Block Cipher Analysis

Miolane, Charlotte Vikkelsø

Publication date:
2009

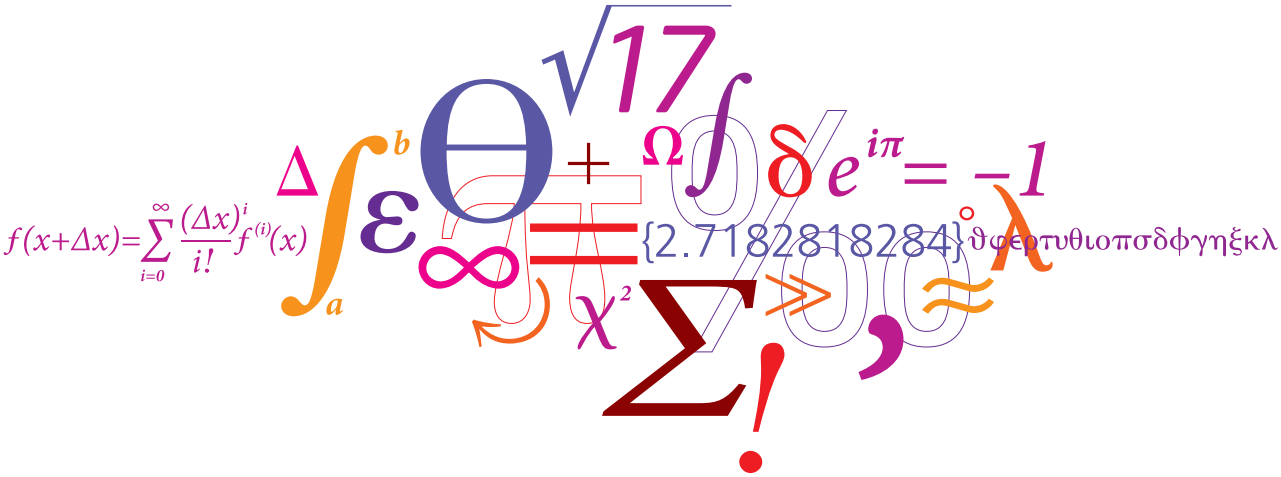
Document Version
Publisher's PDF, also known as Version of record

BLOCK CIPHER ANALYSIS

Charlotte Vikkelsø Miolane

Ph.D Thesis
Technical University of Denmark
Department of Mathematics

December 18, 2008



Date Charlotte Vikkelsø Miolane

Technical University of Denmark
Department of Mathematics
Matematiktorvet 303S
DK-2800 Kgs. Lyngby
Denmark

Contents

Resumé

Blockchifre er kryptografiske primitiver som opererer på tekster af en, af chifferet, specificeret længde. De fleste blockchifre er designet med sikkerhed og effektivitet for øje. Blokchifre anvendes i vid udstrækning til kryptering men indgår eksempelvis også som byggesten i visse hashfuktioner og i såkaldte message authentication codes (MAC). Formålet med kryptoanalyse er at opnå viden om sikkerheden af chifrene. Ideelt set bør der ikke findes bedre angreb end de generiske angreb, udtømmende nøglesøgning og tabel opslag.

Denne afhandling indeholder en general introduktion til kryptografi især med fokus på blokchifre, heriblandt the Advanced Encryption Standard (AES). De mest generelle metoder til krytoanalyse af blokchifre beskrives mens der især fokuseres på de såkaldte algebraiske angreb. Disse har haft en del succes i anvendelse på visse strøm chifre, men har i modsætning hertil ikke bibragt nogle banebrydende resultater på blokchifre. Denne afhandling bidrager med en ny vinkel på de algebraiske angrebs anvendelse på blokchifre. Desuden præsenteres en ny teknik hvor probabilistiske ligninger anvendes til at forbedre de algebraiske angreb. I afhandlingen præsenteres resultaterne af en række praktiske eksperimenter med de algebraiske angreb på små blokchifre. Endelig præsenteres et nyt blokchiffer ved navn Present, og vi undersøger sikkerheden af dette, især mod algebraisk og differentiel kryptoanalyse.

Summary

Block ciphers are cryptographic primitives that operate on fixed size texts (blocks). Most designs aim towards secure and fast encryption of large amounts of data. Block ciphers also serve as the building block of a number of hash functions and message authentication codes (MAC). The task of cryptanalysis is to ensure that no attack violates the security bounds specified by generic attack namely exhaustive key search and table lookup attacks.

This thesis contains a general introduction to cryptography with focus on block ciphers and important block cipher designs, in particular the Advanced Encryption Standard (AES). We describe the most general types of block cipher cryptanalysis but concentrate on the algebraic attacks. While the algebraic techniques have been successful on certain stream ciphers their application to block ciphers has not shown any significant results so far. This thesis contributes to the field of algebraic attacks on block ciphers by an analytic and systematic approach that allows insight to the techniques. Moreover a new procedure of generating and applying probabilistic equations in algebraic attacks on block cipher is proposed and examined. Also, we present practical results, which to our knowledge are the best algebraic results on small scale variants of AES. In the final part of the thesis we present a new block cipher proposal Present and examine its security against algebraic and differential cryptanalysis in particular.

Preface

As technological development progresses doors are opened to new options which increase the flexibility and freedom of both individuals, companies and governments. In the slipstream issues of security and privacy appear which require affordable solutions convenient for all parts. For example we are no longer bounded by (limited) opening hours of our bank, but have access to our accounts any time of the day through the Internet. In Denmark the government offers digital signatures that allow the citizens to access, change, and verify sensitive data. Meanwhile utilizing the services we rely on the security of cryptographic protocols and underlying cryptographic primitives and the key handling. Several Danish banks have been criticized for their solution of the latter where a key file stored on the users hard disk is combined with a user password to access the account. Given a hacker gains access to your hard disk he can obtain the key file and apply a key logging program to obtain your password. To prevent this, your key file should be stored in a place physically separated from your computer e.g. on a key card.
In the real world people make mistakes which may cause that the security is often violated. As cryptographers we usually consider an ideal world where the parties act according to flawless security protocols. In this world the security falls back on the underlying cryptographic primitives. We often divide the cryptographic primitives into the categories of public key cryptography, including e.g., cryptosystems as for instance RSA and digital signature schemes, and symmetric key cryptography including block ciphers, stream ciphers, hash functions, and message authentication codes (MAC). The cryptographic primitives are designed to address various issues of security for example eavesdropping, tampering, message forgery, impersonation. This thesis concerns security of the so-called block ciphers. The thesis is organized as follows.

Chapter 1 provides motivation and a general introduction to the field of cryptography. We define basic concepts and terminology, such as security definitions, bounds, and attack scenarios.

Chapter 2 introduces the category of cryptographic primitives called block ciphers. We provide a short description of Data Encryption Standard (DES) which is and has been the most applied block cipher throughout the past three decades. Also, we describe the Advanced Encryption Standard (AES) which was developed to replace DES as encryption standard.

Chapter 3 introduces the field of cryptanalysis applied to block cipher. We give a short description of exhaustive key search and the time-memory trade-off attack applied to block ciphers. The attack is very important because it sets a bound on the security provided by any block cipher. Also, we provide a description of the most general types of cryptanalysis on block ciphers namely linear and differential cryptanalysis. These have had a great impact on today's block cipher design. A large part of this thesis concerns algebraic attack on block cipher which we introduce as the last part of Chapter 3. We describe how to obtain algebraic descriptions of block ciphers and study this on AES.

Chapter 4 describes Gröbner basis theory and the related algebraic tools which provide an important approach in algebraic attacks. Our interest in Gröbner basis theory is due to its application for solving the algebraic equations over block ciphers. We describe Buchberger's algorithm and give a top level description of the so-called $F4$ algorithm.

Chapter 5 concerns the linearization techniques for solving algebraic equations. We describe respectively basic linearization, relinearization, and extended linearization (XL). We point out the differences and the problems regarding the time and memory complexity of the algorithms.

Chapter 6 presents our approach to analyze XL similar attacks. We present a new variant of the XL algorithm which we name iterated XL. The advantage is that regarding first part of the algorithm we are able to provide exact numbers for the work effort and memory consumption. Moreover we contribute systematic procedures for generating equations and hereby avoid a part of the redundancy of the original XL algorithm. Finally, we outline a number of simulations of the attack on small block ciphers.

Chapter 7 concerns small scale variants of AES proposed in [15]. We outline a number of timing results obtained using Magma's package for Gröbner bases computation over $GF(2)$. The results are, to the best of our knowledge, the best algebraic results on small scale variant of AES and seem to indicate that the approach is more

efficient for solving the algebraic equations of AES over $GF(2)$ than the algebraic equations over $GF(2^8)$ presented in [50].

Chapter 8 presents a new approach in algebraic attacks on block ciphers. The idea is to apply probabilistic equations. We describe two methods for obtaining probabilistic equations over the non-linear S-boxes, which is the building stone of most block ciphers. The techniques are demonstrated on the DES S-boxes.

Chapter 9 describes a new block cipher design Present which we propose in [10]. Present is an ultra light weight block cipher that suits very constrained devices such as RFID chips. The cipher has been analyzed with respect to linear, differential, and algebraic techniques. In this chapter we focus in particular on the differential and algebraic properties of the cipher.

Chapter 10 summarizes the results of this thesis.

This thesis was carried through at the Department of Mathematics, Technical University of Denmark. The project was co-funded one third by Technical University of Denmark (DTU), and two thirds by Department of Mathematics at DTU. The author was supervised by Professor Lars Ramkilde Knudsen.
The thesis describes the work carried out by the author during her PhD studies. Our contributions are presented in Chapters 3, 6, 7, 8, and 9. One technical report under the title "On Algebraic Attack on Block Cipher" [43] and one paper "Present-an ultra lightweight cipher" [10] were published. Moreover a paper "Solving nonlinear equations with applications to block ciphers" has been submitted to an international journal, December 2008. Another paper on probabilistic equations is currently in the process of being written.

Acknowledgement

First of all, I would like to thank my supervisor Lars Ramkilde Knudsen for his help throughout both my master and Ph.D project. Thank you for helping me raise funds for my Ph.D project, enabling me to do this project which I wanted more than anything. Thank you for your devotion to my project, for always taking the time to answer questions and discussing ideas, results and possibilities. Finally, thank you for putting effort in commenting and correcting my thesis. Also I would like to express my appreciation to your wife Heather and your three children Sasha, Kasper, and Mia, for welcoming me in your home when I arrived to Australia in January, 2006. It meant a lot to me.

Sincerely thanks to Tom Høholdt for help and support throughout my studies at DTU. Thank you for introducing me to discrete mathematics and for directing me towards the field to cryptography. Knowing you truly has been priceless to me.

When I first started as a Ph.D in the Crypto group it was a party of only few people. Meanwhile the Crypto group has expanded a lot and is in my opinion an exquisite place to work. I really appreciate the open door policy, the discrete math and crypto seminars, and the Friday lunches. Thank you to the entire discrete math group for creating a great environment for research.

It has been a pleasure to spend the past 4 years at Department of Mathematics and I would like to than everyone there for making it a nice place to work. Wholehearted thanks to the Department of Mathematics for funding two thirds of my Ph.D.

Special thanks to Christian Henriksen and Peter Beelen. I am grateful for your help and comments on my work. A very special thanks to Thomas Hjorth and Julia Borghoff for your thorough work on correcting my thesis. Thank you Erik for bringing the tea ;)

In spring 2006 I visited the Information Security Institute (ISI) at the Technical University of Queensland (QUT) in Australia. Thank you to the entire group at ISI and in particular Ed Dawson for inviting me there.

Thanks to my parents for supporting me throughout my education all the way from primary school to university. Thank you to my husband, the love of my life, Christophe. Thank you for standing by me through good and bad times, in spite of

my at times bad temper. Without your support this project would not have been possible. Thank you to my little daughter Camille for being the sunshine of my life that puts it all into perspective.

Chapter 1

Introduction to Cryptography

1.1 Cryptosystems

Cryptography is the mathematical approach of handling security issues in the world of communication. The original problem consists of two parties wanting to communicate through an insecure channel in a way that makes it impossible for eavesdroppers to intercept the information communicated.
The situation is usually modeled by a transmitter Alice, a receiver Bob and an eavesdropper called Eve. Alice wants to encode the message such that Eve cannot read what is being transmitted to Bob. Bob however must be able to decode the message, such that he can read it. The cryptographic model for solving this problem is a cryptosystem.

Definition 1.1 (Cryptosystem)**.** *A cryptosystem is defined by an injective function called an encryption rule $e_{K_e}(P)$ and its corresponding inverse, the decryption rule $d_{K_d}(C)$.*
P denotes a plaintext from a finite set of plaintexts $\mathcal{P}$.
C denotes a ciphertext from a finite set of ciphertexts $\mathcal{C}$.
K_e, K_d denotes respectively the encryption and the decryption key from a finite set of keys $\mathcal{K}$.
Encryption and decryption are realized by

$$C = e_{K_e}(P) : \mathcal{P} \to \mathcal{C}.$$

$$P = d_{K_d}(C) : \mathcal{C} \to \mathcal{P}.$$

As decryption is supposed to be unique it is necessary that the encryption-decryption rule is injective.

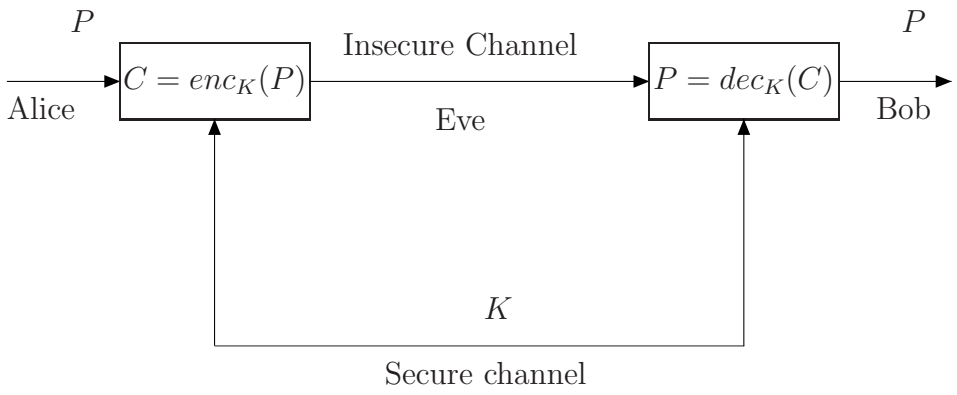


Figure 1.1: Secret key encryption

There are various ways to establish a cryptosystem and the solutions can be divided into the main categories of public key encryption and secret key encryption.
Secret key encryption is also called symmetric key encryption because Alice and Bob hold the same key information i.e. $K_d = K_e$. Figure 1.1 shows how secret key encryption, which is the objective of this project, works. Through a secure channel the key value is communicated to Alice and Bob respectively, and they are then ready to communicate using the secret key cryptosystem.
In a public key cryptosystem Alice has a public encryption key $K_{e_{Alice}}$ and a private decryption key $K_{d_{Alice}}$ (sometimes referred to as Alice's secret key). Whenever someone wishes to send an encrypted message to Alice all he has to do is obtain Alice's public key $K_{e_{Alice}}$ and encrypt the message by applying this in the encryption scheme. The schemes are constructed such that only a holder of Alice's private key $K_{d_{Alice}}$ is capable of decrypting messages encrypted with her public key $K_{e_{Alice}}$.
The public key schemes have the advantage that the sender and receiver do not need to share secret key information as opposed to the secret key systems. For the latter a secret key is needed for every two persons wanting to communicate. However, in practice the public key schemes are not efficient for encrypting a large amount of

data. The public key cryptosystems are therefore often applied to exchange keys for a secret key cryptosystem which is then used to encrypt the data.

1.2 Security evaluation

When analyzing a cryptosystem we want to learn something about the security of the cipher. Evaluating the security of a cipher or even better proving that it is in fact secure is not easy at all. For most of the encryption schemes in use there is no proof of security and they are always endangered by the possibility of new powerful attacks being invented. This brings us to Shannon's distinction [60] between the two basic kinds of security namely unconditional security and computational security.

Definition 1.2 (Unconditional security)**.** *A cipher has unconditional security if it is secure when the adversary has unlimited computational power.*

This type of security is based on probability theory because a cipher being unconditionally secure means that given everybody acts as the protocol prescribes, there is no better way for an adversary to determine the transmitted message than guessing. In other words plaintext and ciphertext sampling must be described as independent variables.

Corollary 1.1. *X and Y are independent variables if and only if*

$$pr(X = x|Y = y) = pr(X = x)$$

for all $x \in X$ and all $y \in Y$.

I.e. for every instance x of X and y of Y the probability of x occurring given y has already occurred equals the probability of x occurring with no constraint on the value of y. Thus knowing y tells us absolutely nothing about x. This is the essence of perfect secrecy.

Definition 1.3 (Perfect secrecy)**.** *A cryptosystem has perfect secrecy if*

$$pr(P = p|C = c) = pr(P = p)$$

for every plaintexts $p \in \mathcal{P}$ and every ciphertext $c \in \mathcal{C}$.

Perfect secrecy is obtained only if an adversary cannot extract any knowledge about the plaintext from knowing the ciphertext. Therefore the ciphertexts must be distributed statistically random over all choices of the key.

The simplest cipher with perfect secrecy is the Vernam Cipher which is also called the One-time pad, as the cipher obtains perfect secrecy when the keys are used only once. The Vernam cipher was proved secure under Shannon's definition of perfect secrecy.

Example 1.1 (The Vernam Cipher)**.** *The Vernam Cipher is a stream cipher over the alphabet* $\Lambda = \{0, 1\}$ *where* $|P| = |C| = |K|$.
A plain-text $P = p_0p_1p_2 \cdots p_{n_k}$ *is encrypted into a ciphertext* $C = c_0c_1c_2 \cdots c_{n_k}$ *using a randomly selected key* $K = k_0k_1k_2 \cdots k_{n_k}$ *where* $p_i, c_i, k_i \in \Lambda$ *for* $i = 0, \ldots, n_k$.
The encryption rule is defined by

$$e_K(P) = e_K(p_0, p_1, p_2, \cdots, p_{n_k}) =$$

$$(p_0 \oplus k_0, p_1 \oplus k_1, p_2 \oplus k_2, \cdots, p_n \oplus k_{n_k}) = c_0, c_1, c_2, \cdots c_{n_k} = C.$$

Since $x \oplus x = 0 \ \forall x \in \Lambda$ *we have that*

$$p_i \oplus k_i \oplus k_i = p_i.$$

Thus decryption is done in the same way as the encryption.
It is absolutely crucial for the security of this scheme that a key is used only once. If this rule is broken all an adversary has to do is add up two cipher-texts encrypted by the same key and this cancels out. Thus the adversary has the value of the two plain-texts exclusive-ored. This can be used in a redundancy study.
If the scheme is used correctly, that is if for every encryption a new random key is selected, then for every key bit k_i *of the key* K

$$pr(k_i = 0) = \frac{1}{2} \quad \text{and} \quad pr(k_i = 1) = \frac{1}{2}$$

hence

$$pr(p_i = 0|c_i = 0) = pr(k_i = 0) = \frac{1}{2}$$

$$pr(p_i = 0|c_i = 1) = pr(k_i = 1) = \frac{1}{2}$$

$$pr(p_i = 1|c_i = 0) = pr(k_i = 1) = \frac{1}{2}$$

$$pr(p_i = 1|c_i = 1) = pr(k_i = 0) = \frac{1}{2}$$

thus for every ciphertext bit c_i *in* $C = c_0c_1 \cdots c_n$

$$pr(p_i = 0) = pr(p_i = 1) = \frac{1}{2}$$

so the condition for perfect secrecy

$$pr(p_i = x_i | c_i = y_i) = pr(p_i = x_i)$$

is accomplished.

Clearly unconditional security is desirable and an ideal for cryptographic primitives. However, it is not practical that the key must be the same length as the plaintext, when it is only used once, as is the case in the One-time pad. Therefore the security of most practical ciphers is based on the computational concept of security rather than unconditional security.
The second type of security defined by Shannon is:

Definition 1.4 (Computational security)**.** *The adversary is modeled to have polynomial computational power. The security is evaluated as the feasibility for the adversary to compute the key using known methods for cryptanalysis.*

In analysis of ciphers different methods, called attacks, are used to estimate the computational requirements for breaking the cipher. The assumption of the Dutch Auguste Kerckhoffs, first time published in 1883 in "Journal des sciences militaires", is the basis of all modern cryptanalysis.

Theorem 1.1 (Kerckhoffs' Principle)**.** *A cryptosystem should be secure even if everything about the cipher, except the key, is public.*

In practice governments, banks etc. apply secret algorithms while civilians make use of public algorithms. In any case Kerckhoffs' principle forms the basis of modern cryptanalysis and invokes different attack scenarios. Figure 1.2 lists these scenarios in descending order of severity.
Some attacks might only have limited application as they require a larger degree of control than the attacker usually possesses. However any attack violating the security threshold is considered a threat to the cryptosystem because it is often possible to convert it into an attack that requires a lower degree of control by for example applying more texts and/or executing more computational work.
It is not clearly defined what a successful attack is. In fact, the concept is used in very different ways. The basic method of breaking ciphers is by exhaustive key search (also called brute force). This security of a cipher based on computational security is bounded by the computational requirements of this attack.

Definition 1.5 (Exhaustive key search)**.** *The method of trying all keys until the right one is found is called exhaustive key search. On average it requires testing half the keys.*

ciphertext-only: The adversary has access to ciphertexts C only. This is weakest attack type, and assumed always possible.

known-plaintext: The adversary has knowledge of plaintext-ciphertext pairs (P, C) but no control of which pairs.

chosen text attack: The adversary can choose the plaintexts P and obtain the corresponding ciphertexts C or vise versa.

adaptively chosen text attack: A chosen text attack where the adversary chooses the texts adaptively.

Figure 1.2: Attack types

A cipher can always be broken by exhaustive key search provided the needed text pairs are available. Therefore a cipher being computationally secure implies that succeeding by exhaustive key search is computationally infeasible. In practice this implies that the key must be of a certain size that makes exhaustive key search intractable.

Definition 1.6 (Total break)**.** *If it is computationally feasible to determine the key, the method by which this is done provides a total break.*

When a total break is achieved the cipher is clearly not suitable for practical use but this is not a very good definition for setting a bound on the security of the cipher. The computational requirements for doing exhaustive key search specify when a method of analysis provides a cryptographic break on a cipher.

Definition 1.7 (Cryptographic break)**.** *If it is theoretically possible to compute the key faster than by exhaustive key search, the method by which this is done provides a cryptographic break.*

It is obviously safer to set security bounds from this definition. However, the fact that a cryptographic break exists does not necessarily imply that an attack is feasible in practice. If for instance the key length is 128 bits then we would have to try approximately 2^{127} different keys in an exhaustive key search. This means an attack determining the key using computational resources of 2^{124} encryptions would do better than exhaustive key search but this is still an intractable job.
The computational requirements can be divided into three categories of complexity.

- *Processing complexity:* The time needed to do the computations.

- *Data complexity:* The amount of data (texts) needed to determine the key.
- *Storage complexity:* The space needed to store data.

Ciphers are designed so that the complexities of the known attacks are maximized while the complexities of encryption are minimized. On one hand, an attack usually has an impact on the cipher when the complexities of the attack is reduced substantially compared to exhaustive search. On the other hand, an attack that compromises the security bound of exhaustive key search reveals an undesirable weakness of the cipher.
It is always preferable for cryptanalysts to break a cipher by minimizing the computational requirements. If for instance an attack requires a large amount of known or chosen text pairs one might argue that it would not be possible to break the cipher in practice. So even if a cipher has already been broken one still tries to find new attacks to see whether it is possible to improve the complexities of the break.
The attacks can be considered as methods of solving problems. The security of a cipher is therefore based on the hardness of solving the problems which the methods of analysis give rise to. This approach is due to Shannon's principle of "reducibility to a known problem". If a specific type of cryptanalysis can be reduced to solving an instance of a problem believed to be difficult, then breaking the cipher by this approach is guaranteed intractable on condition that the underlying problem is intractable. This principle is used in the RSA where the security relies on the hardness of factoring products of big primes. Another example is the Diffie-Hellmann scheme based on the discrete logarithm problem being difficult (given p, x, $a = x^b \mod p$, for a prime p, find b). A class of problems which is of interest for the main objective in this project is the so-called NP-complete problems.

1.3 NP-completeness

The P problems are a subset of the complexity class NP problems.

Definition 1.8 (Complexity class **P**)**.** *The complexity class* **P** *is the set of all decision problems that are solvable in polynomial time.*

Being solvable in polynomial time $O(n^k)$ means that there exist an deterministic algorithm that solves the problem. The running time of the algorithm, that is the number of bit operations for solving the problem, is bounded by a constant degree k polynomial $f(n)$ as a function of the number of input bits n.

$$f(n) = a_0 + a_1 n + \cdots + a_k n^k$$

where the coefficient $\{a_0, a_1, \cdots, a_k\} \in \mathbb{R}$.

Definition 1.9 (Complexity class **NP**)**.** *The complexity class* **NP** *is the set of all decision problems for which a "yes" answer can be verified in polynomial time given some extra information, called a certificate.*

The certificate is some additional information that makes it possible to verify a yes-answer. The equivalent problem of verifying a no-answer problem is called a co-NP problem.

Definition 1.10 (Complexity class $\mathbf{co-NP}$)**.** *The complexity class co-NP is the set of all decision problems for which a no answer can be verified in polynomial time given a certificate.*

If a problem is a class **P**-problem it is therefore also a class NP-problem since if one can find a solution in polynomial run time one can certainly verify the solution in this run time or better.

Definition 1.11 (**NP**-complete problem)**.** *A decision problem is said to be* **NP***-complete if it is contained in the class of* **NP** *complex problems, and if there is a polynomial reduction from any other decision problem to this problem in* **NP***.*

In general we believe that **NP**-complete problems are not **P**-problems and there are only non-deterministic algorithms for solving these in polynomial time.
We now introduce the satisfiability problem [34]. For this we need the following concepts:

Definition 1.12. *Literals are variables or negations of variables. A clause is a disjunction of literals.*

A satisfiability problem is a conjunction of clauses. A solution to a satisfiability problem is an assignment of the literals that satisfies all the clauses raised by the problem.

Example 1.2. *Let for instance*

$$V = \{v_0, v_1, v_2, \cdots, v_n\}$$

be a set of literals in $GF(2)$*. Any clause of assignment of this set of literals is a function which is either true or false. The clause*

$$C_1 = v_0 \vee v_1 \vee \widehat{v_4}$$

will be true only if v_0 *is true or* v_1 *is true or* v_4 *is false. A solution to a satisfiability problem is an assignment of the literals that makes every clause* $C_1, C_2, \cdots C_m$ *for the given problem true.*

Theorem 1.2 (Cook's Theorem)**.** *The Satisfiability problem is* **NP***-complete.*

We believe that there is no deterministic algorithm for solving the satisfiability problem in polynomial time. However, an assignment of literals can be verified true in polynomial time. The satisfiability problem is particularly relevant to this thesis since the objective of the algebraic attacks, namely solving certain algebraic equations, is equivalent to solving an instance of the problem [4]. Chapters 4, 5, 6, 7 and 8 concern, in each their way, methods for solving algebraic equations over block ciphers.

Chapter 2

Block Ciphers

Block ciphers are cryptographic primitives which allows encryption of large amounts of data in a fast and secure way. The ciphers fall in the category of symmetric encryption where we assume that the secret key is pre-distributed. A block cipher is defined as follows:

Definition 2.1 (Block cipher)**.** *A block cipher is a collection of bijections. Under a secret key K the cipher encrypts a plaintext block P of length n_b into a ciphertext block C of equal length. Encryption is realized using the encryption function*

$$C = e_K(P) : \{0,1\}^n \rightarrow \{0,1\}^n.$$

Most block cipher designs aim towards providing a high level of security, efficiency, and applicability in various software and hardware environments. A popular choice of design structure is an SP-network.

Definition 2.2 (SP-network)**.** *An iterated block cipher where the round function is composed of a substitution layer (sBoxLayer), a linear permutation layer (pLayer) and linear key application is called an SP-network.*

The substitution layer provides non-linearity to the cipher. Without this the whole cipher could be written as a series of linear expressions which are easily solved by linear algebra. The substitution layer is often realized by parallel application of small non-linear functions called S-boxes. The permutation layer provides diffusion, that ensures that the output of the substitution layer is mixed thoroughly across the entire block. For a cipher that has good diffusion a small change in the plaintext causes a big change in the ciphertext.
The Data Encryption Standard (DES) is perhaps the most applied block cipher in the world for which reason it has been described many places in literature (e.g.

[52, 60]). In the following section we provide a short description of the cipher not only for completeness of the thesis but also because we in Chapter 8 apply a new technique on the DES S-boxes.

2.1 Data Encryption Standard

Many people have heard of the Data Encryption Standard and most people are in daily contact with it (or at least its variant triple-DES). For the past three decades it had been the most applied block cipher and though it has been withdrawn as a Federal Information Processing Standard (FIPS) standard more than three years ago it remains in use many places in society. The cipher dates back to the early 1970'es where the NBS (National Bureau of Standards), today known as the NIST (National Institute of Security and Technology), made a public request for proposals for a new cryptographic standard. The design criterion were security, availability to the public and low price in implementation. None of the candidates were found suitable for the standard, and a new request was issued. This time IBM submitted a proposal based on a cipher called Lucifer [30]. The cipher belongs to a special category of block ciphers called Feistel networks (named after Horst Feistel), the structure is shown in Figure 2.1.

Definition 2.3 (Feistel Network)**.** *An SP-network where only half the block is transformed in each round is called a Feistel network. Let C_i^R be the right half and C_i^L be the left half of the text block, and let f be the round function, then*

$$(C_i^L, C_i^R) = (C_{i-1}^R, f(C_{i-1}^R) \oplus C_{i-1}^L)\,.$$

The NSA (National Security Agency) was responsible for the security evaluation of DES and recommended to IBM some modifications of Lucifer. As a result the key size was reduced from 128 bits to 56 bits, and the block size from 128 bits to 64 bits. Moreover the the S-boxes, which are the only non-linear part of the cipher, were changed to improve the security of the cipher. The result was DES which became the official encryption standard in November 1976, certified for protecting sensitive unclassified federal data. Because of the secrecy governing the work of NSA many critics have questioned whether the changes made to Lucifer introduced a hidden trap door in DES. Other argued that the key size was too short for long term security.

2.1.1 The algorithm

DES applies a 56-bit cipher key to encrypt a 64-bit block of text (the standard specifies a 64-bit key, however 8 bits only serve as parity check and are not counted

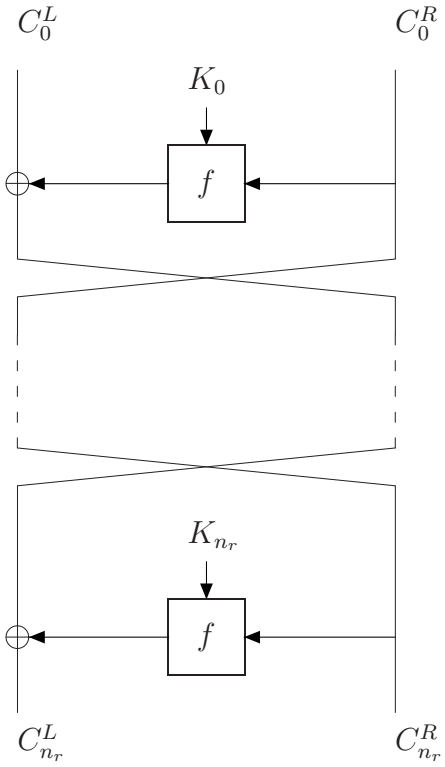


Figure 2.1: Feistel Network. The ciphertext $C = C_{n_r}^L, C_{n_r}^R$ is computed from the plaintext $P = C_0^L, C_0^R$ through n_r rounds of encryption using n_r round keys.

as part of the cipher key). The plaintext is processed by an initial permutation (IP), after which the text is encrypted with a sixteen round Feistel structure (as shown in Figure 2.1). The output of the final round is subjected to a final permutation defined as the inverse of the initial permutation (IP^{-1}).

In each round the round function of DES $f(C_{i-1}^R, K_i)$ encrypts the right half of the text block as follows (see Figure 2.2): The 32-bit block C_i^R is expanded to a 48-bit block by an expansion function that repeats certain bits. The output of the expansion function is exclusive-ored with a 48-bit round key K_i. The 48-bit block is substituted into a 32-bit block, using the eight DES S-boxes. Each S-box substitutes 6 bits into a 4 bits (hence the 48 to 32-bit conversion). Finally, the round function applies a fixed permutation to the 32-bit block.

The round keys are derived by a key schedule that basically selects 48 bits of the cipher key for each round key. The DES decryption is exactly the same as the

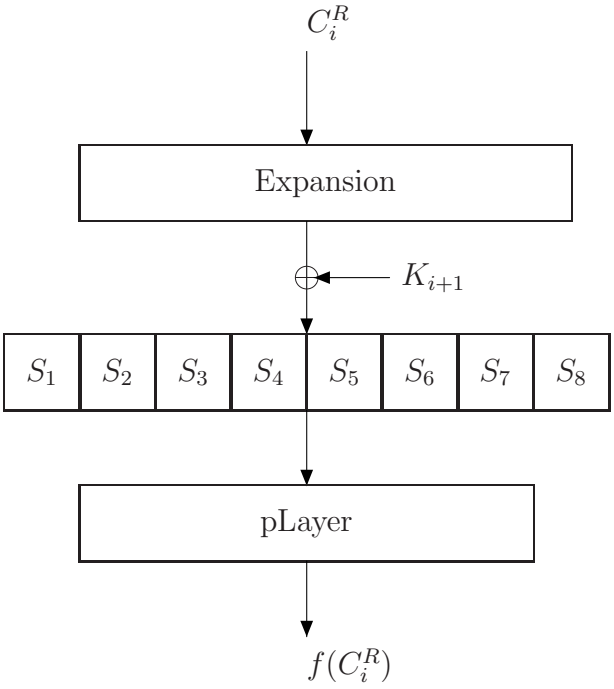


Figure 2.2: The round function $f(\cdot)$ of DES.

encryption, only with the round keys used in reverse order.

2.2 Between standards

As time went by and computation power increased, DES became vulnerable to brute force attacks due to the relative short key length of 56-bit. As a simple enhancement of DES the triple-DES was specified [27]. As indicated by its name the cipher is built from three DES encryption in sequence using three DES keys. While one might think this scheme provides 168-bit security in fact it offers only 112-bit security due to a meet in the middle attack. Another version called two-key triple DES was proposed two enable communication between implementations of respectively single and triple DES. This version is composed of one DES encryption with a first key (K_1), one DES decryption with a second key (K_2) and a final DES encryption with

the first key (K_1). Two-key Triple DES operates as single DES when the user selects $K_2 = K_1$.

2.3 The Advanced Encryption Standard

On September 12, 1997 NIST called for proposals for a new encryption standard to be named the Advanced Encryption Standard (AES). The algorithm was to replace DES and it should be at least as safe as triple-DES while much faster. Fifteen proposals were submitted and after two AES conferences (AES-1 and AES-2) NIST announced in August, 1999, the five finalists: MARS [13], RC6 [57], Rijndael [24], Serpent [6], and Twofish [32]. A third conference followed (AES-3) before NIST on October 2, 2000, announced that the cipher Rijndael, proposed by the Belgians Daemen and Rijmen, had been selected as the new standard AES. On May 19, 2005 (single) DES was finally withdrawn by NIST as FIPS 46-3 [53].
The AES is an iterated block cipher of the type SP-network (see Definition 2.2). There are three variants of the AES, namely a 128-bit, a 192-bit, and a 256-bit key version. In this thesis we often refer to the 128-bit version as AES-128. All three versions are categorized as strong encryption. The number of rounds n_r depends on the key size:

128-bit key, $n_r = 10$

192-bit key, $n_r = 12$

256-bit key $n_r = 14$

The AES proposal Rijndael has variable block size, but for AES the block size is fixed to 128 bit. The 128-bit text block of AES, which is often referred to as the state is arranged in a 4×4 array of 8-bit words.

2.3.1 Encryption

The round function is composed of four sub-functions in the following order (see Figure 2.3):

- SubBytes (substitution layer)
- ShiftRows (part of the linear layer)
- MixColumns (part of the linear layer)
- AddRoundKey (key application)

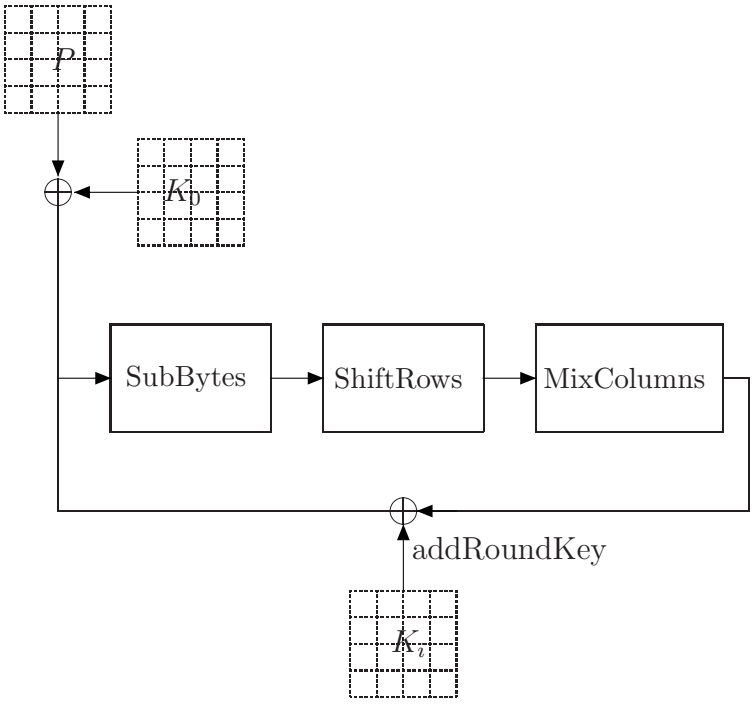


Figure 2.3: Top level description of AES

Initially a pre-whitening key K_0 is exclusive-ored with the plaintext. Next the round function is applied $n_r - 1$ times. Finally, a last round is performed in which the MixColumns function is omitted. This enables decryption to be performed analogous to encryption. Note that the linear layer after the last substitution layer does not improve the security of the cipher. This is due to the fact that the key application is also a linear function, which allows the cryptanalyst to swap the order of the linear layer and the key application and hence peel off the linear layer of the final round.

2.3.2 The round function

Several of the functions operate in the finite field $GF(2^8)$. The field is constructed using the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, referred to as the Rijndael polynomial. The field is denoted by F and θ is a root of $m(x)$:

$$F = GF(2)[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle = GF(2)(\theta).$$

Each 8-bit word is represented by a polynomial in θ where the most significant bit is the coefficient of θ^7.

2.3.3 The substitution layer

The substitution layer operates simulations on the sixteen 8-bit words. Each word is substituted using the 8-bit to 8-bit AES S-box. The S-box is composed of the inversion map

$$X^{-1} \quad \text{over} \quad GF(2^8)\,,$$

for $X \neq 0$, and

$$X \to X\,,$$

for $X = 0$. The inversion is followed by a $GF(2)$-linear map defined as

$$g(X) = X(x^7 + x^6 + x^5 + x^4 + 1) \mod x^8 + 1$$

where X is an element of $F = GF(2^8)$. SubBytes is finalized by addition with the S-box constant 63_x (in hexadecimal notation). The $GF(2)$-linear map is also specified by an 8×8 matrix over $GF(2)$ in [54]. In practice the S-box is realized by a lookup table with 256 entries that inputs one byte and outputs one byte.
The design rationale is that the inversion map provides good resistance towards linear and differential cryptanalysis (see Chapter 3). The $GF(2)$-linear map and the S-box constant are introduced to increase the algebraic complexity.

2.3.4 Diffusion layer

The diffusion/permutation layer is composed of the functions ShiftRows and MixColumns. ShiftRows is defined as cyclic rotations to the left, of the i'th row of the data block, by i positions. MixColumns is designed to create good byte level diffusion in the 32 bits of each column of the state. In MixColumns the columns of the state are considered as polynomials over F^4, i.e. polynomials of degree less than four with coefficients in F. Each column vector $a(X)$ is multiplied by $c(X)$ modulo $X^4 + 1$ to obtain the output column $b(X)$ (see Figure 2.4):

$$b(X) = a(X) \cdot c(X) \mod X^4 + 1\,,$$

where

$$\begin{aligned} c(X) &= (\theta + 1)X^3 + (1)X^2 + (1)X + \theta \\ &= 03_x X^3 + 01_x X^2 + 01_x X + 02_x \end{aligned}$$

(the second representation is in hexadecimal notation). $c(X)$ is not an irreducible polynomial but since it is co-prime to $X^4 + 1$ its multiplicative inverse ($c^{-1}(X)$) is well defined:

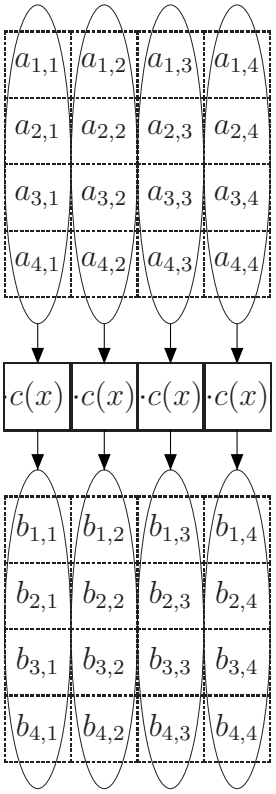


Figure 2.4: The MixColumns function

$$c^{-1}(X) = (\theta^3 + \theta + 1)X^3 + (\theta^3 + \theta^2 + 1)X^2 + (\theta^3 + 1)X + (\theta^3 + \theta^2 + \theta)\,.$$

MixColumns is inverted by multiplying each column vector by $c^{-1}(X)$ modulo X^4+1. The MixColumns transformation can be rewritten as multiplication of the rows in the state by the matrix $\underline{\underline{A}}$ in the field F.

$$\underline{\underline{A}} = \begin{pmatrix} \theta & (\theta+1) & 1 & 1 \\ 1 & \theta & (\theta+1) & 1 \\ 1 & 1 & \theta & (\theta+1) \\ (\theta+1) & 1 & 1 & \theta \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}.$$

θ is a root of $m(x)$ in $GF(2^8)$.

Example 2.1. *Let* $a = (20_x, 40_x, 02_x, 02_x)^T$

be a column of the AES state. This is represented by

$$\begin{aligned} a(X) &= 02_x X^3 + 02_x X^2 + 40_x X + 20_x \\ &= (\theta) X^3 + (\theta) X^2 + (\theta^6) X + \theta^5 \end{aligned}$$

in F^4*. According to the definition of MixColumns we compute*

$$b(X) = a(X)c(X) =$$
$$(\theta^5 + \theta^2 + \theta) X^3 + (\theta^6 + \theta^5 + \theta) X^2 + (\theta^7 + \theta^5 + \theta^2) X + \theta^7 .$$

corresponding to the state column $b = \begin{pmatrix} 80_x \\ a4_x \\ 62_x \\ 26_x \end{pmatrix}$*, which is the same as we obtain from the computation* $\underline{b} = \underline{\underline{A}}\underline{a}$.

2.3.5 Key application

The function AddRoundKey exclusive-ores a 128-bit round key with the 128-bit state. The round keys are derived using the key schedule.

2.3.6 Key schedule

The AES round keys are generated from the cipher key using a key expansion function. For AES-128 the function applies four S-boxes per round to generate the sixteen 8-bit words of the round key.
The 32-bit words w_i each represent one column of a round key. For AES-128 the round keys are derived as follows: The cipher key

$$K = (w_3, w_2, w_1, w_0)$$

is loaded directly into the first round key $K^{(0)}$.
All subsequent round keys

$$K^{(i)} = (w_{3+4i}, w_{2+4i}, w_{1+4i}, w_{4i}) ,$$

for $i = 1, \ldots, 10$, are derived by passing 32 bits of the previous round key (the key word $w_{3+4(i-1)}$) to the function RotWord which performs cyclic shifts on byte level. The output of RotWord is transformed by four S-box applications and finally exclusive-ored by a round constant $const_{i-1}$. The round key is then derived linearly from the 32-bit output of these S-boxes and the previous round key as shown in Figure 2.5. The key schedule is written in details (for later use) in Appendix A.2.

2.4 Modes of operation

Block ciphers operate on fixed length blocks while a message can have any length. To accommodate for this five modes of operation have been defined. Given a block cipher $e_K(\cdot)$ with specified block length of n_b bits, a plaintext P padded into a bit string of $n_b \cdot l$ bits

$$P = P_1, \ldots, P_l$$

is encrypted into a ciphertext

$$C = C_1, \ldots, C_l$$

by applying one of the following four modes of operations.
The **Electronic CodeBook (ECB) mode** applies the block cipher to encrypt plaintext blocks one by one, i.e.

$$C_i = e_K(P_i) \quad \text{for} \quad i = 1, \ldots, l.$$

This means that identical blocks of the plaintext are encrypted to identical blocks of ciphertext, and hence one might recognize patterns in the ciphertext.
The **Cipher Block Chaining (CBC) mode** addresses the problem of pattern recognition by exclusive-oring the ciphertext of the previous block to the input of the block cipher, i.e.

$$C_i = e_K(P_i \oplus C_{i-1}) \quad \text{for} \quad i = 1, \ldots, l\,,$$

where C_0 is initialized to some initial value IV.
In **Cipher FeedBack (CFB) mode** the plaintext P_i is exclusive-ored with the text block $e_K(C_{i-1})$ obtained by encryption the previous ciphertext block C_{i-1} under the key K

$$C_i = P_i \oplus e_K(C_{i-1}) \quad \text{for} \quad i = 1, \ldots, l\,,$$

where where C_0 is initialized to an initial value IV.
Output FeedBack (OFB) mode respectively **Counter (CTR) mode** are stream cipher modes. They both apply the block cipher to produce a key stream (independent of the text) which is exclusive-ored by the plaintext. The advantage of these is that they actually don't require padding of the plaintext since excessive keys bits are simply discarded. Moreover, the counter mode offers parallel encryption of the blocks without precomputing the entire key stream. The reader is referred to [60] for details on modes of operation.

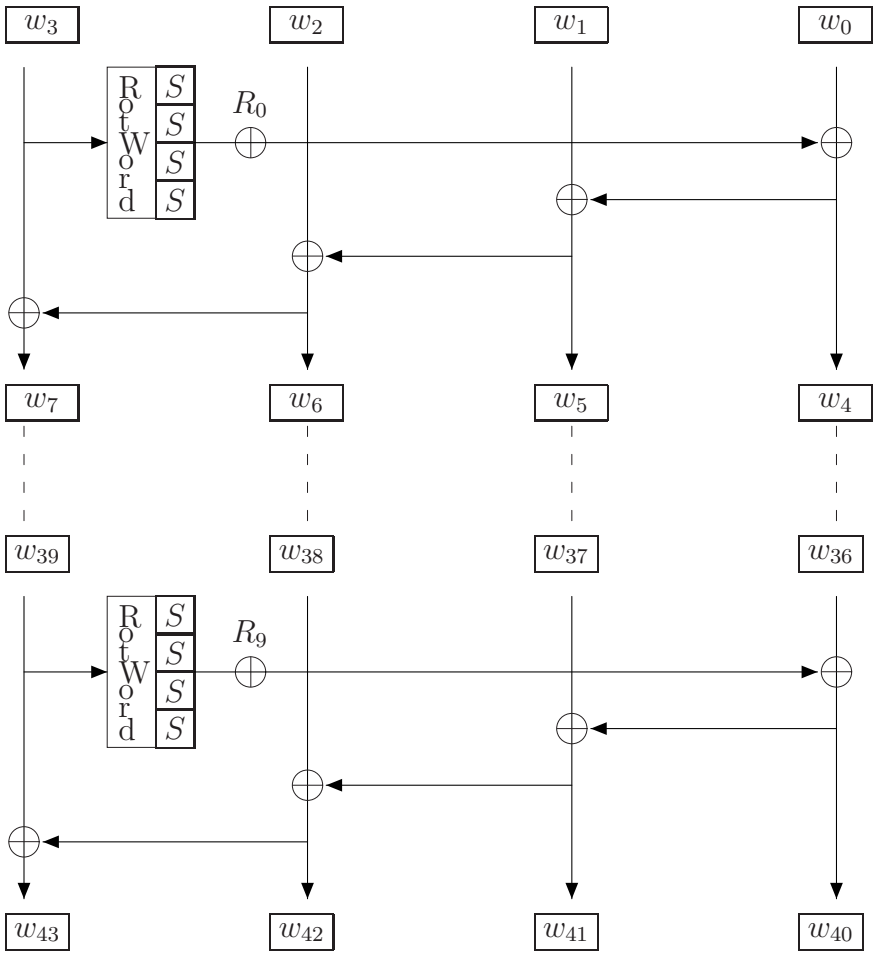


Figure 2.5: Key schedule of AES-128

Chapter 3

Block Cipher Cryptanalysis

The security of block ciphers is continuously evaluated by cryptographers worldwide. The objective is to examine the resistance of the designs towards various types of analysis. In this chapter we first describe the generic attacks namely exhaustive key search, the table lookup attack, and a clever extension of this, the time memory trade-off attack. An important bound on the security offered by block ciphers follows from these attacks. Also, we provide a short introduction to some of the most powerful methods of analysis on block cipher, namely linear and differential cryptanalysis. Finally, we give a short introduction algebraic cryptanalysis which is the main objective of this thesis.

3.1 Exhaustive key search

In Chapter 1 we described the most general attack that applies to any cipher, namely exhaustive key search. Let $e_K(\cdot)$ be a cipher that maps a plaintext P to a ciphertext C, under a secret key $K \in \mathcal{K}$ (where $\mathcal{K}$ is the key space),

$$C = e_K(P)\,.$$

In an exhaustive key search the attacker exhaustively tries all keys until he finds a key K that encrypts P to C. If the key length equals the block length ($n_b = n_k$), the attack requires on average one known text pair (P, C) and exhaustive key search takes on average 2^{n_k-1} operations, when $|\mathcal{K}| = 2^{n_k}$ and one operation is defined as the time spend on one encryption. If the key length is larger than the block length ($n_b > n_k$) the attacker will have to apply more texts for the attack to succeed.

3.2 Table lookup attack

The idea of the table lookup attack is to reduce the online computation time, by doing almost all the work in the precomputation. For this we require some known plaintext prior to the attack. In the online phase we need pairs where the plaintext equals those which were specified in the precomputation phase. The attack therefore falls into the category of chosen plaintext attacks. Again, let $e_K(\cdot)$ be a cipher that maps a plaintext P to a ciphertext C, under a secret key $K \in \mathcal{K}$.

Precomputation Given a plaintext $P = P_0$ generate for each key $K_i \in \mathcal{K}$ the corresponding ciphertext

$$C_i = e_{K_i}(P_0).$$

Sort the pairs C_i, K_i with respect to the ciphertext in a table. If the size of the key space is $|\mathcal{K}| = 2^{n_k}$ the precomputation takes 2^{n_k} operations (one operation correspond to an encryption). The memory required to store the table is 2^{n_k} ciphertext blocks.

Online If $|\mathcal{C}| = |\mathcal{K}|$ (where $\mathcal{C}$ is the ciphertext space) we simply lookup the key in the table at the entry of the ciphertext $C = C_0$. The time to perform the online part of the attack is constant. If $|\mathcal{C}| < |\mathcal{K}|$ we need to apply more texts i.e. a larger table to realize the attack.

For as long as the plaintext stays the same the table generated in the precomputation phase can be used again to determine the secret key. This could occur in for example emails where a certain part of the text is known always to be the same. For realistic sized block ciphers, e.g. a block cipher with a 128-bit key, both the time for doing the precomputation and the memory required to store the table make the attack infeasible.

3.3 Cryptanalytic time-memory trade-off

The Time-Memory Trade-off attack was proposed in [37] by M. Hellman. It applies not only to block ciphers but to any random function, e.g. to hash functions as well. Like in the generic table lookup attack the precomputation phase applies a chosen plaintext, and is therefore categorized as a chosen plaintext attack. The attack was originally designed for DES, for which reason we in this description assume that $|\mathcal{C}| \geq |\mathcal{K}|$.
Let $e_K(P)$ be a block cipher that maps an n_b-bit plaintext into a n_b-bit ciphertext using a n_k-bit secret key.

Let P_0 be a fixed plaintext. Define

$$f(K) = Red(e_K(P_0))$$

where $Red(\cdot)$ is a reduction function that maps elements of the ciphertext space $\mathcal{C}$ into the key space $\mathcal{K}$. For ciphers with $n_k = n_b$, a straightforward choice of f would be the identity map. The reduction function should be chosen such that computing $f(K)$ should be almost as fast as computing $e_K(P_0)$. Note that computing K from $f(K)$ is equivalent to cryptanalysis. The attack is as follows:

Precomputation Choose m starting points $SP_1, SP_2, \ldots, SP_m$ at random from the key space. Set $X_{i,0} = SP_i$ and compute from each starting point a chain of t elements

$$X_{i,j} = f(X_{i,j-1}), \quad j \in [1, \ldots, t],$$

as shown in Figure 3.1. $EP_i = X_{i,t}$ is the endpoint we reach upon t iterations of f starting at the point SP_i. Table 3.1 contains the m chains. The chains are sorted by endpoints and all intermediate points are discarded to save memory. The result of the precomputation phase is in Table 3.2.

Online computation The attacker intercepts a ciphertext C_0 and checks if $Red(C_0)$ is among the endpoints in Table 3.2.

If $EP_j = Red(C_0)$, either the key is given by $X_{j,t-1}$ or we have a *false alarm*.

Else iteratively compute

$$C_0^l = f^l(Red(C_0)) = f(f(\cdots f(Red(C_0)))), \qquad 1 \leq l \leq t-1$$

starting by $l = 1$ and then increasing the value of l until we discover C_0^l among the endpoints of Table 3.2. If $C_0^l = EP_j$ then either the key equals $X_{j,t-l-1}$, or we have a *false alarm*.

A *false alarm* occurs because $f(K)$ is a random function and not necessarily injective. I.e. there might be more keys than the cipher key that satisfy $C_0 = e_K(P_0)$. These invoke a false alarm.
If we assume that all elements of Table 3.1 are distinct, the probability that the key is discovered in the procedure described previously is

$$P(S) = \frac{m \cdot t}{2^{n_k}}.$$

Table 3.1: Table $\underline{\underline{T_1}}$

$X_{1,0} = SP_1$	$\xrightarrow{f}$	$X_{1,1}$	$\xrightarrow{f}$	$X_{1,2}$	$\cdots$	$\xrightarrow{f}$	$X_{1,t} = EP_1$
$X_{2,0} = SP_2$	$\xrightarrow{f}$	$X_{2,1}$	$\xrightarrow{f}$	$X_{2,2}$	$\cdots$	$\xrightarrow{f}$	$X_{2,t} = EP_2$
$X_{3,0} = SP_3$	$\xrightarrow{f}$	$X_{3,1}$	$\xrightarrow{f}$	$X_{3,2}$	$\cdots$	$\xrightarrow{f}$	$X_{3,t} = EP_3$
$\vdots$							
$X_{m,0} = SP_m$	$\xrightarrow{f}$	$X_{m,1}$	$\xrightarrow{f}$	$X_{m,2}$	$\cdots$	$\xrightarrow{f}$	$X_{m,t} = EP_m$

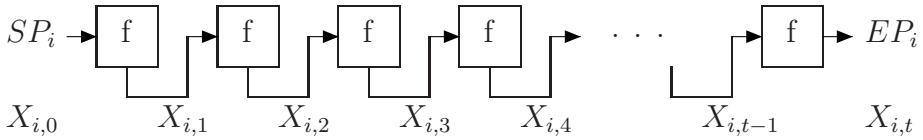


Figure 3.1: Generating a chain

However, since f is a random function we expect from the birthday paradox (Corollary 3.1) to encounter a collision in the output upon $2^{n_k/2}$ generated elements. Due to the nature of the chains they enter a cycle from this point on and no new elements are generated as shown in Figure 3.2.

Corollary 3.1 (Birthday Collisions)**.** *Suppose a random function $f : \mathcal{C} \to \mathcal{K}$ where $|\mathcal{K}| = 2^{n_k}$ one expects a collision in about $2^{n_k/2}$ evaluation of elements chosen at random from $\mathcal{C}$*

As shown in Figure 3.3 the problem occurs both within a single chain, that at some point starts repeating itself, and for several chains.
For this reason the size of Table 3.1 must be chosen such that there is only a reasonable number of overlaps while the probability of success is good. In [37] the bound

Table 3.2: Table $\underline{\underline{T_2}}$

$X_{1,0} = SP_1$	$X_{1,t} = EP_1$
$X_{2,0} = SP_1$	$X_{2,t} = EP_2$
$X_{3,0} = SP_1$	$X_{3,t} = EP_3$
$\vdots$	
$X_{m,0} = SP_1$	$X_{m,t} = EP_1$

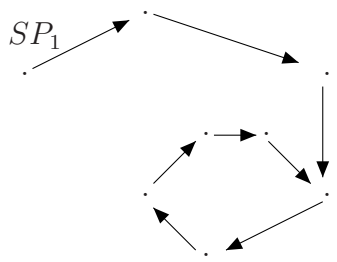


Figure 3.2: Chain repeating itself

for this is derived to be $m \cdot t^2 = 2^{n_k}$.
If we fix $m = t = 2^{n_k/3}$ this is fulfilled but as stated in [37] the probability of success is only is $P(S) = 0.8 \cdot 2^{-n_k/3}$.
To obtain a higher probability one chooses r reduction functions at random

$$f_1, f_2, \ldots, f_r$$

and computes for each function a table as before. Since the functions are random but distinct, there will only be overlaps in the elements of distinct tables according to the birthday paradox. This means that the probability of success when applying all r tables in the attack is given by

$$P(S) = r \cdot 0.8 \cdot 2^{-n_k/3}.$$

By choosing the parameter $r = t = m = 2^{n_k/3}$ we obtain a high probability of success. The memory consumption is $2^{2n_k/3}$ and the online computation time is $2^{2n_k/3}$.
The online phase of both the table lookup and the time-memory attack is really fast. The advantage of the time-memory trade-off is the memory requirements. In Table 3.3 we list the requirements of respectively the exhaustive key search, the table look-up, and the time-memory trade-off attacks.

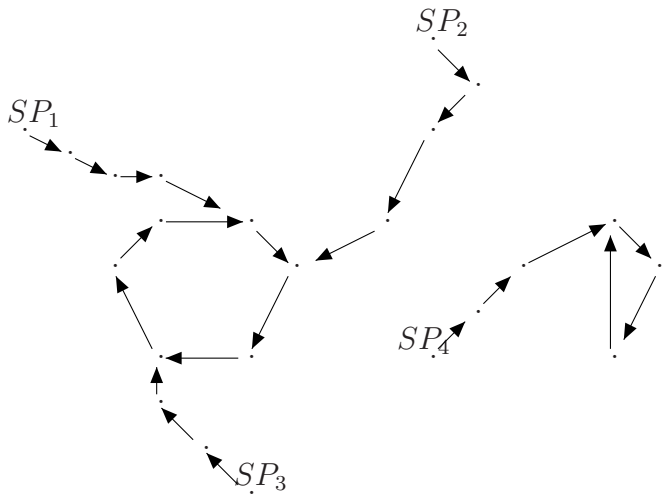


Figure 3.3: Chains overlapping

Table 3.3: Comparing Exhaustive key search, table look-up, and time-memory trade-off

	exhaustive key		table look-up		time-memory	
	pre.	online	pre.	online	pre.	online
time	0	2^{n_k}	2^{n_k}	small	2^{n_k}	$2^{n_k/3}$
memory	0	0	2^{n_k}	2^{n_k}	$2^{2n_k/3}$	$2^{2n_k/3}$

3.4 Differential cryptanalysis

Differential and linear cryptanalysis are the two most general techniques on block ciphers. Differential cryptanalysis was first presented by Biham and Shamir on DES-like cryptosystems [7]. It was noted that DES shows some resistance to differential cryptanalysis which gives reason to believe that the designer (or most likely the NSA) possessed knowledge of the technique at least ten years earlier, back when DES was designed.

A few years later, in 1992, linear cryptanalysis was discovered by Matsui. The overall structure of the technique is similar to differential cryptanalysis. Both attacks are probabilistic techniques that require a significant amount of text.

Differential and linear cryptanalysis have had deep impact on today's ciphers and it is a mandatory part of the security evaluation to investigate the strength of a cipher's against these techniques. The attacks apply not only to block ciphers but

to hash functions and stream ciphers as well.
As implied by its name, differential cryptanalysis is analysis of the relations in text-differences in succeeding rounds of the cipher. The objective is to analyze how differences propagate throughout a cipher. Given an n_r-round cipher the attacker finds a so-called differential characteristic

$$\alpha_0, \alpha_1, \ldots, \alpha_{n_r-1}$$

over n_r-1 rounds of the cipher. The characteristic predicts, with probability p_x, that a text pair with input difference $\Delta C^{(0)} = \alpha_0$ follows the differential characteristic and upon $n_r - 1$ rounds of encryption has the difference $\Delta C^{(n_r-1)} = \alpha_{n_r-1}$. This relation is exploited to distinguish the right from wrong guesses of the final round key. The attack is a chosen plaintext attack because the attacker needs control of differences between plaintexts.

Definition 3.1. *The difference Δ between two texts m_1, m_2 is defined by*

$$\Delta m = \Delta(m_1, m_2) = m_1 \oplus m_2^{-1}$$

where m_2^{-1} is the inverse of m_2 with respect to the group operator $\oplus$.

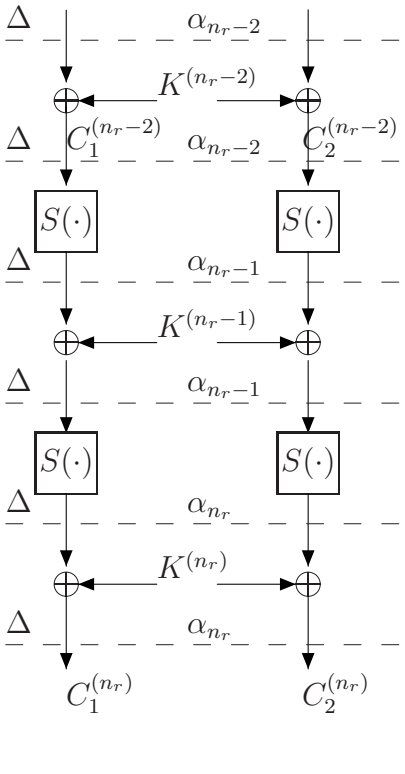
Differential cryptanalysis is based on the observation that differences are non-deterministic for non-linear functions as opposed to linear/affine function. As described in Chapter 2 the non-linear layer of most block ciphers is built from parallel application of small non-linear functions called S-boxes.Throughout the examples of this chapter, we consider a small toy cipher for which the round function $F(C^{(i)}, K^{(i+1)})$ is given by

$$C^{(i+1)} = F(C^{(i)}, K^{(i+1)}) = S(C^{(i)}) \oplus K^{(i+1)}$$

where $S(\cdot)$ is the non-linear S-box given in Table 3.4. The ciphertext $C^{(n_r)}$ is computed by iterating the round function n_r times, where each round applies a different round key chosen uniformly random from the key space. The input to the first round is computed by exclusive-oring the plaintext P with an initial whitening key $K^{(0)}$

$$C^{(0)} = P \oplus K^{(0)}.$$

In Figure 3.4 we follow the difference $\Delta = \alpha_i$ between two texts in the last two rounds of encryption.
The core of differential cryptanalysis is to find high-probability differentials that describe the input to output relation of the non-linear function. A differential $(\alpha \rightarrow \beta)$ is defined by an input difference α and an output difference β. The probability



x

Figure 3.4: Difference between two texts in the last two rounds of the toy cipher.

of a differential is the ratio of text pairs m_1, m_2 with input difference α and output difference β to all text pairs with input difference α. For the small S-boxes applied in many block ciphers one-round differentials are found by establishing a *difference distribution table* as the one shown in Table 3.5. In this the number of text pairs satisfying each possible differential over one S-box are counted.

Example 3.1. *Consider the non-linear S-box given in Table 3.4. We compute for all* $m_i \in [0, \ldots, 2^4 - 1]$ *the value* $S(m_i)$*. For each of the sixteen input differences* α *we compute for all pairs* (m_i, m_j) *where*

$$\Delta(m_i, m_j) = \alpha$$

the value

$$\beta = \Delta(S(m_i), S(m_j)).$$

Table 3.4: The S-box $S(\cdot)$ (in hexadecimal notation) of the toy cipher used in Example 3.1, 3.2 and 3.4.

Input	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Output	b	7	5	4	2	e	9	a	6	f	d	c	1	3	0	8

Table 3.5: Difference distribution for $S(\cdot)$ given in Table 3.4.

$\alpha \to \beta$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	4	2	2	0	0	0	0	2	2	0	0	4	0	0	0
2	0	2	0	4	2	0	0	0	0	0	0	6	0	0	2	0
3	0	0	4	2	0	0	0	2	2	2	2	0	0	0	0	2
4	0	0	0	0	2	0	0	2	0	4	0	0	4	2	2	0
5	0	0	0	0	0	8	0	0	0	0	0	0	2	2	2	2
6	0	0	2	0	0	0	2	4	0	0	2	0	2	2	0	2
7	0	2	0	0	0	0	2	0	0	0	0	2	0	2	6	2
8	0	0	2	2	0	0	0	0	6	2	0	0	0	4	0	0
9	0	6	0	0	2	0	0	0	0	4	2	0	0	0	0	2
a	0	0	2	2	0	0	4	0	2	2	0	4	0	0	0	0
b	0	0	2	0	0	0	0	2	0	0	8	2	0	0	2	0
c	0	2	0	0	6	2	2	0	0	0	2	0	2	0	0	0
d	0	0	0	0	2	2	2	2	4	0	0	0	0	4	0	0
e	0	0	2	0	2	2	0	2	0	0	0	2	0	0	0	6
f	0	0	0	4	0	2	4	2	0	0	0	0	2	0	2	0

We count the number of pairs that follow each of the 2^8 differentials $\alpha \to \beta$. From this we obtain the difference distribution Table 3.5. The rows and columns correspond to differences respectively at the input and output of the S-box.

Example 3.2. *We observe in Table 3.5, that the differential $0 \to 0$ has the highest probability namely $Pr(0 \to 0) = 16/16 = 1$. However, this is trivial since it is obvious that mapping the same text twice results in the same text. We also observe that both the differential*

$$\alpha = 5_x \to \beta = 5_x$$

and the differential

$$\alpha = b_x \to \beta = a_x$$

are satisfied with probability $1/2$.

Once the difference distribution table is established we try to combine the one-round differentials to a characteristic over multiple rounds.

Definition 3.2. *A differential characteristic* $(\alpha_0, \alpha_1, \ldots, \alpha_t)$ *is a chain of differences through* t *rounds of a cipher.*

The probability of a characteristic is the probability that a pair with difference $\Delta C^{(0)} = \Delta P = \alpha_0$ follows the characteristic computed as the average over all keys and all texts.

$$Pr_{P,K}(\alpha_0, \ldots, \alpha_t).$$

While attacking the cipher, the key is fixed and the plaintext is variable, thus the probability is an average over all texts and a fixed key

$$Pr_P(\alpha_0, \ldots, \alpha_t).$$

However, since we don't know the value of the key we cannot compute this probability. We assume that *for virtually all keys*

$$Pr_P(\alpha_0, \ldots, \alpha_t) = Pr_{P,K}(\alpha_0, \ldots, \alpha_t).$$

For certain ciphers called Markov cipher, introduced in [44], it is fairly simple to compute $Pr_{P,K}(\alpha_0, \ldots, \alpha_t)$.

Definition 3.3. *An iterated cipher with round function*

$$C^{(i+1)} = F(C^{(i)}, K^{(i+1)})$$

is a Markov cipher with respect to differential cryptanalysis, if there is a group operation $\oplus$ *defining differences such that*

$$Pr(\Delta C^{(i)} = \beta | \Delta C^{(i-1)} = \alpha)$$

is independent of the text $C^{(i-1)}$ *for all* $\alpha \neq e$ *and* $\beta \neq e$ *(where* e *is the neutral element of the group), when the round keys are chosen uniformly at random.*

For a Markov cipher with independent round keys, the probability of a characteristic is given by

$$Pr_{K,P}(\alpha_0, \ldots, \alpha_t) = Pr_K(\alpha_0, \ldots, \alpha_t) = \prod_{i=1}^{t} (\Delta C^{(i)} = \alpha_i | \Delta C^{(i-1)} = \alpha_{i-1}) \qquad (3.1)$$

AES and DES are examples of Markov ciphers [41]. The toy cipher studied in the examples of this chapter is also a Markov cipher. This is because equal sized round

key and text are exclusive-ored in the round function. Thus for a fixed text pair the probability of a one-round differential over all keys equals the probability of the differential over all text pairs. Therefore when the round keys are independent and chosen uniformly at random, the probability of the differential is independent of the text pair.
The probability of the characteristic p_x is important to the complexity because the number of pairs required for the attack to succeed is proportional to p_x^{-1} [41].
It is a fact that for practical block ciphers the round keys are derived from a key schedule implying that the keys are dependent. However, for a number of ciphers, it has been experimentally verified [41], that the probability given by the formula in Equation 3.1 corresponds nicely with the actual probability.
For realistic block ciphers it is usually very time consuming to find a good characteristic over a sufficient number of rounds. It is therefore attractive to find an iterative characteristic over a few rounds, with a high probability. The iterative characteristic is repeated a number of times to obtain a characteristic over the desired number of rounds.

Definition 3.4. *An s-round iterative characteristic is a differential chain of* $s+1$ *elements*

$$(\alpha_i, \ldots, \alpha_{i+s})$$

where $\alpha_i = \alpha_{i+s}$.

Example 3.3. *Since* $5 \rightarrow 5$ *has the highest probability among the one-round differentials for* $S(\cdot)$ *we iterate the differential* $n_r - 1$ *times and obtain the characteristic*

$$(\alpha_0, \alpha_1, \ldots, \alpha_{n_r-1}) = (5, 5, \ldots, 5).$$

Since the toy cipher is a Markov cipher the probability of this characteristic is

$$Pr(5, 5, \ldots, 5) = (\frac{1}{2})^{n_r-1}.$$

In Chapter 9 we study a practical example of iterative characteristics for the cipher Present.
We now sketch the procedure of a differential attack. Given an n_r round cipher with round function

$$C^{(i+1)} = F(C^{(i)}, K^{(i+1)}),$$

the differential attack is as follows:

Step 1 Establish the difference distributions tables for the non-linear functions of the cipher.

Step 2 Apply the tables of Step 1 to find a characteristic $(\alpha_0, \alpha_1, \ldots, \alpha_{n_r-1})$ with high probability p_x, relating the input difference $\Delta C^{(0)} = \alpha_0$ to the difference at the output of the after $n_r - 1$ rounds of encryption.

Step 3 Choose a text P_1 at random and compute $P_2 = P_1^{-1} \oplus \alpha_0$ such that the pair (P_1, P_2) has difference α_0. Obtain the corresponding ciphertexts $(C_1, C_2) = (e_K(P_1), e_K(P_2))$.

Step 4 Guess values of the last round key $K^{(n_r)}$ and decrypt for each key guess the last round $(F^{-1}(C_1, K^{(n_r)}), F^{-1}(C_2, K^{(n_r)}))$.

Step 5 Compute, for each key guess, the difference $\Delta C^{(n_r-1)} = \Delta(F^{-1}(C_1, K^{(n_r)},), F^{-1}(C_2, K^{(n_r)}))$. If the difference is $\Delta C^{(n_r-1)} = \alpha_{n_r-1}$ we say that the pair suggests the key. In Table 3.6 we increment the counter for each keys suggested by the pair.

Step 6 Repeat Step 3 to 5, N times, each time with a new random text P_1.

Table 3.6: Counting the of number pairs with difference $\Delta P = \Delta C^{(0)} = \alpha_0$, suggesting each of the key guesses $K_0^{(n_r)}, \ldots, K_s^{(n_r)}$.

$K_0^{(n_r)}$	$K_1^{(n_r)}$	$K_2^{(n_r)}$	$\ldots$	$K_s^{(n_r)}$
2	0	0	$\ldots$	1

Example 3.4. *The attacker chooses a plaintexts P_1 at random and computes the other half of the pair $P_2 = P_1 \oplus \alpha_0$, where $\alpha_0 = 5$. The cipher is a Markov cipher so he knows that for this pair one gets the difference $\Delta C^{(n_r-1)} = \alpha_{n_r-1} = 5$ when encrypted, with probability $p_x = (\frac{1}{2})^{n_r-1}$. The attacker obtains the ciphertexts (C_1, C_2) corresponding to the plaintext pair (P_1, P_2). He makes a list of guesses for the last round key $K^{(n_r)}$ (see Table 3.6). For each key guess he inverts the last round function*

$$C_1^{(n_r-1)} = F^{-1}(C_1^{(n_r)} \oplus K^{(n_r)}),$$
$$C_2^{(n_r-1)} = F^{-1}(C_2^{(n_r)} \oplus K^{(n_r)}).$$

He computes the difference $\Delta C^{(n_r-1)}$ for each key guess. If $\Delta C^{(n_r-1)} = \alpha_{n_r-1} = 5$ we say that the pair suggests the key, and the attacker increments the counter for this key. This process is repeated for different plaintexts chosen at random, until a key, namely the right key, is suggested enough times to distinguish it from other keys. If the probability of the differential is high enough the attack succeeds.

There is a number of other important issues in differential cryptanalysis such as differentials over multiple rounds and in relation to this provable security against differential cryptanalysis, impossible differentials, and truncated differentials. However, since this is already well-described [44, 42, 41] and differential cryptanalysis is not the main objective of this thesis we shall not get further into that. In [8] differential cryptanalysis is applied to break DES using 2^{47} chosen plaintexts.

3.5 Linear cryptanalysis

Linear cryptanalysis is a known plaintext attack. As in the differential attack we search for a probabilistic description of the cipher. More specifically we search for affine approximations of the non-linear functions of the cipher. In the basic attack the idea is to combine linear approximations over all rounds to obtain a linear expression in a number of key and text bits. From the linear approximation and a certain amount of known texts we can deduce one bit of key information. In an extended attack we search for a linear characteristic over $n_r - 1$ rounds when analyzing an n_r-round SP-network. We apply the approximation to distinguish the right from wrong guesses of the last round key, similar to what we did in the differential attack.
Given a vector of b binary variables $X = (x_{b-1}, \ldots, x_0)$ and a b-bit vector α, known as a *linear mask*, the inner product

$$\alpha \cdot X,$$

defines a linear expression in the variables of X.

Example 3.5. *The linear mask $\alpha = (0, 1, 1, 0)$ defines the linear expression*

$$0 \cdot x_3 \oplus 1 \cdot x_2 \oplus 1 \cdot x_1 \oplus 0 \cdot x_0 = x_2 \oplus x_1,$$

for the vector $X = (x_3, x_2, x_1, x_0)^T$.

Given a non-linear function $S(\cdot)$ with input vector X and output vector Y, a pair of linear masks (α, β) defines a linear approximation

$$\alpha \cdot X = \beta \cdot Y.$$

The probability of the approximation is

$$Pr(\alpha \cdot X = \beta \cdot Y) = \frac{1}{2} + \epsilon,$$

where ϵ is known as the bias. For the non-linear functions applied in a block cipher we search for an approximation with high probability. To find such approximations we establish linear approximation tables for the S-boxes of the cipher, as described in Example 3.6.

Example 3.6. *Table 3.7 is the linear approximation table of the 4-bit S-box $S(\cdot)$ given in Table 3.4. The rows contain the input masks and the columns contain the output masks. The occurrence of the approximations is measured by their deviation from the mean value. For instance the input mask $\alpha = (1, 1, 0, 1)$ is related to output mask $\beta = (1, 0, 1, 0)$ with probability $p_x = 1/2 + 6/16$ (See Table 3.7 $d \rightarrow a$). One can verify that the approximation is true for all inputs but two, namely $\{3, b\}$.*

Table 3.7: Linear approximations for the S-box $S(\cdot)$.

	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	–2	2	–4	2	0	0	2	2	4	0	–2	0	2	2	0
2	–2	–6	0	0	–2	2	0	2	0	0	–2	2	0	0	–2
3	4	0	0	2	–2	–2	–2	0	0	0	–4	–2	–2	2	–2
4	–2	0	2	–6	0	–2	0	0	2	0	–2	–2	0	2	0
5	0	–2	–2	0	0	–2	6	–2	–2	0	0	–2	–2	0	0
6	0	–2	2	2	2	4	0	–2	2	0	0	–4	0	2	2
7	2	0	2	0	–6	0	2	0	2	0	2	0	2	0	2
8	0	–2	–2	0	0	–2	–2	0	0	2	2	–4	4	–2	–2
9	–2	0	2	2	0	–2	0	–2	0	–2	–4	0	2	–4	2
a	2	0	2	0	2	0	2	–2	0	–2	0	2	4	2	–4
b	0	–2	2	2	2	–4	0	0	4	2	2	2	–2	0	0
c	–2	2	4	2	0	0	2	4	–2	2	0	–2	0	0	–2
d	0	0	0	0	0	0	0	–2	–2	6	–2	2	2	2	2
e	–4	0	0	2	–2	–2	–2	–2	–2	–2	2	0	0	4	0
f	–2	2	0	0	–2	2	0	–4	2	2	0	0	–2	–2	–4

Definition 3.5. *An t-round linear characteristic is a chain of linear masks over t rounds of the cipher*

$$(\alpha_0, \alpha_1, \ldots, \alpha_t).$$

Analogue to differential cryptanalysis we define a Markov cipher with respect to linear cryptanalysis [41].

Definition 3.6. *An iterated cipher is a Markov cipher with respect to linear cryptanalysis, if the probability of approximation $\alpha \cdot X^{(i)} = \beta \cdot Y^{(i)}$ over one round*

$$Pr(\alpha \cdot X^{(i)} = \beta \cdot Y^{(i)})$$

is independent in the text $X^{(i)}$, for all α, β, when the round keys are chosen uniformly at random.

Given a Markov cipher with independent round keys chosen uniformly at random, we can apply Matsui's piling-up lemma to compute the probability of the characteristic $(\alpha_0, \alpha_1, \ldots, \alpha_t)$.

Lemma 3.1 (Piling-up Lemma)**.** *Given $t+1$ independent variables*

$$v_0, v_1 \ldots, v_t$$

that take on values from $GF(2)$, the probability of the event that

$$v_0 \oplus v_1 \oplus \ldots \oplus v_t = 0$$

is given by

$$p_x = \frac{1}{2} + 2^t \prod_{i=0}^{t} (p_i - \frac{1}{2})$$

where $p_0, p_1, \ldots, p_t$ are the probabilities that $v_0 = 0, v_1 = 0, \ldots, v_t = 0$.

For a given characteristic with probability p_x the expected number of texts required to discover a bias in the linear relation is about $(p_x - \frac{1}{2})^{-2} = \epsilon^{-2}$ [41].

Example 3.7. *For the toy cipher applied in our examples we define as follows:*

$$\begin{aligned} X^{(i)} &= (x_3^{(i)}, x_2^{(i)}, x_1^{(i)}, x_0^{(i)}) \\ Y^{(i)} &= (y_3^{(i)}, y_2^{(i)}, y_1^{(i)}, y_0^{(i)}) \\ K^{(i)} &= (k_3^{(i)}, k_2^{(i)}, k_1^{(i)}, k_0^{(i)}) \\ P &= (p_3, p_2, p_1, p_0) \\ C &= (c_3, c_2, c_1, c_0) \end{aligned}$$

as the bit vectors of respectively the input and output of the S-box and the key K of round i, the plaintext P, and the ciphertext C. Consider a two-round version of the toy cipher shown in Figure 3.5.
For this we have

$$X^{(1)} = P \oplus K^{(0)} \tag{3.2}$$
$$Y^{(1)} = S(X^{(1)}) \tag{3.3}$$
$$X^{(2)} = Y^{(1)} \oplus K^{(1)} \tag{3.4}$$
$$Y^{(2)} = S(X^{(2)}) \tag{3.5}$$
$$C = Y^{(2)} \oplus K^{(2)} \tag{3.6}$$

In Table 3.7 we find that the linear approximation defined by the linear masks $(5, 7)$ has bias $\frac{6}{16}$ i.e. it has probability $\frac{14}{16}$. Also, we find that the approximation defined by $(7, 5)$ has bias $\frac{-6}{16}$ i.e. true with probability $2/16$. Thus the approximation

$$7 \cdot X \oplus 1 = 5 \cdot S(X)$$

is true with probability $1 - \frac{2}{16} = \frac{14}{16}$. We apply the approximations

$$5 \cdot X^{(1)} = 7 \cdot Y^{(1)} \tag{3.7}$$
$$7 \cdot X^{(2)} \oplus 1 = 5 \cdot Y^{(2)} \tag{3.8}$$

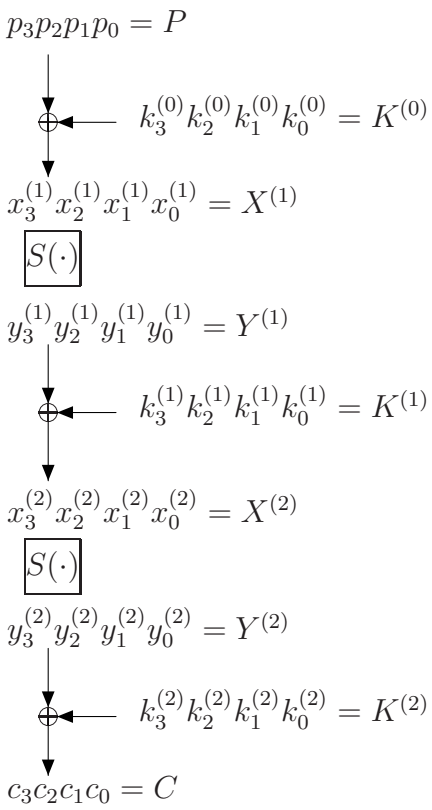


Figure 3.5: Two round toy cipher for the purpose of describing linear cryptanalysis.

and establish the linear relation between the text and the round keys

$$5 \cdot K^{(0)} \oplus 7 \cdot K^{(1)} \oplus 5 \cdot K^{(2)} = 5 \cdot P \oplus 5 \cdot C \oplus 1$$

corresponding to the linear approximation

$$k_2^{(0)} \oplus k_0^{(0)} \oplus k_2^{(1)} \oplus k_1^{(1)} \oplus k_0^{(1)} \oplus k_2^{(2)} \oplus k_0^{(2)} = p_2 \oplus p_0 \oplus c_2 \oplus c_1 \oplus 1.$$

Assuming that the approximations of the round functions are independent we can apply the piling-up lemma to compute the probability of the approximation

$$p_x = \frac{1}{2} + 2(\frac{6}{16} \cdot \frac{6}{16}) = \frac{1}{2} + \frac{9}{32}.$$

From this we see that we can obtain one bit of key information when applying approximately $(p_x - \frac{1}{2})^{-2} \approx 13$ *texts.*

Example 3.7 demonstrates the idea of the linear attack. However, as we would like to recover more than just one bit of key information the attack is extended as follows:

Step 1 Establish a linear characteristic over $n_r - 1$ round

$$(\alpha_0, \alpha_1, \ldots, \alpha_{n_r-1}),$$

with high probability p_x. The characteristic corresponds to a linear approximation of the form

$$\alpha_0 \cdot K^{(0)} \oplus \alpha_1 \cdot K^{(1)} \oplus \ldots \oplus \alpha_{n_r-1} \cdot K^{(n_r-1)} = \alpha_1 \cdot P \oplus \alpha_{n_r-1} \cdot X^{(n_r)} \quad (3.9)$$

where X^{n_r} is the input of the non-linear layer in the last round.

Step 2 Create a list of guesses for the last round key $K^{(n_r)}$, containing two counters T_0 and T_1 for each key guess.

Step 3 For each key guess invert the last round function for the ciphertext C corresponding to a known plaintext P, to obtain the value X^{n_r}. Compute the right hand side of Equation 3.9. For each key increment the counter T_0 if the sum is zero and T_1 if the sum is one (see Table 3.8).

Step 4 Repeat Step 3 for N known texts. For the right key $K^{(n_r)}$ one of the counters, T_0 or T_1, has expected value $p_x N$. For wrong keys the counters both have the expected value $\frac{N}{2}$.

Table 3.8: Counters T_0, T_1 for each guess of the last round key in a linear attack.

	$K_0^{(n_r)}$	$K_1^{(n_r)}$	$K_2^{(n_r)}$	...	$K_s^{(n_r)}$
T_0	14	2	$N-15$	...	$N-14$
T_1	$N-14$	$N-2$	15	...	14

Note that for a fixed key the left hand side of Equations 3.9 has a fixed value. We do not know this value but exploit that for the right key, the sum is either zero or one for $\frac{N}{2} + N \cdot p_x$ of the N texts while for other keys the sum is zero respectively one for $\frac{N}{2}$ of the texts.
In [48] Matsui presented linear cryptanalysis and applied it successfully on DES. The method breaks 8 round DES using 2^{21} known plaintext and the full 16-round DES using 2^{47} known plaintext. AES was designed to be strong towards linear and differential cryptanalysis and seems not to be threatened by these approaches.

3.6 Algebraic attacks

In later years algebraic attacks on symmetric-key ciphers have received much attention. The attacks have had notable impact in the area of stream ciphers due to the discovery of powerful attacks on certain ciphers [19]. The techniques apply in principle to (iterated) block ciphers although most results until now suggest that this does not lead to very effective attacks.
Most modern block ciphers have the structure of an SP-network. As described in Chapter 2, the round function is composed of a non-linear confusion layer, usually realized by a parallel application of small S-boxes, followed by a linear layer that provides diffusion across the block. The round is completed with linear application of a round key. For such a cipher an algebraic attack is established by deriving a set of algebraic equations that describes the linear layers (including the key application) and the non-linear layers of the cipher. Joining these together we obtain a complete description of the cipher. Algebraic attacks have the potential to recover the secret key using only a few known text pairs. By solving a large set of equations a few candidates for the right key are found. The right key is then easily obtained by eliminating other key candidates using a few other text pairs.
AES has from the beginning attracted much attention for its elegant and simple algebraic structure [33, 16]. In [20] an algebraic description of AES (and Serpent) over $GF(2)$ was presented along with some very controversial claims on the complexity of the so-called XSL algorithm for solving these equations. The problem of solving the AES equations is an instance of the well-known MQ-problem, that is the problem of solving multivariate quadratic equations over a finite field. The generic MQ-problem is known to be NP-complete (see Chapter 1). In fact there are cryptosystems that rely on the difficulty of this problem, for instance Hidden Field Equations which we describe in Chapter 5. By now not many people believe that the XSL algorithm works as claimed in [20], and so far no other efficient algorithm for solving the equations has been discovered. However, it is of course important to analyze and examine new ideas to get an impression of whether one could develop an efficient algorithm for solving the equations arising from block ciphers.
In the following sections we describe how to obtain algebraic equations for block ciphers over $GF(2)$. In the final section of this chapter we give a short description of the $GF(2^8)$-description of AES presented in [50].

3.7 Obtaining an algebraic description

As described in Chapter 2 a block cipher is a bijective function (for a fixed key) that maps an n_b-bit input to an n_b-bit output using a n_k-bit cipher key. The goal in an

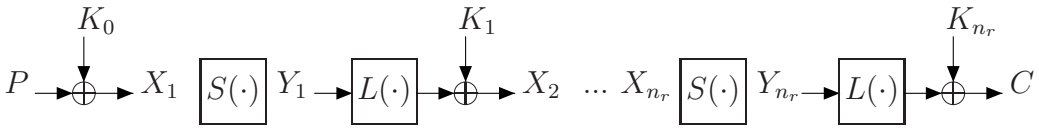


Figure 3.6: A simple block cipher structure.

algebraic attack is to find the cipher key.
Most block ciphers in use are fairly simple constructions. The round function consists a non-linear substitution layer $S(\cdot)$, a linear permutation layer $L(\cdot)$, and a linear key application. The round keys K_i are often mixed with the data via the exclusive-or operation while the non-linear layer typically is implemented by dividing the text into smaller blocks (e.g., 4 or 8 bits) which are then transformed by non-linear table lookups, the so-called S-boxes. By iterating the round function n_r times the ciphertext is obtained.
Our research mainly considers equations over $GF(2)$ and the description of this section concerns only this. Consider the following n_r-round block cipher where the encryption of a plaintext P into a ciphertext C is shown in Figure 3.6.
It follows that $X_1 = P \oplus K_0$, and that

$$X_{i+1} = K_i \oplus L(S(X_i)) = K_i \oplus L(Y_i),$$

for $i = 1, \ldots, n_r$, and $X_{n_r+1} = C$. Of particular interest are block ciphers for which one can obtain equations of low algebraic degree in the (bits of) the pairs (X_i, Y_i) for $i = 1, \ldots, n_r$. These equations are completely general for the cipher and independent of the key, the plaintext, and the ciphertext. From the linear layer one gets a set of linear equations connecting the round keys, the pairs (X_i, Y_i), the plaintext, and the ciphertext. Altogether the equations provide a complete algebraic description of the cipher.

3.7.1 Equations over one S-box

First we show how to find deterministic, multivariate equations over $GF(2)$ for an S-box. The "maximum set of equations of degree-d" is obtained as follows: Consider an S-box that maps b bits to c bits. To find all equations of degree less than or equal to d, construct a binary matrix $\underline{\underline{A}}$ with 2^b rows and $\sum_{i=0}^{d} \binom{b+c}{i}$ columns. The 2^b inputs of the S-box are the row entries and the $\sum_{i=0}^{d} \binom{b+c}{i}$ ordered (according to e.g. graded reverse lexicographic order) monomials of degree less than or equal to d are the

column entries. The bit of row i and column j is set if for the input i, the monomial of column j has the value one. All deterministic equations $f_1 = 0, f_2 = 0, \ldots, f_m = 0$, of degree less than or equal to d, over the S-box, are then obtained by computing the basis of the null space of $\underline{\underline{A}}$. From this we obtain at least

$$e = \sum_{i=0}^{d} \binom{b+c}{i} - 2^b$$

equations of degree less than or equal to d. This is due to the fact that for a matrix with p columns and q rows, where $(p \geq q)$, the null-space is spanned by at least $p-q$ vectors.

3.7.2 The linear layer

There are at least two ways to process the linear layers. For simplicity it is assumed that the round keys K_i are derived from the cipher key K by a simple bit permutation and it is assumed that there are no linear equations over the input and output of an S-box.

A first method is the one proposed in [20]. The input variables to the S-box layer (the bits of the X_is above) are eliminated by inserting a linear expression of the round key and the outputs (the bit of the Y_js above) from the previous round. Let n_{Sbox} denote the total number of S-boxes in the cipher and eq_{Sbox} denote the number of non-linear equations of some degree over $GF(2)$ for one S-box. Using this method a total of $n_{Sbox} \cdot eq_{Sbox} \cdot n_r$ non-linear equations in $n_b \cdot (n_r + 1)$ variables is obtained. The following describes another procedure which we later, in Chapter 6, apply to analyze algebraic attacks on block ciphers. The idea is to *eliminate the key variables*, and thereby keep the linear equations in the system (contrary to the first method above). This is achieved by combining the equations

$$L(Y_i) \oplus K_i = X_{i+1}$$

and

$$L(Y_j) \oplus K_j = X_{j+1}$$

for all $i < j$, by pair wise exclusive-oring equations that contain the same key bit variables. For each pair (i, j) this yields n_b linear equations over $GF(2)$ in bits from X_{i+1}, X_{j+1}, Y_i, and Y_j. The linear equations are separated from the equations of the S-box layer. In total one obtains a system of $n_b \cdot n_r$ linear equations in addition to the $n_{Sbox} \cdot eq_{Sbox} \cdot n_r$ non-linear equations (of a certain degree). The total number of variables is $2n_b \cdot n_r$. Note that although the equations do not contain any key

variables, it is straightforward to extract the value of the secret key from a solution to the other variables in the system.
The method is easily generalized to include ciphers with more complex key-schedules, e.g., those containing S-boxes, which will be demonstrated later.

3.8 Algebraic descriptions of AES

The main observation in [20] is that the AES S-box is described by a set of 39 quadratic, deterministic equations over $GF(2)$ and an additional probabilistic equation which is true with probability $\frac{255}{256}$. The 39 equations can be derived by the method in Section 3.7.1. However, the equations can also be derived by considering the algebraic structure of the AES S-box. Recall that the AES S-box is composed of an inversion over $GF(2^8)$, a $GF(2)$-linear map, and an addition of the constant 63_x. Consider the inversion map over $GF(2^8)$. We can write the 8-bit input X and the 8-bit output Y of the map as

$$X = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$$

and

$$Y = b_0 + b_1x^1 + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5 + b_6x^6 + b_7x^7$$

where

$$X, Y \in GF(2)[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle = GF(2)(\theta).$$

Assume that $X \neq 0, Y \neq 0$, then the following relations are satisfied

$$XY = 1, \tag{3.10}$$
$$X^2Y = X, \tag{3.11}$$
$$YX^4 = X^3, \tag{3.12}$$
$$Y^2X = Y^2, \tag{3.13}$$
$$Y^4X = Y^3. \tag{3.14}$$

These relations each provide 8 quadratic equations in the binary variables

$$a_0, \ldots, a_7, b_0, \ldots b_7$$

over $GF(2)$ because squaring is linear over $GF(2)$. Note that the expression $XY = 1$ is not satisfied for $X = 0$, therefore one of the equations derived from this expression is true only with $\frac{255}{256}$. For completeness the $GF(2)-$equations for the AES S-box are listed in Appendix A.

3.8.1 Eliminating the key variables

In this section we demonstrate the procedure of eliminating the key variables applied to AES-128. We first consider a variant of AES where all round keys are identical with the exception of adding a round-dependent constant. We define as variables in our system all the bits in the inputs and in the outputs of the S-boxes in all rounds. This yields 2560 variables. Each S-box is described by 39 quadratic equations over $GF(2)$, thus in total we obtain 6240 quadratic S-box equations. The output bits of the S-boxes in one round are then connected to the input bits of the S-boxes in the following round by the linear transformation and a 128-bit round key. One combines an expression involving the first round key with an expression involving the second round key and so on. Note that there are 11 round keys in the AES. Hence the round keys in these equations are eliminated and one obtains $10 \cdot 128 = 1280$ linearly independent, linear equations in 2560 (boolean) variables. In total we obtain 7520 equations of degree less than or equal to two in 2560 variables over $GF(2)$.
Let us next consider the real AES-128. Here the user-selected key is loaded directly into the first round key. Each of the following round keys are derived by passing one 32-bit word through a linear mix (RotWord), a layer of four S-boxes, and addition of a round constant (see Appendix A.2 for details). Thus, there is a total of 40 S-box applications in the AES key-schedule. To accommodate for this more complex key-schedule compared to the simple version above, we introduce new variables for all the input and output bits of the S-boxes in the key-schedule. Thus we introduce an additional 640 variables. This allows us to establish 1280 linear equations, as before, but now in 2560+640=3200 variables. In practice this is achieved by eliminating all the key bits in the linear equations which are not input bits to the key-schedule S-boxes. The round keys themselves are related by another 320 linear equations. In total we can establish a set of 1600 linear equations in 3200 variables. It has been checked that these 1600 equations for the (real) AES are linearly independent. The equations are given in detail in Appendix A.3. AES-128 applies in total 200 S-boxes from which we derive 7800 quadratic equations over $GF(2)$. All together the cipher is describes by 9400 equations of degree less than or equal to two in 3200 variables over $GF(2)$.

3.8.2 A description over $GF(2^8)$

The Bigger Encryption Standard (BES) was proposed by Murphy and Robshaw [50] as a larger cipher in which one can embed AES. BES is an iterated cipher that operates on 128-byte blocks with 128-byte keys. The round function is composed of inversion, matrix multiplication, and key addition. All operations are over $GF(2^8)$. When embedding AES in BES, the $GF(2)$-linear map of the substitution layer of

AES (see Section 2.3.3) is replaced by a mapping over $GF(2^8)$ in BES. Thus one can derive an algebraic description of BES hence of AES over $GF(2^8)$. The idea is that the algebraic description over $GF(2^8)$ is simpler and sparser than the description derived in the previous sections over $GF(2)$.

Embedding AES in BES

AES is embedded in BES by mapping each of the sixteen elements of the state from $GF(2^8)$ into for the purpose defined vector conjugates over $(GF(2^8))^8$.

Definition 3.7. *Let F be a finite field of order q and k be an extension field of F of degree d. The elements*

$$a, a^q, a^{q^2}, \ldots, a^{q^{d-1}}$$

are the conjugates of $a \in k$ with respect to F

The vector conjugate mapping $\phi : GF(2^8) \rightarrow (GF(2^8))^8$ is defined by

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

The mapping is closed with respect to addition and preserves inverses because squaring is a linear function over $GF(2)$

$$\phi(b + b') = \phi(b) + \phi(b')$$

$$\phi(b^{-1}) = \phi(b)^{-1}.$$

The conversion from BES to AES via ϕ^{-1} is possible when the BES vector forms an ordered set corresponding to an AES state. Each function of BES is defined such that the property of the vector conjugates is preserved. Most of the functions of BES are defined as a straightforward extension of the corresponding function of AES.

The Linear Layer

Both ShiftRows and MixColumns are defined over $GF(2^8)$ in AES. Recall that in AES MixColumns treats each column of the state (four bytes) as an element of $GF(2^4)$. This element is multiplied by

$$c(X) = (\theta + 1)X^3 + (1)X^2 + (1)X + \theta\,.$$

where θ is a root of the polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ that generates the field

$$GF(2)[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle.$$

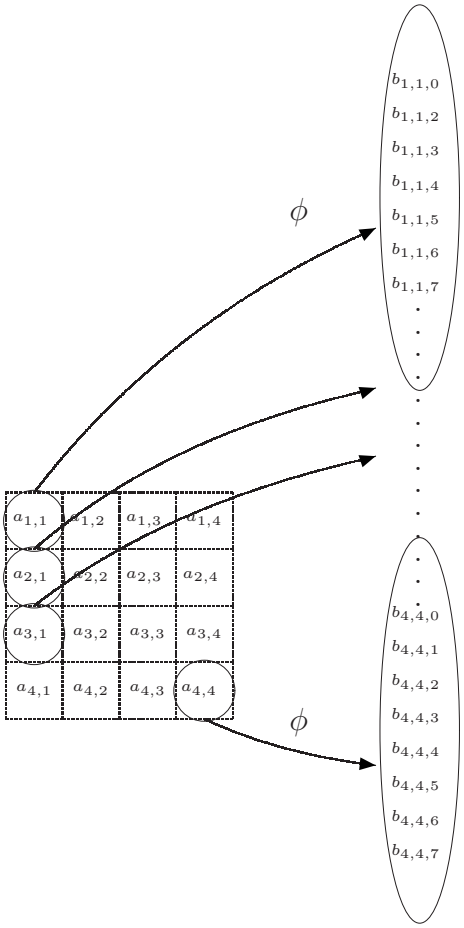


Figure 3.7: Converting the AES state to a BES vector. Each byte of the AES state is mapped into eight bytes of the BES vector.

In BES MixColumns multiplies the elements (for $m = 0, \ldots, 7$) of the four vector conjugates corresponding to a column of the AES state by the polynomial

$$c(X)^{(m)} = (\theta + 1)^{2^m} X^3 + (1)X^2 + (1)X + \theta^{2^m} .$$

The ShiftRows function of AES rotates cyclic each row of the state in different off-sets. In BES the rotation is repeated on all elements of the BES vector.

SubBytes

As described in Section 2.3.3 the SubBytes function is composed of inversion over $GF(2^8)$, a $GF(2)$-linear map, and addition with the constant 63_x. The $GF(2)$-linear

map $g : GF(2) \rightarrow GF(2)$ is actually the only part of AES that prevents an algebraic description of the cipher over $GF(2^8)$. In BES addition of the constant is moved into the key schedule and hence removed from the SubBytes function. Inversion of the BES vector in $(GF(2^8))^8$ is defined by element-wise inversion over $GF(2^8)$

$$b^{-1} = (b_{i,j,0}^{-1}, b_{i,j,1}^{-1}, b_{i,j,2}^{-1}, b_{i,j,3}^{-1}, b_{i,j,4}^{-1}, b_{i,j,5}^{-1}, b_{i,j,6}^{-1}, b_{i,j,7}^{-1}) .$$

Finally, the $GF(2)$-linear function is replaced by a so-called linearized polynomial.

Definition 3.8. *A linearized polynomial $f(X) \in k[X]$ is a polynomial given by*

$$f(X) = c_0 X + c_1 X^q + c_2 X^{q^2} + \ldots + c_{d-1} X^{q^{d-1}}$$

where $c_i \in k$. Thus a linearized polynomial $f(X)$ is a polynomial whose evaluation $f(a)$ for any $a \in k$ gives a linear combination of the d conjugates of a.

There exists a polynomial with coefficients in $GF(2^8)$ which interpolates the $GF(2)$-linear map of the AES S-box in its input-output points. The $GF(2)$-map of AES can be written as

$$f(X) = 05_x X^{2^0} + 09_x X^{2^1} + f9_x X^{2^2} + 25_x X^{2^3} + f4_x X^{2^4} + 01_x X^{2^5} + b5_x X^{2^6} + 8f_x X^{2^7}.$$

The operation is defined in BES by multiplication with an 8×8 matrix over $GF(2^8)$. The matrix replicates the action of the $GF(2)$-linear map of AES on the first byte of the vector conjugate and ensures that the vector conjugate property is preserved on the remaining 7 bytes.

The Round Keys

The AES key addition is defined by bitwise addition of a state and a round key. This operation is well-defined over $GF(2^8)$ hence also in BES and the only thing lacking is to extend the round key derivation to fit the BES vector. The key schedule of AES applies the same functions as the encryption function. As explained all of these are well defined over $GF(2^8)$ in BES, thus the key schedule is also well-defined. As mentioned, addition of the constant in the SubBytes function of AES is in BES transferred to the key schedule. This is done by embedding the image of the constant in BES and adding it to the round keys.

3.8.3 BES equations

The encryption algorithm of BES applies $8 \cdot 10 \cdot 16$ S-boxes. Each S-box gives rise to three quadratic equations over $GF(2^8)$. Let x be the input of an AES S-box,

and $(x_1, \ldots, x_8)$ be the vector conjugate input of eight S-boxes of BES. y is the corresponding S-box output and $(y_1 \ldots, y_8)$ the BES vector conjugate output. Then for $i = 0, \ldots, 7$ we get the equations

$$x_i y_i = 1,$$

$$x_i^2 = x_{i+1}$$

and

$$y_i^2 = y_{i+1},$$

where $i + 1$ is interpreted modulo 8, i.e. 3 quadratic equations per S-box.
In total the encryption algorithm is described by 3840 quadratic equations and 1408 linear equations in 2560 state variables and 1408 key variables. The key schedule applies 320 S-boxes and is therefore described by 960 quadratic equations and 1600 linear equations in the 1408 key variables and 640 auxiliary variables.
As mentioned the motivation of describing AES over $GF(2^8)$ instead of $GF(2)$ is that it is simpler and sparser. The authors of [50] noted that, given that the complexity estimates of the XSL algorithm [20] were correct, the attack would be more efficient for the $GF(2^8)$ description (time complexity 2^{100}). However, since XSL does not work as anticipated by its designers and the complexity estimates have been shown to be incorrect and too optimistic [14], this number means nothing at all. So far it remains unclear whether the $GF(2)$ or the $GF(2^8)$ description of AES is more favorable for mounting an algebraic attack. In Chapter 7 we outline a number of simulations over $GF(2)$ on the small scale variants of AES proposed in [15]. Compared to the results of [15] over $GF(2^8)$ our simulations suggests that after all the $GF(2)$-description seems favorable.

Chapter 4

Gröbner Bases Techniques

The theory of Gröbner bases was founded by Bruno Buchberger in his Ph.D thesis in 1965, and named after his supervisor Wolfgang Gröbner. Gröbner bases are applied for many purposes in for instance combinatorial optimization, coding theory, and robotics. Our interest in Gröbner bases is the application for solving algebraic equations describing the secret key of a cipher.
Given a set of polynomial equations $f_1 = 0, \ldots, f_m = 0$ we compute the reduced Gröbner basis $G = \{g_1, \ldots, g_l\}$ for the polynomial ideal I generated by

$$\langle f_1, \ldots, f_m \rangle.$$

Due to Proposition 4.1 (which we give later in this chapter) solving the polynomial equations $g_1 = 0, \ldots, g_l = 0$ is equivalent to solving $f_1 = 0, \ldots, f_m = 0$. The reason that we prefer to solve $g_1 = 0, \ldots, g_l = 0$ is that it is often easier than solving $f_1 = 0, \ldots, f_m = 0$.
Gröbner basis algorithms are perhaps the most promising methods regarding algebraic attacks on block ciphers. At least the best known algebraic results on block ciphers are accomplished using the Gröbner basis algorithm F4 [28].
In this chapter we describe the concept of Gröbner bases and how to compute them using respectively Buchberger's algorithm [12] and F4. For more details and the proofs, which we omit in this description, we refer the reader to [45, 28, 29].
Throughout this chapter we consider a polynomial ring $k[x_1, \ldots, x_n]$ over a finite field k, in order to simplify the definitions of a Gröbner basis.

4.1 Polynomial ideals

At first we give a few important definitions.

Definition 4.1. *Let k be a field, then $k[x_1, \ldots, x_n]$ is the polynomial ring over k in the variables $x_1, \ldots, x_n$. A polynomial $f \in k[x_1, \ldots, x_n]$ can be written*

$$f = a_1 m_1 + \ldots + a_l m_l,$$

where $a_i \in k$ are the coefficients of the monomials

$$m_i = x_1^{\alpha_{1,i}} \cdots x_n^{\alpha_{n,i}} \quad \alpha_{j,i} \in \mathbb{N}.$$

We denote $t_i = a_i m_i$, where $a_i \neq 0$, as the terms of f. The total degree of f, denoted $deg(f)$, is the maximum

$$|\alpha_i| = \sum_{j=1}^{n} \alpha_{j,i}$$

where $a_i \neq 0$.

Definition 4.2. *A non-empty set of polynomials $I \in k[x_1, \ldots, x_n]$ is said to be an ideal if*

- *$f_i + f_j \in I$ for any $f_i, f_j \in I$,*
- *$mf \in I$ for any $f \in I$ and any $m \in k[x_1, \ldots, x_n]$*

Definition 4.3. *Let $f_1, \ldots, f_m$ be a set of polynomials where $f_i \in k[x_1, \ldots, x_n]$, then $\langle f_1, \ldots, f_m \rangle$ is the ideal generated by $f_1, \ldots, f_m$, i.e.*

$$\langle f_1, \ldots, f_m \rangle = \{m_1 f_1 + \ldots + m_m f_m \quad | \quad m_1, \ldots, m_m \in k[x_1, \ldots, x_m]\}$$

Definition 4.4. *The affine variety $V(f_1, \ldots, f_m)$ of a set of polynomials $f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$ is the set of solutions $(a_1, \ldots, a_n) \in k^n$ satisfying all of the equations*

$$f_1(x_1, \ldots, x_n) = 0,$$

$$\vdots$$

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

simultaneously.

Moreover the variety $V(I)$ defined by an ideal $I \subseteq k[x_1, \ldots, x_n]$ in the affine space k^n is

$$V(I) = \{(a_1, \ldots, a_n) \in k^n | f(a_1, \ldots, a_n) = 0 \text{ for all } f \in I\}.$$

Proposition 4.1. *Let $V(I)$ be the affine variety of the ideal I.*
If

$$I = \langle f_1, \ldots, f_m \rangle,$$

then

$$V(I) = V(f_1, \ldots, f_m).$$

From Proposition 4.1 we have that solving the polynomial equations

$$f_1 = 0, \ldots, f_m = 0$$

is equivalent to determining the variety of any set of polynomials that generates the ideal

$$I = \langle f_1, \ldots, f_m \rangle.$$

A Gröbner basis $G = \{g_1, \ldots, g_k\}$ for the ideal $I = \langle f_1, \ldots, f_m \rangle$, is a generating set for I with some special properties.
For the purpose of Gröbner basis computation we need an ordering on the elements of $k[x_1 \ldots, x_n]$. In the univariate ring $k[x]$ there is a natural way of ordering the monomials $m_i \in k[x]$, namely in ascending order with respect to the degree. As opposed to this there are infinitely many ways to order the elements in the multivariate ring $k[x_1, \ldots, x_n]$. We therefore need to specify a *term order* on the elements of $k[x_1, \ldots, x_n]$. A term order is a total order with extra conditions.

Definition 4.5. *A total order on the monomials $m \in k[x_1, \ldots, x_n]$ is a relation $\leq$ that for all $m_i, m_j, m_l \in k[x_1, \ldots, x_n]$ satisfies*

- *$m_i \leq m_i$ (reflexivity),*
- *$m_i \leq m_j$ and $m_j \leq m_i \Rightarrow m_i = m_j$ (antisymmetry),*
- *$m_i \leq m_j$ and $m_j \leq m_l \Rightarrow m_i \leq m_l$ (transitivity),*
- *$m_i \leq m_j$ or $m_j \leq m_i$ (totality).*

A term order is defined as follows.

Definition 4.6. *A term order $\leq$ on $k[x_1, \ldots, x_n]$ is a total order that for all m_i, m_j, m_l satisfies*

- $0 \leq m_i$
- $m_i \leq m_j \Rightarrow m_i m_l \leq m_j m_l$

We give as an example the graded reverse lexicographic (grevlex) order which is the preferred choice for the F4 algorithm (described in Section 4.3).

Definition 4.7 (Graded Reverse Lexicographic Order)**.** *According to graded reverse lexicographic order*

$$x_1^{\alpha_1} \cdots x_n^{\alpha_n} = x_1^{\beta_1} \cdots x_n^{\beta_n}$$

if $\alpha_i = \beta_i$ *for all* i*, and*

$$x_1^{\beta_1} \cdots x_n^{\beta_n} < x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

if

$$|\beta| = \sum_{i=1}^{k} \beta_i < |\alpha| = \sum_{i=1}^{k} \alpha_i \quad \text{or} \quad |\alpha| = |\beta|$$

and the rightmost non-zero entry of $(\alpha_1 - \beta_1, \alpha_2 - \beta_2, \ldots, \alpha_n - \beta_n)$ *is negative.*

Example 4.1. *Consider the ring* $GF(2)[x_1, x_2, x_3]$*. The elements*

$$\{x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3\}$$

in grevlex order are then

$$x_3 < x_2 < x_1 < x_2x_3 < x_1x_3 < x_1x_2 < x_1x_2x_3.$$

We define the head term of a polynomial f as follows.

Definition 4.8. *The head term of a polynomial* f *with respect to a term order* $\leq$ *is defined as*

$$\mathrm{HT}(f) = a_i m_i$$

where m_i *is the maximum monomial of* f *with respect to* $\leq$ *and where* $a_i \neq 0$*.*

Example 4.2. *The polynomial*

$$f = x_1x_2 + x_1x_3 + x_2x_3 + x_3$$

has head term $\mathrm{HT}(f) = x_1x_2$ *according to grevlex order.*

Buchberger's algorithm for Gröbner bases computation basically consists of two procedures which are repeated until a Gröbner basis is obtained. In the first we generate polynomials, and in the second we reduce the polynomials. In the following we describe the multivariate polynomial division algorithm which is a basic part of Gröbner bases computation.

Given a tuple of non-zero polynomials $F = (f_1, \ldots, f_m)$, a polynomial f, and a fixed term order $\leq$, we can compute a remainder r such that

$$f = a_1 f_1 + \ldots a_m f_m + r,$$

where either none of the terms in r are divisible by $\mathrm{HT}(f_1), \ldots, \mathrm{HT}(f_m)$ or $r = 0$. The *multivariate division algorithm* is described as follows. Given a sequence of polynomials $F = f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$, and a polynomial $f \in k[x_1, \ldots, x_n]$ we fix a term order $\leq$. f is reduced modulo the polynomials in F by repeating the following steps until f is invariant to the procedure:

Step 1 Find smallest i such that $\mathrm{HT}(f_i)$ divides some term t of f.

Step 2 Replace f by $f - \frac{t}{\mathrm{HT}(f_i)} f_i$.

Definition 4.9. *Suppose $f \in k[x_1, \ldots, x_n]$ and let $F = f_1, \ldots, f_m$ be a sequence of non-zero polynomials in $k[x_1, \ldots, x_n]$. Then*

$$f \mod F$$

denotes the remainder r from dividing f by F using the multivariate division algorithm.

In the univariate ring $k[x]$ there is only one order, namely the degree order, thus the remainder coming from the division algorithm is uniquely defined. As shown in the following examples this is not the case in general for multivariate rings $k[x_1, \ldots, x_n]$.

Example 4.3. *Fix the term order $\leq$ to be grevlex. Given a list of polynomials $F = (f_1, f_2) = \{x_1x_2 + x_2, x_2x_3 + x_3\}$ in $GF(2)[x_1, x_2, x_3]$ we reduce the polynomial $f = x_1x_2x_3 + x_1x_3 + x_2$ to*

$$r = f \mod F$$

as follows

$$r_1 = f \mod f_1 = x_1x_3 + x_2x_3 + x_2,$$

$$r = r_1 \mod f_2 = x_1x_3 + x_2 + x_3.$$

Example 4.4. *Consider the same equations as in Example 4.3 only this time we swap the order of the elements in F, thus $F = (f_1, f_2) = (x_2x_3 + x_3, x_1x_2 + x_2)$. The polynomial division algorithm reduces $f = x_1x_2x_3 + x_1x_3 + x_2 \mod F$ to*

$$r_1 = f \mod f_1 = x_1x_2 + x_1x_3 + x_2,$$

$$r = r_1 \mod f_2 = x_1x_3.$$

A Gröbner basis is a generating set for an ideal I with the property that the remainder coming from the division algorithm is independent in the order of its element.

4.2 Buchberger's algorithm

We first define the main object of Buchberger's algorithm, namely the Gröbner basis.

Definition 4.10. *A set of non-zero polynomials is a Gröbner basis*

$$G = \{f_1, \ldots, f_l\}$$

for an ideal

$$I \subseteq k[x_1, \ldots, x_n]$$

with respect to a term order $\leq$ *if* $G \subseteq I$ *and for every* $f \in I \setminus \{0\}$ *there is a* $f_i \in G$ *for which* $\mathrm{HT}(f_i)|\mathrm{HT}(f)$.

Theorem 4.1. *Let* k *be a field,* $\leq$ *a term order, and* $I \subseteq k[x_1, \ldots, x_n]$ *an ideal. Then* I *has a Gröbner basis with respect to* $\leq$.

Buchberger's algorithm computes a Gröbner basis for an ideal

$$I = \langle f_1, \ldots, f_m \rangle$$

in a polynomial ring $k[x_1, \ldots, x_n]$ with respect to a fixed term order $\leq$.
For any non-zero pair $f_1, f_2 \in k[x_1, \ldots, x_n]$ we can define the S-polynomial $S(f_1, f_2)$.

Definition 4.11. *The S-polynomial of a pair of non-zero polynomials* (f_1, f_2) *is defined as*

$$S(f_1, f_2) = \frac{\mathrm{lcm}(\mathrm{HT}(f_1), \mathrm{HT}(f_2))}{\mathrm{HT}(f_1)} f_1 - \frac{\mathrm{lcm}(\mathrm{HT}(f_1), \mathrm{HT}(f_2))}{\mathrm{HT}(f_2)} f_2,$$

with respect to a term ordering $\leq$.

The pair (f_1, f_2) corresponding to the S-polynomial $S(f_1, f_2)$ is commonly referred to as a critical pair.

Corollary 4.1 (Buchberger's S-criteria)**.** *A sequence* $G = \{f_1, \ldots, f_m\}$ *of polynomials is a Gröbner basis if and only if* $S(f_i, f_j) = 0 \mod G$, *for* $1 \leq i < j \leq m$.

Buchberger's Algorithm takes as input a set of polynomials $F = f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$ and outputs a Gröbner basis G for the polynomial ideal
$I = \langle f_1, \ldots, f_m \rangle$.

The algorithm is described as follows:

Step 1 Initialize $G = F$.

Step 2 For all pairs of polynomials $f_i, f_j \in G$, where $i < j$, compute

$$f = S(f_i, f_j) \mod G.$$

If $f \neq 0$ set $G' = G \cup f$.

Step 3 If $G = G'$ return and output G. Else set $G = G'$ and go to Step 2.

Theorem 4.2. *Buchberger's Algorithm terminates and returns a Gröbner basis.*

Theorem 4.2 can be proved by applying either Hilbert's basis theorem or Dickson's lemma [45].

For a number of problems, and in particular those we consider in this chapter, we know that the solutions of the polynomial equations are in k^n, where k is a finite field of $q = 2^m$ elements. We therefore add the field relations $x_i^q + x_i$ for $i = 1, \ldots, n$ to F, i.e.

$$F = f_1, \ldots, f_m, x_1^q + x_1, \ldots x_n^q + x_n.$$

This simplifies the computations because, in practice, x_i^q is reduced to x_i whenever it appears. In this and the following chapters this is denoted by

$$I \subseteq k[x_1, \ldots, x_n]/\langle x_1^q + x_1, \ldots, x_n^q + x_n \rangle.$$

Example 4.5. *This and the following examples shows how we apply Gröbner bases to find the solutions* $(a_1, a_2, a_3) \in GF(2)^3$ *of the polynomial equations*

$$\{f_1 = 0, f_2 = 0, f_3 = 0\} = \{x_2x_3 + x_3 = 0, x_1x_2 + x_2 = 0, x_1x_2x_3 + x_1x_3 + x_2 = 0\}$$

where $f_1, f_2, f_3 \in GF(2)[x_1, x_2, x_3]$.
We fix the term order $\leq$ *to grevlex order and compute a Gröbner basis for the polynomial ideal*

$$I = \langle f_1, f_2, f_3 \rangle$$

where $I \subseteq GF(2)[x_1, x_2, x_3]/\langle x_1^2 + x_1, x_2^2 + x_2, x_3^2 + x_3 \rangle$.

$$\begin{aligned} f_4 &= S(f_1, f_2) = x_1x_3 + x_3, \\ f_5 &= S(f_1, f_3) = x_2, \\ f_6 &= S(f_2, f_3) = x_3, \end{aligned}$$

$$S(f_1, f_4) = S(f_1, f_5) = S(f_1, f_6) = S(f_2, f_4) = S(f_2, f_5) = S(f_2, f_6) = S(f_3, f_4) = S(f_3, f_5) = S(f_3, f_6) = S(f_4, f_5) = S(f_4, f_6) = S(f_5, f_6) = 0.$$

Buchberger's algorithm returns the Gröbner basis $G = \{f_1, f_2, f_3, f_4, f_5, f_6\}$.

We notice that many of the S-polynomials generated in Example 4.5 reduce to zero. This is in fact an important problem concerning the complexity of Buchberger's algorithm. The following lemma optimizes Buchberger's algorithm though it does not solve the problem.

Lemma 4.1. *Let* $\leq$ *be a term order on* $k[x_1 \ldots, x_n]$. *Let* $f_1, f_2 \in k[x_1 \ldots, x_n]$ *and suppose that* $\mathrm{HT}(f_1)$ *and* $\mathrm{HT}(f_2)$ *have no common divisors except constants, then*

$$S(f_1, f_2) \mod (f_1, f_2) = 0.$$

Lemma 4.1, known as Buchberger's first criteria, allows us to sort out a number of critical pairs that reduce to zero and hereby save time in computing and reducing the S-polynomials.

Example 4.6. *According to Lemma 4.1 we can omit the critical pairs*

$$(f_2, f_6), (f_4, f_5), (f_5, f_6)$$

of Example 4.5 since the head terms pair wise have no common divisors.

There exists a Buchberger's second criteria [5] which allows us to sort out more critical pairs but in order to limit our description, we do not discuss this here. The Gröbner basis computed by Buchberger's algorithm is not unique but we can convert it into a Gröbner basis of a special form which is unique. At first we define a minimal Gröbner basis.

Definition 4.12. *A minimal Gröbner basis* $G = \{f_1, \ldots, f_l\}$ *is a Gröbner basis where*

- $\mathrm{HT}(f_i)$ *is not divisible by* $\mathrm{HT}(f_j)$ *for* $i \neq j$
- *The coefficient of* $\mathrm{HT}(f_i)$ *is 1.*

Definition 4.13. *A reduced Gröbner basis,* $G_{red} = \{f_1, \ldots, f_l\}$ *is a minimal Gröbner basis where no term in* f_i *is divisible by* $\mathrm{HT}(f_j)$ *for* $i \neq j$.

Theorem 4.3. *Every ideal* I *has a unique reduced Gröbner basis.*

Once a Gröbner basis is obtained we can apply the polynomial division algorithm to compute the reduced Gröbner basis.

Example 4.7. *For the Gröbner basis computed in Example 4.5*

$$\begin{aligned} f_1 &= x_2x_3 + x_3, \\ f_2 &= x_1x_2 + x_2, \\ f_3 &= x_1x_2x_3 + x_1x_3 + x_2, \\ f_4 &= x_1x_3 + x_3, \\ f_5 &= x_2, \\ f_6 &= x_3. \end{aligned}$$

We compute, by polynomial division with respect to grevlex order, the reduced Gröbner basis

$$G_{red} = \{x_2, x_3\}$$

everything else reduce to zero. Applying Proposition 4.1 we have that the solutions to the polynomial equations $\{f_1 = 0, f_2 = 0, f_3 = 0\}$ *is given by* $(x_1, x_2, x_3) = (t, 0, 0)$ *where* $t \in GF(2)$.

The Gröbner basis reduction can be incorporated into Buchberger's algorithm such that the reduced Gröbner basis is computed directly.
There are several problems regarding Buchberger's algorithm such as the huge number of S-polynomials reducing to zero, which term order is more appropriate for the specific purpose, and in which order we treat the critical pairs.
The complexity of Buchberger's algorithm is closely related to the total degree d of the intermediate S-polynomials. Unfortunately this degree can be very big (in some cases double exponential [16]) and the running time and memory consumption can make the algorithm fail for a number of purposes.

4.3 Faugère's improvements

This section primarily describes the F4 algorithm, which along with F5 is probably the most powerful algorithm for Gröbner basis computations. The algorithms F4 and F5 were proposed by Faugère [28, 29] in respectively 1999 and 2002. Both algorithms optimize Buchberger's algorithm (in different ways). The idea of F4 is to re-use *reductors*, i.e. the polynomials generated for the purpose of reducing other polynomials. F5 works similar to F4 while its main target is to avoid generating polynomials that reduce to zero. In F4 the improvement is carried out by simultaneous reduction of several S-polynomials. This is realized by methods from linear

algebra, where Gaussian elimination is generalized to non-linear polynomial equations. The algorithm does not improve the worst case complexity for Gröbner basis computation but for a number of purposes it is much faster than Buchberger's algorithm. The F4 algorithm works on basis of any admissible order but it was designed to be efficient for grevlex order.
F4 is implemented as the standard algorithm in Magma [59] for computing Gröbner bases and has been applied to solve the first Hidden Field Equations (HFE) challenge of 80 quadratic equations in 80 variables over $GF(2)$ [59]. In Chapter 7 we give a number of timing results of algebraic attacks on small block cipher applying Magma's implementation of F4.
The following example is meant to motivate the idea of F4.

Example 4.8. *Suppose we want to reduce the three polynomials f_1, f_2, f_3 modulo f_4 where*

$$f_1, f_2, f_3, f_4 \in GF(2)[x_1, x_2, x_3, x_4]/\langle x_1^2 + x_1, x_2^2 + x_2, x_3^2 + x_3, x_4^2 + x_4\rangle\,,$$

and

$$\begin{aligned} f_1 &= x_1x_2x_3 + x_1x_3 + x_1 + x_3 + 1,\\ f_2 &= x_1x_2x_4 + x_1x_3 + x_1,\\ f_3 &= x_1x_3x_4 + x_1x_3 + x_1 + x_2,\\ f_4 &= x_1 + x_4\,. \end{aligned}$$

We choose the grevlex term order and compute by polynomial division

$$\begin{aligned} f_1 \mod f_4 &= f_1 - (x_2x_3 + x_3 + 1)f_4 = x_2x_3x_4 + x_3x_4 + x_3 + x_4 + 1,\\ f_2 \mod f_4 &= f_2 - (x_2x_4 + x_3 + 1)f_4 = x_2x_4 + x_3x_4 + x_4,\\ f_3 \mod f_4 &= f_3 - (x_3x_4 + x_3 + 1)f_4 = x_2 + x_4\,. \end{aligned}$$

In Example 4.8 we note that the *reductor* x_3f_4 is generated several times. Such redundancy in the polynomial reduction occurs often in Gröbner basis computation. The next example shows how this problem is handled in the reduction step of F4.

Example 4.9. *Consider the same polynomials as in Example 4.8. We reduce*

$$\begin{aligned} f_1 &= x_1x_2x_3 + x_1x_3 + x_1 + x_3 + 1,\\ f_2 &= x_1x_2x_4 + x_1x_3 + x_1,\\ f_3 &= x_1x_3x_4 + x_1x_3 + x_1 + x_2 \end{aligned}$$

simultaneous modulo $f_4 = x_1 + x_4$ as follows.
Construct a matrix containing the coefficients of all of the polynomials which appear in the division algorithm that is the coefficients of:

$$f_1, f_2, f_3, f_4, x_2x_3f_4, x_2x_4f_4, x_3x_4f_4, x_3f_4\,.$$

The monomials are ordered according to grevlex with the greatest monomial in the left most column

$$x_1x_2x_3 > \ldots > x_2x_3x_4 > x_1x_2 > \ldots > x_3x_4 > x_1 > x_2 > x_3 > x_4 > 1$$

$$\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \\ x_2x_3f_4 \\ x_2x_4f_4 \\ x_3x_4f_4 \\ x_3f_4 \end{array} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Note that as opposed to Example 4.8, x_3f_4 is generated only once. By Gaussian elimination and back substitution we obtain

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

corresponding to the polynomials

$$F = \begin{array}{l} \{x_1x_2x_3 + x_3x_4 + x_3 + x_4 + 1, \\ x_1x_2x_4 + x_3x_4 + x_4, \\ x_1x_3x_4 + x_3x_4 + x_4, \\ x_2x_3x_4 + x_3x_4 + x_3 + x_4 + 1, \\ x_2x_4 + x_3x_4 + x_4, \\ x_1x_3 + x_3x_4, \\ x_1 + x_4, \\ x_2 + x_4\}\,. \end{array}$$

The procedure given in Example 4.9 computes a set of polynomials F where for all $f_i \in F$, no terms of f_i is divisible by $\mathrm{HT}(f_j)$ for $i \neq j$. I.e. if the reduction algorithm is applied to a Gröbner basis it returns the reduced Gröbner basis.

F4 computes a Gröbner basis G for a set of polynomials

$$F = \{f_1, \ldots, f_m\} \in k[x_1, \ldots, x_n] .$$

The basic steps of the algorithm are described below. In Step 4 a list of polynomials L is reduced modulo a temporary polynomial basis G. This is done by constructing a matrix containing both L and G as sketched in Example 4.9 and performing Gaussian elimination and back substitution on this.

Step 1 Initialize $G = F$ and create a list P of all critical pairs (f_i, f_j) where $i < j$ and $f_i, f_j \in F$.

Step 2 Select a subset of the critical pairs $P' \subseteq P$. Update $P = P \setminus P'$.

Step 3 Generate a list L of polynomials which for each critical pair (f_i, f_j) in P' contains two polynomials $f_l = \frac{\mathrm{lcm}(\mathrm{HT}(f_i),\mathrm{HT}(f_j))}{\mathrm{HT}(f_i)} f_i$ and $f_r = \frac{\mathrm{lcm}(\mathrm{HT}(f_i),\mathrm{HT}(f_j))}{\mathrm{HT}(f_j)} f_j$.

Step 4 Compute a list of polynomials L' by reducing the polynomials of L modulo G as described above.

Step 5 Update P by adding the pairs (f_i, f_j) where $f_i \in L' \setminus G$ and $f_j \in G$ to the list. Add the polynomials of $L' \setminus G$ to G.

Step 6 Go to Step 2.

The algorithm terminates and outputs G when the list of pairs P is empty.

Theorem 4.4. *F4 computes a Gröbner basis G for the ideal generated by $F = \{f_1, \ldots, f_m\}$.*

This description does not cover optimizations (e.g. how Buchberger's criteria are incorporated, and how the reduction bases are re-used) or the so-called symbolic pre-processing which is part of the reduction function in Step 4.

The complexity of F4 is hard to estimate. Like for Buchberger's algorithm it depends on the degree of the intermediate polynomials, and the selection strategy. Though F4 is faster than Buchberger's algorithm for some purposes it often requires more memory. The memory consumption is a problem of both the Buchberger's algorithm and F4 regarding the application to algebraic attacks on block ciphers. In Chapter 7 we outline a number of tests on down scaled versions of AES applying both Buchberger's algorithm and F4.

Chapter 5

The Linearization Techniques

The method of Relinearization, XL (Extended Linearization) and XSL (Extended Sparse Linearization) all belong to the category of linearization techniques. The methods have each, at their time of publication, been claimed by their inventors to run in polynomial time. This has been the subject of much discussion and the approximations on their complexities derived in respectively [40, 18, 20] have been widely criticized for being inaccurate and too optimistic (seen from the attackers point of view). In a note [49] Moh describes how numbers, that are crucial to the complexity of the Relinearization technique, are miscounted. Unfortunately the note can be hard to follow and does not seem to cover all problems regarding the complexity of the method.
The Relinearization attack was designed to cryptanalyze Hidden Field Equations (HFE), and its description is somewhat hidden in the analysis of HFE [40]. However, as stated by the authors of [40] its application extends to a large range of problems. The XL attack is an extension of the Relinearization technique with the purpose of being simpler and more adequate for overdefined sets of polynomial equations, as those we encounter in algebraic descriptions of various block ciphers.
To this day, the critical question regarding the linearization techniques is determining the complexity, and in particular of our interest, discovering whether they apply to block ciphers. This chapter aims to give a clear description of the method of Relinearization and XL as well as of the problems regarding their complexities.

5.1 Hidden field equations (HFE)

In this section we give a brief description of HFE to explain the motivation of the Relinearization attack. The HFE is a public key cryptosystem. The cipher maps an n-bit plaintext to an n-bit ciphertext by evaluating n polynomials over $GF(q)$. The

recommended values for n and q are $n = 128$ and $q = 2$.
The private key

$$K_s = \{P(X), S, T\}$$

consists of two linear maps T, S over $GF(q)^n$, and a univariate polynomial $P(X)$, chosen at random, over the extension field $GF(q^n)$:

$$P(X) = \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} p_{ij} X^{q^i+q^j} \quad \in GF(q^n)[X]$$

with $r \leq 13$ in order to bound the degree of $P(X)$.
The extension field $GF(q^n)$ is constructed from $GF(q)[x]/m(x)$ where $m(x)$ is an irreducible polynomial of degree n. To generate the public key, we represent the polynomial $P(X)$ as vector of n multivariate quadratic polynomials in $GF(q)[x_0, \ldots, x_{n-1}]$. The input and the output of the polynomials are processed by respectively S and T to obtain the public key

$$K_p = \{P_0(x_0, \ldots, x_{n-1}), P_1(x_0, \ldots, x_{n-1}), \ldots, P_{n-1}(x_0, \ldots, x_{n-1})\}.$$

Noting that $f(X) = X^{q^i}$ is a linear function over $GF(q^n)$, and $a^q = a$ in $GF(q)$, one can verify that the polynomials of K_p are quadratic and homogeneous.

Example 5.1 (Toy example of HFE)**.** *Consider HFE with $r = 2$, $n = 4$ and $q = 2$. We construct the extension field $GF(2^4)$ over the polynomial ring modulo the irreducible polynomial $m(x) = x^4+x+1$. For the secret key we choose the polynomial*

$$P(X) = xX^2 + x^2X^3 + (x^2 + 1)X^3 + x^3X^4$$

where $X = a_3x^3 + a_2x^2 + a_1x + a_0$ is an element of the extension field $GF(2^4)$. We compute

$$\begin{aligned} P(X) \;=\; & x^3(a_0 + a_1) + x^2(a_0a_2 + a_1a_2 + a_2a_3 + a_0) + \\ & x(a_0a_1 + a_0a_2 + a_1a_3 + a_0 + a_1 + a_2) + \\ & (a_1a_3 + a_2a_3 + a_0 + a_1 + a_2)\,. \end{aligned}$$

Now we interpret the coefficients (a_0, a_1, a_2, a_3) of the polynomial representation of X as variables over $GF(2)$ and extract the following four equations over $GF(2)[x_0, x_1, x_2, x_3]$

$$\begin{aligned} y_3 &= x_0 + x_1, \\ y_2 &= x_0x_2 + x_1x_2 + x_2x_3 + x_0, \\ y_3 &= x_0x_1 + x_0x_2 + x_1x_3 + x_0 + x_1 + x_2, \\ y_0 &= x_1x_3 + x_2x_3 + x_0 + x_1 + x_2\,. \end{aligned}$$

y_0, y_1, y_2, y_3 is transformed by T and x_0, x_1, x_2, x_3 by S to obtain the public key

$$K_p = \{P_0(x_0, \ldots, x_3), P_1(x_0, \ldots, x_3), P_2(x_0, \ldots, x_3), P_3(x_0, \ldots, x_3)\}.$$

The ciphertext is computed by evaluating the n-bit plaintext in the n (public) polynomials

$$P_0(x_0, \ldots, x_{n-1}), \ldots, P_{n-1}(x_0, \ldots, x_{n-1})\,.$$

To decrypt the holder of the secret key applies the inverse of T to the ciphertext, retrieving $P(X)$. $P(X)$ is inverted, using e.g. Berlekamps algorithm [45], and finally the inverse of S is applied to obtain the plaintext.

The security of HFE is based on the difficulty of solving quadratic equations over $GF(q)$. While it is feasible to invert the polynomial $P(X)$ (because of the bound on the degree of $P(X)$), the problem of computing $x_0, \ldots, x_{n-1}$ from

$$P_0(x_0, \ldots, x_{n-1}), \ldots, P_{n-1}(x_0, \ldots, x_{n-1})$$

is NP-complete.

In [40] Kipnis and Shamir convert the original problem of solving n quadratic equations in n variables over $GF(q)$ into a problem of solving $n(n-r)$ quadratic equations in $r(n-r)+n$ variables over $GF(q^n)$.

$$\{G_0(X_0, \ldots, X_{r(n-r)+n}), G_1(X_0, \ldots, X_{r(n-r)+n}), \ldots, G_{n(n-r)}(X_0, \ldots, X_{r(n-r)+n})\}.$$

For $r << n$ this is approximately n^2 equations in rn variables. If one can solve these equations then one can recover the affine transformation T and subsequently $P(X)$ and S (this is proven in [40]). For the purpose of solving these equations Kipnis and Shamir propose a new technique called Relinearization, as an extension of the basic linearization technique. Moreover they claim that it, for a given value $\epsilon \geq 0.1$, is capable of solving a random system of ϵn^2 equations in n variables, in polynomial time. This claim has been widely discussed [49, 18] for several reasons which we describe later.

5.2 Linearization

Consider a set of m equations of degree d in n variables

$$\mathcal{E}: \left(\begin{array}{rcl} f_0(x_0, \ldots, x_{n-1}) & = & 0 \\ f_1(x_0, \ldots, x_{n-1}) & = & 0 \\ \cdots & & \\ \cdots & & \\ f_{m-1}(x_0, \ldots, x_{n-1}) & = & 0. \end{array} \right)$$

The basic linearization technique replaces any monomial $\prod_{i=0}^{n-1} a_i x_i^{b_i}$ of degree greater than one by a new auxiliary variable y_j to obtain a linear system. The system is then, if possible, solved by Gaussian elimination.

Definition 5.1 (Linearization degree)**.** *The linearization degree is the maximum degree of the monomials which are replaced by new auxiliary variables in linearization.*

The time complexity of Gaussian elimination is $\mathcal{O}(l^3)$ and the memory complexity is $\mathcal{O}(l^2)$, where l is the number of variables upon linearization. The method of linearization and performing Gaussian elimination is sometimes referred to as the *multivariate extension of Gaussian elimination*. In some cases the linearized system of equations contains some of the original variables and sometimes the solution to these are found directly upon Gaussian elimination. In other cases, where for instance the monomials

$$x_i^2, x_j^2, x_i x_j$$

are replaced by

$$y_{ii}, y_{jj}, y_{ij},$$

one might find the value of e.g. y_{ii}, y_{jj} and y_{ij} but neither the value of x_i nor x_j. In this case the square roots of y_{ii} and y_{jj} are extracted to find the set of possible solutions to x_i and x_j and subsequently to $x_i x_j$. By comparing the suggested values of $x_i x_j$ to the actual value of y_{ij} some (maybe all) wrong solutions are filtered out. If $\mathcal{E}$ is defined over a finite field $GF(q)$ the field equations $x_i^q + x_i = 0$ are applied to reduce x_i^q to x_i, for all $i = 0, \ldots, n-1$. In the case where $GF(q) = GF(2)$ the monomials x_i^2 reduce to x_i, for $i = 0, \ldots, n-1$, thus extracting square roots and filtering is not necessary. Finally, in some cases Gaussian elimination does not provide the solution to neither the quadratic nor the linear terms of the original system. In this case the method of linearization fails.

Example 5.2. *Consider the four equations over* $GF(2)[x_0, x_1, x_2]$

$$\begin{aligned} 0 &= x_0 + x_1 \\ 0 &= x_0 x_2 + x_1 x_2 + x_0 \\ 0 &= x_0 x_2 + x_0 + x_1 + x_2 \\ 0 &= x_1 x_2 + x_0 + x_1 + x_2 \end{aligned}$$

To solve the system by linearization we introduce a new variable y_{ij} *for each quadratic monomial* $x_i x_j$ *occurring in the equations. There are* $\binom{3}{2} = 3$ *quadratic and* 3 *linear*

monomials in $GF(2)[x_0, x_1, x_2]$ *but only* 5 *of them are present in this case*

$$\begin{aligned} 0 &= x_0 + x_1 \\ 0 &= y_{02} + y_{12} + x_0 \\ 0 &= y_{02} + x_0 + x_1 + x_2 \\ 0 &= y_{12} + x_0 + x_1 + x_2. \end{aligned}$$

After Gaussian elimination (and back substitution) we obtain

$$\begin{aligned} 0 &= y_{12} + x_2 = x_2(x_1 + x_2) \\ 0 &= x_0 \\ 0 &= x_1. \end{aligned}$$

$x_0 = 0$ *and* $x_1 = 0$ *are given directly upon Gaussian elimination while the value of* x_2 *is obtained by insertion of the value of* x_1 *in the first equation* $x_2(x_1 + x_2) = x_2^2 = x_2 = 0$.

For a quadratic set of equations in $GF(2)[x_0, \ldots, x_{n-1}]$ the linearization technique introduces (at most) $n + \binom{n}{2} = \frac{n(n+1)}{2}$ new variables thus in total the system contains $\frac{n(n+3)}{2}$ variables. Provided that the equations are linearly independent and the system has a unique solution one expects to find the solution if $m \geq \frac{n(n+3)}{2}$. If $m << \frac{n(n+3)}{2}$ the method will probably fail.
Recall that in [40] Kipnis and Shamir derived a set of about $m = n^2$ equations in rn variables (for $r << n$) for HFE. By linearizing the equations we obtain a set of $m = n^2$ equations in $rn + \binom{rn}{2} = \frac{rn(rn+1)}{2} \approx \frac{r^2n^2}{2}$ variables. According to our discussion this system is solvable if $n^2 \geq \frac{r^2n^2}{2}$ equivalently $\frac{1}{r^2} \geq \frac{1}{2}$. For HFE $\frac{1}{r^2} < \frac{1}{2}$ and the system is not solvable by Gaussian elimination. For the purpose of solving such equations, Kipnis and Shamir propose the Relinearization method as an extension to the linearization technique.

5.3 Relinearization

The method of Relinearization is a dedicated method for solving $m = \epsilon n^2$ equations in n variables, where $0.1 \leq \epsilon < \frac{1}{2}$. Relinearization extends the method of linearization by adding extra equations to the linearized system of equations and then linearizing again. In the following we explain Relinearization for solving quadratic equations. The method extends in a straight forward way to equations of higher

degree. Consider a set of m quadratic equations in n variables over $GF(q)$

$$\mathcal{E}: \left(\begin{array}{rcl} f_0(x_0, \ldots, x_{n-1}) & = & 0 \\ f_1(x_0, \ldots, x_{n-1}) & = & 0 \\ \cdots & & \\ \cdots & & \\ f_{m-1}(x_0, \ldots, x_{n-1}) & = & 0. \end{array} \right)$$

Step 1 Replace each quadratic term $x_i x_j$ by a new variable y_{ij} (linearization) .

Step 2 Perform Gaussian elimination on the new linear system and write, for each variable y_{ij} a parametric description $y_{ij} = a_0 z_0 + \ldots + a_k z_k$, introducing the new variables $z_0, z_1, \ldots, z_k$.

Step 3 Generate quadratic equations in the variables $z_0, \ldots, z_k$ induced by commutativity of multiplication $y_{ij}y_{kl} = y_{ik}y_{jl} = y_{il}y_{jk}$. Add the generated equations to $\mathcal{E}$

Step 4 Repeat linearization, this time on the new quadratic equations in $z_0, z_1, \ldots, z_k$. Perform Gaussian eliminations on the linearized equations. The goal is to obtain the solution to $z_0, z_1, \ldots, z_k$.

Step 5 The possible set of solutions to y_{ij} and y_{kl} is derived from the values of $z_m = y_{ij}^2$ and $z_n = y_{kl}^2$. Wrong solutions are filtered out by combining the solutions and comparing them to the value of $z_k = y_{ij}y_{kl}$. Once the solution of $y_{ij} = x_i x_j$ is known the filtering process is repeated to obtain the solution of x_i and x_j.

Example 5.3 (Relinearization)**.** *Consider the four linearized equations modulo 7 below*

$$\begin{array}{rcl} y_{12} & = & z + 1 \\ y_{13} & = & z \\ y_{34} & = & z + 5 \\ y_{24} & = & z + 2 \end{array}$$

use the relation $y_{12}y_{34} = y_{13}y_{24}$ *to generate the equation*

$$\begin{array}{rcl} (z+1)(z+5) & = & z(z+2) \mod 7 \Leftrightarrow \\ z^2 + 6z + 5 & = & z^2 + 2z \mod 7 \Leftrightarrow \\ 4z & = & 2 \mod 7 \Leftrightarrow \\ z & = & 4 \mod 7 \end{array}$$

Following $y_{12} = 5, y_{13} = 4, y_{24} = 6, y_{34} = 2$.

In Example 5.3 we do not need to linearize in Step 4 of the algorithm because z^2 cancels out. However, the example shows how we benefit from exploiting the relation $y_{12}y_{34} = y_{13}y_{24}$ which is the idea of the Relinearization method.
According to Kipnis and Shamir [40] the attack is expected to succeed when

$$\frac{m^4}{12} \geq \frac{((\frac{1}{2} - \epsilon)n^2)^2}{2}.$$

However, this bound is based on an inadequate approximation on the generated number of equations and auxiliary variables. In [49] a more accurate bound is given. The procedure is to count the number of equations that are generated in Step 3. Given that the equations are linearly independent we expect to be able to solve the equations by linearization when there are about as many equations as monomials. Consider the following categories of monomials where $x_a \neq x_b \neq x_c \neq x_d$

Type 1 $x_a^4 = y_{aa}^2$: Generates no equations.

Type 2 $x_a^3 x_b = y_{aa}y_{ab}$: Generates no equations.

Type 3 $x_a^2 x_b^2 = x_a x_b x_a x_b \Rightarrow y_{aa}y_{bb} = y_{ab}^2$: Generates one equation. The number of "Type 3" equations is $\binom{n}{2} = \frac{n(n-1)}{2}$.

Type 4 $x_a^2 x_b x_c = x_a x_b x_a x_c \Rightarrow y_{aa}y_{bc} = y_{ab}y_{ac}$: Generates one equation. The number of "Type 4" equations is $n + \binom{n-1}{2} = \frac{n(n-1)(n-2)}{2}$ (because one can select x_a in n ways and x_b, x_c in $\binom{n-1}{2}$ ways).

Type 5 $x_a x_b x_c x_d = x_a x_c x_b x_d = x_a x_d x_b x_c \Rightarrow y_{ab}y_{cd} = y_{ad}y_{bc} = y_{ac}y_{bd}$: Generates two additional equations. The number of "Type 5" equations is $2\binom{n}{4} = \frac{n(n-1)(n-2)(n-3)}{12}$.

In total one can generate

$$E = \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{2} + \frac{n(n-1)(n-2)(n-3)}{12} = \frac{n^4}{12} + \frac{n^2}{12} \qquad (5.1)$$

equations (as stated in [49]). In Step 1 $n + \binom{n}{2} = \frac{n(n+1)}{2}$ new variables are introduced by linearization thus the parametric description introduces $\frac{n(n+1)}{2} - m$ new variables $z_0, \ldots, z_k$. The equations generated in Step 3 contain

$$V = \binom{\frac{n(n+1)}{2} - m}{2} + 2(\frac{n(n+1)}{2} - m) = \frac{(\frac{n(n+1)}{2} - m)(\frac{n(n+1)}{2} - m + 3)}{2} \qquad (5.2)$$

monomials i.e. new variables in Step 4. One therefore expects to be able to solve the system when

$$\frac{n^4}{12} + \frac{n^2}{12} \geq \frac{(\frac{n(n+1)}{2} - m)(\frac{n(n+1)}{2} - m + 3)}{2}. \tag{5.3}$$

5.3.1 Complexity

In [40] the Relinearization attack is claimed to run in polynomial time. The estimate is based on the assumption that all equations generated in Step 3 of the algorithm are linearly independent. If that was the case one could apply the bound of Equation 5.3 to determine which systems the Relinearization technique could solve at linearization degree 4. In this case the runtime of Relinearization would be bounded by the complexity of Gaussian elimination $\mathcal{O}(l^3)$ where l is the number of variables in the linearized system. However, there is no reason to believe that the equations generated in Step 3 are all linearly independent, in fact simulations [18] show the opposite.
At linearization degree 4 (i.e. the equations in $z_1, \ldots, z_k$ have degree 2) it is claimed in [40] that one can prove that the equations generated in Step 3 are linearly independent. If you consider only trivial linear dependencies then for each 4-tuple x_a, x_b, x_c, x_d it is true that two out of three equations

$$y_{ab}y_{cd} = x_a x_b x_c x_d = x_a x_c x_b x_d = y_{ac}y_{bd}$$

$$y_{ab}y_{cd} = x_a x_b x_c x_d = x_a x_d x_c x_b = y_{ad}y_{bc}$$

$$y_{ac}y_{bd} = x_a x_c x_b x_d = x_a x_d x_c x_b = y_{ad}y_{bc}$$

are linearly independent. However, when the parametric descriptions of $y_{ab}y_{cd}, y_{ab}y_{bd}, y_{ad}y_{bc}$ are inserted this property may be lost.

Example 5.4. *Consider the parametric description of the linearized variables:*

$$\begin{aligned}
y_{12} &= x_1x_2 = z_1 + z_2 \\
y_{34} &= x_3x_4 = z_1 - z_2 \\
y_{13} &= x_1x_3 = -z_2 + z_3 \\
y_{24} &= x_2x_4 = z_2 + z_3 \\
y_{56} &= x_5x_6 = z_1 + z_4 \\
y_{78} &= x_7x_8 = z_1 - z_4 \\
y_{57} &= x_5x_7 = z_3 - z_4 \\
y_{68} &= x_6x_8 = z_3 + z_4
\end{aligned}$$

Generating the equations from $y_{12}y_{34} = y_{13}y_{24}$ *and* $y_{56}y_{78} = y_{57}y_{68}$

$$\begin{aligned}(z_1 + z_2)(z_1 - z_2) &= (-z_2 + z_3)(z_2 + z_3)\\ (z_1 + z_4)(z_1 - z_4) &= (z_3 - z_4)(z_3 + z_4)\end{aligned}$$

we get

$$\begin{aligned}z_1^2 - z_2^2 &= -z_2^2 + z_3^2\\ z_1^2 - z_4^2 &= z_3^2 - z_4^2\end{aligned}$$

which are linearly dependent.

If Relinearization does not work at degree 4 a natural extension is to take the method to the next level, namely degree-six linearization, where we consider permutations of six-tuples $(x_a, x_b, x_c, x_d, x_e, x_f)$ instead. Formulas for counting equations and variables are given in [49] and in [18] the linear dependencies are explored further.
As a conclusion the linearization degree required for the Relinearization attack to work remains unknown because no-one so far has been capable of determining the number of linearly independent equations generated by the method.

5.4 The extended linearization attack (XL)

The Extended Linearization (XL) attack was proposed by Courtois, Klimov, Patarin, and Shamir in 2000 [18]. The attack is strongly inspired by the Relinearization method but claimed (by the authors) to be both simpler and more powerful. The attack has successfully been applied to break the stream cipher Toyocrypt [17]. However, as in the case of Relinearization the complexity of the XL algorithm has been subject to much discussion, and to this day no good estimate exists.
Given a set of m equations of degree greater than one in n variables

$$\mathcal{E}: \begin{pmatrix} f_0(x_0, \ldots, x_{n-1}) &= 0\\ f_1(x_0, \ldots, x_{n-1}) &= 0\\ \cdots & \\ \cdots & \\ f_{m-1}(x_0, \ldots, x_{n-1}) &= 0 \end{pmatrix}.$$

Choose the *linearization degree* $D > d$ where d is the lowest degree of an equation appearing in $\mathcal{E}$.

Step 1 Multiply each equation $f_i \in \mathcal{E}$ by all monomials, one at the time, of degree less than or equal to $D - d_i$, where $d_i = deg(f_i)$ is the degree of f_i.

Step 2 Replace each monomial of degree greater than one by a new auxiliary variable. Perform Gaussian elimination on the linear system.

Step 3 If univariate equations appear upon Step 2, solve these by e.g. Berlekamps algorithm (see for example [45]). Else the algorithm terminates.

Step 4 Simplify the equations by the solutions obtained in Step 3 and go to Step 1 to find the solution of other variables.

As explained in Section 5.2 if $\mathcal{E}$ is defined over a finite field $GF(q)$ the field equations $x_i^q - x_i = 0$ are applied to reduce x_i^q to x_i, for all $i = 0, \ldots, n-1$. This is sometimes referred to as *Reduced XL*. If q is small, solving a univariate polynomial equation in Step 3 is done by testing which of the q elements solves the equation. In the case where $q = 2$ any univariate equation reduces to a linear equation which provides the solution directly, when all univariate (i.e. linear) equations are obtained.

Example 5.5. *Given the three polynomial equations*

$$\mathcal{E} : \left(\begin{array}{c} x_0x_1 + x_2 = 0 \\ x_0x_2 + 1 = 0 \\ x_1x_2 + 1 = 0 \end{array} \right)$$

in $GF(2)[x_0, x_1, x_2]$. *By applying XL with linearization degree* $D = 3$ *we obtain in Step 1 the following equations in addition to the equations in* $\mathcal{E}$

$$\begin{aligned} 0 &= x_0x_1 + x_0x_2 \\ 0 &= x_0x_1 + x_1x_2 \\ 0 &= x_0x_1x_2 + x_2 \\ 0 &= x_0x_2 + x_0 \\ 0 &= x_0x_1x_2 + x_1 \\ 0 &= x_0x_2 + x_2 \\ 0 &= x_0x_1x_2 + x_0 \\ 0 &= x_1x_2 + x_1 \\ 0 &= x_1x_2 + x_2 \end{aligned}$$

Upon linearization followed by Gaussian elimination and substitution back into the

original variables we obtain

$$\begin{aligned} 0 &= x_0x_1x_2 + x_2 \\ 0 &= x_0x_1 + x_2 \\ 0 &= x_0x_2 + x_0 \\ 0 &= x_1x_2 + x_1 \\ 0 &= x_0 + x_2 \\ 0 &= x_1 + x_2 \\ 1 &= x_0 \\ 1 &= x_1 \\ 1 &= x_2 \end{aligned}$$

5.4.1 Complexity

The complexity of the XL algorithm depends on the number of linearly independent equations generated by the method. In [18] it is assumed that the equations generated by XL are uniformly random distributed and thus one expects that *almost all equations generated by XL are linearly independent*. Based on this assumption one can derive the linearization degree D as follows. Given a set of m quadratic equations in n variables the XL attack generates

$$E = m \sum_{i=0}^{D-2} \binom{n}{i}$$

equations at linearization degree D. The number of variables in the linearized system is at most

$$V = \sum_{i=0}^{D} \binom{n}{i}.$$

The attack will succeed when the numbers of variables and equations are approximately the same

$$m \sum_{i=0}^{D-2} \binom{n}{i} \approx \sum_{i=0}^{D} \binom{n}{i}.$$

Thus one gets the following bound on D

$$D \approx \frac{n}{\sqrt{m}}. \tag{5.4}$$

Unfortunately (seen from the attackers point of view) the assumption is not true. In practice it (often) turns out that many of the equations generated in the XL attack are linearly dependent. In Example 5.5 there are many linear dependencies among the equations generated in Step 1. In general one can find a number of obvious linear dependencies among the equations generated by XL.

Example 5.6. *Consider the equation $f \in \mathcal{E}$ in $GF(2)[x_0, \ldots, x_{n-1}]$*

$$f : m_1 + m_2 + m_3 = 0$$

where $m_1, m_2, m_3 \in GF(2)[x_0, \ldots, x_{n-1}]$ are monomials of degree d. For linearization degree $D \geq 2d$, $m_1 f$, $m_2 f$, and $m_3 f$ are among the equations generated in Step 1. These are linearly dependent since

$$(m_1 + m_2 + m_3)f = f^2 = f$$

in

$$GF(2)[x_0, \ldots, x_{n-1}]/\langle x_0^2 + x_0, \ldots, x_{n-1}^2 + x_{n-1} \rangle \, .$$

Clearly the bound on the required linearization degree given by Equations 5.4 is not an exact number. We have already argued that it is not an upper bound, however it is not a lower bound either. The reason is that it is assumed that one needs as many equations as there are monomials of degree less than or equal to D. However, in practice some monomial might not appear in the generated equations and some might cancel out after linearization and Gaussian elimination. To this day the complexity of the XL method remains unknown. Moreover as opposed to the Gröbner bases algorithm described in Chapter 4 there is no guarantee that the XL will terminate. If Step 2 does not produce a univariate polynomial the algorithm gets stuck. One can construct examples for which XL fails no matter the choice of the degree D [16].

5.4.2 Other variants

The FXL algorithm is suggested as an extension to XL in [18]. The method basically fixes a number of variables before applying the usual XL algorithm. No results on this approach has (to our knowledge) been reported. Chapter 8 concerns application of probabilistic equations in algebraic attacks. In relation to this we explore an FXL like approach combined with Gröbner bases computations.
As mentioned, XSL [20] was proposed along with some very controversial statements regarding its impact on important ciphers like AES. The idea of XSL is to reduce the number of monomials generated in XL by "carefully selecting" a subset of equations

and monomials. The algorithm has been subject to much critics and after analysis in [14] it is concluded that the method does not provide an effective method for solving the AES system of equations. For this reason we do not discuss the algorithm further.

Chapter 6

Analysis of the Algebraic Attacks

Although it is possible to establish a set of low-degree equations for the secret key (-bits) of many block ciphers, as described in Chapter 3, solving them efficiently is far from trivial. As mentioned the problem regarding the Gröbner bases techniques is to determine the exact complexity. So far the conclusion is that memory requirements and time consumption obstructs the application even on small ciphers (as we explore in Chapter 7). The XL method has been surrounded by criticism. However, the advantage of the XL method is that for a given degree d, one has a good estimate on the time complexity of the approach. The drawback of applying XL to block ciphers is that it is hard to determine the degree d for which the attack will succeed. Moreover, the original XL attack is not guaranteed to succeed. If, at a certain degree d, the number of linearly independent equations is about as big as the number of terms occurring in the equations, the XL attack succeeds. The problem is to determine the number of linearly independent equations generated by XL at degree d. This is a problem no-one has been able to solve yet, though some results suggest that d may be large for modern block ciphers [26].

A first step towards the solution of the problem could be to determine exactly how many linearly independent equations one can obtain by multiplication of an initial set of equations. In this chapter we approach this subject. Initially we describe a variant of XL, which is a natural extension of XL. By using Gröbner bases theory, we argue that this variant is guaranteed to solve the equations (provided that a solution exists) for some degree d. For further information on the relation between the XL algorithm and the Gröbner bases techniques the reader is referred to [61].

The purpose is not to propose a new algebraic attack but rather to provide a tool to analyze the application of XL and related methods on block ciphers. The technique is to treat the equations of respectively the linear layer (which we denote pLayer or simply $L(\cdot)$) and the S-box layer separately in the initial multiplication step. For

this approach we can determine the exact number of linearly independent equations of higher degrees generated from each set. However, it is clear that in order to obtain the solution we will have to combine the two sets of equations but it is yet unclear how the equations interact in e.g. an Gaussian elimination as the amount of linear dependencies is unknown. While the method does not give us the exact complexity of the attacks on block ciphers, it contributes with a new angle on the algebraic attacks.
We proceed as follows. Section 6.1 briefly describes the systems of equations we attempt to solve. In Section 6.2 the XL-like iterated attack is described and we argue that it will return a solution given that one exists. Section 6.3 contains two results about the number of linearly independent equations one can generate from the linear layer $L(\cdot)$ respectively the non-linear (S-box) layer of a block cipher. Moreover we give a systematic way of generating these, which ensures that only linearly independent equations are generated in the first part of the attack. In Section 6.4 the results are applied to AES-128 and to a variant of AES-128. In Section 6.5 we present a number of simulations on small block ciphers and finally in Section 6.6 we discuss the output of the work.

6.1 The equations

As described in Chapter 3 there are more ways to obtain an algebraic description of a block cipher. In this chapter we apply the method where all key variables are eliminated. Consider an n_b-bit block cipher with n_r rounds of encryption, applying n_{Sbox} S-boxes per round. Assume that each (non-linear) S-box is described by eq_{Sbox} equations of some degree d. To keep the description simple we consider a strictly linear key schedule. However, the method extends to non-linear key schedules as well and in Section 6.4.1 the approach is applied to AES.
Let X_i, and Y_i denote respectively the input, the output of the S-box layer of round i and let K_i denote the round key (see Figure 3.6). Recall that we combine the equations

$$L(Y_i) \oplus K_i = X_{i+1}$$

and

$$L(Y_j) \oplus K_j = X_{j+1}$$

for all $i < j$, where $L(\cdot)$ is the linear transformation. For each pair (i, j) this yields n_b linear equations over $GF(2)$ in bits from X_{i+1}, X_{j+1}, Y_i, and Y_j. In total we obtain a system of $n_b \cdot n_r$ linear equations and $n_{Sbox} \cdot eq_{Sbox} \cdot n_r$ non-linear equations. The total number of variables is $2n_b \cdot n_r$.

6.2 Iterated XL

For the purpose of the systematic techniques and the accompanying analysis of this chapter we need to specify an admissible monomial order (compatible with multiplication) in the polynomial ring $GF(2)[x_1, \ldots, x_n]$. The order must be a graded order (an order for which the total degree of the monomial is the main criterion), and for the remainder of this chapter, we specify it to be the graded reverse lexicographic order (grevlex), see Definition 4.7. The notion of the head term HT(eq) of a (polynomial) equation

$$eq(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{k} a_i m_i = 0, \quad a_i \in GF(2) \quad \text{for} \quad \mathrm{i} = 0, 1, \ldots, \mathrm{k}$$

refers to the head term of the polynomial

$$a_0 + \sum_{i=1}^{k} a_i m_i$$

(see Definition 4.8). In the following section we describe an iterated XL-like attack (based on [18] and [20]). The objective is as usual to find the secret key used for encryption. Note that if one has the solution to the linear equations, then these can be used to find the value of the secret key. We assume that for a given plaintext and ciphertext only a (very) few values of the secret key could have been used in the encryption.

6.2.1 The basic attack for degree d

We refer to Chapter 3 (or the review in Section 6.1) for the method of setting up the equations over a block cipher and let $n = 2n_b \cdot n_r$ denote the number of variables in the system. Let $\mathcal{T}$ denote the set of all monomials over $GF(2)[x_1, \ldots, x_n]$. Throughout the attack any monomial of the form x_i^2 is immediately reduced to x_i, for $i = 1, \ldots, n$. In this way we implicitly include the field equations $x_i^2 + x_i = 0$ in $GF(2)[x_1, \ldots, x_n]$. The idea of the algorithm is that by multiplying monomials onto the equations from the linear layer $\mathcal{L}$ and from the S-box layer $\mathcal{S}$ one may get degree-d equations from each of the two sets with identical head terms. In a Gaussian elimination this may result in equations with head terms of degrees less than d, say degree $d-1$. Such equations may have identical head terms with other equations of degree $d-1$, which may result in equations with head terms of even lower degrees. This may in turn lead to linear equations which were not in the initial set of equations. Prior to the attack we specify the degree d.

Step 1 Generate the set of equations $\mathcal{E} = \cup_{\tilde{d} \leq d} \{\mathcal{E}_L^{\tilde{d}} \cup \mathcal{E}_S^{\tilde{d}}\}$:
$\mathcal{E}_L^{\tilde{d}}$ is the maximum set of degree-$\tilde{d}$ equations, one can generate from $\mathcal{L}$. A systematic procedure is given by the algorithm in Figure 6.1, Page 88.
$\mathcal{E}_S^{\tilde{d}}$ is the maximum set of degree-$\tilde{d}$ equations, one can generate from $\mathcal{S}$. A systematic procedure is given by the algorithm in Figure 6.2, Page 92.

Step 2 Arrange the equations of $\mathcal{E}$ in a matrix and perform the multivariate extension of Gaussian elimination on this, resulting in $\mathcal{E}_g$.

Step 3 Let $\mathcal{F} = \emptyset$. For all equations eq in $\mathcal{E}_g$ which have a head term different from the head term of any equation in $\mathcal{E}$, add eq to $\mathcal{F}$.

Step 4 If $\mathcal{F}$ is empty, then stop and return $\mathcal{E}_g$.

Step 5 Compute the maximum set of equations $\mathcal{F}_m$ of degree at most d, one can generate from $\mathcal{F}$. In practice this is done by multiplying all equations $f_i \in \mathcal{F}$ by all monomials $t_i \in T$ for which $deg(t_i f_i)) \leq d$, one at the time. For each generated equation $t_i f_i$ proceed as follows :

If $\mathrm{HT}(t_i f_i)$ is not head term of any equation in $\mathcal{E}_g$, $t_i f_i$ is added to the sets $\mathcal{E}_g$ and $\mathcal{F}_m$.

Else apply the polynomial division algorithm (described in Chapter 4) to reduce f modulo the polynomials of $\mathcal{E}_g$. Let f_r denote the remainder from the polynomial division. If $f_r \neq 0$ add f_r to each of the sets $\mathcal{E}_g$ and $\mathcal{F}_m$.

Step 6 Set $\mathcal{F} = \mathcal{F}_m$ and $\mathcal{F}_m = \emptyset$, and go to Step 4.

After termination, find the possible values of the secret key from the linear equations in $\mathcal{E}$. Note that the set of linear equations at this point could contain many possible (including wrong) values of the key. If too many values remain, the attack has failed for the particular value of d.
Note that upon Step 3 we can discard $\mathcal{E}$. Thus the memory requirements are bounded by the size of $\mathcal{E}_g$ and $\mathcal{F}$.
For a given degree d, this bound is $(\sum_{i=0}^{d} \binom{n}{i})^2$. On the other hand, our attack is not guaranteed to succeed for any value of $d < n$. With $d = n$ our attack is guaranteed to work, since in this case it is a redundant version of Buchberger's algorithm and we have assumed that the system of equations has at least one solution.

6.3 Counting linearly independent equations

In the following sections we devise a method for counting the exact number of linearly independent equations of degrees $d \geq 2$ one can generate from multiplying

monomials onto the elements in each of the two sets. The following two lemmas follow from basic linear algebra.

Lemma 6.1. *Let*

$$\mathcal{F} = \{eq_1, \ldots, eq_m\}$$

be a set of m linearly independent polynomial equations, and $\leq$ an order on the terms of $\mathcal{F}$. Then one can write a set of m linearly independent equations

$$\{eq'_1, \ldots, eq'_m\}$$

where the head terms

$$\mathrm{HT}(eq'_1), \ldots, \mathrm{HT}(eq'_m)$$

are all distinct and where each equation eq'_i is a linear combination of $eq_1, \ldots, eq_m$.

Lemma 6.2. *Let*

$$\mathcal{F} = \{eq_1, \ldots, eq_m\}$$

be a set of polynomial equations and $\leq$ an order on the terms of $\mathcal{F}$. Then, if the head terms

$$\mathrm{HT}(eq_1), \ldots, \mathrm{HT}(eq_m)$$

are all distinct, the equations $eq_1, \ldots, eq_m$ are linearly independent.

The head terms of linearly independent equations are not necessarily distinct. However, the consequence of Lemma 6.1 is that, any set of linearly independent equations can be linearly transformed into a set of equally many linearly independent equations where the head terms, according to a term order $\leq$, are distinct. The remainder of this section provides tools to determine the upper bound on the number of linearly independent equations generated in Step 1 of the algorithm of Section 6.2.1. Later Lemmas 6.1 and 6.2 are applied to determine the lower bound on this number.
First a few definitions [22].

Definition 6.1. *Let $I \subset k[x_1, \ldots, x_n]$ be an ideal. The radical of I is the set*

$$\sqrt{I} = \{g \in k[x_1, \ldots, x_n] \quad | \quad g^m \in I \text{ for some } m \geq 1\}$$

An ideal is said to be a radical ideal if $\sqrt{I} = I$.

In algebraic geometry the algebraic dimension of an ideal is an important concept. A zero-dimensional ideal over k is an ideal where the number of solutions in the algebraic closure of the field k is finite. We restrict ourselves to this because in algebraic attacks on block ciphers we are always dealing with zero-dimensional ideals. Meanwhile the notation $\mathrm{Dim}_k k[x_1, \ldots, x_n]/I$ denotes the dimension of the vector space $k[x_1, \ldots, x_n]/I$. In [21, Theorem 2.10] we find the following useful theorem concerning $\mathrm{Dim}_{\mathbb{C}}\mathbb{C}[x_1, \ldots, x_n]/I$.

Theorem 6.1. *Let I be a zero-dimensional ideal in $\mathbb{C}[x_1, \ldots, x_n]$ and let $A = \mathbb{C}[x_1, \ldots, x_n]/I$. Then $Dim_{\mathbb{C}}A$ is greater than or equal to the number of points in $V(I)$. Moreover, equality occurs if and only if I is a radical ideal.*

Theorem 6.1 concerns a polynomial ring over the complex numbers, but it actually applies to any polynomial ring $A = k[x_1, \ldots, x_n]$, where k is an algebraically closed field.

Definition 6.2. *A field k is said to be algebraically closed if every polynomial in one variables of degree at least one, with coefficients in k, has a root in k.*

Though $GF(2)$ is not algebraically closed the theorem applies if we take for k the algebraic closure of $GF(2)$, which we denote $\overline{GF(2)}$.

Lemma 6.3. *Let $I \in GF(2)[x_1, \ldots, x_n]$ be a polynomial ideal generated by m polynomials $f_1, \ldots, f_m$ and the polynomials $x_i^2 + x_i$, for $i = 1, \ldots, n$*

$$I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle,$$

for which the variety $V(I)$ is a non-empty set. Let $|V(I)|$ denote the number of points in $V(I)$. Then

$$Dim_{GF(2)}(GF(2)[x_1, \ldots, x_n]/\ I) = |V(I)|.$$

Proof. I is a radical ideal since $f^2 = f$ for any element

$$f \in GF(2)[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle.$$

So if $f^m \in I$, then

$$f = (f - f^m) + f^m \in I.$$

Since $x_i^2 + x_i \in I$ for $i = 1, \ldots, n$, it follows that any solution over the algebraic closure $\overline{GF(2)}$ is already defined over $GF(2)$. In other words $|V(I)|$ over $GF(2)$ equals $|V(I)|$ over $\overline{GF(2)}$.
Also, we have that

$$\begin{aligned} \mathrm{Dim}_{\overline{GF(2)}}\overline{GF(2)}[x_1, \ldots, x_n]/I &= \mathrm{Dim}_{\overline{GF(2)}}\overline{GF(2)}[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle \\ &\quad -\mathrm{Dim}_{\overline{GF(2)}} I/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle \end{aligned}$$

and likewise

$$\begin{aligned} \mathrm{Dim}_{GF(2)}GF(2)[x_1, \ldots, x_n]/I &= \mathrm{Dim}_{GF(2)}GF(2)[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle \\ &\quad -\mathrm{Dim}_{GF(2)} I/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle\,. \end{aligned}$$

where

$$\mathrm{Dim}_{GF(2)} GF(2)[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle = \sum_{i=0}^{n} \binom{n}{i} = 2^n$$

and

$$\mathrm{Dim}_{\overline{GF(2)}} \overline{GF(2)}[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle = \sum_{i=0}^{n} \binom{n}{i} = 2^n \,.$$

By considering the vector space $I/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$ over respectively $GF(2)$ and $\overline{GF(2)}$ we find that

$$\mathrm{Dim}_{\overline{GF(2)}} I/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle = \mathrm{Dim}_{GF(2)} I/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle \,,$$

hence

$$\mathrm{Dim}_{\overline{GF(2)}} \overline{GF(2)}[x_1, \ldots, x_n]/I = \mathrm{Dim}_{GF(2)} GF(2)[x_1, \ldots, x_n]/I$$

thus by Theorem 6.1 the result follows. □

We note that Theorem 6.1 is applied over $GF(2)$ several places in literature e.g. in [2] with reference to [21].

Lemma 6.4. *Let there be given m linearly independent equations in n variables over $GF(2)$, and a term order $\leq$. Let $|V|$ be the number of common solutions to these equations. Consider the set of equations obtained from multiplying the equations with all monomials in the n variables, one at the time, and write the maximal set of linearly independent equations*

$$eq_1, \ldots, eq_{max}$$

where the head terms

$$\mathrm{HT}(eq_1), \ldots, \mathrm{HT}(eq_{max})$$

are distinct, then

$$max = 2^n - |V|.$$

Proof. Let

$$I = \langle f_1, \ldots, f_{max}, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$$

where f_i is the polynomial corresponding to the polynomial equations eq_i for

$$i = 1, \ldots, max \,.$$

Consider the linear map

$$\varphi: \quad GF(2)[x_1,\ldots,x_n]/\langle x_1^2+x_1,\ldots,x_n^2+x_n\rangle \rightarrow GF(2)[x_1,\ldots,x_n]/I$$

where

$$\varphi(x_i) = [x_i] \quad \text{for} \quad \text{i} = 1,\ldots,\text{n},$$

where $[x_i]$ is the remainder class of $GF(2)[x_1,\ldots,x_n]/I$. The polynomial ring

$$GF(2)[x_1,\ldots,x_n]/\langle x_1^2+x_1,\ldots,x_n^2+x_n\rangle$$

is isomorphic with the vector space $GF(2)^{2^n}$ thus

$$\text{Dim}_{GF(2)}(GF(2)[x_1,\ldots,x_n]/\langle x_1^2+x_1,\ldots,x_n^2+x_n\rangle = 2^n.$$

From linear algebra we have the dimension formula for homomorphisms (e.g page 704 in [35])

$$2^n = \text{Dim}(Im(\varphi)) + \text{Dim}(ker(\varphi)).$$

From Lemma 6.3 we have

$$\text{Dim}(Im(\varphi)) = \text{Dim}_{GF(2)}(GF(2)[x_1,\ldots,x_n]/I) = |V(I)|,$$

thus

$$2^n = |V(I)| + \text{Dim}(ker(\varphi)).$$

To determine $\text{Dim}(ker(\varphi))$ consider the isomorphic $GF(2)$-vector space where any

$$f = a_1 + a_2x_1 + \ldots a_{n+1}x_n + a_{n+2}x_1x_2 + \ldots + a_{2^n}x_1\cdots x_n$$

in

$$GF(2)[x_1,\ldots,x_n]/\langle x_1^2+x_1,\ldots,x_n^2+x_n\rangle$$

is represented by a vector of $GF(2)^{2^n}$

$$f: \quad (a_1,\ldots,,a_{2^n}).$$

Since by assumption each polynomial (equation) f has a unique head term, the vectors in $GF(2)^{2^n}$ are linearly independent and the vector space has dimension max, i.e $\text{Dim}(ker(\varphi)) = max$ thus

$$2^n = |V(I)| + max.$$

□

6.3.1 Equations generated from pLayer ($L(\cdot)$)

The following theorem determines the exact number of equations with distinct degree-d head terms one can obtain from a set of m linearly independent degree-one equations. Note that $\binom{n}{k} = 0$ for $k > n$ by definition.

Theorem 6.2. *Let $\mathcal{E}_L$ be a system of m linearly independent equations of degree one in n variables over $GF(2)$, i.e. $m \leq n$. Assume that $\mathcal{E}_L$ has at least one solution. Choose $d > 0$ such that $n - m \geq d - 1$, and multiply all equations of $\mathcal{E}_L$ by all monomials of degree $d - 1$. Arrange the generated equations in a matrix with the terms sorted according to a graded ordering. By performing Gaussian elimination one gets exactly*

$$N = \binom{n}{d} - \binom{n-m}{d} \tag{6.1}$$

linearly independent equations with distinct degree-d head terms.

Proof. According to Lemma 6.1 the equations of $\mathcal{E}_L$ can be linearly transformed such that each equation has a unique head term. Thus m variables are head term of an equation and $n - m$ variables are not a head term of any equation. Therefore

$$GF(2)[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$$

contains $\binom{n}{d} - \binom{n-m}{d}$ monomials of degree d which are divisible by the head term of at least one equation of $\mathcal{E}_L$. The remaining $\binom{n-m}{d}$ degree-d monomials are not divisible by the head term of any equation of $\mathcal{E}_L$. Thus one can generate at least $\binom{n}{d} - \binom{n-m}{d}$ equations with distinct degree-d head terms i.e., a system of $\binom{n}{d} - \binom{n-m}{d}$ linearly independent equations.
By summing the expression (6.1) for $d = \{0, 1, \ldots, n\}$ we get

$$\sum_{d=0}^{n} \binom{n}{d} - \binom{n-m}{d} = 2^n - 2^{n-m}.$$

According to Lemma 6.4 the maximal number of linearly independent equations with distinct head terms is

$$max = 2^n - |V(I)| = 2^n - 2^{n-m},$$

since a set of m degree-one linearly independent equations has 2^{n-m} common solutions. Therefore Theorem 6.2 gives the exact number of polynomials with distinct head terms one can generate from the equations of $\mathcal{E}_L$. □

Example 6.1. *$\mathcal{E}_{L_1}$ is a set of linear degree-one equations over $GF(2)[x_1, x_2, x_3, x_4]$*

$$\mathcal{E}_{L_1} = \left(\begin{array}{lcccl} eq_1 & : & x_1 + x_3 + 1 & = & 0 \\ eq_2 & : & x_2 + x_3 + x_4 & = & 0\,. \end{array} \right)$$

By multiplication of all equations with all monomials, one at the time, of degree one we generate the set
$\mathcal{E}_{L_2} = \{x_1eq_1, x_2eq_1, x_3eq_1, x_4eq_1, x_1eq_2, x_2eq_2, x_3eq_2, x_4eq_2\}$.

$$\mathcal{E}_{L_2} = \left(\begin{array}{lcccl} x_1eq_1 & : & x_1x_3 & = & 0 \\ x_2eq_1 & : & x_1x_2 + x_2x_3 + x_2 & = & 0 \\ x_3eq_1 & : & x_1x_3 & = & 0 \\ x_4eq_1 & : & x_1x_4 + x_3x_4 + x_4 & = & 0 \\ x_1eq_2 & : & x_1x_2 + x_1x_3 + x_1x_4 & = & 0 \\ x_2eq_2 & : & x_2 + x_2x_3 + x_2x_4 & = & 0 \\ x_3eq_2 & : & x_2x_3 + x_3 + x_3x_4 & = & 0 \\ x_4eq_2 & : & x_2x_4 + x_3x_4 + x_4 & = & 0 \end{array} \right)$$

The equations are arranged in a matrix with the terms sorted with respect to reverse lexicographic degree monomial order

$$x_1x_2 > x_1x_3 > x_1x_4 > x_2x_3 > x_2x_4 > x_3x_4 > x_1 > x_2 > x_3 > x_4 > 1$$

and upon the multivariate extension of Gaussian elimination we obtain

$$\mathcal{E}_{L_2g} = \left(\begin{array}{cc} x_1x_2 + x_2x_3 + x_2 & = 0 \\ x_1x_3 & = 0 \\ x_1x_4 + x_3x_4 + x_4 & = 0 \\ x_2x_3 + x_3x_4 + x_2 + x_4 & = 0 \\ x_2x_4 + x_3x_4 + x_4 & = 0 \end{array} \right)$$

five linearly independent degree-two equations with distinct degree-two head term, as anticipated from Theorem 6.2

Example 6.2. *Consider the same set of equations $\mathcal{E}_{L_1}$ as in Example 6.1*

$$\mathcal{E}_{L_1} = \left(\begin{array}{lcccl} eq_1 & : & x_1 + x_3 + 1 & = & 0 \\ eq_2 & : & x_2 + x_3 + x_4 & = & 0\,. \end{array} \right)$$

From multiplication by all monomials of degree two

$$T = \{x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_3x_4\}$$

we generate the set $\mathcal{E}_{L_3}$.

$$\begin{aligned}\mathcal{E}_{L_3} &= \{x_1x_2eq_1, x_1x_3eq_1, x_1x_4eq_1, x_2x_3eq_1, x_2x_4eq_1, x_3x_4eq_1,\\ &\quad x_1x_2eq_2, x_1x_3eq_2, x_1x_4eq_2, x_2x_3eq_2, x_2x_4eq_2, x_3x_4eq_2\}\end{aligned}$$

$$\mathcal{E}_{L_3} = \left(\begin{array}{lcccl}
x_1x_2eq_1 & : & x_1x_2x_3 & = & 0\\
x_1x_3eq_1 & : & x_1x_3 & = & 0\\
x_1x_4eq_1 & : & x_1x_3x_4 & = & 0\\
x_2x_3eq_1 & : & x_1x_2x_3 & = & 0\\
x_2x_4eq_1 & : & x_1x_2x_4 + x_2x_3x_4 + x_2x_4 & = & 0\\
x_3x_4eq_1 & : & x_1x_3x_4 & = & 0\\
x_1x_2eq_1 & : & x_1x_2 + x_1x_2x_3 + x_1x_2x_4 & = & 0\\
x_1x_3eq_1 & : & x_1x_2x_3 + x_1x_3 + x_1x_3x_4 & = & 0\\
x_1x_4eq_2 & : & x_1x_2x_4 + x_1x_3x_4 + x_1x_4 & = & 0\\
x_2x_3eq_2 & : & x_2x_3x_4 & = & 0\\
x_2x_4eq_2 & : & x_2x_3x_4 & = & 0\\
x_3x_4eq_2 & : & x_2x_3x_4 & = & 0
\end{array}\right)$$

Performing the multivariate extension of Gaussian elimination we obtain the set

$$\mathcal{E}_{L_{3g}} = \left(\begin{array}{cc}
x_1x_2x_3 & = 0\\
x_1x_2x_4 + x_2x_3x_4 + x_2x_4 & = 0\\
x_1x_3x_4 & = 0\\
x_2x_3x_4 & = 0\\
x_1x_2 + x_2x_4 & = 0\\
x_1x_3 & = 0\\
x_1x_4 + x_2x_4 & = 0\,.
\end{array}\right)$$

As anticipated from Theorem 6.2 we obtain $\binom{4}{3} - 0 = 4$ *linearly independent equations with distinct degree-three head term. In addition to the cubic equations we obtain three quadratic equations. Note that these however are linearly spanned by the equations in* $\mathcal{E}_{L_{2g}}$ *of Example 6.1.*

6.3.2 Systematic procedure for pLayer ($L(\cdot)$)

This section describes a method for generating all equations of degree d, over the linear transformation ($L(\cdot)$) of a cipher.
Let $\mathcal{L} = \{eq_1, eq_2, \ldots, eq_m\}$ be a set of m linearly independent equations of degree one over $GF[2](x_1, x_2, \ldots, x_n)$. Assume the system has at least one solution i.e.

$n \geq m$. Choose an admissible monomial order, e.g. grevlex ($\geq$). Let $ht_i = \mathrm{HT}(eq_i)$ denote the head term of equations eq_i with respect to $\geq$.
Express the equations of $\mathcal{L}$ in a form where each equation has a unique head term and sort them according to grevlex where $ht_i > ht_{i+1}$. Arrange the terms in each equation, eq_i, such that

$$eq_i = ht_i + t_j + \ldots + t_k \,.$$

where $ht_i > t_j > \cdots > t_k$.
Figure 6.1 describes a systematic procedure for generating the equations of the linear layer. T denotes the set of monomials in $GF(2)[x_1, \ldots, x_n]$ of degree $d-1$, for a fixed degree d. The subset $T_i \subseteq T$ denotes the set of monomials that have common divisors with ht_i. All generated equations have distinct head term and are therefore linearly independent.

Step 1 Initialize $\mathcal{E}_L = \emptyset$.

Step 2 For all

$$eq_i \in \mathcal{L} \quad \text{for} \quad \mathrm{i} = 1, \ldots, \mathrm{m},$$

compute for all $t \in T \setminus \{T_1 \cup \cdots \cup T_i\}$

$$f = t \cdot eq_i$$

and update

$$\mathcal{E}_{\mathcal{L}} = \mathcal{E}_{\mathcal{L}} \cup f.$$

Step 3 Return $\mathcal{E}_L$.

Figure 6.1: Equations for $L(\cdot)$

Example 6.3. *Consider again the equations from Example 6.1 over* $GF(2)[x_1, x_2, x_3, x_4]$.

$$\mathcal{L}_1 = \begin{pmatrix} eq_1 & : & x_1 + x_3 + 1 & = & 0 \\ eq_2 & : & x_2 + x_3 + x_4 & = & 0\,. \end{pmatrix}$$

Fix $d = 3$ *(generate all equations of degree 3). According to Theorem 6.2 we can generate* 4 *such equations. The set of all monomials of degree two is*

$$T = \{x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_3x_4\}\,.$$

Following the systematic procedure we multiply eq_1 by $T \setminus T_1 = \{x_2x_3, x_2x_4, x_3x_4\}$ and eq_2 is multiplied by $T \setminus \{T_1 \cup T_2\} = x_3x_4$. As in Example 6.2 we obtain four linearly independent equations with distinct degree three head term.

$$\mathcal{L}_{3_g} = \left(\begin{array}{lcc} x_2x_3eq_1 & : & x_1x_2x_3 = 0 \\ x_2x_4eq_1 & : & x_1x_2x_4 + x_2x_3x_4 + x_2x_4 = 0 \\ x_3x_4eq_1 & : & x_1x_3x_4 = 0 \\ x_3x_4eq_2 & : & x_2x_3x_4 = 0 \end{array} \right)$$

but without generating excessive equations or performing Gaussian elimination.

6.3.3 Equations over the S-box layer

In this section the exact number of degree-d equations which can be generated from several applications of an S-box with b-bit input and c-bit output is determined. The non-linear layer of most block ciphers, including AES, consists of a number of "parallel" applications of one S-box. In the following we consider s applications of such an S-box.

Each S-box is described by a set of multivariate equations over $GF(2)$. Assume the equations are generated according to the procedure in Section 3.7.1 of Chapter 3. Let $h(i)$ denote the number of degree-i monomials in the $b+c$ variables of the S-box which are not a head term of an equation. The constant term is never the head term of an equation so $h(0) = 1$. For most S-boxes one has $h(1) = b + c$, since otherwise there would be linear relations between the input and output of probability one. In general the numbers $h(i)$ for higher values of i are found by simply counting the number of distinct degree-i head terms of the polynomials $f_1, f_2, \ldots, f_m$. Note that for any function mapping b bits to c bits it holds that

$$\sum_{i=0}^{b+c} h(i) = 2^b. \tag{6.2}$$

To see this, note that we find all multivariate equations over a b-bit function by computing the null space of the matrix $\underline{\underline{A}}$ (as describes in Section 3.7.1) for $d = b+c$ one finds exactly $2^{b+c} - 2^b$ deterministic equations.

Example 6.4. *It is well-known [20] that there are 39 quadratic equations over $GF(2)$ for the AES S-box, thus in this case, $h(2) = \binom{16}{2} - 39 = 81$*

Example 6.5. *Consider a four to four-bit S-box with no deterministic linear equations. Since there are 16 inputs and a total of 37 monomials of degree at most two, one can always find at least 21 quadratic equations in the input and output bits as*

described above. Also, one can always find at least 77 *equations of degrees at most three, since there are exactly* $\binom{8}{3} = 56$ *monomials of degree three in eight variables. Thus, if there are* **exactly** *21 quadratic equations, then there are exactly 56 equations of degree three. Note that, in general, if one can generate all degree-d polynomials with distinct head terms, then this is the case also for* $d' > d$*. The maximum set of equations for this four-bit S-box is then defined as the 21 quadratic equations and all degree-d equations for* $d > 2$*. In other words, in this case we have* $h(0) = 1$*,* $h(1) = 8$*,* $h(2) = 7$*,* $h(3) = h(4) = \cdots = h(8) = 0$ *thus* $\sum h(i) = 16$*.*

Next we count the number of degree-d monomials in the variables of s distinct S-boxes which are not divisible by any head term of the s sets of S-box equations. These degree-d monomials are expressed as $v_1 \cdot v_2 \cdot v_3 \cdots v_s$, where v_1 is a monomial from the first S-box which is not head term of any of its equations, v_2 is a monomial of the second S-box which is not head term of any of its equations and so on. Let d_i denote the degree of v_i. Then by computing

$$S_d = \sum_{d=d_1+d_2+\cdots+d_s} \prod_{i=1}^{s} h(d_i) \tag{6.3}$$

we find the exact number of all such degree-d monomials. The sum is taken over all combinations of $d_1, d_2, \ldots, d_s$ which sum to d. Consider the polynomial

$$P(x) = h(0) + h(1)x + h(2)x^2 + \ldots + h(b+c)x^{b+c}. \tag{6.4}$$

It follows that the values of S_d can be found from the coefficients of

$$P(x)^s = S_0 + S_1 x + S_2 x^2 + \ldots + S_{(b+c)s} x^{(b+c)s}. \tag{6.5}$$

Now we can prove the following result.

Theorem 6.3. *Let there be given s applications of a b-bit to c-bit S-box, each described by the maximum set of equations in the* $b + c$ *input and output bits. Assume that within each of these sets,* $h(d_i)$ *monomials of degree* d_i *for* $d_i \in [0; b + c]$ *are not a head term of any equation. Then there exist exactly*

$$N_d = \binom{n}{d} - S_d, \tag{6.6}$$

equations with distinct head terms of degree d in the joint set of $n = (b+c)s$ *variables, where* $S_d = \sum_{d=d_1+d_2+\cdots+d_s} \prod_{i=1}^{s} h(d_i)$*.*

Proof. As there are 2^b solutions to each set of S-box equations, and since the S-boxes have independent inputs, the number of solutions to the complete set of S-box equations is 2^{bs}. It follows that $max = 2^n - 2^{bs}$ in Lemma 6.4. Thus one can generate at most $2^n - 2^{bs}$ equations of degree less than or equal to n. This is exactly what one gets by summing over the expression in (6.6):

$$\sum_{d=0}^{n} N_d \;=\; \sum_{d=0}^{n} \binom{n}{d} - \sum_{d=0}^{n} S_d = 2^n - P(1)^s = 2^n - 2^{bs}. \tag{6.7}$$

□

In addition to the proof we note that we have experimentally verified Theorem 6.3 for various number of S-boxes s and degrees d.

Example 6.6 (Computing N_d)**.** *For a system of four 4-bit S-boxes each described by* 21 *quadratic equations (e.g. the Noekeon [23] S-box)*

$$P(X) = 1 + 8X + 7X^2\,.$$

We have

$$P(X)^4 = (1 + 8X + 7X^2)^4$$

from which we extract $S_5 = 19040$ *as the coefficient of* X^5 *in* $P(X)^4$ *and find that there are*

$$N_5 = \binom{8 \cdot 4}{5} - S_5 = 182336$$

equations with distinct degree-five head terms.

6.3.4 Systematic procedure for S-box layer

The procedure described in Figure 6.2 generates the maximum set of equations of a fixed degree d for a system of n_{Sbox} S-boxes. Each S-box is described by $b + c$ variables thus $n = n_{Sbox} \cdot (b + c)$ is the total number of variables in the system. $S_i^{\tilde{d}}$ is the set of linearly independent equations of degree $\tilde{d}$ that describes S-box S_i over $GF(2)[x_{1+i(b+c)}, \ldots, x_{(i+1)(b+c)}]$. Initially one generates for each S-box S_i the sets of equations

$$S_i^1, S_i^2, S_i^3, \ldots, S_i^d. \tag{6.8}$$

Note that $S_i^1 = \emptyset$ for a non-linear S-box. In the following T^d denotes the set of all monomials $GF(2)[x_1, \ldots, x_n]$ of degree d. $T_i^d \subset T^d$ is the set of monomials of degree d which are divisible by the head term of a least one equation in $\{S_i^1, S_i^2, S_i^3, ..., S_i^d\}$. T_i is the set of monomials divisible by monomials of degree one from S-box S_i. The

Step 1 Initialize $\mathcal{E}_S = \emptyset$.

Step 2 For $i = 0, \ldots, n_{Sbox} - 1$

generate $\{S_i^1, S_i^2, S_i^3, \ldots, S_i^d\}$.

Step 3 For $\tilde{d} = 1, \ldots, d$ over all

$$eq \in S_i^{\tilde{d}}$$

and for all

$$t \in T' = T^{d-\tilde{d}} \setminus \{T_1^{d-\tilde{d}}, \ldots, T_{i-1}^{d-\tilde{d}}, T_i\}$$

compute

$$f = t \cdot eq$$

and update

$$\mathcal{E}_S = \mathcal{E}_S \cup f.$$

Step 4 Return $\mathcal{E}_S$.

Figure 6.2: Equations for S-boxes

point of the procedure given in Figure 6.2 is the same as for the procedure of Figure 6.1, namely to ensure that we generate as many equations, of the chosen degree d, as possible without generating more equations with the same head term.

6.4 Number of equations for some block ciphers

In this section we apply the results of the previous section to determine the number of linearly independent equations one can generate for various ciphers. AES has been criticized for its use of the inverse mapping in $GF(2^8)$ in the construction of the S-box. It is well-known that this leads to the existence of 39 quadratic $GF(2)$ equations over the S-box [20] (see Section 3.8). Thus in total an AES-key can be described as the solution to a system of equations with 1600 linear equations and 7800 quadratic equations in 3200 variables. In the following we compare AES to the variant where the S-box is replaced by an S-box for which no quadratic equations exist. First we describe AES-128 by the methods of this chapter.

6.4.1 AES

As shown in Section 3.8.1 AES-128 is described by a set of 9400 quadratic equations out of which 1600 are linear in 3200 variables over $GF(2)$. We verified (by the matrix method of 3.7.1) for the AES S-box that there are no equations of degree one, 39 equations of degree two, 432 equations of degree three, and 1790 equations of degree four. Note that the number of monomials in sixteen variables of degrees two, three and four are respectively 120, 560 and 1820. Furthermore we confirmed that one can generate $\binom{16}{5}$ equations each with a distinct head term of degree 5, and therefore one gets $\binom{16}{d}$ equations each with a distinct head term of degree d, for $d > 5$. For the AES S-box one gets $h(0) = 1$, $h(1) = 16$, $h(2) = 81$, $h(3) = 128$, $h(4) = 30$, and for $i \geq 5$, $h(i) = 0$. With these values one can apply the result of Theorem 6.3 for the AES S-boxes. The results of this are listed i Table 6.1.

6.4.2 A variant of AES-128

We define a variant of AES-128 simply by replacing the S-box used in AES-128 with a "randomly chosen" eight-bit S-box. Note that this S-box needs to be bijective. All other components of this variant are identical to those of AES-128.
We choose an S-box for which there are no linear equations and no quadratic equations. By using the method of Section 3.7.1 it follows that the probability that there are quadratic equations over a randomly chosen eight-bit S-box is very small. With no linear and quadratic equations it follows that there are

$$(\sum_{i=0}^{3} \binom{16}{i}) - 256 = 441$$

equations of degree three, and thus one gets

$$h(0) = 1, h(1) = 16, h(2) = 120, h(3) = 119,$$

and $h(i) = 0$ for $i \geq 4$.

6.4.3 Comparison of the ciphers

One advantage of our approach compared to that of others, e.g., [20], is that we are able to compare certain algebraic properties of AES to other ciphers rather easily. To do this, we set up a systems of equation for each of the ciphers, then divide the equations into two sets, cf. earlier, and finally compute how many linearly independent equations are obtained by multiplying the equations within each set to a certain degree d.

Table 6.1: The number of linearly independent equations generated from respectively the linear and the S-box layer plus the number of terms for AES-128 and the AES variant.

Degree	# terms	# equations from $L(\cdot)$	# equations from S-box layer	
			AES-128	AES-variant
1	3200	1600	0	0
2	$2^{22.29}$	$2^{21.87}$	7800	0
3	$2^{32.35}$	$2^{32.15}$	$2^{24.57}$	$2^{16.43}$
4	$2^{41.99}$	$2^{41.89}$	$2^{35.21}$	$2^{28.07}$
5	$2^{51.31}$	$2^{51.26}$	$2^{45.26}$	$2^{38.70}$
6	$2^{60.36}$	$2^{60.34}$	$2^{54.90}$	$2^{48.76}$
..	..	..	..	
10	$2^{94.63}$	$2^{94.62}$	$2^{90.72}$	$2^{85.59}$

Table 6.1 lists these numbers for the degrees $d = 1, \ldots, 6$ and 10 together with the total number of degree-d terms for both AES-128 and the variant introduced above. There are several things worth noting in the table. First of all, although not surprisingly, the majority of the degree-d equations are generated from the linear equations. Also, the numbers of equations obtained from the S-boxes in the AES variant are remarkably smaller than for the AES-128.

While the equations within each of the two sets for each cipher are linearly independent, the big question is of course how many linear dependencies one finds when the two set of equations are combined, e.g., as in the approach from Section 6.2. At this point in time we are not able to answer this question, which remains an open problem and a (good) topic for further research. However, we can say something about generic XL-like approaches using our results. Given an initial set of equations in n variables consider an attack for breaking block ciphers, which determines a degree d such that all equations of degree d can be generated. Then one finds the secret key using a multivariate extension of Gaussian elimination. Let $|\mathcal{T}_d|$ be the number of monomials of degree d. Then with n variables one gets $|\mathcal{T}_d| = \binom{n}{d}$. Let $|\mathcal{L}_d|$ be the number of linearly independent equations of degree d one can generate from the linear layer and let $|\mathcal{S}_d|$ be the number of linearly independent equations of degree d one can generate from the S-box layer. Thus for the XL-like approach to succeed it must hold that $|\mathcal{T}_d| \leq |\mathcal{L}_d| + |\mathcal{S}_d|$. For AES-128 this inequality holds for $d \geq 6$. Note that this degree is a minimum, there is nothing in our results that indicate that such an attack will succeed at this degree. It is interesting to note, though, that for the AES variant the inequality holds only for $d \geq 10$. It appears that the AES

variant provides a (much) higher resistance against algebraic attacks. However, in fairness of AES-128 it should be noted that for $d = 6$, one gets $|\mathcal{T}_6| = \binom{3200}{6} \simeq 2^{60}$. Therefore, even under very optimistic assumptions about the success of attacks like the ones we have presented here, with $d = 6$ one would get an attack which is more costly than an exhaustive search for the key and which will require an unrealistically large amount of memory, and as such it is not a threat for the security of AES-128.

6.5 Simulations

We performed a series of algebraic attacks on small(er) ciphers using the iterated XL approach. Consider first an iterated cipher with eight-bit blocks where each round is as follows. First add (modulo 2) a round key to the text, then divide the text into two nibbles each of which are evaluated through a four-bit S-box, and finally apply a linear mapping on the eight-bit block. After the last application of the linear mapping a final round-key is added to produce the ciphertext. The round keys are derived from the user-selected eight-bit key added to a round-dependent constant, the S-box is derived from the inverse mapping in $GF(2^4)$, and the linear mapping is a randomly chosen eight to eight bit invertible matrix. The linear mapping and the set of round key constants were chosen at random in each test, the S-box was the same in all tests. Each S-box can be described by 21 quadratic equations over $GF(2)$. In the following let n be the number of variables in the attack. We picked a randomly chosen plaintext and a randomly chosen key. The resulting ciphertext and the plaintext were used in setting up the linear equations of the cipher. In the tests where the linear equations and/or the S-box equations were multiplied to higher degrees, it was checked and confirmed that the numbers obtained were identical to those found from Theorems 6.2 and 6.3.

20 tests for a 3-round cipher with 6 S-boxes :
Here $n = 48$, so there is a maximum of 17,296 degree-3 monomials, 1128 degree-2 monomials, and 48 degree-1 monomials in the system. The basic attack using $d = 2$ never succeeded. Initially there are 978 quadratic equations and 24 linear equations, while there are 1128 monomials of degree two. By multiplication we obtain 15,272 equations and 5,376 equations of degree three from the linear equations respectively the S-box equations. In total 20,648 cubic equations, while there are 17,296 degree-three monomials. In 20 tests, the attack identified all values of the key which encrypt the plaintext to the ciphertext.

10 tests for a 4-round cipher with 8 S-boxes :
Here $n = 64$, so there is a maximum of 41,664 degree-3 monomials, 2016 degree-2 monomials, and 64 degree-1 monomials in the system. The basic attack using $d = 2$

never succeeded. By multiplication we found 36,704 degree-two equations from the linear equations and 9,856 degree-three equations from the S-boxes. In six of the ten cases, the attack identified all values of the key which encrypt the plaintext to the ciphertext.

20 tests for a 2-round AES-like cipher with 6 S-boxes :
In these tests the cipher is the same as above with the (important) exception that the round keys were generated from the input key in an AES-like fashion, as proposed in [15]. There are four S-box applications in the encryption routine and an additional two S-boxes in the key-schedule. The equations of this mini-AES cipher were set up exactly analogue to the way the equations are set up for the AES, cf. Section 3.8. We have $n = 48$. The number of linear equations and S-box equations obtained is the same as in the above test on the 3-round cipher. All 20 tests succeeded at degree three, the complexity of these tests were very similar to the complexity of the above test for the 3-round cipher.

All the above tests were implemented in C++ using the NTL library [58]. All tests were run on a modern laptop, except those for the 4-round cipher. These were run on a SunFire V440 workstation with 8GB RAM.

6.5.1 Probabilistic equations

Consider the tests on the 4-round cipher above. In four of ten cases the solution was not found at degree $d = 3$. One possible way to try to find the solution is to run the algorithm for $d = 4$. Another way which we explore in Chapter 8 is to generate some additional probabilistic equations. A very simply way is to choose the value of a degree-d monomial (such that a new linearly independent degree-d equation is obtained). E.g., consider the 4-round cipher from the test above. For each of the four failed cases we fed the algorithm with the same initial equations as earlier plus one additional degree-two equation consisting of one degree-two monomial, but such that the equation was true for the correct value of the key. Perhaps surprisingly this additional equation was sufficient for the attack to succeed, and in all four cases the correct value of the secret key was identified. Note that in an actual attack one would guess the value of this equation, run the attack, and if unsuccessful, add the constant '1' to the extra equation and run the attack again. Clearly such an approach can be extended to guess the values of several degree-d monomials and/or equations.

6.6 Discussion

A big issue regarding the XL and similar attacks is estimating the complexity. In [18] the degree d at which the XL is expected to succeed for a system of quadratic equations $f_1, \ldots, f_m$ in n variables is estimated by assuming that almost all generated equations are linearly independent. If this is the case the attack will succeed when there are as many equations as monomials, that is, when

$$m \sum_{i=0}^{d-2} \binom{n}{i} \approx \sum_{i=0}^{d} \binom{n}{i}.$$

For $n >> d$ the approximation $d \approx \frac{n}{\sqrt{m}}$ is applied in [18]. This means that for the system of 8000 quadratic equations established in [20] the degree d is estimated to $d = \frac{1600}{\sqrt{8000}} = 18$. However, the big issue in this estimate is whether the assumption of linearly independence of the involved equations is correct and/or reasonable.
Proposition 6 of [26] provides a lower bound on the degree d given a system of quadratic polynomials. According to this proposition d is at least 20 for our example. As shown in this work the key is also be described by a set of equations out of which 1600 are linear. Proposition 6 of [26] does not seem to be applicable in this case and the complexity of finding a solution remains open.
In this chapter we developed methods for counting the number of linearly independent equations generated from a set of linear equations and a set of S-box equations respectively. We found for AES-128 that for $d = 6$ the sum of these two numbers exceeds the total number of terms. However, it is unclear exactly how many linear dependencies there are among the two sets and one cannot claim that the attack described in this chapter will work at this degree on AES-128. To obtain better estimates of the complexities of XL-similar attacks on block ciphers more knowledge on the interaction of these sets of equations is needed. We remind the reader that so far no real-life block cipher has been broken by algebraic attacks, not even in theory. However, we feel that our approach provides a good basis for further progress in the area.

Chapter 7

Small Scale Variants of AES

When a new cipher is proposed it is common practice to examine its resiliency towards known methods of cryptanalysis by analyzing down scaled versions of the cipher. In particular AES is, for obvious reasons, a cherished target for cryptanalysis.

In [15] a family of small scale variants of AES is proposed to create common ground for algebraic analysis of AES. The ciphers are defined for 4-bit and 8-bit S-boxes, but in this thesis we only consider the 4-bit S-box version, which we name SmallAES(n_r, r, c). n_r is the number of rounds of encryption and r, c are the number of rows and columns of the state. Each sub-function of SmallAES(n_r, r, c) is defined as a down scaled version of the corresponding sub-function of AES.

As described in Section 3.6 AES can be described by algebraic equations in the polynomial ring over $GF(2)$ or $GF(2^8)$ (by embedding AES in BES). In [15] Cid, Murphy and Robshaw primarily explore the latter by deriving a description of SmallAES(n_r, r, c) over $GF(2^4)$ and applying Magma's implementation of F4 for Gröbner bases computation. For details on this approach, we refer the reader to [15].

Our work focuses on the description of SmallAES(n_r, r, c) over $GF(2)$. As in [15] we apply Magma's implementation of F4 for the Gröbner bases computation. The timing results are measured in seconds taken by the CPU. The idea is to study the growth of the Gröbner basis computation time when increasing respectively the block size and the number of rounds. We successfully accomplish this for SmallAES$(n_r, 1, 1)$ and SmallAES$(n_r, 2, 1)$ from one to eleven rounds of encryption. This improves the results of [15] where the simulations on SmallAES$(n_r, 2, 1)$ can handle only one to four rounds of encryption. Our simulations on SmallAES$(n_r, 2, 2)$ can handle only up to two rounds of encryption. Though this is an improvement compared to the results presented in [15] where they are only successful in computing a Gröbner basis for the one round version of SmallAES$(n_r, 2, 2)$, it is not

Table 7.1: The S-box of SmallAES(r, c, n) and SmallAES-2(r, c, n) (in hexadecimal notation).

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[x]	6	b	5	4	2	e	7	a	9	d	f	c	3	1	0	8

very impressive and does not tell us much about the behavior of the attack when increasing the number of rounds.
To take the attack a step further we propose in Chapter 8 a technique which applies probabilistic equations. We discovered that a slightly modified version of SmallAES(n_r, r, c) is interesting regarding this technique. We name this cipher SmallAES-2(n_r, r, c) and in Section 7.2 we describe the difference between this cipher and SmallAES(n_r, r, c).

7.1 SmallAES(n_r, r, c)

SmallAES(n_r, r, c) is an n_r-round cipher which has a state size $r \cdot c \cdot 4$, where r is the number of rows and c is the number of columns of the state. The word size is decreased from 8 to 4 bits compared to AES. This means that a number of operations are computed over $GF(2^4)$ instead of $GF(2^8)$. The field $GF(2^4)$ is established as the extension field $GF(2)[x]/m(x)$ where

$$m(x) = x^4 + x + 1$$

is irreducible in $GF(2)[x]$.

As for the full scale AES the round function is composed of four sub-functions: SubBytes, MixColumns, ShiftRows and AddRoundKey, which we briefly describe in the following.

7.1.1 The substitution layer

The SubBytes function divides the state into 4-bit words which are each substituted using a 4-bit S-box. The S-box, given in Table 7.1, is composed of inversion over $GF(2^4)$ followed by a $GF(2)$-linear map and finally by addition of the constant 6_x. The S-box is described by 21 polynomial equations over $GF(2)$:

$$f_1 = 0, \ldots, f_{21} = 0$$

in the binary variables

$$(x_1, x_2, x_3, x_4), (x_5, x_6, x_7, x_8)$$

of respectively the input and the output of the S-box. The polynomials $f_1, \ldots, f_{21}$ generate the polynomial ideal $I = \langle f_1, \ldots, f_{21} \rangle$, where

$$I \subseteq GF(2)[x_1, \ldots, x_8].$$

In Figure 7.1 we list the reduced Gröbner basis of

$$I \subseteq GF(2)[x_1, \ldots, x_8]/\langle x_1^2 + x_1, \ldots, x_8^2 + x_8 \rangle$$

with respect to grevlex order

7.1.2 The diffusion layer

The diffusion layer is composed of the functions MixColumns and ShiftRows. ShiftRows performs words-based left rotations with different off-sets.
For $0 \leq i \leq r-1$, row $i+1$ is rotated i places to the left. In our simulations we apply the state sizes:

$$(r = 1, c = 1), (r = 2, c = 1), (r = 2, c = 2).$$

When the state has only one column (c=1), ShiftRows is defined as the identity map. In this case, the MixColumns function alone creates the diffusion in the round function.
MixColumns is defined as multiplication of the state by the invertible matrix $\underline{\underline{A}}$ over $GF(2^4)$.
When the state consists of only one row (r=1), MixColumns is given by the identity map

$$\underline{\underline{A}} = \begin{pmatrix} 1 \end{pmatrix}.$$

In this case there is no diffusion in the outputs of the S-boxes of the round function. This means that, for $c > 1$, one can divide the cipher into smaller subsystems which can be analyzed independently. Since such ciphers are not interesting we omit the state size $r = 1, c = 2$ in our simulation.
For r=2, $\underline{\underline{A}}$ is given by

$$\underline{\underline{A}} = \begin{pmatrix} (\theta + 1) & \theta \\ \theta & (\theta + 1) \end{pmatrix}.$$

where θ is a root of $m(x) = x^4 + x + 1$.
Figures 7.2 and 7.3 contain the algebraic equations of the linear layer for respectively the state where $(r = 2, c = 1)$ and $(r = 2, c = 2)$.

$$
\begin{aligned}
x_6x_7x_8 + x_2x_8 + x_5x_8 + x_8,\\
x_1x_2 + x_5x_8 + x_6x_8 + x_2 + x_4 + x_5 + x_8,\\
x_1x_3 + x_2x_8 + x_6x_8 + x_7x_8 + x_2 + x_3 + x_6 + x_7,\\
x_2x_3 + x_6x_7 + x_2x_8 + x_3x_8 + x_4x_8 + x_6x_8 + x_7x_8 + x_2 + x_7 + x_8,\\
x_1x_4 + x_2x_8 + x_5x_8 + x_1 + x_2 + x_4 + x_5 + x_8,\\
x_2x_4 + x_6x_7 + x_2x_8 + x_3x_8 + x_4x_8 + x_1 + x_2 + x_4 + x_5 + x_6 + x_7 + 1,\\
x_3x_4 + x_2x_8 + x_3x_8 + x_4x_8 + x_6x_8 + x_1 + x_3 + x_5 + x_6 + x_7,\\
x_1x_5 + x_2x_8 + x_4x_8 + x_1 + x_2 + x_4 + x_5 + x_8,\\
x_2x_5 + x_6x_7 + x_3x_8 + x_6x_8 + x_2 + x_3 + x_4 + x_5 + x_8 + 1,\\
x_3x_5 + x_6x_7 + x_4x_8 + x_5x_8 + x_7x_8 + x_2 + x_4 + x_7 + x_8,\\
x_4x_5 + x_6x_7 + x_2x_8 + x_3x_8 + x_6x_8 + x_1 + x_2 + x_5 + x_6 + x_7 + x_8 + 1,\\
x_1x_6 + x_4x_8 + x_8,\\
x_2x_6 + x_3x_8 + x_5x_8 + x_6x_8 + x_7x_8 + x_1 + x_4 + x_5 + x_6 + 1,\\
x_3x_6 + x_6x_7 + x_3x_8 + x_5x_8 + x_7x_8 + x_1 + x_4 + x_5 + 1,\\
x_4x_6 + x_6x_7 + x_2x_8 + x_5x_8 + x_7x_8 + x_4 + x_6 + x_7 + x_8 + 1,\\
x_5x_6 + x_6x_7 + x_3x_8 + x_4x_8 + x_5x_8 + x_6x_8 + x_1 + x_5 + x_7 + x_8,\\
x_1x_7 + x_2x_8 + x_5x_8 + x_6x_8 + x_7x_8 + x_2 + x_4 + x_5 + x_8,\\
x_2x_7 + x_6x_7 + x_2x_8 + x_4x_8 + x_6x_8 + x_3 + 1,\\
x_3x_7 + x_2x_8 + x_3x_8 + x_4x_8 + x_5x_8 + x_6x_8 + x_7x_8 + x_3 + x_7 + x_8 + 1,\\
x_4x_7 + x_2x_8 + x_4x_8 + x_5x_8 + x_6x_8 + x_7x_8,\\
x_5x_7 + x_6x_7 + x_2x_8 + x_6x_8 + x_7x_8 + x_2 + x_3 + x_4 + x_5 + x_8 + 1,\\
x_1x_8 + x_3x_8 + x_4x_8 + x_5x_8 + x_6x_8 + x_7x_8 + x_1 + x_3 + x_4 + x_5 + x_6 + x_7.
\end{aligned}
$$

Figure 7.1: Gröbner basis for the polynomial ideal of the S-box of SmallAES(n_r, r, c) and SmallAES-2(n_r, r, c).

$$
\begin{aligned}
y_1 &= x_4 + x_5 + x_8 \\
y_2 &= x_1 + x_4 + x_5 + x_6 + x_8 \\
y_3 &= x_2 + x_6 + x_7 \\
y_4 &= x_3 + x_7 + x_8 \\
y_5 &= x_1 + x_4 + x_8 \\
y_6 &= x_1 + x_2 + x_4 + x_5 + x_8 \\
y_7 &= x_2 + x_3 + x_6 \\
y_8 &= x_3 + x_4 + x_7
\end{aligned}
$$

Figure 7.2: Equations of the diffusion layer of one round of SmallAES($n_r, 2, 1$). The output bit variables $y_1, \ldots, y_8$ are described by linear expressions in the input bit variables $x_1, \ldots, x_8$.

7.1.3 Key application

The function AddRoundKey exclusive-ores a $4rc$-bit round key to the $4rc$-bit state. The round keys are derived by the key schedule.

7.1.4 Key schedule

The key schedule of SmallAES(n_r, r, c) is defined such that each row of the state invokes an S-box application in the key schedule. The cipher key is loaded directly into the first round key. For the state size of four bits ($r = 1, c = 1$) the first round key is

$$K^{(0)} = w_0$$

while the following round keys are derived as

$$K^{(i)} = w_i = S(w_{i-1}) \oplus R_{i-1},$$

for $i = 1, \ldots, n_r$, where R_i is a round constant and $S(\cdot)$ is the same S-box as the function SubBytes applies. For the state size of eight bits ($r = 2, c = 1$) the first round key is

$$K^{(0)} = (w_1, w_0)$$

and the following round keys

$$K^{(i)} = (w_{1+2i}, w_{2i})$$

$$
\begin{aligned}
y_1 &= x_5 + x_8 + x_{12} \\
y_2 &= x_5 + x_6 + x_8 + x_9 + x_{12} \\
y_3 &= x_6 + x_7 + x_{10} \\
y_4 &= x_7 + x_8 + x_{11} \\
y_5 &= x_1 \\
y_6 &= x_1 + x_2 + x_{13} \\
y_7 &= x_2 + x_3 + x_{14} \\
y_8 &= x_3 + x_4 + x_{15} \\
y_9 &= x_8 + x_9 + x_{12} \\
y_{10} &= x_5 + x_8 + x_9 + x_{10} + x_{12} \\
y_{11} &= x_6 + x_{10} + x_{11} \\
y_{12} &= x_7 + x_{11} + x_{12} \\
y_{13} &= x_4 + x_{13} + x_{16} \\
y_{14} &= x_1 + x_4 + x_{13} + x_{14} + x_{16} \\
y_{15} &= x_2 + x_{14} + x_{15} \\
y_{16} &= x_3 + x_{15} + x_{16}
\end{aligned}
$$

Figure 7.3: Equations of the diffusion layer of one round of SmallAES$(n_r, 2, 2)$. The output bit variables $y_1, \ldots, y_{16}$ are described by linear expressions in the input bit variables $x_1, \ldots, x_{16}$.

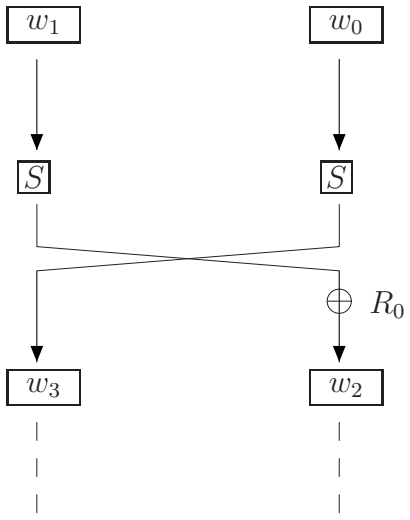


Figure 7.4: The Key schedule of SmallAES($n_r, 2, 1$)

for $i = 1, \ldots, n_r$ are derived using the key schedule shown in Figure 7.4.
Finally, the first round key for the state size of sixteen bits ($r = 2, c = 2$) is

$$K^{(0)} = (w_3, w_2, w_1, w_0),$$

and the following round keys $K^{(i)}$

$$K^{(i)} = (w_{3+4i}, w_{2+4i}, w_{1+4i}, w_{4i}),$$

for $i = 1 \ldots n_r$, are derived by the key schedule shown in Figure 7.5. The key schedule applies the same S-box as the encryption function. Like in AES the key schedule exclusive-ores a round constant R_i upon the S-box application. The number of S-boxes applied in the key schedule is given by $n_r \cdot r$.

7.2 SmallAES-2(n_r, r, c)

The cipher SmallAES-2(n_r, r, c) is very similar to SmallAES(n_r, r, c). In fact, the only difference between the two ciphers is the key schedule. We reduce the number of S-boxes in the key schedule because S-boxes in the key schedule complicate the Gröbner basis computations (we have tested this and Magma is always faster in computing the Gröbner basis when we replace the non-linear key schedule by a linear

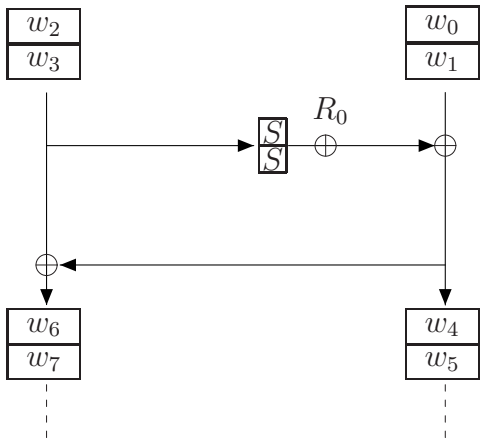


Figure 7.5: The Key schedule of SmallAES$(n_r, 2, 2)$

key schedule). In [15] the number of S-boxes applied in the key schedule equals $n_r \cdot r$ (as mentioned). This means that both SmallAES$(n_r, 2, 1)$ and SmallAES$(n_r, 2, 2)$ applies two S-boxes to derive each of the round keys $K^{(i)}$, for $i = 1, \ldots, n_r$. Thus the ratio in the number of 4-bit words of the round keys to the number of S-box applications is respectively $1 : 1$ and $2 : 1$. The state of AES-128 contains sixteen 8-bit words (so does each round key) and the key schedule applies four 8-bit S-boxes to derive each key $K^{(i)}$, for $i = 1, \ldots, 9$, thus the ratio here is $4 : 1$. For this reason we think it is fair to define a slightly less complex key schedule. The key schedule of SmallAES-2(n_r, r, c) applies when possible the ratio $4 : 1$ in the number of 4-bits words of the state to the number of S-box applications in the round key derivation of the keys $K^{(i)}$, for $i = 1, \ldots, n_r$. E.g for SmallAES-2$(n_r, 2, 2)$ the key schedule applies one S-box to derive each round key (except for the first). Figure 7.6 shows the key schedule of SmallAES$(n_r, 2, 2)$. In our opinion the key schedule of SmallAES-2$(n_r, 2, 2)$ retains more properties of the AES key schedule than that of SmallAES$(n_r, 2, 2)$. As mentioned the former has some interesting properties regarding the technique we present in Chapter 8.

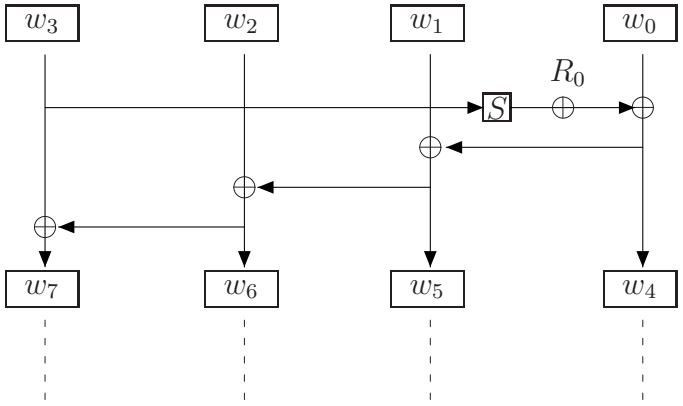


Figure 7.6: The Key Schedule of SmallAES-2$(n_r, 2, 2)$

7.3 Simulations

We have performed a number of simulations on SmallAES(n_r, r, c) and SmallAES-2(n_r, r, c). In the following we list the mean values and variance of the tests on respectively 4, 8 and 16-bit blocks ciphers of up to eleven rounds of encryption. The tests presented in this chapter are performed for keys and plaintexts chosen at random. When nothing else is stated, the ciphertext is computed by encrypting the chosen plaintext under the chosen key. Whenever we establish a system of polynomial equations on basis of such a text pair, we know the system has at least one solution, namely the one that provides us the key which was applied for encryption.

Our test results on the 4-bit block cipher are listed in Table 7.2. For this block size the SmallAES(n_r, r, c) and SmallAES-2(n_r, r, c) are defined to be exactly the same. The round function and the key schedule applies one S-box per round and since the text block has the size of just one S-box input/output the round function has no diffusion layer. Each S-box is described by 21 polynomials equations of degree two. Thus in total the cipher is described by

$$\#\text{equations} = n_r \cdot 2 \cdot 21, \quad \#\text{variables} = n_r \cdot 2 \cdot 4$$

Our results are visualized in Figure 7.7.

A more interesting cipher is SmallAES$(n_r, 2, 1)$. Because the cipher has only one column, MixColumns alone provides the diffusion of the round function. Both the round function and the key schedule applies two S-boxes per round. The cipher is

Table 7.2: Tests over $GF(2)$ on SmallAES$(n_r, 1, 1)$ and SmallAES-2$(n_r, 1, 1)$

n_r	# tests	# variables	# equations	average time	st. deviation
2	100	16	84	$3.0 \cdot 10^{-2}$	$6.7 \cdot 10^{-3}$
3	100	24	126	$1.5 \cdot 10^{-1}$	$8.1 \cdot 10^{-3}$
4	100	32	168	$3.9 \cdot 10^{-1}$	$1.3 \cdot 10^{-2}$
5	100	40	210	$8.5 \cdot 10^{-1}$	$9.0 \cdot 10^{-3}$
6	100	48	252	1.8	$4.0 \cdot 10^{-2}$
7	100	56	294	3.4	$9.8 \cdot 10^{-2}$
8	100	64	336	6.0	$9.8 \cdot 10^{-2}$
9	100	72	378	10.2	$2.6 \cdot 10^{-1}$
10	100	80	420	15.6	$2.9 \cdot 10^{-1}$
11	100	88	462	23.1	$4.4 \cdot 10^{-1}$

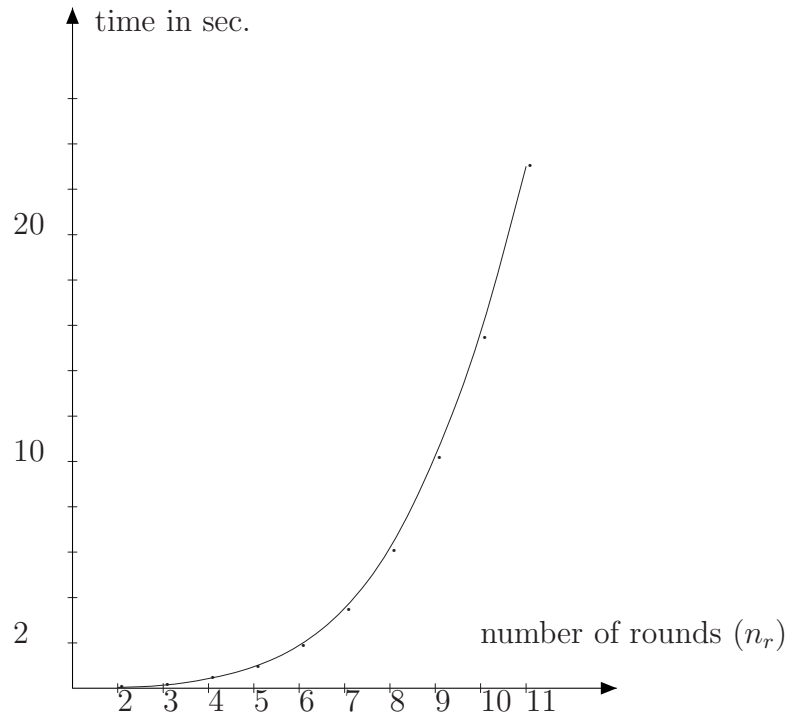


Figure 7.7: Gröbner basis computation times for SmallAES$(n_r, 1, 1)$ and SmallAES-2$(n_r, 1, 1)$

described by

$$\#\text{equations} = n_r \cdot 4 \cdot 21, \quad \#\text{variables} = n_r \cdot 4 \cdot 4$$

over $GF(2)$. In Table 7.3 we list our timing results (in seconds) on SmallAES$(n_r, 2, 1)$. N/A denotes insufficient memory to complete the computation. Our simulations can handle up to eleven rounds of encryption, while the simulations given in [15] can handle only up to four rounds of encryption. We quote the simulations of [15] in Table 7.4. The reader should be advised that the number of tests and the standard deviation are not given in [15].
The authors of [15] note that from their results on SmallAES$(n_r, 1, 1)$ it seems to be faster to compute a Gröbner basis for the $GF(2)$-description of SmallAES$(n_r, 1, 1)$. However, they present no results on Gröbner bases computations over $GF(2)$ for SmallAES$(n_r, 2, 1)$ or SmallAES$(n_r, 2, 2)$. Our results on SmallAES$(n_r, 2, 1)$ are shown in Figure 7.8.

Table 7.3: Tests over $GF(2)$ on SmallAES$(n_r, 2, 1)$

n_r	# tests	# variables	# equations	average time	st. deviation
2	256	32	168	1.8	$3.3 \cdot 10^{-2}$
3	256	48	252	9.5	$6.9 \cdot 10^{-2}$
4	174	64	336	47.7	3.1
5	10	80	420	207.9	11.7
6	10	96	504	755.7	44.5
7	10	112	588	1915.0	252.6
8	10	128	672	3672.2	365.9
9	10	144	756	6702.3	416.2
10	10	160	840	10327.3	358.4
11	10	176	924	15716.0	727.7
12	10	192	1008	N/A	-

In [15] both F4 and Buchberger fail to compute the Gröbner basis for the two round version of SmallAES$(n_r, 2, 2)$. In our simulations on SmallAES$(n_r, 2, 2)$ we can handle only two rounds because of the memory consumption. The average computation time for SmallAES$(2, 2, 2)$ is 1715.72 seconds when the plaintext and the key is chosen at random. For SmallAES-2$(2, 2, 2)$ the computation time is better, however we still fail to compute the Gröbner basis on SmallAES-2$(3, 2, 2)$. The results are listed in Table 7.5.
It would be of great interest to obtain some results on SmallAES$(n_r, 2, 2)$ or SmallAES-2$(n_r, 2, 2)$ over more rounds of encryption because they share more properties with AES than the smaller versions. In particular the diffusion layer is more

Table 7.4: Tests over $GF(2^4)$ on SmallAES$(n_r, 2, 1)$ from [15]

n_r	# tests	# variables	# equations	average time	st. deviation
2	-	72	144	24.55 (F4)	-
3	-	104	208	519.92 (F4)	-
4	-	136	272	28999 (BA)	-
5	-	168	332	N/A	-

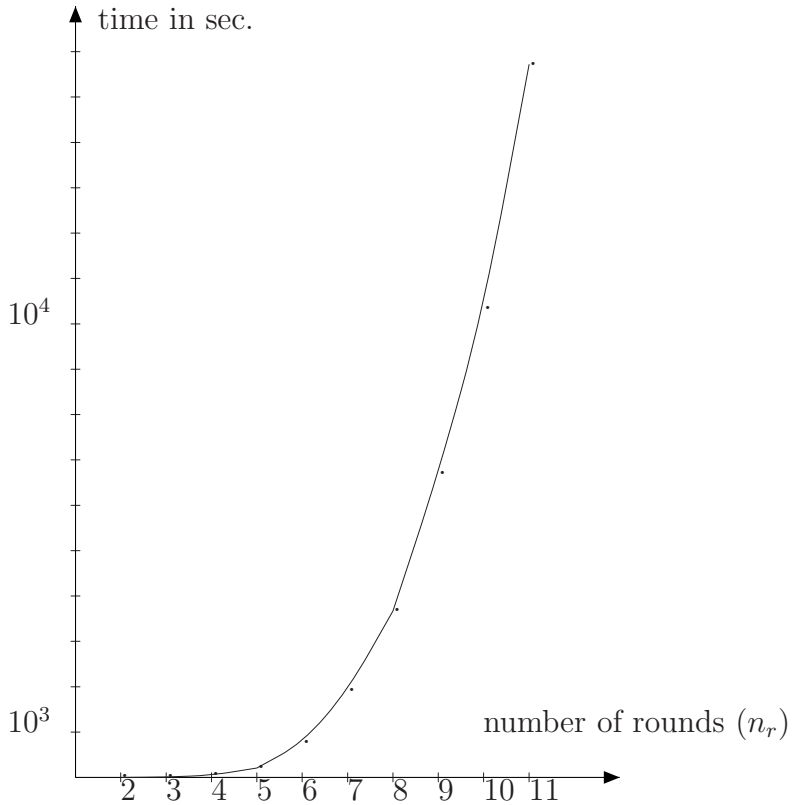


Figure 7.8: Gröbner basis computation times for SmallAES$(n_r, 2, 1)$

realistic because MixColumns and ShiftRows together create the diffusion of the round function. An important observation from our simulations is, that even though the $GF(2)$-descriptions of respectively SmallAES$(3, 2, 1)$ and SmallAES$(2, 2, 2)$ contain the same number of equations and variables, the difficulty of finding the solution for the two by Gröbner bases computation are in contrast. This means that the diffusion layer (which is the vital difference between SmallAES$(n_r, 2, 1)$ and SmallAES$(n_r, 2, 2)$) seems to have great impact on the difficulty of the problem.

Table 7.5: Tests over $GF(2)$ on SmallAES$(n_r, 2, 2)$ and SmallAES-2$(n_r, 2, 2)$

cipher	n_r	# tests	# variables	# equations	average time
SmallAES	2	10	48	252	1715.72
SmallAES-2	2	10	40	210	891
SmallAES	3	10	72	378	N/A
SmallAES-2	3	10	60	315	N/A

From all simulations on SmallAES(n_r, r, c) and SmallAES-2(n_r, r, c), we observe that it does not seem to effect the Gröbner bases computation time whether the polynomial equations have one or more solutions.
As an additional test on SmallAES$(2, 2, 2)$ we chose both the plaintext and the ciphertext at random and measured the Gröbner bases computation time. In half of the tests, Magma outputs the Gröbner basis $G = \{1\}$ i.e. the polynomial equations have no common solution. This phenomena occurs when no key encrypts the specified plaintext to the specified ciphertext. The average computation time of all tests is 1683.22 which is only a marginal different from the 1715.72 seconds listed in Table 7.5. The tests show no relation between the computation time and whether the equations are solvable or not. This result is surprising compared to the results of [1], in which the authors observe the opposite and exploit it in an attack that combines differential and algebraic cryptanalysis.
To conclude this chapter we note that it is not clear whether the advantage of working over $GF(2)$ compared to $GF(2^4)$ is partly because Magma is optimized to work over this field. However, our timing results obtained from simulations over $GF(2)$ are clearly better than those over $GF(2^4)$ presented in [15]. To our knowledge the results presented in this chapter are so far the best algebraic results on small scale variants of AES.

Chapter 8

Probabilistic Equations

Linear and the differential cryptanalysis are probabilistic approaches that often require a large amount of text to cryptanalyze block ciphers. Algebraic cryptanalysis requires in principle only a few known texts to obtain a set of polynomial equations, for which the solution yields the secret key. However, in practice it appears to be very difficult to solve the equations for realistic block ciphers. This makes one wonder whether it is possible to improve the complexity of an algebraic attack by converting it into an approach that benefits from more than just one or two known texts. One idea is to combine algebraic and differential cryptanalysis, this is discussed in [1]. In this chapter we introduce the idea of applying probabilistic equations in algebraic attacks on block ciphers. As described in Section 3.6, it was noted in [20] that for the AES S-box there exists one probabilistic quadratic equations over $GF(2)$ which is true with probability $\frac{255}{256}$. Applying probabilistic equations seems to be a natural extension of the algebraic attacks, and certainly it is important to examine the existence of probabilistic equations and the possibilities of their application. In [11] the idea is explored on stream ciphers. To our knowledge our results are the first of their kind on block ciphers.

There are a couple of things (probably more) that need to be examined. First of all, how to find high-probability equations, and secondly, given a set of probabilistic equations, how to apply them in an algebraic attack? For some ciphers we can derive both deterministic and probabilistic equations. In this case one may ask what is the impact of applying the probabilistic equations and how does it compare to the *deterministic* algebraic approach? This chapter aims to answer some of these questions. For the latter part we outline a number simulations on SmallAES-2(n_r, r, c) (defined in Chapter 7) and compare them to the results of the deterministic approach. Our simulations show that the impact of the high-probability equations is not impressive, however we also discover that a simple and straightforward guessing technique does

improve the computation time. We find that for SmallAES-2(2, 2, 2), it is possible to exploit the structure of the key schedule in this approach. In some cases we obtain a reduction of a factor of 27 in the computation time compared to the deterministic approach. However, when we add an extra round of encryption the method seems to require a notable amount of extra probabilistic equations.
The first part of this chapter concerns how to obtain and apply probabilistic equations in an algebraic attack. We present two ways of finding probabilistic equations for S-boxes, the *matrix method* and the *product method*. The matrix method is suitable for small S-boxes where it is possible to search exhaustively for the equations. The product method is inspired by an observation on a DES S-box where a deterministic quadratic equation arises as the product of two linear approximations of the S-box. The method utilizes linear approximations of the S-boxes to obtain high-probability quadratic equations. This method suits *large* S-boxes in particular for which it is not feasible to perform the exhaustive search for probabilistic equations.

8.1 Applying probabilistic equations

Given a cipher described by a combined set of probabilistic and deterministic equations one idea is simply to apply Buchberger's algorithm or F4 (as described in Chapter 4). If we find that the system has no solution, we apply another set of probabilistic equations and repeat the computation. A natural approach would be a *maximum likelihood approach* where we start out by applying the set of equations which has the highest probability. If this does not provide the solution the computation is repeated, but this time with the set of equations having the next highest probability etc. When we obtain a few solutions, and hereby some key suggestions e.g. K', we test its consistency by encrypting a plaintext P (different from the one applied for the Gröbner basis computation, and for which the ciphertext C is known) under the key K'

$$C' = e_{K'}(P).$$

If $C' = C$ we probably found the right key which can be double checked by applying another known text pair. If $C' \neq C$ for all key suggestion non of them are the right key and we repeat the Gröbner bases computation for the set of probabilistic equations with the next highest probability. We assume that each set of equations yields a limited set of solutions for a fixed text pair. The probability of success, in i iterations of the attack is given by

$$Pr(i) = p_1 + (1 - p_1)p_2 + (1 - p_1)(1 - p_2)p_3 + \ldots + (1 - p_1) \cdots (1 - p_{i-1})p_i$$

where p_i is the probability that the i'th most likely set of equations is true for a key and plaintext chosen at random.
The following sections discuss how to find the probabilistic equations.

8.2 The matrix method

For non-linear functions like the DES S-boxes which map six bits to four bits, or the Serpent, Noekeon and Present S-boxes that map four bits to four bits, it is not difficult to find probabilistic equations. The matrix method offers a simply way to obtain multivariate probabilistic equations of degree less than or equal to d.
Let

$$f : \{0,1\}^m \to \{0,1\}^n$$

be a non-linear function. Arrange a matrix $\underline{\underline{T_d}}$ of 2^m rows and $\sum_{i=0}^{d} \binom{n+m}{d}$ columns. Each row corresponds to an input of f, and each column holds the values of a monomial of degree less than or equal to d, in the input and output bit variables of f. As described in Section 3.6 all deterministic equations of degree d are obtained by computing the null space of $\underline{\underline{T_d}}$.
If

$$\sum_{i=0}^{d} \binom{n+m}{d} > 2^m$$

thus the null space of $\underline{\underline{T_d}}$ is spanned by at least

$$e = \sum_{i=0}^{d} \binom{n+m}{d} - 2^m$$

vectors, i.e. we obtain at least this many linearly independent equations of degree d. To obtain additional probabilistic equations we remove t rows of $\underline{\underline{T_d}}$ before computing the null space.
If we remove one row of $\underline{\underline{T_d}}$ and $\sum_{i=0}^{d} \binom{n+m}{d} > 2^m - 1$, we obtain at least one equation which is true with probability at least $\frac{2^m-1}{2^m}$.
If

$$\sum_{i=0}^{d} \binom{n+m}{d} > 2^m - t,$$

and we remove t rows from $\underline{\underline{T_d}}$ we obtain at least t additional equations which are simultaneously true with probability at least $\frac{2^m-t}{2^m}$.

Example 8.1. *For a random four bit S-box ($n = m = 4$) there are at least*

$$|\mathcal{E}_d| = \binom{8}{2} + \binom{8}{1} + \binom{8}{0} - 2^4 = 21$$

equations of degree less than or equal to two. If there is exactly 21 *deterministic equations and we remove one row of $\underline{\underline{T_2}}$ we get an additional equation of degree at most two, which is true with probability at least $\frac{15}{16}$.*

Example 8.2. *For a random six to four bit S-box ($m = 6, n = 4$) there are not necessarily any deterministic equations of degree less than or equal to two since*

$$\binom{10}{2} + \binom{10}{1} + \binom{10}{0} = 56 < 2^6.$$

If we remove 9 *rows we know with certainty that there is at least one equation of degree at most two which is true with probability $\frac{55}{64}$.*

There are

$$\binom{2^m}{t}$$

ways to remove t rows of $\underline{\underline{T_d}}$. For example for the 8-bit AES S-box this number is big even when t is relatively small.

Example 8.3. *Consider the AES S-box where $m = n = 8$. To find quadratic probabilistic equations we establish $\underline{\underline{T_2}}$, which has dimensions* 256×137. *To search for equations with probability $\frac{255}{256}$ we remove one row from $\underline{\underline{T_2}}$. This can be done in* 256 *ways, which leads us to find only one such equation, namely the one first noted in [20], and which is obtained by removing the row corresponding to the input* 0. *To search for equations which are true with probability $\frac{254}{256}$ we remove two rows of $\underline{\underline{T_2}}$, there are $\binom{2^8}{2} = 32640$ ways to do this. We tried all of these but obtained no additional equations. Considering alone the dimensions of $\underline{\underline{T_2}}$, we do not expect that there are deterministic or high-probability quadratic equations. However, as noted in [20] there are 40 quadratic equations over $GF(2)$ from of which 39 are deterministic and one is true with probability $\frac{255}{256}$.*

8.3 The product method

In Chapter 3 we described how linear cryptanalysis combines linear probabilistic equations in the input/output variables of the S-boxes to linear expressions over

several rounds. The product method applies these equations to obtain equations of higher degree with higher probabilities.
Consider a non-linear function

$$f : \{0,1\}^m \to \{0,1\}^n.$$

Let $(x_1, \ldots, x_{n+m})$ be the input and output bits of f. Given a linear equation

$$a_0 + \sum_{i=1}^{n+m} a_i x_i = 0, \quad a_i \in GF(2) \quad \text{for} \quad i = 0, 1, \ldots$$

that holds for $2^m - k$ of the 2^m inputs. We search for a linear equation

$$b_0 + \sum_{i=1}^{n+m} b_i x_i = 0, \quad b_i \in GF(2) \quad \text{for} \quad i = 0, 1, \ldots$$

that holds for $k - t$, where $t < k$, of the remaining k inputs. For inputs chosen at random we know that the equation

$$(a_0 + \sum_{i=1}^{n+m} a_i x_i)(b_0 + \sum_{i=1}^{n+m} b_i x_i) = 0$$

holds with probability $\frac{2^m - t}{2^m}$ and has degree at most two. This procedure is repeated for several linear approximations such that we obtain a large set of quadratic equations. The method extends in the straightforward way to generating equations of higher degrees. Compared to the matrix method, the advantage of the product method is that it is very fast. This is in particular relevant for large S-boxes where the matrix method is slow even when we remove only a few rows of $\underline{\underline{T_d}}$.

8.4 Application to the DES S-boxes

The eight S-boxes of DES $S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$ each maps six bits to four bits. Given the input bit string $(x_6, x_5, x_4, x_3, x_2, x_1)$, where x_6 is the most significant bit, the integer representation $(2x_6 + x_1)$ of the binary string (x_6, x_1) determines a row entry of the S-box S_i (see e.g. Table 8.1), and the integer representation of (x_5, x_4, x_3, x_2) determines the column entry of the S-box.
For each DES S-box S_i we establish the matrix $\underline{\underline{T_2}}$, which has dimensions 64×56. As stated in Table 8.3, we find that the S-box S_1, S_4, S_5 are described by respectively 1, 5 and 1 quadratic equations which are true with probability one (this was previously

Table 8.1: The S-box S_1 of DES substitutes the bit sting $(x_6, x_5, x_4, x_3, x_2, x_1)$ by (y_4, y_3, y_2, y_1).

	x_5, x_4, x_3, x_2															
x_6, x_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

noted in [39, 62]). However, a total of seven quadratic equations for eight S-boxes is not enough to mount an algebraic attack on DES by neither the Gröbner bases nor the linearization techniques. In this following we expand the set of quadratic equations by a number of probabilistic quadratic equations applying respectively the matrix method and the product method.

Example 8.4. *Consider S-box S_1 of DES. The only deterministic equation of degree at most two (also listed in [62]) is*

$$1 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + y_1 + y_2 + y_3 + y_4 + x_1x_5 + x_1y_1 + x_1y_2 + x_1y_3 + x_1y_4 + x_2x_5 + x_2y_1 + x_2y_2 + x_2y_3 + x_2y_4 + x_3x_5 + x_3y_1 + x_3y_2 + x_3y_3 + x_3y_4 + x_4x_5 + x_4y_1 + x_4y_2 + x_4y_3 + x_4y_4 + x_5x_6 + x_5y_2 + x_5y_3 + x_5y_4 + x_6y_1 + x_6y_2 + x_6y_3 + x_6y_4 + y_1y_2 + y_1y_3 + y_1y_4 = 0.$$

By removing the row of $\underline{\underline{T_2}}$, corresponding to the input string

$$(x_6, x_5, x_4, x_3, x_2, x_1) = (1, 1, 0, 0, 1, 1)$$

we obtain an extra probabilistic equation. No other single row elimination produces additional equations, but there are 75 ways of eliminate two rows which do. Increasing the elimination to three rows we find that 14 ways to eliminate rows in $\underline{\underline{T_2}}$ which all produce three equations.
There are two ways to remove four rows of $\underline{\underline{T_2}}$ where we obtain four equations, namely by removing either rows $(1, 10, 51, 33)$ or $(\overline{23}, 41, 44, 51)$. For example for the latter

Table 8.2: The S-box S_5 of DES substitutes the bit sting $(x_6, x_5, x_4, x_3, x_2, x_1)$ by (y_4, y_3, y_2, y_1).

	x_5, x_4, x_3, x_2															
	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
x_6, x_1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

we obtain

$$1 + x_1 + x_2 + x_4 + x_5 + x_6 + y_1 + y_4 + x_1x_5 + x_1y_1 + x_1y_2 + x_1y_3 + x_1y_4 + x_2x_4 + x_2x_5 + x_2x_6 + x_2y_1 + x_4x_5 + x_4y_4 + x_5x_6 + x_5y_1 + x_6y_4 + y_2y_4 + y_3y_4 = 0$$

$$x_1x_3 + x_1x_4 + x_1y_2 + x_1y_3 + x_1y_4 + x_2x_3 + x_2x_6 + x_3x_4 + x_3x_6 + x_3y_1 + x_3y_2 + x_3y_3 + x_4x_5 + x_4x_6 + x_4y_1 + x_5y_1 + x_6y_1 + x_6y_4 + y_1y_2 + y_1y_3 + y_1y_4 = 0$$

$$1 + x_1 + x_2 + x_4 + x_5 + x_6 + y_1 + y_2 + y_3 + y_4 + x_1x_3 + x_1x_4 + x_1x_5 + x_1y_1 + x_2x_3 + x_2x_5 + x_2x_6 + x_2y_1 + x_2y_2 + x_2y_3 + x_2y_4 + x_3x_4 + x_3x_6 + x_3y_1 + x_3y_2 + x_3y_3 + x_4x_6 + x_4y_2 + x_4y_3x_4y_4 + x_5x_6 + x_6y_2 + x_6y_3 = 0$$

$$x_3 + x_3x_5 + x_3y_1 + x_3y_2 + x_3y_3 + x_3y_4 + x_5y_1 + x_5y_2 + x_5y_3 + x_5y_4 = 0.$$

The four equations are simultaneously true with probability $\frac{60}{64}$.

For each of the eight S-boxes, the maximum number of quadratic equations that are simultaneous true with probabilities from $\frac{58}{64}$ to $\frac{64}{64}$, are listed in Table 8.3. The results obtained by systematically removing from one to six rows of the matrix $\underline{\underline{T_2}}$, as described in Example 8.4.

The matrix method is practical for finding equations that are true with high probability. However, the amount of equations we can derive using the matrix method in a reasonable time *might* not suffice to mount an algebraic attack. To find more equations we will try to lower the probability. For this purpose the matrix method is not practical because it has to run through too many combinations. For the DES S-boxes there are $\binom{64}{t}$ ways to eliminate t rows of the T-matrix. For example this means that to find the largest set of probabilistic equations which are true with probability $\frac{54}{64}$ we would have to compute the null space of the matrix $\underline{\underline{T_2}}$ about 2^{37} times.

Table 8.3: Number of quadratic equations describing the DES S-boxes. For each S-box S_i we tested all combinations of removing from zero to six rows of $\underline{\underline{T_2}}$. This table holds the highest number of equations obtained by this technique.

probability	$= \frac{64}{64}$	$\geq \frac{63}{64}$	$\geq \frac{62}{64}$	$\geq \frac{61}{64}$	$\geq \frac{60}{64}$	$\geq \frac{59}{64}$	$\geq \frac{58}{64}$
S_1	1	2	2	3	4	4	5
S_2	0	1	1	2	3	3	4
S_3	0	0	1	2	2	3	4
S_4	5	5	6	6	7	7	8
S_5	1	2	3	4	4	5	6
S_6	0	0	1	2	3	3	4
S_7	0	1	2	3	3	4	5
S_8	0	1	2	2	3	4	4

In the following we apply the product method to the DES S-boxes. The first step is to find linear probabilistic equations that holds for $2^6 - k$ of the 2^6 inputs of S-box S_i (where k is small). Next we search for another linear probabilistic equation that holds for as many as possible, of the k inputs that do not satisfy the first probabilistic equation.

Example 8.5. *For the DES S-box S_1 we have the linear equation*

$$x_5 + y_4 + y_3 + y_2 + y_1 + 1 = 0$$

which is true for 50 *of the* 64 *inputs. For the 14 remaining inputs we find that the linear equation*

$$1 + x_1 + x_2 + x_3 + x_4 + x_6 + y_2 + y_3 + y_4 = 0$$

is true with probability one. Thus the quadratic equation

$$(x_5 + y_4 + y_3 + y_2 + y_1 + 1)(1 + x_1 + x_2 + x_3 + x_4 + x_6 + y_2 + y_3 + y_4) = 0$$

is true with probability one. We find that the linear equation $1+x_1+x_2+x_4+x_5+x_6+y_2+y_3+y_4 = 0$ *is true for* 13 *out of the* 14 *inputs while* $1+x_2+x_5+y_1+y_2+y_3+y_4 = 0$ *is true for* 12 *of the* 14 *inputs. Thus the quadratic equation*

$$(x_5 + y_4 + y_3 + y_2 + y_1 + 1)(1 + x_1 + x_2 + x_4 + x_5 + x_6 + y_2 + y_3 + y_4) = 0$$

is true with probability $\frac{63}{64}$ *and*

$$(x_5 + y_4 + y_3 + y_2 + y_1 + 1)(1 + x_2 + x_5 + y_1 + y_2 + y_3 + y_4) = 0$$

is true with probability $\frac{62}{64}$.

In [62] it is noted that the deterministic quadratic equations for S-box S_5 can be factorized into two polynomials of degree one, and that "surprisingly" one factor is the best linear approximation for this S-box. Example 8.6 shows that the existence of the quadratic equations is a consequence of linear approximations of the S-box.

Example 8.6. *For the DES S-box S_5 we have the linear equation*

$$l_1(x, y) = x_5 + y_4 + y_3 + y_2 + y_1 + 1 = 0$$

which is true for 52 *of the* 64 *inputs. For the 12 remaining inputs we find two linear equations*

$$l_2(x, y) = (1 + x_6 + x_5 + x_2) = 0$$

$$l_3(x, y) = (1 + x_2 + x_6 + y_1 + y_2 + y_3 + y_4) = 0$$

which are true with probability one. However, we note that

$$l_1 l_3 = l_1(l_2 + 1 + l_1) = l_1 l_2 + l_1 + l_1^2 = l_1 l_2$$

when we apply $l_i^2 = l_i$ for reduction. Thus we obtain only one quadratic equation

$$(x_5 + y_4 + y_3 + y_2 + y_1 + 1)(1 + x_6 + x_5 + x_2) = 0$$

which is true with probability one.

8.5 The product method on 8-bit S-boxes.

The product method was tested on three 8-bit S-boxes namely the AES S-box, the SAFER S-box [47] and a randomly chosen S-box (given in Appendix C). For each S-box we generated a linear approximation table and selected the best linear approximations of the S-box.
For the AES S-box the best linear approximations have bias $\frac{16}{256}$, or $\frac{-16}{256}$, i.e. the corresponding linear equations hold for 144 of the 256 inputs. By choosing three of these, we found for the remaining 112 inputs a number linear approximations with bias $\frac{71}{112}$. Thus the corresponding quadratic equations have probability $\frac{215}{256}$. Below we list three such equations for the AES S-box:

$$\begin{aligned}(x_0 + y_3 + y_7)(x_1 + x_3 + y_0 + y_2 + y_3 + y_4 + y_5 + y_6) &= 0\\ (x_3 + y_1 + y_4 + y_6)(x_0 + x_2 + x_3 + x_4 + y_1 + y_4 + y_6 + y_7) &= 0\\ (x_4 + y_1 + y_2 + y_4 + y_6)(x_1 + y_3 + y_6) &= 0\,.\end{aligned}$$

We note that compared to the deterministic equations (listed in Appendix A.1) these probabilistic equations are far sparser.
For the SAFER S-box the best linear approximation,

$$3a_x \rightarrow 32_x\,,$$

has bias $\frac{-46}{256}$. Applying this approximation we find the following equation which is true with probability $\frac{237}{256}$:

$$(1 + x_1 + x_3 + x_4 + x_5 + y_1 + y_4 + y_5)(x_0 + x_1 + x_5 + y_4 + y_5 + y_6 + y_7) = 0\,.$$

The second best linear approximations for the SAFER S-box have probability $\frac{-45}{256}$. Applying such two approximations the following two equations are derived:

$$(x_0 + x_4 + y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7)(x_1 + x_3) = 0$$
$$(1 + x_1 + x_3 + x_5 + y_0 + y_3 + y_4 + y_7)(x_3 + x_6 + y_0 + y_4) = 0$$

The first equation is true with probability $\frac{236}{256}$, and the second with probability $\frac{237}{256}$. It is interesting to note that we can generate probabilistic equations with probability $\frac{237}{256}$ from both the best and the second best linear approximation. Also, we tried deriving a quadratic equation from a linear approximation with bias $\frac{42}{256}$. From this we obtained the following quadratic equation, which also has probability $\frac{237}{256}$:

$$(x_1 + x_2 + x_5 + y_0 + y_1 + y_3 + y_5 + y_7)(x_0 + x_1 + x_3 + x_4 + x_5 + x_6 + x_7 + y_3 + y_6 + y_7)\,.$$

For the randomly chosen S-box we list the following three equations:

$$(x_0 + y_0 + y_7)(x_0 + x_1 + x_2 + x_4 + x_6 + y_3 + y_4 + y_7) = 0$$
$$(x_0 + y_1 + y_5 + y_7)(x_2 + x_5 + y_0 + y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7) = 0$$
$$(x_1 + y_0 + y_3 + y_4 + y_6 + y_7)(x_0 + x_2 + x_3 + x_4 + x_7 + y_1 + y_2 + y_3 + y_4) = 0\,.$$

The first is true with probability $\frac{223}{256}$, the second with probability $\frac{225}{256}$ and the third with probability $\frac{226}{256}$.
By comparing the equations of the three S-boxes, we note that the highest probabilities were obtained for the SAFER S-box and the lowest probabilities for the AES S-box. However, our search was not exhaustive. We focused on the best linear approximations. For the SAFER S-box we found examples which show that the best linear approximation is not necessarily better than other approximations in application to the product method. For future research one idea is to do an exhaustive search for probabilistic equations using the product method.
Whether there is an intelligent way of applying the probabilistic equations in an

attack is unknown. One idea is to try to exploit the sparsity of the probabilistic equations which for the AES S-box is much more evident than for the deterministic equations (see Appendix A.1). Another idea is to try to exploit their special form of being a product of two linear equations. Also, we note that we have much freedom in choosing the probabilistic equations. For each linear approximation we found that there are many quadratic equations with the same probability. One idea is to choose the probabilistic equations such that certain terms are eliminated in a Gaussian elimination or a Gröbner basis computation.

8.6 Simulations

This section contains the results of our simulations of the probabilistic approach applied on SmallAES-2(n_r, r, c). The simulations were performed using Magma's 2.13-1 implementation of F4 version 1.8 GHz processor with 1GB ram (compared to the results of Chapter 7 the Gröbner basis computation times are improved by a factor of about two). The timing results are measured in seconds taken by the CPU. The goal is to get an impression of whether or not and to which extent algebraic attacks benefit from applying probabilistic equations.

8.6.1 Applying high-probability equations

We begin by applying high-probability equations obtained by the matrix method for the four-bit S-box applied in SmallAES(n_r, r, c) and SmallAES-2(n_r, r, c). At first we search for improvement in the Gröbner basis computation time on SmallAES-2$(n_r, 2, 1)$ when adding probabilistic equations to the set of deterministic equation. Table 8.4 presents the results where we add between zero and six probabilistic S-box equations to the sets of deterministic equations describing respectively S_1 and S_2 (see Figure 8.1). The plaintexts and keys are chosen at random and we report the computation time as the average of all tests, whether the probabilistic equations are true or not.

First of all we note that for two, three and four rounds of encryption there is almost no reduction in the computation time when we add 1, 2, 3 or 4 probabilistic equations. For five rounds of encryption we gain on average 20 percent reduction in computation time (from 101.9 to 80.9 seconds) by adding 4 equations. The probability that the four equations are simultaneous true is $\frac{3}{4}$. To test whether it is better to apply the probabilistic approach to more S-boxes at a time but with less probabilistic equations per S-box we repeated the simulations (same texts, only this time we add two probabilistic equations to each set of equations for respectively S_1 and S_2). As stated in Table 8.4 we found that while the average Gröbner basis

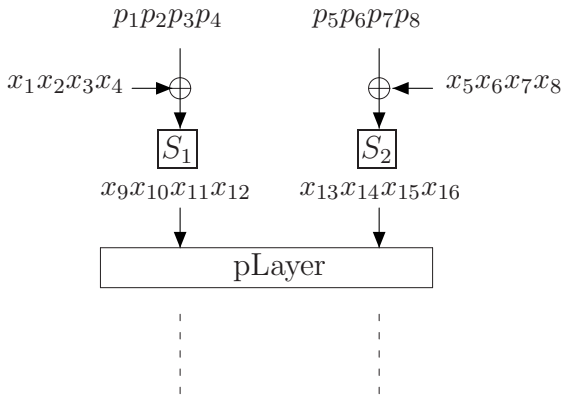


Figure 8.1: SmallAES$(n_r, 2, 1)$

computation time for the first approach was 80.9 seconds, the time for this approach was 89.4 seconds. The probability that the four equations, of the latter, are simultaneous true is $(\frac{14}{16})^2$. We conclude that though our tests are not very extensive, applying high-probability equations for the S-boxes does not appear to be very effective, because the reduction in the computation time is not large enough compared to the trade-off in having to run the attack more times. When we increase the size of the state and consider SmallAES$(n_r, 2, 2)$, then the approach of applying high-probability equations only has very little impact on the computation time compared to the deterministic approach. The approach was also tested on SmallAES-2$(n_r, 2, 2)$ which produced similar results.

8.6.2 Guessing bits

Though the tests reported in the previous section are not very exhaustive, they show only a marginal effect on the Gröbner basis computation time, which indicates the approach of applying high-probability equations in algebraic attacks on SmallAES$(n_r, 2, 2)$ and SmallAES-2$(n_r, 2, 2)$ may not be very effective. Therefore we test a simple guessing technique and examine to which extend this affects the computation time. The technique is simply to fix some state or key bit variables prior to the Gröbner basis computation. Since the variables are binary, each guess is true with probability one half. It turns out that the technique is more efficient on the cipher SmallAES-2$(n_r, 2, 2)$ than on SmallAES$(n_r, 2, 2)$ for which reason we fo-

Table 8.4: Applying high-probability equations on SmallAES$(n_r, 2, 1)$. The simulations apply between 0 and 6 probabilistic S-box equations distributed on the two S-boxes of the first round of encryptions. #prob (S_1, S_2) is the number of probabilistic equation applied over respectively S-box S_1 and S_2.

n_r	# tests	# variables	# equations	# prob (S_1, S_2)	average time
2	100	32	168	(0,0)	0.7
2	100	32	169	(1,0)	0.7
2	100	32	170	(2,0)	0.7
2	100	32	171	(3,0)	0.7
3	100	48	252	(0,0)	3.8
3	100	48	253	(1,0)	3.8
3	100	48	254	(2,0)	3.8
3	100	48	255	(3,0)	3.8
4	10	64	336	(0,0)	22.4
4	10	64	337	(1,0)	22.2
4	10	64	338	(2,0)	22.0
4	10	64	339	(3,0)	22.0
4	10	64	340	(4,0)	17.5
5	10	80	420	(0,0)	**101.9**
5	10	80	424	(4,0)	**80.9**
5	10	80	424	(2,2)	**89.4**
5	20	80	425	(3,2)	70.9
5	10	80	426	(4,2)	57.2
6	10	96	504	(0,0)	315.62
6	10	96	510	(4,2)	180.9

cus on SmallAES-2$(n_r, 2, 2)$. The encryption routine of SmallAES-2$(2, 2, 2)$ is shown in Figure 8.2 and the key schedule in Figure 8.3.

For some parts of our tests (those where we guess only one bit) there are no clear patterns in the results, i.e. the standard deviation in the computation time is large, for which reason we do not list a full series of these. Instead we bring out a few examples for fixed text values. Table 8.5 (on page 128) holds the timing results when fixing the value of one variable prior to the Gröbner bases computation. If we specify the value of one fixed variable, the Gröbner basis computation time varies for different choices of the plaintext and the key.

The results listed in Table 8.5 should be compared to the computation time when no variables are fixed which is 892 seconds. Regarding this example it is interesting,

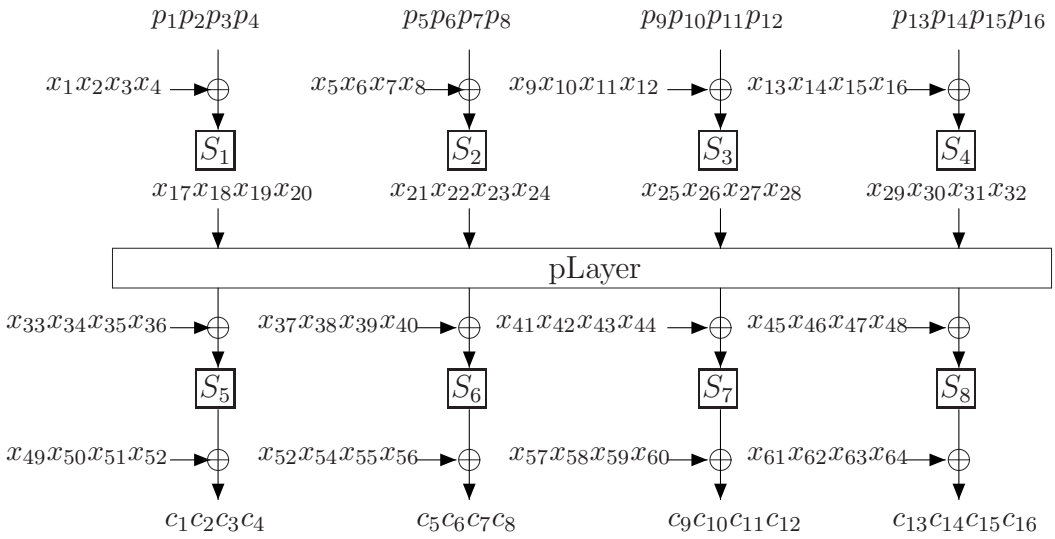


Figure 8.2: SmallAES(2,2,2)/SmallAES-2(2,2,2)

and not very intuitive, to note that while guessing one bit often reduce the Gröbner basis computation time it actually sometimes has the opposite effect. For example by guessing $x_{49} = 0$ the computation time is 948.9 seconds. Also, we note that while fixing $x_{13} = 1$ reduces the computation time to 31.7 seconds, fixing $x_{13} = 0$, which is the correct assignment, only reduces the computation time to 635.1 seconds. A possible explanation for the difference is that in the first case the equations have no solution, whereas in the second (slow) case they do, which supposedly [1] makes the Gröbner bases computation time slower. However, another result also listed in Table 8.5 contradicts this. When fixing $x_{14} = 0$ the computation time is 614.9 seconds, and we find that there is no solution, while when fixing $x_{14} = 1$ the Gröbner basis is computed, and the solution obtained, in only 29.0 seconds. This example serves to show the unclear results we observe when guessing the value of just one variable, but it also shows that the technique in some cases provides a remarkable reduction in the computation time.

The next results are obtained by fixing the value of two variables. Fortunately in this case the results are more structured. In Table 8.6 we give examples of the Gröbner basis computation time, for a fixed plaintext and key, where two variables are fixed prior to the computation.

It is interesting to note, that we obtain the best results by fixing two variables of S-box S_1. In fact we obtain a reduction of more than a factor of 100. To test whether this is only a lucky incident, we performed a number of tests with the plaintext and key chosen at random, and fixed the values of two *input* variables of the S-box S_1.

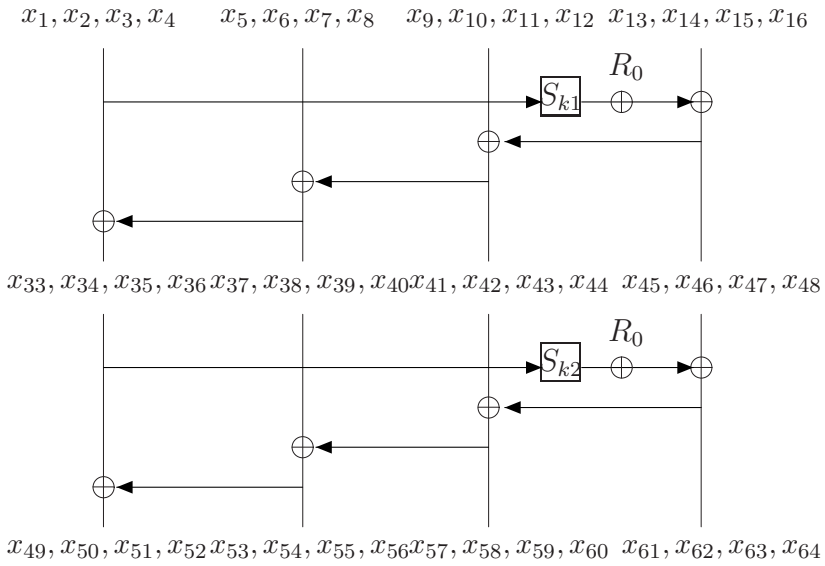


Figure 8.3: Key Schedule of SmallAES-2$(2, 2, 2)$

Similarly we performed tests, where we fixed the value of two input variables of the other S-boxes of the encryption function. Table 8.7 holds the results. We note that for S-box S_1 the Gröbner basis computation time is reduced by a factor 100 for all tests (note that the deviation is very small). In the probabilistic approach we will in worst case have to repeat the Gröbner basis computation (assigning the variables to new values) four times. In this case the total computation time on average is 31.2 seconds. Comparing this to the average computation time of the deterministic approach of 853.0 seconds this is an improvement of more than a factor 27. Also, we note from Table 8.7 that we obtain a reduction in the computation time no-matter which S-box we fix bits for. However, if we consider the results on S-box S_7, we find that if we will have to repeat the Gröbner basis computation four times in the probabilistic approach, the total computation time amounts to 752.4 seconds which is better but not very impressive compared to the results on S-box S_1. Moreover we note that for these guesses the standard deviation in the computation time is rather large. We find that the average computation time and the standard deviation is low for respectively S-box S_1, S_5 and S_8. By considering the key schedule and the encryption algorithm shown in Figures 8.2 and 8.3 we find a possible explanation for this observation. For S-box S_1, we note that the input variables x_1, x_2, x_3, x_4 are also input variables to the S-box S_{k1} of the key schedule. Likewise for S-box S_5, the key bits entering this S-box $(x_{33}, x_{34}, x_{35}, x_{36})$ are also input variables of S-box S_{k2} of the key schedule. Finally, for the S-box S_8 the key bit variables $x_{45}, x_{46}, x_{47}, x_{48}$ entering the S-box are also the output variables of S-box S_{k2}. The conclusion of

Table 8.5: Computation time to obtain a Gröbner basis for SmallAES$(2, 2, 2)$ applying F4 and guessing one bit. + denotes that we find the correct key while ÷ denotes the guess of the bit value is wrong and Magma returns the Gröbner basis $\{1\}$. This example applies the key $a67f_x$, the plaintext 0123_x, and the ciphertext= $62a5_x$.

	time			time			time			time	
-	891.7	+									
x_1	437.7	÷	x_1+1	419.3	+	x_2	440.8	÷	x_2+1	425.8	+
x_3	301.0	÷	x_3+1	431.8	+	x_4	417.1	÷	x_4+1	428.5	+
x_5	506.7	÷	x_5+1	797.0	+	x_6	796.7	÷	x_6+1	553.1	+
x_{13}	635.1	+	$x_{13}+1$	**31.7**	÷	x_{14}	614.9	÷	$x_{14}+1$	**29.0**	+
x_{15}	622.2	+	$x_{15}+1$	732.6	÷	x_{16}	652.3	÷	$x_{16}+1$	**31.4**	+
x_{29}	658.9	÷	$x_{29}+1$	745.1	+	x_{30}	**33.2**	÷	$x_{30}+1$	**26.7**	+
x_{31}	**31.3**	÷	$x_{31}+1$	613.6	+	x_{32}	653.8	÷	$x_{32}+1$	**31.2**	+
x_{37}	712.2	÷	$x_{37}+1$	727.6	+	x_{38}	684.5	÷	$x_{38}+1$	705.1	+
x_{49}	**948.9**	+	$x_{49}+1$	731.9	÷	x_{50}	937.0	÷	$x_{50}+1$	713.9	+
x_{61}	550.0	÷	$x_{61}+1$	696.0	+	x_{62}	555.4	+	$x_{62}+1$	**27.9**	÷
x_{63}	**24.4**	+	$x_{63}+1$	644.7	÷	x_{64}	552.0	+	$x_{64}+1$	**24.9**	÷

this observation is that by guessing variables that simplifies the S-box equations of both an S-box in the key schedule and the encryption algorithm we obtain the best results of the approach. Figures 8.4 and 8.5 show what the S-box polynomials look like when we fix respectively one and two input variable of S-box S_1. Figure 7.1 in Chapter 7 holds the S-box polynomials where no variables are fixed.

Also, we tested the effect of guessing three variables of one S-box. The results are listed in Table 8.8. We see that there is a reduction in the computation time, however it is marginal compared to the computation time obtained when guessing two bits.

Next we apply the guessing approach to SmallAES-2$(3, 2, 2)$. As noted in Chapter 7 Magma runs out of memory when trying to compute the Gröbner basis in the deterministic approach. As a natural choice we tested the approach, which was most efficient on SmallAES-2$(2, 2, 2)$, namely to guess bits that simplifies the S-box equations of both the key schedule and the encryption algorithm. However, the results we obtained from this were not very impressive. We had to guess six bits even for Magma to be able to compute the Gröbner bases, and for this the computation time ranges (in our limited tests) from 22 to 1112 seconds without any apparent explanation of this notable difference in the computation time.

In this chapter we looked into the possibilities of deriving and applying probabilistic equations in algebraic attacks on block cipher. We showed that the approach seems

Table 8.6: Example of guessing two bits prior to the Gröbner basis computation. The key is 0008_x, the plaintext is 0000_x and the ciphertext is 7339_x. The system of equations has more than one solution. + denotes that the right key is found, $\div$ denotes that the system has no solution, i.e. Magma returns the Gröbner basis $\{1\}$, $\cdot$ denotes that a wrong solution is found.

	time			time	
$-$	853.0	+			
x_1, x_2	7.937	+	$x_{13}, x_{14}+1$	12.984	$\div$
x_1, x_2+1	8.219	$\cdot$	$x_{13}+1, x_{14}$	212.266	$\cdot$
x_1+1, x_2	7.890	$\div$	$x_{13}+1, x_{14}+1$	16.141	$\div$
x_1+1, x_2+1	7.844	$\cdot$	x_{13}, x_{15}	13.907	+
x_1, x_3	8.109	+	$x_{13}, x_{15}+1$	13.656	$\cdot$
x_1, x_3+1	7.921	$\div$	$x_{13}+1, x_{15}$	14.250	$\cdot$
x_1+1, x_3	8.062	$\cdot$	$x_{13}+1, x_{15}+1$	13.359	$\div$
x_1+1, x_3+1	8.360	$\div$	x_{13}, x_{16}	224.063	+
x_1, x_4	8.062	$\cdot$	$x_{13}, x_{16}+1$	14.109	$\div$
x_1, x_4+1	8.156	+	$x_{13}+1, x_{16}$	13.750	$\cdot$
x_1+1, x_4	7.984	$\cdot$	$x_{13}+1, x_{16}+1$	14.500	$\div$
x_1+1, x_4+1	8.109	$\div$	x_{14}, x_{15}	224.422	+
x_1, x_5	112.891	+	$x_{14}, x_{15}+1$	13.062	$\cdot$
x_1, x_6	123.422	+	$x_{14}+1, x_{15}$	201.344	$\div$
x_1, x_7	129.781	+	$x_{14}+1, x_{15}+1$	13.204	$\div$
x_1, x_8	118.984	+	x_{14}, x_{16}	12.437	+
x_{13}, x_{14}	13.015	+	$x_{14}, x_{16}+1$	13.172	$\div$

to have some potential though our results do not pose a threat on any cipher. However, more research needs to be done. Regarding the product method it would be interesting to explore whether one can exploit the fact that the equations are products of linear equations. Also, we did not search exhaustively for the best probabilistic equations.

Table 8.7: Tests on SmallAES-2$(2, 2, 2)$ where the value of two variables of S-box S_i is guessed prior to the Gröbner basis computation.

S_i	# tests	# variables	# equations	average time	deviation
–	10	40	210	870	40.0
S_1	60	40	212	7.9	0.34
S_2	60	40	212	100.8	110.7
S_3	60	40	212	77.2	112.8
S_4	60	40	212	40.6	66.4
S_5	12	40	212	10.6	0.13
S_6	12	40	212	182.2	172.2
S_7	12	40	212	188.1	179.9
S_8	12	40	212	13.1	2.6

Table 8.8: Computation time to obtain a Gröbner basis for SmallAES-2$(2, 2, 2)$ when guessing three bits. The text pair is chosen such that there is only one key in the variety of the ideal. + denotes that we find the key.

	time			time			time	
x_1, x_2, x_3	4.249	+	x_1, x_5, x_9	5.391	+	x_1, x_2, x_4	4.046	+
x_1, x_2, x_{13}	4.593	+	x_1, x_4, x_{15}	4.640	+	x_1, x_5, x_{16}	5.531	+

$$
\begin{aligned}
x_2x_4 + x_6x_7 + x_6 + x_7 + x_8 + 1,\\
x_2x_5 + x_6x_7 + x_4 + x_5 + x_6 + x_7 + x_8 + 1,\\
x_4x_5 + x_6x_7 + x_2 + x_5 + x_6 + x_7 + x_8 + 1,\\
x_2x_6 + x_4 + x_5 + x_6 + x_8 + 1,\\
x_4x_6 + x_6x_7 + x_4 + x_6 + x_7 + 1,\\
x_5x_6 + x_6x_7 + x_2 + x_4 + x_7 + x_8,\\
x_2x_7 + x_6x_7 + x_4 + x_5 + x_6 + x_7 + x_8 + 1,\\
x_4x_7 + x_2 + x_4 + x_5,\\
x_5x_7 + x_6x_7 + x_4 + x_5 + x_6 + x_7 + x_8 + 1,\\
x_2x_8 + x_2 + x_4 + x_5,\\
x_4x_8 + x_8,\\
x_5x_8 + x_8,\\
x_6x_8 + x_2 + x_4 + x_5,\\
x_7x_8 + x_2 + x_4 + x_5,\\
x_1,\\
x_3 + x_4 + x_5 + x_6 + x_7
\end{aligned}
$$

Figure 8.4: The Gröbner basis of S-box S_1 when $x_1 = 0$.

$$
\begin{aligned}
x_5x_7,\\
x_5x_8 + x_8,\\
x_7x_8,\\
x_1,\\
x_2,\\
x_3 + x_7 + x_8 + 1,\\
x_4 + x_5,\\
x_6 + x_8 + 1
\end{aligned}
$$

Figure 8.5: The Gröbner basis of S-box S_1 when $x_1 = 0$ and $x_2 = 0$.

Chapter 9

Present

With the establishment of the AES, the need for new block ciphers has been greatly diminished; for almost all block cipher applications the AES is an excellent and preferred choice. However, it is not suitable for extremely constrained environments such as RFID tags and sensor networks. In [10] we propose an ultra-lightweight block cipher, Present. Both security and compact hardware efficiency have been equally important during the design of the cipher. Intriguingly, the hardware requirements for Present make it competitive with the leading compact stream ciphers in the eSTREAM [51] stream cipher initiative.

One defining trend of this century's IT landscape will be the extensive deployment of tiny computing devices. These devices appear routinely in consumer items and also form an integral part of a common communication infrastructure. For example as in RFID-biometric and as artifact of everyday life, such as tags on clothes instructing washing machines and a refrigerator handling the shopping in an intelligent house. It is already recognized that such deployments bring a range of very particular security risks. Yet at the same time the cryptographic solutions, and particularly the cryptographic primitives, we have at hand are unsatisfactory for extreme resource-constrained environments.

Present is a hardware-optimized block cipher that has been carefully designed with area and power constraints uppermost in mind. At the same time we have tried to avoid a compromise in security. Present is inspired by the pioneering work embodied in the DES [52] and complemented with features from two of the AES finalist candidates: Rijndael [25] and Serpent [6] which demonstrated excellent performance in hardware.

It is reasonable to ask why we might want to design a new block cipher. After all, it has become an "accepted" fact that stream ciphers are, potentially, more compact. Indeed, renewed efforts to understand the design of compact stream ciphers

are underway with the eSTREAM project and several promising proposals offer appealing performance profiles. But we note a couple of reasons why we might want to consider a compact block cipher. First, a block cipher is a flexible primitive and by running a block cipher in *counter mode* we get a stream cipher. But second, and perhaps more importantly, the art of block cipher design seems to be a little better understood than that of stream ciphers.
The following sections describes the cipher Present and its security in particular regarding the differential and algebraic properties of the cipher.

9.1 Goals and environment of use

The design criteria of Present are security, simplicity and efficiency. In this section the design decisions are explained. First, however, we describe the anticipated application requirements.
The cipher Present is suitable for extremely constrained environments and its design does not aim for wide-spread use; we already have the AES [54] for this. Instead, it targets specific applications for which the AES is unsuitable. These will generally have the following characteristics.

- Applications will only require moderate security levels. Consequently, 80-bit security will be more than adequate. Note that this is also the position taken for hardware profile stream ciphers submitted to
  eSTREAM [51].
- Applications are unlikely to require the encryption of large amounts of data.
- For many applications the key will be fixed at the time of device manufacture. There will often be no need to re-key a device and many sophisticated key manipulation attacks on block ciphers will not be possible. Attackers trying to exploit some relation between encryption keys would need to physically search among thousands of devices to find an appropriate pair.
- After security, the physical space required for an implementation will be the primary consideration. This is closely followed by peak and average power consumption, with the timing requirements being a third important metric.
- In applications that demand the most efficient use of space, the block cipher will often only be implemented as *encryption-only*. In this way it can be used within challenge-response authentication protocols and, with some careful state management, it could be used for both encryption and decryption of communications to and from the device by using the counter mode [55].

generateRoundKeys()
for $i = 1$ to 31 **do**

addRoundKey(STATE,K_i)
sBoxLayer(STATE)
pLayer(STATE)
end for

addRoundKey(STATE,K_{32})

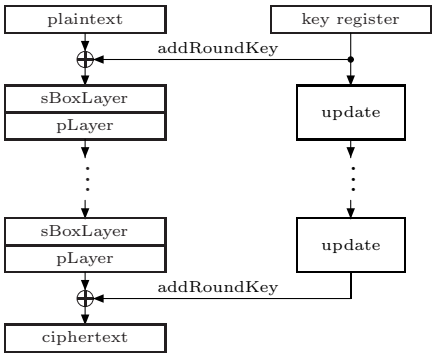


Figure 9.1: A top-level algorithmic description of Present.

Taking all these considerations into account, we decided to make Present a 64-bit block cipher with an 80-bit key. The literature contains a wide range of attacks that manipulate time-memory-data trade-offs [9] or the birthday paradox when encrypting large amounts of data (see Chapter 3). However, such attacks depend alone on the parameters of the block cipher and exploit no inner structure. Our goal is that these attacks will be the best available to an adversary.
Encryption and decryption with Present have roughly the same physical requirements. Opting to support both encryption and decryption will result in a lightweight block cipher implementation and one that is much smaller than an encryption-only implementation of AES. However, opting to implement encryption-only Present will give an ultra-lightweight solution.
We note that side-channel and invasive hardware attacks are likely to be a threat to Present, as they are to all cryptographic primitives. For the likely applications, however, the moderate security requirements reflect the very limited gain any attacker would make in practice. In a risk assessment such attacks are unlikely to be a significant factor.

9.2 The block cipher Present

Present is an SP-network that consists of 31 rounds. The block length is 64 bits and two key lengths of 80 and 128 bits are supported. Given the applications we have in mind, we recommend the version with 80-bit keys. This is more than adequate security for the low-security applications typically required in tag-based deployments, but just as important, this matches the design goals of hardware-oriented stream ciphers in the eSTREAM project and allows us to make a fairer

comparison.
Each of the 31 rounds consists of exclusive-or to introduce a round key K_i for $1 \leq i \leq 32$, where K_{32} is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The non-linear layer uses a single 4-bit S-box S which is applied 16 times in parallel in each round. The cipher is described in pseudo-code in Figure 9.1, and each stage is now specified in turn. Throughout this chapter we number bits from zero, with bit zero as the rightmost bit of the state or word.

addRoundKey.

Given round key $K_i = k^i_{63} \dots k^i_0$ for $1 \leq i \leq 32$ and current state $x_{63} \dots x_0$, addRoundKey consists of the operation for $0 \leq j \leq 63$,

$$x_j \rightarrow x_j \oplus k^i_j.$$

9.2.1 The permutation layer

pLayer

The bit permutation used in Present is given by the following table. Bit i of state is moved to bit position $P(i)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Design of the pLayer

When choosing the mixing layer, our focus on hardware efficiency demands a linear layer that can be implemented with a minimum number of processing elements, *i.e.* transistors. This leads us directly to bit permutations.
To obtain good diffusion, we would like each bit of the state to depend on each of the input bits after a few rounds. If we assume that all four output bits from a four-bit S-box depend on all four input bits, then at least three rounds will be needed to achieve good diffusion for a block length of 64 bits.

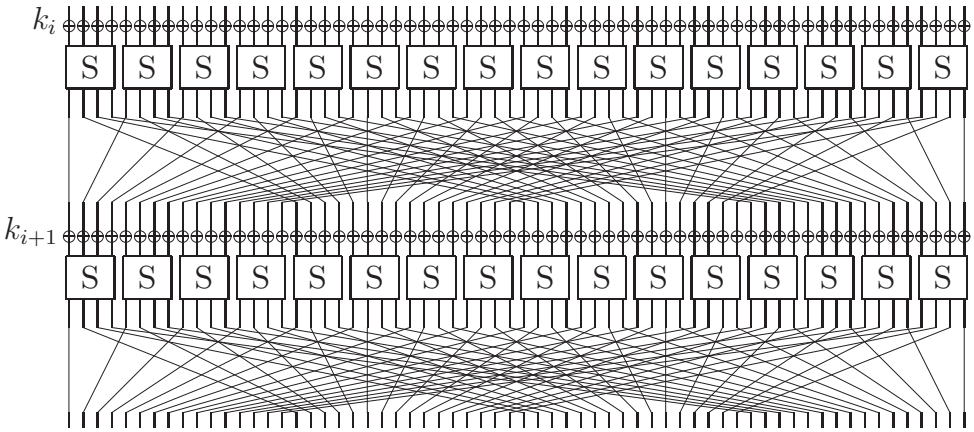


Figure 9.2: The S/P network for Present.

When considering the bit permutation P, one condition for good diffusion can be phrased in terms of matrices. Define the 4×4 and 64×64 GF(2^k) matrices A and S such that

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \text{ and } S = \begin{pmatrix} A & 0 & 0 & \dots & 0 \\ 0 & A & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & A & 0 \\ 0 & \dots & 0 & 0 & A \end{pmatrix}.$$

If L denotes the 64×64 matrix that represents the bit permutation P, then every bit of the three-round output will depend on every bit of input if, and only if, the matrix $SLSLS$ is the "all one" matrix. This is the case for our chosen permutation. Given our focus on simplicity, we have chosen a very regular bit-permutation and this in turn helps to make a clear security analysis.

9.2.2 The S-box

sBoxlayer

The S-box used in Present is a four-bit to four-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$. The action of this function in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

For sBoxLayer the current state $x_{63} \dots x_0$ is considered as sixteen 4-bit words $w_{15} \dots w_0$ where $w_i = x_{4*i+3}||x_{4*i+2}||x_{4*i+1}||x_{4*i}$ for $0 \leq i \leq 15$ and the output nibble $S[w_i]$ provides the updated state values in the obvious way.

Design of the S-box

The cipher applies a single four-bit to four-bit S-box, $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$. Such a small S-box is a direct consequence of the pursuit of hardware efficiency, with the implementation of such an S-box typically being much more compact than that required for an eight-bit S-box.
Since we use a bit permutation for the linear diffusion layer, AES-like diffusion techniques [25] are not an option for Present. Therefore we place some additional conditions on the S-boxes to improve the so-called *avalanche of change*. More precisely, the S-box for Present is chosen to fulfill the following conditions:

1. For any fixed non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and any fixed non-zero output difference $\Delta_O \in \mathbb{F}_2^4$ we require

$$\#\{x \in \mathbb{F}_2^4 \,|S(x) + S(x + \Delta_I) = \Delta_O\} \leq 4.$$

   This means that the best one round differential has probability 2^{-2}.

2. For any fixed non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and any fixed output difference $\Delta_O \in \mathbb{F}_2^4$ such that $\mathrm{wt}(\Delta_I) = \mathrm{wt}(\Delta_O) = 1$, where $\mathrm{wt}(\Delta)$ is the weight of Δ, we have

$$\{x \in \mathbb{F}_2^4 \,|S(x) + S(x + \Delta_I) = \Delta_O\} = \emptyset.$$

   This means that when the S-box input has only one bit set to one, the output has at least two bits set to one.

3. The bias of all linear approximations is at most 2^{-2}.

4. The bias of any single-bit approximation is at most 2^{-3}.

The differential and linear distribution of the Present S-box is given in Table 9.1 and Table 9.2

9.2.3 Key schedule

Present can take keys of either 80 or 128 bits. The user-supplied key is stored in a key register K and represented as $\kappa_{79}\kappa_{78} \dots \kappa_0$. At round i the 64-bit round key

Table 9.1: Difference distribution for the Present S-box

$\alpha \rightarrow \beta$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
a	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
b	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
c	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
d	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
e	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
f	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

Table 9.2: Linear approximations for the Present S-box

$\alpha \rightarrow \beta$	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	0	0	0	0	–4	0	–4	0	0	0	0	0	–4	0	4
2	0	2	2	–2	–2	0	0	2	–2	0	4	0	4	–2	2
3	0	2	2	2	–2	–4	0	–2	2	–4	0	0	0	–2	–2
4	0	–2	2	–2	–2	0	4	–2	–2	0	–4	0	0	–2	2
5	0	–2	2	–2	2	0	0	2	2	–4	0	4	0	2	2
6	0	0	–4	0	0	–4	0	0	–4	0	0	4	0	0	0
7	0	0	4	4	0	0	0	0	–4	0	0	0	0	4	0
8	0	2	–2	0	0	–2	2	–2	2	0	0	–2	2	4	4
9	4	–2	–2	0	0	2	–2	–2	–2	–4	0	–2	2	0	0
a	0	4	0	2	2	2	–2	0	0	0	–4	2	2	–2	2
b	–4	0	0	–2	–2	2	–2	–4	0	0	0	2	2	2	–2
c	0	0	0	–2	–2	–2	–2	4	0	0	–4	–2	2	2	–2
d	4	4	0	–2	–2	2	2	0	0	0	0	2	–2	2	–2
e	0	2	2	–4	4	–2	–2	–2	–2	0	0	–2	–2	0	0
f	4	–2	2	0	0	–2	–2	–2	2	4	0	2	2	0	0

$K_i = k_{63}k_{62}\ldots k_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = k_{63}k_{62}\ldots k_0 = \kappa_{79}\kappa_{78}\ldots\kappa_{16}.$$

After extracting the round key K_i, the key register $K = \kappa_{79}\kappa_{78}\ldots\kappa_0$ is updated as follows.

1. $[\kappa_{79}\kappa_{78}\ldots\kappa_1\kappa_0] = [\kappa_{18}\kappa_{17}\ldots\kappa_{20}\kappa_{19}]$
2. $[\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76}] = S[\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76}]$
3. $[\kappa_{19}\kappa_{18}\kappa_{17}\kappa_{16}\kappa_{15}] = [\kappa_{19}\kappa_{18}\kappa_{17}\kappa_{16}\kappa_{15}] \oplus$ `round_counter`

Thus, the key register is rotated by 61 bit positions to the left, the left-most four bits are passed through the Present S-box, and the `round_counter` value i is exclusive-ored with bits $\kappa_{19}\kappa_{18}\kappa_{17}\kappa_{16}\kappa_{15}$ of K with the least significant bit of `round_counter` on the right.
The key schedule for the version of Present that takes 128-bit keys is described as follows. The user-supplied key is stored in a key register K and represented as $\kappa_{127}\kappa_{126}\ldots\kappa_0$. At round i the 64-bit round key $K_i = k_{63}k_{62}\ldots k_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = k_{63}k_{62}\ldots k_0 = \kappa_{127}\kappa_{126}\ldots\kappa_{64}.$$

After extracting the round key K_i, the key register $K = \kappa_{127}\kappa_{126}\ldots\kappa_0$ is updated as follows.

1. $[\kappa_{127}\kappa_{126}\ldots\kappa_1\kappa_0] = [\kappa_{66}\kappa_{65}\ldots\kappa_{68}\kappa_{67}]$
2. $[\kappa_{127}\kappa_{126}\kappa_{125}\kappa_{124}] = S[\kappa_{127}\kappa_{126}\kappa_{125}\kappa_{124}]$
3. $[\kappa_{123}\kappa_{122}\kappa_{121}\kappa_{120}] = S[\kappa_{123}\kappa_{122}\kappa_{121}\kappa_{120}]$
4. $[\kappa_{66}\kappa_{65}\kappa_{64}\kappa_{63}\kappa_{62}] = [\kappa_{66}\kappa_{65}\kappa_{64}\kappa_{63}\kappa_{62}] \oplus$ `round_counter`

Thus, the key register is rotated by 61 bit positions to the left, the left-most eight bits are passed through two Present S-boxes, and the `round_counter` value i is exclusive-ored with bits $\kappa_{66}\kappa_{65}\kappa_{64}\kappa_{63}\kappa_{62}$ of K with the least significant bit of `round_counter` on the right.

9.3 Hardware performance

In [10] the hardware performance of Present is discussed in detail. The results are summarized in Table 9.3. For each algorithm it states the key length, the block size, and the number of cycles required to encrypt one block of the appropriate length. The throughput is stated in Kbps, which is equivalent to 1000 bits per

Table 9.3: A comparison of the clock cycles, the power consumption, and the gate count required for a variety of hardware efficient ciphers. Where appropriate we have taken the version of the cipher that provides the closest fit to the parameters of Present.

	Key size (bits)	Block size (bits)	Cycles per block	Throughput at 100KHz (Kbps)	Logic process (μW)	Area	
						GE pJ/bit	rel. (GE)
Block ciphers							
Present-80	80	64	32	200	0.18μm	1570	1
AES-128 [31]	128	128	1032	12.4	0.35μm	3400	2.17
HIGHT [38]	128	64	1	6400	0.25μm	3048	1.65
mCrypton [46]	96	64	13	492.3	0.13μm	2681	1.71
Camellia [3]	128	128	20	640	0.35μm	11350	7.23
DES [56]	56	64	144	44.4	0.18μm	2309	1.47
DESXL [56]	184	64	144	44.4	0.18μm	2168	1.38
Stream ciphers							
Trivium [36]	80	1	1	100	0.13μm	2599	1.66
Grain [36]	80	1	1	100	0.13μm	1294	0.82

second. Although power figures are not comparable for different processes, we list both the total power consumption at a clock speed of 100 KHz and the logic process which was used for the implementation. From these figures we derive the energy consumption per bit, which is of great interest for battery powered devices. The last two columns provide the area requirements both in absolute and relative figures. One gate equivalence (GE) equals the area which is occupied by a NAND gate of the appropriate process.

9.4 Differential attacks

This section mainly contains our differential observations on Present. In Section 9.4.1 we give a short description of the differential result on 16 rounds of Present. In order to evaluate the resistance of Present to differential and linear cryptanalysis we provide a lower bound to the number of so-called *active* S-boxes involved in a differential (or linear) characteristic.

Theorem 9.1. *Any five-round differential characteristic of Present has a minimum of 10 active S-boxes.*

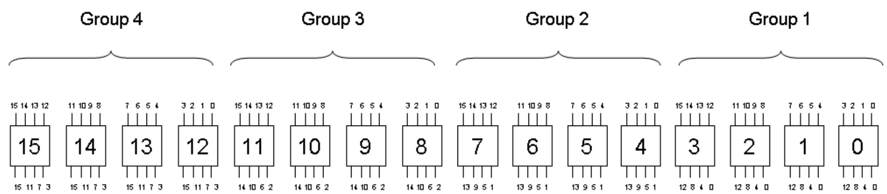


Figure 9.3: The grouping of S-boxes in Present for the purposes of cryptanalysis. The input numbers indicate the S-box origin from the preceding round and the output numbers indicate the destination S-box in the following round.

We make the following observations. We divide the 16 S-boxes in Present into four groups: boxes 0 to 3, boxes 4 to 7, boxes 8 to 11, and boxes 12 to 15 (see Figure 9.3). By examining the permutation layer one can then establish the following.

1. The four input bits to a particular S-box come from 4 distinct S-boxes of the same group.

2. The sixteen input bits to a group of four S-boxes come from 16 different S-boxes.

3. The four output bits from a particular S-box enter four distinct S-boxes, each of which belongs to a distinct group of S-boxes in the subsequent round.

4. The output bits of S-boxes in distinct groups go to distinct S-boxes in the subsequent round.

The proof of Theorem 9.1 follows from these observations.

Proof. Recalling that the rounds are indexed from 1 to 31, consider five consecutive rounds of Present ranging from $i-2$ to $i+2$ for $i \in [3 \ldots 29]$. Let D_j be the number of active S-boxes in round j. If $D_j \geq 2$, for $i-2 \leq j \leq i+2$, then the theorem trivially holds. So let us suppose that one of the D_j is equal to one. We can distinguish several cases:

Case $D_i = 1$. The S-box of Present is such that a difference in a single input bit causes a difference in at least two output bits (*cf.* the second design criterion). Thus $D_{i-1} + D_{i+1} \geq 3$. Using observation 1 above, all active S-boxes of round $i-1$ belong to the same group, and each of these active S-boxes have only a single bit difference in their output. So according to observation 2 we have that $D_{i-2} \geq 2D_{i-1}$. Conversely, according to observation 3, all active S-boxes

in round $i+1$ belong to distinct groups and have only a single bit difference in their input. So according to observation 4 we have that $D_{i+2} \geq 2D_{i+1}$. Together this gives $\sum_{j=i-2}^{i+2} D_j \geq 1+3+2\times 3 = 10$.

Case $D_{i-1} = 1$. If $D_i = 1$ we can refer to the first case, so let us suppose that $D_i \geq 2$. According to observation 3 above, all active S-boxes of round i belong to distinct groups and have only a single bit difference in their input. Thus, according to observation 4, $D_{i+1} \geq 2D_i \geq 4$. Further, all active S-boxes in round $i+1$ have only a single bit difference in their input and they are distributed so that at least two groups of S-boxes contain at least one active S-box. This means that $D_{i+2} \geq 4$ and we can conclude that $\sum_{j=i-2}^{i+2} D_j \geq 1+1+2+4+4 = 12$.

Case $D_{i+1} = 1$. If $D_i = 1$ we can refer to the first case. So let us suppose that $D_i \geq 2$. According to observation 1 above, all active S-boxes of round i belong to the same group and each of these active S-boxes has only a single bit difference in their output. Thus, according to observation 2, $D_{i-1} \geq 2D_i \geq 4$. Further, all active S-boxes of round $i-1$ have only a single bit difference in their output, and they are distributed so that at least two groups contain at least two active S-boxes. Thus, we have that $D_{i-2} \geq 4$ and therefore that $\sum_{j=i-2}^{i+2} D_j \geq 4+4+2+1+1 = 12$.

Cases $D_{i+2} = 1$ or $D_{i-2} = 1$. The reasoning for these cases is similar to those for the second and third cases.

The theorem follows. □

By using Theorem 9.1 we see that any differential characteristic over 25 rounds of Present must have at least $5 \times 10 = 50$ active S-boxes. The maximum differential probability of a Present S-box is 2^{-2} and so the probability of a single 25-round differential characteristic is bounded by 2^{-100}. Advanced techniques allow the cryptanalyst to remove the outer rounds from a cipher to exploit a shorter differential characteristic. However, even if we were to allow an attacker to remove six rounds from the cipher, a situation without precedent, then the amount of data required to exploit the remaining 25-round differential characteristic exceeds the data available. The security bounds are more than we require.

Simulations

We can identify differential characteristics that only involve ten S-boxes over five rounds. For instance, the following two-round iterative differential characteristic

involves two S-boxes per round and holds with probability 2^{-25} over five rounds.

$$\begin{aligned}\Delta &= 0000000000000011\\ &\rightarrow 0000000000030003\\ &\rightarrow 0000000000000011 = \Delta.\end{aligned}$$

We have experimentally confirmed the probability of the two-round iterative differential. In experiments over 100 independent keys using 2^{23} chosen plaintext pairs, the observed probability was as predicted. This seems to suggest that for this particular characteristic there is no accompanying significant differential.

There is also a more complicated five-round differential characteristic that holds with probability 2^{-21} over five rounds.

$$\begin{aligned}\Delta &= 0000000000007070\\ &\rightarrow 000000000000000A\\ &\rightarrow 0001000000000000\\ &\rightarrow 0000000010001000\\ &\rightarrow 0000000000880088\\ &\rightarrow 0033000000330033.\end{aligned}$$

The five round characteristic extends to eight rounds by adding the characteristic

$$\begin{aligned}\Delta &= 0033000000330033.\\ &\rightarrow 0000000000030033.\\ &\rightarrow 000000000000000B.\\ &\rightarrow 0000000000000001.\end{aligned}$$

It looks a bit odd that it comes back to having a single S-box after having had as many as six active S-boxes. For this reason we did some further analysis of the phenomena. We found that it is not an isolated event. In fact we discovered other characteristics that add to the 8 round differential

$$(0000000000007070) \rightarrow (00000000\ 00000001)$$

for example the characteristic given below

$$\begin{aligned}
\Delta &= \texttt{0000000000007070} \\
&\rightarrow \texttt{000000000000000A} \\
&\rightarrow \texttt{0001000000000000} \\
&\rightarrow \texttt{0000000010001000} \\
&\rightarrow \texttt{0000000000110011} \\
&\rightarrow \texttt{0000000000330033} \\
&\rightarrow \texttt{0000000000000033} \\
&\rightarrow \texttt{0000000000000003} \\
&\rightarrow \texttt{0000000000000001}.
\end{aligned}$$

However, these eight round characteristics have probabilities 2^{-42} respectively 2^{-50} and therefore is seems that the eight round differential will not be particular useful. Nevertheless the observation of the 7-round differential

$$(\texttt{000000000000000A}) \rightarrow (\texttt{00000000 00000001})$$

encourage examining whether there are other characteristics which starts and end with only one active S-box, and whether they are accompanied by significant differentials. It turns out that there are many such characteristics and among these some that contribute to the same differential. In Appendix B.1.1 we list a number of five, six, seven and eight round characteristics with the special property that they start and end with only one active S-box.
In the following we examine the *seven* round differential

$$(\texttt{0000000000007070}) \rightarrow (\texttt{00000000 00000001}).$$

We note that this differential has the same differences as the eight-round differential given above. The characteristics (which we found) that contribute to respectively the seven and the eight-round differential are listed in Appendix B.1.1. For the seven round differential we found three characteristics that contribute to the differential. Their probabilities are 2^{-39}, 2^{-40}, and 2^{-46}. This implies that in order to confirm the probability of the differential we would have to encrypt at least 2^{39} pairs which is rather time consuming. Instead we examine the four round differential

$$(\texttt{000000000000000A}) \rightarrow (\texttt{0000000000000033})$$

which is satisfied for four-round subchains of the three seven-round characteristics. The probabilities of the four round characteristics are 2^{-26}, 2^{-27}, and 2^{-33}. The four round differential was tested over 100 keys chosen at random and 2^{26} chosen plaintext pairs, i.e. a total of almost 2^{33} pairs. In the experiments we counted 154 right pairs out of which 52 followed the characteristic

$$\begin{aligned}\Delta &= 0000000000000000\mathtt{A}\\ &\rightarrow 0000000000001000\mathtt{1}\\ &\rightarrow 0000000000110011\\ &\rightarrow 0000000000330033\\ &\rightarrow 0000000000000033,\end{aligned}$$

which has probability 2^{-27}. The remaining 102 pairs followed the characteristic

$$\begin{aligned}\Delta &= 0000000000000000\mathtt{A}\\ &\rightarrow 0000000100000001\\ &\rightarrow 0000000001010101\\ &\rightarrow 0000000000550055\\ &\rightarrow 0000000000000033,\end{aligned}$$

which has probability 2^{-26}. Compared to the anticipated number of right pair $2^{33}2^{-27} = 2^6$, $2^{33}2^{-26} = 2^7$, and $2^{33}2^{-33} = 1$ the outcome of our tests seems reasonable and does not reveal other characteristics. The simulations were also performed for the five-round differential

$$(0000000000000707) \rightarrow (0000000000000033)$$

Again the differential was tested for 100 keys chosen at random and 2^{26} chosen plaintext pairs. We found in total 8 right pairs out of which 5 follow the characteristic

$$\begin{aligned}\Delta &= 0000000000000707\\ &\rightarrow 0000000000000000\mathtt{A}\\ &\rightarrow 0000000100000001\\ &\rightarrow 0000000001010101\\ &\rightarrow 0000000000550055\\ &\rightarrow 0000000000000033.\end{aligned}$$

which has probability 2^{-30}. The remaining 3 pairs follow the characteristic

$$\begin{aligned}
\Delta &= 0000000000000707 \\
&\rightarrow 000000000000000A \\
&\rightarrow 0000000000001001 \\
&\rightarrow 0000000000011011 \\
&\rightarrow 0000000000330033 \\
&\rightarrow 0000000000000033.
\end{aligned}$$

which has probability 2^{-31}. Again the numbers fit nicely with the expected number of right pairs which is respectively $2^{33}/2^{30} = 6$ and $2^{33}/2^{31} = 4$.
As a final observation we give the six-round iterative characteristic given below

$$\begin{aligned}
\Delta &= 0000000000000001 \\
&\rightarrow 0000000000001001 \\
&\rightarrow 0000000000011011 \\
&\rightarrow 0000000000330033 \\
&\rightarrow 0000000000000033 \\
&\rightarrow 0000000000000003 \\
&\rightarrow 0000000000000001
\end{aligned}$$

which has probability 2^{-35}. We examined the characteristic using 2^{29} chosen text pairs and 100 keys chosen at random. As anticipated from we find one pair that satisfies the six-round differential, namely one that follows the characteristic.
Summing up, it seems that even though more characteristics comprise a differential with the property of starting and ending with only one active S-box, the probability is too low to threaten the security of the cipher. However, we are aware that the effect of more characteristics contributing to the same differential is likely to increase with the number of rounds.

9.4.1 Analysis of 16-Round Present

In [63] a differential attack on 16 rounds of Present is described. The paper presents a 14-round characteristic with probability 2^{-62} and a 15-round characteristic with

probability 2^{-66}. The authors find the following four-round iterative characteristic

$$\begin{aligned}
\Delta &= 0000000000004004 \\
&\rightarrow 0000000900000009 \\
&\rightarrow 0000010100000000 \\
&\rightarrow 0500000000000500 \\
&\rightarrow 0000000000004004
\end{aligned}$$

which has probability 2^{-18}. The characteristic is applied to build a 14-round characteristic which has probability 2^{-62}. They also find 23 similar 14-round characteristics that all have probability 2^{-62}. The characteristics, which all have two active S-boxes in both first and last round, are applied to derive 8 bits of the second last round key and 24 bits of the last round key of a 16-round version of Present. The remaining 48 bits of the cipher key are obtained by exhaustive search.
The 14-round characteristics and 16-round differential attack are the best known differential results on Present. However, we note that the attack is very far from compromising the security of Present. Further more we note that no accompanying differential is reported for any of the characteristics presented in [63].

9.4.2 Algebraic attacks

Algebraic attacks have had better success when applied to stream ciphers than block ciphers. Nevertheless, the very simple structure of Present means that they merit serious study. This is particularly the case since, in principle, the key could be recovered from only a few known plaintext/ciphertext pairs.
The Present S-box is described by 21 quadratic equations in the eight input/output-bit variables over $GF(2)$. This is not surprising since it is well-known that any four bit S-box can be described by at least 21 such equations. The entire cipher can then be described by $e = n \times 21$ quadratic equations in $v = n \times 8$ variables, where n is the number of S-boxes in the encryption algorithm and the key schedule. For Present we have $n = (31 \times 16) + 31$ thus the entire system consists of $11,067$ quadratic equations in $4,216$ variables. Applying the method of Chapter 6 we find by Theorem 6.3 and 6.2 that, at degree five, there are less equations than terms. At degree six there are $2^{62.759}$ equations in at most $2^{62.755}$ terms, i.e. the number of equations exceeds the number of terms by 2^{54}. As described it is unclear whether this means that the iterated XL solves the equations at degree five, but even if it would, the computation complexity would be about 2^{191}.

As described in Chapter 7 simulations on small-scale versions of the AES show that for all but the very smallest SP-networks one quickly encounters difficulties in both time and memory complexity (see Chapter 7). The same applies to Present.

Simulations

We ran a number of simulations on small-scale variants of Present using the F_4 algorithm in Magma. When there is a single S-box, *i.e.* a very small block size of four bits, then Magma can solve the resulting system of equations over many rounds. However, by increasing the block size and adding S-boxes, along with an appropriate version of the linear diffusion layer, the system of equations soon becomes too large. Even when considering a system consisting of seven S-boxes, *i.e.* a block size of 28 bits, we were unable to get a solution in a reasonable time to a two-round version of the reduced cipher. Our analysis suggests that algebraic attacks are unlikely to pose a threat to Present.

9.5 Further information

In this chapter we only discussed differential and algebraic attacks on Present. Other important information regarding the security against linear cryptanalysis and more advanced techniques as well as the performance of Present can be found in the cipher proposal [10]. For instance there is a result, similar to Theorem 9.1, that limits the maximum bias of linear characteristics. The theorem implies that a linear attack based on a linear characteristic over 28 rounds of Present requires at least 2^{84} known plaintext/ciphertext which exceeds the size of the text space.

Chapter 10

Conclusion

The objective of this thesis was block cipher analysis. We studied general types of cryptanalysis while the main focus was aimed towards algebraic cryptanalysis. The ambition has been to extend the theory of algebraic attacks and the practical experience on block ciphers. The author also took part in the design of a new block cipher Present. The cipher targets constrained environments like for instance RFID chips for which reason it has a block size of 64 bits. We focused in particular on the security of Present against differential and algebraic attacks.

The most important issue regarding any attack is determining the complexity (both time and memory). While this is well-understood in linear and differential cryptanalysis it is still unanswered for algebraic cryptanalysis. The algebraic attacks can be classified into the categories of linearization techniques and Gröbner bases techniques. Though the categories are closely related the linearization techniques turn out to be advantageous for the analysis presented in Chapter 6. In relation to this we proposed a new method for generating the algebraic equations in an XL similar approach. The idea is to treat the equations of the linear layer and the S-box layer separately in the initial multiplication step of XL. Our approach preserves the structure of the algebraic equations and hereby provides a new angle on the equations over block cipher generated by XL techniques. Two theorems are presented giving the exact number of linearly independent equations one can generate in the initial steps of the algorithm. The results are applied to AES and a variant of AES, and it is suggested that the variant resists algebraic attacks better than AES. Also, we presented a number of simulations on small block ciphers.

While people believe that algebraic attacks in their current form are not likely to compromise the security of any realistic block cipher it is important to seek through possible improvement and ideas. In this work we looked into the possibility of applying probabilistic equations on algebraic attacks. The idea and procedure of

obtaining probabilistic equations for S-boxes of different dimensions was presented and discussed.
Also, this work extends the practical experience with algebraic attacks by a number of simulations of algebraic attacks. We applied Magma's implementation of F4 for Gröbner bases computation on respectively small scale variants of AES and Present. To our knowledge the results presented in this thesis are the best algebraic results on small scale variants of AES. Moreover, the approach of Chapter 6 was tested on some small block ciphers with some success. Finally, the derivation and application of probabilistic equations in algebraic attacks on block cipher was examined. From this we make interesting observations and in some cases improve the computation time significantly. However, for the most realistic down scale of AES the results are limited to only a few rounds of encryption because we are bounded by the maximal memory consumption allowed by the software package Magma.

10.1 Future Research

The chapter on probabilistic equations certainly leaves open questions for research. For example how do we find the best probabilistic equations and how do we apply them in practice? Regarding the probabilistic equations over 8-bit to 8-bit S-boxes it would be interesting to examine whether one could exploit their special form, being products of two linear equations and extremely sparse.
In Chapter 6 we presented methods of counting and generating equations in an XL like approach. Regarding this, there is still one important question left open, namely how many equations will be linearly independent when the equations of the S-box layer and the linear layer are mixed in Gaussian elimination? This appears to be a difficult question to answer, but we think that it is a good topic for further research.

Appendix A

AES

A.1 The AES ecryption

In the following the input of the SubBytes function is expressed as four 32-bit words (state columns)

$$x_{4\cot i+3}x_{4\cot i+2}x_{4\cot i+1}x_{4\cot i}$$

for $i = 0, \ldots, 9$. The output of MixColumns is expressed as

$$y_{4\cot i+3}y_{4\cot i+2}y_{4\cot i+1}y_{4\cot i}$$

for $i = 0, \ldots, 9$. The corresponding plaintext and cipher text is expressed as

$$p_3, p_2, p_1, p_0,$$
$$c_3, c_2, c_1, c_0\,.$$

The round keys are similarly defined as

$$w_{4\cot i+3}w_{4\cot i+2}w_{4\cot i+1}w_{4\cot i}$$

are $i = 0, \ldots, 10$. These 32-bit words are related by the following equations:

$$x_0 = p_0 + w_0, \tag{A.1}$$
$$x_1 = p_1 + w_1, \tag{A.2}$$
$$x_2 = p_2 + w_2, \tag{A.3}$$
$$x_3 = p_3 + w_3. \tag{A.4}$$

$$x_4 = y_0 + w_4, \tag{A.5}$$
$$x_5 = y_1 + w_5, \tag{A.6}$$
$$x_6 = y_2 + w_6, \tag{A.7}$$
$$x_7 = y_3 + w_7. \tag{A.8}$$

$$x_8 = y_4 + w_8, \tag{A.9}$$
$$x_9 = y_5 + w_9, \tag{A.10}$$
$$x_{10} = y_6 + w_{10}, \tag{A.11}$$
$$x_{11} = y_7 + w_{11}. \tag{A.12}$$

$$x_{12} = y_8 + w_{12}, \tag{A.13}$$
$$x_{13} = y_9 + w_{13}, \tag{A.14}$$
$$x_{14} = y_{10} + w_{14}, \tag{A.15}$$
$$x_{15} = y_{11} + w_{15}. \tag{A.16}$$

$$x_{16} = y_{12} + w_{16}, \tag{A.17}$$
$$x_{17} = y_{13} + w_{17}, \tag{A.18}$$
$$x_{18} = y_{14} + w_{18}, \tag{A.19}$$
$$x_{19} = y_{15} + w_{19}. \tag{A.20}$$

$$x_{20} = y_{16} + w_{20}, \tag{A.21}$$
$$x_{21} = y_{17} + w_{21}, \tag{A.22}$$
$$x_{22} = y_{18} + w_{22}, \tag{A.23}$$
$$x_{23} = y_{19} + w_{23}. \tag{A.24}$$

$$x_{24} = y_{20} + w_{24}, \tag{A.25}$$
$$x_{25} = y_{21} + w_{25}, \tag{A.26}$$
$$x_{26} = y_{22} + w_{26}, \tag{A.27}$$
$$x_{27} = y_{23} + w_{27}. \tag{A.28}$$

$$x_{28} = y_{24} + w_{28}, \tag{A.29}$$
$$x_{29} = y_{25} + w_{29}, \tag{A.30}$$
$$x_{30} = y_{26} + w_{30}, \tag{A.31}$$
$$x_{31} = y_{27} + w_{31}. \tag{A.32}$$

$$x_{32} = y_{28} + w_{32}, \tag{A.33}$$
$$x_{33} = y_{29} + w_{33}, \tag{A.34}$$
$$x_{34} = y_{30} + w_{34}, \tag{A.35}$$
$$x_{35} = y_{31} + w_{35}. \tag{A.36}$$

$$x_{36} = y_{32} + w_{36}, \tag{A.37}$$
$$x_{37} = y_{33} + w_{37}, \tag{A.38}$$
$$x_{38} = y_{34} + w_{38}, \tag{A.39}$$
$$x_{39} = y_{35} + w_{39}. \tag{A.40}$$

$$c_0 = y_{36} + w_{40}, \tag{A.41}$$
$$c_1 = y_{37} + w_{41}, \tag{A.42}$$
$$c_2 = y_{38} + w_{42}, \tag{A.43}$$
$$c_3 = y_{39} + w_{43}. \tag{A.44}$$

A.2 The key-schedule of AES with 128 bit keys

The 11 AES round keys, each consisting of four 32-bit key words are derived as follows. The cipher key is loaded directly into the first round key $K^{(0)} = (w_0, w_1, w_2, w_3)$. The subsequent round key for $i \in [0; 9]$
$w_{4(i+1)}, w_{4(i+1)+1}, w_{4(i+1)+2}, w_{4(i+1)+3}$ contains a non-linear part derived by passing $w_{4(i)+3}$ through a linear mix R (RotWord) and an S-box layer S consisting of four S-boxes and subsequently adding a round dependent constant $const_i$. We denote the S-box variables of the key schedule s_i^{in}, s_i^{out}. They are defined by

$$s_i^{out} = S(R(s_i^{in})) = S(R(w_{4i+3})) + const_i\,.$$

$$K^{(0)} = \begin{cases} w_0\,, \\ w_1\,, \\ w_2\,, \\ w_3\,, \end{cases}$$

$$K^{(1)} = \begin{cases} w_4 = S(R(s_0^{in})) + const_0 + w_0 = s_0^{out} + w_0\,, \\ w_5 = w_4 + w_1\,, \\ w_6 = w_5 + w_2\,, \\ w_7 = w_6 + w_3 = s_1^{in}\,, \end{cases}$$

$$K^{(2)} = \begin{cases} w_8 = S(R(s_1^{in})) + const_1 + w_4 = s_1^{out} + w_4\,, \\ w_9 = w_8 + w_5\,, \\ w_{10} = w_9 + w_6\,, \\ w_{11} = w_{10} + w_7 = s_2^{in}\,, \end{cases}$$

$$K^{(3)} = \begin{cases} w_{12} = S(R(s_2^{in})) + const_2 + w_8 = s_2^{out} + w_8\,, \\ w_{13} = w_{12} + w_9\,, \\ w_{14} = w_{13} + w_{10}\,, \\ w_{15} = w_{14} + w_{11} = s_3^{in}\,, \end{cases}$$

$$K^{(4)} = \begin{cases} w_{16} = S(R(s_3^{in})) + const_3 + w_{12} = s_3^{out} + w_{12}\,, \\ w_{17} = w_{16} + w_{13}\,, \\ w_{18} = w_{17} + w_{14}\,, \\ w_{19} = w_{18} + w_{15} = s_4^{in}\,, \end{cases}$$

$$K^{(5)} = \begin{cases} w_{20} = S(R(s_4^{in})) + const_4 + w_{16} = s_4^{out} + w_{16}\,, \\ w_{21} = w_{20} + w_{17}\,, \\ w_{22} = w_{21} + w_{18}\,, \\ w_{23} = w_{22} + w_{19} = s_5^{in}\,, \end{cases}$$

$$K^{(6)} = \begin{cases} w_{24} = S(R(s_5^{in})) + const_5 + w_{20} = s_5^{out} + w_{20}\,, \\ w_{25} = w_{24} + w_{21}\,, \\ w_{26} = w_{25} + w_{22}\,, \\ w_{27} = w_{26} + w_{23} = s_6^{in}\,, \end{cases}$$

$$K^{(7)} = \begin{cases} w_{28} = S(R(s_6^{in})) + const_6 + w_{24} = s_6^{out} + w_{24}\,, \\ w_{29} = w_{28} + w_{25}\,, \\ w_{30} = w_{29} + w_{26}\,, \\ w_{31} = w_{30} + w_{27} = s_7^{in}\,, \end{cases}$$

$$K^{(8)} = \begin{cases} w_{32} = S(R(s_7^{in})) + const_7 + w_{28} = s_7^{out} + w_{28}\,, \\ w_{33} = w_{32} + w_{29}\,, \\ w_{34} = w_{33} + w_{30}\,, \\ w_{35} = w_{34} + w_{31} = s_8^{in}\,, \end{cases}$$

$$K^{(9)} = \begin{cases} w_{36} = S(R(s_8^{in})) + const_8 + w_{32} = s_8^{out} + w_{32}\,, \\ w_{37} = w_{36} + w_{33}\,, \\ w_{38} = w_{37} + w_{34}\,, \\ w_{39} = w_{38} + w_{35} = s_9^{in}\,, \end{cases}$$

$$K^{(10)} = \begin{cases} w_{40} = S(R(s_9^{in})) + const_9 + w_{36} = s_9^{out} + w_{36}\,, \\ w_{41} = w_{40} + w_{37}\,, \\ w_{42} = w_{41} + w_{38}\,, \\ w_{43} = w_{42} + w_{39}\,. \end{cases}$$

A.3 Linear equations for the AES

In the following we list the linear equations for AES derived according to the description of Section 3.8.1. The key variables which are not inputs or output of an S-box in the key schedule. The equations describes 32-bit words and therefore each correspond to 32 equations over $GF(2)$, in total we obtain $50 \cdot 32 = 1600$ linear equations over $GF(2)$.

$$
\begin{aligned}
x_4 + s_0^{out} + y_0 + x_0 + &= p_0 \\
x_5 + s_0^{out} + y_1 + y_0 + x_1 &= p_1 + p_0 \\
x_6 + s_0^{out} + y_2 + x_2 + x_1 + x_0 &= p_2 + p_1 + p_0 \\
x_7 + s_0^{out} + y_3 + x_3 + x_2 + x_1 + x_0 &= p_3 + p_2 + p_1 + p_0 \\
x_8 + s_1^{out} + y_4 + x_4 + y_0 &= 0 \\
x_9 + s_1^{out} + y_5 + x_5 + s_0^{out} + y_1 + x_0 &= p_0 \\
x_{10} + s_1^{out} + y_6 + x_6 + y_2 + x_1 &= p_1 \\
x_{11} + s_1^{out} + y_7 + x_7 + y_7 + s_0^{out} + x_2 + x_0 &= p_2 + p_0 \\
x_{12} + s_2^{out} + y_8 + x_8 + y_4 &= 0 \\
x_{13} + s_2^{out} + y_9 + x_9 + s_1^{out} + y_5 + s_0^{out} + x_0 &= p_0 \\
x_{14} + s_2^{out} + y_{10} + x_{10} + x_6 + s_0^{out} + x_1 + x_0 &= p_1 + p_0 \\
x_{15} + s_2^{out} + y_{11} + x_{11} + y_7 + s_1^{out} + x_2 + x_1 &= p_2 + p_1 \\
x_{16} + s_3^{out} + y_{12} + x_{12} + y_8 &= 0 \\
x_{17} + s_3^{out} + y_{13} + x_{13} + s_2^{out} + y_9 + & \\
s_1^{out} + s_0^{out} + x_0 &= p_0 \\
x_{18} + s_3^{out} + y_{14} + x_{14} + y_{10} + s_1^{out} + x_1 &= p_1 \\
x_{19} + s_3^{out} + y_{15} + x_{15} + y_{11} + s_2^{out} + x_2 &= p_2 \\
x_{20} + s_4^{out} + y_{16} + x_{16} + y_{12} &= 0 \\
x_{21} + s_4^{out} + y_{17} + x_{17} + s_3^{out} + y_{13} + & \\
s_2^{out} + s_1^{out} + s_0^{out} + x_1 + x_0 &= p_1 + p_0 \\
x_{22} + s_4^{out} + y_{18} + x_{18} + y_{14} + s_2^{out} + s_0^{out} + x_0 &= p_0 \\
x_{23} + s_4^{out} + y_{19} + x_{19} + s_3^{out} + y_{15} + & \\
s_0^{out} + x_2 + x_1 + x_0 &= p_2 + p_1 + p_0
\end{aligned}
$$

$$
\begin{aligned}
x_{24} + s_5^{out} + y_{20} + x_{20} + y_{16} &= 0 \\
x_{25} + s_5^{out} + y_{21} + x_{21} + s_4^{out} + y_{17} + & \\
s_3^{out} + s_2^{out} + s_1^{out} + s_0^{out} + x_0 &= p_0 \\
x_{26} + s_5^{out} + y_{22} + x_{22} + y_{18} + s_3^{out} + s_1^{out} + x_1 &= p_1 \\
x_{27} + s_5^{out} + y_{23} + x_{23} + s_4^{out} + y_{19} + s_1^{out} + s_0^{out} + x_2 + x_0 &= p_2 + p_0 \\
x_{28} + s_6^{out} + y_{24} + x_{24} + y_{20} &= 0 \\
x_{29} + s_6^{out} + y_{25} + x_{25} + s_5^{out} + y_{21} + & \\
s_4^{out} + s_3^{out} + s_2^{out} + s_1^{out} + s_0^{out} + x_0 &= p_0 \\
x_{30} + s_6^{out} + y_{26} + x_{26} + y_{22} + & \\
s_4^{out} + s_2^{out} + s_0^{out} + x_1 + x_0 &= p_1 + p_0 \\
x_{31} + s_6^{out} + y_{27} + x_{27} + y_{23} + & \\
s_5^{out} + s_2^{out} + s_1^{out} + x_2 + x_1 &= p_2 + p_1 \\
x_{32} + s_7^{out} + y_{28} + x_{24} + y_{20} &= 0 \\
x_{33} + s_7^{out} + y_{29} + x_{25} + s_6^{out} + y_{21} + & \\
s_5^{out} + s_4^{out} + s_3^{out} + s_2^{out} + s_1^{out} + s_0^{out} + x_0 &= p_0 \\
x_{34} + s_7^{out} + y_{30} + x_{26} + y_{22} + & \\
s_5^{out} + s_3^{out} + s_1^{out} + x_1 &= p_1 \\
x_{35} + s_7^{out} + y_{31} + x_{27} + s_6^{out} + y_{23} + s_3^{out} + x_2 &= p_2 \\
x_{36} + s_8^{out} + y_{32} + x_{28} + y_{24} &= 0 \\
x_{37} + s_8^{out} + y_{33} + x_{29} + s_7^{out} + y_{25} + & \\
s_6^{out} + s_5^{out} + s_4^{out} + s_3^{out} + s_2^{out} + s_1^{out} + s_0^{out} + x_0 &= p_0 \\
x_{38} + s_8^{out} + y_{34} + x_{30} + y_{26} + & \\
s_6^{out} + s_4^{out} + s_2^{out} + s_0^{out} + x_0 &= p_0 \\
x_{39} + s_8^{out} + y_{35} + x_{31} + y_{27} + & \\
s_7^{out} + s_4^{out} + s_3^{out} + s_0^{out} + x_2 + x_1 + x_0 &= p_2 + p_1 + p_0 \\
y_{36} + s_9^{out} + y_{32} + x_{36} &= c_0 \\
y_{37} + s_9^{out} + y_{33} + s_8^{out} + x_{37} + & \\
s_7^{out} + s_6^{out} + s_5^{out} + s_4^{out} + s_3^{out} + s_2^{out} + s_1^{out} + s_0^{out} + x_0 &= c_1 + p_1 \\
y_{38} + s_9^{out} + y_{34} + x_{38} + & \\
s_7^{out} + s_5^{out} + s_3^{out} + s_1^{out} + x_1 &= p_1 + c_2 \\
y_{39} + s_9^{out} + y_{35} + x_{39} + & \\
s_8^{out} + s_5^{out} + s_1^{out} + s_0^{out} + x_2 + x_0 &= p_2 + p_0 + c_3 \\
s_0^{in} + x_3 &= p_3 \\
s_1^{in} + s_0^{out} + x_3 + x_2 + x_1 + x_0 &= p_3 + p_2 + p_1 + p_0 \\
s_2^{in} + s_1^{out} + x_3 + x_1 &= p_3 + p_1 \\
s_3^{in} + s_2^{out} + x_3 + x_2 &= p_3 + p_2 \\
s_4^{in} + s_3^{out} + x_3 &= p_3 \\
s_5^{in} + s_4^{out} + s_1^{out} + x_3 + x_2 + x_1 + x_0 &= p_3 + p_2 + p_1 + p_0 \\
s_6^{in} + s_5^{out} + x_3 + x_1 &= p_3 + p_1 \\
s_7^{in} + s_6^{out} + s_2^{out} + x_3 + x_2 &= p_3 + p_2
\end{aligned}
$$

Figure A.1: Equations for one AES S-box over $GF(2)$ in the input-output bits $x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0, y_7, y_6, y_5, y_4, y_3, y_2, y_1, y_0$

$$y_6y_7 + y_5y_6 + y_4y_7 + y_4y_5 + y_3y_6 + y_2y_6 + y_2y_4 + y_1y_7 + y_1y_5 + y_1y_4 + y_1y_3 + y_0y_5 + y_0y_4 + x_7y_7 + x_7y_6 + x_7y_5 + x_7y_4 + x_7y_3 +$$
$$x_7y_1 + x_7y_0 + x_6y_6 + x_6y_2 + x_6y_0 + x_5y_6 + x_5y_5 + x_5y_2 + x_5y_0 + x_5x_6 + x_4y_7 + x_4y_6 + x_4y_5 + x_4y_2 + x_4y_1 + x_4x_6 + x_4x_5 + x_3y_5 + x_3y_4$$
$$+x_3y_1 + x_3y_0 + x_3x_7 + x_3x_6 + x_3x_5 + x_3x_4 + x_2y_7 + x_2y_6 + x_2y_4 + x_2y_1 + x_2x_7 + x_2x_6 + x_2x_5 + x_2x_3 + x_1y_6 + x_1x_3 + x_0x_6 + 1 = 0$$
$$y_5y_6 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_7 + y_3y_6 + y_2y_7 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_2 + y_0y_7 + y_0y_6 + y_0y_5 + y_0y_3 + y_0y_1 + x_7y_7 +$$
$$x_7y_6 + x_7y_5 + x_7y_3 + x_7y_2 + x_7y_0 + x_6y_7 + x_6y_6 + x_6y_5 + x_6y_4 + x_6y_3 + x_6y_1 + x_5y_6 + x_5y_5 + x_5y_0 + x_4y_6 + x_4y_4 + x_3y_7 + x_3y_4 +$$
$$x_3y_2 + x_3y_1 + x_3y_0 + x_2y_7 + x_2y_5 + x_2y_4 + x_2y_2 + x_1y_7 + x_1y_6 + x_0 = 0$$
$$y_4y_7 + y_4y_6 + y_4y_5 + y_3y_7 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_7 + y_0y_5 + y_0y_4 + y_0y_2 + x_7y_7 +$$
$$x_7y_2 + x_7y_0 + x_6y_3 + x_6y_0 + x_5y_7 + x_5y_6 + x_5y_5 + x_5y_4 + x_5y_2 + x_5y_0 + x_5x_6 + x_4y_6 + x_4y_5 + x_4y_4 + x_4y_1 + x_4y_0 + x_4x_6 + x_4x_5 + x_3y_7$$
$$+x_3y_6 + x_3y_5 + x_3y_3 + x_3y_1 + x_3x_7 + x_3x_6 + x_3x_5 + x_3x_4 + x_2y_7 + x_2y_6 + x_2y_5 + x_2y_4 + x_2y_2 + x_2x_7 + x_2x_6 + x_2x_5 + x_2x_3 + x_1y_7 +$$
$$x_1x_3 + x_0x_6 + x_1 = 0$$
$$y_6y_7 + y_5y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_3y_6 + y_3y_4 + y_2y_6 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_1y_3 + y_0y_7 + y_0y_6 + y_0y_4 +$$
$$y_0y_3 + y_0y_2 + y_0y_1 + x_7y_7 + x_7y_6 + x_7y_4 + x_7y_3 + x_7y_1 + x_7y_0 + x_6y_5 + x_6y_4 + x_6y_3 + x_6y_1 + x_5y_7 + x_5y_6 + x_5y_5 + x_5y_3 + x_4y_1 + x_4y_0$$
$$+x_3y_6 + x_3y_5 + x_3y_4 + x_3y_3 + x_3y_2 + x_3y_1 + x_2y_6 + x_2y_4 + x_2y_2 + x_2y_1 + x_1y_7 + x_1y_6 + x_2 = 0$$
$$y_5y_6 + y_4y_5 + y_3y_7 + y_3y_5 + y_2y_4 + y_0y_7 + y_0y_4 + y_0y_1 + x_7y_7 + x_7y_4 + x_7y_0 + x_6y_6 + x_6y_5 + x_5y_4 + x_5y_3 + x_5y_2 + x_5y_1 + x_4y_5 +$$
$$x_4y_3 + x_4y_2 + x_4y_0 + x_3y_6 + x_3y_5 + x_3y_4 + x_3y_3 + x_3y_2 + x_3y_0 + x_2y_4 + x_2y_2 + x_2y_1 + x_1y_7 + x_1y_6 + x_3 = 0$$
$$y_5y_7 + y_4y_7 + y_4y_6 + y_3y_7 + y_3y_4 + y_2y_6 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_5 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_3 + y_0y_2 + x_7y_5 +$$
$$x_7y_4 + x_7y_3 + x_6y_6 + x_6y_5 + x_6y_4 + x_6y_3 + x_5y_4 + x_5y_3 + x_5y_1 + x_5y_0 + x_5x_6 + x_4y_3 + x_4y_0 + x_4x_6 + x_4x_5 + x_3y_7 + x_3y_6 + x_3y_3 + x_3y_1 +$$
$$x_3x_7 + x_3x_6 + x_3x_5 + x_3x_4 + x_2y_6 + x_2y_3 + x_2x_7 + x_2x_6 + x_2x_5 + x_2x_3 + x_1y_6 + x_1x_3 + x_0x_6 + x_4 = 0$$
$$y_6y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_6 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_5 + y_1y_4 + y_1y_3 + y_0y_6 + y_0y_5 + y_0y_2 +$$
$$y_0y_1 + x_7y_7 + x_7y_6 + x_7y_2 + x_7y_1 + x_7y_0 + x_6y_6 + x_6y_5 + x_6y_4 + x_6y_2 + x_5y_5 + x_5y_3 + x_5y_2 + x_5y_0 + x_4y_6 + x_4y_5 + x_4y_4 + x_4y_3 + x_4y_1 +$$
$$x_3y_7 + x_3y_6 + x_3y_4 + x_3y_2 + x_3y_1 + x_2y_7 + x_2y_3 + x_1y_6 + x_5 = 0$$
$$y_5y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_5 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_7 + y_0y_4 +$$
$$y_0y_3 + y_0y_2 + y_0y_1 + x_7y_7 + x_7y_6 + x_7y_5 + x_7y_3 + x_7y_1 + x_6y_6 + x_6y_5 + x_6y_2 + x_6y_1 + x_5y_7 + x_5y_6 + x_5y_5 + x_5y_4 + x_5y_2 + x_5y_1 + x_5y_0$$
$$+x_5x_6 + x_4y_5 + x_4y_1 + x_4x_6 + x_4x_5 + x_3y_7 + x_3y_5 + x_3y_2 + x_3y_1 + x_3x_7 + x_3x_6 + x_3x_5 + x_3x_4 + x_2y_5 + x_2y_4 + x_2y_3 + x_2y_2 + x_2y_1 +$$
$$x_2x_7 + x_2x_6 + x_2x_5 + x_2x_3 + x_1x_3 + x_0x_6 + x_6 = 0$$
$$y_6y_7 + y_5y_7 + y_4y_5 + y_3y_7 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_5 + y_1y_4 + y_1y_3 + y_0y_5 + y_0y_3 + x_7y_7 + x_7y_5 + x_7y_4 +$$
$$x_7y_3 + x_7y_2 + x_7y_0 + x_6y_7 + x_6y_3 + x_6y_2 + x_5y_7 + x_5y_6 + x_5y_5 + x_5y_2 + x_5y_1 + x_5y_0 + x_4y_6 + x_4y_5 + x_4y_2 + x_4y_1 + x_4y_0 + x_3y_5 + x_3y_3$$
$$+x_3y_2 + x_3y_1 + x_2y_6 + x_2y_5 + x_2y_3 + x_1y_7 + x_7 = 0$$
$$y_6y_7 + y_5y_7 + y_5y_6 + y_4y_6 + y_3y_5 + y_3y_4 + y_2y_7 + y_2y_4 + y_2y_3 + y_1y_4 + y_0y_6 + y_0y_4 + y_0y_3 + y_0y_1 + x_7y_4 + x_7y_3 + x_7y_0 + x_6y_6 +$$
$$x_6y_3 + x_6y_2 + x_6y_1 + x_6y_0 + x_5y_6 + x_5y_3 + x_5y_0 + x_4y_7 + x_4y_5 + x_4y_4 + x_4y_3 + x_4y_1 + x_4y_0 + x_3y_6 + x_3y_2 + x_3y_1 + x_3y_0 + x_2y_7 + x_2y_6$$
$$+x_2y_5 + x_2y_4 + x_2y_3 + x_2y_2 + x_2y_1 + x_1y_7 + y_0 = 0$$
$$y_6y_7 + y_4y_6 + y_3y_7 + y_3y_5 + y_3y_4 + y_2y_7 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_5 + y_1y_4 + y_0y_6 + y_0y_5 + y_0y_4 + x_7y_7 + x_7y_5 + x_7y_4 + x_7y_2 +$$
$$x_7y_1 + x_7y_0 + x_6y_7 + x_6y_6 + x_6y_4 + x_6y_3 + x_6y_1 + x_6y_0 + x_5y_5 + x_5y_4 + x_4y_7 + x_4y_5 + x_4y_0 + x_3y_7 + x_3y_6 + x_3y_5 + x_3y_4 + x_3y_3 + x_2y_6$$
$$+x_2y_5 + x_2y_4 + x_2y_3 + x_2y_2 + x_1y_6 + y_1 = 0$$

$$
\begin{aligned}
&y_5y_7 + y_5y_6 + y_3y_7 + y_3y_5 + y_2y_7 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_2 + y_0y_5 + y_0y_4 + y_0y_2 + y_0y_1 + x_7y_6 + x_7y_4 + x_7y_1 + \\
&x_6y_7 + x_6y_6 + x_6y_5 + x_6y_4 + x_6y_3 + x_6y_0 + x_5y_5 + x_5y_4 + x_5y_3 + x_5y_0 + x_4y_7 + x_4y_6 + x_4y_5 + x_4y_3 + x_4y_2 + x_4y_0 + x_3y_5 + x_3y_4 + x_3y_2 \\
&+x_2y_6 + x_2y_4 + x_2y_2 + x_1y_7 + y_2 = 0 \\
&y_6y_7 + y_4y_5 + y_3y_6 + y_2y_7 + y_2y_3 + y_1y_7 + y_1y_6 + y_0y_6 + y_0y_4 + y_0y_3 + y_0y_2 + x_7y_6 + x_7y_0 + x_6y_7 + x_6y_5 + x_6y_4 + x_6y_2 + x_5y_4 + \\
&x_5y_3 + x_4y_7 + x_4y_6 + x_4y_4 + x_4y_3 + x_3y_5 + x_3y_3 + x_3y_2 + x_2y_6 + x_2y_5 + x_2y_4 + x_2y_2 + x_2y_1 + x_1y_6 + y_3 = 0 \\
&y_5y_6 + y_4y_7 + y_4y_6 + y_2y_7 + y_2y_4 + y_1y_5 + y_1y_4 + y_1y_2 + y_0y_7 + y_0y_4 + x_7y_7 + x_7y_5 + x_7y_0 + x_6y_6 + x_6y_4 + x_6y_1 + x_5y_7 + x_5y_6 + \\
&x_5y_5 + x_5y_4 + x_5y_2 + x_5y_0 + x_4y_7 + x_4y_2 + x_4y_1 + x_3y_4 + x_3y_1 + x_2y_6 + x_2y_2 + x_2y_1 + y_4 = 0 \\
&y_5y_6 + y_4y_6 + y_3y_6 + y_3y_5 + y_2y_6 + y_2y_5 + y_2y_4 + y_1y_7 + y_1y_6 + y_1y_5 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_4 + y_0y_2 + x_7y_7 + x_6y_6 + x_6y_5 + \\
&x_6y_4 + x_6y_1 + x_5y_7 + x_5y_2 + x_5y_1 + x_4y_6 + x_4y_5 + x_4y_2 + x_4y_1 + x_4y_0 + x_3y_6 + x_3y_4 + x_3y_2 + x_2y_6 + x_2y_5 + x_2y_1 + x_1y_7 + y_5 = 0 \\
&y_5y_7 + y_4y_7 + y_4y_5 + y_3y_6 + y_3y_4 + y_2y_4 + y_2y_3 + y_0y_7 + y_0y_3 + y_0y_1 + x_7y_7 + x_7y_6 + x_7y_5 + x_7y_4 + x_7y_2 + x_6y_7 + x_6y_6 + x_6y_5 + \\
&x_6y_4 + x_6y_0 + x_5y_7 + x_5y_3 + x_5y_2 + x_5y_1 + x_5y_0 + x_4y_5 + x_4y_4 + x_4y_2 + x_3y_7 + x_3y_6 + x_3y_4 + x_3y_3 + x_3y_1 + x_2y_7 + x_2y_4 + x_2y_2 + y_6 = 0 \\
&y_5y_7 + y_3y_7 + y_3y_6 + y_3y_4 + y_2y_6 + y_2y_3 + y_1y_7 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_6 + y_0y_5 + y_0y_3 + x_7y_6 + x_7y_3 + x_7y_0 + x_6y_7 + x_6y_4 + \\
&x_6y_3 + x_6y_1 + x_5y_7 + x_5y_3 + x_5y_1 + x_4y_6 + x_4y_4 + x_3y_7 + x_3y_5 + x_3y_4 + x_2y_6 + x_2y_4 + x_1y_7 + x_1y_6 + y_7 = 0 \\
&y_3y_5 + y_3y_4 + y_2y_6 + y_2y_4 + y_1y_3 + y_1y_2 + y_0y_7 + y_0y_5 + y_0y_3 + y_0y_1 + x_7y_6 + x_7y_5 + x_6y_3 + x_6x_7 + x_5y_5 + x_5y_4 + x_5y_2 + x_5x_7 + x_5x_6 \\
&+x_4y_6 + x_4y_5 + x_4y_4 + x_4y_0 + x_4x_5 + x_3y_7 + x_3y_5 + x_3y_4 + x_3y_2 + x_3x_6 + x_3x_5 + x_3x_4 + x_2y_7 + x_2y_6 + x_2y_2 + x_2x_6 + x_2x_3 + x_1y_7 + \\
&x_1y_6 + x_1x_7 + x_1x_3 + x_0x_1 = 0 \\
&y_5y_7 + y_5y_6 + y_4y_5 + y_3y_6 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_7 + y_0y_6 + y_0y_5 + y_0y_4 + y_0y_3 + \\
&y_0y_1 + x_7y_5 + x_7y_3 + x_7y_1 + x_7y_0 + x_6y_7 + x_6y_6 + x_6y_5 + x_6y_2 + x_6y_1 + x_6y_0 + x_6x_7 + x_5y_6 + x_5y_4 + x_5y_2 + x_5y_1 + x_5y_0 + x_4y_4 + x_4y_1 + \\
&x_4x_6 + x_4x_5 + x_3y_7 + x_3y_6 + x_3y_5 + x_3y_3 + x_3y_2 + x_2y_7 + x_2y_5 + x_2y_4 + x_2x_6 + x_2x_5 + x_2x_3 + x_1y_7 + x_1x_7 + x_1x_5 + x_1x_4 + x_1x_3 + x_0x_2 = 0 \\
&y_6y_7 + y_5y_7 + y_5y_6 + y_3y_4 + y_2y_6 + y_2y_3 + y_1y_7 + y_1y_5 + y_1y_4 + y_1y_3 + y_0y_7 + y_0y_5 + y_0y_4 + y_0y_3 + y_0y_1 + x_7y_7 + x_7y_6 + x_7y_4 + \\
&x_7y_3 + x_6y_7 + x_6y_6 + x_6y_5 + x_6y_3 + x_6y_2 + x_6y_1 + x_6x_7 + x_5y_7 + x_5y_3 + x_5y_1 + x_4y_4 + x_4y_1 + x_4x_6 + x_3y_7 + x_3y_5 + x_3y_4 + x_3y_0 + \\
&x_3x_7 + x_2y_7 + x_2y_6 + x_2y_4 + x_2y_2 + x_2y_1 + x_2x_7 + x_1y_7 + x_1y_6 + x_1x_4 + x_0x_6 + x_0x_3 = 0 \\
&y_6y_7 + y_5y_7 + y_4y_5 + y_3y_7 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_5 + y_1y_4 + y_1y_3 + y_0y_5 + y_0y_3 + x_7y_3 + x_6y_6 + x_6y_5 + \\
&x_6y_4 + x_6y_3 + x_6x_7 + x_5y_6 + x_5y_5 + x_5y_3 + x_4y_7 + x_4y_6 + x_4y_5 + x_4y_4 + x_4y_3 + x_4y_2 + x_4y_1 + x_4x_5 + x_3y_7 + x_3y_6 + x_3y_4 + x_3y_2 + x_3y_1 \\
&+x_3x_7 + x_3x_5 + x_2y_5 + x_2y_3 + x_2y_1 + x_2x_5 + x_2x_4 + x_2x_3 + x_1x_7 + x_0x_6 + x_0x_4 = 0 \\
&y_5y_7 + y_5y_6 + y_4y_5 + y_3y_6 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_7 + y_0y_6 + y_0y_5 + y_0y_4 + y_0y_3 + \\
&y_0y_1 + x_7y_4 + x_7y_2 + x_7y_1 + x_7y_0 + x_6y_6 + x_6y_2 + x_6y_1 + x_6y_0 + x_5y_5 + x_5y_4 + x_5y_3 + x_5x_7 + x_4y_7 + x_4x_7 + x_4x_6 + x_3y_6 + x_3y_4 + x_3x_7 + \\
&x_3x_6 + x_3x_5 + x_3x_4 + x_2y_6 + x_2y_4 + x_2y_2 + x_2y_1 + x_2x_5 + x_2x_4 + x_2x_3 + x_1y_7 + x_1x_6 + x_1x_5 + x_1x_3 + x_0x_6 + x_0x_5 = 0 \\
&y_6y_7 + y_5y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_3y_6 + y_3y_4 + y_2y_6 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_1y_3 + y_0y_7 + y_0y_6 + y_0y_4 + \\
&y_0y_3 + y_0y_2 + y_0y_1 + x_7y_3 + x_7y_2 + x_7y_1 + x_6y_6 + x_6y_3 + x_6y_1 + x_5y_5 + x_5y_0 + x_4y_7 + x_4y_6 + x_4y_3 + x_4y_2 + x_4y_0 + x_4x_5 + x_3y_6 + x_3y_5 + \\
&x_3y_4 + x_3y_3 + x_3y_2 + x_3x_7 + x_3x_6 + x_3x_4 + x_2y_7 + x_2y_6 + x_2y_2 + x_2x_7 + x_2x_5 + x_2x_3 + x_1y_7 + x_1y_6 + x_1x_6 + x_1x_4 + x_1x_3 + x_0x_7 = 0 \\
&y_5y_7 + y_5y_6 + y_3y_7 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_5 + y_2y_4 + y_1y_5 + y_1y_3 + y_1y_2 + y_0y_7 + x_7y_6 + x_7y_5 + x_6y_5 + x_6y_2 + x_5y_6 + x_5y_5 + \\
&x_5y_4 + x_5y_3 + x_4y_6 + x_4y_4 + x_4y_3 + x_3y_3 + x_3y_0 + x_2y_7 + x_2y_5 + x_2y_3 + x_2y_1 + x_0y_0 = 0 \\
&y_5y_6 + y_4y_7 + y_4y_6 + y_3y_5 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_0y_7 + y_0y_3 + y_0y_2 + x_7y_6 + x_7y_3 + x_7y_1 + x_6y_7 + x_6y_6 + x_6y_5 + x_6y_4 + \\
&x_6y_2 + x_6y_1 + x_6y_0 + x_5y_4 + x_5y_3 + x_5y_2 + x_5y_0 + x_4y_7 + x_4y_2 + x_4y_1 + x_3y_7 + x_3y_5 + x_3y_4 + x_3y_3 + x_3y_1 + x_2y_7 + x_2y_5 + x_2y_4 + x_2y_3 \\
&+x_2y_1 + x_1y_6 + x_0y_1 = 0 \\
&y_5y_7 + y_3y_7 + y_3y_6 + y_2y_7 + y_1y_6 + y_1y_5 + y_1y_4 + y_1y_2 + y_0y_7 + y_0y_6 + y_0y_4 + y_0y_3 + y_0y_2 + y_0y_1 + x_7y_6 + x_7y_5 + x_7y_3 + x_7y_0 + \\
&x_6y_7 + x_6y_4 + x_6y_3 + x_6y_2 + x_6y_1 + x_5y_7 + x_5y_4 + x_5y_3 + x_5y_2 + x_4y_7 + x_4y_4 + x_4y_3 + x_4y_2 + x_4y_1 + x_4y_0 + x_3y_5 + x_3y_4 + x_3y_3 + x_3y_1 \\
&+x_2y_6 + x_2y_1 + x_0y_2 = 0
\end{aligned}
$$

$$y_6y_7 + y_5y_7 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_7 + y_3y_6 + y_3y_4 + y_2y_7 + y_2y_6 + y_2y_5 + y_2y_4 + y_1y_5 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_6 + y_0y_4 +$$
$$x_7y_7 + x_7y_5 + x_7y_4 + x_7y_2 + x_7y_1 + x_6y_3 + x_6y_1 + x_6y_0 + x_5y_4 + x_5y_1 + x_4y_7 + x_4y_5 + x_4y_4 + x_4y_1 + x_3y_4 + x_3y_3 + x_3y_2 + x_2y_7 + x_2y_5$$
$$+x_2y_4 + x_1y_7 + x_0y_3 = 0$$
$$y_4y_7 + y_4y_6 + y_3y_6 + y_3y_5 + y_2y_7 + y_2y_5 + y_2y_3 + y_1y_7 + y_1y_2 + y_0y_6 + y_0y_5 + y_0y_4 + y_0y_3 + x_7y_7 + x_7y_5 + x_7y_3 + x_7y_0 + x_6y_6 +$$
$$x_6y_5 + x_6y_4 + x_6y_3 + x_6y_2 + x_6y_1 + x_5y_7 + x_5y_5 + x_5y_4 + x_5y_3 + x_5y_2 + x_5y_0 + x_4y_6 + x_4y_2 + x_4y_1 + x_3y_7 + x_3y_6 + x_3y_1 + x_2y_6 + x_2y_5 +$$
$$x_2y_2 + x_2y_1 + x_0y_4 = 0$$
$$y_5y_7 + y_5y_6 + y_4y_5 + y_3y_6 + y_3y_4 + y_2y_7 + y_2y_6 + y_1y_6 + y_1y_5 + y_1y_4 + y_1y_3 + y_0y_7 + y_0y_6 + y_0y_5 + y_0y_2 + y_0y_1 + x_7y_5 + x_7y_4 +$$
$$x_7y_3 + x_7y_2 + x_7y_1 + x_6y_7 + x_6y_1 + x_5y_5 + x_5y_4 + x_5y_1 + x_4y_7 + x_4y_3 + x_4y_1 + x_3y_6 + x_3y_5 + x_2y_6 + x_2y_4 + x_2y_2 + x_1y_7 + x_0y_5 = 0$$
$$y_6y_7 + y_4y_7 + y_4y_6 + y_4y_5 + y_2y_7 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_5 + y_0y_5 + y_0y_4 + y_0y_3 + x_7y_6 + x_7y_4 + x_7y_2 + x_6y_5 + x_6y_2 +$$
$$x_6y_0 + x_5y_6 + x_5y_5 + x_5y_2 + x_5y_0 + x_4y_7 + x_4y_5 + x_4y_3 + x_4y_2 + x_4y_0 + x_3y_7 + x_3y_6 + x_3y_5 + x_2y_7 + x_2y_6 + x_2y_3 + x_1y_6 + x_0y_6 = 0$$
$$y_5y_7 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_6 + y_3y_5 + y_1y_6 + y_0y_7 + y_0y_6 + y_0y_4 + y_0y_3 + y_0y_2 + x_7y_6 + x_7y_3 + x_7y_2 + x_7y_1 + x_6y_6 + x_6y_4 +$$
$$x_6y_2 + x_6y_1 + x_6y_0 + x_5y_7 + x_5y_5 + x_5y_4 + x_5y_2 + x_4y_6 + x_4y_2 + x_4y_1 + x_3y_6 + x_3y_4 + x_3y_3 + x_3y_2 + x_3y_1 + x_2y_7 + x_2y_6 + x_2y_4 + x_2y_2 +$$
$$x_1y_7 + x_0y_7 = 0$$
$$y_5y_6 + y_4y_7 + y_4y_6 + y_3y_4 + y_2y_6 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_5 + y_0y_2 + y_0y_1 + x_7y_5 + x_7y_3 + x_6y_7 +$$
$$x_6y_6 + x_6y_5 + x_6y_3 + x_6y_2 + x_6y_0 + x_5y_7 + x_5y_4 + x_5y_3 + x_5y_2 + x_5y_0 + x_5x_7 + x_4y_7 + x_4y_0 + x_4x_6 + x_4x_5 + x_3y_6 + x_3y_3 + x_3x_6 + x_3x_5 +$$
$$x_2y_7 + x_2y_5 + x_2y_2 + x_2x_7 + x_2x_5 + x_2x_4 + x_2x_3 + x_1y_7 + x_1x_4 + x_1x_2 = 0$$
$$y_6y_7 + y_5y_7 + y_5y_6 + y_4y_5 + y_3y_7 + y_3y_6 + y_3y_5 + y_2y_5 + y_2y_4 + y_1y_2 + y_0y_6 + y_0y_5 + y_0y_3 + y_0y_1 + x_7y_7 + x_7y_4 + x_7y_1 + x_6y_5 +$$
$$x_6y_2 + x_6y_0 + x_5y_4 + x_5y_3 + x_4y_6 + x_4y_4 + x_4y_0 + x_3y_7 + x_3y_5 + x_3y_2 + x_2y_6 + x_2y_5 + x_2y_3 + x_1y_7 + x_1y_6 + x_1y_0 = 0$$
$$y_5y_6 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_7 + y_3y_6 + y_3y_5 + y_2y_5 + y_1y_6 + y_1y_5 + y_0y_7 + y_0y_6 + y_0y_5 + y_0y_4 + y_0y_2 + y_0y_1 + x_7y_6 + x_7y_4 +$$
$$x_7y_3 + x_7y_2 + x_6y_6 + x_6y_4 + x_5y_7 + x_5y_5 + x_5y_4 + x_5y_3 + x_5y_2 + x_5y_1 + x_4y_7 + x_4y_6 + x_4y_5 + x_4y_4 + x_4y_2 + x_4y_1 + x_3y_5 + x_2y_7 + x_2y_6$$
$$+x_2y_5 + x_2y_4 + x_2y_2 + x_1y_1 = 0$$
$$y_6y_7 + y_5y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_3y_6 + y_3y_5 + y_2y_5 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_4 + y_1y_2 + y_0y_6 + y_0y_5 + y_0y_4 + y_0y_2 + x_7y_5 +$$
$$x_7y_2 + x_6y_5 + x_6y_2 + x_6y_1 + x_5y_7 + x_5y_5 + x_5y_3 + x_5y_2 + x_5y_1 + x_5y_0 + x_4y_7 + x_4y_6 + x_4y_5 + x_4y_2 + x_4y_0 + x_3y_7 + x_3y_2 + x_2y_7 +$$
$$x_2y_6 + x_2y_3 + x_1y_7 + x_1y_6 + x_1y_2 = 0$$
$$y_6y_7 + y_4y_7 + y_4y_6 + y_3y_7 + y_3y_6 + y_3y_5 + y_2y_3 + y_1y_7 + y_1y_6 + y_1y_5 + y_1y_4 + y_1y_2 + y_0y_6 + y_0y_4 + y_0y_3 + y_0y_2 + y_0y_1 + x_7y_5 +$$
$$x_7y_3 + x_7y_1 + x_6y_7 + x_6y_3 + x_6y_1 + x_5y_7 + x_5y_5 + x_5y_4 + x_5y_3 + x_5y_2 + x_4y_6 + x_4y_4 + x_4y_3 + x_4y_2 + x_4y_1 + x_3y_5 + x_3y_4 + x_3y_3 + x_3y_2$$
$$+x_3y_1 + x_2y_4 + x_2y_3 + x_2y_2 + x_1y_7 + x_1y_6 + x_1y_3 = 0$$
$$y_4y_5 + y_3y_7 + y_3y_6 + y_3y_5 + y_3y_4 + y_2y_6 + y_2y_5 + y_2y_4 + y_2y_3 + y_1y_7 + y_1y_5 + y_1y_4 + y_1y_3 + y_1y_2 + y_0y_7 + y_0y_6 + y_0y_4 + x_7y_6 +$$
$$x_7y_4 + x_7y_3 + x_6y_7 + x_6y_5 + x_6y_4 + x_6y_3 + x_6y_2 + x_6y_0 + x_5y_7 + x_5y_6 + x_5y_5 + x_5y_3 + x_5y_2 + x_5y_1 + x_5y_0 + x_4y_6 + x_4y_2 + x_4y_1 + x_4y_0$$
$$+x_3y_7 + x_3y_6 + x_3y_5 + x_3y_3 + x_3y_1 + x_2y_7 + x_2y_5 + x_2y_4 + x_2y_3 + x_2y_1 + x_1y_4 = 0$$
$$y_6y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_3y_7 + y_2y_5 + y_2y_4 + y_1y_6 + y_1y_2 + y_0y_7 + y_0y_5 + y_0y_4 + y_0y_2 + y_0y_1 + x_7y_6 + x_7y_4 + x_7y_3 + x_7y_2 +$$
$$x_7y_0 + x_6y_3 + x_6y_2 + x_6y_1 + x_6y_0 + x_5y_6 + x_5y_1 + x_4y_3 + x_3y_7 + x_3y_5 + x_3y_4 + x_2y_4 + x_2y_2 + x_1y_7 + x_1y_6 + x_1y_5 = 0$$
$$y_6y_7 + y_5y_7 + y_5y_6 + y_4y_7 + y_4y_6 + y_4y_5 + y_3y_7 + y_3y_5 + y_1y_6 + y_1y_5 + y_1y_2 + y_0y_3 + y_0y_2 + y_0y_1 + x_7y_5 + x_7y_4 + x_6y_7 + x_6y_5 +$$
$$x_6y_4 + x_6y_1 + x_6y_0 + x_5y_6 + x_5y_5 + x_5y_3 + x_5y_1 + x_5y_0 + x_4y_7 + x_4y_6 + x_4y_5 + x_4y_4 + x_4y_1 + x_4y_0 + x_3y_7 + x_3y_6 + x_3y_4 + x_3y_2 + x_2y_7$$
$$+x_2y_4 + x_2y_3 + x_2y_0 = 0$$

Appendix B

Present

B.1 Test vectors

The test vectors for Present with an 80-bit key are shown below in hexadecimal notation.

plaintext	*key*	*ciphertext*
00000000 00000000	00000000 00000000 0000	5579C138 7B228445
00000000 00000000	ffffffff ffffffff ffff	e72c46c0 f5945049
ffffffff ffffffff	00000000 00000000 0000	a112ffc7 2f68417b
ffffffff ffffffff	ffffffff ffffffff ffff	3333dcd3 213210d2

B.1.1 Differential characteristics -Present

Six round differential characteristic for Present with probability 2^{-35}:

$$
\begin{aligned}
\Delta \;=\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1 \\
\rightarrow\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1 \\
\rightarrow\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1 \\
\rightarrow\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,3\,3\,0\,0\,3\,3 \\
\rightarrow\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,3\,3 \\
\rightarrow\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,3 \\
\rightarrow\; & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1 \\
& pr_{char} = (\frac{1}{8})^7(\frac{1}{4})^7 = 2^{-35}
\end{aligned}
$$

Six round iterative differential characteristic for Present with probability 2^{-45}:

$$\begin{aligned}
\Delta &= 0000000000000000a \\
&\rightarrow 0001000000001000 \\
&\rightarrow 0000000010101010 \\
&\rightarrow 000000000aa00aa \\
&\rightarrow 0033000003300000 \\
&\rightarrow 0000000000003030 \\
&\rightarrow 000000000000000a \\
&\quad pr_{char} = (\frac{1}{8})^{11}(\frac{1}{4})^{6} = 2^{-45}
\end{aligned}$$

Six round differential characteristic for Present with probability 2^{-35}:

$$\begin{aligned}
\Delta &= 0000000000010000 \\
&\rightarrow 0000000000100010 \\
&\rightarrow 0000000000220022 \\
&\rightarrow 0000033000000033 \\
&\rightarrow 0000000000000303 \\
&\rightarrow 0000000000000005 \\
&\rightarrow 0000000000000001 \\
&\quad pr_{char} = (\frac{1}{4})^{7}(\frac{1}{8})^{7} = 2^{-35}\,.
\end{aligned}$$

Five round differential characteristic for Present with probability 2^{-40}:

$$\begin{aligned}
\Delta &= 0000000000010000 \\
&\rightarrow 0000001000100010 \\
&\rightarrow 0000000002220222 \\
&\rightarrow 0000007700000077 \\
&\rightarrow 0000000000000303 \\
&\rightarrow 0000000000000005 \\
&\quad pr_{char} = (\frac{1}{4})^{8}(\frac{1}{8})^{8} = 2^{-40}\,.
\end{aligned}$$

Seven round differential characteristic for Present with probability 2^{-39}:

$$\begin{aligned}
\Delta &= 0000000000007070 \\
&\rightarrow 000000000000000a \\
&\rightarrow 0000000100000001 \\
&\rightarrow 0000000010101011 \\
&\rightarrow 0000000005500550 \\
&\rightarrow 0000000000000033 \\
&\rightarrow 0000000000000003 \\
&\rightarrow 0000000000000001 \\
&\quad pr = (\frac{1}{4})^9(\frac{1}{8})^7 = 2^{-39}\,.
\end{aligned}$$

The four round sub-characteristic

$$\begin{aligned}
&\rightarrow 000000000000000a \\
&\rightarrow 0000000100000001 \\
&\rightarrow 0000000010101011 \\
&\rightarrow 0000000005500550 \\
&\rightarrow 0000000000000033
\end{aligned}$$

has probability 2^{-26}.

Seven round differential characteristic for Present with probability 2^{-40}:

$$\begin{aligned}
\Delta &= 0000000000007070 \\
&\rightarrow 000000000000000a \\
&\rightarrow 0000000000010001 \\
&\rightarrow 0000000000110011 \\
&\rightarrow 0000000000330033 \\
&\rightarrow 0000000000000033 \\
&\rightarrow 0000000000000003 \\
&\rightarrow 0000000000000001 \\
&\quad pr = (\frac{1}{4})^8(\frac{1}{8})^8 = 2^{-40}\,.
\end{aligned}$$

The four round sub-characteristic

$$\begin{aligned}
&\rightarrow \texttt{0000000000000000a}\\
&\rightarrow \texttt{0000000000001000 1}\\
&\rightarrow \texttt{0000000000110011}\\
&\rightarrow \texttt{0000000000330033}\\
&\rightarrow \texttt{0000000000000033}
\end{aligned}$$

has probability 2^{-27}.
Seven round differential characteristic for Present with probability 2^{-46}:

$$\begin{aligned}
\Delta \;=\; &\texttt{0000000000007070}\\
&\rightarrow \texttt{000000000000000a}\\
&\rightarrow \texttt{0001000100010000}\\
&\rightarrow \texttt{000000000ee00ee}\\
&\rightarrow \texttt{0000000000330033}\\
&\rightarrow \texttt{0000000000000033}\\
&\rightarrow \texttt{0000000000000003}\\
&\rightarrow \texttt{0000000000000001}\\
&\quad pr = (\frac{1}{4})^5(\frac{1}{8})^{12} = 2^{-46}\,.
\end{aligned}$$

The four round sub-characteristic

$$\begin{aligned}
&\rightarrow \texttt{000000000000000a}\\
&\rightarrow \texttt{0001000100010000}\\
&\rightarrow \texttt{000000000ee00ee}\\
&\rightarrow \texttt{0000000000330033}\\
&\rightarrow \texttt{0000000000000033}
\end{aligned}$$

has probability 2^{-33}.

Eight round differential characteristic for Present with probability 2^{-42}:

$$\begin{aligned}
\Delta &= 0000000000007070 \\
&\rightarrow 000000000000000a \\
&\rightarrow 0001000000000000 \\
&\rightarrow 0000000010001000 \\
&\rightarrow 0000000000110011 \\
&\rightarrow 0000000000330033 \\
&\rightarrow 0000000000000033 \\
&\rightarrow 0000000000000003 \\
&\rightarrow 0000000000000001 \\
&\quad pr = (\frac{1}{4})^9(\frac{1}{8})^8 = 2^{-42}\,.
\end{aligned}$$

The five round sub-characteristic

$$\begin{aligned}
&\rightarrow 000000000000000a \\
&\rightarrow 0001000000000000 \\
&\rightarrow 0000000010001000 \\
&\rightarrow 0000000000110011 \\
&\rightarrow 0000000000330033 \\
&\rightarrow 0000000000000033
\end{aligned}$$

has probability 2^{-29}.

Eight round differential characteristic for Present with probability 2^{-52}:

$$\begin{aligned}
\Delta &= 0000000000007070 \\
&\rightarrow 000000000000000a \\
&\rightarrow 0001000000010000 \\
&\rightarrow 0000000101010100 \\
&\rightarrow 000000000aa00aa \\
&\rightarrow 0000000000330033 \\
&\rightarrow 0000000000000033 \\
&\rightarrow 0000000000000003 \\
&\rightarrow 0000000000000001 \\
&\quad pr = (\frac{1}{4})^8(\frac{1}{8})^{12} = 2^{-52}\,.
\end{aligned}$$

The five round sub-characteristic

$$\begin{aligned}
&\rightarrow \texttt{0000000000000000a}\\
&\rightarrow \texttt{0001000000010000}\\
&\rightarrow \texttt{0000000010101010}\\
&\rightarrow \texttt{000000000aa00aa}\\
&\rightarrow \texttt{0000000000330033}\\
&\rightarrow \texttt{0000000000000033}
\end{aligned}$$

has probability 2^{-49}.
Eight round differential characteristic for Present with probability 2^{-56}:

$$\begin{aligned}
\Delta \;=\;& \texttt{0000000000007070}\\
&\rightarrow \texttt{000000000000000a}\\
&\rightarrow \texttt{0001000100000001}\\
&\rightarrow \texttt{0000000011011101}\\
&\rightarrow \texttt{000000000ddd00dd}\\
&\rightarrow \texttt{0000000000330033}\\
&\rightarrow \texttt{0000000000000033}\\
&\rightarrow \texttt{0000000000000003}\\
&\rightarrow \texttt{0000000000000001}\\
&\quad pr = (\frac{1}{4})^{10}(\frac{1}{8})^{12} = 2^{-56}.
\end{aligned}$$

The five round sub-characteristic

$$\begin{aligned}
&\rightarrow \texttt{000000000000000a}\\
&\rightarrow \texttt{0001000100000001}\\
&\rightarrow \texttt{0000000011011101}\\
&\rightarrow \texttt{000000000ddd00dd}\\
&\rightarrow \texttt{0000000000330033}\\
&\rightarrow \texttt{0000000000000033}
\end{aligned}$$

has probability 2^{-45}.

Appendix C

Randomly chosen 8-bit S-box

The randomly chosen S-box applied in Section 8.5 is generated in a c++ program using the *rand*() number generator. The S-box is presented in hexadecimal form in Table C.1. For example the element $2a_x$ is substituted by the element of row 2_x, column a_x resulting in 24_x.

Table C.1: Random 8-bit to 8-bit S-box in hexadecimal notation. The S-box is applied in Section 8.5.

x,y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	2d	cf	46	29	4	b4	78	d8	68	a7	ff	3f	2b	f1	fc	d9
1	7a	96	9	2c	a5	57	74	64	c4	af	15	28	a4	e9	db	5e
2	20	fb	38	a8	4e	a6	14	93	25	56	24	44	df	59	8d	43
3	7b	be	90	16	89	9d	7e	77	c6	2f	26	98	88	f5	30	d4
4	34	3a	d	f	bd	a1	f7	f4	3c	35	e5	8f	2a	cd	c2	40
5	c1	52	71	6a	72	23	b0	5b	1b	a3	82	19	e	f2	6b	ae
6	84	49	d6	83	42	17	e7	39	73	e6	8a	fe	21	e4	75	8e
7	87	85	58	dd	5d	8b	5a	a9	ef	ec	91	9c	41	18	eb	6
8	ee	b8	6c	33	70	97	d7	12	d2	94	95	66	6f	fa	ab	aa
9	bf	ba	5f	b1	81	61	45	47	c7	4a	3b	d0	d5	7	bc	cc
a	62	86	55	1c	e3	b3	27	10	9a	5	8	a	3e	d1	e0	65
b	6d	b	60	de	b7	36	bb	a2	48	c9	c5	69	32	c8	51	1d
c	ed	67	fd	ea	53	dc	4c	1a	7f	4f	cb	f6	f9	54	b5	1e
d	63	ac	e1	f0	4d	50	b6	8c	c3	c	5c	1f	11	76	ad	7d
e	0	c0	e2	a0	1	13	9e	80	f8	ce	f3	2e	92	31	22	b2
f	9b	37	ca	7c	6e	99	d3	4b	2	3	da	3d	9f	e8	b9	79

Bibliography

[1] Martin Albrecht and Carlos Cid. Algebraic Techniques in Differential Cryptanalysis. In *First International Conference on Symbolic Computation and Cryptography*, 2008.

[2] Henning Ejnar Andersen. On puncturing of codes from Norm-Trace curves. *Finite Fields and their Applications*, 13 nr 1:136–157, 2007.

[3] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In Douglas R. Stinson, editor, *Proceedings of Selected Areas in Cryptography 2000*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer-Verlag, 2000.

[4] Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers,
Available via `http://eprint.iacr.org/2007/024`, 2007.

[5] Thomas Becker and Volker Weispfenning. *Gröbner Basis a Computational Approach to Commutative Algebra*. Springer-Verlag, 1993.

[6] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption '98*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238, 1998.

[7] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, pages 3–72, 1991.

[8] Eli Biham and Adi Shamir. Differential Cryptanalysis of the Full 16-round DES. In Ernest Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *LectureNotes in Computer Science*, pages 487–496. Springer-Verlag, 1991.

[9] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved Time-memory Trade-offs with Multiple Data. In Bart Preneel and Stafford Tavares, editors, *Proceedings of SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 110–127. Springer-Verlag, 2006.

[10] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsø. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466, 2007.

[11] An Braeken and Bart Preneel. Probabilistic Algebraic Attacks. In *10th IMA International Conference*, volume 3796 of *Lecture Notes in Computer Science*, pages 290–303, 2005.

[12] Bruno Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41:475–511, 2006.

[13] Carolynn Burwick, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas, Jr. Luke, Oconnor Mohammad Peyravian, David Safford, and Nevenko Zunic. MARS - A Candidate Cipher for AES. *NIST AES Proposal*, 1998.

[14] Carlos Cid and G. Leurant. An Analysis of the XSL Algorithm. In B. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 333–352. Springer-Verlag, 2005.

[15] Carlos Cid, Sean Murphy, and Matthew J.B. Robshaw. Small Scale Variants of the AES. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption, 12th International Workshop, Paris,France*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer-Verlag, 2005.

[16] Carlos Cid, Sean Murphy, and Matthew J.B. Robshaw. *Algebraic Aspects of the Advanced Encryption Standard*. Springer-Verlag, 2006.

[17] Nicolas Courtois. Higher Order Correlation Attacks, XL algorithm and Cryptanalysis of Toyocrypt. In P.J. Lee and C.H Lim, editors, *The 5th International Conference on Information Security and Cryptology ICISC-2002, Seoul, Korea*, volume 2587, pages 182–199. Springer-Verlag, 2003.

[18] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt 2000*, volume

1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer-Verlag, 2000.

[19] Nicolas Courtois and Willy Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[20] Nicolas Courtois and Josef Pieprzyk. Cryptoanalysis of Block Ciphers with Overdefined System of equations. In Y. Zheng, editor, *Advances in Cryptology -Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.

[21] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer-Verlag, first edition, 1998.

[22] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms, An Introduction to Computational Geometry and Commutative Algebra*. Springer-Verlag, second edition, 2005.

[23] Joan Daemen, Michäel Peeters, Gilles. Van Assche, and Vincent Rijmen. Nessie proposal: NOEKEON. Submitted as a NESSIE Candidate Algorithm. Available via `http://www.cryptonessie.org`.

[24] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, 1999.

[25] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, Secaucus, NJ, USA, 2002.

[26] Claus Diem. The XL-Algorithm and a Conjecture from Commutative Algebra. In P .J. Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 323–337, 2004.

[27] Withfield Diffie and Martin Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.

[28] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139:61–88, 1999.

[29] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to 0 (F_5). In T. Mora, editor, *Proceedings of ISSAC*, pages 75–83. ACM Press, 2002.

[30] H. Feistel. Cryptography and Computer Privacy. 228:15–23, 1973.

[31] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES algorithm. In M. Joye and J.-J. Quisquater, editors, *Proceedings of CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer-Verlag, 2004.

[32] Niels Ferguson, Chris Hall, John Kelsey, Bruce Schneier, David Wagner, and Doug Whiting. Twofish: A 128-bit Block Cipher. In *First Advanced Encryption Standard (AES) Conference*, 1998.

[33] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley publishing, Inc, 2003.

[34] Michael R. Garey and David S. Johnso. Computers and Intractability. *Bell Telephone Laboratories, Incorporated*, 1979.

[35] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[36] T. Good, W. Chelton, , and M. Benaissa. Hardware Results for Selected Stream Cipher Candidates, Presented at SASC 2007. Available via `http://www.ecrypt.eu.org/stream/`, 2007.

[37] Martin Hellman. A cryptanalytic time-memory trade-off. *Transactions on Information Theory,IEEE*, it-26, no. 4:401–406, 1980.

[38] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In L. Goubin and M. Matsui, editors, *Proceedings of CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer-Verlag, 2006.

[39] Thomas Jakobsen. *Higher Order Cryptanalysis of Block Ciphers*. PhD thesis, Technical University of Denmark, 1999.

[40] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In M. Wiener, editor, *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 1999.

[41] Lars Ramkilde Knudsen. Personal communication.

[42] Lars Ramkilde Knudsen and Kaisa Nyberg. Provable Security Against a Differential Attack. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 8(1):27–37, 1995.

[43] Lars Ramkilde Knudsen and Charlotte Vikkelsø. Technical report: On Algebraic Attacks on Block Ciphers, 2005.

[44] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In *Proceedings of EUROCRYPT'91*, Lecture Notes in Computer Science, pages 17–38. Springer-Verlag, 1991.

[45] Niels Lauritzen. *Concrete Abstract Algebra*, volume 3.0. Cambridge University Press, February 2002.

[46] C. Lim and T. Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors. In J. Song, T. Kwon, and M. Yung, editors, *Workshop on Information Security Applications - WISA'05*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, 2005.

[47] James L. Massey. SAFER K-64: A byte-oriented block ciphering algorithm. In Ross Anderson, editor, *Fast Software Encryption*, volume 809 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[48] M. Matsui. Linear cryptanalysis method for DES cipher. In editor T. Helleseth, editor, *Advances in Cryptology EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1993.

[49] T. Moh. The Method of "Relinearization" of Kipnis and Shamir and its application to TMM.

[50] Sean Murphy and Matthew J.B. Robshaw. Essential Algebraic Structure within AES. In M. Yung, editor, *Advances in Cryptology CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[51] ECRYPT Network of Excellence. The Stream Cipher: estream. Available via `www.ecrypt.eu.org/stream`.

[52] National Bureau of Standards. Data Encryption Standard. *Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C.*, January 1977.

[53] Nationale Institute of Standards and Technology. Accouncement available via http://csrc.nist.gov/publications/fips/05-9945-des-withdrawl.pdf, 2005.

[54] Nationale Institute of Standards and Technology. Advanced Encryption Standard. *FIPS 197, US Department of Commerce, Washington D.C.*, November 2001.

[55] Nationale Institute of Standards and Technology. SP800-38A. Available via www.csrc.gov.

[56] A. Poschmann, G. Leander, K Schramm, and C Paar. A Family of Light-Weight Block Ciphers Based on DES Suited for RFID Applications. In A. Biryukov, editor, *Proceedings of FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[57] Ronald L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin. The RC6 TM Block Cipher. In *First Advanced Encryption Standard (AES) Conference*, 1998.

[58] Victor Shoup. Ntl: A Library for Doing Number Theory. Available via http://www.shoup.net/ntl/.

[59] Allan Steel. Allan Steel's Gröbner Basis Timing Page. Available via http://magma.maths.usyd.edu.au/users/allan/gb/.

[60] Douglas R. Stinson. *Cryptography Theory and Practice*. RC Press LLC, 1995.

[61] Makoto Sugita, Mitsuru Kawazoe, and Hideki Imai. Relation between the XL Algorithm and Gröbner Basis Algorithms. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A:11–18, 2006.

[62] Toshinobu Kaneko Takeshi Shimoyama. Quadratic Relation of S-box and Its Application to the Linear Attack of Full Round DES. In H. Krawczyk, editor, *Advances in Cryptology CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[63] Meiqin Wang. Differential Cryptanalysis of Reduced-Round PRESENT. In *AFRICACRYPT*, pages 40–49, 2008.