Algebraic Methods in Block Cipher Cryptanalysis

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des Grades
Dr. rer. nat. (rerum naturalium)

von

Dipl.-Inform. Ralf-Philipp Weinmann
geboren in Mannheim

Referenten: Prof. Dr. rer. nat. Dr. h.c. Johannes A. Buchmann
Prof. Dr. ir. Vincent Rijmen

Eingereicht am: 1. März 2008
Verteidigt am: 16. April 2008

Darmstadt, 2008
Hochschulkennziffer: D17

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit - abgesehen von den in ihr ausdrücklich genannten Hilfen - selbständing verfaßt habe.

Wissenschaftlicher Werdegang des Verfassers in Kurzfassung

Oktober 1998 – August 2003	Studium der Informatik mit Nebenfach Mathematik an der Technischen Universität Darmstadt
August 2003	Diplomabschluß an der Technischen Universität Darmstadt (Diplom-Informatiker)
Oktober 2003 – heute	Wissenschaftlicher Mitarbeiter und Doktorand am Fachgebiet Theoretische Informatik, Fachbereich Informatik, Technische Universität Darmstadt

Zusammenfassung

Diese Dissertation ist ein Beitrag zum Gebiet der algebraischen Kryptanalyse. Die folgenden Themen werden in ihr behandelt:

- Wir konstruieren und analysieren sowohl Feistel als auch SLN Chiffren die eine fundierte Konstruktionsstrategie gegen lineare und differentielle Kryptanalyse aufweisen. Der Verschlüsselungsprozess für diese Chiffren kann als ein sehr einfaches System polynomieller Gleichungen beschrieben werden. Für eine Block- und Schlüsselgröße von 128 Bits präsentieren wir Chiffren mit bis zu 12 Runden für die praktische Gröbnerbasisangriffe den gesamten Schlüssel errechnen können; mit einer minimalen Anzahl von Klartext-/ Schlüsseltextpaaren. Wir zeigen wie für eine Untermenge der Chiffren Gröbnerbasen mit vernachlässigbarem Rechenaufwand direkt konstruiert werden können. Diese Vorgehensweise reduziert das Problem der Schlüsselrückgewinnung (Key Recovery) auf das Problem, Gröbnerbasen zwischen zwei verschiedenen Termordnungen zu konvertieren. Für FGLM, einen Algorithmus zum Konvertieren von Gröbnerbasen, können wir obere Schranken für seine Laufzeit sowie seinen Speicherplatzbedarf angeben. Hierdurch sind wir in der Lage zu zeigen, dass es Blockchiffren gibt, die resistent gegen lineare und differentielle Kryptanalyse sind, jedoch mit Gröbnerbasisangriffen angreifbar.

  Eine Einreichung zu diesem Thema wurde in *Proceedings of The Cryptographers' Track at the RSA Conference 2006 (CT-RSA 2006)* veröffentlicht [21].

- Wir zeigen eine effiziente Methode zum Berechnen einer Gröbnerbasis für ein null-dimensionales Ideal welches das Schlüsselrückgewinnungsproblem für den vollen AES-128 ausgehend von einem einzigen Klartext-/Chiffretextpaar beschreibt. Diese Gröbnerbasis ist relativ zu einer graduiert-lexikografischen Ordnung. Wir untersuchen, welche Auswirkungen die Existenz dieser Gröbnerbasis auf die Sicherheit von AES hat.

  Dieses Resultat wurde in *Revised Selected Papers of the Fast Software Encryption Workshop 2006 (FSE 2006)* veröffentlicht [22].

- SMS4 ist eine 128-Bit Blockchiffre, die im WAPI Standard verwendet wird um eine Vetraulichkeit der übermittelten Daten in Funknetzwerken zu erreichen. Für diese Chiffre erklären wir, wie eine Einbettung in einen Erweiterungskörper ähnlich zu BES erreicht werden kann. Weiterhin zeigen wir, dass die Konstruktion der Chiffre fragil ist. Varianten der Chiffre weisen 2^{64} schwache Schlüssel auf.

  Die erzielten Ergebnisse wurden in *Proceedings of Information Security and Privacy, 12th Australasian Conference (ACISP 2007)* publiziert [77].

- Cryptomeria ist eine 64-Bit Blockchiffre mit einem 56-Bit Schlüssel, die in dem CPPM/CPRM Standard für den Schutz von Inhalten auf DVD Audio Medien, Video DVD-Rs sowie SD Karten Verwendung findet. Die Spezifikation dieser Chiffre ist bis auf die verwendete S-Box öffentlich. Die S-Box, die anwendungsspezifisch ist, wird als Geschäftsgeheimnis behandelt und muss von 4C Entity, Inc. lizenziert werden. Wir zeigen wie man für Cryptomeria und ähnlich aufgebaute Chiffren die S-Box durch eine Kombination von differenziellen und algebraischen Methoden zurück gewinnen kann, wenn man den Schlüssel sowie die Eingaben der Chiffre wählen kann. Dieser Angriff wurde gegen rundenreduzierte Varianten von Cryptomeria praktisch verifiziert.

  Diese Ergebnisse sind bisher unpubliziert.

- Wir betrachten Algorithmen zum Berechnen von Gröbnerbasen die auf Methoden aus der linearen Algebrau aufbauen. Da diese Algorithmen extrem speicherhungrig sind haben wir Strategien entwickelt, um die reduzierte Zeilenstufenform einer Matrix effizient auf speicherverteilten Systemen berechnen zu können. Wir geben einen Algorithmus an, der dieses Problem effizient im dichtbesetzten Fall löst und diskutieren den dünnbesetzten Fall.

  Ein Extended Abstract wurde im Tagungsband der *The First International Conference on Symbolic Computation and Cryptography (SCC 2008)* veröffentlicht [112].

Abstract

This thesis is a contribution to the field of algebraic cryptanalysis. Specifically the following topics have been studied:

- We construct and analyze Feistel and SLN ciphers that have a sound design strategy against linear and differential cryptanalysis. The encryption process for these cipher can be described by very simple polynomial equations. For a block and key size of 128 bits, we present ciphers for which practical Gröbner Basis Attacks can recover the full cipher key for up to 12 rounds requiring only a minimal number of plaintext/ciphertext pairs. We show how Gröbner bases for a subset of these ciphers can be constructed with negligible computational effort. This reduces the key–recovery problem to a Gröbner basis conversion problem. By bounding the running time of a Gröbner basis conversion algorithm, FGLM, we demonstrate the existence of block ciphers resistant against differential and linear cryptanalysis but vulnerable against Gröbner basis attacks.

  A paper on this subject has been published in the *Proceedings of The Cryptographers' Track at the RSA Conference 2006 (CT-RSA 2006)* [21].

- We demonstrate an efficient method for computing a Gröbner basis of a zero-dimensional ideal describing the key-recovery problem from a single plaintext/ciphertext pair for the full AES-128. This Gröbner basis is relative to a degree-lexicographical order. We investigate whether the existence of this Gröbner basis has any security implications for the AES.

  This result has been published in the *Revised Selected Papers of the Fast Software Encryption Workshop 2006 (FSE 2006)* [22].

- SMS4 is a 128-bit block cipher used in the WAPI standard for providing data confidentiality in wireless networks. For this cipher we explain how to construct a extension field embedding similar to BES, and demonstrate the fragility of the cipher design by giving variants that exhibit 2^{64} weak keys.

These results have been published in the *Proceedings of Information Security and Privacy, 12th Australasian Conference (ACISP 2007)* [77].

- Cryptomeria is a 64-bit block cipher with a 56-bit key used in the CPRM / CPPM standard for content protection on DVD Audio discs, Video DVD-Rs and SD cards. The design of this cipher is public, the S-Box – which is application-specific – is treated as a trade secret which needs to be licensed from the 4C Entity, Inc. We show how for Cryptomeria and similarly structured ciphers the S-Box can be recovered in a chosen-key setting by a combination of differential and algebraic methods. This attack has been practically validated against reduced round versions of Cryptomeria.

  This is unpublished work.

- We look into Gröbner bases algorithms which use linear algebra methods. Because these algorithms are extremely memory-hungry, we have developed strategies for implementing the reduced row-echelon computation efficiently on distributed memory systems. We give an algorithm to efficiently tackle this problem in the dense case and discuss the sparse case.

  A extended abstract on this subject has been submitted to and accepted at *The First International Conference on Symbolic Computation and Cryptography (SCC 2008)* [112].

Acknowledgements

I am indebted to several people whose support and inspiration has proven invaluable during this dissertation. First of all, I would like to thank my thesis advisor, Professor Johannes Buchmann for the generous support and the excellent environment he provided. Next I would like to thank Professor Vincent Rijmen for becoming the co-referee of my thesis. Thirdly, I would like to express gratitude towards my parents for having enabled and encouraged me to pursue my studies. Last but not least I would like to thank my lovely girlfriend Mirja for her support and understanding during the countless nights I spent instead of the computer instead of with her.

Also, I would like to thank all of my colleagues, collaborators and fellow travellers throughout the years. John Gilmore needs to be explicitly mentioned: He brought the topic of DRM and proprietary encryption algorithms in the context of SD cards at the 23rd Chaos Computer Club Congress; I would probably never have looked into Cryptomeria myself without that remark. Thank you!

Parts of the experiments in this thesis were carried out in MAGMA, which has an excellent implementation of the F_4 algorithm, albeit black-box.

I'm grateful to the people involved in the SAGE project for producing such a wonderful and powerful open-source computer algebra system and hope that parts of the Gröbner basis computation code written during the course of this thesis will soon be in shape for them to include.

A special thanks goes to Franconian brewery Loscher for producing the caffeinated drink "Club-Mate" which was heavily perused while writing this document. This thesis was produced using LaTeXand written in Emacs.

Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

AES	Advanced Encryption Standard
CPPM	Content Protection for Pre-Recorded Media
CPRM	Content Protection for Recordable Media
DES	Data Encryption Standard
DMS	Distributed Memory System
FIPS	Federal Information Processing Standard
GE	Gaussian Elimination
GJE	Gauss-Jordan Elimination
GUFN	Generalized Unbalanced Feistel Network
HPS	High-Performance Switch
MPI	Message-Passing Interface
NBS	National Bureau of Standards
NIST	National Institute of Standards
NSA	National Security Agency
PRAM	Parallel Random Access Machine
REF	Row Echelon Form
RREF	Row-Reduced Echelon Form
SLN	Substitution Linear Network
SNI	Switch Network Interface
SPN	Substitution Permutation Network
UFN	Unbalanced Feistel Network

Chapter 0

Introduction

Encryption has become ubiquitous. WPA2 is used to encrypt wirelessly transmitted IEEE 802.11 packets. IPsec encrypts IP packets. SSL provides data confidentiality for TCP connections. S/MIME and PGP are used for encrypted email. Data at rest – stored on media of various form factors – is routinely encrypted both file and block-wise. All of the above technologies are in widespread use today and share a common theme: they use or allow for the use of a block cipher for encrypting the bulk data. These ciphers split their input into a sequence of blocks of the same size. The blocks are then individually encrypted using a parametrized invertible function, the encryption function of the block cipher. The parameter of this function is the so-called cipher key. This function can be represented algebraically: as one or more equations in which the key and the plaintext take the role of the unknowns. Claude Shannon was the first to propose this view as a model to assess a cipher's security [104]:

> *Thus, if we could show that solving a certain system requires at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type, then we would have a lower bound of sorts for the work characteristic.*

Cryptanalysis is the study of cryptographic systems with the intention of breaking them. A cryptanalytic attack is an attack that breaks a cipher faster than brute-force, i.e. enumerating all possible keys. Indeed, before Shannon published the paper from which the above quote was taken, Marian Rejewski sucessfully used algebraic methods against the Enigma [53] to cryptanalyse it. The equations in his analysis describe permutations. This work only saw the light of day much later.

The Data Encryption Standard or rather its publication marks a critical point in the history of modern cryptology. For the first time it allowed

the scientific community to see the full design of a government approved encryption algorithm. In 1975, the National Bureau of Standards (NBS) proposed the block cipher Data Encryption Standard (DES), which was designed by IBM with input from the National Security Agency (NSA). It was approved as federal standard in the United States of America in late 1976 and published in 1977 [88]. Previously, encryption algorithms were only used in military settings and hence not disclosed.

Statistical cryptanalytic attacks are standard attacks against modern block ciphers. The best known attacks against the DES – linear and differential cryptanalysis, which were only discovered in the early 1990s – are of this type and require a large number of plaintext/ciphertext pairs. As the cryptographic community began understanding the power of differential and linear cryptanalysis they found criteria that allowed for the construction of block ciphers immune to these attacks. As general purpose computers became cheaper, a further paradigm change resulted from shifting the focus from hardware implementations to software implementations. In software, bit permutations – used extensively in the DES – often are extremely costly, as most off-the-shelf CPUs do not provide for instructions for this purpose. However, byte and word-sized operations are cheap.

The Advanced Encryption Standard reflects these changes in block cipher design. In 1997, 20 years after the publication of DES, the National Institute of Standards (NIST) – the successor of the NBS – issued a call for a successor, the Advanced Encryption Standard (AES). This was a tremendous opportunity for the academic cryptologic community. As it was an open selection process, fierce competition resulted in significant advances being made in both block cipher design and cryptanalysis. Although the set of functional, security and efficiency requirements for this block cipher were fixed by the NIST, it was the cryptographic community that evaluated the performance and security of the submitted ciphers in a three-round process. It is noteworthy that this contest was not limited to U.S. citizens. The winner of this competition was Rijndael [38], an entry by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. Not only in the U.S., but world-wide the AES has since become the new standard block cipher.

Algebraic approaches to secret-key analysis did not receive much attention in the world of open cryptologic research until the end of the 20th century. Although there were some initial attempts of attacking the DES through a polynomial bit-level representation [60], these were not successful and soon discarded. In the 1990s, block cipher designers began proposing algebraically structured components [90, 91, 10] in order to make their ciphers resistant against linear and differential cryptanalysis; it was a logical

next step to look into exploiting that algebraic structure. The first algebraic attack in this context was Jakobsen and Knudsen's interpolation attack [64] against SHARK [100], a predecessor of Rijndael. Later, after Rijndael [38] had been chosen as the Advanced Encryption Standard, suspicion because of its high algebraic structure arose [50]. In 2002, Courtois and Pieprzyk published a then highly controversial paper on attacking block ciphers by solving systems of polynomials equations using a specialized method which they called XSL (Extended Sparse Linearization) [32]. This method as well as their complexity analysis was later debunked by Cid and Leurent [24]. Although XSL has died, the idea of attacking block ciphers using algebraic methods has stayed alive.

Stream ciphers were easier prey for algebraic attacks than block ciphers. The designs attacked were regularly clocked LFSR-based ciphers such as combiners and filter generators [2, 45]. The structure of the equation systems for these stream ciphers is inherently different from the structure encountered in the case of block ciphers: Independent of the number of clocks, the linear evolution of the internal state causes the output bits to be expressible as a polynomial of fixed degree in the variables representing the bits of the initial internal state. This is different from the case of block ciphers where each round adds a layer of non-linearity to the system of equations, requiring either to introduce variables for representing the intermediate state after a round transformation or obtaining a system in which the degree grows with the number of rounds of the cipher.

Multivariate systems of polynomial equations not only occur in secret-key cryptography. There are also public-key encryption and signature schemes built on the difficulty of solving these systems [94, 71]. Most of these systems have been broken however [47, 52].

Several methods can be used for solving. This thesis will only explore Gröbner basis algorithms as they have a solid mathematical foundation. The XL [31] algorithm sometimes is proposed as an alternative to Gröbner basis algorithms. It was shown however that XL is merely a Gröbner basis algorithm in disguise, and a very inefficient one at that [3].

Other methods include resultant-based methods, SAT solvers and an algorithm proposed by Raddum and Semaev [98]. Whether SAT solvers should be considered algebraic solvers is a philosophical question. The DPLL algorithm, the backbone of SAT-solvers, is a backtracking-based procedure for deciding the satisfiability of formulae in propositional logic. Therefore, this approach in the past has been called "logical cryptanalysis". SAT solvers were recently revisited by Mironov and Zhang for the case of hash functions [86]. Related to this is the work of Courtois, Bard and Wagner [33], who cryptanalysed the block cipher KeeLoq, an NLFSR-based block cipher

used in the automotive industry with a 32-bit state and 64-bit keys. Using a combination of slide attacks and SAT solvers, they were able to successfully attack this cipher.

Outline of this thesis

The thesis is organized as follows:

- Chapter 1 introduces the mathematical and cryptologic basics needed for this thesis. Gröbner bases, block ciphers and attack models are explained. Buchberger's algorithm as well as the Buchberger criteria are presented.
- Chapter 2 deals with the construction and statistical cryptanalysis of block ciphers. We give an overview of the construction principles of iterated block ciphers and the briefly survey a number of statistical attacks that modern block ciphers have been made immune against. We then describe a selection of deployed ciphers that are of further interest for the thesis as well as experimental ciphers that are suitable for experimentation with algebraic attacks. For one of the deployed ciphers presented, we argue that its design is fragile by showing weak keys in a variant.
- Chapter 3 contains algorithmic and textual descriptions of efficient Gröbner basis algorithms which make use of linear algebra routines for polynomial reduction. These algorithms have been implemented by the author.
- Chapter 4 demonstrates how algebraic approaches can be used in the cryptanalysis of block ciphers. Interpolation attacks are briefly described. For FLURRY and CURRY we give explicit polynomial representations. We demonstrate that the bit-level operations of the cipher SMS4 can be embedded into a larger field such that the whole cipher can be represented in a structurally "clean" way. Furthermore we explain the general approach of Gröbner basis attacks and show that in certain instances Gröbner bases can be constructed for ciphers without performing a single polynomial reduction. For a number of FLURRY and CURRY instances we give experimental results for Gröbner basis attacks requiring a single plaintext/ciphertext pair. For other cipher instances with a larger number of rounds we are able to give upper bounds on the complexity of a Gröbner basis attack. For AES-128 we present an explicit and practical way to construct a zero-dimensional Gröbner basis. The impact of this result is then analysed. We show that algebraic attacks can be fruitful against block ciphers with unknown S-Boxes if we allow for a chosen-key, chosen-text model. This is demonstrated by attacks against reduced-round variants of Cryptomeria, a block cipher with a secret S-Box used in digital restriction management schemes.
- Chapter 5 presents a parallelized version of the Gauss-Jordan elimination to tackle the computation of reduced row-echelon forms on

distributed memory systems efficiently. This algorithm requires only unidirectional communication from the master node to the slave nodes and has low communication overhead. Furthermore, inter-node latency becomes almost irrelevant, allowing the computation to be carried out on loosely coupled networks where the machines are physically distributed.

- Chapter 6 concludes this thesis and give an overview of open research problems.

Chapter 1

Preliminaries

In this chapter we lay the groundwork that is needed to understand the rest of the thesis: First we will clarify some notational issues. We then will state a number of basic facts about polynomial rings, explain the concept of Gröbner bases and give a basic algorithm to compute them. The material we present on this subject is narrowly focused on our needs, we kindly refer reader yearning for a more thorough introduction to [7] and [34]. In almost all cases proofs will be omitted.

Secondly we give an abstract description of the concept of block ciphers and define attack models against this class of secret-key primitives.

We close this chapter with a section on the reduction of 3SAT to the solving of multivariate quadratic of equations over $GF(2)$.

1.1 Notation for Data Structures

In the course of this thesis, at times we will need data structures such as lists and tuples with well-defined operations on them. In the following we will define the list data structure:

A list L is an ordered sequence of elements of a type T; specifically, all elements of L must be of the same type. An empty list is denoted by the constant value $()$. The following operations are defined on lists:

cons(m, L)	prepends an element $m \in T$ to the list L
cons(L, m)	appends an element $m \in T$ to the list L
head(L)	returns the first element of the list L or $()$ if the list is empty.
tail(L)	returns a lists that consists of all elements of L except for the first element.

We assume the existence of a fast method for deciding whether an element exists in the list or not. This is usually not given for a typical list

structure but can be obtained by emulating a list through a binary tree. Closely related to lists, alas with a more rigidly typed signature are n-tuples. For a tuple $t = (t_1, t_2)$ we define the functions $\mathsf{first}(t)$ and $\mathsf{second}(t)$ to return t_1 and t_2 respectively.

1.2 Polynomial Rings and Ideals

The only fields we are interested in are finite fields; some results and algorithms presented here may not be applicable for fields with characteristic 0. Let $\mathbb{F}$ be a finite field, $\mathcal{X} := \{x_1, \ldots, x_n\}$ a set of variables and $R := \mathbb{F}[\mathcal{X}]$ a polynomial ring in these variables. Elements of the ring R then are called polynomials, products of the form $t = \prod_{i=1}^{n} x_i^{e_i}$ with $e_i \in \mathbb{N}_0$ for $1 \leq i \neq n$ are called terms, and for $c \in \mathbb{F},\ c \neq 0$, products of the form $c \cdot t$ are called monomials.

The total degree of a term $t \in R$ shall be denoted by $\deg(t)$, the set of all terms of the ring R by $\mathcal{T}(R)$ and the set of terms that have a total degree $\leq d$ by $\mathcal{T}_d(R)$. For a polynomial $p \in R$, the support $\mathrm{supp}(p)$ is defined to be the set of all terms occurring in p with non-zero coefficient. Analogously for a term t, its support $\mathrm{supp}(t)$ is defined to be the set of all variables occuring in t that have a non-zero exponent.

Definition 1.2.1. Let $p \in \mathbb{F}[\mathcal{X}]$ be $p = \sum_{i=0}^{k} c_i t_i$ with $c_i \in \mathbb{F}, t_i \in \mathcal{T}(R)$. We then call the polynomial homogeneous if there exists $d \in \mathbb{N}$ such that $\deg(t_i) = d$ for all $1 \leq i \leq k$. A polynomial is called inhomogeneous if it is not homogeneous.

One of the main objects in the course of this thesis will be polynomial ideals and their representations. An ideal is defined as follows:

Definition 1.2.2. An ideal $I \subset R$ is a set of elements that forms an additive group and has the additional property of being closed under multiplication with elements of R. This means that for every $x \in R, y \in I$, both $xy \in I$ and $yx \in I$.

Emmy Noether first discovered the following finiteness property of polynomial rings:

Definition 1.2.3. For a ring R, an ascending chain of ideals $I_1 \subset \ldots \subset I_2 \ldots \subset R$ is said to become stationary if there exists an n such that $I_n = I_m$ for all $m > n$. If every ascending chain of ideals becomes stationary, the ring R is called Noetherian.

Since polynomial rings are Noetherian rings, Hilbert's basis theorem holds.

Theorem 1.2.1 (Hilbert's basis theorem)**.** *Every ideal $I \in \mathbb{F}[\mathcal{X}]$ is finitely generated, i.e $\exists g_1, \ldots, g_m \in \mathbb{F}[\mathcal{X}]$ such that $I = \langle g_1, \ldots, g_m \rangle$.*

1.2.1 Affine Varieties

Generally, algebraic varieties are defined over algebraically closed fields.

Definition 1.2.4. An algebraic variety $V \in \overline{\mathbb{F}}^n$ is a set of points fulfilling a set of polynomial equations:

$$\begin{aligned} p_1 &= 0 \\ \vdots &\quad \vdots \\ p_m &= 0 \end{aligned}$$

i.e. if we regard each of the polynomials in $P := \{p_1, \ldots, p_m\}$ as an n-valued function, the result of evaluating any $p \in P$ at any point $v \in V$ results in 0 for all of $p \in P$.

We're now in a position to relate the ideals to their corresponding varieties. This can be done by employing Hilbert's Nullstellensatz:

Theorem 1.2.2 (Hilbert's Nullstellensatz)**.** *Let $R = \overline{\mathbb{F}}[x_1, \ldots, x_n]$ and I and ideal of R such that $I \neq R$ and $I \neq 0$. Then there exists an $x \in \mathbb{F}^n$ such that*

$$f(x_1, \ldots, x_n) = 0$$

for all $f \in I$.

The Hilbert Nullstellensatz of course also gives us a relation between polynomial systems of equations and their corresponding ideals in polynomial rings.

Due to the topic we are dealing with – cryptanalysis – we are not very much interested in points at infinity of algebraic varieties. We therefore introduce the concept of so-called *field polynomials* which are used to restrict the points in the algebraic varieties and hence the solutions we are dealing with to points, respectively solutions such that each component respectively variable is an element of the field.

Definition 1.2.5. Let X be a set of variables, $\mathbb{F}_q$ be a finite field of order q and $\mathbb{F}_q[\mathcal{X}]$ a polynomial ring. Then polynomials contained in the set $\mathcal{F}(\mathbb{F}_q[\mathcal{X}]) = \{x^q + q \mid x \in X\}$ are called *field polynomials*. Equations of the form $x^q = x$ or $x^q - x = 0$ with $x \in X$ are called *field equations*.

It then makes sense to compute in the quotient ring of the polynomial ring factored by the set of field polynomials. For $q = 2$, this ring is called the ring of Boolean functions.

Theorem 1.2.3. *The quotient ring $F_q[X]/\langle \mathcal{F}(F_q[X]) \rangle$ contains only a finite number of elements.*

This should be seen in contrast to the polynomial ring, which contains an infinite number of elements.

1.3 Gröbner Bases

Gröbner bases are standard bases of polynomial ideals. In order to precisely define the notion of a Gröbner basis we first need to introduce some concepts, namely term orders, normal forms and polynomial reduction in the multivariate case.

1.3.1 Term Orders

Contrary to the univariate case where we can simply order all terms by their degree, an answer to the question of how to order terms in the multivariate case is not immediately obvious. In fact, there are multiple ways to order the terms of a multivariate polynomial ring]. To make these options precise, we need to define the notion of a term order.

Definition 1.3.1. A linear order (also called total order) is a binary relation $\leq$ on a set X such that the following holds:

- if $(a \leq b) \wedge (b \leq a)$ then $a = b$ (anti-symmetry)
- if $(a \leq b) \wedge (b \leq c)$ then $a \leq c$ (transitivity)
- $\forall a, b \in X : \ (a \leq b) \vee (b \leq a)$ (totality)

Definition 1.3.2. A term order $\leq$ is a linear order on the set of terms $\mathcal{T}(R)$ such that

1. $1 \leq t$ for all terms $t \in \mathcal{T}(R)$
2. for all terms $s, t_1, t_2 \in \mathcal{T}(R)$ whenever $t_1 \leq t_2$ then $st_1 \leq st_2$

The following example shows some of the most commonly used term orders:

Example 1.3.1. Let $\varepsilon : \mathcal{T}(\mathbb{F}[\mathcal{X}]) \to \mathbb{N}^n$ be a map that takes a term $t = \prod_{i=1}^{n} x_i^{e_i}$ to its exponent tuple $e = (e_1, \ldots, e_n)$. We can now define term orders by showing how terms $t_1, t_2 \in \mathcal{T}(\mathbb{F}[\mathcal{X}])$ can be compared by comparing their corresponding exponent tuples $e := \varepsilon(t_1)$ and $d := \varepsilon(t_2)$.

The *lexicographical order* (lex): $t_1 \leq_{\text{lex}} t_2$ iff

$$(d_1, \ldots, d_n) = (e_1, \ldots, e_n)$$

or if there exists $1 \leq i \leq n$ with $d_j = e_j$ for $1 \leq j \leq (i-1)$ and $d_i < e_i$.

The *reverse lexicographical order* (revlex): $t_1 \leq_{\text{revlex}} t_2$ iff

$$(d_1, \ldots, d_n) = (e_1, \ldots, e_n)$$

or if there exists $1 \leq i \leq n$ with $d_j = e_j$ for $i+1 \leq j \leq n$ and $d_i < e_i$.

The *graded lexicographical order* (deglex): $t_1 \leq_{\text{glex}} t_2$ iff

$$\deg(t_1) < \deg(t_2)$$

or if $\deg(t_1) = \deg(t_2)$ and $t_1 \leq_{\text{lex}} t_2$.

The *graded reverse lexicographical order* (degrevlex) $t_1 \leq_{\text{grevlex}} t_2$ iff

$$\deg(t_1) < \deg(t_2)$$

or if $\deg(t_1) = \deg(t_2)$ and $t_1 \leq_{\text{revlex}} t_2$.

Once a term order has been fixed, we define HT(f) to be the greatest term occuring in the polynomial $f \in R$ according to this order – this is the so-called *head term*. Other names used in the literature are *initial term* or *leading term*. Correspondingly HM(f) is the head monomial, i.e. the head term of f multiplied with the matching coefficient which is denoted by HC(f), the head coefficient. Please note that there can be some confusion arising from the fact that parts of the literature use the notions *term* and *monomial* in a fashion that is interchanged from our definitions (see [34] for an example).

Definition 1.3.3 (Syzygy polynomial)**.** Let $f, g \in \mathbb{F}[\mathcal{X}]$. The syzygy polynomial of f and g is defined as

$$\text{spol}(f,g) = \frac{\text{lcm}(\text{HM}(f), \text{HM}(g))}{\text{HM}(f)} f - \frac{\text{lcm}(\text{HM}(f), \text{HM}(g))}{\text{HM}(g)} g$$

Proposition 1.3.1. *A subset of polynomials $G \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$ with $G \neq 0$ is a Gröbner basis if $\langle HT(g) : g \in G\rangle = \langle G\rangle$*

A zero-dimensional ideal is an ideal that has a finite number of solutions over the closure of the field. It is advantageous to have this property for Gröbner basis computations, because usually Gröbner bases for these cases can be computed faster. Using Corollary 6.56 of [7] we can determine whether an ideal I is zero-dimensional. Below we state a reduced version of this corollary:

Lemma 1.3.2. *Let I be a proper ideal of $F[x_1, \ldots, x_n]$. Then the following assertions are equivalent:*

- $\dim(I) = 0$
- *There exists a term order $\leq$ such that for each $1 \leq i \leq n$ there is $g_i \in I$ with $HT(g_i) = x_i^{\nu_i}$ for some $0 \leq \nu_i \in \mathbb{N}$.*

Definition 1.3.4. A polynomial $f \in R$ is in normal form respective to G (or reduced by G) if there exists no $g \in G$ with $HT(g) | \mathcal{T}(f)$. We say that Gröbner basis G is reduced if all its elements are in normal form and the head coefficient of each element is 1.

Algorithm 1 can be used for computing normal forms of polynomials respective to a set of polynomials. Please note that normal forms are not canonical. In the univariate setting, this algorithm reduces to the computation of a GCD of the polynomials; in the multivariate setting the algorithm degrades to Gaussian elimination if the total degree of all polynomials is < 2.

Algorithm 1 NormalForm

Input: $P = (p_1, \ldots, p_m) \subset F[x_1, \ldots, x_n], \quad f \in F[x_1, \ldots, x_n]$
Output: g – a normal form of f respective to P

$g \leftarrow f$
while IsReducible(g, P) **do**
  select $p \in P$ such that g is reducible modulo p.
  determine a monomial m with $g \rightarrow_p g - mp$
  $g \leftarrow g - mp$
end while

1.3.2 Buchberger's Algorithm

In his Ph.D. thesis [19], Bruno Buchberger proposed the following algorithm for computing the standard basis of an ideal and hence for solving the ideal membership problem. The function SelectPair used in this algorithm can

Algorithm 2 Buchberger

Require: $(p_1, \ldots, p_m) \subset F[x_1, \ldots, x_n]$

$G \leftarrow \{p_1, \ldots, p_m\}$
$B \leftarrow \{\{g_1, g_2\} \mid g_1, g_2 \in G \text{ with } g_1 \neq g_2\}$
while $B \neq \emptyset$ **do**
  $\{g_1, g_2\} \leftarrow$ SelectPair(B)
  $B \leftarrow B \setminus \{g_1, g_2\}$
  $h \leftarrow$ spol(g_1, g_2)
  $h_0 \leftarrow$ NormalForm(h, G)
  if $h_0 \neq 0$ **then**
    {add critical pair to B}
    $B \leftarrow B \cup \{\{g, h_0\} \mid g \in G\}$
    $G \leftarrow G \cup \{h_0\}$
  end if
end while

be any function that selects an element from a set of unordered polynomial tuples and returns them. How to select these pairs is what is referred to as the "strategy" of the Gröbner basis algorithm. Different strategies exist, for the one most commonly used see below (Section 1.3.3).

Because of the properties of the normal form computation, this algorithm degenerates to Gaussian Elimination if we input a set of linear equations and to the computation of a greatest common divisor of the polynomials if the polynomials $p_1, \ldots, p_m$ are univariate.

1.3.3 Buchberger's Criteria

During the computation of a Gröbner basis many "reductions to zero" can be observed when computing normal forms. These computations do not contribute anything towards finding the Gröbner basis and henceforth are considered useless. By reducing the amount of reductions to zero, the computation of the Gröbner basis can be significantly sped up. In 1979, Buchberger [20] presented two criteria that can be used for exactly that purpose, the so-called first and second Buchberger criterion.

Proposition 1.3.3 (Buchberger's 1st criterion)**.** *Let $f, g \in \mathbb{F}[\mathcal{X}]$ with*

$$\gcd(\mathit{HT}(f), \mathit{HT}(g)) = 1.$$

Then

$$\mathit{spol}(f, g) \xrightarrow[\{f,g\}]{*} 0.$$

The first Buchberger criterion tells us that we can discard all pairs whose head terms are pairwise prime in the Buchberger algorithm. To state the second Buchberger criterion, we need to introduce a rather technical concept, namely the concept of t-representations:

Definition 1.3.5. Let $t \in \mathcal{T}(\mathbb{F}[\mathcal{X}]), f \in \mathbb{F}[\mathcal{X}]$ with $f \neq 0$ and $P \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$. If f can be represented as follows

$$f = \sum_{i=1}^{k} m_i p_i \tag{1.1}$$

with non-zero monomials $p_i \in P$ not necessarily being pairwise different for $1 \leq i \leq k$ and

$$\max\{\text{HT}(m_i p_i \mid 1 \leq i \leq k\} \leq t$$

we say that the right-hand side of equation 1.1 is a t-representation of f w.r.t. P.

Proposition 1.3.4 (Buchberger's 2nd criterion)**.** *Let $F \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$ and $g_1, g_2, p \in \mathbb{F}[\mathcal{X}]$. Furthermore, assume both of the following conditions hold:*

- *HT*$(p) \mid$ *lcm*$(HT(g_1), HT(g_2))$ *and*
- *spol*(g_i, p) *has a* t_i*-representation w.r.t.* F *with*

$$t_i < \textit{lcm}(HT(g_i), HT(p))$$

for $i \in \{1, 2\}$.

Then it follows that there exists a $t <$ *lcm*$(HT(g_1), HT(g_2))$ *such that spol*(g_1, g_2) *has a* t*-representation w.r.t.* F.

Making use of both of Buchberger's criteria allows to skip so-called "reductions to zero" in the Buchberger algorithm. Avoiding these useless reduction steps does not change the theoretical complexity but results in a tremendous speed-up of the algorithm from a practical side.

If we make use of the Buchberger criteria, it is advantageous to use the so-called "normal strategy": This means that SelectPair always selects a pair $\{f, g\}$ such that lcm$(\text{HT}(f), \text{HT}(g))$ is minimal under the chosen term order.

1.3.4 Macaulay Matrices

In the early 20th century, Francis Sowerby Macaulay wrote a groundbreaking paper that had significant impact on the field of algebraic geometry [79]. Among other things, he describes a novel method to check whether a polynomial system of more than two equations can be solved using resultants. This method was later refined in a book of his [80].

The matrices occuring in this technique establish a link between polynomial systems and linear systems. They were later called *Macaulay matrices* in his honor[1]. The homogenization of the polynomial system that is described in the paper is not required for these type of matrices – and will not be required by us, when we speak of Macaulay matrices. The technique of forming the Macaulay matrix of polynomial system is called *linearization*.

Let $P = \{p_1, \ldots, p_m\} \subset F[x_1, \ldots, x_n]$ be a set of polynomials. To form a Macaulay matrix of P we first collect all terms $\mathcal{T}(P) = \bigcup_{p \in P} \text{supp}(p)$ occuring in the polynomials of P and order them using a term order. Let m be the total number of terms occurring. The Macaulay matrix is an $n \times m$ matrix over the field F. Each line of the Macaulay matrix represents a polynomial. For this line each entry is set to the coefficient of the term corresponding to the column in question. Entries of the line in a column position corresponding to a term not occuring in the polynomial are set to 0.

Example 1.3.2. Let $P := \{x_1x_2 + x_3, 2x_2 + 1, x_1x_3 + 2\} \in \mathbb{F}_3[x_1, \ldots, x_3]$. Then $\mathcal{T}(P) = \{1, x_2, x_3, x_1x_2, x_1x_3\}$. The corresponding Macaulay matrix

[1] Sometimes they are simply called the coefficient matrix of the polynomial system.

reads as follows:

$$\begin{pmatrix} x_1x_3 & x_1x_2 & x_2 & x_3 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 & 2 \end{pmatrix}$$

1.4 Block Ciphers

Definition 1.4.1. A block cipher is a family of functions $E_k : \mathcal{P} \to \mathcal{C}$, each of which maps an element of the plaintext space $\mathcal{P}$ to an element of the ciphertext space $\mathcal{C}$. The elements $k \in \mathcal{K}$ are the keys of the block cipher. For each function E_k a corresponding function $\mathcal{D}_k$ exists such that $\mathcal{D}_k(\mathcal{E}_k(x)) = x$ for all $x \in \mathcal{P}$. This corresponding function is called decryption function and must be efficiently computable given the key k.

Remark. For most block ciphers in symmetric-key cryptography $\mathcal{P} = \mathcal{C} = GF(2)^n$, with n being a multiple of 8. Of course, public-key encryption schemes such as RSA can also be modelled as block ciphers. This however is seldomly done.

For a more detailed description of how modern block ciphers are built, please see Chapter 2.

1.4.1 Diffusion and Confusion

In [104], Shannon describes an "algebra of secrecy systems". Specifically, he comes up with the concept of "product ciphers", namely ciphers are composed of different components.

In the same paper, Shannon introduces the notions of *confusion* and *diffusion*. These are not rigorously defined: Confusion is meant to make it difficult for an eavesdropper to come up with a simple relation between the statistical properties of the intercepted messages and the actual keys that were used. Diffusion on the other hand is to "dissipate" the statistical structure of the original message into "long range statistics", meaning that the locality of statistical properties of the messages is removed.

Modern block ciphers all build on the concept of alternating multiple layers of confusion and diffusion operations, which will be described in Section 2.1.1 in more detail.

1.4.2 Attack Models

Auguste Kerckhoffs[2] stated six principles that (military) ciphers should fulfill in his article *La cryptographie militaire* [70]; the second of these principles

[2]Scanned and OCRed versions of Kerckhoffs' seminal articles have been made available by Fabien A.P. Petitcolas: `http://www.petitcolas.net/fabien/kerckhoffs`

has become known as Kerckhoffs' principle:

> *Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi.*

Claude Shannon later reformulated the above as "the enemy knows the system"; in this form it is known as Shannon's maxim.

Taking this as a given, namely that the design of the cipher is publicly known, however the key is not, the most obvious way to recover the key is to search through the whole key space. This is known as brute force. The worst case time complexity for an attack by brute force obviously is $\#\mathcal{K}$ encryption operations, the expected time complexity is $\#\mathcal{K}/2$ encryption operations.

The following attack scenarios against ciphers are known in the literature:

ciphertext only attacks Ciphertext-only attacks can be considered to be the most powerful attacks against any cipher. The attacker is able to deduce (parts of) the key merely by knowing a certain amount of ciphertext encrypted under the given key.

known plaintext attacks Known plaintext attacks are scenarios in which the attacker has access to a number of plaintexts toghether with their corresponding ciphertexts.

chosen text attacks Chosen text attacks assume the attacker to have a higher level of access. Namely, it is assumed that the attacker has one-time access to an oracle and may submit a batch of either chosen plaintext or chosen ciphertexts which the oracle then encrypts respectively decrypts. The attacker has access to the output of the oracle for the submitted values after all values have been submitted.

adaptive chosen text attacks This attack is similar to the chosen text attack but instead of having one time access to the oracle the adversary may interactively change his queries to the oracle during the attack depending on the responses received. Sometimes it is assumed that the attacker only has access to an encryption oracle, sometimes to the decryption oracle at other times he has access to both.

The above attacks are carried out with the aim of key recovery. Sometimes, the aim of an attacker may be different however:

chosen key attacks This scenario is relevant in cases where Kerckhoffs' principle is violated, for example if components of a cipher such as S-Boxes are kept secret (see for instance [93], [1]). The attacker has full access to an encryption and/or decryption oracle that he can key. Chosen key attacks need not be adaptive chosen text attacks but can be combined with them.

distinguishing attacks Distinguishing attacks can be seen as the weakest class of attacks. A distinguishing attack merely allows an attacker to distinguish ciphertext output of a cipher A either from a (pseudo) random sequence of data or from the ciphertext output of another B with a probability > 0.5.

Please note that these two scenarios are completely orthogonal to the previous types of attacks listed.

In a cases where Kerckhoffs' principle does not apply, it can also be considered valid for an attacker to use chosen-key, chosen-text attacks to discover the structure or the inner workings of a block cipher. In this case it is assumed that the adversary has full access to an implementation of the cipher that he may freely key. The first chosen-key attack in the literature is an against the GOST 28147-89 block cipher [93], a Russian design developed in the 1970s with domain-specific, secret S-Boxes. For this cipher Markku-Juhani Saarinen proposed a chosen-key attack that recovers the S-Boxes [102]. Certain ciphers such as the NSA's Type-1 ciphers BATON and JUNIPER prevent chosen-key scenarios by requiring the key to be provided together with a 160-bit checksum [101].

In Section 2.3.3 of this thesis we will look at Cryptomeria [1], a block cipher used for content protection of data on Video DVD-R's, DVD Audios and SD cards, for which the cipher specification itself is public, the S-Boxes of the cipher however are considered trade secrets which need to be licensed.

The distinguishing model oftentimes is used for reduced-round ciphers to guess the keys of the last round. A distinguishing attack on $(r-1)$ rounds of a block cipher – which distinguishes the output of those $(r-1)$ rounds from a pseudo-random permutation – can often be exploited and turned into a key-recovery attack on the same block cipher with r rounds. Please note however, that there also exists another interesting use case for distinguishing attacks: assume the attacker has access to an oracle – i.e. an encryption or decryption device – with a fixed key, unknown to him, for a limited time. A distinguishing attack for this cipher will then allow him to determine whether this oracle uses cipher $\mathcal{A}$ or not.

The models we have discussed thus far treat the cipher as an mathematically idealized building block with fixed inputs and outputs, namely the plaintext, the key and the ciphertext. In practice however, an adversary of course does not have to comply with this idealized model but may choose to attack the implementation instead. This means he is able to observe or even control more aspects of the execution of the actual ciphering algorithm. This gives rise to so-called side-channel attacks [73] and fault attacks [17]. Although these classes of attacks are interesting in themselves, they will not further been discussed in this thesis as they lie outside of our attack model.

Weak Keys

For some block ciphers, such as DES [88] and IDEA [74] and Blowfish, so-called weak keys have been found [109, 14]. Using a cipher with these keys will cause the cipher to have cryptographic weaknesses. For example: For E being the DES encryption function, a weak K causes the encryption to be self-inverting, i.e. $E_K(E_K(M)) = M$. For DES, IDEA and Blowfish, the weak keys found represent only a small fraction of the key space. IDEA's weak keys cause the cipher to be susceptible to linear cryptanalysis (see 2.2.2), Blowfish's weak keys generate S-Boxes that make allow a version reduced to 14 rounds to be distinguished from a random permutation. Ciphers that do not exhibit weak keys clearly are preferable, we call their key space "flat".

1.5 The MQ-Problem

We know that linear systems of equations over finite fields can be solved in polynomial time – Gaussian elimination for example solves them in $O(n^3)$ with n being the number of variables. Given a random set of n quadratic equations in n variables over a finite field $\mathbb{F}_q$, we pose ourselves the following question: How difficult is this problem to solve asymptotically? Can this problem be solved in time polynomial in the number of variables?

To answer this question, we first have to perform a brief excursion into complexity theory: A decision problem is a problem to which the answer is a simple "yes" or "no". Decision problems are the basis of the theory of NP-completeness, the foundations for which were laid by Stephen Cook in a 1971 paper called "The Complexity of Theorem Proving procedures" [27]. To understand the notion of NP-completeness, we first have to explain what NP stands for:

Definition 1.5.1 (non-deterministic polynomial time (NP))**.** Let P be a decision problem of size n. Given an input C, called "certificate" C, an algorithm can check whether or not the answer to P is correct in time polynomial in n.

The class of NP-complete problems informally speaking is a class of problems for which no polynomial time algorithm is known for solving them. On the other hand, although there is no proof, it is widely believed that these algorithms cannot be solved by polynomial time algorithms. More importantly by using the concept of polynomial-time reducibility, each of the problems in the class of NP-complete problems can be used to solve another arbitrary NP-complete problem:

Definition 1.5.2 (polynomial-time reducible)**.** Let P_1, P_2 be decision problems and Ω an oracle solving P_1. We call P_2 polynomial-time reducible to

P_1 if a polynomial-time algorithm for solving P_2 using Ω exists. This means that the number of calls to Ω is polynomially bounded.

More formally we state the MQ problem as follows:

Problem 1.5.1. Given a set of polynomials $p_1, \ldots, p_n \in \mathbb{F}[\mathcal{X}]$ with $\deg(p_i) = 2$ for all $1 \leq i \leq n$. Decide whether there is an assignment of variables $a_1, \ldots, a_n \in \mathbb{F}$ such that

$$\begin{aligned} p_1(a_1, \ldots, a_n) &= 0 \\ \vdots &\quad \vdots \\ p_1(a_1, \ldots, a_n) &= 0 \end{aligned}$$

holds.

It is trivial to see that the MQ problem indeed is a problem in NP; given an assignment for the variables we can evaluate the polynomials in polynomial time and check whether the result is 0. It turns out that solving the general MQ problem in fact is an NP-complete problem. Unfortunately, in the standard reference on NP-completeness [54], no reduction is given. Merely a reference to an unpublished manuscript and a private communication is cited.

Therefore we give an outline for a reduction of 3SAT – one of Karp's original 21 NP-complete problems[69] – to the MQ problem for the case of $q = 2$ in the following. A full proof for the general case can be found in [114]; it covers not just the case of finite fields, it even transfers to the MQ problem over domains.

Problem 1.5.2 (3SAT). Let $k, n \in \mathbb{N}$, $L = \{L_1, \ldots, L_n\}$ be a set of literals, $N = \left\{\overline{L_1}, \ldots, \overline{L_n}\right\}$ the corresponding set of negated literatals, $\vee$ the boolean "or" operator, $\wedge$ the boolean "and" operator and

$$(a_{1,1} \vee a_{1,2} \vee a_{1,3}) \wedge \ldots . \wedge (a_{k,1} \vee a_{k,2} \vee a_{k,3})$$

be a boolean formula for $a_{i,j} \in (L \cup N)$, $1 \leq i \leq k$, $1 \leq j \leq 3$. Decide whether this formula is true or false.

Theorem 1.5.3. *The 3SAT problem is polynomial-time reducible to the MQ problem.*

Given an n-literal, k-clause instance of the 3SAT problem, we want to transform the problem into a set of quadratic polynomials of the ring $GF(2)[x_1, \ldots, x_t]$ such that the number of variables t grows at most polynomially in n. This works by first transforming the boolean formula into set of cubic polynomials from which quadratic polynomials are derived by introducing additional "intermediate" variables.

In the following we need to introduce temporary variables y_i, $1 \leq i \leq n$ to deal with negated literals. We then perform the following transformations to obtain polynomials from the formula:

1. Negated literals N_i become expressions of the form $y_i = (1 - x_i)$, positive literals L_i simply get transformed into $y_i = x_i$
2. Replace each clause $(a_u \vee a_v \vee a_w)$ by $C_k := (y_u y_v y_w + y_u y_v + y_u y_w + y_v y_w + y_u + y_v + y_w)$
3. Turn each converted clause C_k into a polynomial $1 + C_k$, expanding the temporary variables y_i appropriately.

All of the converted polynomials $1 + C_k$ are of degree three. By introducing $t = \binom{n}{2}$ new variables for quadratic terms we can transform the cubic system into a quadratic system of t equations.

Chapter 2

Iterated Block Ciphers

In this chapter we give an overview of both the high-level construction principles of as well as the most common attacks against iterated block ciphers. This class of block ciphers breaks up the encryption and decryption process into a sequence of steps, each of which is called a "round". The rounds themselves may be cryptographically weak in the sense of being easy to invert (without knowledge of the round key), however this is made up by iterating the round transformation F over a number of times r. The technique of iterating a weak transformation multiple times to obtain a strong transformation is used to increase both hardware and software efficiency of the cipher and to make it easier to analyse – in hardware less gates are needed, in software less instructions.

The third objective of this chapter is to describe both deployed iterated block ciphers and iterated block ciphers designed for experimentation with algebraic attacks. The experimental ciphers are evaluated with regards to their resistance against linear and differential attacks.

2.1 High-Level Structures of Block Ciphers

Several high-level structures have been used for building iterated block ciphers. We will restrict ourselves to the the two most commonly used constructions: Substitution linear networks and Feistel networks – including generalizations of the latter. A third, less commonly used construction is the Lai-Massey scheme [74]. This construction will not be explained as it is of no further relevance to this thesis. We note however it is used in the block ciphers IDEA [75] and FOX[1] [65]; the 8-bit S-Box of the hash function Whirlpool [99] also is constructed using the Lay-Massey scheme from a 4-bit S-Box.

[1] Now renamed to IDEA NXT by the licensor, MediaCrypt.

2.1.1 Substitution Linear Networks

Shannon was the first to come up with the concept of *product ciphers* in his seminal work "Communication Theory of Secrecy Systems" [104]. In this article he also describes the concept of alternatingly applying substitutions and permutation operations to a message, leading to the term Substitution Permutation Network (SPN), which has since been used to describe every cipher that follows this concept. However, since permutations usually allude to bit permutations and not arbitrary permutations on the message space, the term Substitution Linear Network (SLN) has been introduced to refer to ciphers that are built of an alternation of layers of Substitution boxes (S-Boxes) and linear layers.

Definition 2.1.1 (Substitution-Linear Network)**.** A substitution-linear network is a cipher $\mathcal{C}$ that is the result of a composition of invertible substitution maps S_i and (affine-)linear maps L_i with $1 \leq i \leq r$:

$$\mathcal{C} = L_r \circ S_r \circ \ldots \circ L_2 \circ S_2 \circ L_1 \circ S_1$$

For $1 \leq i \leq r$, the composition $(L_i \circ S_i)$ is the round transformation of round i.

Rijmen and Daemen later introduced the term *bricklayer transformation* [39] to describe functions which can be decomposed into a number of smaller Boolean vectorial functions that are applied in parallel to a partition on the bits – these are called *bundles* of the input. The term "bricklayer transformations" can both be used to refer to a parallel application of non-linear components such as S-Boxes as well as to parallel applications of linear components, in which case the component is called a D-Box. Invertible bricklayer transforms are called *bricklayer permutations*. The SubBytes transform used in Rijndael is an prominent example of a bricklayer permutation.

2.1.2 Feistel Networks

This design concept was first described by its inventor Horst Feistel in 1973 [48]. A Feistel network breaks the input into two pieces of equal length and only operates on half of the state in each round. More specifically:

Definition 2.1.2 (Feistel network)**.** Let $L \in \{0,1\}^n$ and $R \in \{0,1\}^n$ be the left and right halves of the input to a round. The round transformation $F : \{0,1\}^{2n} \to \{0,1\}^{2n}$ of a Feistel network then is $(L, R) \mapsto (R, L \oplus f_k(L, R))$ where $f_k : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a family of functions that is indexed by the round key k. The function f_k is called the round function.

Remark. Sometimes unkeyed Feistel constructions are used in cryptography for mixing purposes, for example in the OAEP scheme [8]. In this case the family of functions f_k collapses into a single function f.

Although the rate of diffusion per round achievable in Feistel networks is less than that of substitution-linear networks, this design principle still is widely used. This can be attributed to the fact that the round function f_k does not have to be invertible. Moreover, the same round transformation can be used for both encryption and decryption. For decrypting a block, the order of the round keys simply needs to be reversed.

The DES [88] was the first commercially available and widely deployed Feistel cipher.

2.1.3 Generalized Unbalanced Feistel Networks

In contrast to the balanced Feistel networks described in 2.1.2, an Unbalanced Feistel Network (UFN) splits the round input into two parts $L \in \{0,1\}^s$ and $R \in \{0,1\}^t$ such that $s \neq t$. A taxonomy of unbalanced cipher constructions was given by Schneier and Kelsey [103].

The family of round functions f_k used in the round transformation of course needs to have a different signature, $f_k : \{0,1\}^s \times \{0,1\}^t \rightarrow \{0,1\}^s$. For $s > t$, the UFN is called *source-heavy* and the function f *contracting*, for $s < t$ it is called *target-heavy* and the function f *expanding*. A network using the same family of functions f_k in each round is called *homogeneous*, a network where the family f_k changes in each round is called *heterogeneous*. Schneier and Kelsey even generalized the concept of UFN by not requiring the two blocks to be combined by an XOR operation – for a Generalized Unbalanced Feistel Network (GUFN) it is sufficient that one part of the input block controls another part of the input block. A GUFN in which all bits of the internal state of a round are used in the round transformation is called *complete*, if bits are left invariant by the round transformation, the cipher is called *incomplete*. The number of rounds required such that each bit of the block has been part of both the source and the target block at least once is called *cycle*.

2.2 Last-Round Attacks against Block Ciphers

Most attacks against block ciphers with r rounds aim at computing or guessing the round key of the last round. Having determined this round key, the last round can be peeled off, reducing the problem to attacking a cipher with $r-1$ rounds. Most of the times, the last round key is obtained by distinguishing the $(r-1)$-round cipher from a random permutation.

This process is then repeated until all round keys have been determined. In this section we will examine a number of last-round attacks and metrics for measuring the resistance of a cipher against these attacks.

2.2.1 Differential Cryptanalysis

Differential cryptanalysis is a chosen-plaintext attack that was first publicly demonstrated by Biham and Shamir against the block cipher FEAL [11] and later refined into an attack on the full-round DES [13]. It was later revealed by Don Coppersmith that this method of cryptanalysis was already known to both the NSA and IBM during the design phase of DES in 1974. This explains why the full-round DES shows good resistance against this method – in order to carry out an attack against all of the 16 rounds of DES, an attacker needs to be able to chose 2^{47} plaintexts and obtain the resulting ciphertexts.

By encrypting a pair of carefully selected plaintexts under the same key to ciphertexts, the attacker is able to predict whether certain bits of the input to the last round are equal or not. This is achieved by using a difference pattern on the input. Let $f : GF(2)^n \rightarrow GF(2)^n$ be a vectorial boolean function. A difference pattern is defined as follows:

Definition 2.2.1 (Difference pattern)**.** A difference pattern for f is a tuple (Δ_I, Δ_O) such that for an input difference Δ_I the output difference $f(x) + f(x + \Delta_I)$ possibly has the value Δ_O.

Usually, the difference pattern is only defined on the round transformation. To follow a path of difference patterns, the notion of "characteristic" is defined:

Definition 2.2.2 (Characteristic)**.** A r-round characteristic is a $(r + 1)$-tuple of difference patterns $(\Lambda_1, \ldots, \Lambda_{r+1})$.

For a one-round characteristic, the probability that a given difference in the inputs to the rounds results in the predescribed output difference is called the probability of the characteristic, for an arbitrary function $f : GF(2)^n \rightarrow GF(2)^n$ we speak of the differential probability:

Definition 2.2.3. Let X denote a random variable uniformly distributed in $GF(2)^n$. The differential probability for a pair $(\Delta_I, \Delta_O) \in GF(2)^n \times GF(2)^n$ with $\Delta_I \neq 0$ is defined as

$$\mathrm{DP}(\Delta_I, \Delta_O) = \Pr_X \{\rho(X) + \rho(X + \Delta_I) = \Delta_O\}$$

Assuming that the input values for each round are distributed randomly and independent of each other (Markov assumption), the probability of an r-round characteristic simply is the product of the probabilities of the one-round characteristics.

After the invention of differential cryptanalysis by Biham and Shamir, Lai, Massey and Murphy introduced the notion of the *differential*[75]. They noticed that for the success of the attack it is irrelevant whether the intermediate values of the characteristic match, as long as the output at round r

matches the output difference for an input matching the input difference in the first.

Definition 2.2.4 (Differential)**.** A r-round differential is a tuple (Δ_I, Δ_O) such that Δ_I is the input difference and Δ_O is the output difference after r rounds.

Differential cryptanalysis was significantly refined after its discovery: Truncated differential attacks, higher-order differentials, boomerang attacks. For hash functions of the MD4 family, Wang showed how to combine the modular-additive differentials with XOR differentials [110]. This result has changed the landscape of hash function cryptanalysis significantly.

When measuring the resistance of a cipher against differential cryptanalysis, only "basic" differential cryptanalysis is taken into account. In Section 4.5.1 we present a simple application of differential techniques.

2.2.2 Linear Cryptanalysis

Linear cryptanalysis is a known-plaintext attack that was first proposed by Matsui and Yamagashi against the block cipher FEAL [83]. Again, just like in the case of differential cryptanalysis, after its successful use against FEAL it was turned into an attack against DES by Matsui [82] that was also practically demonstrated [81]. This attack required 2^{43} known plaintexts and corresponding ciphertexts. To this date, linear cryptanalysis is the most powerful attack known against DES – if we disregard brute-force.

Linear cryptanalysis works by modelling the non-linear components of a cipher by affine-linear approximations. This sounds somewhat easier than it turns out to be in practice: Usually, one starts by determining "good" linear approximations for individual components of the cipher, then builds an approximation for a single round from these and finally searches for a path through the cipher that makes use of the round approximations. By a "good" approximation, an affine-linear function approximating the original function with a probability $p = 0.5 + \varepsilon$ with $|\varepsilon|$ as large as possible is meant. This variable ε is called the bias.

Linear cryptanalysis uses bit masks which indicate which bits of the input and output are used in a linear approximation:

Definition 2.2.5. Let $(a, b) \in GF(2)^n \times GF(2)^n$ be a pair with $a \neq 0$ being the input mask and b being the output mask. The linear probability for (a, b) then is defined as

$$\mathrm{LP}(a, b) = (2 \cdot \Pr_X \{\langle a, X\rangle = \langle b, \rho(X)\rangle\} - 1)^2$$

Similar to the case of differential cryptanalysis, a vector of masks $A = (a_1, \ldots, a_{r+1})$ with $a_i \neq 0$ for all $1 \leq i \leq r$ is called *linear characteristic* of a cipher.

Mitsuru Matsui proposed the following lemma, called Piling-Up Lemma in [82]:

Lemma 2.2.1 (Piling-up lemma)**.** *Assume $X_1, \ldots, X_n$ are independent random variables representing bits and $\varepsilon_1, \ldots, \varepsilon_n$ are their respective biases. We can then calculate the bias ϵ of $X_1 \oplus \ldots \oplus X_n$ as follows:*

$$\varepsilon = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

Using the Piling-Up Lemma, one can estimate the probability of success of a linear attack if the probabilities for individual approximations are known. This however under the assumption that we are dealing with a Markov cipher. Given the affine-linear expression approximating a cipher with probability p we can expect to an attack using linear cryptanalysis to require $\approx p^{-2}$ known plaintext/ciphertext pairs.

Daemen, Govaerts and Vandewalle subsequently introduced the concept of *correlation matrices* for Boolean mappings [36], which provide better insight into the mechanisms of linear cryptanalysis.

2.2.3 Integral Cryptanalysis

The concept of integral cryptanalysis was invented by Lars Knudsen as an attack against the block cipher Square and is presented in the paper describing the very same block cipher [37] as the so-called *Square attack*. The term *integral cryptanalysis* was coined only later however, in a paper by Knudsen and Wagner with this precise title. Integral cryptanalysis can most successfully applied against ciphers with a byte-wise structure and work best when bijective components are used. Both criteria are fulfilled in the case of Rijndael and indeed an extension of the original Square attack is the best known current cryptanalysis of AES-256, breaking 9 out of 14 rounds [49].

Lucks later extended the concept [78] in an attack on a reduced-round version of Twofish and called it *saturation attack*.

In [15], Biryukov and Shamir describe the concept of multiset attacks which are closely related to integral cryptanalysis. Multiset attacks are attacks against block ciphers that exploit properties of multisets that remain invariant under the round transformation, regardless of the choice of the round key. We will not describe neither integral cryptanalysis nor multi-set attacks in further detail, but rather suggest as an open research problem to model these attacks algebraically. Making advances in this field is expected lead to new cryptanalytic methods.

2.3 Selected Standardized Block Ciphers

This section will give descriptions of block ciphers that have been proposed as a standard or as a part thereof. The criterion that was applied for a block cipher to appear in this section was that it is of importance to a result on algebraic cryptanalysis presented in this thesis.

2.3.1 The Advanced Encryption Standard (AES)

The cipher Rijndael [38] is a SLN that has been selected as the AES in 2001 [89], a United States encryption standard that since has been adopted world wide. This section briefly lists the components used in this cipher. The only difference between Rijndael and the standardized AES is that AES only supports a block size of 128 bits, whereas Rijndael supports 128, 192 and 256 bit block sizes.

- SubBytes – applies the S-Box to all elements of the internal state
- ShiftRows – shifts rows of the internal state cyclically to the right
- MixColumns – mixes the columns using a $GF(2^8)$-linear transformation
- AddRoundKey – $GF(2)$-addition of the round key to the internal state

We do not describe Rijndael here, as Section 2.4.1 gives an algorithmic description of a parametrized variant called Mini-AES that can be scaled up to AES-128.

2.3.2 SMS4

The cipher SMS4 is a Chinese design proposed for the WAPI standard, a wireless LAN protocol that will be enforced throughout China in the near future. This section will describe the cipher and show that the design is fragile by giving weak keys for a variant that only differs in the round constants.

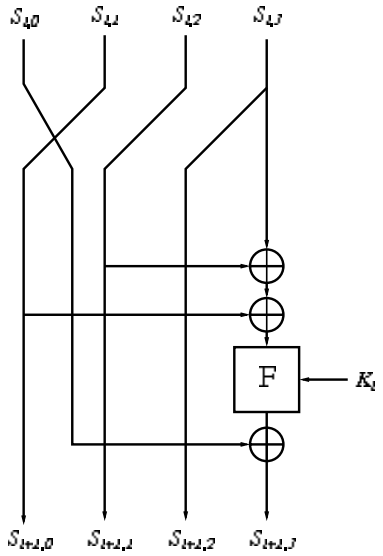
SMS4 is a 32 round unbalanced Feistel network with a block and key size of 128 bits. Using the taxonomy of Section 2.1.3, the cipher is a homogeneous, complete, source-heavy (96:32) UFN with 8 cycles.

Let the internal state be denoted by $\mathcal{S} = (S_1, S_2, S_3, S_4)$ where $S_i \in GF(2)^{32}$. The round keys of the cipher shall be denoted by $K_i \in GF(2)^{32}$.

Define the linear diffusion function λ as

$$\begin{aligned} \lambda : GF(2)^{32} &\rightarrow GF(2)^{32} \\ x &\mapsto x \oplus (x \lll 2) \oplus (x \lll 10) \oplus (x \lll 18) \oplus (x \lll 24) \end{aligned}$$

Figure 2.1: One round of the SMS4 Unbalanced Feistel Network



and the brick-layer function γ applying an 8-bit S-Box to the input 4 times in parallel as:

$$\begin{aligned} \gamma : GF(2)^{32} &\rightarrow GF(2)^{32} \\ x &\mapsto (\rho(x_{[31...24]}), \rho(x_{[23...16]}), \rho(x_{[15...8]}), \rho(x_{[7...0]})) \end{aligned}$$

The round function then simply is the composition of the functions λ and γ:

$$\begin{aligned} F : GF(2)^{32} \times GF(2)^{32} &\rightarrow GF(2)^{32} \\ (X, K_i) &\rightarrow \lambda(\gamma(X \oplus K_i)) \end{aligned}$$

and the round transformation R that maps $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$ under the round key K_i is defined as:

$$\begin{aligned} R : GF(2)^{128} \times GF(2)^{32} &\rightarrow GF(2)^{128} \\ (S_1, S_2, S_3, S_4, K_i) &\mapsto (S_2, S_3, S_4, S_1 \oplus F(S_2 \oplus S_3 \oplus S_4, K_i)) \end{aligned}$$

The S-Box used in SMS4 was shown to be based on an inversion mapping over $GF(2)[\theta]$ with both an input and an output affine linear transform over $GF(2)$ in [77].

The Key Schedule

In total, 32 round key words k_i are generated from a 128-bit cipher key. For the key schedule a function F' is used that is almost identical to the round

transformation; the only thing changed is the linear transform. Instead of λ, the following mapping λ' is used:

$$\begin{aligned} \lambda' : GF(2)^{32} &\rightarrow GF(2)^{32} \\ x &\mapsto x \oplus (x \lll 13) \oplus (x \lll 23) \end{aligned}$$

In order to obtain the round keys, the cipher key K is first masked with a so-called system parameter

$$T = \texttt{0xA3B1BAC656AA3350677D9197B27022DC}$$

as follows:

$$\begin{aligned} k_{-4} &= K_{[127..96]} \oplus T_{[127..96]} \\ k_{-3} &= K_{[95..64]} \oplus T_{[95..64]} \\ k_{-2} &= K_{[63..32]} \oplus T_{[63..32]} \\ k_{-1} &= K_{[31..0]} \oplus T_{[31..0]} \end{aligned}$$

The reasoning behind the masking of the cipher key is not explained in the design document. The round key of the i-th round is computed as follows:

$$k_i = k_{i-4} \oplus \lambda'(\gamma(k_{i-3} \oplus k_{i-2} \oplus k_{i-1} \oplus \kappa_i))$$

where κ_i are key constants. The key constants κ_i are of the form

$$\kappa_i = ((28 \cdot i), (28 \cdot i + 7), (28 \cdot i + 14), (28 \cdot i + 21))$$

where each component of the above vector is a byte, the operators $\cdot$ and $+$ denote the multiplication respectively addition in $\mathbb{Z}_{256}$.

Weak Keys for Modified Round Key Constants

For slightly modified round key constants in the key schedule, the cipher will exhibit a class of 2^{64} weak keys. For all of these keys, the cipher exhibits an invariant property over an arbitrary number of rounds. This invariance can be used to effectively distinguish the encryption function from a random permutation. Once the use of a weak key is detected, the key search space for an attacker of course shrinks from 2^{128} to 2^{64}. The property shows an unexpected fragility of the cipher design and in our opinion casts serious doubt on its strength.

Definition 2.3.1. Let $a \in GF(2)^{2n}$. If $a = b||b$ for an element $b \in GF(2)^n$, then we say that the element a has a $1/2$-repetition property; alternatively a may be called $1/2$-repeated.

Theorem 2.3.1. *Let $(s_1, \ldots, s_k) \in \mathbb{Z}^k$ be a vector of shift offsets. Any $2n$-bit function $g : GF(2)^{2n} \to GF(2)^{2n}$ of the form*

$$x \mapsto \bigoplus_{i=1}^{k} (x \lll s_i)$$

preserves the $1/2$-repetition property.

Proof. Obviously the invariance condition is preserved under addition if it holds for all elements of the sum. By induction the invariance condition for n-bit cyclic shifts can be derived for 1-bit shits. □ □

Modifying all round key constants κ_i to be $1/2$-repeated, we obtain 2^{64} cipher keys for which all round keys possess the $1/2$-repetition property; note that due to the masking of the cipher key with the system parameter in the key generation the 2^{64} actual cipher keys are not $1/2$-repeated though. Both the function for generating the round keys and the round transformation preserve the invariance property for these keys. It follows that for plaintexs in which each word is $1/2$-repeated, we obtain ciphertexts that also are $1/2$-repeated. Henceforth, these cipher variants are insecure. A similar idea has been investigated by Gilbert and Handschuh for SHA-256 in [56].

2.3.3 Cryptomeria

The Cryptomeria block cipher is a proprietary 64-bit Feistel cipher with a 56-bit key length used in the and standards. It is used to protect content on DVD-Audio discs, recordable DVD-Video media and on SD cards. Although the design of Cryptomeria itself is public [1], the actual S-Box used is proprietary; it is considered a trade secret by the licensing body 4C Entity LLC. Moreover, the S-Box seems to be application-specific. A S-Box different from the DVD-Audio case is used for the recordable DVD-Video case. The S-Box used in the case of SD cards is not publicly known.

The Encryption Operation

Unlike other Feistel ciphers, Cryptomeria does not use vectorial addition modulo $GF(2)$ for mixing the left and the right half of the block. Instead modular addition in the residue class group $\mathbb{Z}_{2^{32}}$ is used for the Feistel step. The cipher employs a total of 10 rounds, however each round only applies a single 8×8 S-box instead of a brick layer transform. We name this transformation S:

$$S : GF(2)^{32} \to GF(2)^{32} : (x_1, x_2, x_3, x_4)x \mapsto (S(x_1), x_2, x_3, x_4) \qquad (2.1)$$

The following three functions are used for diffusion within the round transformation:

$$t_2 : GF(2)^{16} \to GF(2)^8 : (x, y) \mapsto ((x \oplus \texttt{0x65}) \lll 1) \oplus y \tag{2.2}$$
$$t_3 : GF(2)^{16} \to GF(2)^8 : (x, y) \mapsto ((x \oplus \texttt{0x2b}) \lll 2) \oplus y \tag{2.3}$$
$$t_4 : GF(2)^{16} \to GF(2)^8 : (x, y) \mapsto ((x \oplus \texttt{0xc9}) \lll 5) \oplus y \tag{2.4}$$

namely in the linear transform L:

$$L : GF(2)^{32} \to GF(2)^{32} : \tag{2.5}$$
$$(x_1, x_2, x_3, x_4) \mapsto (x_1, t_2(x_1, x_2), t_3(x_1, x_3), t_4(x_1, x_4)) \tag{2.6}$$

Using all of the above definitions, the round function F of Cryptomeria reads as follows:

$$F : \mathbb{Z}_{2^{32}} \to \mathbb{Z}_{2^{32}} : x \mapsto L(S(x)) \tag{2.7}$$

The plaintext (L_0, R_0) is encrypted to the ciphertext (L_{10}, R_{10}) using the following sequence of steps:

$$(L_i, R_i) = F(R_{i-1}, L_{i-1} + K_i)$$

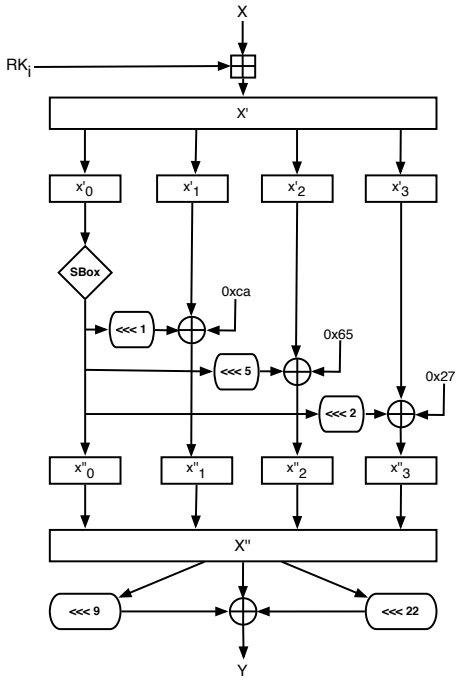
where K_i denotes the round key of round i and $+$ denotes addition in the residue class ring $\mathbb{Z}/\mathbb{Z}_{2^{32}}$.

The Key Schedule

The key schedule of Cryptomeria is simple, yet non-linear. The 56-bit cipher key is rotated by a 17 bit positions to the left in each round. Subsequently a modular addition of an 12-bit quantity is performed. This 12-bit quantity is derived from parts of the cipher key and the round number which are XORed and fed through the S-Box. More precisely, the key schedule for any round key can be described as a function taking as input the cipher key x and the round number r:

$$GF(2)^{56} \times \mathbb{N} \to GF(2)^{56} : (x, r) \mapsto (x \lll (17r \bmod 56)) + (S(x_{[0..7]} \oplus r) \ggg 4)$$

Figure 2.2: The Cryptomeria round function



2.4 Experimental Block Ciphers

Practically attacking reduced-round versions of the ciphers presented in the last section does not work so well as the number of variables needed to represent them as a polynomial system is too large. It therefore makes sense to invent experimental ciphers that are structurally similar to deployed block ciphers but which can be modelled with a lower number of variables.

In this section we describe block ciphers that have been proposed to facilitate easier experimentation in the field of algebraic cryptanalysis. The first family, Mini-AES, is a scaled-down version of Rijndael while the ciphers described in Section 2.4.2, Flurry and Curry, are experimental ciphers in the sense that they were invented to specifically prove the point that algebraic attacks against block ciphers with a real-world block and key size which resist differential and linear cryptanalysis indeed are possible.

2.4.1 Mini-AES

Mini-AES is a parametrized family of ciphers derived from Rijndael. It was proposed by the author in his Diplom thesis [111]. It is very similar to the SR family of ciphers proposed by Cid, Murphy and Robshaw [25] – the difference being that Mini-AES gives more degrees of freedom in the

parameters than SR – and mimics the design criteria of the Rijndael cipher; both families are instantiatable with much smaller parameters than Rijndael and are thus much better suited for experimentation with algebraic attacks.

The internal state of the Rijndael as well as our family of Mini-Rijndaels for this chapter is represented as a matrix with the elements being ordered columnwise.

$$S := \begin{pmatrix} a_{0,0} & \cdots & a_{0,N_b-1} \\ \vdots & \ddots & \vdots \\ a_{N_a-1,0} & \cdots & a_{N_a-1,N_b-1} \end{pmatrix} \in \mathbb{F}^{N_a \times N_b}$$

Parameters

The following table lists parameters of our cipher that can be easily adapted. Note that changes such as assigning $A = 0$ and $b = 0$ respectively may make algebraic cryptanalysis easier, but then again does not reflect the Rijndael design criteria[2].

$s \in \mathbb{N}$	width of the S-Box in bits
$m \in \mathbb{F}_2[\theta]$	minimal polynomial of the finite field $\mathbb{F}_{2^s}$
$N_r \in \mathbb{N}$	number of rounds in the cipher
$N_a \in \mathbb{N}$	number of rows in the state/key matrix
$N_b \in \mathbb{N}$	number of columns in the state/key matrix
$N_k \in \mathbb{N}$	number of columns in the key matrix
$A \in \mathbb{F}_2^{N_s \times N_s}$	matrix for affine transformation in SubElement
$b \in \mathbb{F}_2^{N_s}$	vector for affine transformation in SubElement
$M_{\text{mix}} \in \mathbb{F}_{2^s}^{N_a \times N_a}$	matrix for the MixColumns step

AES-128, that is Rijndael with a block and key size of 128 bits, is obtained by using the following parameters:

- $s = 8,\ N_r = 10,\ N_a = 4,\ N_b = 4,\ N_k = 4$
- $m = \theta^8 + \theta^4 + \theta^3 + \theta + 1$
- $M_{\text{mix}} = \begin{pmatrix} \theta & \theta+1 & 1 & 1 \\ 1 & \theta & \theta+1 & 1 \\ 1 & 1 & \theta & \theta+1 \\ \theta+1 & 1 & 1 & \theta \end{pmatrix}$

[2]since the resulting cipher is not protected against interpolation attacks then

- $A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$

An Algorithmic Cipher Description

The sequence of operations to be performed is the following:

- Add round key K_0 in round zero (this effectively is the cipher key for $N_k = N_b$)
- $N_r - 1$ rounds of the following transformations:
  - Apply SubElement to each element of the state
  - Shift rows cyclically to the left
  - Diffuse internal state columnwise with MixColumns
  - Add round key K_i for round i.
- In the last round the MixColumns step is left out:
  - Apply SubElement to each element of the state
  - Shift rows cyclically to the left
  - Add round key K_i for round i.

AddRoundKey The function AddRoundKey simply performs a bitwise XOR of the internal state with the round subkey.

SubElement SubElement applies the non-linear invertible function γ – the S-Box function – to each element $a_{i,j}$ of the state. The S-box uses the same construction as Rijndael, generalized to an arbitrary input/output size of s bits.

The S-Box is composed of two functions f, g, which are defined as follows:

$$f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}, \qquad x \mapsto \begin{cases} x^{-1} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

$$g : \mathbb{F}_2^n \to \mathbb{F}_2^n, \qquad x \mapsto Ax + b$$

Algorithm 3 MiniRijndaelEncrypt

Input: $P \in \mathbb{F}^{N_a \times N_b}$
Input: $K_0, \ldots, K_r \in \mathbb{F}^{N_a \times N_b}$
Output: $C \in \mathbb{F}^{N_a \times N_b}$

$S_0 \leftarrow \mathsf{AddRoundKey}(P, K_0)$
for $r \leftarrow 1$ to N_r **do**
  $S_i \leftarrow \mathsf{SubElement}(S_{i-1})$
  $S_i \leftarrow \mathsf{ShiftRows}(S_i)$
  if $r \neq N_r$ **then**
    $S_i \leftarrow \mathsf{MixColumns}(S_i)$
  end if
  $S_i \leftarrow \mathsf{AddRoundKey}(S_i, K_r)$
end for
$C \leftarrow S_r$

Algorithm 4 AddRoundKey

Input: $S \in \mathbb{F}^{N_a \times N_b}$
Input: $K \in \mathbb{F}^{N_a \times N_b}$
Output: $S' \in \mathbb{F}^{N_a \times N_b}$

$S' \leftarrow S \oplus K$

with A invertible. The transformation γ then is defined as a sequence of applying the inversion f, the canonical mapping from $\mathbb{F}_{2^s}$ to $\mathbb{F}_2^s$, the affine transformation g and finally the canonical mapping from $\mathbb{F}_2^s$ to $\mathbb{F}_{2^s}$ Note that f can also be expressed as $x \mapsto x^{2^n-2}$ for $n \geq 2$. We further require the matrix A used in the mapping g to be circulant, imposing the condition

$$A_{i,j} = A_{0,i-j \bmod n} \text{ for all } i, j$$

on its elements and the S-Box to have no fixed and no opposite fixed points.

ShiftRows Each row i of the internal state is cyclically shifted λ_i positions to the left by the ShiftRows transformation.

MixColumns Each column of the state is multiplied by an invertible and circulant matrix M_{mix}. The matrix M_{mix} used in Rijndael was chosen such that its branch number (see Definition 2.4.3) is maximal – to obtain optimal

Algorithm 5 SubElement

Input: $S \in \mathbb{F}^{N_a \times N_b}$
Output: $S' \in \mathbb{F}^{N_a \times N_b}$

for $i \leftarrow 0$ to $N_a - 1$ **do**
  for $j \leftarrow 0$ to $N_b - 1$ **do**
    $S'_{i,j} \leftarrow \gamma(S_{i,j})$
  end for
end for

Algorithm 6 ShiftRows

Input: $S \in \mathbb{F}^{N_a \times N_b}$
Output: $S' \in \mathbb{F}^{N_a \times N_b}$

for $i \leftarrow 0$ to $N_a - 1$ **do**
  for $j \leftarrow 0$ to $N_b - 1$ **do**
    $l \leftarrow (j + \lambda_i) \bmod N_b$
    $S'_{i,j} \leftarrow S_{i,l}$
  end for
end for

diffusion; thus for Mini-Rijndael we should do likewise. For a single column i of a $4 \times N_b$ state the MixColumns transformation looks as follows:

$$M_{\text{mix}} \cdot \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix} \quad \mapsto \quad \begin{pmatrix} a'_{0,i} \\ a'_{1,i} \\ a'_{2,i} \\ a'_{3,i} \end{pmatrix}$$

Algorithm 7 MixColumns

Input: $S \in \mathbb{F}^{N_a \times N_b}$
Output: $S' \in \mathbb{F}^{N_a \times N_b}$

for $i \leftarrow 0$ to $N_b - 1$ **do**
  $v \leftarrow S_{0 \ldots N_a - 1, i}$
  $v \leftarrow M \cdot v^T$
  $S'_{0 \ldots N_a - 1, i} \leftarrow v^T$
end for

Key Scheduling

The key schedule of Rijndael defines two types of key expansion, one variant for $N_k \leq 6$ and a slightly different one for $N_k > 6$. For the sake of simplicity we only define one version of the key expansion.

The columns of the expanded key are recursively computed. Columns $(k_0, \ldots, k_{N_k-1})$ of the expanded key are equal to the cipher key, column $i \geq N_k$ can be calculated as follows:

(i) if $N_k \nmid i$, column i is the bitwise XOR of column $i - N_k$ and column $i - 1$

(ii) if $N_k \mid i$, column i is the bitwise XOR of column $i - N_k$ and the result of the application of a non-linear function to column $i-1$. This non-linear function applies the S-Box transformation to each of the elements in the column, cyclically rotates the column and then adds the round constant $\mu_i := \theta_{\mathbb{F}}^{\lfloor \frac{i}{N_k} \rfloor - 1}$.

Algorithm 8 MiniRijndaelKeySchedule

Input: $k \in \mathbb{F}^{N_a \times N_k}$
Output: $K_0, \ldots, K_r \in \mathbb{F}^{N_a \times N_b}$

for $r \leftarrow 1$ to N_r **do**
  $K_r \leftarrow K_{r-1}$
  for $i \leftarrow 0$ to $N_b - 1$ **do**
    for $j \leftarrow 0$ to $N_a - 1$ **do**
      if $i = 0$ **then**
        $K_{r,j,i} \leftarrow K_{r,j,i} + \mathsf{SubElement}(K_{r-1,\ (j+1) \bmod N_b,\ m-1})$
        if $j = 0$ **then**
          $K_{r,0,0} \leftarrow K_{r,0,0} + \theta_{\mathbb{F}}^{r-1}$
        end if
      else
        $K_{r,j,i} \leftarrow K_{r,j,i} + K_{r,j,i-1}$
      end if
    end for
  end for
end for

2.4.2 Flurry and Curry

Flurry, a Feistel Network and Curry, a substitution linear network are two block cipher families proposed by Andrei Pyshkin and the author of this thesis. The goal of constructing these families of ciphers was to show that block

ciphers with a sound design strategy against linear and differential cryptanalysis as well as brute-force attacks can be built that fall prey to algebraic Gröbner basis attacks. This goal was reached by using certain monomial and inversion S-Box functions over extension fields of $GF(2)$. For the selected S-Box functions theoretical analyses of their resistance against linear and differential cryptanalysis exist – showing them to be close to optimal. However, in contrast to real-world block ciphers, we do not compose these functions with affine-linear functions over $GF(2)$ as was done in Rijndael and KASUMI; this in order to make the cipher easily representable over the extension field. Also, most of the instances we were able to successfully attack use S-Boxes that are significantly wider than the ones in use in deployed block ciphers.

The description of these ciphers has been published in the Proceedings of CT-RSA 2006.

The Feistel Case: Flurry

We construct the family FLURRY(n, m, r, f, D) of Feistel ciphers. The parameters used are:

- $m \in \mathbb{N}$: the plaintext space, the ciphertext space and the cipher key space are F^{2m}.
- $r \in \mathbb{N}$: the number of rounds
- $f : F \to F$: a non-linear mapping giving the S-Box of the round function
- $D = (d_{i,j}) \in F^{m \times m}$: a matrix describing the linear diffusion mapping of the round function.

We set $R = (r_1, \ldots, r_m) \in F^m$, $L = (l_1, \ldots, l_m) \in F^m$ and $K = (k_1, \ldots, k_m) \in F^m$. The round transformation $\rho : F^m \times F^m \times F^m \to F^m \times F^m$ of a FLURRY cipher is then defined as:

$$\rho(L, R, K) = (R, G(R, K) + L)$$

with $G : F^m \times F^m \to F^m$ being the parallel application of m S-Boxes followed by a linear transform:

$$G(r_1, \ldots, r_m, k_1, \ldots, k_m) = D \times \begin{pmatrix} f(r_1 + k_1) \\ f(r_2 + k_2) \\ \vdots \\ f(r_m + k_m) \end{pmatrix}.$$

A plaintext (L_0, R_0) is encrypted into a ciphertext (L_r, R_r) by iterating the round transformation ρ over r rounds:

$$\begin{aligned}(L_i, R_i) &= \rho(L_{i-1}, R_{i-1}, K_{i-1}) \qquad i = 1, 2, \ldots, r-1 \\ (L_r, R_r) &= \rho(L_{r-1}, R_{r-1}, K_{r-1}) + (K_r, K_{r+1})\end{aligned}$$

After the last round transformation, an additional key addition is performed on both halves of the state. Analogously, using the inverse round transformation ρ^{-1}

$$\rho^{-1}(L, R, K) = (G(L, K) + R, L)$$

we can decrypt a ciphertext with the following sequence of steps:

$$\begin{aligned}(L_{r-1}, R_{r-1}) &= \rho^{-1}(L_r + K_r, R_r + K_{r+1}, K_{r-1}) \\ (L_{i-1}, R_{i-1}) &= \rho^{-1}(L_i, R_i, K_{i-1}) \qquad i = r-1, r-2, \ldots, 1\end{aligned}$$

The number of F-components of a cipher key, plaintext or ciphertext is denoted by $t = 2m$.

The key schedule The key schedule is affine over F. We write the cipher key as a tuple of vectors $(K_0, K_1) \in F^m \times F^m$. Let the round keys for the first two rounds be K_0, K_1 and recursively compute subsequent round keys for $2 \leq i \leq r+1$ as follows:

$$K_i = D \cdot K_{i-1}^T + K_{i-2} + v_i$$

where D is the same matrix used in the round function of the cipher and the v_i are round constants:

$$v_i = ((\theta + 1)^i, (\theta + 1)^{i+1}, \ldots, (\theta + 1)^{i+m-1})$$

The SPN Case: Curry

In this section we construct a family CURRY(n, m, r, f, D) of ciphers similar to SQUARE [37]. We explain the parameters used:

- $m \in \mathbb{N}$: the plaintext space, the ciphertext space and the cipher key space are $F^{m \times m}$.
- $r \in \mathbb{N}$: the number of rounds
- $f : F \to F$: a bijective non-linear mapping giving the S-Box of the round function
- $D = (d_{i,j}) \in F^{m \times m}$: an invertible matrix used for diffusion

The round function $\rho : F^{m\times m} \times F^{m\times m} \rightarrow F^{m\times m}$ of a CURRY cipher is defined as:

$$\rho(S, K) = D \cdot G(S + K)^T$$

with $G : F^{m\times m} \rightarrow F^{m\times m}$ being the parallel application of m^2 S-Boxes:

$$G((s_{i,j})) = (f(s_{i,j}))$$

A plaintext S_0 is encrypted into a ciphertext S_r by iterating the round transformation ρ exactly r times followed by an additional key addition after the last round:

$$\begin{aligned} S_i &= \rho(S_{i-1}, K_{i-1}) \qquad i = 1, 2, \ldots, r-1 \\ S_r &= \rho(S_{r-1}, K_{r-1}) + K_r \end{aligned}$$

Analogously, using the inverse round transformation ρ^{-1}

$$\rho^{-1}(S, K) = G^{-1}((D^{-1} \cdot S)^T) + K$$

we can decrypt a ciphertext with the following sequence of steps:

$$\begin{aligned} S_{r-1} &= \rho^{-1}(S_r + K_r, K_{r-1}) \\ S_{i-1} &= \rho^{-1}(S_i, K_i) \qquad i = r-1, r-2, \ldots, 1 \end{aligned}$$

Just as for FLURRY, let the number of F-components of a key, plaintext or ciphertext be denoted by t, this time $t = m^2$.

The key schedule For CURRY the first round key is equivalent to the cipher key $K_0 \in F^{m\times m}$. Just as for FLURRY the key schedule is affine over F. Subsequent round keys $K_i, i \geq 1$ are recursively computed as follows:

$$K_i = D \cdot K_{i-1} + M_i$$

where D is the same matrix used in the round function and $M_i = ((a_{j,l}))$ with $a_{j,l} = \theta^{i+(j-1)m+l}$. The matrices M_i are round constants.

Selected Parameters

We will now specify suitable parameters for the S-Box function and the linear transformation. These will be used to more thoroughly investigate instances of our cipher constructions later in this thesis. The number of rounds shall be left unspecified for now.

Table 2.1: S-Box mappings over $GF(2^n)$ with $n \in \{8, 16, 32, 64\}$

function	mapping	bijective	δ-uniformity	$\mathcal{N}(f)$
f_{-1}	$x \mapsto \begin{cases} x^{-1} & \text{iff } x \neq 0 \\ 0 & \text{iff } x = 0 \end{cases}$	yes	4	$2^{n-1} - 2^{\frac{n}{2}}$
f_3	$x \mapsto x^3$	no	2	$\geq 2^{n-1} - 2^{\frac{n}{2}}$
f_5	$x \mapsto x^5$	no	4	$\geq 2^{n-1} - 2^{\frac{n}{2}+1}$
f_7	$x \mapsto x^7$	yes	≤ 6	$\geq 2^{n-1} - 3 \cdot 2^{\frac{n}{2}}$

The S-Box functions The only non-linear components of FLURRY and CURRY are the S-Boxes. In order to achieve resistance against differential and linear cryptanalysis even for a small number of rounds these must be chosen very carefully. The strength that an S-Box provides against these attacks is measured by its differential uniformity and its nonlinearity respectively. These are defined as follows:

Definition 2.4.1. Let $f : F \to F$ be a mapping and

$$\delta = \max_{\substack{a,b \in F \\ a \neq 0}} \#\{x \in F \ : \ f(x+a) = f(x) + b\}.$$

Then f is called differentially δ-uniform.

In the following definition we use the bijective map

$$F \to GF(2)^n, a = \sum_{i=0}^{n-1} \left(a_i \theta^i\right) \mapsto (a_0, \ldots, a_{n-1})$$

to identify F with $GF(2)^n$. For $a = (a_0, \ldots, a_{n-1}),\ b = (b_0, \ldots, b_{n-1})$ we set

$$\langle a, b \rangle = \sum_{i=0}^{n-1} a_i b_i$$

Definition 2.4.2. The nonlinearity of a function $f : F \to F$ is defined as

$$\mathcal{N}(f) = \min_{\substack{a,b \in F \\ b \neq 0}} \#\{x \in F \mid \langle x, a \rangle \neq \langle f(x), b \rangle\}$$

For monomial functions as well as the multiplicative inverse over finite fields of characteristic two the δ-uniformity and the nonlinearity have been well studied in the literature [90, 10, 40]. In order to make Gröbner basis attacks feasible, we keep the degree of our S-Box functions low. Table 2.1 shows the S-Box functions that we have picked.

We call f_3, f_5 and f_7 *monomial S-Boxes* and f_{-1} the *inversion S-box*.

Lemma 2.4.1. *1. f_3 is a 2-uniform mapping*

2. f_{-1} and f_5 are 4-uniform mappings.

3. f_7 has δ-uniformity of 6 or less.

Proof. Obviously for all $a, b \in F$ with $a \neq 0$ the equation $x^7 + (x + a)^7 = b$ has at most 6 roots. For claims 1 and 2, see [90]. □

Lemma 2.4.2. *1. The nonlinearity of f_{-1} is $2^{n-2} - 2^{\frac{n}{2}}$.*

2. For a polynomial function $f : F \to F$ of degree d the following holds true: $\mathcal{N}(f) \geq 2^{n-1} - \lfloor \frac{d-1}{2} \rfloor 2^{\frac{n}{2}}$

Proof. For claim 1, see [40], for claim 2 see [23]. □

The linear transformations We use matrices of Maximum Distance Separable codes – *MDS matrices* for short – for the matrix D in the linear layer and the key schedule. We chose these types of linear transformations since they have optimal diffusion properties. This strategy is widely used in modern block cipher design; all ciphers following the wide-trail design use diffusion optimal matrices. The matrix D_4 below actually is the matrix used in the `MixColumns` step of Rijndael, D_2 is equivalent to a Pseudo-Hadamard Transform over F.

$$D_2 = \begin{pmatrix} \theta & 1 \\ 1 & 1 \end{pmatrix} \qquad D_4 = \begin{pmatrix} \theta & \theta+1 & 1 & 1 \\ 1 & \theta & \theta+1 & 1 \\ 1 & 1 & \theta & \theta+1 \\ \theta+1 & 1 & 1 & \theta \end{pmatrix}$$

Rijmen and Daemen introduced the notion of the *branch number* of a linear transformation to measure the quality of the diffusion provided. For a F-vector $X := (x_1, \ldots, x_m)$ we define $w(X)$ to be the hamming weight of X, i.e. the count of all non-zero coordinates of this vector. The following definition is according to [39]:

Definition 2.4.3. Let $M \in F^{m \times m}$ be a matrix describing a be a linear map. The differential branch number $\mathcal{B}_d(M)$ of M is then defined as

$$\mathcal{B}_d(M) = \min_{\substack{X \in F^m \\ X \neq 0}} (w(X) + w(MX))$$

while the linear branch number $\mathcal{B}_l(M)$ is defined as $\mathcal{B}_l(M) = \mathcal{B}_d(M^T)$.

For a symmetric matrix such as D_2, the linear and the differential branch number clearly coincide. For the circulant matrix D_4 the linear and differential branch number coincide as well [39]. Thus in our case it suffices to speak of *the branch number* $\mathcal{B}(M)$ of a matrix M. For MDS matrices the branch number is maximal [39], i.e. $\mathcal{B}(M) = m + 1$ with m being the size of the matrix M. For block ciphers with $m = 1$ we use the identity matrix of size one, I_1, trivially resulting in $\mathcal{B}(I_1) = 2$.

Resilience against Linear and Differential Cryptanalysis

The notion of *practical security* of block ciphers against differential and linear cryptanalysis was introduced by Knudsen [72]. We will show how compute the minimum number of rounds that will make FLURRY and CURRY practically secure against differential and linear cryptanalysis.

Note that our objective was not to evaluate the strength of our ciphers against all known attacks. Our ciphers may very well be vulnerable against one or several advanced attacks even if they resist standard linear and differential cryptanalysis. Indeed, as an example we argue that the choices we have made for the S-Boxes are very weak against interpolation attacks.

A fundamental parameter that influences the complexity of differential and linear attacks is the minimum number of active S-Boxes N over consecutive rounds of the cipher. Kanda [68] gives useful results on both SLN ciphers and Feistel ciphers with a substitution-linear round function; from these we derive the following lemma:

Lemma 2.4.3. *The minimum number of active S-boxes in 4, 6, 8 consecutive rounds of a Feistel cipher with substitution-linear round function is lower bounded by $\mathcal{B}(D)$, $\mathcal{B}(D)+2$ and $2\mathcal{B}(D)+1$ respectively. For an SLN cipher the minimum number of active S-Boxes for $2r$ consecutive rounds is lower bounded by $r\mathcal{B}(D)$.*

Definition 2.4.4. Let Ω_L be the set of all linear characteristics and Ω_D the set of all differential characteristics of a cipher C. The maximum linear characteristic probability (MLCP) of C then is

$$\text{MLCP}(C) = \max_{A \in \Omega_L} \prod_{i=1}^{r} \text{LP}(a_i, a_{i+1})$$

Analogously the maximum differential characteristic probability (MDCP) of C is

$$\text{MDCP}(C) = \max_{A \in \Omega_D} \prod_{i=1}^{r} \text{DP}(a_i, a_{i+1})$$

The maximum differential probability of a function $f : F \to F$ can be calculated from δ as $p(f) = \frac{\delta}{\#F}$ where δ is according to Definition 2.4.1. The maximum linear probability of a mapping $f : F \to F$ can be computed as

$$q(f) = \left(1 - \frac{2\mathcal{N}(f)}{\#F}\right)^2$$

where $\mathcal{N}(f)$ is defined as in Definition 2.4.2. For SLN ciphers and Feistel ciphers with a substitution-linear round function the MDCP is bounded by $p(f)^N$ while the MLCP is bounded by $q(f)^N$ [68], where N is the miminum number of active S-Boxes.

From these bounds we can deduce the number of rounds required to make an instance practically secure against differential and linear cryptanalysis. According to Knudsen [72], a block cipher with dependent round keys is practically secure against differential and linear cryptanalysis if the MLCP and the MDCP is too low for an attack to work under the assumption of independent round keys. Note however that for both r-round Feistel and r-round SPN ciphers, we need to consider the MLCP and MDCP of $r - 2$ rounds because of attacks that guess bits of the first and the last round key, so-called 2R attacks.

Chapter 3

Efficient Gröbner Basis Algorithms

For the computation of normal forms state-of-the-art Gröbner basis algorithms exploit a link between Gaussian elimination and Gröbner basis computation first characterized by Daniel Lazard in 1983 [76]. This chapter presents a number of algorithms based on the Lazard's ideas in historical order. The algorithms described in this chapter are already implemented in the computer algebra package Magma [108]. However, Magma provides the algorithms as a black box implementation with only few parameters that the user can influence. This is not sufficient to advance the state of the art in algebraic cryptanalysis. To wit: in order to efficiently solve the HFE challenge 1 using F_4, a new parameter `HFE` was added to the `GroebnerBasis` command in Magma. This parameter makes the algorithm behave in a way that is tailored to specific classes of HFE instances: Magma in this case supresses all pairs of degree > 4. Using this parameter on the other hand is distinctly different from simply computing a truncated Gröbner basis at degree 4 using Magma.

It is therefore important to not only be able to operate the tools we have but to understand their inner workings and to sharpen them: Open-source software clearly gives an advantage here. The author of this thesis implemented the algorithms described in this chapter for the case of finite fields as base fields of the polynomial rings in a software package called Xylirt. This software package is written in C++ and will be released under an open-source license and integrated into the SAGE computer algebra system in the near future.

A straightforward enhancement – that has been integrated into Xylirt – is to not just efficiently deal with the case of field polynomials but also to allow to use so-called "conjugate polynomials" in the case of cipher embeddings such as the ones described in [87] for AES and in Section 4.2.2 for SMS4.

3.1 The FGLM Algorithm

In general, computing a Gröbner basis of an ideal relative to a lexicographical order results in a significantly higher computational cost than computing a Gröbner basis for the same ideal relative to a degree-reverse lexicographical order. To take advantage of this, Faugére and Lazard as well as Gianni and Mora developed algorithms for changing the order of a Gröbner basis. The merged version of these algorithms resulted in a paper proposing an algorithm that is now referred to as the "FGLM algorithm" [46].

Algorithm 9 FGLM

Input: $<_2$ – the target term ordering
Input: $G_1 \subset \mathbb{F}[x_1, \ldots, x_n]$ – Gröbner basis w.r.t. $<_1$
Output: $G_2 \subset \mathbb{F}[x_1, \ldots, x_n]$ – Gröbner basis w.r.t. $<_2$

$m = 1$ {monomial}
$M \leftarrow \emptyset$ {monomial basis}
$G_2 \leftarrow \emptyset$ {new basis}
$L \leftarrow ()$ {list of nexts}
while $m \neq \emptyset$ **do**
  if $\exists m' \in \mathrm{HM}(G_2)$ such that $m | m'$ **then**
    $v \leftarrow \phi_1(m)$
    if $\forall w \in M . \exists \lambda_w \in \mathbb{F}$ such that $v + \sum_{w \in M} \lambda_w \mathrm{second}(w) = 0$ **then**
      $p \leftarrow m + \sum_{w \in M} \lambda_w \mathrm{first}(w)$
      $G_2 \leftarrow \mathrm{cons}([p, v], M)$
    end if
    for $v \in \{x_1, \ldots, x_n\}$ **do**
      if $(v \cdot m)$ does not exist in L **then**
        $\mathsf{insert}(L, v \cdot m)$
      end if
    end for
  end if
  $m \leftarrow \mathsf{head}(L)$
end while

Given a reduced Gröbner basis G of an zero-dimensional ideal $\mathfrak{I}$ relative to a term order $<_1$ and a different term order $<_2$, the FGLM algorithm computes the Gröbner basis of I relative to $<_2$. We will now describe the FGLM algorithm and give upper bounds on the space and time complexity.

An important characteristic of the ideal is the vector space dimension of the residue class ring obtained when factoring the polynomial ring R by the ideal I:

Definition 3.1.1. Let $R := F[x_1, \ldots, x_n]$. Then the F-space dimension of the ideal $I \subset R$ shall be denoted by $\dim(R/I)$.

The complexity of the FGLM algorithm hinges on two parameters of the input G: the number of variables of the polynomial ring R and the vector space dimension of the residue class ring $R/\mathfrak{I}$, where $\mathfrak{I}$ is the ideal generated by the Gröbner basis $G \subset R$. The following theorem [7] shows how this invariant of an ideal can be computed.

Theorem 3.1.1. *Let G be a Gröbner basis of the ideal $\mathfrak{I}$. Then*

$$\dim(R/\mathfrak{I}) = \#\left\{t \in \mathcal{T}(R) : HT(f) \nmid t \text{ for all } f \in G\right\} \tag{3.1}$$

This theorem turns out to have a trivial, but useful Corollary if the head terms of all polynomials are univariate:

Corollary 3.1.2. *Let $G = \{g_1, \ldots, g_k\}$ be a Gröbner basis for the ideal $\mathfrak{I} \subset F[x_1, \ldots, x_k]$ with head terms $x_1^{d_1}, \ldots, x_k^{d_k}$. Then $\dim(R/\mathfrak{I}) = \prod_{i=1}^{k} d_i$.*

Together with the following theorem this will later – in Chapter 4 – be used to give an upper bound on the time and space complexity of a Gröbner basis conversion for concrete instances of polynomial systems derived from block ciphers.

Theorem 3.1.3. *Let K be a finite field and $R = K[x_1, \ldots, x_k]$. Furthermore $G_1 \subset R$ is the Gröebner basis relative to a term order $<_1$ of an ideal $\mathfrak{I}$, and $d = dim(R/\mathfrak{I})$. We can then convert G_1 into a Gröbner basis G_2 relative to a term order $<_2$ in $O(kd^3)$ field operations.*

3.2 The F_4 Algorithm

The F_4 algorithm is an algorithm proposed by Jean-Charles Faugére to compute Gröbner bases more efficiently by employing a more powerful reduction algorithm. The algorithm uses Macaulay matrices to achieve this and builds on Lazard's ideas [76]. This also is the reason why these type of algorithms sometimes are referred to as Faugère-Lazard solvers.

Algorithm 10 presents the so-called "improved" version of the F_4 algorithm by Jean-Charles Faugère. Two improvements make it superior to the first algorithm – the "normal" version of F_4: it applies the Buchberger criteria and it tries to apply results from previous steps of the computation to simplify polynomials. It is recommended to use the Gebauer-Möller installation for the Update function implementing the Buchberger criteria. This function is given in Algorithm 14.

To understand the following algorithm, we first have to introduce a slightly different definition of a critical pair:

Definition 3.2.1. A critical pair of two polynomials $f, g \in \mathbb{F}[\mathcal{X}]$ for the algorithm F_4 is defined by the function $\mathsf{Pair}(f, g) = (\text{lcm}_{fg}, t_f, f, t_g, g)$ such that the following holds:

$$\text{lcm}(\mathsf{Pair}(f, g)) = \text{lcm}_{fg} = \text{HT}(t_f \cdot f) = \text{HT}(t_g \cdot g) = \text{lcm}(\text{HT}(f), \text{HT}(g))$$

with $\text{lcm}_{fg}, t_f, t_g \in \mathcal{T}(R)$.

We now define some properties of these critical pairs:

Definition 3.2.2. Let $p = (\text{lcm}_{fg}, t_f, f, t_g, g)$ be a critical pair. The degree of p is defined as $\deg(\text{lcm}_{fg})$, two projections are defined to work on the components of the pair: $\mathsf{Left}(p) = (t_f, f)$ and $\mathsf{Right}(p) = (t_g, g)$. Furthermore for convenience, for a tuple $(t, f) \in \mathcal{T}(\mathbb{F}[\mathcal{X}]) \times \mathbb{F}[\mathcal{X}]$ we define $\mathsf{mult}(t, f) = t \cdot f$.

Algorithm 10 F4Improved

Require: $P \subset \mathbb{F}[\mathcal{X}]$
Ensure: $G \subset \mathbb{F}[\mathcal{X}]$ is a Gröbner basis
  $(G, P, d) \leftarrow (\emptyset, \emptyset, 0)$
  while $F \neq \emptyset$ **do**
    $f \leftarrow \mathsf{first}(F)$
    $F \leftarrow F \setminus \{f\}$
    $(G, P) \leftarrow \mathsf{Update}(G, P, f)$
  end while
  while $P \neq \emptyset$ **do**
    $d \leftarrow d + 1$
    $P_d \leftarrow \mathsf{SelectPairs}(P)$
    $P \leftarrow P \setminus P_d$
    $L_d \leftarrow \mathsf{Left}(P_d) \cup \mathsf{Right}(P_d)$
    $(\tilde{F}_d^+, F_d) \leftarrow \mathsf{F4Reduction}(L_d, G, (F_i)_{d=1,\ldots,(d-1)})$
    for $h \in \tilde{F}_d^+$ **do**
      $(G, P) \leftarrow \mathsf{Update}(G, P, h)$
    end for
  end while

Comparing this algorithm to the Buchberger algorithm described in 1.3.2, we see similarities. As mentioned before, the Reduction function returns more than one polynomial. For all of these polynomials the Update function is executed, which checks the Buchberger criteria. Furthermore we notice that more state than in the original Buchberger algorithm is kept. The Reduction method returns subsets $F_d^+ \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$ and $F_d \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$ the second of which is preserved until the algorithm terminates. The counter d indicates the number of "steps" the F_4 algorithm has already executed. In step d, all of the previous F_i, are used in the reduction.

To understand the reduction algorithm, we first have to agree on the convention of identifying polynomials by their corresponding row vectors in the Macaulay matrix of the set of polynomials processed. The function $\mathsf{REF}(F, \leq)$ implicitly changes from the polynomial representation to the representation of the polynomials by their Macaulay matrix such that columns are ordered by the term order $\leq$, computes a row-echelon form of the Macaulay matrix representation and returns the result as a set of polynomials.

In practice however, this switch is only done once, when reading the input system. After this point, all of the polynomial manipulations are translated into operations on row vectors. The naïve method of switching back and forth between polynomial and matrix representations causes a slowdown.

Algorithm 11 F4Reduction

Require: $\begin{cases} L \subset_{fin} T \times R[\mathbb{F}[\mathcal{X}]] \\ G \subset_{fin} R[\mathbb{F}[\mathcal{X}]] \\ \mathcal{F} = (F_k)_{k=1,\ldots,(d-1)} \end{cases}$

Ensure: $(F_1, F_2) \subset R[\mathbb{F}[\mathcal{X}]] \times R[\mathbb{F}[\mathcal{X}]]$

$F \leftarrow \mathsf{SymbolicPreprocessing}(L, G, \mathcal{F})$

$\widetilde{\mathcal{F}} = \mathsf{REF}(F, <)$

$\widetilde{\mathcal{F}}^+ \leftarrow \left\{ f \in \widetilde{\mathcal{F}} : \mathrm{HT}(f) \notin \mathrm{HT}(F) \right\}$

The reduction algorithm in turn makes use of another sub-algorithm called "symbolic preprocessing". Symbolic preprocessing is the main step of F_4 that builds the matrix to be reduced. Three auxiliarly functions are used in this algorithm, the function $\mathsf{RandomPick}(S)$ does exactly what its names suggests, it simply picks a random element of a set S. The function $\mathsf{IsTopReducible}(m, G)$ checks whether m is top-reducible modulo G and the function $\mathsf{FindReductor}(m, G)$ finds an element of $f \in G$ and a term $m' \in \mathcal{T}(\mathbb{F}[\mathcal{X}])$ such that $m = m' \cdot \mathrm{HT}(f)$.

The goal of the Simplify is to replace rows representing the evaluated product $\mathsf{mult}(m, f)$ occurring in the matrix F by equivalent rows representing $\mathsf{mult}(m', f')$ such that $m' \leq m$. As input the function Simplify takes a non-evaluated tuple $(m', f) \in \mathcal{T}(\mathbb{F}[\mathcal{X}]) \times \mathbb{F}[\mathcal{X}]$ as well all previous F_i and recursively determines the simplification described. We note that Simplify actually computes a row-echelon form of the matrices F_i again. Therefore it would make sense to store the row-echelon forms that were already computed in reduction step. However, from a practical point of view, the overhead of storing the intermediate matrices is prohibitive. In the F_4 implementation contained in Xylirt, Simplify therefore is turned off and simply

Algorithm 12 SymbolicPreprocessing

Require: $\begin{cases} L \subset_{fin} T \times \mathbb{F}[\mathcal{X}] \\ G \subset_{fin} \mathbb{F}[\mathcal{X}] \\ \mathcal{F} = (F_k)_{k=1,\ldots,(d-1)} \end{cases}$

$F \leftarrow \{\mathsf{mult}(\mathsf{Simplify}(m, f, \mathcal{F} \in L\}$
$Done \leftarrow \mathrm{HT}(F)$
while $\mathcal{T}(F) \neq Done$ **do**
  $m \leftarrow \mathsf{RandomPick}(\mathcal{T}(F) \setminus Done)$
  if $\mathsf{IsTopReducible}(m, G)$ **then**
    $(m', f) \leftarrow \mathsf{FindReductor}(m, G)$
    $m \leftarrow m' \cdot \mathrm{HT}(f)$
    $F \leftarrow F \cup \{\mathsf{mult}(\mathsf{Simplify}(m', f, \mathcal{F})\}$
  end if
end while

returns the first two parameters passed to the function. This means that the matrices returned by the F4Reduction function only are used in the loop directly following the reduction step and can be discarded afterwards.

Algorithm 13 Simplify

Require: $\begin{cases} t \in \mathcal{T}(R[\underline{X}]) \\ f \in R[\underline{X}] \\ \mathcal{F} = (F_k)_{k=1,\ldots,(d-1)} \text{ with } F_k \subset_{fin} R[\underline{X}] \\ \widetilde{\mathcal{F}} = (\widetilde{F}_k)_{k=1,\ldots,(d-1)} \text{ with } \widetilde{F}_k \subset_{fin} R[\underline{X}] \end{cases}$

$U \leftarrow \mathsf{ListOfDivisors}(t)$
for $u \in U$ **do**
  for $j \in \{1, \ldots, d\}$ **do**
    if $uf \in F_j$ **then**
      $p \leftarrow \mathsf{FindP}(\widetilde{F}_j^+, f, u)$
      if $u \neq t$ **then**
        $\mathsf{Simplify}(\frac{t}{u}, p, \mathcal{F}, \widetilde{F})$
      end if
    end if
  end for
end for

3.2.1 The Gebauer-Moeller Installation

Algorithm 14 gives an algorithmic description of the so-called Gebauer-Moeller installation [55] implementing the Buchberger criteria. This is the

proposed update strategy that should be used in the improved F_4 algorithm. The algorithmic description is taken from [7].

Again, we first give the auxiliarly functions needed before we give the actual algorithm: The function DisjointHT(f, g) is defined to return true iff $gcd(\text{HT}(f), \text{HT}(g)) = 1$, LCMHT$(f, g)$ is defined as lcm$(\text{HT}(f), \text{HT}(g))$ and the function CT(g_1, h, S) returns true iff LCMHT$(h, g_2) \nmid$ LCMHT(h, g_1) for all $\{h, g_2\} \in S$. As usual when employing the Buchberger criteria, the normal strategy should be used as selection strategy. For F_4, the function SelectPairs now returns all critical pairs with the degree of their LCM being minimal.

3.3 On the Complexity of Gröbner Basis Computations

Meyer and Mayr gave worst case complexities for Gröbner basis computations in [84]. For the case of algebraically closed fields, the worst-case complexity can become doubly exponential in the number of variables. For the cryptanalytic case – where we only consider solutions in the ground field – the worst case complexity is single exponential though.

For the degrevlex order, exact complexity bounds are known for so-called regular sequences [76]. Regular sequences are defined as follows:

Definition 3.3.1 (Regular sequence)**.** Given a sequence of homogeneous polynomials $(f_1, \ldots, f_m) \in \mathbb{F}[\mathcal{X}]^m$ we call this sequence regular if for all $1 \leq i \leq m$ and an arbitrary $g \in \mathbb{F}[\mathcal{X}]$

$$gf_i \in \langle f_1, \ldots, f_{i-1} \rangle$$

implies that g also is in $\langle f_1, \ldots, f_{i-1} \rangle$. An affine sequence of polynomials $(f_1, \ldots, f_m) \in \mathbb{F}[\mathcal{X}]^m$ is said to be regular if $(H(f_1), \ldots, H(f_m))$ is a regular sequence.

An interesting property of regular sequences is that it can be proved that no reductions to zero occur in the F_5 algorithm if they are used as input [44].

We clearly see that the notion of regularity only works for sequences of polynomials where the number of polynomials does not exceed the number of variables. Henceforth the notion of regularity needs to be adapted for systems where this is not the case; we call the systems corresponding to these sequences "overdetermined systems". These systems are relevant in cryptanalysis. For these cases, Magali Bardet introduced the notion of semi-regularity in her Ph.D. thesis [5], which introduces a bound on the product of g and f_i. This bound is called the "degree of regularity" and is defined as follows:

Algorithm 14 Update

Input: $G_{\text{old}} \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$
Input: $B_{\text{old}} \subset_{\text{fin}} (\mathbb{F}[\mathcal{X}] \times \mathbb{F}[\mathcal{X}])$
Input: $h \in \mathbb{F}[\mathcal{X}], h \neq 0$
Output: $G_{\text{new}} \subset_{\text{fin}} \mathbb{F}[\mathcal{X}]$
Output: $B_{\text{new}} \subset_{\text{fin}} (\mathbb{F}[\mathcal{X}] \times \mathbb{F}[\mathcal{X}])$

$C \leftarrow \{\{h, g\} \,|g \in G_{\text{old}}\}$
$D \leftarrow \emptyset$
while $C \neq \emptyset$ **do**
  $\{h_1, g\} \leftarrow \mathsf{RandomPick}(C)$
  $C \leftarrow C \setminus \{h_1, g\}$
  if $\mathsf{DisjointHT}(h, g_1) \vee (\mathsf{CT}(h, g_1, g_2, C) \wedge \mathsf{CT}(h, g_1, g_2, D))$ **then**
    $D \leftarrow D \cup \{h, g_1\}$
  end if
end while
$E \leftarrow \emptyset$
while $D \neq \emptyset$ **do**
  $\{h, g\} \leftarrow \mathsf{RandomPick}(D)$
  if $\mathsf{DisjointHT}(h, g)$ **then**
    $E \leftarrow E \cup \{h, g\}$
  end if
end while
$B_{\text{new}} \leftarrow E$
while $B_{\text{old}} \neq \emptyset$ **do**
  $\{g_1, g_2\} \leftarrow \mathsf{RandomPick}(B_{\text{old}})$
  $B_{\text{old}} \leftarrow B_{\text{old}} \setminus \{\{g_1, g_2\}\}$
  if $(\mathrm{HT}(h) \nmid \mathsf{LCMHT}(g_1, g_2)) \vee (\mathsf{LCMHT}(g_1, h) = \mathsf{LCMHT}(g_1, g_2)) \vee$
  $(\mathsf{LCMHT}(g_2, h) = \mathsf{LCMHT}(g_1, g_2))$ **then**
    $B_{\text{new}} \leftarrow B_{\text{new}} \cup \{\{g_1, g_2\}\}$
  end if
end while
$G_{\text{new}} \leftarrow \{h\}$
while $G_{\text{old}} \neq \emptyset$ **do**
  $g \leftarrow \mathsf{RandomPick}(B_{\text{old}})$
  $G_{\text{old}} \leftarrow G_{\text{old}} \setminus \{g\}$
  if $\mathrm{HT}(h) \nmid \mathrm{HT}(g)$ **then**
    $G_{\text{new}} \leftarrow G_{\text{new}} \cup \{g\}$
  end if
end while

Definition 3.3.2 (degree of regularity)**.** Let $\mathcal{I} = \langle f_1, \ldots, f_m \rangle \subset \mathbb{F}[\mathcal{X}]$. The

degree of regularity of $\mathcal{I}$ is a function of $\mathcal{I}$ that is defined as

$$d_{\text{reg}}(\mathcal{I}) = \min\{d \geq 0 \mid \dim_{\mathbb{F}}(f \in \mathcal{I}, \deg(f) = d) = aaa\}$$

To go from regular sequences to semi-regular sequences, Definition 3.3.1 only needs to be changed slightly. We simply bound the degree of $g \cdot f_i$ below the degree of regularity for all f_i:

Definition 3.3.3 (semi-regular sequence)**.** Given a sequence of homogeneous polynomials $F = (f_1, \ldots, f_m) \in \mathbb{F}[\mathcal{X}]^m$ we call it semi-regular if for all $1 \leq i \leq m$ and an arbitrary $g \in \mathbb{F}[\mathcal{X}]$

$$gf_i \in \langle f_1, \ldots, f_{i-1}\rangle \text{ and } \deg(gf_i) < d_{\text{reg}}(F)$$

implies that g also is in $\langle f_1, \ldots, f_{i-1}\rangle$. An affine sequence of polynomials $(f_1, \ldots, f_m) \in \mathbb{F}[\mathcal{X}]^m$ is said to be semi-regular if $(H(f_1), \ldots, H(f_m))$ is a regular sequence.

Bardet, Faugére, Salvy and Yang later gave worst-case complexity estimates for so-called semi-regular systems over $GF(2)$ [4]. These were obtained by bounding the maximum degree of the polynomials and subsequently deriving the size of the matrices in the case of Lazard-Faugére solvers.

Chapter 4

Algebraic Approaches To Cryptanalysis

In this chapter we show how polynomial systems of equations can be used for analyzing the security of ciphers. We explain the concept of interpolation attacks, give an explicit construction for the polynomial representation of FLURRY and CURRY and show how the bit-level operations that are used in SMS4 can be embedded into the extension field $GF(2^8)$, yielding a cipher with a structurally "clean" representation over this field, ESMS4. Furthermore we give a high-level description of Gröbner basis attacks with minimal data complexity; for FLURRY and CURRY we present experimental results. We explain a method that can be used to avoid polynomial reductions completely when computing a Gröbner basis of certain ciphers and use it to obtain a zero-dimensional Gröbner basis for AES-128. For this Gröbner basis we analyze the impact that it has on the security of the cipher. Last but not least we show how a chosen-key attack on the cipher Cryptomeria can be used to recover its secret S-Box. This attack is a combination of differential and algebraic methods.

4.1 Interpolation Attacks on Block Ciphers

Jakobsen and Knudsen presented interpolation attacks in [64] as a reaction to ciphers using algebraically constructed S-Boxes such as those proposed by Nyberg [90]. In fact, interpolation attacks were the first demonstration of successful polynomial-based algebraic attacks against block ciphers. Interpolation attacks work by expressing the relationship between the plaintext and ciphertext for a fixed key as either one or as a vector of polynomials.

If the degree of these polynomials is low enough, the coefficients of the polynomials can be interpolated from a number of plaintext/ciphertext pairs. A key–dependent equivalent of the encryption or the decryption algorithm has then been determined. In [64] upper bounds on the data complex-

ity – the number of required pairs for known-plaintext interpolation attacks – are given for selected examples. In general this number increases exponentially with the degree of the polynomial function describing the S-Box, the number of rounds and the number of elements in the internal state, while for the attacks we present in the next section the data complexity remains a constant quantity.

Courtois later improved on the work of Jakobsen and Knudsen and introduced an attack called General Linear Cryptanalysis [30]. In the same paper he also gives several examples of insecure ciphers based on inversion based S-Boxes that resist differential and linear cryptanalysis. His approach and his goals are quite different from ours however.

4.2 Deriving Systems of Polynomial Equations

Multiple approaches can be used for obtaining a polynomial system of equations describing the cipher process: Depending on the elementary operations used in the block cipher, a suitable ground field is chosen. In most cases the equations are formulated over $GF(2)$ or over an extension fields of $GF(2)$; sometimes the cipher be seen as a restriction of a more general cipher, this is called an embedding. To practically express and manipulate the polynomials, the introduction of so-called intermediate state variables almost always is necessary. Without the introduction of these intermediate variables, the degree of the polynomials grows with each round, causing "expression swell": the number of terms in the polynomials to grow exponentially. This section describes different approaches by example: We look at round-based descriptions over extension fields for the experimental cipher FLURRY and CURRY as well as a way to obtain an embedded cipher description of the block cipher SMS4. By "embedded" we mean that the bit-level structure of the cipher is carried over into an extension field, causing the bit-level operations to become compatible with other algebraic operations such as inversion. Later, in Section 4.5 we also describe how to obtain polynomial systems over $GF(2)$ for ciphers involving modular addition in $\mathbb{Z}/\mathbb{Z}_{2^k}$

4.2.1 Polynomial Representation of FLURRY and CURRY

In the following we will detail how to obtain a system of polynomial equations that describes the transformation of a plaintext into a ciphertext block round by round using intermediate state variables. Please note that our description is slightly simplified. For the sake of legibility we have omitted the round key addition after the final round; for the experiments described in Section 4.3.1 the final key addition has of course been retained.

- FLURRY
  For Feistel ciphers the left half of the state in round e is identical to the right half of the state in round $e-1$, giving rise to the following mr trivial linear equations:

$$x_j^{(e)} + x_{j+m}^{(e-1)} = 0$$

  Each monomial S-Box of the cipher induces a polynomial equation of degree $\deg(f)$. Thus we get a total of mr non-linear equations of the form:

$$x_{m+j}^{(e)} + x_j^{(e-1)} + \sum_{l=1}^{m} d_{j,l} \cdot f\left(x_{m+l}^{(e-1)} + k_l^{(e-1)}\right) = 0$$

  with $1 \leq e \leq r,\ 1 \leq j \leq m$. When using the inversion S-Box the polynomial system is correct only with probability $\left(\frac{2^n-1}{2^n}\right)^{mr}$. The equations in this case are of a different form:

$$\left(x_j^{(e-1)} + x_{m+j}^{(e)}\right) \prod_{i=1}^{m} \left(x_{m+i}^{(e-1)} + k_i^{(e-1)}\right) + \sum_{l=1}^{m} d_{j,l} \prod_{\substack{i=1 \\ i \neq l}}^{m} \left(x_{m+i}^{(e-1)} + k_i^{(e-1)}\right) = 0$$

  The linear equations for the key schedule of FLURRY can be written as:

$$k_j^{(e)} + k_j^{(e-2)} + (\theta+1)^{et+j} + \sum_{l=1}^{m} d_{j,l} k_l^{(e-1)} = 0$$

  with $2 \leq e \leq r,\ 1 \leq j \leq m$.

- CURRY
  No trivial linear equations hold between intermediate state variables.

  Denote by $x_{(i,j)}^{(e)}$ the variable in row i, column j of the state in round e, analogously for $k_{(i,j)}^{(e)}$. Then for all rounds $e > 0$ the following equations hold with $1 \leq i, j \leq m$:

$$x_{i,j}^{(e)} + \sum_{l=1}^{m} d_{i,l} \cdot f\left(x_{j,l}^{(e-1)} + k_{j,l}^{(e-1)}\right) = 0$$

  Again for f_{-1} the non-linear equations look different:

$$x_{i,j}^{(e)} \prod_{u=1}^{m} \left(x_{j,u}^{(e-1)} + k_{j,u}^{(e-1)}\right) + \sum_{l=1}^{m} d_{i,l} \prod_{\substack{u=1 \\ u \neq l}}^{m} \left(x_{j,u}^{(e-1)} + k_{j,u}^{(e-1)}\right) = 0$$

  Using the above equations, the polynomial system also does not hold with probability one but with probability $\left(\frac{2^n-1}{2^n}\right)^{m^2 r}$.

The linear equations for the key schedule can be expressed as follows:

$$k_{i,j}^{(e)} + (\theta)^{e+(i-1)m+j} + \sum_{l=1}^{m} d_{i,l} k_{l,j}^{(e-1)} = 0$$

with $1 \leq e \leq r,\ 1 \leq i, j \leq m$.

The field polynomials will not be used in our system.

4.2.2 An Embedded Representation of SMS4

Similar to the embedding defined by Murphy and Robshaw for AES–128 [87], we can embed SMS4 into a more elegant and structured cipher ESMS4 in which all operations are performed over the finite field $GF(2^8)$. In this section we will show how this can be done. First note that the description we give is probabilistic, since we do not allow the inversion of the value 0 to occur. The overall number of S-Boxes in the cipher and key schedule is 256, henceforth the probability that an arbitrary plaintext can be encrypted under an arbitrary key without causing a zero inversion can be approximated by $\left(\frac{255}{256}\right)^{256} \approx 1/e \approx 36.7\%$.

First of all, let F denote the field ESMS4 will be defined over:

$$F = GF(2^8) = \frac{GF(2)[x]}{x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1} = GF(2)(\theta)$$

The state space, the key space and the message space of ESMS4 then are F^{128}, the round key space is F^{32}. In accordance with [87] we define a vector conjugate mapping ϕ that maps an element $a \in F$ to an 8-tuple $a' \in F^8$

$$\phi(a) = \left(a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7}\right)$$

and analogously maps a vector $A \in F^n$ to $A' \in F^{8n}$. The inverse of ϕ, $Im(\phi)$ shall be called extraction mapping. For a $GF(2)$-linear function L operating on a byte $b := (b_8, b_7, b_6, b_5, b_4, b_3, b_2, b_1)$ we obtain a F-linear function $\mathcal{L}$ that performs the equivalent operation on the vector $\phi(b)$ by first computing the coefficients $\beta_1, \ldots, \beta_8$ of the the linearized polynomial

$$\mathcal{L}(b) = \sum_{k=1}^{8} \beta_k a^{2^{k-1}}$$

and then computing the matrix $M_{\mathcal{L}} = (\alpha_{i,j})$ with $\alpha_{i,j} = \beta_{1+((j-i) \bmod 8)}^{2^{i-1}}$. The function $\mathcal{L}$ then is defined as $\mathcal{L} : F^8 \to F^8, v \to M_{\mathcal{L}} \cdot v$. We call $M_{\mathcal{L}}$ the *linearized polynomial matrix form* of L.

The S-Box Layer

The S-Box of SMS4 can be decomposed into the form $A \circ I \circ A$, with A an affine-linear function over $GF(2)$ [77]. Analogously, for ESMS4, the S-Box operation can be performed by $\mathcal{A} \circ \mathcal{I} \circ \mathcal{A}$, with $\mathcal{A}$ being an affine-linear transform over F and $\mathcal{I}$ being the componentwise inversion of elements on a vector $v \in F^8$. The linear part of $\mathcal{A}$ can be expressed by multiplication of the linearized polynomial matrix form $M_{\mathcal{A}} \in F^{8\times 8}$ of the linear part of A, whilst the constant can simply be embedded using ϕ. We define $\widetilde{C} = (\phi(C_1), \phi(C_1), \phi(C_1), \phi(C_1))$ and $\widetilde{A} = \text{Diag}_4(M_{\mathcal{A}})$

The Linear Transform λ

Let $P \in GF(2)^{32\times 32}$ be the permutation matrix such that for $v \in GF(2)^{32}$, the product $P \cdot v$ corresponds to a cyclic shift of elements of v by one position to the left. This matrix can be decomposed into the following form

$$P = \begin{pmatrix} M_1 & 0 & 0 & M_2 \\ M_2 & M_1 & 0 & 0 \\ 0 & M_2 & M_1 & 0 \\ 0 & 0 & M_2 & M_1 \end{pmatrix}, \quad M_1, M_2 \in GF(2)^{8\times 8}$$

By computing the linearized polynomial matrix forms for M_1 and M_2

$$\widetilde{M_1} = \mathcal{L}(M_1), \;\; \widetilde{M_2} = \mathcal{L}(M_2)$$

we obtain the following matrix that performs the equivalent action on a 32-tuple of elements representing 4 bytes of the state:

$$P = \begin{pmatrix} \widetilde{M_1} & 0 & 0 & \widetilde{M_2} \\ \widetilde{M_2} & \widetilde{M_1} & 0 & 0 \\ 0 & \widetilde{M_2} & \widetilde{M_1} & 0 \\ 0 & 0 & \widetilde{M_2} & \widetilde{M_1} \end{pmatrix}, \quad \widetilde{M_1}, \widetilde{M_2} \in F^{8\times 8}$$

Then the transformation λ is equivalent to the multiplication from the left with the matrix

$$\Lambda_1 = P^0 + P^2 + P^{10} + P^{18} + P^{24}$$

whilst for λ' the corresponding matrix is

$$\Lambda_2 = P^0 + P^{13} + P^{24}.$$

The Round Function

The round function of ESMS4 can be expressed as:

$$\begin{aligned} \widetilde{F} : F^{32} \times F^{32} &\rightarrow F^{32}, \\ (\widetilde{X}, \widetilde{K}) &\mapsto \Lambda_1 \cdot \left(\widetilde{A} \cdot \mathcal{I} \left(\widetilde{A} \cdot \left(\widetilde{X} + \widetilde{K} \right) + \widetilde{C} \right) + \widetilde{C} \right) \end{aligned}$$

The Key Schedule

The key generation function of ESMS4 is almost identical to the round function; merely Λ_1 needs to be replaced by Λ_2.

Impact

The existence of the embedding arises from SMS4 solely using $GF(2)$-linear operations and inversions over $GF(2^8)$. Since the number of S-Boxes per cipher round for ESMS4 is only a quarter of that of BES–128, we expect ESMS4 to be more amenable to experimenting with algebraic attacks without having to resort to scaling down the field or block size.

4.3 Gröbner Basis Attacks with Minimal Data Complexity

This section describes the general concept of a Gröbner Basis Attack. We show that for several ciphers, notably some instances of Flurry and Curry, Gröbner Bases can be computed with minimal computational effort. This then reduces the problem of finding the key to a Gröbner basis conversion problem, which can be solved with a Gröbner basis conversion algorithm, e.g. FGLM.

Estimating the time and space complexity of Gröbner basis algorithms is no easy feat. For polynomial systems induced by block ciphers, theoretical results estimating the performance of Gröbner basis algorithms were previously unknown. We therefore carried out experiments to study the resistance of FLURRY and CURRY against Gröbner Basis attacks. Results of these experiments are presented and analysed in section 4.3.1.

The Gröbner basis attack we have successfully used against instances of FLURRY and CURRY yields a key recovery. This attack has minimal data complexity, i.e. a single pair of plaintext and corresponding ciphertext will suffice unless an inversion-based S-Box is used. Another pair is used to verify key candidates. The attack works as follows:

1. Set up a polynomial system $\mathcal{P} = \{p_i = 0\}$ for the cipher in question with $p_i \in R$ as described in Section 4.2.1. The system $\mathcal{P}$ consists of both cipher and key schedule equations.

2. Request a plaintext/ciphertext pair $((P_1, \dots P_t), (C_1, \dots, C_t))$. This gives rise to the following additional system of linear equations $\mathcal{G} = \{g_i = 0\}$: Let $\mathfrak{I}$ be the ideal generated by the set of polynomials $\mathcal{L} = (\bigcup_i \{p_i\}) \cup (\bigcup_i \{g_i\})$. We call this ideal the *key recovery ideal*.

3. Compute a degree-reverse lexicographic Gröbner basis G_{DRL} of $\mathfrak{I}$. For ciphers using a multiplicative inverse as S-Box function, the system may be inconsistent, resulting in $G_{DRL} = 1$.
4. If $G_{DRL} = 1$ go to Step 2, otherwise proceed.
5. Use a Gröbner basis conversion algorithm to obtain a lexicographical Gröbner basis G_{lex} from G_{DRL}. The variable ordering should be such that the key variables of the first round are the least elements.
6. Compute the variety Z of $\mathfrak{I}$ using the Gröbner basis G_{lex}.
7. Request another plaintext/ciphertext pair (P', C').
8. Try all elements $k \in Z$ as key candidates to encrypt P'. If k does not encrypt P' to C', remove k from Z, otherwise retain.
9. If Z contains more than one element, go to step 7.
10. Terminate

Considerable complexity is hidden in step 6. To compute the variety of an ideal using a lexicographical Gröbner basis, we need to successively eliminate variables by computing zeroes of univariate polynomials and back-substituting results. The complexity of this step depends on the number of solutions of the polynomial system (zeroes of the ideal) and the complexity of the algorithm for finding roots of univariate polynomials. The best algorithm for factoring polynomials is due to Kaltofen and Shoup [67] and has a complexity of $O(d^{1.815}n)$ field operations, where d is the degree of the polynomial. This degree if bounded by $2^n - 1$. The number zeroes is equivalent to the number of distinct keys encrypting the plaintext to a ciphertext. In general we can expect this number to be small.

4.3.1 Experimental Results

We have performed experiments to analyze the resistance of FLURRY and CURRY using the computer algebra system MAGMA [108], version 2.11-8, on an AMD Athlon 64 3200+ equipped with 1024 Megabytes of RAM running Linux. MAGMA implements Faugére's F4 algorithm [43] and is widely considered the best publicly available tool for computing Gröbner bases. We have chosen n and m such that the ciphers evaluated are 128-bit block ciphers.

Table 4.1 lists a number of instantiations of FLURRY and CURRY ciphers for which we were able to successfully recover the secret key; the FLURRY ciphers listed with 6 and more rounds are resistant to linear and differential cryptanalysis. We see that ciphers with inversion-based S-boxes are easier

to break than ciphers which use a monomial S-box, even if the monomial is of very low degree. For monomial S-Boxes, we clearly see that the degree of the function used in the S-Box influences the complexity of the attack. The influence of the degree however is smaller than the influence of an additional two rounds.

Unfortunately we were unable to determine an a priori indicator for selecting the most efficient Gröbner basis conversion algorithm – in some cases FGLM was faster, in other cases the Gröbner walk; the same holds for the memory consumption.

Table 4.1: Experimental results obtained with MAGMA (from [21])

cipher	conversion	CPU time	memory used
FLURRY$(64, 1, 4, f_{-1}, I_1)$	Walk	0.011 s	3.48 MBytes
FLURRY$(64, 1, 4, f_{-1}, I_1)$	FGLM	0.011 s	3.48 MBytes
FLURRY$(64, 1, 4, f_3, I_1)$	Walk	0.04 s	3.48 MBytes
FLURRY$(64, 1, 4, f_3, I_1)$	FGLM	0.029 s	3.58 MBytes
FLURRY$(64, 1, 4, f_5, I_1)$	Walk	1.28 s	3.97 MBytes
FLURRY$(64, 1, 4, f_5, I_1)$	FGLM	2.3 s	6.36 MBytes
FLURRY$(64, 1, 4, f_7, I_1)$	Walk	13.61 s	6.22 MBytes
FLURRY$(64, 1, 4, f_7, I_1)$	FGLM	82.62 s	33.4 MBytes
FLURRY$(64, 1, 6, f_{-1}, I_1)$	Walk	0.15 s	3.58 MBytes
FLURRY$(64, 1, 6, f_{-1}, I_1)$	FGLM	0.059 s	3.58 MBytes
FLURRY$(64, 1, 6, f_3, I_1)$	Walk	59.91 s	10.63 MBytes
FLURRY$(64, 1, 6, f_3, I_1)$	FGLM	145.08 s	193.24 MBytes
FLURRY$(64, 1, 8, f_{-1}, I_1)$	Walk	3.43 s	4.51 MBytes
FLURRY$(64, 1, 8, f_{-1}, I_1)$	FGLM	1.46 s	4.46 MBytes
FLURRY$(64, 1, 10, f_{-1}, I_1)$	Walk	115.44 s	14.74 MBytes
FLURRY$(64, 1, 10, f_{-1}, I_1)$	FGLM	60.61 s	12.39 MBytes
FLURRY$(64, 1, 12, f_{-1}, I_1)$	Walk	4194.28 s	99.97 MBytes
FLURRY$(64, 1, 12, f_{-1}, I_1)$	FGLM	2064 s	142.90 MBytes
FLURRY$(32, 2, 4, f_{-1}, D_2)$	Walk	216.53 s	25.58 MBytes
FLURRY$(32, 2, 4, f_{-1}, D_2)$	FGLM	65.78 s	41.62 MBytes
FLURRY$(16, 4, 2, f_{-1}, D_4)$	Walk	264 s	37.13 MBytes
FLURRY$(16, 4, 2, f_{-1}, D_4)$	FGLM	26.119 s	18.56 MBytes
CURRY$(32, 2, 3, f_{-1}, D_2)$	Walk	1750.87 sec	138.77 MBytes
CURRY$(32, 2, 3, f_{-1}, D_2)$	FGLM	3676.26 sec	107.54 MBytes

4.3.2 Gröbner Bases without Polynomial Reductions

In certain situations one can determine whether a set of polynomials forms a Gröbner basis without computing normal forms. This is an interesting observation, as it allows to obtain a Gröbner basis for the polynomial systems

of some instances of FLURRY and CURRY instantaneously. In the following let be $G \subset R$ be a finite set of polynomials with $0 \neq G$.

The first Buchberger criterion (1.3.3) together with the following theorem given in [34] allows us to decide whether a sequence of polynomials is a Gröbner basis. This is done by simply looking at their head terms:

Theorem 4.3.1. *The set G is a Gröbner basis iff $spol(f,g) \rightarrow_G 0$ for all $f, g \in G$ with $f \neq g$.*

The following lemma results:

Lemma 4.3.2. *Let G be a set of polynomials and $H = \{HT(f) : f \in G\}$. If all elements in H are pairwise prime, then G is a Gröbner basis.*

When using polynomial S-boxes, this enables us to compute a degree-reverse lexicographic Gröbner bases of the key-recovery ideals of FLURRY and CURRY without performing polynomial reductions; the head terms of all polynomials of $\mathfrak{I}$ are univariate. For each polynomial of round e, either a power of a state variable of the preceeding round or a power of a key variable of the current round occur as head term. Some head terms however occur more than once.

By using an appropriate variable order we can force the set of head terms of each round to be disjunct from the set of head terms of all other rounds:

- CURRY
  For better legibility, we identify $x_{i,j}^{(e)}$ with $x_{et+im+j}$ and $k_{i,j}^{(e)}$ with $k_{et+im+j}$. We then fix the following variable order:

$$\underbrace{x_0 < \ldots < x_{t-1}}_{\text{plaintext variables}} < \underbrace{x_{tr} < \ldots < x_{t(r+1)-1}}_{\text{ciphertext variables}} < \underbrace{k_0 < \ldots < k_{t(r+1)-1}}_{\text{key variables}} < \underbrace{x_t < \ldots < x_{tr-1}}_{\text{internal state variables}}$$

- FLURRY
  Again we decrease the number of indexes: we identify $x_i^{(e)}$ with x_{et+i} and $k_i^{(e)}$ with k_{et+i}. We then fix the following variable order:

$$\underbrace{x_0 < \ldots < x_{t-1}}_{\text{plaintext variables}} < \underbrace{x_{tr} < \ldots < x_{(t+1)r-1}}_{\text{ciphertext variables}} < \underbrace{x_{t(r-1)+m} < \ldots < x_{tr-1}}_{\text{state variables of the right half of the second last round}} <$$

$$\underbrace{k_0 < \ldots < k_{m-1}}_{\text{key variables of the first round}} < \underbrace{k_{m(r-1)} < \ldots < k_{mr-1}}_{\text{key variables of round } r} <$$

$$\underbrace{k_m < \ldots < k_{m(r-1)-1} < k_{mr} < \ldots < k_{m(r+2)-1}}_{\text{remaining key variables}} < \underbrace{x_t < \ldots < x_{t(r-1)+m-1}}_{\text{remaining state variables}}$$

To make the following linear transformation easier to describe we use a vectorial representation for FLURRY and a matrix representation for CURRY. The entries in the vector and matrix of each round are the left-hand side polynomials of the nonlinear cipher equations.

We can multiply the vectors respectively matrices of all rounds by D^{-1} to obtain pairwise prime head terms within each and across rounds. For CURRY this is sufficient. For FLURRY we also need to adjust the key schedule equations. The nonlinear polynomials of the first and the last round have powers of key variables as head terms. These key variables are of the first and the last round respectively. For the first round this poses no problem. However for the last round the key schedule polynomials that produce the last round key have the same head terms. Thus we rewrite the key schedule equations. We express all round keys except for the last round key as a linear combination of the first two round keys. Then we write the second round key as a linear combination of the first and the last round key. This results in all head terms being pairwise prime. In order for this to work for FLURRY, the order of the matrix used in the key schedule needs to be greater than the number of rounds.

We have shown how to make the head terms of all polynomials pairwise prime. Hence by Theorem 4.3.1, we have obtained a Gröbner basis. This strategy however does not work FLURRY and CURRY instances with inversion S-Boxes, as the head terms in these cases are never univariate.

By using the result on the complexity of FGLM, Theorem 3.1.3, we are in a position to give theoretical upper bounds on the time complexity required to break FLURRY and CURRY instances with polynomial S-Boxes.

We conjecture the constant factor in Theorem 3.1.3 to be approximately one cipher operation. For the space complexity of the algorithm, no bound is given in the original FGLM paper. We note that the dominant memory requirement of the FGLM algorithm is a $d \times kd$ matrix over F. Thus the memory usage of the algorithm is upper bounded by $\lceil (kd^2 n)/8 \rceil + o(1)$ bytes.

This allows us to estimate the maximum resistance of FLURRY and CURRY ciphers with polynomial S-Boxes against Gröbner basis attacks (see Table 4.2). Note that for the CURRY cipher we need to use a bijective S-Box in the round function; the lowest degree S-Box function that is bijective is f_7.

Table 4.2: Upper bounds on the complexity of breaking 128-bit FLURRY and CURRY ciphers with FGLM

cipher	n	$\dim(R/I)$	# of ops	memory (bytes)
FLURRY$(32, 2, 4, f_3, D_2)$	8	$3^8 \approx 2^{12.68}$	$O(2^{41.0})$	$2^{30.4}$
FLURRY$(32, 2, 4, f_5, D_2)$	8	$5^8 \approx 2^{18.58}$	$O(2^{58.7})$	$2^{42.2}$
FLURRY$(32, 2, 4, f_7, D_2)$	8	$7^8 \approx 2^{22.46}$	$O(2^{70.4})$	$2^{49.9}$
FLURRY$(32, 2, 6, f_3, D_2)$	12	$3^{12} \approx 2^{19.02}$	$O(2^{60.6})$	$2^{43.2}$
FLURRY$(32, 2, 6, f_5, D_2)$	12	$5^{12} \approx 2^{27.86}$	$O(2^{87.2})$	$2^{61.3}$
FLURRY$(32, 2, 6, f_7, D_2)$	12	$7^{12} \approx 2^{33.69}$	$O(2^{104.7})$	$2^{73.0}$
FLURRY$(32, 2, 8, f_3, D_2)$	16	$3^{16} \approx 2^{25.36}$	$O(2^{80.0})$	$2^{56.7}$
FLURRY$(32, 2, 8, f_5, D_2)$	16	$5^{16} \approx 2^{37.15}$	$O(2^{115.5})$	$2^{80.3}$
FLURRY$(32, 2, 8, f_7, D_2)$	16	$7^{16} \approx 2^{44.92}$	$O(2^{138.8})$	$2^{95.8}$
FLURRY$(16, 4, 4, f_3, D_2)$	16	$3^{16} \approx 2^{25.36}$	$O(2^{80.0})$	$2^{55.7}$
FLURRY$(16, 4, 4, f_5, D_2)$	16	$5^{16} \approx 2^{37.15}$	$O(2^{115.5})$	$2^{79.3}$
FLURRY$(16, 4, 4, f_7, D_2)$	16	$7^{16} \approx 2^{44.92}$	$O(2^{138.8})$	$2^{94.8}$
CURRY$(32, 2, 3, f_7, D_2)$	12	$7^{12} \approx 2^{33.69}$	$O(2^{104.6})$	$2^{73.0}$

Example 4.3.1. The following sequence of polynomials G for the polynomial representation of FLURRY $(32, 2, 4, f_3, D_2)$ together with polynomials for the plaintext and the ciphertext is a degree-reverse lexicographic Gröbner basis with the following variable ordering:
$x_0 < x_1 < x_2 < x_3 < x_{16} < x_{17} < x_{18} < x_{19} < x_{14} < x_{15} < k_0 < k_1 < k_6 < k_7 < k_2 < k_3 < k_4 < k_5 < k_8 < k_9 < k_{10} < k_{11} < x_4 < x_5 < x_6 < x_7 < x_8 < x_9 < x_{10} < x_{11} < x_{12} < x_{13}$

$G = \{$

plaintext:

$$x_0 + \theta^{31} + \theta^{29} + \theta^{27} + \theta^{24} + \theta^{22} + \theta^{21} + \theta^{19} + \theta^{13} + \theta^{11} + \theta^8 + \theta^7 + \theta^6 + \theta^4 + 1$$
$$x_1 + \theta^{31} + \theta^{30} + \theta^{29} + \theta^{22} + \theta^{21} + \theta^{15} + \theta^{14} + \theta^{11} + \theta^{10} + \theta^7 + \theta^6 + \theta^5 + \theta^3 + \theta$$
$$x_2 + \theta^{26} + \theta^{25} + \theta^{24} + \theta^{21} + \theta^{19} + \theta^{18} + \theta^{16} + \theta^{14} + \theta^8 + \theta^7 + \theta^6 + \theta^4 + \theta + 1$$
$$x_3 + \theta^{27} + \theta^{26} + \theta^{24} + \theta^{21} + \theta^{17} + \theta^{15} + \theta^{13} + \theta^{11} + \theta^9 + \theta^6 + \theta^4 + \theta$$

ciphertext:

$$x_{16} + \theta^{31} + \theta^{29} + \theta^{21} + \theta^{19} + \theta^{18} + \theta^{16} + \theta^{15} + \theta^{14} + \theta^{12} + \theta^4 + 1$$
$$x_{17} + \theta^{24} + \theta^{21} + \theta^{20} + \theta^{18} + \theta^{16} + \theta^{13} + \theta^{10} + \theta^9 + \theta^8 + \theta^6 + \theta^5 + \theta^3 + \theta + 1$$
$$x_{18} + \theta^{29} + \theta^{25} + \theta^{21} + \theta^{20} + \theta^{19} + \theta^{13} + \theta^{10} + \theta^9 + \theta^8 + \theta^7 + \theta^6 + \theta^5 + \theta^3$$
$$x_{19} + \theta^{29} + \theta^{27} + \theta^{26} + \theta^{20} + \theta^{13} + \theta^{10} + \theta^8 + \theta^5 + \theta^2$$

round 1:

$$x_4 + x_2$$
$$x_5 + x_3$$
$$k_0^3 + k_0^2 x_2 + k_0 x_2^2 + x_2^3 + C_1 x_7 + C_1 x_6 + C_1 x_1 + C_1 x_0$$
$$k_1^3 + k_1^2 x_3 + k_1 x_3^2 + x_3^3 + C_2 x_7 + C_1 x_6 + C_2 x_1 + C_1 x_0$$

round 2:

$$x_8 + x_6$$
$$x_9 + x_7$$
$$x_6^3 + x_6^2 k_2 + x_6 k_2^2 + k_2^3 + C_1 x_{11} + C_1 x_{10} + C_1 x_5 + C_1 x_4$$
$$x_7^3 + x_7^2 k_3 + x_7 k_3^2 + k_3^3 + C_2 x_{11} + C_1 x_{10} + C_2 x_5 + C_1 x_4$$

round 3:

$$x_{12} + x_{10}$$
$$x_{13} + x_{11}$$
$$x_{10}^3 + x_{10}^2 k_4 + x_{10} k_4^2 + k_4^3 + C_1 x_9 + C_1 x_8 + C_1 k_9 + C_1 k_8 + C_1 x_{15} + C_1 x_{14}$$
$$x_{11}^3 + x_{11}^2 k_5 + x_{11} k_5^2 + k_5^3 + C_2 x_9 + C_1 x_8 + C_2 k_9 + C_1 k_8 + C_2 x_{15} + C_1 x_{14}$$

round 4:

$$x_{14} + x_{16}$$
$$x_{15} + x_{17}$$
$$k_6^3 + k_6^2 x_{14} + k_6 x_{14}^2 + x_{14}^3 + C_1 x_{13} + C_1 x_{12} + C_1 k_{11} + C_1 k_{10} + C_1 x_{19} + C_1 x_{18}$$
$$k_7^3 + k_7^2 x_{15} + k_7 x_{15}^2 + x_{15}^3 + C_2 x_{13} + C_1 x_{12} + C_2 k_{11} + C_1 k_{10} + C_2 x_{19} + C_1 x_{18}$$

key expansion:

$$k_{11} + \theta^2 k_7 + (\theta^2 + \theta + 1) k_1 + \theta k_0 + \theta^4 + \theta^2$$
$$k_{10} + \theta^2 k_6 + \theta k_1 + k_0 + \theta^3 + \theta$$
$$k_9 + (\theta^2 + \theta) k_7 + (\theta + 1) k_6 + \theta^2 k_1 + (\theta + 1) k_0 + \theta^6 + \theta^5 + \theta^3 + 1$$
$$k_8 + (\theta + 1) k_7 + (\theta + 1) k_6 + (\theta + 1) k_1 + k_0 + \theta^5 + \theta^3$$
$$k_5 + (\theta^2 + \theta + 1) k_7 + \theta k_6 + \theta^2 k_1 + (\theta + 1) k_0 + \theta^6 + \theta^4 + \theta^3 + \theta$$
$$k_4 + \theta k_7 + k_6 + (\theta + 1) k_1 + k_0 + \theta^5 + \theta^4 + \theta^3 + 1$$
$$k_3 + \theta^2 k_7 + (\theta + 1) k_6 + (\theta^2 + \theta + 1) k_1 + \theta k_0 + \theta^6 + \theta^5 + \theta^4 + \theta$$
$$k_2 + (\theta + 1) k_7 + k_6 + \theta k_1 + k_0 + \theta^5 + \theta^2 + \theta + 1$$

$\}$

with $C_1 = (\theta + 1)^{-1}$ and $C_2 = 1 + (\theta + 1)^{-1}$

4.4 A Gröbner Basis for AES-128

In this section we will describe how to obtain a Gröbner basis for AES-128. These results have been published in "Selected and Revised Papers, Fast Software Encryption 2006".

In the following we restrict ourselves to AES-128, i.e. Rijndael with a block and key size of 128 bits. We will deviate from the standard representation by using a column vector instead of a matrix for the internal state and the round keys. The elements in the column vector are identified with the elements of the matrix in a column-wise fashion by the following map:

$$\varphi : F^{4\times 4} \to F^{16}, \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \mapsto (s_{0,0}, s_{1,0}, \ldots, s_{0,1}, s_{1,1}, \ldots)^T \tag{4.1}$$

The 16×16 matrix P is defined to be the permutation matrix that expresses the exchange of elements in the column vector that is equivalent to transposing the state matrix.

The above notation allows us to combine the diffusion performed by the MixColumns and ShiftRows operations into a single matrix multiplication.

Let $x_{i,j}$ denote the variable referring to the ith component of the state vector after the jth round execution. By this definition the variables $x_{i,0}$ are called *plaintext variables*, correspondingly $x_{i,10}$ are called *ciphertext variables*. All other variables $x_{i,j}$ are called *intermediate state variables*; variables $k_{i,j}$ are called *key variables*. We will also refer to $k_{i,0}$ as *cipher key variables*.

The field F is the finite field $GF(2^8)$ as defined for Rijndael. The polynomial ring R is defined as

$$R := F[x_{i,j}, k_{i,j} : \{0 \leq i \leq 15, 0 \leq j \leq 10\}]$$

4.4.1 The S-Box

The S-Box used in Rijndael can be interpolated as a sparse polynomial over F:

$$\begin{aligned} \sigma : F \to F, \quad x \quad \mapsto \quad & \mathtt{05}x^{254} + \mathtt{09}x^{253} + \mathtt{F9}x^{251} + \mathtt{25}x^{247} + \mathtt{F4}x^{239} + \\ & \mathtt{B5}x^{223} + \mathtt{B9}x^{191} + \mathtt{8F}x^{127} + \mathtt{63} \end{aligned} \tag{4.2}$$

whilst the interpolation polynomial of the inverse S-Box

$$\sigma^{-1} : F \to F, \quad x \mapsto \sum_{i=0}^{254} c_i x^i \tag{4.3}$$

is dense.

4.4.2 The Linear Transformation

The linear transformation of AES consists of two operations, ShiftRows and MixColumns. We can perform the linear transform by multiplying the state column vector with a 16×16-matrix D from the left. In the following, we calculate D; however at the start of each round we apply the transposition matrix P since it makes expressing the operations as matrices easier. At the end we multiply with the matrix P to undo the initial transposition.

A matrix that shifts the elements of a 1×4 row vector cyclically by an offset t is of the following form:

$$D_{\mathrm{SR_t}} = \left(\Delta_{i,(j-t) \bmod 4}\right) \in F^{4\times 4} \tag{4.4}$$

where $\Delta_{i,j}$ is the Kronecker delta. The ShiftRows operation is equivalent to multiplying by the matrix D_{SR}:

$$D_{\mathrm{SR}} = \begin{pmatrix} D_{\mathrm{SR}_0} & 0 & 0 & 0 \\ 0 & D_{\mathrm{SR}_1} & 0 & 0 \\ 0 & 0 & D_{\mathrm{SR}_2} & 0 \\ 0 & 0 & 0 & D_{\mathrm{SR}_3} \end{pmatrix} \in F^{16\times 16} \tag{4.5}$$

The MixColumns operation is applied to each row of the internal state. We use the matrix D_{MC} to transform the column vector equivalently:

$$D_{\mathrm{MC}} = \begin{pmatrix} \mathtt{02} & \mathtt{03} & \mathtt{01} & \mathtt{01} \\ \mathtt{01} & \mathtt{02} & \mathtt{03} & \mathtt{01} \\ \mathtt{01} & \mathtt{01} & \mathtt{02} & \mathtt{03} \\ \mathtt{03} & \mathtt{01} & \mathtt{01} & \mathtt{02} \end{pmatrix} \otimes I_4 \in F^{16\times 16} \tag{4.6}$$

where $\otimes$ denotes the tensor product. Concatenation of the two operations in the diffusion layer is achieved by multiplying the above matrices, yielding the matrix D:

$$D = P \cdot D_{\mathrm{MC}} \cdot D_{\mathrm{SR}} \cdot P \tag{4.7}$$

The diffusion layer of the last round is missing the MixColumns transformation; it will be described by the matrix $\tilde{D}$:

$$\tilde{D} = P \cdot D_{\mathrm{SR}} \cdot P \tag{4.8}$$

This enables us to obtain the following vectorial representation of a system of 16 polynomial equations that holds for rounds $1 \leq j \leq 9$ of the cipher:

$$\begin{pmatrix} \sigma(x_{0,(j-1)} + k_{0,(j-1)}) \\ \vdots \\ \sigma(x_{15,(j-1)} + k_{15,(j-1)}) \end{pmatrix} + D^{-1} \begin{pmatrix} x_{0,j} \\ \vdots \\ x_{15,j} \end{pmatrix} = 0 \tag{4.9}$$

For the last round we need to take the simplified diffusion layer and the final key addition into account:

$$\begin{pmatrix} \sigma\left(x_{0,9} + k_{0,9}\right) \\ \vdots \\ \sigma\left(x_{15,9} + k_{15,9}\right) \end{pmatrix} + \tilde{D}^{-1} \begin{pmatrix} x_{0,10} + k_{0,10} \\ \vdots \\ x_{15,10} + k_{15,10} \end{pmatrix} = 0 \tag{4.10}$$

Choosing any degree lexicographical term order, either a term $x_{i,j}^{254}$ or a term $k_{i,j}^{254}$ occurs as head term of each polynomial. We take note that none of the head terms is a power of a plaintext nor of a ciphertext variable. Moreover all of the head terms are pairwise prime. The variable order chosen will influence whether the head term is a power of a key variable or of an intermediate state variable.

4.4.3 The Key Schedule

In order to obtain a Gröbner basis of both the cipher and the key scheduling polynomials, we need to set up the key scheduling in a slightly different way. Usually, the key scheduling expresses the elements of the round subkey of round $1 \leq j \leq 10$ as a vector of polynomials in the key variables of the previous round as follows:

$$\begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \\ k_{4,j} \\ \vdots \\ k_{15,j} \end{pmatrix} = \begin{pmatrix} k_{0,j-1} \\ k_{1,j-1} \\ k_{2,j-1} \\ k_{3,j-1} \\ k_{4,j-1} \\ \vdots \\ k_{15,j-1} \end{pmatrix} + \begin{pmatrix} \sigma(k_{15,j-1}) \\ \sigma(k_{12,j-1}) \\ \sigma(k_{13,j-1}) \\ \sigma(k_{14,j-1}) \\ k_{0,j} \\ \vdots \\ k_{11,j} \end{pmatrix} + \begin{pmatrix} \gamma_{j-1} \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4.11}$$

where the $\gamma_0, \ldots, \gamma_9$ are the round constants. To make all head terms pairwise prime (see also Section 4.4.4 on the term order chosen), we have to proceed in reverse order:

$$\begin{pmatrix} \sigma^{-1}(k_{0,j} + k_{0,j-1} + \gamma_{j-1}) \\ \sigma^{-1}(k_{1,j} + k_{1,j-1}) \\ \sigma^{-1}(k_{2,j} + k_{2,j-1}) \\ \sigma^{-1}(k_{3,j} + k_{3,j-1}) \\ k_{4,j} + k_{4,j-1} \\ \vdots \\ k_{15,j} + k_{15,j-1} \end{pmatrix} + \begin{pmatrix} k_{15,j-1} \\ k_{12,j-1} \\ k_{13,j-1} \\ k_{14,j-1} \\ k_{0,j} \\ \vdots \\ k_{11,j} \end{pmatrix} = 0 \tag{4.12}$$

4.4.4 Choosing a Suitable Variable Order

The plaintext and ciphertext polynomials simply are of the form

$$x_{i,0} + p_i \qquad p_i \in F, 0 \leq i \leq 15 \tag{4.13}$$

respectively

$$x_{i,0} + c_i \qquad c_i \in F, 0 \leq i \leq 15. \tag{4.14}$$

Let $\mathcal{A}$ be the union of the left-hand side of equations (4.9), (4.10) and (4.12) for all rounds $1 \leq j \leq 10$ as well as the plaintext and ciphertext polynomials. Ordering the variables as follows makes all head terms pairwise prime:

1. plaintext variables: $x_{0,0} < \ldots < x_{15,0}$
2. ciphertext variables: $x_{0,10} < \ldots < x_{15,10}$
3. key variables of all rounds in natural order: $k_{0,0} < k_{1,0} < \ldots < k_{15,10}$
4. intermediate state variables in their natural order

The degree lexicographical term order with the above variable order will be in the following be referred to as $<_{\mathcal{A}}$. By Theorem 4.3.2, the set of polynomials $\mathcal{A}$ is a Gröbner basis relative to this term order! Moreover, checking Lemma 1.3.2 we verify that this ideal is zero-dimensional.

4.4.5 Impact Analysis

In the previous section we have shown how to obtain a zero-dimensional Gröbner basis $\mathcal{A}$ for AES-128. In this section we explore the cryptanalytic impact of this finding. To this end, we investigate the complexity of a Gröbner basis conversion algorithm, find an invariant under the elimination of variables and explain why the naïve way of applying the ideal membership test does not work for guessing parts of the round key.

Complexity of Gröbner Basis Conversions

An obvious question is whether the Gröbner basis we have computed in the previous section can be efficiently converted to a different, more suitable order, i.e. a lexicographical order or an elimination order [6].

Two algorithms and variations of them are known for performing Gröbner basis conversions, the FGLM algorithm and the Gröbner Walk [26]. Since we have established that $\mathcal{A}$ is zero-dimensional, we are in a position to use FGLM and give an estimate for its time complexity below.

Since the head terms of the polynomials in the Gröbner basis are pairwise prime and univariate, Corollary 3.1.2 together with Theorem 3.1.3 are sufficient to give a bound on the complexity of the Gröbner basis conversion using FGLM. We conclude that the vector space dimension of the ideal generated by the Gröbner basis $\mathcal{A}$ is far too big for the FGLM algorithm be useful for cryptanalytic purposes in this case:

$$dim(R/\mathcal{A}) = 254^{200} \approx 2^{1598} \tag{4.15}$$

For the Gröbner Walk, the running time strongly depends on the source and the target term order. It is an open problem to give bounds on the time and space complexity for this algorithm. The only bounds known are local bounds, namely for adjacent term orders, due to Kalkbrener [66].

Elimination of Variables

In this section we establish that the dimension of the vector space of the ideal remains invariant when eliminating certain variables. We first prove the following more general statement:

Proposition 4.4.1. *Let I' be a zero-dimensional ideal of $R' := F[x_1, \ldots, x_n]$, I an ideal of $R := R'[x_{n+1}]$ and $I' = I \cap R'$. Then $\dim R/I = \dim R'/I'$ iff there exists a polynomial $g \in R'$ such that $x_{n+1} + g \in I$.*

Proof. W.l.o.g. we fix a lexicographical term ordering such that x_{n+1} is the greatest variable. Let RT(I) and RT(I') be defined as follows:

$$\begin{aligned} \text{RT}(I) &= \{t \in \mathcal{T}(R) : s \nmid t \text{ for all } s \in \text{HT}(I)\} \\ \text{RT}(I') &= \left\{t \in \mathcal{T}(R') : s \nmid t \text{ for all } s \in \text{HT}(I')\right\} \subset \text{RT}(I) \end{aligned}$$

By Lemma 3.1.1, $\dim_K(R/I) = \#\text{RT}(I)$ holds. Thus it is sufficient to prove that $\#\text{RT}(I) = \#\text{RT}(I')$. Since $x_{n+1} \nmid t$ for $t \in \mathcal{T}(R')$, the equality RT(I) = RT(I') holds iff $x_{n+1} \in \text{HT}(I)$, i.e. exists a $g \in R'$ for which $x_{n+1} + g \in I$. □

Corollary 4.4.2. *For the set of polynomials $\mathcal{A}$ the dimension $\dim(R/I)$ is invariant under the elimination of all variables except the round key variables $k_{i,0}$ with $0 \leq i \leq 15$ and $k_{i,j}$ with $0 \leq i \leq 3,\ 1 \leq j \leq 9$.*

Proof. By induction using Proposition 4.4.1. □

So even eliminating a significant amount of variables – resulting in a system only in the round key variables – does not reduce the complexity of converting the Gröbner basis to a term order suitable for key recovery.

Taking the Field Equations into Account

The main problem that we have is that the Gröbner basis basis does not capture that we are only interested in the solutions over the ground field $GF(2^8)$. It contains so-called "parasitic solutions", meaning solutions that do not help cryptanalytically because they are only contained in the closure of the ground field but not in the ground field itself.

A potential way to deal with this issue is to try to adjoin the set of field polynomials to the Gröbner basis. The set of roots of each of these polynomials is the set of all elements of the field F. By adjoining the set of all field polynomials $\mathcal{F}$ to the set of polynomials $\mathcal{A}$, we eliminate all points of the variety that only exist in the closure but not in the ground field. The resulting set does not form a Gröbner basis, however.

What we have to do is to compute the intersection of two varieties; this is usually achieved by computing the Gröbner basis of the sum of the corresponding ideals. We have a set of polynomials $\mathcal{A}$, describing AES which is a Gröbner basis relative to the order $<_{\mathcal{A}}$, and a second set of polynomials $\mathcal{F}$, which also forms a Gröbner basis relative to the same order. It is however unclear how to exploit the Gröbner basis property of the input.

Testing Keys

Gröbner bases were invented to solve the ideal membership problem. So why are we not able to simply test whether a linear polynomial of the form

$$k_i + C, \qquad C \in F \tag{4.16}$$

— with C being a guess for a key variable — lies in the ideal? After all, this would allow us to determine the key piecementally by guessing each byte.

A serious problem presents itself here. The polynomial system has solutions over the closure of the ground field, which means that we have to test for a polynomial:

$$g = p \cdot \prod (k_i + C_j)^{t_j}, \quad t_j \in \mathbb{N}_0,\ C_j \in F$$

instead of the above polynomial. The C_j denote candidate values for the key variable and p is a product of irreducible non-linear polynomials. The dimension of the ideal again plays an important role here: it is an upper

bound on the number of solutions of the corresponding polynomial system in the closure of the field. Hence the degree of g is expected to be very large, which prohibits this approach from working in practice.

Summary

As far as the author is aware at the time of writing this thesis, the existence of the above Gröbner basis has no implications for the security of the AES. We conjecture that methods similar to the one presented in this paper can be used to produce total-degree Gröbner bases for many other iterated block ciphers – however we like to point out that because of the high algebraic structure of Rijndael, it makes for an excellent example.

4.5 Secret S-Boxes and Algebraic Attacks

Cryptomeria, the block cipher presented in Section 2.3.3 allows for a novel algebraic attack. Its 8×8 S-Box – which is application-specific – is kept secret. In this section we show a combination of differential and algebraic methods which recovers the values of this S-Box in a chosen-text, chosen-key attack for reduced round versions of this cipher.

During an execution of the Cryptomeria cipher as a function with a variable plaintext and key, up to 20 different S-Box entries can be active, one for each round in both the round key generation and in the actual encryption process.

4.5.1 Constructing a Polynomial System

From a high-level perspective, our attack exploits simple differential characteristics and one chosen key which triggers only a low number of S-Boxes in the key schedule. This is done in order to reduce the number of active S-boxes over a small number of chosen plaintext/ciphertext pairs; our goal is to maximize the number of S-Boxes shared for these pairs. After having queried an encryption oracle for the chosen number of plaintexts we can set up a quadratic system of equations. The variables in this system model both the outputs of the S-Boxes as well as the carry bits occurring during the modular additions in the cipher and the key schedule. By obtaining a solution for the bits representing the outputs of the S-Boxes we are able to determine a subset of the S-Box entries. A repeated application of this method allows us to recover the complete S-Box.

Active S-Box Entries in the Key Schedule

Ad-hoc methods allowed for a minimization of the number of active S-Boxes down to four in the key schedule – this was more or less based on luck and

intution though. Whether this number was the minimum was not clear initially.

In the end we used a simple combinatorial method to systematically reduce the number of S-Boxes in the key schedule. Let r be the number of rounds in the cipher and k the number of distinct S-Boxes we think we can get by with as a minimum number in the key schedule. We enumerate all possible assignments of k S-Boxes to r positions and check whether there are keys fulfilling the restrictions imposed by the key schedule by solving a system of linear equations. This system of equations is small, for the full Cryptomeria cipher with 10 rounds we needed to check the solvability of linear systems consisting of 56 equations in 48 variables. The number of these systems we needed to solve was large: we had to check $> 10^6$ of these systems.

We found that four S-Boxes indeed is the minimum number of distinct S-Boxes that are active in the key schedule for any given key. A total of 1024 keys trigger only four S-Boxes, one possible key is `0x26042284118C08` [1]

Minimizing the Number of Active S-Boxes in the Cipher

In order to set up a polynomial system of equations that is solvable it is a necessity to keep both the number of unknowns and the degree of the equations low. At the same time there is an information theoretic minimum of variables we need to model in order to obtain a unique solution. In general m executions of r rounds of the cipher will potentially trigger $(m+1)r$ different entries of the S-Box, resulting in $8(m+1)r$ variables. However, since the output of the cipher is only 64 bits wide, we see that for the full 10 rounds we cannot hope to perform the attack with a single plaintext/ciphertext pair.

Differentials on the plaintext allow us to select three plaintexts which will trigger the same S-Boxes in the first 4 rounds. Moreover, by imposing a filtering condition on the ciphertext, we can then ensure that the same S-Box is triggered in the last round as well. Using these techniques we cut down the total number of potentially distinct S-Boxes for these three plaintext/ciphertext pairs to 24. Given the width of the S-Box, in this scenario sufficient information is contained in the plaintext/ciphertext pairs to determine 24 S-Box entries.

The following difference patterns trigger the same S-Boxes in the first 4 rounds with probability 0.5. If the Feistel step were an XOR instead of a modular addition they would hold with probability 1, in our case the difference can get affected by a carry bit of the modular addition.

[1] The author of this thesis later realized that this statement is wrong: the minimum number of distinct S-Boxes is three. This was overlooked due to a silly mistake in one of the author's programs.

$$\Delta_1 = (\texttt{0x0000000000000080})$$
$$\Delta_2 = (\texttt{0x0000000080000000})$$
$$\Delta_3 = (\texttt{0x0000000080000080})$$

The filter used in the last round works as follows: We simply undo the last Feistel step, apply L^{-1} and then check whether the same S-Box was applied by checking whether the S-Box output x_0'' is the same. This means that we need to find a 3-collision on the S-Boxes of the last round for the plaintext/ciphertext pairs considered.

The Polynomial System

In the following we will demonstrate how to set up a quadratic system of polynomial equations for the S-Box recovery problem. These polynomial systems are interesting in their own right because they only model a series of XORs, modular additions and bit shifts of byte and 32-bit word sized quantities. Systems of equations that can be constructed for the stream cipher SALSA-20 [9] or the block cipher TEA [113] are constructed from the same set of operations.

Equation Systems for Modular Addition

The following gives equations for a full modular addition of two k-bit values $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$. The result is $(z_1, \ldots, z_k)$. The variables $c_1, \ldots, c_{k-1}$ are intermediate variables used to store the carry bits; these are introduced to make the system a quadratic one. The equation system for the modular addition of two k-bit numbers modulo 2^k looks as follows:

$$\begin{aligned}
z_1 &= x_1 + y_1 \\
c_1 &= x_1 y_1 \\
z_2 &= x_2 + y_2 + c_1 \\
c_2 &= x_2 y_2 + c_2 x_2 + c_2 y_2 \\
&\vdots \quad\quad \vdots \\
z_k &= x_k + y_k + c_{k_1}
\end{aligned}$$

Putting it together

As previously stated, each S-box will be modelled by 8 variables representing its output. Figure 4.1 shows which S-Boxes are used in which rounds and which S-Boxes are used in the key schedule and which in the cipher. Each modular addition incurs the cost of 31 carry bits and after each round a new

Table 4.3: Equation systems for reduced round versions of Cryptomeria

rounds	S-Boxes	p/c pairs	variables	equations
4	5	2	916	1036
5	8	2	1159	1255
6	10	2	1394	1482
7	13	2	1637	1701
8	15	2	1872	1912
9	21	3	3009	3129
10	24	3	3322	3442

set of 32 state variables is introduced. Thus, if a total of s S-boxes is used in r rounds for l plaintext/ciphertext pairs, we need a total of $8s + 32lr + 62lr + 31r$ variables.

The following observation helps us construct an overdetermined system: the S-boxes are only modelled by their output, however we know that for the first 4 rounds and for the last round the input must match as well for the pairs we have chosen. We can express this by equations of the form $p_i + p_j = 0$ where p_i denotes the linear combination of variables that becomes the S-Box input for plaintext A; likewise p_j for the same S-Box in plaintext B.

For the full 10 round Cryptomeria, we obtain a total of 3442 equations in 3322 variables. Of these, 192 variables are S-Box variables.

4.5.2 The Attack: Solving the Polynomial System

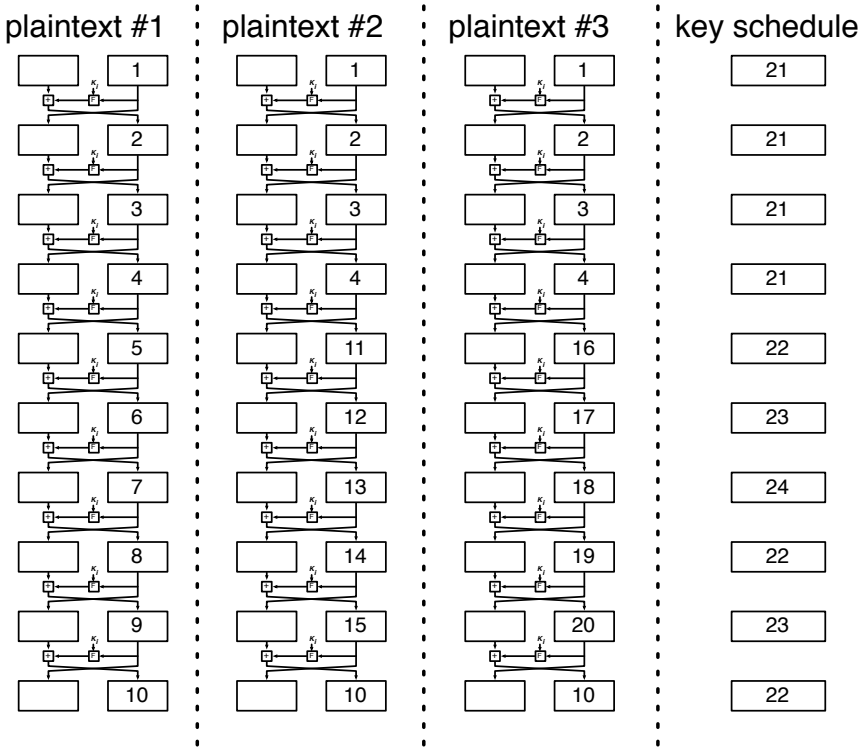
Whereas the difficulty of solving previous systems arose from the equations describing their S-Boxes, the systems we try to solve in the Cryptomeria case are hard to solve because of the long chains of non-linear carry bit calculations contained in them. In the following we present a number of methods that can be used for dealing with this specific case.

Tricks used for Solving the Polynomial System

Several methods can be used to make a Gröbner basis computation and thus a solution of the polynomial system possible.

- Carry chains can be broken by guessing variables.
- Internally, S-boxes outputs can be guessed to be identical (chance for that happening is approx. $\sqrt{2^{-8}}$)

Figure 4.1: Active S-Boxes in Cryptomeria (10 rounds and three p/c pairs)



- By knowing certain bits – i.e. the lowermost 4 bits in the modular addition of the key schedule – are always or almost always zero, the number of variables can be reduced.

- Truncated Gröbner bases can be used to avoid terms occuring in the computation above a pre-specified degree bound. Caveat: this can produce incorrect results if not applied carefully!

Experimental Results

We have written code for the open-source computer algebra system SAGE [105] to generate the systems of equations described. We then attempted to solve these systems using both the F_4 [43] implementation contained in MAGMA [92] as well as the PolyBoRi framework [18] shipped with recent versions of SAGE.

We have succeeded in breaking up to four rounds using MAGMA. Two plaintext/ciphertext pairs were needed for this attack, and a total of 5 S-Boxes were used in the encryption and the key schedule for this particular case. On an Opteron 2218 clocked at 2.6GHz the solution of this system – in 916 variables – took 42 seconds and 134MB of memory using Magma 2.13-11. As the key remains fixed, we recover four S-Box entries per iteration in this attack. With 42 seconds per entry, this attack is entirely practical. Scaling this attack up however, a we hit a hard limit. Even for five rounds were not able to attack the problem with Magma anymore, on a machine with 8GB of RAM. At this point we switched to our own implementation of F_4, as it allowed us more freedom in experimentation than Magma.

We were then able to solve polynomial systems for 8 rounds – in 1872 variables – on a machine equipped with 64 Gigabytes of memory using custom software that implements a specialized variant of the improved F_4 algorithm. This also required two plaintext/ciphertext pairs; a total of 15 S-Boxes need to be determined. In order to succeed at this, the tricks – except for the guessing of colliding S-Boxes – listed in the previous subsection were employed. We needed to guess a total of 6 carry variables to zero to obtain a polynomial system that could be solved. A total of 20GB of memory and 15 CPU hours on a Power5 p565+ were consumed.

A practical attack against the full cipher has not yet been achieved. This attack would need three plaintext/ciphertext pairs in order to build a system that allows for a unique solution as there are a total of 24 S-Boxes to be recovered in this case.

4.5.3 Results Achieved against Cryptomeria

We have demonstrated a practical attack against 8 out of 10 rounds of Cryptomeria which recovers the content of the S-Box. For this we have shown

how differential techniques can be successfully combined with algebraic cryptanalysis. Our work should be seen as a proof that algebraic cryptanalysis against deployed block ciphers can indeed be performed and. It also should serve as a warning to designers that not adhering to Kerkhoffs' principle only increases the number of possible attack vectors against a cipher. Especially when it has to be relied upon the fact that certain parts of the design stay secret.

Chapter 5

Distributed Memory Computation of RREFs

The progress of practical algebraic cryptanalysis currently is not limited by the amount of CPU power available to an attacker but rather by the amount of memory. Faugère-Lazard style Gröbner basis algorithms are extremely memory hungry, as the size of the matrix they triangulate grows exponentially in the degree of the polynomials. In this chapter we present a way to handle this problem: We propose to use distributed memory systems for Gröbner basis computations – allowing us to scale the problem better.

We show how to adapt the Gauss-Jordan algorithm to efficiently compute a Row-Reduced Echelon Form (RREF) of a matrix over a finite field in dense representation on a distributed memory system. By efficient we mean that both the data duplication is kept to a minimum and that the speed-up over a serial implementation is close to optimal. The algorithm does not require low latency for inter-node communication.

5.1 Motivation

Algorithms for computing Gröbner bases that follow the Faugère-Lazard principle [43, 76] of using linear algebra in the reduction step – such as F_4 [43] and F_5 [44] – depend on highly efficient linear algebra routines. The size of the problems we are able to solve with these algorithms crucially depends on the maximum size of matrices for which we are able to compute a Row Echelon Form (REF). Especially in the field of cryptanalysis the problems are large in the number of variables and equations. On the upside, in most cryptanalytically relevant cases the problems are specified over finite fields. Thus we do not have to worry about potential stability and convergence problems of our algorithms.

Definition 5.1.1 (REF)**.** Let $M \in F^{n \times m}$ be a matrix. We say that M is in

row echelon form if all of the following conditions hold:

- All zero rows are at the bottom of the matrix
- The leading entry of each nonzero row after the first occurs to the right of the leading entry of the previous row.
- The leading entry in any nonzero row is 1.

If a matrix is in REF and additionally, all entries in the column above and below a leading 1 are zero, the matrix is said to be in RREF.

Several methods can be chosen for computing a REF of a matrix. Iterative methods for solving systems of equations such as Block-Lanczos [28, 62] or Block-Wiedemann [29] have been proposed in the F_4 paper. [43] These can be employed to compute possible assignments for the vector x in the equation $Ax - b = 0$ with $A \in \mathbb{F}^{n \times m}, b, x \in \mathbb{F}^n$ for fixed A, b. These solutions can then be used to reconstruct a REF of the matrix. However, this appproach is not efficient for matrices with large rank deficit. As the solution vectors generated by these algorithms are random elements of the nullspace, a we effectively have to perform elimination on these vectors to regenerate a REF.

Two commonly used methods exist for directly computing the REF of a matrix: The Gaussian Elimination (GE) method and the Gauss-Jordan Elimination (GJE) method. A regular GE brings a matrix into REF whereas a GJE produces a RREF. In the case of GE, a RREF is obtained from the REF by an additional step called back substitution. In the GJE method the RREF is achieved directly by forcing the column elements above and below the pivot to zero. This is the reason why this method is referred to as "one-sweep". The problem of inverting a matrix is closely related to the task of obtaining a RREF, however the latter task is more general: we have to be able to deal with rectangular and singular matrices as well.

For inverting a matrix in parallel, a block matrix version of the Gauss-Jordan algorithm [107, 96] has been proposed. This allows to split the task of computing a RREF up into several jobs of inverting, multiplying and adding matrices that can be performed on a Distributed Memory System (DMS) Essentially, we want to do the computation in-place with the entries of the matrix being stored distributed among the nodes. Implementations of these parallel block-matrix Gauss-Jordan algorithms have already been analyzed in the literature [85, 95]. Unfortunately however, using the block-matrix approach does not seem to be easily possible for computing RREFs since it requires submatrices to be invertible. Henceforth we analyze the direct approach of a parallel Gauss-Jordan algorithm in our setting. The advantage of *not using* a block-matrix algorithm however is that we can more easily adapt the algorithm to a situation that requires additional constraints in the elimination process such as those imposed by the F_5 criterion [44]. Parallel GJE has been investigated for Hypercube architectures in

[61], showing a parallel efficiency of 90%. This led us to revisit these results and propose the algorithm presented in this chapter.

5.2 A Model for Distributed Memory Computations

Several models for computation on DMSs exist. One of the earliest proposed models was the Parallel Random Access Machine (PRAM) [51]. This is a theoretical model of a register machine in which multiple registers can perform computations at the same time, subsequently storing the results in memory. Several variants of this model exist which differ in whether concurrent write access to a memory cell by different registers at the same time is allowed.

We will be using a model that is much more coarse-grained and geared towards the specific application we have in mind. Just like the PRAM model, we assume a common clock – synchronous operation of the nodes. We distinguish between a master node and slave nodes, where the master node is pushing data towards the slave nodes. The assignment of the master node is not static. It may change during the computation.

At the beginning of the reduced-row echelon form computation, we will decide on a fixed number of nodes t of the cluster to run our code. Each of these nodes may have multiple CPUs with RAM shared between them. The configuration of the nodes is identical in all characteristics except in terms of CPUs per node. Node i, with $1 \leq i \leq t$ has N_i CPUs for and the RAM on each of the nodes is limited to hold at most M elements of the field F.

5.3 A Parallelized version of Gauss-Jordan

Let $A = (a_{i,j}) \in \mathbb{F}^{n \times m}$. The central idea of Gauss-Jordan is to subtract an appropriate multiple of a line k from all other lines such that every element in column k becomes zero:

$$a_{i,j} = a_{i,j} - (a_{i,k} \cdot a_{k,k}^{-1} \cdot a_{k,j})$$

We propose to compute a RREF in parallel using this idea with the following code being executed on the master node:

1. Slice matrix into t submatrices $B_1, \ldots, B_t$ such that $B_1 || \ldots || B_t = A$.
2. Set master node to $i = 1$
3. Set $r = 0$ (used for computing the rank)
4. Set $k = 1$ and $l = 1$ (row and column index)

5. Check whether $a_{k,l} \neq 0$. If so, broadcast row multiplier $a_{k,l}^{-1}$ to all nodes
   [MULTIPLY($a_{k,l}^{-1}, k$)], increase r by 1 and go to step 8.
6. Find row u with $u > k$ such that $a_{u,l} \neq 0$. If no such row exists, broadcast [SYNC(l)], increase l by 1 and go to step 5
7. Broadcast request to add row u to row k to all nodes [ADD_ROW(u, k)] and go to step 5
8. Find multipliers to clear all entries in column above and below $a_{k,l}$
9. Broadcast column of multipliers $b := (b_1, \ldots, b_n)$ and row index k to all nodes
   [CLEAR_COLUMN(b, k)].
10. Increase k by 1, increase l by 1.
11. If $r = m$, broadcast request to set all rows below row k to zero [ZEROIZE(k)] and terminate (rank maximal).
12. If column l does not reside in master's local memory, increase i and pass control to next node.

Please note that the requests for computation that the master node broadcasts to all nodes are not only executed on the slave nodes but also on the master node itself. The node-local row-data of row j will in the following be referred to as R_j. Each node performs one of the following operations:

MULTIPLY_ROW(c, k)	$R_k \leftarrow c \cdot R_k$
CLEAR_COLUMN($(b_1, \ldots, b_n), k$)	for all $1 \leq j \leq n$: $R_j \leftarrow R_j + b_j \cdot R_k$
ADD_ROW(u, k)	$R_k \leftarrow R_k + R_u$
ZEROIZE(k)	for all $k < j \leq n$: $R_j = (0, \ldots, 0)$

The operation SYNC keeps nodes synchronized and assures that each slave node's row and column index is identical to the values of the master node. Otherwise the slave nodes have no reliable information what row is operated on and when they have to switch over, becoming the master node.

We found that slicing the matrix vertically rather than horizontally has the benefit of resulting in a much simpler algorithm. Because all of the information to clear a column are local to the master node, only unidirectional communication towards the slave nodes is needed.

5.4 Notes on the Performance of the Algorithm

Looking at the above outline of the algorithm, we see that a maximum number of each of the following operations requests to be broadcast is bounded

by n: MULTIPLY_ROW, ADD_ROW, CLEAR_COLUMN. The maximum communication cost incurred therefore is $4n$ integer values and $(n+1)m$ field elements.

In most cases, especially when dealing with very small finite fields such as $GF(2)$, the performance of the algorithm may not be constrained by the cost of arithmetic operations but rather by speed between the CPUs of the node and RAM. Caches between the CPU and RAM can help to improve performance when data locality can be exploited. This is something to keep in mind for the data layout in RAM to achieve an efficient implementation of the algorithm.

We see that the impact of latency of intra-node communication for our algorithm is almost negligible. Because of the unidirectional communication, the requests may actually be queued both on the sending and the receiving side. Latency issues only kick in when a master node yields his control to another node. This however only happens a total of t times. This makes the speed-up almost linear in the number of nodes involved in the computation.

5.5 Properties of the Algorithm and Implementation

The algorithm proposed has an undesirable asymptotic complexity, namely cubic time complexity in the size of the matrix. However, we are able to obtain an almost linear speed-up in the number of nodes. Moreover the algorithm does not need low-latency networks to perform well. This indicates that it may even be suited for running a computation on a network of physically distributed nodes, such as the internet or a company intranet.

The algorithm proposed has been implemented in ANSI C. This implementation does not make use of the Message-Passing Interface (MPI); rather the requests between the nodes are communicated through TCP. A tiny command line program for broadcasts receives the requests on each node and passes them on through a pipe. Future plans include to integrate the implementation into the Xylirt package, a piece of software written by the author that implements the linear-algebra based Gröbner basis algorithms described in Chapter 3.

It is an open problem to adapt the algorithm to the case of sparse matrices. In this case a strategy for avoiding rows to become dense during the computation is needed. Most strategies for reducing the fill-in require reordering the rows. In order to decide how to reorder the rows, heuristics are applied that require a global view on the individual rows. This in turn is likely to require bi-directional communication between the master and slave nodes which would destroy one of the most attractive properties of the algorithm.

5.6 Experimental Results

We have benchmarked our implementation against the implementations in Magma and in SAGE. For this we have chosen $GF(257)$ as a base field and have chosen matrix sizes between 1500×1500 and 6000×7500. The matrix dimensions were chosen to be multiples of three since in our experiments we have restricted the number of nodes too three.

SAGE contains two implementations for computing the row-echelon form of a matrix. LinBox's implementation is described in the FFPACK [41] paper. It makes use of LQUP factorizations [63] and Coppersmith-Winograd matrix multiplication to achieve a sub-cubic time complexity in the matrix dimension. The results for this implementation are listed in the column labelled "Linbox", the results for the vanilla GJE implementation are listed in the next column. The columns with the labels PGJE show our own C implementation running on $n = 1$ node (without communication) and on a network with $n = 3$ nodes.

matrix size	Linbox	Gauss	PGJE ($n = 1$)	PGJE ($n = 3$)
1500×1500	4.2 s	30.4 s	27.4 s	9.3 s
1500×3000	11.1 s	90.0 s	52.1 s	17.5 s
3000×3000	20.9 s	240.9 s	209.6 s	70.1 s
3000×4500	39.8 s	484.3 s	309.4 s	103.4 s
4500×4500	56.4 s	817.1 s	697.8 s	232.8 s
4500×6000	95.3 s	1310.0 s	923.8 s	308.3 s
6000×6000	141.8 s	1868.8 s	1644.0 s	548.5 s
6000×7500	326.0 s	2903.3 s	2042.5 s	681.7 s

The experiments were performed on a network of Linux PCs with Intel Celeron CPUs clocked at 2.4GHz. The machines were connected through Gigabit Ethernet and running kernel version 2.6.x. We see that in practice the algorithm achieves almost linear speed-up – as claimed.

Chapter 6

Conclusions

This thesis shows several things: First of all, we have demonstrated that there are indeed block ciphers that resist differential and linear cryptanalysis as well as brute-force attacks for which Gröbner bases can be used to recover the key from a plaintext/ciphertext pair. Secondly, we have shown how to construct a Gröbner basis for AES-128. We have however been unable to leverage this Gröbner basis into a cryptanalytic attack. For a reduced-round version of a deployed cipher, Cryptomeria, we have shown how to reconstruct the contents of it secret S-Box using algebraic and differential methods. Last but not least we have demonstrated that row-reduced echelon forms of dense matrices can be efficiently computed on a distributed memory system by giving a parallelized version of the Gauss-Jordan algorithm that has almost linear speedup in the number of nodes.

Open research problems Some of the most fundamental problems in the field of algebraic cryptanalysis are not yet settled. For example, it has not been established whether the iterated structure of modern block ciphers gives symmetry in the polynomial equations that is actually exploitable in cryptanalysis. To answer this question positively, customized algorithms need to be developed and used instead of using generic algorithms like F_4; in fact customized variants of efficient general purpose Gröbner basis algorithms may be sufficient. Answering this question negatively potentially is much harder. Thus far we have only seen experimental evidence suggesting that general purpose algorithms are not able to exploit the structure.

Attacks that work with only a single plaintext/ciphertext pair are the holy grail in cryptanalysis. In algebraic cryptanalysis only a single plaintext/ciphertext pairs is used to keep the number of variables as low as possible. However, Faugère showed that for certain instances of FLURRY using several pairs of related plaintext/ciphertext pairs indeed allow to attack the cipher faster by using Gröbner basis methods [42]. It is an open problem to carry this method over to other block ciphers and improve upon it.

Looking at the last-round attacks presented in Chapter 2 we see that all of these attacks can be reduced to distinguishing the input of the last round from the input that would have been generated by a random permutation. Henceforth the question arises whether we can perform this distinguishing process in an algebraic way. Recent results on the multivariate cryptosystem HFE show that algebraic distinguishers can be built in practice [58], whether they can be built for block ciphers is an open problem.

Combining algebraic cryptanalysis with other types of cryptanalytic attacks seems to be a worthwhile research topic. Integral cryptanalysis for example can be modelled algebraically, it remains to be seen whether this approach yields cryptanalytic advances.

Guessing variables is a very simple yet effective method to trade off memory complexity for time complexity in algebraic attacks. More work needs to be done to systematically determine which variables should best be guessed. In certain situations it makes sense to guess variables during the Gröbner basis computation.

From a practical side, more work needs to be done on dealing with the issue of memory complexity of Gröbner basis algorithms. Using distributed memory architectures may be the key to success here. The Parallel Gauss Jordan algorithm presented in the last chapter of this thesis is a step into this direction. It only deals with densely populated matrices however, whereas the Macaulay matrices we encounter in practice are only sparsely populated. It is an open problem to come up with algorithms that allow us to efficiently compute row-echelon forms of sparsely populated matrices on distributed memory systems. For the dense case it should be investigated whether and if so, how blocking methods can be used in the algorithm. This would allow to obtain an algorithm performing better asymptotically by using fast matrix multiplication algorithms.

Last but not least: Although I am very much in favour of using techniques with a solid mathematical foundation such as Gröbner bases, other more heuristic methods should not be easily discarded but be studied in detail. Also, methods using different polynomial representations such as BDDs, as was recently suggested by Brickenstein and Dreyer [18] and implemented in their PolyBoRi package clearly seem to be worth investigating.

Bibliography

[1] 4C Entity, LLC. *C2 Block Cipher Specification*, January 2003.

[2] Frederik Armknecht and Matthias Krause. Algebraic attacks on combiners with memory. In Boneh [16], pages 162–175.

[3] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and Gröbner basis algorithms. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2004.

[4] Magali Bardet, Jean Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the degreeof regularity of semi-regular polynomial systems. In *In MEGA 2005, 8th International Symposium on Effective Methods in Algebraic Geometry*, 2005. 15 pages.

[5] Magali Turrel Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*. PhD thesis, Université Paris 6, December 2004.

[6] David Bayer and Michael Stillman. On the complexity of computing syzygies. *Journal of Symbolic Computation*, 6(2/3):135–147, 1988.

[7] Thomas Becker and Volker Weispfenning. *Gröbner Bases – A Computational Approach to Commutative Algebra*. Springer-Verlag, 1991.

[8] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1995.

[9] Daniel J. Bernstein. Salsa20. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/025, 2005. `http://www.ecrypt.eu.org/stream`.

[10] Thomas Beth and Cunsheng Ding. On almost perfect nonlinear permutations. In Helleseth [59], pages 65–76.

[11] Eli Biham. How to make a difference: Early history of differential cryptanalysis. presentation given at Fast Software Encryption (FSE) 2006 in Graz, Austria on March 16th, 2006.

[12] Eli Biham, editor. *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*. Springer, 1997.

[13] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1991.

[14] Alex Biryukov, Jorge Nakahara Jr., Bart Preneel, and Joos Vandewalle. New weak-key classes of IDEA. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *ICICS 2002*, volume 2513 of *Lecture Notes in Computer Science*, pages 315–326. Springer, 2002.

[15] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.

[16] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.

[17] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[18] Michael Brickenstein and Alexander Dreyer. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. In *Electronic Proceedings of MEGA 2007*, pages 48–65, 2007.

[19] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[20] Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of Groebner bases. In Edward W. Ng, editor, *EUROSAM 1979*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 1979.

[21] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. Block ciphers sensitive to Gröbner basis attacks. In David Pointcheval,

editor, *CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 313–331. Springer, 2006.

[22] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. A zero-dimensional Gröbner basis for AES-128. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 78–88. Springer, 2006.

[23] Jung Hee Cheon, Seongtaek Chee, and Choonsik Park. S-boxes with controllable nonlinearity. In Stern [106], pages 286–294.

[24] Carlos Cid and Gaëtan Leurent. An analysis of the XSL algorithm. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 333–352. Springer, 2005.

[25] Carlos Cid, Sean Murphy, and Matt Robshaw. Small scale variants of the AES. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer, 2005.

[26] Stéphane Collart, Michael Kalkbrener, and Daniel Mall. Converting bases with the Gröbner walk. *Journal of Symbolic Computation*, 24(3/4):465–469, 1997.

[27] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, STOC'71*, pages 151–158, New York, 1971. ACM, ACM Press.

[28] Don Coppersmith. Solving linear equations over $\mathrm{GF}(2)$: block Lanczos algorithm. *Linear Algebra and its Applications*, 192:33–60, 1993.

[29] Don Coppersmith. Solving homogeneous linear equations over $\mathrm{GF}(2)$ via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.

[30] Nicolas Courtois. Feistel schemes and bi-linear cryptanalysis. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer, 23–40.

[31] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[32] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[33] Nicolas T. Courtois, Gregory V. Bard, and David Wagner. Algebraic and slide attacks on KeeLoq. In Kaisa Nyberg, editor, *FSE 2008*, Lecture Notes in Computer Science. Springer, 2008. to be published.

[34] David A. Cox, John B. Little, and Don O'Shea. *Ideals, Varieties, and Algorithms*. Springer-Verlag, NY, 2nd edition, 1996. 536 pages.

[35] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[36] Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In Preneel [97], pages 275–285.

[37] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In Biham [12], pages 149–165.

[38] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. Submission to NIST for AES call, 1998.

[39] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: The Wide Trail Strategy*. Springer-Verlag, 2001.

[40] Hans Dobbertin. One-to-one highly nonlinear power functions on $GF(2^n)$. *Applicable Algebra in Engineering, Communication and Computing*, 9(2):139–152, 1998.

[41] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. Ffpack: finite field linear algebra package. In Jaime Gutierrez, editor, *ISSAC 2004*, pages 119–126. ACM, 2004.

[42] Jean-Charles Faugere. Gröbner bases. applications in cryptology. invited talk given at Fast Software Encryption (FSE) 2007 in Luxembourg on March 27th, 2007.

[43] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

[44] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *ISSAC*, pages 75–83. ACM, 2002.

[45] Jean-Charles Faugère and Gwénolé Ars. An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases. Rapport de recherche de l'INRIA 4739, INRIA, February 2003. `http://www.inria.fr/rrrt/rr-4739.html`.

[46] Jean-Charles Faugère, P. Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[47] Jean Charles Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Cryptosystems using Gröbner bases. In Boneh [16], pages 44–60.

[48] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, May 1973.

[49] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2001.

[50] Niels Ferguson, Richard Schroeppel, and Doug Whiting. A simple algebraic representation of rijndael. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 103–111. Springer, 2001.

[51] Steven Fortune and James Wyllie. Parallelism in random access machines. In *STOC 1978*, pages 114–118. ACM, 1978.

[52] Pierre-Alain Fouque, Louis Granboulan, and Jacques Stern. Differential cryptanalysis for multivariate schemes. In Cramer [35], pages 341–353.

[53] Kris Gaj and Arkadiusz Orlowski. Facts and myths of enigma: Breaking stereotypes. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2003.

[54] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W.H. Freeman and Co., New York, 1979.

[55] Rüdiger Gebauer and H. Michael Möller. On an installation of Buchberger's algorithm. *J. Symb. Comput.*, 6(2/3):275–286, 1988.

[56] Henri Gilbert and Helena Handschuh. Security analysis of SHA-256 and sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 175–193. Springer, 2004.

[57] Dieter Gollmann, editor. *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*. Springer, 1996.

[58] Louis Granboulan, Antoine Joux, and Jacques Stern. Inverting HFE is quasipolynomial. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[59] Tor Helleseth, editor. *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*. Springer, 1994.

[60] Martin E. Hellman, Ralph Merkle, Richard Schroeppel, Lawrence Washington, Whit Diffie, Stephen Pohlig, and P. Schweitzer. Results of an initial attempt to cryptanalyze the data encryption standard. Technical Report SEL 76-042, Stanford University, Information Systems Laboratory, September 1976.

[61] Paul G. Hipes and Aron Kuppermann. Gauss-Jordan inversion with pivoting on the Caltech Mark II hypercube. In *The 3rd Conference on Hypercube Concurrent Computers and Applications*, volume II, Applications, pages 1621–1634, Pasadena, CA, January 1988. ACM. Caltech.

[62] Bradford Hovinen and Wayne Eberly. A reliable block lanczos algorithm over small finite fields. In Manuel Kauers, editor, *ISSAC 2005*, pages 177–184. ACM, 2005.

[63] Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast lup matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, 1982.

[64] Thomas Jakobsen and Lars Knudsen. The interpolation attack on block ciphers. In Biham [12], pages 28–40.

[65] Pascal Junod and Serge Vaudenay. FOX: A new family of block ciphers. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2004.

[66] Michael Kalkbrener. On the complexity of Gröbner bases conversion. *Journal of Symbolic Computation*, 28(1-2):265–273, 1999.

[67] Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. *Mathematics of Computation*, 67(223):1179–1197, 1998.

[68] Masayuki Kanda. Practical security evaluation against differential and linear cryptanalyses for Feistel ciphers with SPN round function.

In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography 2000*, volume 2012 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2001.

[69] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations (Proceedings of a Symposium on the Complexity of Computer Computations, March, 1972, Yorktown Heights, NY)*, pages 85–103. Plenum Press, New York, 1972.

[70] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:3–72, January 1883.

[71] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Stern [106], pages 206–222.

[72] Lars R. Knudsen. Practically secure Feistel ciphers. In Ross J. Anderson, editor, *FSE 1993*, volume 809 of *Lecture Notes in Computer Science*, pages 211–221. Springer, 1994.

[73] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[74] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Damgård, editor, *EUROCRYPT 1990*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1991.

[75] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[76] Daniel Lazard. Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *EUROCAL 1983*, volume 162 of *Lecture Notes in Computer Science*, pages 146–156. Springer, 1983.

[77] Fen Liu, Wen Ji, Lei Hu, Jintai Ding, Shuwang Lv, Andrei Pyshkin, and Ralf-Philipp Weinmann. Analysis of the SMS4 block cipher. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *ACISP 2007*, volume 4586 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2007.

[78] Stefan Lucks. The saturation attack - a bait for Twofish. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[79] F. S. Macaulay. On some formulæ in elimination. *Proceedings of the London Mathematical Society*, 33(1):3–27, 1902.

[80] Francis Sowerby Macaulay. On the resolution of a given modular system into primary systems including some properties of Hilbert numbers. *Mathematische Annalen*, 74(1):66–121, 1913.

[81] M. Matsui. Linear cryptanalysis method for DES cipher. In Douglas R. Stinson, editor, *CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 386 – 387. Springer, 1994.

[82] Mitsuru Matsui. Linear cryptoanalysis method for DES cipher. In Helleseth [59], pages 386–397.

[83] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of FEAL cipher. In Rainer A. Rueppel, editor, *EUROCRYPT 1992*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1993.

[84] E. Mayr and A. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. Math., Beijing*, 46(3):305–329, 12 1982.

[85] Nouredine Melab, El-Ghazali Talbi, and Serge G. Petiton. A parallel adaptive Gauss-Jordan algorithm. *The Journal of Supercomputing*, 17(2):167–185, 2000.

[86] Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In Armin Biere and Carla P. Gomes, editors, *SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006.

[87] Sean Murphy and Matthew J.B. Robshaw. Essential algebraic structure within the AES. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.

[88] National Bureau of Standards. The Data Encryption Standard. Federal Information Processing Standards Publication (FIPS) 46, 1977.

[89] National Institute of Standards and Technology (NIST). Announcing the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication (FIPS) 197, November 2001.

[90] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Helleseth [59], pages 55–64.

[91] Kaisa Nyberg and Lars R. Knudsen. Provable security against differential cryptanalysis. In Ernest F. Brickell, editor, *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 1993.

[92] University of Sydney Computational Algebra Group. The Magma computational algebra system, 2004. `http://magma.maths.usyd.edu.au/magma/`.

[93] Government Committee of the USSR for Standards. Gosudarstvennyi standard 28147-89. Cryptographic Protection for Data Processing Systems, 1989.

[94] Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996.

[95] Serge G. Petiton and Lamine M. Aouad. Large scale peer to peer performance evaluations, with Gauss-Jordan method as an example. In Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Wasniewski, editors, *PPAM 2003*, volume 3019 of *Lecture Notes in Computer Science*, pages 938–945. Springer, 2004.

[96] Brad Pierce and D. Stott Parker. A block matrix generalization of Gauss-Jordan elimination using Haynsworth's quotient formula for Schur complements. Technical report CSD-950063, University of California, Los Angeles, 1995.

[97] Bart Preneel, editor. *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 1995.

[98] Håvard Raddum and Igor Semaev. New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475, 2006. `http://eprint.iacr.org/2006/475`.

[99] Vincent Rijmen and Paulo S. L. M. Barreto. The WHIRLPOOL hashing function. NESSIE submission, standardized in ISO/IEC 10118-3:2004, may 2003.

[100] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher SHARK. In Gollmann [57], pages 99–111.

[101] RSA Laboratories. *PKCS #11 v2.11: Cryptographic Token Interface Standard*. RSA Data Security, Inc., November 2001.

[102] Markku-Juhani Saarinen. A chosen key attack against the secret S-boxes of GOST, 1998.

[103] Bruce Schneier and John Kelsey. Unbalanced Feistel networks and block cipher design. In Gollmann [57], pages 121–144. full version also at http://www.schneier.com/paper-unbalanced-feistel.pdf.

[104] Claude Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949. A footnote on the initial page says: "The material in this paper appeared in a confidential report, 'A Mathematical Theory of Cryptography', dated Sept. 1, 1946, which has now been declassified.".

[105] William Stein. *Sage Mathematics Software (Version 2.10.1)*. The Sage Group, 2008. `http://www.sagemath.org`.

[106] Jacques Stern, editor. *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*. Springer, 1999.

[107] Ole Tingleff. Systems of linear equations solved by block Gauss-Jordan method using a transputer cube. Technical report IMM-REP-1995-08, Institute of Mathematical Modelling, Technical University of Denmark, 1995.

[108] University of Sydney Computational Algebra Group. The Magma Computational Algebra System, 2004. `http://magma.maths.usyd.edu.au/magma/`.

[109] Serge Vaudenay. On the weak keys of Blowfish. In Gollmann [57], pages 27–32.

[110] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Cramer [35], pages 19–35.

[111] Ralf-Philipp Weinmann. Evaluating algebraic attacks on the AES. Diplom thesis, Technische Universität Darmstadt, Sep 2003.

[112] Ralf-Philipp Weinmann and Johannes Buchmann. Distributed memory computation of row-reduced echelon forms over finite fields. submitted to First International Conference on Symbolic Computation and Cryptography (SCC 2008).

[113] David J. Wheeler and Roger M. Needham. TEA, a tiny encryption algorithm. In Preneel [97], pages 363–366.

[114] Christopher Wolf. “Hidden Field Equations" (HFE) - variations and attacks. Master’s thesis, Universität Ulm, December 2002. http://www.christopher-wolf.de/dpl.