Applications of Finite Field Computation to Cryptology: Extension Field Arithmetic in Public Key Systems and Algebraic Attacks on Stream Ciphers

Kenneth Koon-Ho Wong

Bachelor of Applied Science (First Class Honours)
Queensland University of Technology, 2003

Thesis submitted in accordance with the regulations for the Degree of Doctor of Philosophy

Information Security Institute
Faculty of Information Technology
Queensland University of Technology

2008
Keywords

algebraic attacks, clock control, cyclotomic fields, CEILIDH, extension fields, Gauss periods, Karatsuba multiplication, Pomaranch, RC4, stream ciphers, torus-based cryptography, XTR
Abstract

In this digital age, cryptography is largely built in computer hardware or software as discrete structures. One of the most useful of these structures is finite fields. In this thesis, we explore a variety of applications of the theory and applications of arithmetic and computation in finite fields in both the areas of cryptography and cryptanalysis.

First, multiplication algorithms in finite extensions of prime fields are explored. A new algebraic description of implementing the subquadratic Karatsuba algorithm and its variants for extension field multiplication are presented. The use of cyclotomic fields and Gauss periods in constructing suitable extensions of virtually all sizes for efficient arithmetic are described. These multiplication techniques are then applied on some previously proposed public key cryptosystem based on extension fields. These include the trace-based cryptosystems such as XTR, and torus-based cryptosystems such as CEILIDH. Improvements to the cost of arithmetic were achieved in some constructions due to the capability of thorough optimisation using the algebraic description.

Then, for symmetric key systems, the focus is on algebraic analysis and attacks of stream ciphers. Different techniques of computing solutions to an arbitrary system of boolean equations were considered, and a method of analysing and simplifying the system using truth tables and graph theory have been investigated. Algebraic analyses were performed on stream ciphers based on linear feedback shift registers where clock control mechanisms are employed, a category of ciphers that have not been previously analysed before using this method. The results are successful
algebraic attacks on various clock-controlled generators and cascade generators, and a full algebraic analyses for the eSTREAM cipher candidate Pomaranch. Some weaknesses in the filter functions used in Pomaranch have also been found.

Finally, some non-traditional algebraic analysis of stream ciphers are presented. An algebraic analysis on the word-based RC4 family of stream ciphers is performed by constructing algebraic expressions for each of the operations involved, and it is concluded that each of these operations are significant in contributing to the overall security of the system. As far as we know, this is the first algebraic analysis on a stream cipher that is not based on linear feedback shift registers. The possibility of using binary extension fields and quotient rings for algebraic analysis of stream ciphers based on linear feedback shift registers are then investigated. Feasible algebraic attacks for generators with nonlinear filters are obtained and algebraic analyses for more complicated generators with multiple registers are presented. This new form of algebraic analysis may prove useful and thereby complement the traditional algebraic attacks.

This thesis concludes with some future directions that can be taken and some open questions. Arithmetic and computation in finite fields will certainly be an important area for ongoing research as we are confronted with new developments in theory and exponentially growing computer power.
Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed:........................................... Date:.........................
Acknowledgements

The quality and quantity of the research presented in this thesis would not have been achieved without my supervisors, Gary Carter and Ed Dawson, who have provided me with every help, support and encouragement throughout the seemingly. They have spent numerous hours with me on suggesting research directions, discussing problems and reviewing writings. I sincerely pay my highest regards to their kindness and professionalism.

I have been fortunate to be able to work with many other researchers during my research. I would like to thank Winfried Müller, who warmly hosted my visit for three weeks at the Department of Mathematics, University of Klagenfurt, Austria. I have gained much from working with colleagues while being there. I would like to thank Sultan Al-Hinai, Lynn Batten, Bernard Colbert, and Subhamoy Maitra, whom I have had the honour to meet in the last few years. They have provided me comments and suggestions to improve on various aspects of my research. In particular, I have enjoyed the close collaboration with Sultan Al-Hinai, my fellow PhD student, as well as his supervisors, Matt Henricksen, Bill Millan and Leonie Simpson, at the Information Security Institute. Together, Sultan and I have achieved two joint publications, where each of us contributed in our strength toward some nice results. The collaborative work appears in Sections 5.4-5.6 of this thesis. I would like to also thank Lynn Batten for her organisation in making the two publications possible, and also for inviting me to speak at one of her workshops. I would like to thank Gregory Bard and Richard Brent for providing valuable comments that have improved some of the work contained in Section 3.3 significantly.
My colleagues at both the Information Security Institute and the School of Mathematical Sciences, Faculty of Science have undoubtedly given me a warm atmosphere under which I can comfortably work on my research. I greatly appreciate their friendship to me and discussions that inspire me much.

I would like to thank the Australian Commonwealth Government, Queensland University of Technology, and the Information Security Institute for providing generous scholarships and subsidies for my studies. I would also like to thank the Information Security Institute and the School of Software Engineering and Data Communications, Faculty of Information Technology for providing me with both the opportunities and funds for my local, interstate and overseas travels and studies at various conferences and workshops.

I would also like to thank the internal review panel, comprising of Colin Boyd, Gary Carter, Ed Dawson and Ian Turner, for spending time to review my thesis and final seminar, and providing valuable comments to improve the quality of the thesis submission.

Last but not least, I would like to give sincere appreciation the Dean’s Scholars Program offered to me through the Faculty of Science as part of my undergraduate studies, which has prepared me the skills and knowledge to conduct research at the academic level, and paved my way towards the completion of a doctoral degree.
Previously Published Material


Contents

Keywords ................................................................. iii
Abstract ................................................................. v
Declaration ............................................................... vii
Acknowledgements .................................................... ix
Previously Published Material ........................................ xi

1 Introduction ......................................................... 1

1.1 Modern Cryptology .................................................. 2
  1.1.1 Finite Field Arithmetic in Prime Fields ...................... 2
  1.1.2 Algebraic Attacks in Binary Fields ............................ 2

1.2 Aims and Objectives ................................................ 3
  1.2.1 Extension Field Arithmetic ................................. 3
  1.2.2 Algebraic Analysis and Attacks ......................... 3

1.3 Main Outcomes .................................................... 4

1.4 Structure of Thesis ................................................. 5
2 Solving Equations over Finite Fields

2.1 Introduction ......................................................... 7

2.2 Linear Systems ..................................................... 8

2.2.1 Gaussian Elimination ......................................... 8

2.2.2 Solution Methods over Finite Fields ......................... 9

2.3 Univariate Polynomials ........................................... 9

2.3.1 Polynomial Factorisation and Root Finding .................. 10

2.3.2 Cantor-Zassenhaus Equal Degree Factorisation ............... 11

2.3.3 Common Roots of Univariate Polynomials .................... 13

2.4 Multivariate Polynomial Systems ................................. 14

2.4.1 Linearisation .................................................... 14

2.4.2 Gröbner Bases .................................................. 16

2.4.3 Truth Tables and Graphs ...................................... 20

2.5 Summary .......................................................... 23

3 Finite Field Arithmetic ............................................. 25

3.1 Introduction ....................................................... 25

3.2 Karatsuba Multiplication ......................................... 27

3.2.1 Traditional Recursive Algorithm ............................. 29

3.2.2 Generalised Recursive Algorithm ............................ 31
CONTENTS

3.3 Arithmetic in Extension Fields
   3.3.1 Polynomial Bases
   3.3.2 Normal Bases
3.4 Cyclotomic Fields
   3.4.1 Multiplication in Extensions of Degree $2^m$
   3.4.2 Multiplication in Even Extensions
3.5 Gauss Periods
   3.5.1 Multiplication with Single Extensions
   3.5.2 Multiplication with Multiple Extensions
3.6 Summary

4 Public Key Systems in Extension Fields
   4.1 Introduction
   4.1.1 Public Key Cryptography
   4.2 Trace-Based Cryptography
      4.2.1 LUC
      4.2.2 XTR
   4.3 Torus-Based Cryptography
      4.3.1 CEILIDH
      4.3.2 Systems Based on Higher Dimensional Tori
5 Algebraic Analysis of Stream Ciphers

5.1 Introduction ................................................. 63
5.1.1 Stream Ciphers ........................................... 64
5.1.2 Algebraic Analyses and Attacks ......................... 66
5.2 Stream Cipher Design ...................................... 67
5.2.1 Linear Feedback Shift Registers ......................... 67
5.2.2 Expressions for Register States ......................... 68
5.2.3 Expressions for Nonlinear Components ................ 70
5.3 Algebraic Attacks ........................................... 72
5.3.1 Traditional Algebraic Attacks ......................... 72
5.3.2 Improving Algebraic Attacks ......................... 74
5.4 Clock Controls .............................................. 75
5.4.1 Algebraic Analysis of Clock-Controlled Registers .... 77
5.5 Clock-Controlled Stream Ciphers ......................... 80
5.5.1 The Stop-and-Go Generator ......................... 81
5.5.2 The Step-1/Step-2 Generator ......................... 82
5.5.3 The Alternating Step Generator ..................... 85
5.5.4 The Self-Decimated Generator ..................... 87
CONTENTS

5.5.5 The Strengthened Beth-Piper Generator ............... 89

5.6 Clock-Controlled Cascade Ciphers ......................... 92
  5.6.1 The Gollmann Cascade Generator ....................... 94
  5.6.2 Variable Relabelling ................................. 98
  5.6.3 Pomaranch ............................................. 100
  5.6.4 Algebraic Analysis of Pomaranch ....................... 102
  5.6.5 The Filter Functions in Pomaranch ..................... 108

5.7 Further Notes ............................................. 110

5.8 Summary ................................................. 111

6 Algebraic Analyses with Binary Fields ..................... 113
  6.1 Introduction ............................................. 113

6.2 The RC4 Family of Stream Ciphers ......................... 115
  6.2.1 Description of RC4 .................................... 116
  6.2.2 Algebraic Analysis of RC4 ............................. 117
    State Extraction ........................................... 118
    Word Addition .............................................. 119
    State Permutation ......................................... 120

6.2.3 Equation Generation ..................................... 122
    Pointer Increment .......................................... 122
CONTENTS

Pointer Addition ........................................ 123
State Permutation ...................................... 124
Keystream Generation ................................. 126

6.2.4 Summary of Equations in RC4 .................. 127
6.2.5 RC4 as an Algebraic Cipher .................... 127
6.2.6 Algebraic Attacks on RC4 ...................... 128
6.2.7 Section Summary ................................. 129

6.3 Extension Field Algebraic Analysis .............. 129
6.3.1 Introduction ..................................... 130
6.3.2 Algebraic Analysis of LFSR Generators ........ 130
6.3.3 Non-Linear Components ......................... 133
6.3.4 Extension Field Elements ....................... 136
6.3.5 Multiple Registers .............................. 137
6.3.6 Comparison with Traditional Algebraic Attacks . 140
6.3.7 Section Summary ................................. 141

6.4 Summary ........................................... 142

7 Conclusion ............................................ 143

7.1 Summary of Thesis .................................. 143
7.2 Future Directions ................................... 145
CONTENTS

7.3 Open Questions ........................................ 146
7.4 Closing Remarks ........................................ 147

A Algebraic Preliminaries 149
A.1 Abstract Algebra ........................................ 149
   A.1.1 Quotients ........................................... 151
   A.1.2 Field Theory ....................................... 153
A.2 Number Theory .......................................... 154
A.3 Commutative Algebra .................................... 154
   A.3.1 Finite Fields ....................................... 156
   A.3.2 Gröbner Bases ..................................... 157
A.4 Ideals and Varieties ..................................... 159

B Tables 161
B.1 List of Cyclotomic Fields ............................... 161
B.2 List of Gaussian Normal Bases ....................... 161
List of Figures

5.1 Schematic of a Stream Cipher .............................................. 64
5.2 A Bit-Based Linear Feedback Shift Register ............................. 68
5.3 A Nonlinear Filter Generator .............................................. 71
5.4 A Clock-Controlled Generator .............................................. 76
5.5 The Stop-and-Go Generator ................................................ 81
5.6 The Step-1/Step-2 Generator .............................................. 82
5.7 The Alternating Step Generator ............................................ 85
5.8 The Self-Decimated Generator .............................................. 88
5.9 The Strengthened Beth-Piper Generator ................................ 90
5.10 The Gollman Cascade Generator .......................................... 94
5.11 The Pomaranch Stream Cipher (Version 3) .............................. 101
5.12 The Pomaranch S-Box ...................................................... 104
5.13 The Algebraic Normal Form (ANF) of Filter Function $f$ .......... 105
5.14 Annihilators and Low Degree Multiples $f$ in Pomaranch .......... 109
5.15 Annihilators and Low Degree Multiples of $g$ in Pomaranch . . . . . 110

6.1 The KSA and PRNG of RC4 . . . . . . . . . . . . . . . . . . . . . . . . . . . 116

6.2 A Nonlinear Filter Generator . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Cyclotomic Fields $\mathbb{F}_{q^n}$ for $2 \leq n \leq 16$</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Recursive Karatsuba multiplication over cyclotomic fields $\mathbb{F}_{p^{2m}}$</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Traditional Recursive Karatsuba Algorithm over $\mathbb{F}_{p^{2t}}$</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Generalised Recursive Karatsuba Algorithm over $\mathbb{F}_{p^{2t}}$</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>Gaussian Normal Bases for $2 \leq n \leq 7$ with Least $k$</td>
<td>42</td>
</tr>
<tr>
<td>3.6</td>
<td>Multiplication with Gaussian normal bases</td>
<td>43</td>
</tr>
<tr>
<td>3.7</td>
<td>Constructing Multiple Extensions</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Arithmetic Costs in LUC in [90]</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Arithmetic Costs in XTR in [90]</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Arithmetic Costs of CEILIDH in [45]</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>Optimised Arithmetic Costs of CEILIDH</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>Cost of Karatsuba Arithmetic in $\mathbb{F}_{p^6}$</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>Arithmetic Costs in the $T_{30}$ System in [94]</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>Optimised Arithmetic Costs in the $T_{30}$ System</td>
<td>60</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8</td>
<td>Primitive Arithmetic Costs in the $T_{210}$ System</td>
</tr>
<tr>
<td>4.9</td>
<td>Towered Arithmetic Costs in the $T_{210}$ System</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparison of Attack Complexities on the Step1/Step2 Generator</td>
</tr>
<tr>
<td>5.2</td>
<td>Attack Times for the Step1/Step2 Generator</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparison of Attacks on the Alternating Step Generator</td>
</tr>
<tr>
<td>5.4</td>
<td>Algebraic Attack Times for the Alternating Step Generator</td>
</tr>
<tr>
<td>5.5</td>
<td>Attack Complexities on the Self-Decimated Generator</td>
</tr>
<tr>
<td>5.6</td>
<td>Algebraic Attack Times on the Self-Decimated Generator</td>
</tr>
<tr>
<td>5.7</td>
<td>Comparison of Attacks on the Strengthened Beth-Piper Generator</td>
</tr>
<tr>
<td>5.8</td>
<td>Attack Times for the Strengthened Beth-Piper Stop-and-Go Generator</td>
</tr>
<tr>
<td>5.9</td>
<td>Comparison of Attack Complexities on the Gollmann Cascade Generator</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary of Equations Generated for RC4</td>
</tr>
<tr>
<td>6.2</td>
<td>Algebraic Attacks on the Filter Generator</td>
</tr>
<tr>
<td>B.1</td>
<td>Cyclotomic Fields for $2 \leq n \leq 88$</td>
</tr>
<tr>
<td>B.2</td>
<td>Gaussian Normal Bases for $2 \leq n \leq 14$</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

As we know it today, cryptology primarily deals with discrete structures and algebraic manipulations inside hardware and software. Finite fields are well-studied discrete structures with a vast array of useful properties and are indispensable in the theory and application of cryptology. Efficient computation in finite fields is crucial for the feasibility of cryptographic systems built on them, and also for the successful cryptanalyses of such systems. Research progress in the arithmetic and computation in finite fields with a view to improving cryptological processes is constantly being made. This thesis intends to become part of the endeavour to analyse more deeply and improve on cryptology over finite fields.

The theory and applications of arithmetic over finite fields have been a major research area, particularly since the advent of public key cryptography. Recently, public key systems based on extension fields have been proposed. In this thesis, we present a structural analysis of extension field multiplication, in an attempt to achieve the most optimised algorithm based on Karatsuba arithmetic.

In relatively recent times, a powerful new technique has been added to the cryptanalysis arsenal. Algebraic analysis, as it is known, essentially represents what is happening inside a cryptosystem as a system of equations and then proceeds to solve these equations to reveal the keys or initial states of the system. In this
thesis, this technique is applied to ciphers not previously analysed in this way, and
a new approach to generating the system of equations is presented.

1.1 Modern Cryptology

Two primary categories of cryptographic schemes exist in the modern setting of
cryptology. They are symmetric key schemes for fast encryption and decryption,
and public key schemes for key exchange and protocols. In this thesis, we in-
vestigate certain aspects of both of these schemes in relation to arithmetic and
computation in finite fields. We will mainly study efficient field arithmetic in pub-
lic key cryptography, and algebraic attacks as part of symmetric key cryptanalysis.

1.1.1 Finite Field Arithmetic in Prime Fields

The prime fields $\mathbb{F}_p$, having the simplest representation of all finite fields, simply
behave as integers modulo $p$. The arithmetic in prime fields forms a basis for all
algorithms in other finite fields. For example, arithmetic in extension fields $\mathbb{F}_{p^n}$
can be performed entirely using algorithms built on modulo arithmetic. Public
key systems based on various discrete logarithm problems are frequently imple-
menced over finite fields or curves defined over finite fields, to provide structure
and efficient arithmetic. In this thesis, we will present new efficient arithmetic for
use in extension fields.

1.1.2 Algebraic Attacks in Binary Fields

The binary prime field $\mathbb{F}_2$, which acts in the same way as a boolean algebra, serves
as a great tool for development and analysis of symmetric ciphers, since many
of them can be described using boolean functions. The binary extension fields
$\mathbb{F}_{2^n}$ are used in both public key cryptography in implementing efficient arithmetic,
and in symmetric key cryptography in designing cipher components. Algebraic attacks on symmetric ciphers rely heavily on the properties of binary fields for the equation generation and solution. In this thesis, we will use algebraic attacks to analyse a collection of stream ciphers not previously analysed and comment on their susceptibility to these forms of attacks.

## 1.2 Aims and Objectives

The overall aim of this thesis is to investigate the relationship of finite fields to both cryptography and cryptanalysis. In cryptography, the aim is to improve the arithmetic on extension fields and study the consequent positive effect on the implementation of public key systems based on extension fields. In cryptanalysis, the aim is to apply traditional algebraic analysis to a suite of ciphers and to develop improved means of generating and solving systems of equations representing the actions of ciphers.

### 1.2.1 Extension Field Arithmetic

The arithmetic of finite extensions of prime fields used in public key cryptography is studied. Since there are many representations of extension fields, an analysis into which ones can result in efficient arithmetic will be very useful. It is intended that a systematic construction of extension fields will be carried out and it is believed that this will result in more efficient algorithms for arithmetic.

### 1.2.2 Algebraic Analysis and Attacks

Algebraic analysis and attacks are to be performed on bit-based clock-controlled stream ciphers, a class of stream ciphers that has not been widely analysed algebraically. It is believed that some ciphers in this class may be susceptible to
algebraic attacks. This study will start with basic clock-controlled generators, and progress to ciphers for industry use.

To date, algebraic analysis has been fairly narrowly applied. It is the intention of this study to look for other novel uses of algebraic analysis, which can extend its range of applications. As an aside, the theory of generating and solving equations in binary fields as part of an algebraic attack will be studied, with a view to improve on currently available methods.

1.3 Main Outcomes

A novel contribution to the development of the theory of extension field arithmetic through the Karatsuba multiplication algorithm and its variants have been made. The processes involved in extension field multiplication and the methods of optimisation have been systematically analysed, and a new algebraic description of the construction of suitable extension fields and their corresponding field multiplication algorithms have been developed. These multiplication algorithms are then applied to public key systems based on extension fields and compared with previously known results.

The methods of algebraic analysis attacks have been extended to cryptanalyse some special types of stream ciphers based on linear feedback shift registers. These include the clock-controlled stream ciphers and cascaded clock-controlled stream ciphers. To our knowledge, this has not been done previously. New methods of generating and simplifying equations describing the ciphers and experiments on solving these equations have been documented. An algebraic analysis of Pomaranch, an eCRYPT stream cipher project candidate, has also been performed.

Some new and interesting techniques in the application of finite field algebraic cryptanalysis were discovered. These show that some unconventional stream ciphers, like the RC4 family of stream ciphers, can be algebraically analysed. An alternative form of algebraic analysis using extension fields and related structures
1.4 Structure of Thesis

In Chapter 2, various methods of solving equations over finite fields are presented. We also introduce some novel ideas of simplifying equations, which could be useful in increasing the efficiency of solving certain systems. These include utilising the truth tables of the boolean functions in the systems, and graph theory techniques on the boolean variables.

Chapter 3 presents a systematic analysis on extension field multiplication is performed. An algebraic description of the Karatsuba algorithm for polynomial multiplication is given, and it is applied to multiplication of extension field elements. It is shown that due to the reduction of elements to their minimal representations, multiplication in extension fields can be optimised beyond what multiplication of polynomials can achieve. A description of the construction of extension fields with appropriate bases is presented, such that efficient arithmetic using the Karatsuba algorithm can be used. These include the use of cyclotomic fields and Gauss periods, as well as their combinations. The results from this chapter were submitted as a paper to the proceedings of Finite Fields and Applications conference held in 2007 in Melbourne, Australia.

The results in Chapter 3 are applied to various public key systems based on extension fields in Chapter 4. These include the trace-based cryptosystems and torus-based cryptosystems. We show that the extension field arithmetic derived using our methods are at least as efficient as results obtained in the literature. Partial results were published in [101].

Algebraic cryptanalysis of stream ciphers based on linear feedback shift registers is the topic of Chapter 5. We focus on a class of such ciphers for which algebraic attacks have not been readily applied, namely irregularly clocked generators. These
include the various clock-controlled generators and cascaded clock-controlled generators. As an end result, an algebraic analysis of the stream cipher Pomaranch, whose hardware implementation is in phase three of the eCRYPT stream cipher project, is presented. The results from this chapter have been published in [2, 102].

In the penultimate Chapter 6, some novel ideas in algebraic cryptanalysis are investigated. As far as we know, the first algebraic analysis of the RC4 family of stream ciphers is presented here. By analysing the operations involved in the cipher, equations are derived representing the cipher, it is then shown how they miraculously interact with each other to give a high overall security against algebraic attacks. A method of algebraic analysis using extension fields and related structures is then presented, as opposed to the traditional algebraic analysis where boolean functions are usually used. It is shown that feasible algebraic attacks can be launched on keystream generators based on linear feedback shift registers. The systems of equations derived can be univariate, and in general less memory is required to solve them compared to the traditional method. The results from this chapter were submitted to the Australasian Journal of Combinatorics and the proceedings of Finite Fields and Applications held in 2007 in Melbourne, Australia.

Finally, Chapter 7 summaries the main findings of the research, and draws conclusions are drawn about their significance in their respective fields. Future directions to which this research can be taken are also discussed, followed by the proposal of some open questions.

Two appendices are provided. Appendix A gives a treatment of the algebraic concepts used in this thesis. Appendix B provides tables for the construction of extension fields for efficient arithmetic, as investigated in Chapter 3.
Chapter 2

Solving Equations over Finite Fields

Over the fields of real and complex numbers, equations or systems of equations can be solved by many of the available efficient direct and indirect methods from the area of numerical analysis. This, however, is not the case for finite fields. The difficulty of solving systems of equations in finite fields is well known, and has been identified as a basis for security in many cryptographic algorithms [88]. In this chapter, methods of solving various kinds of equations and equation systems over finite fields are presented, and novel ideas in equation solving, which can lead to simplification of the equations systems and efficiency gains in computing their solutions, are discussed.

2.1 Introduction

It is widely believed that the security of cryptographic systems can be translated to the difficulty of solving large systems of equations over finite fields. The relationship between cryptographic security and systems of equations has become more apparent since the introduction of index calculus methods for cryptanalysis
of public key systems, and algebraic analysis and attacks on symmetric key systems. In recent years, much effort has been made to improve the efficiency of the methods for solving these systems of equations over finite fields, and implications on security of cryptographic systems have been constantly evaluated [61, 82, 103].

Three kinds of equations and equation systems, their solution methods, and their uses in cryptology will be presented in this chapter. These are the linear systems in Section 2.2, the univariate polynomial equations and systems in Section 2.3, and the multivariate polynomial systems in Section 2.4. For multivariate polynomial systems, some novel methods using truth tables and graph theory to simplify large systems of equations are developed, which allow the solutions to be computed more efficiently.

2.2 Linear Systems

Linear systems are one of the most frequently used tools in cryptanalysis. They are found in index calculus methods for the solution of discrete logarithm problems in public key systems, as well as in algebraic attacks where linearisation is used to convert a multivariate polynomial system into a linear one. Expressed in matrix form, equations in a linear system can be solved simultaneously for its variables.

2.2.1 Gaussian Elimination

Gaussian elimination is a systematic method for reducing a matrix to its echelon or reduced echelon form. For a square matrix of size $n$, the asymptotic complexity of Gaussian elimination is $O(n^\omega)$, where $\omega$ depends on the matrix multiplication algorithm used. For classical matrix multiplication, $\omega = 3$. The value of $\omega$ was first reduced to $\log_2 7$ by Strassen, whose algorithm [91] remains a popular choice for Gaussian elimination. The Coppersmith-Winograd algorithm [18] achieves the currently lowest value of $\omega = 2.376$, but is not widely implemented due to its high complexity coefficient. These subcubic complexities are often quoted in index
2.3. **UNIVARIATE POLYNOMIALS**

calculus and linearisation methods. Since there are \( n^2 \) entries in a matrix product, the lower bound for \( \omega \) is 2, which has not been achieved for generic matrices. However, for sparse matrices, the complexity of Gaussian elimination can be near quadratic.

Unless otherwise specified, throughout the thesis it is assumed that Gaussian elimination is performed with the worst case complexity with \( \omega = 3 \). It is noted that most systems we deal with are large and sparse, and the value of \( \omega \) for solving these systems can be significantly better than the worst case.

### 2.2.2 Solution Methods over Finite Fields

In general, efficient methods of solving linear systems over the complex field cannot be used in finite fields. However, more sophisticated methods have been developed specifically to improve the efficiency of computing solutions to linear systems over finite fields. The method of structured Gaussian elimination [77] can be used to convert a large sparse linear system into an equivalent small dense system, which can be processed using less storage. The Lanczos method for solving linear systems is an iterative method that was originally used for computing eigenvalues of a matrix [62]. It has since been adapted to solve linear systems over finite fields [61]. The conjuate gradient method is very similar to the Lanczos method. The Wiedermann method [99] tries to reconstruct the characteristic polynomial of the matrix representing the linear system, using the efficient Berlekamp-Massey algorithm for linear recurrences. All of these methods have been experimentally shown to be feasible at solving large linear systems over finite fields, and have been used in factorisation and index calculus algorithms [52, 53].

### 2.3 Univariate Polynomials

In a polynomial ring \( \mathbb{F}_q[x] \), obtaining the \( \mathbb{F}_q \) roots of a univariate polynomial \( f \in \mathbb{F}_q[x] \) is closely related to factoring \( f \), since each linear factor \((x - \gamma)\) of \( f \) gives
the root \( \gamma \). Several methods for finding roots of univariate polynomial equations over finite fields are available. The simplest approach is an exhaustive search algorithm over the solution space \( \mathbb{F}_q \). This is known in coding theory as Chien’s search [15]. In practice, roots are generally obtained from polynomial factorisation methods.

### 2.3.1 Polynomial Factorisation and Root Finding

Let \( \mathbb{F}_q[x] \) be a univariate polynomial ring in the indeterminate \( x \) over the finite field \( \mathbb{F}_q \) of \( q \) elements, and \( f \) be a polynomial in \( \mathbb{F}_q[x] \) of degree \( d \). The polynomial \( f \) can be factorised as

\[
f = \prod_{r=1}^{d} \prod_{i=1}^{s_r} f_{r,i}^{e_{r,i}}, \quad f_{r,i} \in \mathbb{F}_q[x],
\]

where the polynomials \( f_{r,1}, f_{r,2}, \ldots, f_{r,s_r} \) are the \( s_r \) factors of degree \( r \), having multiplicities \( e_{r,1}, e_{r,2}, \ldots, e_{r,s_r} \) respectively. The number of factors and their multiplicities must satisfy

\[
\sum_{r=1}^{d} r \sum_{i=1}^{s_r} e_{r,i} = d.
\]

**Definition 2.1.** Let \( \mathbb{F}_q[x] \) be a univariate polynomial ring over the finite field \( \mathbb{F}_q \). A polynomial \( f \in \mathbb{F}_q[x] \) is **squarefree** if and only if \( g^2 \nmid f \) for all \( g \in \mathbb{F}_q[x] \).

**Definition 2.2.** Let \( \mathbb{F}_q[x] \) be a univariate polynomial ring over the finite field \( \mathbb{F}_q \). A polynomial \( f \in \mathbb{F}_q[x] \) is an **equal-degree polynomial** if and only if all irreducible factors of \( f \) in \( \mathbb{F}_q[x] \) are of the same degree.

The algorithms for polynomial factorisation mainly consist of three stages.

**Squarefree Factorisation (SFF)** Given a monic polynomial \( f \in \mathbb{F}_q[x] \) of degree \( d \), SFF computes unique monic squarefree and pairwise coprime polynomials \( g_1, g_2, \ldots, g_d \) such that

\[
f = \prod_{i=1}^{d} g_i.
\]
Distinct Degree Factorisation (DDF) Given a monic polynomial \( f \in \mathbb{F}_q[x] \) of degree \( d \), DDF computes unique monic polynomials

\[
f = \prod_{r=1}^{d} h_r
\]

such that each \( h_r \) is an equal-degree polynomial with irreducible factors of degree \( r \).

Equal Degree Factorisation (EDF) Given a squarefree equal-degree polynomial \( f \in \mathbb{F}_q[x] \) of degree \( d \), EDF computes all of its irreducible factors.

As far as root finding in \( \mathbb{F}_q[x] \) is concerned, we need only to compute the linear factors of \( f \). Observe that

\[
x^{q^n} - x = x(x^{q^n-1} - 1) = \prod_{\gamma \in \mathbb{F}_q} (x - \gamma)
\]

is the product of all possible linear polynomials in \( \mathbb{F}_q[x] \) with each polynomial appearing exactly once. Therefore, the polynomial

\[
g = \gcd(f, x^{q^n-1} - 1) \in \mathbb{F}_q[x]
\]

is squarefree and is the product of all linear factors of \( f \) except for \( x \). It is then only necessary to perform equal degree factorisation on \( g \) to recover all roots of \( f \) in \( \mathbb{F}_q \).

2.3.2 Cantor-Zassenhaus Equal Degree Factorisation

From here on, root finding in binary fields \( \mathbb{F}_{2^n} \) will be specifically discussed for the purpose of algebraic analysis of stream ciphers. As discussed above, it is possible to obtain a squarefree equal-degree univariate polynomial in \( \mathbb{F}_{2^n}[x] \) with only linear factors, whose roots in \( \mathbb{F}_{2^n} \) are to be sought. Let such a polynomial be expressed
as
\[ f = \sum_{i=1}^{2^n-1} a_i x^i, \quad a_i \in \mathbb{F}_{2^n}. \] (2.3.7)

Here, the Cantor-Zassenhaus algorithm in \( \mathbb{F}_{2^n}[x] \) for equal degree factorisation is presented. Suppose \( f \) is of degree \( d \) and has roots \( \gamma_1, \gamma_2, \ldots, \gamma_d \in \mathbb{F}_{2^n} \), then
\[ f = \prod_{i=1}^{d} (x - \gamma_i), \quad \gamma_i \in \mathbb{F}_{2^n}. \] (2.3.8)

By the Chinese remainder theorem, the following isomorphism of quotient rings holds.
\[ \mathbb{R} \equiv \mathbb{F}_{2^n}[x]/\langle f \rangle \cong \bigoplus_{i=0}^{d-1} \mathbb{F}_{2^n}[x]/\langle x - \gamma_i \rangle \] (2.3.9)

Since each linear polynomial \( x - \gamma_i \) is irreducible, each quotient ring in the above direct sum is isomorphic to \( \mathbb{F}_{2^n} \). Therefore, every element \( \alpha \in \mathbb{R} \) satisfies
\[ \alpha^{2^n} - \alpha = 0. \] (2.3.10)

Define the polynomial trace map in \( \mathbb{F}_{2^n}[x] \) over \( \mathbb{F}_2[x] \) as
\[ \text{Tr} : \mathbb{F}_{2^n}[x] \rightarrow \mathbb{F}_2[x] \]
\[ a \mapsto \sum_{i=0}^{n-1} a^2^i. \]

Then, the following factorisation over \( \mathbb{F}_{2^n}[x] \) can be observed.
\[ a^{2^n} - a = \text{Tr}(a)(\text{Tr}(a) + 1), \quad a \in \mathbb{F}_{2^n}[x] \] (2.3.11)

By (2.3.10), this means that \( \mathbb{R} \) is not an integral domain, as expected, since \( f \) is reducible in \( \mathbb{F}_{2^n}[x] \). More importantly, \( \gcd(f, \text{Tr}(a)), \gcd(f, \text{Tr}(a) + 1) \) are factors of \( f \). It can be shown that computing
\[ g = \gcd(f, \text{Tr}(a)). \] (2.3.12)
for a random \( a \in \mathbb{F}_2^n \) gives a factor with probability at least \( \frac{1}{2} \). Once \( g \) is known, the Cantor-Zassenhaus algorithm can be repeated for \( \frac{L}{g} \) to give a second factor of \( f \). This process can be continued until all factors are recovered. See [36] for a more thorough treatment of this algorithm.

### 2.3.3 Common Roots of Univariate Polynomials

From the factorisation of a polynomial \( f \in \mathbb{F}_{2^n}[x] \), The \( \mathbb{F}_2 \) roots of \( f \) can be deduced from the linear factors. If there are \( m \) polynomials \( f_1, f_2, \ldots, f_m \in \mathbb{F}_2[x] \) whose common roots in \( \mathbb{F}_2 \) are to be sought, factoring each \( f_i \) to compute the respective roots, and then extracting the common ones will achieve this goal. However, it is time consuming to factor each polynomial. Since the common \( \mathbb{F}_{2^n} \) roots \( r_1, r_2, \ldots, r_s \) of \( f_1, f_2, \ldots, f_m \) must constitute a common factor

\[
u = \prod_{i=1}^{s} (x - r_s) \in \mathbb{F}_2[x]
\]

(2.3.13)

of \( f_1, f_2, \ldots, f_m \), this means that we have

\[u \mid \gcd(f_1, f_2, \ldots, f_m).\]  

(2.3.14)

Specifically, we have

\[uv = \gcd(f_1, f_2, \ldots, f_m),\]  

(2.3.15)

where \( u \) is a product of common linear factors of \( f_1, f_2, \ldots, f_m \) as defined above, and \( v \) is a product of nonlinear common factors of \( f_1, f_2, \ldots, f_m \) that are irreducible in \( \mathbb{F}_{2^n}[x] \), whose roots are not in \( \mathbb{F}_{2^n} \). It can then be seen that \( u \) can be computed by

\[u = \gcd(f_1, f_2, \ldots, f_m, x^{2^n} - x),\]

(2.3.16)

since \((x^{2^n} - x)\) is a product of exactly all linear polynomials in \( \mathbb{F}_{2^n}[x] \). Factoring \( u \) gives the common linear factors and in turn the common roots of \( f_1, f_2, \ldots, f_m \).

The greatest common divisor of two polynomials is often computed by the Eu-
clidean algorithm, which is one of the oldest algorithms in use today for computing greatest common divisors in Euclidean domains. Since $\mathbb{F}_{2^n}[x]$ is an Euclidean domain, we can use the Euclidean algorithm to compute common factors and in turn the common roots of polynomials. The time complexity of the Euclidean algorithm on polynomials of degree $n$ is $O(n)$ divisions, which corresponds to at most $O(n^2)$ operations in $\mathbb{F}_{2^n}[x]$. Furthermore, in characteristic two fields, the Euclidean algorithm can be implemented more efficiently. The Euclidean GCD algorithm will be used for computation of common polynomial roots in the extension field algebraic analysis of stream ciphers in Chapter 6.

### 2.4 Multivariate Polynomial Systems

In this section, several methods of solving multivariate polynomial systems of equations are presented. These systems constantly arise in algebraic analysis. The solutions of these systems are discussed in the context of algebraic attacks against stream ciphers, targeting systems of boolean equations in particular. Novel methods of computing solutions and simplifying the systems of equations will also be introduced.

#### 2.4.1 Linearisation

Linearisation involves converting a system of multivariate polynomial equations into a linear system of equations with linearised variables. Consider a multivariate polynomial system

$$
\begin{align*}
  f_1(x_1, x_2, \ldots, x_n) &= b_1 \\
  f_2(x_1, x_2, \ldots, x_n) &= b_2 \\
  \vdots & \quad \vdots \\
  f_m(x_1, x_2, \ldots, x_n) &= b_m.
\end{align*}
$$

(2.4.1)

Let $\mathbf{v}$ be the vector of monomials present in $f_1, f_2, \ldots, f_m$. Let $\mathbf{w}$ be a vector of linear variables, such that each monomial in $\mathbf{v}$ becomes a new linear variable in
w. Construct a matrix $A$ such that

$$Av = b. \quad (2.4.2)$$

Since $w$ is just a relabelling of $v$, we must also have

$$Aw = b. \quad (2.4.3)$$

The system of multivariate polynomial equations becomes a system of linear equations in the new variables. This process of introducing these new linear variables is called linearisation. The linear system can then be solved using conventional techniques such as Gaussian elimination. The computation time depends on the length of $v$, which is the number of monomials present in $f_1, f_2, \ldots, f_n$. For a system of equations of maximum degree $d$, the number of monomials is of

$$O(m) = O\left(\sum_{i=0}^{d} \binom{n}{i}\right) \quad (2.4.4)$$

Note that to obtain a solution to the original multivariate polynomial system, it is only necessary to compute the values of those variables in $w$ that represent the linear variables $x_1, x_2, \ldots, x_n$ in $v$. However, it is unknown how this can be done without computing the rest of the solution to the linear system.

**Example 2.3.** Let the polynomial ring be $\mathbb{F}_2[x_1, x_2]$. The solutions to the bivariate boolean polynomial system in the variables $x_1, x_2 \in \mathbb{F}_2$ below is sought.

\begin{align*}
x_1x_2 &= 0 \quad (2.4.5) \\
x_1x_2 + x_1 &= 0 \quad (2.4.6) \\
x_1 + x_2 &= 1. \quad (2.4.7)
\end{align*}

The monomials present in this system are

$$v = (x_1, x_2, x_1x_2)^T. \quad (2.4.8)$$
These can be relabelled as the linear variables

\[ \mathbf{w} = (w_1, w_2, w_3)^T = (x_1, x_2, x_1x_2)^T, \]  
(2.4.9)

A linear system can then be generated as follows.

\[ \begin{align*}
    w_3 &= 0 \quad (2.4.10) \\
    w_1 + w_3 &= 0 \quad (2.4.11) \\
    w_1 + w_2 &= 1 \quad (2.4.12)
\end{align*} \]

The corresponding matrix equation is then

\[
\begin{pmatrix}
    1 & 0 & 0 \\
    1 & 1 & 0 \\
    0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    w_1 \\
    w_2 \\
    w_3
\end{pmatrix}
= 
\begin{pmatrix}
    0 \\
    1 \\
    0
\end{pmatrix}
\]
(2.4.13)

Using Gaussian elimination or similar methods, the solution to the above equation can be computed as

\[ \mathbf{w} = (0, 1, 0)^T. \]  
(2.4.14)

This means that the solution to the original bivariate polynomial system is

\[ \mathbf{x} = (0, 1)^T. \]  
(2.4.15)

### 2.4.2 Gröbner Bases

Gröbner bases were first introduced in [12] as a tool for computation with multivariate polynomial ideals. For background on Gröbner bases, see Appendix A. It was then discovered that Gröbner bases can also be used to analytically solve a system of multivariate polynomial equations though its elimination property, which is analogous to Gaussian elimination for linear systems. The main results of the elimination property from [6] is presented here.

**Definition 2.4.** Let \( k \) be a field, and \( I \subset k[x_1, \ldots, x_n] \) be an ideal of the multi-
2.4. MULTIVARIATE POLYNOMIAL SYSTEMS

Multivariate polynomial ring \( k[x_i, \ldots, x_n] \) in \( n \) indeterminates. The \( j \)-th elimination ideal of \( I \) is

\[
I_j = I \cap k[x_{j+1}, \ldots, x_n].
\]  

(2.4.16)

**Theorem 2.5 (The Elimination Property).** Let \( G \) be a Gröbner basis of \( I \) with respect to the lexicographic order with \( x_1 \succ x_2 \succ \ldots \succ x_n \). Then, for every \( 0 \leq j \leq n \), the set

\[
G_j = G \cap k[x_{j+1}, \ldots, x_n]
\]  

(2.4.17)

is a Gröbner basis of \( I_j \).

From the ideal-variety correspondence, solving a system of multivariate equations can be converted to finding simple generators of ideals. This means that Gröbner bases can be used to successively eliminate variables in multivariate systems.

Our interest in Gröbner bases methods is in its use for solving multivariate boolean equations for algebraic attacks on stream ciphers. In recovering an \( n \)-bit key or initial states \( x_i \) of a cipher with algebraic attacks, a system of multivariate polynomial equations over with boolean variables in \( \mathbb{F}_2 \) are generated. The polynomials are in fact in the quotient ring

\[
R = \frac{\mathbb{F}_2[x_0, x_1, \ldots, x_{n-1}]}{(x_0^2 + x_0, x_1^2 + x_1, \ldots, x_{n-1}^2 + x_{n-1})}
\]  

(2.4.18)

where the \( x_i^2 + x_i \) are called the field equations of \( R \). Here, the consideration is the implementation of Gröbner bases methods specifically for this structure. Let a system of \( m \) equations be

\[
\begin{align*}
  f_0(x_0, x_1, \ldots, x_n) &= 0 \\
  f_1(x_0, x_1, \ldots, x_n) &= 0 \\
  &\vdots \quad \vdots \\
  f_{m-1}(x_0, x_1, \ldots, x_n) &= 0
\end{align*}
\]  

(2.4.19)

where each \( x_i \in \mathbb{F}_2 \). This means the following field equations can be added into
the equation system.
\[
\begin{align*}
x_0^2 + x_0 &= 0 \\
x_1^2 + x_1 &= 0 \\
&\vdots \\
x_{n-1}^2 + x_{n-1} &= 0.
\end{align*}
\] (2.4.20)

Let
\[
g_i = x_i^2 + x_i, \quad 0 \leq i \leq n - 1.
\] (2.4.21)

Then, the ideal \(\langle f_0, f_1, \ldots, f_{m-1}, g_0, g_1, \ldots, g_{n-1} \rangle\) corresponds to the multivariate polynomial system with roots in \(\mathbb{F}_2\). These roots can then be found through the computation a Gröbner basis \(G\) of \(I\).

Let \(I\) be an ideal of \(k[x_1, \ldots, x_n]\). Recall that a finite set \(G = \{g_1, \ldots, g_n\}\) is a Gröbner basis of \(I\) if and only if the subtraction polynomial \(S(g_i, g_j) = 0\) for all pairs \(g_i \neq g_j\) in \(G\). The Buchberger algorithm [12] uses this criterion as a test for Gröbner bases and to compute them from a given set of polynomials. Given a set of generators \(F = \{f_1, f_2, \ldots, f_n\}\) of \(I\), it returns \(G\) as a Gröbner basis of \(I\). While the Buchberger algorithm provide a solution for computing a Gröbner basis \(G\) for an ideal \(I\), the pairs in the generators of \(I\) are reduced separately. The pair selection strategy plays an important role in determining the efficiency of the Buchberger algorithm. For large systems, the algorithm is inefficient. Alternatives to the Buchberger algorithm have been developed, one of which are the algorithms due to Faugère [31, 32].

Faugère has published a series of algorithms to compute Gröbner bases. Currently, the most widely adopted is the \(F_4\) algorithm [31]. Instead of using a pair selection strategy to process one pair of polynomials at a time, the algorithm uses linear algebra to process many pairs simultaneously. It was observed that reducing a polynomial via polynomial division can be performed by computing row echelon forms. Suppose the task to reduce polynomials \(f_0, f_1, \ldots, f_{r-1}\) by polynomials \(g_0, g_1, \ldots, g_{s-1}\) with respect to a certain monomial order. In order to simulate polynomial division through row reduction of matrices, polynomials \(h_0, h_1, \ldots, h_k\) corresponding to all intermediate results in the polynomial division of \(f_i\) by \(g_j\) are
computed. A concise description of this process can be found in [82]. Let \( \mathbf{x} \) be the vector of monomials ordered with respect to the monomial order \( \preceq \). Construct the matrix \( \mathbf{A} \) such that

\[
\mathbf{A}\mathbf{x} = (h_0, h_1, \ldots, h_{k-1}, f_0, f_1, \ldots, f_{r-1})^T
\]  

(2.4.22)

Let \( \tilde{\mathbf{A}} \) be a row echelon form of \( \mathbf{A} \). It follows that

\[
\tilde{\mathbf{A}}\mathbf{x} = (\tilde{h}_0, \tilde{h}_1, \ldots, \tilde{h}_{k-1}, \tilde{f}_0, \tilde{f}_1, \ldots, \tilde{f}_{r-1})^T
\]  

(2.4.23)

where \( \tilde{f}_0, \tilde{f}_1, \ldots, \tilde{f}_{r-1} \) are the the reduced polynomials of \( f_0, f_1, \ldots, f_{r-1} \) respectively. This matrix method allows the reduction step in the Buchberger algorithm to be performed on multiple polynomials simultaneously, and provides a key improvement to the efficiency of Gröbner bases computation.

During the polynomial reduction process in \( F_4 \), many of the polynomials will reduce to zero. A separate algorithm, the \( F_5 \) algorithm [32] was introduced to avoid these reductions to zero, which improves the computation time for Gröbner bases compared to \( F_4 \). The algorithm is more complicated and its implementations are not widely available, so \( F_4 \) has been selected as the algorithm used in this research. All Gröbner bases computations presented here are performed with the implementation of the \( F_4 \) algorithm on the MAGMA computer algebra package [10] with specialised routines for boolean equations, and is considered the fastest of its kind at the time of writing.

**Example 2.6.** Let the polynomial ring be \( \mathbb{F}_2[x_1, x_2] \). The solutions to the bivariate boolean polynomial system in the variables \( x_1, x_2 \in \mathbb{F}_2 \) below is to be sought.

\[
x_1 x_2 + x_1 = 0 \tag{2.4.24}
\]

\[
x_1 + x_2 = 1. \tag{2.4.25}
\]

Only the \( \mathbb{F}_2 \) solutions of the system are to be computed, so the field equations
below are added to the system.

\[
\begin{align*}
x_1^2 + x_1 &= 0 \\
x_2^2 + x_2 &= 0
\end{align*}
\] (2.4.26)
(2.4.27)

From the ideal-variety correspondence, the solutions of the above system are the same as the common roots of the polynomials in the ideal

\[ I = \langle x_1 x_2 + x_1, x_1 + x_2 + 1, x_1^2 + x_1, x_2^2 + x_2 \rangle \subset \mathbb{F}_2[x] \] (2.4.28)

Computing a Gröbner basis \( G \) of \( I \) gives

\[ G = \{ x_1 + 1, x_2 \} \] (2.4.29)

This means that the solution to the original bivariate polynomial system is

\[ x = (1, 0)^T. \] (2.4.30)

### 2.4.3 Truth Tables and Graphs

In this section, new ways of using truth tables and graph theory to solve systems of equations are presented. Truth tables are well known in the theory of boolean functions. Their sole use in equation solution is generally not considered, since the memory requirement is exponential in the number of variables. Presented here are some methods in which truth tables can be utilised to aid in solving systems of equations. Consider the multivariate polynomial system

\[
\begin{align*}
f_1(x_1, x_2, \ldots, x_n) &= 0 \\
f_2(x_1, x_2, \ldots, x_n) &= 0 \\
\vdots & \vdots \\
f_m(x_1, x_2, \ldots, x_n) &= 0.
\end{align*}
\] (2.4.31)
Each polynomial $f_1, f_2, \ldots, f_m$ corresponds to a boolean function in its algebraic normal form (ANF). The truth tables of $f_1, f_2, \ldots, f_m$ can be constructed from their ANFs such that the maps

$$f_i : \mathbb{F}_2^n \to \mathbb{F}_2 \quad (x_1, x_2, \ldots, x_n) \mapsto 0,$$

are all determined. The sets

$$S_i = \{(x_1, x_2, \ldots, x_n) : f_i(x_1, x_2, \ldots, x_n) = 0\}.$$  \hspace{1cm} (2.4.33)

are then computed, and the solution can then be recovered by finding the intersection

$$S = \bigcap_{i=1}^m S_i.$$

(2.4.34)

The computation time and memory required for this method is exponential and equivalent to that of an exhaustive search. However, since the solution must satisfy each of the equations in the system, it is possible to process several equations simultaneously against a set of truth table entries so that incorrect solutions can be eliminated more efficiently. For example, the analysis shown in [81] uses information gathered for each separate equation to improve the efficiency of finding the unique solution without exhaustively checking every entry of the truth tables.

Another method is presented here, which combines truth tables with Gröbner bases methods. Let $X$ be the set $\{x_1, x_2, \ldots, x_n\}$ of $n$ variables. Choose $U \subset X = \{u_1, u_2, \ldots, u_r\}$ of cardinality $r$ and let $V = X \setminus U = \{v_1, v_2, \ldots, v_s\}$ be the set of remaining variables with cardinality $s = n - r$. Compute maps

$$f_i : \mathbb{F}_2^r \to \mathbb{F}_2[v_1, v_2, \ldots, v_s] \quad (u_1, u_2, \ldots, u_r) \mapsto g_i(u_1, u_2, \ldots, u_r)$$

where each $g_i$ is obtained by substituting the values of $u_1, u_2, \ldots, u_r$ into the respective $f_i$. The result is a truth table of $2^r$ rows with $m$ entries in each row, where each entry consists of boolean polynomials in $s$ variables. Each row corresponds
to the result of a partial key guess of the variables in $U$.

Let $g_{i,1}, g_{i,2}, \ldots, g_{i,s}$ be the polynomials obtained from a row of the truth table. If $v_1, v_2, \ldots, v_s$ can be found such that each $g_{i,j} = 0$, then the partial key guess $u_1, u_2, \ldots, u_n$ is correct and together with $v_1, v_2, \ldots, v_s$ forms the solution to the original system of the $m$ multivariate polynomial equations.

It is possible to discover contradictions in some truth table entries before computing Gröbner bases, which rule out the validities of some partial key guesses. For example, if there exists a linear combination of $g_{i,j}$ such that

$$\sum_{j=1}^{m} g_{i,j} = 1,$$

then it immediately follows that no common root exist for the polynomials $g_{i,j}$.

The effectiveness of the above method relies on finding a good set $U$ such that computing the truth tables and solving the reduced polynomial systems are easier than computing the solution of the entire system. It is possible to use graph theory to discover some properties of the system of equations in question, which might lead to methods of finding good sets $U$. In an algebraic attack, the set $U$ is called a partial key guess set, since the values of those variables are recovered by brute force.

Let $f$ be a boolean function in $n$ variables $x_0, x_2, \ldots, x_{n-1}$ of maximum degree $d$. Let the set of monomials present in $f$ be $m_i$. Let $H = (V, E)$ be a hypergraph with vertices $V = \{v_0, v_2, \ldots, v_{n-1}\}$. For each monomial $m_i = x_0^{c_0} x_1^{c_1} \cdots x_{n-1}^{c_{n-1}}$ in $f$, we create an edge $e_i \in E$ of $H$ such that $e_i = \{v_i | c_i = 1\}$. This gives a map $\phi$ between boolean functions in $n$ variables and hypergraphs of order $n$, where

$$\phi : \frac{\mathbb{F}_2[x_0, x_1, \ldots, x_n]}{(x_0^2 + x_0 x_1 + x_1, \ldots, x_{n-1}^2 + x_{n-1})} \to H_n$$

$$f \mapsto (V, E)$$

$$x_i \mapsto v_i$$

$$m_i \mapsto e_i$$

(2.4.37)
2.5 Summary

The degree $d$ of $f$ is the maximum cardinality of $e_i$ in $H$, denoted by $t(H)$. Let $U \subset V$, and $G_0 = (U, D_0)$ be the subhypergraph of $H$ generated by $U$. The hypergraph $G_0$ then corresponds to $f$ with 0 substituted for each of the variables in $V \setminus U$. Let $G_1 = (U, D_1)$ be a subhypergraph of $H$, where $d_{1,i} = e_i \cap U$. The hypergraph $G_1$ then corresponds to $f$ with 1 substituted to each of the variables in $V \setminus U$. Clearly, we have $t(G_1) \leq t(G_0) \leq t(H)$.

A set of boolean functions $\{f_0, f_1, \ldots, f_{n-1}\}$ in $n$ variables can be described as a set of hypergraphs $\{H_0, H_1, \ldots, H_{n-1}\}$ where $H_i = (V, E_i)$. Let $G_{0,j}, G_{1,j}, \ldots, G_{n-1,j}$ where $j \in \{0, 1\}$ be the subhypergraphs obtained in the same way as above. To find the most effective partial key guess set of $k$ variables is to find a set $U$ of cardinality $n - k$ such that each $t(G_{i,0})$ is minimal.

The feasibility of these methods on systems of multivariate equation relies heavily on the form of the equations. Before we can conclude whether they work well for algebraic attacks, more experiments with different ciphers are required. This is left for future investigation.

2.5 Summary

In this chapter, existing methods of solving systems of multivariate linear and polynomial equations and finding roots of a univariate polynomial and common roots of a set of polynomials have been reviewed. In Chapters 5-6, these methods will be applied to cryptanalyse various stream ciphers. Some novel methods of solving multivariate polynomial equations using truth tables and graphs have also been described, which might worth further investigation.
Chapter 3

Finite Field Arithmetic

The search for the most optimised algorithms for arithmetic has always fascinated the computing world. In recent years this has been especially the case in finite fields, due to the invention of public key systems based on finite fields [29]. In this chapter, a general framework is provided for constructing multiplication algorithms over finite fields, with a focus on the Karatsuba algorithm [55] and its variants [7, 108]. Such a framework has been lacking in the past. The advantage of a general framework is that provides a sound algebraic model that can be used for ease of optimisation and implementation, compared to the straight algorithmic approach to designing efficient arithmetic, which has been the norm in the past. The resulting complexity of multiplication algorithms matches, and sometimes outperforms, those previously reported.

3.1 Introduction

Throughout the history of computer science, there has been a need for efficient computational techniques for arithmetic operations. Karatsuba and Ofman [55] discovered one of the first methods of reducing the number of multiplications required to compute a product of two polynomials. This is commonly known as
Karatsuba multiplication, whose variants are still in use today for computational tasks on a large scale. In this chapter, the application of some variants of Karatsuba multiplication to arithmetic in finite extension fields is investigated, and resulting optimised complexities of arithmetic of using this method with different field representations are shown.

Traditionally, the time complexity for computing the product of two polynomials is quadratic in the degree of the polynomial. The discovery of Karatsuba multiplication showed that the computation can be performed in subquadratic time. Karatsuba multiplication has since been generalised [7], resulting in algorithms of even lower asymptotic complexity. However, the original Karatsuba multiplication is still more efficient for polynomials of small size, due to its simple construction. The Fast Fourier Transform (FFT) multiplication methods [87] are currently of the lowest asymptotic complexity, but again are slower than Karatsuba multiplication for multiplication of polynomials of low degree [108].

The advent of applications of finite fields in cryptology has led to a need for efficient algorithms for operations on elements of finite fields, for example in elliptic curve cryptography [5] and torus-based cryptography [45]. Karatsuba multiplication can be used naturally for multiplication of elements in extension fields, when these elements are expressed with respect to a polynomial basis. These extension field elements can then be multiplied in the same way with the Karatsuba algorithm as polynomials, with the additional step that the product is to be reduced to its minimal element in the extension field. Extension field multiplication, although containing an extra step, generally uses fewer additions than polynomial multiplication [101]. This is because the reduction process causes the number of coefficients in the product to decrease. However, the choice of the field representation is important to the complexity of the reduction.

In this chapter, it is shown how elements in extensions of general prime fields can be multiplied efficiently using variants of the Karatsuba algorithm, rather than just for binary fields, which appear in most literature. In Section 3.2, the traditional [58] and general [98] recursive Karatsuba algorithms will be used as a basis to develop a new algebraic description of extension field multiplication. In section 3.3, various
methods of constructing extension fields and their respective representations of extension field elements for fast arithmetic are discussed. The optimised costs of finite field multiplication with these extension fields and representations using the algebraic description are then presented. These include cyclotomic extensions with polynomial bases in Section 3.4 and extensions with Gaussian normal bases in Section 3.5. With the use of primitive roots of unity in the constructions, the existence of extension fields with these representations can be easily determined. The lists of possible extensions with these constructions are shown in Appendix B.

3.2 Karatsuba Multiplication

Let $R$ be a commutative ring with unity, and $a, b \in R[x]$ be two polynomials in the variable $x$ of maximum degree $(n-1)$ having at most $n$ coefficients in $R$. The polynomials $a, b$ can be expressed in the form

$$a = \sum_{i=0}^{n-1} a_i x^i, \quad (3.2.1)$$

$$b = \sum_{i=0}^{n-1} b_i x^i. \quad (3.2.2)$$

Let $c = ab$ be the product of the polynomials $a, b$, such that

$$c = \sum_{i=0}^{2n-2} c_i x^i. \quad (3.2.3)$$

Traditionally, this product can be obtained from $a, b$ by observing that

$$c = \left( \sum_{i=0}^{n-1} a_i x^i \right) \left( \sum_{i=0}^{n-1} b_i x^i \right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j}, \quad (3.2.4)$$

and the coefficients of $c$ can be found by computing all possible products $a_i b_j$ of the coefficients of $a, b$. This takes $n^2$ multiplications in $R$. These products can then be
combined as coefficients of c for each power of x using only additions in R. This is commonly known as the schoolbook algorithm for polynomial multiplication, which is considered inefficient compared to the subquadratic methods available, one of which being Karatsuba multiplication.

Presented here is a new algebraic description of Karatsuba multiplication, in place of the traditional algorithmic description found originally in [55]. As will be shown in consequent sections, this new description leads to more optimised algorithms for extension field arithmetic.

Define vectors \( \mathbf{a}, \mathbf{b}, \mathbf{c} \) to represent the coefficients of polynomials \( a, b, c \) respectively, such that

\[
\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})^T, \\
\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})^T, \\
\mathbf{c} = (c_0, c_1, \ldots, c_{2n-2})^T.
\]

(3.2.5) (3.2.6) (3.2.7)

The Karatsuba algorithm for finding \( \mathbf{c} \) can be done in two stages using two parameters, namely the multiplicands matrix \( \mathbf{T} \) and the Karatsuba matrix \( \mathbf{M} \), which ultimately determines the cost of a particular Karatsuba algorithm. Let the number of rows in \( \mathbf{T} \) be \( w \). Firstly, the vector \( \mathbf{k} \) of Karatsuba multiplicands is computed with entries

\[
k_i = (\mathbf{Ta})_i (\mathbf{Tb})_i, \quad 1 \leq i \leq w.
\]

(3.2.8)

The value of \( w \) is the maximum number of multiplications necessary for computing \( \mathbf{c} \). Secondly, the coefficients \( \mathbf{c} \) of the product are obtained by combining the Karatsuba multiplicands such that

\[
\mathbf{c} = \mathbf{Mk}.
\]

(3.2.9)

The sizes of \( \mathbf{T}, \mathbf{M} \) depend on the degree of the polynomials, and their form depends on the type of Karatsuba algorithm used. Let \( I = \langle k_0, k_1, \ldots, k_w \rangle \) be the ideal generated by the set of Karatsuba multiplicands in \( \mathbf{k} \). A set of Karatsuba
3.2. KARATSUBA MULTIPLICATION

multiplicands \(k\) generated from \(T\) can be combined to form \(c\) if and only if

\[ c_i \in I, \quad 0 \leq i \leq n - 1. \]  \hspace{1cm} (3.2.10)

The problem of finding the smallest \(w\) such that this holds for an arbitrary \(n\) is the topic of ongoing research. Two of the available Karatsuba algorithms will be presented in Sections 3.2.1-3.2.2.

3.2.1 Traditional Recursive Algorithm

Let \(a = a_0 + a_1x, b = b_0 + b_1x\) be linear polynomials in \(R[x]\). Thus, \(a = (a_0, a_1)^T, b = (b_0, b_1)^T\). The multiplicands matrix is

\[ T = T_2 = \begin{pmatrix} 1 & 0 \\ 1 & s \\ 0 & 1 \end{pmatrix} \]  \hspace{1cm} (3.2.11)

where \(s = 1 \, [55]\) or \(s = -1\). All cost analyses here are done with \(s = -1\), which is more efficient in general, but for completeness we will present the parameters for variable \(s\). The Karatsuba matrix is

\[ M = M_2 = \begin{pmatrix} 1 & 0 & 0 \\ -s & s & -s \\ 0 & 0 & 1 \end{pmatrix}. \]  \hspace{1cm} (3.2.12)

The cost of computing the product \(ab\) using this algorithm is 3 multiplications in \(R\) and 4 additions/subtractions in \(R\). This cost will be written as \(3M + 4A\), where \(M\) stands for a multiplication and \(A\) stands for an addition/subtraction, with the ring \(R\) understood from context. From here on this simplified notation will be used. For polynomials \(a, b\) of degree \(n - 1 = 2^m - 1\) where \(m > 1\), the Karatsuba
algorithm can be used recursively. Let \( y = x^{2^m-1} \), then

\[
a = \sum_{i=0}^{2^m-1} a_i x^i + \sum_{i=2^m-1}^{2^m-1} a_i x^i y, \tag{3.2.13}
\]

\[
b = \sum_{i=0}^{2^m-1} b_i x^i + \sum_{i=2^m-1}^{2^m-1} b_i x^i y. \tag{3.2.14}
\]

The two polynomials then become linear in the variable \( y \) with the polynomials in \( x \) as coefficients, on which the Karatsuba algorithm can be applied. These polynomial coefficients can be further split into two terms at a time and the Karatsuba algorithm can be run for each split until linear polynomials in \( x \) are reached, leading to a recursion of \( m \) levels. From the complexity of the traditional Karatsuba algorithm for linear polynomials, it can be derived that the recursive Karatsuba algorithm costs \( 3^m M + 2(3^m - 2^m)A \). The algorithm can be straightforwardly implemented, but this description lacks flexibility for optimisation at the algebraic level. Therefore, the algorithm will instead be represented as an equivalent one with this description using no actual recursion. The multiplicands matrix is obtained by

\[
T = \bigotimes_{j=1}^{m} T_2 \tag{3.2.15}
\]

where \( \otimes \) denotes tensor product. The Karatsuba matrix \( M \) is of the form

\[
M = P \left( \bigotimes_{j=1}^{m} M_2 \right) \tag{3.2.16}
\]

where \( P \) is a \((2n - 1) \times 3^m\) matrix. The factor \( P \) arises from the recursion process, where various intermediate results overlap at the same powers of \( x \), resulting in only \( 2n - 1 \) coefficients in the product from \( 3^m \) computations. If the degrees of the polynomials \( a, b \) are not of the form \( 2^m - 1 \), say \( u, v \) respectively, we can find the
smallest \( n \) such that \( n > u, n > v, n = 2^m \) and express the polynomials as

\[
a = \sum_{i=0}^{u} a_i x^i + \sum_{i=u+1}^{n-1} 0 \cdot x^i,
\]

(3.2.17)

\[
b = \sum_{i=0}^{v} b_i x^i + \sum_{i=v+1}^{n-1} 0 \cdot x^i,
\]

(3.2.18)

and then use the recursive Karatsuba algorithm for degree \( n - 1 \) polynomials. With this algebraic representation of the traditional recursive Karatsuba multiplication, the algorithm can be optimised more easily and more succinctly than with the algorithmic representation of the same algorithm. As will be shown later in the chapter, the number of multiplications in \( R \) required for most values of \( n \) is less than those for the algorithm in [98], and equal to those for the algorithm in [75].

### 3.2.2 Generalised Recursive Algorithm

It is possible to perform Karatsuba multiplication by considering polynomials of degrees greater than one. This leads to the generalised recursive Karatsuba multiplication [98]. For example, one can compute directly the product of quadratic polynomials \( a, b \in R \) with

\[
T = T_3 = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & s & 0 \\
1 & 0 & s \\
0 & 1 & s
\end{pmatrix}
\]

(3.2.19)
and the Karatsuba matrix

$$M = M_3 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
-s & -s & 0 & s & 0 & 0 \\
-s & 1 & -s & 0 & s & 0 \\
0 & -s & -s & 0 & 0 & s \\
0 & 0 & 1 & 0 & 0 & 0
\end{pmatrix}$$  \hspace{1cm} (3.2.20)

Using this algorithm, the product of two quadratic polynomials \(a, b \in R[x]\) can be computed in \(6M + 10A\). This algorithm can be used in recursion with the traditional Karatsuba algorithm for linear polynomials to construct multiplication algorithms for polynomials of degree \(2^k 3^l - 1\), where \(k > 0, l > 0\). For example, for quintic polynomials

\[
a = \sum_{i=0}^{5} a^i x^i,
\]

\[
b = \sum_{i=0}^{5} b^i x^i,
\]

letting \(y = x^3\) splits \(a, b\) into

\[
a = A_0 + A_1 y
\]

\[
b = B_0 + B_1 y,
\]

where

\[
A_0 = a_0 + a_1 x + a_2 x^2,
\]

\[
A_1 = a_3 + a_4 x + a_5 x^2,
\]

\[
B_0 = b_0 + b_1 x + b_2 x^2,
\]

\[
B_1 = b_3 + b_4 x + b_5 x^2,
\]

and the polynomials \(a, b\) can be multiplied as if they are linear, with their coefficients being quadratic polynomials in \(R[x]\) rather than constants in \(R\). Similarly to the traditional recursive algorithm, the entire algorithm can be expressed with
3.3. ARITHMETIC IN EXTENSION FIELDS

no actual recursion with the matrix parameters

\[ T = T_3 \otimes T_2, \tag{3.2.29} \]
\[ M = P(M_3 \otimes M_2), \tag{3.2.30} \]

where \( P \) is an \( 11 \times 18 \) matrix. It is possible to construct \( T_n, M_n \) for larger \( n \), where \( n \) is the number of coefficients in \( a, b \), which leads to further efficiency gain when used in recursion. The algorithmic version of the generalised Karatsuba algorithm has been derived for all prime \( n \leq 127 \) in [98]. The matrices corresponding to their results will be used in this chapter. More recently, improved variants for \( n = 5, 6, 7 \) have appeared in [75].

3.3 Arithmetic in Extension Fields

Let \( \mathbb{F}_{p^n} \) be a degree \( n \) extension of a prime field \( \mathbb{F}_p \). Representation of elements in \( \mathbb{F}_{p^n} \) can be achieved in several ways. Let \( f \) be an irreducible polynomial of degree \( n \) over \( \mathbb{F}_p[x] \) and \( \alpha \) be a root of \( f \). Define

\[ \mathbb{F}_{p^n} \cong \frac{\mathbb{F}_p[x]}{(f)} \cong \mathbb{F}_p(\alpha). \tag{3.3.1} \]

The simplest power representation expresses each element in \( \mathbb{F}_{p^n} \) as \( \alpha^i \), except for the zero element, which is expressed as 0. For a minimal representation, \( i \in [0, p^n - 1] \). This representation is useful for multiplication and exponentiation, but is highly inefficient for addition, since computing \( \alpha^i + \alpha^j = \alpha^k \) with known \( i, j \) requires finding discrete logarithms. Therefore, for finite field arithmetic, a basis representation is normally used. This expresses each element \( a \in \mathbb{F}_{p^n} \) as

\[ a = \sum_{i=0}^{n-1} a_i \beta_i, \tag{3.3.2} \]

where the set \( \mathcal{B} = \{ \beta_0, \beta_1, \ldots, \beta_{n-1} \} \), called a basis of \( \mathbb{F}_{p^n} \), spans \( \mathbb{F}_{p^n} \). Common types of bases used for efficient multiplication are discussed in Sections 3.3.1-3.3.2.
3.3.1 Polynomial Bases

**Definition 3.1.** Let $\mathbb{F}_q$ be a prime field, and $\mathbb{F}_{q^n}$ be its extension of degree $n$. Let $\alpha$ be a primitive element of $\mathbb{F}_{q^n}$. The basis

$$\mathcal{B} = \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$$

is called a polynomial basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.

The above definition can be also generalised into

$$\mathcal{B}_i = \{\alpha^i, \alpha^{i+1}, \ldots, \alpha^{i+n-1}\},$$

where $0 \leq i \leq n - 1$. This gives up to $n$ polynomial bases for a primitive element $\alpha \in \mathbb{F}_{q^n}$. For simplicity, only arithmetic with the canonical polynomial bases $\mathcal{B}_0$ will be used for the cost analyses in the chapter. Results for generalised polynomial bases can be adapted from therein.

Elements in extension fields represented with polynomial bases are most suitable for Karatsuba multiplication because they resemble polynomials, except that all results are to be reduced modulo the defining polynomial $f$ of $\mathbb{F}_{q^n}$. Karatsuba multiplication in a finite field $\mathbb{F}_{q^n}$ proceeds as follows. Recall that, in the first stage, the vector $\mathbf{k}$ of Karatsuba multiplicands is formed from the $n$ coefficients of the polynomials $a, b$. Similarly, elements $a, b \in \mathbb{F}_{q^n}$ expressed with respect to a polynomial basis are of the form

$$a = \sum_{i=0}^{n-1} a_i \alpha^i, \quad a_i \in \mathbb{F}_p,$$

$$b = \sum_{i=0}^{n-1} b_i \alpha^i, \quad b_i \in \mathbb{F}_p.$$

Treating $\alpha$ as an indeterminate, $a, b$ can be uniquely identified by the vectors $\mathbf{a}, \mathbf{b}$ of $n$ coefficients. Therefore, the vector $\mathbf{k}$ is obtained in the same way as in multiplication in a univariate polynomial ring of $\mathbb{F}_p$. In the second stage, if we were
computing in an infinite ring \( R \), there would be \((2n-1)\) coefficients in the product. However, in the finite field \( \mathbb{F}_{q^n} \), there will only be \( n \) coefficients, since elements are reduced to their minimal representation with respect to the basis \( \mathcal{B} \). Therefore, the Karatsuba matrix \( M \) has to be modified to account for the reduction. Performing the multiplication with the standard Karatsuba matrix results in a product of the form

\[
c = \sum_{i=0}^{2n-2} c_i \alpha^i.
\] (3.3.7)

To reduce the higher powers of \( \alpha^i \) so that the product \( c \) is of degree less than \( n \), observe that

\[
\mathcal{B} = \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}
\] (3.3.8)

is a basis for \( \mathbb{F}_{q^n} \). Hence, the \( \alpha^i \) in the product can be reduced as

\[
\alpha^i = \sum_{j=0}^{n-1} u_{i,j} \alpha^j, \quad 0 \leq i \leq 2n - 2
\] (3.3.9)

From this, an \( n \times (2n-1) \) reduction matrix \( U_{\mathcal{B}} = (u_{i,j}) \) for the basis \( \mathcal{B} \) can be obtained. The reduced Karatsuba matrix \( M_{\mathcal{B}} \) can then be computed as

\[
M_{\mathcal{B}} = U_{\mathcal{B}} M,
\] (3.3.10)

and the coefficients of the product \( c = ab \) are computed instead by

\[
c = M_{\mathcal{B}} k.
\] (3.3.11)

Details on the choice and cost of using of polynomial bases for Karatsuba multiplication will be discussed in Section 3.4.
3.3.2 Normal Bases

Definition 3.2. Let $\mathbb{F}_q$ be a finite field, and $\mathbb{F}_{q^n}$ be its extension of degree $n$. Let $\alpha$ be a normal element of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. The basis

$$\mathcal{B} = \{\alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{q^{n-1}}\}$$  

(3.3.12)

is called a normal basis for $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.

The main advantage of normal bases is that computing Frobenius automorphisms requires no actual arithmetic, since

$$\left(\sum_{i=0}^{n-1} a_i \alpha^{p^i}\right)^p = \sum_{i=0}^{n-1} (a_{(i-1) \mod n}) \alpha^{p^i}$$  

(3.3.13)

and it is equivalent to a rotation of coefficients. However, normal bases are not generally suited for multiplication, since the representations of elements under these bases do not resemble polynomials. Nevertheless, it has been shown that there exists a bijective map from Gaussian normal bases to polynomial bases, which allows Karatsuba multiplication to be performed with the same asymptotic complexity as if polynomial bases were used [35]. However, this method is not practical for small extension fields, because of its high complexity constant. Hence, for some normal bases, a set of multiplicands that takes $O(n^2)$ multiplications will be presented, but for small extension fields the algorithm will remain efficient.

Details on the choice and cost of using of normal bases for extension field multiplication will be discussed in Section 3.5.

3.4 Cyclotomic Fields

The use of cyclotomic polynomials to construct extension fields for efficient arithmetic will be discussed in this section. Such construction allows easy derivation of polynomials bases and their reduction matrices for use with Karatsuba arithmetic.
3.4. CYCLOTONIC FIELDS

Definition 3.3. Let \( r \) be a positive integer. The \( r \)-th cyclotomic polynomial \( \Phi_r \) is the minimal polynomial of the \( r \)-th non-trivial primitive root \( \zeta_r \) of unity. This can be expressed as

\[
\Phi_r(x) = \prod_{\gcd(i,r) = 1} (x - \zeta_r^i),
\]

(3.4.1)

where \( \phi \) is the Euler totient function.

Cyclotomic polynomials can be defined over any ring. For the purpose of finite field arithmetic cyclotomic polynomials over finite fields are used. Numerous properties of cyclotomic polynomials exist. Here, we use the fact that \( \Phi_r \) is of degree \( \phi(r) \). This means that if \( \Phi_r \in \mathbb{F}_q[x] \) is irreducible, then

\[
\mathbb{F}_q[x]/\langle \Phi_r \rangle \cong \mathbb{F}_{q^{\phi(r)}},
\]

(3.4.2)

since the degree of extension is the degree of \( \Phi_r \). The inverse map of \( \phi \) is given by

\[
\phi^{-1}(n) = \{ m : m \in \mathbb{Z}, \phi(m) = n \}.
\]

(3.4.3)

Hence, to construct an extension field \( \mathbb{F}_{q^n} \) given \( q, n \), choose \( r \in \phi^{-1}(n) \) such that \( \Phi_r \in \mathbb{F}_q[x] \) is irreducible, and let

\[
\mathbb{F}_{q^n} \cong \mathbb{F}_q[x]/\langle \Phi_r \rangle \cong \mathbb{F}_q(\zeta_r).
\]

(3.4.4)

The field \( \mathbb{F}_q(\zeta_r) \) is called a cyclotomic extension of \( \mathbb{F}_q \), and a cyclotomic field. Note that cyclotomic fields are commonly used to denote cyclotomic extensions over the rationals \( \mathbb{Q} \), but here this term is used for finite fields.

A list of possible cyclotomic extensions \( \mathbb{F}_{q^n} \) of \( \mathbb{F}_q \) for small \( n \) is shown in Table 3.1. An entry \((r, l)\) denotes that \( \mathbb{F}_{q^n} \cong \mathbb{F}_q(\zeta_r) \) for \( q \equiv l \pmod{r} \). A more comprehensive list is shown in Appendix B. Once a suitable cyclotomic field \( \mathbb{F}_{q^n} \cong \mathbb{F}_q(\zeta_r) \) is found, its elements can be represented using the polynomial basis

\[
\mathcal{B}_{r,0} = \{1, \zeta_r, \zeta_r^2, \ldots, \zeta_r^{n-1}\},
\]

(3.4.5)
CHAPTER 3. FINITE FIELD ARITHMETIC

Table 3.1: Cyclotomic Fields $\mathbb{F}_{q^n}$ for $2 \leq n \leq 16$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$(r, l)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(3, 2), (4, 3), (6, 2), (6, 5)</td>
</tr>
<tr>
<td>4</td>
<td>(5, 2), (5, 3), (10, 2), (10, 3), (10, 7)</td>
</tr>
<tr>
<td>6</td>
<td>(7, 3), (7, 5), (9, 2), (9, 5), (14, 3), (14, 5), (18, 2), (18, 5), (18, 11)</td>
</tr>
<tr>
<td>10</td>
<td>(11, 2), (11, 6), (11, 7), (11, 8), (22, 2), (22, 7), (22, 13), (22, 17), (22, 19)</td>
</tr>
<tr>
<td>12</td>
<td>(13, 2), (13, 6), (13, 7), (13, 11), (26, 2), (26, 7), (26, 11), (26, 15), (26, 19)</td>
</tr>
<tr>
<td>16</td>
<td>(17, 3), (17, 5), (17, 6), (17, 7), (17, 10), (17, 11), (17, 12), (17, 14), (34, 3), (34, 5), (34, 7), (34, 11), (34, 23), (34, 27), (34, 29), (34, 31)</td>
</tr>
</tbody>
</table>

or a generalised polynomial basis

$$\mathcal{B}_{r,i} = \{\zeta^i r, \zeta^{i+1} r, \ldots, \zeta^{i+n-1} r\}.$$  \hfill (3.4.6)

Any two elements $a, b \in \mathbb{F}_{q^n}$ represented in this form allow the Karatsuba algorithm to be used for computing their product $c = ab$.

3.4.1 Multiplication in Extensions of Degree $2^m$

Let $\mathbb{F}_p$ be a prime field. If we can find a cyclotomic field $\mathbb{F}_{p^n} \cong \mathbb{F}_p(\zeta_r)$ for $n = 2^m$, the traditional recursive Karatsuba algorithm can be used along with reductions of higher powers of $\zeta_r$ to compute the product of two elements in $\mathbb{F}_{p^n}$. That is,

$$T = \bigotimes_{i=1}^{m} T_2, \quad M = P \left( \bigotimes_{i=1}^{m} M_2 \right).$$  \hfill (3.4.7)

Additionally, we need the reduction matrix $U_B$ whose entries $u_{i,j}$ are obtained from the relation

$$\zeta^i_r = \sum_{j=0}^{n-1} u_{i,j} \zeta^j_r, \quad 0 \leq i \leq 2n - 1.$$  \hfill (3.4.8)

The complexity of finite field multiplication using Karatsuba arithmetic for some $\mathbb{F}_{p^{2^n}}$ is shown in Table 3.2. The form of the Karatsuba matrix $M$ is described by the following three values in each case.
3.4. CYCLOTOMIC FIELDS

- Complexity $C(\mathbf{M})$, the number of non-zero elements in $\mathbf{M}$
- Norm $N(\mathbf{M})$, the square of the Frobenius norm of $\mathbf{M}$
- Weight $W(\mathbf{M})$, the largest entry in absolute value of $\mathbf{M}$

Clearly, if $W(\mathbf{M}) = 1$, then $C(\mathbf{M}) = N(\mathbf{M})$. Values of $C(\mathbf{M}), N(\mathbf{M}), W(\mathbf{M})$ will be used for various tables throughout this chapter. The arithmetic cost of the two stages of the Karatsuba algorithm is shown together with its total cost. These costs are derived from the matrix multiplications from the algebraic description of the Karatsuba algorithm, which are optimised using the symbolic algebra package Maple 10.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>$C(\mathbf{M})$</th>
<th>$N(\mathbf{M})$</th>
<th>$W(\mathbf{M})$</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>$3M + 2A$</td>
<td>2A</td>
<td>$3M + 4A$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>17</td>
<td>17</td>
<td>1</td>
<td>$9M + 10A$</td>
<td>11A</td>
<td>$9M + 21A$</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>493</td>
<td>493</td>
<td>1</td>
<td>$81M + 130A$</td>
<td>201A</td>
<td>$81M + 331A$</td>
</tr>
</tbody>
</table>

Table 3.2: Recursive Karatsuba multiplication over cyclotomic fields $\mathbb{F}_{p^{2^m}}$

3.4.2 Multiplication in Even Extensions

If the extension degree $n = 2t$ is even, but not a power of 2, and a cyclotomic extension of degree $n$ exists, the method in Section 3.2.1 of padding the polynomials with zero coefficients can be used. The product is then to be reduced in the field $\mathbb{F}_{p^{2t}}$, using the basis generated by $\zeta_r$. Firstly, the multiplicands matrix $\mathbf{T}$ for $2^m$ is used, taking only the first $(2^m - n)$ columns, since the rest are all zeroes when multiplied with the coefficient vectors $\mathbf{a}, \mathbf{b}$. Some rows in $\mathbf{T}$ will be all zeroes or the same after the truncation, which leads to savings in the number of multiplications needed. The Karatsuba matrix $\mathbf{M}$ can then be optimised according to the modifications in $\mathbf{T}$. The costs of this algorithm for various extension fields are shown in Table 3.3. It can be seen that an algorithm for a degree $n$ extension is most efficient using the lowest $r$ possible.
Alternatively, the generalised Karatsuba algorithm can be used to match the degree of extension with the length of the polynomials, so that no zero coefficients need to be added. This method preserves the symmetries of the Karatsuba multiplicands, and the resulting reduced Karatsuba matrices $M_B$ are generally less complicated.

The costs of this algorithm for various extension fields are shown in Table 3.4. It can be seen that if the traditional Karatsuba algorithm does not give advantages by having fewer multiplications, then the generalised version is a preferred way of implementing arithmetic on cyclotomic extensions, since it generally requires fewer additions. Note that if the Karatsuba multiplicands from [75] are used, the generalised Karatsuba algorithm will be more efficient in more cases than the traditional Karatsuba algorithm. However, these results are not incorporated, because there is not yet a simple construction for the multiplicands matrix $T_n$ for general $n$ with this variant.

### Table 3.3: Traditional Recursive Karatsuba Algorithm over $\mathbb{F}_{p^{2t}}$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>$C(M)$</th>
<th>$N(M)$</th>
<th>$W(M)$</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>49</td>
<td>49</td>
<td>1</td>
<td>$18M + 24A$</td>
<td>$29A$</td>
<td>$18M + 53A$</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>56</td>
<td>56</td>
<td>1</td>
<td>$18M + 24A$</td>
<td>$35A$</td>
<td>$18M + 59A$</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>174</td>
<td>183</td>
<td>2</td>
<td>$42M + 64A$</td>
<td>$96A$</td>
<td>$42M + 160A$</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>216</td>
<td>351</td>
<td>2</td>
<td>$42M + 64A$</td>
<td>$126A$</td>
<td>$42M + 190A$</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>258</td>
<td>261</td>
<td>2</td>
<td>$54M + 86A$</td>
<td>$124A$</td>
<td>$54M + 210A$</td>
</tr>
<tr>
<td>12</td>
<td>26</td>
<td>332</td>
<td>557</td>
<td>2</td>
<td>$54M + 86A$</td>
<td>$185A$</td>
<td>$54M + 271A$</td>
</tr>
</tbody>
</table>

### Table 3.4: Generalised Recursive Karatsuba Algorithm over $\mathbb{F}_{p^{2t}}$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>$C(M)$</th>
<th>$N(M)$</th>
<th>$W(M)$</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>48</td>
<td>48</td>
<td>1</td>
<td>$18M + 24A$</td>
<td>$26A$</td>
<td>$18M + 50A$</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>48</td>
<td>48</td>
<td>1</td>
<td>$18M + 24A$</td>
<td>$29A$</td>
<td>$18M + 53A$</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>144</td>
<td>144</td>
<td>1</td>
<td>$45M + 71A$</td>
<td>$64A$</td>
<td>$45M + 135A$</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>203</td>
<td>380</td>
<td>2</td>
<td>$45M + 71A$</td>
<td>$122A$</td>
<td>$45M + 193A$</td>
</tr>
</tbody>
</table>
3.5 Gauss Periods

The use of Gauss periods to construct extension fields for efficient arithmetic will be discussed in this section. Such construction allows easy derivation of Gaussian normal bases and their reduction matrices for use with efficient arithmetic.

If no cyclotomic extensions exist for a certain degree $n$, which is the case for all odd $n$ and some even $n$, then a cyclotomic field with degree $n$ to construct a reduction matrix cannot be used. However, alternative extensions can still be constructed using primitive roots of unity through the use of Gauss periods and their associated Gaussian normal bases.

**Definition 3.4.** Let $n, k$ be positive integers such that $r = nk + 1$ is prime, and $\mathbb{F}_q$ be a finite field of $q$ elements, where $q$ is a prime power such that $\gcd(q, r) = 1$. The multiplicative group $(\mathbb{Z}/r\mathbb{Z})^*$ is cyclic with $nk$ elements, and the primitive $r$-th root $\zeta_r$ of unity lies in $\mathbb{F}_{q^{nk}}$. Let $\mathcal{K}$ be the unique subgroup of $(\mathbb{Z}/r\mathbb{Z})^*$ with $k$ elements, then

$$\alpha_{(n,k)} = \sum_{i \in \mathcal{K}} \zeta_r^i$$

(3.5.1)

is called a *Gauss period* of type $(n, k)$ over $\mathbb{F}_q$.

**Definition 3.5.** Let $n, k$ be positive integers such that $r = nk + 1$ is prime, and $\alpha_{(n,k)}$ be a Gauss period of type $(n, k)$. If $\gcd(e, n) = 1$, where $e$ is the index of $q$ modulo $r$, then $\alpha_{(n,k)}$ is a normal element of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. The normal basis

$$\mathcal{N}_{(n,k)} = \{\alpha_{(n,k)}, \alpha_{(n,k)}^q, \ldots, \alpha_{(n,k)}^{q^{n-1}}\}$$

(3.5.2)

generated by such $\alpha_{(n,k)}$ is called a *Gaussian normal basis* of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.

A list of Gaussian normal bases for small $n$ and their least accompanying $k$ is shown in Table 3.5. From the table, a Gaussian normal basis $\mathcal{N}_{(n,k)}$ exists for an extension field $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ if $q \not\equiv l \pmod{r}$, where $r = nk + 1$. A more comprehensive list is shown in Appendix B.
3.5.1 Multiplication with Single Extensions

Consider a prime field \( \mathbb{F}_p \). If a Gauss period \( \alpha \) of type \((n, k)\) over \( \mathbb{F}_p \) exists for some \( k \), a Gaussian normal basis \( N_{(n,k)} \) can be used to represent elements in the extension \( \mathbb{F}_{p^n} \). In [35] it is shown that every normal basis in \( \mathbb{F}_{p^n} \) can be mapped to a polynomial basis in \( \mathbb{F}_{p^{nk}} \) for use with Karatsuba multiplication. However, the factor \( k \) has a significant effect on the cost of multiplication for relatively small values of \( n \). Therefore, for small \( n \), it is preferable to use the following multiplicands that allow multiplication on \( N_{(n,k)} \) to be carried out directly. Let \( v_{i,j} \) be the column vector of length \( n \) with 1 as the \( i \)-th entry and \( -1 \) as the \( j \)-th entry, and 0 for all other entries, for \( 0 \leq i \leq n-2 \), \( i+1 \leq j \leq n-1 \). There are in total \( \binom{n}{2} \) such vectors. Let \( I \) be the identity matrix of order \( n \), and \( V \) be the concatenation of all vectors \( v_{i,j} \) as an \( n \times \binom{n}{2} \) matrix. The multiplicands matrix is given by

\[
T = (I | V)^T \quad (3.5.3)
\]

The number of rows in \( T \), that is, the number of multiplications in \( \mathbb{F}_p \) required for this algorithm, is \( n + \binom{n}{2} = \frac{1}{2}(n^2 + n) \). This algorithm is asymptotically worse than Karatsuba multiplication, but it saves approximately half the number of multiplications compared to the schoolbook algorithm. In Table 3.6, the number \( M_1 \) of multiplications required for the algorithm in [35] is compared to the number \( M_2 \) of multiplications for the latter algorithm, along with the complexities, norms and weights of the matrices for the latter algorithm.

It can be seen that the algorithm presented here for small \( n \) is more efficient than
3.5. GAUSS PERIODS

Table 3.6: Multiplication with Gaussian normal bases

<table>
<thead>
<tr>
<th>n</th>
<th>k</th>
<th>r</th>
<th>M₁</th>
<th>M₂</th>
<th>C(M)</th>
<th>N(M)</th>
<th>W(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>11</td>
<td>42</td>
<td>15</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>13</td>
<td>54</td>
<td>21</td>
<td>36</td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>29</td>
<td>207</td>
<td>28</td>
<td>84</td>
<td>91</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>19</td>
<td>108</td>
<td>45</td>
<td>81</td>
<td>81</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>61</td>
<td>684</td>
<td>55</td>
<td>240</td>
<td>280</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.6: Multiplication with Gaussian normal bases

Karatsuba multiplication with mapping to polynomial bases, particularly for those Gaussian normal bases generated from Gauss periods of type \((n, k)\) with larger \(k\). Also, the reduced Karatsuba matrices for Gaussian normal bases are in general more complicated than those for polynomial bases over cyclotomic fields, again particularly for those bases with large \(k\). Nevertheless, for applications needing normal bases representation or small extensions of odd degree, this is still a very practical algorithm for multiplying elements.

3.5.2 Multiplication with Multiple Extensions

Since cyclotomic fields and Gaussian normal bases can be defined on non-prime fields \(F_q\), multiple extensions from the prime field \(F_p\) can be used to construct the finite extension field in which arithmetic is to be performed. This is useful when a single extension does not exist for \(F_{p^n}\) with the previous methods of cyclotomic fields and Gaussian normal bases. It is known that

\[
F_{p^n} \cong F_{(((p^{n_1})^{n_2})\cdots)^{n_t}},
\]

where \(\prod_{i=0}^{t} n_i = n\). Therefore, if we can find values \(n_i\) such that an extension exists for each of them, the extension field \(F_{p^n}\) can be constructed with the fast arithmetic discussed in this chapter. The parameters \(M, T\) for the multiplication algorithm in \(F_{p^n}\) can be found in the usual way as tensor products of the respective matrices for each extension of degree \(n_i\).

Table 3.7 shows the use of this method for some small extension fields. In these
examples, the largest even factor of \( n \) is used for a cyclotomic extension, since it has been shown that multiplication in cyclotomic extensions is generally more efficient than the use of Gaussian normal bases. Gauss periods were used for the rest of the extensions. The numbers \( M(n_i) \) of multiplications for each \( n_i \) is shown, along with the total cost \( M \) of multiplications for the entire algorithm. With the combinations of cyclotomic fields and Gaussian normal bases, the construction of fields with virtually any degree of extension for fast field multiplication is possible.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n_i )</th>
<th>Basis</th>
<th>( M(n_i) )</th>
<th>( M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4, 2</td>
<td>( B_{5, i} \times N_{(2, 2)} )</td>
<td>9, 3</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>3, 3</td>
<td>( B_{3, i} \times N_{(3, 2)} )</td>
<td>6, 6</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>5, 2</td>
<td>( B_{5, i} \times N_{(2, 5)} )</td>
<td>14, 3</td>
<td>42</td>
</tr>
<tr>
<td>16</td>
<td>4, 2, 2</td>
<td>( B_{4, i} \times N_{(2, 2)} \times N_{(2, 5)} )</td>
<td>9, 3, 3</td>
<td>81</td>
</tr>
</tbody>
</table>

Table 3.7: Constructing Multiple Extensions

Other factors might affect the choice of the extension degrees. For example, in the \( T_{30} \) torus-based cryptosystem [94], the extension field \( \mathbb{F}_{p^{30}} \) is constructed by using the cyclotomic extension \( \mathbb{F}_{q^6} \cong \mathbb{F}_q(\zeta_9) \) on top a field \( \mathbb{F}_q \equiv \mathbb{F}_{p^5} \) that uses the Gaussian normal basis \( N_{(5, 2)} \). This is done to preserve a specially designed fast arithmetic over \( \mathbb{F}_{q^6} \). In Chapter 4, the extension field constructions and multiplication algorithms developed in this chapter will be applied to public key systems based on finite fields.

### 3.6 Summary

In this chapter, an algebraic description of the Karatsuba algorithm for multiplication is presented, which allows a better view of variants of the algorithm and easier optimisation. In fact, most polynomial multiplication algorithms can be described in the same way by changing the parameters involved in the description.

Methods of construction of extension fields using primitive roots of unity suitable for Karatsuba multiplication are discussed. Cyclotomic fields, which allow ele-
ments to be represented with respect to a polynomial basis, are most suitable for Karatsuba multiplication. A limitation is that cyclotomic fields do not exist for odd extensions and some even extensions. Gauss periods, which are used for construction of Gaussian normal bases, are also discussed. These can be defined over more varieties of extensions. It has been shown how Gaussian normal bases and to multiply elements using these bases, which is efficient for low degree extensions and is as practical as Karatsuba multiplication. Combining these two constructions with the idea of multiple extensions, fields of nearly all possible cardinalities can be constructed using primitive roots of unity for fast arithmetic.

Under our algebraic description of Karatsuba multiplication, the best optimised algorithms can be obtained for the traditional recursive and generalised recursive Karatsuba algorithms. Although the recent discovery [75] of variants of the Karatsuba algorithm with a fewer number of multiplications improve on these results, it is believed that our algebraic representation of the recursion can be applied to these variants to gain further improvements in efficiency. This is left for ongoing future work.

The extension field multiplication algorithms in this chapter will be applied to the implementation of public key cryptography in extension fields in Chapter 4.
Chapter 4

Public Key Systems in Extension Fields

Efficiency is one of the most crucial aspects for the success of a public key cryptographic system, and the ability to perform fast arithmetic underlies how efficient computation can be carried out in these systems. In this chapter, we apply the extension field multiplication techniques developed in Chapter 3 to public key systems built on extension fields. It is shown that our algebraic method of deriving multiplication algorithms achieves results as good as, or better than, those previously reported on some of these public key systems. Experiments with different bases will also be shown to demonstrate the flexibility of our algebraic description of extension field multiplication.

4.1 Introduction

Since its introduction in 1976, many public key systems based on the discrete logarithm problem have been proposed by numerous researchers. The most influential systems until today have been the Diffie-Hellman key exchange [29] and the ElGamal encryption scheme [30]. The hardness of solving the discrete logarithm
problem has a wide acceptance, although the recent advancements in subexponential algorithms attacking the discrete logarithm problem in finite fields implies that the key size needed for such cryptosystems to be secure, which currently stands at around 1024 bits, will soon become increasingly large. A way of countering this problem was introduced in 1985 with the use of elliptic curves [59, 73], for which there are no known subexponential algorithms to solve its underlying discrete logarithm problem. Since 1993, methods for reducing the key size for systems over extension fields while keeping the same amount of security have also been proposed [43, 64, 89]. Their associated systems involve the creation of isomorphisms between a cyclic subgroup of an extension field and a smaller extension field, and this allows elements in the cyclic subgroup to be represented with a smaller data size, while keeping the security of the system. In 2004, these methods were modified and generalised into what is now known as torus-based cryptosystems [83].

The trace-based public key systems and the torus-based public key systems will be discussed in Section 4.2 and Section 4.3 respectively. The extension field arithmetic involved in each these system will be constructed using the algebraic description from Chapter 3, and the optimised arithmetic costs will be presented.

4.1.1 Public Key Cryptography

Discrete logarithms, together with integer factorisation, constitute the two main intractable problems in number theory that are widely used to design trapdoors in public key systems. In contrast to integer factorisation, which is defined in the integer ring \( \mathbb{Z} \), the discrete logarithm problem can be defined over any cyclic group. The simplest implementation of public key systems based on the discrete logarithm problem is over cyclic groups in prime fields \( \mathbb{F}_p \), which require only modulo arithmetic. Protocols such as Diffie-Hellman key exchange [29] and ElGamal encryption [30] can then be performed. The feasibility of implementing public key cryptography in cyclic subgroups of general extension fields based on the discrete logarithm problem has been discussed since the publication of the ElGamal public key system. It has not gained wide attention since computation in prime fields
and binary fields is generally much easier to implement than in extension fields.

**Definition 4.1 (The Discrete Logarithm Problem in Extension Fields).** Let $G$ be a cyclic subgroup of the multiplicative group $\mathbb{F}_q^*$ of an extension field $\mathbb{F}_q$, and $g \in G$ be a generator of $G$. Let $h \in G$. Given $g, h$, the *discrete logarithm problem* in the finite field is the problem of computing the smallest integer $x$ such that

$$g^x = h. \quad (4.1.1)$$

In a public key system based on the discrete logarithm problem, the public key is derived from $g$ and the private key from $h$. The security of the system then lies in the difficulty of finding $x$ with the knowledge of $g, h$. As cryptanalytic efficiency increases, the cardinality of the cyclic group $\langle g \rangle$ must increase in order for the public key system to remain secure. At the time of writing, the recommended size of $g$ is at least 1024 bits for security in the immediate future, and at least 2048 bits for security over a longer period. Attempts to reduce the key lengths while keeping the security have been made in order to provide more condensed implementations of public key systems, especially in devices with limited memory. Elliptic curve cryptography [59, 73] has been introduced with this in mind. At the current state of cryptanalytic techniques and computing power, a key size of around 160 bits is sufficient to resist attacks on elliptic curve discrete logarithms. However, since arithmetic on elliptic curves is usually cumbersome, alternative public key systems based on extension fields that also achieve small key sizes have been proposed. These include trace-based cryptography [89, 64] and torus-based cryptography [83], which use special structures to achieve compact representations of group elements.

### 4.2 Trace-Based Cryptography

A series of public key systems was introduced for compact representation of extension field elements, beginning with the LUC public key system in 1993 [89]. LUC first provided the possibility of using traces as a means for compact representa-
tion. In 2000, the XTR public key system [64] then made an impact in the area of trace-based cryptography as a feasible alternative to elliptic curve cryptography. This was followed by a few variants and speculation of further trace-based public key systems [69, 11], until the emergence of torus-based cryptography [83], which we will discuss in section 4.3. In this section, the arithmetic for LUC and XTR will be described.

**Definition 4.2.** Let $\mathbb{F}_q$ be a finite field with $q$ elements. The cyclotomic subgroup $G_{q,n}$ of $\mathbb{F}_{q^n}$ is the multiplicative subgroup of $\mathbb{F}_{q^n}^*$ of order $\Phi_n(q)$, the $n$-th cyclotomic polynomial evaluated at $q$.

**Definition 4.3.** Let $\mathbb{F}_q$ be a finite field with $q$ elements, and $\mathbb{F}_{q^n}$ be an extension field of $\mathbb{F}_q$ of degree $n$. The trace map of $\mathbb{F}_{q^n}$ with respect to $\mathbb{F}_q$ is defined as

$$\text{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q} : \mathbb{F}_{q^n} \to \mathbb{F}_q$$

$$x \mapsto \sum_{i=0}^{n-1} x^{q^i}. \quad (4.2.1)$$

The value of $\text{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q}(x)$ is called the trace of $x$ with respect to $\mathbb{F}_q$.

From here on, we simply use $\text{Tr}(x)$ to denote the trace of $x$ in the above context. Trace-based public key systems make use of properties of the traces of elements in cyclotomic subgroups to achieve compact representations and efficient computations. We mention here in passing that a special case of the GH family of public key systems [43] based on third order linear feedbacks shift registers over $\mathbb{F}_{p^2}$ also gives the XTR public key system.

### 4.2.1 LUC

The LUC public key system was introduced in [89], taking its name from the use of Lucas functions in the system. It is a variant of public key systems based on discrete logarithms, built upon the cyclotomic subgroup $G_{p,3} \subset \mathbb{F}_{p^2}$. In LUC, an element $g \subset G_{p,3}$ can be uniquely represented by its trace $\text{Tr}(g) \in \mathbb{F}_p$ up to its
conjugates. Let \( V_n = \text{Tr}(g^n) \). Exponentiation in LUC can then be performed with the trace representation in \( \mathbb{F}_p \) through the recurrence relation

\[
V_{n+m} = V_n V_m - V_{n-m}, \quad (4.2.2)
\]

which is a special case of Lucas functions. This costs one multiplication in \( \mathbb{F}_p \) for each recursion. The LUC system can then be implemented entirely in \( \mathbb{F}_p \) using modular arithmetic. It was then suggested that the recurrence relation is cumbersome to use in, for example, double exponentiation. An implementation of LUC in \( \mathbb{F}_p^2 \), with elements mapped to their traces in \( \mathbb{F}_p \) only when transmission is needed, can be more flexible [90]. However, the cost of an \( \mathbb{F}_p^2 \) implementation is significantly higher than an \( \mathbb{F}_p \) implementation. For the quadratic extension, the cyclotomic fields \( \mathbb{F}_p(\zeta_3) \) or \( \mathbb{F}_p(\zeta_4) \) can be used for efficient arithmetic with bases \( \mathbb{B}_{3,i} \) or \( \mathbb{B}_{4,i} \) respectively. The arithmetic costs for both implementations of LUC are shown in Table 4.1.

\begin{center}
Table 4.1: Arithmetic Costs in LUC in [90]
\end{center}

4.2.2 XTR

The XTR public key system was introduced in [64]. XTR stands for Efficient Compact Subgroup Trace Representation (ECSTR). It is a variant of public key systems based on discrete logarithms, built upon a cyclic subgroup of the cyclotomic subgroup \( G_{p,6} \subset \mathbb{F}_{p^6} \). The main results on XTR from [67] are presented below.

**Definition 4.4.** Let \( \mathbb{F}_p \) be a prime field of \( p \) elements, Let \( \mathbb{F}_{p^6} \) be a sixth degree extension of \( \mathbb{F}_p \), and \( \mathbb{F}_{p^6}^* \) be its multiplicative group. The unique multiplicative
subgroup of $\mathbb{F}_{p^6}^*$ of order $\Phi_6(p)$ is called the XTR supergroup. Essentially, the XTR supergroup is just the cyclotomic subgroup $G_{p,6}$.

**Lemma 4.5.** Let $\mathbb{F}_p$ be a prime field of $p$ elements and $\mathbb{F}_{p^2}$ be a quadratic extension of $\mathbb{F}_p$. Elements in the XTR supergroup can be uniquely represented by their traces with respect to $\mathbb{F}_{p^2}$ up to their conjugates.

The elements in the XTR subgroup can then be represented by their traces, which are elements in $\mathbb{F}_{p^2}$. Computational efficiency in XTR is achieved by the fact that all arithmetic can be performed in $\mathbb{F}_{p^2}$.

Let $g$ be a generator of a cyclic subgroup $\langle g \rangle$ of the XTR supergroup. The discrete logarithm problem in cyclic subgroup $\langle g \rangle$ forms the security basis of the XTR public key system. Therefore, the order of $g$ determines the security of XTR from subgroup attacks such as Pollard’s methods [67]. As to the subexponential methods over fields, the security of XTR relies on the fact that the smallest field in which $\langle g \rangle$ can be embedded is $\mathbb{F}_{p^6}$, so the security of XTR against these methods is the same as systems based on $\mathbb{F}_{p^6}$. It is then said that XTR achieves $\mathbb{F}_{p^6}$ security using only $\mathbb{F}_{p^2}$ implementation. However, this claim was challenged by the discovery of efficient index calculus methods over algebraic tori, in which the XTR group can be embedded [46].

Similar to the LUC public key system, XTR had its own specialised arithmetic in $\mathbb{F}_{p^2}$. However, it was found in [90] that it is comparably efficient to use an implementation in $\mathbb{F}_{p^6}$, with mapping of elements to their traces when storage or transmission is necessary. Furthermore, the implementation in $\mathbb{F}_{p^6}$ is more flexible and can accommodate more applications. In [60], the use of Gaussian normal bases for XTR was investigated as well, which shows that more primes $p$ are eligible for use in the system. The resulting implementations are reasonably efficient compared to the original one with optimal normal bases.

Table 4.2 shows the costs of basic arithmetic used for the XTR public key system. The arithmetic of XTR has been investigated and implemented in software [100]. Algorithms in the implementation are drawn from [66, 65, 67, 63, 64]. Due to the availability of efficient exponentiation methods in the full $\mathbb{F}_{p^6}$ representation,
Table 4.2: Arithmetic Costs in XTR in [90]

the difference in efficiency between the two implementations is minimal, despite a much higher multiplication cost in the larger field.

4.3 Torus-Based Cryptography

Torus-based cryptography [83] was proposed in as a general framework for designing public key systems based on algebraic tori. The concept was drawn from a modification of the trace-based public key systems. Similar to trace-based cryptography, the purpose of torus-based cryptography is to provide compact representations of public key systems based on the discrete logarithm problem, while not compromising security. The main results from [83] on using algebraic tori in cryptography are presented below.

Definition 4.6. Let $k$ be a field, $n > 0$ be an integer, and $L \equiv \mathbb{F}_{p^n}$. The $n$-th algebraic torus $T_n$ is the set

$$T_n(k) = \bigcap_{k \subset F \subset L} \ker(N_{L/F}),$$

(4.3.1)

where $N_{L/F}$ denotes the norm map of $L$ with respect to a subfield $F$ of $L$.

All elements in $T_n(k)$ have norm one with respect to all proper subfields of $L$.

Lemma 4.7. Let $T_n(\mathbb{F}_p)$ be the $n$-th algebraic torus over a prime field $\mathbb{F}_p$. The following properties hold for a torus $T_n(k)$. 
1. $T_n(F_p) \cong G_{p,n}$,

2. $|T_n(F_p)| = \Phi_n(p)$,

3. If $h \in T_n(F_p)$ is an element of prime order not dividing $n$, then $h$ does not lie in a proper subfield of $F_{p^n}/F_p$.

Since

$$\Phi_n(p) \nmid p^r - 1, \quad r < n \quad (4.3.2)$$

$G_{p,n}$ cannot be embedded in any proper subfield of $F_{p^n}$, and hence is as secure as $F_{p^n}$ against subexponential index calculus methods over finite fields. Also, the norm of each element in $G_{p,n}$ is one with respect to each proper subfield of $F_{p^n}$, so no information can be obtained about the discrete logarithm of any element through the norm function. A definition of a rational algebraic torus for our purposes is given as follows.

**Definition 4.8.** Suppose $T$ is an algebraic torus of dimension $d$ defined over $F_q$. $T$ is rational if and only if there exists a birational map

$$\rho : T \rightarrow \mathbb{A}^d, \quad (4.3.3)$$

where $\mathbb{A}^d$ is an affine space of dimension $d$. That is, if $T$ is contained in an affine space $\mathbb{A}^t$, then $T$ is rational if and only if there exist Zariski open subsets $W \subset T$ and $U \subset \mathbb{A}^d$, and rational functions $\rho_1, \rho_2, \ldots, \rho_d \in F_q(x_1, x_2, \ldots, x_t)$ and $\psi_1, \psi_2, \ldots, \psi_d \in F_q(y_1, y_2, \ldots, y_d)$ such that

$$\rho = (\rho_1, \rho_2, \ldots, \rho_d) : W \rightarrow U \quad (4.3.4)$$

and

$$\psi = (\psi_1, \psi_2, \ldots, \psi_d) : U \rightarrow W \quad (4.3.5)$$

are inverse isomorphisms. The map $\rho$ is called a rational parameterisation of $T$.

The existence of birational parameterisations of $T_n(F_p)$ is utilised to give rise to the class of torus-based cryptosystems. For each $n$ where $T_n(F_p)$ is rational, elements
in $T_n$ can be represented in the affine space $\mathbb{A}^{\phi(n)}(\mathbb{F}_p)$ containing $\phi(n)$ elements of $\mathbb{F}_p$, compared with $n$ elements in $\mathbb{F}_p$ if they were to be represented as elements in $\mathbb{F}_{p^n}$.

**Conjecture 4.9 (Voskrenskii [97]).** Let $n > 0$. The algebraic torus $T_n$ is rational.

The rationalities of $T_n$ have been proven true when $n$ is a prime or a product of two primes. For general $n$ the results are inconclusive. Furthermore, for those $n$ where $T_n$ is rational, it is still necessary to construct a rational parameterisation of the algebraic tori $T_n$ for use in a public key system. In [83], explicit isomorphisms for $T_2$ and $T_6$ have been shown.

The public key system using $T_2$ has been shown in [83] as the smallest example of torus-based cryptosystems. Similar to LUC, elements in the $T_2$-based system can be embedded in a quadratic extension field $\mathbb{F}_{p^2}$, and can be represented by elements in $\mathbb{F}_p$. However, instead of using the trace map to obtain the compact representation as in LUC, the $T_2$ system uses the birational isomorphism

$$T_2(\mathbb{F}_p) \cong \mathbb{A}(\mathbb{F}_p).$$

(4.3.6)

The rational parameterisation of $T_2$ is also contained in [45] within the construction of the CEILIDH public key system.

### 4.3.1 CEILIDH

The CEILIDH public key system [83] is a torus-based cryptosystem constructed on the algebraic torus $T_6(\mathbb{F}_p)$ over a prime field $\mathbb{F}_p$. Since $6 = 2 \cdot 3$ is a product of two primes, $T_6$ is rational. A birational isomorphism can be constructed between $T_6(\mathbb{F}_p)$ and $\mathbb{A}^2(\mathbb{F}_p)$, so that elements in CEILIDH can be represented using two elements in $\mathbb{F}_p$. The original description of CEILIDH involved constructing algorithms to perform $T_6(\mathbb{F}_p)$ arithmetic through $\mathbb{A}^2(\mathbb{F}_p)$. However, this was found in [45] to be inefficient compared to other cryptosystems. Therefore, in the construction of
CEILIDH, arithmetic in $T_6(F_p)$ is performed in the cyclotomic subgroup $G_{p,6}$ in $F_{p^6}$, with a compression map to $\mathbb{A}^2(F_p)$ when storage or transmission of CEILIDH elements is required. The explicit birational maps between the torus $T_6$ and $\mathbb{A}^2$ as used in this implementation of CEILIDH have been obtained from [83].

In [45], the CEILIDH public key system is implemented using four representations $F_1, F_2, F_3, \mathbb{A}^2(F_p)$ of the algebraic torus $T_6(F_p)$ and the mappings between neighbouring representations. The Karatsuba algorithm is used with multiplication in $F_{p^6}$ in $F_1$ using the basis $\mathcal{B}_{9,1}$. This is the same representation as suggested for XTR. $F_2, F_3$ use the Karatsuba algorithm for both multiplication and squaring in the multiple extension field $F_{(p^2)^3}$, with

\[
\begin{align*}
\mathcal{B}_{2,0} &= \{1, \zeta_3\}, \\
\mathcal{B}_3 &= \{-\zeta_9^3 - \zeta_9^6, \zeta_9, \zeta_9^2, \zeta_9^7\}
\end{align*}
\]

as the bases for the quadratic and cubic extensions respectively. Table 4.3 shows the cost of arithmetic from [45], while Table 4.4 shows the cost of arithmetic in different representations from our analysis. It can be observed that the multiplication costs in $F_2, F_3$ are reduced to $18M + 48A$ from the previously reported $18M + 54A$.

For $F_1$, we provide experimental results of the computational costs of Karatsuba multiplication and squaring over different bases that can be used in its implementation. First, we define multiplication tables and their complexities.

As discussed in Chapter 3, the traditional method of multiplying two elements $a, b \in F_{p^6}$ with respect to a basis $\mathcal{B} = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ is to evaluate the coefficients
4.3. TORUS-BASED CRYPTOGRAPHY

<table>
<thead>
<tr>
<th>Representation</th>
<th>$F_1$</th>
<th>$F_2, F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction</td>
<td>$\mathbb{F}_p$</td>
<td>$\mathbb{F}_{(p^2)^2}$</td>
</tr>
<tr>
<td>Basis</td>
<td>$\mathcal{B}_{9,1}$</td>
<td>$\mathcal{B}<em>{2.0} \times \mathcal{B}</em>{3}$</td>
</tr>
<tr>
<td>Addition</td>
<td>6A</td>
<td>6A</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$18M + 53A$</td>
<td>$18M + 48A$</td>
</tr>
<tr>
<td>Squaring</td>
<td>$6M + 21A$</td>
<td>$12M + 33A$</td>
</tr>
</tbody>
</table>

Table 4.4: Optimised Arithmetic Costs of CEILIDH

of each $\gamma_i \gamma_j$ in the expansion of

$$ab = \left( \sum_{i=0}^{n-1} a_i \gamma_i \right) \left( \sum_{i=0}^{n-1} b_i \gamma_i \right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \gamma_i \gamma_j.$$  

Since $\mathcal{B}$ is a basis, we can write

$$\gamma_i \gamma_j = \sum_{k=0}^{n-1} t_{ij}^{(k)} \gamma_k, \quad t_{ij}^{(k)} \in \mathbb{F}_p,$$

and the coefficients of the product $c = ab$ can be evaluated as

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} t_{ij}^{(k)} a_i b_j.$$  

This can be written as a bilinear form with

$$c_k = a^T T^{(k)} b,$$

where $T^{(k)} = (t_{ij}^{(k)})$ is defined in [76] as the multiplication table for normal bases $N$ of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. For normal bases, the matrix $T^{(k)}$ is independent of $k$, and we can drop the index and simply denote it as $T$. The complexity $C_N$ of a normal basis $N$ is the number of non-zero elements in $T$ associated with $N$. It has been proven in [76] that $C_N \geq 2n - 1$. When $C_N = 2n - 1$, $N$ is called an optimal normal basis. As described in [34], multiplication using normal bases can be performed more efficiently in the case of optimal normal bases, compared to those normal bases that have higher complexities. The multiplication table $T$ can be found efficiently
using the algorithm described in [92].

For polynomial bases this is not the case. Each $T^{(k)}$ is, in general, different for different $k$. Let $B$ be a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ and $s_B^{(k)}$ be the number of non-zero entries in the multiplication table $T^{(k)}$ of $B$. We define the complexity of $B$ as

$$C_B = \frac{1}{n} \sum_{k=0}^{n-1} s_B^{(k)}. \quad (4.3.7)$$

If $N$ is a normal basis, all $s_N^{(k)}$ have the same value and we get

$$C_N = s_N^{(k)} \quad (4.3.8)$$

for any $k$, which is the same as in [76]. With this definition, we carry out complexity comparisons among polynomial bases suitable for the $F_1$ representation of the CEILIDH public key system.

In Table 4.5, $C(T)$ is the complexity of the multiplication table of the bases, $W(M), W(N)$ are the weights of the matrices in the Karatsuba algorithm for multiplication and squaring, respectively, and $A(M), A(N)$ are the cost of multiplication and squaring through $M, N$ using the Karatsuba algorithm respectively. From the table, it seems that the complexities of $T, M, N$ do not play a big part in determining the cost of the algorithms. This is because the extents of optimisation are determined by the actual structures of $M, N$, rather than from their complexities. The similarly low cost of arithmetic with different bases shows that more options are available for the implementation of the $F_1$ representation.

Finally, note that $F_4$ is simply the native torus representation in $A^3(\mathbb{F}_p)$, which is used for transmission only, and requires no arithmetic routines. Therefore, the representation is not discussed here.
4.3. TORUS-BASED CRYPTOGRAPHY

**Table 4.5: Cost of Karatsuba Arithmetic in \( \mathbb{F}_{p^6} \)**

<table>
<thead>
<tr>
<th>Basis</th>
<th>( C(T) )</th>
<th>( W(M) )</th>
<th>( A(M) )</th>
<th>( W(N) )</th>
<th>( A(N) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{7,0} )</td>
<td>10.16</td>
<td>48</td>
<td>( 18M + 50A )</td>
<td>31</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{7,1} )</td>
<td>11</td>
<td>45</td>
<td>( 18M + 51A )</td>
<td>32</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{7,2} )</td>
<td>10.16</td>
<td>48</td>
<td>( 18M + 50A )</td>
<td>32</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{7,3} )</td>
<td>10.16</td>
<td>43</td>
<td>( 18M + 50A )</td>
<td>32</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{7,4} )</td>
<td>10.16</td>
<td>41</td>
<td>( 18M + 49A )</td>
<td>32</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{7,5} )</td>
<td>10.16</td>
<td>41</td>
<td>( 18M + 50A )</td>
<td>31</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{7,6} )</td>
<td>10.16</td>
<td>39</td>
<td>( 18M + 49A )</td>
<td>26</td>
<td>( 12M + 36A )</td>
</tr>
<tr>
<td>( B_{9,0} )</td>
<td>8</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>28</td>
<td>( 12M + 38A )</td>
</tr>
<tr>
<td>( B_{9,1} )</td>
<td>8.5</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>32</td>
<td>( 12M + 39A )</td>
</tr>
<tr>
<td>( B_{9,2} )</td>
<td>8.6</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>35</td>
<td>( 12M + 39A )</td>
</tr>
<tr>
<td>( B_{9,3} )</td>
<td>8.5</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>36</td>
<td>( 12M + 39A )</td>
</tr>
<tr>
<td>( B_{9,4} )</td>
<td>8</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>35</td>
<td>( 12M + 39A )</td>
</tr>
<tr>
<td>( B_{9,5} )</td>
<td>7.6</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>35</td>
<td>( 12M + 39A )</td>
</tr>
<tr>
<td>( B_{9,6} )</td>
<td>7.5</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>32</td>
<td>( 12M + 39A )</td>
</tr>
<tr>
<td>( B_{9,7} )</td>
<td>7.5</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>28</td>
<td>( 12M + 38A )</td>
</tr>
<tr>
<td>( B_{9,8} )</td>
<td>7.6</td>
<td>48</td>
<td>( 18M + 53A )</td>
<td>26</td>
<td>( 12M + 38A )</td>
</tr>
</tbody>
</table>

**4.3.2 Systems Based on Higher Dimensional Tori**

For an algebraic torus \( T_n \) of dimension \( n \) to be suitable for use in a public key system, it must be rational. This poses a rather large restriction on the choice of \( n \). This problem was overcome in [95] by adjoining asymptotically small structures to algebraic tori \( T_n \) for any \( n \), to create slightly modified algebraic varieties that are rational, in order for public key systems to be implemented. It was shown that there exist birational isomorphisms

\[
T_n \times \mathbb{A}^k \cong \mathbb{A}^d
\]  

(4.3.9)

Specifically, a public key cryptosystem based on the algebraic torus \( T_{30} \) was proposed in [94]. Although \( T_{30} \) has not been shown rational, a birational isomorphism

\[
T_{30} \times \mathbb{A}^2 \cong \mathbb{A}^{10}
\]  

(4.3.10)
CHAPTER 4. PUBLIC KEY SYSTEMS IN EXTENSION FIELDS

Table 4.6: Arithmetic Costs in the $T_{30}$ System in [94]

<table>
<thead>
<tr>
<th>Representation</th>
<th>$\mathbb{F}_p$</th>
<th>$T_6(\mathbb{F}_p)$</th>
<th>$T_6(\mathbb{F}_p^5)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bases</td>
<td>$N_{(5,2)}$</td>
<td>$B_{0,i}$</td>
<td>$B_{0,i} \times N_{(5,2)}$</td>
</tr>
<tr>
<td>Addition</td>
<td>5A</td>
<td>6A</td>
<td>30A</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$15M + 40A$</td>
<td>$18M + 53A$</td>
<td>$270M + 985A$</td>
</tr>
<tr>
<td>Squaring</td>
<td>$-$</td>
<td>$6M + 21A$</td>
<td>$90M + 345A$</td>
</tr>
</tbody>
</table>

Table 4.7: Optimised Arithmetic Costs in the $T_{30}$ System

was found in [94], improving on an idea in [95]. This allows an element in $T_{30}$ to be represented with 8 elements provided that 2 extra elements from $A^2$ are appended to it. In [94], the isomorphism

$$T_{30}(\mathbb{F}_p) \times T_6(\mathbb{F}_p) \cong T_6(\mathbb{F}_p^5)$$

(4.3.11)

was also presented, allowing the system to perform arithmetic in $T_6(\mathbb{F}_p^5)$ instead of $T_{30}(\mathbb{F}_p)$. This is done so that the efficient $T_6$ arithmetic for CEILIDH [45] can be carried across to this system, with the only difference being that the arithmetic is performed over $\mathbb{F}_{p^5}$ rather than $\mathbb{F}_p$. Hence, efficient arithmetic in $\mathbb{F}_{p^5}$ is required in addition to that in $\mathbb{F}_p$. Table 4.6 shows the arithmetic costs of the $T_{30}$ system as reported in [94], while Table 4.7 shows the cost of operations from our analysis of extension field arithmetic. In our implementation, multiplication costs in $\mathbb{F}_{p^5}$ have been reduced to $18M + 40A$ from the previously reported $18M + 75A$ in [94]. This in turn leads to a reduction in the costs of multiplication and squaring in $T_6(\mathbb{F}_p^5)$.

In addition to the $T_{30}$ system, the torus-based cryptosystem based on $T_{210}$ has been described in [94]. We can infer the arithmetic costs of such cryptosystems by finding a suitable construction of arithmetic in $\mathbb{F}_{p^{210}}$. One way of creating the
extension field is with
\[ \mathbb{F}_{p^{210}} \equiv \mathbb{F}_{(p^7)^6}. \] (4.3.12)

In addition to the construction for \( T_{30} \), we use the degree seven extension with a Gaussian normal basis of type \((7, 4)\). For this we require that \( p \not\equiv l \pmod{29} \) where \( l \in \{1, 12, 17, 28\} \). the same quintic extension as in \( T_{30} \) can again be used to take advantage of the fast squaring algorithm in \( T_6 \) as in the \( T_{30} \)-based system.

<table>
<thead>
<tr>
<th>Representation</th>
<th>( \mathbb{F}_{p^7} )</th>
<th>( \mathbb{F}_{p^5} )</th>
<th>( T_6(\mathbb{F}_p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bases</td>
<td>( N_{(7,4)} )</td>
<td>( N_{(5,2)} )</td>
<td>( \mathcal{B}_{9,i} )</td>
</tr>
<tr>
<td>Addition</td>
<td>( 7A )</td>
<td>( 5A )</td>
<td>( 6A )</td>
</tr>
<tr>
<td>Multiplication</td>
<td>( 28M + 133A )</td>
<td>( 15M + 40A )</td>
<td>( 18M + 53A )</td>
</tr>
<tr>
<td>Squaring</td>
<td>---</td>
<td>---</td>
<td>( 6M + 21A )</td>
</tr>
</tbody>
</table>

Table 4.8: Primitive Arithmetic Costs in the \( T_{210} \) System

<table>
<thead>
<tr>
<th>Representation</th>
<th>( \mathbb{F}_{p^7} )</th>
<th>( \mathbb{F}_{p^{35}} )</th>
<th>( T_6(\mathbb{F}_{p^{35}}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bases</td>
<td>( N_{(7,4)} )</td>
<td>( N_{(7,4)} \times N_{(5,2)} )</td>
<td>( N_{(7,4)} \times N_{(5,2)} \times \mathcal{B}_{9,i} )</td>
</tr>
<tr>
<td>Addition</td>
<td>( 7A )</td>
<td>( 35A )</td>
<td>( 210A )</td>
</tr>
<tr>
<td>Multiplication</td>
<td>( 28M + 133A )</td>
<td>( 420M + 2275A )</td>
<td>( 7560M + 42805A )</td>
</tr>
<tr>
<td>Squaring</td>
<td>---</td>
<td>---</td>
<td>( 2520M + 14280A )</td>
</tr>
</tbody>
</table>

Table 4.9: Towered Arithmetic Costs in the \( T_{210} \) System

Table 4.8 presents the costs of arithmetic in each separate extension and structure, and Table 4.9 combines the results to show the full cost of implementing arithmetic in the public key system based on the torus \( T_{210}(\mathbb{F}_p) \) over a prime field \( \mathbb{F}_p \). The high costs of the arithmetic in this implementation of the system may mean that CEILIDH or the \( T_{30} \) system are preferred in practice. However, due to the high degree of the extension field, other representations or algorithms may be available that would reduce these costs significantly, such as with those found in [75, 96, 98].
4.4 Summary

A plethora of bases are available for representing elements in extension fields with
the appropriate choice of primes. Some of these bases are more suitable for imple-
mentation of extension field arithmetic in public key systems for efficiency. In this
chapter, we have presented applications of efficient extension field arithmetic in
public key systems based on the discrete logarithm problem over extension fields,
namely trace-based cryptosystems and torus-based cryptosystems. Using our new
algebraic description of extension field arithmetic, we have and designed bases
compatible with each system and the optimised the cost of arithmetic. In some
cases improvements from previously described costs have been achieved. These
methods of implementing extension field arithmetic can be adapted to current and
future cryptosystems based on extension fields.
Chapter 5

Algebraic Analysis of Nonlinear Stream Ciphers

Since its introduction, the effectiveness of algebraic analysis and attacks has become an indispensable security consideration for symmetric ciphers [27]. Algebraic attacks have proved their success on stream ciphers based on linear feedback shift registers [25]. In this chapter, investigation of a class of stream ciphers that may be susceptible to algebraic attacks, namely the irregularly clocked or clock-controlled stream ciphers [41] will be carried out. To our knowledge, this has not been done before. New methods of algebraic attacks on several clock-controlled generators are presented, some of which are more efficient than those previously known. Furthermore, the algebraic attacks developed use minimal amount of keystream for successful recovery of the initial states of the stream ciphers considered. Experimental results on actual attacks are also presented.

5.1 Introduction

The study of algebraic analysis and attacks currently encompasses a wide variety of topics including symmetric ciphers, boolean functions and systems of multivariate
equations. In this section, we give an overview of stream ciphers and the algebraic
attacks on stream ciphers, for the purpose of this thesis. Section 5.2 gives an alge-
braic description of stream cipher design. Details of algebraic analysis and attacks
on stream ciphers based on linear feedback shift registers are then presented in
Section 5.3. In Section 5.4, the approach to algebraic attacks on clock-controlled
stream ciphers will be presented. New algebraic analysis and attacks on various
clock-controlled generators are developed in Section 5.5. This is followed by an-
other set of new algebraic analysis and attacks on cascaded clock-controlled stream
ciphers, namely the Gollmann cascade generator and the eCRYPT stream cipher
project candidate Pomaranch, in Section 5.6. The results presented in this chapter
represent the first attempt at launching algebraic attacks on these types of stream
ciphers.

5.1.1 Stream Ciphers

Stream ciphers are a type of symmetric key system, in which plaintext is encrypted
in small units into ciphertext using the keystream that is continuously generated
by the cipher. Originally, this unit was one bit in size, and the corresponding
stream ciphers were called bit-based stream ciphers. The basic mechanism of a
bit-based stream cipher is shown in Figure 5.1.

Figure 5.1: Schematic of a Stream Cipher

The part of the stream cipher responsible for the keystream output is called the
pseudo-random sequence generator or the keystream generator. The generator
produces keystream according to the key given to it. At each clock, one bit of keystream is output from the pseudorandom generator, and a bitwise exclusive-or (XOR) operation is performed on one bit of plaintext and the keystream bit to form the ciphertext. The cryptographic strength of such stream ciphers can be measured by the randomness properties of the keystream outputs of the pseudorandom generators. It has been a primary goal of stream cipher research to find pseudorandom sequence generators of small size that produce sequences least distinguishable from a purely random one, and can resist all known and possible future cryptanalysis.

As processor design evolved throughout the past decades, the unit of choice for computation has generally shifted from the bit to the word. In light of this, recent new stream cipher proposals begin to use word-based structures and word operations in their pseudorandom generators. For example, if a linear feedback shift register is used, each state in the register would be a word. The stream ciphers that use words as the basis of computation are called word-based stream ciphers. At each clock, the output of a word-based stream cipher is a word, and is combined with the plaintext using the wordwise exclusive-or (XOR) operation. Word-based stream ciphers have become increasingly popular due to the speed attained, since words are a native unit of computation in modern processors. However, bit-based stream ciphers still remain a popular choice for stream cipher designs due to their simple design and performance in hardware, as well as the known security properties of bit-based cipher components. In this chapter, we are only concerned with bit-based stream ciphers. An algebraic analysis of the word-based stream cipher RC4 will be presented in Chapter 6.

Most bit-based stream ciphers use linear feedback shift registers (LFSRs) as a primary component of the pseudorandom sequence generator. This is mainly due to their simplicity, speed and provable properties. A comprehensive review of stream cipher generators based on LFSRs can be found in [84]. Non-linear feedback shift registers (NLFSRs) have also been proposed as a more sophisticated stream cipher component. While NLFSRs have been found empirically to achieve high cryptographic strength, not much theoretical study has thus far been performed.
Therefore, the use of LFSRs remains a popular choice for the design of modern day stream ciphers with security proofs.

To cryptanalyse a stream cipher, we assume that the design of the pseudorandom generator is known, and enough plaintext ciphertext pairs are obtained, so that we can derive keystream outputs from the pseudorandom generator. The keystream can then be examined together with the cipher design for any weaknesses which could enable the key to be recovered more efficiently than brute force. Common cryptanalytic techniques include time-memory tradeoff attacks, correlation attacks, and more recently, algebraic attacks.

### 5.1.2 Algebraic Analyses and Attacks

Algebraic analysis is a method of cryptanalysis in which the target cipher is described as a system of equations. It was first applied to block ciphers and public key cryptosystems in [19, 27]. Since then, algebraic analysis has been found to be successful in breaking stream ciphers based on linear feedback shift registers (LFSRs) [4, 3, 16, 21, 20, 25].

In the context of stream ciphers, the systems of equations relate the variable key or initial states of the cipher and the keystream generated from them. Solutions to the unknowns of the systems lead to the recovery of the key or initial states used in the cipher. Therefore, the complexity of solving the systems of equations describing the cipher is used as a security measure of the cipher against algebraic analysis. The process of determining the initial state or the key of a cipher using algebraic analysis through generation and solution of systems of equations is called an algebraic attack.

The success of algebraic attacks on stream ciphers led to the recent developments of new security considerations for new stream cipher proposals. Our interest in algebraic attacks is on their applications to a class of bit-based LFSRs, the irregularly clocked stream ciphers, or clock-controlled stream ciphers. Previous attacks on clock-controlled stream ciphers involve guessing the states of the clock control
mechanism so that the cipher becomes regularly clocked. The work presented here shows that algebraic attacks can be effective against this class of stream ciphers by incorporating the clock control as part of the equation generation process. Furthermore, some of these attacks improve on the currently known ones.

5.2 Stream Cipher Design

In order to analyse stream ciphers, a mathematical model is built so that the way in which the pseudorandom generator output keystream can be clearly shown. In this section, we present the algebraic model commonly used for stream ciphers based on linear feedback shift registers. This forms the basis of algebraic attacks on stream ciphers, which will be discussed in Section 5.3.

5.2.1 Linear Feedback Shift Registers

A bit-based linear feedback shift register (LFSR) $S$ of length $n$ is shown in Figure 5.2. It consists of an array of $n$ bit-sized registers, such that the states of the bit registers are shifted uniformly across the array at each clock of the register. The output bit at each clock is the state that is shifted out at the end of the LFSR. The input bit or the feedback bit at each clock is the state that is shifted out at the end of the LFSR. The input bit or the feedback bit at each clock is the new state introduced at the start of the LFSR at each clock, expressed as a linear function of the register states at the previous clock. Unless otherwise specified, all linear feedback shift registers (LFSRs) discussed in this chapter are bit-based.

**Definition 5.1.** Let $S$ be a bit-based linear feedback shift register of length $n$. The feedback polynomial

$$p = \sum_{i=0}^{n} c_i x^i \in \mathbb{F}_2[x]$$

of $S$ is a polynomial of degree $n$ that governs the linear recurrence relation for the input bit at each clock.
CHAPTER 5. ALGEBRAIC ANALYSIS OF STREAM CIPHERS

In an LFSR $S$, the coefficients $c_i$ of a feedback polynomial $p$ determine which bit position are involved in the feedback for the next input bit, and in turn the output sequence $z$ from $S$. Therefore, the properties of an output sequence from an LFSR are determined by its size $n$ and feedback polynomial $p$. A detailed treatment of linear feedback shift registers and their sequences can be found in [68].

An LFSR $S$ of length $n$ is usually represented as a vector in $\mathbb{F}_2^n$. We denote $s_i^t$ for $0 \leq i \leq n - 1$ to be the $i$-th least significant bit of register $S$ at time $t$, so that the register state $s^t$ at time $t$ is expressed as

$$s^t = (s_0^t, s_1^t, \ldots, s_{n-1}^t) \in \mathbb{F}_2^n \quad (5.2.1)$$

When more than one linear feedback shift register is involved in a cipher, we sometimes use $S_0, S_1, \ldots, S_{k-1}$ to denote a set of registers of length $n_0, n_1, \ldots, n_{k-1}$ respectively. In this case, we use $s_{i,j}^t$ for $0 \leq j \leq n-1$ to be the $j$-th least significant bit of register $S_i$ at time $t$. All registers shift to the next less significant bit position at each clock. The output bit at time $t$ of an LFSR sequence is denoted by $z^t$ for $t > 0$.

### 5.2.2 Expressions for Register States

The algebraic description of linear feedback shift registers provides a straightforward method for determining the register states at each successive clock given
their initial states. Let $S$ be a linear feedback shift register of length $n$, and

$$p = \sum_{i=0}^{n} c_i x^i \in \mathbb{F}_2[x], \quad c_i \in \mathbb{F}_2$$

be the primitive feedback polynomial of $S$, where $c_0, c_n$ are necessarily unity. In an algebraic analysis, we first label the initial states of the register $S$ by unknowns $x_i \in \mathbb{F}_2$, such that

$$s^0 = (x_0, x_1, \ldots, x_{n-1}).$$

This assigns each bit of the initial states of the register with the variable

$$s^0_i = x_i, \quad 0 \leq i \leq n - 1$$

From the feedback recurrence relations, we obtain

$$s_{i+1}^t = \begin{cases} s_i^t, & 0 \leq i \leq n - 1 \\ \sum_{i=0}^{n-1} c_i s_i^t & i = n. \end{cases}$$

Using (5.2.4) and (5.2.5), the expression of each state bit $s_i^t$ at each clock $t > 0$ can be calculated. As a linear recurrence is used, all expressions for $s_i^t$ are linear in the initial state variables $x_0, x_1, \ldots, x_{n-1}$. Alternatively, we can represent the shift of the register and the feedback bit as a matrix multiplication.

**Definition 5.2.** Let $p \in \mathbb{F}_2[x]$ be a polynomial of degree $n$ of the form

$$p = \sum_{i=0}^{n-1} c_i x^i + x^n, \quad c_i \in \mathbb{F}_2.$$ 

The *companion matrix* of $p$ is the $n \times n$ matrix

$$L = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ c_0 & c_1 & c_2 & \ldots & c_{n-1} \end{pmatrix}. $$
Let $L$ be the companion matrix of the feedback polynomial $p$ of $S$. The state transition, or clocking, of $S$ from time $t$ to time $t+1$ can be algebraically expressed as the action of $L$ on $s^t$.

$$s^{t+1} = Ls^t$$

(5.2.8)

The transition of the individual state bits in $S$ is then given by

$$s_{i}^{t+1} = \sum_{j=0}^{n-1} L_{ij} s_{j}^{t}, \quad 0 \leq i \leq n - 1.$$ 

(5.2.9)

If we wish to compute the state of $S$ after $u$ clockings, then the $u$-th power $L^u$ of $L$ is used, which gives

$$s^{t+u} = L^u s^t$$

(5.2.10)

for the entire state transition and

$$s_{i}^{t+u} = \sum_{j=0}^{n-1} (L^u)_{ij} s_{j}^{t}, \quad 0 \leq i \leq n - 1$$

(5.2.11)

for the individual bits. The matrix representation of the state transition gives the same result as using the linear recurrence relations.

If $k$ registers $S_0, S_1, \ldots, S_k$ are used in a stream cipher, we can similarly assign initial state variables to each register and compute their recurrence relations individually.

### 5.2.3 Expressions for Nonlinear Components

The outputs of the linear feedback shift registers are usually not directly used in a stream cipher, since the feedback recurrence relations are linear, and the initial states of the register can be efficiently computed from their outputs. Let $S$ be a linear feedback shift register of length $n$, and $x_0, x_1, \ldots, x_{n-1}$ be its initial states.
The output $z^t$ of $S$ at each clock can then be expressed as a linear combination

$$z^t = \sum_{j=0}^{n-1} (L^t)_{ij} x_j, \quad 0 \leq i \leq n - 1. \quad (5.2.12)$$

These are linear equations in the initial state variables. If at least $n$ bits of keystream are known, a system of at least $n$ equations in $n$ variables is created. This can be efficiently solved by techniques such as Gaussian elimination in at most $O(n^3)$ time. This shows that using the outputs of a linear feedback shift register as keystream in a stream cipher is generally not secure.

In a practical stream cipher, the state bits $s^i_t$ for each clock $t$ are often used as input to nonlinear components. One of the most popular of these nonlinear components is a nonlinear filter. The basic setup of a nonlinear filter generator is shown in Figure 5.3. At each clock of the linear feedback shift register $S$, the states of $S$ are passed through a nonlinear filter $f$ to obtain the keystream output $z$ at every time $t$.

![Figure 5.3: A Nonlinear Filter Generator](image)

Algebraically, the nonlinear filter is modelled using a polynomial boolean function $f$ of degree $d$, with inputs as the state bits $s^i_t$ from $S$. The equations obtained through the filter are of the form

$$f(s^0_t, s^1_t, \ldots, s^{n-1}_t) = z^t, \quad t > 0 \quad (5.2.13)$$

where $z^t$ are the observed keystream output from the filter at time $t$. The equations
CHAPTER 5. ALGEBRAIC ANALYSIS OF STREAM CIPHERS

generated are multivariate polynomial with maximum degree \( d \).

5.3 Algebraic Attacks

When enough equations are generated, a multivariate system of polynomial equation is formed. It can be observed that such a system can be solved for a unique solution, which corresponds to the values of the initial states \( x_0, x_1, \ldots, x_{n-1} \) of \( S \). This method of recovering the initial state variables formed the basis of algebraic attacks on stream ciphers [25]. An algebraic attack consists of two stages. The first stage involves generating equations relating the initial states and keystream. The second stage involves computing the solution to the system of equations generated.

5.3.1 Traditional Algebraic Attacks

In an algebraic attack against a stream cipher, it is assumed that the complete design of the stream cipher is known, and that enough keystream has been obtained in order to recover the unique key that generated the keystream from the cipher. In Section 5.2, the derivation of polynomial systems of equations describing a nonlinear filter generator was presented. Here, we give an example of an actual algebraic attack on such ciphers.

Example 5.3. Let \( S \) be a bit-based linear feedback shift register of length \( n = 4 \), with feedback polynomial \( p = t^4 + t + 1 \in \mathbb{F}_2[t] \). The initial states of register \( S \) are given the variables \( s_i^0 = x_i \). The state bits at the first two clocks are given by

\[
\begin{align*}
t = 0, & \quad s = (x_0, x_1, x_2, x_3) \\
t = 1, & \quad s = (x_1, x_2, x_3, x_0 + x_1)
\end{align*}
\]

(5.3.1) (5.3.2)

The state bits of \( S \) then passes through a filter \( f \) at each clock, which determines
the keystream bit $z^t$. The keystream bits at the first two clocks are given by

$$
z^t = f(s_t) = s_{t,0}s_{t,1} + s_{t,1}s_{t,3} + s_{t,2} + s_{t,3}$$  (5.3.3)
$$z^0 = f(s_0) = x_0x_1 + x_1x_3 + x_2 + x_3$$  (5.3.4)
$$z^1 = f(s_1) = x_0x_2 + x_0 + x_1 + x_3$$  (5.3.5)

The keystream bits $z^t$ are assumed known in algebraic attacks. Let the observed keystream of this filter generator be $z = (0, 1, 1, 0)$. Collecting $n$ equations which represents $n$ clocks of the generator results in the following polynomial system of equations in $n$ variables.

$$g_0 = x_0x_1 + x_1x_3 + x_2 + x_3 = 0$$  (5.3.6)
$$g_1 = x_0x_2 + x_0 + x_1 + x_3 + 1 = 0$$  (5.3.7)
$$g_2 = x_1x_3 + x_0 + x_2 + 1 = 0$$  (5.3.8)
$$g_3 = x_0x_2 + x_1x_2 + x_1 + x_3 = 0$$  (5.3.9)

As discussed in 2.4, since $x_i \in \mathbb{F}_2$ are boolean variables, we can also add to the system the field equations

$$h_i = x_i^2 + x_i = 0, \quad 0 \leq i \leq 3$$  (5.3.10)

so that the solutions that are not in $\mathbb{F}_2$ are avoided. Let

$$I = \langle g_0, g_1, g_2, g_3, h_0, h_1, h_2, h_3 \rangle$$  (5.3.11)

be the ideal generated from the above equations. Computing a Gröbner basis of $I$ we get

$$I = \langle x_0 + 1, x_1, x_2, x_3 \rangle$$  (5.3.12)

This yields the solution

$$x \in V(I) = \{(1, 0, 0, 0)\}$$  (5.3.13)

and the initial state bits $x_i$ of the register are recovered.
5.3.2 Improving Algebraic Attacks

The effectiveness of algebraic attacks on stream ciphers with nonlinear mechanisms, such as nonlinear filters, relies on solving the multivariate systems efficiently. As discussed in section 2.4, the complexities of linearisation and Gröbner bases computations both depend on the maximum degree of the polynomials in the system. In a filter generator, this degree is the degree of the filter function $f$. However, Courtois [20] showed that alternative sets of equations of lower maximum degree can be constructed by the use of annihilators and low degree multiples of $f$, which lead to significant efficiency gain in finding the solutions of the system.

**Definition 5.4.** Let $f \in \mathbb{F}_2[x_0, x_1, \ldots, x_{n-1}]$ be a boolean polynomial of degree $d$ in the variables $x_0, x_2, \ldots, x_{n-1}$. The set

$$\text{Ann}(f) = \{g : fg = 0\}$$

(5.3.14)

is an ideal in $\mathbb{F}_2[x_0, x_1, \ldots, x_{n-1}]$ called the *annihilator ideal* of $f$. An element $g \in \text{Ann}(f)$ is called an annihilator of $f$.

**Definition 5.5.** Let $f \in \mathbb{F}_2[x]$ be a boolean polynomial of degree $d$. If there exists $g \in \mathbb{F}_2[x]$ such that

$$fg = h$$

(5.3.15)

where $h$ is of degree $e < d$, then $g$ is called a low degree multiple of $f$.

If an annihilator $g$ of degree $e < d$ exists for $f$, equations of degree $e$ can be constructed to give the same set of solutions. When the keystream bit at time $t$ is 1, we have the equation

$$f(s_{0}^{t}, s_{1}^{t}, \ldots, s_{n-1}^{t}) = 1.$$  

(5.3.16)

Since $g$ is an annihilator of $f$, we must also have

$$g(s_{0}^{t}, s_{1}^{t}, \ldots, s_{n-1}^{t}) = 0$$

(5.3.17)

This gives an equation of degree $e$, instead of degree $d$ for the original one. Recall that, with linearisation algorithms, the complexity of solving multivariate poly-
mial equations of maximum degree $r$ is $O(m^3)$, where

$$m = \sum_{i=0}^{n-1} \binom{r}{i}. \quad (5.3.18)$$

Comparing the maximum degree of the original polynomial system of equations and the new one derived from the use of annihilators, we obtain

$$\sum_{i=0}^{n-1} \binom{\epsilon}{i} < \sum_{i=0}^{n-1} \binom{d}{i}. \quad (5.3.19)$$

This shows that the complexity of solving the new system of equations is lower than that of the original one.

**Definition 5.6.** The *algebraic immunity* $\text{AI}(f)$ of a boolean polynomial $f \in \mathbb{F}_{2^n}$ is the minimum degree of all polynomials in $\text{Ann}(f) \cup \text{Ann}(f+1)$.

The discovery of annihilators and low degree multiples has led to the introduction of algebraic immunity as a security measure for filter functions. The analysis of boolean functions and their algebraic immunity are beyond the scope of this thesis. However, the idea of using annihilators and low degree multiples to reduce the maximum degree of equations generated from stream ciphers will be used implicitly throughout this chapter for clock-controlled stream ciphers, in order to obtain the best attack complexities possible. In Section 5.6.5, we will also discuss the filter functions of the Pomaranch stream cipher in terms of their annihilators and low degree multiples.

## 5.4 Clock Controls

In this thesis, we divert from the algebraic analysis of filter functions, and apply a similar type of analysis to bit-based clock-controlled generators. Piecewise functions are used to describe the branching of states that occur in those generators due to the clock control mechanisms, which are then converted to polynomial
boolean equations for launching algebraic attacks.

Irregular clocking in LFSRs was originally designed to enhance their complexity and consequently their security. For a regularly clocked LFSR, the register shifts by one bit at each clock. For an irregularly clocked LFSR, the LFSR shifts according to a clock control, which governs how many bits each LFSR shifts at each clock. The clock control is usually from the output of another LFSR, which can in turn be regularly or irregularly clocked. A typical clock-controlled generator is shown in Figure 5.4.

![Figure 5.4: A Clock-Controlled Generator](image)

When more irregular clocked LFSRs are connected to each other, with each LFSR being controlled by the previous one, we obtain the cascaded clock-controlled generators [41]. These will be discussed in Section 5.6.

Clock-controlled stream ciphers assume the existence of an underlying clock that maintains a consistent set of basic time intervals against which a register and its output can be compared. A bit-based LFSR system can then be established in a number of ways. A register can be stepped in synchrony with the underlying clock, or it may move more slowly than the underlying clock, taking more than one basic unit to shift the registers. However, it can be assumed that the register will never shift faster than the clock, as otherwise we can adjust the basic clocking time to the step time of the register. Similarly, the output from a system of clock-controlled LFSRs can be synchronised with the clock time or can be slowed down or varied against the clock time. If a register shifts with the basic time interval, we refer to it as regularly clocked. If the output is delivered with the basic time interval, we refer to it as regular output.

Zenner [105] has developed a general approach to attacking such ciphers by guessing at the clocking through a clock cycle, and has applied this approach to several
ciphers including A5/1, the stop-and-go generator, the alternating step generator and the step1/step2 generator with varying levels of success. Molland [74] introduces a general approach for dealing with LFSR systems with two registers where one register controls the clocking of the other. This applies to the basic stop-and-go generator, LILI-128 and step1/step2 generator. To our knowledge, the only algebraic attack on such a cipher is [25] on LILI-128 [28], but guessing the clock control is an integral part of the approach.

We present our generic approach to determining algebraic equations involving the initial state bits from bit-based clock-controlled stream ciphers. The aim is to recover the initial state bits. We ignore key initialisation and use initial state bits as variables in the algebraic analyses. In subsequent sections, we use these equations to find the initial states of clock-controlled stream ciphers. In the case of the strengthened Beth-Piper generator and of the self-decimated generator, we obtain the initial state of the registers in a significantly faster time than any other known attack. In the other two situations, we do better than or as well as all attacks except the correlation attack. In all cases, we demonstrate relationships between the registers indicating that a low degree multiple of the polynomials corresponding to irregularly clocked registers can be quadratic. We also present our computational results on actual attacks with Gröbner basis methods.

### 5.4.1 Algebraic Analysis of Clock-Controlled Registers

Let $R$, $S$ be linear feedback shift registers (LFSRs) of length $k$, $l$ respectively. Suppose register $R$ controls the clocking of $S$ such that, at time $t$, $S$ is clocked $u$ times if $r_0 = 0$, and $v$ times if $r_0 = 1$. The register states of $S$ at time $t + 1$ can be modelled by the piecewise expression

$$s_{i+1}^t = \begin{cases} s_{i+u}^t, & r_0^t = 0, \\ s_{i+v}^t, & r_0^t = 1. \end{cases} \quad 0 \leq i \leq l - 1 \quad (5.4.1)$$
Such a clock control is called a $[u, v]$ generator. Since we are dealing with boolean functions, the piecewise expression (5.4.1) can be written as the sum

$$s_{i+1}^t = s_i^t (r_0^t + 1) + s_{i+u}^t r_0^t, \quad 0 \leq i \leq l - 1$$  

This method can be generalised to multiple clock control bits and multiple clockings on multiple registers, which will be discussed by way of examples in the rest of the chapter.

With a $[u, v]$ generator on an LFSR, we need to shift the register up to $v$ times. Our algebraic description so far does not account for the linear feedback that occurs when the register shifts. The generation of equations incorporating the feedback is shown below. Consider a LFSR $S$ of length $l$ and its feedback polynomial

$$f = \sum_{i=0}^{l} c_i x^i \in \mathbb{F}_2[x], \quad c_l = 1.$$  

Let the state of $S$ at time $t$ be $S^t = (s^t_0, s^t_1, \ldots, s^t_{l-1})$ and its output be $s^t_0$. Let $R$ be another LFSR of length $k$ that is used as a clock control for $S$ with output $r_0^t$ at time $t$. Let $L$ be the companion matrix of $f$, such that

$$L = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ c_0 & c_1 & c_2 & \ldots & c_{l-1} \end{pmatrix}.$$  

At time $t+1$, the LFSR shifts left. If it is clocked once, then

$$s_{i+1}^{t+1} = \sum_{j=0}^{l-1} L_{ij} s_j^t, \quad 0 \leq i \leq l - 1.$$  

(5.4.5)
Expanding the sum in (5.4.5) gives

\[ s_i^{t+1} = s_{i+1}^t, \quad 0 \leq i \leq l - 2 \]  
\[ s_{i-1}^{t+1} = \sum_{j=0}^{c_j} s_j^t. \]  

(5.4.6)

(5.4.7)

If \( S \) is clocked more than once, say \( u \) times, then we need to use the \( u \)-th power of \( L \) to work out the state after \( u \) clockings.

\[ s_i^{t+1} = \sum_{j=0}^{l-2} (L^u)_{ij} s_j^t, \quad 0 \leq i \leq l - 1. \]  

(5.4.8)

This continues to apply up to all clockings \( u \). Hence, for a \([u, v]\) generator, we obtain

\[ s_i^{t+1} = \begin{cases} 
\sum_{j=0}^{l-2} (L^u)_{ij} s_j^t, & r_0^t = 0, \\
\sum_{j=0}^{l-2} (L^v)_{ij} s_j^t, & r_0^t = 1, 
\end{cases} \]

\[ 0 \leq i \leq l - 1. \]  

(5.4.9)

In boolean algebraic terms, this becomes

\[ s_i^{t+1} = (r_0^t + 1) \sum_{j=0}^{l-2} (L^u)_{ij} s_j^t + r_0^t \sum_{j=0}^{l-2} (L^v)_{ij} s_j^t, \quad 0 \leq i \leq l - 1. \]  

(5.4.10)

Algebraic attacks can then be launched on a \([u, v]\)-generator without error. However, for simplicity, from here on we shall use \( s_{i+u}^t, s_{i+v}^t \) even when \( i \geq l \) to denote the shifted bits incorporating the feedback at the new timestep. The expression for the shifted bits in (5.4.10) becomes

\[ s_i^{t+1} = (r_0^t + 1) s_{i+u}^t + r_0^t s_{i+v}^t, \quad 0 \leq i \leq l - 1. \]  

(5.4.11)
5.5 Clock-Controlled Stream Ciphers

In the sections that follow, we will show how equation systems for various clock-controlled stream ciphers are obtained and algebraic attacks could be launched. Let $R_0, R_1, \ldots, R_{n-1}$ be regularly clocked linear feedback shift registers of length $k_0, k_1, \ldots, k_{n-1}$ respectively, which are used to control the clocking of a linear feedback shift register $S$ of length $l$. Let the clock control function $f^t$ at time $t$ be a linear polynomial in the output bits $r_{0,k_0-1}^t, r_{1,k_1-1}^t, \ldots, r_{n-1,k_{n-1}-1}^t$ of $R_0, R_1, \ldots, R_{n-1}$ respectively at time $t$. Let the keystream bit $z^t$ at time $t$ be a linear function $g$ in the output of all the registers. The state bits of register $S$ clock as follows.

$$s_{i+1}^t = s_i^t + u(f^t + 1) + s_i^t v f^t, \quad 0 \leq i \leq l - 1.$$  \hspace{1cm} (5.5.1)

The keystream is given by

$$z^t = \sum_{i=0}^{n-1} r_{i,k_i-1}^t + s_0^t.$$  \hspace{1cm} (5.5.2)

At the next timestep, the keystream is given by

$$z^{t+1} = \sum_{i=0}^{n-1} r_{i,k_i-1}^{t+1} + s_0^{t+1}.$$  \hspace{1cm} (5.5.3)

From (5.5.1), we can rewrite (5.5.3) as

$$z^{t+1} = \sum_{i=0}^{n-1} r_{i,k_i-1}^{t+1} + s_u^t (f^t + 1) + s_v^t f^t.$$  \hspace{1cm} (5.5.4)

Adding (5.5.2) and (5.5.4) gives the derivative of $z^t$ with respect to $t$ as

$$z^t + z^{t+1} = \sum_{i=0}^{n-1} (r_{i,k_i-1}^t + r_{i,k_i-1}^{t+1}) + f^t (s_u^t + s_v^t) + s_0^t + s_u^t.$$  \hspace{1cm} (5.5.5)
Multiplying (5.5.5) by \((f^t + 1)\) gives

\[
(f^t + 1)(z^t + z^{t+1}) = (f^t + 1) \sum_{i=0}^{n-1} (r_{i,k_i-1}^t + r_{i,k_i-1}^{t+1}) + (f^t + 1)(s^t_0 + s^t_u). \tag{5.5.6}
\]

For a \([0,v]\) clock-controlled generator, (5.5.6) becomes

\[
(f^t + 1)(z^t + z^{t+1}) = (f^t + 1) \sum_{i=0}^{n-1} (r_{i,k_i-1}^t + r_{i,k_i-1}^{t+1}). \tag{5.5.7}
\]

### 5.5.1 The Stop-and-Go Generator

Introduced in [8] and reviewed in [41], the Beth-Piper stop-and-go generator uses one of the simplest clock control mechanisms. Its structure is shown in Figure 5.5.

**Figure 5.5: The Stop-and-Go Generator**

The generator consists of two registers \(R, S\) of lengths \(k, l\) respectively, such that the output of \(R\) is used to control the clocking of \(S\). The clock control mechanism is defined by

\[
s_{i+1} = \begin{cases} 
  s_i^t, & r_0^t = 0, \\
  s_{i+1}^t, & r_0^t = 1.
\end{cases}, \quad 0 \leq i \leq l - 1 \tag{5.5.8}
\]

In boolean algebraic form, this becomes

\[
s_i^t = s_{i+1}^{t-1}(r_0^{t-1} + 1) + s_{i+1}^{t-1}r_0^{t-1}, \quad 0 \leq i \leq l_1 - 1 \tag{5.5.9}
\]

The keystream is given by

\[
z^t = s_0^t \tag{5.5.10}
\]
Taking the derivative at $z^t$ with respect to $t$ gives

$$\Delta z^t = z^t + z^{t+1} = (s_0^t + s_1^t) r_0^t$$

(5.5.11)

It can be observed that whenever $\Delta z^t = 1$, the clock control at time $t$ must be 1, and $(s_0^t + s_{i-2}^t)$ must also be 1. This equation confirms and encompasses what was observed by Gollman and Chambers, Kanso [54] and Rueppel [84]. Additionally, it gives an attack method, since in (5.5.11) the nonlinear term on the right hand side is reduced to a constant on the left hand side. Consequently, after at least $k$ keystream bits we obtain enough equations to solve the linear system in the bits of $R$ in $O(k^2)$ time. Once the initial states of $R$ are determined, the clocking of $S$ is known, and the initial states of $S$ can be found directly from the keystream.

### 5.5.2 The Step-1/Step-2 Generator

The Step-1/Step-2 generator, proposed in [41], is a modification of the stop-and-go generator, in which the clock-controlled register always shifts at each clock. Its structure is shown in Figure 5.6.

![Figure 5.6: The Step-1/Step-2 Generator](image)

At time $t$, if the output of $R$ is zero, $S$ is shifted once. If the output of $R$ is one, $S$ is shifted twice. The clock control mechanism is defined by

$$s_{i+1}^{t+1} = \begin{cases} 
    s_{i+1}^t, & r_i^t = 0 \\
    s_{i+2}^t, & r_i^t = 1
\end{cases}$$

(5.5.12)

which, in algebraic form, is

$$s_{i+1}^{t+1} = s_{i+1}^t(r_0^t + 1) + s_{i+2}^t r_0^t, \quad 0 \leq i \leq t - 1.$$  

(5.5.13)
5.5. CLOCK-CONTROLLED STREAM CIPHERS

The keystream of the Step-1/Step-2 generator is given by

\[ z^t = s_0^t. \] (5.5.14)

Taking the derivative of \( z^t \) with respect to \( t \) gives

\[ \Delta z^t = z^t + z^{t+1} = (s_1^t + s_2^t)r_0^t + (s_0^t + s_1^t). \] (5.5.15)

Multiplying through by \((r_0^t + 1)\) gives

\[ (z^t + z^{t+1})(r_0^t + 1) = (s_0^t + s_1^t)(r_0^t + 1). \] (5.5.16)

As discussed in Section 5.4 and from observing (5.5.15), if the register \( S \) always
clocks, an extra term appears in \( \Delta z \). This increases the complexity of the equations
generated and consequently the security of the cipher.

Two methods of attack can be launched using this system of equations. If we
guess the initial state bits \( R \), all equations immediately becomes linear, since the
clocking of \( S \) would be known. With each guess of \( R \), we generate a system of
linear equations, which we can solve using Gaussian elimination. If a solution
is found, then the guess of \( R \) is correct and we have also recovered the initial
states of \( S \), completing the attack. This is more or less equivalent to the clock
control guessing attack. The complexity of this attack is \( O(2^k l^3) \), and the minimum
keystream required is \( l \). If we do not guess any bits of \( R, S \), we would generate
a system of equations of degree \( l + 1 \) with \( l + m \) variables. Using Gröbner bases
methods we can obtain a unique solution to the initial states of \( R, S \). This means
that the minimum keystream required for the attack is \( k + l \).

Table 5.1 shows a comparison of attacks on the Step-1/Step-2 generator. For
registers \( R, S \) of length \( k, l \) respectively, \( m(k, l) \) is the minimum keystream re-
quired for a generator, and \( C_0(k, l), C_1(k, l) \) are the time complexities for precom-
putation and the attack respectively. The total attack complexity is given by

\[ C(k, l) = C_0(k, l) + C_1(k, l). \] Where there is essentially no common computation
for different sets of keystreams, the entry is left blank. We also give values of the
keystream and the total complexity for register sizes of 64 bits. Separate rows of complexities are used for linearisation methods and Gröbner bases methods respectively. The precomputation complexity shown in the last line is derived from polynomial multiplication. The entry * indicates that the complexity of the work using Gröbner bases methods is generally unknown. We provide empirical data derived from using Gröbner bases methods as shown in Table 5.2. Since multivariate polynomial multiplication requires exponential time in the number of variables, we restricted bit sizes to values we were able to run in under two days.

<table>
<thead>
<tr>
<th>Attack</th>
<th>$m(k,l)$</th>
<th>$C_0(k,l)$</th>
<th>$C_1(k,l)$</th>
<th>$C(k,l)$</th>
<th>$m(64,64)$</th>
<th>$C(64,64)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding</td>
<td>$5l$</td>
<td>$2^l$</td>
<td>$\leq l^3$</td>
<td>$&gt; 2^l5l$</td>
<td>$2^8$</td>
<td>$2^{72}$</td>
</tr>
<tr>
<td>Correlation [106]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock Control</td>
<td>$k+l$</td>
<td>$O((k+l)^{3.2(k+l)/2})$</td>
<td>$O((k+l)^{3.2(k+l)/2})$</td>
<td>$2^7$</td>
<td>$2^{84}$</td>
<td></td>
</tr>
<tr>
<td>Guessing [105]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improved</td>
<td>$\frac{3}{2}l$</td>
<td>$O(l^3)$</td>
<td>$O(2^l)$</td>
<td>$O(2^l)$</td>
<td>$2^7$</td>
<td>$2^{64}$</td>
</tr>
<tr>
<td>LCT [74]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>$l$</td>
<td>$O(2^{k^3})$</td>
<td>$O(2^{k^3})$</td>
<td>$2^7$</td>
<td>$2^{82}$</td>
<td></td>
</tr>
<tr>
<td>(Gaussian)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>$k+l$</td>
<td>$O(2^l)$</td>
<td>*</td>
<td>*</td>
<td>$2^7$</td>
<td>*</td>
</tr>
<tr>
<td>(Gröbner)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of Attack Complexities on the Step1/Step2 Generator

<table>
<thead>
<tr>
<th>$l_0, l_2$</th>
<th>$n$</th>
<th>$m$</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>16</td>
<td>13 s</td>
<td>3 s</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>24</td>
<td>153699 s</td>
<td>5958 s</td>
</tr>
</tbody>
</table>

Table 5.2: Attack Times for the Step1/Step2 Generator

It can be seen that our algebraic attacks on the Step-1/Step-2 generator are comparable with the other available attacks. The keystream requirement is at its minimum possible.
5.5.3 The Alternating Step Generator

The alternating step generator, introduced in [47], is an example of one regularly clocked LFSR controlling the clocking of two other registers. Its structure is shown in Figure 5.7.

![Figure 5.7: The Alternating Step Generator](image)

The alternating step generator uses three registers $R, S_0, S_1$ of lengths $k, l_0, l_1$ respectively. Register $R$ is regularly clocked, and controls whether $S_0$ or $S_1$ is clocked at any time $t$. If the output of $R$ at time $t$ is zero, then only $S_0$ is clocked. Otherwise, only $S_1$ is clocked. Hence, the registers $S_0, S_1$ moves in an alternating fashion. The output of the generator is the sum of the outputs of registers $S_0, S_1$.

The clock control mechanism is given by

$$s_{0,i}^{t+1} = \begin{cases} s_{0,i}^t, & r_0^t = 0, \\ s_{0,i+1}^t, & r_0^t = 1 \end{cases}, \quad 0 \leq i \leq l_0 - 1, \quad (5.5.17)$$

$$s_{1,i}^{t+1} = \begin{cases} s_{1,i}^t, & r_0^t = 1, \\ s_{1,i+1}^t, & r_0^t = 0 \end{cases}, \quad 0 \leq i \leq l_1 - 1. \quad (5.5.18)$$

which, in algebraic form, is

$$s_{0,i}^{t+1} = s_{0,i}^t (r_0^t + 1) + s_{0,i+1}^t r_0^t, \quad 0 \leq i \leq l_0 - 1, \quad (5.5.19)$$

$$s_{1,i}^{t+1} = s_{1,i}^t r_0^t + s_{1,i+1}^t (r_0^t + 1), \quad 0 \leq i \leq l_1 - 1. \quad (5.5.20)$$

The keystream is given by

$$z^t = s_{0,0}^t + s_{1,0}^t. \quad (5.5.21)$$
CHAPTER 5. ALGEBRAIC ANALYSIS OF STREAM CIPHERS

Taking the derivative at $z^t$ with respect to $t$ gives

$$\Delta z^t = z^t + z^{t+1} = s^t_{0,0} + s^t_{1,0} + s^{t+1}_{0,0} + s^{t+1}_{1,0},$$

which can be rewritten in terms of variables at time $t$ only as

$$z^t + z^{t+1} = (s^t_{0,0} + s^t_{0,1})r^t_0 + (s^t_{1,0} + s^t_{1,1})(r^t_0 + 1).$$

Multiplying through by $(r^t_0 + 1)$ gives

$$(r^t_0 + 1)(z^t + z^{t+1}) = (r^t_0 + 1)(s^t_{1,0} + s^t_{1,1}).$$

Alternatively, multiplying through by $r^t_0$ gives

$$r^t_0(z^t + z^{t+1}) = r^t_0(s^t_{0,0} + s^t_{0,1}).$$

Since registers $S_0, S_1$ are controlled solely by the register $R$, by guessing the bits of $R$ we obtain linear equations for (5.5.24),(5.5.25). The system of equations can then be solved in $O(t^3_0)$ or $O(t^3_1)$ time. After recovering the initial states of one of $S_0, S_1$, we can then use (5.5.21) to recover the initial states of the remaining register. This takes only $O(l_0)$ or $O(l_1)$ time since registers $S_0, S_1$ are clocked independently of each other. The time complexity of recovering the initial states of the alternating step generator is then $O(2^k \min(l_0, l_1)^3)$.

If we do not guess any bits of $R, S_0, S_1$, we would generate a system of equations of degree $k + 1$ with $k + l_1 + l_2$ variables. Using Gröbner bases methods we can obtain a unique solution to the initial states of all registers. This means that the minimum keystream required for the attack is $k + l_1 + l_2$.

Table 5.3 shows a comparison of attacks on the alternating step generator. Here, $m(k, l_i)$ is the minimum keystream required for the attack, and $C_0(k, l_i), C_1(k, l_i)$ are the time complexities for precomputation and the attack respectively. The total attack complexity is given by $C(k, l_i) = C_0(k, l_i) + C_1(k, l_i)$. Where there are essentially no common computation for different sets of keystreams, the entry is left blank. We also give values of the keystream and the total complexity for register
sizes of 64 bits. Separate rows are used for complexities derived using guessing methods and Gröbner bases methods. The precomputation complexity shown in the last line is derived from polynomial multiplication. The entry ∗ indicates that the complexity of the work using Gröbner bases methods is generally unknown.

Table 5.4 shows the empirical results on recovering the initial states of the alternating step generator using the $F_4$ algorithm for different register sizes $k, l$. The number of variables in the equations is $n$, and the number of keystream bits used, which is the number of equations, is $m$. $T_1$ is the time to generate the system of equations, and $T_2$ is the time to obtain a solution to the system.

<table>
<thead>
<tr>
<th>Attack</th>
<th>$m(k, l)$</th>
<th>$n(k, l)$</th>
<th>$C_1(k, l)$</th>
<th>$C(k, l)$</th>
<th>$m(64, 64)$</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock-controlled guessing</td>
<td>$O(l_0 + l_1)$</td>
<td>$O(2^{l_0+1}(l_0 + l_1))$</td>
<td>$O(2^{l_0+1}(l_0 + l_1))$</td>
<td>2$^9$</td>
<td>2$^{135}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock control guessing</td>
<td>$O((k + l_0 + l_1)^2l_1^{(k+l_0+l_1)/2})$</td>
<td>$O((k + l_0 + l_1)^2l_1^{(k+l_0+l_1)/2})$</td>
<td>2$^8$</td>
<td>2$^{119}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improved edit distance correlation</td>
<td>$O(l)$</td>
<td>$O(2^l)$</td>
<td>$O(2^{max(l_0-l_1) max(l_0, l_1)}$</td>
<td>2$^{11}$</td>
<td>2$^{70}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours (Gaussian)</td>
<td>$max(l_0, l_1)$</td>
<td>$O(2^{l_0 + l_1})$</td>
<td>$O(2^{l_0 + l_1})$</td>
<td>2$^7$</td>
<td>2$^{53}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours (Gröbner)</td>
<td>$k + min(l_0, l_1)$</td>
<td>$O(2^k)$</td>
<td>$*$</td>
<td>$*$</td>
<td>2$^8$</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Comparison of Attacks on the Alternating Step Generator

<table>
<thead>
<tr>
<th>$k$, $min(l_0, l_1)$</th>
<th>$n$</th>
<th>$m$</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>24</td>
<td>27 s</td>
<td>7 s</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>2706 s</td>
<td>1830 s</td>
</tr>
</tbody>
</table>

Table 5.4: Algebraic Attack Times for the Alternating Step Generator

It can be observed that our algebraic attacks require the least amount of keystream compared to other available attacks. The time complexity for the attack with guessing is lower than the edit distance correlation and the clock control guessing attacks.

5.5.4 The Self-Decimated Generator

The self-decimated generator is an example where the clock control register and the clock-controlled register are the same register. Its structure is shown in Figure 5.8.
The \([u, v]\) self-decimated generator proposed by Rueppel \[85\] consists of a single register \(S\) of length \(l\) with its output as the clock control bit. The clock control mechanism of the self-decimated generator is given by

\[
\begin{align*}
  s_{t+1}^i &= \begin{cases} 
    s_{t+u}^i, & s_0 = 0, \\
    s_{t+v}^i, & s_0 = 1.
  \end{cases}
\end{align*}
\]  

(5.5.26)

In algebraic form, this becomes

\[
    s_{t+1}^i = s_{t+u}^i (s_0 + 1) + s_{t+v}^i s_0.
\]

(5.5.27)

The keystream is given by

\[
    z^t = s_0^t,
\]

(5.5.28)

which can be rewritten as

\[
    z^{t+1} = s_u^t (s_0^t + 1) + s_v^t s_0^t.
\]

(5.5.29)

Substituting the keystream bit at time \(t\) gives

\[
    z^{t+1} = (s_u^t + s_v^t) z^t + s_u^t
\]

(5.5.30)

Since the keystream is assumed to be known, (5.5.30) becomes linear in state bits \(s_i^t\), and in turn the initial state bits \(x_i\). Once enough equations are generated, the resulting system can be solved by elimination techniques for the initial states of the register \(S\). The time complexity of the attack is \(O(l^3)\), and the minimum keystream required is \(l\) bits.
Table 5.5 shows the attack complexities for the self-decimated generator. For a register $S$ of length $l$, $m(l)$ is the minimum keystream required for a generator. $C_0(l), C_1(l)$ are the time complexities for precomputation and the attack, and the total attack complexity is given by $C(l) = C_0(l) + C_1(l)$.

Table 5.6 shows the implementation results for the algebraic attack on the self-decimated generator. From the table, $l$ is the size of the register $S$, $n$ is the number of variables in the system of equations, $m$ is the amount of keystream bits used for the solution of the system, $T_1$ is the time used to generate the system, and $T_2$ is the time to compute the solution of the system.

<table>
<thead>
<tr>
<th>Attack (Gaussian)</th>
<th>$m(l)$</th>
<th>$C_0(l)$</th>
<th>$C_1(l)$</th>
<th>$C(l)$</th>
<th>$m(128)$</th>
<th>$C(128)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>$O(l)$</td>
<td>$O(1)$</td>
<td>$O(l^3)$</td>
<td>$O(l^3)$</td>
<td>$2^7$</td>
<td>$2^{21}$</td>
</tr>
</tbody>
</table>

Table 5.5: Attack Complexities on the Self-Decimated Generator

<table>
<thead>
<tr>
<th>$l$</th>
<th>$n$</th>
<th>$m$</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>128</td>
<td>160</td>
<td>6 s</td>
<td>0.07 s</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>384</td>
<td>71 s</td>
<td>0.6 s</td>
</tr>
<tr>
<td>512</td>
<td>512</td>
<td>640</td>
<td>182 s</td>
<td>3 s</td>
</tr>
</tbody>
</table>

Table 5.6: Algebraic Attack Times on the Self-Decimated Generator

From the results shown, it can be observed that the algebraic attack renders the self-decimated generator insecure when it is used alone for keystream generation.

### 5.5.5 The Strengthened Beth-Piper Generator

The strengthened version of the Beth-Piper stop-and-go generator improves on the basic stop-and-go generator by adding a third LFSR to increase the security of the generator. Its structure is shown in Figure 5.9.

The generator consists of registers $R, S$ of lengths $k, l$ respectively, as well as another regularly clocked register $A$ of length $d$. The output of $A$ is added to the
output of $S$ to form the keystream. The first two registers behave the same way as the stop-and-go generator as

$$s_{i+1}^t = \begin{cases} s_i^t, & r_0^t = 0 \\ s_{i+1}^t, & r_0^t = 1. \end{cases} \quad (5.5.31)$$

In algebraic form, this becomes

$$s_i^t = s_{i-1}^t (r_0^t + 1) + s_{i+1}^{t-1} r_0^{t-1}, \quad 0 \leq i \leq l - 1. \quad (5.5.32)$$

The keystream of the strengthened generator is given by

$$z^t = s_0^t + a_0^t. \quad (5.5.33)$$

Taking the derivative of $z^t$ with respect to $t$ gives

$$z^t + z^{t+1} = (s_0^t + s_1^t) r_0^t + a_0^t + a_0^{t+1}. \quad (5.5.34)$$

Multiplying both sides by $(r_0^t + 1)$ gives

$$(r_0^t + 1)(z^t + z^{t+1}) = (r_0^t + 1)(a_0^t + a_0^{t+1}). \quad (5.5.35)$$

Since $R, A$ are both clocked linearly, (5.5.35) is quadratic in the state bits $r_i^t, a_i^t$ of $R, A$. Note that the state bits of the irregularly clocked register $S$ do not appear in the equations. This means that it is not necessary to consider $S$ at all in the equation generation, and we can avoid the degree increase in the equations due to the clock control mechanism.
5.5. CLOCK-CONTROLLED STREAM CIPHERS

The quadratic equations obtained for $R, A$ has $k+d$ variables, and the initial states of $R, A$ can be solved using Gröbner bases methods. The minimum keystream required for this is $k + d$. Once the initial states in $R, A$ are recovered, we can use (5.5.33) to obtain the initial states of $S$. This requires $l$ bits of keystream. Therefore, the minimum keystream required for the whole attack is $\max(k + d, l)$ bits.

Table 5.7 shows a comparison of attacks on the strengthened Beth-Piper generator. For registers $R, A$ of length $k, d$ respectively, $m(k, d)$ is the minimum keystream required for a generator, and $C_0(k, d), C_1(k, d)$ are the time complexities for pre-computation and the attack respectively. The total attack complexity is given by $C(k, d) = C_0(k, d) + C_1(k, d)$. We also give values of the keystream and the total complexity for register sizes of 64 bits. The entry $*$ indicates that the complexity of the work using a Gröbner bases method is generally unknown. To obtain a measure of this complexity we obtained the empirical results shown in Table 5.8.

Table 5.8 shows the implementation results for the algebraic attack on the strengthened Beth-Piper generator. From the table, $k, d$ are the sizes of registers $R, A$ respectively, $n$ is the number of variables in the system of equations, $m$ is the amount of keystream bits used for the solution of the system, $T_1$ is the time used to generate the system, and $T_2$ is the time to compute the solution of the system.

It can be observed that the quadratic system with initial state variables in register $S$ has a large amount of linear dependencies in the equations generated. In our experiments, significantly more than $n$ equations were required for a unique solution. It can be seen that the more keystream we have, the more efficient it is to find a solution of the system of equations we generate. The keystream needed to obtain a system with a unique solution is approximately quadratic in the number of variables present in the system.

One significant previous attack against the strengthened Beth-Piper stop-and-go generator is the linear syndrome attack [104]. The attack depends on the generation of polynomials of weight three, which requires the computation of discrete logarithms. Our attack is independent of the weight of the polynomial used, and also requires less keystream.
CHAPTER 5. ALGEBRAIC ANALYSIS OF STREAM CIPHERS

Chapter 5

Section 5.5

Linear syndrome [104]

\[ \max(l, n) \]

Table 5.7: Comparison of Attacks on the Strengthened Beth-Piper Generator

<table>
<thead>
<tr>
<th>Attack</th>
<th>( m(k, d) )</th>
<th>( C_0(k, d) )</th>
<th>( C_1(k, d) )</th>
<th>( C(k, d) )</th>
<th>( m(64, 64) )</th>
<th>( C(64, 64) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>( 37 ) ( \max(l, n) ) ( 2^{n^{2/3} + 2^{n^{2(n^{1/3} - 1)/3}}} ) ( (l + n)^3 ) ( 2^{n^{2/3} + 2^{n^{2(n^{1/3} - 1)/3}}} ) ( (l + n)^3 ) ( 2^{12} )</td>
<td>*</td>
<td>*</td>
<td>( 2^7 )</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Our (Gröbner)</td>
<td>( \max(k + d, l) ) ( O(l^2) ) ( O(l^2) ) ( O(l^2) ) ( O(l^2) ) ( O(l^2) )</td>
<td>*</td>
<td>*</td>
<td>( 2^7 )</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8: Attack Times for the Strengthened Beth-Piper Stop-and-Go Generator

<table>
<thead>
<tr>
<th>( k, d )</th>
<th>( n )</th>
<th>( m )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>32</td>
<td>128</td>
<td>0.8 s</td>
<td>32 s</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>256</td>
<td>1.6 s</td>
<td>0.75 s</td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>1024</td>
<td>40 s</td>
<td>60 s</td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>1536</td>
<td>60 s</td>
<td>10 s</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>4096</td>
<td>513 s</td>
<td>3889 s</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>5120</td>
<td>649 s</td>
<td>618 s</td>
</tr>
</tbody>
</table>

It should be noted that it is not possible to solve the system obtained from equation (5.5.35) by linearisation of the system followed by the application of Gaussian elimination. This is because when the equations are multiplied through by, for example, \((a_0 + 1)\), a new set of monomials appears and they must also be linearised into more variables, while the rank of the system stays constant. The system then becomes underdetermined and hence cannot be solved for a unique solution. Therefore, we must instead solve the system by multivariate methods such as those using Gröbner bases.

5.6 Clock-Controlled Cascade Ciphers

The clock-control mechanism for linear feedback shift registers can be repeated used in a multiple register design in the form of clock-controlled cascade stream ciphers. In such ciphers, \( n \) registers \( S_0, S_1, \ldots, S_{n-1} \) are connected such that registers \( S_0, S_1, \ldots, S_{i-1} \) are used to control the clocking of \( S_i \). The output of the
ciphertext can then be taken from combinations of the outputs of all registers. In this section, we present algebraic analyses of two examples of cascade ciphers, namely the Gollmann cascade generator [40] and Pomaranch [49, 50, 51].

The idea of cascading a set of LFSRs was due to Gollmann [40] and was further studied by Chambers and Gollmann in 1988 [41]. In the latter study, they conclude that better security is achieved with a large number of short LFSRs instead of a small number of long ones. Park, Lee and Goh [78], having extended the attack of Menicocci on a 2-register cascade using statistical techniques [72], successfully broke 9-register cascades where each register has fixed length 100. They suggested also that 10-register cascades might be insecure. In 1994, Chambers [14] proposed a clock-controlled cascade cipher in which each 32-bit portion of the output sequence of each LFSR passes through an invertible s-box with the result being used to clock the next register. Several years later, the idea of cascade ciphers resurfaced in a proposal by Jansen, Helleseth and Kholosha [49] to the 2005 SKEW workshop, which became the eSTREAM candidate Pomaranch. Pomaranch can be viewed as a variant of the Gollmann cascade in which a several bits from each register are filtered using a nonlinear function, and the result is used to control the clocking of the next register.

In the case of the Gollmann cascade, the key is the combined initial states of the registers, and the keystream output is the output of the final register. Pomaranch uses an initialization vector for key loading and its keystream output is the sum of certain bits taken from each of the registers. In the subsequent sections, we present our algebraic attacks on the Gollmann cascade generator. This leads us into our algebraic analysis of Pomaranch, where the cipher construction is more complicated than the Gollman cascade. Unless otherwise specified, additions and multiplications presented here are defined over $\mathbb{F}_2$. 

5.6. CLOCK-CONTROLLED CASCADE CIPHERS
5.6.1 The Gollmann Cascade Generator

The Gollmann cascade generator [40] consists of $n$ linear feedback shift registers $S_0, S_1, \ldots, S_{n-1}$ of lengths $l_0, l_1, \ldots, l_{n-1}$ connected serially such that each register except for the first one is clock-controlled by its predecessors. The structure of a Gollmann cascade generator is shown in Figure 5.10.

For $1 \leq i \leq n - 1$, the outputs of $S_0, S_1, \ldots, S_{i-1}$ is used as a clock control for $S_i$. Initially, all registers are filled independently with key bits. The behaviour of the registers is as follows. Let the input bit to the $i$-th register at time $t$ be $a_i^t$, for $i \geq 2$. The $i$-th register is clocked if and only if $a_i^t = 1$. The output bit of the $i$-th register is then added to $a_i^t$, and the result becomes the input bit $a_{i+1}^t$ to the $(i+1)$-th register. Let $s_{i,j}$ be the $j$-th bit of the $i$-th register in the cascade. The clock-control mechanism can then be algebraically expressed as

$$
\begin{align*}
s_{i,j}^{t+1} &= \begin{cases} 
s_{i,j}^t, & \sum_{k=0}^{i-1} s_{k,0}^t = 0, \\
sp{-1} s_{i,j}^t + \sum_{k=0}^{i-1} s_{k,0}^t = 1, & 1 \leq i \leq n - 1.
\end{cases}
\end{align*}
$$

(5.6.1)

After each clock, the keystream output of the generator is the output $s_{n-1,l-1}$ of the final register $S_{n-1}$.

Gollmann first proposed cascading $n$ cyclic registers of the same prime length $p$ with feedback polynomial $f(x) = x^p + 1$. This is known as the $p$-cycle. A variation of the Gollmann cascade, called an $m$-sequence cascade, has the cyclic registers replaced by maximum length LFSRs of the same length $l$. We will algebraically analyse this type of Gollmann cascade generator with registers of variable length,
and present an attack showing how we can recover the initial states of the registers.

We present an algebraic analysis on the clock-controlled Gollmann cascade generator for \( n = 4 \). However, the analysis presented here can be generalised to cascades with any number of registers. Let \( S_0, S_1, S_2, S_3 \) be LFSRs of arbitrary lengths. Let, \( s_{i,j}^t \) be the \( j \)-th bit of register \( s_i \) at time \( t \). Since the first register is regularly clocked, we have

\[
s_{0,i}^{t+1} = s_{0,i+1}.
\]

(5.6.2)

The states of the remaining registers at time \( t \) can be expressed as follows.

\[
s_{1,i}^t = s_{1,i}^{t-1}(s_{0,0}^{t-1} + 1) + s_{1,i+1}^{t-1}s_{0,0}^{t-1},
\]

(5.6.3)

\[
s_{2,i}^t = s_{2,i}^{t-1}(s_{0,0}^{t-1} + s_{1,0}^{t-1} + 1) + s_{2,i+1}^{t-1}(s_{0,0}^{t-1} + s_{1,0}^{t-1}),
\]

(5.6.4)

\[
s_{3,i}^t = s_{3,i}^{t-1}(s_{0,0}^{t-1} + s_{1,0}^{t-1} + s_{2,0}^{t-1} + 1) + s_{3,i+1}^{t-1}(s_{0,0}^{t-1} + s_{1,0}^{t-1} + s_{2,0}^{t-1}).
\]

(5.6.5)

Since the keystream of the generator is given by the output of the final register, we have

\[
z^t = s_{3,0}^t.
\]

(5.6.6)

Using (5.6.5) with \( i = 0 \) and (5.6.6) at time \( t + 1 \), we obtain

\[
z^{t+1} = z^t(s_{0,0}^t + s_{1,0}^t + s_{2,0}^t + 1) + s_{3,1}^t(s_{0,0}^t + s_{1,0}^t + s_{2,0}^t),
\]

(5.6.7)

which can be expressed as

\[
z^t + z^{t+1} = (z^t + s_{3,1}^t)(s_{0,0}^t + s_{1,0}^t + s_{2,0}^t).
\]

(5.6.8)

From (5.6.8), we observe that if \( z^t + z^{t+1} = 1 \), then

\[
1 = s_{0,0}^t + s_{1,0}^t + s_{2,0}^t.
\]

(5.6.9)

From observing (5.6.9), the internal states of all registers in the Gollmann cascade can be recovered as follows. Consider again the case where \( n = 4 \). We know that whenever \( z^t + z^{t+1} = 1 \), then \( s_{0,0}^t + s_{1,0}^t + s_{2,0}^t = 1 \). By guessing the initial states of \( S_0, S_1 \), which will enable us generate linear equations (5.6.4) for \( S_2 \) with the
substitution \( s_{t,0}^t = s_{0,0}^t + s_{1,0}^t + 1 \) for the output of \( S_2 \) when \( z^t + z^{t+1} = 1 \). We then obtain the equations

\[
s_{0,0}^t + s_{1,0}^t + 1 = s_{2,0}^{t-1} (s_{0,0}^{t-1} + s_{1,0}^{t-1} + 1) + s_{2,1}^{t-1} (s_{0,0}^{t-1} + s_{1,0}^{t-1}), \quad t > 0 \quad (5.6.10)
\]

Once we have enough linearly independent equations we can then recover the initial states of \( S_2 \) by solving the system. We can then recover the initial states of register \( S_3 \) by solving the linear equations (5.6.5) by substituting the values for \( S_0, S_1, S_2 \) and the keystream. If we obtain a solution for both \( S_2, S_3 \) after these computations, then the guess of \( S_0, S_1 \) was correct, and we have found the initial states of all registers. The complexity of this approach is given by the complexity of guessing \( S_0, S_1 \) multiplied by the complexity of solving linear equations in the other registers. The time complexity of this attack is then \( 2^{l_0 + l_1} (l_2^3 + l_3^3) \). The minimum keystream requirement for recovering \( S_2 \) is \( 2l_2 \) on average, since we can only obtain a valid equation if \( z^t + z^{t+1} = 1 \). The minimum keystream requirement for recovering \( S_3 \) is \( l_3 \), since every clock gives a valid equation. The same keystream can be used to construct equations for \( S_2 \) and \( S_3 \), so the attack needs \( \max(2l_2, l_3) \) bits of keystream. In practice, we might need a small percentage more than this requirement due to linear dependencies among the linear equations generated.

In general, for a cascade cipher with \( n \) LFSRs, the complexity of the presented approach will be \( 2^u (l_{n-2}^3 + l_{n-1}^3) \), where \( l_{n-2}, l_{n-1} \) are the length of the last two registers respectively, and \( u \) is the total length of the remaining registers. Clearly, the proposed attack is better than exhaustive key search, which has complexity \( 2^{u+l_1+l_2} \).

As far as we are aware, there have been four significant attacks on the Gollmann cascade generator. These are the lock-in effect attack by Gollman and Chambers [13], the attack by Menococci [72], the clock control guessing attack by Zenner [105], and the attack by Park et al. [78]. The complexities of the lock-in effect attack and the Menicocci attack are far higher than ours, so here we will only make comparisons of our attack with the ones that are more effective. The clock control guessing attack has a relatively closer complexity and has a similar approach to ours. Zenner applies linear consistency tests to the Gollmann cascade using a
technique of guessing the clock control bits resulting in a search tree. This attack
is similar to ours in that it forms a set of equations after guessing a number of bits.
It then solves the equations discarding those that are inconsistent. The complexity
of the attack is of order $2^{(l_0+l_1+l_2+l_3)/2(l_0 + l_1 + l_2 + l_3)^3}$.

The only attack that outperforms ours is the probabilistic attack by Park et al.
[78]. This attack is based on guessing the initial states of each register successively
in the cascade with some desired probability. Their analysis starts by building a
matrix of conditional probabilities between the inputs and outputs of a register.
This matrix is used to determine the probabilities of particular outputs to the
cipher given particular inputs to the registers. These probabilities are biased when
a run of zeros or ones occurs, yielding an efficient algorithm for finding the initial
states of the registers by scanning the given keystream for runs of at least $u$
consecutive zero or ones, where $u$ is determined by the desired error rate of the
algorithm.

The algorithm builds linear equations in the unknowns of the initial state bits of
the first register of the cascade. Random equations from these are solved until
a solution with high probability is found. The algorithm then uses this solution
to build equations for the next register. This process is repeated until the initial
states of every register are recovered except for the last. Finally, the initial
states of the last register are recovered by using previously known techniques. The
theoretical complexity of this attack is not given in [78], so we have estimated it
from the algorithm and the experimental data. Let the registers be of the same
length $l$. Gaussian elimination is present, so $O(l^3)$ is used for the asymptotic time
complexity. From the experimental data we deduce that $O(l^2)$ bits of keystream
are required to find the desired run of zeros or ones.

Although Park’s attack is very efficient, it does have some limitations. Park’s
attack requires consecutive keystream bits with runs of zeros or ones, but our
attack does not even require consecutive keystream bits. We can use any subset
of the keystream as long as the equations describing them are dependent on the
values of all of the initial state bits. Our keystream requirement is also much
less than that of Park’s attack. Hence, our attack would be the only one that is
feasible for implementations where rekeying occurs frequently such as the frame based communication systems for mobile telephones and wireless networks. Table 5.9 summarizes the abovementioned attacks on the Gollmann cascade generator for \( n = 4 \).

<table>
<thead>
<tr>
<th>Attack</th>
<th>( m(l_i) )</th>
<th>( C(l_i) )</th>
<th>( m(64) )</th>
<th>( C(64) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock-in Effect [13]</td>
<td>( 4l^2 )</td>
<td>( \frac{16}{l^2}(2^l - 2)^4 )</td>
<td>( 2^{14} )</td>
<td>( 2^{248} )</td>
</tr>
<tr>
<td>Park [78]</td>
<td>( 6l^2 )</td>
<td>( 36l^3 )</td>
<td>( 2^{15} )</td>
<td>( 2^{24} )</td>
</tr>
<tr>
<td>Clock Control Guessing [105]</td>
<td>( \sum l_i )</td>
<td>( 2^{\sum l_i/2}(\sum l_i)^3 )</td>
<td>( 2^{8} )</td>
<td>( 2^{152} )</td>
</tr>
<tr>
<td>Ours</td>
<td>( \max(2l_2, l_3) )</td>
<td>( 2^{l_0+l_1}(l_2^3 + l_3^3) )</td>
<td>( 2^{7} )</td>
<td>( 2^{147} )</td>
</tr>
</tbody>
</table>

Table 5.9: Comparison of Attack Complexities on the Gollmann Cascade Generator

### 5.6.2 Variable Relabelling

In this section, we describe another method of equation generation using tradeoffs between the number of variables and the degree of the equations. The resistance of clock-controlled stream ciphers to traditional algebraic attacks is due to the fact that clock controls cause increases in the degree of the equations generated. In the case of the Gollmann cascade, we obtain (5.6.3)-(5.6.5). The clock control variables are multiplied into the states of the registers, causing an increase in the degree of the equations. Since the clock control of the Gollmann cascade is linear in the register states, the degrees of the equations increase by one at every clock. For example, let the initial state bits of \( R, S \) be \( x_i, y_i \) respectively. The outputs of \( R \) at the first two clocks are then

\[
\begin{align*}
    r_0^0 &= x_0 \\
    r_0^1 &= x_1,
\end{align*}
\]
and the states of $S$ at the first three clocks are expressed as

$$s_i^0 = y_i \quad \text{(5.6.13)}$$
$$s_i^1 = y_i(x_0 + 1) + y_{i-1}x_0 \quad \text{(5.6.14)}$$
$$s_i^2 = y_i(x_0 + 1)(x_1 + 1) + y_{i-1} \{(x_0 + 1)x_1 + x_0(x_1 + 1)\} + y_{i-2}x_0x_1. \quad \text{(5.6.15)}$$

The equations formed in the variables $x_i, y_i$ will increase in degree as the cipher is clocked. However, if we instead use new variables for the register states at every clock, this degree accumulation can be prevented and we obtain quadratic equations in terms of the register states at each time $t$, as shown in (5.6.3), (5.6.4), (5.6.5). Initially, we have $l_0 + l_1 + l_2 + l_3$ variables representing the initial states of the registers. Since $S_0$ is regularly clocked, we do not need to introduce new variables for it. Also, since $s_3^t = z^t$, we can replace that state bit with the keystream output. This means that a total of $l_1 + l_2 + l_3 - 1$ new variables are introduced at each time $t$. Also, a total of $l_1 + l_2 + l_3$ equations are introduced as relations between old and new variables. Since the internal state size is $l_0 + l_1 + l_2 + l_3$, we would need at least $l_0 + l_1 + l_2 + l_3$ clocks to form a system of equations with a unique solution. From the analysis above, we obtain $(l_1 + l_2 + l_3)(l_0 + l_1 + l_2 + l_3)$ quadratic equations in $(l_1 + l_2 + l_3)(l_0 + l_1 + l_2 + l_3)$ unknowns for the Gollmann cascade. These equations can be solved by techniques such as Gröbner basis methods, but not by linearisation. This is because as we introduce more variables, the number of monomials in the system increases at a much higher rate, and it is not possible to obtain enough equations in the linearised variables.

Not so much is known about the practical complexities of solving polynomial equations by Gröbner basis methods, such as those by using Faugère’s $F_4$ and the $XL$ and related algorithms. It is known that the worst case complexity of these algorithms on random systems of equations in $\mathbb{F}_2$ is $O(2^v)$, where $v$ is the number of equations. However, for specific systems and as the number of equations exceeds the number of variables, the complexity of $XL$ and its variants can drop significantly, to even polynomial time. See, for example, [24, 26, 103] for descriptions and analyses of these algorithms. In our case, since the variables are in $\mathbb{F}_2$, we do not need to consider solutions in the algebraic closure of $\mathbb{F}_2$ and can solve the
equations subject to the field equations \( x^2 + x = 0 \) for each variable \( x \) in the system. Therefore, we obtain another \((l_1 + l_2 + l_3)(l_0 + l_1 + l_2 + l_3)\) equations, giving us twice as many equations as variables. Furthermore, the equations are very sparse, since each variable is used only for two clocks. These properties might prove useful at reducing the complexity of finding the solution to the system. The actual efficiency of this attack is yet to be gauged by further research in this area. This concept of variable relabelling as an attack method will be explored in Section 5.6.3 on the Pomaranch stream cipher.

5.6.3 Pomaranch

In this section, we provide an algebraic analysis of version 3 of Pomaranch [49, 50, 51], an eSTREAM stream cipher phase 3 candidate. Here we mainly discuss in the context of the 128-bit version of Pomaranch, but the analysis also holds for the 80-bit version. Although only the 80-bit version of Pomaranch is officially in phase 3 of the eSTREAM project, the 128-bit version will also be important since there are doubts about the security of 80-bit ciphers, as discussed in the eSTREAM project. Earlier versions of Pomaranch have been cryptanalysed in [17, 48, 56]. As of the beginning of phase 2 of the project, version 3 of Pomaranch has been published to address weaknesses found in the above papers with modifications of several components compared to previous versions. As far as we are aware, this is one of the first analyses of version 3 of the cipher.

The overall structure of Pomaranch is shown in Figure 5.11. Pomaranch is a clock-controlled cascade generator consisting of 9 jump registers \( R_1, R_2, \ldots, R_9 \). The jump registers are implemented as autonomous Linear Finite State Machines (LFSM), each containing 18 memory cells, with each cell containing a bit value. Each cell in a jump register behaves as a shift cell or a feedback cell. At each clock, a shift cell simply shifts its state to the next cell, whereas a feedback cell feeds its state back to itself and adds it to the state from the previous cell, as well as performing a normal shift like a shift cell. The behaviour of each cell depends on the value of a Jump Control (JC) signal to the jump register where the cell
belongs. Algebraically, a transition matrix $A$ governs the behaviour of an LFSM. The transition matrices of the jump registers in Pomaranch take the form

$$A = \begin{pmatrix}
d_{18} & 0 & 0 & \ldots & 0 & 1 \\
1 & d_{17} & 0 & \ldots & 0 & t_{17} \\
0 & 1 & d_{16} & \ddots & \vdots & \vdots \\
0 & 0 & \ddots & \ddots & 0 & \vdots \\
\vdots & \vdots & \ddots & 1 & d_2 & t_2 \\
0 & 0 & \ldots & 0 & 1 & d_1 + t_1
\end{pmatrix},$$

where $t_i$ determines the positions of the feedback taps, and $d_i$ determines whether the cells are shift cells or feedback cells. At any moment, half of the cells in the registers behave as shift cells, and the other half as feedback cells, so half the $d_i$ are 0 and the other half are 1. If $JC = 0$ for a certain register, the register is clocked according to its transition matrix $A$. If $JC = 1$, all cells are switched to the opposite behaviour. This is equivalent to switching the transition matrix to $A + I$, where $I$ is the identity matrix. Two different transition matrices $A_1, A_2$ are used for odd (type 1) and even (type 2) numbered registers respectively, with different values of $t_i, d_i$. Each jump register is then connected to a key map, which consists of an s-box and a nonlinear filter. Key bits are diffused into the key map and a one bit output is drawn. The jump control of a register is then taken as the
sum of all outputs from the key maps from all previous registers. The keystream output is given by the sum of the contents of the 17th cells of all registers.

The 128-bit key $k$ of Pomaranch is divided into eight 16-bit subkeys $K_1$ to $K_8$, where $K_i$ represents the key bits $k_{i,1}, k_{i,2}, \ldots, k_{i,16}$. Each section of Pomaranch except the last contains a jump register of length 18 and a nonlinear function composed of an s-box and a degree 4 boolean filter function $f$. The last section of Pomaranch only has a jump register. The key map at section $i$ first takes a 9-bit vector $v$ from cells numbered 1, 2, 4, 5, 6, 7, 9, 10, 11 from jump registers of type 1 or 1, 2, 3, 4, 5, 7, 9, 10, 11 from jump registers of type 2. Then the 9 least significant bits of $K_i$ are added to $v$. The sum is passed through a 9-to-7-bit inversion s-box over GF($2^9$) defined by the primitive polynomial $x^9 + x + 1$. The resulting 7 bits are then added to the 7 most significant bits of $K_i$. This is fed into the boolean function $f$ of degree 4, and the output of $f$ is called the jump control out bit of the section and is denoted as $c_i$. The $c_i$ from each section is used to produce the jump control bits $JC_2$ to $JC_9$ controlling the registers $R_2$ to $R_9$ at time $t$ respectively, as follows.

$$JC_i^t = \sum_{j=2}^{i-1} c_j^t, \quad 2 \leq i \leq 9.$$  

The jump control bit $JC_1$ of register $R_1$ is permanently set to zero.

### 5.6.4 Algebraic Analysis of Pomaranch

In this section we provide the first full algebraic analysis of Pomaranch. From the description of the way that the cipher is clocked, each register can be represented as

$$R^t = (A + JC^t \cdot I)R^{t-1}, \quad (5.6.16)$$

where $R^t$ is the state of a register at time $t$, $A$ is its transition matrix, $JC^t$ is its jump control, and $I$ is the identity matrix. The keystream is given by

$$z^t = \sum_{i=1}^{9} r_{i,17}^t, \quad (5.6.17)$$
5.6. CLOCK-CONTROLLED CASCADE CIPHERS

where \( r_{i,17}^t \) is the 17-th cell of the register \( i \) at time \( t \). In the 80-bit version employing only 6 sections, the keystream is given by

\[
\begin{align*}
    z^t &= g(r_{1,17}^t, r_{2,17}^t, \ldots, r_{5,17}^t) + r_{6,17}^t, \\
    \text{(5.6.18)}
\end{align*}
\]

where \( g \) is a nonlinear filter of degree 3. In order to understand how the key is related to the output and what the relations among different subkey bits are, we analyse what happens in the first section containing \( R_1 \) as the key bits are mixed in a nonlinear manner, and how the key bits are carried across into the second section. Similar analysis follows for the remaining sections.

Let the \( i \)-th bit of \( R_1 \) at time \( t \) be \( r_i \) for \( 1 \leq i \leq 18 \). Nine selected bits \( r_j \) are added to \( k_1, k_2, \ldots, k_9 \) respectively, according to the register type. These are then fed into the key map that consists of an s-box and a boolean function \( f \) of degree 4. Firstly, the bits pass through an inversion s-box over \( \text{GF}(2^9) \). Let the input to the s-box be \( a_1, a_2, \ldots, a_9 \), then we have \( a_i = r_j + k_i \) for \( 1 \leq i \leq 9 \). Let the output from the s-box be \( b_1, b_2, \ldots, b_9 \). Each \( b_i \) can be represented by equations in \( a_i \). An explicit function of each \( b_i \) in terms of \( a_i \) will be of high degree, in our case it is of degree 9, i.e.

\[
b_i = \sum_{e \in \text{GF}(2)^9} (s_e \prod_{i=1}^{9} a_{e_i}) , \quad 1 \leq i \leq 7,
\]

where \( s_e \) are coefficients in \( \text{GF}(2) \). We can also form implicit relations between the inputs and outputs of the s-boxes of degree 2, as was shown in [22]. The relations between the inputs \( a_i \) and the outputs \( b_i \) of the Pomaranch s-box are shown in Figure 5.12.

From the truth table of the s-box shown in [51], we can see that the seven output bits \( b_2, b_3, \ldots, b_8 \) are used for the next component, discarding \( b_1, b_9 \). Those output bits from the s-box are then added to the next 7 key bits and the result \( b_i + k_{8+i} \) for \( 2 \leq i \leq 8 \) is filtered through \( f \), a degree 4 function. The algebraic normal form (ANF) of \( f \) is shown in Figure 5.13.

Assuming we take the explicit functions for the s-box and the ANF of the filter, the output \( c_1 \) from the filter function will be an expression of degree 13 in 16 key
By the end of the first clock, we will have equations of degree 104 in 128 variables. The key map of the second section will mix the next 16-bit subkey bits. This expression is then fed into the next register in the first section, which becomes an input in the second section. As from the analysis above, a degree 13 equation in 16 key bits is generated from the output. The high degree expression and variables from the first section are carried across as well. The key map of the second section will mix the next 16-bit subkey $K_2$ into the expression, raising the degree by 13 to 26 and the number of variables by 16 to 32. This accumulation continues to carry across to $R_3$ and beyond. The number of possible monomials each $JC_i$ possesses at the first clock is then

$$M_{JC_i} = \max \left\{ \sum_{j=0}^{13i} \binom{16i}{j}, 2^{128} \right\}, \quad 1 \leq i \leq 9. \quad (5.6.19)$$

By the end of the first clock, we will have equations of degree 104 in 128 variables.
\[ f(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = x_1x_2x_3x_4 + x_1x_2x_3x_5 + x_1x_2x_3x_6 + x_1x_2x_4x_5 + x_1x_2x_4x_7 + x_1x_2x_6x_7 + x_1x_3x_4x_6 + x_1x_3x_4x_7 + x_1x_3x_5x_7 + x_1x_3x_6x_7 + x_1x_4x_5x_7 + x_2x_3x_4x_6 + x_2x_3x_4x_7 + x_2x_3x_5x_6 + x_2x_3x_5x_7 + x_2x_3x_6x_7 + x_2x_3x_7 + x_1x_4x_5 + x_1x_4x_6 + x_1x_4x_7 + x_1x_5x_7 + x_1x_6x_7 + x_1x_7 + x_2x_4 + x_1x_6 + x_1x_7 + x_1x_9 + x_1x_10 + x_1x_11 + x_1x_12 + x_2x_13 + x_2x_14 + x_3x_15 + x_3x_16 + x_3x_17 + x_3x_18 + x_3x_19 + x_1 + x_2 + x_3 + x_5 + x_6 + x_7 \]

Figure 5.13: The Algebraic Normal Form (ANF) of Filter Function \( f \)

This degree accumulation does not carry over to the next clock, since \( JC_1 \) is set to be constantly zero. Therefore, in order to perform an algebraic attack in this manner, we need at least 128 bits of keystream, giving us 128 equations of degree 104. Although the keystream requirement is low, generating such equations is time and memory consuming and the effort needed in solving them is likely to be much more than that of exhaustive key search.

There are three main components in each section of Pomaranch, namely a jump register, an s-box, and a nonlinear filter. The outputs of each component are nonlinear in its inputs, and the expressions describing the outputs will have a higher degree than those of the inputs. Since each output is fed into the next component, the degree of the expressions accumulates. To prevent this degree accumulation, we introduce new variables so that the nonlinearities are not carried across the components. Let \( r_{i,m}^t \) be the \( m \)-th bit of register \( R_i \) at time \( t \), \( b_{i,m}^t \) be the output bits of the s-box at the \( i \)-th section at time \( t \), \( JC_i^t \) be the jump control input to \( R_i \) at time \( t \), and \( k_{i,m} \) be the key bits used in the \( i \)-th section. We successively introduce the above variables as we step through the keystream generation. At the start, we have 128 variables \( k_{i,m} \) whose values are to be determined. We go through each component and try to discover equations that relate to \( k_{i,m} \). At time
$t$, we proceed as follows. The relations between the jump controls and the registers are

$$R_t^i = \begin{cases} 
A_1 R_{i-1}^t & i = 1 \\
(A_2 + JC_{i-1}^t) R_{i-1}^t & i \in \{2, 4, 6, 8\} \\
(A_1 + JC_{i-1}^t) R_{i-1}^t & i \in \{3, 5, 7, 9\}
\end{cases}$$

The first register will always contain known bits, since it is not under the effect of jump controls. Hence, we have 8 registers with 18 variables each to introduce in each clock. Each new $R_t^i$ is a function of $R_{i-1}^t$. This yields 144 quadratic equations in 144 variables. The s-box is defined by inversion over $\text{GF}(2^9)$, which means that the relations between its input and output bits can be expressed as a system of 9 quadratic equations. The inputs are linear sums of certain register bits and key bits, so we have the equations

$$s_0(r_{i,1} t + k_{i,1}, \ldots, r_{i,11} t + k_{i,9}, b_{i,1}, \ldots, b_{i,9}) = 1$$
$$s_j(r_{i,1} t + k_{i,1}, \ldots, r_{i,11} t + k_{i,9}, b_{i,1}, \ldots, b_{i,9}) = 0, \quad 1 \leq j \leq 8.$$

In each clock we have 8 s-boxes, which yield 72 quadratic equations and 72 new variables $b_{i,m}$. Let $c_i^t$ be the jump control output from the nonlinear filter $f$ of the $i$-th section at time $t$. Then

$$c_i^t = f(b_{i,2} t + k_{i,10}, b_{i,2} t + k_{i,11}, \ldots, b_{i,8} + k_{i,16}) \quad (5.6.20)$$

giving a degree 4 equation. The jump control input to the next register is then

$$JC_{i+1}^t = JC_i^t + c_i^t.$$

We do not assign new variables to $c_i^t$. Therefore, we obtain a degree 4 equation with the new variable $JC_{i+1}^t$. Since there are 8 filters in one clock, we get 8 equations and 8 variables. Finally, at the end of each clock we get the keystream bit

$$z^t = \sum_{i=0}^{9} r_{i,17}^t.$$
We can use the above equation and rewrite it as, for example,

\[ r_{9,17}^t = z^t + \sum_{i=0}^{8} r_{i,17}^t, \]

and we can replace \( r_{9,17} \) with the above expression, thereby eliminating a variable at each clock. As a whole, we obtain 224 equations and 223 new variables at each clock, plus the original 128 variables representing the key bits. In order to obtain a unique solution, we would require at least 128 bits of keystream, giving 28672 equations in 28672 variables. Of these equations, 1024 are of degree four and 27648 are quadratic. In the case of the 80-bit version, we have

\[ z^t = g(r_{1,17}^t, r_{2,17}^t, \ldots, r_{5,17}^t) + r_{6,17}^t, \]

and we can rewrite it as

\[ r_{6,17}^t = z^t + g(r_{1,17}^t, r_{2,17}^t, \ldots, r_{5,17}^t), \]

where \( g \) is the cubic nonlinear filter. This contributes a cubic equation to the system. We then obtain 140 equations and 139 new variables at each clock, plus the original 80 variables which represent the key bits. In order to obtain a unique solution, we would require at least 80 bits of keystream, giving 11200 equations in 11200 variables. Of these equations, 400 are of degree four, 80 are cubic and 10800 are quadratic.

In order to solve the set of equations, we would require techniques such as Gröbner basis methods, since we cannot get enough equations for linearisation. With the addition of field equations into the system, we obtain twice as many equations as variables. The sparse structure of the equations may also give rise to complexity reductions in solving the system by the \( XL \) and related algorithms. This would reduce the complexity of finding the solution significantly from \( O(2^v) \) for random systems, where \( v \) is the number of equations. See, for example, [24, 26, 103] for analyses and details of implementing these algorithms. All key bits can be recovered when the solution to this set of equations is found. While there is yet no
evidence for or against whether this type of attack would be better than exhaustive key search, the size and the form of the equations generated can be used intuitively to judge the cipher’s strength. Algebraic attacks are still a widely discussed and controversial topic in the cryptographic community, so their consequences should not be overlooked.

We note here that the designers of Pomaranch have increased the size of the registers from 14 to 18. This increase has no effect on the size or the degree of the final equations. In fact, the degrees of the equations are independent of the size of the registers. They are affected by the clock control mechanism and nonlinear components.

5.6.5 The Filter Functions in Pomaranch

As discussed in Section 5.3.2, the existence annihilators and low degree multiples of the boolean functions used in stream ciphers may have significant impacts on the security of the ciphers. In Pomaranch, two nonlinear filter functions $f, g$ are used as components of the cipher. As part of the algebra analysis of Pomaranch, several annihilators and low degree multiples of $f, g$ have been discovered.

For both 80-bit and 128-bit versions of Pomaranch, we have computed a cubic annihilator $y_f$ of the degree four filter $f$ such that $fy_f = 0$ and cubic multiples $e_{f,1}, e_{f,2}, e_{f,3}$ such that $h_{f,i} = fe_{f,i}$ are cubic for all $e_i$. These are shown in Figure 5.14.

In particular, the following relations hold.

\[ y_f = e_{f,1} + e_{f,2} \quad (5.6.21) \]
\[ (f + 1)e_{f,1} = 0 \quad (5.6.22) \]
\[ (f + 1)e_{f,3} = 0 \quad (5.6.23) \]

This shows a potential weakness in the filter function $f$. However, in terms of our algebraic attack these multipliers are not useful, because of the newly introduced
which is still of degree 4, due to the presence of $c_i$. If we apply a cubic multiplier $e$ to (5.6.20), we obtain

$$c_i^l e = fe,$$  \hspace{1cm} (5.6.24)

which is still of degree 4, due to the presence of $c_i^l$. To successfully reduce the degree of our equations, we need to find annihilators or low degree multiples of $f + c_i^l$, which we have not been able to do.

For the 80-bit version of Pomaranch, we have also computed a quadratic annihilator $y_g$ of the cubic filter $g$ such that $gy_g = 0$ and quadratic polynomials $e_{g,1}, e_{g,2}$ such that $h_{g,i} = ge_{g,i}$ are cubic for all $e_{g,i}$. These are shown in Figure 5.15.
\[ y_g(x_1, x_2, x_3, x_4, x_5) = x_1x_2 + x_1x_3 + x_2x_3 + x_3x_4 + x_2x_5 + x_2 + x_3 \]
\[ e_{g,1}(x_1, x_2, x_3, x_4, x_5) = x_1x_2 + x_1x_3 + x_3x_4 + x_2x_5 \]
\[ e_{g,2}(x_1, x_2, x_3, x_4, x_5) = x_2x_3 + x_2 + x_3 \]

Figure 5.15: Annihilators and Low Degree Multiples of \( g \) in Pomaranch

In particular, the following relations hold.

\[ y_g = e_{g,1} + e_{g,2} \]  \hspace{1cm} (5.6.25)
\[ (f + 1)g_1 = 0 \]  \hspace{1cm} (5.6.26)

Again, there could be weaknesses in the filter function \( g \). However, our algebraic attack cannot make use of this since we need annihilators or low degree multiples of \( g + r_{6,17} \) in order to reduce the equations describing the cipher.

We leave as an open question other possible uses of these results. However, we believe that the filter function \( f, g \) should be changed so that no annihilators or low degree multiples exist for \( f, g \). This would resist possible future attacks based on algebraic techniques.

### 5.7 Further Notes

The clock control mechanism can be further developed such that each register in a generator is responsible for controlling all other registers. The clock controls can usually be represented by nonlinear functions of certain bits in the available registers in the generator. Examples of mutual clock controls include the A5 family of stream ciphers used in GSM protocols.

Algebraic analyses of stream ciphers can also be extended beyond those based on linear feedback shift registers. In Chapter 6, we present an algebraic analysis of other nonlinear mechanisms that can also be modelled using piecewise boolean
In this chapter, we have presented an algebraic method of attacking a general type of clock-controlled bit-based stream cipher. The method of algebraic analysis on clock controls can be used for a broader range of ciphers in determining their levels of security, and we have demonstrated its effectiveness on the strengthened Beth-Piper Stop-and-Go, self-decimated, Step-1/Step-2 and alternating step generators. In comparing the efficiency of our attacks with the previously known one on the strengthened Beth-Piper generator, it is observed that when the feedback polynomial is not a trinomial, the previously attack needed to generate trinomial polynomials, which is a highly complex procedure. Our results, however, do not depend on constructions of these polynomials. In fact, all attacks presented in this chapter are equally effective regardless of the feedback polynomials of the registers. For the Beth-Piper stop-and-go generator strengthened version and for the self-decimated generator, we obtain significantly better attacks than any other known attack. For the other two ciphers, our attack complexity is not far from the best known attacks.

As far as we know, this is the first attempt at a general algebraic attack approach to bit-based clock-controlled stream ciphers based on linear feedback shift registers, in which one register determines the clocking of other registers in the system. Our results indicate that long register lengths are needed to protect these ciphers against algebraic attacks.

We have also described the first algebraic attacks against clock controlled cascade stream ciphers, in particular the Gollmann cascade and the eSTREAM candidate Pomaranch. We have established relations between the initial state bits and the output bits of the Gollmann clock controlled cascade stream ciphers, and demonstrated that the initial state bits of the last two registers can be determined from

functions. The RC4 family of stream ciphers will be examined as an example of a well-known stream cipher based on dynamic tables.
those of the previous registers, yielding an attack better than exhaustive key search.

For the Gollmann cascade, we also showed that the effect of a clock control on algebraic attacks is that the degree of the equations generated increases with each clock. An alternative attack on the cascade was developed to eliminate the effect of this degree accumulation, resulting in a large sparse low degree polynomial system, which can be generated and solved more efficiently using algorithms such as Faugère’s $F_4$, $F_5$, and the many variants of $XL$.

Our algebraic analysis on Pomaranch further showed that a cipher with nonlinear components can be expressed as a system of equations with maximum degree no higher than the maximum degree of the components. In the analysis, we showed how to generate degree four equations in the key bits and other component bits of the system. The bottleneck to reducing this degree further is due to the filter function of degree four. The input and output sizes of the components determine the number of new variables to be introduced and therefore the size of the system of equations needed to describe the cipher. We also make the observation that increasing the size of the jump registers has no effect on increasing the degree of the equations.

Finally, we have found annihilators and low degree multiples of both filter functions proposed for use in of Pomaranch, which indicates a possible weakness in the cipher.
Chapter 6

Algebraic Analyses with Binary Fields

As shown in Chapter 5, algebraic analysis is a powerful cryptanalytic method on symmetric ciphers. Thus far, their use on stream ciphers is mainly limited to the analysis of bit-based symmetric ciphers with boolean functions. In this chapter, we present expositions of algebraic attacks on alternative forms of stream ciphers. While algebraic analyses are usually performed on bit-based ciphers based on linear feedback shift registers, we apply algebraic analysis to the RC4 family of stream ciphers, which uses a word-based construction with dynamic tables. Additionally, the possibility of using binary extension fields for algebraic analysis of bit-based stream ciphers is investigated, and its equation generation and solution methods are described. We believe that the material presented are the first of their type of in the field of algebraic cryptanalysis.

6.1 Introduction

A comprehensive review of algebraic analyses and attacks, as well as their applications on stream ciphers is presented in Chapter 5. The traditional algebraic
analyses and attacks are often performed on bit-based stream ciphers. Each of the key or initial state bits are assigned a bit variable, and boolean equations are generated from them. This has been a primary method of algebraic analysis of stream ciphers, and many ciphers have been broken in this way. We have already presented algebraic analyses on bit-based clock-controlled stream ciphers and cascade ciphers in Chapter 5 using this traditional algebraic analysis and successfully implemented attacks using Gröbner bases methods.

Algebraic analysis has been demonstrated to be a very useful tool for stream ciphers based on linear feedback shift registers (LFSRs). In contrast, little progress has been made on algebraic analysis and attacks on word-based stream ciphers. One of the few examples of algebraic attacks on word-based stream ciphers can be found in [16]. It is therefore appropriate to extend algebraic analyses to other well-known ciphers that are not based on LFSRs, in order to evaluate the possibility of successful algebraic attacks on them. This is the primary aim of our algebraic analysis of the RC4 family of stream ciphers. In Section 6.2, we will show by way of the RC4 family of stream ciphers how we perform a full algebraic analysis on a word-based stream cipher with boolean equations. The result is a system of multivariate polynomial equations with solutions in $F_2$ representing individual bits of the initial state words of the RC4 family of stream ciphers.

To further widen the applicability of algebraic analysis, the possibility of using extension fields to perform algebraic analyses and attacks of bit-based stream ciphers is discussed in Section 6.3. It is well-known that sequences generated from bit-based linear feedback shift registers can be modelled as sequences in binary extension fields. We take this idea further to model all state bits of the linear feedback shift registers using elements in binary extension fields, and consequently generate equations from them to describe stream ciphers based on the registers.
6.2 The RC4 Family of Stream Ciphers

Our investigation into the resistance of non-LFSR based stream ciphers to algebraic attacks begins with the RC4 family of stream ciphers, which is based on dynamic tables. It will be shown how valid algebraic relations between the internal states of the ciphers and the output bits, as generated from the cipher, are obtained. The resistance of such ciphers to algebraic attacks will be judged by the complexity and the form of the equations generated. To the best of our knowledge, this is the first investigation of the resistance of the RC4 family of stream ciphers to algebraic attacks. The types and number of equations generated from the cipher are discussed, which can be used as a guide for the security of RC4 against algebraic attacks, both at present and for future reference, as methods of solution of large equation systems become more efficient over time. The results presented here may also be extended to RC4 variants, such as RC4A [80] and VMPC [107].

RC4 is a widely used stream cipher on the Internet, in wireless applications and in many commercial products. It is a simple and elegant word-based stream cipher, designed by Rivest in 1987, which was kept secret until 1994 [86]. After the specification of RC4 was revealed, the cipher became the target for cryptanalysis. The first published cryptanalysis of RC4 was [37], followed by a number of interesting papers [57, 33, 70, 79, 80]. Weaknesses identified in the RC4 cipher have motivated the proposal of several strengthened versions of RC4, such as RC4A [80]. Other researchers were inspired by the design of the cipher and proposed stream ciphers based on the design of RC4 such as the 32 and 64-bit RC4 [42] and VMPC [107]. However, more recently, distinguishing attacks were shown to be very effective on both abovementioned strengthened proposals of RC4 and on new RC4 variants [71, 93].

In order to provide an algebraic analysis of the RC4 stream cipher, we first show how relationships between the internal state of the cipher and the outputs can be obtained. The three main operations used in RC4, namely word addition, state extraction and state permutation will be analysed. We will derive algebraic representations of each of the three operations, and construct low degree algebraic
equations from them. The number of equations generated and their maximum degrees will be given. It will be shown that the natures of modular additions, state extractions and the state permutations in RC4 determine the number of equations generated, their degrees and their form. We also arrive at an interesting observation that these three main operations have their own unique contributions to the system of equations derived from the RC4 stream cipher, giving a high degree of resistance to algebraic attacks only when they are combined into one cipher.

6.2.1 Description of RC4

The RC4 stream cipher has a very large internal state compared to the key size. An $n$-bit RC4 consists of a permutation table of $2^n$ words and two pointers $i, j$ of one word each. The total internal state space of RC4 is therefore of size $\log_2(2^n!(2^n)^2)$ bits. For $n = 8$, this is approximately 1700 bits. Two algorithms govern the RC4 stream cipher, namely the key scheduling algorithm (KSA) and the pseudo-random generation algorithm (PRGA). In the KSA, a secret key $k$ is used to load and mix the internal states $S_i$ of the register $S$, resulting in $S$ having some permutation of the $2^n$ possible $n$-bit words. The PRGA then proceeds to generate keystream using the states obtained from the KSA. The KSA and PRGA for RC4 are shown in Figure 6.1. The operations described in the pseudocode are wordwise and the keystream output is denoted by $z$. During the KSA, the permutation

\begin{align*}
KSA(k) & \quad \text{for } i = 0 \text{ to } 2^n - 1 \\
& \quad \quad S_i \leftarrow i \\
& \quad \quad j \leftarrow 0 \\
& \quad \text{for } i = 0 \text{ to } 2^n - 1 \\
& \quad \quad j \leftarrow j + S_i + k_i \mod 2^n \\
& \quad \quad \text{Swap}(S_i, S_j) \\
PRNG(S) & \quad \text{while key generation} \\
& \quad i \leftarrow i + 1 \mod 2^n \\
& \quad j \leftarrow j + S_i \mod 2^n \\
& \quad \text{Swap}(S_i, S_j) \\
& \quad z \leftarrow S_{S_i+S_j}
\end{align*}

Figure 6.1: The KSA of RC4 (left), The PRNG of RC4 (right)
6.2. THE RC4 FAMILY OF STREAM CIPHERS

(0, 1, ..., 2^n - 1) is loaded into the register S. The secret key k is then used to initialize S to a random permutation by shuffling the words in S according to the KSA. Once the KSA is complete, the cipher is ready for keystream generation. The PRGA is used to produce random words based on the permutation in S. Each iteration of the PRGA loop produces one output word, which constitutes n-bits of keystream. As with other stream ciphers, the keystream is bit-wise XORed with the plaintext to obtain the ciphertext.

6.2.2 Algebraic Analysis of RC4

The RC4 class of stream ciphers of word size n uses one register S of length 2^n - 1 with n-bit words representing each state of S. Commonly, RC4 is used with n = 8. However, we will present an algebraic analysis that is applicable for arbitrary n. Before key initialisation, the register states are as follows.

\[ S = (0, 1, \ldots, 2^n - 1) \]

The cipher then initialises according to the KSA. Let the initial state of S after the KSA be

\[ S^0 = (x_0, x_1, \ldots, x_{2^n-1}) \]

where \( x_i \in \mathbb{Z}/(2^n) \). Throughout this analysis, we realise the isomorphism between the residue class ring \( \mathbb{Z}/(2^n) \) and the product ring \( \mathbb{F}_2^n \), so that all equations are generated in a polynomial quotient ring over \( \mathbb{F}_2 \). From here onwards, for \( u \in \mathbb{Z}/(2^n) \) and \( 0 \leq b \leq n - 1 \), let \( u_{(b)} \in \mathbb{F}_2 \) be the b-th least significant bit (LSB) of \( u \in \mathbb{Z}/(2^n) \). Additionally, to denote the b-th least significant bit of a state \( k \) of register S, we use the notation \( S_{k,(b)} \). We now derive the algebraic expressions of operations involved in RC4.
State Extraction

The value of one state $S_i$ in the register $S$ is at times needed. As the $i$ are to be considered as unknown, we must derive an expression that gives the correct value for any possible $i = (i_0, i_1, \ldots, i_{n-1})$. This state extraction operation is algebraically equivalent to computing the value of the piecewise expression

$$S_i = \begin{cases} 
S_0, & i = 0 \\
S_1, & i = 1 \\
\vdots \\
S_{2^{n-1}}, & i = 2^n - 1
\end{cases}$$

where each bit of the states of $S$ and $i$ are the variables. Since the variables are in $\mathbb{F}_2$, the above can be arranged analogously as a boolean expression and written as

$$S_i = \sum_{u=0}^{2^n-1} \left( S_u \prod_{b=0}^{n-1} (i_{(b)} + u_{(b)} + 1) \right).$$

The above expression is ordered by $S_u$ with polynomials in $i_{(b)}$ as coefficients. This can be rewritten to order by monomials in $i_{(b)}$ with $S_k$ as coefficients as

$$S_i = \sum_{c=0}^{2^n-1} \left( \prod_{f=0}^{n-1} i_{(f)}^{e_{(f)}} \left( \sum_{k=1}^{2^n-1} S_k \left( \prod_{g=0}^{n-1} e_{(g)}(i_{(g)} + 1) + 1 \right) \right) \right).$$

Consider a bit position $S_{i_{(b)}}$ throughout the entire register. Due to the fact that $S$ is just a permutation of its initial states, the values of that position must contain equal numbers of zeros and ones, which means that

$$\sum_{k=0}^{2^n-1} S_{k_{(b)}} = 0, \quad 0 \leq b \leq n - 1.$$
Therefore, the expressions for state extraction can be written as

\[
S_{i,(b)} = \sum_{e=0}^{2^n-2} \left( \prod_{f=0}^{n-1} t_{(f)}^e \left( \sum_{k=1}^{2^n-1} \prod_{g=0}^{n-1} e(g)(i(g) + 1) + 1 \right) \right).
\]

This removes the degree \( n + 1 \) terms in the expression, and we are left with expressions of maximum degree \( n \) for the extraction of \( S_i \). For example, with \( n = 2 \) the expression is

\[
S_{i,(b)} = i(0)i(1)S_{0,(b)} + i(0)i(1)S_{1,(b)} + i(0)i(1)S_{2,(b)} + i(0)i(1)S_{3,(b)}
+ i(0)S_{0,(b)} + i(1)S_{3,(b)} + i(1)S_{1,(b)} + i(1)S_{2,(b)} + S_{3,(b)}
= i(0)S_{0,(b)} + i(0)S_{3,(b)} + i(1)S_{1,(b)} + i(1)S_{2,(b)} + S_{3,(b)},
\]

since \( S_0 + S_1 + S_2 + S_3 = 0 \). The expression for \( S_{i,(b)} \) is therefore of degree 2.

**Word Addition**

The addition operation is defined as addition modulo \( 2^n \). To obtain the equivalent operations using variables in \( \mathbb{F}_2 \), we create a group isomorphism between \((\mathbb{Z}/(2^n); +)\) and \((\mathbb{F}_2^n; \star)\), where + is the canonical addition modulo \( 2^n \), and \( \star \) is as defined below.

\[
u \star v = (u_{(0)}, u_{(1)}, \ldots, u_{(n-1)}) \star (v_{(0)}, v_{(1)}, \ldots, v_{(n-1)})
= (w_{(0)}, w_{(1)}, \ldots, w_{(n-1)}).
\]

where \( w_{(b)} \) satisfies

\[
w_{(b)} = \sum_{k=0}^{b-1} \left( u_{(k)}v_{(k)} \prod_{l=k+1}^{b-1} (u_{(l)} + v_{(l)}) \right) + u_{(b)} + v_{(b)}, \quad 0 \leq b \leq n - 1.
\]

With this definition we then have the homomorphism

\[
w_{(b)} = (u \star v)_{(b)} = u_{(b)} + v_{(b)}.
\]
This amounts to degree \( n + 1 \) expressions in the bit variables for addition. It can be seen that these expressions are independent of \( n \). The first few expressions are as follows.

\[
\begin{align*}
  w_0 &= u_0 + v_0, \\
  w_1 &= u_0v_0 + u_1 + v_1, \\
  w_2 &= u_1u_0v_0 + v_1u_0v_0 + u_1v_1 + u_2 + v_2, \\
  w_3 &= u_1u_2u_0v_0 + u_1v_2u_0v_0 + u_2v_1u_0v_0 + v_1v_2u_0v_0 \\
  &\quad + u_1u_2v_1 + u_1v_1v_2 + u_2v_2 + u_3 + v_3.
\end{align*}
\]

For simplicity, from here on we denote all addition operations, including that over \((\mathbb{F}_2^2; \oplus)\), using +, where there the appropriate operation can be inferred from context.

**State Permutation**

Swapping states \( i, j \) in \( S \) can be algebraically described as the action of a permutation matrix \( M \) on \( S \). Given \( i, j \), the entries \( m_{r,s} \) of \( M \) are constructed as follows.

- A diagonal entry \( m_{r,r} \) is set if \( i = j \) or both \( i \neq r \) and \( j \neq r \)
- An off-diagonal entry \( m_{r,s} \) is set if \( \{i, j\} = \{r, s\} \)

Using the above rules we can create the appropriate boolean function used by each entry \( m_{r,s} \) of \( M \) in the bits \( i_{(b)}, j_{(b)} \). Algebraically, the diagonal entries are given
by

\[ m_{r,r} = \prod_{b=0}^{n-1} (i(b) + j(b) + 1) + \left( 1 + \prod_{b=0}^{n-1} (i(b) + r(b) + 1) \right) \left( 1 + \prod_{b=0}^{n-1} (j(b) + r(b) + 1) \right) \]

\[ + \prod_{b=0}^{n-1} (i(b) + j(b) + 1) \left( 1 + \prod_{b=0}^{n-1} (i(b) + r(b) + 1) \right) \left( 1 + \prod_{b=0}^{n-1} (j(b) + r(b) + 1) \right). \]

(6.2.1)

The off-diagonal entries are given by

\[ m_{r,s} = m_{s,r} = \prod_{b=0}^{n-1} (i(b) + r(b) + 1) \prod_{b=0}^{n-1} (j(b) + s(b) + 1) + \prod_{b=0}^{n-1} (i(b) + s(b) + 1) \prod_{b=0}^{n-1} (j(b) + r(b) + 1). \]

For example, the symmetric permutation matrix with \( n = 2 \) is

\[
\mathbf{M} = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{0,1} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{0,2} & m_{1,2} & m_{2,2} & m_{2,3} \\ m_{0,3} & m_{1,3} & m_{2,2} & m_{3,3} \end{pmatrix}
\]

with entries

\[
m_{0,0} = i(0) i(1) + j(0) j(1) + i(0) + i(1) + j(0) + j(1) + 1,
\]

\[
m_{0,1} = i(0) i(1) j(1) + i(1) j(0) j(1) + i(0) i(1) + i(0) j(1) + i(1) j(0) + j(0) j(1) + i(0) + j(0),
\]

\[
m_{1,1} = i(0) i(1) + j(0) j(1) + i(0) + j(0) + 1,
\]

\[
m_{0,2} = i(0) i(1) j(0) + i(0) j(0) j(1) + i(0) i(1) + i(0) j(1) + i(1) j(0) + j(0) j(1) + i(1) + j(1),
\]

\[
m_{1,2} = i(0) i(1) j(0) + i(0) i(1) j(1) + i(0) j(0) j(1) + i(1) j(0) j(1) + i(0) j(1) + i(1) j(0),
\]

\[
m_{2,2} = i(0) i(1) j(0) + i(0) j(0) j(1) + i(1) + j(1) + 1,
\]

\[
m_{0,3} = i(0) i(1) j(0) + i(0) i(1) j(1) + i(0) j(0) j(1) + i(1) j(0) j(1) + i(0) i(1) + j(0) j(1),
\]

\[
m_{1,3} = i(0) i(1) j(0) + i(0) j(0) j(1),
\]

\[
m_{2,3} = i(0) i(1) j(1) + i(1) j(0) j(1),
\]

\[
m_{3,3} = i(0) i(1) + j(0) j(1) + 1.
\]
The entries of $M$ have maximum degree 3. In the next sections we will show how these algebraic representations can be used to describe RC4.

### 6.2.3 Equation Generation

In this section we show that most operations in RC4 are prone algebraically to annihilators and low degree multiples. By introducing variables at each step of the algorithm we can use these to simplify most of the equations generated. Let $S^t, i^t, j^t$ be the values of $S, i, j$ respectively at the end of clock $t$. The relations between the parameters of RC4 can then be expressed as follows.

\[
\begin{align*}
    i^t &= i^{t-1} + 1 & \text{(pointer increment)} \\
    j^t &= j^{t-1} + S_i^{t-1} & \text{(pointer addition)} \\
    S^t &= MS^{t-1} & \text{(state permutation)} \\
    z^t &= S_{S_i^t + S_j^t}^t & \text{(keystream generation)}
\end{align*}
\]

Each operation shown above will be algebraically analysed below.

#### Pointer Increment

In the first step, $i$ is incremented by 1. This addition is represented by

\[
\begin{align*}
    i^t_{(0)} &= i^{t-1}_{(0)} + 1, & \text{(6.2.2)} \\
    i^t_{(b)} &= i^{t-1}_{(b)} + \prod_{k=0}^{b-1} i^{t-1}_{(k)}, & 1 \leq b \leq n - 1. & \text{(6.2.3)}
\end{align*}
\]
6.2. THE RC4 FAMILY OF STREAM CIPHERS

Moving all terms to the left hand side and multiplying through by \((i_{(0)}^{t-1} + 1)\) gives, for the first bit of \(i^t\),

\[
i_{(0)}^t + i_{(0)}^{t-1} + 1 = 0 \quad (6.2.4)
\]

\[
(i_{(0)}^{t-1} + 1)(i_{(0)}^t + i_{(0)}^{t-1} + 1) = 0 \quad (6.2.5)
\]

\[
(i_{(0)}^{t-1} + 1)(i_{(0)}^t + 1) = 0 \quad (6.2.6)
\]

and for the rest,

\[
i_{(b)}^t + i_{(b)}^{t-1} + \prod_{k=0}^{b-1} i_{(k)}^{t-1} = 0 \quad (6.2.7)
\]

\[
(i_{(0)}^{t-1} + 1)
\left( i_{(b)}^t + i_{(b)}^{t-1} + \prod_{k=0}^{b-1} i_{(k)}^{t-1} \right) = 0 \quad (6.2.8)
\]

\[
(i_{(0)}^{t-1} + 1)(i_{(b)}^{t-1} + i_{(b)}) = 0, \quad 1 \leq b \leq n - 1. \quad (6.2.9)
\]

These equations make sense, because when adding 1 to the pointer \(i\), the LSB of \(i\) must change. This gives (6.2.6). Also, if the LSB of \(i\) is originally 0, then there is no carry and the rest of the bits stay the same. This gives (6.2.9). The result is that all equations describing this addition are of maximum degree 2.

**Pointer Addition**

The contents of \(S_i\) are then extracted, which gives

\[
S_{i,(b)}^t = \sum_{k=0}^{2^n-1} \left( S_{k,(b)}^t \prod_{l=0}^{n-1} (i_{(l)}^{t} + k(l) + 1) \right), \quad 0 \leq b \leq n - 1.
\]

From the analysis in section 6.2.2, this can be expressed as

\[
S_{i,(b)}^t = \sum_{e=0}^{2^n-1} \left( \prod_{f=0}^{n-1} i_{(f)}^{e(f)} \left( \sum_{k=1}^{2^n-1} S_{k,(k)}^t \left( \prod_{g=0}^{n-1} e(g)(i_{(g)} + 1) + 1 \right) \right) \right).
\]
This gives a degree \( n \) equation. The addition for \( j \) is then given as follows.

\[
\begin{align*}
j_t(0) &= j_t^{t-1} + S_{i_t(0)}, \\
j_t(b) &= \sum_{k=0}^{b-1} \left( j_t^{t-1} S_{i_t(k)} \prod_{l=k+1}^{b-1} (j_t^{t-1} + S_{i_t(l)}) \right) + j_t^{t-1} + S_{i_t(b)}, \quad 1 \leq b \leq n - 1.
\end{align*}
\]

Moving all terms to the left hand side and multiplying the expressions for \( j_t^b \) by \((j_t^{t-1} + 1)(S_{i_t(b-1)} + 1)\) gives

\[
(j_t^{t-1} + 1)(S_{i_t(b-1)} + 1)(j_t^{t-1} + S_{i_t(b)} + j_t^b) = 0.
\]

This gives equations of maximum degree 3 for the full word addition operation.

**State Permutation**

The new pointers \( i^t, j^t \) are then used for state swapping in register \( S \). As mentioned before, the diagonal entries of the permutation matrix \( M \) are given by

\[
m_{r,r}^t = \prod_{b=0}^{n-1} (i_t^b + j_t^b + 1) + \left( 1 + \prod_{b=0}^{n-1} (i_t^b + r_t^b + 1) \right) \left( 1 + \prod_{b=0}^{n-1} (j_t^b + r_t^b + 1) \right)
+ \prod_{b=0}^{n-1} (i_t^b + j_t^b + 1) \left( 1 + \prod_{b=0}^{n-1} (i_t^b + r_t^b + 1) \right) \left( 1 + \prod_{b=0}^{n-1} (j_t^b + r_t^b + 1) \right)
\]

(6.2.10)

The off-diagonal entries are given by

\[
m_{r,s}^t = m_{s,r}^t = \prod_{b=0}^{n-1} (i_t^b + r_t^b + 1) \prod_{b=0}^{n-1} (j_t^b + s_t^b + 1) + \prod_{b=0}^{n-1} (i_t^b + s_t^b + 1) \prod_{b=0}^{n-1} (j_t^b + r_t^b + 1)
\]

It can be observed that multiplying each entry \( m_{r,s} \) of the permutation matrix \( M \) by

\[
\sigma_{r,s}^t = \left( \sum_{b=0}^{n-1} i_t^b \right) \left( \sum_{b=0}^{n-1} r_t^b \right) \left( \sum_{b=0}^{n-1} j_t^b \right) \left( \sum_{b=0}^{n-1} s_t^b \right)
\]
gives a low degree multiple of the original expression of the entry, which is of maximum degree 3. In order to incorporate \( \sigma \) into our equations, we can relabel and multiply each entry of \( M \) to obtain the degree 3 expression \( \sigma m_{r,s} \). The number of equations introduced as a result would be \( 2^n - 1 \), since \( M \) is symmetric. An additional \( 2^n - 1 \) linear expressions are required for the row sums of the matrix, which are the new states of register \( S \). This is quite uneconomical for an algebraic analysis. Alternatively, let

\[
M = \begin{pmatrix}
  m_{0,0} & m_{0,1} & \cdots & m_{0,2^n-1} \\
  m_{0,1} & m_{1,1} & \cdots & m_{1,2^n-1} \\
  \vdots & \vdots & \ddots & \vdots \\
  m_{0,2^n-1} & m_{1,2^n-1} & \cdots & m_{2^n-1,2^n-1}
\end{pmatrix},
\]

\[
S^t = \begin{pmatrix}
  S^t_{0,(0)} & S^t_{0,(1)} & \cdots & S^t_{0,(n-1)} \\
  S^t_{1,(0)} & S^t_{1,(1)} & \cdots & S^t_{1,(n-1)} \\
  \vdots & \vdots & \ddots & \vdots \\
  S^t_{2^n-1,(0)} & S^t_{2^n-1,(1)} & \cdots & S^t_{2^n-1,(n-1)}
\end{pmatrix}.
\]

The swapping action can then be described as the multiplication

\[
S^t = MS^{t-1}.
\]

Hence, we have

\[
S^t_{i,(b)} = \sum_{k=0}^{2^n-1} m_{k,b} S^{t-1}_{k,(b)}, \quad 0 \leq i \leq 2^n - 1,
\]
where $m_{u,v}$ are the matrix entries of $M$. Applying the multiplier to the above equation gives

$$S^t_{i,(b)} + \sum_{k=0}^{2^n-1} m_{b,k} S^{t-1}_{k,(b)} = 0,$$

$$\sigma_{r,s}^t \left( S^t_{i,(b)} + \sum_{k=0}^{2^n-1} m_{b,k} S^{t-1}_{k,(b)} \right) = 0,$$

$$\sigma_{r,s}^t \left( S^t_{i,(b)} + \sum_{k=0}^{2^n-1} S^{t-1}_{k,(b)} \right) = 0.$$ 

This amounts to $n$ equations of maximum degree 3, with no relabeling of matrix entries needed. Equations generated using this method will be considered in our algebraic analysis in subsequent sections.

**Keystream Generation**

Finally, state extraction is used twice to obtain a keystream word at each clock. Let $r^t$ be the index of the state from which the keystream output is to be taken.

$$r^t_{(b)} = S^t_{i,(b)} + S^t_{j,(b)} = \sum_{k=0}^{2^n-1} \left( S_{k,(b)} \prod_{b=0}^{n-1} (i^t_{(b)} + k_{(b)}) \right) + \sum_{k=0}^{2^n-1} \left( S_{k,(b)} \prod_{b=0}^{n-1} (j^t_{(b)} + k_{(b)}) \right)$$

The keystream $z^t$ is then given by extracting state $r^t$ of register $S$. Thus,

$$z^t_{(b)} = S_{r^t_{(b)}} = \sum_{k=0}^{2^n-1} \left( S_{k,(b)} \prod_{b=0}^{n-1} (r^t_{(b)} + u_{(b)}) \right)$$

This amounts to 2 equations of degree $n + 1$ for each bit with one introduced variable $r^t$ at each clock, since $z^t$ is assumed to be known.
6.2. THE RC4 FAMILY OF STREAM CIPHERS

6.2.4 Summary of Equations in RC4

Based on the investigation from the last section, the number of equations generated for each bit per clock at each operation is shown in Table 6.1.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of Equations</th>
<th>Number of Variables</th>
<th>Maximum Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer Increment for $i$</td>
<td>$n$</td>
<td>$n$</td>
<td>2</td>
</tr>
<tr>
<td>State Extraction for $S_i$</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Pointer Addition for $j$</td>
<td>$n$</td>
<td>$n$</td>
<td>2</td>
</tr>
<tr>
<td>State Permutation</td>
<td>$n2^n$</td>
<td>$n2^n$</td>
<td>3</td>
</tr>
<tr>
<td>Keystream Generation</td>
<td>$2n$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

Table 6.1: Summary of Equations Generated for RC4

From these results, we present an analysis of the RC4 cipher in the following sections.

6.2.5 RC4 as an Algebraic Cipher

From the cipher description, one would normally expect the high nonlinearity of RC4 to arise from the state permutation operation. However, it has been shown in Table 6.1 that it is in fact due to the state extraction, since there seems to be no low degree equations that can describe the operation in terms of the internal states. On the other hand, the state permutation makes a primary contribution to the number of equations generated from the cipher, because the operation affects the register as a whole. Also, note that the equations for the individual bits in a word become mixed only at the word additions, due to the effect of the carry; otherwise, the equation system could be separated, and each bit position can be solved for separately. It is quite interesting to see that each of the three main operations involved in the RC4 stream cipher has its own role in increasing the security of the cipher when realised from an algebraic point of view, since algebraic attacks have not made their appearance in the literature at the time when RC4 was designed.
Together, the three operations yield a strong algebraic system, whose properties will be described below.

6.2.6 Algebraic Attacks on RC4

From the equation analysis, we obtain $n2^n + 5n$ equations in $n2^n + 4n$ variables per bit in each clock of the cipher. With a register size of $2^n$ words, hence $n2^n$ additional initial state variables, we require at least $2^n$ clocks to result in a system with a unique solution, since each clock gives $n$ bits of output. This amounts to at least $2^n(n2^n + 6n)$ equations in $2^n(n2^n + 6n)$ variables. A total of $n$ systems of equations of this size are to be solved to recover all $n$ bits of the words in the initial states. For the common 8-bit RC4 cipher, this gives a system of at least 536576 equations of maximum degree 8 in at least 536576 variables, with 6144 of them having maximum degree more than 3. These equations are very sparse, as each variable is only related to those at the immediately preceding, current, and immediately succeeding clocks. This is an estimate, as the numbers may rise if dependencies are found among the equations, or may fall if more low degree multiples are found.

The purpose of this analysis on RC4 is to consider the impact of applying algebraic analyses techniques to non-algebraically oriented stream ciphers. As such, these techniques do not provide a measure of the absolute or relative effectiveness of an algebraic attack against the RC4 stream cipher and its variants. Therefore, no comparisons are drawn with existing attacks, and we do not claim any advantages or disadvantages of this method over any existing ones. Sound complexity analyses on algebraic attacks are often difficult to achieve due to their reliance on algorithms for solving large sparse multivariate systems of equations of varying forms, which in turn belongs to an area whose theory is yet to be fully developed and documented.

Nevertheless, recent progress suggests that by implementing specialised routines to target equations generated from particular ciphers, one can improve greatly the efficiency of solution. These include the Gröbner basis [26] and the Boolean Satis-
6.3. EXTENSION FIELD ALGEBRAIC ANALYSIS

fiability (SAT) [23] algorithms. As the research on algebraic attacks progresses it is quite reasonable to believe that solution techniques will improve in the foreseeable future. Therefore it is important to discuss equation generation so that the feasibility of solution to ciphers can be constantly monitored as time passes.

6.2.7 Section Summary

As far as we are aware, we have presented here the first algebraic analysis of a non-LFSR based stream cipher. A method was shown for obtaining relationships between the internal states and the outputs of the well-known RC4 stream cipher. It was shown how the state extraction, word addition, and state permutation operations can be represented in terms of algebraic equations. From these equations, we can see that having state extractions yields a system of high degree, having word addition makes the equation system inseparable, and state permutation is the main source of equations. If any of these components are compromised (for example, low degree multiples are found for the state extraction operation), it will have a significant effect on the security of RC4 and its variants against algebraic attacks. This is left to the reader as an open question.

6.3 Extension Field Algebraic Analysis

In Section 6.2, we have seen the method of deriving boolean equations for word-based stream ciphers. In this section, we describe an alternative method of deriving equations for bit-based stream ciphers with Linear Feedback Shift Registers (LFSRs), by considering the states of the LFSRs as variables in extension fields. The polynomial equations generated can be univariate and the methods of solving such equations are in general more easily implemented than that of multivariate polynomial equations. The complexity of solving univariate equations is also well studied. This may be a good alternative to traditional algebraic attacks where each state bit of the LFSRs is assigned as a variable.
6.3.1 Introduction

The methods of algebraic attacks proposed recently have led to the rise of a new generation of symmetric ciphers with a focus on resisting these attacks. However, the true effectiveness of algebraic attacks remains unknown in general, and can vary greatly for different ciphers. This is because the complexity of solving large systems of polynomial equations of a certain form cannot be easily derived. In light of this, we propose an alternative algebraic analysis. In a keystream generator based on LFSRs, the complexity of recovering the initial states translates to that of solving a set of univariate polynomial equations.

Analyses of linear feedback shift registers (LFSRs) and variants using extension fields have existed for a long time [44, 84]. These analyses primarily focus on the nonlinear or statistical properties of the LFSR outputs. With the new designs of stream ciphers and recent discoveries of algebraic attacks, it makes sense to investigate whether extension fields can be used in algebraic analyses of stream ciphers. This might give some interesting results that confirm or complement those from the traditional analyses with boolean equations.

As with the traditional method of algebraic analysis, our objectives in the extension field algebraic analysis are to express the keys or initial states of a bit-based keystream generator as variables in extension fields or other derived structures, and to construct valid equations relating the keystream to these variables, such that there is a method of efficiently computing the solution to these equations.

6.3.2 Algebraic Analysis of LFSR Generators

For details of linear feedback shift registers and the traditional algebraic attack methods, see Section 5.3. This type of algebraic attack requires methods of solving multivariate polynomial equations with variables in \( \mathbb{F}_2 \). These include the usage of, for example, Gröbner bases and the boolean satisfiability algorithms. The complexity of these are often difficult to analyse.
Nevertheless, algebraic analysis in $\mathbb{F}_2$ has easy access to the internal state bits, since each initial state bit is individually labelled as a variable. This provides the flexibility to describe complex ciphers, even those based on non-traditional structures such as RC4, as presented in Section 6.2. However, the number of variables in the system of equations usually grows very large for ciphers of reasonable security. Together with the possibility of obtaining high degree equations from the cipher, computing solutions from these multivariate systems of equations may be time consuming. This leads us into the consideration of reducing the number of variables by performing algebraic analysis in extension fields. The extension field representation of linear feedback shift registers is closely related to the trace map in binary fields [84]. The trace map has been used for public key cryptography in Chapter 4. Here, we present the trace map specifically for binary fields.

**Definition 6.1.** Let $\mathbb{F}_2$ be the field of two elements, and $\mathbb{F}_{2^n}$ be an extension field of degree $n$ over $\mathbb{F}_2$. The trace map of $\mathbb{F}_{2^n}$ with respect to its ground field $\mathbb{F}_2$ is defined as

$$\text{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2} : \mathbb{F}_{2^n} \to \mathbb{F}_2$$

$$x \mapsto \sum_{i=0}^{n-1} x^{2^i}. \quad (6.3.1)$$

The value of $\text{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(x)$ is called the trace of $x$ with respect to $\mathbb{F}_2$.

From here on, we simply use $\text{Tr}(x)$ to denote the trace of $x$ in the above context. Let $S_n$ be the set of all possible states of an $n$-bit LFSR. Clearly, we have $|S_n| = 2^n$. It is well-known that each $u \in \mathbb{F}_{2^n}$ can be associated with a unique state $s \in S_n$. This means that there exists a bijection between $\mathbb{F}_{2^n}$ and $S_n$. The explicit map can be found in [44] as follows.

**Lemma 6.2.** Let $S$ be a linear feedback shift register of length $n$ with feedback polynomial $p \in \mathbb{F}_2[x]$. Define the extension field

$$\mathbb{F}_{2^n} \equiv \frac{\mathbb{F}_2[x]}{(p)} \equiv \mathbb{F}_2(\alpha), \quad (6.3.2)$$

where $\alpha \in \mathbb{F}_2^n$ is a root of $p$. The bijection between the extension field $\mathbb{F}_{2^n}$ and the
set of all possible states $S_n$ is given by

$$
\varphi : \mathbb{F}_{2^n} \rightarrow S_n,
$$

$$
x \mapsto (\text{Tr}(x), \text{Tr}(\alpha x), \text{Tr}(\alpha^2 x), \ldots, \text{Tr}(\alpha^{n-1} x)).
$$

(6.3.3)

**Example 6.3.** Let $S$ be a linear feedback shift register of length 4. Let the primitive feedback polynomial of $S$ be

$$
p = t^4 + t + 1 \in \mathbb{F}_2[t].
$$

(6.3.4)

Define the quadratic binary extension field

$$
\mathbb{F}_{2^4} \cong \mathbb{F}_2[x]/\langle f(x) \rangle.
$$

(6.3.5)

The field elements of $\mathbb{F}_{2^4}$ form a bijection with the states of the variables. Take an element say $x = \alpha^2 + 1 \in \mathbb{F}_{2^4}$. Its corresponding state of $S$ can be computed as

$$
(\text{Tr}(x), \text{Tr}(\alpha x), \text{Tr}(\alpha^2 x), \text{Tr}(\alpha^3 x)) = (1, 0, 1, 1).
$$

(6.3.6)

Conversely, if $x$ is to be found from the initial states $s_i^0$, we solve the system

$$
s_i^0 = \text{Tr}(\alpha^i x), \quad 0 \leq i \leq n - 1
$$

(6.3.7)

where

$$
x = \sum_{i=0}^{n-1} x_i \alpha^i.
$$

(6.3.8)

This computation takes at most $O(n^3)$ time, which is polynomial in the length of the register.

From this description of linear feedback shift registers, we arrive at the following classic result on the output of the an LFSR using extension fields and traces. This can be found in [84].

**Lemma 6.4.** Let $S$ be a linear feedback shift register of length $n$ with feedback
polynomial \( p \in \mathbb{F}_2[x] \). Define the extension field

\[
\mathbb{F}_{2^n} \equiv \frac{\mathbb{F}_2[x]}{\langle p \rangle} \equiv \mathbb{F}_2(\alpha),
\]

(6.3.9)

where \( \alpha \in \mathbb{F}_2^n \) is a root of \( p \). Suppose that the initial state of \( S \) corresponds to \( x \) by the bijection defined earlier. Then, the sequence \( z^0, z^1, z^2, \ldots \) generated by \( S \) is given by

\[
z^t = \text{Tr}(\alpha^t x), \quad t > 0.
\]

(6.3.10)

### 6.3.3 Non-Linear Components

Let \( S \) be a linear feedback shift register of length \( n \), and \( \mathbb{F}_{2^n} \) be its corresponding extension field. Let \( x \in \mathbb{F}_2^n \) be the initial state of \( S \), and \( s^t_i \) be the value of the \( i \)-th bit of the register at time \( t \). Since \( s^t_i \) will be the output of the register at time \( t + i \), the state bit can be expressed as

\[
s^t_i = \text{Tr}(\alpha^{t+i} x)
\]

The ability to extract each bit of a state at time \( t \) is important as nonlinear components often act on a combination of individual bits of the register states. One example is in the analysis of nonlinear filter generators. A schematic of a nonlinear filter generator is shown on Figure 6.2. See Chapter 5 for details and an example of a traditional algebraic analysis on the generator.

![Figure 6.2: A Nonlinear Filter Generator](image)
A nonlinear filter $f$ applied on a register $S$ is traditionally expressed as a boolean function in several variables. Let $z^t$ be the keystream output of the filter generator with inputs from the bits of the register $S$ at time $t$, then

$$z^t = f(s^t_0, s^t_1, \ldots, s^t_{l-1}), \quad t > 0.$$  \hfill (6.3.11)

With the extension field representation with respect to the initial state $x$ of register $S$, this becomes

$$z^t = f(\text{Tr}(\alpha^t x), \text{Tr}(\alpha^{t+1} x), \ldots, \text{Tr}(\alpha^{t+n-1} x)), \quad t > 0$$  \hfill (6.3.12)

As discussed in Section 2.3.3, we can obtain common solutions to these equations by using greatest common divisor (GCD) computations. When $m$ is large enough, we obtain a linear polynomial describing the common root $x$, which is the initial state of $S$. In general, we can recover a unique $x \in \mathbb{F}_{2^n}$ with $m = O(n)$ equations. From $x$ we can compute

$$s_i = \text{Tr}(\alpha^i x).$$  \hfill (6.3.13)

to recover the initial state bits of the register.

**Example 6.5.** Consider a filter generator with a linear feedback shift register $S$ and a filter function $f$. Let the feedback be given by the primitive polynomial

$$p = t^4 + t + 1 \in \mathbb{F}_2[t].$$  \hfill (6.3.14)

Let the filter function in 4 inputs be

$$f(s_t) = s_{t,0}s_{t,1} + s_{t,1}s_{t,3} + s_{t,2} + s_{t,3}
= \text{Tr}(\alpha^{t+0} x) \text{Tr}(\alpha^{t+1} x) + \text{Tr}(\alpha^{t+1} x) \text{Tr}(\alpha^{t+3} x) + \text{Tr}(\alpha^{t+2} x) + \text{Tr}(\alpha^{t+3} x).$$  \hfill (6.3.15)

To recover the initial state $x$ of register $S$ from known keystream bits $z^0, z^1, \ldots, z^m$, we first generate the equations

$$z^t = f(s_t), \quad 1 \leq t \leq m.$$  \hfill (6.3.16)
The function evaluates to the following at the first four clocks.

\[
f(s_0) = x^{16} + \alpha^{13}x^{12} + \alpha^5x^{10} + \alpha^{11}x^9 + \alpha^{14}x^8 + \alpha^{14}x^6 \\
\quad + \alpha^{10}x^5 + \alpha^7x^4 + \alpha^7x^3 + \alpha^{11}x^2 + \alpha^6x,
\]

\[
f(s_1) = \alpha x^{16} + \alpha^{10}x^{12} + x^{10} + \alpha^5x^9 + \alpha^7x^8 + \alpha^5x^6 + x^5 \\
\quad + \alpha^{11}x^4 + \alpha^{10}x^3 + \alpha^{13}x^2 + \alpha^7x,
\]

\[
f(s_2) = \alpha^2x^{16} + \alpha^7x^{12} + \alpha^{10}x^{10} + \alpha^{14}x^9 + x^8 + \alpha^{11}x^6 \\
\quad + \alpha^5x^5 + x^4 + \alpha^{13}x^3 + x^2 + \alpha^8x,
\]

\[
f(s_3) = \alpha^3x^{16} + \alpha^4x^{12} + \alpha^5x^{10} + \alpha^8x^9 + \alpha^8x^8 + \alpha^2x^6 \\
\quad + \alpha^{10}x^5 + \alpha^4x^4 + \alpha x^3 + \alpha^2x^2 + \alpha^9x
\]

We then construct the polynomials

\[
g_0 = f(s_0) + z^0, \quad (6.3.17)
\]

\[
g_1 = f(s_1) + z^1, \quad (6.3.18)
\]

\[
g_2 = f(s_2) + z^2, \quad (6.3.19)
\]

\[
g_3 = f(s_3) + z^3, \quad (6.3.20)
\]

whose common roots are to be sought. Computing the GCD of these polynomial gives

\[
h = \gcd(g_0, g_1, g_2, g_3) = x + \alpha^3 + 1. \quad (6.3.21)
\]

Since \( h \) is a linear polynomial, it immediately follows without the need to factorise that the initial state of \( S \) is \( \alpha^3 + 1 \).

Actual algebraic attacks have been implemented for filter generators of small sizes. Table 6.2 shows some experiments on recovering the initial states from filter generators using the traditional method of algebraic attacks and the method described above. In the table, \( n \) is the length of the register. Nonlinear filter functions with \( e \) input bits and degree \( d \) are used. The traditional method uses \( T_1 \) time and \( M_1 \) memory, while the extension field method uses \( T_2 \) time and \( M_2 \) memory. All computations are performed on version 2.9 of the MAGMA computer algebra package.
using an SGI Origin 3000 on the IRIX 64 platform.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$e$</th>
<th>$d$</th>
<th>$T_1$</th>
<th>$M_1$</th>
<th>$T_2$</th>
<th>$M_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>4</td>
<td>0.020 s</td>
<td>196 kB</td>
<td>0.011 s</td>
<td>$\ll 1$ kB</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>4</td>
<td>2.63 s</td>
<td>9206 kB</td>
<td>0.52 s</td>
<td>308 kB</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>4</td>
<td>42.36 s</td>
<td>106610 kB</td>
<td>170.85 s</td>
<td>20311 kB</td>
</tr>
</tbody>
</table>

Table 6.2: Algebraic Attacks on the Filter Generator

It can be observed that algebraic attacks on the filter generator using the extension field representation are feasible. The attack times are similar in order of magnitude to those of the traditional method. Note that the MAGMA package has an extremely efficient Gröbner basis algorithm for boolean polynomials, so the times might be slightly biased toward the traditional attack. Also, when $n = 16$, the polynomials generated from the extension field attack are of maximum degree $2^{16}$, which might start to be too large for the GCD algorithm to run as proportionally efficiently. This could explain the reason the traditional attack becomes more efficient than the extension field attack in the last row of the table. There is also smaller memory requirements in computing greatest common divisors compared to computing Gröbner bases. This is reflected in the table as the extension field algebraic attack uses about an order of magnitude less memory than the traditional algebraic attack.

6.3.4 Extension Field Elements

One drawback of the extension field algebraic analysis is that the equations generated contain polynomials of degree $2^n$, where $n$ is the length of the LFSR in the generator. It is then natural to ask the question of whether we can reduce the degree of the polynomials. This is possible if we know that the order of $x \in \mathbb{F}_{2^n}$ is less than $2^n$.

The multiplicative group $\mathbb{F}^*_2$ is cyclic, so the number of elements in $\mathbb{F}_{2^n}$ of order $m$ is $\phi(m)$, where $\phi$ is the Euler totient function. The proportion of initial states from the space $\mathbb{F}^n_2$ whose corresponding elements in $\mathbb{F}_{2^n}$ are not of maximal order
is
\[ \rho_n = \frac{(2^n - 1) - \phi(2^n - 1)}{2^n - 1}. \]
The value of \( \rho \) would increase as the number of prime factors of \( 2^n - 1 \) decreases. In the case where \( n = 2^r \), which is often a popular choice in modern stream ciphers, we have \( \phi(2^{2^r} - 1) = 2^{2^r} - 1 \) and
\[ \rho_{2^r} = \frac{2^{2^r} - 1}{2^{2^r} - 1}, \]
which tends to \( \frac{1}{2} \) as \( r \) grows. This means that nearly half of the initial states would be considered as weak. On the other hand, if \( 2^n - 1 \) is a Mersenne prime \( p \), then \( \phi(2^n - 1) = 2^n - 2 \) and we obtain
\[ \rho_p = \frac{1}{2^p - 1}, \]
which becomes negligible as \( p \) grows. Therefore, it would be better to use register lengths \( n \) such that \( 2^n - 1 \) is a Mersenne prime. A necessary condition for this is that \( n \) is prime.

Other security measures that a cipher should have to resist extension field analysis can be inferred from the above analysis. For example, nonlinear components should be chosen so that all powers of the element appear in the equations generated, since lacunary polynomials may reduce GCD computation time and keystream requirement.

### 6.3.5 Multiple Registers

Let \( S_0, S_1, \ldots, S_{n-1} \) be LFSRs of lengths \( l_0, l_1, \ldots, l_{n-1} \), and \( p_0, p_1, \ldots, p_{r-1} \in \mathbb{F}_2[x] \) be the primitive feedback polynomials of \( S_0, S_1, \ldots, S_{n-1} \) respectively. Let
\[ K_i = \frac{\mathbb{F}_2[x]}{(p_i)}, \quad 0 \leq i \leq r - 1, \]
and $\alpha_i$ be a root of $p_i$. As discussed in the single register case, the $j$-th state bit of a register $S_i$ at time $t$ can be denoted as $s_{i,j}^t \in K_i$. Let the initial states of $S_i$ be $x_i$. We can then construct algebraic equations in the variables $x_0, x_1, \ldots, x_{n-1}$ for the initial states of $S_0, S_1, \ldots, S_{n-1}$ respectively. This leads to a system of multivariate polynomial equations, and requires tools such as Gröbner bases in order to find solutions.

Alternatively, this can be analysed as follows to create a univariate system. Since $p_0, p_1, \ldots, p_{n-1}$ are primitive, there exists the isomorphism

$$R \cong \mathbb{F}_2[x] / \left( \prod_{i=0}^{n-1} p_i \right) \cong \bigoplus_{i=0}^{n-1} K_i. \quad (6.3.22)$$

Additionally, we can construct the homomorphisms

$$\phi_i : R \to R$$

$$x \mapsto x \mod p_i, \quad 0 \leq i \leq r - 1,$$

and trace maps

$$\text{Tr}_i : R \to R$$

$$x \mapsto \left( \sum_{j=0}^{l_i-1} x^{2^j} \right) \mod p_i, \quad 0 \leq i \leq n - 1.$$

It is then possible to represent the combined states of the registers as ordered $n$-tuples

$$s = (s_0, s_2, \ldots, s_{n-1}) \in \bigoplus_{i=0}^{n-1} K_i. \quad (6.3.23)$$

These $n$-tuples can be expressed with one element in $R$ by

$$s = (\phi_1(s), \phi_2(s), \ldots, \phi_{r-1}(s)) \in R. \quad (6.3.24)$$

The set of states with all LFSRs non-zero is the set of units $R^*$ in $R$, which can
be given by
\[ R^* = \{ s \in R : \phi_i(s) \neq 0, 0 \leq i \leq n - 1 \}. \] (6.3.25)
It follows that \( |R^*| = \prod_{i=0}^{n-1} (2^l - 1) \), the number of possible states of the cipher with no register having the all zero state. Let \( x = (x_1, x_2, \ldots, x_{r-1}) \in R \) be the combined initial state of the cipher. Let \( \alpha^i \) be a primitive element of \( K_i \). Analogous to the one register case, the bits \( s^t_{i,j} \) of each register state \( s_{i,j} \) of register \( S_i \) at time \( t \) can be expressed in terms of \( x \) as
\[ s^t_{i,j} = \text{Tr}_i (\alpha^{t+j} x_i). \] (6.3.26)
This can be computed in \( R \) as
\[ s^t_{i,j} = \text{Tr}_i (\phi_i (\alpha^{t+j}) \phi_i (x)). \] (6.3.27)

**Example 6.6.** Define a simple additive generator with \( r \) linear feedback shift registers, where the keystream is given by the boolean sum of \( r \) registers.

\[ z^t = \sum_{i=0}^{r-1} s^t_{i,0} = \sum_{i=0}^{r-1} \text{Tr}_i (\phi_i (\alpha^t) \phi_i (x)). \] (6.3.28)

Here we present an additive generator with three registers. Let
\[ p_0 = x^3 + x + 1 \] (6.3.29)
\[ p_1 = x^4 + x + 1 \] (6.3.30)
\[ p_2 = x^5 + x^2 + 1 \] (6.3.31)
be the primitive feedback polynomials for LFSRs \( S_1, S_2, S_3 \) respectively. We then have
\[ R \equiv \frac{F_2[x]}{(p_1 p_2 p_3)} \cong \frac{F_2[x]}{(p_1)} \times \frac{F_2[x]}{(p_2)} \times \frac{F_2[x]}{(p_3)}. \] (6.3.32)
Let the initial state of the generator be
\[ x = (x_0, x_1, x_2) \in R. \] (6.3.33)
where \( x_0 \in \mathbb{F}_{2^3}, x_1 \in \mathbb{F}_{2^4}, x_2 \in \mathbb{F}_{2^5} \) are the initial states of \( S_0, S_1, S_2 \) respectively. If the outputs \( y_i \) from filters \( f_i \) of each register \( S_i \) are used in the summation instead, the keystream is given by

\[
z^t = \sum_{i=0}^{r-1} y_i^t = \sum_{i=0}^{r-1} f_i(s^t_{i,0}, s^t_{i,1}, \ldots, s^t_{i,l_i}).
\] (6.3.34)

If, additionally, a security amplifier \( F \) is used to combine outputs \( y_i \) from each filter \( f_i \), then

\[
z^t = F(y_0, y_1, \ldots, y_{r-1}).
\] (6.3.35)

All of the above settings can be analysed algebraically, and equations can be generated to relate the keystream with the initial states of the LFSRs.

The quotient ring analysis reduces to extension field analysis in the single register case with \( n = 1 \). In the multiple register case, we obtain polynomials over quotient rings. The computation of GCDs of these polynomials is theoretically possible, since we are still working with Euclidean domains, but we are not aware of implementations of GCD algorithms of this kind. If an efficient method is available, this method of algebraic analysis can be applied to many more realistic cipher designs, such as the clock control mechanisms discussed in Chapter 5.

### 6.3.6 Comparison with Traditional Algebraic Attacks

In algebraic attacks on bit-based stream ciphers, each initial state bit is treated as a variable in \( \mathbb{F}_2 \). Therefore, to describe a cipher with an \( n \)-bit register, we have to use equations in \( n \) variables. It is well-known that in order to compute a unique solution for equations in \( n \) variables, at least \( n \) equations are required. Hence, breaking the cipher reduces to solving a system of \( n \) equations in \( n \) variables over \( \mathbb{F}_2 \). As discussed in Chapter 2, the most widely known methods of solving a generic multivariate system are via Gröbner bases computation. For a zero-dimensional variety over \( \mathbb{F}_2 \), the worst time complexity of these algorithms is \( O(2^{2^n}) \), although in practice it can be much less. Claims have been made that the complexity can be
less than that of exhaustive key search for algorithms that target specific systems [103].

In the method presented in this section, we only use one variable to represent the initial states of the cipher, so the equations derived are univariate. This means that we can solve the equations independently of each other. Root finding algorithms have complexity $O(d^2)$, where $d$ is the degree of the polynomial. It is clear that if the degree of the equations derived from the cipher can be made lower, then the algebraic attack can succeed with complexity less than that of an exhaustive key search. However, it is unknown at this time whether this degree $d$ can become less than exponential in the size of the registers.

### 6.3.7 Section Summary

In this section, we have presented a novel alternative to the traditional algebraic analysis with boolean equations. By representing the states of linear feedback shift registers as elements in extension fields and quotient rings, it is possible to derive univariate polynomial equations describing the cipher. In the single register case, we are able to solve the systems generated using greatest common divisor (GCD) computations. This method requires less memory than the computation of Gröbner bases, which are often used in computing the solutions of multivariate polynomial systems in traditional algebraic analysis. Parallel computation is also possible since the GCD computations can be easily split into separate parts.

Furthermore, although much is already known about the algebraic structure of LFSR-based stream ciphers, exploiting the structure in extension fields could reveal other properties, which could lead to simplifying algebraic attacks and security considerations. We leave as an open question whether this method of algebraic analysis would be useful in future theory and practice, or can eventually supersede or complement existing algebraic analyses with boolean equations.
6.4 Summary

In this chapter, two non-traditional applications of algebraic analysis and attacks were presented. First, a full algebraic analysis of the RC4 family of stream ciphers, an example of a stream cipher that is not based on linear feedback shift registers, was given. The operations involved in RC4 were described algebraically, which shows how each of them contribute to the overall security of the cipher against. A general method of algebraic analysis and attacks on stream ciphers based on linear feedback shift registers was then presented, by the use of extension fields and quotient rings. The systems of equations generated can be univariate, compared to multivariate in many variables in traditional algebraic analysis. The computation of solutions to these univariate systems require less memory, and is easily parallelisble. With the two novel methods presented in this chapter, we hope to widen the range of possible applications of algebraic analyses and attacks.
Chapter 7

Conclusion

This thesis shows how finite fields serve as an important element in cryptography and cryptanalysis of both public key and symmetric key systems. As a concluding chapter, we summarise the major outcomes presented in this thesis. Future directions and open questions arising from this research are also discussed.

7.1 Summary of Thesis

This thesis demonstrates the relevance of the theory and applications of arithmetic and computation in finite fields for use in cryptology. Extensions $\mathbb{F}_{p^n}$ of general prime fields $\mathbb{F}_p$ have been used for public key cryptography, and binary fields $\mathbb{F}_{2^n}$ have been used for symmetric key cryptanalysis.

In Chapter 2, we have reviewed methods of solving different types of equations and equations systems over finite fields, which are important for algebraic cryptanalysis. Novel methods were also developed to simply and solve multivariate systems of boolean equations through the use of truth tables and graph theory, which merits further investigation.

In Chapter 3, we have developed the first systematic method for extension field
arithmetic, by using an algebraic approach to describe the multiplication algorithm in extension fields. The method can be used for Karatsuba multiplication and its variants. In many instances, better optimisation of the multiplication algorithms than those previously reported were achieved. In Chapter 4, these algorithms were then applied to public key systems based on extension fields such as the trace-based and torus-based cryptosystems. Again, we were able to improve on the efficiency of the arithmetic in this systems.

In Chapter 5, we have provided the first algebraic analysis on some irregularly clock-controlled stream ciphers, a topic which has not been previously discussed widely in the literature. Our target stream ciphers included the various clock-controlled generators, the Gollmann cascade generator, and the eCRYPT stream cipher project candidate Pomaranch. For each of these ciphers we have generated the equations and discussed their security against algebraic attacks, and some have been shown to be weak. For some ciphers, we have also run experiments of actual algebraic attacks, and showed the feasibility of these attacks.

In Chapter 6, we have made what we believe to be the first algebraic analysis of the RC4 family of stream ciphers in the open literature, and commented on its design and its security against algebraic attacks from the perspective of its algebraic structure. It has been shown that each operation in RC4 has its role in contributing to the overall large, high degree and inseparable system of equations generated from the cipher, which makes the solution of the system difficult. We have also shown that all the operations involved are necessary for RC4 to remain secure against algebraic attacks.

Also in Chapter 6, we have suggested a novel alternative of algebraic cryptanalysis by the use of equations over binary extension fields, as opposed to the use of boolean equations in the traditional method. The equations generated using this method can be univariate, and the solution techniques are simpler and consume less memory. We have shown the feasibility of the method by successfully launching algebraic attacks on filter generators, and also discussed the possibility of extending the algebraic analysis to ciphers with multiple registers.
7.2 Future Directions

One promising application of efficient extension field arithmetic in cryptology is in the area of pairing based cryptography. Pairing-based cryptography [9] is a category of public key cryptography where the public key is the encryptor’s identity. The protocols that are built within this framework usually use a pairing map from an elliptic curve over a finite field to an extension field. Since its discovery, many protocols have been developed, and much research has been done to improve the efficiency of these protocols. Along with the pairing computation, extension field arithmetic is an important element in these schemes. It would then be interesting to see how the work presented in this thesis can be adapted into the field of pairing-based cryptography.

Since the discovery of algebraic cryptanalysis, there have been significant changes to the design criteria of symmetric ciphers. The method of algebraic analysis and attacks on clock-controlled stream ciphers presented in this thesis also introduce some security considerations in future cipher proposals. This work can be extended to include a wider variety of irregularly clocked stream ciphers. One example of these is mutually clock-controlled stream ciphers, the algebraic analyses of which has been discussed in [1]. It would be worthwhile to perform algebraic analysis on industrial strength ciphers of this type to gauge their security against possible attacks.

Algebraic cryptanalysis has mostly targeted ciphers with the usual components such as linear feedback shift registers, nonlinear filters and substitution boxes. From the work presented in this thesis on the algebraic analysis of the RC4 family of stream ciphers, it can be seen that more ciphers with distinct components may be vulnerable to algebraic cryptanalysis. These components can include, but are not limited to, registers with carry, registers with memory, and nonlinear registers.

The extension field and quotient ring representation of linear feedback shift registers allow univariate polynomial equations describing stream ciphers to be generated. It is believed that the expository work presented in this thesis can be taken
further to include different types of stream ciphers. From the theory developed, more work can be done on performing experiments on ciphers with multiple registers. There would also be significant improvements to this method of algebraic analysis if the equations generated can be simplified.

7.3 Open Questions

Here, we present several open questions based on the material presented in this thesis.

It is not a trivial task to develop new algorithms for solving multivariate systems of boolean equations do not often appear in literature. The material presented in this thesis on using truth tables and graph theory to simplify these systems of equations seems not to have discussed previously in open literature. Could this lead to a novel solution technique, and be used in algebraic attacks? Would equations systems generated from stream ciphers of a certain form be vulnerable to analysis with truth tables and graph theory?

The most popular method of performing extension field multiplication seems to be the subquadratic Karatsuba algorithm, where extension field elements are represented as polynomials. However, asymptotically, there are other polynomial multiplication algorithms such as those due to Toom-Cook and Schönhage-Strassen. Although for small extension degree, Karatsuba is usually more efficient, at which point should we switch to the more sophisticated algorithms, and can we improve on these algorithms to match the performance of the Karatsuba algorithm? In addition, can we develop subquadratic algorithms for extension field multiplication where elements are not represented with respect to polynomial bases, but, for example, with normal bases?

Algebraic cryptanalysis consists of two stages, which are the generation of equations describing the cipher, followed by the solution of the equations generated. The first stage is usually straightforward, but sometimes inefficient due to the
exponential time complexity of polynomial multiplication. Is there any more efficient method for equation generation other than having to multiply polynomials at each clock, given that the polynomials are very similar? In the current literature, the second stage seems to be more important as it directly affects how secure the target cipher is. Can any improvement be made to the solution of the systems of equations generated from the first stage, either for particular ciphers or in general?

The algebraic analysis of the RC4 family of stream ciphers presented in this thesis has given us insights into the security of RC4 against algebraic attacks. Is it possible to launch an actual attack on RC4 and its variants based on these results? Can we perform similar analyses on other word-based ciphers or ciphers with unusual structures? Algebraic analysis on stream ciphers based on linear feedback shift registers using extension fields has been shown feasible and revealed some interesting properties of these stream ciphers. How much room is there for the improvement of this method of algebraic analysis? Can we infer more security considerations that complement those from traditional algebraic analysis? Does this method scale up for multiple registers, or more complex designs with various components?

7.4 Closing Remarks

The aims and objectives set out in Chapter 1 of this thesis have been successfully completed. Both reviews and contributions to the theory and applications of arithmetic and computation in finite fields in the area of cryptology have been made. It is worth noting that this thesis contains novel material for both public and private key schemes, and both cryptography and cryptanalysis. This result emphasises the importance of the role that finite fields play in all major areas of cryptology. Future directions and open questions arising from this thesis have also been discussed. Investigations into these may lead to challenging and rewarding research.
Appendix A

Algebraic Preliminaries

The theory of cryptology is based on many ideas in mathematics, particularly from discrete mathematics. This chapter provides most of the necessary background this thesis, which includes areas such as abstract algebra, finite fields and ideal theory. The material presented here are primarily drawn from [6, 68].

A.1 Abstract Algebra

Definition A.1. A set $S$ of is a collection of distinct elements $s_0, s_1, \ldots, s_{n-1}$, commonly expressed as $S = \{s_0, s_1, s_2, \ldots\}$. The number of elements in $S$, called the cardinality of $S$, is denoted $|S|$. If $S$ has a finite number of elements, say $n$, then $|S| = n$. If $S$ has an infinite number of elements, then $|S| = \infty$.

Definition A.2. A group $(G; \bullet)$ is a set $G$ with an operation $\bullet$ over $G$ that satisfies the following properties.

- **Closure** For all $a, b \in G$, $a \bullet b \in G$

- **Associativity** For all $a, b, c \in G$, $a \bullet (b \bullet c) = (a \bullet b) \bullet c \in G$

- **Identity** For all $a \in G$, there exists $e \in G$ such that $a \bullet e = e \bullet a = a$
Inverse For all $a \in G$, there exists $c \in G$ such that $a \bullet c = c \bullet a = e$

The group $(G, \bullet)$ is commonly denoted as $G$ when the operation can be inferred from context. If the addition operation is used, $G$ is called an additive group. If the multiplication operation is used, $G$ is called a multiplicative group.

Definition A.3. A commutative group is a group that satisfies the following additional property.

Commutativity For all $a, b \in G$, $a \bullet b = b \bullet a$

Example A.4. The group $\mathbb{Z}$ of integers is an additive commutative group.

Definition A.5. Let $g \in G$ be an element in a multiplicative group $G$, the $n$-th power $g^n$ of $G$ is computed by

\[ g^n = g \bullet g \bullet g \bullet \ldots \]

For simplicity, from here on we work only with addition and multiplication operations. It is noted that these can be replaced by arbitrary operations in full generality.

Definition A.6. A ring $(R; +, \times)$ is an additive group $(R; +)$ with an extra multiplication operation $\times$ over $R$ that satisfies the following properties.

Associativity over Multiplication For all $a, b \in G$, $a \times b \in G$

Distributivity For all $a, b, c \in G$, $a \times (b + c) = a \times b + a \times c$

The ring $(R; +, \times)$ is usually denoted as $R$ with the canonical addition and multiplication over $R$.

Definition A.7. A commutative ring $R$ with unity is a ring $R$ with the following extra properties.
Commutativity over Multiplication For all $a, b \in G$, $a \times b = b \times a$

Unity Element For all $a \in R$, there exists $u \in R$ such that $a \times u = u \times a = a$

Unless specified, every ring discussed in this thesis is a commutative ring with unity.

Definition A.8. A field $K$ is a ring $R$ that satisfies the following additional properties.

No Zero Divisors For all $a, b \in K$, if $a \times b = 0$, then $a = 0$ or $b = 0$

Inverses over Multiplication For all $a \in F$ with $a \neq 0$, there exists an element denoted by $a^{-1}$ such that $a \times a^{-1} = a^{-1} \times a = 1$.

A.1.1 Quotients

Definition A.9. Let $G$ be a commutative group, $H$ be a subgroup of $G$, and $g \in G$. The coset $gH$ of $H$ in $G$ is given by

$$gH = \{gh : h \in H\}$$

Definition A.10. A relation $r$ on a set $S$ is a subset $R$ of the cartesian product $S \times S$. If $(a, b) \in R$, $a$ is said to be related to $b$ through $r$, and is written as $a \, r \, b$.

Definition A.11. Let $S$ be a set. An equivalence relation $\sim$ on $S$ is a relation such that the following properties hold.

Reflexivity For all $a \in S$, $a \sim a$

Transitivity For all $a, b \in S$, if $a \sim b$ and $b \sim c$, then $a \sim c$

Symmetry For all $a, b \in S$, if $a \sim b$, then $b \sim a$
**Definition A.12.** Let $S$ be a set, and $\sim$ be an equivalence relation on $S$. Let $a \in S$. An *equivalence class* of $a$ with respect to $\sim$ is the set

$$[a] = \{b \in S : b \sim a\}$$

The element $a$ is called a *representative* of the equivalence class $[a]$.

**Example A.13.** Let $R$ be a ring, and $I$ be an ideal of $R$. The congruence relation $\equiv$ on $R$ defined by

$$a \equiv b \quad \text{if and only if} \quad a + I = b + I$$

is an equivalence relation on $R$. The equivalence class an element $a \in R$ with respect to $\equiv$ is the additive cosets of $a$ with respect to $I$ in $R$, namely

$$[a] = a + I, \quad a \in R$$

**Definition A.14.** Let $R$ be a commutative ring with unity and $I$ be an ideal of $R$. The *quotient ring*, also called the *residue class ring* or *factor ring*, $R/I$ is the ring of all equivalence classes of $R$ modulo $I$. The map

$$\varphi : R \to R/I$$

$$x \mapsto [x]$$

is a homomorphism, and is called the natural homomorphism from $R$ to $R/I$.

Integer arithmetic modulo $n$ is arithmetic in the residue class ring $\mathbb{Z}/n\mathbb{Z}$. For convenience, when it is known from context that we are working in a quotient ring, the minimal representative from each equivalence class is used instead of the equivalence classes itself.

**Theorem A.15 (First Isomorphism Theorem).**

$$\frac{R}{\ker(\varphi)} = \text{img}(\varphi)$$


A.1.2 Field Theory

**Definition A.16.** Let $L$ be a field. If there exists $K \subset L$ such that $K$ is a field under the operations of $L$, then $K$ is called a subfield of $L$, and $L$ is called an extension field, or an extension field of the base field $K$. The extension is denoted as $L/K$. If $K \neq L$, then $K$ is a *proper subfield* of $L$.

**Definition A.17.** A *prime field* is a field containing no proper subfields.

**Definition A.18.** Let $L$ be an extension field of $K$. Consider $L$ as a vector space over $K$. The *degree* of the extension $L/K$ is the dimension of the vector space of $L$ over $K$, denoted as $[L : K]$. If $[L : K]$ is finite, then $L$ is called a *finite extension* of $K$.

**Theorem A.19.** If $M$ be an extension field of $L$, and $L$ be an extension field of $K$, then $M$ is an extension field of $K$ with

$$[M : K] = [M : L][L : K]$$

**Example A.20.** The complex numbers $\mathbb{C}$ is an extension field of real numbers $\mathbb{R}$ with extension degree $2$.

Fields such as the real and complex numbers are called infinite fields, since they contain an infinite number of elements. In modern cryptology, we are mainly concerned with fields with a finite number of elements, since they are more suitable discrete nature of digital computation.

**Definition A.21.** A *finite field* $K$ is a field with a finite number of elements.

**Theorem A.22.** The cardinality of a finite field $K$ is $p^n$, where $p$ is prime and $n > 0$.

All finite fields of the same cardinality are isomorphic to each other. Therefore, a general finite field is usually written as $\mathbb{F}_{q^n}$, where $q$ is a prime power and $n > 0$ is the degree of extension from $\mathbb{F}_q$. When $q = p$ is a prime, we write $\mathbb{F}_{p^n}$ instead.
**Definition A.23.** Let $\mathbb{F}_{p^n}$ be a finite field, where $p$ is prime and $n > 0$. The finite field $\mathbb{F}_p$ is called the *base field* of $\mathbb{F}_{p^n}$. The prime $p$ is called the *characteristic* of $\mathbb{F}_{p^n}$.

### A.2 Number Theory

**Definition A.24.** Let $a, b$ be positive integers and $a \neq 0$. We say $a$ *divides* $b$, or $a$ is a *divisor* of $b$, if $b = ac$ for some $c \in \mathbb{Z}$. This is denoted as $a \mid b$.

**Definition A.25.** An integer $p \geq 2$ is a *prime* if and only if the only divisors of $p$ are $1, p$.

**Definition A.26.** Let $a, b \in \mathbb{Z}$. The *greatest common divisor* of $a$ and $b$ is an largest integer $c$ such that $a \mid c$ and $b \mid c$. We write $\text{gcd}(a, b) = c$.

**Definition A.27.** Two integers $a, b$ are *relatively prime* to each other if $\text{gcd}(a, b) = 1$.

**Definition A.28.** The $n$-th *Mersenne number* $M_n$ for $n > 0$ is a number of the form $(2^n - 1)$.

**Definition A.29.** A *Mersenne prime* is a Mersenne number that is a prime.

A Mersenne number $M_n$ can be prime only if $n$ is prime, since if $n = rs$, then $(2^r - 1)$ is a divisor of $M_n$.

**Definition A.30.** The totient $\phi(n)$ of a positive integer $n$ is defined as number of positive integers less than $n$ that are relatively prime to $n$. The function $\phi$ is often called the Euler totient function.

### A.3 Commutative Algebra

**Definition A.31.** Let $R$ be a commutative ring with unity. An *ideal* $I \subset R$ of $R$ is a nonempty subset of $R$ that satisfies the following properties.
i. If \( a, b \in I \), then \( a - b \in I \)

ii. If \( a \in I \) and \( r \in R \), then \( ar \in I \)

**Definition A.32.** Let \( R \) be a commutative ring with unity and \( a_0, a_1, \ldots, a_{n-1} \in R \). An ideal \( I \subseteq R \) generated by \( a_0, a_1, \ldots, a_{n-1} \) is the smallest ideal of \( R \) containing \( a_0, a_1, \ldots, a_n \). This is denoted by \( I = \langle a_0, a_1, \ldots, a_{n-1} \rangle \).

From here on, all rings \( R \) are commutative with a unity element.

**Definition A.33.** Let \( R \) be a ring and let \( I, J \) be ideals of \( R \). The arithmetic of ideals are as follows.

\[
I + J = \{a + b : a \in I, b \in J\}
\]

\[
IJ = \{ab : a \in I, b \in J\}
\]

**Definition A.34.** Let \( R \) be a commutative ring with unity. A non-zero ideal \( I \) of \( R \) is called a *prime ideal* if \( I \mid I_1I_2 \), then \( I \mid I_1 \) or \( I \mid I_2 \).

**Definition A.35.** Let \( R \) be a commutative ring with unity. An ideal \( I \) of \( R \) is called a *maximal ideal* if there exists an ideal \( J \) in \( R \) such that \( I \subseteq J \), then \( I = J \) or \( J = R \).

**Definition A.36.** A *polynomial ring* \( R[x_0, x_1, \ldots, x_{n-1}] \) in \( n \) indeterminates is a ring consisting of all polynomials in the indeterminates \( x_0, x_1, \ldots, x_n \) with coefficients from the ring \( R \).

**Definition A.37.** Let \( K \) be a field. A *simple extension* \( K(\alpha) \) of \( K \) is the smallest field containing the polynomial ring \( K[\alpha] \). The element \( \alpha \) is called the primitive element of \( K(\alpha) \).

**Theorem A.38.** Let \( R[x] \) be a polynomial ring in the variable \( x \) and \( f \in R[x] \). Let \( I = \langle f \rangle \) be an ideal of \( R[x] \) generated by \( f \). The quotient ring \( R[x]/I \) is a field if \( I \) is a maximal ideal, which is with \( f \) is irreducible.
A.3.1 Finite Fields

**Definition A.39.** Let \( F_q \) be a finite field of \( q = p^n \) elements, where \( p \) is a prime and \( n \in \mathbb{N} \). If \( n = 1 \), then \( F_q = F_p \) a prime field with \( p \) elements.

**Theorem A.40.** A prime field \( F_p \) is isomorphic to the residue class ring \( \mathbb{Z}/(p\mathbb{Z}) \).

**Definition A.41.** Let \( F_q \) be a finite field of \( q = p^n \) elements. Let \( f \in F_q[x] \) be an irreducible polynomial of degree \( n \). The quotient \( F_q[x]/\langle f \rangle \) is a finite extension field \( F_q^n \) of \( q^n \) elements, and is called an extension of \( F_q \) of degree \( n \). We can also write the extension field as \( F_q(\alpha) \), where \( \alpha \in F_q^n \) is a root of \( f \).

**Definition A.42.** Let \( L = F_q^n, K = F_q \). The trace \( \text{Tr}_{L/K} \) is a map defined by

\[
\text{Tr}_{L/K} : F_q^n \to F_q \\
x \mapsto x + x^q + \ldots + x^{q^{n-1}}
\]

**Definition A.43.** Let \( L = F_q^n, K = F_q \). The norm \( N_{L/K} \) is a map defined by

\[
N_{L/K} : F_q^n \to F_q \\
x \mapsto x^{q^{n-1}}
\]

**Definition A.44.** Let \( L = F_q^n, K = F_q \). The dimension of \( L \), considered as a vector space over \( K \), is \( n \). A basis \( B \) of \( L \) over \( K \) is a set of \( n \) elements

\[
B = \{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}, \quad \alpha_i \in L
\]

such that every element \( c \in L \) can be represented as

\[
c = c_0\alpha_0 + c_1\alpha_1 + c_{n-1}\alpha_{n-1}, \quad c_i \in K
\]

**Definition A.45.** Let \( L = F_q^n, K = F_q \). A polynomial basis of \( L \) over \( K \) is a basis of the form \( \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\} \), where \( \alpha \in L \) is called a primitive element.

**Definition A.46.** Let \( L = F_q^n, K = F_q \) and \( \alpha \in L \). A normal basis of \( L \) over \( K \) is a basis of the form \( \{\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}}\} \), where \( \alpha \in L \) is called a normal element.
A.3. COMMUTATIVE ALGEBRA

A.3.2 Gröbner Bases

Definition A.47. Let $R$ be a ring. A total order, linear order or order $\preceq$ on $R$ is a relation that satisfies the following properties.

- **Reflexivity** For all $a \in R$, $a \preceq a$.
- **Transitivity** For all $a, b \in R$, if $a \preceq b$ and $b \preceq c$, then $a \preceq c$
- **Antisymmetry** For all $a, b \in R$, if $a \preceq b$ and $b \preceq a$, then $a = b$
- **Totality** For all $a, b \in R$, either $a \preceq b$ or $b \preceq a$

Definition A.48. Let $\preceq$ be a linear order on a ring $R$. An admissible order $\preceq$ is an order that satisfies the following properties.

- $0 \preceq a$
- If $a \prec b$, then $a + c \prec b + c$

Definition A.49. A monomial $m$ in a polynomial ring $R[x_1, x_2, \ldots, x_n]$ is a power product of the form

$$m = \prod_{i=1}^{n} x_i^{e_i} = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$$

The exponent map of a monomial $m$ is

$$\eta : R[x_1, x_1, \ldots, x_n] \rightarrow (\mathbb{N} \cup \{0\})^n$$

$$x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n} \mapsto (e_1, e_2, \ldots, e_n)$$

The vector $e$ is called the exponent vector of $m$. A monomial $m \in R[x_1, x_2, \ldots, x_n]$ can then be expressed as

$$m = x^{\eta(m)} = x^{(e_1, e_2, \ldots, e_n)} = x^e.$$

Definition A.50. A monomial order $\preceq$ on a polynomial ring $R[x_1, x_2, \ldots, x_n]$ is an admissible order on the set of monomials in $R[x_1, x_2, \ldots, x_n]$. 
Definition A.51. Let $R[x_1, x_2, \ldots, x_n]$ be a polynomial ring with monomial order $\preceq$. Let $f \in R[x_1, x_2, \ldots, x_n]$. We introduce the following functions

$\text{LM}(f) =$ largest monomial of $f$

$\text{LC}(f) =$ coefficient of $\text{LM}(f)$

$\text{LT}(f) = \text{LC}(f) \text{LM}(f)$

Definition A.52. Let $R[x_1, x_2, \ldots, x_n]$ be a polynomial ring with monomial order $\preceq$. Let $f, g \in R[x_1, x_2, \ldots, x_n]$. The $S$-polynomial of $f, g$ is given by

$$S(f, g) = \frac{\text{lcm}(\text{LM}(f), \text{LM}(g))}{\text{LT}(f)} f - \frac{\text{lcm}(\text{LM}(f), \text{LM}(g))}{\text{LT}(g)} g$$

Definition A.53. Let $R[x_1, x_2, \ldots, x_n]$ be a polynomial ring with monomial order $\preceq$. Let $I$ be an ideal of $R[x_1, x_2, \ldots, x_n]$, and $G = \{f_1, f_2, \ldots, f_n\}$ be a set of generators of $I$. $G$ is called a Gröbner basis of $I$ if and only if $S(f_i, f_j) = 0$ for all pairs $f_i \neq f_j$

From the above criterion for Gröbner bases, the Buchberger algorithm was designed to compute a Gröbner basis $G$ of $I$ given a set of generators of $F = \{f_1, f_2, \ldots, f_n\}$ of $I$.

Algorithm 1 Buchberger Algorithm

$G \leftarrow F$

repeat

$H \leftarrow G$

for all $u \in H, v \in H, u \neq v$ do

$S \leftarrow S(u, v)$

$H \leftarrow H \setminus \{S\}$

if $S \neq 0$ then

$G \leftarrow G \cup S$

end if
end for

until $G = H$
\section*{A.4 Ideals and Varieties}

\textbf{Definition A.54.} Let $k$ be a field, and $k[x_1, \ldots, x_n]$ be the polynomial ring of $k$ in $n$ variables. Let $f_1, f_2, \ldots, f_r \in k[x_1, \ldots, x_n]$. The \textit{affine variety} $V(f_1, f_2, \ldots, f_r)$ of $f_1, \ldots, f_r$ is the set of all solutions in the algebraic closure of $k$ to the system

$$
\begin{align*}
    f_1(x_1, \ldots, x_n) &= 0 \\
    f_2(x_1, \ldots, x_n) &= 0 \\
    \vdots \\
    f_r(x_1, \ldots, x_n) &= 0
\end{align*}
$$

\textbf{Theorem A.55.} Let $k$ be a field, and $k[x_1, \ldots, x_n]$ be the polynomial ring of $k$ in $n$ variables. Let $f_1, f_2, \ldots, f_r \in k[x_1, \ldots, x_n]$, and $g_1, g_2, \ldots, g_s \in k[x_1, \ldots, x_n]$. If we have equality of the ideals

$$
\langle f_1, f_2, \ldots, f_r \rangle = \langle g_1, g_2, \ldots, g_s \rangle \quad (A.4.1)
$$

then we have equality of the varieties

$$
V(f_1, f_2, \ldots, f_r) = V(g_1, g_2, \ldots, g_s) \quad (A.4.2)
$$

This means that converting a system of equations

$$
\begin{align*}
    f_1(x_1, \ldots, x_n) &= 0 \\
    f_2(x_1, \ldots, x_n) &= 0 \\
    \vdots \\
    f_r(x_1, \ldots, x_n) &= 0
\end{align*}
$$
into an alternative system

\[ g_1(x_1, \ldots, x_n) = 0 \]
\[ g_2(x_1, \ldots, x_n) = 0 \]
\[ \vdots \]
\[ g_s(x_1, \ldots, x_n) = 0 \]

with the same solution is equivalent to computing a new set of generators \( G = \{g_1, g_2, \ldots, g_s\} \) of the ideal \( I = \langle f_1, f_2, \ldots, f_r \rangle \). For linear systems, this process can be done without the use of ideals by Gaussian elimination. For polynomial systems, the elimination of variables is done by computing \( G \) from \( I \) Gröbner bases techniques. This will be explained in Chapter 2.
Appendix B

Tables

B.1 List of Cyclotomic Fields

Table B.1 shows a list of cyclotomic fields $\mathbb{F}_{q^n}$ for degree $n$ extensions up to $n = 88$. A cyclotomic extension $\mathbb{F}_{q^n} \cong \mathbb{F}_q(\zeta_r)$ of $\mathbb{F}_q$ exists if $q \equiv l \pmod{r}$ and $q$ is prime.

B.2 List of Gaussian Normal Bases

Table B.2 shows a list of Gaussian normal bases for degree $n$ extensions up to $n = 11$. A Gaussian normal bases $\mathcal{N}_{(n,k)}$ of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ exists if $q \not\equiv l \pmod{r}$, where $r = nk + 1$. 

161
| n | 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 3 | 4 | 6 | 8 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 4 | 5 | 6 | 7 | 9 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 6 | 7 | 8 | 9 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 7 | 8 | 9 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 8 | 9 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
| 9 | 10 | 11 | 13 | 15 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |

Table B.1: Cyclotomic Fields for $2 \leq n \leq 88$
Table B.2: Gaussian Normal Bases for $2 \leq n \leq 14$
Bibliography


BIBLIOGRAPHY


