ON THE SECURITY OF THE ADVANCED ENCRYPTION STANDARD

Paul D. Yacoumis

Supervisor: Dr. Robert Clarke

November 2005

Thesis submitted for the degree of Honours in Pure Mathematics

THE UNIVERSITY OF ADELAIDE
AUSTRALIA

Contents

Chapter 1

Introduction

On November 26, 2001, the block cipher Rijndael [19] was officially selected as the Advanced Encryption Standard (AES) and published as FIPS 197 [1], following a 5-year selection process by the US National Institute of Standards and Technology (NIST). The AES is intended to replace the aging Data Encryption Standard (DES) in protecting sensitive, unclassified data within US Government organisations. It has also fast become a worldwide standard within financial and commercial institutions, is the default cipher employed in many software and hardware applications that use encryption, and has been approved for even up to "top secret" level security by the US National Security Agency [42]. NIST predicts that the cipher will remain secure for at least 20-30 years [44]. It is therefore of immense interest to consider just how secure the AES actually is, and whether this prediction will be upheld. In this paper, we do just this, by evaluating attacks against the AES that are considered of significance in determining its suitability as a security measure.

Rijndael was developed according to the Wide Trail Strategy [20], a design proven to provide resistance against the well-known linear and differential cryptanalysis. However, although the cipher may have been heavily optimised against statistical attacks, the structure of Rijndael is still extremely simple, as has been emphasised by its authors, Dr. Joan Daemen and Dr. Vincent Rijmen. Recent observations on the structural properties of Rijndael [15, 19, 24, 39, 40] have prompted much research into effective ways of exploiting its simple design. From this research have stemmed several prominent attacks, and we study these and their implications on the security of the AES in this thesis.

In Chapters 2 and 3, we set up the mathematical and cryptographic background required to understand the concepts introduced in this paper. We then describe the AES algorithm in some detail in Chapter 4, and comment briefly on the cipher's resistance to classic attacks in Chapter 5.

In Chapters 6, 7 and 8, we explore, in detail, a family of Multiset Attacks on reduced-round versions of the AES. Beginning with the fundamental six-round Square-6 Attack, we then show how variants on this basic attack can be extended to 7, 8 and 9 rounds.

Turning our attention to the cipher's algebraic structure in Chapter 9, we show that a single encryption of the AES can be represented by an equation that can be seen as a generalised continued fraction. Furthermore, we show that AES can be embedded within another cipher, BES, which allows an AES encryption to be described by a very structured system of multivariate quadratic equations (an MQ-system).

In Chapters 10 and 11, we begin by discussing general methods for solving MQ-systems. We then introduce two recently devised techniques, XL and XSL, designed to handle large overdetermined systems, and evaluate their effectiveness on the AES MQ-systems.

We conclude that, barring any further unexpected advancements, the AES is currently secure, and will remain so for at least 10 years.

1.1 Glossary

We provide a glossary of terms that will be used throughout this paper.

Term	**Description**
AES	The AES algorithm.
Bit	A digit with value 0 or 1.
Byte	An ordered sequence of 8 bits.
Ciphertext	Data resulting from an encryption, or more generally, any data that is output from an encryption cipher or input to a decryption cipher.
Decryption	The reverse process of encryption, converting ciphertext into plaintext.
Encryption	Any method of converting plaintext into an illegible form (ciphertext).
Plaintext	Any data that is in non-encrypted form, or more generally, any data that is input to an encryption cipher or output from a decryption cipher.
Rijndael	The algorithm upon which the AES is based.
State	The intermediate 4×4 $\mathcal{F}$-matrix upon which the AES operations are performed.
Word	An ordered collection of 4 bytes.
XOR ($\oplus$)	Bit-wise addition (modulo 2).

Chapter 2

Mathematical Preliminaries

2.1 In the field $\mathcal{F} = \mathbf{GF}(2^8)$

There are some important concepts that are needed in order to understand, not only the the workings of the AES, but the attacks that have been developed to attempt to break it. We begin by introducing the most basic building blocks.

Definition 1. A **bit** is a digit with value 0 or 1. A **byte** is an ordered sequence of eight (8) bits. A **word** is an ordered collection of four (4) bytes.

Here a *bit* is considered an element of the finite field GF(2) $\cong \mathbb{Z}_2$, and a *byte* can therefore be viewed as an element of $\mathcal{F} = \mathrm{GF}(2^8)$. For the purpose of this paper a byte of data is presented as either the concatenation of its 8 bits, or a degree-7 polynomial (in the indeterminate x) with coefficients in $\mathbb{Z}_2$, i.e., the byte b in its concatenated form

$$b = \{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}, \tag{2.1}$$

can also be expressed as the polynomial

$$b(x) := b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i.$$

This is trivial since all representations of a finite field of prime power (in particular, $\mathcal{F}$) are isomorphic. Nevertheless, it will become helpful further down the track to utilise both representations. Furthermore, it is sometimes convenient to assign a *hexadecimal value* to a byte according to the following table:

Binary	Value	Binary	Value	Binary	Value	Binary	Value
0000	`0`	0100	`4`	1000	`8`	1100	`c`
0001	`1`	0101	`5`	1001	`9`	1101	`d`
0010	`2`	0110	`6`	1010	`a`	1110	`e`
0011	`3`	0111	`7`	1011	`b`	1111	`f`

Example. The element $\{01101101\}$ has hexadecimal value `6d` and the polynomial representation $x^6 + x^5 + x^3 + x^2 + 1$.

2.1.1 Addition

Using the polynomial representation, there is a canonical way of "adding" together two elements: Simple addition (modulo 2) of the coefficients of corresponding powers of x.

Example. Modulo-2 addition of polynomials:

$$(x^7 + x^4 + x^3 + 1) + (x^5 + x^4 + x^3 + x + 1) = x^7 + x^5 + x.$$

Definition 2. The **XOR** (**Exclusive-OR**) operation, denoted $\oplus$, is a computation corresponding to bit-wise addition (modulo 2); i.e., $1 \oplus 1 = 0 = 0 \oplus 0$ and $1 \oplus 0 = 1 = 0 \oplus 1$.

Example. Using the same two elements as in the previous example:

$$\{10011001\} \oplus \{00111011\} = \{10100010\},$$

or equivalently

$$\texttt{99} \oplus \texttt{3b} = \texttt{a2}.$$

It is clear that the XOR operation is equivalent to polynomial addition (modulo 2), and so we can use this as a more convenient method for element addition.

A useful relation on elements of the field $\mathcal{F}$ that should be noted is the following:

$$\bigoplus_{x \in \mathcal{F}} x = \texttt{00}. \tag{2.2}$$

2.1.2 Multiplication

Multiplication of two elements in $\mathcal{F}$ is not so straight forward. For instance, it is not clear how one would multiply, say, the elements `62` and `1b` to form another element of $\mathcal{F}$. Obviously, the polynomial representation will be necessary again, but with a requirement that the multiplication is performed "modulo an *irreducible* polynomial of degree 8". The fact that this polynomial has degree 8 ensures the resulting polynomial has degree ≤ 7. The irreducible polynomial chosen for the AES is

$$m(x) := x^8 + x^4 + x^3 + x + 1. \tag{2.3}$$

Multiplication modulo $m(x)$ in $\mathcal{F}$ is denoted $\bullet$

Example. `62` $\bullet$ `1b` $=$ `e1`,
or equivalently

$$\begin{aligned}(x^6+x^5+x)(x^4+x^3+x+1) &= x^{10}+x^9+x^7+x^6+\\ &\quad\; x^9+x^8+x^6+x^5+\\ &\quad\; x^5+x^4+x^2+x\\ &\equiv x^{10}+x^8+x^7+x^4+x^2+x,\end{aligned}$$

and

$$\begin{aligned}x^{10}+x^8+x^7+x^4+x^2+x \;\text{ modulo }\; &(x^8+x^4+x^3+x+1)\\ &= x^7+x^6+x^5+1.\end{aligned}$$

Note that the "modulo multiplication" as defined above is distributive, i.e., for any polynomials $a(x), b(x)$ and $c(x)$,

$$a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x).$$

Theorem 1 (**The Extended Euclidean Algorithm**). Let $\mathbb{F}$ be a field. Given any two polynomials $b(x), m(x) \in \mathbb{F}[x]$ s.t. $m(x) \neq 0$, then there exist unique polynomials $a(x)$, $c(x) \in \mathbb{F}[x]$ satisfying

$$b(x)a(x) + m(x)c(x) = d(x),$$

where $d(x)$ is the greatest common divisor of $m(x)$ and $b(x)$.

Using this theorem, it is easy to find the inverse of an element in $\mathcal{F}$. If $b(x) \neq 0$ is the polynomial representation of a byte and $m(x)$ is the polynomial in Equation (2.3), then since $\deg(b(x)) < \deg(m(x))$ and $m(x)$ is irreducible, their greatest common divisor $d(x) \equiv 1$. Therefore, by the theorem, there are polynomials $a(x)$ and $c(x)$ such that

$$b(x)a(x) + m(x)c(x) = 1.$$

This is equivalent to

$$b(x)a(x) \equiv 1 \qquad (\text{mod } m(x)),$$

and therefore

$$b^{-1}(x) \equiv a(x) \quad (\text{mod } m(x)).$$

It follows that the set of 256 possible byte values, equipped with the operations $\oplus$ and $\bullet$ as defined above, has the structure of the finite field

$$\mathcal{F} = \mathrm{GF}(2^8) \cong \frac{\mathrm{GF}(2)[X]}{(X^8+X^4+X^3+X+1)} \cong \mathrm{GF}(2)(x),$$

where x is a root of the polynomial $m(X)$.

2.1.3 Polynomials over $\mathcal{F}$

We have already seen how polynomials can be used to represent bytes. A similar representation can also be made of *words*. A word a can be viewed as a collection of four bytes

$$a = \{a_0, a_1, a_2, a_3\},$$

or the polynomial

$$a(x) := a_3x^3 + a_2x^2 + a_1x + a_0 = \sum_{i=0}^{3} a_ix^i.$$

Note the difference in convention of the indices on a word as compared to the indices on a byte (Equation (2.1)).

Addition on words can be defined, again using the bit-wise XOR operation on the coefficients of the powers of x in the polynomial representation, noting that now the coefficients are themselves elements of $\mathcal{F}$. Multiplication, on the other hand, is again a little tricky and also slightly different to the case in §2.1.2.

Suppose we have two polynomials

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad \text{and} \quad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0.$$

Then let $c(x) := a(x)b(x)$ such that

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0, \tag{2.4}$$

where

$$\begin{aligned} c_0 &= a_0 \bullet b_0, & c_4 &= a_1 \bullet b_3 \oplus a_2 \bullet b_2 \oplus a_3 \bullet b_1,\\ c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1, & c_5 &= a_2 \bullet b_3 \oplus a_3 \bullet b_2,\\ c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2, & c_6 &= a_3 \bullet b_3.\\ c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3, \end{aligned}$$

We need to perform this multiplication "modulo a polynomial of degree 4" if we are to get a word. The polynomial chosen for the AES is

$$M(x) := x^4 + 1.$$

Denote this multiplication modulo $M(x)$ by $\otimes$

Let $d(x) := a(x) \otimes b(x)$ such that

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0,$$

where $d_0, \ldots d_3$ are functions of the coefficients c_i in Equation (2.4). It is useful here to recognise that

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4},$$

in order to calculate these relations from $c(x)$. It turns out that:

$$\begin{aligned}
d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3,\\
d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3,\\
d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3,\\
d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3,
\end{aligned}$$

and if $a(x)$ is held fixed, this can be represented in the matrix form

$$\begin{bmatrix} d_0\\ d_1\\ d_2\\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1\\ a_1 & a_0 & a_3 & a_2\\ a_2 & a_1 & a_0 & a_3\\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \bullet \begin{bmatrix} b_0\\ b_1\\ b_2\\ b_3 \end{bmatrix}. \tag{2.5}$$

It is important to note that since x^4+1 is *not* irreducible over $\mathcal{F}$, multiplying by $a(x)$ does not necessarily have an inverse. Multiplication by the polynomial

$$a(x) = \mathtt{03}x^3 + \mathtt{01}x^2 + \mathtt{01}x + \mathtt{02}, \tag{2.6}$$

however does have an inverse,

$$a^{-1}(x) = \mathtt{0b}x^3 + \mathtt{0d}x^2 + \mathtt{09}x + \mathtt{0e}, \tag{2.7}$$

and is used in the AES.

2.2 Branch Number

A useful notion that describes the 'diffusion' power of a linear transformation is the *Branch Number*. Let the *byte weight* of a word, a, be the number of non-zero bytes,[1] denoted $W(a)$.

Definition 3. The **Branch Number**, $\beta(L)$, of a linear transformation L is

$$\beta(L) = \min_{a\neq 0}\Big(W(a) + W\big(L(a)\big)\Big).$$

2.3 The Birthday Paradox

The *birthday paradox* is a well known statement that if there are 23 people in a room, there is a chance of more than 50% that at least 2 of them have the same birthday. Note that this is not a paradox in the usual sense, but it is a paradox in that it is a mathematical fact yet seems to contradict intuition.

Consider 23 people in a room. Then the number of ways to choose a pair is $\frac{23\times 22}{2} = 253$. From this it does not seem unlikely that at least two of the people share the same birthday. To calculate this probability we use the formula

$$p(n) \approx 1 - \exp\Big(-\frac{n^2}{2N}\Big),$$

where, more generally, n is the number of elements and N is the number of possible values each element can take; in this case, $n = 23$ and $N = 365$.

[1]Note that the concept of the byte weight differs from that of the more common Hamming weight, which denotes the number of non-zero bits.

Chapter 3

Cryptographic Terminology and Preliminaries

3.1 Block Ciphers

Block ciphers, such as the AES, play a fundamental role in modern cryptography. A block cipher is an algorithm that takes a fixed-length string of input bits (known as the *plaintext*) and transforms it, using several iterations (or *rounds*) of complicated operations, into a string of output bits of the same length (known as the *ciphertext*). This fixed length is known as the *block length.* A *cipher key* is also used within the cipher to customise its operations, so that decryption is possible only with knowledge of the particular key used for encryption.

Block ciphers form a class of *private-key* (or symmetric) cryptosystems, in which the key is known only to the sender and receiver. Private-key systems are generally faster than *public-key* (or asymmetric) cryptosystems, such as RSA, in which the encryption key, known to the public, differs from the secret decryption key.

3.2 Cryptanalysis of a Cipher

Cryptanalysis of a cipher involves studying weaknesses in the implementation of the algorithm, or the algorithm itself, in order to gather previously unknown information about the plaintext or the cipher key. Information regarding the key is generally more useful as it would allow decryption of all ciphertexts formed using that key.

Any cryptanalytic technique imposed on a cipher is known as an *attack.* An attack which takes less time than trying all possible keys is called a *shortcut attack.* A cipher is said to be *weakened* when a shortcut attack is found, and *broken* when a computationally feasible attack (one that will take both a reasonable amount of computer resources by todays standards and a reasonable amount of time to complete) is found. The terms broken and weakened are typically used interchangeably.

3.2.1 Attack Scenarios

There are, in general, four levels of mathematical attack that can be mounted on an algorithm, depending on the degree of access the cryptanalyst has to the data being processed:

1. **Brute Force Attack:**
   The adversary systematically tries every possible key until the correct one is found. For example, if the algorithm uses a 128-bit key length, one would need to try all 2^{128} possible combinations (on average, only 2^{127} combinations are required) before the correct one is found.

2. **Ciphertext-Only Attacks:**
   In this form of attack, the cryptanalyst has access only to a set of ciphertexts and no knowledge of the plaintext. Most applications require educated guesses as to the contents or wording of the corresponding plaintexts in order to deduce information about the key.

3. **Known Plaintext Attacks:**
   Here, a collection of data in both plaintext and ciphertext forms is known to the adversary. Again, the goal is usually to find the key which results in these plaintext/ciphertext pairs.

4. **Chosen Plaintext Attacks:**
   Sometimes referred to as *differential cryptanalysis*; the cryptanalyst in this case can choose which plaintext/ciphertext pairs to use in their analysis of the algorithm. This usually requires access to the cipher itself, making it a less practical, yet much more powerful attack.

3.2.2 Reduced Ciphers

When designing new attacks on block ciphers, we usually cannot expect them to immediately work on the full-size cipher. This is why cryptanalysts generally begin with analysing "reduced" versions of the cipher, where "reduced" typically means that the number of rounds are decreased. Attacks are first formulated for the reduced cipher and then adapted and refined to work on increasingly more rounds.

Although an attack may be found to be effective at "breaking" a reduced cipher, if it does not extend efficiently, it will generally be of no immediate threat to the security of the cipher. Nevertheless, further improvements could be found that do extend to the full-blown cipher, and so shortcut attacks on reduced ciphers should not be readily dismissed. We consider several reduced cipher attacks in Chapters 5–8.

3.3 Complexity Analysis of Algorithms

In the analysis of an algorithm (in particular, a cryptographic attack), it is usually important to talk about its *complexity*. Using this concept, we can study an algorithm's efficiency and compare its performance against other algorithms, without having to implement it on a specific computer. There are two types of complexity usually considered, namely *time complexity* and *space complexity*. The former is concerned with approximating the upper bound for the algorithm's running time, whilst the latter approximates the temporary storage or memory required for running the algorithm. We will be dealing primarily with time complexity (also known as *running time*) in this paper.

3.3.1 The $\mathcal{O}$ Notation

In order to be able to talk about complexity, we first need the following notation on functions:

Definition 4. A function $g(n)$ is said to be $\mathcal{O}(f(n))$, for another function $f(n)$ (pronounced "Oh", or "Big Oh" of $f(n)$), if $\exists\, c, N \in \mathbb{R}$ such that, $\forall\, n \geq N$ we have $g(n) \leq cf(n)$. We write $g(n) = \mathcal{O}(f(n))$.

Remark. The $\mathcal{O}$ notation serves as an upper bound for $g(n)$. The function $cf(n)$ is not unique and could, in fact, be substantially larger than $g(n)$.

Example. $5n^2 + 15 = \mathcal{O}(n^2)$ since $5n^2 + 15 \leq 6n^2$ for $n \geq 4$, but also $5n^2 + 15 = \mathcal{O}(n^3)$ since $5n^2 + 15 \leq n^3$ for all $n \geq 6$.

There are some typical $\mathcal{O}$-bounds that are used extensively in analysing the complexity of an algorithm. These are: $\mathcal{O}(1), \mathcal{O}(n), \mathcal{O}(n^a)$, and $\mathcal{O}(a^n)$. They are called constant, linear, polynomial and exponential, respectively. If we have a bound of $\mathcal{O}(a^{a^n})$, then the complexity is said to be double exponential.

3.3.2 Time Complexity

How do we analyse an algorithm's running time without running it? One would expect that we need to count the number of steps it performs. But just counting the steps will not be sufficient, since there may be many different steps, each of which take a different amount of time to execute. Listing all of the types of steps accurately would (in most cases) be very difficult, if not impossible, given that the time taken to complete a task generally depends on the input.

For this reason, we usually approximate the number of operations by the number of *basic steps*. A basic step is one which constitutes the major part of the algorithm. If $\mathcal{O}(f(n))$ is a time bound for the number n of basic step, then $\mathcal{O}(f(n))$ is also a bound for the total number of operations and we say that the time complexity, or running time, of the algorithm is $\mathcal{O}(f(n))$.

Time complexity is also sometimes expressed simply in units of a particular operation. For instance, in a brute force attack on a cipher with 128-bit keys, we have to check all 2^{128} key combinations by decrypting the ciphertext with each of these values. Using a single decryption (or equivalently, a single encryption) as our "operation", we then say that the brute force attack has a running time of 2^{128}. It turns out that units of encryptions are very useful for complexity evaluations of cryptographic attacks, and we will use these units in this paper, unless otherwise stated.

3.3.3 A Note on Gaussian Elimination

In Chapters 10 and 11, we introduce several techniques for solving large systems of polynomial equations. These techniques incorporate the well known *Gaussian Elimination* method for determining solutions to linear systems. The method, when applied to a matrix, produces what is known as "reduced row echelon form". The number of steps required for complete Gaussian elimination is approximately proportional to n^3 if the matrix is of dimension $n \times n$, and therefore is said to have complexity $\mathcal{O}(n^\omega)$, where $\omega = 3$ is called the Gaussian complexity exponent.

Several improvements have been developed, such as Strassen's algorithm, with exponent $\omega = 2.807$, and the Coppersmith-Winograd algorithm, with the current best known exponent of $\omega = 2.3766$ [33]. The Coppersmith-Winograd algorithm is often used to provide time complexity bounds for other algorithms, but appears not to be particularly practical for actual applications.

3.4 Current Computing Capabilities

The term "broken" can hold a different meaning for a cryptographer than an a engineer. To a cryptographer, any shortcut attack on a cipher will leave it broken, however the attack may still be infeasible, and in this case an engineer would not consider the cipher broken, merely weakened. Current computing capabilities and time restraints obviously determine whether an attack is feasible and therefore "breaks" the cipher.

In 2004, it was estimated that it would take a billion modern computer processers around 30 years to perform 2^{90} computations [43]. Currently, the world's fastest computer is capable of performing 135.5 trillion ($\approx 2^{47}$) computations per second, and by 2011, Japan hope to have a supercomputer able to perform over a quadrillion ($\approx 2^{50}$) computations per second [50]. An AES encryption, as we will see, requires many computations, say 2^8.

As an example, if an attack has complexity of (only) 2^{67}, it would take approximately 1 year at 2^{50} computations per second, which is clearly infeasible for most applications.

According to Moore's law [38], it is expected that computing power will double every 18 months. Therefore, even if a low complexity attack is currently infeasible, it may well become practical in a few years. Furthermore, with the speculation that quantum computing may become viable in the future, we could potentially see some problems reduced from years to seconds [45].

Chapter 4

The AES Algorithm

We now have enough background to introduce the design of the AES algorithm. The AES is a block cipher with a 128-bit block length and variable key length of 128, 192, or 256 bits.[1] The three AES versions are thus termed AES-128, AES-192 and AES-256, respectively.

4.1 The State

Operations within the AES are based on functions acting on matrices of bytes. For this reason, a plaintext block is arranged into a *byte array* (or $\mathcal{F}$-matrix) of dimension 4×4,[2] and this byte-structure is fully respected throughout an encryption. The array can be viewed as either a matrix of bytes, or as columns of words.

Definition 5. The **State** is the intermediate 4×4 $\mathcal{F}$-matrix upon which the cipher's operations are performed.

A data block $\{b_0, \ldots, b_{15}\}$ of bytes is arranged into the state and relabelled in the following way

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix} =: \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix},$$

where any element $s_{i,j}$ is considered as $s_{i \bmod 4, j \bmod 4}$.

[1]Rijndael is more versatile than the AES in that it has a variable block length and key length of multiples of 32 between 128 and 256 bits, inclusive.

[2]The general version of Rijndael uses a $\mathcal{F}$-matrix of dimension $4 \times N_b$, where N_b denotes the block length divided by 32, see [1].

4.2 Round Structure

An AES encryption has a minimum of 9 full rounds (depending on the key size), as well as an initial key addition step and modified final round. The total number of rounds used is determined from the following table:

Version	AES-128	AES-192	AES-256
# of Rounds	10	12	14

Each full round (also known as an *inner round*) utilises four operations: ByteSub, ShiftRows, MixColumns and KeyAddition. These operations are grouped into three functional steps termed "layers". We now describe each layer in detail.

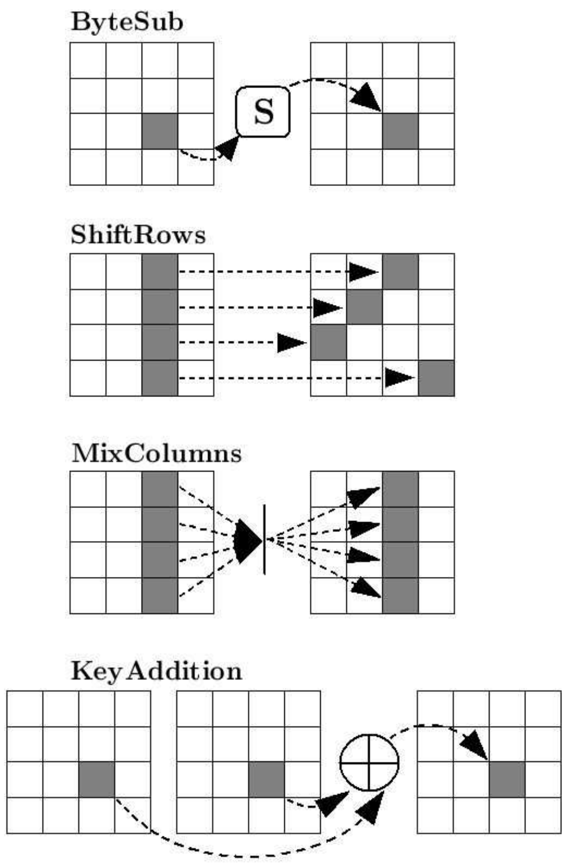

Figure 4.1: The action of the round operations on bytes of the state.

4.2.1 The Non-Linear Layer

This first layer makes use of a non-linear bijective substitution table called an **S-box**. The S-box element $z := f(g(x))$ corresponding to an input byte x is given by the following transformation:

1. Map x to $y = g(x) := x^{(-1)}$ where $x^{(-1)}$ is defined by

$$x^{(-1)} := x^{254} = \begin{cases} x^{-1} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

   Here, x^{-1} is the multiplicative inverse of x over $\mathcal{F}$ as described in §2.1.2.

2. Define $f(y) := (L_A \cdot y) \oplus \texttt{63}$, where L_A is an 8×8 GF(2)-matrix. The affine transformation $z = f(y)$ is described by the matrix equation

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \tag{4.1}$$

Note that the only non-linear part of this transformation is the map $x \mapsto x^{-1}$ (in fact, it is the only non-linear operation performed within the AES). Define a function, `BS` (ByteSub),[3] which acts on the state (or any collection of bytes), substituting each element x with its S-box replacement z. The inverse of this function, $\texttt{BS}^{-1}$, uses an **Inverse S-box** lookup.

Note. The non-linear layer is essentially described by a layer of 16 S-boxes applied in parallel on the bytes of the state (see Figure 4.1).

4.2.2 The Diffusion Layer

The linear layer is comprised of two operations: The first in the sequence is the linear function `SR` (ShiftRows). This performs a row-wise shift on the state, whereby row i gets left-shifted by i places. The inverse of this function, $\texttt{SR}^{-1}$, necessarily right-shifts row i by i places. The functions `SR` and $\texttt{SR}^{-1}$ acting on the state are given explicitly by

$$\begin{aligned} \texttt{SR} &: \quad s_{i,j} \mapsto s_{i,j-i \bmod 4}, \\ \texttt{SR}^{-1} &: \quad s_{i,j} \mapsto s_{i,j+i \bmod 4}. \end{aligned}$$

[3]The AES specification [1] denotes this function as SubBytes.

The second function, denoted `MC` (MixColumns), performs column-wise mixing of the state. The 'mixing' is performed by multiplying each column s_j of the state by a fixed word a to form a new column s'_j, i.e.,

$$s'_j(x) = \mu(s_j(x)) := a(x) \otimes s_j(x).$$

Recall the invertible polynomial $a(x)$ as defined in Equation (2.6):

$$a(x) = \texttt{03}x^3 + \texttt{01}x^2 + \texttt{01}x + \texttt{02}.$$

Substituting this into the matrix Equation (2.5) gives

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix}, \tag{4.2}$$

where $\{s_{0,j}, \ldots s_{3,j}\}$ represents column j of the state, and $\{s'_{0,j}, \ldots s'_{3,j}\}$ forms column j of the transformed state. This transformation equation describes the column-wise action, μ, of `MC` on the state.

The inverse operation, $\texttt{MC}^{-1}$, uses the polynomial $a^{-1}(x)$ as in Equation (2.7):

$$a^{-1}(x) = \texttt{0b}x^3 + \texttt{0d}x^2 + \texttt{09}x + \texttt{0e}.$$

Observe that $\mathcal{F}$-matrix multiplication is with respect to the element-wise operation $\bullet$. However, we will drop this notation when it is clear from the context that this is the operation performed.

Note. The coefficients of the polynomial $a(x)$ above were chosen for the AES to maximise the Branch Number (§2.2) of the operation `MC`, i.e., if a column of the state has only one non-zero (active) byte, then `MC` will produce an output column with all four bytes active. Therefore the achieved maximum Branch Number is 5. If a column has 2 active bytes, the output will be a column with at least 3 active bytes, etc. This ensures optimal "mixing" of the elements. Furthermore, any linear relationship describing the operation involves at least 5 bytes from input and output. The same is also true of the inverse function $\texttt{MC}^{-1}$.

4.3 The Key Addition Layer

This final layer involves a simple XOR of a *Round Key* with the state. A round key is a 128-bit "sub-key" derived from the cipher key through a process called *Key Expansion*. Let N_k denote the key length divided by 32. The cipher key, therefore, is made up of N_k words.

4.3.1 Key Expansion

As was mentioned at the beginning of §4.2, the round structure of the AES consists of an initial key addition step, followed by a series of iterated rounds. Each of these rounds involve a key addition step in which a round key is XORed to the state. Let N_r denote the number of rounds. There are therefore $N_r + 1$ round keys required; a total of $4 \cdot (N_r + 1)$ words.

The key expansion method involves an operation on words in which the word $\{a_0, a_1, a_2, a_3\}$ is 'left-shifted' to form the word $\{a_1, a_2, a_3, a_0\}$. We will refer to this operation as `RotWord`. Define also, the 'round constant' word array

$$\texttt{Rcon}[i] := \{x^{i-1}, \texttt{00}, \texttt{00}, \texttt{00}\} \qquad \text{for } i > 0,$$

where x represents the byte `02` and is multiplied modulo our irreducible polynomial $m(x)$. Furthermore, let $w[i]$ be the word in position i of the expanded key, where $0 \leq i \leq 4 \cdot (N_r + 1) - 1$. The method of constructing the expanded key is as follows:

For $N_k = 4, 6$: The cipher key forms the first N_k words, $w[0], \ldots w[N_k - 1]$, of the expanded key. We now define the remaining words.
For $N_k \leq i \leq 4 \cdot (N_r + 1) - 1$, define:

$$w[i] := w[i-1] \oplus w[i - N_k] \qquad \text{for } i \bmod N_k \neq 0, \tag{4.3}$$

and if $i \bmod N_k = 0$,

$$w[i] := \texttt{RotWord}\Big(\texttt{BS}\big(w[i-1]\big)\Big) \oplus \texttt{Rcon}[i] \oplus w[i - N_k].$$

So any word not in a position of a multiple of N_k is simply defined as the XOR of the previous word and the word N_k positions earlier. For a word $w[i]$ where i is a multiple of N_k, the operations `RotWord` and `BS` are performed on $w[i-1]$ and the constants $\texttt{Rcon}[i]$ and $w[i - N_k]$ are subsequently XORed to the result.

For $N_k = 8$: The method is similar to that of $N_k \leq 6$, but with the extra condition that

$$w[i] := \texttt{BS}\big(w[i-1]\big) \oplus w[i - N_k] \qquad \text{if } i \bmod N_k = 4;$$

that is, if $i - 4$ is a multiple of N_k then the operation `BS` is applied to $w[i-1]$ before XORing with $w[i - N_k]$.

Note. It is clear that any word $w[i]$ can be found if we know $w[i-1]$ and $w[i - N_k]$ of the expanded key. Similarly, working backwards and using $w[i]$ and $w[i-1]$, we can calculate $w[i - N_k]$.

4.3.2 The Key Schedule

Once the expanded key has been calculated from the cipher key, it is partitioned into blocks, $K^0, \ldots, K^{N_r}$, each of 4 words. Each block is arranged into columns of these words to form a 4×4 $\mathcal{F}$-matrix. The round keys $K^0, \ldots, K^{N_r}$ then form the basis for the key addition step:

- The initial key addition step involves byte-wise XORing the input state with K^0.
- The key addition layer in round i involves XORing the state with the block K^i.

This process is known as the *Key Schedule*. Denote the key addition operation by `KA` (KeyAddition).[4] Since the XOR operation has order 2, `KA` is its own inverse.

4.4 The Cipher and Inverse Cipher

The key addition step and the inner round structure have been discussed in the previous sections. The final round of the cipher has the same form as an inner round with the exception that the operation `MC` is omitted. This does not affect the security of the AES in any way [19]. One application of the AES on a block of plaintext can be described by the following schematic:

$$\begin{tikzcd}[row sep=small]
\boxed{\text{Plaintext}} \arrow[d, Rightarrow] & \leftarrow \text{Input} \\
\mathtt{KA} \arrow[d, Rightarrow] & \leftarrow \text{Key addition step} \\
\mathtt{KA} \circ \mathtt{MC} \circ \mathtt{SR} \circ \mathtt{BS} \arrow[d, Rightarrow] & \leftarrow \text{Round } 1 \\
\vdots \arrow[d, Rightarrow] & \\
\mathtt{KA} \circ \mathtt{MC} \circ \mathtt{SR} \circ \mathtt{BS} \arrow[d, Rightarrow] & \leftarrow \text{Round } N_r - 1 \\
\mathtt{KA} \circ \mathtt{SR} \circ \mathtt{BS} \arrow[d, Rightarrow] & \leftarrow \text{Round } N_r \\
\boxed{\text{Ciphertext}} & \leftarrow \text{Output}
\end{tikzcd}$$

To decrypt the ciphertext, the inverse cipher must be used. This is simply defined as the AES applied in the reverse direction.[5] An inner round of the inverse cipher is therefore represented by the sequence:

$$\mathtt{BS}^{-1} \circ \mathtt{SR}^{-1} \circ \mathtt{MC}^{-1} \circ \mathtt{KA}.$$

[4] The AES specification [1] denotes this function as AddRoundKey.

[5] [19] also describes another, equivalent inverse cipher, which is not discussed here.

4.5 Some Notation on Bytes

For the purpose of ease of description of attacks on the AES, we will use the notation of [23] in describing the position of bytes within the cipher. This notation is as follows:

$a_{i,j}^{(r)}$	The byte at position (i, j) at the beginning of round r.
$b_{i,j}^{(r)}$	The byte at position (i, j) at the output of the KeyAddition operation in round r.
$K_{i,j}^{r}$	The byte at position (i, j) of the round key of round r.
$m_{i,j}^{(r)}$	The byte at position (i, j) at the output of the `MC` operation in round r.
$s_{i,j}^{(r)}$	The byte at position (i, j) at the output of the `BS` operation in round r.
$t_{i,j}^{(r)}$	The byte at position (i, j) at the output of the `SR` operation in round r.

Chapter 5

AES Against Classic Attacks

There is an abundance of attacks that have been developed for breaking block ciphers. Several are specific to families of ciphers with similar characteristics, whilst others have been developed with the most generic block cipher structure in mind. Our focus is on academic attacks on the AES, rather than attacks on physical implementations of the algorithm (known as *side-channel attacks*), which depend significantly on the implementation itself. In this chapter, we give a brief overview of the resistance of the AES against several classic attacks.

5.1 Differential Cryptanalysis

Differential Cryptanalysis (*DC*) is a chosen plaintext attack, first described by Biham and Shamir [6] as a method for attacking DES in 1990. In this attack, we analyse the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs, in order to find the cipher key. The usual approach is to find high probability difference pairs for the active S-boxes (i.e., the S-boxes used in the attack). We do this by calculating the probability that the input difference $\Delta X = X \oplus X'$, for inputs X and X', will result in the output difference $\Delta Y = Y \oplus Y'$, for outputs Y and Y'. This done for all pairs $(\Delta X, \Delta Y)$.

A *differential trail* is formed by composing active S-box differences such that the output difference from one round corresponds to the input difference of the next round. A *differential characteristic* is then constructed for the entire cipher by composing all differential trails with the given input and output difference. A higher differential characteristic probability corresponds to a higher probability of recovering the cipher key.

For Rijndael, it is proven that there are no 4-round differential trails with probability above 2^{-150}, and no 8-round differential trails with probability above 2^{-300} [19]. This is sufficient to provide resistance against DC.

5.2 Linear Cryptanalysis

The method of *Linear Cryptanalysis* (*LC*) was first described by Matsui [35] in 1993, as a method for breaking DES. This is a known plaintext attack that takes advantage of probabilistic linear relationships between the input and output of a cipher, to find the cipher key. This is usually accomplished by approximating the active S-boxes by linear expressions that hold with high probability bias (i.e., large deviation from 1/2), and then combining them to form an approximation of the entire cipher involving only plaintext, ciphertext and key bits. This approximation must also have a high bias for the attack to work and uncover the key bits.

For Rijndael, it is proven that there are no 4-round linear trails with a bias above 2^{-75}, and no 8-round linear trails with a bias above 2^{-150} [19]. This is sufficient to provide resistance against this attack. It is clear from the above that the Wide Trail Design strategy [20], used in the design of Rijndael, achieves its stated goals of providing resistance against both LC and DC.

5.3 Truncated Differentials

Since their publication, linear and differential attacks have been extended in several ways. One such extension is known as the *Truncated Differential Attack*, introduced by Knudsen [31] in 1994. This attack exploits the fact that in some ciphers, differential trails tend to *cluster*. Clustering takes place if for certain pairs of plaintext and ciphertext differences, the number of differential trails is exceedingly large. Ciphers which act on well aligned blocks, such as the AES (byte-oriented operations, rather than bit-oriented) are more susceptible to this attack, even if proven secure against LC/DC.

Truncated differentials were already taken into account in the design of Rijndael [21]. Currently, the best known attack using this property is on AES-192 reduced to 6 rounds [28].

5.4 Boomerang Attacks

In some constructions, it is possible that there are no good difference patterns that propagate through the whole cipher, but there are some highly probable patterns that propagate half-way. When this happens, the *Boomerang Attack* [51] can be applied. The adversary essentially propagates differences from both "ends" of the cipher and finds which differences agree in the middle. This type of attack is therefore closely related to what are known as *meet-in-the-middle* attacks.

Due to the very low differential probabilities (i.e., 2^{-150} for 4 rounds) and highly effective diffusion layer, the AES provides adequate security against boomerang attacks. Recently, the boomerang attack has been used to break 5 and 6-round reduced AES-128 [7].

5.5 Impossible Differentials

This attack is a variant on the truncated differential attack in which a differential predicts that particular differences should *not* occur (i.e., that their probability is exactly zero). We call these differentials *Impossible Differentials* [5]. If a pair of ciphertexts is decrypted to one such difference under a guessed key, then the key choice is obviously incorrect. The correct key is found by eliminating all the other keys that lead to contradictions. This type of approach is typically known as a *miss-in-the-middle* attack.

An impossible differential attack on AES-128 reduced to 6 rounds [11], and more recently, on AES-192 reduced to 8 rounds [28] using encryptions under *related keys* (i.e., a collection of cipher keys with known differences), have been found to be faster than exhaustive search.

5.6 Interpolation Attacks

An *Interpolation Attack* [29] consists of forming polynomials from plaintext and ciphertext pairs. If the entire cipher is expressed as a polynomial with small degree, then the (key-dependent) coefficients can be solved for using only a few plaintext/ciphertext pairs. The attack is effective against ciphers in which the components themselves have simple polynomial expressions (i.e., are of low degree).

The inversion $x \mapsto x^{(-1)}$ over $\mathcal{F}$, combined with the affine transformation of the S-box has a very high-degree algebraic representation (degree 254). Combined with the effect of the diffusion layers, this complex expression prohibits interpolation attacks beyond a few rounds [19].

5.7 Slide Attacks

Slide attacks, first introduced by Biryukov and Wagner [9], exploit the degree of self-similarity of a block cipher. If we view the cipher as a product of identical round functions, $F(x, k)$, under key k, and consider a pair of plaintexts $(P, C), (P', C')$ such that $F(P, k) = P'$, then by self-similarity of the rounds and key schedule, we have also $F(C, k) = C'$, for the corresponding ciphertexts. This pair is known as a *slid pair*. By the birthday paradox (§2.3), it is possible to find a slid pair in $2^{\frac{n}{2}}$ known texts with high probability. If the round function F is considered "weak", i.e., given two equations $F(x_1, k) = y_1$ and $F(x_2, k) = y_2$, it is easy to deduce k, then we can recover the key with one slid pair.

Due to its strong key scheduling and different constants in each round, the AES exhibits enough dissimilarity between rounds to provide resistance against slide attacks.

Chapter 6

Square-6 – A Multiset Attack

Rijndael was based on a cipher named Square [18], developed by the same authors with Lars Knudsen. For details on the Square cipher itself, we refer the reader to [18]. The main motivation for the structure of Square was resistance against linear and differential cryptanalysis. However, a 'dedicated' chosen plaintext attack (described in the same paper as the cipher itself), which exploits the rather unique structure of Square, was soon found by Knudsen. This four-round attack, known as the *Square Attack*, can be extended naturally to the AES. The Square attack can be increased to 6 rounds and is referred to as the *Square-6 Attack* in this paper. Several extensions and variations on Square-6 have been found, and collectively this class of attack is known as a *Multiset Attack*. Other proposed names include 'Saturation attack', 'Structural attack', and 'Integral cryptanalysis'.

A *multiset* differs from the normal notion of a set by the fact that it allows the same value to appear multiple times. An element of a multiset is therefore a pair (*value*,*multiplicity*). In a multiset attack, the adversary carefully chooses multisets of plaintexts and studies their propagation through the cipher. While the element values obviously change, other properties such as multiplicity or "integral" (i.e., sum of all components) can remain unchanged, allowing cryptanalysis. Multiset attacks are currently thought to be the best known attacks against the AES [8, 21].

6.1 Preliminaries

In order to describe the attacks we first need the notion of a "Λ-set". Let a Λ-set be a collection of 256 states (4×4 $\mathcal{F}$-matrices) which are identical, except in one or more common array positions where the value of the byte in each of these positions is distinct for every element in the set. The bytes that vary over the Λ-set will be called *active* bytes and those that do not, *passive* bytes, i.e.,

$$\forall\, x, y \in \Lambda : \begin{cases} x_{i,j} \neq y_{i,j} & \text{if } (i,j) \text{ active} \\ x_{i,j} = y_{i,j} & \text{otherwise, i.e., if } (i,j) \text{ passive,} \end{cases}$$

where $x_{i,j}$ denotes the byte in the $(i,j)^{\text{th}}$ position of the state x. Furthermore, let Λ^n denote a Λ-set with n active bytes.

Let P^0 be an input Λ^1-set to the first round of the attack, and subsequently let P^i be the set of 256 output states from round i. It will become advantageous to use an 'intermediate' set of states, Q^i, defined by

$$Q^i := \{y \mid y = \mathtt{SR}^{-1}\bigl(\mathtt{MC}^{-1}(x)\bigr),\ \forall\, x \in P^i\}.$$

Note that Q^i is viewed as a set of states "between" P^{i-1} and P^i. Obviously, Q^i and P^i are trivially related, and so knowledge of one translates to knowledge of the other. In the same setting, we will define the "pseudo-round key"

$$L^i := \mathtt{SR}^{-1}\bigl(\mathtt{MC}^{-1}(K^i)\bigr), \tag{6.1}$$

where K^i is the round key of round i. Note that knowing L^i is equivalent to knowing K^i, and furthermore, knowing a column of K^i is equivalent to knowing 4 bytes of L^i.

Let K^i_j and L^i_j denote the j^{th} columns of the keys K^i and L^i, respectively.

For simplicity we assume that the final round is also an inner round. As we have seen in §4.2, this has no effect on the security of the cipher. If necessary, however, this attack can be adapted to take into account the final round structure.

6.2 The Square Attack

Consider, firstly, a chosen plaintext set P^0 being processed by 3 inner rounds. We will discuss the propagation of the set of plaintexts through the cipher round by round, and show that we are able to distinguish between a 3-round encryption and a random permutation. We will then show how this makes it possible to uncover the fourth round key.

6.2.1 Λ-set Propagation

Firstly note that the operations `BS` and `KA` do not affect the position of the active bytes since they both act on the state element-wise. For simplicity, we will assume that the active byte is in position $(0,0)$ (although any active byte position chosen is valid for this attack).

First Round: `SR` does not change the position of the active byte. `MC` then, due to the maximal Branch Number, produces an output Λ^4-set with the entire

first column of active bytes.
Second Round: `SR` permutes each row such that there is one active byte in each column. The subsequent application of `MC` converts this to four columns of active bytes (a Λ^{16}-set).
Third Round: Every byte remains active after the operation `SR`. However, when `MC` is applied, the result is not necessarily a Λ-set.

This 3-round process can be illustrated by the following diagram where the shaded boxes represent the active bytes of Λ.

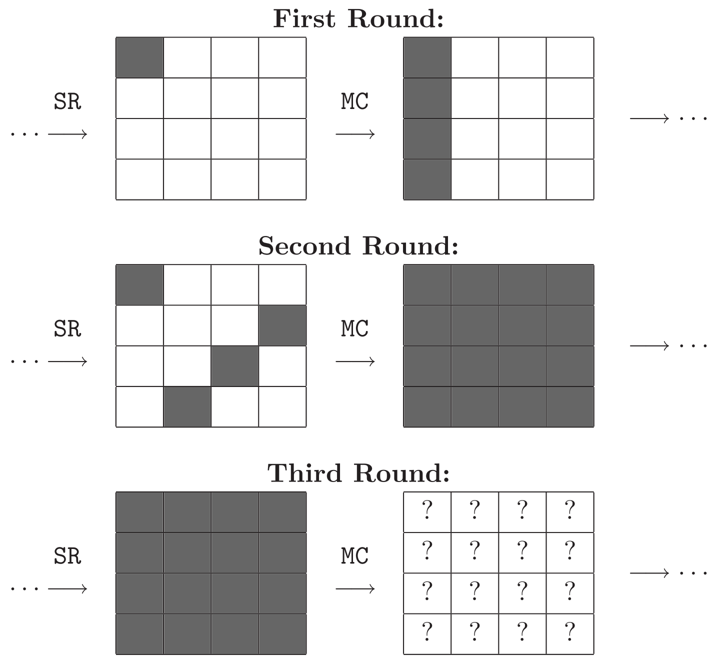

Figure 6.1: Λ-set propagation through three rounds.

It is useful here to note that if $y_{i,j}$ represents an active byte of the state $y \in \Lambda$, then from Equation (2.2), we get the relation:

$$\bigoplus_{y \in \Lambda} y_{i,j} = \bigoplus_{x \in \mathcal{F}} x = \mathtt{00} \equiv 0. \tag{6.2}$$

Let $t^{(3)} \in \Lambda^{16}$ be an input state to `MC` of the 3rd round of the attack (i.e., every byte is active), and let $m^{(3)}$ represent the output state, then from Equation (4.2) we get, for all (i, j):

$$\begin{aligned} m_{i,j}^{(3)} &= \mathtt{02} \cdot t_{i,j}^{(3)} \oplus \mathtt{03} \cdot t_{i+1,j}^{(3)} \oplus \mathtt{01} \cdot t_{i+2,j}^{(3)} \oplus \mathtt{01} \cdot t_{i+3,j}^{(3)} \\ &\equiv 2t_{i,j}^{(3)} + 3t_{i+1,j}^{(3)} + t_{i+2,j}^{(3)} + t_{i+3,j}^{(3)}, \end{aligned}$$

and therefore, using Equation (6.2),

$$\begin{aligned}
\bigoplus_{\substack{m^{(3)}=\mathtt{MC}(t^{(3)})\\ t^{(3)}\in\Lambda^{16}}} m^{(3)}_{i,j} &= \bigoplus_{t^{(3)}\in\Lambda^{16}} \left(2t^{(3)}_{i,j} + 3t^{(3)}_{i+1,j} + t^{(3)}_{i+2,j} + t^{(3)}_{i+3,j}\right)\\
&= 2\bigoplus_{t^{(3)}} t^{(3)}_{i,j} + 3\bigoplus_{t^{(3)}} t^{(3)}_{i+1,j} + \bigoplus_{t^{(3)}} t^{(3)}_{i+2,j} + \bigoplus_{t^{(3)}} t^{(3)}_{i+3,j}\\
&= 0+0+0+0\\
&= 0. \qquad (6.3)
\end{aligned}$$

Satisfying this property, every byte position of $a^{(4)} \in P^3$ (the set of input states to the 4th round) is said to be *balanced.* This property clearly distinguishes a 3-round encryption from a random permutation. Note, however, that the values of the bytes are unknown due to the addition of round keys.

The operation $\oplus$ is implied above by '+'. We will use this notation for the remainder of this paper whenever it is clear from the context that $\oplus$ is to be performed.

6.2.2 The Fourth Round

An important feature in using the 'alternative' values L^i and Q^i as defined in §6.1 is that the cipher has another, functionally equivalent, round structure. The comparison is as follows:

Standard Round Structure:	Alternative Round Structure:
$\mathtt{KA} \circ \mathtt{MC} \circ \mathtt{SR} \circ \mathtt{BS}$	$\mathtt{MC} \circ \mathtt{SR} \circ \mathtt{KA*} \circ \mathtt{BS}$

where KA* denotes XORing the pseudo-round key L^i with the state (as distinguished from KA which deals with the round key K^i) and the other operations are as previously described.

Using this alternative representation for the round structure of the AES, it is clear that an element $b^{(4)} \in Q^4$ can be expressed as a function of the last round input state $a^{(4)} \in P^3$ and the pseudo-round key L^4 by

$$b^{(4)}_{i,j} = \mathtt{BS}(a^{(4)}_{i,j}) + L^4_{i,j},$$

or equivalently

$$a^{(4)}_{i,j} = \mathtt{BS}^{-1}(b^{(4)}_{i,j} + L^4_{i,j}).$$

Therefore, by choosing a value of $L^4_{i,j}$, one can deduce the value $a^{(4)}_{i,j}$ from the corresponding intermediate $b^{(4)}_{i,j}$ "ciphertext" values. From above, the byte $a^{(4)}_{i,j}$ is expected to be balanced if the choice of $L^4_{i,j}$ is correct, and therefore any value of $L^4_{i,j}$ resulting in a non-balanced input byte must be incorrect.

The 4-round attack can essentially be described by the following algorithm:

```
1. Encrypt the $2^8$ plaintexts of $P^0$ with 4-round AES.
2. For all $(i,j) \in \{0,1,2,3\}^2$:
    3. For all $c \in \{0,1\}^8$:
        4. Calculate $s(c) := \bigoplus_{b^{(4)} \in Q^4} \mathtt{BS}^{-1}(b^{(4)}_{i,j} + c)$.
        5. If $s(c) \neq 0$ then $L^4_{i,j} \neq c$.
```

Moreover, it is expected that an incorrect key value will produce a balanced result with probability 2^{-8}, and therefore this process will generally provide 2 candidate key values – the correct byte and an incorrect one [32]. The process can be repeated for the other 15 bytes of the pseudo-round key L^4, so as to reveal a collection of (usually) less than 2^{16} candidates for the (equivalent) round key K^4.

The correct round key can be found with very high probability by either applying this attack again with a new 2^8-set of plaintexts, or by simple exhaustive search over the key candidates. Of course the first method uses twice as many plaintexts as the second. Nevertheless, either approach uniquely determines the round key with negligible memory requirements and a work factor of only about $2^4 \cdot 2^8 \cdot 2^8 = 2^{20}$ byte-wise XOR operations and around 2^9 cipher executions. This basic method is known as the 4-round *Square Attack*.

Note. This 4-round attack is performed in order to uncover four words of the expanded key (a single round key). For AES-128, this allows us to run the key schedule backwards to uniquely determine the cipher key. For AES-192 and AES-256, however, we require 6 and 8 words respectively, and so this attack is not sufficient for those key lengths.

6.3 The Square-6 Attack

The Square attack can be increased to 6 rounds through extension by an additional round at the end, and at the beginning. We will call this the *Square-6 Attack*. The Square-6 attack allows us to uncover two round keys, thus providing us with enough key information to uniquely determine the cipher key for any key length. We begin with a fifth-round extension of the basic four-round attack.

6.3.1 5th Round Extension

Given P^0 as above, we can compute Q^5 from the output, P^5, of a 5th round. Values for P^3 can be calculated by assuming values of 5 bytes of the pseudo-round keys L^5 and L^4 and then decrypting $1\frac{1}{2}$ rounds using Q^5.

Example. In tracking, say, the byte $a^{(4)}_{0,0}$ propagating forward through these $1\frac{1}{2}$ rounds, BS is first applied, then KA* adds the byte $L^4_{0,0}$. SR has no effect whilst MC diffuses the byte to the entire first column, ending the 4th round. The beginning of the 5th round provides another byte substitution and KA* is applied again, this time adding the first column of key values of L^5. The 5 key values, $L^4_{0,0}$ and $L^5_0 = (L^5_{0,0}, L^5_{1,0}, L^5_{2,0}, L^5_{3,0})$, are therefore required to find $a^{(4)}_{0,0}$.

The $1\frac{1}{2}$ round encryption is described by the function:

$$\texttt{KA*} \circ \texttt{BS} \circ \texttt{MC} \circ \texttt{SR} \circ \texttt{KA*} \circ \texttt{BS}.$$

As in the four-round attack, if the bytes of P^3 are not balanced then at least one of the key values is wrong. Again, it is expected that 1 in 2^8 incorrect quintuples of key values will provide a false-positive result [32]. The round keys K^4 and K^5 can be recovered by exhaustive key search using 5 Λ^1-sets of plaintexts.

In order to measure the time complexity of this attack, we will define a 'basic operation', BO, by:

1. For $i = 0$ to 3: $V_i := \texttt{BS}^{-1}(b^{(r)}_{i,j} + K^r_{i,j})$
2. $W := \mu^{-1}(V)$
3. $BO(b^{(r)}_j, K^r_j, K^{r-1}_{k,j}) := \texttt{BS}^{-1}(W_k + K^{r-1}_{k,j})$,

where μ represents the column-wise mixing action of MC. One AES encryption can be approximated by 2^8 of these basic operations [23] and thus we will use 2^8 BO's (encryptions) as our unit of time complexity.

To check the correctness of a quintuple of key values, we have to do 2^8 basic operations followed by XORing the results to verify they satisfy Equation (6.3). This is done for every possible collection of key values and so the 5-round attack requires the time of about $(2^8)^5 \cdot 2^8 = 2^{48}$ basic operations and hence has a time complexity of 2^{40}.

6.3.2 0th Round Extension

The 0th round extension involves choosing a plaintext set P^{-1} that will result in P^0 being a Λ^1-set. To find candidates for P^{-1}, the adversary could begin a Λ^1-set with, for example, the active byte in position $(0, 0)$, and perform a single round decryption. This results in a Λ^4-set with active bytes in positions $(0, 0)$, $(1, 1)$, $(2, 2)$ and $(3, 3)$ as determined by the $\texttt{MC}^{-1}$ and subsequent $\texttt{SR}^{-1}$ operations on the active P^0-byte.

Consider an arbitrary collection of 2^{32} plaintexts such that the 4 bytes corresponding to these active bytes range over all possible values and the remaining 12 passive bytes are fixed over the set. From this set, we can choose many candidates for P^{-1} (i.e., choose sets of 256 plaintexts that result in P^0 being a Λ^1-set) and assume values of the key bytes $K^{-1}_{0,0}$, $K^{-1}_{1,1}$, $K^{-1}_{2,2}$ and $K^{-1}_{3,3}$ required to calculate the corresponding P^0 sets. If the chosen key values are incorrect, then when these P^0 sets are input to the 5-round attack, the last round key calculated for each set will be inconsistent. The correct combination of key values results in the correct round key being recovered. It is estimated that we require at least 10 P^{-1} sets for this attack to be successful [23].

For each of the 2^{32} chosen key bytes, we have to perform the above 5-round attack and so the Square-6 attack has a running time of approximately $2^{40} \cdot 2^{32} = 2^{72}$. We also have the memory requirement of storing the 2^{32} ciphertexts. Essentially, the Square-6 attack requires us to guess 9 key bytes in order to recover the round keys K^4 and K^5.

Note. We only use the alternative round structure in the final 2 rounds of the Square-6 attack.

Chapter 7

Basic Seven-Round Variants

At the Third AES Candidate Conference (AES3) in April 2000, two independent extensions of the Square-6 attack, to seven rounds of the AES, were presented. The first of these, presented by Lucks [32], is a simple extension facilitated by guessing a further round key. Lucks refers to this attack as a *Saturation Attack*. The second attack, by Gilbert and Minier [27], is a variation on the Square-6 attack that makes use of a phenomenon known as a "collision" between induced functions within the cipher. This attack is therefore termed a *Collision Attack*.

7.1 The Saturation Attack

In performing the 7-round extension of Square-6 as described by Lucks, the adversary begins by choosing 2^{32} plaintexts suitable for the Square-6 attack, encrypts them with 7 rounds of the AES (rounds 1–7, say), and for every choice of $K^7 \in \mathbb{Z}_2^{128}$, decrypts the 2^{32} resulting ciphertexts by one round. Performing the Square-6 attack on the original plaintexts will recover the round keys K^5, K^6 and furthermore, K^7, by comparing the 6-round encryption with the 7^{th}-round decryption. This gives us 3 round keys, more than enough to deduce the cipher key and verify the result.

The 7-round attack requires the same number of plaintexts as Square-6, but the number of operations performed increases by 2^{128} and so we have a time complexity of $2^{72} \cdot 2^{128} = 2^{200}$. This means, therefore, that this attack is slower than exhaustive search for both AES-128 and AES-192, making it unsuitable for those versions.

7.1.1 Attacking 7 Rounds of AES-256

Although the generic 7-round attack is faster than exhaustive key search for AES-256, the key expansion method of this version can be exploited to facilitate a further improvement by a factor of 2^8.

As was noted in §4.3.1, knowledge of the words $w[i]$ and $w[i-1]$ gives the word $w[i - N_k]$ of the expanded key. Once we have uncovered the round key K^7, we know 4 consecutive words of the expanded key, namely $w[28], \ldots, w[31]$. Using Equation (4.3), we find three more words:

$$\begin{aligned} w[21] &= w[28] + w[29], \\ w[22] &= w[29] + w[30], \text{ and} \\ w[23] &= w[30] + w[31]. \end{aligned}$$

Therefore we know the 3 columns, K^5_1, K^5_2 and K^5_3. This equates to knowing 12 bytes of L^5, in particular the byte $L^5_{0,1}$. As in the Square-6 attack, in order to check the bytes of P^4 are balanced at, say, position $(0,1)$, we need the bytes in column 1 of Q^6, and the key bytes of L^6_1 and $L^5_{0,1}$. The seven round attack on AES-256 is performed as follows:

```
1. Choose $2^{32}$ plaintexts suitable for the Square-6 attack (with active bytes in
   positions $(0,0),(1,1),(2,2)$ and $(3,3)$) and encrypt them with 7 rounds of
   the AES.
2. For all $2^{32}$ possible combinations of $K^0_{0,0}, K^0_{1,1}, K^0_{2,2}$ and $K^0_{3,3}$:
   3. Choose 32 distinct $2^8$-sets of plaintexts, $P^0[i]$ for $i = 0, \ldots, 31$, such that
      $P^1[i]$ is a $\Lambda^1$-set.
   4. For all $2^{128}$ possible choices for $K^7$:
      5. Decrypt the 32 ciphertext sets $P^7[i]$ by one round to get $P^6[i]$ and
         thus $Q^6[i]$.
      6. Calculate the pseudo-round key byte $L^5_{0,1}$.
      7. For all $2^{32}$ possible combinations of the bytes $L^6_{0,1}, L^6_{1,1}, L^6_{2,1}$ and $L^6_{3,1}$:
         8. For $i = 0, \ldots, 31$: Compute
               $$y[i] := \bigoplus_{b^{(6)} \in Q^6[i]} BO(b^{(6)}_1, L^6_1, L^5_{0,1}).$$
            if $y[i] \neq 0$ then stop iterating and try the next combination.
         9. If $y[i] = 0$ for all $i = 0, \ldots, 31$ then the key bytes are correct and
            the attack is successful.
```

This algorithm essentially performs exhaustive search over a subspace of size $2^{32} \cdot 2^{128} \cdot 2^{32} = 2^{192}$ of the expanded key space. As was the case in §6.3.1, if any combination of key bytes is incorrect then $y[i] = 0$ will hold with probability 2^{-8} for each i. The probability that $y[i] = 0$ for all $i = 0, \ldots, 31$, for a particular set of incorrect key bytes, therefore will be $(2^{-8})^{32} = 2^{-256}$. Since we are only considering 2^{192} combinations in total, the probability that this algorithm will recover an incorrect collection of key bytes is less than $2^{192} \cdot 2^{-256} = 2^{-64}$ which is certainly negligible.

Once completed, the algorithm uncovers the round key K^7 and the 2nd column of K^6. The other 12 bytes of K^6 can be found easily by exhaustive search, and the 2 complete round keys can then be used to determine uniquely, the entire expanded key.

Similar to Square-6, we require 2^{32} plaintexts and the storage of 2^{32} ciphertexts for this attack. The time requirement is calculated as follows: Step 2 loops 2^{32} times, Step 4 loops 2^{128} times, Step 7 loops 2^{32} times, Step 8 is iterated $1 + 2^{-8} + 2^{-16} + \ldots \approx 1$ time and requires 2^8 basic operations per iteration. Therefore the algorithm performs approximately $2^{32} \cdot 2^{128} \cdot 2^{32} \cdot 1 \cdot 2^8 = 2^{200}$ basic operations and thus has a running time of 2^{192}; certainly an improvement on the generic 7-round attack, and much faster than brute force.

7.1.2 Attacking 7 Rounds of AES-192

It is clear that an acceleration of 2^8 on the generic attack will not suffice to break AES-192 faster than brute force, but as in the case of AES-256, there is a weakness in the key expansion method that in fact allows improvement by a factor of 2^{24} to the generic case.

The three further words that the known K^7 provides in the case of AES-192 are $w[23], w[24]$ and $w[25]$. They correspond to the columns K_3^5, K_0^6 and K_1^6, respectively. The key bytes $L_{0,3}^5, L_{1,3}^6$ and $L_{2,3}^6$, for example, can therefore be calculated (from Equation (6.1)) and so we need to find $L_{0,3}^6$ and $L_{3,3}^6$ by mounting a similar 7-round attack on AES-192.

The difference between the attack on AES-256 and this attack on AES-192 is that we have 3 necessary key bytes and thus only need to guess 2 further key values rather than 4. Step 7 of the above algorithm is therefore performed 2^{16} times instead of 2^{32} times. So we have a time complexity of $2^{192-16} = 2^{176}$, which is certainly faster than exhaustive key search.

7.2 A Collision Attack on the AES

The main premise behind the Square-6 attack (and extensions) is the 3-round distinguisher between encryption and a random permutation. Gilbert and Minier [27] discovered an extension of this, to a 4-round distinguisher, by exploiting the existence of *internal collisions* between some partial functions induced by the cipher. A collision within a function f is one in which we can find two inputs, x_1 and x_2, such that $x_1 \neq x_2$, and $f(x_1) = f(x_2)$.

The term “internal collision” is used because, in general, the collision will not propagate to the output of the cipher. We assume that internal collisions for particular plaintext (resp. ciphertext) encryptions (resp. decryptions) are somehow correlated with the cipher key used. Finding such collisions forms the basis of the 4-round distinguisher, and the subsequent 7-round attack described in this section.

7.2.1 Notation

Let P^0 be the input 2^8-set to round 1 and let P^i be the output from round i, for $i \in \{1,2,3,4\}$. We will assign shorthand notation to some byte positions within the P^i that play a crucial role in this attack:

$$\begin{aligned} \text{Let } y &= P^0_{0,0} \text{ and } c_i = P^0_{i,0} \text{ for } i \in \{1,2,3\}; \\ p_i &= P^1_{i,0} \text{ for } i \in \{0,1,2,3\}; \\ q_i &= P^2_{i,4-i} \text{ for } i \in \{0,1,2,3\}; \\ r &= P^3_{0,0}; \text{ and} \\ z_i &= P^4_{i,0} \text{ for } i \in \{0,1,2,3\}. \end{aligned}$$

These are shown in Figure 7.1. Let P^0 be a Λ^1-set, with y as the active byte, such that we know the constant triple $c = (c_1, c_2, c_3)$ and the other byte positions are arbitrary constants. The remaining bytes p_i, q_i, r and z_i can be considered as c-dependent functions of y, i.e., $p_i = p_i^c[y], q_i = q_i^c[y], r = r^c[y]$ and $z_i = z_i^c[y]$.

7.2.2 A 4-Round Distinguisher

As we have seen, the 3-round distinguisher in the previous attacks is based on the following observations:

1. The bytes $p_0, \ldots, p_3$ are 1-1 functions of y and the remaining 12 bytes are constant;
2. The bytes $q_0, \ldots, q_3$ (and the other 12 bytes) are 1-1 functions of y; and
3. The byte r is the XOR of four 1-1 functions of y and therefore $\bigoplus_{y \in \mathcal{F}} r[y] = 0$.

These 3 properties are sufficient for 3 rounds but do not provide any information about multiset propagation through the 4th round. There are more detailed observations that can be made on each of these steps that involve the constant c and, as we will see, can determine a 4-round distinguisher. Beginning with the first round:

First Round: `SR` permutes rows 1–3 and, in doing so, shifts the constants c_i from the first column. `MC` then converts the first column into 4 active bytes, independent of c. For similar reasons, columns 1–3 are independent of y. The bytes p_i can thus be described by

$$p_i = a_{i,0} \cdot \mathtt{BS}(y) + K^1_{i,0},$$

and the other bytes of P^1 are of the form

$$p_{i,j} := a_{i,j} \cdot \mathtt{BS}(c_{4-j}) + K^1_{i,j},$$

for $i \in \{0,1,2,3\}$ and $j \in \{1,2,3\}$, where $a_{i,j}$ are constants related to the `MC` operation. Each p_i is therefore determined by a key-dependent byte, and each $p_{i,j}$ is determined by a c- and key-dependent byte.

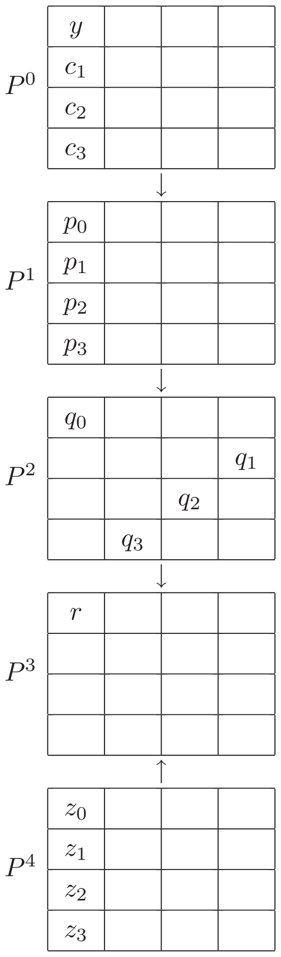

Figure 7.1: Four inner rounds of the AES.

Second Round: SR shifts the active bytes p_i to separate columns followed by the MC operation, creating a Λ^{16}-set. Each of the q_i's are therefore functions of both p_i and c. The transformation is as follows: There exist 16 MC coefficients $\alpha_i, \beta_i, \gamma_i$ and δ_i such that

$$q_i = \alpha_i \cdot \mathtt{BS}(p_i) + \beta_i \cdot \mathtt{BS}(p_{1+i,1}) + \gamma_i \cdot \mathtt{BS}(p_{2+i,2}) + \delta_i \cdot \mathtt{BS}(p_{3+i,3}) + K^2_{i,4-i},$$

for $i \in \{0, 1, 2, 3\}$. The q_i are therefore related to y and c by:

$$q^c_i[y] = \alpha_i \cdot \mathtt{BS}(a_{i,0} \cdot \mathtt{BS}(y) + K^1_{i,0}) + b_i,$$

where

$$\begin{aligned} b_i = {} & [\beta_i \cdot \mathtt{BS}(a_{1+i,1} \cdot \mathtt{BS}(c_3) + K^1_{1+i,1})] + [\gamma_i \cdot \mathtt{BS}(a_{2+i,2} \cdot \mathtt{BS}(c_2) + K^1_{2+i,2})] \\ & + [\delta_i \cdot \mathtt{BS}(a_{3+i,3} \cdot \mathtt{BS}(c_1) + K^1_{3+i,3})] + K^2_{i,4-i}. \end{aligned}$$

Each q_i is a function of y entirely determined by one key-dependent byte, $K^1_{i,0}$, and one c- and key-dependent byte, b_i.

Third Round: The r byte can be expressed as a similar function of the q_i and one key-dependent byte, $K^3_{0,0}$. Therefore, $r^c[y]$ is a function of y, determined by 4 unknown c- and key-dependent constants and 5 unknown key-dependent constants.

Fourth Round: Considering a fourth round 'decryption', we can express r as a function of the z_i and a key-dependent byte k, using Equations (2.5) and (2.7), as follows:

$$r = \mathtt{BS}^{-1}[(\mathtt{0e} \cdot z_0 + \mathtt{0b} \cdot z_1 + \mathtt{0d} \cdot z_2 + \mathtt{09} \cdot z_3) + k].$$

This means that $L^c := (\mathtt{0e} \cdot z^c_0 + \mathtt{0b} \cdot z^c_1 + \mathtt{0d} \cdot z^c_2 + \mathtt{09} \cdot z^c_3)$ is a 1-1 function of r, determined by k. Hence, the $z^c_i[y]$ are functions of y determined entirely by 6 key bytes and 4 c- and key-dependent bytes.

The above property of $z^c_i[y]$, for $i \in \{0, 1, 2, 3\}$, therefore determines an efficient 4-round distinguisher. We can view $r^c[y]$ as a (key-dependent) function of y, which depends only on the 4 c-dependent constants $b_0, \ldots, b_3$. Let us assume heuristically that the b_i act as random functions of c. Then, according to the birthday paradox (§2.3), given a set C of 2^{16} c values, there exist two values c' and c'' in C such that $r^{c'}[y] = r^{c''}[y]$ for all $y \in \mathcal{F}$, with non-negligible probability. From this, we can say that $L^{c'}[y] = L^{c''}[y]$ for all $y \in \mathcal{F}$ iff c' and c'' produce a collision.

Note. It usually suffices to use only, say, 16 values of y to determine (with significant probability) whether we have a collision. [27] found experimentally that these collisions do in fact occur, and for some key values, more than 256 such collisions were found.

We now introduce the 4-round distinguisher, which can be used to attack seven rounds of the AES:

1. Choose a set C of 2^{16} c values, and choose a set $\Gamma \subset \mathcal{F}$ of 16 y values, eg. if we view $\mathcal{F}$ as the set $\mathcal{F} = \{0, 1, \ldots, 255\}$ then we choose $\Gamma = \{0, \ldots, 15\}$, say.

2. For each $c \in C$, compute $L^c[y] = (\texttt{0e} \cdot z_0^c + \texttt{0b} \cdot z_1^c + \texttt{0d} \cdot z_2^c + \texttt{09} \cdot z_3^c)$, $\forall\, y \in \Gamma$. The authors of [27] claim that calculation of a set of 16 L^c values takes substantially less time than a single AES operation.

3. Check whether any two of these lists, $L^{c'}$ and $L^{c''}$, collide.

The above algorithm requires approximately 2^{20} chosen plaintexts, and less than 2^{20} AES encryptions [27].

7.2.3 Attacking 7 Rounds of AES-192 and AES-256

To extend this 4-round distinguisher to a 7-round attack, we need to set up a little more notation on bytes of the input states. Let y, c, r, z_i and P^i be defined as before, then let $x_i = P^{-1}_{i,i}$ for $i \in \{0, 1, 2, 3\}$, where P^{-1} is the input 2^8-set to round 0. Round 0 is considered to be an initial key addition followed by an inner round. Furthermore, let P^6 be the output from round 6, where round 6 is considered to be a final round (i.e., not including the `MC` operation).

We need to take a collection of 2^{32} plaintexts such that all bytes are constant, except for the values x_i which range over all possible combinations. If the four key bytes corresponding to the x_i are known, we can split the collection into 2^{24} 2^8-sets P^{-1}, corresponding to Λ^1-sets P^0, where the active byte is in position $(0, 0)$ and the constant triple c is unique for each Λ^1-set. All other 12 bytes are constant over all subsets. Note that this is the same procedure used in §6.3.2.

Note also that the bytes y and c of the corresponding P^0 sets are known up to unknown key values. Denote by $y*$ and $c*$ the known values, which differ from the actual y and c values by the fixed unknown key bytes. Let the 4 unknown key bytes be denoted k_{ini}.

Similar to the 'fourth round decryption' method above, we can express each byte $z_0, \ldots, z_3$ as a function of 4 bytes of P^6 and 5 key-dependent bytes. These key-dependent bytes consist of 4 key bytes from the final round and one linear combination of key bytes from the penultimate round.

We can view $L^c = (\texttt{0e} \cdot z_0^c + \texttt{0b} \cdot z_1^c + \texttt{0d} \cdot z_2^c + \texttt{09} \cdot z_3^c)$ as the XOR of two functions $\tau_1^c := (\texttt{0e} \cdot z_0^c + \texttt{0b} \cdot z_1^c)$ and $\tau_2^c := (\texttt{0d} \cdot z_2^c + \texttt{09} \cdot z_3^c)$, each of which are therefore functions of 8 ciphertext bytes and 10 unknown key bytes. Denote by k_{τ_1} and k_{τ_2}, these 10 key bytes associated with τ_1 and τ_2, respectively.

The seven round "collision" attack is then performed via the following algorithm:

```
1. Select a subset of size $2^{16}$ from the set of $2^{24}$ possible $P^{-1}$ sets.
2. For each of the $2^{32}$ possible combinations for $k_{\text{ini}}$:
  3. Choose 16 states from each $P^{-1}$ set corresponding to $y* \in \Gamma$ and
     encrypt them to get the ciphertexts.
  4. For all $2^{80}$ possible $k_{\tau_1}$ values:
    5. For all $2^{16}$ $c*$ values associated with each $k_{\text{ini}}$ combination, calculate
       from the ciphertexts, the 16-tuple $\tau_1^{c*}[y*]$ for $y* \in \Gamma$.
    6. For each $(c'*, c''*)$ pair of distinct $c*$ values, calculate
       $(\tau_1^{c'*}[y*] + \tau_1^{c''*}[y*])$ for $y* \in \Gamma$.
  7. For all $2^{80}$ possible $k_{\tau_2}$ values:
    8. For all $2^{16}$ $c*$ values, calculate from the ciphertexts, the 16-tuple
       $\tau_2^{c*}[y*]$ for $y* \in \Gamma$.
    9. For each $(c'*, c''*)$ pair of distinct $c*$ values, calculate
       $(\tau_2^{c'*}[y*] + \tau_2^{c''*}[y*])$ for $y* \in \Gamma$.
    10. If $(\tau_2^{c'*}[y*] + \tau_2^{c''*}[y*]) = (\tau_1^{c'*}[y*] + \tau_1^{c''*}[y*])$ for all $y* \in \Gamma$, then
        encrypt the remaining states from the $P^{-1}$ sets associated with $c'*$
        and $c''*$ and verify with the remaining $y*$ values. If the equality is
        verified, stop the algorithm. Otherwise, try the next $k_{\tau_2}$ value.
```

The equality between $(\tau_1^{c'*}[y*] + \tau_1^{c''*}[y*])$ and $(\tau_2^{c'*}[y*] + \tau_2^{c''*}[y*])$ indicates equality between $L^{c'*}[y*]$ and $L^{c''*}[y*]$ and thus a collision between the functions $r^{c'}[y]$ and $r^{c''}[y]$. This collision shows that information about the key is 'leaked' by the ciphertexts and therefore provides, with "overwhelming probability" [27], the key values $k_{\text{ini}}, k_{\tau_1}$ and k_{τ_2}. This equates to knowing the 4 initial bytes k_{ini}, 4 bytes of the second last round key and the entire final round key.

The authors of [27] claim that the probability of the procedure finding a collision and thus ending successfully is approximately 50%. The time complexity of the algorithm is about 2^{140} [27], making this a very effective attack on 7 rounds of either AES-192 or AES-256. However, the running time clearly exceeds that of exhaustive search on AES-128. An improvement to this attack dedicated to the 128-bit version is described in [27], where it is claimed that the procedure is "marginally faster" than brute force.

7.3 Summary

There is a generic seven-round extension of Square-6 that requires 2^{32} chosen plaintexts and has complexity 2^{200}. This running time can be improved to 2^{176} for AES-192 and 2^{192} for AES-256, but seemingly the smallest running time is still too large to break AES-128 faster than brute force.

A collision attack that can be mounted on AES-192 and AES-256 reduced to seven rounds has also been discussed here. It makes use of a 4-round distinguisher and requires 2^{32} chosen plaintexts, with a complexity of 2^{140}.

Chapter 8

Extending the Square-6 Attack

As we have seen, multiset attacks can be used to break AES reduced to 6 and 7 rounds. In this chapter we look at several improvements to the Square-6 attack, and show how this translates to an improved 7 round attack. We will show how we can further extend this to break 8 rounds of AES-192 and AES-256. Finally, we show that there is a "related-key" attack that can be mounted on AES-256 reduced to 9 rounds. These improvements and extensions to the basic Square-6 attack are attributable to the authors of [23]. Currently, the attacks introduced in this chapter are the best known cryptanalytic techniques established against the AES.

8.1 An Improvement of Square-6

Instead of guessing the four first-round key bytes in the Square-6 attack, we can simply use all 2^{32} plaintexts. We take a set of 2^{32} plaintexts suitable for the Square-6 attack and encrypt them. Guessing the 5 key bytes, denoted $k_0, \ldots, k_4$, at the end of the cipher, we can then partially decrypt the 2^{32} ciphertexts by $1\frac{1}{2}$ rounds to a single byte. We then sum over all 2^{32} values and check for a zero result.

Consider this partial decryption. From the example of §6.3.1, we use 4 bytes of the ciphertext Q^5 to find one byte of P^3. The function describing this $1\frac{1}{2}$ round decryption is:

$$\texttt{BS}^{-1} \circ \texttt{KA*} \circ \texttt{SR}^{-1} \circ \texttt{MC}^{-1} \circ \texttt{BS}^{-1} \circ \texttt{KA*}.$$

This is equivalent to one basic operation, BO. Let the 4 ciphertext bytes used in the partial decryption be denoted c_j for $j = \{0, 1, 2, 3\}$, and if we are considering the i^{th} ciphertext block in particular, denoted this $c_{i,j}$. Then we want to compute:

$$\bigoplus_i \mathtt{BS}^{-1}(S_0[c_{i,0}+k_0]+S_1[c_{i,1}+k_1]+S_2[c_{i,2}+k_2]+S_3[c_{i,3}+k_3]+k_4), \quad (8.1)$$

where $S_j = \varepsilon_j \cdot \mathtt{BS}^{-1}$ for suitable constants ε_j associated with the $\mathtt{MC}^{-1}$ operation. This XOR over 2^{32} ciphertexts is performed for all 2^{40} possible key guesses and so we sum 2^{72} values. This corresponds to the work of about 2^{64} encryptions, an improvement of 2^8 on the Square-6 attack. Furthermore, we are only required to guess 5 key bytes instead of 9. However, to uniquely identify the proper value for these 5 bytes, we need about $6 \cdot 2^{32}$ plaintexts [23].

The method described above can be performed much more efficiently, reducing the complexity by a further 2^{20}. For each m, define the "partial sum":

$$x_m := \bigoplus_{j=0}^{m} S_j[c_j + k_j].$$

This gives us a map

$$(c_0, c_1, c_2, c_3) \mapsto (x_m, c_{m+1}, \ldots, c_3),$$

that we can apply to each ciphertext block if we know $k_0, \ldots, k_m$.

The attack is performed as follows:

```
1. Begin with the $2^{32}$ ciphertexts.
2. For each of the $2^{16}$ possible combinations for $k_0$ and $k_1$:
   3. Compute a list of $2^{32}$ triples, $(x_1, c_2, c_3)_i$, where the triple
      $(x_1, c_2, c_3)_j := (x_{j,1}, c_{j,2}, c_{j,3})$ is calculated for the jth ciphertext block.
      Count how often each triple occurs in the list.
   4. For all $2^8$ possible $k_2$ values:
      5. Compute a list of $2^{32}$ tuples, $(x_2, c_3)_i$. Count how often each tuple
         occurs in the list.
      6. For each of the $2^8$ possible combinations for $k_3$:
         7. Compute a list of $2^{32}$ values, $x_{i,3}$. Count how often each value
            occurs in the list.
         8. For all $2^8$ possible values of $k_4$, compute the expression in
            Equation (8.1).
```

Note that there are 2^{24} possible combinations for the triple in Step 3 and so we need 2^{24} counters. Since we are taking sums modulo 2 (i.e., the XOR operation), it suffices to count modulo 2. Therefore, we can use a single bit for each counter, and so we only require 2^{24} bits of memory for the counters. The counters provide a lowered work factor by reducing the number of XORs required in the final step.

In the first stage we guess 2 key bytes and process 2^{32} ciphertexts for each guess, and so has complexity of approximately $2^{16} \cdot 2^{32} = 2^{48}$ S-box lookups. Next, we guess 1 more byte (a total of 3 key bytes) and process 2^{24} triples for each guess, which therefore also costs $2^{24} \cdot 2^{24} = 2^{48}$. Similarly, the last 2 stages (processing the tuples, and finally the $x_{i,3}$ values) have a work factor of 2^{48} each.

In total, for one structure of 2^{32} plaintexts, the computation requires the equivalent of about 2^{50} S-box lookups. However, again we require around $6 \cdot 2^{32}$ plaintexts and so the total number of S-box applications is approximately 2^{52}; a running time of about 2^{44}, using a rough equivalence of 2^8 S-box lookups to an encryption. This is certainly a very significant improvement on the 2^{72} work factor of Square-6.

8.2 The Saturation Attack Improved

We can apply this improved method to Lucks' [32] attack on 7 rounds of AES-256 and AES-192 (§7.1.1,§7.1.2).

For AES-192: For each of the 2^{128} possible combinations of K^7, we can reduce each structure of 2^{32} plaintext/ciphertext pairs to 2^{24} counters as above. We then guess one of the unknown bytes and reduce the partial sum to 2^{16} counters in around 2^{24} steps. Finally, we guess the last unknown byte and calculate the desired sum. Each of these three steps involve around 2^{160} S-box lookups for one structure. We need three structures to begin eliminating wrong key guesses for K^7 [23] and so we have a total of about 2^{163} lookups, corresponding to a running time of 2^{155}; an improvement of 2^{21} on the original attack. To uniquely identify the correct key bytes we require approximately $19 \cdot 2^{32}$ plaintexts [23].

For AES-256: Similar to AES-192, we initially process the 2^{32} ciphertexts for each of the 2^{128} key values K^7, using around 2^{160} S-box lookups. We then guess 2 of the unknown key bytes and calculate 2^{24} counter values for a cost of 2^{176} lookups. The remaining steps also cost 2^{176}, resulting in an overall cost of 2^{178} lookups or 2^{170} encryptions for each structure. Although we require about 21 structures to find the unique key bytes, the workload is dominated by the first 5 structures used to eliminate incorrect guesses for K^7 and therefore the overall complexity of this attack is approximately 2^{172} [23].

8.3 A Further Improvement

It is possible to improve this attack further to a complexity of 2^{120}. However with this decrease in time complexity comes the requirement of increasing the data complexity to 2^{128} chosen plaintexts, i.e., we use the entire codebook.

We divide the collection into 2^{96} packs of 2^{32} encryptions that vary in 4 bytes of $m^{(1)}$ (i.e., after the first `MC` operation; recall this notation from §4.5) suitable for the attack of §8.1; that is, each pack consists of 2^{24} sets of 2^8 encryptions that vary in a single byte of $m^{(2)}$. Essentially we have a collection of 2^{120} 2^8-sets that vary in a single byte of $m^{(2)}$. As in §6.2.2, when we sum a single byte of $a^{(6)}$ over each 2^8-set we will get a zero result, and therefore summing over all 2^{128} encryptions also yields a zero result. This property is the basis for the following attacks.

If we now fix a fifth byte of $m^{(1)}$ (different to the 4 bytes selected above), say $m^{(1)}_{a,b} = x$, then we will have a total of $2^{120-8} = 2^{112}$ sets of 2^8 encryptions that vary in a single byte of $m^{(2)}$. This structure of 2^{120} encryptions is called a *herd*.

Taking the collection of 2^{128} plaintexts and guessing 4 necessary key bytes, we can calculate $m^{(1)}_{a,b}$ for each encryption and separate the collection into herds. We then guess five bytes at the end to calculate a byte in $a^{(6)}$. If the key bytes were correct, summing the byte of $a^{(6)}$ over each herd will have zero result. Moreover, this result is unlikely to occur if the guessed bytes are incorrect [23]. This method essentially uses the 6 round attack of §8.1 extended by a round at the beginning with a complexity of around 2^{200} computational steps.

Using the fact that a byte of $a^{(6)}$ depends on 4 ciphertext bytes ($c_0, \ldots, c_3$, say) and the byte $m^{(1)}_{a,b}$ depends on 4 plaintext bytes ($p_4, \ldots, p_7$, say), we can implement the above method more efficiently. We use a three-phase attack as follows:

Phase 1: Using 2^{64} counters, m_y, we count (modulo 2) the number of times the 8 byte quantity $y = (c_0, \ldots, c_3, p_4, \ldots, p_7)$ occurs in the collection of 2^{128} plaintext/ciphertext pairs.

Phase 2: We guess the four necessary key bytes of the first round, and separate the encryptions and counters into herds by computing $m^{(1)}_{a,b}$ for each counter position using the quadruple $(p_4, \ldots, p_7)$. We then select a single herd, and for every combination of $z = (c_0, \ldots, c_3)$ we update a counter n_z by adding m_y to it for each y that agrees with z and is in the correct herd.

Phase 3: We guess five key bytes at the end of the cipher and calculate a single byte of $a^{(6)}$ for each combination of z. Summing this byte over all 2^{32} values of z will yield a zero result if the key guesses were correct.

The first phase requires us to perform 2^{128} memory lookups in order to compute the m_y values. Using a rough equivalence of 2^8 memory lookups to an encryption [23], Phase 1 is comparable to 2^{120} encryptions. The remaining phases have a negligible work factor of 2^{96} encryptions and so exhaustive search over the 5 guessed key bytes of Phase 3 will suffice instead of using the partial sum method. There is also a 2^{64} bit memory requirement for storing the counters.

8.3.1 Reducing the Data Requirement

We are using the entire codebook in performing this attack, but we can reduce the number of required encryptions by 2^{119} if we focus our attention again on the bytes of $m^{(1)}$.

Recall that we fixed a single byte of $m^{(1)}$ to define our herds. The four plaintext bytes $p_4, \ldots, p_7$ and the four initial key bytes used to calculate $m^{(1)}_{a,b}$, however, define 4 bytes of $m^{(1)}$. We obviously cannot fix all four bytes since at least one of these bytes has to be active within the pack. Therefore we do the next best thing. Fixing three bytes of $m^{(1)}$ for each herd, we get 2^{24} herds each of 2^{104} encryptions.

If we consider the collection of 2^{128} encryptions and remove all encryptions in which the 4 plaintext bytes $p_4, \ldots, p_7$ take on any value from an arbitrary set of 2^{23} combinations (i.e., we keep $2^{32} - 2^{23}$ out of the possible 2^{32} combinations), then about half ($\approx 2^{23}$) of the herds will have missing plaintext/ciphertext pairs while the other half remain complete [23]. We can use these complete herds in the attack to reduce the data requirement to $2^{128} - 2^{119}$ plaintexts. Note that the reduction in the number of plaintexts does not affect the time complexity of 2^{120} and so this 7 round attack is applicable to all three AES versions.

8.4 Eight-Round Extension

Having improved the 7 round attack significantly, we now consider extension of this attack by an eighth round for both AES-192 and AES-256.

8.4.1 Attacking AES-256

As before, we have $2^{128} - 2^{119}$ plaintexts (23 herds) and summing a single byte of $a^{(6)}$ over all 2^{104} encryptions in any herd will result in zero. However, the byte in $a^{(6)}$ now depends on 21 key bytes at the end of the cipher and all 16 ciphertext bytes, $c_0, \ldots, c_{15}$, so we must use the partial sum method of §8.1. Defining our herds by guessing 4 key bytes at the beginning and then calculating the partial sums x_m leads to an attack using 2^{104} bits of memory for counters and around 2^{202} encryptions. We need to use four herds before we can start eliminating wrong key guesses [23], and so we have an overall complexity of 2^{204}; faster than exhaustive search for AES-256, but clearly not for AES-192.

8.4.2 Attacking AES-192

Using the key schedule weakness of AES-192, we can improve this attack by a factor of 2^{16}. We begin by guessing the last 3 columns of K^8, bytes $k_0, \ldots, k_{11}$, say, and computing Equation (8.1) for each column. This gives us 2^{56} counters for

$$(x_{0,\ldots,3}, x_{4,\ldots,7}, x_{8,\ldots,11}, c_{12}, c_{13}, c_{14}, c_{15}),$$

where we have introduced the slightly different notation

$$x_{a,\ldots,b} := \bigoplus_{j=a}^{b} S_j[c_j + k_j].$$

Each $x_{a,\ldots,b}$ of the above septuple therefore corresponds to a guessed column of K^8. Evaluating the 2 relevant bytes of K^7 (for example $K^7_{1,0}$ and $K^7_{2,1}$ as in §7.1.2) that we get for free using the key schedule, we can now perform Step 3 of the algorithm from §8.1. This gives us

$$(x_{0,\ldots,7}, x_{8,\ldots,11}, c_{12}, c_{13}, c_{14}, c_{15}),$$

and so we have essentially reduced the number of counters from 2^{56} to 2^{48}. We now guess the final column of K^8 and by evaluating Equation (8.1), we get 2^{24} counters for

$$(x_{0,\ldots,7}, x_{8,\ldots,11}, x_{12,\ldots,15}).$$

Guessing the remaining 2 necessary bytes of K^7, we get 2^8 counters for $x_{0,\ldots,15} = x_{15}$. Evaluating the byte of K^6 that we get from the key schedule we can now sum the corresponding byte of $a^{(6)}$ over all 2^{104} encryptions to check for a zero result.

We have evaluated Equation (8.1) five times in this improved attack; one time for each column of K^8, and once finally using these 4 results instead of the ciphertexts $c_{i,0}, \ldots, c_{i,3}$.

We have again used $2^{128} - 2^{119}$ plaintexts but now have a work factor of 2^{188}. The 2^{16} factor improvement comes from the fact that we evaluate the first two bytes of K^7 before we guess the final column of K^8, reducing the number of required counters.

8.5 A 9-Round Related-Key Attack

At Eurocrypt '93, Biham [4] introduced a new type of attack on block ciphers known as a *Related Key* attack. This involves creating a set of keys that are in some way related to the original unknown cipher key, eg. they differ from the original by a fixed amount. These differences are tracked throughout the key schedule and encryptions are then performed using each of the related keys, revealing information about the value of the cipher key.

This attack may, at first, seem unrealisable in that it would be unlikely that an adversary could persuade a human cryptographer to encrypt plaintexts under a set of keys with some fixed relation. However, modern cryptography tends to be implemented using complex computer protocols, and so in some cases this attack can become very feasible.

The authors of Rijndael, in their AES proposal submission [19], make the statement, "The key schedule of Rijndael, with its high diffusion and non-linearity, makes it very improbable that [related-key attacks] can be successful

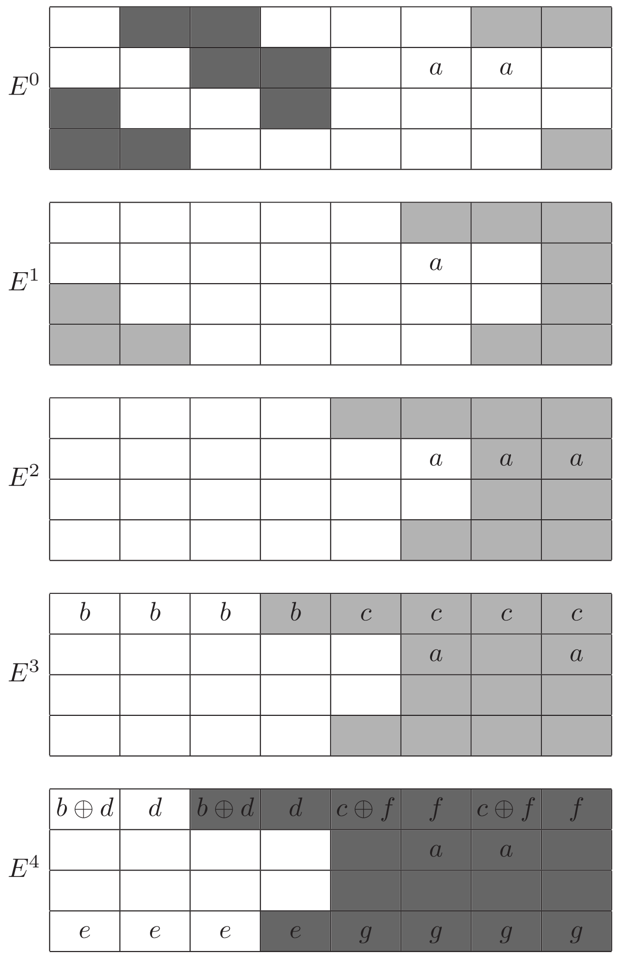
E^0
a a
E^1
a
E^2
a a a
E^3
b b b b c c c c
a a
E^4
$b \oplus d$ d $b \oplus d$ d $c \oplus f$ f $c \oplus f$ f
a a
e e e e g g g g

Figure 8.1: Difference and guessing pattern in the keys of the 9-round attack.

for Rijndael", and list resistance to related-key attacks as one of the requirements for the design of the Rijndael key schedule. However, despite this requirement, Ferguson et al. [23] were able to develop a 9-round attack on the AES using the principle of related keys.

Using a variant on the Square-6 attack, i.e., using 256 related keys that differ only in a single byte of the fourth round key, and carefully chosen plaintexts, we can get three bytes of $a^{(7)}$ that sum to zero over the 256 encryptions. The attack enables us to deduce information about key bytes of the last three rounds. It has a running time of 2^{248}, and therefore only applies to AES-256.

8.5.1 Key Difference Propagation

The 32 bytes of a 256-bit quantity, K, are arranged as follows:

$$K = \begin{bmatrix} K_{0,0} & K_{0,1} & K_{0,2} & K_{0,3} & K_{0,4} & K_{0,5} & K_{0,6} & K_{0,7} \\ K_{1,0} & K_{1,1} & K_{1,2} & K_{1,3} & K_{1,4} & K_{1,5} & K_{1,6} & K_{1,7} \\ K_{2,0} & K_{2,1} & K_{2,2} & K_{2,3} & K_{2,4} & K_{2,5} & K_{2,6} & K_{2,7} \\ K_{3,0} & K_{3,1} & K_{3,2} & K_{3,3} & K_{3,4} & K_{3,5} & K_{3,6} & K_{3,7} \end{bmatrix}.$$

Starting with an unknown 256-bit base key Y, we derive a set of 256 related keys $Y_0, \ldots, Y_{255}$, such that the difference $Y_a \oplus Y$ is zero everywhere except in positions $(1,5)$ and $(1,6)$ where it takes the value a. For AES-256, 5 cycles of the key schedule produces the 256-bit quantities $E^0, \ldots, E^4$, from which we generate the 10 necessary round keys $K^0, \ldots, K^9$ by "splitting" the E^i into halves. We generate the E^i for each of the 256 related keys Y_j and track their difference propagation.

In the first cycle we have a difference a in $E^0_{1,5}$ and $E^0_{1,6}$. The next cycle provides a difference of a in $E^1_{1,5}$. Thirdly, we have a difference of a in $E^2_{1,5}, E^2_{1,6}$ and $E^2_{1,7}$.

In the fourth cycle, the difference is faced with the first applications of `BS`. To calculate the output difference b of `BS` from the input difference a, we need to guess (or know) the byte $E^2_{1,7}$ for the base key Y. The bytes $E^3_{0,i}$ for $i = 0, \ldots, 3$ take on the difference value b due to the application of `BS` and `RotWord` on $E^2_{1,7}$. The byte difference of $E^3_{0,4}$ is then altered by the `BS` operation acting on $E^3_{0,3}$ and so this byte must be guessed for the key Y to compute the output difference c. This gives a difference of c in $E^3_{0,i}$ for $i = 4, \ldots, 7$. $E^3_{1,5}$ and $E^3_{1,7}$ also have a difference of a induced from the previous cycle.

The differences within the fifth cycle are determined in a similar way, and all these differences can be seen in Figure 8.1. The dark-grey bytes are the bytes of the expanded key of Y that we guess for this attack. The light-grey bytes can be deduced from our guesses using the key schedule. Although we will need all 27 guessed bytes, it turns out that we need only 6 of these to track the differences throughout the key schedule.

8.5.2 The Attack

We begin by guessing the dark-grey bytes in Figure 8.1, and since we know the differences in K^1 we can choose a collection of 256 plaintexts such that when we encrypt one plaintext under each key, all encryptions end up in the same state of $a^{(2)}$. Since K^4 has a single difference, we end up with a single active byte in $a^{(4)}$, i.e., the set of 256 states at $a^{(4)}$ form a Λ^1-set. This now means that $m^{(5)}$ is a Λ^{16}-set.

The next few steps are shown in Figure 8.2. Round 8 is considered to have an equivalent round structure such that the KeyAddition and MixColumns operations are swapped. We therefore add the round key $K^{8'} = \texttt{MC}^{-1}(K^8)$ before the `MC` operation. From the ciphertext and our guessed key bytes, we can compute back to the KeyAddition operation of round 8. We guess the 4

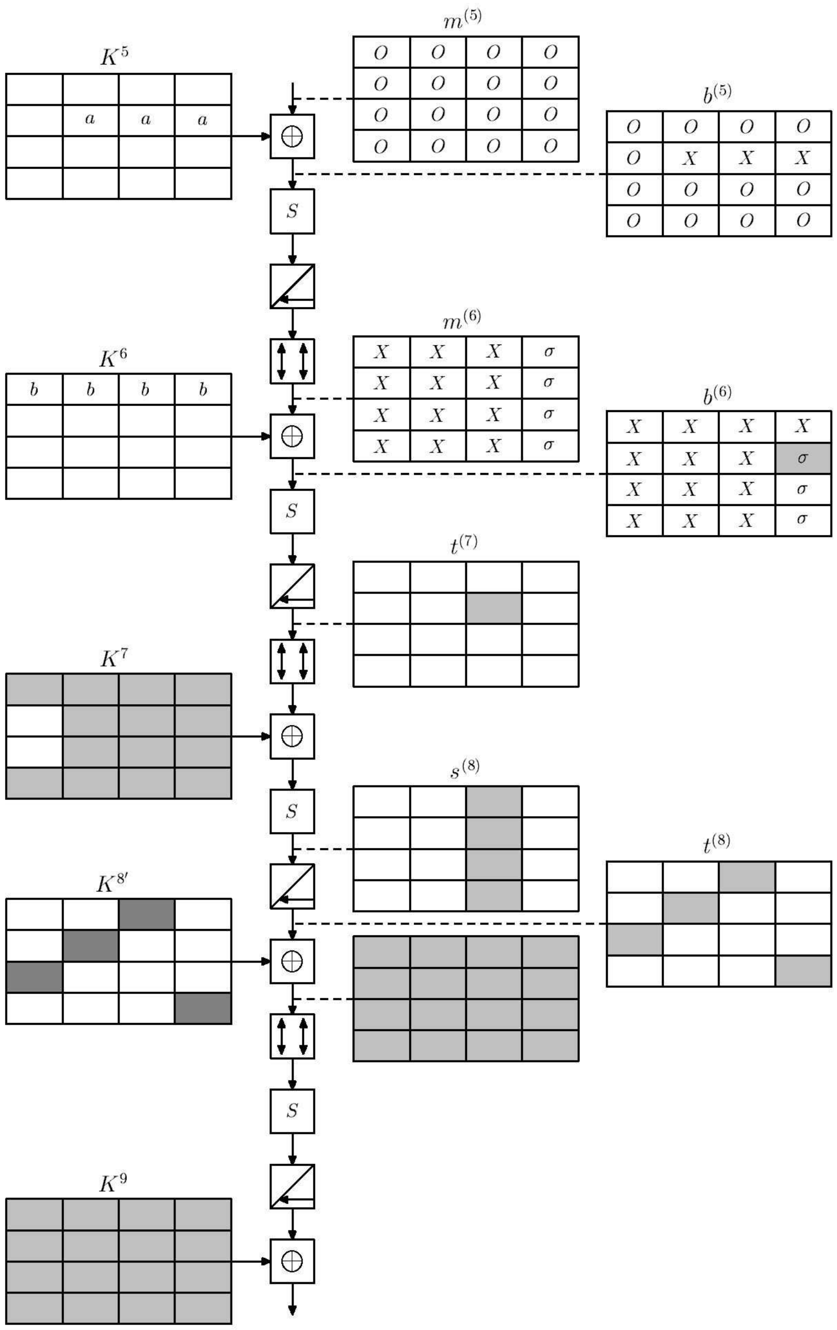

Figure 8.2: Rounds 6–9 of the attack. Adapted from [23].

marked bytes of $K^{8'}$ and therefore can calculate the corresponding bytes of $t^{(8)}$ and the column $s_2^{(8)}$. Using the key schedule we can deduce K_2^7 from our initial guessed key bytes in Figure 8.1, and therefore we compute $t_{1,2}^{(7)}$ and finally $a_{1,3}^{(7)}$. Summing the byte $a_{1,3}^{(7)}$ over the 256 encryptions will yield zero if all key byte guesses were correct.

All in all, we have guessed 31 key bytes and from the key schedule can deduce a further 35 bytes. For each guess the amount of work we perform is approximately equivalent to a single encryption and so this attack has complexity 2^{248}. Since we are only varying 8 bytes of the plaintext to give us the same state of $a^{(2)}$, it is enough to encrypt a collection of 2^{64} plaintexts with each of the 256 related keys to form this attack. This gives us a data requirement of 2^{72} plaintexts.

8.5.3 An Improvement

Using the improvement described in §8.1 we can reduce the work factor to $5 \cdot 2^{224}$. Instead of guessing the 8 bytes of K^0, we can use all 2^{64} plaintexts for each of the 256 related keys and sum over all 2^{72} encryptions to get a zero result. About 32 structures of 2^{72} plaintext/ciphertext pairs are required to identify the correct key byte guesses [23], and so we increase the plaintext requirement to 2^{77}.

We now decrypt, to $s^{(8)}$ say, a single structure of 2^{72} ciphertexts by guessing the 19 dark-grey bytes of E^4, and count how many times each combination for the column $s_2^{(8)}$ occurs. This now requires us to evaluate an expression similar to Equation (8.1) with a work factor of 2^{48} XORs. This has to be done for 5 structures to start removing incorrect key guesses and so the complexity is reduced to $5 \cdot 2^{224}$.

8.6 Summary

We have discussed an improvement on the Square-6 attack that, although requiring $6 \cdot 2^{32}$ plaintexts, has a reduced complexity of 2^{44}. This improvement can be extended to the 7-round attacks of Lucks [32]. The 192-bit attack now has a complexity of 2^{155}, requiring $19 \cdot 2^{32}$ chosen plaintexts, while the 256-bit version has running time 2^{170} and data complexity $21 \cdot 2^{32}$.

A further improvement to the generic 7 round attack sees a reduced complexity of 2^{120}, enabling it to be mounted on any key size. However, we now have a data requirement nearly equivalent to the entire codebook ($2^{128} - 2^{119}$ plaintexts). This improvement lends itself to an 8-round attack on AES-192 and AES-256 with complexities 2^{188} and 2^{204}, respectively.

There is also a related-key attack on AES-256 reduced to 9 rounds. This uses 2^{77} plaintexts, encrypted under 256 related keys, and has complexity $5 \cdot 2^{224}$.

Chapter 9

Algebraic Representation of the AES

We have now looked at some very prominent attacks on the AES that can be extended up to 9 rounds. There are, however, several other promising methods of retrieving the cipher key that do not have the downfall of being restricted to a reduced number of rounds, and can generally be implemented in a known (as opposed to chosen) plaintext setting. These involve finding an algebraic representation for a single encryption of the AES (as a function of the plaintext, ciphertext and round keys) and using advanced techniques to solve for the unknown key bytes.

Firstly, we will discuss a closed algebraic form for the AES as a continued fraction; we find a simple expression for a single round and extend this to an entire encryption. Finally, we show that the AES can be embedded within a larger cipher known as the BES. One consequence of this embedding is that an AES encryption can be written as a system of *Multivariate Quadratic* polynomials (an MQ-system) that is both sparse (i.e., most of the coefficients of the possible terms are zero) and overdetermined (i.e., there are more equations than variables). We look at methods for solving such MQ-systems in the subsequent chapters.

9.1 A Closed Algebraic Form for the AES

In their 2001 paper, Ferguson, Schroeppel and Whiting [24] discovered that the security of the AES can be reduced to the problem of solving what could be considered a generalised version of continued fractions. To see how this comes about, we first consider the individual round transformations of an AES encryption.

Recall that the S-box transformation $z = f(g(x))$ of a byte x can be described as follows: The element $g(x) := x^{(-1)} \in \mathrm{GF}(2^8)$ is viewed as an element of

$\mathrm{GF}(2)^8$, multiplied by an 8×8 GF(2)-matrix L_A of Equation (4.1), XORed with the constant `63`, and the result is then viewed in the natural way as an element of $\mathrm{GF}(2^8)$. The function $f : \mathcal{F} \to \mathcal{F}$ can be written as follows for $a \in \mathcal{F}$:

$$f(a) = \psi^{-1}\Big(L_A\big(\psi(a)\big) + \texttt{63}\Big),$$

where ψ is the natural map $\psi : \mathcal{F} = \mathrm{GF}(2^8) \to \mathrm{GF}(2)^8$.

Cryptanalysis of the AES is made more difficult by the need for the functions ψ and ψ^{-1}. It turns out that there is an equivalent definition for the function f which has coefficients in the field $\mathcal{F}$ and thus does not require the mapping ψ [19]. It replicates the action of the affine transformation over GF(2) whilst working entirely in $\mathcal{F}$. This equivalent function acting on an element $x \in \mathcal{F}$ is given by

$$f(x) = \sum_{e=0}^{7} \lambda_e x^{2^e} + \lambda_8,$$

where $(\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8) = (\texttt{05}, \texttt{09}, \texttt{f9}, \texttt{25}, \texttt{f4}, \texttt{01}, \texttt{b5}, \texttt{8f}, \texttt{63})$. Combined with the $\mathcal{F}$ "inversion" $x \mapsto x^{254}$, this gives the S-box polynomial employed in an interpolation attack (§5.6).

The constant `63` can be 'removed' from this equation by incorporating it into a slightly modified key schedule [39] to produce the simplified transformation

$$f(x) = \sum_{e=0}^{7} \lambda_e x^{2^e}. \tag{9.1}$$

For the purposes of this chapter, we will use this simplified version, keeping in mind that we now have to use a modified key schedule.

Let $a^{(r)}$ be the input state to round r (as in §4.5). Then from above, the S-box transformation of this state can be described by

$$s_{i,j}^{(r)} = f\big((a_{i,j}^{(r)})^{(-1)}\big) = \sum_{e=0}^{7} \lambda_e \big((a_{i,j}^{(r)})^{(-1)}\big)^{2^e}.$$

This is true for all $a_{i,j}^{(r)}$. If we adopt the convention that $a/0 := 0$, then this can be rewritten

$$s_{i,j}^{(r)} = \sum_{e=0}^{7} \lambda_e (a_{i,j}^{(r)})^{-2^e},$$

true for all $a_{i,j}^{(r)}$.

The application of `SR` transforms this as

$$t_{i,j}^{(r)} = s_{i,i+j}^{(r)} = \sum_{e=0}^{7} \lambda_e (a_{i,i+j}^{(r)})^{-2^e}.$$

Thirdly, `MC` provides

$$m_{i,j}^{(r)} = \sum_{d=0}^{3} v_{i,d} t_{d,j}^{(r)},$$

where $v_{i,d}$ are the coefficients of the `MC` matrix in Equation (4.2). Substitution then gives

$$\begin{aligned} m_{i,j}^{(r)} &= \sum_{d=0}^{3} v_{i,d} \sum_{e=0}^{7} \lambda_e (a_{d,d+j}^{(r)})^{-2^e} && (9.2) \\ &= \sum_{d=0}^{3} \sum_{e=0}^{7} w_{i,d,e} (a_{d,d+j}^{(r)})^{-2^e}, && (9.3) \end{aligned}$$

for suitable constants $w_{i,d,t}$.

Finally comes the key addition step, simply described by

$$a_{i,j}^{(r+1)} = K_{i,j}^{r} + \sum_{d=0}^{3} \sum_{e=0}^{7} w_{i,d,e} (a_{d,d+j}^{(r)})^{-2^e},$$

where K^r is the round key of round r and $a^{(r+1)}$ is the input to round $r+1$.

Rewriting this equation, using $\mathcal{D} := \{0, \ldots, 3\}$ and $\mathcal{E} := \{0, \ldots, 7\}$, gives

$$a_{i,j}^{(r+1)} = K_{i,j}^{r} + \sum_{\substack{d \in \mathcal{D} \\ e \in \mathcal{E}}} \frac{w_{i,d,e}}{(a_{d,d+j}^{(r)})^{2^e}};$$

a simple algebraic expression describing an inner round of the AES.

9.1.1 Multiple Round Equations

Considering two rounds of the AES, we get the expression

$$a_{i,j}^{(3)} = K_{i,j}^{2} + \sum_{\substack{d_2 \in \mathcal{D} \\ e_2 \in \mathcal{E}}} \frac{w_{i,d_2,e_2}}{\left(K_{d_2,d_2+j}^{1} + \sum_{\substack{d_1 \in \mathcal{D} \\ e_1 \in \mathcal{E}}} \frac{w_{d_2,d_1,e_1}}{(a_{d_1,d_1+d_2+j}^{(1)})^{2^{e_1}}} \right)^{2^{e_2}}}.$$

Since we working in a field with characteristic 2, we can apply the "Freshman's Dream", $(a+b)^2 = a^2 + b^2$, so that

$$a_{i,j}^{(3)} = K_{i,j}^{2} + \sum_{\substack{d_2 \in \mathcal{D} \\ e_2 \in \mathcal{E}}} \frac{w_{i,d_2,e_2}}{(K_{d_2,d_2+j}^{1})^{2^{e_2}} + \sum_{\substack{d_1 \in \mathcal{D} \\ e_1 \in \mathcal{E}}} \frac{w_{d_2,d_1,e_1}^{2^{e_2}}}{(a_{d_1,d_1+d_2+j}^{(1)})^{2^{e_1+e_2}}}}.$$

We can continue this process to write down an algebraic equation for more rounds, but the number of terms increases very rapidly and the expression becomes quite complicated. For this reason, we will introduce some rather 'casual' notation to clarify the overall structure. Any known constants will be denoted by C, any key bytes by K and known powers and subscripts by $\star$. Using the fact that the initial key addition step is simply given by $a^{(1)} = p + K^0$ where p is the plaintext, we can now write down a five-round expression as follows:

$$a^{(6)}_{i,j} = K + \sum_{\substack{d_5 \in \mathcal{D} \\ e_5 \in \mathcal{E}}} \cfrac{C}{K^\star + \sum\limits_{\substack{d_4 \in \mathcal{D} \\ e_4 \in \mathcal{E}}} \cfrac{C}{K^\star + \sum\limits_{\substack{d_3 \in \mathcal{D} \\ e_3 \in \mathcal{E}}} \cfrac{C}{K^\star + \sum\limits_{\substack{d_2 \in \mathcal{D} \\ e_2 \in \mathcal{E}}} \cfrac{C}{K^\star + \sum\limits_{\substack{d_1 \in \mathcal{D} \\ e_1 \in \mathcal{E}}} \cfrac{C}{K^\star + p^\star_\star}}}}}, \qquad (9.4)$$

remembering that each C and $\star$ are known quantities and each K is an unknown key byte, and that all of these values depend on the enclosing summation.

9.1.2 An Attack?

A 5-round equation such as Equation (9.4) involves approximately 2^{25} terms of the form $C/(K^\star + p^\star_\star)$, and we can write out a full 10-round expression for AES-128 with around 2^{50} terms. For AES-256 (14 rounds), the equation for half the cipher would have about 2^{35} terms and the expanded formula for the full cipher would comprise about 2^{70} terms.

Alternatively, one could write out a similar 5-round equation to the above, describing the inverse cipher acting on the ciphertext. Since this expression will result in the same intermediate state as Equation (9.4), the two can be equated, forming a closed algebraic equation describing the full 10-round cipher, but with only approximately 2^{26} terms. By repeating this equation for $2^{26}/16 = 2^{22}$ plaintext/ciphertext pairs, we would have enough information to theoretically solve for the unknown key bytes [24].

At the present time, we are aware of no efficient algorithms for solving such generalised continued fractions (and thus forming an effective attack on the AES using these equations). The authors of [24], however, have made the observation that an algorithm designed to solve the equations could afford a time complexity of $\mathcal{O}(n^4)$, in the number n of terms in the equation, before exceeding the workload of brute force on AES-128. Furthermore, an attack on AES-256 could use an algorithm of order $\mathcal{O}(n^7)$. It is an open problem as to whether a method can be found (or indeed already exists) to solve for the unknown key bytes K given enough plaintext/ciphertext pairs.

9.2 The Big Encryption System (BES)

In his famous paper from 1949, Claude Shannon [49] states that breaking a good cipher should require “at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type”. It turns out that every cipher can, in fact, be expressed as such a system over GF(2). The AES is no exception. Courtois and Pieprzyk [15] managed to produce an MQ-system of 8000 equations with approximately 89600 terms in 1600 variables over GF(2), describing a single AES-128 encryption. For AES-256 we get a system of 22400 equations with about 125440 terms in 4480 variables. In this paper, we are not interested in the derivation of these MQ-systems (GF(2)-systems), which can be found in [15], however we will use their mentioned properties in the following chapters.

Murphy and Robshaw, at Crypto ‘02 [40], found that it is more interesting to work over the field $\mathcal{F}$. They developed a new cipher called the *Big Encryption System* (BES), an extension of the AES with the advantage that all operations in the BES are performed over $\mathcal{F}$, instead of combining operations over both GF(2) and $\mathcal{F}$. This allows for greater ease of cryptanalysis of the embedded AES. One consequence of the new cipher is that an AES encryption can be written as an extremely sparse overdetermined MQ-system over the field $\mathcal{F}$ ($\mathcal{F}$-system). It turns out that this $\mathcal{F}$-system is far simpler than the GF(2)-system of [15] and this will become important for the XSL attack, described in Chapter 11.

9.2.1 Notation and Preliminaries

We define the *vector conjugate mapping*, $\varphi : \mathcal{F} \to \mathcal{F}^8$ by

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

for any element $a \in \mathcal{F}$. This can be then extended to higher dimensions:
Let $\phi : \mathcal{F}^n \to \mathcal{F}^{8n}$, and if $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1}) \in \mathcal{F}^n$, define

$$\phi(\mathbf{a}) := \big(\varphi(a_0), \varphi(a_1), \ldots, \varphi(a_{n-1})\big).$$

The field inversion in $\mathcal{F}$ can also be extended in a natural way by considering component-wise inversion of an element in $\mathcal{F}^n$, i.e.,

$$\mathbf{a}^{(-1)} := (a_0^{(-1)}, a_1^{(-1)}, \ldots, a_{n-1}^{(-1)}).$$

The BES as is defined in the next section, has a 128-*byte* block length and 16-*byte* effective key length. It is essentially an extension of AES-128. Let **A** and **B** be the state spaces (the set of all possible states) of the AES and BES respectively. Then **A** and **B** respectively correspond to the vector spaces $\mathcal{F}^{16}$ and $\mathcal{F}^{128}$. Furthermore, the AES is embedded within the BES such that $\mathbf{B_A} := \phi(\mathbf{A}) \subset \mathbf{B}$.

An element $\mathbf{a} \in \mathbf{A}$ is usually represented as a 4×4 array of bytes in the AES. However, for the purposes of the description in this section, we will view $\mathbf{a}$ as a column vector by rearranging the state in the following way:

$$\mathbf{a} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} = (a_{00}, \ldots, a_{30}, a_{01}, \ldots, a_{31}, \ldots, a_{03}, \ldots, a_{33})^T.$$

In the same way, a state vector $\mathbf{b} \in \mathbf{B}$ has the form

$$\mathbf{b} = (b_{000}, \ldots, b_{007}, b_{100}, \ldots, b_{107}, \ldots, \ldots, b_{330}, \ldots, b_{337})^T,$$

where

$$(b_{ij0}, \ldots, b_{ij7}) := (a_{ij}^{2^0}, \ldots, a_{ij}^{2^7}) = \varphi(a_{ij}).$$

The idea is that every byte a in a state $\mathbf{a} \in \mathbf{A}$ of the AES will remain represented by its conjugate vector $\varphi(a)$ in $\mathbf{b} \in \mathbf{B}$ of the BES as it is processed by the cipher. Furthermore, the BES is defined in such a way that, for every AES cipher key K, we have

$$\mathrm{AES}_K(\mathbf{a}) = \phi^{-1}\Big(\mathrm{BES}_{\phi(K)}\big(\phi(\mathbf{a})\big)\Big).$$

9.2.2 Operations within the BES

The round structure of the BES is precisely that of the AES, however the individual operations are modified due to the change in the way we are viewing the state.

ByteSub:

The AES S-box substitution is composed of an $\mathcal{F}$-inversion g, followed by a GF(2)-linear transformation f. It is the tension between the fields $\mathcal{F}$ and GF(2) that makes it difficult to represent the AES S-box layer as a matrix multiplication. However, in the BES there is a simple matrix representation of this operation.

Component-wise inversion of an element $\mathbf{a} \in \mathbf{A}$ of the AES simply corresponds to the map $\mathbf{a} \mapsto \mathbf{a}^{(-1)}$. Similarly, for $\mathbf{b} \in \mathbf{B}$ of the BES, $\mathbf{b} \mapsto \mathbf{b}^{(-1)}$ has the same result, as expected. Inversion is with respect to $\mathcal{F}$ in both cases.

Recall from Equation (9.1) that (using a modified key schedule) the GF(2)-linear transformation $f : GF(2)^8 \to GF(2)^8$ has an equivalent $\mathcal{F}$-polynomial form

$$f(x) = \sum_{m=0}^{7} \lambda_m x^{2^m},$$

for suitable coefficients $\lambda_0, \ldots, \lambda_7 \in \mathcal{F}$. Note that f is described by a linear combination of conjugates.

We can define a matrix L_B that when acting on a vector of the form $\varphi(a) \in \mathcal{F}^8$ performs this 'linear transformation' f on the first byte and preserves the vector conjugacy property on the successive bytes, i.e.,

$$L_B \cdot \varphi(a) \equiv \varphi\big(f(a)\big).$$

This matrix is given by

$$L_B = \begin{bmatrix} (\lambda_0)^{2^0} & (\lambda_1)^{2^0} & (\lambda_2)^{2^0} & \cdots & (\lambda_7)^{2^0} \\ (\lambda_7)^{2^1} & (\lambda_0)^{2^1} & (\lambda_1)^{2^1} & \cdots & (\lambda_6)^{2^1} \\ (\lambda_6)^{2^2} & (\lambda_7)^{2^2} & (\lambda_0)^{2^2} & \cdots & (\lambda_5)^{2^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\lambda_1)^{2^7} & (\lambda_2)^{2^7} & (\lambda_3)^{2^7} & \cdots & (\lambda_0)^{2^7} \end{bmatrix}.$$

The action of this matrix represents the linear transformation on one byte of the state in the AES. The action on the entire state is then given by the 128×128 $\mathcal{F}$-matrix in the BES, $\mathbf{Lin}_B = Diag_{16}(L_B)$, where $Diag_{16}(L_B)$ denotes a block diagonal matrix with 16 identical submatrices L_B.

ShiftRows:

The `SR` operation can be seen to be equivalent to performing a permutation on the elements of the state $\mathbf{a} \in \mathbf{A}$. This permutation can be accomplished via multiplication by a 16×16 $\mathcal{F}$-matrix, $\mathbf{R}_A$. Similarly, this permutation can be extended to $\mathbf{b} \in \mathbf{B}$ as a 128×128 $\mathcal{F}$-matrix, $\mathbf{R}_B$, with the condition that the octuples of conjugates are shifted as one entity.

MixColumns:

Recall that `MC` performs column-wise mixing of the state through multiplication by the 4×4 $\mathcal{F}$-matrix

$$C_A = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix}.$$

The action in the AES can thus be represented by the 16×16 $\mathcal{F}$-matrix, $\mathbf{Mix}_A = Diag_4(C_A)$.

To generalise this to the BES, thereby preserving the conjugacy property, we have to consider 8 versions of this matrix, namely

$$C_B^{(m)} = \begin{bmatrix} (\texttt{02})^{2^m} & (\texttt{03})^{2^m} & \texttt{01} & \texttt{01} \\ \texttt{01} & (\texttt{02})^{2^m} & (\texttt{03})^{2^m} & \texttt{01} \\ \texttt{01} & \texttt{01} & (\texttt{02})^{2^m} & (\texttt{03})^{2^m} \\ (\texttt{03})^{2^m} & \texttt{01} & \texttt{01} & (\texttt{02})^{2^m} \end{bmatrix}, \quad \text{for } m = 0, \ldots, 7.$$

These matrices can be arranged into a 128×128 $\mathcal{F}$-matrix, $\mathbf{Mix}_B$ which satisfies this vector conjugacy condition on the BES and provides the required action of `MC` on the AES.

KeyAddition:

Key addition is performed via vector (XOR) addition. If $(\mathbf{k}_A)_i \in \mathbf{A}$ is the round key for round i of the AES, then key addition with a state $\mathbf{a} \in \mathbf{A}$ is obviously the map $\mathbf{a} \mapsto \mathbf{a} + (\mathbf{k}_A)_i$. Similarly for the BES, $\mathbf{b} \mapsto \mathbf{b} + (\mathbf{k}_B)_i$, where the round key $(\mathbf{k}_B)_i \in \mathbf{B}$ corresponding to $(\mathbf{k}_A)_i$ of the AES is defined as $(\mathbf{k}_B)_i := \phi\big((\mathbf{k}_A)_i\big)$.

The BES Round Function

Having defined all round operations, one full round of the BES (and hence the AES) has the action

$$\mathbf{b} \mapsto (\mathbf{Mix}_B \cdot \mathbf{R}_B \cdot \mathbf{Lin}_B \cdot \mathbf{b}^{(-1)}) + (\mathbf{k}_B)_i,$$

or simply

$$\mathbf{b} \mapsto (\mathbf{M}_B \cdot \mathbf{b}^{(-1)}) + (\mathbf{k}_B)_i, \tag{9.5}$$

for round $i = 1, \ldots, 9$, where $\mathbf{M}_B$ is a 128×128 $\mathcal{F}$-matrix incorporating the linear functionality of the diffusion layers. As in the AES, the final round does not include the $\mathbf{Mix}_B$ operation, and so replacing $\mathbf{M}_B$ with $\mathbf{M}_B^* := \mathbf{Mix}_B^{-1} \cdot \mathbf{M}_B$ in (9.5) describes the necessary final round transformation.

9.2.3 Observations on the BES

Several interesting properties on the BES are noted in [40]. One observation is that the diffusion matrix $\mathbf{M}_B$ has order 16 (in fact, the diffusion layer of the unembedded AES also has order 16 [39]). While this is rather surprising, the round function also includes a round key addition, which necessarily counteracts this property and hence any implications on the AES. More remarkably, the authors of [40] show that there exists a differential-type effect in the BES that holds with probability one, which can be extended to any number of rounds (!). This observation, however, does not apply when specific details of the key schedule are considered and thus does not affect the security of the embedded AES.

9.3 AES as a Simple MQ-System

We can exploit this "simplified" version of the AES to form a rather well-structured $\mathcal{F}$-system. Denote the plaintext by $\mathbf{p} \in \mathbf{B}$ and the ciphertext by $\mathbf{c} \in \mathbf{B}$. Let $\mathbf{x}_i$ be the output from the i^{th} round ($i = 0, \ldots 9$) of the BES, and subsequently define $\mathbf{y}_i := \mathbf{x}_i^{(-1)}$. If $\mathbf{k}_i$ is the round key for round i, then a BES encryption can be described by the following system of equations:

$$\begin{aligned}
\mathbf{x}_0 &= \mathbf{p} + \mathbf{k}_0, & \\
\mathbf{y}_i &= \mathbf{x}_i^{(-1)} & \text{for } i = 0, \ldots, 9, \\
\mathbf{x}_i &= \mathbf{M}_B \cdot \mathbf{y}_{i-1} + \mathbf{k}_i & \text{for } i = 1, \ldots, 9, \\
\mathbf{c} &= \mathbf{M}_B^* \cdot \mathbf{y}_9 + \mathbf{k}_{10}. &
\end{aligned}$$

Letting the $(8j+m)^{\text{th}}$ component of a vector $\mathbf{v}_i$ be $v_{i,(j,m)}$, then we can rewrite this system, for each component (j,m), as

$$\begin{aligned}
x_{0,(j,m)} &= p_{(j,m)} + k_{0,(j,m)}, && && (9.6)\\
y_{i,(j,m)} &= x_{i,(j,m)}^{(-1)} && \text{for } i = 0,\dots,9, && (9.7)\\
x_{i,(j,m)} &= (\mathbf{M}_B \cdot \mathbf{y}_{i-1})_{(j,m)} + k_{i,(j,m)} && \text{for } i = 1,\dots,9, && (9.8)\\
c_{(j,m)} &= (\mathbf{M}_B^* \cdot \mathbf{y}_9)_{(j,m)} + k_{10,(j,m)}, && && (9.9)
\end{aligned}$$

where $0 \le j \le 15$ and $0 \le m \le 7$. If we assume that $x_{i,(j,m)} \neq 0$, Equation (9.7) can be rewritten as

$$y_{i,(j,m)} x_{i,(j,m)} = 1.$$

Even if this assumption (which is valid for 53% of encryptions and 85% of 128-bit keys [40]) is false, only a small number of the below equations are incorrect. Letting $\mathbf{M}_B$ and $\mathbf{M}_B^*$ be α and β respectively, we can rearrange Equations (9.6), ... (9.9) to get

$$\begin{aligned}
0 &= p_{(j,m)} + x_{0,(j,m)} + k_{0,(j,m)}, && \\
0 &= y_{i,(j,m)} x_{i,(j,m)} + 1 && \text{for } i = 0,\dots,9,\\
0 &= x_{i,(j,m)} + k_{i,(j,m)} + \textstyle\sum_{(j',m')} \alpha_{(j,m),(j',m')} y_{i-1,(j',m')} && \text{for } i = 1,\dots,9,\\
0 &= c_{(j,m)} + k_{10,(j,m)} + \textstyle\sum_{(j',m')} \beta_{(j,m),(j',m')} y_{9,(j',m')};
\end{aligned}$$

which is a system of 2688 multivariate equations over $\mathcal{F}$ describing one complete BES encryption.

Since an AES state embedded in the BES is a vector of conjugates, the components $y_{i,(j,m)}$ and $x_{i,(j,m)}$ also satisfy the respective relations:

$$\begin{aligned}
y_{i,(j,m)}^2 &= y_{i,(j,m+1 \bmod 8)}, \text{ and}\\
x_{i,(j,m)}^2 &= x_{i,(j,m+1 \bmod 8)},
\end{aligned}$$

for $i = 0,\dots,9$. This adds a further 2560 equations to the system, making a total of 5248 multivariate equations with 7808 terms describing an AES encryption. These 7808 terms are comprised of 2560 state variables and 1408 key variables, and the equations consist of 3840 extremely sparse quadratic equations and 1408 linear equations.

In its most sparsest form, the key schedule can be written as a similar system of equations consisting of 2560 equations (960 extremely sparse quadratic and 1600 linear) with 2368 terms, composed of 1408 key variables and 640 auxiliary variables [40].

All in all, we have 7808 multivariate equations with a total of 10176 terms, which is certainly much more sparse than the GF(2)-system for AES-128. It still remains to be seen whether these equations can be formulated into an attack on the AES, and this is considered in the following chapters.

Chapter 10

The MQ-Problem

The problem of solving MQ-systems is known as the MQ-problem (sometimes simply referred to as MQ). The MQ-problem is known to be NP-hard over any field (see [25]). There are many schemes for solving the MQ-problem but most of them fail to produce a solution for large systems such as those of the AES due to the enormous complexity of the task. Recently, several techniques have been developed to specifically combat highly structured, overdetermined systems such as this, however, quite a lot of debate has arisen as to whether they are actually as practical and effective as the developers have claimed.

We first look at general methods for solving systems of multivariate polynomial equations. We then introduce an algorithm known as XL, developed specifically for solving large overdefined multivariate systems, and give a prototypical complexity evaluation using the AES-128 GF(2)-system.

Definition 6. A **monomial** in $\mathbf{x} := \{x_1, x_2, \ldots, x_n\}$ is a product of the form

$$c\, x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}, \qquad \text{for } \alpha_i \in \mathbb{N},\ 1 \leq i \leq n, \text{ and } c \in \mathcal{F}.$$

Denote the set of monomials in $\mathbf{x}$ by $M(\mathbf{x})$. Note here that the constant term is also a valid monomial by this definition ($\alpha_i = 0\ \forall i$).

Definition 7. The **lexicographical order**, denoted $\leq_{\text{lex}}$, on $M(\mathbf{x})$ is defined by: $x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n} \leq_{\text{lex}} x_1^{e_1} x_2^{e_2} \ldots x_n^{e_n}$ iff either $(d_1, d_2, \ldots, d_n) = (e_1, e_2, \ldots, e_n)$ or for some $k \leq n$, $d_k < e_k$ and $d_i = e_i\ \forall\, i < k$.

10.1 Gröbner Bases

The classical algorithms for solving MQ-systems are designed to construct Gröbner bases. A Gröbner basis can be viewed as a multivariate, non-linear generalisation of Gaussian elimination for linear systems.[1] These algorithms (such as Buchberger's algorithm, F4 and F5, see [48]) essentially order the monomials

[1]For a full treatment on Gröbner bases, we refer the reader to [48].

(typically in lexicographic order) and eliminate the "top" monomial by combining two equations with appropriate polynomial coefficients. This process is repeated until we end up with a polynomial equation in one variable. There are several methods for solving such univariate polynomial equations, one of which is known as Berlekamp's algorithm [3].

Unfortunately, the degrees of the remaining monomials increase rapidly during the elimination process and so even the most efficient of these Gröbner bases algorithms cannot handle quadratic equations with more than about $n = 15$ variables. In the worst case, the Buchberger algorithm is known to run in double exponential time [14]. There are, however, much more efficient algorithms for solving large systems of equations.

10.2 Linearisation and Relinearisation

Consider a random (overdetermined) system of $n(n+1)/2$ homogeneous quadratic equations in n variables $x_1, \dots, x_n$. Replacing each pair $x_i x_j$ by a new variable y_{ij}, we can convert this quadratic system into a linear one with $n(n+1)/2$ equations in about $n(n+1)/2$ variables. Using Gaussian elimination, we can find all y_{ij} values. Taking the square root of y_{ii} in the field gives two possible values of x_i and the correct one can be found by using the values of y_{ij} to confirm the correct roots of y_{ii} and y_{jj} for all i and j. This well-known process for solving MQ-systems is called *linearisation.*

In 1999, Kipnis and Shamir [30] refined this process by noting that the values y_{ij} are not independent. Due to commutativity of field elements, we have for any a, b, c, d,

$$(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c),$$

and therefore

$$y_{ab} y_{cd} = y_{ac} y_{bd} = y_{ad} y_{bc},$$

providing additional non-linear equations. This technique is known as (degree 4) *relinearisation.* We can increase this to degree 6 using equations of the form

$$y_{ab} y_{cd} y_{ef} = y_{ad} y_{be} y_{cf} = \dots$$

and so on for higher degrees. The relinearisation technique is designed to handle systems of ϵn^2 equations in n variables, where $\epsilon < \frac{1}{2}$.

10.3 Extended Linearisation (XL)

At Eurocrypt '00, Courtois et al. [14] showed that many of the equations generated by higher-degree relinearisation are provably dependent on other equations and so can be eliminated. Although this reduces the size of the linearised systems, it also limits the types of polynomial equations that can be successfully solved by the technique. The XL (e**X**tended **L**inearisation) technique was developed by the authors of [14] as a simplified and improved version of relinearisation. The XL algorithm is at least as powerful as (and in practice, is in fact more efficient than) relinearisation since the independent equations produced by degree D relinearisation are equivalent to a subset of those obtained using XL with the same parameter D (see [14] for an outline proof).

10.3.1 The XL Algorithm

Let K be a field. Let $\mathcal{A}$ be a system of m multivariate quadratic equations $\ell_k = 0$ ($1 \leq k \leq m$) in n variables, where each ℓ_k is the multivariate polynomial $f_k(x_1, x_2, \ldots, x_n) - c_k$ for some $f_k \in K[x_1, x_2, \ldots, x_n]$ and $c_k \in K$. We will assume that $\mathcal{A}$ has a unique solution $(x_1, x_2, \ldots, x_n) = (b_1, b_2, \ldots, b_n) \in K^n$.

Let $D \in \mathbb{N}$ be a parameter of the XL algorithm. We consider all the polynomials $\prod_j x_{i_j} \cdot \ell_i$ of total degree $\leq D$. Let $\mathcal{I}_D$ be the ideal spanned by these equations. We have $\mathcal{I}_D \subset \mathcal{I}_\infty$, where $\mathcal{I}_\infty$ is the ideal generated by the ℓ_i.

Definition 8. ***The XL Algorithm.*** Execute the following steps:

1. ***Multiply.*** Generate all the products $\prod_{j=1}^{k} x_{i_j} \cdot \ell_i \in \mathcal{I}_D$ where $k \leq D-2$.
2. ***Linearise.*** Consider each monomial of degree $\leq D$ as a new variable and perform Gaussian elimination on the equations obtained in Step 1. The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last. A lexicographic order on the terms, for instance, will satisfy this condition.
3. ***Solve.*** If we have chosen D such that Step 2 yields at least one univariate equation in the powers of x_1 then solve this equation over K, using eg. Berlekamp's algorithm.
4. ***Repeat.*** Simplify the equations from Step 3 and repeat the process to find the values of the other variables.

For a "toy example" of the XL technique, see Appendix A.

Let R be the total number of equations generated in Step 1 and let T denote the number of monomials in these equations, including the constant term. Furthermore, let *Free* denote the number of equations in this set that are linearly independent (i.e., the dimension of $\mathcal{I}_D$). We have $\textit{Free} \leq R$ and necessarily $\textit{Free} \leq T$.

The basic principle of XL is that, if the system has a unique solution, for some D we will have $R \geq T$. When this occurs, it is expected that $\textit{Free} \approx T$. There is no need to have R much bigger than T, since *Free* cannot exceed T, and therefore we generally look for $R/T \approx 1$.

A necessary condition for XL to succeed over GF(q) is for $\textit{Free} \geq T - \min(D, q-1)$ [17]. Therefore, for $q = 2$, we have the condition $\textit{Free} \geq T - 1$. If $\textit{Free} = T$ then the system is insoluble if one of the equations contains a constant term. This can be shown by contradiction: if *Free* $= T$ then by elimination of $T-1$ non-constant monomials we will be left with a set of equations that can be combined, through some linear combination, to equal 1, and if there is a solution to these equations, we can substitute to get $0 = 1$.

Remark. Sometimes it is more efficient to consider only a subset of the possible monomials when performing Step 1. When all equations in a system are homogeneous quadratic equations, for example, it is sufficient to use only monomials of even (or odd) degrees (see the example, Appendix A).

10.3.2 Complexity Evaluation

Given m quadratic equations in n variables, the number of equations generated by the XL algorithm is about $R \approx \frac{n^{D-2}}{(D-2)!} \cdot m$. The total number of monomials in these equations is about $T \approx \frac{n^D}{D!}$, and therefore the inequality $R \geq T$ gives $m \geq \frac{n^2}{D(D-1)}$. The authors of [14] assume heuristically that $D_{\min}$, the minimum D needed for XL to succeed, is not far removed from what makes $R \geq T$, and thus obtain the rough estimate

$$D_{\min} \approx \frac{n}{\sqrt{m}}.$$

More recently, a strict lower bound of

$$D_{\min} \geq \frac{n}{\sqrt{c-1}+1}, \tag{10.1}$$

for $m = n + c,\ c \geq 1$ over any field K, has been found by [22], while the authors of [53] give a rigorous minimal degree requirement of

$$D_{\min} = \min\left\{ D \,\middle|\, [t^D]\frac{1}{1-t}\left(\frac{1-t^q}{1-t}\right)^n \left(\frac{1-t^2}{1-t^{2q}}\right)^m \leq \min(D, q-1) \right\}$$

over GF(q), where the combinatorial notation $[u]p$ denotes “the coefficient of term u in the expansion of p,” eg. $[x^2](1+x)^4 = 6$.

If $m \approx n$ and if we expect most of the generated equations to be independent (see below for a note on this hypothesis), then we can expect the complexity of the algorithm to be lower bounded by the complexity of Gaussian elimination on about $\frac{n^D}{D!}$ variables. The work factor is therefore at least

$$WF \geq \left(\frac{n^D}{D!}\right)^{\omega}, \tag{10.2}$$

where ω is the Gaussian complexity exponent (§3.3.3).

Note. Computer simulations performed in [17] show that it is, in fact, not possible to assume that “almost all” the generated equations over GF(2) are linearly independent. For example, when $D = 4$, $n = 20$ and $m = 30$, only 92.65% of the equations generated by XL are independent. Furthermore, the ratio Free/R appears to slowly decrease as the values of m and D increase. This can impact on both the speed and applicability of XL over GF(2).

10.3.3 Attempted Cryptanalysis of the AES using XL

Unfortunately (for the adversary), the XL attack is extremely inefficient when applied to the AES. Recall that the GF(2)-system describing AES-128 has 8000 equations in 1600 variables. Even using the lower bound in Equation (10.1) and the evaluation in Equation (10.2), we get a complexity of $WF \approx 2^{360}$ GF(2)-operations using the best known Gaussian exponent, $\omega = 2.3766$.

This exceptionally large complexity occurs because, in a randomly generated system of $R_{\text{ini}} = m = 8000$ quadratic equations in $n = 1600$ variables, we have $T_{\text{ini}} \approx n^2/2 \approx 2^{20}$ terms. This gives $R_{\text{ini}}/T_{\text{ini}} \approx 2^{-7.3}$, which is very small and so the XL algorithm has to do extensive work (corresponding to a very large value for D) to achieve an expanded system with $R/T \approx 1$.

As we have seen for AES-128, we actually have approximately $T_{\text{ini}} \approx 2^{16.5}$ terms over GF(2). This gives $R_{\text{ini}}/T_{\text{ini}} \approx 2^{-3.5}$ which is certainly an improvement but still quite small. This suggests that there **should** be a better attack.

10.4 Relatives of XL

There are several proposed improvements and variations on the XL technique:

XL': [17] This algorithm operates like XL, except that we try to eliminate down to r equations that involve only monomials in r of the variables, say $x_1, \ldots, x_r$, then solve the remaining system by brute-force substitution;
FXL: [14] Stands for **F**ixing and **XL**. We fix μ variables (for some μ), and then solve the resulting system of m equations in $n - \mu$ variables using XL;
XLF: [13] Stands for **XL** and apply **F**robenius mappings. The idea is to add the Frobenius mappings $x^q = x$ to the initial system over GF(q) where $q = 2^k$, by considering each term $x, x^2, x^4, \ldots, x^{2^{k-1}}$ as a separate variable, replicating all R equations k times by repeatedly squaring them, and using the equivalence of identical monomials as extra equations; and
XL2: [17] With XL it is possible to solve the system only when $T - \mathit{Free}$ is very small. XL2 modifies the final step of XL in order to attempt to increase the number of linearly independent equations, such that the algorithm will succeed with $T - \mathit{Free}$ much larger. XL2 resembles the "T' method" of the XSL technique of Chapter 11.

Asymptotically, these variants do not appreciably increase speed compared to the original XL over GF(2) [53, 55], and so are not discussed further in this paper; we merely introduce them here for completeness.

Chapter 11

Extended Sparse Linearisation (XSL)

In 1999, Kipnis and Shamir [30] made the important observation that solving the MQ-problem should be much easier for overdetermined systems. Furthermore, in 2002, Courtois and Pieprzyk [15] discovered that if the MQ-system is sparse, it is even easier to solve, and this lead to a new improvement on the XL algorithm which takes advantage of the structure of such systems. This now refined attack is known as XSL, which stands for "e**X**tended **S**parse **L**inearisation" or "multiply(**X**) by **S**elected monomials and **L**inearise".

The XSL technique has gained much attention since it was first published in 2002. It is claimed that XSL can break the AES with a work factor of as little as 2^{79} when applied to the Murphy-Robshaw $\mathcal{F}$-system. However, there are doubts as to whether the technique performs as efficiently as the authors claim, and this has caused much debate on the effectiveness and reliability of XSL in solving large systems. In particular, the claims against the AES may be inaccurate.

11.1 Core of the XSL Attack

In the XL algorithm above, we multiply each equation by every possible monomial of degree $\leq D - 2$. The XSL algorithm instead only multiplies them by "carefully selected" monomials, namely products of monomials that already appear in other equations.

Let $\mathcal{A}$ be our initial system and let $P \in \mathbb{N}$ be a parameter of the XSL algorithm. We partition $\mathcal{A}$ into several smaller systems $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and multiply the set of equations in each system $\mathcal{A}_i$ by products of up to P terms from other systems. The partition chosen by the authors of [15] is to treat each S-box and the linear layer of each round as a separate system (see §11.3). For some P, we expect to reach a certain threshold of linearly independent equations *Free* and then we apply a “final step”, using 2 or 3 system variables, to further increase this number such that we can linearise the system.

11.2 The T′ Method

Let x_1 be a variable and $\mathcal{T}$ be the set of all T terms in our system. Define $\mathcal{T}'$ as the set of all monomials m that are in $\mathcal{T}$ such that we also have $x_1 \cdot m \in \mathcal{T}$. Let T' be the number of these monomials. Assuming that we have reached $\mathit{Free} \geq T - T' + C$ for small $C \in \mathbb{N}$, we then apply the following final step, known as “the T' method”:

1. Using a single Gaussian elimination, bring the system to a form where each term is a linear combination of the terms in $\mathcal{T}'$. Perform this calculation twice, for $\mathcal{T}'$ defined separately for, say, x_1 and x_2.
2. In each of the two systems, there is a subsystem of C “exceeding” equations that contain only terms of $\mathcal{T}'$. Multiplying each subsystem by x_1 and x_2 respectively and then substituting the expressions from Step 1 gives a **new** set of equations that contain only terms of $\mathcal{T}'$, but for the other variable. These new equations are most likely linearly independent from the equations we already have [15]. Therefore, we are essentially increasing the number of equations by up to $2C$.
3. Repeat this process and if the initial system had a unique solution, we expect to end up with $\mathit{Free} = T$ or $\mathit{Free} = T - 1$. When this occurs we can linearise the system to solve for the unknowns.
4. If the attack fails, try another two different variables x_3 and x_4, say, or use three variables (and hence three systems) from the start.

See Appendix B for a working example of the T' method.

Note. It is expected that the number of new equations will grow at an exponential rate [15], however even if it grows by 1 each time, the attack will still work.

The aim of the XSL technique is to not have to iteratively solve the system, as in XL; but rather to end up with *Free* independent equations for $\mathit{Free} + 1$ monomials, thereby allowing the system to be solved uniquely. The T' method described above essentially attempts to increase the number of independent equations *Free* when $\mathit{Free}/T \approx 1$, without increasing the number of terms T.

For each equation containing only terms in $\mathcal{T}'$, the cost to generate an additional equation will be about T'^2 [15]. Since we are in deficit of T' such equations, we expect the T' method will perform about T'^3 operations. This can probably be improved to T'^ω and thus be negligible compared to T^ω (the cost of the XSL attack itself, as we will see later). For example, for AES-128 over GF(2), we find $T \approx 2^{96}$ and $T' \approx 2^{90}$, and for AES-256 we have $T \approx 2^{125}$ and $T' \approx 2^{114}$ [15].

11.3 Application to the AES

As was mentioned in §11.1, a particular partition of the original system is to treat each S-box and linear layer as a separate system. This actually enables us to formulate a (rough) complexity estimation of the XSL attack on block ciphers. There are two versions of the XSL attack that can be applied to block ciphers, namely the first and second XSL attacks, both studied in [16]. The first XSL attack is more general and does not consider the specific key schedule; the second XSL attack, on the other hand, does take into account the key schedule and thus is designed to obtain "concrete results" on ciphers such as the AES and Serpent (the second most popular cipher after Rijndael during the AES selection process). We consider only the second XSL attack for this reason.

11.3.1 Overdefined Equations on the S-box

Consider the byte inversion, $y = x^{(-1)}$ within the S-box transformation. As we have already seen, we can rewrite this as

$$xy = 1, \qquad \forall\, x \neq 0.$$

This is a single bi-linear equation relating x and y, but instead we can choose to look at the individual bits by equating powers of the polynomial representations of the bytes. This gives us 8 linearly independent bi-affine quadratic equations in the bits of x and y. Although converting from y to z introduces some linear complications, we still end up with 8 quadratic equations in the bits of x and z. Even if $x = 0$, seven of these equations still hold (the eighth equation contains the constant monomial 1).

The expressions $xy^2 = y$ and $x^2y = x$ also yield 8 bi-affine quadratic equations each. It turns out that despite the algebraic equivalence of the 3 bi-linear equations, all 24 bi-affine equations are linearly independent. Since we are in a characteristic 2 field ($\mathcal{F}$), the squaring operation does not increase the degree of the equations as we might expect, since

$$\begin{aligned}(b_7x^7 + \cdots + b_0)^2 &= b_7^2x^{14} + \cdots + b_0^2\\ &= b_7x^{14} + \cdots + b_0,\end{aligned}$$

and reduction modulo our irreducible polynomial is also linear in the bits.

These are in fact the only bi-affine equations that occur due to the fact that, since squaring is a linear operation, each of the equations

$$\begin{cases} x = x^2y, \\ x^2 = x^4y^2, \\ \quad\vdots \\ x^{128} = xy^{128}, \end{cases}$$

generate the same set (modulo a linear combination) of 8 bi-affine equations. Within these equations are $t = 81$ monomials. They are:

$$\{1, x_0, \ldots, x_7, z_0, \ldots, z_7, x_0z_0, \ldots, x_7z_7\}.$$

From the above, we have 23 quadratic equations in x and z that hold with probability 1. The 24th equation holds with probability 255/256 for each S-box. Furthermore, this equation holds with probability 53% throughout an entire encryption with AES-128 to approximately 11% for AES-256 [15]. Therefore, we should use all $r = 24$ equations if an attack performs only one or two executions of the cipher, otherwise we use $r = 23$.

11.3.2 Product of Terms

Let ρ be the number of plaintexts required to uniquely identify the key in this attack (i.e., the number of cipher executions required). For AES-128, we need only use one 128-bit plaintext block, but for AES-256 we require two. If S is the total number of S-boxes used in this attack, then since we consider ρ executions of the cipher, we have

$$S = \rho \cdot 16 \cdot N_r + D + E,$$

where D is the number of S-boxes used in the key schedule and E is a constant related to the number of key variables used within the diffusion layers (see [16]). We have $E = 0$ for AES-128 and $E = 1$ for the other two versions.

Recall the parameter P of the XSL attack. In the second XSL attack, we multiply each of the r equations of one S-box by all possible terms, t, for all subsets of $P - 1$ other S-boxes. The total number of equations generated by this method is about

$$R \approx r \cdot S \cdot t^{P-1} \binom{S-1}{P-1}.$$

The total number of terms in these equations is approximately

$$T \approx t^P \cdot \binom{S}{P}.$$

There are, however, some obvious linear dependencies within these equations. To see this, we consider the case when $P = 2$. Let $E_1, \dots, E_r$ and $E'_1, \dots, E'_r$ be the equations that exist respectively for two S-boxes. Let $T_1 \dots, T_t$ be the terms that appear in the E_i. Instead of writing the products $T_1E'_1, \dots, T_tE'_1$, we could equivalently write $T_1E'_1, \dots, T_{t-r}E'_1$ and then finish the set with $E_1E'_1, \dots, E_rE'_1$. Making this transformation for all of the equations introduced above, we find that each of the $E_iE'_j$ occur twice, creating a linear dependence.

From this simple example, we see that we should alter our equation generation method slightly. Instead of multiplying each of the equations by all t terms, we should only multiply them by the first $t - r$ terms, and then add the equations consisting of “products” of S-boxes. This gives a now smaller number of equations:

$$R \approx \sum_{i=1}^{P} \binom{S}{i} r^i \cdot \binom{S-i}{P-i} (t-r)^{P-i} = (t^P - (t-r)^P) \binom{S}{P},$$

in the same number of terms T.

11.3.3 Summary of Equations

We now summarise the number of remaining equations and terms used in the attack. For details on the derivations of these approximations, we refer the reader to [16].

We have (see the definition of T' given in §11.2):

$$T' \approx t' t^{P-1} \binom{S-1}{P-1},$$

where $t' < t$ is the number of terms within the basis for one S-box, that can be multiplied by some fixed variable and are still within the basis. For example, with $r = 23$ and $t = 81$ as in §11.3.1, we have $t' = 9$.

Within the diffusion layers we have:

$$R' \approx 128 \cdot \rho \cdot (N_r + 1) \cdot (t - r)^{P-1} \cdot \binom{S}{P-1}.$$

The final set of equations comes from the key schedule and the number of these is:

$$R'' \approx (S_k - L_k) \cdot (t - r)^{P-1} \cdot \binom{S}{P-1},$$

where L_k is the number of linearly independent key variables within the expanded key, and S_k is the number of key variables used in the attack.

11.4 Complexity Evaluation

The aim of the first phase of the XSL attack is to achieve

$$\frac{R + R' + R''}{T - T'} > 1.$$

We attempt to find the smallest P such that this is true and, assuming that the attack works for this value of P, calculate the complexity. The running time of the XSL attack is expected to be comparable to a Gaussian reduction on T variables, and so for this P the complexity is about:

$$WF \approx T^{\omega} \approx t^{\omega P} \cdot \binom{S}{P}^{\omega}.$$

11.4.1 AES over GF(2)

Upon calculating the number of equations generated in the XSL attack on the GF(2)-system of AES-128, the value $P = 8$ with $\rho = 1$ is found to produce $\frac{R+R'+R''}{T-T'} = 1.005$. The resulting work factor of $\approx 2^{230}$ steps, or

$$T^{\omega} \approx 2^{222}$$

encryptions (using a rough estimation of 2^8 computations for an AES encryption), is much higher than that of exhaustive search. It appears that, similar to XL, this technique also fails to break AES-128 using equations over GF(2).

Attempting the same evaluation for AES-256 gives $\frac{R+R'+R''}{T-T'} = 1.006$ for $\rho = 2$ and $P = 8$, for a complexity of

$$T^\omega \approx 2^{247};$$

which is slightly faster than exhaustive search.

Note that we are using $r = 24$ and $t = 81$ in these calculations. However, it is possible to obtain cubic equations on the S-box. Simulations performed by [16] show that we can achieve $r = 471, t = 697$ and $t' = 242$ with cubic equations. Then for AES-256 with $\rho = 2$ and $P = 5$ we have $\frac{R+R'+R''}{T-T'} = 1.0005$ and a complexity of about

$$T^\omega \approx 2^{195}.$$

This is calculated using the current best known value of $\omega = 2.3766$ but even if we use $\omega = 3$ (the usual Gaussian reduction exponent), we still achieve a complexity of 2^{242}. Thus, if the XSL attack works as well as expected over GF(2), it will break AES-256.

11.4.2 AES over $\mathcal{F}$

In [41] the authors make comparisons between their $\mathcal{F}$-system and the GF(2)-system of [15]. The following table summarises their results:

Parameter	Symbol	GF(2)-value	$\mathcal{F}$-value
Field		GF(2)	GF(2^8)
Block size		128	128
Key size		128	128
S-box equations	r	24	24
S-box terms	t	81	41
Key schedule S-boxes	$D+E$	41	41
Total S-boxes	S	201	201
Key variables	S_k	704	704
Independent key variables	L_k	448	448

Using these $\mathcal{F}$-parameter values and smallest P value, it was found that, for $P = 3$:

$$\begin{aligned} R &= 85.19 \times 10^9, \\ R' &= 8.18 \times 10^9, \\ R'' &= 2.97 \times 10^9, \\ \Rightarrow R + R' + R'' &= 95.18 \times 10^9, \text{ and} \\ T &= 91.94 \times 10^9 \approx 2^{36}. \end{aligned}$$

This indicates that there are more equations than terms and so might suggest that, if almost all of these equations are linearly independent, the complexity of the XSL attack on AES-128 over $\mathcal{F}$ will be comparable to a Gaussian elimination on 2^{36} variables. This would give an effort of about $(2^{36})^{\omega}$ $\mathcal{F}$-operations. Using a rough equivalence of 2^8 $\mathcal{F}$-operations to an AES encryption [41], we find that $WF \approx 2^{100}$ for $\omega = 3$ and possibly even $WF \approx 2^{79}$ (!) for the optimistic $\omega = 2.3766$; a huge improvement over the 2^{222} work factor for the GF(2)-system.

11.5 Comments on the XSL Technique

Although this attack in 2^{79} is currently infeasible, it does raise concerns over the future security of the AES. Several criticisms about the effectiveness of the XSL technique, however, have arisen since it was introduced in 2002, suggesting that the method may not work. These criticisms generally stem from similar criticisms of XL.

11.5.1 Solutions at Infinity

In his paper on the effectiveness of the XL technique, T. Moh [36] gives a sufficient condition for XL to succeed:

Let $\ell_1, \ldots, \ell_m$ be a system of equations in the variables $x_1, \ldots, x_n$ over a field K. Let $\mathcal{I}_\infty = (\ell_1, \ldots, \ell_m) \subset K[x_1, \ldots, x_n]$ be the ideal generated by $\ell_1, \ldots, \ell_m$ and $\mathcal{I}_\infty^h$ be the *homogenisation* of $\mathcal{I}_\infty$, i.e., $\mathcal{I}_\infty^h = (\ell_1^h, \ldots, \ell_m^h) \subset K[x_0, \ldots, x_n]$ where $\ell_i^h = x_0^{d_i} \ell_i(x_1/x_0, \ldots, x_n/x_0)$ with $d_i = \deg\ \ell_i$ and $x_0 \in K$ a new variable.

Proposition 1. If the solution set of $(\ell_1, \ldots, \ell_m)$ is 0-dimensional and the solution set in the projective space at infinity of $(\ell_1^h, \ldots, \ell_m^h)$ is 0-dimensional or empty, then the XL technique can be solved for D large enough.
Proof. See [36].

We see from the above proposition, that if the projective variety of the solutions has a subspace of strictly positive dimension at infinity, then the XL technique will not work. Courtois and Patarin [17] acknowledge this idea by noting that the well-known trick of adding the equations of the field $x_i^q = x_i$ over GF(q) will make the component at infinity empty such that the XL technique *will* work (see the brief description of XLF in §10.4).

On his website entitled "AES is NOT broken" [37], Moh, referring to the XSL technique applied to the Murphy-Robshaw $\mathcal{F}$-system, states that "this trick can not be used for the AES situation, since the corresponding equations would be $x_i^{256} + x_i = 0$, the degrees 256 would be too high for practical purpose."

This suggests that the attack may not work for the Murphy-Robshaw system. His argument, however, is void since the equations $x_i^{256} + x_i = 0$ are in fact already indirectly included. For each variable x a **separate variable** exists for each of the following powers: $x, x^2, x^4 \ldots, x^{128}$. Then using the quadratic equations $(x)^2 = x^2, (x^2)^2 = x^4, \ldots, (x^{128})^2 = x$, we find that we have the desired equation $x^{256} = x$. This excludes all unwanted solutions in extension fields, and the projective space at infinity, that could prevent the XSL technique from working.

Furthermore, Yang and Chen [54] claim that the above proposition does not even apply to XSL, since the entire ideal $\mathcal{I} = (\ell_1, \ldots, \ell_m, p_1, p_2, \ldots, p_\kappa)$ is not used (where the p_i are extra polynomial equations added by the attacker).

11.5.2 Number of Linearly Independent Equations

In 2000, Moh [36] proved (falsely) that there are not enough independent equations produced by the XL technique. He was able to show that $Free/R \approx \frac{(n+D)(n+D-1)}{D(D-1)m} = \omega$ and that obviously $\omega \to \frac{1}{m}$ as $D \to \infty$, indicating that most of the equations produced are actually linearly dependent. However, the assumption that D is much larger than the number of variables n is false, and therefore invalidates his findings. In fact, if we assume that $D \approx \frac{n}{\sqrt{m}}$, as in §10.3.2, then we find $\omega \approx 1$.

Don Coppersmith [46] once posted this comment on XSL: "I believe that the Courtois-Pieprzyk work is flawed. They overcount the number of linearly independent equations. The result is that they do not in fact have enough linear equations to solve the system, and the method does not break Rijndael."

Coppersmith claimed that there is a problem evident in the T'-method of §11.2. He suggested that any of the $t'[t^P - (t-r)^P]\binom{S}{P}$ equations produced by multiplying a basic equation by an S-box monomial, are already contained within the R equations, and so can't be counted again.

This, in fact, is not the case, and Coppersmith later revoked his comments, acknowledging that he had written them **before** he actually understood the full XSL and T'-method.

Nevertheless, there is still debate as to the problem of the number of linearly independent equations produced by XSL. The technique is an ad-hoc method, based on a number of heuristic arguments, and this makes it difficult to formally analyse the algorithm's practicability. Coppersmith [12] suggests that there is still very little known about the problem.

11.5.3 Working Examples

A further criticism of XSL is that the technique has not been shown to work on even a modest sized system of equations. Schneier [46], in his Crypto-Gram newsletter, is quoted as saying: "I can say with certainty that no one knows for certain if XSL can break Rijndael or Serpent or anything else. Actually, I can say something stronger: no one has produced an actual demonstration of XSL breaking even a simplified version of Rijndael or Serpent or anything else. This makes a lot of people skeptical."

On the other hand, we can also say that no one knows for certain that the XSL technique *cannot* break the AES, since at present, we do not have the computer resources to implement the attack. If the attack on the AES indeed does work, and is as effective as claimed, we could very well see implementations on real-world systems in around 10 years time [46].

Chapter 12

Conclusion

In this paper we have studied the most prominent attacks on the AES to date. We have seen that the AES resists classic attacks on block ciphers such as linear and differential cryptanalysis, interpolation attacks, and slide attacks.

The most powerful cryptanalysis of the AES to date is the multiset attack. We have demonstrated extensions of up to 7, 8 and 9 rounds. While these attacks are unsettling, they require an extremely large number of chosen plaintexts ($2^{128} - 2^{119}$ in some cases) and are, for the most part, computationally infeasible. Moreover, they have no practical significance to the full cipher of 10-14 rounds. We conclude that these attacks do not compromise the security of AES-encrypted data in any way.

We have shown that an AES encryption can be expressed (quite elegantly) in the form of a generalised continued fraction, with around 2^{26} terms for the AES-128 version. At present, there are no known methods for solving such an equation, and therefore this representation currently has no implications on the security of the AES.

The most important result, we find, is that the XSL technique could potentially break the AES in 2^{100}, or even as little as 2^{79}, using the probably unrealistic complexity exponent, $\omega = 2.3766$. Although this is currently computationally infeasible, the attack may become practical in around 10 years if it is as efficient as claimed. However, the technique is widely criticised and many believe that it does not work. More research into the XSL technique is required.

It has been predicted that the AES will remain secure for at least 30 years. Based on our research, it is our standpoint that, provided there are no unexpected computing or cryptographic advancements, the AES will be secure for at least 10 of those years. However, we should not be complacent. The XSL attack in particular, we feel is cause for concern. Even if the attack turns out not to be faster than exhaustive search, the AES still has a very simple algebraic structure, and this could lead to further improvements, or even more interesting representations and attacks. Thus, we are of the opinion that the AES should *eventually* be replaced, simply as a precautionary measure. However, at present there is certainly no cause for urgency.

Chapter 13

Acknowledgements

First and foremost, I would like to thank my family, David, Julie and Lee, for their undying support, and for putting up with me through thick and thin; I know I have not been the easiest to get along with. Thankyou to all my good friends, band members, and uni mates for their support, and understanding of my time constraints and commitment to this thesis, and this year. To my supervisor, Bob, I thankyou for your comments and suggestions; our meetings were brief but helpful. Finally, I would like to thank Mikey and Jenny for their encouragement, compassion and friendship.

Bibliography

[1] "Advanced Encryption Standard (AES)", *Federal Information Processing Standard Publication 197, NIST*, available from:
`http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

[2] "New Attacks on AES / Rijndael", *Independent AES Security Observatory, maintained by N. Courtois, June 14, 2005*, website:
`http://www.cryptosystem.net/aes/`

[3] E. Berlekamp: "Algebraic Coding Theory", *McGraw-Hill Inc., New York, NY, 1968*

[4] E. Biham: "New Types of Cryptanalytic Attacks Using Related Keys", *Eurocrypt 1993, Springer LNCS 765, pp. 398–409*

[5] E. Biham, A. Biryukov, A. Shamir: "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials", *Eurocrypt 1999, Springer LNCS 1592, pp. 12–23*

[6] E. Biham, A. Shamir: "Differential Cryptanalysis of DES-like Cryptosystems", *Crypto 1990, Springer LNCS 537, pp. 2–21*

[7] A. Biryukov: "The Boomerang Attack on 5 and 6-round Reduced AES", *Fourth AES Conference (AES4), NIST*, available from:
`http://www.cosic.esat.kuleuven.be/publications/article-206.pdf`

[8] A. Biryukov: "Multiset Attack", available from:
`http://www.esat.kuleuven.be/~abiryuko/Enc/b.pdf`

[9] A. Biryukov, D. Wagner: "Slide Attacks", *Fast Software Encryption 1999, Springer LNCS 1636, pp. 245–259*

[10] "Birthday paradox", *Wikipedia online encyclopedia*, available from:
`http://en.wikipedia.org/wiki/Birthday_paradox`

[11] J. Cheon, M. Kim, K. Kim, J.-Y. Lee, S. Kang: "Improved Impossible Differential Cryptanalysis of Rijndael and Crypton", *Information Security and Cryptology 2001, Springer LNCS 2288, pp. 39–49*

[12] Don Coppersmith, Personal Communication, *T.J. Watson Research Center, IBM Corporation, 2005*

[13] N. Courtois: "Algebraic Attacks over GF(2^k), Application to HFE Challenge 2 and Sflash-v2", *Public Key Cryptography 2004, Springer LNCS 2947, pp. 201–217*

[14] N. Courtois, A. Klimov, J. Patarin, A. Shamir: "Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations", *Eurocrypt 2000, Springer LNCS 1807, pp. 392–407*

[15] N. Courtois, J. Pieprzyk: "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations", *Asiacrypt 2002, Springer LNCS 2501, pp. 267–287*

[16] N. Courtois, J. Pieprzyk: "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations" (Extended version), *IACR ePrint Archive, April 2002*, available from:
`http://eprint.iacr.org/2002/044.pdf`

[17] N. Courtois, J. Patarin: "About the XL Algorithm over GF(2)", *The Cryptographers' Track at the RSA Conference 2003, Springer LNCS 2612, pp. 141–157*

[18] J. Daemen, L. Knudsen, V. Rijmen: "The Block Cipher Square", *Fast Software Encryption 1997, Springer LNCS 1267, pp. 149–165*

[19] J. Daemen, V. Rijmen: "The Rijndael Block Cipher - AES Proposal", available from:
`http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf`

[20] J. Daemen, V. Rijmen: "The Wide Trail Design Strategy", *Cryptography and Coding 2001, Springer LNCS 2260, pp. 222–238*

[21] J. Daemen, V. Rijmen: "Security of a Wide Trail Design", *Indocrypt 2002, Springer LNCS 2551, pp. 1–11*

[22] C. Diem: "The XL-Algorithm and a Conjecture from Commutative Algebra", *Asiacrypt 2004, Springer LNCS 3329, pp. 323–337*

[23] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, D. Whiting: "Improved Cryptanalysis of Rijndael", *Fast Software Encryption 2000, Springer LNCS 1978, pp. 213–230*

[24] N. Ferguson, R. Schroeppel, D. Whiting: "A simple algebraic representation of Rijndael", *Selected Areas in Cryptography 2001, Springer LNCS 2259, pp. 103–111*

[25] M. Garey, D. Johnson: "Computers and Intractibility : A Guide to the Theory of NP-completeness", *W. H. Freeman, San Fransisco, 1979*

[26] "Gaussian elimination", *Wikipedia online encyclopedia*, available from:
`http://en.wikipedia.org/wiki/Gaussian_elimination`

[27] H. Gilbert, M. Minier: "A collision attack on 7 rounds of Rijndael", *Third AES Conference, NIST*, available from:
`http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/papers/11-hgilbert.pdf`

[28] G. Jakimoski, Y. Desmedt: "Related-Key Differential Cryptanalysis of 192-bit Key AES Variants", *Selected Areas in Cryptography 2003, Springer LNCS 3006, pp. 208–221*

[29] T. Jakobsen, L. Knudsen: "The Interpolation Attack on Block Ciphers", *Fast Software Encryption 1997, Springer LNCS 1267, pp. 28–40*

[30] A. Kipnis, A. Shamir: "Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization", *Crypto 1999, Springer LNCS 1666, pp. 19–30*

[31] L. Knudsen: "Truncated and Higher Order Differentials", *Fast Software Encryption 1994, Springer LNCS 1008, pp. 196–211*

[32] S. Lucks: "Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys", *Third AES Conference (AES3), NIST*, available from:
`http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/papers/04-slucks.pdf`

[33] U. Manber: "Introduction to Algorithms: A Creative Approach", *Addison-Wesley Publishing Company Inc., Boston, MA, 1989*

[34] "Matrix multiplication", *Wikipedia online encyclopedia*, available from:
`http://en.wikipedia.org/wiki/Matrix_multiplication`

[35] M. Matsui: "Linear cryptanalysis method for DES cipher", *Eurocrypt 1993, Springer LNCS 765, pp. 386–397*

[36] T. Moh: "On The Method of "XL" And Its Inefficiency to TTM", *IACR ePrint Archive, January 2000*, available from:
`http://eprint.iacr.org/2001/047.ps`

[37] T. Moh: "On The Courtois-Pieprzyk's Attack on Rijndael", *September 18, 2002*, website:
`http://www.usdsi.com/aes.html`

[38] "Moore's law", *Wikipedia online encyclopedia*, available from:
`http://en.wikipedia.org/wiki/Moore's_law`

[39] S. Murphy, M. Robshaw: "New Observations on Rijndael", *Preliminary draft, August 7, 2000*, available from:
`http://www.isg.rhul.ac.uk/∼sean/rijn_newobs.pdf`

[40] S. Murphy, M. Robshaw: "Essential Algebraic Structure within the AES", *Crypto 2002, Springer LNCS 2442, pp. 1–16*

[41] S. Murphy, M. Robshaw: "Comments on the Security of the AES and the XSL Technique", *Nessie report*, available from:
`https://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/Xslbes8_Ness.pdf`

[42] United States National Security Agency, website:
`http://www.nsa.gov`

[43] H. Nover: "Algebraic Cryptanalysis of AES: An Overview", available from:
`http://www.math.wisc.edu/∼boston/nover.pdf`

[44] "OBM Guidance to Federal Agencies on Data Availability and Encryption", *NIST Encryption Guidance Policy*, available from:
`http://csrc.nist.gov/policies/ombencryption-guidance.pdf`

[45] "Quantum computer", *Wikipedia online encyplopedia*, available from:
`http://en.wikipedia.org/wiki/Quantum_computing`

[46] B. Schneier: Crypto-Gram Newsletter, *October 15, 2002*, available from: `http://www.schneier.com/crypto-gram-0210.html`

[47] K. Schramm, T. Wollinger, C. Paar: "A New Class of Collision Attacks and its Application to DES", *Fast Software Encryption 2003, Springer LNCS 2887, pp. 206–222*

[48] A.J.M. Segers: "Algebraic Attacks from a Gröbner Basis Perspective" (Masters Thesis), available from: `http://www.win.tue.nl/~henkvt/images/ReportSegersGB2-11-04.pdf`

[49] C. Shannon: "Communication Theory of Secrecy Systems", available from: `http://www.cs.ucla.edu/~jkong/research/security/shannon1949.pdf`

[50] "TOP500 Supercomputer Sites", *accessed October, 2005*, website: `http://www.top500.org/`

[51] D. Wagner: "The boomerang attack", *Fast Software Encryption 1999, Springer LNCS 1636, pp. 156–170*

[52] R. Weinmann: "Evaluating Algebraic Attacks on the AES", available from: `http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/weinmann/Diplomarbeit.pdf`

[53] B.-Y. Yang, J.-M. Chen: "Theoretical Analysis of XL over Small Fields", *Information Security and Privacy 2004, Springer LNCS 3108, pp. 277–288*

[54] B.-Y. Yang, J.-M. Chen: "All in the XL Family: Theory and Practice", *Information Security and Cryptology 2004, Springer LNCS 3506, pp. 67–86*

[55] B.-Y. Yang, J.M. Chen, N. Courtois: "On Asymptotic Security Estimates in XL and Gröbner Bases-Related Algebraic Cryptanalysis", *Information and Communications Security 2004, Springer LNCS 3269, pp. 401–413*

Appendix A

Mini-Example of XL

To understand how exactly the XL algorithm (§10.3.1) works, let's look at a toy example: Consider the problem of solving the following system of 2 homogeneous quadratic equations in 2 unknowns:

$$x_1^2 + \mu x_1 x_2 = \alpha, \tag{A.1}$$

$$x_2^2 + \nu x_1 x_2 = \beta, \tag{A.2}$$

where μ, ν, α and β are known constants with $\mu \neq 0$.

For $D = 4$, in Step 1 we need only multiply (A.1) and (A.2) by all possible monomials of degree 2: $x_1^2, x_2^2, x_1x_2 \in \mathbf{x}^2$. Therefore we get

$$x_1^4 + \mu x_1^3 x_2 = \alpha x_1^2, \tag{A.3}$$

$$x_1^2 x_2^2 + \nu x_1^3 x_2 = \beta x_1^2, \tag{A.4}$$

$$x_1^2 x_2^2 + \mu x_1 x_2^3 = \alpha x_2^2, \tag{A.5}$$

$$x_2^4 + \nu x_1 x_2^3 = \beta x_2^2, \tag{A.6}$$

$$x_1^3 x_2 + \mu x_1^2 x_2^2 = \alpha x_1 x_2, \tag{A.7}$$

$$x_1 x_2^3 + \nu x_1^2 x_2^2 = \beta x_1 x_2. \tag{A.8}$$

Now for Step 2: Elimination of x_2.
Using (A.1): $x_1x_2 = \frac{\alpha}{\mu} - \frac{1}{\mu}x_1^2$;
Using (A.2): $x_2^2 = (\beta - \frac{\alpha\nu}{\mu}) + \frac{\nu}{\mu}x_1^2$;
Using (A.3): $x_1^3x_2 = \frac{\alpha}{\mu}x_1^2 - \frac{1}{\mu}x_1^4$;
Using (A.4): $x_1^2x_2^2 = (\beta - \frac{\alpha\nu}{\mu})x_1^2 + \frac{\nu}{\mu}x_1^4$;
Using (A.8): $x_1x_2^3 = \frac{\alpha\beta}{\mu} + (\frac{\alpha\nu^2}{\mu} - \beta\nu - \frac{\beta}{\mu})x_1^2 - \frac{\nu^2}{\mu}x_1^4$;
Using (A.6): $x_2^4 = (\beta^2 - \frac{2\alpha\beta\nu}{\mu}) + (\frac{2\nu\beta}{\mu} + \beta\nu^2 - \frac{\alpha\nu^2}{\mu})x_1^2 + \frac{\nu^3}{\mu}x_1^4$.
Finally, using (A.5) we get a univariate equation in x_1:

$$\alpha^2 + (\alpha\mu\nu - \beta\mu^2 - 2\alpha)x_1^2 + (1 - \mu\nu)x_1^4 = 0.$$

Appendix B

Mini-Example of the "T′ Method"

Consider a small system with $n = 5$ variables, and thus $T = 16$ and $T' = 10$. We look at a "toy example" of the T' method of §11.2. We start with a random system over GF(2) that has exactly one solution, with $\mathit{Free} > T - T'$ and with 2 exceeding equations, i.e., $\mathit{Free} = T - T' + 2$. Here is a system in which $\mathcal{T}'$ is defined with respect to x_1:

$$\begin{cases} x_3x_2 = x_1x_3 + x_2, \\ x_3x_4 = x_1x_4 + x_1x_5 + x_5, \\ x_3x_5 = x_1x_5 + x_4 + 1, \\ x_2x_4 = x_1x_3 + x_1x_5 + 1, \\ x_2x_5 = x_1x_3 + x_1x_2 + x_3 + x_4, \\ x_4x_5 = x_1x_2 + x_1x_5 + x_2 + 1, \\ 0 = x_1x_3 + x_1x_4 + x_1 + x_5, \\ 1 = x_1x_4 + x_1x_5 + x_1 + x_5. \end{cases}$$

We have the same system, with $\mathcal{T}'$ now defined for x_2:

$$\begin{cases} x_1x_3 = x_3x_2 + x_2, \\ x_1x_4 = x_3x_2 + x_2 + x_1 + x_5, \\ x_1x_5 = x_2x_4 + x_3x_2 + x_2 + 1, \\ x_3x_5 = x_2x_4 + x_3x_2 + x_2 + 1 + x_4 + 1, \\ x_3x_4 = x_2x_4 + x_1 + 1, \\ x_4x_5 = x_1x_2 + x_2x_4 + x_3x_2, \\ 0 = x_1x_2 + x_2x_5 + x_3x_2 + x_2 + x_3 + x_4, \\ 0 = x_2x_4. \end{cases}$$

We have $\mathit{rank} = 8$. Now multiply the two exceeding equations of the first version of the system by x_1:

$$\begin{cases} 0 = x_1x_3 + x_1x_4 + x_1 + x_1x_5, \\ 0 = x_1x_4. \end{cases}$$

We have $rank = 10$ since these equations are linearly independent from the others.

We rewrite these equations, using the second system, only with terms that can be multiplied by x_2. Now we have 4 exceeding equations for the second system (two old and two new):

$$\begin{cases} 0 = x_1x_2 + x_2x_5 + x_3x_2 + x_2 + x_3 + x_4, \\ 0 = x_2x_4, \\ 0 = x_2x_4 + x_3x_2 + x_5 + x_2 + 1, \\ 0 = x_3x_2 + x_2 + x_1 + x_5. \end{cases}$$

We multiply these four equations by x_2:

$$\begin{cases} 0 = x_1x_2 + x_2x_5 + x_2x_4 + x_2, \\ 0 = x_2x_4, \\ 0 = x_2x_4 + x_3x_2 + x_5x_2, \\ 0 = x_3x_2 + x_2 + x_1x_2 + x_2x_5. \end{cases}$$

Unfortunately, we see that the second equation is invariant under this transformation. Still, we get three new linearly independent equations. We have $rank = 13$.

We rewrite, using the first system, the three new equations with terms that can be multiplied by x_1:

$$\begin{cases} 1 = x_1x_5 + x_2 + x_3 + x_4, \\ 1 = x_1x_2 + x_1x_3 + x_1x_5 + x_2 + x_3 + x_4, \\ 0 = x_3 + x_4. \end{cases}$$

We still have $rank = 13$. Then we multiply the three new equations by x_1:

$$\begin{cases} x_1 = x_1x_5 + x_1x_2 + x_1x_3 + x_1x_4, \\ x_1 = x_1x_5 + x_1x_4, \\ 0 = x_1x_3 + x_1x_4. \end{cases}$$

We get one more linearly independent equation (the two other are redundant), therefore we now have $rank = 14$. Now we rewrite the first equation with terms that can be multiplied by x_2:

$$0 = x_1x_2 + x_2x_4 + x_3x_2 + x_1 + x_2 + x_5.$$

We still have $rank = 14$. Then we multiply the new equation by x_2:

$$0 = x_2x_4 + x_3x_2 + x_2x_5 + x_2.$$

We get another new linearly independent equation, and thus $rank = 15$. This is the maximum rank achievable, there are 15 non-zero monomials here, and $rank = 16$ can only be achieved for a system that is contradictory.