Eindhoven University of Technology
Department of Mathematics and Computer Science

MASTER'S THESIS

Multiple right-hand side equations

A new tool for cryptanalysis

by
A.C.C. Schoonen

Eindhoven, May 2008

Supervisor:
Prof. dr. ir. H.C.A. van Tilborg
Eindhoven University of Technology

Advisors:
Prof. dr. I.A. Semaev
Dr. H. Raddum
University of Bergen

Abstract

The design of a typical modern block cipher enables potential attackers to represent the cipher as a system of multivariate algebraic equations. Solving this equation system is then equivalent to breaking the cipher, and the corresponding attack is commonly referred to as an algebraic attack. Essentially, algebraic attacks can be distinguished in two separate classes. The first class are the attacks that are tailor-made for specific ciphers, by carefully analyzing specific properties of the encryption process. The second class consists of more general attacks that focus on solving multivariate equation systems, such as the ones representing ciphers. In practice, this second class mainly consists of techniques that employ well-documented notions like Gröbner-bases and the Boolean satisfiability problem.

Although considerable effort has been spent on solving equation systems that describe ciphers, practical results have generally failed to materialize, which may indicate that the representation of a cipher as a system of algebraic equations is not the most suitable approach. In this thesis, we present an alternative type of equation, which is particularly effective for representing a large class of ciphers. Although these equations are not linear by definition, they have some linear properties that can be exploited by an algorithm for solving them. We propose such an algorithm and study its behavior when applied to equation systems that represent small-scale variants of the block ciphers DES and AES.

Preface

This thesis concludes my graduation project, which is part of the Master of Science program "Industrial and Applied Mathematics" at the Eindhoven University of Technology. The research herein was performed at the Selmer center, which is the center for coding theory and cryptology of the University of Bergen, Norway.

It is very interesting to see how recent cryptanalytic research has focused on the solution of large systems of algebraic (typically quadratic) equations that represent cryptosystems. For some reason, the cryptographic community seems convinced that algebraically defined ciphers (i.e. ciphers that are defined using algebraic transformations) should be analyzed using this particular representation. However, in many cases, practical results have yet to be reported and the described attacks typically have a computational complexity that is much higher than the brute force attack, at least when they are applied to real-life ciphers. This might imply that alternative representation methods should be considered. One of these alternative methods is to use extended versions of ordinary linear equations that can have multiple outcomes, or right-hand sides. We will show that a large class of ciphers can be represented very efficiently using these so-called multiple right-hand side (MRHS) equations.

The concept of multiple right-hand side equations and their possible use in the field of cryptanalysis is a rather new and unconventional subject, and there is only a small body of work on this approach. Although the ideas contained in this thesis have been published earlier, e.g. in [RS07] and [Rad07], we generalize this new type of equation in more detail, as well as a possible algorithm for solving them. Independent experimental results are presented and several examples are included to clarify the theory.

The thesis is organized as follows:

- Chapter 1 briefly describes the mathematical prerequisites and notations that are used throughout the rest of the thesis;
- Chapter 2 serves as an introduction to symmetric cryptography and algebraic attacks in general. In particular, some previous attacks are described and analyzed;
- Chapter 3 introduces multiple right-hand side equations and a general algorithm for solving them;
- Chapter 4 describes a particular implementation of the algorithm;
- Chapter 5 covers the use of MRHS equations in cryptanalysis and gives experimental results;
- Chapter 6 concludes the theory and describes, in varying detail, some possible topics for future research, including an alternative problem description;
- The appendices contain details of the investigated ciphers and some reference code.

I would like to thank the people that were involved in this project, in particular Igor Semaev and Håvard Raddum for many fruitful discussions on the subject and suggestions

on this thesis. Furthermore, Tor Helleseth for offering me the opportunity to perform my research in beautiful Norway, and Henk van Tilborg for overall supervision and for reviewing updated versions of this thesis numerous times. Also, thanks to Benne de Weger and Hennie Wilbrink for being on my graduation committee. A general thanks to the people at the Selmer center, who have provided a very comfortable and enjoyable working atmosphere.

Finally, I want to thank Sonja and Lex for being nearly infinitely patient, and my father for being the most supportive person I know, no matter what the circumstances, and for giving me the opportunity to finish my education in the first place. A special word of thanks to my mother, who I know would have been particularly proud.

Ad Schoonen, 2008

Contents

1

Mathematical preliminaries

This chapter will summarize some of the mathematical basics used throughout this thesis. We will not go into the theory with too much detail; interested readers are referred to, for instance, [Mey01].

1.1 Finite fields

All operations are in the finite field $\mathbb{F}_q = GF(q)$, where q is a prime power. We will denote the field addition by + and the corresponding zero-element by 0. In the special case $q = 2^n$ for some n (i.e. fields of characteristic 2), the addition may be denoted by $\oplus$. The field multiplication will be denoted by $\cdot$, although commonly it is simply omitted and we may use x_1x_2 to indicate the product of elements x_1 and x_2. The corresponding unit element is denoted by 1.

Elements of $\mathbb{F}_2$ are called bits. A special field used in Section 5.4 is $\mathbb{F}_{2^8}$. Elements of this field, also called bytes, may be represented in three different ways:

- as a **binary vector** of eight bits, cf. 1.2,
- as a **polynomial** with coefficients in $\mathbb{F}_2$ and degree at most seven,
- as a **pair of hexadecimals**, each representing four bits.

We will use the so-called most-significant bit first notation, which means that the leftmost element in vector notation corresponds to X^7 in polynomial notation, etc. For example, the binary vector (10011100) may be written as the polynomial $X^7 + X^4 + X^3 + X^2$ or in hexadecimals as `9C`.

1.2 Vector spaces

We can define an n-dimensional vector space V over a field K. Elements of V are represented as vectors of length n, whose entries are elements of K. A subset W of a vector space V is called a subspace if it satisfies two axioms:

- W **is closed under addition.** For any pair of vectors $w_1, w_2 \in W$, the sum $w_1 + w_2$ is also in W.

- W **is closed under (scalar) multiplication.** For any vector $w \in W$ and any scalar $\alpha \in K$, the product αw is also in W.

Consider a vector space V and $v_1, \ldots, v_m \in V$, then a linear combination of these vectors is a vector of the form

$$\alpha_1 v_1 + \ldots + \alpha_m v_m,$$

where $\alpha_1, \ldots, \alpha_m \in K$. A vector v is called (linearly) dependent of a given set of vectors if it can be written as a linear combination of these vectors, otherwise the vector is independent.

In any vector space V, there is always a set of independent vectors (called a basis) such that any vector in V can be written as a linear combination of these basis vectors. For an n-dimensional vector space, any basis consists of n vectors. In our case, the vector space V is defined over the finite field $K = \mathbb{F}_q$, and we will denote the n-dimensional vector space as $\mathbb{F}_q^n$.

In this thesis the vector consisting of only ones will be denoted by $\underline{1} = (1, \ldots, 1)$. Its length is determined by the context. Similarly, the all-zero vector is denoted by $\underline{0}$, and we may refer to this as the zero vector (or simply zero). The general vector consisting of only r's is then denoted by $r \cdot \underline{1}$. The concatenation of vectors x and y will be denoted by $(x \mid\mid y)$, i.e.

$$(x \mid\mid y) = (x_1, \ldots, x_n, y_1, \ldots, y_n).$$

Finally, the inner product of vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ is defined as

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i.$$

The inner product exists only if x and y have the same length.

1.3 Matrices

Given vector spaces $\mathbb{F}_q^n$ and $\mathbb{F}_q^m$, we can define a function (or transformation) $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$. This function is called linear if it satisfies

$$\forall \alpha, \beta \in \mathbb{F}_q \ \ \forall x, y \in \mathbb{F}_q^n \ \ : \ \ f(\alpha x + \beta y) = \alpha f(x) + \beta f(y).$$

Assume the set $\{e_1, \ldots, e_n\}$ is a basis of $\mathbb{F}_q^n$ and let x be a column vector in K. Then, we can alternatively describe the linear transformation $f(x)$ as the product Fx, where F is the $m \times n$ matrix whose columns are $f(e_i)$ for $i = 1, \ldots, n$. Note that matrix-vector multiplication is defined as

$$Fx = \begin{pmatrix} \langle F_1, x \rangle \\ \vdots \\ \langle F_m, x \rangle \end{pmatrix},$$

where we denote the the i-th row of F as F_i. The i-th column of F will be denoted as F^i. In this thesis, we will not use powers of matrices, so there should be no confusion when using this notation.

Three important tools when processing matrices are the elementary row operations:

- Interchange two rows;
- Multiply a row by a non-zero constant;

- Add a non-zero multiple of one row to another row.

A special matrix is the $n \times n$ identity matrix, denoted by I_n, which contains only zeroes except on the diagonal, where it has only ones. Also, $\mathcal{O}$ denotes the all-zero matrix (not necessarily square), whose dimensions will be determined by the context. Note that for all vectors x of appropriate length, it holds that $I_n x = x$ and $\mathcal{O}x = \underline{0}$.

The number of linearly independent rows of a matrix A is called the (row) rank of A, and will be denoted by $\rho(A)$. If this number equals the number of rows of A, the matrix is said to have full row rank (i.e. the rows form an independent set of vectors). The row rank of a matrix can be determined by using an algorithm known as Gaussian elimination, cf. 1.4. In this thesis, we will only be concerned with the row rank of a matrix, and we will omit the term "row". Hence, full rank formally means "full row rank".

1.4 Gaussian elimination

An important algorithm used in this thesis is the process of Gaussian elimination. It is an algorithm that uses elementary row operations to transform a matrix into reduced row echelon form (RREF). A matrix is in RREF if

1. Any non-zero row is above any zero row,
2. The leading coefficient of any row equals one,
3. The leading coefficient of any row is strictly to the right of the leading coefficient of the row above it,
4. All entries in the same column above a leading coefficient are zero,

where the leading coefficient of a row is defined as the leftmost non-zero coefficient. In essence, the RREF of a matrix A gives the minimal set of vectors (formed by the non-zero rows of the resulting matrix) such that each row of A can be written as a linear combination of these vectors. If the RREF of A contains one or more zero rows, this implies that A has linearly dependent rows.

We will not present the details of the algorithm here, but mention that it consists of two separate stages. In the first stage, also called the elimination step, the matrix is transformed into row echelon form, for which conditions 1, 2 and 3 are required. The second stage, also referred to as backward substitution, transforms the matrix into RREF by forcing it to satisfy condition 4.

The rank of a matrix A can be easily determined by transforming A into RREF. The number of non-zero rows in the RREF of A is then equal to $\rho(A)$.

Example 1.1. Consider over $\mathbb{F}_2$ the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then, by interchanging rows and adding multiples of one row to another, this matrix is transformed into RREF as follows:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \overset{(1)}{\sim} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \overset{(2)}{\sim} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \overset{(3)}{\sim} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Here, (1) follows from moving the third row to the top of the matrix. The second step (2) is to subtract the second row from the first, and (3) is obtained by subtracting the third row from the second row. Note that the final matrix satisfies the properties for the reduced row echelon form of a matrix. The rank of the initial matrix A equals three, as the RREF of A has three non-zero rows.

1.5 Linear equation systems

A linear equation system in the variables $X = \{x_1, \ldots, x_n\}$ is commonly represented as an equation of the form

$$AX = b, \tag{1.1}$$

where A is an $m \times n$ coefficient matrix with elements in $\mathbb{F}_q$, and $b \in \mathbb{F}_q^m$ is referred to as the right-hand side. The set of variables X is represented as a column vector, i.e.

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

Solutions to (1.1) are those assignments of values to the variables $x_1, \ldots, x_n$ such that equality holds. If a solution exists, the system is called consistent.

A common method to solve a linear equation system uses the Gaussian elimination algorithm to determine the RREF of the augmented matrix

$$(A \mid\mid b),$$

after which the solutions can be easily read out.

Example 1.2. Consider over $\mathbb{F}_2$ the linear equation in variables $X = (x_1, \ldots, x_5)$ given by

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} X = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}. \tag{1.2}$$

This equation can be solved by applying the Gaussian elimination procedure to the augmented matrix A (cf. Example 1.1). We have already determined that the RREF of A is

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

This implies that the solution to (1.2) is described by the equations

$$\begin{cases} x_1 = 0 \\ x_2 + x_4 + x_5 = 1 \\ x_3 + x_5 = 1. \end{cases}$$

1.6 Computational complexity

Computational complexity analysis is a branch of computer science that studies the amount of resources needed to run a particular algorithm. An excellent reference on the topic of

algorithms and (computational) complexity is [Kob98]. In this thesis, we will be interested in two measurable quantities:

- Time complexity, i.e. the amount of time required to run the algorithm,
- Memory complexity or memory requirement, i.e. the amount of memory consumed by the algorithm.

Although the memory requirement of an algorithm is rather easy to describe (for example, an algorithm may require to store a particular number of bits), the time complexity is rather hard to specify. After all, a faster computer will process an algorithm in less time than a slower one, and we want the time complexity to measure the efficiency of the algorithm, and not of the particular implementation or hardware used. For this reason, it is customary to express the time complexity of an algorithm as the number of elementary operations (additions, multiplications) needed by the algorithm. This quantity is implementation and hardware-independent, and as such is a good measure for the efficiency of an algorithm.

The size of the input to an algorithm affects both the time and memory complexities of the algorithm: running the same algorithm on a larger instance of a problem will typically require more operations and memory. Generally, these quantities are expressed as a function of the size n of the input, and we are especially interested in the asymptotic behavior (i.e. for $n \to \infty$) of algorithms. To formalize this, we introduce the O or big-O notation, which is defined as

$$\text{if } f(n)/g(n) \to \text{ constant as } n \to \infty, \text{ then } f(n) = O(g(n)).$$

In less formal terms, $O(\cdot)$ only gives the most dominant term(s) of an expression, and disregards scalar multiplications and lower order terms. For example, the polynomial

$$f(n) = 3n^3 + 2n + 1$$

in big-O notation is $O(n^3)$. Alternatively, we may state that f is in the order of n^3 or even that f is approximately n^3. Indeed, the difference between f and n^3 approaches zero as n approaches ∞.

An algorithm that has time complexity $O(n^c)$ for some $c > 1$ is said to be polynomial. Some additional terminology is in Table 1.1, where $c > 1$ is a constant. Each entry increases faster (for increasing n) than the entries above it, so of these examples an exponential algorithm is the least efficient. In practice, any algorithm that has a time complexity that is

Notation	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n^c)$	Polynomial
$O(c^n)$	Exponential

Table 1.1: Big-O notation.

polynomial or less is considered efficient. However, many algebraic attacks (cf. 2.2) involve exponential algorithms, which makes them impractical (unless the value of n is kept small).

Note that Table 1.1 is not complete, e.g. there are algorithms that behave worse than polynomial but better than exponential, etc. We will return to the concept of complexity in Section 2.3.

2

Algebraic cryptanalysis

The purpose of this chapter is to give a brief introduction into the concepts of cryptography and in particular algebraic attacks on block ciphers, as well as a general description of a few currently known and studied attacks.

2.1 Symmetric cryptography

Cryptography is the study of the design and analysis of systems that enable its users to exchange information with a certain degree of confidentiality or authenticity. A method to achieve this is by using a symmetric cryptographic system (or cryptosystem), in which the involved parties use a pre-shared secret (the encryption key) to mask their messages using a process called encryption. In arguably cryptography's most influential paper [Sha49], Shannon suggested the following model for a general secrecy system, which basically describes our notion of a symmetric cryptosystem.

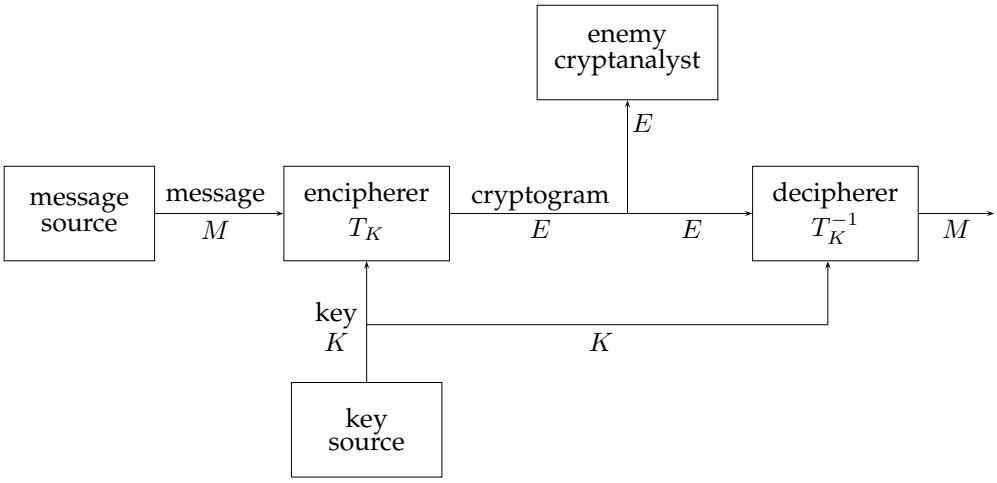

Figure 2.1: Shannon's model of a "secrecy system".

In this model, the message M is transformed into a seemingly random cryptogram E by a process called the encipherer T_K, which has the pre-shared key K as additional input. The cryptogram is then sent to the intended recipient, where the channel used for communication is possibly insecure. This means that it is assumed that not only the intended recipient but also an external party (called an enemy cryptanalyst) can intercept E. The received cryptogram can then be translated into the original message M by an operation T_K^{-1} called the decipherer, which basically is the inverse of T_K for a fixed key K. Since only the intended recipient has access to K (it is important that K is kept secret by both parties), no enemy cryptanalyst should be able to retrieve M from E. This form of cryptography, in which the key used for encryption is equal to the key used for decryption, is called symmetric cryptography and will be the focus of this thesis.

It was already recognized by Shannon that for fixed K, the encipherer T_K acts as a function from the set of all possible messages to all possible cryptograms, and that the particular choice for this function mainly determines the level of security of the system: if T_K is not able to mask the message M sufficiently, an enemy cryptanalyst might be able to, upon reception of a cryptogram, retrieve the corresponding message without knowing the key, by a process called cryptanalysis. If the enemy succeeds, the system is said to be broken. In principle, if the number of possible keys is finite, as is often the case in practice, an enemy can break any system by trying all possible keys (this method is referred to as an exhaustive search or brute force attack). However, if the number of possibilities is sufficiently large, this will take an undesirable amount of time. This gives a requirement on the function T_K, which Shannon further generalized by stating that

> "we may construct our cipher in such a way that breaking it is equivalent to (or requires at some point in the process) the solution of some problem known to be laborious".

The problem he suggested is the simultaneous solution of a large number of equations[1] in a large number of variables, cf. 2.3. So, if we are able to describe the cryptosystem as a huge equation system, whose solution implies a break of the system, the difficulty of solving these equations should ensure the security of the cryptosystem.

Shannon's model is still used, although the nomenclature has been slightly adjusted: we will use

- plaintext p instead of message M,
- ciphertext c instead of cryptogram E,
- encryption E_K instead of encipherer T_K,
- decryption D_K instead of decipherer T_K^{-1},
- adversary or attacker instead of enemy cryptanalyst,
- cryptosystem or cipher instead of secrecy system.

The description of a particular process of cryptanalysis may also be referred to as an attack. Note that for decryption to be possible, it should hold for any K that if $c = E_K(p)$, then $p = D_K(c)$.

[1] Shannon mentions these equations should be "complex", which is to be interpreted as hard or difficult.

2.2 Algebraic analysis

Where most pre-Shannon ciphers use either basic pen-and-paper or mechanical techniques, modern ciphers are typically implemented using integrated circuits, i.e. chips and computers. Such ciphers use operations defined in some finite field $\mathbb{F}_q$, usually $\mathbb{F}_2$ or one of its related fields, to translate a plaintext p into a seemingly random ciphertext c using a secret key K. In practice, the plain- and ciphertext and the key are represented as vectors over the finite field $\mathbb{F}_q$, and encryption can be viewed as a transformation from a particular vector space to another (or itself). Typically, this transformation is non-linear, which makes it hard to reverse without any further knowledge, e.g. the encryption key.

Clearly, any attack will benefit from more known information. For example, if only a received ciphertext is given (the so-called ciphertext only setting), it is very unlikely that an attacker will be able to choose between all possible plaintexts. Another possibility is that an attacker has access to one or more plaintexts and the ciphertexts that were obtained by encrypting these plaintexts using the same encryption key. This is the known plaintext setting, and it will be assumed from now on, unless indicated otherwise. In this case, most encryption and decryption processes can be described by a system of equations in many variables (a so-called multivariate equation system), in which the encryption key acts as one of the variables and the plain-/ciphertext pairs are known constants. An extension of the known plaintext setting is the chosen plaintext setting, in which the attacker can even generate the corresponding ciphertext for any desired plaintext.

If the multivariate equation system describing a cipher is solved, the encryption key can be extracted from the solution and the cipher is essentially broken. In terms of Shannon's theory, this is an attack on the cryptosystem, which will be referred to as an algebraic attack: to (attempt to) solve a multivariate equation system that represents a cipher. Most modern ciphers require a huge number of (typically quadratic) equations and variables to be described by such a system. For example, the 128-bit version of AES (Section 5.4) can be represented by 8,000 quadratic equations in 1,600 variables over $\mathbb{F}_2$. Since the systems describing a cipher typically contain a lot of non-linear equations in a lot of variables (as described by Shannon's early requirement for a cipher), which are hard to solve in comparison to linear equations, the search for an efficient algorithm for solving these systems is a topic of cryptographic research.

So far, several proposals have been made, but only a few have led to successful and practical attacks on hardly used cryptosystems. It seems that cipher designers are careful enough to make it impossible (or at least very difficult) to describe their ciphers as a system of easily solvable equations. On the other hand, cryptanalysts seem to believe that cipher designers (unknowingly) rely on the intractability of multivariate non-linear equation systems a bit too much, and that the equation system describing a typical cipher is actually easier to solve than a system that is randomly generated. We will formally determine how difficult solving a random multivariate equation system is.

2.3 Complexity of multivariate equations

As noted, it is not unlikely that the multivariate equation system describing a typical cipher is not as complex as a random system. A natural question is then how complex a truly random system is, and we will build a common framework to formalize this and explain its consequences.

2.3.1 General definitions

We start this section with some definitions.

Definition 2.1. *A decision problem is a problem P that can be answered by either "yes" or "no".*

Definition 2.2. *The class $\mathcal{NP}$ of non-deterministic polynomial time problems is defined as the set of all decision problems P to which a yes-answer and a corresponding proof can be provided by a non-deterministic machine in a number of steps that is bounded by a polynomial in the size of the input to P (so-called polynomial time). In case the answer is yes, the proof can be verified in polynomial time by a deterministic machine. The name $\mathcal{NP}$ stands for Non-deterministic Polynomial time.*

> **Example 2.1.** A classic example of an $\mathcal{NP}$-problem is the problem of determining whether a given natural number is either prime or composite. In this case, although it was believed impossible for many years, this problem can be solved in polynomial time by a deterministic machine [AKS04], so it is actually in the subset $\mathcal{P}$ of $\mathcal{NP}$ that contains all problems that can be solved in polynomial time by a deterministic machine.

In less formal terms, the set $\mathcal{NP}$ contains all decision problems to which the proof of a yes-answer can be efficiently verified. Note that the answer is not necessarily efficiently determined; in case it is the problem is in the subset $\mathcal{P}$. One of the most notorious open questions in mathematics is whether $\mathcal{P} = \mathcal{NP}$.

Definition 2.3. *A decision problem $P_1 \in \mathcal{NP}$ can be reduced to another decision problem $P_2 \in \mathcal{NP}$ if, given a solution to P_2, a solution to P_1 can be determined in polynomial time. We also say that P_1 reduces to P_2.*

In other words, if P_1 can be reduced to P_2, it is not harder to solve P_1 than it is to solve P_2.

Definition 2.4. *A decision problem P is called $\mathcal{NP}$-hard if every problem in $\mathcal{NP}$ reduces to P. If also $P \in \mathcal{NP}$, the problem is called $\mathcal{NP}$-complete.*

So, if an algorithm that solves a particular $\mathcal{NP}$-complete problem in polynomial time (on a deterministic machine) is ever found, each problem in $\mathcal{NP}$ can be solved in polynomial time. In that case, the $\mathcal{P} = \mathcal{NP}$ question can be answered 'yes'.

In practice, $\mathcal{NP}$-hardness is proved by reducing a problem that is known to be $\mathcal{NP}$-complete (for instance one of the many problems described in [GJ79]) to the studied problem. The $\mathcal{NP}$-complete problems are considered to be among the most difficult problems, and they are often encountered in algebraic attacks on ciphers.

2.3.2 SAT

An important decision problem in algebraic cryptanalysis is the Boolean satisfiability problem SAT. Boolean variables are variables whose value is either "true" (often denoted by 1) or "false" (denoted by 0). We can then combine several Boolean variables by using the conjunction AND denoted by $\wedge$, the disjunction OR denoted by $\vee$ and the complement NOT(x) denoted by $\overline{x}$. If a formula contains only OR and NOT operators, it is called a clause; a variable or a negation of a variable is a literal. If several clauses are combined using AND operators, the resulting Boolean formula is said to be in CNF (Conjunctive Normal Form).

Definition 2.5. *The satisfiability problem (or SAT) is the problem of, given a Boolean formula in CNF, determining whether there exists an assignment of values to the variables such that the formula holds. If the number of literals per clause is exactly k, we may also refer to the problem as k-SAT.*

Example 2.2. Consider the Boolean formula

$$(x_1 \vee x_2 \vee x_4) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4),$$

which is clearly in CNF. It contains three clauses, and the number of literals per clause is three. Hence, this induces an instance of 3-SAT. The answer in this case is yes, which can be verified by using e.g. $(x_1, x_2, x_3, x_4) = (0, 0, 0, 1)$.

A known result in complexity theory is the following theorem, which is due to Cook [Coo71] and is often referred to as Cook's theorem or the Cook-Levin theorem. We will not formally prove it here, the interested reader is referred to [GJ79] or Cook's original paper.

Theorem 2.1 (**Cook's theorem**)**.** The Boolean satisfiability problem SAT is $\mathcal{NP}$-complete.

In other words, every instance of a problem in $\mathcal{NP}$ can be transformed into a SAT-instance in polynomial time. Also, SAT itself is in $\mathcal{NP}$.

Following Definition 2.4, we can then prove the following theorem about 3-SAT.

Lemma 2.2. 3-SAT is $\mathcal{NP}$-complete.

Proof. Given an instance of SAT, i.e. a Boolean formula in CNF, we can replace each clause by a combination of several clauses of three literals by applying these simple steps to each of the clauses in the original formula:

- If the clause contains three literals, do nothing.
- If the clause contains more than three literals, say $(x_1 \vee \ldots \vee x_t)$ for $t \geq 4$, write the clause as the combination $(x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge \ldots \wedge (\overline{y_{t-3}} \vee x_{t-1} \vee x_t)$, where $y_1, \ldots, y_{t-3}$ are new variables.
- If the clause contains one or two literals, pad it by adding variables that are forced to have the value false. For example, the clause $(x_1 \vee x_2)$ can be replaced by

  $$(x_1 \vee x_2 \vee y) \wedge (\overline{y} \vee \overline{y} \vee \overline{y}).$$

  To satisfy the second clause, the value of y must be set to false, canceling its effect on the first clause.

After all clauses have been checked, each of them has been replaced by an equivalent combination of clauses of three literals. This implies that the resulting formula (which is clearly in CNF) consists of three-literal clauses and is satisfied if and only if the original formula is satisfied. This reduction from SAT to 3-SAT can be performed in polynomial time. Since 3-SAT is a special case of SAT which is in $\mathcal{NP}$, so is 3-SAT. Actually, SAT is even $\mathcal{NP}$-complete by Cook's theorem, which completes the proof. □

2.3.3 Multivariate quadratic equation systems

As mentioned earlier, many modern ciphers can be described by a multivariate quadratic equation system over a finite field. We are now ready to prove the following statement, that supports Shannon's requirement. The proof is due to Garey and Johnson in [GJ79, pp. 251].

Theorem 2.3. The problem MQ of determining whether an arbitrary multivariate quadratic equation system over a finite field $\mathbb{F}_q$ can be solved is $\mathcal{NP}$-complete.

Proof. First, let $q = 2$. Consider an instance of 3-SAT, i.e. a Boolean formula in which each clause contains precisely three literals. We prove the theorem by reducing an instance of 3-SAT to an instance of MQ. To achieve this, we replace the logical operators and values with their algebraic counterparts: AND by multiplication, OR by addition, true by 1 and false by 0. Note that then $\overline{x} = (1 - x)$. Without loss of generality, we can assume that each clause of the given Boolean function is of the form

$$(x_{i_1} \vee x_{i_2} \vee x_{i_3}), \tag{2.1}$$

which translates to the sum

$$x_{i_1} + x_{i_2} + x_{i_3} = 1. \tag{2.2}$$

Although a solution to (2.2) is a solution to (2.1), the converse is not necessarily true: if exactly two of the variables are set to 1, their sum equals 0, but the clause is satisfied. To include all possibilities, we replace each clause by the equations

$$\begin{cases} x_{i_1} + x_{i_2} + x_{i_3} &= x_{i_4} \\ x_{i_1}x_{i_2} + x_{i_2}x_{i_3} + x_{i_1}x_{i_3} &= x_{i_5} \\ x_{i_4} + x_{i_5} + x_{i_4}x_{i_5} &= 1. \end{cases} \tag{2.3}$$

It is an easy task to verify that a solution to (2.1) is a solution to (2.3) and vice versa.

For the general field $\mathbb{F}_q$, it suffices to force the values of the variables to be either 0 or 1. This is accomplished by adding the equations

$$x_i^2 - x_i = 0$$

for each variable x_i. Then, we have determined an instance of MQ that corresponds to the initial instance of 3-SAT, which proves that MQ is $\mathcal{NP}$-hard. The fact that $MQ \in \mathcal{NP}$ is trivial (a provided solution can be easily checked), which concludes the proof. □

Note that the straightforward way to prove a yes-answer to MQ is to provide the actual solution to the system of equations, which can then be easily checked. At this moment this is the only known proof, and we can only solve MQ if we solve the system. For this reason, the problem of actually solving a multivariate quadratic equation system is also referred to as MQ. Formally, this is no longer a decision problem, but it is equivalent to one.

Recall Shannon's requirement that a cipher should employ an encryption function $E_K(p)$ that can be described using a large number of "complex" equations in a large number of variables. Theorem 2.3 suggests that such a system is indeed very hard to solve in the average case, provided the equations are quadratic. However, the fact that decryption must be possible implies that the system of equations describing a particular cipher is not random at all: there has to be some structure that is exploited by the decryption function D_K. Hence, the MQ-instances that describe ciphers contain some additional information that might imply that they are easier to solve than fully random systems. It is this important observation that motivates cryptographic researchers to look for algorithms that find this hidden structure (which is usually not an easy task) and use it to break the corresponding cryptosystem.

Example 2.3. As an example that not every multivariate quadratic equation system is difficult, we note that it is known that 2-SAT is in $\mathcal{P}$. Then, a clause of the general form $(x_{i_1} \vee x_{i_2})$ is described by the single equation

$$x_{i_1} + x_{i_2} + x_{i_1} x_{i_2} = 1$$

over $\mathbb{F}_2$. But, since 2-SAT can be solved in polynomial time, a system of equations of this form is not $\mathcal{NP}$-complete and can be easily solved. Hence, a cipher that can be described using only this type of equation may not be considered secure, as it does not satisfy Shannon's requirement.

2.4 Recent attacks

This section will describe some of the techniques that are currently used to solve MQ and related problems for systems representing ciphers. References are included for readers that are interested in the details of a specific attack. These and further examples are also described in Sections 5.3.2 and 5.4.2

2.4.1 XL

A basic algebraic approach to MQ is the XL-attack suggested by Courtois et al. in 2000 [CKP00]. In XL (eXtended Linearization), the system of equations is expanded by multiplying each of the equations by all monomials (a monomial is the product of certain variables, e.g. $x_1 x_3 x_9$) of a certain degree. The higher this degree, the higher the number of equations, but the resulting system may actually become more efficiently solvable.

The system is then viewed as a matrix, in which each row represents an equation and each column contains the coefficients corresponding to a given monomial. This matrix is then reduced to RREF using Gaussian elimination. If the matrix is approximately square, the solution to the initial problem can then be easily extracted. It was estimated by the authors that this attack is considerably slower than the brute force attack when applied to ciphers such as AES (cf. 5.4), making it impractical. However, further research led to an improved version of the algorithm called XSL.

2.4.2 XSL

One of the main drawbacks of XL is the fact that each equation is multiplied by all monomials of a given degree. In that case, the number of equations explodes quite rapidly, while some of the new equations turn out to be more helpful than others. A refinement of XL, called XSL (eXtended Sparse Linearization) [CP02], carefully selects which equations are to be added to the equation system, thus trying to keep the number of equations within reasonable bounds. Several procedures for selecting these new equations have been studied, and from the most optimistic results the authors conclude that their technique "might be able" to break AES. However, the original paper and therefore the authors' claim is believed to be flawed [Die04]; in practice the algorithm does not behave as promised, and it may even occur that the resulting system can not be solved.

2.4.3 Gröbner basis techniques

A different approach is the use of Gröbner bases [BW93] to solve MQ. Although it goes beyond the scope of this report to properly describe this theory, we mention that a large portion of promising algebraic attacks involves the use of an algorithm for determining a Gröbner basis. At this moment, Faugère's F5 is considered one of the fastest algorithms; it was used to break an instance of the cipher HFE (Hidden Field Equations) in just over 48 hours on a standard computer [FJ03]. In this experiment, 80 quadratic equations in 80 variables over $\mathbb{F}_2$ were solved by adding new equations, representing the system as a huge matrix and converting this matrix into a simple form, much like the XL technique. As an illustration of the magnitude of this problem, the largest matrix encountered when breaking HFE had dimensions $307,126 \times 1,667,009$, which would require about 60 GB to be stored (if no compression is applied).

It is suggested that XL is closely related to some known Gröbner basis attacks [SKI04], which may imply that the attacks are equivalent.

2.4.4 SAT-solvers

The previous attacks are all focused on solving a multivariate equation system that describes a cipher. However, an alternative description of a cipher is as an instance of k-SAT, as is exploited in the proof of Theorem 2.3. A different attack is then to solve the instance of k-SAT that describes a cipher, and many algorithms (called SAT-solvers) have been suggested. Most of these algorithms traverse the space of all possible solutions in an "optimal" way, often based on the Davis-Putnam algorithm [DP60]. The same motivation as for the MQ-solvers applies: although it is believed that random k-SAT is very difficult, the instance describing a typical cipher may be much less complex.

A recent SAT-solver that has drawn quite some interest, mainly due to its elegant description, is MiniSAT [ES03]. The behavior of k-SAT-solvers for small values of k is studied in [Iwa04]. Note that strictly speaking, this is a logical and not an algebraic attack as it is not fully based on the theory of algebra. However, since the problems SAT and MQ are so closely linked, this type of attack is commonly referred to as algebraic nonetheless.

In this thesis, we present a new type of equation that can be used to attack a large class of cryptosystems that satisfy a certain requirement. These equations have some linear properties, which as we will see does not necessarily imply that they are easy to work with. We present a possible algorithm for solving a system of such equations and study its behavior when applied to many different systems of equations. When compared to some known techniques, we believe that this new approach makes an interesting subject for future research.

3

Multiple right-hand side equations

This section will introduce multiple right-hand side equations and describe a general algorithm for solving them. Details of a particular implementation of this algorithm can be found in Chapter 4.

3.1 Systems of algebraic equations

Let $X = \{x_1, \ldots, x_n\}$ be a set of ordered variables with values in $\mathbb{F}_q$. Let $X_i \subset X$ with $|X_i| = k_i \leq k$ for $i = 1, \ldots, M$ and some $k < n$. Further, let $f_i : \mathbb{F}_q^{k_i} \to \mathbb{F}_q$ be functions with variables in X_i (given as multivariate polynomials and/or explicitly by means of function tables). Consider a system of M algebraic equations

$$f_1(X_1) = 0, \ldots, f_M(X_M) = 0 \tag{3.1}$$

whose solution set is defined as

$$\Omega = \{X \in \mathbb{F}_q^n \mid f_i(X_i) = 0 \text{ for all } i = 1, \ldots, M\}.$$

If $q = 2$, the variables and equations are called *Boolean*. As mentioned before, such a system may fully represent a particular cipher.

Although there are numerous methods to solve such systems, this section will focus on the approach suggested by Raddum and Semaev in [RS06, Rad04] and introduced independently by Zakrevskij and Vasilkova in [ZV00]. Their suggestion is not to use the functions f_i explicitly, but to manipulate their individual solutions called *configurations*:

Definition 3.1. *A configuration for an equation $f(X)$ is an assignment of values to the variables X such that $f(X) = 0$.*

Now, in order to extract a solution for Equation (3.1), each function f_i is represented by its corresponding symbol $S_i := S_{f_i}$ as defined below.

Definition 3.2. *A symbol $S_f = (X, L)$ corresponding to a function f consists of an ordered set of variables $X = X(S_f)$ and a list $L = L(S_f)$ of all configurations for $f(X)$.*

The set $L_i = L(S_i)$ is commonly referred to as the (algebraic) variety of f_i in $\mathbb{F}_q^{k_i}$ [BW93, pp. 312].

Without using the explicit form of f_i, the list $L(S_i)$ of configurations can only be compiled by an exhaustive search. So, for this approach to be practical it is required that the number of variables per equation, i.e. k, is relatively small. For equation systems describing block ciphers this is generally the case.

Example 3.1. Consider the functions $f_i : \mathbb{F}_2^4 \to \mathbb{F}_2$ given by

$$f_1(x_1, x_2, x_3, x_4) = x_1x_2 + x_2x_4 + 1,$$
$$f_2(x_1, x_2, x_3, x_4) = x_1 + x_1x_3x_4.$$

We may then attempt to solve the joint problem of finding a simultaneous root of f_1 and f_2 directly (e.g. by using a Gröbner-basis method), but instead we describe the equations by their corresponding symbols:

$$X(S_1) = (x_1, x_2, x_4),$$
$$X(S_2) = (x_1, x_3, x_4),$$
$$L(S_1) = \{(0, 1, 1), (1, 1, 0)\},$$
$$L(S_2) = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 1, 1)\}.$$

It is important to note the variables that are present in these configuration lists. For instance, although $(0, 1, 1)$ is an element of both configuration lists, it represents different variables, indicated by $X(S_1)$ and $X(S_2)$, respectively.
Now, from the first symbol it follows that any simultaneous root of f_1 and f_2 must satisfy $x_1 + x_4 = 1$ (in this particular case, a condition in the form of a linear equation can be extracted, but this is not generally true). Now, any configuration in $L(S_2)$ that does not satisfy this condition can not contribute to a solution of the joint problem, since it contradicts the first symbol, and may therefore be disregarded. In this case, all but the second and fourth elements of $L(S_2)$ can be deleted.
After this step the configuration lists are

$$L(S_1) = \{(0, 1, 1), (1, 1, 0)\},$$
$$L(S_2) = \{(0, 0, 1), (0, 1, 1)\}.$$

Obviously, the sets of involved variables $X(S_1)$ and $X(S_2)$ remain unchanged. Then, the second element of $L(S_1)$ can be deleted, since it coincides with no configuration for S_2. Finally, no more configurations can be deleted; the symbols are said to agree (cf. Section 3.3.2). Indeed, the final configuration lists correspond directly to the solutions of the joint problem:

$$L(S_1) = \{(0, 1, 1)\},$$
$$L(S_2) = \{(0, 0, 1), (0, 1, 1)\},$$

which correspond to the joint solutions $(0, 1, 0, 1)$ and $(0, 1, 1, 1)$.

The example illustrates the main idea behind the (pairwise) agreeing procedure described in [RS06]: to sequentially delete configurations in S_i that do not coincide with any of the configurations in S_j. If the joint problem has a unique solution (which is highly probable for the equation systems that describe the ciphers we will consider, cf. Chapter 5), this solution can be easily extracted if each configuration list has exactly one element. When more than two symbols are considered, configurations can be deleted by agreeing each pair of symbols until all pairs agree. If at any point in the algorithm one of the symbols contains an empty configuration list, no solution exists. In any other case, no conclusion can be

drawn and another procedure has to be applied to solve the system. The straightforward procedure to achieve this is to "join" together two symbols by combining their configuration and variable lists into larger lists, effectively considering a new symbol. This new symbol then replaces the two original ones, and the agreeing procedure can be applied again to possibly delete obsolete configurations, etc.

An in-depth analysis of this approach for solving sparse (i.e. depending on a low number of variables) algebraic equation systems can be found in [Sem07]. This thesis will, however, focus on a variant that uses multiple right-hand side equations, as proposed in [RS07]. Although there is a simple correspondence between both problem descriptions (i.e. given an algebraic equation $f(X) = 0$ and a particular solution $\bar{X}$, there exists a MRHS equation S to which $\bar{X}$ is a solution, and vice versa, cf. 3.2.1), the analysis of the latter is a bit more intuitive as it exploits known results from linear algebra. For example, we will be able to define a very elegant procedure for agreeing a pair of multiple right-hand side equations.

3.2 Definition

Let q be a prime power and let X be a set of n variables, represented as a column vector. A multiple right-hand side equation is defined as follows:

Definition 3.3. *An equation of the form*

$$S : AX = [L],$$

with A a $k \times n$ matrix of full rank and L a set of s column vectors of length k, represented as a $k \times s$ matrix, is a multiple right-hand side (MRHS) equation. Its solution set is given by

$$\Omega_S = \bigcup_{j=1}^{s} \{X \in \mathbb{F}_q^n \mid AX = l_j\} \tag{3.2a}$$

$$= \{X \in \mathbb{F}_q^n \mid AX \in L\}, \tag{3.2b}$$

where $l_1, \ldots, l_s$ (the "right-hand sides") are the elements of L. The matrix A may be referred to as the coefficient matrix of S.

Note that the matrix L is written in square brackets to avoid possible confusion with a classical matrix (i.e. L does not represent a linear transformation in any sense). It is important to realize that it is an ordered set of columns, merely represented as a matrix. We will, however, refer to an element of L as a column of L, as in Equation (3.2b), and denote it as, for instance, L^1. If $s = 1$, an ordinary system of linear equations is obtained, and the brackets may be omitted. For completeness, we stress that there are three types of equations used in this thesis.

- Algebraic equations, i.e. equations of the form $f(X) = 0$, where f is an algebraic function.
- Linear equations, which are algebraic equations for which f is a linear function.
- MRHS equations, as in Definition 3.3.

Note that linear equations are considerably easier to solve than the other two types.

We will alternatively refer to a MRHS equation $S : AX = [L]$ as a symbol. Note that if each row of A has exactly one non-zero element, the corresponding MRHS equation can be

regarded as a symbol as described in Definition 3.2. Unless stated otherwise, the dimensions of matrix A are $k \times n$, with $k \leq n$. The dimensions of L are $k \times s$. Also we will assume that elements of L have multiplicity 1 (i.e. L consists of unique elements), which is reasonable since a duplicate element does not contain any additional information. Similarly, note that the coefficient matrix A has full rank, so the linear system $AX = l$ can be solved for any column $l \in L$.

3.2.1 MRHS vs. algebraic equations

Two equations are said to be equivalent if their solution sets are equal. As mentioned, given an algebraic equation we can determine an equivalent MRHS equation, and vice versa. To prove this, we introduce, given a function $f : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q$, the set $G(f)$ as

$$G(f) = \{b \in \mathbb{F}_q^n \mid f(X + b) = f(X) \text{ for all } X \in \mathbb{F}_q^n\}.$$

It is an easy exercise to check that $G(f)$ is actually a subspace of $\mathbb{F}_q^n$, for which we have to ensure that it is closed under addition and scalar multiplication, cf. Section 1.2:

- Let $b_1, b_2 \in G(f)$, then for all $X \in \mathbb{F}_q^n$ it holds that

  $$f(X + b_1 + b_2) = f(X + b_1) = f(X),$$

  where the first equality holds since $b_2 \in G(f)$ and $X + b_1 \in \mathbb{F}_q^n$.

- Let $b \in G(f)$ and $\alpha \in \mathbb{F}_q$, then for all $X \in \mathbb{F}_q^n$ it holds that

  $$\begin{aligned} f(X + \alpha b) &= f(X + (\alpha - 1)b + b) \\ &= f(X + (\alpha - 1)b) \\ &= f(X + (\alpha - 2)b) \\ &\vdots \\ &= f(X + b) \\ &= f(X). \end{aligned}$$

Because it is a space, $G(f)$ has a dimension, say $m \leq n$, and a basis $\{b_1, \ldots, b_m\}$. Let $\{a_1, \ldots, a_k\}$ be an independent set of $k = n - m$ vectors that satisfy $\langle a_i, b_j \rangle = 0$ for all i and j. Denote the $k \times n$ matrix whose rows are $a_1, \ldots, a_k$ by A, so for all j it holds that $Ab_j = \underline{0}$. Finally, define g as

$$g(Y) = f(X)$$

for $Y = AX$. We then prove the following statement

Lemma 3.1. The function g is properly defined and $f(X) = g(AX)$.

Proof. Let $Y \in \mathbb{F}_q^k$ and consider $g(Y)$. Since the matrix A possibly consists of more columns than rows, there may exist X_1 and X_2, with $X_1 \neq X_2$, such that

$$AX_1 = AX_2 = Y,$$

in which case it is not clear whether $g(Y) = f(X_1)$ or $f(X_2)$. So, for arbitrary matrices A, the function g is not properly defined. However, this particular choice of A implies that $f(X_1) = f(X_2)$, which ensures that g is a function.

To see this, consider distinct X_1 and X_2 for which $AX_1 = AX_2$, and observe that

$$\begin{aligned} AX_1 = AX_2 &\Leftrightarrow A(X_1 - X_2) = 0 \\ &\Leftrightarrow X_1 - X_2 \in G(f), \end{aligned}$$

since $G(f)$ is the space whose elements b have the property that $Ab = \underline{0}$ (it is called the nullspace of A). But then

$$f(X_1) = f(X_1 + X_2 - X_2) = f(X_2 + (X_1 - X_2)) = f(X_2),$$

which finishes the proof, since $f(X) = g(AX)$ is obvious. □

Let us look at this lemma a bit closer. It states that for any not necessarily linear function f in n variables, we can determine a function g in $k \leq n$ variables and a linear function $\mathcal{A} : \mathbb{F}_q^n \to \mathbb{F}_q^k$ (defined by $\mathcal{A}(X) = AX$) such that $f(X) = g(\mathcal{A}(X))$. So if $k < n$, instead of processing the function f directly, we may split this function into a linear function $\mathcal{A}$ and a non-linear function g in less variables than f, and focus on these functions instead. If the described method for constructing A and g is used, it is ensured that the number of variables in g is minimal. Splitting a function into a linear part and a non-linear part is illustrated in Figure 3.1.

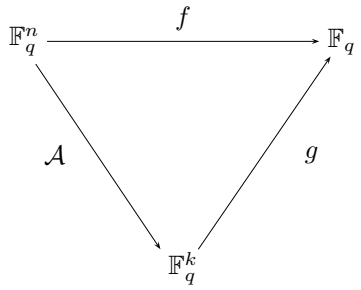

Figure 3.1: Splitting a function f.

Essentially, Lemma 3.1 describes the approach we suggest in this thesis. Instead of analyzing ciphers using a representation in (mostly non-linear) algebraic equations, we split these equations into a linear part (the coefficient matrices) and a non-linear part (the right-hand side matrices) and analyze these alternative functions. In practice, as we will describe in Section 5.1, the coefficient and right-hand side matrices that represent ciphers can be easily determined without using this inefficient construction, i.e. we do not need the representation in algebraic equations to construct the corresponding MRHS equations.

We can now prove the following theorems that link algebraic equations to MRHS equations and vice versa.

Theorem 3.2. Given a function $f : \mathbb{F}_q^n \to \mathbb{F}_q$, there exists a symbol S that is equivalent to the equation $f(X) = 0$.

Proof. A trivial equivalent symbol is obtained if we use the identity matrix I_n as the coefficient matrix A and the matrix consisting of the elements $X \in \mathbb{F}_q^n$ that satisfy $f(X) = 0$ as the right-hand side matrix L. However, this representation has n rows. If we apply Lemma 3.1, we may find a more efficient representation with $k < n$ rows.

Following the described construction, we determine the space $G(f)$ and its dimension m. If $m = n$ the trivial representation is optimal: we can not describe the equation using less rows. But if $m < n$, we determine the matrix A and the function g. Then, let L be the variety of the function g, i.e. the set consisting of all vectors $Y \in \mathbb{F}_q^k$ for which $g(Y) = 0$. Then it holds that $f(X) = 0$ if and only if $AX = [L]$. This follows from the simple observation that

$$\begin{aligned} f(X) = 0 &\Leftrightarrow g(AX) = 0 \\ &\Leftrightarrow AX \in L \\ &\Leftrightarrow AX = [L], \end{aligned}$$

where the last equivalence follows from Definition 3.3. Note that this representation has only k rows. This equivalence is illustrated in Example 3.2. □

Theorem 3.3. Given an MRHS equation $S : AX = [L]$ there exists an algebraic function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ such that the equation $f(X) = 0$ is equivalent to S.

Proof. Consider the MRHS equation $AX = [L]$. Define the algebraic function $g : \mathbb{F}_q^k \to \mathbb{F}_q$ by means of a function table:

$$g(Y) = \begin{cases} 0 & \text{if } Y \text{ is a column of } L, \\ 1 & \text{otherwise,} \end{cases}$$

and define the function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ as

$$f(X) = g(AX).$$

Then $f(X) = 0$ is an algebraic equation in n variables, and any solution to $f(X) = 0$ is a solution to S and vice versa. The proof of this statement is identical to that in the proof of Theorem 3.2. □

In fact, we can set $g(Y)$ equal to any non-zero element of $\mathbb{F}_q$ in case Y is not a column of L, and the resulting function will still be equivalent to the symbol S. This implies that the construction for transforming a symbol into an algebraic equation is not generally bijective, unless $q = 2$. Similarly, several MRHS equations correspond to a given algebraic equation, depending on the particular choice for the matrix A.

Example 3.2. Following (for $q = 2$) the constructions described in the proof of Theorem 3.2, the function

$$f_2(x_1, x_3, x_4) = x_1 + x_1x_3x_4$$

of Example 3.1 corresponds to the trivial MRHS equation given by

$$S_2' : \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

in the four variables $X = (x_1, x_2, x_3, x_4)$. But, it is also equivalent to the smaller symbol

$$S_2 : \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The all-zero column in the coefficient matrix indicates that the corresponding algebraic equation is independent of x_2.
Similarly, if

$$f_1(x_1, x_2, x_4) = x_1x_2 + x_2x_4 + 1$$

from Example 3.1 is considered, the resulting MRHS equation has a unique right-hand side:

$$S_1 : \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

in other words the linear equations $x_1 + x_4 = 1$ and $x_2 = 1$ are obtained. Note that this symbol corresponds to the factorization of f_1 described by

$$f_1(x_1, x_2, x_4) = x_2(x_1 + x_4) + 1.$$

Analogously, the procedure described in the proof of Theorem 3.3 can be used to determine the algebraic equation that corresponds to the MRHS equations S_1 and S_2. Since $q = 2$, these algebraic equations are uniquely defined and we obtain the function tables for f_1 and f_2, respectively.
Since the number of variables in these cases is very low, the equivalences are easily verified.

3.3 Systems of MRHS equations

From Section 1.5, we recall that a system of (ordinary) linear equations is consistent if at least one simultaneous solution exists. Consider a symbol $S : AX = [L]$. Since we have assumed that A has full rank, the system $AX = l$ is consistent for all $l \in L$. Also, if the rank of A is denoted by $\rho(A)$ and $s = 1$, the symbol S (which, in this case, is a system of linear equations) has $q^{n-\rho(A)}$ solutions. Hence, for a general number of s right-hand sides, the number of solutions to S is given by

$$\mid \Omega_S \mid = sq^{n-\rho(A)} = sq^{n-k},$$

which can be a very large number. However, if we consider M symbols simultaneously, i.e. the system

$$S_1 : A_1X = [L_1], \ \ldots, \ S_M : A_MX = [L_M] \tag{3.3}$$

whose solution set is given by

$$\Omega = \bigcap_{i=1}^{M} \Omega_{S_i},$$

the number of solutions will be considerably lower. In fact, as mentioned before, for equation systems derived from ciphers it is reasonable to assume that there is a unique solution, in which case $\mid \Omega \mid = 1$. From now on, this assumption shall be made without mention.

To extract the solution to (3.3), this section will introduce four procedures to manipulate a system of MRHS equations. Some suggested implementations of these procedures and an analysis of their corresponding time and memory requirements can be found in Chapter 4.

Some words on notation: from now on, the dimensions of A_i are $k_i \times n$, and the dimensions of L_i are $k_i \times s_i$. Both matrices contain elements of $\mathbb{F}_q$. The number of rows k_i is upper-bounded by $k < n$. Since L_i contains unique elements, the number of right-hand sides s_i is at most $q^{k_i} \leq q^k$.

3.3.1 Procedure 1: Extracting linear equations

In order to solve a system of symbols in n variables, the goal is to obtain an equivalent system of n independent linear equations. Assuming a unique solution exists, this solution can then easily be found by solving this extracted system of linear equations. Obviously, if any symbol S_i contains only one right-hand side, $\rho(A_i) = k_i$ linear equations can be directly read from the symbol as illustrated by Example 3.2. But there is another method to extract less obvious linear equations from a single symbol.

Consider symbol $S_i : A_iX = [L_i]$. From Equation (3.2a) it follows that a symbol can be solved by subsequently solving the systems of linear equations $A_iX = L^l$, where l runs through the set $\{1, \ldots, s_i\}$. Note that for any vector $t \in \mathbb{F}_q^k$

$$A_iX = L^l \;\Rightarrow\; tA_iX = tL^l$$

which can be translated into (using Definitions 3.2a and 3.2b)

$$A_iX = [L_i] \;\Rightarrow\; tA_iX = [tL_i]. \tag{3.4}$$

We will now use the following definition.

Definition 3.4. *The k-row matrix A is said to generate the vector v if there exists a non-zero $t \in \mathbb{F}_q^k$ such that $tA = v$. Alternatively, v is said to be generated by A.*

Now, let X be a solution to the symbol S_i and assume L_i generates the vector $r \cdot \underline{1} = (r, \ldots, r)$, with $r \in \mathbb{F}_q$. Then there exists a $t \in \mathbb{F}_q^k \backslash \{\underline{0}\}$ such that

$$tA_iX = [r \cdot \underline{1}]$$

by Equation (3.4), which implies the linear equation $tA_iX = r$. This linear equation forms a necessary, but not generally sufficient condition for X (cf. Example 3.3).

Note that several independent equations may be extracted satisfying $r = 0$ (i.e. homogeneous equations), but that extracting one equation for $r \neq 0$ suffices (a non-homogeneous equation). Every non-homogeneous equation (with arbitrary r) can then be expressed as a linear combination of the homogeneous equations and the selected non-homogeneous one. If $k_i \geq s_i$, at least one linear equation can be extracted, since in that case $\underline{0}$ is generated by L_i.

The generation of all-r vectors can be extended to formulate a procedure that is more probabilistic. This is discussed in Section 6.2.1.

Example 3.3. Consider the symbol over $\mathbb{F}_2$ given by

$$S_i : \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} X = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and note that L_i generates both $\underline{0}$ (for $t_1 = (1, 0, 1, 0)$) and $\underline{1}$ (for $t_2 = (1, 1, 0, 1)$). Then, by considering t_1A_iX and t_2A_iX respectively, the linear equations $x_3 = 0$ and $x_3 + x_4 + x_5 = 1$ can be extracted.
It is an easy check that all vectors $(x_1, \ldots, x_5)$ that satisfy S_i also satisfy these linear equations. The converse, however, is not true. For instance, the vector $(0, 0, 0, 0, 1)$ satisfies both linear equations, but

$$A_i(0, 0, 0, 0, 1)^T \;=\; (1, 1, 1, 1)^T \notin L_i.$$

After a linear equation has been extracted, one involved variable becomes "fixed" (either its value or its dependence on a number of other variables), so it may be disregarded. Although there is a more direct method to implement this elimination, the method proposed in this thesis uses a procedure that has not been introduced yet. Therefore, we will return to this feature in Section 3.3.4.

3.3.2 Procedure 2: Agreeing

In Section 3.2.1 we showed the connection between MRHS and "ordinary" (algebraic) equations. Naturally, this connection extends to systems of equations, and in the introduction to this chapter we already briefly mentioned the method for solving a system of algebraic equations by removing all information that does not contribute to the solution of the system. In this section, a simple yet effective MRHS-version of this method is described.

Consider a pair of symbols

$$S_i : A_i X = [L_i] \text{ and } S_j : A_j X = [L_j],$$

with $1 \leq i, j \leq m$ and $i \neq j$. We then define (pairwise) agreement as follows:

Definition 3.5. *Two symbols S_i and S_j agree if and only if*

- *for all $l_1 \in L_i$ there exists an $l_2 \in L_j$ such that the simultaneous linear system*

$$A_i X = l_1 \text{ and } A_j X = l_2 \tag{3.5}$$

  is consistent, and

- *for all $l_2 \in L_j$ there exists an $l_1 \in L_i$ such that Equation* (3.5) *is consistent.*

Alternatively, we say that S_i and S_j are in agreement, or that S_i agrees with S_j or vice versa[2]*. In any other case the symbols disagree or are in disagreement.*

Now, suppose there exists an $l_1 \in L_i$ such that $S_i' : A_i X = l_1$ and S_j do not agree. In other words, there is no assignment of values to the variables X such that both S_i' and S_j are satisfied. Then l_1 can be deleted from L_j, since it does not contribute to the solution of Equation (3.3). Similarly, any column in L_j that causes disagreement can be deleted.

The procedure of comparing two symbols while deleting columns that cause disagreement forms the second procedure, called (pairwise) agreeing. Note that after deleting a column from L_i, say, there may be a third symbol S_k that no longer agrees with S_i, in which case either another column from L_i or a column from L_k can be deleted. Continuing this procedure, there may be a "chain reaction" of deleted columns. If at some point all right-hand sides of a particular symbol have been deleted, the initial system has no solution.

Example 3.4. Consider the two symbols over $\mathbb{F}_2$ of Example 3.2, given by

$$S_1 : \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and

$$S_2 : \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

[2]Which is an equivalent statement, since agreement clearly is a symmetric property.

Then S_1 states that a simultaneous solution to S_1 and S_2 must satisfy $x_1 + x_4 = 1$ and $x_2 = 1$. Since S_2 is independent of x_2, this second linear equation will not cause the deletion of any columns in L_2, but the first equation will.
To realize this, consider $S_2' : A_2X = l_2$ where l_2 is the first column of L_2. Then any solution to S_2' satisfies $x_1 = x_4 = 0$ and can therefore not satisfy the constraints that are induced by S_1. We can not expect to find a simultaneous solution to S_1 and S_2', and column $l_2 \in L_2$ can be deleted.
It is an easy exercise to continue this process and conclude that all but the third and fourth column of L_2 can be deleted to obtain the new symbol $\hat{S}_2$, given by

$$\hat{S}_2 : \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

This implies the additional (i.e. independent of the two extracted earlier) linear equation $x_1 = 0$, or, alternatively, $x_4 = 1$. Both symbols now agree, and no fourth linear equation can be extracted. So, the solution set corresponding to the initial system S_1, S_2 is described by the linear equations

$$\begin{cases} x_1 + x_4 = 1 \\ x_2 = 1 \\ x_1 = 0 \end{cases}$$

The analogy with the agreeing procedure illustrated in Example 3.1 is obvious.

Note that if S_i and S_j effectively operate on disjoint variable sets (i.e. the joint coefficient matrix $\binom{A_i}{A_j}$ has full rank), no columns can be deleted, regardless of the contents of L_i and L_j.

If the agreeing procedure alone is not powerful enough, e.g. if there is insufficient overlap among the coefficient matrices or if the number of right-hand sides is simply too large, deletions may be incited by extracting linear equations (and consequently, eliminating variables) and removing right-hand sides that do not satisfy these equations. If at any point a linear equation is extracted that contradicts one or more equations found earlier, we conclude that a solution to the system does not exist.

3.3.3 Procedure 3: Gluing

In practice, the solution to Equation (3.3) can not be found by consecutively agreeing each pair of symbols and extracting linear equations alone. In order to delete additional columns, multiple (i.e. more than two) symbols have to be compared simultaneously to find inconsistencies. The simplest way to achieve this is by joining two symbols S_i and S_j into a single new symbol $S_{ij} = S_i \circ S_j$ (called the gluing of S_i and S_j) which contains all information contained in both S_i and S_j. So the solution set of S_{ij} is given by

$$\Omega_{S_{ij}} = \Omega_{S_i} \cap \Omega_{S_j}, \tag{3.6}$$

for $1 \leq i, j \leq m$ and $i \neq j$. If the two symbols S_i and S_j are replaced by the single symbol S_{ij}, agreeing the updated system

$$S_{ij}, S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_{j-1}, S_{j+1}, \ldots, S_M$$

(which now contains one symbol less) may be useful. If not, another pair of symbols can be glued to enhance deletions, etc. Since any solution to S_i and S_j is a solution to the gluing

S_{ij}, any solution to the updated system is a solution to the initial system. Hence, no vital information is lost by gluing.

The details of a basic gluing procedure are rather straightforward. The coefficient matrix A_{ij} of S_{ij} is given by $A_{ij} = \binom{A_i}{A_j}$. Note that there is a distinct probability that A_{ij} contains duplicate or, at least, linearly dependent rows, a property that will be exploited in the detailed implementation described in Chapter 4. For now, A_{ij} will remain in this simple, possibly overdetermined form. The set of right-hand sides L_{ij} is formed by concatenating all columns of L_i and L_j, so

$$L_{ij} = \{(l_i \mid\mid l_j) \mid l_i \in L_i, l_j \in L_j\}.$$

This fully describes both symbols S_i and S_j, and Equation (3.6) is easily verified. Using this approach, the dimensions of the coefficient matrix A_{ij} are $(k_i + k_j) \times n$ and the dimensions of L_{ij} are $(k_i + k_j) \times s_i s_j$.

However, the number of right-hand sides of S_{ij}, denoted by s_{ij}, can be greatly reduced by observing inconsistencies: if the joint system of linear equations described in Equation (3.5) is inconsistent for a certain $l_1 \in L_i$ and $l_2 \in L_j$, the vector (l_1, l_2) can be excluded from L_{ij} since it does not describe a common solution to S_i and S_j. Hence, s_{ij} satisfies $s_{ij} \leq s_i s_j$.

When gluing, the same restrictions as for agreeing apply: only if A_{ij} has non-full rank we can expect to find inconsistencies. Otherwise the number of right-hand sides will achieve the upper bound $s_{ij} = s_i s_j$. Obviously, gluing S_i and S_j also agrees them: if $l_1 \in L_i$ such that no $l_2 \in L_j$ exists for which Equation (3.5) is consistent, l_1 will not appear in the set of right-hand sides of S_{ij} and effectively is deleted. Gluing is a bit more powerful though, as it even describes which choices of (l_1, l_2) result in a consistent linear system.

Example 3.5. Consider the symbols S_1 and S_2 introduced in Example 3.2. Then the straightforward gluing of S_1 and S_2 is given by

$$S_{12} : \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},$$

from which the joint solutions to S_1 and S_2 can be directly extracted. Note that we effectively glue the symbols S_1 and $\hat{S}_2$ of Example 3.4.
As mentioned earlier, the fact that this coefficient matrix is overdetermined is not exploited in this basic procedure.

It is important to realize that even though gluing produces very specific results, it is a rather expensive procedure in terms of memory requirements and time complexity, as the number of right-hand sides can increase drastically. In practice, gluing is only affordable if the resulting number of right-hand sides is below some fixed threshold, mainly determined by the available amount of system memory and desired running time. This will be further discussed in Section 4.3.

3.3.4 Procedure 4: Guessing and eliminating variables

If all symbols agree and gluing is too expensive (i.e. there are no two symbols whose gluing "fits" in the available system memory), another procedure is needed to enhance deletions of columns by the agreeing algorithm. The simplest method to achieve this is by guessing the value of some variable(s).

In essence, guessing the values of the variables $x_{i_1}, \ldots, x_{i_N}$ introduces the linear equations $x_{i_r} = v_{i_r}$, where $v = (v_{i_1}, \ldots, v_{i_N}) \in \mathbb{F}_q^N$ are the values of the guesses[3]. These linear equations can be regarded as an N-row symbol S_0 with a single right-hand side v (viewed as a column vector). In fact, this newly formed symbol can also be used to store any linear equation that is extracted from the system. If the symbol S_0 is then glued to a symbol S_i, only those columns of L_i that do not contradict the linear equations contained in S_0 survive. Since s_0 (the number of right-hand sides of S_0) equals one, this gluing introduces an efficient method for eliminating variables, once a linear equation has been extracted (cf. 3.3.1). In practice, we only need to glue new rows of S_0, which further improves the efficiency of this elimination procedure.

Using the symbol S_0, an algorithm for solving Equation (3.3) should run until one of the following conditions is true:

- **C1.** One of the symbols $S_1, \ldots, S_M$ has no right-hand sides;
- **C2.** An extracted linear equation contradicts one or more of the equations induced by S_0;
- **C3.** The symbol S_0 contains n linearly independent equations.

If C1 or C2 is true, the system has no solution. If C3 holds, any new rows of the symbol S_0 are glued to all remaining symbols, and if the resulting system does not satisfy C1, the solution to the initial system of MRHS equations can be easily extracted by solving the equations contained in S_0. Otherwise we can conclude that the system has no solution.

3.4 The algorithm SOLVEMRHS

The four described procedures suffice to define an algorithm for solving any system of MRHS equations. The algorithm used in this thesis is adapted such that it always outputs the solution, which exists by assumption. It is depicted in Algorithm 3.1.

Here, *changesmade* is a Boolean variable (initial value true) that is set to true if the previous iteration of the main loop changed the system (i.e. if at least one column was deleted, a new linear equation was extracted or two symbols were glued), and false otherwise. The procedure *LinSolve* is a procedure for solving a system of linear equations (e.g. using Gaussian elimination).

Only when the main loop fails to alter the system one variable is guessed. We will see that the number of guesses needed to solve the system is affected by which particular variables are guessed and in which order. If after t guesses either C1 or C2 holds, it can be concluded that the guess was wrong (since we assume the initial system can be solved uniquely). In that case the system is reset to the situation after $t-1$ guesses and another t-th guess is made. If none of these t-th guesses produce a solution, the $(t-1)$-th guess is rejected, etc. Using this method, a q-ary deterministic search tree with backtracking is traversed until the solution is found [Woo92].

[3] The same analysis applies if not the value of a single variable but of a linear combination of variables is guessed, i.e. to guesses of the general form $\sum_{j=1}^{n} \alpha_j x_j = v_{i_r}$, where the $\alpha_j \in \mathbb{F}_q$ are not all equal to zero.

Algorithm 3.1: The algorithm SOLVEMRHS.

Input: A system of MRHS equations $S_1, \ldots, S_M$
Output: The solution

```
repeat
    while changesmade do
        // begin main loop
        agree all symbols
        extract new linear equations and add to S_0
        glue new rows of S_0 to all symbols
        glue one pair of symbols (if possible)
        // end main loop
    guess one variable
until C1 or C2 or C3

if C3 then
    glue new rows of S_0 to all symbols
if C1 or C2 then
    reject guess; backtrack
else
    LinSolve(S_0)
```

4

Procedure details and analysis

After having properly introduced multiple-right hand side equations, this chapter will suggest and analyze implementations of the described procedures for solving them. These implementations were used to obtain the results described in Chapter 5. An example is included at the end of each section to clarify the details of the procedures. The implementations are generalizations of those previously described in [RS07].

A word on notation. In the following, the r-row identity matrix is denoted by I_r and $\mathcal{O}$ indicates the all-zero matrix, whose dimensions follow from the context. For $r > 0$, the submatrix of A formed by its first r rows is denoted by $A^{(r)}$, whereas $A^{(-r)}$ consists of the last r rows of A. The all-zero vector is denoted by $\underline{0}$, the all-one vector by $\underline{1}$ and in general $r \cdot \underline{1} = (r, \ldots, r)$. As before, the number of variables is n and the dimensions of A_i are $k_i \times n$, where $k_i < k$ for some k. The dimensions of L_i are $k_i \times s_i$.

4.1 Procedure 1: EXTRACTLINEAR

The first procedure EXTRACTLINEAR attempts to extract ordinary linear equations (i.e. linear equations with a single right-hand side) from a given symbol. If successful, one involved variable per extracted equation can be eliminated.

Consider a single symbol $S : AX = [L]$. The main step in extracting linear equations from S is to determine a maximal set of independent vectors $t_1, \ldots, t_N$ such that

$$t_l L = \underline{0} \tag{4.1a}$$

for $l = 1, \ldots, N-1$ and

$$t_N L = r \cdot \underline{1} \tag{4.1b}$$

for some $r \neq 0$. As mentioned in section 3.3.1, any vector $r' \cdot \underline{1} = (r', \ldots, r')$ is then generated by L using a vector t' that can be written as a linear combination of $t_1, \ldots, t_N$. Once the solutions to Equations (4.1a) and (4.1b) are known, the linear equations induced by S are easily extracted.

The suggested implementation is Algorithm 4.1 outlined below. Note that if $s = 1$, the linear equations can be read out from the symbol directly and the algorithm is not required. In this algorithm, Equation (4.1b) is further specified by setting $r = 1$.

Algorithm 4.1: EXTRACTLINEAR(S).

Input: A symbol $S : AX = [L]$ with $s > 1$
Output: The set of linear equations induced by S

concatenate A and L (row-wise) to create $B = (L \mid\mid A)$
$B' = (L' \mid\mid A') = Gauss(B)$
$l = k$
while row l of L' equals $\underline{0}$ **do**
    `// output homogeneous equations`
    return row l of A'
    $l = l - 1$
if $\sum_{j=1}^{l}$ row j of $L' = \underline{1}$ **then**
    `// output non-homogeneous equation`
    return $\sum_{j=1}^{l}$ row j of A'

Here $Gauss$ is an implementation of the Gaussian elimination procedure with backward substitution. Note that the dimensions of L' and A' equal those of L and A, respectively. Further, note that EXTRACTLINEAR outputs equations as vectors of coefficients, i.e. the linear equation $x_1 + 2x_4 = 0$ in the variables (x_1, x_2, x_3, x_4) is returned as $(1, 0, 0, 2)$ in the first part of the algorithm.

The algorithm initializes by forming the matrix B, which consists of corresponding rows of L and A. This matrix is then transformed into reduced row echelon form (RREF). A direct consequence of the RREF of B' is that L generates the all-zero vector only if the portion of B' that corresponds to L (named L' in the algorithm, which is also in RREF) has all-zeroes in its last rows. Hence, starting at the bottom, the coefficient lists of all homogeneous equations can be directly read from A', as long as the corresponding row of L' contains only zeroes.

After using the bottom rows to determine the homogeneous linear equations from B', we can use the remaining rows in the second part of the algorithm. At this point, the value l indicates the highest index such that L'_l is non-zero. Naturally, if L generates $\underline{1}$, so does L' and vice versa. But since L' is in RREF, this property is very easy to check: if L' generates $\underline{1}$, the rows of L' must sum to $\underline{1}$. If this property holds, the corresponding linear equation can be easily determined by summing the appropriate rows of A'.

This gives a simple construction for extracting linear equations from the symbol S, by transforming its right-hand side matrix L into reduced row echelon form. If a linear equation is extracted from a symbol, one involved variable can be eliminated. We will return to the details of elimination of variables later.

It is known that the Gaussian elimination of an $l \times m$ matrix requires $O(l^2 m)$ operations [Mey01]. Although there are more efficient algorithms available, these are not employed in this thesis [Str69, CW87]. Since the time complexity of the other steps in Algorithm 4.1 is obviously lower than that of the Gaussian elimination step, we obtain (after substituting the appropriate parameters)

$$O(k^2(s + n))$$

as the time complexity of EXTRACTLINEAR. There is no significant memory requirement for

this procedure, other than the memory required for storing the involved symbols.

Example 4.1. Consider for $q = 2$ the symbol $S : AX = [L]$ given by

$$S : \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

We then run EXTRACTLINEAR(S) to determine which (if any) linear equations are induced by S. To this end we create the matrix B as

$$B = \left(\begin{array}{cc|ccccc} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$$

and reduce this into RREF to obtain

$$B' = \left(\begin{array}{cc|ccccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right) = \left(L' \mid\mid A'\right).$$

It immediately follows that L generates $\underline{0}$, and the first part of the algorithm will produce the last row of A', which represents the linear equation $x_1 + x_3 = 0$. The second part of the algorithm will produce the vector $(0, 1, 0, 1, 0)$, which represents the non-homogeneous linear equation $x_2 + x_4 = 1$, as is implied by adding the first and second rows of B'. Note that in this simple case, this linear equation could have been read from the second row of B directly.

4.2 Procedure 2: AGREE

This section will introduce the procedure AGREE, which pairwise agrees a set of symbols. To this end, several subprocedures are introduced to facilitate the implementation. In particular, we are able to improve the performance of the algorithm in case the underlying field is $\mathbb{F}_2$.

4.2.1 The function $U(A)$

As mentioned in section 3.3.2, agreeing symbols S_i and S_j can only lead to the deletion of columns if the joint coefficient matrix A defined by

$$A = \begin{pmatrix} A_i \\ A_j \end{pmatrix}$$

has non-full rank, i.e. if $\rho(A) < \kappa$, where $\kappa = k_i + k_j$. To exploit this fact, we use a function $U(A)$ that returns a square, non-singular matrix U, such that

- **U1.** UA has exactly $\rho(A)$ independent non-zero rows, and
- **U2.** any zero row of UA is below all its non-zero rows.

The suggested implementation of $U(A)$ is Algorithm 4.2.

The function $Gauss'(A)$ is the first part of the function $Gauss$ used before, i.e. we apply Gaussian elimination but not backward substitution, as this is not required. In fact, we

Algorithm 4.2: $U(A)$.

Input: Any κ-row matrix A
Output: A matrix U, such that U1 and U2

$B = (A \mid\mid I_\kappa)$
$B' = Gauss'(B) = (A' \mid\mid U)$
return U

manipulate A such that the result UA is upper-triangular. So, if we set $U = U(A)$ using this algorithm, the product UA has zero rows at the bottom and $\rho(A)$ independent rows on top: conditions U1 and U2 are clearly satisfied. It is important to realize that UA also satisfies an additional condition (i.e. that the leading coefficient of a row is always strictly to the right of the leading coefficient of the row above it), which is not directly necessary for this method to work.

4.2.2 Agreeing a pair of symbols

We can now employ the function $U(A)$ to agree a pair of symbols. The suggested implementation is the algorithm AGREEPAIR. Note that $S'_i \subset S_i$ means that $A'_i = A_i$ and $L'_i \subset L_i$.

Algorithm 4.3: AGREEPAIR(S_i, S_j).

Input: Two (possibly disagreeing) symbols S_i and S_j
Output: Two agreed symbols $S'_i \subset S_i$ and $S'_j \subset S_j$

$A = \binom{A_i}{A_j}$
$\kappa = k_i + k_j$ `// number of rows in A`
$U = U(A)$
$r =$ number of zero rows in UA
if $r \neq 0$ **then**
  set $T_i = \binom{L_i}{\mathcal{O}}$ and $T_j = \binom{\mathcal{O}}{L_j}$ such that T_i and T_j have κ rows
  determine $P_i = (UT_i)^{(-r)}$ and $P_j = (UT_j)^{(-r)}$
  for $l = 1, \ldots, s_i$ **do**
    if $-P_i^l \notin P_j$ **then**
      remove the l-th column from L_i
    $l = l + 1$
  for $l = 1, \ldots, s_j$ **do**
    if $-P_j^l \notin P_i$ **then**
      remove the l-th column from L_j
    $l = l + 1$
$S'_i : A_i X = [L_i];\ S'_j : A_j X = [L_j]$
return (S'_i, S'_j)

Recall that the notation $(UT_i)^{(-r)}$, since $r > 0$, indicates the submatrix of UT_i formed by its last r rows. Further, note that P_i and P_j are, like L_i and L_j, regarded as sets of columns. So, we a statement $a \in P_i$ means "a is a column of P_i". It is important to realize that, although this is assumed impossible for L_i, the set P_i may contain duplicate elements.

Also, since the l-th element of P_i corresponds to the l-th element of L_i, it is only permitted to change the order of the elements of P_i if the same ordering is applied to the elements of L_i. These remarks also hold for P_j and L_j, naturally.

This algorithm is a vital part of the final procedure for solving MRHS equations, used to obtain the results described in Chapter 5. Therefore, it is very important to understand its details.

AGREEPAIR initializes by constructing the joint coefficient matrix A and $U = U(A)$ using Algorithm 4.2. The number of all-zero rows of UA is denoted by r (note that $r = \kappa - \rho(A)$). If $r \neq 0$, we determine the matrices T_i and T_j, and multiply these by U. As the analysis will focus on the last r rows of UT_i and UT_j, we facilitate notation by referring to these submatrices as P_i and P_j, respectively. The details of AGREEPAIR are then clarified by the following theorem.

Theorem 4.1. Two symbols S_i and S_j agree if and only if either $r = 0$ or

- for all $p_1 \in P_i$ there exists a $p_2 \in P_j$ such that $p_1 = -p_2$ and
- for all $p_2 \in P_j$ there exists a $p_1 \in P_i$ such that $p_2 = -p_1$.

Proof. Recall that the coefficient matrices A_i and A_j individually have full rank, hence for all $l_1 \in L_i$ and for all $l_2 \in L_j$, the linear systems $A_iX = l_1$ and $A_jX = l_2$ are consistent (which does not necessarily imply the existence of a simultaneous solution). If $r = 0$, i.e. if A has full rank, the symbols S_i and S_j effectively operate on disjoint variable sets. So the simultaneous linear system

$$A_iX = l_1 \text{ and } A_jX = l_2, \tag{4.2}$$

with $l_1 \in L_i$ and $l_2 \in L_j$ is consistent if and only if the individual equations $A_iX = l_1$ and $A_jX = l_2$ are consistent, which is true by definition. In that case the symbols S_i and S_j agree and no columns can be deleted. So for the case $r = 0$ the proof is obvious.

Now, suppose $r \neq 0$, and assume that the conditions in Theorem 4.1 hold. We then prove that it is equivalent to state that S_i and S_j agree, by introducing intermediate equivalences.

Let $p_1 \in P_i$. Then by the first condition there is a column $p_2 \in P_j$ such that $p_1 = -p_2$. Let α_1 be the index of p_1 in P_i, so $p_1 = P_i^{\alpha_1}$. Similarly, let α_2 be the index of p_2 in P_j, thus

$$P_i^{\alpha_1} = -P_j^{\alpha_2}. \tag{4.3}$$

Since the transformation matrix U is such that UA has zeroes in its last r rows, we can alternatively state that

$$\underline{0} = (UA)^{(-r)}X = P_i^{\alpha_1} + P_j^{\alpha_2},$$

which can be rewritten as

$$(UA)^{(-r)}X = U^{(-r)}T_i^{\alpha_1} + U^{(-r)}T_j^{\alpha_2}. \tag{4.4}$$

Also, it is known by conditions U1 and U2 that UA has rank $\rho(A)$ and that all its zero rows are below the non-zero rows. This implies that the submatrix $(UA)^{(\rho(A))}$ has full rank, so any linear equation that has this submatrix as coefficient matrix and an arbitrary right-hand side can be solved, in particular

$$(UA)^{(\rho(A))}X = U^{(\rho(A))}(T_i^{\alpha_1} + T_j^{\alpha_2}). \tag{4.5}$$

Finally, observe that the number of rows of UA is κ, and that $\kappa = \rho(A) + r$. So we can combine Equations (4.4) and (4.5) and obtain

$$UAX = U(T_i^{\alpha_1} + T_j^{\alpha_2}),$$

which after multiplication with the inverse of U (which exists since U is non-singular by definition) proves the consistency of

$$AX = T_i^{\alpha_1} + T_j^{\alpha_2} = \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}, \tag{4.6}$$

where $l_1 = L_i^{\alpha_1}$ and $l_2 = L_j^{\alpha_2}$. Note that (4.6) is equivalent to (4.2), and that a similar analysis applies if we start with $p_2 \in P_j$. This implies that the corresponding symbols S_i and S_j agree. Since all described steps are equivalences, this proves the theorem. □

The algorithm is designed to remove columns that cause disagreement using the conditions introduced in Theorem 4.1. For the equation systems we consider, the number r is very low, making these conditions for agreement particularly easy to check.

4.2.3 Agreeing a pair of symbols in $\mathbb{F}_2$

If $q = 2$, as is the case for the equation systems we will consider, the algorithm for agreeing a pair of symbols can be simplified by observing that Equation (4.3) is then equivalent to

$$P_i^{\alpha_1} = P_j^{\alpha_2}.$$

Hence the minus signs in Algorithm 4.3 vanish, and the improved Algorithm 4.4 can be used.

Note that, since P_i and P_j are regarded as sets, the statement $P = P_i \cap P_j$ makes sense. For fixed $p_1 \in P_i$, the algorithm runs through the elements of the set P (which has at most 2^r elements), rather than the elements of the set P_j (with s_j elements, which may be larger than 2^r) to check which columns of L_i cause disagreement. We may even further speed up this search for identical elements by using sorted versions of the lists P_i and P_j.

This alternative algorithm is designed to remove columns according to the conditions for agreement in $\mathbb{F}_2$ described in the following corollary, which is immediately clear after noting that for $q = 2$ the minus signs in Theorem 4.1 can be omitted.

Corollary 4.2. Two symbols S_i and S_j over $\mathbb{F}_2$ agree if and only if $r = 0$ or $P_i = P_j$.

Note that Algorithms 4.3 and 4.4 can be significantly improved by allowing precomputations. Since the matrices $U = U(A)$ and the values r are independent of the right-hand side matrices, these are not affected by agreeing and need only be computed once. Furthermore, we only need to compute the matrices P_i and P_j once if we, upon deletion of a column from L_i, also delete the corresponding column in P_i, etc.

4.2.4 Agreeing a set of symbols

Once an implementation of pairwise agreeing is available, a set of M symbols can be agreed by sequentially applying AGREEPAIR until each pair of symbols agrees. This simple process is described in Algorithm 4.5.

Algorithm 4.4: AGREEPAIR(S_i, S_j) in $\mathbb{F}_2$.

Input: Two (possibly disagreeing) symbols S_i and S_j over $\mathbb{F}_2$
Output: Two agreeing symbols $S_i' \subset S_i$ and $S_j' \subset S_j$

$A = \binom{A_i}{A_j}$
$\kappa = k_i + k_j$ `// number of rows in A`
$U = U(A)$
$r =$ number of zero rows in UA
if $r \neq 0$ **then**
  set $T_i = \binom{L_i}{\mathcal{O}}$ and $T_j = \binom{\mathcal{O}}{L_j}$
  determine $P_i = (UT_i)^{(-r)}$ and $P_j = (UT_j)^{(-r)}$
  if $P_i \neq P_j$ **then**
    determine $P = P_i \cap P_j$
    for $l = 1, \ldots, s_i$ **do**
      if $P_i^l \notin P$ **then**
        remove column l from L_i
      $l = l + 1$
    for $l = 1, \ldots, s_j$ **do**
      if $P_j^l \notin P$ **then**
        remove column l from L_j
      $l = l + 1$
$S_i' : A_i X = [L_i];\ S_j' : A_j X = [L_j]$
return (S_i', S_j')

Algorithm 4.5: AGREE$(S_1, \ldots, S_M)$

Input: A set of symbols $S_1, \ldots, S_M$ that possibly disagree
Output: The set of agreed symbols $S_i' \subset S_i$, for $i = 1, \ldots, M$

for $i = 1, \ldots, M-1$ **do**
  $j = i + 1$
  while $j \leq M$ **do**
    $(S_i, S_j) =$ AGREEPAIR(S_i, S_j)
    if *columnsdeleted* **then**
      $i = 0$; break
    else
      $j = j + 1$
  $i = i + 1$
for $i = 1, \ldots, M$ **do**
  return $S_i' = S_i;\ i = i + 1$

Here *columnsdeleted* is a Boolean variable that is set to true whenever AGREEPAIR(S_i, S_j) deletes at least one column, and false otherwise. The command "break" returns the algorithm directly to the end of the closest current loop. In this case, if one or more columns are deleted from any right-hand side matrix, the pairwise agreeing starts over by checking symbols S_1 and S_2, then S_1 and S_3, etc. Note that, since agreement is both a symmetric and

a reflexive (i.e. any symbol agrees with itself) property by definition, there is no need for j to run through $\{1, \ldots, i\}$. Depending on the value of q, Algorithm 4.3 or 4.4 is used as implementation of AGREEPAIR(S_i, S_j).

Although in this particular implementation pairs of symbols are checked in the most straightforward order, the actual order in which symbols are compared and agreed does not affect the outcome of an agreeing procedure [RS07, Lemma 3.1].

4.2.5 Complexity issues

The average time complexity of Algorithm 4.5 is obviously very hard to determine, as it depends strongly on the particular coefficient and right-hand side matrices. Some rigorous estimates are available in [Sem07]. We can, however, obtain an upper bound on the time complexity by reasoning as follows. Recall that the number of symbols is M, and that each symbol contains at most k rows.

In the worst case (in terms of time complexity), Algorithm 4.5 deletes one column per iteration, and runs until there are no columns left. To delete a column, the algorithm tries to agree all pairs of symbols until a pair of disagreeing symbols is found, i.e. we need to compare at most

$$\frac{M(M-1)}{2} = O(M^2) \tag{4.7}$$

pairs of symbols. If precomputations are allowed, the main step in AGREEPAIR(S_i, S_j) is to find $p_1 \in P_i$ such that $-p_1$ does not occur in P_j, and similarly for $p_2 \in P_j$. This can be done by sorting P_i and P_j, after which an element that can be deleted is easily found. The most efficient algorithm for sorting a list is the *MergeSort* algorithm [Knu98, pp. 158-168], which requires $O(N \log N)$ operations if the list contains N elements. Since P_i and P_j contain at most q^k elements, agreeing a pair of symbols takes at most $O(q^k \log q^k)$ operations on vectors of $r \leq k$ bits. Hence the number of $\mathbb{F}_q$-operations required for agreeing two symbols (disregarding the cost of any precomputations) is bounded by

$$O(kq^k \log q^k). \tag{4.8}$$

In practice, k is relatively small, in which case Algorithm 4.3 behaves (roughly) exponential in the worst case. However, since the exponent k is considerably smaller than the number of variables n, in some cases the implementation suggested in this chapter may find the key used for encryption faster than an exhaustive search, whose time complexity is $O(q^n)$. We will return to this in Section 5.3.4.

An upper bound for the time complexity of Algorithm 4.5 is found by combining Equations (4.7) and (4.8) and multiplying by the maximal number of right-hand sides, which is Mq^k. We obtain

$$O(M^3 k q^{2k} \log q^k)$$

as the maximal time complexity of AGREE$(S_1, \ldots, S_M)$, which in $\mathbb{F}_2$ translates into

$$O(M^3 k^2 2^{2k}).$$

The precomputations consist of the computations of the transformation matrices $U = U(A)$ for $A = \binom{A_i}{A_j}$ and some less costly matrix operations. The main step in Algorithm 4.2 is the Gaussian elimination step, which takes $O(\kappa^2(n+\kappa))$ operations, where $\kappa = k_i + k_j \leq 2k$. Hence, the time complexity of the precomputations is at most

$$O(M^2 k^2 (n+k)).$$

Clearly allowing precomputations requires some additional memory. When agreeing symbols S_i and S_j, the lists P_i and P_j are used. Both lists contain at most kq^k elements of $\mathbb{F}_q$, and there are $O(M^2)$ pairs of symbols to be compared. Note that the other matrices used in the precomputations are not actually needed in the main algorithm. The additional memory required for Algorithm 4.5 is then at most

$$O(M^2 kq^k)$$

elements of $\mathbb{F}_q$.

Besides Algorithm 4.5, [RS07] describes the alternative *Agreeing2* algorithm for agreeing a set of symbols over $\mathbb{F}_2$. This algorithm uses a precomputation stage of similar time complexity, but its main loop is significantly faster: using *Agreeing2*, a set of symbols can be agreed in at most $O(M^2k^2 2^k)$ bit operations and $O(M^2 2^k)$ table look-ups. The additional memory needed is only marginal compared to Algorithm 4.5. Although the analysis presented only proves that it is more efficient than Algorithm 4.5 in the worst case, our experiments indicate that *Agreeing2* is also favorable on the average, at least for equation systems derived from ciphers.

Example 4.2. Consider for $q = 2$ the symbols $S_1 : A_1X = [L_1]$ and $S_2 : A_2X = [L_2]$ given by

$$S_1 : \begin{pmatrix} 1&1&0&0&0\\ 0&1&0&1&0\\ 0&0&1&1&0 \end{pmatrix} X = \begin{bmatrix} 0&0&1&1\\ 0&1&0&1\\ 1&1&1&0 \end{bmatrix} \text{ and } S_2 : \begin{pmatrix} 0&0&1&0&1\\ 0&1&0&1&0\\ 0&1&1&0&0 \end{pmatrix} X = \begin{bmatrix} 0&0&1&1\\ 0&1&0&1\\ 0&0&0&1 \end{bmatrix}$$

and run AGREEPAIR(S_1, S_2), for which we can use Algorithm 4.4 since $q = 2$. Then we compute the joint coefficient matrix $A = \binom{A_1}{A_2}$, the transformation matrix $U = U(A)$ and the product UA to obtain

$$A = \begin{pmatrix} 1&1&0&0&0\\ 0&1&0&1&0\\ 0&0&1&1&0\\ 0&0&1&0&1\\ 0&1&0&1&0\\ 0&1&1&0&0 \end{pmatrix}, \qquad U = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&1&1&0&0\\ 0&1&0&0&1&0\\ 0&1&1&0&0&1 \end{pmatrix}, \qquad UA = \begin{pmatrix} 1&1&0&0&0\\ 0&1&0&1&0\\ 0&0&1&1&0\\ 0&0&0&1&1\\ 0&0&0&0&0\\ 0&0&0&0&0 \end{pmatrix}.$$

Indeed UA is upper-triangular. Then we determine the matrices $UT_1 = U\binom{L_1}{\mathcal{O}}$ and $UT_2 = U\binom{\mathcal{O}}{L_2}$:

$$UT_1 = \begin{pmatrix} 0&0&1&1\\ 0&1&0&1\\ 1&1&1&0\\ 1&1&1&0\\ 0&1&0&1\\ 1&0&1&1 \end{pmatrix}, \qquad UT_2 = \begin{pmatrix} 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&1&1\\ 0&1&0&1\\ 0&0&0&1 \end{pmatrix}.$$

Since $r = 2$ (as the last two rows of UA are all-zero) the matrices P_1 and P_2 are given by the last 2 rows of UT_1 and UT_2, respectively:

$$P_1 = \begin{pmatrix} 0&1&0&1\\ 1&0&1&1 \end{pmatrix}, \qquad P_2 = \begin{pmatrix} 0&1&0&1\\ 0&0&0&1 \end{pmatrix}.$$

So $P = \{\binom{1}{0}, \binom{1}{1}\}$, and the first and third columns of both L_1 and L_2 can be deleted to obtain the symbols

$$S_1' : \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \text{ and } S_2' : \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

4.3 Procedure 3: GLUE

The procedure GLUE describes how to join two symbols, by extending the procedure AGREE. In practice, the usability of this procedure will turn out to be rather limited.

4.3.1 Gluing a pair of symbols

In Section 3.3.3 we defined the gluing of two symbols S_i and S_j. By using the basic procedure described there we find the gluing $S_i \circ S_j : AX = [L]$, where

$$A = \begin{pmatrix} A_i \\ A_j \end{pmatrix}, \tag{4.9}$$

$$L = \{(l_1 \mid\mid l_2) \mid l_1 \in L_i, l_2 \in L_j\}. \tag{4.10}$$

But if the linear system

$$AX = \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}$$

is inconsistent, the column (l_1, l_2) can be deleted from L_{ij} as it does not contribute to a joint solution of S_i and S_j. So, there may be a more efficient approach to construct L in case the joint coefficient matrix has non-full rank (i.e. if there exists a matrix U such that UA has at least one all-zero row). The suggested approach is very similar to the algorithm AGREEPAIR(S_i, S_j), it is depicted in Algorithm 4.6.

Indeed, if the joint coefficient matrix has full rank (i.e. $r = 0$), GLUE(S_i, S_j) returns the gluing according to Equations (4.9) and (4.10). But if $r > 0$, the algorithm returns only those right-hand sides for which a joint solution actually exists.

4.3.2 Complexity analysis

In Section 4.2.5 we estimated the time complexity of agreeing a pair of symbols by using the sorting algorithm *MergeSort*. This allowed us to give an upper bound on the time complexity of AGREE. A similar approach can be used to determine an upper bound on the time complexity of GLUE.

First, we sort the lists P_i and P_j using *MergeSort*. The algorithm then consists of the computation of a number of lists (with negligible complexity), the sortings of P_i and P_j and the computation of a column of L, in case a particular condition is satisfied. Let the number of columns of L be denoted by s, then Algorithm 4.6 requires

$$O(s_i \log s_i + s_j \log s_j + s)$$

elementary operations on vectors of length at most $\kappa \leq 2k$. Hence, the time complexity of GLUE(S_i, S_j) is upper bounded by

$$O(k(s_i \log s_i + s_j \log s_j + s)). \tag{4.11}$$

Algorithm 4.6: GLUE(S_i, S_j).

Input: A pair of symbols (S_i, S_j)
Output: The gluing $S_{ij} = S_i \circ S_j$

$A = \binom{A_i}{A_j}$
$\kappa = k_i + k_j$ `// number of rows in A`
$U = U(A)$
$r =$ number of zero rows in UA

if $r \neq 0$ **then**
  set $T_i = \binom{L_i}{\mathcal{O}}$ and $T_j = \binom{\mathcal{O}}{L_j}$
  determine $P_i = (UT_i)^{(-r)}$ and $P_j = (UT_j)^{(-r)}$
  determine $R_i = (UT_i)^{(\kappa - r)}$ and $R_j = (UT_j)^{(\kappa - r)}$
  $L =$ empty set
  for $l = 1, \ldots, s_i$ **do**
    for $m = 1, \ldots, s_j$ **do**
      if $(P_i)^l = -(P_j)^m$ **then**
        add $\{(R_i)^l + (R_j)^m\}$ to L
      $m = m + 1$
    $l = l + 1$
  return $S_{ij} : (UA)^{\kappa - r} X = [L]$
else
  set $L = \{(l_1, l_2) \mid l_1 \in L_i, l_2 \in L_j\}$
  return $S_{ij} : AX = [L]$

A full run of the algorithm requires, besides the initial memory needed to store the input symbols, the storage of the lists UT_i and UT_j. Furthermore, the list of right-hand sides L needs to be stored. The memory requirement of GLUE(S_i, S_j) is therefore

$$O(\kappa(s_i + s_j) + (\kappa - r)s) \simeq O(k(s_i + s_j + s)). \tag{4.12}$$

elements of $\mathbb{F}_q$.

4.3.3 Practical issues

The main drawback of Algorithm 4.6 (or any other gluing procedure, for that matter) is the fact that the number of right-hand sides s may increase drastically, which clearly affects the time complexity and the memory requirement of the algorithm (illustrated by the fact that Equations (4.11) and (4.12) depend on s). In practice, s may be much larger than $s_i + s_j$, which is the number of right-hand sides before gluing. In fact, this is the reason we can not simply glue all symbols to solve the whole system, as that process involves too many right-hand sides (although, in the end, only one will remain). We will only glue the pair of symbols that results in the lowest number of right-hand sides, provided this number is below some pre-set threshold Θ. After gluing we may try to agree the new set of symbols or extract linear equations, etc.

The value of the threshold Θ, mainly determined by the available amount of system memory and the desired maximal running time of the algorithm, is a very important parameter when solving a system of MRHS equations. We will see that a higher Θ results in

a lower number of guesses needed to uniquely solve the system. This is more or less what we would expect, as the algorithm can then run longer before making any guesses, with a higher probability of deleting columns. We will even see that, for the equation systems derived from the ciphers we consider, there is a peculiar yet very interesting trade-off between the value of Θ and the number of required guesses. This notion will be discussed further when the results are presented in Chapter 5.

Example 4.3. Consider for $q = 2$ the agreed symbols S_1' and S_2' from Example 4.2. The goal is to produce the gluing $S_{12} = S_1' \circ S_2'$. Note that we employ the already determined matrices from Example 4.2.
The coefficient matrix A_{12} is found by computing UA and selecting the non-zero rows, hence

$$A_{12} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix},$$

which is a full rank matrix by condition U1 (cf. 4.2.1) of the transformation matrix U.
The right-hand sides are found by computing the sum of those pairs of a column of UT_1 and a column of UT_2 whose last r coordinates coincide. The last r coordinates of the sum are disregarded. In this case we obtain the matrix

$$L_{12} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Note that there is no actual need to agree the symbols before gluing them. If GLUE had been applied to the symbols S_1 and S_2 defined at the beginning of Example 4.2, we would have obtained the same A_{12} and L_{12}.

4.4 Guessing and eliminating variables

We already described how the additional symbol $S_0 : A_0X = l_0$ can be used to keep track of extracted linear equations and guessed variables. Now, assume the first part of EXTRACTLINEAR(S_i) returns the vector a, i.e. a is the coefficient vector of a homogeneous linear equation induced by S_i. Then it should be checked that

- a is not already induced by S_0, and
- a does not contradict S_0.

If a can be written as a linear combination of the equations already present in S_0 (in which case the former condition is not satisfied), it is discarded as it does not contain any new information. If the latter condition is not satisfied, we can conclude that a made guess was wrong and reject guesses accordingly. Finally, if both conditions are satisfied, the extracted linear equation should be added to S_0 by adding the vector a to A_0 and a zero to l_0. Similarly, if the second part of EXTRACTLINEAR(S_i) returns a vector a and both conditions hold, we add a to A_0 and a one to l_0.

Now, when all symbols pairwise agree, no new or contradicting linear equations can be extracted and gluing is too costly, a guessing procedure is needed in order to possibly delete

some columns and move towards finding the solution to the system. Obviously, a guess is added to S_0 in a similar manner: if $\alpha_1 x_1 + \ldots + \alpha_n x_n = v$ is the intended guess, the vector $(\alpha_1, \ldots, \alpha_n)$ is added to A_0 and the constant v is added to l_0. Note that it should be ensured that the guess satisfies both requirements described above.

Once a linear equation or guess has been added to S_0, one involved variable can be eliminated by gluing the addition to S_0 to all symbols. Since S_0 has only one right-hand side, eliminating a variable does not increase the total number of right-hand sides. In fact, this number is even likely to decrease, as only those right-hand sides that satisfy the linear equations induced by S_0 will survive.

5

MRHS equation systems from ciphers

This chapter will describe how a large class of ciphers can be represented as a system of multiple right-hand side equations. Solving this system by applying the procedures of Chapter 4 is then a method to retrieve the used encryption key, and as such is an algebraic attack on the cipher. This type of attack is the focus of this chapter, and will be referred to as "our" attack.

Besides a general description of suitable ciphers, we will give explicit details and results of this new type of attack applied to scaled versions of DES and AES. Unless indicated otherwise, the presented results were obtained independently of those published earlier, e.g. in [RS07] and [Rad07].

5.1 Suitable ciphers

Consider an iterated block[4] cipher, i.e. a cipher that consists of a round function $F : \mathbb{F}_q^b \times \mathbb{F}_q^\lambda \to \mathbb{F}_q^b$, where b is the block length, that is applied recursively R times. Input to the i-th application of the round function is the internal state of the cipher and a round key $K_i \in \mathbb{F}_q^\lambda$, which is derived from the main key K only (i.e. it is independent of the plaintext and any external parameters) in a process called the key schedule. In order for decryption to be possible, the round function F should be injective if the second argument (i.e. the round key) is fixed. In other words there should exist a function F' (called the inverse of F) such that

$$F'(F(x, K_i), K_i) = x$$

for all x and K_i [Sti05].

In practice, the round function F often consists of a linear part L and a non-linear part N which are applied sequentially. The encryption, often denoted by means of the encryption function E_K, can then be described by the following recursion (where we use the interme-

[4] As our analysis will focus on block ciphers only, we will omit the term block from now on.

diate states X_i to describe the output of the i-th round function)

$$E_K(p) : \begin{cases} X_0 &= p, \\ X_i &= N(L(X_{i-1}, K_i)) \text{ for } i = 1, \ldots, R, \\ X_R &= c. \end{cases} \tag{5.1}$$

Here, p and c are the plain- and ciphertext blocks, respectively. The recursion can be graphically represented as in Figure 5.1.

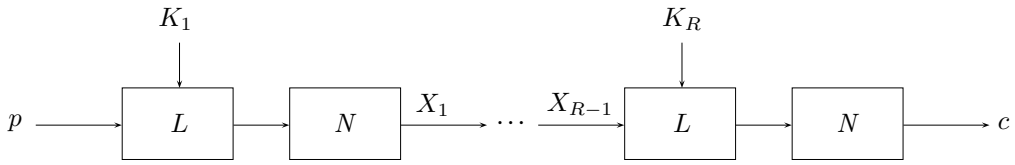

Figure 5.1: Iterated block cipher.

As we have assumed the known plaintext setting, we will not be concerned with the corresponding decryption $D_K(c)$. Instead, the objective is to determine the key[5] K given only one plaintext p and ciphertext c that satisfy $c = E_K(p)$. Note that most block ciphers are designed such that given a plaintext p, each key K results (after encryption) in a distinct ciphertext c such that $c = E_K(p)$. This justifies the assumption made earlier that the key corresponding to a given plain-/ciphertext pair is unique.

Most modern ciphers have this general form, although there may be variations. For example, the transformations L and N may vary for different rounds, their order may be reversed, or there may be some initial or final operations applied. However, to allow for an efficient implementation, most cipher designs only employ a very low number of operations, so these variations are often only marginal. The AES, for example, which contains four different operations, has an initial operation and a last round that slightly differs from the intermediate rounds (cf. 5.4.1).

The non-linear transformation N is commonly implemented as a co-called S-box (or a series of S-boxes):

Definition 5.1. *An $l \times m$ or l-to-m S-box or substitution box is a component of a cipher that maps an element of $\mathbb{F}_q^l$ to an element of $\mathbb{F}_q^m$.*

In practice, S-boxes are non-linear transformations, but this is not required by definition.

It is important to realize that it is essentially the non-linear portion of the round function that determines a cipher's resistance against algebraic attacks, as linear operations (which can be written as matrix-vector multiplications and additions) are relatively easy to invert. Therefore, in practice S-boxes are carefully designed to make it difficult, if not infeasible, to find a linear function that behaves approximately the same as the S-box (so-called linear approximation). Many successful algebraic attacks have a linear approximation at their core. S-box design is a very interesting and dynamic subfield in cryptography, but it goes beyond the scope of this thesis to discuss it in detail.

[5] Actually, it suffices to determine the round keys K_i, but it is generally a simple task to determine K once these are known.

As will be shown in Section 5.2, all iterated block ciphers in which the only non-linear components are S-boxes can be represented as a system of MRHS equations (we will refer to such ciphers as suitable). In particular, this requirement must also hold for the key schedule: if the encryption process (5.1) satisfies the requirement, but the key schedule contains a non-linear transformation that can not be viewed as an S-box, the cipher is unsuited for our attack. Most modern ciphers (not necessarily block ciphers) are suitable, although designers do not consistently use the term S-box. Some descriptions simply use the term non-linear function. However, if this function has a fixed-length input and output, it is essentially an S-box and the cipher satisfies the requirements (e.g. the cipher KeeLoq [BDI08] employs the non-linear function NLF that has five bits of input and one bit of output; essentially this is an S-box, making KeeLoq suitable for our attack).

The straightforward example of a non-linear transformation that can not be represented by an S-box is a hash function [MOV96, Ch. 9], as that takes a variable length vector as input, i.e. the value of l in Definition 5.1 is not fixed. We are able to represent a hash function using MRHS equations if we introduce some restrictions, for example by only considering messages of a fixed length. Although this representation does not describe the whole cipher, it may provide some insight into possible weaknesses of the studied hash function. Note that many hash functions have a block cipher-like internal structure, which does make them suitable for our representation.

5.2 Constructing the equations

Assume the studied cipher satisfies the requirements above. We then introduce variables to enable a description of the cipher using MRHS equations. Defining variables is a very important step in constructing MRHS equations: since an algorithm for solving the equations will run more efficiently if the number of variables (denoted by n) is small, the objective is to introduce a minimal number of variables that allows for a full description of the cipher.

For iterated ciphers, it is often useful to introduce, for $i = 1, \ldots, R$, the state variables $X_i = (x_i^1, \ldots, x_i^b)$ to describe either the inputs or the outputs of the round functions, and the round key variables $K_i = (K_i^1, \ldots, K_i^\lambda)$ to describe the round keys. For now, let X_i describe the output of the i-th round function as illustrated in Figure 5.1. Note that we may omit the variables X_R, since $X_R = c$ is known.

Denote the vector of all variables by X, so $X = (K_1, \ldots, K_R, X_1, \ldots, X_{R-1})$. An encryption process corresponds to a unique assignment of values to the variables X (i.e. there is exactly one vector X such that (5.1) holds) by assumption, and the objective is to retrieve this assignment given one plain-/ciphertext pair, which we may consider known constants.

Focus on an S-box $\mathcal{S}$ (cf. Figure 5.2) that is employed in a certain round, and observe (since we assumed there are no other non-linear operations involved) that its input vector $u = (u_1, \ldots, u_l)$ can be written as a linear combination of the variables X and the known constants $C = (p, c)$, so we can determine the coefficient vectors α_t and β_t such that

$$u : \begin{cases} u_1 &= \langle \alpha_1, X \rangle + \langle \beta_1, C \rangle \\ u_2 &= \langle \alpha_2, X \rangle + \langle \beta_2, C \rangle \\ &\vdots \\ u_l &= \langle \alpha_l, X \rangle + \langle \beta_l, C \rangle \end{cases} \tag{5.2a}$$

where $\langle x, y \rangle$ denotes the inner product of vectors x and y. Similarly, the output vector

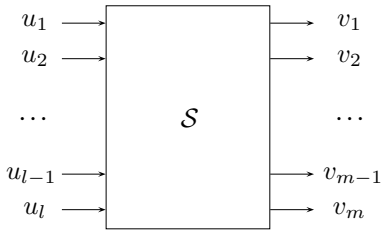

Figure 5.2: General S-box.

$v = (v_1, \ldots, v_m)$ of S can be written as

$$v : \begin{cases} v_1 &= \langle \gamma_1, X \rangle + \langle \delta_1, C \rangle \\ v_2 &= \langle \gamma_2, X \rangle + \langle \delta_2, C \rangle \\ &\vdots \\ v_m &= \langle \gamma_m, X \rangle + \langle \delta_m, C \rangle \end{cases} \tag{5.2b}$$

by using the coefficient vectors γ_t and δ_t. Now consider the $(l+m) \times n$ matrices

$$A = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_l \\ \gamma_1 \\ \vdots \\ \gamma_m \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_l \\ \delta_1 \\ \vdots \\ \delta_m \end{pmatrix} \tag{5.3}$$

and observe that Equations (5.2a) and (5.2b) are equivalent to

$$(u \mid\mid v)^T = AX + BC,$$

where BC is a known constant vector.

Viewed as $AX + BC$, the vector $(u \mid\mid v)$ can have any of q^{l+m} possible values. However, since v is the image of u under $\mathcal{S}$, it holds that

$$(u \mid\mid v)^T = (u \mid\mid \mathcal{S}(u))^T.$$

There are only q^l possible choices for u (i.e. inputs to the S-box), so this is the actual number of choices for $(u \mid\mid v)$. In other words, if X corresponds to a full encryption, then the vector $AX + BC$ can be written as $(x \mid\mid \mathcal{S}(x))$ for some x. So, AX is an element of the set

$$L = \{(x \mid\mid \mathcal{S}(x))^T - BC \mid x \in \mathbb{F}_q^l\}.$$

That implies that X satisfies the MRHS equation

$$S : AX = [L],$$

which has $l + m$ rows and q^l right-hand sides.

The input and output of a specific S-box typically depend on a small fraction of the variables X only. For instance, an S-box involved in round i of the encryption process has an input that only depends on X_{i-1} and K_i, and its output only determines the values of the variables X_i. In this case only the columns in the coefficient matrix A that correspond to these particular variables are non-zero. So, if we consider a single symbol S there may be a huge number of possible solutions. However, recall that the encryption is fully described by X and that we assumed that X is unique. Hence, there will be a single solution if we set up a symbol S_j for each S-box involved in the encryption process and the key schedule and consider the so-formed system of symbols

$$S_1, \ldots, S_M,$$

where M is the total number of S-boxes in the cipher, typically R times the number of S-boxes involved in a single round of encryption (including the key schedule). This is the idea of the algebraic attack described in this paper: construct one MRHS equation for each non-linear component involved in the encryption process and key schedule, and simultaneously solve the obtained symbols using Algorithm 3.1.

The specific implementations described in Chapter 4 have been extensively tested on scaled versions of DES and AES. The next sections will describe these ciphers, their representations using MRHS equations and the results obtained by running the algorithm. The primary interest of our research is the number of guesses needed to uniquely solve the system, as this number is independent of the used implementation of agreeing and gluing and only depends on the allowed number of right-hand sides Θ. Even if a significantly faster implementation of these general procedures is available, the required number of guesses will remain unchanged.

Since both DES and AES are binary ciphers, from now on it will be assumed that $q = 2$.

5.3 Experiments on DES-variants

The Data Encryption Standard (DES) was designed in 1975 by Ehrsam et al. [EMP75] and was published as Federal Information Processing Standard (FIPS) 46 in 1977 [Nat99]. It has been used in several applications, and there is a vast amount of information on possible attacks (cf. 5.3.2). Mainly due to its relatively small (effective) key length of 56 bits, it was superseded by the Advanced Encryption Standard (AES) in 2002 and eventually withdrawn as a standard in May, 2005. The closely related cipher Triple DES (which basically consists of three DES operations with two or three distinct keys) is still used in electronic banking.

5.3.1 Cipher description

DES is a block cipher operating on blocks of 64 bits that has an effective key length of 56 bits (8 bits of the 64-bit encryption key are used for parity checking only). It contains sixteen rounds of encryption. However, in practice the cipher is often analyzed using reduced-round variants, i.e. variants that use a lower number of rounds for encryption. Although the name DES is reserved for the full-round version, we may refer to the reduced-round variants as DES nonetheless. If required, we will mention the number of used rounds, denoted by R, explicitly. The encryption process is summarized in Figure 5.3.

Before applying the round function, the plaintext p is permuted using the permutation IP and split into two 32-bit half-blocks, say p^l and p^r. If the input to round i is denoted by

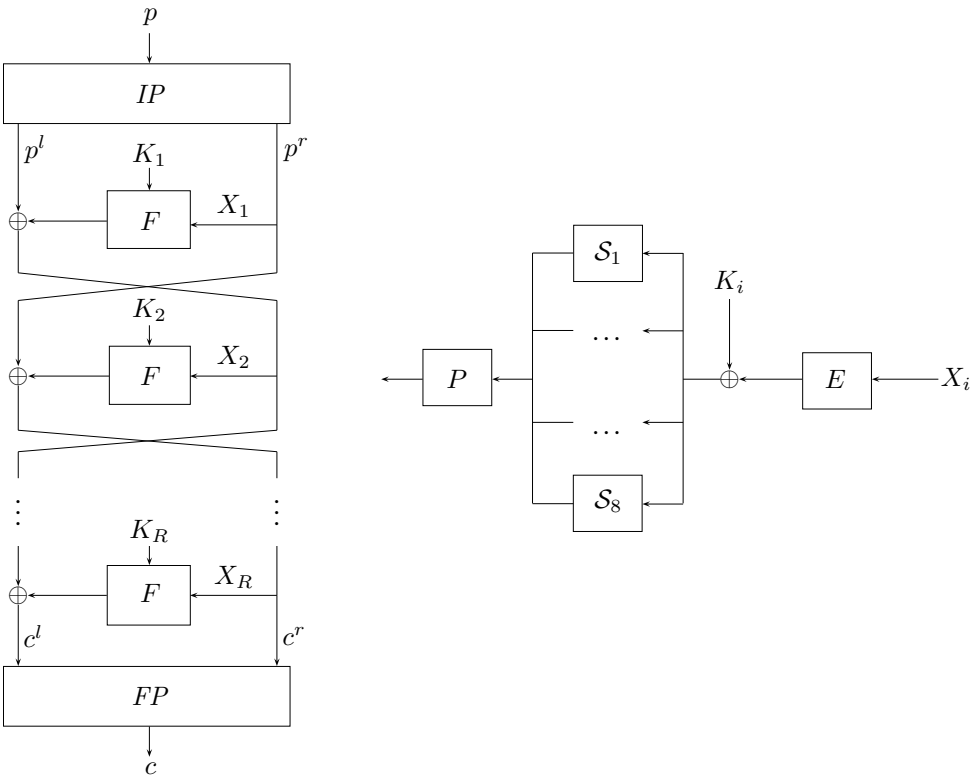

Figure 5.3: DES encryption overview (left) and round function.

X_i, the encryption process is then recursively defined by

$$E_K^{DES}(p) : \begin{cases} (X_0 \,||\, X_1) &= (p^l \,||\, p^r) \\ X_i &= F(X_{i-1}, K_{i-1}) \oplus X_{i-2} \text{ for } i = 2, \ldots, R+1, \\ (X_{R+1} \,||\, X_R) &= (c^l \,||\, c^r), \end{cases} \tag{5.4}$$

where $\oplus$ is the bitwise xor-operation. We will describe the key schedule for extracting the round keys $K_1, \ldots, K_R$ from K below. The ciphertext is finally obtained by concatenating c^l and c^r and applying the final permutation FP, which is the inverse of IP.

The DES round function F operates on 32-bit half-blocks and consists of four stages:

(i) **Expansion.** The 32-bit half-block is expanded into a 48-bit vector using a permutation E. Some input bits are used twice.

(ii) **Round key addition.** The 48-bit round key is added to this 48-bit vector.

(iii) **Substitution.** The resulting 48-bit vector is split into eight equal parts and part j is fed through a 6×4 S-box S_j, for $j = 1, \ldots, 8$.

(iv) **Permutation.** The eight 4-bit S-box outputs are concatenated and permuted using a permutation P, resulting in the 32-bit output of F

The key schedule is illustrated in Figure 5.4. It initializes by compressing the 64-bit key K into 56 effective bits by applying the permutation PC_1, note that eight bits of K are

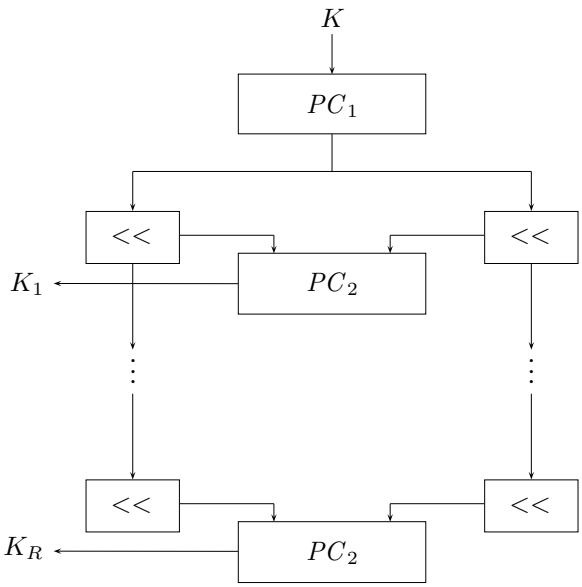

Figure 5.4: DES key schedule.

simply discarded. Then, the resulting 56-bit effective key is split into two equal halves, and both halves are treated separately. The extraction of a round key has two stages:

(i) **Rotation.** The 28-bit vectors are cyclically rotated to the left by either one (for rounds 1, 2, 9 and 16) or two positions.

(ii) **Permutation.** The two rotated 28-bit vectors are concatenated and compressed into a 48-bit round key by using the permutation PC_2, 24 bits are selected from each of the input vectors.

The details of the used permutations and S-boxes can be found in Appendix A.

As is clear from the above cipher description, DES consists of linear operations only (permutations, additions and cyclic rotations), except for the substitution stage (iii) in encryption. Since this stage is described using only S-boxes as non-linear components, DES satisfies the requirements for a suitable cipher, and we are able to describe it by means of MRHS equations.

5.3.2 Previous attacks

Before we go into the exact description of the attack on DES proposed in this paper, we mention some previous results in order to obtain some historic perspective.

Differential cryptanalysis

After several studies without mentionable results, e.g. [HMS76], the first reported attack on DES-like ciphers was claimed by Biham and Shamir in 1991 [BS91]. Their attack employs the newly introduced concept of differential cryptanalysis (a study of how small changes in the plaintext affect the resulting ciphertext when encrypted using the same key) to break reduced-round variants of DES. For fifteen rounds, their attack is still slightly faster than

$O(2^{56})$, which is the time complexity of the exhaustive search. However, it requires a huge amount of $O(2^{51})$ chosen plaintext pairs. For sixteen rounds, the time complexity of the attack is higher than the exhaustive search, although a variant that was able to break the full cipher slightly faster than exhaustive search was published in 1992 [BS92]. This latter attack requires $O(2^{47})$ chosen plaintexts.

The designers of DES have since claimed that the cipher was designed to resist this type of attack, indicating that differential cryptanalysis was already known at the time DES was introduced.

Linear cryptanalysis

In 1993, Matsui [Mat93] published an attack on DES, based on linear approximations of the S-boxes. The attack has an average time complexity of $O(2^{42})$ for sixteen-round DES, making it still the best analytical method to break the full-round cipher faster than the exhaustive search, although its data requirement of $O(2^{43})$ known plaintexts makes it hardly practical. Some improvements on the required amount of known plaintexts have been published, the best of which require about one fourth of the initial number.

Davies' attack

Where the previously described attacks are general in nature and may be applied to any cipher (with varying results), the attack suggested by Davies and Murphy in 1987 [BB97] is designed to exploit properties that are unique to DES. This attack focuses on the outputs of pairs of adjacent S-boxes, which turns out to have interesting statistical properties. Where the other attacks rely heavily on algebraic techniques, Davies' attack has a more probabilistic nature. The best improved version of the attack requires $O(2^{49})$ operations using $O(2^{50})$ known plaintexts.

Brute force attacks

As it has a relatively low effective key length of 56 bits and with computing power ever increasing, DES has become more and more vulnerable to the brute force attack, i.e. trying each possible key until the correct key has been identified. Using huge networks of computers and/or highly efficient but costly machines designed specifically for this task, the current record for recovering a DES key given one known plaintext dates from January, 1999, and stands at just over 22 hours.

Although DES' vulnerability to the brute force attack has been shown, it is important to realize that no practical algebraic attack exists. For this reason it is still an interesting cipher for algebraic analysis, and many newly suggested attacks are applied to DES as well as to ciphers that are considered more secure.

5.3.3 Representation using MRHS equations

This section will describe how R-round DES can be described using MRHS equations only. Recall that we set up one symbol for each S-box encountered. As there are eight S-boxes involved in each round, we will need $8R$ symbols.

Before constructing the equations, note that the permutations IP, FP have no actual cryptographic meaning, as their inputs and outputs can be easily described as linear combinations of the plaintext p and the ciphertext c, which are known. The same more or less

holds for the permutation PC_1 employed in the key schedule, as that only controls which bits of the encryption key K are used and in which order. Hence, these three permutations are commonly disregarded when analyzing the cipher.

We introduce variables to describe the inputs and outputs to all S-boxes. First, we note that the round key K_i in round i consists of certain bits of the encryption key: for each round key a different set of bits is selected. This implies that we do not need any additional variables to describe the round keys, and we can represent all round keys by the key variables

$$K = (k_1, \ldots, k_{56}).$$

To describe the intermediate states of the encryption process, we introduce the additional variables

$$X_i = (x_i^1, \ldots, x_i^{32})$$

for $i = 2, \ldots, R-1$, where X_i is the 32-bit input to the i-th round function as illustrated in Figure 5.3 and Equation (5.4). An alternative is to introduce variables that describe the output of the round function, which would lead to the same number of variables and an equivalent analysis. This is, however, the minimal number of variables needed to describe R-round DES. Note that we do not need X_1, X_R and X_{R+1}, as those are already known. Using these $n = 56 + 32(R-2)$ variables and $M = 8R$ symbols, we can fully describe the cipher using ten-row symbols with $2^6 = 64$ right-hand sides. This is illustrated in Examples 5.1 and 5.2.

Example 5.1. Consider DES' third round function (assume $R > 4$), whose input is given by

$$X_3 = (x_3^1, \ldots, x_3^{32}).$$

This is then first expanded into 48 bits by the permutation E, after which the third roundkey is added. The first six bits of this result are sent to S-box $\mathcal{S}_1$, etc. Combining this with the details of DES given in Appendix A, we obtain

$$u = (x_3^{32}, x_3^1, x_3^2, x_3^3, x_3^4, x_3^5) \oplus (k_{18}, k_{21}, k_{15}, k_{28}, k_5, k_9)$$

as the input to $\mathcal{S}_1$ in the third round of encryption.

To express the output of $\mathcal{S}_1$ as a linear combination of the available variables, observe that Equation (5.4) implies that

$$F(X_3, K_3) = X_2 \oplus X_4.$$

If we denote the inverse of the permutation P by P', the output bits of $\mathcal{S}_1$ are then given by the first four bits of $P'(F(X_3, K_3)) = P'(X_2 \oplus X_4)$. Using the details of P, we obtain

$$v = (x_2^9, x_2^{17}, x_2^{23}, x_2^{31}) \oplus (x_4^9, x_4^{17}, x_4^{23}, x_4^{31})$$

as the output of $\mathcal{S}_1$.

In terms of Section 5.2, we have determined the coefficient vectors α_t and γ_t. For example, α_1 has all-zeroes, except at those positions corresponding to the variables x_3^{32} and k_{18}, where it has a one. Since both the input and output of $\mathcal{S}_1$ in the third round are independent of any constants, the coefficients β_t and δ_t are zero. By Equation (5.3), the coefficient matrix

A is given by

$$A = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_6 \\ \gamma_1 \\ \vdots \\ \gamma_4 \end{pmatrix}$$

so $AX = (u, v)$ describes the input and output of $\mathcal{S}_1$ in terms of the variables. On the other hand, the vector (u, v) also satisfies $(u, v) \in L_1$, where L_1 is the right-hand side matrix associated with $\mathcal{S}_1$, i.e.

$$L_1 = \{(x, \mathcal{S}_1(x)) \mid x \in \mathbb{F}_2^6\}.$$

This leads to the symbol

$$AX = [L_1],$$

which fully describes the third round S-box $\mathcal{S}_1$.

For $R = 6$, the coefficient matrix A is the 10×184 matrix depicted in Figure 5.5, where each dot indicates a one. The variables in this case are sorted as $X = (K, X_2, \ldots, X_5)$. The coefficients in the first block correspond to K, those in the second block to X_2, etc.

Figure 5.5: Third round DES coefficient matrix ($R = 6$).

Following this procedure for each of the S-boxes, we obtain $8R$ symbols describing the cipher. Note that symbols describing the S-boxes from rounds 1, 2, $R - 1$ and R depend on the known constants p and c as well, but on less (10 for the rounds 1 and R, 16 for rounds 2 and $R-1$) variables than the intermediate rounds (which depend on 20 variables). In case a round function depends on one or more of the known constants, the analysis is comparable to that illustrated in Example 5.1, as is illustrated below.

Example 5.2. Consider the second round function (assume $R > 3$), whose input is given by

$$X_2 = (x_2^1, \ldots, x_2^{32}).$$

Applying the permutation E and the round key addition, we obtain

$$u = (x_2^{32}, x_2^1, x_2^2, x_2^3, x_2^4, x_2^5) \oplus (k_{16}, k_{19}, k_{13}, k_{26}, k_3, k_7)$$

as the input to $\mathcal{S}_1$.

The main difference is in the output of the round function. Equation (5.4) implies that

$$F(X_2, K_2) = p^r \oplus X_3,$$

as $X_1 = p^r$ is known. Using P' from Example 5.1, the output bits v of $\mathcal{S}_1$ are now the first four bits of $P'(F(X_2, K_2)) = P'(p^r \oplus X_3)$, hence

$$v = (p^{r9}, p^{r17}, p^{r23}, p^{r31}) \oplus (x_3^9, x_3^{17}, x_3^{23}, x_3^{31}).$$

Note that the first term of this expression is known.

In this case, only the coefficient vectors β_t are all-zero, so we determine both matrices A and B. Using the right-hand side matrix L_1 of Example 5.1, we obtain the symbol

$$AX = [L_1']$$

with

$$L_1' = \{L_1^j \oplus BC \mid j = 1, \ldots, 2^6\}.$$

For $R = 6$, the coefficient matrix A is depicted in Figure 5.6, with the same ordering of the variables as before.

Figure 5.6: Second round DES coefficient matrix ($R = 6$).

5.3.4 Experimental results

The algorithms and their implementations described in Chapters 3 and 4 have been extensively applied to systems of symbols corresponding to DES encryptions for various values of R. Our main interest was how the number of guesses needed to uniquely solve the systems is affected if the value of the threshold Θ is varied. As mentioned in 5.2, this behavior is implementation-independent as long as the SOLVEMRHS algorithm of Section 3.4 is used to solve the system.

It is important to think about which variables to guess, and in which order. As the main objective of solving a system of symbols is to retrieve the encryption key, it makes sense to guess key variables. In the case of DES these are the variables in $K = (k_1, \ldots, k_{56})$. However, it is known that DES has the property that some key bits are used more often than others, so we may expect better results if guessing is done in order of usage: the most used variables first, etc. This method is referred to as method G_1. The straightforward alternative is method G_2: to guess the values of key variables in their ascending order, so k_1, k_2, etc.

For $\Theta = 2^8$ and $\Theta = 2^{16}$, the results described in Table 5.1 were obtained. For comparison, the results for $\Theta = 2^{18}$ published earlier by Raddum in [Rad07] are also included. As can be seen, the results for $\Theta = 2^8$ and 2^{16} support Raddum's results. As described in Section 3.4, the algorithm traverses a binary search tree with backtracking, until correctly guessing the required number of variables solves the system. Our simulations indicate that the number of guesses needed to make the algorithm halt is not higher (typically, it is one less) than the number of correct guesses needed to solve the system. Hence, we never need more than the indicated number of guesses to either solve the system or reject a made guess. To speed up the simulations, we did not let the algorithm traverse the whole search tree, but made sure that each made guess was the correct one. For instance, with $\Theta = 2^8$, we simply fed the algorithm with the correct fourteen guesses required to solve four-round DES (using guessing method G_2), instead of trying all 2^{14} possible guesses.

As it turns out, the method of guessing the most used key bits first (method G_1) is less efficient than the straightforward approach of guessing key bits in ascending order. This somewhat contradicts our expectations, and a possible reason may be the specific form of the DES key schedule, as is argued in [Rad07, pp. 240].

Also, the first row of the table implies that the algorithm can extract the solution without guessing any variables for three-round DES. As that is not a very complex cipher, this result is hardly interesting. For larger values of R, the number of required guesses increases quite

	$\Theta = 2^8$		$\Theta = 2^{16}$		$\Theta = 2^{18}$	
R	method G_1	method G_2	method G_1	method G_2	method G_1	method G_2
3	0	0	0	0		
4	15	14	6	6	3	3
5	32	28	28	20	26	17
6	48	46	36	34	34	28
7	48	48	44	40	38	38
8	48	48	45	40	38	38
9	48	48	45	40		
10	49	48	45	40	38	38
11	49	48	45	40		
12	49	48	45	40	38	38
13	50	48	45	40		
14	50	48	45	41		
15	51	48	45	41		
16	51	49	46	41	41	38

Table 5.1: Number of guesses for R-round DES-systems for various values of Θ.

rapidly, until reaching some maximum at about seven rounds, after which the number of guesses stagnates.

However, arguably the most interesting conjecture is that there seems to be a trade-off between the number of guesses and the allowed number of right-hand sides Θ, summarized in the following hypothesis.

Hypothesis 5.1. If $\Theta = 2^l$ right-hand sides are allowed, then R-round DES, for $7 \leq R \leq 16$, can be broken by guessing $56 - l$ key bits, provided variables are guessed according to method G_2.

Several values of Θ have been tested to try to contradict this hypothesis, but the only counterexamples found still roughly satisfy this hypothesis, e.g. when we set $\Theta = 2^{16}$, we need 41 guesses instead of 40 to break fourteen rounds or more. An overview of some values of Θ and the corresponding number of guesses needed for $R = 16$ is summarized in Table 5.2. Note that the value of Θ largely determines the amount of memory and time needed

Θ	2^6	2^8	2^{12}	2^{16}	2^{18}	2^{20}
Number of guesses	50	49	45	41	38	36

Table 5.2: Trade-off for various values of Θ.

to run Algorithm 3.1, which would imply some sort of "time-memory-number of guesses" tradeoff, possibly comparable to [Hel80].

If Hypothesis 5.1 is true, this does not imply a break of the R-round (with $R \geq 7$) DES, as the main loop (i.e. agreeing-gluing-extracting linear equations) of SOLVEMRHS with Θ right-hand sides requires at least Θ operations. To see this, observe that each right-hand side is to be checked at least once (i.e. when it is deleted). In practice, the time complexity of SOLVEMRHS may be much higher, unfortunately there are no estimates available yet. It is expected that it can be written as $O(a\Theta)$, where a is a constant that depends on the number of symbols and variables, but not on the value of Θ.

The hypothesis may imply a break of one of the smaller variants of DES. If we let $\Theta = 2^{18}$ and assume that the main loop of the algorithm requires less than 2^{28} operations (i.e. 2^{10} times the maximal number of right-hand sides), the time complexity of breaking six-round DES, for which we need to guess 28 key bits, is trictly lower than

$$O(2^{28} \cdot 2^{28}) = O(2^{56}),$$

which implies a break faster than the brute force attack. For the five and four-round variants these results are even better. Note that this attack is carried out using only one plain-/ciphertext pair. However, as long as the time complexity of the algorithm is not properly estimated, we can not formally claim to have successfully broken these DES variants.

5.4 Experiments on AES-variants

In January, 1997, the National Institute of Standards and Technology (NIST) publicly requested proposals for a successor of DES, to be named the Advanced Encryption Standard (AES) . After an extensive process of selecting and testing candidates suggested by the cryptographic community, the Rijndael-cipher [DR02] designed by Daemen and Rijmen was selected as the official successor to DES in October, 2000. It was formally presented as FIPS 197 [Nat01] in November, 2001, and has been used in numerous applications since.

Satisfying the requirements set by the NIST, AES[6] is a cipher that operates on blocks of 128 bits, and it supports key lengths of 128, 192 and 256 bits, making a brute force attack infeasible for now and the near future (at least, according to the NIST). Its details are remarkably simple, containing only one S-box. This makes it an ideal candidate for our approach. As DES, it is commonly studied by using smaller variants. Although there are several variants available, the one considered here is the cipher SR*(R, Nr, Nc) from [CMR05], which will also be referred to as simply SR*.

5.4.1 Cipher description

Where DES is mainly based on table lookups, SR* depends more on finite field arithmetic. It operates on an array of bytes (i.e. vectors of eight bits) called the internal state, where the total number of bits in the internal state equals the block length b. The cipher is parametrized as follows:

- R is the number of rounds, $3 \leq R \leq 10$,
- Nr is the number of rows in the internal state, $Nr = 1, 2$ or 4,
- Nc is the number of columns in the internal state, $Nc = 1, 2$ or 4.

Observe that

$$b = 8NrNc. \tag{5.5}$$

Calculations are performed in the finite field $\mathbb{F}_{2^8}$, which is defined as

$$\mathbb{F}_{2^8} = \frac{\mathbb{F}_2[X]}{(X^8 + X^4 + X^3 + X + 1)} = \mathbb{F}_2(\theta),$$

where θ is a root of the polynomial $X^8 + X^4 + X^3 + X + 1$.

[6] We refer to Rijndael as the AES, although there are some formal differences.

In the encryption process, the plaintext block is written as an $Nr \times Nc$ array of bytes, which is then further manipulated to obtain the $Nr \times Nc$ ciphertext block array after R rounds. For example, if $Nr = Nc = 2$, the block length of SR* equals $2 \cdot 2 \cdot 8 = 32$ bits. From the possible values of Nr and Nc, it follows that SR* has a block length of either 8, 16, 32, 64 or 128 bits. The key length is equal to the block length, and, like the plain- and ciphertext, the key is regarded as an $Nr \times Nc$ array. The same holds for the round keys.

The round function of SR*(R, Nr, Nc) contains four operations, that are applied sequentially. Further details can be found in Appendix B.

1. `SubBytes` - The only non-linear operation, which employs the 8×8 S-box $\mathcal{S}$. This S-box is generated by determining the multiplicative inverse of the input (viewed as a polynomial), multiplying this element of $\mathbb{F}_{2^8}$ by a matrix M_{SB} and adding a constant C_{SB}. Each element of the internal state is fed through $\mathcal{S}$ separately, resulting in an $Nr \times Nc$ array of S-box outputs, which forms the output of `SubBytes`.
2. `ShiftRows` - Row i of the internal state is cyclically rotated to the left by $i - 1$ positions, for $i = 1, \ldots, Nr$. Note that the top row is left unchanged (and even the full state if $Nr = 1$).
3. `MixColumns` - The internal state is multiplied by the matrix MC, which depends on the specific value of Nr.
4. `AddRoundKey` - The $Nr \times Nc$ round key is xor-ed to the internal state. The key schedule itself will be described later.

Formally, the i-th round function F_i is defined as

$$F_i(X) = \begin{cases} \texttt{AddRoundKey}(\texttt{MixColumns}(\texttt{ShiftRows}(\texttt{SubBytes}(X))), K_i) & \text{for } i < R, \\ \texttt{AddRoundKey}(\texttt{ShiftRows}(\texttt{SubBytes}(X)), K_i) & \text{for } i = R, \end{cases}$$

where X is an $Nr \times Nc$ array of bytes. Furthermore, an initial `AddRoundKey` is applied before running the round functions.

Graphically, an SR*-encryption can be represented as in Figure 5.7, where the operation names are abbreviated in the straightforward manner (i.e. `SB` is `SubBytes`, etc.), and `AddRoundKey`, as that is basically a bitwise xor, is denoted by $\oplus$.

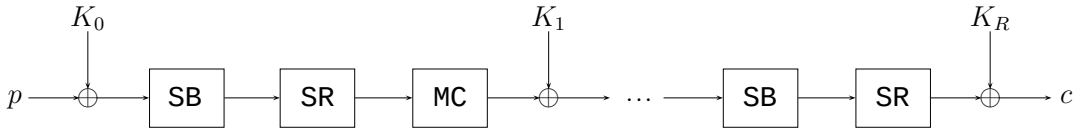

Figure 5.7: SR* encryption process.

The SR* key schedule uses the same non-linear operation as the round function, so there is actually only one S-box involved in the whole cipher. It also employs an additional round constant RC_i, whose value depends on the round index i. The input to the key schedule is the b-bit encryption key K, regarded as an $Nr \times Nc$ array of bytes. This key, also denoted by K_0, is added to the plaintext in the initial application of `AddRoundKey`. The round keys are then defined recursively, depending on the value of Nc. The detailed recursions can be found in Appendix B.

The AES with 128-bit key length is then equal to SR*$(10, 4, 4)$. The AES versions with key lengths 192 and 256 bits are defined in a slightly different manner and are not considered here.

Example 5.3. Consider the second round of SR*$(4, 2, 2)$, and assume its input is given by

$$\begin{pmatrix} 11011010 & 01101001 \\ 01011000 & 00001111 \end{pmatrix} = \begin{pmatrix} \texttt{DA} & \texttt{69} \\ \texttt{58} & \texttt{0F} \end{pmatrix},$$

and the round key K_2 by

$$K_2 = \begin{pmatrix} 00000101 & 11001101 \\ 11101010 & 10000001 \end{pmatrix} = \begin{pmatrix} \texttt{05} & \texttt{CD} \\ \texttt{EA} & \texttt{81} \end{pmatrix}.$$

Then the output of the round function is obtained as follows:

$$\begin{pmatrix} \texttt{DA} & \texttt{69} \\ \texttt{58} & \texttt{0F} \end{pmatrix} \xrightarrow{\texttt{SB}} \begin{pmatrix} \texttt{57} & \texttt{F9} \\ \texttt{6A} & \texttt{76} \end{pmatrix} \xrightarrow{\texttt{SR}} \begin{pmatrix} \texttt{57} & \texttt{F9} \\ \texttt{76} & \texttt{6A} \end{pmatrix} \xrightarrow{\texttt{MC}} \begin{pmatrix} \texttt{8F} & \texttt{7A} \\ \texttt{CD} & \texttt{47} \end{pmatrix} \xrightarrow{\oplus K_2} \begin{pmatrix} \texttt{8A} & \texttt{B7} \\ \texttt{27} & \texttt{C6} \end{pmatrix}$$

To determine the first (i.e. leftmost) column of the third round key, we need the S-box $\mathcal{S}$ and the round constant $RC_3 = \texttt{04}$:

$$K_3^1 = \begin{pmatrix} \mathcal{S}(\texttt{81}) \\ \mathcal{S}(\texttt{CD}) \end{pmatrix} + \begin{pmatrix} \texttt{04} \\ \texttt{00} \end{pmatrix} + \begin{pmatrix} \texttt{05} \\ \texttt{EA} \end{pmatrix} = \begin{pmatrix} \texttt{0D} \\ \texttt{57} \end{pmatrix}.$$

The second column is determined by adding the second column of K_2 to the first column of K_3:

$$K_3^2 = \begin{pmatrix} \texttt{0D} \\ \texttt{57} \end{pmatrix} + \begin{pmatrix} \texttt{CD} \\ \texttt{81} \end{pmatrix} = \begin{pmatrix} \texttt{C0} \\ \texttt{26} \end{pmatrix}.$$

Hence, the third round key is

$$K_3 = (K_3^1\ K_3^2) = \begin{pmatrix} \texttt{0D} & \texttt{C0} \\ \texttt{57} & \texttt{26} \end{pmatrix}.$$

5.4.2 Previous attacks

The AES has been the subject of numerous articles, in which its security is analyzed using both existing and new techniques. We summarize some of the most notable ones.

Related key attacks

The related key attack was first introduced by Biham in [Bih93]. In this type of attack, it is assumed that the attacker has access to several plain-/ciphertext pairs that were encrypted using unknown keys that have a certain known relation (e.g. the last 20 bits may be the same). The version described in [FKL00] can break nine-round AES-256 (i.e. the AES version with a key length of 256 bits) using $O(2^{85})$ chosen plaintexts. It has a time complexity of $O(2^{224})$, which is considerably faster than the exhaustive search. Note that this version of AES has fourteen rounds.

Integral cryptanalysis

Another type of attack suggested by Ferguson et al. is integral cryptanalysis, although it is also referred to as the saturation, structural or partial sum attack [FKL00]. It was introduced by Knudsen in an attack on the cipher Square [DKR97], upon which the AES was based. Integral cryptanalysis can be seen as an extension of differential cryptanalysis (cf. 5.3.2), which uses sets of chosen plaintexts of which a particular part is held constant, while the rest runs through all possibilities. At present, the best implementation can break the seven-round version of AES-128 and the eight-round versions of AES-192 and AES-256. The attack on AES-128 requires $O(2^{120})$ operations and $O(2^{122})$ chosen plaintexts.

Multivariate equations analysis

As already mentioned in Chapter 2, a typical cipher can be described as a huge system of multivariate algebraic equations. Because of the particular form of AES' S-box, the corresponding equations for AES are quadratic (i.e. the equations describing AES contain no products of three or more variables). Solving the equation system essentially breaks the cipher, but is generally very hard to perform. Numerous articles describe algorithms for solving multivariate quadratic equations, which includes the XSL (eXtended Sparse Linearization) attack by Courtois and Pieprzyk [CP02] that has been the source of a lot of controversy in the cryptographic society. Although it was presented as a ground breaking attack that might be able to break AES, this claim has never been taken too seriously.

In [CMR05], the small scale AES-variant SR* is embedded into a new, larger cipher called BES (Big Encryption System), which is supposedly easier to break than SR* itself. The equations are then solved by a Gröbner basis-approach. The paper does not give any estimates on the time complexity of the attack, but some experimental timing results are available, which indicate that this approach is not very useful from a practical point of view. For example, using this attack, the eight-bit cipher SR*$(R, 1, 1)$ is broken for $R = 4$ after over 20,000 seconds of computation. For comparison, the method using MRHS equations can break this cipher for any value of R in less than a second.

5.4.3 Representation using MRHS equations

To describe the cipher using MRHS equations, we introduce suitable variables. Since the key schedule contains a non-linear operation, we are unable to simply write each round key bit as a linear combination of the bits in the encryption key K. However, note that the l-th (with $l \neq 1$) column of round key K_i is defined in (B.1) as

$$K_i^l = K_i^1 \oplus K_{i-1}^2 \oplus \ldots \oplus K_{i-1}^l,$$

so each of these columns can be written as a linear combination of K_i^1 (the first column of round key K_i), and columns of the previous round key K_{i-1}. This implies that if we introduce the variables

$$K_0 = (k_0^1, \ldots, k_0^b)$$

to describe the initial round key (which is equal to the encryption key K), and the additional variables

$$K_i^1 = (k_i^1, \ldots, k_i^{8Nr}) \text{ for } i = 1, \ldots, R$$

to describe the first columns of the remaining round keys, we can fully describe any round key as a linear combination of the variables. Hence, the number of variables and symbols

required to represent the key schedule is

$$\begin{aligned} n_{\text{key}} &= b + 8RNr = 8Nr(Nc + R), \\ M_{\text{key}} &= RNr. \end{aligned}$$

The internal state is described by introducing the state variables

$$X_i = (x_{i,1}, \ldots, x_{i,b}) \text{ for } i = 1, \ldots, R,$$

which represent the outputs of the non-linear operations `SubBytes` contained in the round functions. However, we may omit the state variables X_R, if we note that the output of `SubBytes` in the last round can be written as a linear combination of the last round key and the ciphertext c, which is known. The number of required variables and symbols for the encryption process is therefore

$$\begin{aligned} n_{\text{enc}} &= (R-1)b = 8(R-1)NrNc, \\ M_{\text{enc}} &= RNrNc. \end{aligned}$$

Hence, the total number of variables and symbols needed to describe SR*(R, Nr, Nc) is

$$\begin{aligned} n &= n_{\text{enc}} + n_{\text{key}} = 8RNr(Nc + 1), \\ M &= M_{\text{enc}} + M_{\text{key}} = RNr(Nc + 1). \end{aligned}$$

Setting up the equations requires a bit more attention than in the case of DES, but is still a relatively easy task, if we keep track of which variables go into and come out of the S-boxes.

Example 5.4. Consider SR*$(R, 2, 2)$, and the operation `SubBytes` employed in the second round function. Note that each round function uses the S-box four times, once for each entry in the 2×2 internal state array. In this case, the output of `SubBytes` is given by the variables X_2, represented as

$$X_2 = \left(\begin{array}{c|c} x_2^1 \ldots x_2^8 & x_2^{17} \ldots x_2^{24} \\ \hline x_2^9 \ldots x_2^{16} & x_2^{25} \ldots x_2^{32} \end{array} \right). \tag{5.6}$$

To determine its input, we start at the last known internal state (which is just after `SubBytes` in the first round function) and apply the operations accordingly.
The internal state after `ShiftRows` in the first round function can be easily determined by

$$\left(\begin{array}{c|c} x_1^1 \ldots x_1^8 & x_1^{17} \ldots x_1^{24} \\ \hline x_1^9 \ldots x_1^{16} & x_1^{25} \ldots x_1^{32} \end{array} \right) \xrightarrow{\text{SR}} \left(\begin{array}{c|c} x_1^1 \ldots x_1^8 & x_1^{17} \ldots x_1^{24} \\ \hline x_1^{25} \ldots x_1^{32} & x_1^9 \ldots x_1^{16} \end{array} \right).$$

The application of `MixColumns` requires some more attention. To apply this operation, we multiply the internal state with the matrix MC. Note that the left column of the internal state at this point equals

$$B = \begin{pmatrix} x_1^1, \ldots, x_1^8 \\ x_1^{25}, \ldots, x_1^{32} \end{pmatrix} = \begin{pmatrix} x_1^1\theta^7 + \ldots + x_1^7\theta + x_1^8 \\ x_1^{25}\theta^7 + \ldots + x_1^{31}\theta + x_1^{32} \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$$

and the left column of the output of `MixColumns` is given by

$$MC.B = \begin{pmatrix} (\theta+1)B_1 + B_2 \\ B_1 + (\theta+1)B_2 \end{pmatrix}.$$

If we expand this and use that $\theta^8 = \theta^4 + \theta^3 + \theta + 1$ by the definition of θ, this column is equal to the product HB, where

$$H = \left(\begin{array}{cccccccc|cccccccc} 1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0\\ 1&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0\\ 1&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0\\ 1&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0\\ 1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\ \hline 1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0\\ 0&0&0&1&0&0&0&0&1&0&0&1&1&0&0&0\\ 0&0&0&0&1&0&0&0&1&0&0&0&1&1&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&0&1&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1 \end{array}\right) = \left(\begin{array}{c|c} H' & I_8 \\ \hline I_8 & H' \end{array}\right).$$

Using the matrix H', we can describe the output of `MixColumns` as a linear combination of the variables X_1. Since `AddRoundKey` is a simple bitwise xor, we can describe the input to the S-box when used to determine the top left entry of the output of the second round function as

$$u = H'(x_1^1, \ldots, x_1^8) \oplus (x_1^{25}, \ldots, x_1^{32}) \oplus (k_1^1, \ldots, k_1^8).$$

This describes the input to the S-box in terms of the variables, the output v is determined by the top left entry of Equation (5.6), hence

$$v = (x_2^1, \ldots, x_2^8).$$

Since there are no constants involved, the vector (u, v) must satisfy

$$(u, v) = [L_{\mathcal{S}}],$$

where

$$L_{\mathcal{S}} = \{(x, \mathcal{S}(x)) \mid x \in \mathbb{F}_2^8\}$$

is the right-hand side matrix associated with the S-box $\mathcal{S}$. This leads to an MRHS equation with $8 + 8 = 16$ rows and $2^8 = 256$ right-hand sides.

For $R = 3$, the 16×144 coefficient matrix of this MRHS equation is depicted in Figure 5.8. The indicated blocks correspond to the variables $K_0, K_1^1, K_2^1, K_3^1, X_1$ and X_2, respectively. The matrix H' is clearly visible. The operation `ShiftRows` is illustrated by the fact that the H' and I_8 block acting on the variables X_1 are separated. If `ShiftRows` were omitted from the round function, these blocks would be adjacent.

Figure 5.8: SR*$(3, 2, 2)$ second round coefficient matrix.

5.4.4 Experimental results

For different values of the gluing threshold Θ and the cipher parameters R, Nr and Nc (except for $Nr = Nc = 4$, which could not be simulated due to limitations in the used hardware), we have applied SOLVEMRHS to the systems of MRHS equations derived from SR*(R, Nr, Nc). For guessing method G_2 (i.e. guessing key variables in their ascending order $k_0^1, k_0^2, \ldots$) the results are summarized in Table 5.3, which confirms the results published earlier in [RS07]. For the sake of completeness, the results for SR*$(R, 4, 4)$ from that article are also included. Several other guessing methods produced less favorable results which are not presented here. As an illustration, we mention that all eight-bit variants SR*$(R, 1, 1)$ were solved in less than a second, which is significantly faster than the Gröbner-basis approach suggested in [CMR05]. We will return to this in Section 6.1.

	$\Theta = 2^8$		$\Theta = 2^{16}$			
	SR*$(R, 1, 1)$	SR*$(R, 2, 1)$	SR*$(R, 2, 2)$	SR*$(R, 2, 4)$	SR*$(R, 4, 2)$	SR*$(R, 4, 4)$
R	8-bit key	16-bit key	32-bit key	64-bit key	64-bit key	128-bit key
3	0	5	16	48	48	112
4	0	8	16	48	48	112
5	0	8	16	48	48	112
6	0	8	16	48	48	112
7	0	8	16	48	48	112
8	0	8	16	48	48	112
9	0	8	16	48	48	112
10	0	8	16	48	48	112

Table 5.3: Number of guesses for SR* for various values of Θ.

As for the DES variants, there seems to be a trade-off between the allowed number of right-hand sides and the number of keybits needed to solve the system. In fact, there is only one case in which the following hypothesis does not precisely hold.

Hypothesis 5.2. If $\Theta = 2^l$ right-hand sides are allowed, then any instance of SR*(R, Nr, Nc) can be broken by guessing $b-l$ key bits, provided variables are guessed according to method G_2.

Note that the block length b is as in Equation (5.5).

We have tried many possible settings for the threshold Θ and the cipher parameters (R, Nr, Nc), but have not been able to find any contradiction to this hypothesis except SR*$(3, 2, 1)$ with $\Theta = 2^8$. Again, this does not yet imply a break of any of these ciphers as each of the $O(\Theta)$ right-hand sides needs to be checked at least once, but the trade-offs found for both DES and SR* are certainly remarkable.

The fact that we have not found any significant contradictions to Hypotheses 5.1 and 5.2 does not imply that a similar statement may be made for any suitable cipher. In fact, recently started research on our method applied to small-scale versions of the ciphers Noekeon [DPA00] and the already mentioned KeeLoq shows much weaker results, indicating that the attack is less efficient than the brute force attack, even for a very low number of rounds. On the other hand, the first results for reduced-round versions of the stream cipher Trivium [CP06] are actually better than those obtained for the DES and AES variants. However, the research on these ciphers is still ongoing, and we will not formally present the results here

as they have not yet been fully validated.

6

Conclusions and future work

We conclude this thesis with an overview of the presented material. Furthermore, we describe some topics for future research, including an alternative problem description that may provide an interesting perspective.

6.1 Conclusions

In this thesis, we have presented multiple right-hand side (MRHS) equations, and their possible use in the analysis of iterated block ciphers that consist of linear operations and S-boxes (like DES and AES). By introducing variables to describe the round keys and the state of the cipher at certain points during the encryption process, we can efficiently represent any involved S-box as an MRHS equation, where the number of right-hand sides equals the number of possible inputs to the S-box. Setting up such an equation for each S-box in the encryption process and key schedule gives a new representation of the cipher, which we believe is an interesting new tool for cryptanalysis, to compete (or possibly be combined) with already known approaches that either involve the solution of large systems of non-linear equations or SAT-instances.

The main advantage of our approach when compared to known techniques appears to be the fact that, although they are quite different from (ordinary) linear equations, MRHS equations have two linear properties:

- Elementary row operations may be applied to the involved matrices at any point, as long as the same operation is applied to both the coefficient and right-hand side matrices. This does not affect the solution set that corresponds to the particular equation;
- All relations are described using only linear combinations of the variables, i.e. no products of variables are involved.

The former is quite a useful observation when designing an algorithm for solving MRHS equation systems, since it may be possible that a particular sequence of row operations transforms an equation into an equation that is easier to solve. For example, the procedure for extracting linear equations described in Section 3.3.1 is based on this principle.

Another advantage is the fact that the size of the equations is not affected when the S-boxes become more intricate, provided the number of possible inputs and outputs remains the same. Typically, the efficiency of algorithms for solving multivariate algebraic equation

systems that represent ciphers is affected if the involved S-boxes become more complex: the more complex the S-boxes are, the more complex the corresponding equation system will be. The MRHS representation does not suffer from this drawback.

Finally, we mention the advantage of being able to efficiently extract linear equations that may be hidden in the cipher. To our knowledge, there is no similar procedure for algebraic equation systems.

It is important to realize that the algorithm SOLVEMRHS described in Section 3.4 with the particular implementation specified in Chapter 4 was not designed to solve a system of MRHS equations in the most efficient manner. Our first concern was to determine how many guesses are needed to obtain the solution, since this number is independent of the chosen implementation of SOLVEMRHS. The initial experimental results obtained for small-scale variants of DES and AES indicate that there is an interesting tradeoff between the maximum number of right-hand sides allowed when gluing the equations and the number of guesses that is needed to uniquely solve the system. Although this particular tradeoff was only found for variants of DES and AES, there may be more ciphers that have this interesting property.

As there are no practical estimates on the complexity of the studied algorithm available, at this point this tradeoff does not imply a successful attack on either DES or AES. On the contrary, the complexity of the algorithm is at least in the order of the number of allowed right-hand sides, since each of these must be processed at least once, in which case the complexity of our attack is at least equal to that of the exhaustive search.

But, when applied to the simple eight-bit AES-variant SR*$(n, 1, 1)$, our method, which solves the corresponding MRHS equation system in less than a second, significantly overcomes the Gröbner basis approach suggested in [CMR05], which requires over 20,000 seconds to solve the equivalent quadratic equation system. Although this difference is promising, it is important to realize that it gives hardly any information on the behavior of the algorithm when applied to larger, more complex systems. However, this comparison and the fact that many ciphers can be efficiently represented by a system of MRHS equations do support our belief that the study of this new type of equation makes an interesting and useful subject for future research, and that an MRHS representation is possibly more suitable for S-box based ciphers than a representation using algebraic equations, even though another breakthrough (most likely a more efficient procedure for agreeing) is needed to define a really useful attack based on this representation. The fact that, so far, practical attacks on real-life ciphers using algebraic equations have more or less failed to materialize has not discouraged the cryptographic community to keep research on this topic ongoing, and we see no reason why the same should not hold for attacks involving MRHS equations.

6.2 Future work

In this section, we describe some possible topics for further research.

6.2.1 Higher-probability guessing

If agreeing, gluing and extracting linear equations do not produce any new results, the algorithm applies the straightforward approach of simply guessing the value of a particular variable or linear combination of variables. Our results indicate that guessing the value of key variables in their ascending order is most helpful, at least for DES and AES variants. The

value of this guess is fully random, as there is no reason to assume that the probability that a particular key bit is zero is higher than the probability that it is one, or lower for that matter. We would like to be able to use guesses that have a higher probability of being correct. An improvement that comes to mind is to extend the procedure for extracting linear equations (that hold with certainty) into a procedure for extracting linear equations that hold with a specific probability. This is possible if we make an assumption involving the right-hand sides in the studied MRHS equation system.

Let

$$S_1 : A_1X = [L_1],\ \ldots,\ S_M : A_MX = [L_M] \tag{6.1}$$

be a system of MRHS equations that fully describes a particular cipher. Recall that the number of right-hand sides in symbol S_i is denoted by s_i. We have already assumed that this system has a unique solution X, which implies that there is a unique set of columns $\{l_1, \ldots, l_M\}$, where $l_i \in L_i$, such that

$$A_1X = l_1,\ \ldots,\ A_MX = l_M,$$

since the right-hand side matrices do not contain duplicate elements. Note that determining these columns is equivalent to solving system (6.1). We then assume that the index of column l_i is uniformly distributed over the set $\{1, \ldots, s_i\}$ for $i = 1, \ldots, M$. In other words, it is assumed that

$$\Pr(l_i = L^j) = \frac{1}{s_i}$$

for $j = 1, \ldots, s_i$. Note that if the assumption does not hold, this implies that the indices are biased, a property that may possibly be exploited in another improvement of the algorithm. From now on, we will make this assumption without mention.

Now, observe the procedure for extracting linear equations from symbol S over $\mathbb{F}_q$. It basically applies elementary row operations (described by the vector t) to the right-hand side matrix L, and in case tL equals the all-r vector it outputs the linear equation

$$tAX = r, \tag{6.2}$$

which holds with certainty. However, suppose there exists a t such that tL contains a fraction of r's that is significantly larger than $\frac{1}{q}$. Then, by the above assumption, the probability that (6.2) holds is equal to this fraction, and in particular larger than $\frac{1}{q}$. That implies that (6.2) has a higher probability of being correct than simply guessing the value of a variable randomly, which is correct with probability $\frac{1}{q}$. A linear equation that holds with maximal probability (unequal to one) will be referred to as a most likely guess. The straightforward but very inefficient way to determine a most likely guess is to try all possible vectors t and determine which lead to a tL with a maximal number of r's, with $r \in \mathbb{F}_q$. Note that there may be more than one most likely guess, as illustrated in the following example.

Example 6.1. Consider the symbol over $\mathbb{F}_2$ given by

$$S : \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} X = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

and observe that no linear equations can be extracted, since the right-hand side matrix does not generate either $\underline{0}$ or $\underline{1}$. However, from the second row we can extract the linear equation

$$x_2 + x_3 + x_4 = 1,$$

which holds with probability $\frac{3}{4}$, and there are more equations that hold with this probability. The set of all most likely guesses in this case is given by

$$\begin{cases} x_2 + x_3 + x_4 = 1 \\ x_1 + x_3 + x_4 + x_5 = 0 \\ x_2 + x_3 + x_5 = 1 \\ x_1 + x_3 = 1. \end{cases}$$

It is possible that the set of most likely guesses forms an inconsistent equation system (as is the case here). We should always check whether a most likely guess contradicts linear equations extracted or guessed earlier. If so, the guess is immediately rejected.

Note that it is not straightforward to determine a vector t such that tL has a maximal number of, say, zeroes, which would correspond to a most likely guess with value zero. Readers familiar with coding theory may recognize this problem as closely related to the problem of determining the minimum Hamming distance of the linear code that has L as its generator matrix. This latter problem is notoriously difficult, and was proven to be $\mathcal{NP}$-hard in [Var97]. The most likely guess with arbitrary value r among a set of symbols $S_1, \ldots, S_M$ is even harder to determine. In fact, the straightforward method described above may be the most efficient way to determine the most likely guess, and we should be aware that incorporating this procedure in an alternative algorithm for solving MRHS equation systems will lead to a considerable increase in time complexity. We can not expect to find an algorithm that finds a most likely guess in polynomial time, as the existence of such an algorithm would contradict Vardy's result.

We already observed (cf. Table 5.1) that the number of required guesses greatly depends on which particular variables are guessed. Hence, it is not unlikely that, even though the probability that a made guess is correct is increased by this method, so is the number of guesses needed to solve the system. From this, combined with the inevitable increase in time complexity required to determine the most likely guess(es), we may actually expect an algorithm that is less efficient than the one that guesses the values of variables randomly. Initial experimental results on DES and AES equation systems, where we determined the most likely guesses by the straightforward and costly method described earlier, confirm this expectation: the made guesses did have a larger probability of being correct, but we needed considerably more guesses to solve the systems. The cost for using most likely guesses was simply too high, even when disregarding the additional complexity of determining the most likely guesses.

Based on these observations, we expect that transforming the current "random-value" guessing procedure into a procedure that uses most likely guesses instead will affect the workload of the algorithm too severely, and is therefore not a practical option to improve our attack.

6.2.2 t-agreeing

One of the main drawbacks of the current algorithm is that pairwise agreeing is typically unable to delete sufficiently many columns, making gluing of symbols necessary. Since the number of right-hand sides tends to explode when symbols are glued, we are interested in a procedure that is able to detect more inconsistencies than pairwise agreeing, possibly avoiding the use of the gluing procedure altogether. The natural candidate is to extend pairwise agreeing into a new procedure, which we will call t-agreeing.

Recall Definition 3.5 of pairwise agreement. We can easily extend this definition into a definition of t-agreement as follows.

Definition 6.1. *A set of symbols $S_1, \ldots, S_t$ t-agrees if and only if for all $s \in \{1, \ldots, t\}$ it holds that for all $l_i \in L_i$ with $i \neq s$, there exists an $l_s \in L_s$ such that the simultaneous linear equation system*

$$A_1X = l_1 \text{ and } \ldots \text{ and } A_tX = l_t$$

is consistent. A set of $M > t$ symbols t-agrees if and only if each subset of t symbols t-agrees.

For a set of t symbols, this induces t conditions for agreement (one for each possible value of s). Note that this definition for $t = 2$ is equivalent to Definition 3.5.

If a particular right-hand side in one of the involved symbols is fixed, we essentially put a number of constraints on the allowed solutions. By fixing more right-hand sides, we increase the number of constraints. So the number of possible solutions decreases, unless the new constraints are linear combinations of the already set constraints, in which case the number of solutions remains the same. In fact, if the number of constraints is made sufficiently large, there is an increasing probability that no solution exists. In the extreme case of $t = M$, there is only one set of columns that results in a consistent system (as there is a unique solution to the initial system). Hence, the probability that fixing $t - 1$ columns leads to an inconsistent system approaches 1 for $t \to M$. Even though t-agreeing is clearly a more complex procedure than the current 2-agreeing, the probability that it is able to detect inconsistencies becomes higher for increasing t, so the additional computational cost may be acceptable.

The idea of exploiting t-agreement in an improved version of the algorithm is straightforward at first sight: check each set of t symbols for inconsistencies and delete columns accordingly. Since the probability of finding inconsistencies increases for $t \to M$, the probability of being able to delete columns also increases, so we should be able to create an algorithm for solving MRHS equations that uses t-agreeing and extracting linear equations, but not gluing, as was our intent. There is, however, one huge advantage pairwise agreeing has over all other cases: only if $t = 2$ we can delete columns if inconsistencies are encountered.

For example, imagine the 2-agreeing of symbols S_i and S_j, and suppose that for the first column of L_i there exists no column in L_j such that the resulting linear system is consistent. We have already established that the first column of L_i may then be deleted, as it can not contribute to the solution. Now, we 3-agree the symbols S_i, S_j and S_k to illustrate the difference. Suppose that we find an inconsistency if the first columns of L_i and L_j have been fixed. This implies that this combination of columns does not contribute to the solution, but we can conclude nothing about the columns individually. So, instead of deleting a whole column, we have to somehow mark the pair (L_i^1, L_j^1), to indicate that this combination does not contribute to the solution. Hence, t-agreeing does not affect the right-hand side matrices directly, and we will have to keep track of which combinations of right-hand sides lead to consistent systems and which do not.

At this point we do not present an efficient algorithm for t-agreeing, but introduce it as a useful improvement to the SOLVEMRHS algorithm. As an initial proposal, we analyze an extension of the pairwise agreeing algorithm AGREEPAIR of Section 4.2. By setting $t = M$, we obtain an alternative problem description which may provide an interesting perspective on the problem of solving MRHS equation systems.

6.2.3 Alternative problem description

Consider the problem of solving a set of M symbols as in (6.1). Now, define the joint coefficient matrix A as

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_M \end{pmatrix}$$

and let $\kappa = \sum_{j=1}^{M} k_j$ denote the number of rows of A. Then, determine the $\kappa \times s_i$ matrices

$$T_i = \begin{pmatrix} \mathcal{O} \\ L_i \\ \mathcal{O} \end{pmatrix}$$

for $i = 1, \ldots, M$, where the number of rows in the top $\mathcal{O}$-block is $\sum_{j=1}^{i-1} k_j$. So

$$T_1 = \begin{pmatrix} L_1 \\ \mathcal{O} \\ \vdots \\ \mathcal{O} \end{pmatrix}, \quad T_2 = \begin{pmatrix} \mathcal{O} \\ L_2 \\ \vdots \\ \mathcal{O} \end{pmatrix} \ldots, \quad T_M = \begin{pmatrix} \mathcal{O} \\ \vdots \\ \mathcal{O} \\ L_M \end{pmatrix}.$$

Now, compute the matrix $U = U(A)$ such that UA has all its

$$r = \kappa - \rho(A)$$

zero-rows at the bottom. Since it is natural to assume that each of the n variables is involved in at least one of the symbols (otherwise it can be eliminated first) and that A contains more than n rows (otherwise there need not be a unique solution), we may conclude that $\rho(A) = n$. Now, let the matrices P_i be given by the last r rows of UT_i, i.e.

$$P_i = (UT_i)^{(-r)}.$$

Finally, let $\alpha = (\alpha_1, \ldots, \alpha_M)$ be such that

$$\sum_{i=1}^{M} P_i^{\alpha_i} = \underline{0}. \tag{6.3}$$

We may then use this equation in an alternative problem description.

Theorem 6.1. Solving (6.3) for α is equivalent to solving (6.1) for X.

Proof. The proof of this statement is similar to the proof of Theorem 4.1, with $p_i = P_i^{\alpha_i}$ and $l_i = L_i^{\alpha_i}$ for $i = 1, \ldots, M$. □

Let us study this alternative description a bit closer. Roughly it says that given a set of matrices of κ rows over $\mathbb{F}_q$, we must select exactly one column from each matrix such that the mod-q sum of these columns equals the zero-vector. By the equivalence to the problem of solving an MRHS equation system, this alternative problem has a unique solution. The challenge is now to devise an algorithm that finds this solution significantly faster than simply trying all possible combinations of columns. Since matrix P_i contains s_i columns, the number of possible combinations is

$$\prod_{j=1}^{M} s_j,$$

which may be a huge number.

The problem of finding the combination of columns that sum to zero is clearly a very difficult problem; it is not unlikely that it can be proven to be $\mathcal{NP}$-hard. However, it is important to realize that the matrices P_i are not random, as they are derived from equations that describe a cipher. For this reason, the problem of solving Equation (6.3) for $\alpha = (\alpha_1, \ldots, \alpha_M)$ may be a much simpler problem than the version in which the P_i are random.

To illustrate the fact that the matrices P_i are, in fact, far from random, consider the four-round DES variant described in Section 5.3, which results in an MRHS equation system of $M = 32$ symbols, each of which has $2^6 = 64$ right-hand sides and ten rows. We then determine the matrices P_i, which have dimensions 200×64. It turns out that these matrices contain a lot of zero-rows. Furthermore, the location of these zero-rows is independent of the initial right-hand side matrices L_i, which implies that they are independent of the particular plain-/ciphertext pair used in the corresponding attack. Hence, this observation holds for all MRHS equation systems derived from four-round DES, and in fact we can describe a similar observation for all variants of DES and even AES. For several values of j, we have summarized this for four-round DES in Table 6.1, which gives the indices of the matrices P_i whose j-th row is non-zero. For example, there are only two matrices, P_{16} and P_{32}, whose first four rows are non-zero.

Row index j	Matrix indices i
1	16, 32
2	16, 32
3	16, 32
4	16, 32
5	5, 32
19	6, 15, 31
20	7, 31
85	2, 12, 15,24
86	8, 15, 24
139	5, 9, 10, 19
140	3, 13, 19

Table 6.1: Four-round DES matrices P_i that have a non-zero row j.

For four-round DES, the maximum number of matrices that have a non-zero row j is four (which is obtained for $j = 85, 139, 147$ and 148), the minimum number is two and the average, over all j, is 2.48. Note that if the matrices P_i were fully random, these numbers would all be 32 (the number of symbols) with probability close to 1. For the full version of DES, these numbers are 10, 2 and 6.03, respectively. For random matrices, these would be 128 with high probability.

This illustrates that the matrices P_i are far from random, and that the problem of solving (6.3) for instances derived from ciphers is possibly not as hard as may be expected. The fact that this alternative problem is equivalent to the original problem of solving an MRHS equation system may provide a new perspective that turns out useful when designing a more efficient version of the SOLVEMRHS algorithm.

6.2.4 Further topics

Without going into too much detail, we suggest some additional projects in arbitrary order.

Further experiments using the current algorithm

A straightforward subject is to keep applying our method to other ciphers, and possibly encounter the "time-memory-number of guesses" tradeoff for these other instances as well. Since we have already tried a number of block ciphers, we are especially interested in the behavior of our attack when applied to stream ciphers, which are also suitable for this type of attack, provided the involved non-linear functions can be viewed as an S-box.

Increase the number of known plaintexts

In this thesis and most of the literature on the subject of MRHS equations in cryptanalysis, only one known plain-/ciphertext pair is used to attack the investigated cipher. There is a distinct probability that increasing this number may result in a more efficient attack, in which less key variables need to be guessed, or the probability that a particular guess is correct can be improved. The addition of one or more plain-/ciphertext pairs that have been encrypted using the same encryption key will lead to an increase of the number of internal state variables, but not of the number of (round) key variables. The number of equations will increase linearly, e.g. if the number of plain-/ciphertext pairs is doubled, so is the number of equations.

Estimate complexities

As already mentioned, we are very interested in an estimate of the time complexity (and to a lesser extent the memory requirement) of our algorithm. Although we are aware that it is currently not strictly faster than the exhaustive search for larger systems, we will be able to formally compare our method to known approaches once a complexity estimate is available.

Exploit properties of coefficient matrices

Typically, the coefficient matrices describing S-boxes contain only a very low number of non-zero elements (they are said to be very sparse). For example, for six-round DES the matrices A_i contain about 1% ones or less, and for higher-round DES this number is even lower. This is due to the fact that the in- and outputs of an S-box only depend on a low number of variables, for instance the variables describing two internal states and one round key. At the very least, this implies that the memory requirements for solving MRHS equation systems that describe ciphers can be improved considerably by using more efficient methods for storing the coefficient matrices.

However, the theory of sparse matrices is quite a well-known topic in linear algebra, e.g. [DER89], and there exist numerous algorithms for efficiently processing them. It may be possible to exploit one of the techniques from this field in our attack. The algorithm suggested in this thesis does not explicitly use the sparseness of the coefficient matrices at all, and we expect there is room for improvement.

Exploit properties of right-hand side matrices

Likewise, we may exploit some of the properties of the right-hand side matrices, as these are not fully random, given a set of symbols. For example, the right-hand side matrices describing one of the AES variants are all equal up to the addition of a vector that depends on the value of the plain- and/or ciphertext. Besides the obvious improvement in the storage of these matrices, there may be an efficient way of simultaneously processing them.

Hybrid systems

Our algorithm does not use the fact that the operations of the involved S-boxes may be explicitly known in an algebraic form (as is the case for AES). There may be a way to incorporate this explicit form in a hybrid system of equations, in which both MRHS and algebraic equations are used to describe the cipher. This system will no longer have the linear properties described in Section 6.1, but may be more efficiently solvable nonetheless.

Appendices

A

DES details

The following pages specify the permutations and S-boxes used in the Data Encryption Standard (DES) as described in Section 5.3.

Before applying the round functions, the 64-bit plaintext block p is permuted using IP to obtain (p^l, p^r). After running the round functions, the ciphertext is obtained by applying FP (which is the inverse of IP).

The round function itself contains the S-boxes $\mathcal{S}_1, \ldots, \mathcal{S}_8$ and the two additional permutations E and P. The expansion permutation E transforms a 32-bit half-block into 48 bits, which are subsequently split into eight equally-sized parts that are sent to the S-boxes. Note that 16 of the 32 bits of the input to E occur twice in its output. The mixing permutation P shuffles the 32 bits coming from the S-boxes to produce the output of the round function.

The key schedule employs two permutations PC_1 and PC_2 and a cyclic rotation. The former specifies which 56 bits of the 64-bit encryption key form the left and right parts of the key schedule and in which order, whereas the latter shuffles cyclically rotated versions of these halves and outputs the 48-bit round keys themselves. The number of rotations depends on the round index and is also specified.

The permutation tables describe which bit of the input occurs at which position(s) in the output. For example, the first output bit of IP is given by the 58-th bit of its input. The S-box tables should be read as follows: the row of the table is determined by the outer two bits of the input, i.e. the first row is selected if the input is of the form 0xxxx0, the second row if it is 0xxxx1, etc. Similarly, the column is determined by the inner four bits of the input. The output of the S-box is then the value in the table, converted into the binary format. For example, $\mathcal{S}_3(011010) = 1011$.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Initial permutation IP

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Final permutation FP

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Expansion permutation E

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Mixing permutation P

Left part	57	49	41	33	25	17	9
	1	58	50	42	34	26	18
	10	2	59	51	43	35	27
	19	11	3	60	52	44	36
Right part	63	55	47	39	31	23	15
	7	62	54	46	38	30	22
	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

Key schedule permutation PC_1

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Key schedule permutation PC_2

Round index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of rotations	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Number of rotations in the key schedule

$\mathcal{S}_1$	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

$\mathcal{S}_2$	15	1	8	14	6	1	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

$\mathcal{S}_3$	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

$\mathcal{S}_4$	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

$\mathcal{S}_5$	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	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

$\mathcal{S}_6$	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

$\mathcal{S}_7$	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

$\mathcal{S}_8$	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

B

SR* details

This appendix contains details of the cipher SR*(R, Nr, Nc) of Section 5.4.

The plaintext is viewed as an $Nr \times Nc$ array of bytes, and an initial round key (equal to the encryption key) is added before the R round functions are applied. The round function contains a round key addition, a cyclic shift, a matrix multiplication and an S-box. The latter two operations will be specified below.

- `MixColumns`. This operation is performed by viewing the internal state as an $Nr \times Nc$ matrix of polynomials over $\mathbb{F}_2$ with degree at most 7. The output is then determined by multiplying this matrix by the matrix MC, whose value depends on the number of rows as specified in Table B.1.

Number of rows	MC
$Nr = 1$	$\begin{pmatrix} 1 \end{pmatrix}$
$Nr = 2$	$\begin{pmatrix} \theta+1 & 1 \\ 1 & \theta+1 \end{pmatrix}$
$Nr = 4$	$\begin{pmatrix} \theta & \theta+1 & 1 & 1 \\ 1 & \theta & \theta+1 & 1 \\ 1 & 1 & \theta & \theta+1 \\ \theta+1 & 1 & 1 & \theta \end{pmatrix}$

Table B.1: MC for possible choices of Nr.

  Remember that `MixColumns` is omitted from the last round function.

- `SubBytes`. This operation is the non-linear part of the round function, and it is essentially an 8×8 S-box, applied to each entry of the internal state array. The S-box itself is generated by applying three simple steps, of which the first is non-linear:

  1. View the input as a polynomial and determine its multiplicative inverse in $\mathbb{F}_{2^8}$, where the inverse of the polynomial 0 is defined to be 0;
  2. Regard this inverse as an 8-bit vector, and multiply with

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

3. Add the constant vector $C_{SB} = (011000011) = \texttt{63}$ to obtain the output of the S-box.

Note that M_{SB} differs from the matrix given in the official specification [Nat01], as we use a different method for ordering the bits in a byte, cf. 1.1. However, the resulting lookup table, depicted in Table B.2, is identical. It is to be read as follows: the first four bits (in hexadecimal) of the input determine the row of the output; the last four bits determine the column. So, $\mathcal{S}(\texttt{DA}) = \texttt{57}$ and $\mathcal{S}(\texttt{58}) = \texttt{6A}$.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1_	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2_	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3_	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4_	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5_	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6_	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7_	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8_	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9_	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A_	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B_	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C_	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D_	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E_	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F_	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table B.2: SR* S-box (in hexadecimals)

The key schedule is defined recursively, depending on the particular value of Nc. It employs the operation `SubBytes` and the round constants

$$RC_i = \theta^{i-1}.$$

Let i denote the round index, and let $K_0 = K$ be the initial round key. Recall that the l-th column of round key K_i is denoted by K_i^l. Then the recursion is as follows:

- One column ($Nc = 1$):

$$K_i^1 = \texttt{SubBytes}(K_{i-1}^1) \oplus RC_i'.$$

- More than one column ($Nc \neq 1$):

$$K_i^l = \begin{cases} \texttt{SubBytes}(\mathcal{R}(K_{i-1}^{Nc})) \oplus RC_i' \oplus K_{i-1}^1 & \text{for } l = 1, \\ K_i^{l-1} \oplus K_{i-1}^l & \text{for } l = 2, \ldots, Nc. \end{cases} \tag{B.1}$$

Here, $\mathcal{R}$ is an operator that cyclically shifts its input upwards by one byte, e.g.

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} \xrightarrow{\mathcal{R}} \begin{pmatrix} \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_1 \end{pmatrix} \tag{B.2}$$

where $\alpha_1, \ldots, \alpha_4$ are bytes. Note that the definition of the operator $\mathcal{R}$ in (B.2) slightly differs from the one used in [CMR05], in which $\mathcal{R}$ reverses its input. However, only if we use $\mathcal{R}$ as described here will the cipher SR*$(10, 4, 4)$ be equivalent to the AES. The round constant vectors RC_i' are determined by the value of Nr as summarized in Table B.3.

Number of rows	RC_i'
$Nr = 1$	$\begin{pmatrix} RC_i \end{pmatrix}$
$Nr = 2$	$\begin{pmatrix} RC_i \\ 0 \end{pmatrix}$
$Nr = 4$	$\begin{pmatrix} RC_i \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Table B.3: RC_i' for possible choices of Nr.

C

Reference code

This appendix contains some of the code used to obtain most of the results described in Chapter 5. All experiments were run on a 3.0 GHz personal computer with 1 GB of RAM. Note that for the larger experiments (i.e. those with 2^{12} right-hand sides or more), a more efficient alternative implementation was used. However, for readability, we only include the basic code, which has not been optimized in any sense.

The simulation was written in *Mathematica*, and even though the overhead of using this sofware is considerable, the results were obtained in a reasonable amount of time. For the sake of completeness, we mention that the results contained herein are independent of those mentioned in any earlier publications. Where already published results have been included, this is indicated explicitly.

It is important to note that in our simulations, the right-hand side matrices are represented as lists of rows, instead of columns. So, strictly speaking, we use the transposes of the right-hand side matrices. Symbols are represented by two-dimensional lists, where the first entry is the coefficient matrix and the second entry is the (transpose of the) right-hand side matrix. Finally, note that we have implemented the case $q = 2$ only. For the considered ciphers this suffices. The procedure names are self-explanatory.

The particular implementation of the function $U(A)$ slightly differs from the one described in Section 4.2.1. Instead of applying the first part of Gaussian elimination, we let *Mathematica* determine the nullspace of the matrix A, and replace rows of an intermediate matrix (which is initially an identity matrix) with the vectors of this nullspace, such that the resulting matrix U is non-singular and the product UA satisfies requirements U1 and U2. The same holds for the procedure EXTRACTLINEAR, where we prefer to let *Mathematica* solve the linear equations directly, instead of explicitly transforming involved matrices into reduced row echelon form. These procedures are considerably faster than those in which the built-in command `RowReduce` for Gaussian elimination was used.

```
U[min_]:=Module[{},
    mat=IdentityMatrix[Length[min]];
    R=NullSpace[Transpose[min],Modulus->2];

    For[s=1,s<=Length[R],
    rr=Length[min];
    While[R[[s,rr]]==0,rr--];
```

```
        (*Find appropriate location*)
    mat=Delete[mat,rr];
    mat=Append[mat,R[[s]]];
        (*Insert nullspace-vector at the bottom*)
    s++];

    Return[{mat,Length[R]}];
    ]


Agree[in1_,in2_]:=Module[{},
    M=Join[in1[[1]],in2[[1]]];
    out1=in1[[2]];
    out2=in2[[2]];
    {u,r}=U[M];

    If[r>0,
    T1=Transpose[Table[PadRight[in1[[2,i]],Length[u]],
    {i,Length[in1[[2]]]}]];
    T2=Transpose[Table[PadLeft[in2[[2,i]],Length[u]],
    {i,Length[in2[[2]]]}]];
    Pr1=Transpose[Take[Mod[u.T1,2],-r]];
    Pr2=Transpose[Take[Mod[u.T2,2],-r]];
    C1=Complement[Pr1,Pr2];
    C2=Complement[Pr2,Pr1];
    If[C1=!={}||C2=!={},
    out1=Delete[in1[[2]],Flatten[Table[Position[Pr1,C1[[i]]],
    {i,Length[C1]}],1]];
    out2=Delete[in2[[2]],Flatten[Table[Position[Pr2,C2[[i]]],
    {i,Length[C2]}],1]];
    status=2,status=1],
        (*status=2: columns were deleted*)
        (*status=1: no columns were deleted*)
    status=1;
    ];

    Return[{status,{in1[[1]],out1},{in2[[1]],out2}}]
    ]


AgreeSet[in_]:=Module[{},
    l=Length[in];
    out=in;
    changed=True;

    While[changed,
    changed=False;
    Do[{a,out[[i]],out[[j]]}=Agree[in[[i]],in[[j]]];
    If[a==2,changed=True;
    Break[]],
    {i,l},{j,i+1,l}]];
```

```
    Print[Table[Length[out[[i,2]]],{i,l}]];
        (*prints the number of right-hand sides in*)
        (*each symbol after agreeing the set*)
    Return[out];
    ]


ExtractLinear[in_]:=Module[{},
    Quiet[
    HS=DeleteCases[Mod[NullSpace[in[[2]],Modulus->2].in[[1]],2],
    Table[0,{Length[in[[1,1]]]}]];

    NHS=Mod[LinearSolve[in[[2]],Table[1,{Length[in[[2]]]}],
    Modulus->2].in[[1]],2]];

    Return[{If[Head[HS]=!=Mod&&HS=!={},HS],If[Head[NHS]=!=Mod,NHS]}]
        (*returns coefficient lists of linear equations formatted as*)
        (*{{homogeneous equations},{non-homogeneous equations}}*)
    ]


Glue[in1_,in2_]:=Module[{},
    M=Join[in1[[1]],in2[[1]]];
    {u,r}=U[M];

    T1=Mod[Transpose[u.Table[PadRight[in1[[2,i]],Length[M]],
    {i,Length[in1[[2]]]}]],2];
    T2=Mod[Transpose[u.Table[PadLeft[in2[[2,i]],Length[M]],
    {i,Length[in2[[2]]]}]],2];
    If[r=!=0,
    B=Drop[Mod[u.M,2],-r];
        (*remove zero-rows*)
    Pr1=Transpose[Take[T1,-r]];
    Pr2=Transpose[Take[T2,-r]];
    P=Table[Flatten[Position[Pr2,Pr1[[i]]],1],{i,Length[Pr1]}];
        (*indices of those elements of Pr2 that equal Pr1[[i]]*)
    LL=
    Flatten[Table[Drop[BitXor[Transpose[T1][[i]],
    Transpose[T2][[P[[i,j]]]]],-r],{i,Length[T1[[1]]]},
    {j,Length[P[[i]]]}],1],
        (*if r=!=0, add those elements T1[[i]] and T2[[j]] for which*)
        (*Pr1[[i]]=Pr2[[j]], disregarding last r entries*)
    B=M;
    LL=Flatten[Table[BitXor[Transpose[T1][[i]],Transpose[T2][[j]]],
    {i,Length[T1[[1]]]},{j,Length[T2[[1]]]}],1]];
        (*if r=0, simply combine all possible pairs of elements*)

    Return[{B,LL}]
    ]
```

Used notations and parameters

A or A_i	Left-hand side or coefficient matrix
b	Block length
C	Set of ordered constants
$\mathbb{F}_q$	Finite field of q elements
K	Encryption key
K_i	Round key for round i
k	
- for algebraic equations	Maximal number of involved variables per equation
- for MRHS equations	Maximal number of rows
k_i	
- for algebraic equations	Number of variables involved in equation i
- for MRHS equations	Number of rows in equation i
L or L_i	Right-hand side matrix
M	Number of equations
Nc	Number of columns in the AES internal state array
Nr	Number of rows in the AES internal state array
n	Number of variables
q	Number of elements in the used finite field
R	Number of rounds of the studied block cipher
S or S_i	Symbol, i.e. $S : AX = [L]$ or $S_i : A_iX = [L_i]$
$\mathcal{S}$ or $\mathcal{S}_i$	S-box
s or s_i	Number of right-hand sides in symbol S or S_i
X	Set of ordered variables $\{x_1, \ldots, x_n\}$
λ	Round key length
Ω	Set of solutions, may be indexed to indicate the corresponding equation, e.g. Ω_{S_i}
$\rho(A)$	(Row) rank of matrix A
Θ	Maximal number of right-hand sides allowed when gluing
$\mathcal{O}$	All-zero matrix, dimensions determined by context
$\oplus$	Bitwise xor, i.e. addition modulo 2

$\underline{0}$	All-zero vector, length determined by context
$\underline{1}$	All-one vector, length determined by context
I_r	$r \times r$ identity matrix
$(x \mid\mid y)$	Concatenation of vectors x and y
$\langle x, y \rangle$	Inner product of vectors x and y
$S_i \circ S_j$	Gluing of symbols S_i and S_j
for $l \geq 1$:	
A_l	Row l of matrix A
A^l	Column l of matrix A
$A^{(l)}$	Submatrix formed by the first l rows of A
$A^{(-l)}$	Submatrix formed by the last l rows of A
$O(\ )$	if $f(n)/g(n) \rightarrow$ constant as $n \rightarrow \infty$, then $f(n) = O(g(n))$

The dimensions of the coefficient matrices A_i are $k_i \times n$, the dimensions of the right-hand side matrices L_i are $k_i \times s_i$.

Bibliography

[AKS04] Manindra Agrawal, Neeraj Kayal and Nitin Saxena. *Primes is in P*, 2004. Annals of Mathematics, vol. 160, issue 2, pp. 781-793.

[BB97] Eli Biham and Alex Biryukov. *An Improvement of Davies' Attack on DES*, 1997. Journal of Cryptology, vol. 10, issue 3, pp. 195-206.

[BDI08] Eli Biham, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller and Bart Preneel. *A Practical Attack on KeeLoq*. In Advances in Cryptology - EuroCrypt 2008, pp.1-18, 2008.

[Bih93] Eli Biham. *New Types of Cryptanalytic Attacks Using Related Keys*. In Advances in Cryptology - EUROCRYPT 1993, pp. 398-409, 1993.

[BS91] Eli Biham and Adi Shamir. *Differential Cryptanalysis of DES-like cryptosystems*, 1991. Journal of Cryptology, vol. 4, issue 1, pp. 3-72.

[BS92] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Full 16-Round DES*. In Advances in Cryptology - Crypto '92, pp. 487-496, 1992.

[BW93] Thomas Becker and Volker Weispfenning. *Gröbner bases: A Computational Approach to Commutative Algebra, 1st Edition*. Springer-Verlag, 1993.

[CKP00] Nicolas Courtois, Alexander Klimov, Jacques Patarin and Adi Shamir. *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*. In Advances in Cryptology - Eurocrypt 2000, pp. 392-407, 2000.

[CMR05] C. Cid, S. Murphy and M. Robshaw. *Small Scale Variants of the AES*. In 12th International Workshop, FSE 2005, Revised Selected Papers, pp. 145-162, 2005.

[Coo71] Stephen Cook. *The Complexity of Theorem Proving Procedures*. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151-158, 1971.

[CP02] Nicolas Courtois and Josef Pieprzyk. *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*. In Advances in Cryptology - Asiacrypt 2002, pp. 267-287, 2002.

[CP06] Christophe De Cannière and Bart Preneel. *Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principle*, 2006. eStream submitted papers.

[CW87] Don Coppersmith and Shmuel Winograd. *Matrix Multiplication via Arithmetic Progressions*. In Proceedings of the 19th Annual ACM Symposium on Theory of Computing, pp. 1-6, 1987.

[DER89] I. Duff, A. Erisman and J. Reid. *Direct Methods for Sparse Matrices*. Oxford Science Publications, 1989.

[Die04] Claus Diem. *The XL-Algorithm and a Conjecture from Commutative Algebra*. In Advances in Cryptology - Asiacrypt 2004, pp. 323-337, 2004.

[DKR97] Joan Daemen, Lars Knudsen and Vincent Rijmen. *The Block Cipher Square*. In Fast Software Encryption '97, pp. 149-165, 1997.

[DP60] Martin Davis and Hilary Putnam. *A Computing Procedure for Quantification Theory*, 1960. Journal of the Association for Computing Machinery, vol. 7, issue 3, pp. 201-215.

[DPA00] Joan Daemen, Michaël Peeters, Gilles van Assche and Vincent Rijmen. *Nessie Proposal: Noekeon*, 2000. The New European Schemes for Signatures, Integrity and Encryption (Nessie) Project.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.

[EMP75] William Ehrsam, Carl Meyer, Robert Powers, John Smith and Walter Tuchman. *Product Block Cipher System for Data Security*, 1975. United States Patent # 3,962,539.

[ES03] Niklas Een and Niklas Sörensson. *An Extensible SAT-solver*. In Theory and Applications of Satisfiability Testing, pp. 502-518, 2003.

[FJ03] Jean-Charles Faugère and Antoine Joux. *Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases*. In Advances in Cryptology - Crypto 2003, pp. 44-60, 2003.

[FKL00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner and Doug Whiting. *Improved Cryptanalysis of Rijndael*. In Seventh Fast Software Encryption Workshop, pp. 213-230, 2000.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[Hel80] Martin Hellman. *A Cryptanalytic Time-Memory Trade-off*, 1980. IEEE Transactions on Information Theory, vol. 26, issue 4, pp. 401-406.

[HMS76] M. Hellman, R. Merkle, R. Schroppel, L. Washington, W. Diffie, S. Pohlig and P. Schweitzer. *Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard*, 1976. Technical Report SEL 76-042, Stanford University.

[Iwa04] Kazuo Iwama. *Worst-Case Upper Bounds for kSAT*, 2004. Bulletin of the EATCS, vol. 82, pp. 61-71.

[Knu98] Donald Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition*. Addison-Wesley, 1998.

[Kob98] Neal Koblitz. *Algebraic Aspects of Cryptography - Algorithms and Computation in Mathematics*. Springer-Verlag, 1998.

[Mat93] Mitsuru Matsui. *Linear Cryptanalysis Method for DES Cipher*. In Advances in Cryptology - EUROCRYPT 1993, pp. 386-397, 1993.

[Mey01] Carl Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), 2001.

[MOV96] Alfred Menezes, Paul van Oorschot and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Nat99] National Institute of Standards and Technology. *Data Encryption Standard (DES)*, 1999. Federal Information Processing Standard 46, Version 3.

[Nat01] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*, 2001. Federal Information Processing Standard 197.

[Rad04] Håvard Raddum. *Solving Non-Linear Sparse Equation Systems over GF(2) Using Graphs*, 2004. University of Bergen Preprint.

[Rad07] Håvard Raddum. *MRHS Equation Systems*. In Selected Areas in Cryptography 2007, 14th International Workshop, LNCS 4876, pp. 232-245, 2007.

[RS06] Håvard Raddum and Igor Semaev. *New Technique for Solving Sparse Equation Systems*, 2006. See Cryptology ePrint Archive, Report 475.

[RS07] Håvard Raddum and Igor Semaev. *Solving MRHS Equations*, 2007. To Appear in Designs, Codes and Cryptography.

[Sem07] Igor Semaev. *On Solving Sparse Algebraic Equations Over Finite Fields*, 2007. To Appear in Designs, Codes and Cryptography.

[Sha49] C.E. Shannon. *Communication Theory of Secrecy Systems*, 1949. Bell System Technical Journal, vol. 28, pp. 656-715.

[SKI04] Makoto Sugita, Mitsuru Kawazoe and Hideki Imai. *Relation Between XL Algorithm and Gröbner Basis Algorithms*, 2004. See Cryptology ePrint Archive, Report 112.

[Sti05] Douglas Stinson. *Cryptography: Theory and Practice, Third Edition*. Chapman & Hall/CRC, 2005.

[Str69] Volker Strassen. *Gaussian Elimination is not Optimal*, 1969. Numerische Mathematik vol. 13, issue 4, pp. 354-356.

[Var97] Alexander Vardy. *Algorithmic Complexity in Coding Theory and the Minimum Distance Problem*. In Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 92-109, 1997.

[Woo92] Derick Wood. *Data Structures, Algorithms and Performance*. Addison-Wesley, 1992.

[ZV00] Arkadij Zakrevskij and Irina Vasilkova. *Reducing Large Systems of Boolean Equations*. In Proceedings of the 4th International Workshop on Boolean Problems, Freiberg University, 2000.

Index