

Diplomarbeit

Evaluating Algebraic Attacks on the AES

Ralf-Philipp Weinmann
<weinmann@cdc.informatik.tu-darmstadt.de>

Betreuer:
Prof. Dr. J. Buchmann,
Fachbereich Informatik
Fachgebiet Kryptographie und Computeralgebra,
Technische Universität Darmstadt

Contents

1	Introduction	5
1.1	Algebraic descriptions of AES	5
1.2	Block ciphers	6
1.2.1	Iterated Block Ciphers	6
1.2.2	Key-Iterated Block Ciphers	6
1.3	Classification of attacks on block ciphers	7
1.4	Scope of this thesis	7
2	The family of Mini-Rijndael	9
2.1	Parameters	10
2.2	An algorithmic cipher description	10
2.2.1	AddRoundKey	10
2.2.2	SubElement	11
2.2.3	ShiftRows	12
2.2.4	MixColumns	12
2.2.5	Key scheduling	13
3	Systems of polynomial equations	15
3.1	Terminology	15
3.2	Constructing the equations	16
3.3	Constructing a system over \mathbb{F}_2	18
3.3.1	The S-Boxes	18
3.3.2	The linear layer	19
3.3.3	The key schedule	20
3.4	Embedding the cipher	20
3.4.1	The S-Boxes	20
3.4.2	The linear layer	21
3.4.3	The key schedule	22
4	Linearization attacks	31
4.1	Linearization	31
4.2	The XL Algorithm	32
4.3	Relinearization	32
4.4	Extended Sparse Linearization	33
4.4.1	The final step	33

4.4.2	An example for XSL	34
5	Observations and experimental results	43
5.1	Implementation	43
5.2	The original examples	44
5.3	Applying XSL to a Mini-Rijndael	44
5.4	Our own toy example	45
5.5	XSL over an extension field of \mathbb{F}_2	45

Chapter 1

Introduction

Block ciphers are an important and omnipresent building block of modern cryptography. In August 2000, the block cipher Rijndael was selected for the Advanced Encryption Standard (AES). This happened in a then unprecedented way – an open contest with international participation was held by the NIST to find a successor for the then 24-year old Data Encryption Standard (DES).

Rijndael is a key-iterated block cipher with a very strong algebraic structure. The block and key length are variable in steps of 32 bits between 128 and 256 bits. The only valid block length for AES is 128 bits however; the key length for AES may be either 128, 192 or 256 bits.

1.1 Algebraic descriptions of AES

In late 2001, Ferguson, Schroepel and Whiting showed how to write the Rijndael in one algebraically closed equation [8], a continued fraction, over $GF(2^8)$. This equation contains about 2^{50} terms for a key and block size of both 128 bits.

In early 2002, Courtois and Pieprzyk described [4] how to express Rijndael and Serpent as a system of multivariate polynomial equations over $GF(2)$. The key property of their representation is that the equations are quadratic, very sparse and the system overdefined. The authors claim that Rijndael is susceptible to a new attack designed by them which aims at exactly the task of solving sparse and overdefined systems of quadratic equations. This technique is called Extended Sparse Linearization (XSL) and is described in the same paper. Two versions of the attack are presented, one with and one without taking the key schedule into account.

Subsequently Murphy and Robshaw establish an embedding of AES-128 into another cipher called Big Encryption System (BES) [17] which can be expressed as a quadratic system over $GF(2^8)$ that is even sparser. In a note submitted to the New European Schemes for Signature, Integrity, and Encryption (NESSIE) process, Murphy and Robshaw clarify the relationship between the BES and the XSL technique [16].

A slightly different version of the XSL paper [5] was presented at Asiacrypt 2002, describing Compact XSL, a slight modification to the originally proposed XSL technique.

Most of the known attacks on block ciphers – such as differential [2] and linear crypt-

analysis [15], the Square attack [7], the boomerang attack [21] etc. – are probabilistic and assume a that the attacker has access to a large number of known plain-text/cipher-text pairs.

This is what sets the proposed XSL attack apart. Depending on the key and block size and whether the key schedule is used or not, 1–14 plain-text/cipher-text pairs should be sufficient for determining a full AES cipher key.

1.2 Block ciphers

A *block cipher* is a family of invertible functions mapping an input string of fixed length n_b , the *plaintext* p to an output string, the *ciphertext* c . This family is indexed by the *cipher key* K . Usually the input and output domain as well as the domain of the cipher key are Boolean vector spaces. It is a reasonable choice to restrict the notion of a block cipher to families of functions that can be represented as algorithms parameterized by the cipher key.

Applying this algorithm to a plaintext is called *encryption*, the inverse operation is called *decryption*.

1.2.1 Iterated Block Ciphers

A block cipher is called *iterated block cipher* if the ciphertext is obtained from the plaintext by applying a *round function* R multiple times (N_r) to the plaintext ¹

$$S_i = R(S_{i-1}, K_i) \quad \forall 1 \leq i \leq N_r$$

We call the values S_i the *state of the cipher in round i* , where the state S_0 denotes the plaintext and S_{N_r} the ciphertext. The argument K_i to the function R is called *round subkey*; for K_i fixed, the function R has to be invertible.

The round keys are computed from the *cipher key* with an algorithm that we call the *key schedule* of the cipher. Concatenation of all round keys yields the *expanded key*, denoted by K :

$$K = K_1|K_2|\dots|K_{N_r}$$

The length of the binary expansion of the K is denoted by n_K .

1.2.2 Key-Iterated Block Ciphers

Several options exist for combining the round key K_i with the state S_{i-1} in the round function. Perhaps the most simple is to use addition over $GF(2)$ – bitwise XOR. If the round function is a composition of a key-independent round transformation and a key addition over $GF(2)$, the cipher is a *key-alternating cipher*. Furthermore, a key-alternating iterated block cipher is also called *key-iterated block cipher*.

¹iterated block ciphers are a specialization of the class of iterative block ciphers, which allows for several distinct round functions.

1.3 Classification of attacks on block ciphers

Following the notation introduced in [13], we distinguish the following types of attack on block ciphers:

Ciphertext-only: The adversary only has access to the ciphertext. The task for the cryptanalyst is to determine information about either the plaintext or the keying data.

Known plaintext: The adversary has access to plaintext and corresponding ciphertext. The objective is to recover keying data.

Chosen plaintext: Not only does the adversary know the plaintext but he may use the cipher as an oracle to which he can send plaintext and receive ciphertext from. After one oracle call – which may involve sending more than one plaintext block – his task is to recover keying data.

Adaptive chosen plaintext: Consider the same setting as in the *chosen plaintext* scenario, however allow the adversary may make multiple calls to the oracle and thus interactively choose the input from the observations/computations he made on earlier plaintext/ciphertext pairs. Between oracle calls, the keying data will not be changed.

Obviously these attacks are of different strength, meaning that the amount of information and access to the encryption device required to successfully carry out the attack varies. The above attacks are ordered by decreasing strength. Analogous chosen-text attacks also exist for the ciphertext, requiring access to a decryption oracle/device.

1.4 Scope of this thesis

Even reduced-round versions of Rijndael induce systems of polynomial equations that are too complex to be practically attacked with the XSL technique. Therefore it is our goal to study the XSL on “miniature ciphers” that are designed with a structure resembling the one found in Rijndael.

These ciphers which we will call “Mini-Rijndaels” are expected to yield systems of equations that are practically solvable with today’s off-the-shelf computers.

We shall proceed in four steps – first, we show how the Mini-Rijndaels are constructed and which parameters are variable, then how to generate systems of quadratic equations for these ciphers in an algorithmic fashion. This is followed by a detailed description of the algorithms employed in our XSL implementation. We conclude by giving experimental results on the viability of the attack.

Chapter 2

The family of Mini-Rijndael

When experimenting with new attack strategies, we usually cannot expect them to immediately work against the full-blown block cipher – unless its design is so weak that it succumbs to the attack without further detailed analysis. This is why cryptanalysts usually first investigate “reduced” versions of the cipher they attack, “reduced” usually meaning that the number of rounds of the cipher has been decreased. Attacks then are attempted first on a small number of rounds of the cipher and are refined and adapted to work on increasingly more rounds.

For our purposes it seems lucrative to scale Rijndael down not only in the number of rounds but to also employ smaller S-Boxes as well as less entries in the state and the subkey matrix. Thus we are able to cut down the complexity of algebraic representations we try use for the to attack enormously, yet still work with a structurally similar cipher.

In the following we identify the vector $x = (x_0, \dots, x_{n-1}) \in \mathbb{F}_2^n$ with the element $(\sum_{i=0}^{n-1} x_i \theta^i =: x) \in \mathbb{F}_2[\theta]/m(\theta)$ with $m(\theta) \in \mathbb{F}_2[\theta]$ an irreducible polynomial. Having fixed the minimal polynomial of the finite field, we also write \mathbb{F} or \mathbb{F}_{2^n} instead of $\mathbb{F}_2[\theta]/m(\theta)$, where $n = \deg(m)$. If the field in question is not unambiguous we may sometimes also write $\theta_{\mathcal{F}}$ for the defining element instead of simply θ .

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

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

In later chapters, when describing the cipher algebraically we will switch to interpreting the state as a column vector with the following ordering of elements:

$$\tilde{S} := (a_{0,0}, \dots, a_{N_a-1,0}, \dots, a_{1,0}, \dots, a_{0,N_b-1}, \dots, a_{N_a-1,N_b-1})^T$$

Since we are only interested in cryptanalysing the cipher algebraically and not actually using it, we only specify the encryption operation and the key schedule.

2.1 Parameters

The following table lists parameters of our cipher that can be easily adapted. Note that changes such as assigning $A = 0$ and $b = 0$ respectively may make algebraic cryptanalysis easier, but then again does not reflect the Rijndael design criteria ¹.

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

2.2 An algorithmic cipher description

The sequence of operations to be performed is the following:

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

2.2.1 AddRoundKey

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

¹since interpolation attacks are possible then

Algorithm 1 MiniRijndaelEncrypt

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

```

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

```

Algorithm 2 AddRoundKey

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

```

 $S' \leftarrow S \oplus K$ 

```

2.2.2 SubElement

SubElement applies the non-linear invertible function γ – the S-Box function – to each element $a_{i,j}$ of the state. The S-box uses the same construction as Rijndael, generalized to an arbitrary input/output size of s bits. Rijndael’s design principles lean heavily on the work of K. Nyberg [18].

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

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

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

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

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

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

Algorithm 3 SubElement

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

```

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

```

2.2.3 ShiftRows

Each row i of the internal state is cyclically shifted λ_i positions to the left by the ShiftRows transformation. Since this operation is a permutation, it is invertible.

Assuming we have a 4×4 -state such as in Rijndael, the operation looks as follows:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{pmatrix}$$

Algorithm 4 ShiftRows

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

```

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

```

2.2.4 MixColumns

Each column of the state is multiplied by an invertible and circulant matrix M_{mix} . The matrix M_{mix} used in Rijndael was chosen such that its branch number is maximal – to obtain optimal diffusion; thus for a Mini-Rijndael test cipher we should do likewise. For a single column i of a $4 \times N_b$ state the MixColumns transformation looks as follows:

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

Algorithm 5 MixColumns

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

```

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

```

2.2.5 Key scheduling

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

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

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

Algorithm 6 MiniRijndaelKeySchedule

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

```

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

```

Chapter 3

Systems of polynomial equations

In this chapter we will show how to represent the problem of a known-plaintext attack on a certain class of block ciphers – nominally on the family of Mini-Rijndael – as a system of polynomial equations over a finite field of characteristic two. We consider two algebraic descriptions of the cipher in detail, one of them being a quadratic system over \mathbb{F}_2 , the other one a quadratic system over \mathbb{F}_{2^s} , with s being the width of the S-Box in the Mini-Rijndael cipher represented.

3.1 Terminology

For defining terms, monomials as well as term orders we follow the description of [1]. A *term* in the variables of $X := \{x_1, \dots, x_n\}$ is a power product of the form

$$x_1^{e_1} \cdots x_n^{e_n} \quad \text{with } e_i \in \mathbb{N}_0 \quad \forall 1 \leq i \leq n$$

Note that the constant $x_1^0 \cdots x_n^0 = 1$ also is a valid term by this definition. By $T(X)$ we denote the set of all terms in variables of X . We define a *monomial* in the variables of X over the field \mathbb{F} as a polynomial of the form $m = at$ with $0 \neq a \in \mathbb{F}$ and $t \in T(X)$, calling a the *coefficient* of m and t the term of m .

Let f be written as a sum of pairwise inequivalent monomials. Then the set of all monomials occurring in this representation is denoted by $M(f)$ and the set of all terms of f , denoted by $T(f)$, is the set of all terms of monomials $m \in M(f)$. We define $\deg(f)$ to be the total degree of the polynomial f .

Definition 3.1.1. Let \leq be a term order on $T(X)$. For any finite, non-empty subset A of M consisting of pairwise inequivalent monomials, let $\max(A)$ be the unique maximal element of A w.r.t. the order \leq . For any non-zero polynomial $f \in \mathbb{F}[X]$ we define the **head term** $\text{HT}(f)$, the **head monomial** $\text{HM}(f)$ and the **head coefficient** $\text{HC}(f)$ of f w.r.t. \leq as follows:

$$\begin{aligned} \text{HT}(f) &= \max(T(f)) \\ \text{HM}(f) &= \max(M(f)) \\ \text{HT}(f) &= \text{the coefficient of HM}(f) \end{aligned}$$

Definition 3.1.2. Let M be a non-empty set and $r \subseteq M \times M$ a relation on M . The set

$$r^{-1} := \{(a, b) \mid (b, a) \in r\}$$

is called the **inverse relation** to r . Furthermore, a partial order on M for which $r \cup r^{-1} = M \times M$ holds is called a **linear order**.

Definition 3.1.3. A term order is a linear order on $T(X)$ that satisfies the following conditions:

- (i) $1 \leq t$ for all $t \in T$.
- (ii) $t_1 \leq t_2$ implies $t_1 \cdot s \leq t_2 \cdot s$ for all $s, t_1, t_2 \in T$.

Example 3.1.4.

lexicographical order We say that $X_1^{d_1} \cdots X_n^{d_n} \leq_{lex} X_1^{e_1} \cdots X_n^{e_n}$ iff $(d_1, \dots, d_n) = (e_1, \dots, e_n)$ or there exists $1 \leq i \leq n$ with $d_j = e_j$ for $1 \leq j \leq i - 1$ and $d_i < e_i$.

inverse lexicographical order $X_1^{d_1} \cdots X_n^{d_n} \leq_{invlex} X_1^{e_1} \cdots X_n^{e_n}$ iff $(d_1, \dots, d_n) = (e_1, \dots, e_n)$ or there exists $1 \leq i \leq n$ with $d_j = e_j$ for $i + 1 \leq j \leq n$ and $d_i < e_i$

total degree-lexicographical order Let \leq be a term order or an inverse term order on $T(X)$. Let $f = X_1^{d_1} \cdots X_n^{d_n}$ and $X_1^{e_1} \cdots X_n^{e_n}$. Set $f \leq' g$ iff either $\deg(f) < \deg(g)$ or $\deg(f) = \deg(g)$ and $f \leq g$.

3.2 Constructing the equations

Theorem 3.2.1. Let $u, v \in \mathbb{N}$, P_0, \dots, P_u be variables representing the plaintext, C_0, \dots, C_u the ciphertext and K_0, \dots, K_v the cipher key. Then every block cipher can be written as a system of polynomial equations over \mathbb{F}_2 or \mathbb{F}_{2^n} for some $n \in \mathbb{N}$ with variables P_i , C_i and K_i .

Proof.

- (i) By definition a block cipher is a function between boolean vector spaces that maps an input P to an output C under a key K . Every boolean function of the block cipher can be represented by a polynomial f in P_i , K_i and C_i . Thus we can write the block cipher as a system of polynomial equations over \mathbb{F}_2 .
- (ii) Pick an integer n such that $n|N_k$ and $n|N_s$. Having fixed a minimum polynomial for \mathbb{F}_{2^n} , express the key, the plaintext and the ciphertext as vectors of elements of \mathbb{F}_{2^n} . Each element of the ciphertext vector can then be interpolated by a polynomial in the plaintext and key variables, i.e. by Lagrange interpolation.

□

Since we are dealing with an iterated block-cipher having a non-trivial round function, a system with no variables for the intermediate states during execution will become rather large both in the number of terms and in the total degree of the polynomials involved.

Thus we wish to trade off the expression swell in a straightforward representation of the flavor presented above against the inconvenience of having a larger number of equations and variables. This has the advantage that the polynomial system is made up of less terms, the total degree of the polynomials is smaller, and the polynomials become more sparse, meaning there are less terms involved per polynomial.

The representation with the least maximum total degree of all polynomials involved in the system is a quadratic system of equations.

Proposition 3.2.2. *Every system of polynomial equations $\mathcal{F} \in \mathbb{F}[X]$ over a finite field \mathbb{F} can be transformed into an equivalent system of quadratic equations $\mathcal{F}' \in \mathbb{F}[X \cup X']$ by introducing new variables $\{x_k \notin X, \dots, x_l \notin X\} := X'$, assigning each new variable a sub-expression of at most degree two and rewriting the original equations accordingly.*

Example 3.2.3. Consider the following polynomial system over $\mathbb{F}[x_0, \dots, x_3]$

$$\begin{aligned} 3x_0^2 + 2x_1^3x_2^2 + x_0x_1^2 + 2 &= 0 \\ x_0 + x_1^4 + 2x_1x_2x_3 + x_3^{16} &= 0 \\ x_1x_3 + 2x_2 &= 0 \\ 3x_0x_2^5 + x_3^7 &= 0 \end{aligned}$$

This is equivalent to the following quadratic system:

$$\begin{aligned} 3x_0^2 + 2x_6x_2 + x_0x_4 + 2 &= 0 & x_5 - x_1x_2 &= 0 \\ x_0 + x_4^2 + 2x_5x_3 + x_9^2 &= 0 & x_6 - x_4x_5 &= 0 \\ x_1x_3 + 2x_2 &= 0 & x_7 - x_3^2 &= 0 \\ 3x_0x_2^5 + x_{10}x_3 &= 0 & x_8 - x_7^2 &= 0 \\ x_4 - x_1^2 &= 0 & x_9 - x_8^2 &= 0 \\ & & x_{10} - x_8x_7 &= 0 \end{aligned}$$

Consider a system of multivariate polynomial equations \mathcal{L} over a finite field \mathbb{F} :

$$\begin{aligned} f_1(x_1, \dots, x_m) &= 0 \\ &\vdots \\ f_l(x_1, \dots, x_m) &= 0 \end{aligned}$$

Clearly every polynomial system can be transformed into the above form, for if we have an equation $f(x) = g(x)$ the difference $h = f - g$ also is a polynomial and we can replace the above equation by $h(x) = 0$.

The set of polynomials $f_1, \dots, f_l \in \mathbb{F}[X]$ such that $f_i(x) = 0$ for all $1 \leq i \leq l$ generates an ideal \mathcal{I} in $\mathbb{F}[X]$. If the number of zeros of this ideal in $\mathbb{F}^{|X|}$ is one, the

corresponding system \mathcal{L} obviously only has one solution – in this case \mathcal{I} is called 0-dimensional.¹

3.3 Constructing a system over \mathbb{F}_2

Constructing a system over \mathbb{F}_2 is mostly straightforward. We need to build equations for the S-Boxes of each round, the linear layer of each round and for the key schedule. An algorithmic description of this procedure is given in `generate_equations_gf2`.

3.3.1 The S-Boxes

Assume the input variables of a S-Box are $v_0, \dots, v_{s-1} \in X$ and the output variables are $w_0, \dots, w_{s-1} \in X$. Write the following polynomials $f_1, f_2 \in \mathbb{F}_{2^s}[X]$:

$$f_1 = \sum_{i=0}^{s-1} v_i \theta^i, \quad f_2 = \sum_{i=0}^{s-1} w_i \theta^i$$

Were the S-Box to perform a simple inversion in the field \mathbb{F}_{2^s} , the following equations would hold:

$$f_1 \cdot f_2 = 0$$

However, since we also need to apply the affine transformation over \mathbb{F}_2 , the equations become slightly more complicated:

$$f_2' = (\theta^0, \dots, \theta^{s-1}) \cdot (A^{-1} \cdot (w_0, \dots, w_{s-1})^T + b)$$

The equation now looks as follows:

$$f_1 \cdot f_2' - 1 = 0$$

Going from the representation over the field \mathbb{F}_{2^s} to one over the vector space \mathbb{F}_2^s we obtain an equivalent system of s simultaneous equations over \mathbb{F}_2 . This is performed by the function `seperate_polynomial`. Of these equations, $s - 1$ hold with probability 1, and the remaining equation only with probability $\frac{2^s - 1}{2^s}$; this is due to the fact the S-Box in contrary to inversion in a finite field is also defined for the value 0.

What Courtois and Pieprzyk observed is that one can get even more linearly independent equations per S-Box by taking the following relations into account:

$$\begin{aligned} f_1^2 \cdot f_2' - f_1 &= 0 \\ f_1 \cdot f_2'^2 - f_2 &= 0 \end{aligned}$$

¹Note however that a 0-dimensional ideal only needs to have finitely many zeros, not necessarily a single one.

All of these equations are bi-affine, meaning the terms occurring in the polynomials are either the constant term, terms of degree one such as v_i or w_i or from the set $\{v_i w_j \mid (0 \leq i < s) \wedge (0 \leq j < s)\}$. The total number of equations per S-Box is r , the total number of terms occurring in these r equations is t . Their objective is to maximize the ratio $\frac{r}{t}$, which is the main reason why we do not consider relations of the form $f_1^4 \cdot f_2^3 - f_1^3$, yielding “fully quadratic” equations over \mathbb{F}_2 – meaning quadratic equations that are not bi-affine.

For the case presented above $r = 3s - 1$ and $t = s^2 + 2s + 1$.

Example 3.3.1.

$$m(x) = \theta^3 + \theta + 1, \quad \mathbb{F} := \mathbb{F}_2[m(x)], \quad A := \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad b := \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{aligned} f_1 &= \theta^2 \cdot x_2 + \theta \cdot x_1 + x_0 \\ f_2 &= (\theta^2 + \theta + 1) \cdot x_5 + (\theta + 1) \cdot x_4 + x_3 + \theta \end{aligned}$$

$$\begin{aligned} x_2 x_4 + x_2 + x_1 x_4 + x_1 x_3 + x_0 x_5 + x_0 x_4 + x_0 &= 0 \\ x_2 x_4 + x_2 x_3 + x_1 x_5 + x_1 x_4 + x_1 + x_0 x_5 &= 0 \\ x_2 x_4 + x_2 + x_1 x_5 + x_1 x_4 + x_1 + x_0 x_5 + x_0 x_4 + x_0 x_3 + x_0 &= 0 \\ x_2 x_3 + x_2 + x_1 x_4 + x_0 x_5 + x_0 x_4 + x_0 &= 0 \\ x_2 x_5 + x_2 x_3 + x_1 x_4 + x_1 x_3 + x_0 x_5 &= 0 \\ x_5 + x_4 + x_3 + x_2 x_5 + x_1 x_4 + x_1 + x_0 x_5 + x_0 x_4 + x_0 x_3 &= 0 \\ x_5 + x_4 + x_2 x_5 + x_2 x_4 + x_2 + x_1 x_5 + x_1 x_3 + x_1 + x_0 x_5 + 1 &= 0 \\ x_5 + x_2 x_5 + x_2 x_3 + x_2 + x_1 x_5 + x_0 x_4 + x_0 &= 0 \end{aligned}$$

3.3.2 The linear layer

The linear layer of the cipher consists of an addition of the round key, of the `ShiftRows` operation and of the `MixColumns` operation. Note that the internal state of the cipher is represented by a column vector of \mathbb{F} -elements. Key addition is quite trivially described as the addition of the round key variables to the state variables, for `ShiftRows` and the `MixColumns` steps we need to derive the $(N_a N_b) \times (N_a N_b)$ matrices M_S and M_C respectively.

Since `MixColumns` applies the same matrix M_{mix} to each column of the state, the corresponding M_C matrix is the block-diagonal matrix $M_C = \text{diag}(4) \otimes M_{\text{mix}}$.

The effect of the `ShiftRows` operation, cyclical shifts of each of the rows, can be expressed by a matrix with exactly one 1-entry in each row, which is calculated by the function `compute_shiftrows_matrix`.

Assume the vector of input variables to the linear layer is $v_0, \dots, v_{N_a \cdot N_b \cdot s - 1}$ and the output vector is $w_0, \dots, w_{N_a \cdot N_b \cdot s - 1}$.

We then write the vectors of polynomials

$$F = (f_0, \dots, f_{N_a \cdot N_b - 1}) \text{ and } G = (g_0, \dots, g_{N_a \cdot N_b - 1})$$

with

$$f_i = \sum_{j=0}^{s-1} v_{is+j} \theta^j, \quad g_i = \sum_{i=0}^{s-1} w_{is+i} \theta^j \quad \text{for all } 0 \leq i \leq (N_a \cdot N_b - 1).$$

For round zero, the linear layer is a simple key addition, for every subsequent round except the last, the effect of the linear layer can be expressed by the relation $F \cdot M_S \cdot M_C - G = 0$ and for the last round by $F \cdot M_S \cdot M_C - G = 0$. Taking each row of the vector on the left-hand side of these equations and interpreting it over the vector space \mathbb{F}_2^s instead of the field \mathbb{F}_{2^s} – again by using `seperate_polynomial` – yields a system of $N_a \cdot N_b \cdot s$ simultaneous equations over \mathbb{F}_2 .

3.3.3 The key schedule

Equations for round keys 1 to N_r can be directly read off from the algorithm `generate_subkey_equations_gf2`.

3.4 Embedding the cipher

In [17] Robshaw and Murphy introduce a technique for embedding the AES cipher within an extended cipher called Big Encryption System (BES) by replacing each element a of the state with its vector conjugate \tilde{a} defined as follows:

$$\tilde{a} := (a^{2^0}, \dots, a^{2^s}) \in \mathbb{F}_{2^s}^s$$

Now all transformations used can be expressed as simple \mathbb{F}_{2^s} -operations, including the affine map over the S-Box which for Rijndael is defined over \mathbb{F}_2 . The same technique can be applied to any of our Mini-Rijndaels.

3.4.1 The S-Boxes

For expressing the \mathbb{F}_2 -linear part of the S-Boxes we first need to compute the linearized polynomial[14] for the affine matrix A :

$$L(a) = \sum_{i=0}^{s-1} \lambda_i a^{2^i} \in \mathbb{F}_{2^s}[a]$$

This polynomial has the desirable property of inducing a linear operator on F if considering \mathbb{F}_{2^s} as a vector space over \mathbb{F}_2 .

Set the \mathbb{F}_{2^s} -vector $\alpha = (\theta^0, \dots, \theta^{s-1})$. Following a description given in [19] we compute the coefficients λ_i as

$$\begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_{s-1} \end{pmatrix} = \begin{pmatrix} \alpha_0^{2^0} & \dots & \alpha_0^{2^{s-1}} \\ \vdots & \ddots & \vdots \\ \alpha_{s-1}^{2^0} & \dots & \alpha_{s-1}^{2^{s-1}} \end{pmatrix}^{-1} \cdot A^T \cdot \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{s-1} \end{pmatrix}$$

The following matrix then describes the \mathbb{F}_2 -linear part of the S-Box:

$$M_L = \begin{pmatrix} \lambda_0^{2^0} & \lambda_1^{2^0} & \cdots & \lambda_{s-1}^{2^0} \\ \lambda_{s-1}^{2^1} & \lambda_0^{2^1} & \cdots & \lambda_{s-2}^{2^1} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^{2^{s-1}} & \lambda_2^{2^{s-1}} & \cdots & \lambda_0^{2^{s-1}} \end{pmatrix}$$

The constant of the affine \mathbb{F}_2 -mapping can be moved either into the key schedule or into the linear layer (which should then be called affine layer). The non-linear part of the S-Box has thus been reduced to a simple inversion in \mathbb{F}_{2^s} ². Again assume the input variables of a S-Box are $v_0, \dots, v_{s-1} \in X$ and the output variables are $w_0, \dots, w_{s-1} \in X$. Then the equations

$$v_i \cdot w_i - 1 = 0 \quad \text{for all } 0 \leq i \leq s-1$$

hold. However, knowing that the conjugacy property must also hold, we obtain the additional equations

$$\begin{aligned} v_i^2 - v_{i+1 \bmod s} &= 0 \quad \text{for all } 0 \leq i \leq s-1 \\ w_i^2 - w_{i+1 \bmod s} &= 0 \quad \text{for all } 0 \leq i \leq s-1 \end{aligned}$$

and thus have an overdefined system of quadratic equations for each S-Box. In contrast to the equations over \mathbb{F}_2 we note that these equations do not contain any products between input and output variables of the S-Box but rather only products of the form $v_i v_j$ or of the form $w_i w_j$. The total number of terms per S-Box is $5 \cdot s + 1$ and the total number of equations per S-Box is $3s$.

3.4.2 The linear layer

We first need to compute the matrices M_C and M_S as described in the \mathbb{F}_2 case. These then need to be converted into a form compatible with the vector conjugated state by the function `compute_conjugated_matrix`, resulting in $(N_a \cdot N_b \cdot s) \times (N_a \cdot N_b \cdot s)$ matrices M'_C and M'_S with coefficients in \mathbb{F}_{2^s} . Computing a product of the block diagonal matrix $\text{diag}(N_a \cdot N_b) \otimes M_L$, M'_S , and M'_C we obtain a matrix completely describing the linear layer.

The only thing missing is the affine part of the transformation which we inherited from the S-Box. This situation can be rectified by considering the vector b as an element of \mathbb{F}_{2^s} which we shall call b' .

The vectors

$$B := M_S \cdot M_C \cdot \left(\underbrace{(1, \dots, 1)^T}_{N_a \cdot N_b \text{ repetitions of } 1} \otimes (b'^{2^0}, \dots, b'^{2^s-1})^T \right)$$

for every but the last round respectively and

$$B' := M_S \cdot \left(\underbrace{(1, \dots, 1)^T}_{N_a \cdot N_b \text{ repetitions of } 1} \otimes (b'^{2^0}, \dots, b'^{2^s-1})^T \right)$$

for the last round then represent the constant in the affine transformation.

²we assume that the value 0 never appears as input to the S-Box, because it is not a patched inverse function.

3.4.3 The key schedule

Just like for the \mathbb{F}_2 case, equations for round keys 1 to N_r can be directly read off from the algorithm `generate_subkey_equations_gf2n`. The only nit to be aware of is that we have to add elements of the conjugated vector of the corresponding round constant in the first column of the round key instead of single bits.

Algorithm 7 generate_equations_gf2

Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $M_2 \leftarrow \text{compute\_shiftrows\_matrix}(\varphi)$ 
 $M_1 \leftarrow M_2 \cdot (\text{diag}(N_b) \otimes M_{\text{mix}})$ 
 $\mathcal{F} \leftarrow \text{generate\_linear\_equations\_gf2}(0, M_2)$ 
for  $r \leftarrow 0$  to  $N_r$  do
  for  $i \leftarrow 0$  to  $N_b - 1$  do
    for  $j \leftarrow 0$  to  $N_a - 1$  do
       $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_sbox\_equations\_gf2}(r, i, j)$ 
    end for
  end for
  if  $r \neq N_r$  then
     $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_linear\_equations\_gf2}(r, M_1)$ 
  else
     $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_linear\_equations\_gf2}(r, M_2)$ 
  end if
   $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_subkey\_equations\_gf2}(r)$ 
end for

```

Algorithm 8 compute_shiftrows_matrix

Input: $(\lambda_0, \dots, \lambda_{N_a-1}) \in \mathbb{N}_0^{N_a}$ **Output:** $M \in \mathbb{F}_{2^s}^{(N_a \cdot N_b) \times (N_a \cdot N_b)}$

```

for  $i \leftarrow 0$  to  $N_a - 1$  do
  for  $j \leftarrow 0$  to  $N_b - 1$  do
     $x \leftarrow ((j - \lambda_i + N_a) \cdot N_a + i) \bmod (N_a \cdot N_b)$ 
     $y \leftarrow j \cdot N_b + i$ 
     $M_{x,y} \leftarrow 1$ 
  end for
end for

```

Algorithm 9 generate_subkey_equations_gf2

Input: $r \in \mathbb{N}$, $1 \leq r \leq N_r$
Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $N_c \leftarrow N_a \cdot s$ 
for  $i \leftarrow r \cdot N_k$  to  $(r + 1) \cdot N_k$  do
  if  $i \bmod N_k = 0$  then
    { non-linear equations for key schedule }
    for  $j \leftarrow 0$  to  $N_a - 1$  do
       $u \leftarrow ((i - 1) \cdot N_a + j) \cdot s$ 
       $v \leftarrow ((r - 1) \cdot N_a + j) \cdot s$ 
       $\mathcal{F}' \leftarrow \text{gen\_sbox\_equations\_gf2}((K_u, \dots, K_{u+s-1}), (t_v, \dots, t_{v+s-1}))$ 
       $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}'$ 
    end for
    { linear equations for key schedule }
    for  $j \leftarrow 0$  to  $N_a \cdot s - 1$  do
       $f \leftarrow K_{i \cdot N_c + j} + K_{(i - N_k) \cdot N_c + j}$ 
       $f \leftarrow f + t_{((r - 1) \cdot N_c + (j + s)) \bmod N_c}$ 
      if  $j < s \wedge \theta_{\mathbb{F}}^r \bmod \theta_{\mathbb{F}}^j \neq 0$  then
         $f \leftarrow f + 1$ 
      end if
       $\mathcal{F} \leftarrow \mathcal{F} \cup f$ 
    end for
  else
    { linear equations for key schedule }
    for  $i \leftarrow 0$  to  $N_c - 1$  do
       $F \leftarrow \mathcal{F} \cup (K_{N_c \cdot i + j} + K_{N_c \cdot (i - 1) + j} + K_{N_c \cdot (i - N_k) + j})$ 
    end for
  end if
end for

```

Algorithm 10 generate_linear_equations_gf2

Input: $r \in \mathbb{N}, 0 \leq r \leq N_r$
Input: $M \in \mathbb{F}_{2^s}^{(N_a \cdot N_b) \times (N_a \cdot N_b)}$
Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $\mathcal{F} \leftarrow \emptyset$ 
{ zero round is a simple key addition }
if  $r = 0$  then
  for  $i \leftarrow 0$  to  $N_a \cdot N_b \cdot s - 1$  do
     $f \leftarrow p_i + K_i + x_i$ 
     $\mathcal{F} \leftarrow \mathcal{F} \cup f$ 
  end for
else
  for  $i \leftarrow 0$  to  $N_a \cdot N_b - 1$  do
    for  $j \leftarrow 0$  to  $s - 1$  do
       $a_i \leftarrow a_i + \theta_{\mathbb{F}}^j \cdot x_{k+i \cdot s+j}$ 
    end for
  end for
   $a \leftarrow M \cdot a$ 
  for  $i \leftarrow 0$  to  $N_a \cdot N_b - 1$  do
     $a' \leftarrow \text{seperate\_polynomial}(a_i)$ 
    for  $j \leftarrow 0$  to  $s - 1$  do
      { last round has ciphertext as output variables }
      if  $r = N_r$  then
         $f \leftarrow c_{i \cdot s+j}$ 
      else
         $f \leftarrow x_{2r \cdot (N_a \cdot N_b + i) \cdot s+j}$ 
      end if
       $f \leftarrow f + a'_j + K_{r \cdot (N_a \cdot N_b + i) \cdot s+j}$ 
       $\mathcal{F} \leftarrow \mathcal{F} \cup f$ 
    end for
  end for
end if

```

Algorithm 11 generate_sbox_equations_gf2

Input: $(v_0, \dots, v_{s-1}) \in X^s$ **Input:** $(w_0, \dots, w_{s-1}) \in X^s$ **Output:** $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $f_1 \leftarrow 0$ 
 $f_2 \leftarrow 0$ 
 $\mathcal{F} \leftarrow \emptyset$ 
for  $i \leftarrow 0$  to  $s - 1$  do
   $f_1 \leftarrow f_1 + \theta_{\mathbb{F}}^i \cdot v_i$ 
  for  $j \leftarrow 0$  to  $s - 1$  do
    if  $A_{i,j}^{-1} = 1$  then
       $f_2 \leftarrow f_2 + \theta_{\mathbb{F}}^i \cdot w_j + b_j \cdot \theta_{\mathbb{F}}^i$ 
    end if
  end for
end for
 $F \leftarrow \text{seperate\_polynomial}(f_1 \cdot f_2 + 1)$ 
for  $i \leftarrow 0$  to  $s - 1$  do
   $\mathcal{F} \leftarrow \mathcal{F} \cup F_i$ 
end for
 $F \leftarrow \text{seperate\_polynomial}(f_1^2 \cdot f_2 + f_1)$ 
for  $i \leftarrow 0$  to  $s - 1$  do
   $\mathcal{F} \leftarrow \mathcal{F} \cup F_i$ 
end for
 $F \leftarrow \text{seperate\_polynomial}(f_2^2 \cdot f_1 + f_2)$ 
for  $i \leftarrow 0$  to  $s - 1$  do
   $\mathcal{F} \leftarrow \mathcal{F} \cup F_i$ 
end for

```

Algorithm 12 seperate_polynomial

Input: $f \in \mathbb{F}_{2^s}[X]$ **Output:** $(F_0, \dots, F_{s-1}) \in \mathbb{F}_2[X]^s$

```

 $f' \leftarrow f$ 
while  $f' \neq 0$  do
   $c \leftarrow \text{HC}(f')$ 
  for  $i \leftarrow 0$  to  $s - 1$  do
    if  $c \bmod \theta_{\mathbb{F}}^i \neq 0$  then
       $F_i \leftarrow F_i + \text{reduce}(\text{HT}(f'))$ 
    end if
     $f' \leftarrow f' - \text{HM}(f')$ 
  end for
end while

```

Algorithm 13 generate_equations_gf2n

Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $M_3 \leftarrow \text{compute\_sbox\_matrix}(\text{compute\_coeffs\_linpoly}(A))$ 
 $M_2 \leftarrow \text{compute\_shiftrows\_matrix}(\varphi)$ 
 $M_1 \leftarrow M_2 \cdot (\text{diag}(N_b) \otimes M_{\text{mix}}) \cdot (\text{diag}(N_a \cdot N_b) \otimes M_3)$ 
 $M_2 \leftarrow M_2 \cdot (\text{diag}(N_a \cdot N_b) \otimes M_3)$ 
 $\mathcal{F} \leftarrow \text{generate\_linear\_equations\_gf2}(0, M_2)$ 
for  $r \leftarrow 0$  to  $N_r$  do
  for  $i \leftarrow 0$  to  $N_b - 1$  do
    for  $j \leftarrow 0$  to  $N_a - 1$  do
       $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_sbox\_equations\_gf2n}(r, i, j)$ 
    end for
  end for
  if  $r \neq N_r$  then
     $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_linear\_equations\_gf2n}(r, M_1)$ 
  else
     $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_linear\_equations\_gf2n}(r, M_2)$ 
  end if
   $\mathcal{F} \leftarrow \mathcal{F} \cup \text{generate\_subkey\_equations\_gf2n}(r, M_3)$ 
end for

```

Algorithm 14 compute_conjugated_element

Input: $a \in \mathbb{F}_{2^n}$, $n \in \mathbb{N}$ **Output:** $a' \in \mathbb{F}_{2^n}^n$

```

for  $i \leftarrow 0$  to  $n - 1$  do
   $a' \leftarrow a^{2^i}$ 
end for

```

Algorithm 15 compute_conjugated_matrix

Input: $M \in \mathbb{F}_{2^n}^{u \times v}$, $n, u, v \in \mathbb{N}$ **Output:** $M' \in \mathbb{F}_{2^n}^{un \times vn}$

```

for  $k \leftarrow 0$  to  $n - 1$  do
  for  $i \leftarrow 0$  to  $u - 1$  do
    for  $j \leftarrow 0$  to  $v - 1$  do
       $M'_{in+k, jn+k} \leftarrow M_{i,j}^{2^k}$ 
    end for
  end for
end for

```

Algorithm 16 compute_coeffs_linpoly

Input: $M \in \mathbb{F}_2^{n \times n}, n \in \mathbb{N}$
Output: $\lambda \in \mathbb{F}_{2^n}^n$

```

for  $i \leftarrow 0$  to  $s - 1$  do
   $\alpha_i \leftarrow \theta_{\mathbb{F}}^i$ 
  for  $j \leftarrow 0$  to  $s - 1$  do
     $B_{i,j} \leftarrow \alpha_i^{2^j}$ 
  end for
end for
{ embed  $A$  into  $\mathbb{F}_{2^n}$  }
 $\lambda \leftarrow B^{-1} \cdot A^T \cdot \alpha$ 

```

Algorithm 17 compute_sbox_matrix

Input: $\lambda \in \mathbb{F}_{2^n}^n, n \in \mathbb{N}$
Output: $M' \in \mathbb{F}_{2^n}^{n \times n}$

```

for  $i \leftarrow 0$  to  $n - 1$  do
  for  $j \leftarrow 0$  to  $n - 1$  do
     $M'_{i,j} \leftarrow \lambda_{i-j+n \bmod n}^{2^i}$ 
  end for
end for

```

Algorithm 18 generate_linear_equations_gf2n

Input: $r \in \mathbb{N}, 0 \leq r \leq N_r$
Input: $M \in \mathbb{F}_{2^s}^{(N_a \cdot N_b \cdot s) \times (N_a \cdot N_b \cdot s)}$
Input: $B \in \mathbb{F}_{2^s}^{N_a \cdot N_b \cdot s}$
Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $\mathcal{F} \leftarrow \emptyset$ 
 $N_s \leftarrow N_a \cdot N_b \cdot s$ 
{ zero round is a simple key addition }
if  $r = 0$  then
  for  $i \leftarrow 0$  to  $N_s - 1$  do
     $f \leftarrow p_i + K_i + x_i$ 
     $\mathcal{F} \leftarrow \mathcal{F} \cup f$ 
  end for
else
  for  $i \leftarrow 0$  to  $N_s - 1$  do
    { last round has ciphertext as output variables }
    if  $r = N_r$  then
       $f \leftarrow c_i$ 
    else
       $f \leftarrow x_{2r \cdot N_s + i}$ 
    end if
    for  $j \leftarrow 0$  to  $N_s - 1$  do
      if  $M_{i,j} \neq 0$  then
         $f \leftarrow f + M_{i,j} x_{(2r-1) \cdot N_s + j}$ 
      end if
    end for
     $f \leftarrow f + B_i$ 
     $\mathcal{F} \leftarrow \mathcal{F} \cup f$ 
  end for
end if

```

Algorithm 19 generate_sbox_equations_gf2n

Input: $(v_0, \dots, v_{s-1}) \in X^s$
Input: $(w_0, \dots, w_{s-1}) \in X^s$
Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $\mathcal{F} \leftarrow \emptyset$ 
for  $i \leftarrow 0$  to  $s - 1$  do
   $f_1 \leftarrow v_i \cdot w_i + 1$ 
   $f_2 \leftarrow v_i^2 + v_{(i+1) \bmod s}$ 
   $f_3 \leftarrow w_i^2 + w_{(i+1) \bmod s}$ 
   $\mathcal{F} \leftarrow \mathcal{F} \cup f_1 \cup f_2 \cup f_3$ 
end for

```

Algorithm 20 generate_subkey_equations_gf2n

Input: $r \in \mathbb{N}$, $1 \leq r \leq N_r$
Output: $\mathcal{F} \subseteq \mathbb{F}[X]$

```

 $\delta' \leftarrow \text{compute\_conjugated\_element}(\theta_{\mathbb{F}}^r)$ 
 $N_c \leftarrow N_a \cdot s$ 
for  $i \leftarrow r \cdot N_k$  to  $(r + 1) \cdot N_k$  do
  if  $i \bmod N_k = 0$  then
    { non-linear equations for key schedule }
    for  $j \leftarrow 0$  to  $N_a - 1$  do
       $u \leftarrow ((i - 1) \cdot N_c + j \cdot s) \cdot s$ 
       $v \leftarrow ((r - 1) \cdot N_c + j \cdot s) \cdot s$ 
       $\mathcal{F}' \leftarrow \text{gen\_sbox\_equations\_gf2n}((K_u, \dots, K_{u+s-1}), (t_v, \dots, t_{v+s-1}))$ 
       $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}'$ 
    end for
    { linear equations for key schedule }
    for  $j \leftarrow 0$  to  $N_a \cdot s - 1$  do
       $f \leftarrow K_{i \cdot N_c + j} + K_{(i - N_k) \cdot N_c + j}$ 
       $f \leftarrow f + t_{((r-1) \cdot N_c + (j+s) \bmod N_c)}$ 
      if  $j < s \wedge \delta'_j \neq 0$  then
         $f \leftarrow f + \delta'_j$ 
      end if
       $\mathcal{F} \leftarrow \mathcal{F} \cup f$ 
    end for
  else
    { linear equations for key schedule }
    for  $i \leftarrow 0$  to  $N_c - 1$  do
       $F \leftarrow \mathcal{F} \cup (K_{N_c \cdot i + j} + K_{N_c \cdot (i-1) + j} + K_{N_c \cdot (i - N_k) + j})$ 
    end for
  end if
end for

```

Chapter 4

Linearization attacks

The problem of solving multivariate quadratic systems of equations is generally known as MQ-Problem. It can be shown that even for the smallest field of all, \mathbb{F}_2 , and more generally for any finite field \mathcal{F} , deciding whether a multivariate quadratic system over \mathcal{F} actually has a solution is an NP-complete problem [9], [10] (though [9] contains no proof but only a reference to an unpublished manuscript and personal communications).

The standard method for solving systems of polynomial equations is to compute a lexicographically ordered Gröbner basis (i.e. with the Buchberger algorithm) of the system in question and then to determine solutions by incrementally solving univariate equations and eliminating variables for which a solution already has been obtained.

However this method is usually only feasible for a low number of variables.

Algorithms for computing Gröbner bases are quite general tools – in this chapter we present more specialized algorithms and techniques which are being touted to be more suited for the task and subexponential in running time.

4.1 Linearization

The basic idea necessary for understanding the contents of this chapter is the concept of linearization. Consider an overdetermined system of polynomial equations where the number of equations is close to the number of terms occurring or even exceeds it. In this case it may be fruitful to consider the terms of the polynomial system as basis of a vector space. The polynomial equations then have a dual representation as linear combinations of terms in this vector space. This gives rise to the following definition:

Definition 4.1.1. A set of multivariate polynomial equations \mathcal{L} is called linearly independent if the dual linear equations are linearly independent.

If the number of linearly independent equations in the polynomial system is very close to the number of terms, we can reduce the problem of solving a multivariate polynomial system of equations to the problem of solving a linear one. For this a multitude of efficient algorithms exist, i.e. Gaussian elimination.

We are only interested in solutions lying in the base field \mathbb{F} and not in the algebraic closure $\overline{\mathbb{F}}$; therefore we reduce each equation modulo the ideal $\mathfrak{J} := \{x^q - x \mid \forall x \in X\}$.

This is done by the algorithm `reduce`, which is not specified. In \mathbb{F}_2 this has the convenient consequence that the $t \cdot x = t$ for all variables x of t .

4.2 The XL Algorithm

Again, consider a system of low-degree polynomial equations $\mathcal{L} = \{l_i = 0\}$ in variables X over a finite field \mathbb{F} . The XL algorithm incrementally increases the size of the initial set of equations \mathcal{L} by adding new algebraically dependent, yet linearly independent equations to it. This is achieved by multiplying each equation $(l = 0) \in \mathcal{L}$ with each possible term $t \in T(X)$ such that $\deg(t \cdot l) \leq D$, where D is the degree bound – the maximum degree an equation may have.

We first give a verbatim description from the original paper and then comment on the individual steps of this basic, yet effective algorithm:

Multiply: Generate all the products $\prod_{j=1}^k x_{i_j} \cdot l_i$ with $k \leq D - 2$.

Linearize: Consider each variable in the x_i of degree $\leq D$ as a new variable and perform Gaussian elimination on the equations obtained in 1. The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last.

Solve: Assume Step 2 yields at least one univariate equation in the powers of x_1 . Solve this equation over the finite field.

Repeat: Simplify the equations and repeat the process to find the values of the other variables.

Remark 4.2.1. The number of linearly independent equations obtained by combining all of the equations obtained in step 1 is generally called *Free*. Let the number of terms – including the constant term – in all of these equations be called T .

We then hope to achieve $Free = T - C$ for obtaining a solution, ideally $C = 1$. If $Free = T$ and one of the equations contains a constant term, then the polynomial system obviously is unsolvable.

If $Free = T - 1$, the term ordering chosen in step 2 is irrelevant, otherwise lexicographically ordering the terms will fulfill the condition in step 2. Note well, however, that a permutation of variables (swapping x_1 and x_3 for example before lexicographically ordering the terms) may lead to success if Gaussian elimination has failed to produce a univariate equation for the current order of variables.

When solving the equations in step 3, the univariate equation obtained may have more than one root in the field \mathbb{F} .

4.3 Relinearization

Historically, relinearization [12] was an intermediate step to the XL algorithm. It has been proven ([3]) that relinearization for a given parameter D generates a subset of

equations the XL algorithm produces for the same degree bound D , therefore we will not elaborate on it.

4.4 Extended Sparse Linearization

Instead of generating all equations of the equations up to degree D as in the XL case, we can be more selective. What the XSL technique advocates is splitting a large system of equations \mathcal{L} into several smaller systems $\mathcal{L}_1, \dots, \mathcal{L}_k$, and multiplying the equations of each of these systems with products of terms occurring in other systems. The partitioning chosen by the authors of [5] is to regard each S-Box and the linear layer of each round as a separate system. Having reached a certain threshold of linearly independent equations, a final step is applied to generate some more equations such that we can linearize the system. In contrast to the XL algorithm, the goal of the XSL technique is to not have to iteratively solve the system; the expectation rather is to have *Free* linearly independent equations for $Free + 1$ terms and thus being able to uniquely solve the system.

In order to correctly identify the different polynomial systems after the partitioning we need to name or tag them. In the algorithms presented this is done by having a mapping from the natural numbers into the subsets of the polynomial ring $\mathbb{F}[X]$, thus assigning each system a number.

The “expansion” step of the XSL technique can be found in the algorithm `xsl_expansion`.

Two avenues of attack are presented in [4] of which only the first is published in [5]:

1. Equations for $N_r + 1$ executions of the cipher are used without paying attention to the key schedule.
2. Equations for $\lceil N_k/N_b \rceil$ executions of the cipher **and** equations for the key schedule are used.

Equations for the second case are generated by the algorithms shown in the previous chapter.

4.4.1 The final step

The final step tries to increase the number of linearly independent equations *Free* when $Free/T \approx 1$ without increasing the number of terms T . Let \mathcal{F} be a set of polynomials over \mathbb{F} and the set $T(\mathcal{F})$ the set of terms occurring in \mathcal{F} .

Then we define $T_x(\mathcal{F})$ to be the set of all terms that can be multiplied by a variable $x \in X$ and still are in $T(\mathcal{F})$:

$$T_x(\mathcal{F}) := \{t \in T(\mathcal{F}) \mid (t \cdot x) \in T(\mathcal{F})\}$$

For the final step we first need to pick two variables $x_i, x_j \in X$. We then obtain a system \mathcal{F}_{x_i} by rewriting all polynomials of \mathcal{F} such that each term $t \notin T(\mathcal{F})_{x_i}$ is represented as a linear combination of terms in $T(\mathcal{F})_{x_i}$. A certain number of equations need not be rewritten, because already all terms occurring in them are in $T(\mathcal{F})_{x_i}$. These equations

are called “excessive equations” for x_i . We do likewise for \mathcal{F}_{x_j} . This procedure can either succeed or fail. If it fails, depending on which rewrite step failed, we need to choose either $x_i \in X$, $x_j \in X$ or both x_i, x_j anew until it succeeds.

Now we can multiply the exceeding equations for x_i with the variable x_j – let us call the set of those \mathcal{L}_{x_i} – and the exceeding equations for x_j with x_i – which we shall call \mathcal{L}_{x_j} – and obtain new equations, most likely linearly independent from the equations we already have. By rewriting all terms of \mathcal{L}_{x_i} not occurring in \mathcal{L}_{x_j} with the representation we have in \mathcal{F}_{x_j} we can iterate the process. A more precise description can be found in the algorithm `final_step`.

In [6] the above final step is called XL2.

4.4.2 An example for XSL

Example 4.4.1. Consider the 0-dimensional ideal $\mathcal{I} \triangleleft \mathbb{F}_2[x_0, \dots, x_3]$ generated by the following equations:

$$\left. \begin{array}{l} x_0x_2 + x_3 + x_1 + 1 = 0 \\ x_0x_1 + x_2 + x_3 = 0 \\ x_1x_3 + x_2x_3 + x_2 = 0 \end{array} \right\} \mathcal{L}_1$$

$$\left. \begin{array}{l} x_3 + x_1x_2 + 1 = 0 \\ x_0 + x_1 + x_2 + 1 = 0 \end{array} \right\} \mathcal{L}_2$$

The sets T_i denote all terms occurring in system \mathcal{L}_i except for the constant term:

$$T_1 = \{ x_0x_2, x_3, x_1, x_0x_1, x_2, x_1x_3, x_2x_3 \}$$

$$T_2 = \{ x_3, x_1x_2, x_0, x_1, x_2 \}$$

We multiply all equations in \mathcal{L}_1 with all terms occurring in \mathcal{L}_2 and vice versa:

$$\left. \begin{array}{l} x_0x_1x_2 + x_0x_1 = 0 \\ x_0x_1x_3 + x_0x_1x_2 + x_0x_1 = 0 \\ x_0x_2 + x_0x_1x_2 = 0 \\ x_0x_2x_3 + x_0x_2 + x_0x_1x_2 = 0 \\ x_1x_2 + x_0x_1 = 0 \\ x_1x_2 + x_0x_2 = 0 \\ x_1x_2x_3 = 0 \\ x_1x_2x_3 + x_0x_1x_3 = 0 \\ x_1x_2x_3 + x_0x_2x_3 = 0 \\ x_1x_3 + x_1x_2 + x_1 = 0 \\ x_2x_3 + x_2 + x_1x_2 = 0 \\ x_3 + x_2x_3 + x_1x_3 = 0 \\ x_3 + x_2x_3 + x_1x_3 + x_0x_3 = 0 \end{array} \right\} T_1 \cdot \mathcal{L}_2$$

$$\left. \begin{array}{l} x_0x_2x_3 + x_0x_1x_3 + x_0 = 0 \\ x_0x_3 + x_0x_2 + x_0x_1 = 0 \\ x_0x_3 + x_0x_2 + x_0x_1 + x_0 = 0 \\ x_1x_2 = 0 \\ x_1x_2x_3 + x_0x_1x_2 = 0 \\ x_1x_2x_3 + x_1x_2 + x_0x_1x_2 = 0 \\ x_1x_3 + x_0x_1x_2 = 0 \\ x_1x_3 + x_0x_2x_3 = 0 \\ x_1x_3 + x_1x_2 + x_0x_1 = 0 \\ x_1x_3 + x_1x_2x_3 + x_1 = 0 \\ x_2x_3 + x_2 + x_0x_1x_2 = 0 \\ x_2x_3 + x_2 + x_1x_2 + x_0x_2 = 0 \\ x_2x_3 + x_2 + x_1x_2x_3 = 0 \\ x_3 + x_2x_3 + x_0x_1x_3 = 0 \\ x_3 + x_2x_3 + x_1x_3 = 0 \end{array} \right\} T_2 \cdot \mathcal{L}_1$$

Each term of $\mathcal{L}' = \{ \mathcal{L}_1, \mathcal{L}_2, T_1 \cdot \mathcal{L}_2, T_2 \cdot \mathcal{L}_1 \}$ including the constant term can now be considered to be an independent variable, thus transforming the problem of solving a multivariate polynomial system into solving a linear system:

$$\begin{pmatrix} x_3 & x_2x_3 & x_2 & x_1x_3 & x_1x_2x_3 & x_1x_2 & x_1 & x_0x_3 & x_0x_2x_3 & x_0x_2 & x_0x_1x_3 & x_0x_1x_2 & x_0x_1 & x_0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Diagonalizing this system yields:

$$\begin{pmatrix} x_3 & x_2x_3 & x_2 & x_1x_3 & x_1x_2x_3 & x_1x_2 & x_1 & x_0x_3 & x_0x_2x_3 & x_0x_2 & x_0x_1x_3 & x_0x_1x_2 & x_0x_1 & x_0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

from which we can read off the solution $x_0 = x_1 = 0$, $x_2 = x_3 = 1$.

Algorithm 21 collect_terms

Input: $\mathcal{F} \subseteq \mathbb{F}[X]$

Output: $\mathcal{T} \subseteq T(X)$

```

 $\mathcal{T} \leftarrow \emptyset$ 
for  $f \in \mathcal{F}$  do
   $f' \leftarrow f$ 
  while  $f' \neq \emptyset$  do
     $\mathcal{T} \leftarrow \mathcal{T} \cup \text{HT}(f')$ 
     $f' \leftarrow f' - \text{HM}(f')$ 
  end while
end for

```

Algorithm 22 linearize

Input: $\mathcal{F} \subseteq \mathbb{F}[X]$ **Input:** $\lambda : T(X) \rightarrow Y, \rho : Y \rightarrow T(X)$ **Output:** $\mathcal{F}' \subseteq \mathbb{F}[X]$

```

for  $f \in \mathcal{F}$  do
   $f' \leftarrow f$ 
  while  $f' \neq 0$  do
     $j \leftarrow \lambda(\text{HT}(f'))$ 
     $M_{i,j} \leftarrow \text{HC}(f')$ 
     $f' \leftarrow f' - \text{HM}(f')$ 
  end while
end for
 $M \leftarrow \text{gauss}(M)$ 
 $\mathcal{F}_s \leftarrow \emptyset$ 
for  $i \leftarrow 0$  to  $\text{Rows}(M) - 1$  do
   $f \leftarrow 0$ 
  for  $j \leftarrow 0$  to  $\text{Cols}(M) - 1$  do
    if  $M_{i,j} \neq 0$  then
       $f \leftarrow f + M_{i,j} \cdot \rho(j)$ 
    end if
  end for
   $\mathcal{F}_s \leftarrow \mathcal{F}_s \cup f$ 
end for

```

Algorithm 23 `xsl_expansion`

Input: $(\mathcal{F}_0 \subset \mathbb{F}[X], \dots, \mathcal{F}_{N-1} \subset \mathbb{F}[X])$ **Input:** $P \in \mathbb{N}$ **Output:** $F' \in \mathbb{F}[X]$

```

 $F' \leftarrow \emptyset$ 
for  $i \leftarrow 0$  to  $N - 1$  do
   $F' \leftarrow F' \cup F_i$ 
   $(\pi_0, \dots, \pi_{P-2}) \leftarrow (0, 1, \dots, P - 2)$ 
  while do
    for  $j \leftarrow 0$  to  $P - 2$  do
      if  $\pi_j \neq i$  then
         $T' \leftarrow \text{collect\_terms}(\mathcal{F}_{\pi_j})$ 
         $T \leftarrow T \times T'$ 
      end if
    end for
     $F' \leftarrow F' \cup \text{multiply\_selectively}(F_i, T)$ 
     $\pi \leftarrow \text{next\_selection}(\pi, P - 2, n)$ 
  end while
end for

```

Algorithm 24 `can be multiplied with`

Input: $\mathcal{T} \subseteq T(X)$ **Input:** $x \in X$ **Output:** $\mathcal{T}' \subseteq T(X)$

```

 $\mathcal{T}' \leftarrow \emptyset$ 
for  $t \in \mathcal{T}$  do
  if  $(x \cdot t) \in \mathcal{T}$  then
     $\mathcal{T}' \leftarrow \mathcal{T}' \cup t$ 
  end if
end for

```

Algorithm 25 multiply_selectively

Input: $\mathcal{F} \subseteq \mathbb{F}[X]$ **Input:** $T \in T(X)$ **Input:** $D \in \mathbb{N}_0$ **Output:** $\mathcal{F}' \subseteq \mathbb{F}[X]$

```

 $\mathcal{F}' \leftarrow \emptyset$ 
for  $f \in \mathcal{F}$  do
  for  $t \in T$  do
     $f' \leftarrow t \cdot f$ 
    if  $D = 0 \vee \deg(f') \leq D$  then
       $\mathcal{F}' \leftarrow \mathcal{F}' \cup f'$ 
    end if
  end for
end for

```

Algorithm 26 create_rewrite_map

Input: $M \in \mathbb{F}^{u \times v}$ **Input:** $n \in \mathbb{N}$ **Input:** $\kappa : \mathbb{N} \rightarrow T(X)$ **Output:** $\omega : T \rightarrow \mathbb{F}[X]$ **Output:** $\mathcal{F}' \subseteq \mathbb{F}[X]$

```

for  $i = 0$  to  $\text{Rows}(M)$  do
   $f \leftarrow 0$ 
  for  $j = n$  to  $\text{Cols}(M)$  do
     $f \leftarrow f + \kappa(j) \cdot M_{i,j}$ 
  end for
  if  $i < n$  then
     $\omega(\kappa(j)) \leftarrow f$ 
  else
    if  $f \neq 0$  then
       $\mathcal{F}' \cup f$ 
    end if
  end if
end for

```

Algorithm 27 create_equiv_map

Input: $\mathcal{T} \subseteq T(X)$ **Input:** $x \in X$ **Output:** $\lambda : T(X) \rightarrow \mathbb{N}$ **Output:** $\kappa : \mathbb{N} \rightarrow T(X)$ **Output:** $n \in \mathbb{N}$ $T_x \leftarrow \text{can_be_multiplied_with}(T, x)$ $T_r \leftarrow T \setminus T_x$ $v \leftarrow 1$ **for** $t \in T_r$ **do** $\lambda(t) \leftarrow v$ $\kappa(v) \leftarrow t$ $v \leftarrow v + 1$ **end for****for** $t \in T_x$ **do** $\lambda(t) \leftarrow v$ $\kappa(v) \leftarrow t$ $v \leftarrow v + 1$ **end for** $n \leftarrow T_r$

Algorithm 28 create_rewrite_map

Input: $\mathcal{T} \subseteq T(X)$ **Input:** $x \in X$ **Output:** $\lambda : T(X) \rightarrow \mathbb{N}$ **Output:** $\kappa : \mathbb{N} \rightarrow T(X)$ **Output:** $n \in \mathbb{N}$ $T_x \leftarrow \text{can_be_multiplied_with}(T, x)$ $T_r \leftarrow T \setminus T_x$ $v \leftarrow 1$ **for** $t \in T_r$ **do** $\lambda(t) \leftarrow v$ $\kappa(v) \leftarrow t$ $v \leftarrow v + 1$ **end for****for** $t \in T_x$ **do** $\lambda(t) \leftarrow v$ $\kappa(v) \leftarrow t$ $v \leftarrow v + 1$ **end for** $n \leftarrow T_r$

Algorithm 29 final_step

Input: $\mathcal{F} \subset \mathbb{F}[X]$ **Input:** $v_0, v_1 \in X$ **Output:** $\mathcal{F}' \subset \mathbb{F}[X]$

```

 $\mathcal{T} \leftarrow \text{collect\_terms}(F)$ 
for  $i = 0$  to  $1$  do
   $(\lambda, \kappa, n) \leftarrow \text{create\_equiv\_map}(\mathcal{T}, v_i)$ 
   $M \leftarrow \text{linearize}(\mathcal{F}, \lambda)$ 
  if  $\text{rank}(M) < n$  then
    return failure
  end if
  if  $\text{diagonalize}(M, n) < n$  then
    return failure
  end if
   $(\mathcal{G}_i, \rho_i) \leftarrow \text{create\_rewrite\_map}(M, n, \kappa)$ 
end for
 $i \leftarrow 0$ 
success  $\leftarrow 2$ 
while success  $> 0$  do
   $\tilde{\mathcal{F}} \leftarrow \emptyset$ 
  for  $g \in G_i$  do
     $\tilde{F} \leftarrow \tilde{F} \cup g \cdot v_i$ 
  end for
   $\tilde{F} \leftarrow \text{rewrite}(\tilde{F}, \rho_{1-i})$ 
  if  $\#(G_i \cup \tilde{F}) = \#G_i$  then
    success  $\leftarrow \text{success} - 1$ 
  else
     $G_i \leftarrow G_i \cup \tilde{F}$ 
    success  $\leftarrow 2$ 
  end if
   $i \leftarrow 1 - i$ 
end while

```

Algorithm 30 diagonalize

Input: $M \in \mathcal{F}^{u \times v}$ **Output:** $M' \in \mathcal{F}^{u \times v}$ *{ compute row-echolon form of matrix M }* $M' \leftarrow \text{gauss}(M)$ **for** $k \leftarrow n - 1$ **downto** 0 **do** **if** $M'_{k,k} = 0$ **then** **return failure** **end if** **if** $M'_{k,k} \neq 1$ **then** $c \leftarrow M'_{k,k}$ **for** $l \leftarrow k$ **to** $\text{Cols}(M')$ **do** $M'_{k,l} \leftarrow \frac{M'_{k,l}}{c}$ **end for** **end if** **for** $l \leftarrow k + 1$ **to** n **do** **if** $M'_{k,l} = 0$ **then** *{ subtract row l from row k }* $M'_k \leftarrow M'_k - M'_l$ **end if** **end for****end for**

Chapter 5

Observations and experimental results

5.1 Implementation

The author has implemented all algorithms presented in C++. For the finite field arithmetic and the linear algebra operations the package NTL [20] was used. The Standard Template Library (STL) was employed for building data structures such as terms, polynomials and sets thereof.

Two programs were the result of this activity

GREG The **G**eneralized **R**ijndael **E**quation **G**enerator. A tool to quickly build equations from a set of Mini-Rijndael parameters. For consumption by the XSL program, the equations are partitioned into sets – one for each linear layer of a round and one for every S-Box for example – and given short symbolical names. A first generation prototype of this program was built in Singular [11], a computer algebra system for polynomial computations which was also used to verify the correctness of the equations generated.

GREG can also be used to trace the cipher in question step by step and to scan the key space for duplicate keys, given a plaintext/ciphertext pair – if the key space is small enough. The last option can be used to make sure a unique solution exists for a known plaintext attack in the simulations.

XSL A program which allows the user to experiment with the XSL strategy. The program executes the following steps:

1. Parse several named systems of equations
2. Read in specification which systems shall be multiplied with the terms of which other systems. I.e. if systems $\mathbb{L}_1, \dots, \mathbb{L}_k$ exist which are named $11, \dots, 1k$ the line containing the specification $11\ 12$ means that the program is to multiply the equations of \mathbb{L}_1 with all terms occurring in \mathbb{L}_2 .
3. For a pair of variables x_i, x_j specified perform the Final Step operation

4. Linearize the system
5. Find all univariate equations and print possible solutions for each.

5.2 The original examples

Unfortunately, the simulations presented in the appendix of [3] do not demonstrate that XSL indeed works. The final step computation apparently has been left out.

The tables in C.1 of the paper among other parameters lists $\frac{Free}{T-T'}$ for each toy cipher variation – however $\frac{Free}{T-T'} \geq 1$ only is a necessary but not a sufficient condition. Finding variables for which the final step works is not always possible.

Indeed, when the author tried to reproduce the results for toy ciphers with 2 and 10 rounds (for which example files are available for download), he found that for no variable x the set T'_x is suitable for rewriting the equations. However, we note that solving both systems of equations by computing a suitable lexicographical Groebner basis using SINGULAR is readily possible and does neither require extensive amounts of memory nor does it consume much time.

5.3 Applying XSL to a Mini-Rijndael

The authors of [5] in section 6.4 estimate the the parameter P for which an XSL attack of the first type should work (without key schedule equations) to be

$$P = \left\lceil \frac{t-r}{s} \right\rceil + o(1)$$

Note that the attack described in the paper is the Compact XSL variant, this approximation however applies to both the original XSL [4] as the optimized Compact XSL.

For a 3-bit S-Box we find the number of equations to be $r = 8$ and the number of terms occurring in these equations to be $t = 16$. We thus expect the attack to work for $P = 2$. We choose the following parameters for our Mini-Rijndael:

- $r = 1, N_a = 2, N_b = 2, N_k = 2$
- $m(\theta) = \theta^3 + \theta + 1$. This an irreducible polynomial for the modulus 2 according to [14].
- $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$
- $M_{\text{mix}} = \begin{pmatrix} \theta + 1 & 1 \\ 1 & \theta + 1 \end{pmatrix}$

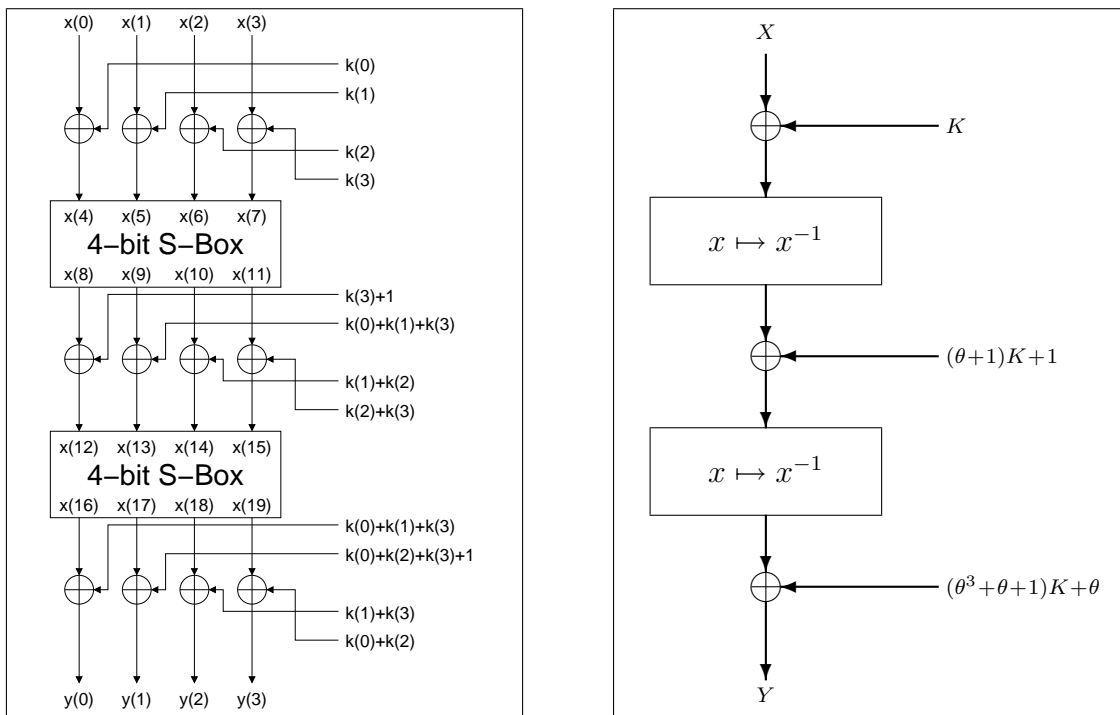
We obtain 84 equations in 109 terms. After applying XSL expansion, we obtain 8226 equations, of which 5226 are linearly independent. Trying to find $\max(\{\#T'_x \mid x \in X\})$

we obtain 4566 equations to rewrite 4627 terms. This obviously means we are missing 60 linearly independent equations to perform the final step. Moreover, the author toyed with different variations of this, e.g. not all S-Boxes were included in the XSL expansion step, terms which had variables in common with the active equation were not used when expanding, however in all cases 60 linearly independent equations were missing.

When going from 1 to 2 rounds, the situation became even worse: Now we were missing > 400 equations for a successful final step!

5.4 Our own toy example

For another test of the XSL technique, we constructed the following 4-bit block cipher which is comprised of only two S-Boxes, one per round. Due to its very simplistic structure, the linear diffusion layer is completely absent.



Applying XSL to a quadratic system of equations over \mathbb{F}_2 for the above construction works remarkably well. The system is in 20 variables, contains 53 terms and 34 equations. After the XSL expansion step we obtain 142 linearly independent equations for 155 terms. Using the final step multiple times with different variables we find a unique solution for the key variables.

5.5 XSL over an extension field of \mathbb{F}_2

Our analysis and experiments suggest that an attack with XSL on Rijndael over an extension field \mathbb{F}_{2^s} with the final step proposed is unlikely to succeed.

Consider the case $\mathbb{F} := \mathbb{F}_2$: Let $\mathcal{F} \subset \mathbb{F}[X]$ be a set of polynomials over \mathbb{F} and $T_x(\mathcal{F})$ be the set of terms for any variable $x \in X$. The final step then seems to work because of the inter-relationship of terms t and t' in the set $T_x(\mathcal{F})$ such that $t = x \cdot t'$. In extension fields however, the exponent of any variable in a term may be > 1 and thus the chance we had before of obtaining a term of total degree $\leq d$ from another term with total degree d when multiplying with a variable $x \in X$ has almost completely vanished.

List of Algorithms

1	MiniRijndaelEncrypt	11
2	AddRoundKey	11
3	SubElement	12
4	ShiftRows	12
5	MixColumns	13
6	MiniRijndaelKeySchedule	14
7	generate_equations_gf2	23
8	compute_shiftrows_matrix	23
9	generate_subkey_equations_gf2	24
10	generate_linear_equations_gf2	25
11	generate_sbox_equations_gf2	26
12	seperate_polynomial	26
13	generate_equations_gf2n	27
14	compute_conjugated_element	27
15	compute_conjugated_matrix	27
16	compute_coeffs_linpoly	28
17	compute_sbox_matrix	28
18	generate_linear_equations_gf2n	29
19	generate_sbox_equations_gf2n	29
20	generate_subkey_equations_gf2n	30
21	collect_terms	35
22	linearize	36
23	xsl_expansion	37
24	can_be_multiplied_with	37
25	multiply_selectively	38
26	create_rewrite_map	38
27	create_equiv_map	39
28	create_rewrite_map	39
29	final_step	40
30	diagonalize	41

Bibliography

- [1] T. Becker and V. Weispfenning. *Gröbner Bases – A Computational Approach to Commutative Algebra*. Springer Verlag Heidelberg, 1991.
- [2] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology - CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages pp. 2 – 21. Springer Verlag Heidelberg, 1991.
- [3] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, page 392 ff. Springer Verlag Heidelberg, 2000.
- [4] N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. Preprint in Cryptology ePrint Archive: <http://eprint.iacr.org/2002/044/>.
- [5] N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages pp. 267 – 287. Springer Verlag Heidelberg, 2002.
- [6] N. T. Courtois and J. Patarin. About the XL Algorithm over $gf(2)$. In *Topics in Cryptology - CT-RSA 2003: The Cryptographers' Track at the RSA Conference 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages pp. 141 – 157. Springer Verlag Heidelberg, 2003.
- [7] J. Daemen, L. Knudsen, and V. Rijmen. The block cipher Square. In *Fast Software Encryption: 4th International Workshop, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages pp. 149 – 165. Springer Verlag Heidelberg, 1997.
- [8] N. Ferguson, R. Schroepel, and D. Whiting. A Simple Algebraic Representation of Rijndael. In *Selected Areas of Cryptography: 8th Annual International Workshop*, volume 2259 of *Lecture Notes in Computer Science*, page pp. 103 ff. Springer Verlag Heidelberg, 2001.
- [9] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*, chapter Appendix A7.2. W. H. Freeman and Company, 1979.

- [10] L. Goubin and J. Patarin. Trapdoor one-way permutations and multivariate polynomials. In *ICICS '97 – International Conference on Information and Communications Security*, volume 1334 of *Lecture Notes in Computer Science*. Springer Verlag Heidelberg, 1997.
- [11] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 2.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2001. <http://www.singular.uni-kl.de>.
- [12] A. Kipnis and A. Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, page pp. 19 ff. Springer Verlag Heidelberg, 1999.
- [13] L. R. Knudsen. Contemporary Block Ciphers. In I. Damgård, editor, *Lectures on Data Security. Modern Cryptology in Theory and Practice*, volume 1561 of *Lecture Notes in Computer Science*, pages pp. 105–126. Springer Verlag Heidelberg, 1999.
- [14] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.
- [15] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages pp. 386 – 387. Springer Verlag Heidelberg, 1994.
- [16] S. Murphy and M. J. Robshaw. Comments on the Security of the AES and the XSL Technique, 2002.
- [17] S. Murphy and M. J. Robshaw. Essential Algebraic Structure within the AES. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages pp. 1 – 16. Springer Verlag Heidelberg, 2002.
- [18] K. Nyberg. Differentially Uniform Mappings for Cryptography. In *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, page pp. 55 ff. Springer Verlag Heidelberg, 1994.
- [19] J. Rosenthal. A Polynomial Description of the Rijndael Advanced Encryption Standard. *Journal of Algebra and Its Applications*, 2(2):pp. 223–236, 2003.
- [20] V. Shoup. NTL: A Library for doing Number Theory. <http://www.shoup.net/ntl>.
- [21] D. Wagner. The Boomerang Attack. In *Fast Software Encryption: 6th International Workshop, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages pp. 156 – 170. Springer Verlag Heidelberg, 1999.

Erklärung zur Diplomarbeit

Hiermit versichere ich, dass ich meine Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Bensheim, den 15.09.2003

.....
Ralf-Philipp Weinmann