procedure will not be a cost effective process as the proposed AES S-boxes involve three levels of isomorphisms. Furthermore, the complicated nature of the CFA AES S-boxes tend to induce excessive switching activities. However, as our circuits are now converted to ANF modules, pipeline cuts can now be easily placed across the signal branches where parallel arithmetic modules in each stage has the same execution time. Thus, only a minimal increase of the overall power consumption is required in exchange for a significant improvement in the speed performance.

Overall, we have reviewed two novel architectures (Case I and II) which utilised the
benefit of having norms of field polynomials’ equal unity rather than traces. Most importantly, we have succeeded in combining both norm and trace equal to unity in different field polynomials which lead to the composite field that has minimum complexity and the shortest critical path in its multiplicative inversion. Furthermore, isomorphic mapping with minimal area cost implementation is chosen through the exploitation of our new CSE algorithm. These efforts, together with a sequence of algorithmic and architectural optimisations, lead to the construction of the optimum CFA AES S-box. We have analytically and empirically demonstrated that our CFA AES S-box in Case III is capable of leading to a smaller, lower power and higher throughput architecture.
Table 7.1: Multiplicative inversion for field element of $GF((2^2)^2)$ in ANF representation. Taking an element of four bits as $a \in GF((2^2)^2)$ as $\{a_3, a_2, a_1, a_0\}$, its multiplicative inversion is $a^{-1} = \{a^{-1}_3, a^{-1}_2, a^{-1}_1, a^{-1}_0\}$.

<table>
<thead>
<tr>
<th>Cases</th>
<th>trace / norm = $W$</th>
<th>trace / norm = $W^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE II: polynomial basis norm unity</td>
<td>$a_3^{-1} = a_3 + a_0a_2 + a_0a_1a_3 + a_1a_2a_3$</td>
<td>$a_3^{-1} = a_3 + a_0a_2 + a_1a_3 + a_0a_2a_3$</td>
</tr>
<tr>
<td></td>
<td>$a_2^{-1} = a_2 + a_3 + a_1 + a_0a_3 + a_0a_2a_3$</td>
<td>$a_2^{-1} = a_2 + a_3 + a_1a_3 + a_0a_2a_3$</td>
</tr>
<tr>
<td></td>
<td>$a_1^{-1} = a_1 + a_2 + a_1a_3 + a_0a_3 + a_0a_1a_3 + a_1a_2a_3$</td>
<td>$a_1^{-1} = a_1 + a_2 + a_3 + a_0a_3 + a_1a_3 + a_0a_2a_3$</td>
</tr>
<tr>
<td></td>
<td>$a_0^{-1} = a_0 + a_1 + a_3 + a_1a_3 + a_0a_3 + a_0a_1a_3 + a_1a_2a_3 + a_0a_3a_2 + a_1a_3 + a_0a_2a_3 + a_1a_2a_3$</td>
<td>$a_0^{-1} = a_0 + a_1 + a_3 + a_0a_3 + a_1a_3 + a_0a_2a_3$</td>
</tr>
<tr>
<td>CASES II &amp; III: normal basis norm unity</td>
<td>$a_3^{-1} = a_0 + a_1a_3 + a_0a_1 + a_0a_1a_3$</td>
<td>$a_3^{-1} = a_0 + a_1a_3 + a_0a_1a_3 + a_0a_3$</td>
</tr>
<tr>
<td></td>
<td>$a_2^{-1} = a_1 + a_2a_3 + a_0a_3 + a_0a_1a_3$</td>
<td>$a_2^{-1} = a_1 + a_2a_3 + a_0a_3 + a_1a_3$</td>
</tr>
<tr>
<td></td>
<td>$a_1^{-1} = a_2 + a_1a_3 + a_1a_2a_3 + a_0a_3a_2$</td>
<td>$a_1^{-1} = a_2 + a_1a_3 + a_0a_3 + a_1a_3$</td>
</tr>
<tr>
<td></td>
<td>$a_0^{-1} = a_3 + a_1a_2 + a_0a_3 + a_0a_2a_3$</td>
<td>$a_0^{-1} = a_3 + a_1a_2 + a_0a_2a_3 + a_1a_2a_3$</td>
</tr>
</tbody>
</table>
Table 7.2: Complexity in terms of total gate count and critical path for $GF((2^2)^2)$ multiplicative inversion in our works and the prior works.

<table>
<thead>
<tr>
<th>Work</th>
<th>trace / norm = $W$</th>
<th>trace / norm = $W^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total Gate</td>
<td>Critical Path</td>
</tr>
<tr>
<td>Zhang and Parhi [47]</td>
<td>8 AND</td>
<td>2 AND</td>
</tr>
<tr>
<td></td>
<td>14 XOR</td>
<td>3 XOR</td>
</tr>
<tr>
<td>Zhang and Parhi [40]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Boyar and Peralta [42]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mehran et al.[41]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Case I</td>
<td>8 AND</td>
<td>2 AND</td>
</tr>
<tr>
<td></td>
<td>15 XOR</td>
<td>3 XOR</td>
</tr>
<tr>
<td>Cases II &amp; III</td>
<td>8 AND</td>
<td>1 AND</td>
</tr>
<tr>
<td></td>
<td>12 XOR</td>
<td>3 XOR</td>
</tr>
</tbody>
</table>

Table 7.3: Area requirement, Timing analysis and Power consumption of FPGA implementation on Cyclone II EP2C5T144C6 device for our original CFA AES S-boxes and our seven stages pipelined ANF-CFA AES S-boxes.

<table>
<thead>
<tr>
<th>Hardware Performance/Requirement</th>
<th>CFA AES S-box</th>
<th>Pipelined ANF-CFA AES S-box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total LE (4608)</td>
<td>Case I</td>
<td>Case II</td>
</tr>
<tr>
<td>Combinatorial Function (4608)</td>
<td>81</td>
<td>80</td>
</tr>
<tr>
<td>Logic Register (4608)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>120.83</td>
<td>121.09</td>
</tr>
<tr>
<td>Total Thermal Power Dissipation (mW)</td>
<td>34.88</td>
<td>33.31</td>
</tr>
<tr>
<td>Core Dynamic Thermal Power Dissipation (mW)</td>
<td>2.64</td>
<td>1.42</td>
</tr>
<tr>
<td>Core Static Power Dissipation (mW)</td>
<td>18.02</td>
<td>18.01</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.96</td>
<td>0.97</td>
</tr>
</tbody>
</table>
Composite Field AES S-box
Construction Using Fermat’s Little Theorem

8.1 Fermat’s Little Theorem in Composite Field AES S-box

Fermat’s Little Theorem (FLT) is another potential tool used for computing the multiplicative inversion apart from Extended Euclidean Algorithm (EEA). Unlike EEA, which is commonly employed in CFA AES S-box construction, FLT-based inversion is rather uncommon in the previous works of AES.

Subsequently, this leads to the need to justify the optimality and feasibility of the FLT-based inversion as opposed to our proposed CFA AES S-box. In this chapter, we perform a detailed study on the effectiveness and the efficiency of the FLT-based inversion, the ITI algorithm for CFA AES S-box. Details on the FLT-based inversion algorithm can be found in Section 2.1.5, Chapter 2. Therefore, two CFA new AES S-boxes over $GF((2^4)^2)$ and $GF((2^2)^4)$ in polynomial basis representation are constructed using the ITI algorithm. The resultant complexity and the performance for both architectures are studied and analyzed.

8.2 The Proposed AES S-boxes Using Fermat’s Little Theorem

The CFA AES S-boxes using ITI algorithm over $GF((2^4)^2)$ and $GF((2^2)^4)$ are presented in Section 8.2.1 and Section 8.2.2 respectively.
8.2.1 Multiplicative Inversion in Field $GF((2^4)^2)$

Let $GF((2^4)^2)$ be constructed using irreducible polynomials $P(x) = x^2 + x + w^{14}$ and $Q(y) = y^4 + y + 1$. The norm $w^{14}$ is an element of $GF(2^4)$ which can be denoted as $\{1001\}_2$. For element $A(x) = \{A_1, A_0\} \in GF((2^4)^2)$, its multiplicative inversion $A^{-1}$ is computed using the four steps as follows.

**Step 1:** $A_{r-1} = A_r(x) = A^{16} = A^{2^4} = \{A_{i0}, A_{i1}\}$

$$
\begin{bmatrix}
A_{i0} \\
A_{i1}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 \\
0 & 1
\end{bmatrix} \times 
\begin{bmatrix}
A_0 \\
A_1
\end{bmatrix}
$$

$A_{i0} = A_0 + A_1$

$A_{i1} = A_1$

**Step 2:** $A' = A^{-1} \cdot A$

$$
A' = A^{-1} \cdot A \\
= h_0 + h_2 \cdot w^{14}
$$

$h_0 = A_0 \cdot A_{i0}$

$h_2 = A_1 \cdot A_{i1}$

Derivation of $h_0$ and $h_2$ involved a multiplication over the subfield, $GF(2^4)$. Instead of performing direct $GF(2^4)$ multiplication, we choose to work on their respective power over the primitive element. Multiplication of finite field elements is equivalent to addition of their respective powers over the primitive element.

As all the elements of $GF(2^n)$ form a cyclic group, the element $a_i \in GF(2^n)$ can be expressed as a multiple of a primitive element $w$, where $w = w^i$. All of the pair $(w_i, i)$ can be stored in two tables, log-table sorted on the first component $(w_i)$ and antilog-table sorted on the second component $(i)$ [2, 73]. Each of the tables took up $2^n$ of $n$ bits, resulting in a total memory requirement of 64 bits LUT for field $GF(2^4)$.

The power-element relation for the primitive element $w$ in the field $GF(2^4)$ is summarised in Table 8.1. For instance, based on Table 8.1, the primitive element $w$ with the power of 14 is equivalent to the element $\{1001\}_2$ in field $GF(2^4)$.
Table 8.1: Power-element relation for the primitive element \( w \) in the field \( GF(2^4) \)

<table>
<thead>
<tr>
<th>Power</th>
<th>Element in Polynomial Form</th>
<th>Element in Binary Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w^0 )</td>
<td>1</td>
<td>{0001}_2</td>
</tr>
<tr>
<td>( w^1 )</td>
<td>( w )</td>
<td>{0010}_2</td>
</tr>
<tr>
<td>( w^2 )</td>
<td>( w^2 )</td>
<td>{0100}_2</td>
</tr>
<tr>
<td>( w^3 )</td>
<td>( w^3 )</td>
<td>{1000}_2</td>
</tr>
<tr>
<td>( w^4 )</td>
<td>( w + 1 )</td>
<td>{0011}_2</td>
</tr>
<tr>
<td>( w^5 )</td>
<td>( w^2 + w )</td>
<td>{0110}_2</td>
</tr>
<tr>
<td>( w^6 )</td>
<td>( w^3 + w^2 )</td>
<td>{1100}_2</td>
</tr>
<tr>
<td>( w^7 )</td>
<td>( w^3 + w + 1 )</td>
<td>{1011}_2</td>
</tr>
<tr>
<td>( w^8 )</td>
<td>( w^2 + 1 )</td>
<td>{0101}_2</td>
</tr>
<tr>
<td>( w^9 )</td>
<td>( w^3 + w )</td>
<td>{1010}_2</td>
</tr>
<tr>
<td>( w^{10} )</td>
<td>( w^2 + w + 1 )</td>
<td>{0111}_2</td>
</tr>
<tr>
<td>( w^{11} )</td>
<td>( w^3 + w^2 + w )</td>
<td>{1110}_2</td>
</tr>
<tr>
<td>( w^{12} )</td>
<td>( w^3 + w^2 + w + 1 )</td>
<td>{1111}_2</td>
</tr>
<tr>
<td>( w^{13} )</td>
<td>( w^3 + w^3 + 1 )</td>
<td>{1101}_2</td>
</tr>
<tr>
<td>( w^{14} )</td>
<td>( w^3 + 1 )</td>
<td>{1001}_2</td>
</tr>
</tbody>
</table>

Using this approach, the product of field elements \( w_j, w_k \in GF(2^n) \) can be derived as,

\[
w_j w_k = \text{antilog} \left[ \log(w_k) + \log(w_k) \right] \pmod{2^n - 1}.
\]  \hspace{1cm} (8.2.1)

In summary, this technique requires determination of the power with respect to the primitive element (log conversion) that is equivalent to the operands. Next, addition is performed on the powers and the resultant value is mapped back to the field element (antilog conversion). Based on (8.2.1), one multiplication over the subfield \( GF(2^4) \) would require three LUTs.

**Step 3: \((A')^{-1}\)**

In this case, we utilise a new \( GF(2^4) \) multiplicative inversion which is similar to our proposed \( GF(2^4) \) multiplication. In general, for \( w_x \in GF(2^n) \) the multiplicative inversion operation can be expressed as,

\[
w_x^{-1} = \text{antilog} \left[ -\log(w_x) \right] \pmod{2^n - 1}.
\]  \hspace{1cm} (8.2.2)
Thus, the same log and antilog conversions (refer Table 8.1) are required here. Let $B \in GF(2^4)$ denoted as $w^x$. Its multiplicative inverse $B^{-1} = w^y$ is then,

\[
B \cdot B^{-1} = 1
\]

\[
w^x \cdot w^y = 1
\]

\[
= w^0
\]

\[
\equiv w^{15}
\]

and therefore $y = 15 - x$. After the log conversion, $y$ can be easily determined through subtraction and then followed by an antilog conversion to obtain $(A')^{-1}$. Therefore, one multiplicative inversion over the subfield $GF(2^4)$ would take up two LUTs.

**Step 4:** $(A')^{-1} \cdot A^{-1}$

Last, two $GF(2^4)$ multiplications are required to multiply $(A')^{-1}$ from Step 3 and $A^{-1}$ from Step 2.

### 8.2.2 Multiplicative Inversion in Field $GF((2^2)^4)$

Let $GF((2^2)^4)$ be constructed using irreducible polynomials $P(x) = x^4 + x^3 + x^2 + w$ and $Q(y) = y^2 + y + 1$. The norm $w$ is a primitive element of $GF(2^2)$ of which can be denoted as \{10\}_2. For element $A(x) = \{A_3, A_2, A_1, A_0\} \in GF((2^2)^4)$, its multiplicative inversion, $A^{-1}$ is computed using fours steps as stated in the following.

**Step 1:** $A^{-1} = A_j(x) = A^{84} = A^{2^2} \cdot A^{2^1} \cdot A^2$

\[
A^{2^2} = A_1(x)
= \sum_{i=0}^{3} A_1 x^{i4}
= A_0 + A_1 x^4 + A_2 x^8 + A_3 x^{12}
\]
CHAPTER 8: COMPOSITE FIELD AES S-BOX CONSTRUCTION USING FERMAT’S LITTLE THEOREM

\[
\begin{bmatrix}
A_{I0} \\
A_{I1} \\
A_{I2} \\
A_{I3}
\end{bmatrix} = \begin{bmatrix}
1 & w & 1 & 1 \\
0 & 0 & w & w \\
0 & 1 & w^2 & w \\
0 & 1 & 0 & w
\end{bmatrix} \times \begin{bmatrix}
A_0 \\
A_1 \\
A_2 \\
A_3
\end{bmatrix}
\]

\[A_{I0} = A_0 + wA_1 + A_2 + A_3\]
\[A_{I1} = wA_2 + wA_3\]
\[A_{I2} = A_1 + w^2A_2 + wA_3\]
\[A_{I3} = A_1 + wA_3\]

\[A^{2^4} = A_{II}(x)\]
\[= \sum_{i=0}^{3} A_i x^{16^i}\]
\[= A_0 + A_1x^{16} + A_2x^{32} + A_3x^{48}\]

\[
\begin{bmatrix}
A_{II0} \\
A_{II1} \\
A_{II2} \\
A_{II3}
\end{bmatrix} = \begin{bmatrix}
1 & w & 1 & w \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & w & w & 1
\end{bmatrix} \times \begin{bmatrix}
A_0 \\
A_1 \\
A_2 \\
A_3
\end{bmatrix}
\]

\[A_{II0} = A_0 + wA_1 + A_2 + wA_3\]
\[A_{II1} = A_2\]
\[A_{II2} = A_1\]
\[A_{II3} = wA_1 + wA_2 + A_3\]

93
CHAPTER 8: COMPOSITE FIELD AES S-BOX CONSTRUCTION USING FERMAT’S LITTLE THEOREM

\[ A^{2^6} = A_{III}(x) = \sum_{i=0}^{3} A_i x^{64} = A_0 + A_1 x^{64} + A_2 x^{128} + A_3 x^{192} \]

\[
\begin{bmatrix}
A_{III0} \\
A_{III1} \\
A_{III2} \\
A_{III3}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & w^2 \\
0 & 1 & w^2 & w \\
0 & 0 & w & w \\
0 & w^2 & w & w
\end{bmatrix}
\times
\begin{bmatrix}
A_0 \\
A_1 \\
A_2 \\
A_3
\end{bmatrix}
\]

\[
A_{III0} = A_0 + A_1 + A_2 + w^2 A_3 \\
A_{III1} = A_1 + w^2 A_2 + w A_3 \\
A_{III2} = w A_2 + w A_3 \\
A_{III3} = w^2 A_1 + w A_2 + w A_3
\]

Eventually, we utilised Karatsuba’s algorithm in [74] to perform multiplications of \( A^{r-1} = A^{2^2} \cdot A^{2^4} \cdot A^{2^6} \).

**Step 2:** \( A^r = A^{r-1} \cdot A \)

\[
A^r = A^{r-1} \cdot A = h_0 + (h_4 + h_5) w
\]

\[
h_0 = A_0 \cdot A_{j0}
\]

\[
h_4 = A_1 \cdot A_{j3} + A_2 \cdot A_{j2} + A_3 \cdot A_{j1}
\]

\[
h_5 = A_2 \cdot A_{j3} + A_3 \cdot A_{j2}
\]

Derivation of \( h_0, h_4 \) and \( h_5 \) involved multiplication over subfield, \( GF(2^2) \). In this case, we use direct multiplication since \( GF(2^2) \) is a relatively small field. Let \( h(x), g(x) \in GF(2^2) \) and that \( f(x) = h(x) \cdot g(x) \) is defined as,
Chapter 8: Composite Field AES S-box Construction Using Fermat’s Little Theorem

\[ f_1 = (g_1 + g_0)(d_1 + d_0) + (g_0 d_0) \]
\[ f_0 = g_1 d_1 + g_0 d_0 \]

**Step 3:** \((A^{-1})\)

Here we utilised EEA for \(GF(2^2)\) multiplicative inversion. For \(A = \{g_1, g_0\}\), its multiplicative inversion, \((A^{-1}) = \{d_1, d_0\}\) is computed as,

\[ d_1 = g_1 \]
\[ d_0 = g_1 + g_0 \]

**Step 4:** \((A^{-1}) \cdot A^{-1}\)

Last but not the least, two \(GF(2^2)\) multiplication is required to multiply \((A^{-1})^{-1}\) from Step 3 and \(A^{-1}\) from Step 2.

### 8.3 Implementation and Results

Using the multiplicative inversion algorithms as outlined in Section 8.2.1 and Section 8.2.2, two AES S-box architectures are derived (see Figure 8.1 and 8.2). Analytical comparisons of the circuit complexities between our FLT-based constructions and our best construction, Case III are summarised in Table 8.2. Both of our FLT-based AES S-boxes are implemented in Cyclone II EP2C5T144C6 and are synthesised using Quartus II 7.2sp3. The timing analysis of the architectures is deduced using TimeQuest Timing Analyzer. A summary of the hardware requirements and the performances of the designs in comparison with our Case III is tabulated in Table 8.3. For fair comparison, the Case III used here is prior to the employment of speed enhancement and power reduction optimisation. Therefore, we do not justify the required power consumption as well.
Figure 8.1: AES S-box over $GF((2^4)^2)$ and using generalised ITI algorithm

Table 8.2: Comparison of hardware complexity (total gate count) between our Case III and FLT-based inversion approach.

<table>
<thead>
<tr>
<th>Work</th>
<th>AND</th>
<th>XOR</th>
<th>LUT (64 bits)</th>
<th>Critical Path</th>
<th>AND</th>
<th>XOR</th>
<th>LUT (64 bits)</th>
<th>Adder/Subtractor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case III</td>
<td>36</td>
<td>96</td>
<td>-</td>
<td>4</td>
<td>20</td>
<td>-</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Our $GF((2^4)^4)$</td>
<td>-</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>-</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Our $GF((2^2)^4)$</td>
<td>59</td>
<td>98</td>
<td>-</td>
<td>9</td>
<td>19</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.3: Area requirement and Timing analysis of FPGA implementation of AES S-box in (i) $GF((2^4)^2)$ and (ii) $GF((2^2)^4)$ (iii) Case III

<table>
<thead>
<tr>
<th>Hardware Performance/Requirement</th>
<th>AES S-box in $GF((2^4)^2)$</th>
<th>AES S-box in $GF((2^2)^4)$</th>
<th>Case III in $GF(((2^2)^2)^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total LE (4608)</td>
<td>88</td>
<td>165</td>
<td>83</td>
</tr>
<tr>
<td>Combinatorial Function (4608)</td>
<td>88</td>
<td>165</td>
<td>83</td>
</tr>
<tr>
<td>Logic Register (4608)</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Total Register (4608)</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>90.35</td>
<td>93.83</td>
<td>122.28</td>
</tr>
<tr>
<td>Throughput (Mbps)</td>
<td>722.8</td>
<td>750.6</td>
<td>978.24</td>
</tr>
</tbody>
</table>

8.4 Discussion

From Table 8.3, CFA AES S-box over $GF((2^4)^2)$ required a total of 88 LEs, with the highest achievable frequency of 90.35MHz. Meanwhile the CFA AES S-box over $GF((2^2)^4)$ required a total of 165 LEs, with the highest achievable frequency of 93.83MHz. This is the first work to review the hardware implementation of CFA AES S-box utilising
FLT-based inversion, specifically in $GF((2^4)^2)$ and $GF((2^2)^4)$. Based on the result, AES S-box over $GF((2^4)^2)$ is smaller in size but the AES S-box over $GF((2^2)^4)$ has better performance at a higher cost. Comparatively, AES S-box over $GF((2^4)^2)$ would be a better choice as it is both efficient and cost-effective.

The major drawback of using FLT-based inversion is that it requires exponentiation and multiplication over the extensions field (Step 1). At first glance, this technique appears to be a disadvantage for field element using the polynomial basis representation. However, this is not always the case. In our multiplicative inversion over field $GF((2^4)^2)$ (Section 8.2.1), only one exponentiation $A^2^4$ is required and therefore, no multiplication over the extension field is required.

As such, Step 1 is easily accomplished by using a 4-bit XOR gate. The remaining three steps involved arithmetic in the subfield, $GF(2^4)$, which can be accomplished using log and antilog conversions. This technique requires log table, adder/subtractor and antilog table which will be more suitable for software implementation.

In our second construction, multiplicative inversion over $GF((2^2)^4)$, we show that opting for the smallest subfield may not be the best solution at times. The subfield $GF(2^2)$ operations in this construction are smaller (Step 2-4), but this advantage is traded off with a more complicated exponentiation in Step 1. Furthermore, it is not possible to choose a irreducible polynomial consisting of only binary coefficients which is irre-
ducible over $GF((2^2)^4)$. Consequently, the derivation of the exponentiation becomes increasingly complicated as well.

On the other hand, our Case III outperformed both of the FLT-based CFA AES S-boxes in terms of performance and hardware cost saving. However, we would like to point out that the FLT provides a rather simple and straightforward multiplicative inversion algorithm compared to our initial approach in Case III. Though the Case III results in smaller area of implementation, the required mathematical manipulations are more complex. In this case, the effort is worthwhile for AES works in the field of $GF(2^8)$, which is relatively small.

For Galois field of the higher order, FLT-based inversion such as ITI algorithm would deem to be more useful. The algorithm can conveniently reduce the multiplicative inversion of the extension field to the lower field, of which can then be solved using EEA, for instance. Apart from that, it is worth noting that the exponentiation required in the FLT can be performed using cyclic shifter when the elements involved are represented in normal basis.

Further elaboration on the advantages of FLT-based inversion can be found in implementation of the elliptic curve (EC) hardware cryptosystem, the second part of this thesis.
Part II

Elliptic Curve (EC) Cryptosystems

This part of thesis addresses the study on the optimisation methods for EC Hardware Cryptosystem
Elliptic Curve Cryptography

9.1 Introduction to Elliptic Curve Cryptography

As public key cryptography algorithms are applicable in any general abelian groups, one needs to select a potential group such that the protocols can be efficiently implemented in both the hardware and the software applications. In the mid 1980’s, Koblitz [75] and Miller [76] independently proposed the use of a group of elliptic curve points, defined in a finite field, for public key cryptography. Following the proposal by Koblitz and Miller, the use of elliptic curve in cryptography protocols have been a popular research area.

Arithmetic in elliptic curve provides a very unique mathematical structure for cryptography. For instance, the process of adding two points on a specific curve will result in another point on the same curve. This special feature is important in cryptography as it is inherently difficult to determine the original points that were used to get the resultant new point [77]. Elliptic curve cryptography (ECC) therefore, provides a few basic operations and rules that define how the arithmetic such as addition, subtraction, multiplication and doubling are performed.

There are two main reasons that the elliptic curve has become an attractive choice for cryptographic use. First, the required key lengths are significantly shorter than the other public-key systems, e.g. those based on integer factorisation or finite field discrete logarithm problem (DLP). One of the major issues in cryptographic protocols is finding the key length that is sufficient to maintain a certain level of security. In particular, the key length will determine the complexity of the group operations performed in the finite field.
For a general cryptographic system with a security level of $N$ bits, it will take approximately $2^n$ steps for the best algorithm to break. On the other hand, EC cryptosystems with a $n$-bit key will offer the same security level as a $N$-bit key in standard cryptosystems [78] where,

$$n = 4.91N^{1/3} \left(\log(N \log 2)\right)^{2/3}. \quad (9.1.1)$$

In other words, by using elliptic curve, the same level of security is attainable by using shorter keys. This is due to the DLP in the group of elliptic curves is computationally more complex than the other candidate groups [79]. Based on the IEEE P1363 standards specification, an EC cryptosystem with the key of 172 bits has a security of equivalent level to a RSA key of 1024 bits [80].

Second, the structure of the point group on elliptic curve allows flexibility in choosing the parameters of the cryptosystems. One can work on different fields (of which an elliptic curve is defined) in order to construct EC cryptosystem of the desired performances.

For these above reasons, ECC has been included in several security standards. The first standard proposed was the IEEE P1363 in 1994 and was initiated as a standard in 2000 [80]. This standard contains all the public-key algorithms covering ECDH, ECDSA, ECMQV and ECIES. Other relevant ECC standards are listed in the following in below [81]:

- **ISO/IEC**: In ISO 14888-3, the standard specifies the elliptic curve analogues of some ElGamal-like signature algorithms [82]. ISO 15946-2 on the other hand covers mainly on ECDSA [83].

- **ANSI X9 (Financial Services)**: In ANSI X9.62, the standard focuses on elliptic curve that deals with the Elliptic Curve Digital Signature Algorithm (ECDSA) [84]. Meanwhile, ANSI X9.63 focuses on Elliptic Curve Key Agreement and Transport Protocols [85]. Both specify formats of the used message and the list of recommended curves.

- **FIPS 186.2**: This NIST standard for digital signatures is an update of the earlier FIPS 186 (focus on DSA algorithm only). On the other hand, FIPS 186.2 specifies
both DSA and ECDSA, and provides a list of recommended curves which are mandated for the use in U.S. government installations.

- **IETF**: The standard describes a key agreement protocol that is a variant of the Diffie-Hellman protocol.
- **SECG**: The SECG standard was written by an industrial group led by Certicom. The standard mainly mirrors the content of the ANSI standards but is more readily available on the Web (http://www.secg.org).

### 9.2 Elliptic Curve

The general form of an elliptic curve is a set points which solves the Weierstrass equation, stated in Definition 9.2.1 [19],

**Definition 9.2.1.** An elliptic curve $E$ over a field $K$ is defined by the equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$  \hspace{1cm} (9.2.1)

where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$. $\Delta$ is the discriminant of $E$ which is defined as the following:

$$\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6$$

where

$$
\begin{align*}
    d_2 & = a_1^2 + 4a_2 \\
    d_4 & = 2a_4 + a_1a_3 \\
    d_6 & = a_3^2 + 4a_6 \\
    d_8 & = a_1a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_3^2.
\end{align*}
$$

The condition $\Delta \neq 0$ indicates that there are no points at which the curve has two or more distinct tangent lines and therefore the curve is “smooth”.

The points of an elliptic curve must both satisfy the curve equation and be at the same defined field, $K$. This can be expressed in the following way. Let $L$ be any extension field of $K$, the set of $L$-rational points on $E$ is,
Chapter 9: Elliptic Curve Cryptography

\[ E(L) = \{(x, y) \in L \times L : y^2 + a_1 xy + a_3 y - x^3 - a_2 x^2 - a_4 x - a_6 = 0\} \cup \{O\} \]

where \( O \) is the point of infinity. A simple example of elliptic curves defined over field of real number, \( \mathbb{R} \) is shown in Figure 9.1.

The elliptic curves with the simplified Weierstrass equations can be divided into three main categories and are summarised in Table 9.1. Note that the arithmetic rules for each category differ from one another.

9.3 Group Law

Points that satisfy the Weierstrass equation in (9.2.1) are known as the affine points. These points on the elliptic curve form an abelian group which the point addition within is defined through a simple chord-tangent method [19, 20]. This group is also used to form the EC cryptosystems.
formulas can be adapted to work on the field of any characteristic. Nevertheless, the derived formulas can be adapted to work on the field of any characteristic.

The reflection \( P \) of a point \( P = (x, y) \in E(K) \setminus \{O\} \) on the curve is given by \( \bar{P} = (x, -y - a_1x - a_3) \).

Point addition operation on elliptic curve can be classified into two separate classes; adding two distinct points and adding a point to itself, which is termed as point doubling. These two summation instructions are known as the chord-method and the tangent-method respectively, and are valid for both finite and infinite fields. The summation instruction is best explained geometrically. Brief descriptions of point addition and point doubling are explained along with the accompanying figures in Section 9.3.1.

However, for cryptographic purposes, we would be interested only in the elliptic curves defined over finite fields. Note that the graphical demonstration of the elliptic curve group law is only possible over the field of real numbers \( \mathbb{R} \). Nevertheless, the derived formulas can be adapted to work on the field of any characteristic.

### Table 9.1: The simplified Weierstrass equations

<table>
<thead>
<tr>
<th>Characteristic of field ( K )</th>
<th>Admissible Change of Variable</th>
<th>Simplified Equations</th>
<th>Determinant</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K \neq 2, 3 )</td>
<td>((x, y) \rightarrow (\frac{-3a_1}{3b_2}, \frac{3b_1 - 2a_1^3}{2b_2^3}))</td>
<td>( y^3 = x^3 + ax + b ) where ( a, b \in K )</td>
<td>( \Delta = -16(4a^3 + 27b^2) )</td>
</tr>
<tr>
<td>( K = 2 ) and ( a_1 \neq 0 )</td>
<td>((x, y) \rightarrow (a_1^2x + \frac{2}{3}a_1, a_1^3y + \frac{2a_3 + 6a_1}{3}))</td>
<td>( y^3 + xy = x^3 + ax^2 + b ) where ( a, b \in K ) and ( b \neq 0 )</td>
<td>( \Delta = b ) (non-supersingular curve)</td>
</tr>
<tr>
<td>( K = 2 ) and ( a_1 = 0 )</td>
<td>((x, y) \rightarrow (x + a_2, y))</td>
<td>( y^3 + cy = x^3 + ax^2 + b ) where ( a, b, c \in K ) and ( c \neq 0 )</td>
<td>( \Delta = c^2 ) (supersingular curve)</td>
</tr>
<tr>
<td>( K = 3 ) and ( a_1^2 \neq -a_2 )</td>
<td>((x, y) \rightarrow (x + \frac{b_1 + 2a_1}{2a_1}, y + a_1x + a_1))</td>
<td>( y^3 = x^3 + ax^2 + b ) where ( a, b \in K ) and ( a, b \neq 0 )</td>
<td>( \Delta = -a_1b ) (non-supersingular curve)</td>
</tr>
<tr>
<td>( K = 3 ) and ( a_1^2 = -a_2 )</td>
<td>((x, y) \rightarrow (x, y + a_1x + a_3))</td>
<td>( y^3 = x^3 + ax + b ) where ( a, b \in K ) and ( a \neq 0 )</td>
<td>( \Delta = -a^2 ) (supersingular curve)</td>
</tr>
</tbody>
</table>

It is important to point out that in order to perform the computation on the set \( E(K) \), we have to define the arithmetic for the point addition operation. In doing so, we need to first understand Propositions 9.3.1 and 9.3.2 [86].

**Proposition 9.3.1.** When a line \( L \) is drawn through two distinct points \( P, Q \in E(K) \), the line will always intersect at the third point \( R \in E(K) \). The same holds, when a tangent \( L \) is laid through a point \( P \in E(K) \). Then, this tangent also intersects at the third point \( R \in E(K) \), while \( P \) is counted twice.

**Proposition 9.3.2.** The reflection \( \bar{P} \) of a point \( P = (x, y) \in E(K) \setminus \{O\} \) on the curve is given by \( \bar{P} = (x, -y - a_1x - a_3) \).
9.3.1 Point Addition

Summation of two distinct points \( P_1(x_1, y_1), P_2(x_2, y_2) \in E \), is formed by drawing a straight line that passes through both points. The intersection of the line with the curve at another point is noted as \(-P_3(x_3, -y_3)\). By reflecting the point \(-P_3\) across the x-axis, we will obtain \( P_3 \), the sum of \( P_1 \) and \( P_2 \), as shown in Figure 9.2.

![Figure 9.2: Point addition in elliptic curve](image)

On the other hand, adding a point \( P_1 \) to itself (doubling), we need to use a tangent line to the curve at \( P \) and the second point of intersection of the line and the curve gives \(-P_3\). Taking its reflection over x-axis, we will obtain \( P_3 = 2P_1 \) as shown in Figure 9.3.

![Figure 9.3: Point doubling in elliptic curve](image)
The explicit formulas for calculating the coordinates of \( P_3 \) can be found by using simple geometry and algebraic calculations. The formulae for coordinates \( P_3 = (x_3, y_3) \) are noted in (9.3.1) and (9.3.2):

\[
\begin{align*}
  x_3 &= \lambda^2 - x_1 - x_2 \quad (9.3.1) \\
  y_3 &= \lambda(x_1 - x_3) - y_1 \quad (9.3.2)
\end{align*}
\]

where,

\[
\lambda = \begin{cases} 
  \frac{y_2 - y_1}{x_2 - x_1}, & \text{if } P_1 \neq P_2 \\
  \frac{3x_1^2 + a}{2y_1}, & \text{if } P_1 = P_2
\end{cases}
\]

### 9.3.2 The Point at Infinity \( O \)

By adding a point \( P_1 = (x_1, y_1) \) to its negative \( -P_1 = (x_1, -y_1) \), the straight line through them would be a vertical line with no apparent third point of intersection with the elliptic curve as shown in Figure 9.4. Based on the projective geometry concepts, the intersection does exist and it is called the *the point of infinity*, denoted as \( O \) [20]. The point lies at infinite distance along the \( y \)-axis and it acts as the identity element of the abelian group.

### 9.4 Scalar Point Multiplication

Multiplication is not directly defined on elliptic curve. Instead, we have scalar multiplication (or sometime known as the point multiplication), which is a product of a point \( P \), with an integer, \( k \) to produce another point on the curve. The operation is denoted as \( kP \) and it is performed on the highest level of elliptic curve arithmetic. Scalar multiplication is computed through an amount of repeated addition (using point addition and doubling) such as the following,

\[
kP = \underbrace{P + P + P + \cdots + P}_{k \text{ summations}} \quad (9.4.1)
\]
Chapter 9: Elliptic Curve Cryptography

![Diagram of elliptic curve with points labeled P1, -P1, and their relationships]

**Figure 9.4:** Addition of a point to its negative

Calculating elliptic curve scalar multiplication is an analogue of the exponentiation in a group and is defined in the following manner,

\[
kP = \begin{cases} 
[k - 1]P + P & : k > 0 \\
0 & : k = 0 \\
-[-k]P & : k < 0 
\end{cases}
\]

and \( kP \) has the following properties:

- \([0]P = 0, [1]P = 1.\)
- \((m + n)P = mP + nP.\)
- \(k(P + Q) = kP + kQ.\)
- \(m([n]P) = [mn]P.\)

Scalar multiplication is the fundamental to all of the elliptic curve based cryptographic schemes. Furthermore, its efficiency has a direct impact towards the efficiency of the cryptographic protocols. Therefore, research in this field mainly focused on optimising the scalar multiplication algorithms.
9.5 Elliptic Curve in Finite Fields

A finite field is identified with the notation $GF(p^m)$ for $p$ is a prime and $m$ is a positive integer. Any of such field is isomorphic to $GF(p)[x]/(P(x))$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$, $p_i \in GF(p)$ is a irreducible polynomial of degree $m$ over $GF(p)$. The major factor that determines the performance and the complexity of the ECC operations lies on the choices of $p$, $m$ and $P(x)$.

The two most commonly used finite fields for EC are the prime field and binary finite field [87]. Both constructions have been included in the standards, such as the ISO [82] and IEEE [80]. Apart from these fields, there are other feasible finite fields that have gained their popularity in elliptic curve applications. These fields are described in the following subsections.

9.5.1 Binary Field

Binary field is the field of characteristic two with ($p = 2$) and is often a favourable choice for hardware implementation of ECC. An elliptic curve $E$ over binary field $GF(2^m)$ is the set of solutions $(x, y)$ which satisfies the simplified Weierstrass equations; $E : y^2 + xy = x^3 + ax^2 + b$ where $a, b \in GF(2^m)$ and $b \neq 0$, together with the point at infinity $O$ [88]. In order to prevent the Weil Descent attack, $m$ is chosen to be a prime number.

As the elements of the subfield $GF(2)$ can be represented by logical signals 0 and 1, binary field is therefore a suitable choice to construct a fast and area efficient ECC hardware circuit. In binary field, $P(x)$ can either be a trinomial or a pentanomial, which will lead to efficient methods for extension field modular reduction.

9.5.2 Composite Binary Field

Many authors have suggested the use of composite binary field in ECC cryptographic applications [89, 79, 73, 2]. Similar to binary field $GF(2^m)$, the characteristic of the field is 2 except that $m$ is a composite number. As such, the multiplication and multiplicative inversion algorithm can be performed in the subfield which is relatively easier and faster. Therefore, composite binary field provides superior performance compared to the case of binary fields. Note that, recent attacks against ECCs over composite binary field question the feasibility of its implementation in practice. However, a careful selec-
Chapter 9: Elliptic Curve Cryptography

...tion of the field order can prevent the possibility of attack on the composite field. More details are discussed in Chapter 11.

9.5.3 Prime Field

Prime field where \( m \) of \( GF(p^m) \) is equal to 1 is one of the most generally used finite field in cryptographic application. An elliptic curve \( E \) over \( GF(p) \) (characteristic not equal to 2 or 3) is the set of solutions \((x, y)\) which satisfies the simplified Weierstrass equation, \( E : y^2 + x^3 + ax + b \) where \( a, b \in GF(p^m) \) and \( 4a^3 + 27b^2 \neq 0 \), together with a point at infinity \( O \).

For ECC implementation, the chosen order of \( p \) has to be larger than \( 2^{160} \) and has to be stored in multiple computer words. The major disadvantage for this representation is that the carries between the words must be propagated throughout the computation. Subsequently, the reduction modulo \( p \) has to be performed over several words. One of the popular methods in this context is one that based on Montgomery reduction. Though ECC of prime field is common in software implementation of ECC, it is not practical for hardware implementation due to its large area consumption resulting from the complex computation.

9.5.4 Optimal Extension Field

The Optimal Extension Fields (OEF), as introduced by Bailey and Paar [90], is a special case of finite fields. OEF of \( GF(p) \) is a field having \( p \) of the form \( 2^n \pm c \), for \( n \) and \( c \) are arbitrary positive integers, where \( \log_2(c) \leq \lfloor \frac{1}{2}n \rfloor \). Here, one would choose \( p \) such that it is an appropriate size for the multiplication instructions available on the targeted microcontroller. Furthermore, \( m \) is chosen in a way that an irreducible binomial \( P(w) = x^m - \omega \) exists, \( \omega \in GF(p) \). However, in contrast to Bailey and Paar, the work by Gura et al. in [91] has shown that the use of OEF might not necessarily lead to performance improvement or area reduction.

9.6 Affine and Projective Coordinate Systems

The points and the algebraic geometry in elliptic curve can be represented in the affine and the projective coordinate systems, which are defined below:
Definition 9.6.1. Let $K$ be an arbitrary field. The affine space $K^n$ over $K$ is the set of all $n$-tuples $(x_1, x_2, \ldots, x_n)$ with $x_i \in K \forall i = 1, \ldots, n$.

Definition 9.6.2. Let $K$ be an arbitrary field. The projective space $\mathbb{P}^n$ over $K$ is the set of all $(n+1)$-tuples $(x_1, x_2, \ldots, x_{n+1}) \neq 0$ with $x_i \in K \forall i = 1, \ldots, n+1$. On these $(n+1)$-tuples, we have the following equivalence relation:

$$(x_1, x_2, \ldots, x_{n+1}) \equiv (x'_1, x'_2, \ldots, x'_{n+1})$$

if $\lambda \neq 0$, with $\lambda \in K$ and such that $x_i = \lambda x'_i \forall i = 1, \ldots, n+1$. We can write $(x_1 : x_2 : x_3 : \cdots : x_{n+1})$ for projective points.

By using the projective coordinate system, the $(n+1)$-dimensional affine coordinate system is scaled down to $n$ dimension. In other words, the definition for an equivalence relation on the set $K^3 \{ (0, 0, 0) \}$ for the non-zero triples over $K$ is given in the following notation as: $(X : Y : Z) = (\lambda^c X, \lambda^d Y, \lambda Z)$ with $c$ and $d$ are positive integers; $\lambda \neq 0$ and $\lambda \in K$. $(X : Y : Z)$ is called projective point and $(X, Y, Z)$ are called a representative of $(X : Y : Z)$.

Both point addition and doubling in affine coordinate system involve the computation of a multiplicative inversion, which is the most complicated operation in finite field arithmetic. On the other hand, by using the projective coordinate system, we are able to define the points at infinity. With this, multiplicative inversion of the field elements can be eliminated and thus save the computation time in hardware implementation [86]. However, this benefit is traded off with more multiplications are required, which will result in larger hardware area is required.

In order to use projective coordinate system for elliptic curve points, the Weierstrass equation (see (9.2.1)) needs to be transformed. This transformation can be performed by replacing $x$ by $X/Z^c$ and $y$ by $Y/Z^d$. Therefore, if $(X, Y, Z)$ satisfies the projection equation, we can say that projective point $(X, Y, Z)$ lies on $E$ [19].

The fundamental projective coordinate systems, on which most of the recent projective systems are built of, will be discussed next in this section. Detailed information with regard of the new and enhanced projective coordinate systems are widely available in the recent literature, such as in [92, 93].
Different type of projective coordinates known for elliptic curve \( E : y^2 = x^3 + ax^2 + b \), are as the following [94]:

- **Standard projective coordinate** system is having the positive integer \( c \) and \( d \) equal to 1, with point of infinity \( O \) is corresponded to \( (0 : 1 : 0) \). The projective point \( (X : Y : Z), Z \neq 0 \), corresponds to the affine point \( (X/Z, Y/Z) \). The negative of \( (X : Y : Z) \) is \( (X : -Y : Z) \).

- **Jacobian projective coordinate** system is having the positive integer \( c = 2 \) and \( d = 3 \), with point of infinity \( O \) is corresponded to \( (1 : 1 : 0) \). The projective point \( (X : Y : Z), Z \neq 0 \), corresponds to the affine point \( (X/Z^2, Y/Z^3) \). The negative of \( (X : Y : Z) \) is \( (X : -Y : Z) \).

- **Chudnovsky projective coordinate** system is formed by representing Jacobian coordinate \( (X : Y : Z) \) as \( (X : Y : Z : Z^2 : Z^3) \).

The field operation for point addition and doubling in the above mentioned elliptic curve with different coordinate systems are summarized in Table 9.2.

**Table 9.2**: Cost of point addition and doubling for different types of coordinates in elliptic curve \( E : y^2 = x^3 + ax^2 + b \) (M-Multiplication, S-Squaring, I-Inversion).

<table>
<thead>
<tr>
<th>Coordinates</th>
<th>Addition</th>
<th>Doubling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affine</td>
<td>2M, 1S, 1I</td>
<td>2M, 2S, 1I</td>
</tr>
<tr>
<td>Standard Projective</td>
<td>12M, 2S</td>
<td>7M, 3S</td>
</tr>
<tr>
<td>Jacobian Projective</td>
<td>12M, 4S</td>
<td>4M, 4S</td>
</tr>
<tr>
<td>Chudnovsky projective</td>
<td>11M, 3S</td>
<td>5M, 4S</td>
</tr>
</tbody>
</table>

On the other hand, the known projective coordinates for elliptic curve \( E : y^2 + xy = x^3 + ax^2 + b \) are as the following [95]:

- **Standard projective coordinate** system is with the positive integer \( c \) and \( d \) equal 1 and point of infinity \( O \) is corresponded to \( (0 : 1 : 0) \). The projective point \( (X : Y : Z), Z \neq 0 \), corresponds to the affine point \( (X/Z, Y/Z) \). The negative of \( (X : Y : Z) \) is \( (X : X + Y : Z) \).

- **Jacobian projective coordinate** system is with the positive integer \( c = 2 \) and \( d = 3 \) and point of infinity \( O \) is corresponded to \( (1 : 1 : 0) \). The projective point \( (X : Y : Z), Z \neq 0 \), corresponds to the affine point \( (X/Z^2, Y/Z^3) \). The negative of \( (X : Y : Z) \) is \( (X : X + Y : Z) \).
López-Dahab projective coordinate system is having the positive integer \( c = 1 \) and \( d = 2 \) and point of infinity \( O \) is corresponded to \((1 : 0 : 0)\). The projective point \((X : Y : Z), Z \neq 0\), corresponds to the affine point \((X/Z, Y/Z^2)\). The negative of \((X : Y : Z)\) is \((X : X + Y : Z)\).

Field operation for point addition and doubling in the above mentioned elliptic curve with different coordinate systems are summarized in Table 9.3.

<table>
<thead>
<tr>
<th>Coordinates</th>
<th>Addition</th>
<th>Doubling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affine</td>
<td>1V + 1M</td>
<td>1V + 1M</td>
</tr>
<tr>
<td>Standard Projective</td>
<td>13M</td>
<td>12M</td>
</tr>
<tr>
<td>Jacobian Projective</td>
<td>14M</td>
<td>10M</td>
</tr>
<tr>
<td>López-Dahab projective</td>
<td>14M</td>
<td>8M</td>
</tr>
</tbody>
</table>

Based on Table 9.2 and Table 9.3, the complexity for the point addition and doubling will differ depending on the different types of coordinates systems used. As a result, this will affect the complexity of the scalar multiplication involved. While projective coordinate systems are well suited for high speed implementation, affine coordinate system is still more suitable for low-cost hardware implementation.

9.7 Elliptic Curve Discrete Logarithm Problem

Security level of any public key cryptosystems relies on the underlying mathematical problem. For instance, in the case of Integer Factorisation Systems (such as the RSA algorithm), the security is based on the integer factorisation problem defined as follows. Given a number \( x = p \cdot q \), determine \( p \) and \( q \), its large prime factors. On the other hand, for Discrete Logarithm Systems (such as the ElGamal encryption and Digital Signature Schemes), the security rests upon the DLP modulo \( n \) [96]. Meanwhile, the security of the elliptic curve based public key cryptosystems depends upon the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP). DLP for an abstract algebraic group can be stated in Definition 9.7.1.

**Definition 9.7.1.** Given a cyclic group \( G \) of order \( n \) and that \( a, b \in G \) with \( a \) is a generator of \( G \) and \( x \) is an unknown integer in the range \( 0 < x < n \) such that \( b = a^x \), therefore calculate \( x \).
The same can be translated for elliptic curve group which is an additive abelian group. The scalar multiplication of integer $k$ and a point on curve $P$ can be relatively straightforward, however the reverse would not be the same. In other words, calculating $k$ given $P$ and also the $kP$ is known as ECDLP which is defined in Definition 9.7.2.

**Definition 9.7.2.** Given an elliptic curve $E$, defined over a finite field $GF(q)$, a point $P \in E(GF(q))$ of order $r$, and a second point $Q \in \langle P \rangle$, determine the integer $l \in [0, r - 1]$ such that $Q = [l]P$.

Based on the current state of mathematical knowledge, the integer factorisation problem, the DLP modulo $n$ and ECDLP are believed to be intractable. Summary of the security analysis in public-key cryptosystems is found in Table 9.4.

<table>
<thead>
<tr>
<th>Public Key System</th>
<th>Mathematical Problem</th>
<th>Best known mathematical problem solving method</th>
<th>Running Times</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Factorisation</td>
<td>Given a number $x = p \cdot q$, determine the prime factors $p$ and $q$.</td>
<td>General number field sieve (GNFS): Sub-exponential</td>
<td>$O(d^{1.923}(\log n)^{1/3}(\log(\log n))^{2/3})$</td>
<td>RSA</td>
</tr>
<tr>
<td>Discrete Logarithm</td>
<td>Given number $a$ and $b$, find $x$ such that $b = a^x$.</td>
<td>General number field sieve (GNFS): Sub-exponential</td>
<td>$O(d^{1.923}(\log n)^{1/3}(\log(\log n))^{2/3})$</td>
<td>ElGamal</td>
</tr>
<tr>
<td>Elliptic Curve Discrete Logarithm</td>
<td>Given an elliptic curve $E$ and points $P$ and $Q$, find $x$ such that $Q = xP$.</td>
<td>Pollard-rho algorithm: Fully exponential</td>
<td>$\sqrt{n}$</td>
<td>EC-Diffie-Hellman</td>
</tr>
</tbody>
</table>

There are dedicated algorithms that can quickly solve the ECDLP for two small classes, namely the supersingular curves and the anomalous curves. However, one can run simple tests to check if the selected elliptic curve does not belong to both classes. Apart from that, ECDLP is known to be mathematically harder to solve than both the integer factorisation problem and the DLP modulo $n$ [96, 97]. All of the known algorithms for solving the ECDLP on the chosen group take up a full exponential time.

For this reason, the strength-per-key bit is substantially greater in EC cryptosystems [97]. ECC only requires an elliptic curve of smaller order group to achieve a security level that is comparable to other public-key cryptosystems. This means that the operand sizes required are smaller. These advantages play a vital role in environments where processing power, storage and bandwidth are constrained.

### 9.8 Elliptic Curve Based Protocol

Elliptic curves have been primarily used in several cryptographic protocols. These cryptographic protocols that are suitable for elliptic curve can be classified into three
areas: the key agreement, encryption and digital signatures. The three popular EC based protocols: Diffie-Hellman key exchange, Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Integrated Encryption Scheme (ECIES).

In general, the cryptographic protocol using elliptic curve requires generation of a private and a public key pair, or sometimes known as the cryptographic keys. The cryptographic keys are generated as the following:

- Choose an elliptic curve, $E$ defined by equation in (9.2.1) and mod $n$.
- Identify $P$, a point on the curve.
- Select $k$, an integer in the interval $[1, n - 1]$.
- Compute $Q = kP$ through scalar multiplication.
- The public cryptographic key consists of $[E, P, n, Q]$.
- The private key is $k$.

Note that the private key has to be randomly selected and to be stored in a manner that the adversary can not obtain any information concerning the key.

### 9.9 Criteria in Elliptic Curve Cryptosystem Implementation

In summary, elliptic curves have been extensively used in several cryptographic protocols defined by various standard bodies. As discussed previously, the most crucial operation required in the ECC is the scalar multiplication, $kP$ which involve point additions and point doublings. Each of the point addition and point doubling can be computed using multiplication, squaring and multiplicative inversion over finite field in affine coordinates representation. Therefore, the underlying finite field arithmetic will be the main factor to determine the performance and the efficiency of the cryptosystems. Subsequently, one needs to choose to work on the suitable finite field and at the same ensures that the resulting ECDLP is relatively hard. Overall, the efficient implementation of an EC cryptosystem has become a popular research topic in public-key cryptography.

Basically, we can look into the implementation of an EC cryptosystem in three different hierarchy levels; the field level, the elliptic curve level and the protocol level [88].
For each level, there is a number of decisions needed to be taken into account, depending on the underlying hardware and the implementation goals. These implementation criteria are summarised in Table 9.5.

**Table 9.5: Criteria in EC cryptosystem implementation**

<table>
<thead>
<tr>
<th>Hierarchy Level</th>
<th>Required Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Level</td>
<td>• Choosing the underlying finite field $GF(q)$ such that its ECDLP is intractable in order to ensure its security level.</td>
</tr>
<tr>
<td></td>
<td>• Choosing the irreducible polynomial $P(x)$ and the field elements’ representations to simplify the operation arithmetic.</td>
</tr>
<tr>
<td></td>
<td>• Selection of the finite field arithmetic algorithms for field addition, multiplication, multiplicative inversion and reduction.</td>
</tr>
<tr>
<td>Elliptic Curve Level</td>
<td>• Selection of the curve point representation type such as the affine coordinates or the projective coordinates.</td>
</tr>
<tr>
<td></td>
<td>• Choosing the efficient algorithm for point addition and doubling.</td>
</tr>
<tr>
<td>Protocol Level</td>
<td>• Selection of the appropriate encryption protocol.</td>
</tr>
<tr>
<td></td>
<td>• Selection of the algorithm for the scalar multiplication $kP$.</td>
</tr>
</tbody>
</table>

At the protocol level, the emphasis is placed on selecting the suitable encryption protocol and scalar multiplication algorithm for implementation. On the next level, the elliptic curve level, the optimisations involve careful selection of the curve representation. Different representations such as the affine coordinates or projective coordinates results in different field arithmetic required. In addition to that, optimisation in the algorithm for point addition and point doubling are essential to reduce the overall complexity. In the lowest level, the field level, the optimisation is mainly related to the underlying field arithmetic. This includes choosing the optimal field $GF(p)$, the irreducible polynomial $P(x)$, field representation, as well as the optimisation of the arithmetic algorithms.

These criteria provide high flexibility for ECC to be viable in both resource-constrained devices and high performance servers. In this work, we aim at optimising the EC cryptosystems in terms of area reduction in the field level which will be explained in both Chapters 11 and 12.
CHAPTER 10

Related Works in Elliptic Curve Cryptosystems

10.1 Previous Works in Elliptic Curve Cryptosystem

Development of the ECC can be divided into two major fields; the hardware and the software realisations of elliptic curve (EC) cryptosystems. In addition to that, public-key cryptography itself is one of the well-known applications in finite field arithmetic. Therefore, the development in the finite field architecture would have a direct influence towards the ECC research as well. These related works on both the software and hardware implementations along with the finite field architectures will be summarised in this chapter.

10.2 Software Elliptic Curve Cryptosystem Implementations

Software development in EC cryptosystems mainly focused on the optimisations of the underlying field arithmetic in order to fulfill the design requirements. These requirements involved the user defined parameters, the system’s requirements and its performance. Further details on the underlying mathematic for ECC can be found in the book written by Rosing, which is the first book dedicated to software implementation of ECC [13]. Here, we are particularly interested in the software EC cryptosystems that worked on the binary finite field.

However, software ECC implementations in binary field are relatively slow in term of performance and are commonly used in small computing devices [98]. Therefore, the
study of ECC in hardware/software (co-design) implementation and in pure hardware implementation are more actively reported in the recent literature. While the hardware ECC implementation will be discussed in Section 10.3, further information on the co-design ECC implementation can be found in the recent work by Hassan et al. [99, 100, 101].

10.2.1 Binary Finite Field

One of the first software ECC implementations was reported by Schroeppel et al. The authors implemented an EC cryptosystem on $GF(2^{155})$ for the Diffie-Hellman key exchange protocol [102]. The work was proven to have a better performance than the non-elliptic curve protocol of the similar level of security. Therefore, this work had inspired the future research in ECC software implementation and remained as a benchmark for several implementations in the later dates.

The generic arithmetic for the arbitrary binary fields, in the context of ECC, was precisely reported by Weimerskirch et al. in [103]. The authors presented explained how efficient field multiplications could be derived for ECC of the arbitrary field sizes and field polynomials. At the arithmetic level, multiplication and multiplicative inversion are the most time consuming operations in finite field. Consequently, both have received the most attention in the research of arithmetic optimisations in binary finite field. For instance, López and Dahab had presented a high speed multiplication algorithm for $GF(2^m)$ in software ECC implementations [104]. Having the field elements represented in polynomial basis, the authors proved their proposed method was significantly faster than the “shift-and-add” method.

Further discussion of the software ECC implementation over binary finite field can be found in [95]. The authors, Hankerson et al., studied the software implementation on a Pentium II 400MHz workstations of the NIST-recommended elliptic curves over binary field, $GF(2^{163}), GF(2^{233}), GF(2^{409})$ and $GF(2^{571})$. Note that the NIST binary curves are defined over $GF(2^m)$ where $m$ is the prime number.

The recent work of software ECC implementation on a tiny 8-bit embedded microcontroller was reported by Hassan and Bernaissa in [105]. The author investigated a full software solution approach for low resource and scalable implementation of ECC. A novel word-level modular reduction methodology at the arithmetic level was proposed to achieve optimisation in resource reduction. In addition to that, Mahboob
and Ikram have proposed a new method for performing binary finite field squaring in polynomial basis using a LUT which is applicable to software environment [106]. The proposed architecture is fast in term of performance but the benefit is traded off with larger code size is required. A comprehensive study on other recent software ECC implementations on processors with different word sizes is summarized in [98].

10.2.2 Composite Field

Apart from the binary curve recommended by the NIST, the composite binary field of $GF(2^m)$ where $m$ is a divisible number had received several attention in the software ECC implementation as well. Previous studies employed the CFA for it is efficient in reducing complexity of the field arithmetic in ECC.

Composite field EC cryptosystem of the form $GF((2^n)^m)$ was first proposed in the field $GF((2^8)^{13})$ by Harper et al. [89]. It was one of the first ECC software implementations using general purpose processor, where LUTs were used to speed up the finite field multiplication. As a result, the authors managed to implement the ElGamal cryptosystem that achieved an encryption rate of 2 Kbits/sec in software.

Soon after, several works in composite field $GF((2^{16})^{11})$ for EC cryptosystems were also reported independently in [79, 73, 2]. These prior studies shared a common feature where the arithmetic were performed using polynomial basis representations and the LUTs were utilised in the subfield arithmetic. The major difference laid on the chosen multiplicative inversion algorithm. De Win et al. employed exponentiation based inversion algorithm in [73]. On the other hand, Guajardo and Paar utilised Euclidean based inversion algorithm for their work in [2].

10.3 Hardware Elliptic Curve Cryptosystem Implementations

Hardware realisation of EC cryptosystems often results in higher performance and security level compared to the software implementations. However, this advantage comes at the expense of higher cost and constrained flexibility. Therefore, one of the fundamental issues arisen in the development of the hardware EC cryptosystems is to choose the appropriate efficiency-flexibility trade off in the design. For instance, in network server applications, high speed public-key cryptosystems are essential. On the other hand, fast development time with the freedom to customise the design accord-
Chapter 10: Related Works in Elliptic Curve Cryptosystems

According to the vendor preferences and the security level are required in other applications. Subsequently, there is a classification of work, in relation to the level of efficiency and flexibility, for EC cryptosystems. In particular, these classifications are subjected to the issue of reconfigurable and dedicated hardware, generator, versatility and general purpose processor [107]. The studies that contributed in each of the classifications are briefly explained in the following.

10.3.1 Reconfigurable FPGA and Dedicated ASIC

Hauck et al. had presented a fully hardwired asynchronous ASIC for EC cryptosystems in [108]. The design utilised the asynchronous wave pipelines (AWPs) techniques, which allowed fine-grained pipelining to be done without imposing much of the latency. Another ASIC implementation of EC cryptosystems, which performed parallelism to maximise the use of function units, was reported by Sozzani et al. [109]. The proposed EC coprocessor provides a fast computation in the synchronous polynomial basis multiplier for efficient scalar multiplication $kP$.

On the other hand, Orlando and Paar [110] initiated the design of reconfigurable EC processor over the finite field $GF(2^{167})$ on FPGA. The presented architecture consisted of an optimised bit-parallel squarer, a digit-serial multiplier, and two programmable processors. In addition to that, Sandoval and Uribe proposed a hardware architecture that performs three different ECC algorithms, namely the elliptic curve Diffie-Hellman (ECDH), elliptic curve digital signature (ECDSA) and elliptic curve integrated encryption scheme (ECIES) [111]. Other EC cryptosystems implemented on reconfigurable platform can be read in [112, 113].

10.3.2 Generators of Elliptic Curve Processor

Requiring only the system’s parameters, a generator based architectures can be automatically synthesised with scripts. In other words, the design is fully parameterisable before synthesis took place. This feature enables designers to rapidly explore and implement a design with the best trade-offs in speed, size and level of security required by the users.

An example of such architecture can be found in [114]. The authors, Kerins et al., proposed a methodology that accepts a number of design parameters to specify the speed/security requirements. Therefore, a HDL description suitable for the synthesis
will be automatically generated into an FPGA.

Furthermore, the architectures that worked on ONB multipliers and focusing on hardware/software partitioning problem were reported in [115, 116]. Cheung et al. presented a system-on-chip (SoC) architecture for EC cryptosystems on a reconfigurable hardware [115]. In their work, a design generator was used to generate a parameterisable blocks for the reconfigurable SoC architecture. Telle et al. had similarly presented a methodology to generate the hardware design for EC cryptosystems over $GF(2^m)$ in ONB [116]. The proposed generator is capable of generating a customised hardware design based on the user-defined requirements.

### 10.3.3 Versatile Scalar Multiplier

Versatile processor manages to accommodate field of various sizes and irreducible polynomials without reconfiguration. It would be very useful in heterogeneous environments, where the desired performance, vendor preferences, security requirements and processor capabilities are unknown [107].

Potgieter and van Dyk presented two scalable FPGA implementations of elliptic curve scalar multiplication in [117]. Scalability here refers to being able to change both the field size and the elliptic curve parameters without reprogramming the hardware. Despite of the scalable nature of the hardware, optimisation measures were taken into account during the design process to ensure the near-optimal performance for the given set of algorithms.

In another similar example, Kerins et al. reported a hardware architecture for flexible EC cryptographic processor for arbitrary elliptic curves over $GF(2^{255})$ [118]. All of the underlying field operations had the same calculation time and was suitable for implementation in a constrained environment. On the other hand, Daneshbeh and Hasan proposed a scalable unidirectional bit serial systolic architecture for elliptic curve scalar multiplication [119]. The architecture was able to be performed on finite field of any dimension and any defining irreducible polynomial.

### 10.3.4 General Purpose Processor

The use of generic datapath can provide a wider range of flexibility compared to a versatile processor. However this advantage comes at the expense of performances
degradation. Generic purpose processor is especially attractive while dealing with different cryptosystems, which requires different kind of arithmetic of different field sizes. The general purpose processor usually works on a 32-bit or 64-bit datapath.

Itoh et al. had developed a $GF(p)$ general processor that was suitable for RSA, DSA and ECDSA on a DSP TMS320C6201 [120]. The authors proposed an improved modular of Montgomery multiplication, which enabled efficient pipeline processing. Furthermore, another optimised algorithm for elliptic curve implementation in CalmRISC with MAC2424 math coprocessor was presented by Chung et al. in [121]. The authors employed fast finite field arithmetic and elliptic curve algorithm that were useful for embedding cryptographic functions on a high performance device. As a result, most of the instructions can be accomplished within one clock cycle.

On the other hand, two 64-bit dual-field processors that support RSA, DSA, ECDSA and DH were reported in [122, 123]. Both works utilised the Montgomery multipliers and were capable to compute integer and binary polynomial arithmetic over $GF(p)$ and $GF(2^m)$. Satoh and Takano presented a Montgomery multiplier with an optimised data bus and an on-the-fly redundant binary converter to boost the throughput of the elliptic curve scalar multiplication [122]. Meanwhile, Erbele et al. employed a novel dual-based field multiplier on a modified carry-save adder (CSA) tree [123]. In particular, the architecture proposed in [122] was deployed on ASIC platform while the architecture in [123] was implemented on FPGA with ASIC extrapolations.

10.4 Finite Field Arithmetic in Hardware Cryptosystem

Most of the published works on finite field architecture can be extended for the use in EC cryptosystems. Here, we emphasised on the architecture that worked in the binary field $GF(2^m)$ in hardware implementation. For instance, Barua and Sengupta provided an overview of architectures in $GF(2^m)$ [124]. The authors presented an array-based circuits for normal basis multiplier, polynomial basis squarer, multiplicative inverter and exponentiation algorithm over the binary finite field $GF(2^m)$. Meanwhile, Paar had reported a detailed survey of the implementation options of finite field arithmetic specifically for EC cryptosystems [125].

These works in finite field optimisation had a significant contribution in the development of EC cryptosystem. The related works on finite field architecture for crypto-
graphic applications can be categorised into several classifications, which will be explained in the following subsections.

### 10.4.1 Binary Finite Field Operations

Scalar multiplication in EC cryptosystem involves multiplicative inversion and multiplication, which are the most hardware cost expensive operations in the finite field. Efficient multiplier for binary finite field of $GF(2^m)$ can be found in [126, 127, 128]. Mastrovito developed a parallel polynomial basis multiplier for $GF(2^m)$ that was both speed and area efficient [126]. The proposed multiplier was proven to be able to exhibit a high degree of modularity and regularity. Another parallel multipliers of the field $GF(2^m)$ which were based on irreducible All-One-Polynomials (AOP) and equally spaced polynomials (ESP) were presented by Hasan et al in [127]. The structures enabled fast implementation of squaring and multiplication algorithms such that it made fast exponentiation and multiplicative inversion possible.

On the other hand, several finite field multiplicative inversions are also available in the literature. Gutub et al. proposed a scalable and unified architecture for Montgomery inversion hardware for both $GF(p)$ and $GF(2^m)$ [129]. The architectures were intended to be scalable such that it allows computation of long precision numbers to be done in a repetitive way. Apart from scalability, domain-specific reconfigurable multiplicative inversion datapath over $GF(2^m)$ were also reported in [130, 131]. Both architectures can be adapted to various area/performance constraints and finite field sizes.

In the later dates, fast multiplicative inversion algorithm has gained several attention as well. Rodríguez-Henríquez et al. derived a novel parallel formulation of the standard Itoh-Tsujii algorithm for multiplicative inversion over $GF(2^m)$ [132]. In their work, a special class of irreducible trinomials were used in order to achieve the maximum performance.

In another development, Deng et al. proposed a fast hardware implementation of multiplicative inversion in $GF(2^m)$ using ONBI basis [133]. The authors deployed the Sunar-Koc multiplier and the Itoh-Tsujii inversion algorithm in their work. Though the multiplicative inverter costs a relatively larger area compared to [129, 130], it can perform perfectly for the high speed applications. Later on, Wei presented a modular inversion over $GF(2^m)$ using FLT [134]. A parallel modular multiplication algorithm and a cascade modular square block were employed in order to reduce the time re-
required for modular exponentiation. The authors showed that only 0.5µs is required to compute a modular inversion over $GF(2^{191})$.

10.4.2 Basis Representations

Another classification of the finite field architectures is with respect to the basis representations of the field elements [135]. These include the polynomial basis, normal basis and dual basis. So far, only polynomial and normal basis representations have been extensively used for cryptographic applications. Though all of the representations are isomorphic to one another, they provide different computational complexities in certain applications and over different computational environments.

Arithmetic in polynomial basis is conceptually simpler to implement but modular reduction following the multiplication operation is required. Therefore, in certain cases, normal basis would appear to be more favourable in computational point of view for it allows efficient exponentiation [20]. General comparison of multipliers of different basis representations can be studied in [136, 137]. The studies concluded that both polynomial basis and dual basis multiplier share similar behaviour and require smaller area of implementation. Normal basis multiplier on the other hand, is more effective in finding the multiplicative inversion element, squaring and exponentiation in finite field arithmetic. However, this comes with the price of larger area of consumption, especially if the larger field is used. Nevertheless, the area imposed can be effectively reduced through algorithmic optimisation and it is attractive for bit serial application which is not reported in the studies.

Furthermore, comparison between polynomial basis and normal basis with reference to public-key cryptosystem was reported in [138]. The authors deduced that normal basis designs are indicated in particular situations where there are substantially less multiplications than squaring. This can be achieved through special exponentiation algorithms. The ONB representation is special case of normal basis representation. It is an attractive option in this context because of their moderate complexity. Unlike polynomial basis and normal basis, ONB only exists in certain field sizes.

10.4.3 Serial and Parallel architecture

Finite field architecture can be realised into bit serial (one output bit per clock cycle) and bit parallel (all output bits per clock cycle) implementation. Bit parallel architecture
provides better performance in term of speed, however it consumes larger chip area in hardware implementations.

Therefore, bit serial multipliers, having space complexity of \( O(m) \) for arithmetic in \( GF(2^m) \), is a more practical solution compared to the bit parallel multiplier of \( O(m^2) \) complexity bound. However, more recently, new types of bit parallel architectures with complexities below \( O(m^2) \) have been proposed in [139, 140, 141]. These architectures are based on either the multiple field extensions or the fast convolution methods.

Paar introduced a composite field \( GF((2^n)^m) \) (with \( k = nm \)) parallel multiplier which gave a significant improvement to the conventional multiplier [139]. The multiplier was highly modular and having a complexity of \( O(k \log_2 3) \), it was suitable for VLSI implementation. Paar et al. further developed a new class of multiplier for \( GF((2^n)^4) \), which was a modified version of the KOA in [140]. Through the optimisation of field polynomials of degree four, the last stage of the KOA and the modulo reduction can be combined and saved the computation time and implementation area.

In addition to that, Paar et al. had developed architectures of the hybrid-type (partially serial and partially parallel) in [135]. The underlying idea was to apply bit parallel architectures for the arithmetic in the subfield, \( GF(2^n) \) and a serial approach to the extension field arithmetic \( GF((2^n)^m) \). This mixed parallel-serial (hybrid) approach was claimed to be able to lead to a faster and more efficient hardware implementations.

10.5 Summaries Notes

As reported in this chapter, there are several studies that lead to the development trend in the EC cryptosystems. In the software development, the work mostly revolved in algorithmic optimisation along with the usage of LUTs, in order to reduce the complexity of the underlying field arithmetic and thus leads to saving the computation time.

Hardware realisation of EC cryptosystems on the other hand are mostly emphasised on scalability and flexibility in design. Given the required performance, vendor preferences, security requirements and processor capabilities, the desired cryptosystem can be generated directly and thus shorten the development time. For further optimisation, the finite field architecture reported in the literature, such as the multiplier in field \( GF(2^n) \) can be employed to enhance the performance in cryptosystems. However, not much of the reported studies emphasised on hardware size reduction in elliptic hard-
In the next chapter, we shall proceed to present our work on the hardware development for EC cryptosystem. Rather than focusing on the flexibility in design, we emphasise on deriving small area and low cost cryptosystem, with an acceptable speed performance. In short, we aim at optimising the multiplicative inversion over the finite field (the most expensive operation) in EC cryptosystems. We will employ CFA, which has only been applied in the software ECC realisation. Unlike the previous works, we promote the employment of combinatorial circuit where the need of LUTs is eliminated completely. Therefore, our multiplicative inverter is suitable for the area-constrained hardware cryptographic architecture.

With the detailed literature review given in this chapter, we are ready to move on to the detailed algorithm and implementation issues involved in EC cryptosystems design, of which will be presented in the next chapter.
Chapter 11

Composite Binary Field for Elliptic Curve Hardware Cryptosystem

11.1 Optimisation in Elliptic Curve Hardware Cryptosystem

As discussed in the preceding chapters, optimisation in the EC cryptosystems can be viewed in three different hierarchy levels; the field level, the elliptic curve level and the protocol levels. In this study, we work on field level optimisation to propose area cost reduction for hardware implementation of the EC cryptosystem. We introduce a new composite field that is both secure and having minimal complexity in ECC. In addition to that, we derive a compact and efficient composite field multiplicative inverter for EC hardware cryptosystem.

11.2 Field level Optimisation in Elliptic Curve Cryptography

In the literature, EC hardware cryptosystems were commonly designed for arbitrary finite field (either prime field or binary field) instead for a specific field. The design efforts were mostly subjected to the issue of reconfigurability and scalability of the ECC processor in achieving high speed or compact cryptosystem. On the contrary, very little studies had emphasised on the algorithmic optimisation in EC hardware cryptosystem.

Note that ECC works in the finite field. The complexity of the finite field arithmetic will determine the total resources required and the performance of the cryptosystem. Thus, optimisation at the field level is a promising approach to promote area reduction and performance enhancement in EC hardware cryptosystems. In general, field level
optimisation for EC cryptosystem involved three major steps [88] (also refer Table 9.5):

1. Choosing the finite field $GF(q)$ such that the underlying arithmetic is both efficient and intractable.

2. Choosing the irreducible polynomial $P(x)$ and the field elements’ representations that will simplify the operations’ arithmetic.

3. Selecting the efficient finite field arithmetic algorithms for field addition, multiplication, multiplicative inversion and reduction.

Subsequently, the most essential step in constructing a compact and efficient EC hardware cryptosystem, is to choose the optimal field along with its respective irreducible polynomial and basis representation for ECC computation. The optimality that we seek for is the field that is both secure and result in compact arithmetic operations which are feasible for small cryptographic applications. Composite field is chosen here as it provides greater computational efficiency compared to other finite field.

However, the work of Frey, Galbraith, Gaudry, Hess and Smart in [142, 143, 144] have cast doubt on the security offered by the composite field as compared to those field defined in the standards. In this study, we provide a counter argument that ECC defined in composite field is not necessarily weak. Further discussion on the CFA in ECC, in terms of the security and the complexity it offers will be discussed in the following sections.

### 11.3 Elliptic Curve Discrete Logarithm Problem in Composite Field

The formal definition of ECDLP can be found in Definition 9.7.2, Section 9.7 in Chapter 9. Basically, the intractability of the ECDLP will determine the level of security of the EC cryptosystems [145, 146].

The best algorithm known for solving the general ECDLP, the Pollard’s rho method [147], took up an exponential expected running time of $\sqrt{\pi r}/2$ (with $r$ is the number of elements) point additions in solving the ECDLP in ECC [145, 146]. As composite field is an attractive choice for the hardware realisation of EC cryptosystem, it is important to determine a composite field that is not weak for ECC. Here, weak is in the sense that
any instance of the ECDLP for the field can be solved in a significantly lesser time than it takes Pollard’s rho method to solve the hardest instances.

In 2000, Gaudry, Hess and Smart (GHS) [142] showed that the Weil descent attack methodology proposed by Frey [148] can be used to reduce the ECDLP in the elliptic curves over composite field to an instance of DLP in the Jacobian of a hyperelliptic curve over $GF(2^N)$. Only for the case of $N \in [160, 600]$ is prime, $GF(2^N)$ is proven to be secure from the GHS attack [149]. In other words, the use of elliptic curves over $GF(2^N)$ with $N$ is a composite number is not recommended.

For this reason, various investigations had been conducted to confirm the vulnerability caused by using the composite field in ECC [143, 144, 150, 146, 145]. The work reported in [143, 144, 145] provided some evidences that the curves over the fields of composite degree divisible by 3, 4 and 5 should be avoided in cryptographic applications. However, their deductions were proven for curves of a small range only.

Menzes and Teske further explored the cryptographic implications of the generalised GHS attack (by Hess in [151]) in binary composite field, $GF(2^N)$ [146]. In their work, they concluded that the field where $N$ is not divisible by 3, 5, 6, 7 or 8 are not (potentially) weak under Hess’s generalised GHS attack. In the later date, the applicability of the GHS attack on the ECDLP for elliptic curves over $GF(2^N)$ for composite $N \in [160, 600]$ was precisely analyzed by Maurer et al. in [150]. The elliptic curves over composite field that are susceptible to the GHS attack were identified and listed in their paper. Therefore, this allows us to select the composite field that is not weak under GHS attack.

### 11.4 Complexity of the Composite Field

Complexity of finite field arithmetic is dependent on several field construction factors. These factors include the order of the field and the field representations (e.g. the type of irreducible (field) polynomials and basis representation used) (refer Section 5.1 in Chapter 5). These field constructions factors need to be selected wisely in order to derive an efficient EC hardware cryptosystem. As discussed in Chapter 10, the previous works only studied two-level composite field for EC cryptosystems [89, 79, 73, 2]. As an improvement to the previous works, we extend these studies by employing three-level composite field, i.e. $GF((2^n)^m^1)$ for EC cryptosystem.
In addition to that, the arithmetic of the polynomial basis represented elements are often followed by the modulo reduction using the irreducible polynomial $P(x)$. For instance, the complexity of the modulo reduction varies based on the coefficient chosen in $P(x)$. The complexity of a hardware multiplier can be reduced if the irreducible polynomial is a trinomial or is in the form of AOP [152]. Moreover, the exponentiations in the field $GF((2^m)^n)$ with binary irreducible polynomial is substantially lower than that of a general multiplications [153]. Note that it is possible to choose $P(x)$, the irreducible polynomial of degree $m$ over $GF(2^n)$, with binary coefficient if $gcd(n, m) = 1$ [8].

On the other hand, normal basis representation is often a preferred choice over the polynomial basis representation in hardware implementation. The reason is that in normal basis representation, squaring of an element can be performed through a simple cyclic shift. Furthermore, ONB, the special case in normal basis representations, manages to further reduce the complexity of the complicated normal basis multipliers [12]. Based on these advantages, we choose to work on elliptic curve over composite field in ONB representation.

For security purposes, the extension degree $l$, has to be a considerably large prime number and therefore, ONB type II representation is sought here. Meanwhile, $n$ and $m$, are chosen to be relatively smaller in order to simplify the computation in ECC. Practically, the common choices for the composite binary field $GF(2^N)$ are in the range of $N \in [160, 600]$ [146]. Our main objective is to cater the resource-constrained cryptographic applications and ensure that our composite field EC cryptosystem is not vulnerable towards the generalised GHS attack. Therefore, we have chosen our field $GF(q^l)$ where the subfield is of the size $q = (2^n)^m = (2^2)^2 \approx 2^4$ and $l = 41$.

### 11.5 Area minimisation in Scalar Multiplication

Following the selection of the optimal composite field, we explore optimisation in the field level to search for the potential area minimisation for scalar multiplication, the most crucial and yet the most complicated operation in ECC. Scalar multiplication in affine coordinate system involves a repetition of point additions and point doublings, would require multiplicative inversions over the finite field [19].

We can avoid the use of the multiplicative inversion operation by using the projective
coordinates systems. However, this benefit is traded off as more multiplications are required. In fact, multiplication is a relatively complex and resource consuming operation as well. Instead of using the projective coordinates systems, we present a new efficient multiplicative inversion circuit through the exploitation of CFA for EC hardware cryptosystem.

Based on the literature surveys, two main approaches for the multiplicative inversion over finite field were employed in ECC. These are namely the EEA and the FLT (refer Chapter 2.1.5 in Chapter 2). For instance, Win et. al. applied the optimised version of EEA, the Almost Inverse Algorithm [102] over $GF((2^{16})^{11})$ in [73]. On the other hand, the FLT-based inversion, realised as the ITI algorithm [21, 153] for polynomial basis representation over $GF((2^{16})^{11})$ was proposed by Guajardo and Paar [2]. However, all of these prior studies were designed with the software implementation in mind, where look-up tables (LUTs) were utilised in the computation of the subfield arithmetic.

Apart from consuming large silicon areas, the use of LUTs in hardware realisation will introduce an unbreakable timing delay, which in turn will predominate the minimum clock rate attainable by the final cryptosystem. To avoid this delay while saving on silicon areas, one can replace the LUT with using only the combinatorial logic components for the final computation. This approach also allows the designer to further improve the clock rate through inserting standard pipeline registers at appropriate points of the system.

This is the main reason we have chosen to work on three-level composite field, i.e. $GF(((2^n)^m)^l)$ as opposed to the two-level composite field. By performing an addition level of computation in the subfield, the required arithmetic can be constructed by using logical AND and XOR gates only. Therefore, we propose a novel three-level composite field multiplicative inverter using the Itoh and Tsujii inversion (ITI) algorithm in optimal normal type II basis representation (ONBII) to eliminate the need for LUTs completely.

In addition to that, each multiplicative inversion requires several multiplications over the extension field and the subfield. Therefore, we propose a series of algorithmic optimisations in the subfield $GF((2^2)^2)$ operations, and also a hybrid serial-parallel Sunar-Koc multiplier in the extension field, in order to promote maximum area reduction in the design. Detailed description of our composite field multiplicative inverter, along with the optimisations in the multiplication will be presented in the next chapter.
11.6 Discussion

To summarise, optimisation in the field level involves three major steps. This includes choosing the finite field that is both efficient and intractable. We also need to consider the choices of the irreducible polynomial as well as the field element’s representation that would result in maximum reduction in the underlying arithmetic. Next, we are required to select the optimal finite field algorithms, specifically the multiplicative inversion and the multiplication that are suitable for EC hardware cryptosystem.

Through series of survey and consideration as reported in this chapter, we will be working on three-level composite field, $GF((2^2)^2_{41})$ in the ONBII representation. Using the both ITI algorithm for the multiplicative inversion and fast Sunar-Koc multiplication algorithm, our compact and low cost multiplicative inverter for EC hardware cryptosystem will be precisely presented in the next chapter.
Efficient Composite Fields

Multiplicative Inversion over 

\( GF(((2^n)^m)^l) \)

12.1 Itoh and Tsujii Inversion Algorithm for Multiplicative Inversion over \( GF((2^2)^41) \)

According to FLT, with \( A \in GF((2^n)^m) \) and \( P(x) \) is an irreducible polynomial defined over \( GF((2^n)^m) \), one can obtain 

\[ A^{2^{nm} - 1} = A \cdot A^{2^{nm} - 2} = 1 \mod P(x). \]

Therefore, we can determine the multiplicative inversion of \( A \) as;

\[
A^{-1} = A^{2^{nm} - 2} \\
\equiv (A^{2^{q-1}-1})^2
\]

with \( q - 1 \) as;

\[
q - 1 = \sum_{i=1}^{l} 2^{k_i}, \text{ where } k_1 > k_2 > \cdots > k_l
\]

Using both (12.1.1) and (12.1.2), \( A^{-1} \) can be derived using the chain addition as follows:
\[ rCl(A^{2^1 \cdot -1})^2 = [(A^{2^{y_1} \cdot -1})(A^{2^{y_2} \cdot -1}) \ldots (A^{2^{y_l} \cdot -1})
(A^{2^{y_l} \cdot -1})^{2^{2^{l} \cdot 2^{l} \cdot \ldots \cdot 2^{y_l}}}^2 \] (12.1.3)

In this work, we employ the ITI algorithm (Theorem 2.1.2, Chapter 2) which reduces the multiplicative inversion in the extension field \( GF(q^l) \) to the multiplicative inversion in its subfield, \( GF(q) \) [21, 153]. The ITI algorithm can be accomplished in four major steps, of which each will be discussed individually in the following subsections. Hereafter, we denote our field as \( GF(q^l) \) with \( q = (2^2)^2 \) and \( l = 41 \).

### 12.1.1 Step 1: Exponentiation of \( A^{r-1} \in GF(q^l) \)

Referring to the multiplicative inversion algorithm in Theorem 2.1.2, we need to first determine the computation of \( A^{r-1} \) where \( A \in GF(q^l) \). Take note that the exponent \( r - 1 \) can be expressed as a sum of powers,

\[
r - 1 = \frac{ql - 1}{q - 1} - 1 = q + q^2 + q^3 + \ldots + q^{l-1} \quad (12.1.4)
\]

Using (12.1.3) and (12.1.4), we can compute the exponentiation of \( A^{r-1} = A^{q^0 + q^3 + q^8 + \ldots + q^2 + q} \) through a series of repeated power raising and multiplications such as follows:

\[
\begin{align*}
A^{q^2} &= (A^q)^q \\
A^q \cdot A^{q^2} &= A^{q^2 + q} \\
(A^{q^2 + q})^2 \cdot (A^{q^2 + q}) &= A^{q^4 + q^3 + q^2 + q} = A^{\sum_{i=1}^{4} q^i}_4 \\
(A^{\sum_{i=1}^{4} q^i}_4)^4 \cdot A^{\sum_{i=1}^{4} q^i}_4 &= A^{q^8 + q^7 + \ldots + q} = A^{\sum_{i=1}^{8} q^i}_8 \\
(A^{\sum_{i=1}^{8} q^i}_8)^8 \cdot A^{\sum_{i=1}^{8} q^i}_8 &= A^{q^{16} + q^{15} + \ldots + q} = A^{\sum_{i=1}^{16} q^i}_{16} \\
(A^{\sum_{i=1}^{16} q^i}_{16})^{16} \cdot A^{\sum_{i=1}^{16} q^i}_{16} &= A^{q^{32} + q^{31} + \ldots + q} = A^{\sum_{i=1}^{32} q^i}_{32} \\
A^{r-1} &= (A^{q^{32} + q^{31} + \ldots + q})^{q^8} .
\end{align*}
\] (12.1.5)
Chapter 12: Efficient Composite Fields Multiplicative Inversion Over \( GF((2^n)^m)^l \)

Subsequently, the computation of \( A^{-1} \) requires \([21]\);

\[
\begin{align*}
\text{#MUL} &= \lfloor \log_2 (l - 1) \rfloor + \text{HW}(l-1) - 1 \\
\text{#EXP} &= l - 1
\end{align*}
\] (12.1.6)

The complexity to compute \( A^{-1} \) using addition chain (see (12.1.5)) is found to be 6 multiplications in \( GF(q^{41}) \) and 40 exponentiations to the power of \( q^{th} \), which agrees with (12.1.6). Exponentiation to the \( q^{th} \) power in normal basis requires only \( q \) cyclic shifts. This is significantly more efficient than the one using polynomial basis representation of which modular reduction is required [2]. On the other hand, the \( GF((2^2)^{41}) \) multiplier will be described in Section 12.2.

12.1.2 Step 2: Multiplication of \( A \) and \( A^{-1} \) that yield \( A^r \in GF(q) \)

In the second step, multiplication of the two operands, \( A \) and \( A^{-1} \in GF(q^{41}) \) will result in \( A^r \in GF(q) \). Therefore, we need a specific multiplier which performs only the computation of the first coefficient, \( a_0 \in GF(q) \), in the general multiplication in \( GF(q^{41}) \).

We employ the Sunar-Koc multiplication algorithm, which will be presented in Section 12.2. In this case, we are only required to cater the computation for \( \beta_1 \) in \( C_1 \), \( D_1 \) and \( D_2 \) (see Tables 2.2, 2.3 and 2.4 in Chapter 2). Similar to the usual ONBII multiplication, permutation (see Table 12.2) is first performed on both operands. Next, we will have,

\[
a_0 = (a_1 b_2 + a_2 b_1) + (a_2 b_3 + a_3 b_2) + \ldots + (a_{40} b_{41} + a_{41} b_{40}) + (a_{41} b_{41})
\] (12.1.7)

which requires a total of 81 multiplications and 81 additions over \( GF((2^2)^2) \). Last, the resultant \( a_0 \) is converted back to its original basis through the inverse permutation (see Table 12.2). Note that in the finite field of characteristic 2, both subtraction and addition are implemented using a XOR operation.
CHAPTER 12: EFFICIENT COMPOSITE FIELDS MULTIPLICATIVE INVERSION OVER GF\(((2^n)^m)^l\)

Table 12.1: Multiplicative inversion over $GF((2^2)^2)$ in ANF representation. Taking an element of four bits as $a \in GF((2^2)^2)$ as $\{a_3, a_2, a_1, a_0\}$ thus its multiplicative inversion is $a^{-1} = \{a_3^{-1}, a_2^{-1}, a_1^{-1}, a_0^{-1}\}$.

<table>
<thead>
<tr>
<th>Inversion bit</th>
<th>Arithmetic in ANF form</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_3^{-1}$</td>
<td>$a_0 + a_0a_2 + a_1a_3 + a_1a_2 + a_0a_1a_3$</td>
</tr>
<tr>
<td>$a_2^{-1}$</td>
<td>$a_1 + a_0a_2 + a_0a_1a_2 + a_0a_1a_3$</td>
</tr>
<tr>
<td>$a_1^{-1}$</td>
<td>$a_2 + a_0a_2 + a_1a_3 + a_0a_3 + a_1a_2a_3$</td>
</tr>
<tr>
<td>$a_0^{-1}$</td>
<td>$a_3 + a_0a_2 + a_0a_2a_3 + a_1a_2a_3$</td>
</tr>
</tbody>
</table>

12.1.3 Step 3: Multiplicative Inversion in $GF(q)$ yields $(A^r)^{-1}$

To avoid the use of LUTs, we propose a pure combinatorial circuit to perform the multiplicative inversion over the composite field $GF(q)$ with respect to the normal basis representation. Therefore, we can apply the approach based on our best CFA AES S-box, Case III in this work. Here, we briefly summarise the arithmetic required in deriving multiplicative inversion of $GF(q)$. Further explanation can be found in Chapters 5, 6 and 7.

All the operations that are essential for performing the multiplicative inversion over the field $GF((2^2)^2)$ in normal basis representation are noted in Table 12.1 and as depicted in Figure 12.1. Total complexity of this multiplicative inverter is 8 ANDs and 12 XORs.

Figure 12.1: Implementation of multiplicative inversion over $GF(2^4)$ using CFA in normal basis. (a) Multiplicative inversion over $GF((2^2)^2)$, (b) Multiplicative inversion over $GF(2^2)$, (c) Multiplication in $GF(2^2)$

12.1.4 Step 4: Multiplication of $(A^r)^{-1}A^r^{-1}$

In the final step, we need to multiply $A^r^{-1} \in GF(q^l)$ (from Step 1) and $(A^r)^{-1} \in GF(q)$ (from Step 3) to deduce $A^{-1}$. This step requires 41 multiplications in $GF(q)$. Here, we
can employ the $GF((2^2)^2)$ multiplier from our CFA AES S-box in Case III which is as depicted in Figure 12.2.

![Figure 12.2: Multiplication in $GF((2^2)^2)$](image)

Next, we eliminate the redundant common factors in the $GF((2^2)^2)$ multiplier and followed by merging certain suboperations within, to enhance further area reduction. Note that in $GF((2^2)^2)$ multiplier, there are two $T\Gamma$ scalers and a $T^2\Gamma$ scaler.

Furthermore, there are three $GF(2^2)$ multiplications, of which each is followed by a $T\Gamma$ scaler or a $T^2\Gamma$ scaler (see the dotted box in Figure 12.2). Therefore, we can combine the $GF(2^2)$ multiplier with their respective scaler; (5.4.13) or (5.4.12) and result in the following,

$$
GF(2^2) \text{ mult with } T\Gamma = [(g_1 + g_0)(d_1 + d_0) + g_0d_0]W^2 + (g_1d_1 + g_0d_0)W \tag{12.1.8}
$$

$$
GF(2^2) \text{ mult with } T^2\Gamma = (g_1d_1 + g_0d_0)W^2 + [(g_1 + g_0)(d_1 + d_0) + g_1d_1]W \tag{12.1.9}
$$

The complexity of our $GF((2^2)^2)$ multiplier is 9 ANDs and 20 XORs only. Further information on this $GF(2^4)$ multiplier can be found in Chapter 5.

### 12.2 Parallel Sunar-Koc Multiplication in Composite Field

We have chosen to employ Sunar-Koc multiplication algorithm for our $GF(q^l)$ multiplication in this study. Note that there will be a slight modification in the Sunar-Koc multiplier deployed here as the algorithm was initially proposed for the multiplication in
CHAPTER 12: EFFICIENT COMPOSITE FIELDS MULTIPLICATIVE INVERSION OVER $GF((2^n)^m)$

Table 12.2: Permutation mapping of element in the basis $M$ to the basis $N$ (shifted canonical). Given value in the second column is $i$ of element $A_i$ (in M basis) and element $A'_i$ (in N basis).

<table>
<thead>
<tr>
<th>Canonical Basis ($A_i$)</th>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Basis ($A'_i$)</td>
<td>1 2 4 8 16 32 19 38 7 14 28 27 29 25 33 17 34 15 30 23 37</td>
</tr>
<tr>
<td>Canonical Basis ($A_i$)</td>
<td>22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41</td>
</tr>
<tr>
<td>Original Basis ($A'_i$)</td>
<td>9 18 36 11 22 39 5 10 20 40 3 6 12 24 35 13 26 31 21 41</td>
</tr>
</tbody>
</table>

binary field of characteristic 2 only. Instead of constructing a $GF(2^{164}) \cong GF((2^2)^2)^{41}$ multiplier, the work is reduced to having a $GF(2^{41})$ Sunar-Koc multiplier. The difference lies only in the suboperations within, which will be performed over the subfield, $GF((2^2)^2)$ (cf. Table 12.2).

Let $A, B \in GF(q^{41})$, we need to first convert the operands to basis $N$;

\[
A = \sum_{i=1}^{41} a_i \beta_i = \sum_{i=1}^{41} a_i (\gamma^i + \gamma^{-i}),
\]

\[
B = \sum_{i=1}^{41} b_i \beta_i = \sum_{i=1}^{41} b_i (\gamma^i + \gamma^{-i})
\]

Next, the product $C = A \cdot B$ is computed as follows:

\[
C = (\sum_{i=1}^{41} a_i (\gamma^i + \gamma^{-i})) \cdot (\sum_{j=1}^{41} b_j (\gamma^j + \gamma^{-j}))
\]

\[
= C_1 + D_1 + D_2 \text{ (see Table 12.3)}
\]

with

\[
C_1 = \sum_{1 \leq i, j \leq 41} a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)}) \quad (12.2.1)
\]

\[
D_1 = \sum_{1=1}^{41} \sum_{j=1}^{41-i} a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) \quad (12.2.2)
\]

\[
D_2 = \sum_{1=1}^{41} \sum_{j=42-i}^{41} a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) \quad (12.2.3)
\]

The multiplication in basis $N$ constructs $C_1, D_1$ and $D_2$. Detailed expressions of all of the terms in $C_1, D_1$ and $D_2$ are tabulated in Tables 12.3. The sums of the appropriate terms result in the product of the operands $A$ and $B$, $C = C_1 + D_1 + D_2$. Last, we need
The complexity of the multiplier is summarised below:

<table>
<thead>
<tr>
<th>$\beta_1$</th>
<th>$\beta_2$</th>
<th>$\beta_3$</th>
<th>...</th>
<th>$\beta_{39}$</th>
<th>$\beta_{40}$</th>
<th>$\beta_{41}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1b_2 + a_2b_1$</td>
<td>$a_1b_3 + a_3b_1$</td>
<td>$a_1b_4 + a_4b_1$</td>
<td>...</td>
<td>$a_1b_{40} + a_{40}b_1$</td>
<td>$a_1b_{41} + a_{41}b_1$</td>
<td>...</td>
</tr>
<tr>
<td>$a_2b_3 + a_3b_2$</td>
<td>$a_2b_4 + a_4b_2$</td>
<td>$a_2b_5 + a_5b_2$</td>
<td>...</td>
<td>$a_2b_{41} + a_{41}b_2$</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$a_{40}b_{41} + a_{41}b_{40}$</td>
<td>$a_{39}b_{40} + a_{40}b_{39}$</td>
<td>$a_{38}b_{39} + a_{39}b_{38}$</td>
<td>...</td>
<td>$a_{31}b_{32} + a_{32}b_{31}$</td>
<td>$a_{22}b_{23} + a_{23}b_{22}$</td>
<td>$a_{21}b_{22}$</td>
</tr>
<tr>
<td>$D_1$</td>
<td>$a_1b_1$</td>
<td>$a_1b_2 + a_2b_1$</td>
<td>...</td>
<td>$a_1b_{38} + a_{38}b_1$</td>
<td>$a_1b_{39} + a_{39}b_1$</td>
<td>$a_1b_{40} + a_{40}b_1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$D_2$</td>
<td>$a_{41}b_{41}$</td>
<td>$a_{40}b_{41} + a_{41}b_{40}$</td>
<td>$a_{39}b_{40} + a_{40}b_{39}$</td>
<td>...</td>
<td>$a_{31}b_{32} + a_{32}b_{31}$</td>
<td>$a_{22}b_{23} + a_{23}b_{22}$</td>
</tr>
</tbody>
</table>

The complexity of the multiplier is summarised below:

$$\# \text{ 4-bit Multiplier} = m^2 = 1681$$
$$\# \text{4-bit XOR} = \frac{3}{2}m(m - 1) = 2460$$

Therefore, a $GF(((2^2)^{41})$ multiplier requires 1681 4-bit multipliers in deriving the $a_ib_j$ for $i, j = 1, 2, 3 \ldots 41$ (in $C_1, D_1$ and $D_2$). The 4-bit multiplier will be decomposed to the smaller field of $GF(2)$, which is computed using CFA as discussed in Section 12.1.4. Here, we propose a hybrid (partially serial and partially parallel) mechanism to perform the 1681 subfield multiplications.

The principle of hybrid architecture was first introduced by Mastrovito in [154]. The reference, however, did not address any optimisation scheme and the application in cryptography area as it was later presented by Paar et al. in [135]. Paar et al. proposed an optimised approach that explores bit parallel arithmetic in the subfield and serial processing for the extension field arithmetic. This hybrid approach is hardware cost effective and at the same time leads to fast implementations.

Here, we employ the methodology in [135] to case (a) and we extend the work to another case (b) and case (c). While the multiplication in the extension field is processed in serial manner, the multiplication in the subfield will be performed in fully parallel in (a) and partially serial and partially parallel in (b) and (c). To be precise, we propose
three hybrid approaches; (a) 41 (b) 21 and (c) 11 4-bit multipliers running in parallel and each would take up 41, 82 and 164 clock cycles for completion. By exploring the time-space trade-off paradigm in all three cases, the required implementation cost and the respective performances in Cyclone III EP3C120F780I7 FPGA are summarised in Table 12.4.

Based on the results in Table 12.4, we have chosen the approach (a) for our final hardware implementation such as depicted in Figure 12.3. Using this approach, we only need 41 $GF((2^2)^41)$ multipliers to perform 1681 multiplications, which will be completed in 41 clock cycles. Although the approach (a) requires 1.79% and 2.93% more logical elements than the approaches (b) and (c), it leads to a performance improvement in terms of clock rate of 8.07% and 10.17% comparatively. Based on this result, we deduce that the complexity of a $GF((2^2)^2)$ multiplier is relatively low.

Last but not the least, we need to perform inverse permutation where the resultant $C$ is converted back to its original basis (refer to Table 12.2).

<table>
<thead>
<tr>
<th>Complexity and Performance</th>
<th>(a) Mult41</th>
<th>(b) Mult21</th>
<th>(c) Mult11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total LE</td>
<td>8511</td>
<td>8609</td>
<td>8597</td>
</tr>
<tr>
<td>Total Combinatorial Functions</td>
<td>8511</td>
<td>8361</td>
<td>8269</td>
</tr>
<tr>
<td>Total Register</td>
<td>502</td>
<td>765</td>
<td>766</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>126.47</td>
<td>117.03</td>
<td>114.80</td>
</tr>
<tr>
<td>Cycle of completion</td>
<td>41</td>
<td>82</td>
<td>164</td>
</tr>
</tbody>
</table>

### 12.3 Implementation and Results

The overview of our proposed $GF((2^2)^2)^{41}$ multiplicative inverter is as depicted in Figure 12.4. In order to demonstrate the efficacies of our multiplicative inverter in EC hardware cryptosystem, its computational cost is benchmarked with the previous works. From our literature review, the most recent and comparable work that we can find is the work by Guajardo and Paar, reported in 1997 [2, 155].

Guajardo and Paar proposed a conversion scheme that is closely comparable to our work but in software implementation. Similar to our work, they had also employed the ITI algorithm for multiplicative inversion over composite field elements. How-
CHAPTER 12: EFFICIENT COMPOSITE FIELDS MULTIPLICATIVE INVERSION OVER $GF(((2^n)^m)^l)$

**Figure 12.3:** Hybrid multiplier for $GF(((2^2)^2)^{41})$

- Multiplication in $GF((2^2)^2)$: $A = \{a_0, a_1, a_2, \ldots, a_{40}\}$
- Concatenation: $B = \{b_0, b_1, b_2, \ldots, b_{40}\}$
- $W = A \times B = \{w_0, w_1, w_2, \ldots, w_{40}\}$

**Figure 12.4:** Overview of our proposed compact and efficient multiplicative inversion over $GF(((2^2)^2)^{41})$. The $GF(((2^2)^2)^{41})$ multiplier in Step 1 is illustrated in Figure 12.3.
CHAPTER 12: EFFICIENT COMPOSITE FIELDS MULTIPlicative INVERSION OVER $GF((2^m)^l)$

Table 12.5: Complexity of $GF((2^4)^{41})$ multiplicative inversion using our proposed scheme and the work by Guajardo and Paar in [2].

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
<th>#AND</th>
<th>#XOR</th>
<th>[2] in $GF((2^4)^{41})$</th>
<th>#LUT (64 bits)</th>
<th>#XOR</th>
<th>#EXPA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$GF(2^4)$ Multiplier</td>
<td>4-bit XOR</td>
<td></td>
<td>$GF(2^4)$ Multiplier</td>
<td>4-bit XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6 * 1681</td>
<td>6 * 2460</td>
<td>90774</td>
<td>6 * 1282</td>
<td>(6 * 1364) + 31200</td>
<td>7920</td>
<td>157536</td>
</tr>
<tr>
<td>2</td>
<td>81</td>
<td>81</td>
<td>729</td>
<td>1944</td>
<td>78</td>
<td>234</td>
<td>308</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>12</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>41</td>
<td>-</td>
<td>369</td>
<td>820</td>
<td>41</td>
<td>123</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>10208</td>
<td>14841</td>
<td>91880</td>
<td>263536</td>
<td>7811</td>
<td>8279</td>
<td>157844</td>
</tr>
</tbody>
</table>

ever, their arithmetic was performed in two levels composite field of $GF((2^m)^l)$, using the polynomial basis representation. They had applied the KOA for multiplication of polynomials over field $GF(2^n)$ of degree $m - 1$, where LUTs were utilised for multiplication in $GF(2^n)$. In their work, two new operations were defined; look-up table (LUT) and exponent addition (EXPA) for KOA in composite field multiplications. In essence, EXPA is an adder modulo the order of the multiplication group.

As their architecture was designed for software realisation, we can only perform an analytical comparison between their scheme in the field $GF((2^4)^{41})$ and our work. The circuits’ complexities of both multiplicative inverter are summarised in Table 12.5. Detailed description of Guajardo and Paar multiplicative inversion algorithm defined over $GF((2^4)^{41})$ is presented in Appendix C.

From Table 12.5, our work requires a total of 10,208 $GF((2^2)^2)$ multipliers and a total of 14,841 4-bits XORs. Without the use of any LUTs, our architecture consumes a total of 91,880 ANDs and 263,536 XORs. On the other hand, two types of LUTs were required in their work. One multiplicative inversion over the subfield $GF(2^4)$ involved two LUTs and one exponent adder. Meanwhile, one multiplication in the subfield would take up three LUTs and one exponent adder. On the contrary, both our multiplicative inverter and multiplier are constructed using ANDs and XORs only. For comparison, our work requires 23.48% more subfield multipliers compared to the work in [2]. On the other hand, their work requires 62.39% more 4-bit XORs compared to ours.

In order to show the advantages of using CFA, we also include several reported designs for the computation of multiplicative inversion over $GF(2^m)$ in hardware platforms for benchmarking. The computational cost and area of these works and our work are summarised in Table 12.6. From Table 12.6, the time complexity of the work by Deng et al. in [133] is the smallest but it comes with a higher hardware cost. Meanwhile, our architecture is slower in terms of timing performance but it consumes relatively
CHAPTER 12: EFFICIENT COMPOSITE FIELDS MULTIPLICATIVE INVERSION OVER \( GF((2^n)^m)^f \)

Table 12.6: Computational cost comparison of various multiplicative inverter architectures in hardware platform (CLB = Configurable Logic Block, LE = Logic Element)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Finite Field</th>
<th>Cycles</th>
<th>Freq (MHz)</th>
<th>Timings (µS)</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodman et al. [130]</td>
<td>0.25µCMOS</td>
<td>( GF(2^{256}) )</td>
<td>3,712</td>
<td>50</td>
<td>74.24</td>
</tr>
<tr>
<td>Gutub et al. [129]</td>
<td>0.5µCMOS</td>
<td>( GF(2^{256}) )</td>
<td>5,000</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Rodríguez-Henríquez et al. [156]</td>
<td>Xilinx Virtex</td>
<td>( GF(2^{193}) )</td>
<td>-</td>
<td>-</td>
<td>1.37</td>
</tr>
<tr>
<td>Gao et al. [157]</td>
<td>EPIC20FC4000</td>
<td>( GF(2^{133}) )</td>
<td>-</td>
<td>-</td>
<td>3.64</td>
</tr>
<tr>
<td>Rodríguez-Henríquez et al.</td>
<td>Xilinx VirtexE</td>
<td>( GF(2^{133}) )</td>
<td>28</td>
<td>21.2</td>
<td>1.32</td>
</tr>
<tr>
<td>(Standard) [132]</td>
<td>Xilinx VirtexE</td>
<td>( GF(2^{133}) )</td>
<td>20</td>
<td>21.2</td>
<td>0.943</td>
</tr>
<tr>
<td>Rodríguez-Henríquez et al.</td>
<td>Xilinx VirtexE</td>
<td>( GF(2^{133}) )</td>
<td>-</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td>(Parallel) [132]</td>
<td>Xilinx VirtexE</td>
<td>( GF(2^{133}) )</td>
<td>-</td>
<td>-</td>
<td>0.867</td>
</tr>
<tr>
<td>Deng et al. [133]</td>
<td>EP1C20FC400C6</td>
<td>( GF(2^{133}) )</td>
<td>26</td>
<td>30</td>
<td>0.26</td>
</tr>
<tr>
<td>Deng et al. [133]</td>
<td>0.18µCMOS</td>
<td>( GF(2^{133}) )</td>
<td>26</td>
<td>100</td>
<td>0.5</td>
</tr>
<tr>
<td>Wei [134]</td>
<td>Xilinx Virtex5</td>
<td>( GF(2^{233}) )</td>
<td>25</td>
<td>50</td>
<td>-</td>
</tr>
<tr>
<td>Our work</td>
<td>Cyclone III</td>
<td>( GF((2^2)^{41}) )</td>
<td>247</td>
<td>100</td>
<td>2.47</td>
</tr>
</tbody>
</table>

smaller amount of hardware resources, of which would be suitable for medium range or lightweight applications.

As tabulated in Table 12.5, an multiplicative inversion over \( GF((2^2)^{41}) \) requires one multiplicative inversion and several multiplications in the subfield, \( GF((2^2)^2) \). As a result, the complexity of the \( GF((2^2)^2) \) multiplier is the major factor that would determine the amount of hardware resources required (area and power) and the performance of the overall multiplicative inverter architecture. In this study, we would like to point out the advantages of using combinatorial finite field multiplicative inverter in hardware implementation as opposed to the pre-computed table method.

The previous works of the composite field EC cryptosystems [2, 79, 73, 89] employed a pre-computed table method in the subfield arithmetic. This method is by the means of performing log and antilog conversion which involved two types of tables. Each of the tables took up \( 2^n \) of \( n \) bits, resulting in a total memory requirement of 64 bits for field \( GF(2^4) \). These tables are used to calculate the multiplication (see (8.2.1)) and the multiplicative inversion (see (8.2.2)) of field elements, such as explained in Chapter 8. Based on these equations, one multiplication and one multiplicative inversion over the subfield \( GF(2^4) \) would take up three and two 64 bits LUTs each.

Another pre-computed table method can be seen in the conventional LUT approach. This is done by having all the possible input combinations and their respective multiplication outputs pre-computed and stored in memory form. In this case, only one LUT is required for each of the multiplication and multiplicative inversion operations. This results in a total memory requirement of 480 bits for multiplication and 64 bits for
CHAPTER 12: EFFICIENT COMPOSITE FIELDS MULTIPLICATIVE INVERSION OVER $GF((2^n)^m)^4$ 

multiplicative inversion.

On the contrary, without using any LUT, our $GF((2^2)^2)$ multiplicative inverter can be constructed using 8 ANDs and 12 XORs, while our $GF((2^2)^2)$ multiplier is built up of 9 ANDs and 20 XORs. Apart from our multiplicative inverter, we have constructed another two composite field multiplicative inverters, by having the $GF(2^4)$ multipliers implemented using the log and antilog conversion and the conventional LUT method. Note that, the log and antilog tables can be implemented using FPGA such as shown in Figure 12.5. For verification and validation purposes, we have implemented the three architectures in Cyclone III EP3C120F780I7 FPGA. Having the architectures synthesised using Quartus II 7.2sp3, the summary of the hardware requirements are tabulated in Table 12.7.

![Figure 12.5: Multiplicative inversion and multiplication in $GF(2^4)$ using log and antilog conversions](image)

**Table 12.7:** Hardware analysis of FPGA implementation for $GF(((2^2)^2)^41)$ multiplicative inverter with the $GF(2^4)/GF((2^2)^2)$ multiplier implemented using (i) combinational circuitry as proposed in our work (ii) log and antilog conversions and (iii) multiplication LUT. Device used for all of the implementations is EP3C120F780I7 Cyclone III.

<table>
<thead>
<tr>
<th></th>
<th>(i) Combinatorial Circuitry</th>
<th>(ii) Log and Antilog Conversions</th>
<th>(iii) Multiplication LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total LE</td>
<td>52,660</td>
<td>53,878</td>
<td>65,979</td>
</tr>
<tr>
<td>Total Combinatorial Functions</td>
<td>51,159</td>
<td>52,326</td>
<td>64,318</td>
</tr>
<tr>
<td>Dedicated logic register</td>
<td>4,663</td>
<td>4,663</td>
<td>4,663</td>
</tr>
<tr>
<td>Total Register</td>
<td>4,663</td>
<td>4,663</td>
<td>4,663</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>107.76</td>
<td>90.52</td>
<td>102.31</td>
</tr>
<tr>
<td>Total Thermal Power Dissipation (mW)</td>
<td>503.73</td>
<td>640.85</td>
<td>885.10</td>
</tr>
<tr>
<td>Core Dynamic Thermal Power Dissipation (mW)</td>
<td>375.69</td>
<td>456.24</td>
<td>667.19</td>
</tr>
<tr>
<td>Core Static Power Dissipation (mW)</td>
<td>69.61</td>
<td>70.06</td>
<td>70.90</td>
</tr>
<tr>
<td>I/O Thermal Power Dissipation (mW)</td>
<td>58.43</td>
<td>114.54</td>
<td>147.00</td>
</tr>
</tbody>
</table>

From Table 12.7, it is evident that our approach has outperformed the LUT-based approaches in terms of performance and at the same time it required less amount of hardware resources. In addition to that, such approach enables further optimisation
in architectural level such as pipelining for speed enhancement. On the contrary, the LUT which is implemented as a single memory block in FPGA, limits the possibility for subpipelining in the architecture, and therefore constrained the highest achievable processing speed.

12.4 Discussion

Our contributions in this work are twofold. First, we introduce an efficient three-level composite field, \( GF(((2^2)^2)^41) \), suitable for hardware EC cryptosystems. Our field is selected such that it is secure and promotes simplicity in CFA at the same time. Second, we propose a novel three-level composite field multiplicative inverter using the ITI algorithm in optimal normal type II basis representation (ONBII). Unlike the previous reports, we performed further isomorphisms in the subfield, \( GF(2^4) \cong GF((2^2)^2) \), such that the need for LUTs can be eliminated completely.

In addition to that, we present a series of algorithmic optimisations in the subfield \( GF((2^2)^2) \) operations, and also a hybrid serial-parallel Sunar-Koc multiplier in the extension field, in order to promote maximum area reduction in the design. In essence, the complexity of the subfield arithmetic will determine the complexity of the CFA as a whole. A multiplicative inversion over \( GF(((2^2)^2)^41) \) would require a multiplicative inverter and several multipliers in \( GF((2^2)^2) \). Here, we presented a series of algorithmic optimisation that successfully reduced complexity of the subfield operators. As a result, without the use of LUT, our multiplicative inverter and multiplier in the sub-field are having the complexity of 8 ANDs and 12 XORs and 9 ANDs and 20 XORs respectively.

Overall, our results in this study have shown that, 1, our three-level composite field multiplicative inversion for ECC is distinctive from the previous works as commonly deployed LUTs can be replaced with combinatorial circuitry 2, Our composite field multiplicative inverter is indeed more compact and with reasonable performance compared to the pre-computed LUT methods suggested in the previous studies. 3, The composite field multiplicative inverter is highly desirable in the EC hardware cryptosystem as it offers better computational efficiency and effectiveness, in terms of hardware cost, compared to those that worked in binary finite field \( GF(2^m) \).
Conclusion and Future Works

13.1 Summary

In this study, we presented a series of optimisation approaches for AES and EC cryptosystems in hardware implementation. We particularly aimed at deriving efficient and effective multiplicative inversion circuitries for both ciphers. The computation of multiplicative inversion is the major bottleneck in achieving compact, low power consumption and high performance in cipher implementation. The followings are the brief summary of our work in AES and EC cryptosystems.

13.1.1 Part I: Advanced Encryption Standard

In the first part of this thesis, a detailed study on composite field construction for the S-box function in AES was presented. As a result, three new constructions of CFA AES S-boxes were deduced, namely the Case I, Case II and Case III. Next, a series of algorithmic reduction followed by substructure sharing in the architectural level were performed on these CFA AES S-boxes.

Furthermore, we had explored all of the possible isomorphic mappings for each of the composite field construction and employed a new CSE algorithm to derive the optimum isomorphic and inverse isomorphic mapping followed by an affine transformation. As a result, the optimum CFA AES S-box that achieved an optimally balanced construction in terms of area of implementation and critical path, compared to the previous studies was found in Case III. This served as one of the major contributions of our work.

In addition to that, we had found that there was a substantial gain in our CFA AES
S-box in achieving a high throughput FPGA implementation. We had successfully demonstrated the capability of fine-grained pipelining in achieving power reduction and speed enhancement. Therefore, the role of pipelining in this work was dual-pronged: 1) to ensure consistency within each pipeline stage such that unnecessary switching activity was avoided; and 2) to reduce the critical path to the minimum, and therefore promote high clock rate performance. In order to maximise the effectiveness of pipelining, we had converted our CFA AES S-box into five modules of ANF, consisting only AND gates and XOR gates.

With the employment of ANF representation and seven stages fine-grained pipelining, our best architecture, Case III managed to achieve a high throughput of 3.49 Gbps with the utilisation of 96 LEs in Cyclone II EP2C5T144C6 FPGA and thermal power consumption of 34.80mW. Analytically, this best architecture possesses a total of 35 AND gates and 95 XOR gates with critical path of 3 AND gates and 18 XORs.

Apart from that, we had reported two CFA AES S-box constructions using the FLT-based inversion for benchmarking purposes. The architectures employed the generalised ITI algorithm, over $GF((2^4)^2)$ and $GF((2^2)^4)$. Among the two architectures, the construction over $GF((2^4)^2)$ appeared to be a better choice. However, our initial development, the Case III had outperformed both the FLT-based constructions in terms of performance and hardware cost.

Though the approach in Case III has successfully resulted in smaller area of implementation, the required mathematical calculations are more complex. However, the effort is worthwhile since AES works in $GF(2^8)$, which is a relatively small field. For Galois field of the higher order, FLT-based inversion is a better choice. The algorithm can conveniently reduce the multiplicative inversion of at the extension field to the lower field, of which can be solved using EEA, for instance. This was better discussed in the optimisation of EC hardware cryptosystem, which was explained in the second part of this thesis.

### 13.1.2 Part II: Elliptic Curve Hardware Cryptosystem

Previous studies on the hardware realisation of EC cryptosystems mostly emphasised on scalability and flexibility in design. Given the required performance, vendor preferences, security requirements and processor capabilities, the desired cryptosystem can be generated directly and thus shorten the development time. On the other hand, in-
Chapter 13: Conclusion and Future Works

Instead of focusing on the flexibility in the design, we emphasised on efficient area reduction, using specific parameters, to achieve compact computation for area-constrained hardware cryptosystems. Therefore, in this work, we explored the optimisation approaches in the field level.

In particular, the scalar multiplication, $kP$, is the most crucial and yet also the most complicated operation in ECC for it requires multiplicative inversions over the finite field. The use of multiplicative inversion operation can be eliminated completely by choosing the projective coordinates systems instead of the affine coordinate systems. However, this benefit is traded off as more multiplications are required, which is another resource consuming operation. Instead of doing so, we aimed at designing a compact and efficient multiplicative inverter that is suitable for EC hardware cryptosystems.

We presented a new efficient composite field multiplicative inverter of the form $GF(q^l)$, with $q = 2^{n \cdot m}$ that is suitable for the hardware realisation of an EC cryptosystem. The subfield was chosen to work in the order of $q = 4 = 2 \cdot 2$ while having the extension field $l = 41$. Considering both the security aspect and the hardware cost required, we proposed the utilisation of the composite field $GF(((2^2)^2)^{41})$ for EC cryptosystem. To our best knowledge, we are the first to implement three-level isomorphism CFA in EC hardware cryptosystem.

For efficient implementation, we had derived a combinatorial multiplicative inversion circuit over $GF(2^{164}) \cong GF(((2^2)^2)^{41})$ to achieve an optimal saving in the hardware cost required. With this, we substantially reduced the hardware cost of the scalar multiplication in ECC. Unlike the previous reports, we performed further isomorphisms in the subfield, $GF(2^4) \cong GF((2^2)^2)$, such that the need for LUTs was eliminated completely. While the extension field $l$ has to be a prime number, we used the ONBII representation, which enabled the 40 exponentiations to be done easily using simple cyclic shifts.

Furthermore, we had also employed a hybrid architecture of serial-parallel Sunar-Koc multiplier from the work [135] for the multiplication in the extension field. It performed in a way that the multiplication in the extension field was processed in a serial manner while the multiplication in the subfield was performed in a parallel manner. This hybrid approach was proven to be hardware cost effective and the same with an acceptable speed performance.

In essence, the complexity of the subfield arithmetic would determine the complexity
of the CFA as a whole. A multiplicative inversion over $GF((2^2)^{41})$ would require a multiplicative inverter and several multipliers in $GF((2^2)^2)$. Here, we have also presented a series of algorithmic optimisation that successfully reduced the complexity of the subfield operators. As a result, without the use of LUT, our multiplicative inverter and multiplier in the subfield are having the complexity of 8 ANDs and 12 XORs and 9 ANDs and 20 XORs respectively.

In this study, we have successfully proven that our approach outperformed the existing measure in achieving small area and low cost composite field multiplicative inverter, which offers a reasonable performance, that is well suited for hardware EC cryptosystem.

### 13.2 Directions for Future Works

This section will provide the reader with an overview of the possible areas in which further work could be pursued. Several ideas have come up as a result of this research and provide opportunities to investigate the potential works in continuation of this work.

One of the possible directions is to construct composite field with isomorphism to the multi-level representation basis. Our work, as well as the reported studies so far, only attempted on CFA with single basis representations. However, it is possible in mathematical sense, to construct composite fields of different representation basis. Therefore, this is worth of a study as we may derived a new CFA AES S-box of different complexities and performance. Apart from that, we can extend our CSE algorithm to the optimisation in the MixColumns/invMixColumns transformations in AES. The transformations involve multiplication with constant matrices, of which substructure sharing is a potential optimisation approach to promote efficient area reduction.

For EC cryptosystem, we can also perform optimisation at the architectural levels, such as pipelining or subpipelining in order to achieve further enhancement in terms of throughput. Not only that, we can employ the power optimisation schemes which are performed in our CFA AES S-box to the multiplicative inversion in EC cryptosystems. Furthermore, previous studies in composite field EC cryptosystems only worked on two-level isomorphism, therefore it would be useful to have a thorough study on the security aspects in our proposed works.
All of the work proposed in this study are specialised for the cryptographic application in reconfigurable hardware platform (FPGA). For future work, we would like to extend these optimisation schemes in another commonly deployed hardware, which is the ASIC platform.

13.3 Concluding Remarks

Overall, implementation of certain cipher algorithms are heavily dependent on the application’s requirements. These are namely the hardware area cost, power consumption and the speed performance.

Therefore, we hope that the work presented here will provide some insight into the hardware implementation of the complex cryptographic algorithms, specifically the AES and EC cryptosystems. Our work mainly aimed at optimising the most complicated operation required in the cipher, which is the multiplicative inversion. We constructed optimum composite field which avoids the usage of LUT and therefore allows efficient multiplicative inversion operations. The optimality that we sought for is the architecture that consumes minimum amount of hardware area cost, low power consumption and with a considerable high performance. However, these features are often contradictory to one another in most VLSI designs. In this work, we managed to construct the composite field circuitries that strike a balance in terms of hardware requirements and performance.

Apart from AES and EC cryptosystems, the methodologies proposed in this work are also applicable for development of any similar cryptographic circuits that involved finite field arithmetic.
References


Available: http://www.sciencedirect.com/science/article/B6WGK-4DX43J5W/2/a118807a9c5129a1a3076b1d0122edad


REFERENCES


REFERENCES

[107] G. M. de Dormale and J.-J. Quisquater, “High-speed hardware implementa-


[110] G. Orlando and C. Paar, “A high-performance reconfigurable elliptic curve pro-


162
REFERENCES


[150] M. Maurer, A. Menezes, and E. Teske, “Analysis of the GHS Weil descent attack on the ECDLP over characteristic two finite fields of composite degree,” in


Appendix A

List of possible coefficients in $GF(((2^2)^2)^2)$ irreducible polynomials

Three sets of irreducible polynomials are required in the computation of multiplicative inversion over $GF(((2^2)^2)^2)$ (refer Chapter 5). The coefficient in these polynomials play a vital role in determining the complexity of the resultant arithmetic. However, only certain coefficients that ensure the irreducibility of the polynomials are valid.

Here, we list the choices of $\tau, \nu, T$ and $N$ in the irreducible polynomials of (5.2.1), (5.2.2) and (5.2.3). From the irreducibility test for $\tau$ and $T$ equal unity, $N$ could either be $\{10\}_2$ or $\{11\}_2$ for polynomial basis. The respective 8 possible values of $\nu$ are deduced in Table A.1. On the other hand, $N$ could either be $\{01\}_2$ or $\{10\}_2$ for normal basis and the respective 8 possible values of $\nu$ are tabulated in Table A.2.

In addition to that, based on the irreducibility test for $\nu$ and $N$ equal unity, $T$ could either be $\{10\}_2$ or $\{11\}_2$ for polynomial basis. Their respective 8 choices of $\tau$ are noted in Table A.3. For normal basis, $T$ could either be $\{01\}_2$ or $\{10\}_2$. Hence, all the 8 possible values of $\tau$ are as shown in Table A.4.
**APPENDIX A: LIST OF POSSIBLE COEFFICIENTS IN $GF((2^2)^2)$ IRREDUCIBLE POLYNOMIALS**

**Table A.1:** Choices of norm value ($\nu$) for $N = \{10\}_2$ and $N = \{11\}_2$ in polynomial basis representations

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\nu$ in Polynomial Vector</th>
<th>$\nu$ in Polynomial Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>${10}_2, {11}_2$</td>
<td>$\Gamma z$</td>
<td>${1000}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + 1$</td>
<td>${1001}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + \Gamma$</td>
<td>${1010}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + \Gamma^2$</td>
<td>${1011}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z$</td>
<td>${1100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + 1$</td>
<td>${1101}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + \Gamma$</td>
<td>${1110}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + \Gamma^2$</td>
<td>${1111}_2$</td>
</tr>
</tbody>
</table>

**Table A.2:** Choices of norm value ($\nu$) for $N = \{01\}_2$ and $N = \{10\}_2$ in normal basis representations

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\nu$ Polynomial Vector</th>
<th>$\nu$ in Normal Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>${01}_2, {10}_2$</td>
<td>$\Gamma Z^4$</td>
<td>${0100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + Z$</td>
<td>${0111}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + \Gamma Z$</td>
<td>${0101}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + \Gamma^2 Z$</td>
<td>${0110}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4$</td>
<td>${1000}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + Z$</td>
<td>${1011}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + \Gamma Z$</td>
<td>${1001}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + \Gamma^2 Z$</td>
<td>${1010}_2$</td>
</tr>
</tbody>
</table>
**APPENDIX A: LIST OF POSSIBLE COEFFICIENTS IN GF((2^2)^2) IRREDUCIBLE POLYNOMIALS**

**Table A.3:** Choices of trace value $\tau$ for $T = \{10\}_2$ and $T = \{11\}_2$ in polynomial basis representations

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\tau$ in Polynomial Vector</th>
<th>$\tau$ in Polynomial Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>${10}_2$</td>
<td>$z$</td>
<td>${0100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z$</td>
<td>${1100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + 1$</td>
<td>${1001}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + 1$</td>
<td>${1101}_2$</td>
</tr>
<tr>
<td></td>
<td>$z + \Gamma$</td>
<td>${0110}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + \Gamma$</td>
<td>${1010}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + \Gamma$</td>
<td>${1110}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + \Gamma^2$</td>
<td>${1111}_2$</td>
</tr>
<tr>
<td>${11}_2$</td>
<td>$z$</td>
<td>${0100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z$</td>
<td>${1000}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + 1$</td>
<td>${1001}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + 1$</td>
<td>${1101}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + \Gamma$</td>
<td>${1010}_2$</td>
</tr>
<tr>
<td></td>
<td>$z + \Gamma^2$</td>
<td>${0111}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma z + \Gamma^2$</td>
<td>${1011}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 z + \Gamma^2$</td>
<td>${1111}_2$</td>
</tr>
</tbody>
</table>

**Table A.4:** Choices of trace value for $T = \{01\}_2$ and $T = \{10\}_2$ in normal basis representations

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\tau$ in Polynomial Vector</th>
<th>$\tau$ in Normal Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>${01}_2$</td>
<td>$Z^4$</td>
<td>${1100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4$</td>
<td>${1000}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + Z$</td>
<td>${0111}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + Z$</td>
<td>${1011}_2$</td>
</tr>
<tr>
<td></td>
<td>$Z^4 + \Gamma Z$</td>
<td>${1101}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + \Gamma Z$</td>
<td>${0101}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + \Gamma Z$</td>
<td>${1001}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + \Gamma^2 Z$</td>
<td>${1010}_2$</td>
</tr>
<tr>
<td>${10}_2$</td>
<td>$Z^4$</td>
<td>${1100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4$</td>
<td>${0100}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + Z$</td>
<td>${0111}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + Z$</td>
<td>${1011}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + \Gamma Z$</td>
<td>${0101}_2$</td>
</tr>
<tr>
<td></td>
<td>$Z^4 + \Gamma^2 Z$</td>
<td>${1110}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma Z^4 + \Gamma^2 Z$</td>
<td>${0110}_2$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma^2 Z^4 + \Gamma^2 Z$</td>
<td>${1010}_2$</td>
</tr>
</tbody>
</table>
CFA AES S-box over $GF(((2^2)^2)^2)$ of other formations

The CFA S-boxes over $GF(((2^2)^2)^2)$ that were proposed in the previous studies can be classified into two constructions, Case $i$ and Case $ii$ which are listed below.

1. Case $i$ [40, 47]: Using polynomial basis representation with field polynomials’ traces equal unity (c.f. Figure B.1).

2. Case $ii$ [39, 41, 42]: Using normal basis representation with field polynomials’ traces equal unity (c.f. Figure B.2).

The algorithmic reduction for both Case $i$ and Case $ii$ are reviewed and the resulting best architecture for both cases are summarised in Table B.3.

Case $i$ Optimisations. First, we consider the case with polynomial basis which the traces of the irreducible polynomials are equal to unity. In $GF(2^8)/GF(2^4)$, the required multiplicative inversion, which is defined in the following:

\[
\delta_1 = (\gamma_1^2 v + \gamma_1 \gamma_0 + \gamma_0^2)^{-1} \gamma_1 \quad (B.0.1)
\]

\[
\delta_0 = (\gamma_1^2 v + \gamma_1 \gamma_0 + \gamma_0^2)^{-1} (\gamma_0 + \gamma_1) \quad (B.0.2)
\]

It is worth noting that the multiplicative inversion is simplified to multiplicative inversion and multiplications (scaling and squaring) in field of $GF(2^4)$. For further optimisation, multiplication by a known constant, $v$, which is usually referred to as scaling, ought to have a separate circuit rather than using the generic $GF(2^4)$ multipliers. The same is true for squaring, $\gamma^2$ as well. Another improvement can be obtained through...
Figure B.1: Case i: Implementation of multiplicative inversion of $GF(2^5)$ using CFA in polynomial basis with field polynomials’ traces equal unity. (a) Multiplicative inversion over $GF(2^5)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$

Figure B.2: Case ii: Implementation of multiplicative inverse over $GF(2^8)$ using CFA in normal basis with field polynomials’ traces equal unity. (a) Multiplicative inversion over $GF(2^8)$, (b) Multiplicative inversion over $GF(2^4)$, (c) Multiplicative inversion over $GF(2^2)$, (d) Multiplication in $GF(2^4)$, (e) Multiplication in $GF(2^2)$
Combining these scaling of $\gamma = \Gamma_1 z + \Gamma_0 (GF(2^4))$ by $\nu = \Delta_1 z + \Delta_0$ with a squarer in $GF(2^4)$. As such, we need only one additional specific circuit for $\nu \gamma^2$.

We define the squarer for $GF(2^4)$ in the following,

$$\gamma^2 = (\Gamma_1 z + \Gamma_0)(\Gamma_1 z + \Gamma_0) \mod (z^2 + z + N)$$

$$= (\Gamma_1 z + \Gamma_0)(\Gamma_1 z + \Gamma_0) + (\Gamma_1 \Gamma_1)(z^2 + z + N)$$

$$= (\Gamma_1^2)z + (\Gamma_0^2 + \Gamma_1^2 N) \quad (B.0.3)$$

Combining the scaling operation with the squaring operation (see (B.0.3)), i.e. $\nu \gamma^2$, gives:

$$\nu \gamma^2 = v \otimes (\Gamma_1 z + \Gamma_0)^2$$

$$= [\Delta_1 z + \Delta_0] \otimes [(\Gamma_1^2)z + (\Gamma_0^2 + \Gamma_1^2 N)] \mod (z^2 + z + N)$$

$$= [\Delta_1 (\Gamma_0^2 + \Gamma_1^2 + N \Gamma_1^2) + \Delta_0 \Gamma_1^2]z + [\Delta_0 (\Gamma_0^2 N + \Gamma_0^2) + \Delta_1 (N \Gamma_1^2)] \quad (B.0.4)$$

The operation in (B.0.4) can be further simplified based on the choice of $\nu$. Elements in $GF(2^2)$ are $\{0, 1, N, N + 1\}$ but only $N$ and $N + 1$ are valid for $s(z) = z^2 + z + N$ to be irreducible. Based on (B.0.4), we need both scaler and squarer of $GF(2^2)$ as defined in the following:

$$(w) \otimes (g_1 w + g_0) = (g_1 \oplus g_0)w + g_1 \quad (B.0.5)$$

$$(w^2) \otimes (g_1 w + g_0) = (g_0)w + (g_0 \oplus g_1) \quad (B.0.6)$$

and,

$$\Gamma^2 = (g_1 w + g_0)(g_1 w + g_0) \mod (w^2 + w + 1)$$

$$= (g_1)w + (g_0 + g_1) \quad (B.0.7)$$

Hence, both scaling (see (B.0.5) and (B.0.6)) and squaring (see (B.0.7)) in $GF(2^2)$ require only one addition each. In addition to that, as every nonzero element in $GF(2^2)$, $\Gamma$, satisfies $\Gamma^3 = 1$, hence $N^{-1} = N^2 = N + 1$ i.e., multiplicative inverter is the same as the
**Table B.1:** Optimisation in multiplicative inversion using polynomial basis (trace unity), with $\gamma = \Gamma_1 z + \Gamma_0$ and $\nu = \Delta_1 z + \Delta_0$. Using Case i and polynomial basis representation, the possible pairs of $N$ and $\nu$ that result in minimal complexity in the $\nu\gamma^2$ operation are determined.

<table>
<thead>
<tr>
<th>$\Delta_1$</th>
<th>$\Delta_0$</th>
<th>Value of $N$</th>
<th>Value of $\nu$</th>
<th>$\nu\gamma^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{\pi}$</td>
<td>0</td>
<td>${10}_2$</td>
<td>${1100}_2$</td>
<td>$\nu\gamma^2 = [{N\Gamma_1^2} + N{N\Gamma_0^2}]z + N^2{N\Gamma_1^2}$</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>${11}_2$</td>
<td>${1000}_2$</td>
<td>$\nu\gamma^2 = [{N\Gamma_0^2} + N^2{N\Gamma_1^2}]z + N{N\Gamma_1^2}$</td>
</tr>
<tr>
<td>$\pi$</td>
<td>0</td>
<td>${10}_2$</td>
<td>${1000}_2$</td>
<td>$\nu\gamma^2 = [{N\Gamma_0^2} + N^2{N\Gamma_1^2}]z + N{N\Gamma_1^2}$</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{\pi}$</td>
<td>${10}_2$</td>
<td>${1111}_2$</td>
<td>$\nu\gamma^2 = [N{N\Gamma_0^2} + N^2{N\Gamma_1^2}]z + N{N\Gamma_0^2}$</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>${11}_2$</td>
<td>${1010}_2$</td>
<td>$\nu\gamma^2 = [N{N\Gamma_0^2} + N^2{N\Gamma_1^2}]z + N{N\Gamma_0^2}$</td>
</tr>
</tbody>
</table>

It is worth noticing that combination of the squaring, (B.0.7) and scaling by $w$, (B.0.5) in the field of $GF(2^2)$,

\[
(w) \otimes \Gamma^2 = (w) \otimes (g_1 w + g_0)^2 \\
= (w) \otimes (g_1) w + (g_0 + g_1) \\
= (g_0) w + g_1
\]  

is a free operation as only swapping is involved and therefore no logic gate is required.

From (B.0.4), there are three combinations of $\Delta_1$ and $\Delta_0$ that result in minimal constructions for $\nu\gamma^2$ as deduced in Table B.1. All of the constructions have the complexity of one addition and two scalings since the operation in $\{\}$ is free. Scaling in $GF(2^2)$ requires only 1 addition thus, overall any of the $\nu\gamma^2$ above gives the complexity of 3 additions.

**Case ii Optimisations.** Similar optimisations are available in normal basis as well, though some details have changed. A specific circuit for combination of scaling and squaring, $\nu\gamma^2$ is needed just like in Case I. Squaring for normal basis $GF(2^4)$ is defined as the following:

\[
\gamma^2 = (\Gamma_1Z^4 + \Gamma_0Z)(\Gamma_1Z^4 + \Gamma_0Z) \\
= [(\Gamma_1 + \Gamma_0)^2 N + \Gamma_1^2]Z^4 + [(\Gamma_1 + \Gamma_0)^2 N + \Gamma_0^2]Z
\]  

Therefore $\nu\gamma^2$ would be,
which requires sub-operations in the field of \( GF(2^2) \). Hence, scaling of both \( N \) and \( N + 1 \) are derived as following:

\[
(W) \otimes (g_1 W^2 + g_0 W) = (g_0 \oplus g_1) W^2 + (g_1) W \quad (\text{B.0.11})
\]

\[
(W^2) \otimes (g_1 W^2 + g_0 W) = (g_0) W^2 + (g_0 \oplus g_1) W \quad (\text{B.0.12})
\]

Meanwhile, squaring of \( \Gamma = g_1 W^2 + g_0 W \) in \( GF(2^2) \) is as follows:

\[
\Gamma^2 = (g_1 W^2 + g_0 W)(g_1 W^2 + g_0 W)
\]

\[
= (g_0 W^2 + g_1 W) \quad (\text{B.0.13})
\]

and the combination of scaling (refer (B.0.11)) and squaring (refer (B.0.13)) gives,

\[
WT^2 = W(g_0 W^2 + g_1 W)
\]

\[
= (g_0 + g_1) W^2 + g_0 W
\]

Note that, \( WT \) (see (B.0.11)) and \( WT^2 \) (see (B.0.12)) require only 1 addition each. For normal basis, squaring in field \( GF(2^2) \), \( \Gamma^2 \) (see (B.0.13)), is a free operation, which requires only swapping. There are four choices of \( \nu \) that result in minimal construction of \( \nu \gamma^2 \) which is as summarised in Table B.2. This specific circuit requires only 2 additions, which are smaller than the one represented using polynomial basis.
Table B.2: Optimisation in multiplicative inversion using normal basis (trace unity), with \( \gamma = \gamma_1 z + \gamma_0 \) and \( \nu = \Delta_1 z + \Delta_0 \). Using Case ii and normal basis representation, the possible pairs of \( N \) and \( \nu \) that result in minimal complexity in operation \( \nu \gamma^2 \) are determined.

<table>
<thead>
<tr>
<th>Relation of ( \Delta_1 )</th>
<th>Relation of ( \Delta_0 )</th>
<th>Value of ( N )</th>
<th>Value of ( \nu )</th>
<th>( \nu \gamma^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \frac{1}{N} )</td>
<td>{01}_2</td>
<td>{1110}_2</td>
<td>( \nu \gamma^2 = [\Gamma_0^2] Z^4 + [NT_1^2 + \Gamma_0^2] )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{10}_2</td>
<td>{1101}_2</td>
<td></td>
</tr>
<tr>
<td>( \frac{1}{N} )</td>
<td>1</td>
<td>{01}_2</td>
<td>{1011}_2</td>
<td>( \nu \gamma^2 = [NT_0^2 + \Gamma_1^2] Z^4 + [\Gamma_1^2] )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{10}_2</td>
<td>{0111}_2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( \frac{1}{N} )</td>
<td>{01}_2</td>
<td>{0010}_2</td>
<td>( \nu \gamma^2 = [\Gamma_0 + \Gamma_1^2] Z^4 + [N^2 \Gamma_0^2] )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{10}_2</td>
<td>{0001}_2</td>
<td></td>
</tr>
<tr>
<td>( \frac{1}{N} )</td>
<td>0</td>
<td>{01}_2</td>
<td>{1000}_2</td>
<td>( \nu \gamma^2 = [N^2 \Gamma_1^2] Z^4 + [\Gamma_0 + \Gamma_1^2] )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{10}_2</td>
<td>{0100}_2</td>
<td></td>
</tr>
</tbody>
</table>

Table B.3: The best cases of CFA from Case i and Case ii

<table>
<thead>
<tr>
<th>Computation</th>
<th>Case i</th>
<th>Case ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicative inversion in ( GF(2^3) )</td>
<td>( d_1 = (\gamma_1 \nu + \gamma_1 \gamma_0 + \gamma_0) \gamma_1 )</td>
<td>( d_1 = (\gamma_1 \gamma_0 + (\gamma_1^2 + \gamma_0^2) \nu) \gamma_1 \gamma_0 )</td>
</tr>
<tr>
<td></td>
<td>( \Delta_1 = (\Gamma_1^2 N + \Gamma_1 \Gamma_0 + \Gamma_0^2)^{-1} \Gamma_1 )</td>
<td>( \Delta_1 = \Gamma_1 \Gamma_0 + (\Gamma_1^2 + \Gamma_0^2) \nu )</td>
</tr>
<tr>
<td></td>
<td>( \Delta_0 = (\Gamma_1^2 N + \Gamma_1 \Gamma_0 + \Gamma_0^2)^{-1}(\Gamma_0 + \Gamma_1) )</td>
<td>( \Delta_0 = \Gamma_0 \Gamma_0 + (\Gamma_1^2 + \Gamma_0^2) \nu )</td>
</tr>
<tr>
<td>NT(^2) = ( g_0 \nu + \gamma_1 )</td>
<td>NT(^2) = ( g_0 \nu + \gamma_1 \nu^2 + \gamma_0 )</td>
<td></td>
</tr>
<tr>
<td>Multiplicative inversion in ( GF(2^5) )</td>
<td>( d_1 = \gamma_1 )</td>
<td>( d_1 = \gamma_0 )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma_0 = \gamma_1 \gamma_0 + \gamma_0 )</td>
<td>( \Gamma_0 = \gamma_1 \gamma_0 + \gamma_0 )</td>
</tr>
<tr>
<td>Multiplication in ( GF(2^4) )</td>
<td>( [(\Gamma_1 + \Gamma_0)(\Delta_1 + \Delta_0) + \Gamma_0 \Delta_0] \nu )</td>
<td>( [(\Gamma_1 + \Gamma_0)(\Delta_0 + \Delta_1)N + \Gamma_1 \Delta_1] Z^4 )</td>
</tr>
<tr>
<td></td>
<td>( + (\Gamma_0 \Delta_0 + \Gamma_1 \Delta_1) N )</td>
<td>( + (\Gamma_1 + \Gamma_0)(\Delta_0 + \Delta_1) N + \Gamma_0 \Delta_0 )</td>
</tr>
<tr>
<td>Multiplication in ( GF(2^2) )</td>
<td>( [(\gamma_1 + \gamma_0)(\gamma_1 + \gamma_0)] \nu )</td>
<td>( [(\gamma_1 + \gamma_0)(\gamma_1 + \gamma_0) + (\gamma_1 \gamma_0)] W^2 )</td>
</tr>
<tr>
<td></td>
<td>( + (\gamma_1 \gamma_0) \nu )</td>
<td>( + (\gamma_1 \gamma_0)(\gamma_1 + \gamma_0 + (\gamma_1 \gamma_0)] W )</td>
</tr>
</tbody>
</table>
**APPENDIX C**

**Gujardo and Paar Inversion Algorithm Over** $GF((2^4)^{41})$

**Step 1: Exponentiation of** $A^r - 1 \in GF(2^4)$

Similar to the steps in Section 12.1.1, $A^r - 1$ can be computed through (12.1.5) with the complexity of 6 multiplications in $GF((2^4)^{41})$ and 40 exponentiations to the power of $2^4$. While exponentiation of a normal basis element can be done by cyclic shifting, exponentiation in polynomial basis requires modular reduction such as described next.

Let $B$ and $C$ be the elements of $GF((2^4)^{41})$. In order to find $C(x) = B^{2^4}$, where $B(x) = \sum_{i=0}^{40} b_i x^i$. This can be deduced as,

$$
C(x) = \sum_{i=0}^{40} c_i x^i \\
= \left( \sum_{i=0}^{40} b_i x^i \right)^{2^4} \\
= \sum_{i=0}^{40} b_i x^{2^i}, b_i \in GF(2^4) \tag{C.0.1}
$$

Following which, the power notation in residue classes modulo $P(x)$ can be derived as follows:

$$
x^{2^i} = s_{0,i} + s_{1,i}x + \cdots + s_{40,i}x^{40} \mod i = 1, \ldots, 40 \tag{C.0.2}
$$

Using the notation in (C.0.2), the exponentiation in (C.0.1) can be expressed in matrix
Computation in (C.0.3) would be simplified if \( P(x) \) is chosen to have only binary coefficients. As a result, all elements powers of \( x^n \mod P(x) \) belong to a subfield whose elements are represented by binary polynomials. Hence, all the coefficients \( s_{ij} \) in (C.0.3) are binaries. Therefore, one exponentiation in polynomial basis using the above mentioned algorithm requires \((40)^2/4\) additions in average. On the other hand, the multiplication in \( GF((2^4)^{41}) \) is performed using the KOA.

**Step 2: Multiplication of \( A \) and \( A^{r-1} \) that yield \( A^r \in GF(2^4) \)**

The multiplication in Section 12.1.2 is not applicable in polynomial basis elements. In this case, we need to consider pure polynomial multiplication \( A'(x) = B(x)C(x) \);

\[
A'(x) = \left( \sum_{i=0}^{40} b_i x^i \right) \cdot \left( \sum_{i=0}^{40} c_i x^i \right) = \left( \sum_{i=0}^{80} a_i' x^i \right) \quad \text{(C.0.4)}
\]

After the reduction modulo \( P(x) \), all except the first coefficient of \( A'(x) \) will be eliminated as following:

\[
A'(x) \equiv A(x) = a_0 \mod P(x) \quad \text{(C.0.5)}
\]

The reduction modulo \( P(x) \) can be viewed as linear mapping of the \( 2(41) - 1 \) coefficient of \( A'(x) \) into 41 coefficients of \( A(x) \) which can be noted as follows [155]:
Appendix C: Gujardo and Paar Inversion Algorithm over $GF((2^4)^{41})$

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{40} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \ldots & 0 & r_{0,0} & \ldots & r_{0,39} \\ 0 & 1 & \ldots & 0 & r_{1,0} & \ldots & r_{1,39} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & r_{40,0} & \ldots & r_{40,39} \end{bmatrix} \times \begin{bmatrix} a_0' \\ a_{40}' \\ a_{41}' \\ \vdots \\ a_{80}' \end{bmatrix} \quad \text{(C.0.6)}$$

The matrix in (C.0.6) consists of an $41 \times 41$ identity matrix combined with a $41 \times 40$ reduction matrix, which is a function of polynomial $P(x)$ [9]. It would be advantageous if $P(x)$ has only coefficient of $GF(2)$. As a result, the entries $r_{i,j}$ of the reduction matrix are binary, allowing $A(x)$ to be computed by additions in the ground field. Eventually, $A(x)$ is expressed as:

$$D(x) = d_0 = d_0' + \sum_{i=0}^{39} r_{0,i} a_m' \pmod{P(x)} \quad \text{(C.0.7)}$$

**Step 3: Multiplicative inversion in $GF(2^4)$ yields $(A')^{-1}$**

Multiplicative inversion of $A'$ involves $\log$ and $antilog$ conversion which requires two types of LUTs such as stated in (8.2.2) (refer Chapter 8). Therefore, this step can be explained analytically as the following,

$$A^{-r} = \text{antilog} \left[ - \log(A') \pmod{2^4 - 1} \right]. \quad \text{(C.0.8)}$$

**Step 4: Multiplication of $(A')^{-1} \cdot A'^{-1}$**

Two elements of the ground field $GF(2^4)$ can be multiplied using three LUTs operations and one addition modulo the order of the multiplicative group (exponent addition, EXPA). Therefore, the product of two elements $\omega_j, \omega_k \in GF(2^4)$ can be obtained such as the following by referring to (8.2.1)(refer Chapter 8).
\[(A_r)^{-1} \cdot A^{r-1} = \text{antilog} \left[ -\log((A_r)^{-1}) + \log(A^{r-1}) \pmod{2^4 - 1} \right]. \quad (C.0.9)\]