

Yet Another Algebraic Cryptanalysis of Small Scale Variants of AES

Marek Bielik¹^a, Martin Jureček¹^b, Olha Jurečková¹^c and Róbert Lórencz¹^d

¹*Department of Information Security, Faculty of Information Technology, Czech Technical University in Prague
mail@marek.onl, {martin.jurecek, jurecolh, robert.lorencz}@fit.cvut.cz*

Keywords: Small Scale Variants of AES, Algebraic Cryptanalysis, Gröbner Bases.

Abstract: This work presents new advances in algebraic cryptanalysis of small scale derivatives of AES. We model the cipher as a system of polynomial equations over $\text{GF}(2)$, which involves only the variables of the initial key, and we subsequently attempt to solve this system using Gröbner bases. We show, for example, that one of the attacks can recover the secret key for one round of AES-128 under one minute on a contemporary CPU. This attack requires only two known plaintexts and their corresponding ciphertexts. We also compare the performance of Gröbner bases to a SAT solver, and provide an insight into the propagation of diffusion within the cipher.

1 INTRODUCTION

The original name of the cipher for the Advanced Encryption Standard (AES) is Rijndael, based on the names of two cryptographers—Joan Daemen and Vincent Rijmen—who originally designed the cipher. In 1997, the U.S. National Institute of Standards and Technology (NIST) announced the development of AES and subsequently organized an open competition, which the Rijndael cipher won. NIST published the cipher as the Federal Information Processing Standard (FIPS) 197 (Pub, 2001) in 2001.


Algebraic cryptanalysis (AC) is an area of cryptanalysis that has gained much attention in recent years (Bard, 2009). The principle of AC consists in transferring the problem of breaking the cryptosystem to the problem of solving a system of multivariate polynomial equations over a finite field that belongs to the set of NP-complete problems. The process of AC is divided into the following two steps. The first step consists of using the cipher’s structure and supplemental information to create a system of equations that describe the behavior of the cipher for a specific case. Several papers (Cid et al., 2005), (Simmons, 2009) present approaches for constructing polynomial equations with auxiliary variables for AES. The paper (Bulygin and Brickenstein, 2010a) presents a method for obtaining equations in key variables only, which is


based on Gröbner bases. In Section 3.2.4, we present another approach for obtaining polynomial equations that contain only the variables of the initial key, which is based on gradual substitution.


The second step of AC involves solving the polynomial system to derive the secret key. While the method for deriving the system of equations depends on the cipher, the method for solving the system may be independent of the cipher. In our work, we leverage the fact that the derived equation systems contain only the variables of the initial key, and we present some reduction techniques for reducing the computational complexity of solving the polynomial systems.


Several previous studies have dealt with algebraic cryptanalysis of small scale variants of AES. In (Courtois and Pieprzyk, 2002), the authors described AES as a system of overdefined sparse quadratic equations over $\text{GF}(2)$, and proposed an XSL attack for the family of XSL-ciphers to which AES belongs to. The XSL algorithm was later analyzed concerning AES in (Cid and Leurent, 2005). The work (Bulygin and Brickenstein, 2010a) also presented methods for solving polynomial systems derived from AES using Gröbner bases. The interpretation of AES as a system of equations over $\text{GF}(2^8)$ is presented in (Murphy and Robshaw, 2002). The work (Nover, 2005) reviewed different techniques for solving systems of multivariate quadratic equations over arbitrary fields, such as relinearization and XL algorithm, that were used on equations derived from AES.

We begin our work by a brief discussion of

^a <https://orcid.org/0000-0002-9426-8467>

^b <https://orcid.org/0000-0002-6546-8953>

^c <https://orcid.org/0000-0002-8858-4826>

^d <https://orcid.org/0000-0001-5444-8511>

Gröbner bases, and we show how these can be used to solve systems of multivariate non-linear polynomial equations. In the third section, we derive multivariate non-linear polynomial systems over GF(2) for small scale variants of AES. We will eliminate all auxiliary variables by a gradual substitution so that the polynomial systems will contain only the variables describing the secret key. The elimination will make the polynomial systems fully dependent on the provided pairs of plaintext and ciphertext, which will allow us to apply the reductions for faster solving.

The fourth section discusses the results of our experiments. We demonstrate the current capabilities of Gröbner bases in solving the polynomial systems described in the third section, and we compare their performance to a SAT solver. We show how the performance of Gröbner bases and the SAT solver can be increased using several pairs of plaintexts and their corresponding ciphertexts. We also discuss the progress of diffusion within the reduced versions of AES. In summary, our main contributions are: (1) the derivation of the equations described in Section 3.2.4; (2) the processing of equations (see Section 4.1) to speed up the computing of Gröbner bases.

2 ALGEBRAIC BACKGROUND

Gröbner bases were introduced by Bruno Buchberger (Buchberger, 2006), who named the concept in honor of his advisor Wolfgang Gröbner (1899–1980). Buchberger also developed the fundamental algorithm for the computation of a Gröbner basis known as Buchberger’s algorithm.

Gröbner bases are nowadays discussed in multiple books including (Becker, 1993) and (Cox, 2015). We will follow these books along the way as we gradually unveil the elegance of Gröbner bases in solving systems of polynomial equations. Further information can be also found in (Adams, 1994) and (Hibi, 2013).

The set of all polynomials in x_1, \dots, x_n with coefficients in a field \mathbb{F} will be denoted $\mathbb{F}[x_1, \dots, x_n]$. When the particular variables are of no relevance, we will denote the set by $\mathbb{F}[\mathbf{x}]$ for short. We will also employ the standard letters x, y and z instead of x_1, x_2 and x_3 when we discuss illustrative polynomials. Univariate polynomials will be denoted by $f(x) \in \mathbb{F}[x]$. We will denote by $\mathcal{M}(x_1, \dots, x_n)$, $\mathcal{M}(\mathbf{x})$ or simply \mathcal{M} , the set of all monomials in the variables x_1, \dots, x_n .

Definition 2.1. Let $h = \sum c_\alpha x^\alpha \in \mathbb{F}[\mathbf{x}] \setminus \{0\}$ be a nonzero polynomial, F a subset of $\mathbb{F}[\mathbf{x}] \setminus \{0\}$ and let \succeq be a monomial order on $\mathcal{M}(\mathbf{x})$. The **multi-degree** of h is $\text{multideg}(h) = \max(\alpha \in \mathbb{N}_0^n \mid c_\alpha \neq 0)$. The maximum is taken with respect to \succeq . The

leading coefficient of h is $\text{LC}(h) = c_{\text{multideg}(h)} \in \mathbb{F}$. $\text{LC}(F) = \{\text{LC}(f) \mid f \in F\}$. The **leading monomial** of h is $\text{LM}(h) = x^{\text{multideg}(h)}$. The **leading term** of h is $\text{LT}(h) = \text{LC}(h) \cdot \text{LM}(h)$.

Definition 2.2. Let $\{f_1, \dots, f_s\} \subset \mathbb{F}[\mathbf{x}]$ be a set of polynomials. Then we set

$$\langle f_1, \dots, f_s \rangle = \left\{ \sum_{i=1}^s h_i f_i \mid h_1, \dots, h_s \in \mathbb{F}[\mathbf{x}] \right\}$$

to be an ideal I of $\mathbb{F}[\mathbf{x}]$, where the set $\{f_1, \dots, f_s\}$ is a **basis** of I . We also call $\langle f_1, \dots, f_s \rangle$ the **ideal generated by** $\{f_1, \dots, f_s\}$.

Definition 2.3. Let $I \subseteq \mathbb{F}[\mathbf{x}]$ be an ideal different from $\{0\}$. We denote by $\text{LT}(I) = \{\text{LT}(f) \mid f \in I\}$ the set of leading terms of nonzero elements of I . The **ideal of leading terms** of I , generated by $\text{LT}(I)$, will be denoted by $\langle \text{LT}(I) \rangle$.

Definition 2.4. Fix a monomial order on $\mathcal{M}(\mathbf{x})$. A finite basis $G \subseteq I$ of a nonzero ideal $I \subseteq \mathbb{F}[\mathbf{x}]$ is a **Gröbner basis** if

$$\langle \text{LT}(G) \rangle = \langle \text{LT}(I) \rangle.$$

The definition above says that a set $\{g_1, \dots, g_m\} \subseteq I$ is a Gröbner basis if and only if the leading term of any element of I is divisible by some of the $\text{LT}(g_i)$.

Definition 2.5. Let $G \subseteq I$ be a Gröbner basis of I . We call G a **reduced Gröbner basis** if for all $g \in G$:

- (i) $\text{LC}(g) = 1$.
- (ii) No monomial of g is in $\langle \text{LT}(G \setminus \{g\}) \rangle$.

Most computer algebra systems actually compute reduced Gröbner bases by default. Any ideal has its reduced Gröbner basis and this basis is unique.

Definition 2.6. Let $I = \langle f_1, \dots, f_m \rangle \subseteq \mathbb{F}[x_1, \dots, x_n]$ be an ideal. The l -th **elimination ideal** I_l is the ideal of $\mathbb{F}[x_{l+1}, \dots, x_n]$ given by $I_l = I \cap \mathbb{F}[x_{l+1}, \dots, x_n]$.

Theorem 2.7 (The Elimination Theorem). *Let $G \subseteq I \subseteq \mathbb{F}[x_1, \dots, x_n]$ be a Gröbner basis of I so that $x_1 \succeq_{\text{lex}} x_2 \succeq_{\text{lex}} \dots \succeq_{\text{lex}} x_n$, where \succeq_{lex} is the lexicographic monomial order. Then, for every $0 \leq l < n$, the set $G_l = G \cap \mathbb{F}[x_{l+1}, \dots, x_n]$ is a Gröbner basis of the l -th elimination ideal I_l .*

Proof. See (Cox, 2015, p. 123). □

Definition 2.8. Let $\mathbb{F}_q[x_1, \dots, x_n]$ be a polynomial ring over the finite field \mathbb{F}_q with order $q = p^m$ where p is a prime number and $m \in \mathbb{N}_{>0}$. The **field equations** of \mathbb{F}_q are the polynomials $x_i^q - x_i$ for every $x_i \in \{x_1, \dots, x_n\}$.

Definition 2.9. Let $\{f_1, \dots, f_s\} \subset \mathbb{F}[x_1, \dots, x_n]$ be a set of polynomials and \mathbb{F}^n an affine space. The **affine variety** $V(f_1, \dots, f_s)$ defined by $\{f_1, \dots, f_s\}$ is the set

$$V(f_1, \dots, f_s) = \left\{ \mathbf{a} \in \mathbb{F}^n \mid f_i(\mathbf{a}) = 0 \text{ for all } 1 \leq i \leq s \right\}$$

of all roots of all the polynomials in $\{f_1, \dots, f_s\}$.

Theorem 2.10 (Finiteness Theorem). *Let $f_1, \dots, f_m \in \mathbb{F}[x_1, \dots, x_n]$ be polynomials. If we have $\langle f_1, \dots, f_m \rangle \cap \mathbb{F}[x_i] \neq 0$ for all x_i , then $V(\langle f_1, \dots, f_m \rangle) \subseteq \mathbb{F}^n$ is finite.*

Proof. See (Cox, 2015, p. 252). \square

Considering the Finiteness Theorem above, adding the field equations into our polynomial system ensures that the system will have finitely many solutions.

Theorem 2.11 (Hilbert's Weak Nullstellensatz). *Let $f_1, \dots, f_m \in \mathbb{F}[\mathbf{x}]$ be polynomials. Then the following are equivalent:*

- (i) *There exists an extension field \mathbb{E} of \mathbb{F} and $\mathbf{a} \in \mathbb{E}^n$ such that for all f_i we have $f_i(\mathbf{a}) = 0$.*
- (ii) $1 \notin \langle f_1, \dots, f_m \rangle$.

Proof. See (Becker, 1993, p. 281). \square

Note that $1 \in I \subseteq \mathbb{F}[\mathbf{x}]$, where I is an ideal, means $I = \mathbb{F}[\mathbf{x}]$ since $1h = h$ for all $h \in \mathbb{F}[\mathbf{x}]$. Also note that whenever we have a finite field \mathbb{F} and its extension \mathbb{E} , all elements from \mathbb{F} satisfy all of the field equations of \mathbb{F} and no element in $\mathbb{E} \setminus \mathbb{F}$ satisfies any of these equations. Therefore, if we add the field equations into a polynomial system that consists of polynomials in $\mathbb{F}[\mathbf{x}]$, we restrict our solutions to the field \mathbb{F} . Let us demonstrate this fact in combination with the Hilbert's Weak Nullstellensatz on the following two examples.

Example 2.12. Consider a system of equations where $f_1 = f_2 = f_3 = 0$ are polynomials in $\text{GF}(2)$ with

$$f_1 = x + y + z, \quad f_2 = xy + xz + yz, \quad f_3 = xyz + 1.$$

If we compute the reduced Gröbner basis with $z \succeq_{\text{lex}} y \succeq_{\text{lex}} x$, we get the following polynomials:

$$g_1 = x + y + z, \quad g_2 = y^2 + yz + z^2, \quad g_3 = z^3 + 1.$$

We see that the only solution to the last polynomial is $z = 1$. When we substitute this solution into g_2 , we get $g_2' = y^2 + y + 1$, which has no solution in $\text{GF}(2)$, and therefore the initial polynomial system has no solution in $\text{GF}(2)$ either. Since g_2' is irreducible over $\text{GF}(2)$ we get the extension field $\text{GF}(2)[\alpha]/\langle \alpha^2 + \alpha + 1 \rangle = \text{GF}(2^2)$ with the elements $0, 1, \alpha, \alpha + 1$. If $z = 1$, the polynomial g_2 has two solutions in $\text{GF}(2^2)$, namely $y = \alpha$ and $y = \alpha + 1$, since

$(\alpha + 1)^2 = \alpha$. The polynomial g_1 has then also two solutions in $\text{GF}(2^2)$, namely $x = \alpha$ and $x = \alpha + 1$. All of these solutions also satisfy our initial system f_1, f_2, f_3 . We could also obtain further solutions if we set $z = \alpha$.

Example 2.13. Considering the previous example, if we add the field equations of \mathbb{F} into the system, we get the following reduced Gröbner basis:

$$g_1 = 1.$$

According to the Hilbert's Weak Nullstellensatz, we can already see that the initial polynomial system f_1, f_2, f_3 has no solutions in $\text{GF}(2)$.

One of the fastest and practically implementable algorithms for computing Gröbner bases, i.e., F4, was introduced in (Faugère, 1999). The algorithm is implemented in the computer algebra system Magma (Bosma et al., 1997), which we employ in our experiments.

3 EQUATION SYSTEMS FOR SMALL SCALE AES

This section describes the derivation of multivariate non-linear polynomial systems over $\text{GF}(2)$ for small scale variants of AES.

3.1 Small Scale AES

Before we discuss the scaled-down derivatives of AES, let us try to estimate how long it would take to attack the full AES-128 by brute force. The actual time complexity of guessing a key with 128 bits can be illustrated by a brief thought experiment.

Suppose we are in possession of a computer cluster with ten billion nodes, each of which runs at 3.3 GHz. Also suppose that one use of AES-128 takes only one clock cycle on each node. Say that one year has around $3 \cdot 10^7$ seconds. Our cluster will then go through $3 \cdot 10^7 \cdot 3.3 \cdot 10^9 \cdot 10^{10} \approx 10^{27} \approx 2^{90}$ keys in one year. This means that in the worst case, the total time required to guess the correct key will be around 2^{38} years, which is about 250 billion; while the age of the universe is currently estimated to be around 13.8 billion years.

Now suppose that the average consumption of each node is only 1 W and that 1 kWh of energy costs only 0.01 € (the average price of 1 kWh for European household consumers was around 0.2 € in 2020). This means that the energy cost required for our attack is around $10^{10} \cdot 0.001 \cdot 0.01 \cdot 24 \cdot 365 \cdot 2^{38} \approx 10^{20}$ €.

A quick estimate like this immediately leads to the conclusion that the feasibility of the classic brute-

force approach is beyond reality. This striking infeasibility of attacking the full AES-128 motivated researchers to come up with scaled down versions of the cipher in order to provide manageable insight into its internals. Carlos Cid et al. introduced such versions in (Cid et al., 2005) and (Cid et al., 2006). The reductions emerge naturally and the new cipher can be described by the following parameters:

- (i) the number of rounds n , $1 \leq n \leq 10$;
- (ii) the number of rows r of the state, $r = 1, 2, 4$;
- (iii) the number of columns c of the state, $c = 1, 2, 4$;
- (iv) the number of bits e of the elements of the state, $e = 4, 8$.

We will denote the scaled-down version of AES by $\text{SR}(n, r, c, e)$. This notation is consistent with (Cid et al., 2005) and (Cid et al., 2006). The standard AES-128 can be then defined by $\text{SR}(10, 4, 4, 8)$ with one subtle difference described in the following paragraph.

The last round of AES differs from the previous ones inasmuch as the MixColumns operation is omitted in it. This omission is due to the design of the inverse of AES. The new $\text{SR}(n, r, c, e)$ cipher keeps the MixColumns operation in the last round. This operation is a linear transformation, so the overall complexity of the cryptanalysis of both ciphers remains the same, since a solution of a system of polynomial equations for one cipher would provide a solution for the other cipher. This omission is the only difference between AES-128 and $\text{SR}(10, 4, 4, 8)$.

Let us now go through the scaled-down versions of the actual encryption operations used in $\text{SR}(n, r, c, e)$. The cipher operates over the field $\text{GF}(2^e)$, defined by the quotient ring $\mathbb{F}_2[x]/\langle f(x) \rangle$ where $f(x) = x^4 + x + 1$ when $e = 4$ and $f(x) = x^8 + x^4 + x^3 + x + 1$ when $e = 8$. Note that the polynomial $f(x)$ is irreducible over $\mathbb{F}_2[x]$ in both cases and when $e = 8$, it is identical to the polynomial used in the original AES-128.

The SubBytes operation is also identical to the one used in AES-128 when $e = 8$. When $e = 4$, the operation is a composition of the following two transformations:

- (i) Take the multiplicative inverse in $\text{GF}(2^4)$, the element 0_{16} is mapped to itself.
- (ii) Apply the following affine transformation over $\text{GF}(2^4)$:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}. \quad (3.1)$$

The ShiftRows operation cyclically rotates the row i of the state by i positions, $0 \leq i < r - 1$. Notice that we index the rows from zero so that the first row is always left intact. When $r = 4$, we use the matrix

$$R_4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad (3.2)$$

to rotate a single row. When $r = 2$, the matrix becomes

$$R_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

When $c = 4$, we can use the following expression to model the whole ShiftRows operation

$$\begin{pmatrix} r'_0 \\ r'_1 \\ r'_2 \\ r'_3 \end{pmatrix} = \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & R_4 & 0 & 0 \\ 0 & 0 & R_4^2 & 0 \\ 0 & 0 & 0 & R_4^3 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix}. \quad (3.3)$$

We substitute R_2 instead of R_4 when $r = 2$. When $c = 2$, the expression simply becomes

$$\begin{pmatrix} r'_0 \\ r'_1 \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & R \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} \quad (3.4)$$

where R is either R_4 or R_2 and I is the identity matrix of corresponding size. When $r = 1$ or $c = 1$, the operation has no effect since either R_2 becomes (1) or the matrix from the expression above becomes I .

The MixColumns operation in AES multiplies each column of the state by the polynomial

$$a(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16} \in \text{GF}(2^8)[x].$$

Multiplication by a fixed polynomial modulo another fixed polynomial can be regarded as a linear transformation so that the MixColumns operation can be seen as a linear transformation as well. If we associate the coefficients 03_{16} , 02_{16} and 01_{16} of the polynomial $a(x) \in \text{GF}(2^8)[x]$ with the polynomials $x + 1$, x and 1 , respectively, we can model the MixColumns operation by the following expression

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix} \quad (3.5)$$

for $0 \leq c < 4$, which indexes the columns. When $r = 2$, the operation can be described by the following linear transformation

$$\begin{pmatrix} s'_{0,j} \\ s'_{1,j} \end{pmatrix} = \begin{pmatrix} x+1 & x \\ x & x+1 \end{pmatrix} \begin{pmatrix} s_{0,j} \\ s_{1,j} \end{pmatrix} \quad (3.6)$$

for $0 \leq j < 2$, which indexes the columns, similarly to expression (3.5). When $r = 1$, the matrix defining

the MixColumns operation simply becomes (1), so the operation has no effect.

When $c = 4$, the new cipher uses the same key schedule as in AES-128. For $c = 2$ and $c = 1$, the structure is naturally reduced and depicted in Figure 1, left and right respectively. Similarly to AES-128, the AddRoundKey operation takes in c words of length r . Each word contains the elements of $\text{GF}(2^e)$. These elements are added to the state—each word is added to a column of the state. The RotWord and SubWord operations take in r -tuples containing the elements of $\text{GF}(2^e)$. The round constant array also contains r -tuples, in which the only non-zero element is the first one, namely $x^{j-1} \in \text{GF}(2^e)$ being the powers of $x \in \text{GF}(2^e)$ where j is the round number. Notice that the initial key has rc bits. Also recall that this initial key is added to the plaintext before starting the encryption and generating the subsequent sub-keys, just as in AES-128.

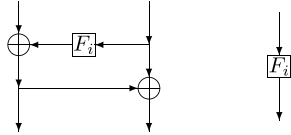


Figure 1: A schematic depiction of the scaled-down key schedule (Cid et al., 2005).

3.2 Equations Systems

Let us now model AES and its scaled-down variants as a system of multivariate polynomial equations over $\text{GF}(2)$. We will focus our attention mainly to $\text{SR}(n, 2, 2, 4)$ and derive a system of equations for this cipher. A solution to this system will provide us with the encryption key. Other scaled-down derivatives can be modeled in the same way, including AES itself. Note that we will use one ciphertext with its corresponding plaintext for our model. Our method therefore comes under the known-plaintext type of cryptanalysis.

3.2.1 Non-linear Equations

Let us start by considering the inversion part of the S-box. We know that $bc = 1$, where $b \in \text{GF}(2^e)$ is the input and $c \in \text{GF}(2^e)$ is the output of the S-box. This equation holds unless $b = 0$, in which case we have $b = c = 0$ and we will say that a 0-inversion has taken place. The probability of a 0-inversion occurring is quite low, namely $\frac{1}{16}$ when $e = 4$ and $\frac{1}{256}$ when $e = 8$, so the probability of no 0-inversion occurring is $1 - \frac{1}{16} = \frac{15}{16}$ and $1 - \frac{1}{256} = \frac{255}{256}$. Notice, however, that these probabilities hold for a single application of the S-box. In $\text{SR}(n, 2, 2, 4)$, there are four applications of the S-box during the encryption in one round,

so the probability of no 0-inversions occurring during the encryption is $(\frac{15}{16})^{4n}$. There are also two applications of the S-box during the key schedule in one round, so the probability of no 0-inversions occurring during the key schedule is $(\frac{15}{16})^{2n}$. We presume statistical independence of the 0-inversions.

The actual occurrence of a 0-inversion either during the encryption or key schedule is deterministically given by the choice of the plaintext and initial key. If we happen to hit a 0-inversion during the generation of the ciphertext, we can simply disregard the current combination of the plaintext and key, and pick another combination. The issue, as we will see later on, is that one of the equations that model the S-box would have to change, and from a cryptanalyst point of view, we would not know which one it would have to be since we do not know the key. For this reason, we will assume that no 0-inversions have occurred for the given plaintext/key combination when we start generating the equations.

We may regard both $b = \sum_{i=0}^3 b_i x^i$ and $c = \sum_{i=0}^3 c_i x^i$ as polynomials in $\text{GF}(2)[x]$. The product bc modulo the polynomial $m(x) = x^4 + x + 1$ is $r(x) = r_3 x^3 + r_2 x^2 + r_1 x + r_0$ where

$$\begin{aligned} r_0 &= b_0 c_0 \oplus b_3 c_1 \oplus b_2 c_2 \oplus b_1 c_3, \\ r_1 &= b_1 c_0 \oplus b_0 c_1 \oplus b_3 c_2 \oplus b_2 c_3 \oplus b_3 c_1 \oplus b_2 c_2 \oplus b_1 c_3, \\ r_2 &= b_2 c_0 \oplus b_1 c_1 \oplus b_0 c_2 \oplus b_3 c_3 \oplus b_3 c_2 \oplus b_2 c_3, \\ r_3 &= b_3 c_0 \oplus b_2 c_1 \oplus b_1 c_2 \oplus b_0 c_3 \oplus b_3 c_3. \end{aligned} \tag{3.7}$$

It is important to note that the coefficients b_i and c_i are the elements of $\text{GF}(2)$. We have $bc = r = 1$. This gives us four multivariate quadratic equations over $\text{GF}(2)$: $r_0 = 1$ and $r_i = 0$ where $i = 1, 2, 3$. These equations are bilinear in the variables b_i and c_i . For $e = 8$, we would have got eight multivariate quadratic equations in the variables b_i and c_i instead of four.

If there was a 0-inversion, either during the encryption or key schedule, the first equation would change to $r_0 = 0$. However as already mentioned, we do not consider this case, since we can detect 0-inversions before we start generating the equations and disregard the plaintext/key combinations that produce them.

Along with these equations, it is possible to obtain further quadratic equations from the relation $bc = 1$. Notice that we also have $bc^2 = c$ and $b^2 c = b$. Let us focus on the first relation and compute the resulting equations. The equations for $b^2 c = b$ can be produced in the same fashion. Since we work over $\text{GF}(2)$, we can write $bc^2 + c = 0$. We have already computed the product bc , so we could just multiply it by c and get the result. This computation would require unnecessary steps as it would lead to many intermediate cu-

bic terms which we would have to cross out before obtaining the final coefficients. We can instead compute the square of c and pre-multiply it by b . We are working over a commutative structure, so the order in which we perform the multiplication is of no relevance. In order to work out the square of c , we can use (3.7) and substitute c for b . We get the polynomial $d = c^2$ where $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$ with

$$d_0 = c_0 \oplus c_2 \quad d_1 = c_2 \quad d_2 = c_1 \oplus c_3 \quad d_3 = c_3.$$

We can now obtain the final result $t = bd + c$ where $t(x) = t_3x^3 + t_2x^2 + t_1x + t_0$ with

$$\begin{aligned} t_0 &= b_0c_0 \oplus b_0c_2 \oplus b_3c_2 \oplus b_2c_1 \oplus b_2c_3 \oplus b_1c_3 \oplus c_1, \\ t_1 &= b_1c_0 \oplus b_1c_2 \oplus b_0c_2 \oplus b_3c_1 \oplus b_3c_3 \oplus b_3c_2 \oplus b_2c_1 \\ &\quad \oplus b_1c_3 \oplus c_1, \\ t_2 &= b_2c_0 \oplus b_2c_2 \oplus b_1c_2 \oplus b_0c_1 \oplus b_0c_3 \oplus b_3c_1 \oplus b_2c_3 \\ &\quad \oplus c_2, \\ t_3 &= b_3c_0 \oplus b_3c_2 \oplus b_2c_2 \oplus b_1c_1 \oplus b_1c_3 \oplus b_0c_3 \oplus b_3c_3 \\ &\quad \oplus c_3. \end{aligned}$$

We know that $t = 0$, so we have four equations $t_i = 0$ for $0 \leq i < 4$. Notice that these equations are quadratic as well. We can obtain reciprocal equations from $b^2c = b$. All of these eight equations are biaffine in the b_i and c_i variables.

It is possible to obtain even more quadratic equations by considering the relations $bc^4 = c^3$ and $b^4c = b^3$. As in the previous case, let us focus on the first relation. We can square d to obtain c^4 and multiply d by c to obtain c^3 . The result will then be $u = bc^4 + c^3 = 0$ where $u(x) = u_3x^3 + u_2x^2 + u_1x + u_0$ with

$$\begin{aligned} u_0 &= b_3c_3 \oplus b_3c_1 \oplus b_2c_3 \oplus b_2c_2 \oplus b_1c_3 \oplus b_0c_3 \oplus b_0c_2 \\ &\quad \oplus b_0c_1 \oplus b_0c_0 \oplus c_3c_1 \oplus c_2c_1 \oplus c_2c_0 \oplus c_0, \\ u_1 &= b_3c_2 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_2 \oplus b_1c_1 \oplus b_1c_0 \oplus b_0c_3 \\ &\quad \oplus b_0c_1 \oplus c_3c_2 \oplus c_2c_0 \oplus c_1c_0 \oplus c_3, \\ u_2 &= b_3c_2 \oplus b_2c_2 \oplus b_2c_1 \oplus b_2c_0 \oplus b_1c_3 \oplus b_1c_1 \oplus b_0c_3 \\ &\quad \oplus b_0c_2 \oplus c_3c_2 \oplus c_3c_1 \oplus c_3c_0 \oplus c_2c_1 \oplus c_2c_0 \\ &\quad \oplus c_1c_0 \oplus c_2, \\ u_3 &= b_3c_2 \oplus b_3c_1 \oplus b_3c_0 \oplus b_2c_3 \oplus b_2c_1 \oplus b_1c_3 \oplus b_1c_2 \\ &\quad \oplus b_0c_3 \oplus c_3c_2 \oplus c_3c_2 \oplus c_3c_1 \oplus c_3 \oplus c_2 \oplus c_1. \end{aligned}$$

We have another four equations $u_i = 0$ for $0 \leq i < 4$. Observe that these equations are still quadratic. We can obtain reciprocal equations from $b^4c = b^3$.

So far, we have derived 20 multivariate quadratic equations from the relation $bc = 1$. A natural question arises whether we have identified all quadratic equations in the b_i and c_i variables. Notice, for example, that we have skipped the relation $bc^3 = c^2$. The reason is that it would produce equations with cubic terms. Relations involving higher powers than c^4 would also

lead to equations with higher than quadratic terms. In fact, the 20 equations we have derived are all the quadratic equations over $\text{GF}(2)$. A further discussion can be found in (Cid et al., 2006, p. 77). As also advised in (Cid et al., 2006, p. 77), we will focus on the first 12 bilinear and biaffine quadratic equations we have obtained and we will omit the remaining eight ones. For $e = 8$, we would have got 40 multivariate quadratic equations in the variables b_i and c_i instead of 20.

3.2.2 Linear Equations

The equations we have derived for the inversion part of the S-box account for the only non-linear equations in the whole system that models the $\text{SR}(n, 2, 2, 4)$ cipher. In fact, the inversion in the AES S-box represents the only non-linear operation in the whole cipher. Let us now derive the linear equations for the remaining transformations in AES.

The affine transformation of the S-box can be expressed directly by (3.1), where the input is the polynomial $c(x)$ from the previous subsection. This gives us four linear equations in the variables c_i . These equations together with the non-linear equations from the previous subsection fully describe a single S-box. Let L_s denote the matrix from (3.1). In order to describe the whole `SubBytes` operation, we can extend the matrix L_s to the whole state array of $\text{SR}(n, 2, 2, 4)$, so we have the matrix

$$L = \begin{pmatrix} L_s & 0 & 0 & 0 \\ 0 & L_s & 0 & 0 \\ 0 & 0 & L_s & 0 \\ 0 & 0 & 0 & L_s \end{pmatrix}.$$

We can also extend the S-box constant vector $(0, 1, 1, 0)^T = \mathbf{6}_{16}$ to the vector $\mathbf{6} = (\mathbf{6}_{16}, \mathbf{6}_{16}, \mathbf{6}_{16}, \mathbf{6}_{16})$ so that we cover the whole state array. We will use \mathbf{b} to denote the input vector of the `SubBytes` operation, and \mathbf{b}^{-1} to denote its output—the vector of the inverted elements in $\text{GF}(2^4)$. Note that each component in these vectors is made of the four coefficients of the polynomials $b(x)$ and $c(x)$, respectively; so we have 12 non-linear equations for each component.

s_0	s_2
s_1	s_3

Figure 2: The state array of the $\text{SR}(n, 2, 2, e)$ cipher.

The actual state array is depicted in Figure 2. We will represent it as the vector $(s_0, s_1, s_2, s_3)^T$. The `ShiftRows` operation can be then described by the

matrix

$$R = \begin{pmatrix} I_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_4 \\ 0 & 0 & I_4 & 0 \\ 0 & I_4 & 0 & 0 \end{pmatrix}$$

where I_4 is the identity matrix of size four. Before we describe the `MixColumns` operation, let us rewrite (3.7) into matrix form:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 \oplus b_3 & b_3 \oplus b_2 & b_2 \oplus b_1 \\ b_2 & b_1 & b_0 \oplus b_3 & b_3 \oplus b_2 \\ b_3 & b_2 & b_1 & b_0 \oplus b_3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

If we substitute the binary values of the coefficients of the polynomials $x + 1$ and x into the matrix in the expression above, we get the matrices

$$\begin{aligned} M_{x+1} &= \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \text{and} \\ M_x &= \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \end{aligned} \quad (3.8)$$

These matrices represent the multiplication by the polynomials $x + 1$ and x modulo the polynomial $x^4 + x + 1$. The `MixColumns` operation, defined by (3.6), can then be expressed by the matrix

$$M = \begin{pmatrix} M_{x+1} & M_x & 0 & 0 \\ M_x & M_{x+1} & 0 & 0 \\ 0 & 0 & M_{x+1} & M_x \\ 0 & 0 & M_x & M_{x+1} \end{pmatrix}.$$

We can now describe one round of $\text{SR}(n, 2, 2, 4)$ by the expression

$$\mathbf{b}_i = MR(L\mathbf{b}_{i-1}^{-1} + \mathbf{6}) + \mathbf{k}_i \quad \text{for } i < 0 \leq n$$

where \mathbf{k}_i is a vector containing 16 binary variables of the round key described in the following subsection and i is the round number. The vector \mathbf{b}_{i-1}^{-1} contains four components—the outputs from the S-boxes—each of which has four binary variables. It is straightforward to check that $R\mathbf{6} = M\mathbf{6} = \mathbf{6}$. We can then write

$$\mathbf{b}_i = MRL\mathbf{b}_{i-1}^{-1} + \mathbf{k}_i + \mathbf{6} \quad \text{for } i < 0 \leq n.$$

The relation above gives 16 linear equations, which represent one round of $\text{SR}(n, 2, 2, 4)$. In addition, we have 12 non-linear equations for each component in \mathbf{b}_{i-1}^{-1} , so in total, we have $16 + 4 \cdot 12 = 64$ equations describing one round of encryption in the $\text{SR}(n, 2, 2, 4)$ cipher. When $i = n$, we have

$$\mathbf{c}_t = MRL\mathbf{b}_{n-1}^{-1} + \mathbf{k}_n + \mathbf{6}$$

where \mathbf{c}_t is the known ciphertext, which is a vector of 16 binary values. We obtain \mathbf{b}_0 by adding the initial unknown key \mathbf{k}_0 to the known plaintext \mathbf{p}_t , so we have

$$\mathbf{b}_0 = \mathbf{p}_t + \mathbf{k}_0.$$

This addition gives us further 16 initial equations where \mathbf{p}_t is a vector of 16 binary values and \mathbf{k}_0 is a vector of 16 binary variables. Our goal is to actually compute the values of \mathbf{k}_0 since this is the user's key. All other variables are auxiliary.

3.2.3 Key Schedule

The generation of round keys for $\text{SR}(n, r, c, e)$ is thoroughly described in Appendix A of (Cid et al., 2005). Let us now describe the equations for $\text{SR}(n, 2, 2, 4)$. Let $\mathbf{k}_i = (k_{i,0}, k_{i,1}, k_{i,2}, k_{i,3})^T \in \text{GF}(2^4)^4$ be the round key of round i . The round key can be then defined by

$$\begin{aligned} \begin{pmatrix} k_{i,2q} \\ k_{i,2q+1} \end{pmatrix} &= \begin{pmatrix} Lk_{i-1,3}^{-1} \\ Lk_{i-1,2}^{-1} \end{pmatrix} + \begin{pmatrix} 6_{16} \\ 6_{16} \end{pmatrix} + \begin{pmatrix} x^{i-1} \\ 0 \end{pmatrix} \\ &+ \sum_{t=0}^q \begin{pmatrix} k_{i-1,2t} \\ k_{i-1,2t+1} \end{pmatrix} \end{aligned} \quad (3.9)$$

for $0 \leq q < 2$ where x^{i-1} is an element of $\text{GF}(2^4)$. This expression gives 16 linear equations for each \mathbf{k}_i . Note that \mathbf{k}_0 is not provided by the user—it is a vector of 16 binary variables that we, as the cryptanalyst, are trying to compute. We also get $2 \cdot 12 = 24$ non-linear equations since the computation of each \mathbf{k}_i requires two applications of the S-box. One round of the key schedule in $\text{SR}(n, 2, 2, 4)$ is then described by 40 equations.

3.2.4 Equations Without Auxiliary Variables

In this section, we propose the derivation of equations that contain only the variables of the initial key. In order to obtain such a system, we can eliminate the auxiliary variables by a gradual substitution of the variables of the initial key since we know that the cipher starts by adding the initial key to the known plaintext. It is straightforward to perform this substitution for the linear equations. For the non-linear equations, which model the S-box, we can leverage Gröbner bases. Consider the four polynomials r_0, \dots, r_3 from (3.7) as polynomials in $\mathbb{F}[c_0, \dots, c_3, b_0, \dots, b_3]$. We see that it is not straightforward to express the output bits c_i in terms of the input bits b_i by ordinary manipulation techniques. If we impose the graded reverse lexicographic block order $\succeq_{\text{grlex,grlex}}$ on $\mathbb{F}[c_0, \dots, c_3, b_0, \dots, b_3]$ with \succeq_{grlex} on both $\mathbb{F}[c_0, \dots, c_3]$ and $\mathbb{F}[b_0, \dots, b_3]$, and compute

the reduced Gröbner basis, we get the following polynomial system:

$$\begin{aligned}
f_1 &= c_0 \oplus b_2 b_1 b_0 \oplus b_3 b_2 b_1 \oplus b_2 b_0 \oplus b_2 b_1 \oplus b_0 \oplus b_1 \\
&\quad \oplus b_2 \oplus b_3, \\
f_2 &= c_1 \oplus b_3 b_1 b_0 \oplus b_1 b_0 \oplus b_2 b_0 \oplus b_2 b_1 \oplus b_3 b_1 \oplus b_3, \\
f_3 &= c_2 \oplus b_3 b_2 b_0 \oplus b_1 b_0 \oplus b_2 b_0 \oplus b_3 b_0 \oplus b_2 \oplus b_3, \\
f_4 &= c_3 \oplus b_3 b_2 b_1 \oplus b_3 b_0 \oplus b_3 b_1 \oplus b_3 b_2 \oplus b_1 \oplus b_2 \\
&\quad \oplus b_3, \\
f_5 &= b_3 b_2 b_1 b_0 \oplus b_2 b_1 b_0 \oplus b_3 b_1 b_0 \oplus b_3 b_2 b_0 \oplus b_3 b_2 b_1 \\
&\quad \oplus b_1 b_0 \oplus b_2 b_0 \oplus b_2 b_1 \oplus b_3 b_0 \oplus b_3 b_1 \\
&\quad \oplus b_3 b_2 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus 1.
\end{aligned}$$

We see that the last polynomial f_5 involves only the variables b_i . Notice that this polynomial is not satisfied only if all $b_i = 0$, and it holds whenever we have at least one $b_i = 1$. Recall that we do not consider 0-inversions. This polynomial is therefore always satisfied, and we can omit it from the system. We also see that in the remaining polynomials, the output variables c_0, \dots, c_3 are expressed solely by the input variables b_i . This allows us to perform the gradual substitution of the unknown variables of the initial key \mathbf{k}_0 throughout the whole polynomial system. Notice that we obtain $|\mathbf{k}_0| = 16$ polynomials after we finish the substitution. We note that the size of the polynomials is close to $2^{|\mathbf{k}_0|-1}$ at full diffusion of the cipher. The diffusion grows rapidly with each round. For example, as our experiments will reveal, the cipher SR($n, 2, 2, 4$) reaches its full diffusion at round $n = 3$. This way of generating the polynomials is therefore suitable only for low values of n . A different method for obtaining polynomials without auxiliary variables is described in (Bulygin and Brickenstein, 2010b).

4 RESULTS OF EXPERIMENTS

The experiments were carried out on GNU/Linux 5.4 running on two Intel® Xeon® Gold 6136 processors with 768 GB DDR4 memory evenly split up into 12 modules. The baseboard was Supermicro X11DPi-NT. The initial polynomial systems containing auxiliary variables were generated by utilizing Martin Albrecht’s implementation of the small scale variants of AES in SageMath 9.1 (The Sage Developers, 2020), which also uses Python 3.7.3 and PolyBoRi (Brickenstein and Dreyer, 2009). The systems were solved in Magma V2.25-5 (Bosma et al., 1997) and CryptoMiniSat (Soos et al., 2009). The source code for the experiments can be found at <https://gitlab.com/upbqdn/yaac>. The generation and preprocessing of the polynomial systems was implemented in paral-

lel utilizing all 24 available cores. Magma, however, was able to solve one system on one core only, so in order to keep the comparison even, we explicitly restricted CryptoMiniSat to one core as well.

As stated in Definition 2.2, we may regard a system of polynomials as a basis of an ideal I . We can then compute the reduced Gröbner basis of I under the lexicographic order, and by applying the Elimination Theorem, we can quickly obtain the solution. We have demonstrated the use of this theorem in Example 2.12, and as we have discussed in the previous section, the solution represents the secret key.

Table 1 shows the results of initial experiments with systems of equations containing auxiliary variables. We generated the systems in SageMath for various versions of SR(n, r, c, e), and we subsequently attempted to solve these systems by the F4 algorithm implemented in Magma and by CryptoMiniSat.

Table 1: Initial experiments with systems containing auxiliary variables.

Cipher	Key bits	Vars	Polys	F4		SAT
				Time	Mem.	Time
SR(1, 2, 2, 4)	16	72	120	1 s	33 MB	2 s
SR(2, 2, 2, 4)	16	128	224	19 s	848 MB	12 s
SR(3, 2, 2, 4)	16	184	328	4 h	76 GB	17 s
SR(4, 2, 2, 4)	16	240	432	—	—	27 s
SR(10, 2, 2, 4)	16	576	1056	—	—	50 s
SR(1, 4, 2, 4)	32	144	240	48 s	981 MB	9 s
SR(2, 4, 2, 4)	32	256	448	—	—	1.5 m
SR(3, 4, 2, 4)	32	368	656	—	—	63 h
SR(1, 2, 4, 4)	32	136	216	3 s	67 MB	11 s
SR(2, 2, 4, 4)	32	240	400	—	—	33 s
SR(3, 2, 4, 4)	32	344	584	—	—	15.5 m
SR(4, 2, 4, 4)	32	448	768	—	—	34 h
SR(1, 4, 4, 4)	64	272	432	—	—	2.5 m
SR(1, 2, 2, 8)	32	144	240	1 m	2.2 GB	22 s
SR(2, 2, 2, 8)	32	256	448	—	—	11.5 m
SR(1, 4, 2, 8)	64	288	480	—	—	41.5 m
SR(1, 2, 4, 8)	64	272	432	—	—	4 m
SR(1, 4, 4, 8)	128	544	864	—	—	—

Since we work over GF(2), the polynomials can be seen as logical formulas in algebraic normal form (ANF). SageMath supports a conversion from ANF to CNF (conjunctive normal form). Formulas in CNF can be passed to CryptoMiniSat and the initial key can be then quickly recovered from the solution. We have included the SAT solver so that we can compare it to the performance of the F4 algorithm and we can see in the table that the solver performs significantly better. The SAT solver also takes a negligible amount of memory, so this value is not stated in the table.

The average number of monomials per polynomial is between 6 and 8 when $e = 4$ and between 18 and 20 when $e = 8$. Both the average and highest degree of the monomials are equal to two, so all polynomials are quadratic or linear, as the case may be.

In our experiments, we do not consider ciphers with $r < 2$ or $c < 2$ as these have the matrices for the operations `MixColumns` and `ShiftRows` reduced to (1). Recall that the dimensions of the state array r and c are restricted to the values 1, 2 and 4; the exponent e can be either 4 or 8; and for the number of rounds n , we have $1 \leq n \leq 10$.

The column named `Vars` contains the number of variables in the whole polynomial system and the column named `Polys` contains the number of polynomials in the system. We measured the runtime and memory consumption only during the solving of the polynomials since the preparation of the system takes only a fraction of the resources relative to solving it.

Recall that the key size for $\text{SR}(n, r, c, e)$ is given by the product rce . Notice that we were not able to compute the solution for even one round of $\text{SR}(n, 4, 4, 8)$, the key size of which is 128 bits. On the other hand, the SAT solver could quickly compute the solution for all ten rounds of $\text{SR}(n, 2, 2, 4)$. We limited the time of each computation to 100 hours. Missing values in the tables denote computations that exceeded this time.

Table 2 contains the results of experiments with systems that contain only the variables of the initial secret key. We eliminated the auxiliary variables by a gradual substitution of the variables of the initial key through the system, starting by adding the known plaintext bits and ending by adding the known ciphertext bits. The time required for this substitution is stated in the column named `PT`. This system always contains k polynomials in k variables where k is the number of the key bits. Since k is the number of variables and we work over $\text{GF}(2)$, k is also the maximal limit of the total degree of the polynomials.

Table 2: Experiments with systems with no auxiliary variables.

Cipher	Key bits	PT ^a	AMP ^b	F4		SAT
				Time	Mem.	Time
SR(1, 2, 2, 4)	16	1 s	20	1 s	33 MB	1 s
SR(2, 2, 2, 4)	16	1 s	2475	2.5 m	4.8 GB	1 m
SR(3, 2, 2, 4)	16	8 s	32784	8.5 m	18.5 GB	13 m
SR(10, 2, 2, 4)	16	2.5 m	32814	9 m	19.5 GB	14 m
SR(1, 4, 2, 4)	32	1 s	37	55 s	1.2 GB	1 s
SR(1, 2, 4, 4)	32	1 s	23	13 s	671 MB	1 s
SR(1, 4, 4, 4)	64	4 s	40	—	—	2 m
SR(1, 2, 2, 8)	32	8 s	314	—	—	1.5 m
SR(1, 4, 2, 8)	64	18 s	567	—	—	33 m
SR(1, 2, 4, 8)	64	14 s	348	—	—	1.5 h

^a Preprocessing Time — the time required to obtain the system

^b Average number of Monomials per Polynomial

All further experiments will be carried out with

systems of polynomials involving only the variables of the initial key. In systems with auxiliary variables, the structure of the polynomial systems derived from different plaintexts remains unchanged. Only the initial and final polynomials that add the bits of the plaintext and ciphertext differ by this bitwise addition. Since we have eliminated the auxiliary variables by a gradual substitution of the initial key bits starting from the initial plaintext addition, each of the k polynomials now depends on the choice of plaintext and its corresponding ciphertext. Since the structure of each polynomial system is now different, the time and memory required for obtaining the solution started to differ as well, especially the time required by the SAT solver. For this reason, all the following tables contain average results of five different runs for each experiment. We can still see that the results for the SAT solver differ across tables for the same experiment, so even more than five runs would be required for further investigation. Nevertheless, we restricted ourselves to such number due to limited time resources.

The column named `AMP` contains the average number of monomials per polynomial in the whole system. We can see that this number grows fast as n increases. The maximal limit of the number of monomials in one polynomial is $2^k - 1$. When $n = 1$ and $e = 4$, the average degree of monomials is 2 and the highest degree is 3. When $n = 2$, the average and highest degrees are 5 and 9, respectively. Note that the average degree has its maximum at $\frac{k}{2}$. We were not able to generate systems with $n > 2$ and $r, c > 2$ for $e = 4$. For $n = 1$ and $e = 8$, the average degree is 4 and the maximal degree is 7. We were not able to generate systems with $e = 8$ and $n > 1$ (recall that we do not consider the cases when $r < 2$ or $c < 2$). We can see in the table that the overall performance is worse compared to the previous table and that the SAT solver still outperforms the F4 algorithm. Moreover, we were able to solve less systems than in the previous experiments.

In the table above, we can see that the the AMP value and the solving time and memory are almost the same for $\text{SR}(3, 2, 2, 4)$ and $\text{SR}(10, 2, 2, 4)$. This means that the full diffusion for $\text{SR}(n, 2, 2, 4)$ is reached in the third round of the cipher and the subsequent rounds do not provide any further security as regards the algebraic cryptanalysis, except for a longer time required for the generation of the polynomial system. This observation is in line with the statements made in (Aumasson, 2019). Table 3 provides a deeper insight into the distribution of monomials in $\text{SR}(3, 2, 2, 4)$.

At full diffusion, the expected degree of monomials should be equal to $\frac{1}{2} \binom{k}{d}$ where k is the number of

Table 3: Distribution of monomials of a given degree in SR(3, 2, 2, 4).

Poly	Number of monomials of the given degree																	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	all
Avg.	0.7	8	59	279	917	2193	4010	5730	6436	5742	4014	2194	909	275	60	8	0.6	32834
Exp.	0.5	8	60	280	910	2184	4004	5720	6435	5720	4004	2184	910	280	60	8	0.5	32768

variables and d is the degree. Since we have SR(3, 2, 2, 4), we get $k = 2 \cdot 2 \cdot 4 = 16$. Recall that we also have k polynomials in the whole system. In Table 3, the expected value is stated in the last row. We see that all the polynomials follow this value very closely, meaning that it is not possible to get much closer to the expected value in the subsequent rounds. For this reason, we do not consider the rounds following after the third one. The table also shows that the average monomial degree is 8 for each polynomial, which is half of the maximal degree, and that no polynomial significantly differs from the expected values for monomial degrees. The second last row shows the average value for all of the polynomials—the average of the whole column above.

The last column contains the number of all monomials in the polynomial. At full diffusion, this number should be equal to $\sum_{d=0}^{16} \frac{1}{2} \binom{k}{d} = \frac{2^{16}}{2} = 32768$ so that every polynomial contains half of all of the possible monomials. We see that the number of monomials is close to the expected value for each of the polynomials as well. We may also be interested in the frequency of the variables in the polynomial system. Considering the full diffusion again, each variable should be contained in half of the monomials in every polynomial, so the expected value is $\frac{2^{16}}{4} = 16384$. In the actual system described in Table 3, the most frequent variable had 16446 occurrences and the least frequent variable had 16393 occurrences, these are aggregated values.

4.1 Reduced Polynomial Systems

Let us now reduce the polynomial systems and see if we can obtain any better results than those in Table 2.

Definition 4.2. Let $V \subseteq \mathbb{F}^n$ be an affine variety. We define

$$I(V) = \left\{ f \in \mathbb{F}[x_1, \dots, x_n] \mid f(\mathbf{a}) = 0 \text{ for all } \mathbf{a} \in V \right\}.$$

Proposition 4.3. If $V \subseteq \mathbb{F}^n$ is an affine variety, then $I(V) \subseteq \mathbb{F}[x_1, \dots, x_n]$ is an ideal. We call $I(V)$ the *ideal of V* .

Proof. See (Cox, 2015, p. 32). \square

Let \mathbf{k} be the initial key of AES or its small scale variant. By Proposition 4.3, we know that $I(\mathbf{k})$ is

an ideal. Now let $\{f_1, \dots, f_k\}$ and $\{g_1, \dots, g_k\}$ be two polynomial systems generated from two different pairs of plaintext and its corresponding ciphertext under the same key \mathbf{k} . Since each $f_i(\mathbf{k}) = 0$ and $g_j(\mathbf{k}) = 0$, we have $I = \langle f_1, \dots, f_k, g_1, \dots, g_k \rangle \subseteq I(\mathbf{k})$. In general, in order to obtain the ideal I , we may combine any number of polynomial systems. We can now compute the Gröbner basis for I and we still get the initial key \mathbf{k} . The ideal I represents an overdefined system for which it could be easier to obtain the solution. We will call one pair of plaintext and its corresponding ciphertext a **PC pair**. In our further experiments, we assume that all PC pairs use the same key.

Table 4: Experiments with two combined systems.

Cipher	Key bits	PT ^a	AMP ^b	F4		SAT
				Time	Mem.	Time
SR(1, 2, 2, 4)	16	1 s	21	1 s	33 MB	1 s
SR(2, 2, 2, 4)	16	2 s	2469	5 s	100 MB	1 m
SR(3, 2, 2, 4)	16	9 s	32798	13 m	19.8 GB	45.5 m
SR(10, 2, 2, 4)	16	3 m	32774	11 m	25.5 GB	31.5 m
SR(1, 4, 2, 4)	32	2 s	37	1 s	33 MB	1 s
SR(2, 4, 2, 4)	32	6 s	33360	—	—	—
SR(1, 2, 4, 4)	32	2 s	23	1 s	33 MB	1 s
SR(2, 2, 4, 4)	32	3 s	6701	—	—	—
SR(1, 4, 4, 4)	64	4 s	39	1 s	33 MB	2 s
SR(1, 2, 2, 8)	32	10 s	316	1 s	33 MB	8 s
SR(1, 4, 2, 8)	64	18 s	568	2 s	33 MB	17 s
SR(1, 2, 4, 8)	64	15 s	348	1 s	33 MB	17 s
SR(1, 4, 4, 8)	128	34 s	599	4 s	33 MB	35 s

^a Preprocessing Time — the time required to obtain the system

^b Average number of Monomials per Polynomial

Table 4 summarizes the experimental results for two combined systems, as described in the previous paragraph. We can see that the results are significantly better compared to Table 2 and that the F4 algorithm often outperforms the SAT solver. We can also see that we are able to solve more polynomial systems and even the system for SR(1, 4, 4, 8) is solved in a few seconds. Recall that we were unable to obtain this solution for systems with auxiliary variables. This practically means that one round of AES-128 provides no security against this attack. We were not able to obtain any solution for SR($n, 4, 4, e$) with $n > 1$

Table 5: Experiments with reduced polynomial systems.

Cipher	Key bits	PT ^a	AMPR ^b	l ^c	F4		SAT
					Time	Mem.	Time
SR(2, 2, 2, 4)	16	5 s	601	1	1 s	33 MB	29 s
SR(2, 2, 2, 4)	16	5 s	519	5	1 s	33 MB	24 s
SR(3, 2, 2, 4)	16	25 s	32592	1	16 m	17.9 GB	37.5 m
SR(3, 2, 2, 4)	16	40 s	32555	5	18 m	23.1 GB	41 m
SR(2, 4, 2, 4)	32	26 s	4938	1	—	—	—
SR(2, 4, 2, 4)	32	1 m	4563	5	—	—	—
SR(2, 2, 4, 4)	32	14 s	3410	1	—	—	1 h 23 m
SR(2, 2, 4, 4)	32	18 s	1192	5	60 m	34.5 GB	50 m

^a Preprocessing Time — the time required to obtain the system

^b Average number of Monomials per Polynomial after Reduction

^c Number of polynomial systems of the reduction set

though. Observe that we used two PC pairs in this scenario. We carried out further experiments with more than two pairs, but we did not obtain any better results. After adding more than five systems, the time required to obtain a solution started increasing.

We note that it would not be possible to combine the systems if we did not eliminate the auxiliary variables. The reason is that the auxiliary variables do not depend on the PC pair—when we use two different PC pairs, we get the same equations, up to the initial additions of the plaintext and ciphertext. On the other hand, when we express the equations only in the variables of the initial key, we get a different system for each PC pair.

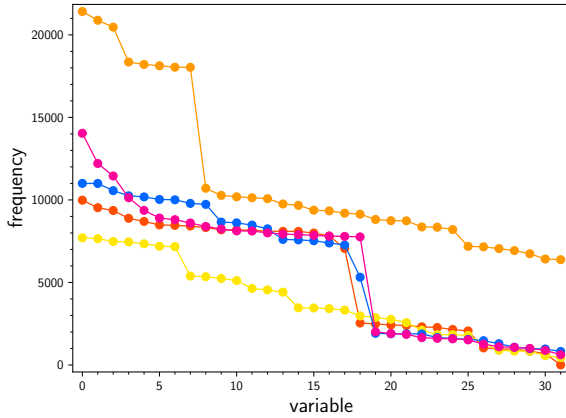
Table 4 shows that the hardest systems to solve were the ones with high AMPR. Let us see if we can reduce this value.

Definition 4.4. Let $f, g \in \mathbb{F}[\mathbf{x}]$ be two polynomials. We define their similarity as $\sigma(f, g) = |M(f) \cap M(g)|$, where $M(h)$ is the set of monomials in h .

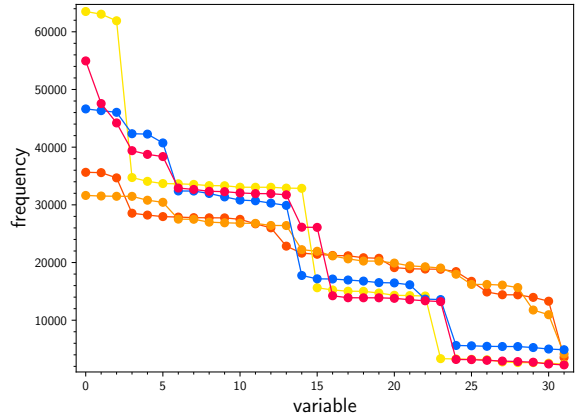
Consider again a polynomial system $F = \{f_1, \dots, f_k\}$ and a set of l polynomial systems $G = \{g_1, \dots, g_m\}$ where $m = kl$. We will refer to F as the *primal system* and to G as the *reduction set*. Each polynomial system is generated from a different PC pair under the same key \mathbf{k} . For each f_i we find a g_j so that $\sigma(f_i, g_j)$ is maximal and compute $h_i = f_i + g_j \in I(\mathbf{k})$. We get an ideal $I = \langle h_1, \dots, h_k \rangle \subseteq I(\mathbf{k})$. Similarly to the previous experiments, we can now compute the Gröbner basis and obtain the solution \mathbf{k} . Since we work over $\text{GF}(2)$, if the polynomials f_i and g_j are similar enough, the alike monomials cancel each other out and the resulting polynomials h_i might be smaller than f_i . As a result, this might allow faster computation.

As already mentioned, we get a different system for each PC pair. How much different depends on the degree of diffusion in the cipher. In Table 3, we have shown that the polynomials for $\text{SR}(n, 2, 2, 4)$ with $n \geq 3$ are essentially random. This reflects in Table 5, which contains the results of experiments with the reduced polynomials h_i . The times stated in the table are always the overall wall times, and each value in the table is the average for five independent experiments. The value l in the table is the number of polynomial systems of the reduction set, as described in the paragraph above. We see that for $\text{SR}(3, 2, 2, 4)$, the Average number of Monomials per Polynomial after the Reduction (AMPR) does not differ from the AMP value in Table 4. On the other hand, for example, for $\text{SR}(2, 4, 2, 4)$ and $l = 5$, AMPR is reduced by 86%. Unfortunately, we could still not compute the solution. For $\text{SR}(2, 2, 4, 4)$ and $l = 5$, the reduction allowed us to solve the system, but for $l = 2$, it did so only for the SAT solver. For $\text{SR}(2, 2, 2, 4)$, the reduction shortened the computation time. We note that we considered only the ciphers that required more than five seconds to solve in the previous table. We can also see that the number of polynomial systems for reduction l considerably lowered the AMPR value only for $\text{SR}(2, 2, 4, 4)$ and for other ciphers it had no, or very subtle effect. We have also tried other values of l , all of which were ≤ 50 due to limited time, with no significant effect either, even for $\text{SR}(2, 2, 4, 4)$. The column labeled PT now includes the time required for the reduction.

In order to increase the reduction even further, we tried generating the plaintexts in the PC pairs for the polynomial systems in G so that each of them would differ only by one bit from the plaintext for F . It emerged that this approach did not bring any signifi-



(a) SR(2, 2, 4, 4)



(b) SR(2, 4, 2, 4)

Figure 3: Frequencies of the key variables for five instances of SR(2, 2, 4, 4) and SR(2, 4, 2, 4). The variables are ordered according to their frequency.

cant improvement.

Since the F4 algorithm and the SAT solver run in a single thread, and we had a parallel architecture at our disposal, we performed guess-and-determine attack and tried guessing some variables in the reduced polynomial systems with $l = 5$. This means that we determined the values of the guessed variables, we substituted these values into the system, and then we attempted to solve the system. Observe that substituting concrete values of some variables not only eliminates the variables, but also shortens the polynomials—for example, a zero occurring in a monomial makes it vanish. On the other hand, substituting a one can lead to two equal monomials which cancel each other out. We used a brute-force approach for guessing the variables so we got 2^v different systems to solve where v is the number of guessed variables. Instead of guessing random variables, we tried to guess the most frequent ones in order to shorten the polynomials even further. The reason can be seen in figure 3. This figure contains the frequencies of the variables for five instances of SR(2, 2, 4, 4) and SR(2, 4, 2, 4). The variables are ordered in a descending order, so their labels correspond to their relative positions in the plot according to their frequency—the zeroth variable is the most frequent one. We can see that some of the frequencies differ significantly. Recall that, on the other hand, the frequencies of the variables of SR(3, 2, 2, 4) are evenly distributed as we already showed. We have tried guessing the eight most frequent variables, so we had $2^8 = 256$ parallel threads, one thread for each guess. The results are presented in Table 6.

The table shows that we were able to obtain the solution for SR(2, 4, 2, 4) and that the solving time is reduced significantly for the other two ciphers. Note that the F4 algorithm outperforms the SAT solver.

Table 6: Experiments with reduced polynomial systems and guessed variables.

Cipher	Key bits	PT ^a	F4		SAT
			Time	Mem.	Time
SR(3, 2, 2, 4)	16	8 m	6 s	33 MB	35 s
SR(2, 4, 2, 4)	32	2.5 m	43 s	620 MB	9 m
SR(2, 2, 4, 4)	32	31 s	14 s	72 MB	5.5 m

^a Preprocessing Time — the time required to obtain the system

Also, observe that the preprocessing time for SR(3, 2, 2, 4) has significantly increased. This is caused by counting the frequencies since each of the 16 polynomials has around 2^{14} monomials. We have also tried guessing eight of the least frequent variables and we were not able to obtain the solutions for SR(2, 4, 2, 4) and SR(2, 2, 4, 4) even though we solved the system for SR(2, 2, 4, 4) in the previous table. This was due to memory limitations as each of the parallel processes allocated dozens of gigabytes—we see in Table 5 that the F4 algorithm allocated on average 34.5 GB when solving SR(2, 2, 4, 4) with no guessed variables. We note that each of the threads finished its computation in a different time. The threads that provided no solution usually ended earlier. This could be leveraged in further analysis since this observation also provides information about the correct key. We have also tried guessing different numbers of variables. Guessing more than eight variables produced even longer solving times. This was caused by creating too many threads. On the other hand, we were often unable to obtain the solutions for SR(2, 4, 2, 4) when we guessed less than six variables.

5 CONCLUSIONS

In our experiments, we demonstrated the capabilities of solving systems of polynomial equations by means of Gröbner bases and a SAT solver. Initially, we generated systems that contain the auxiliary variables, and we saw that the SAT solver significantly outperformed Gröbner bases. We subsequently eliminated the auxiliary variables by a gradual substitution so that the systems contained only the variables of the initial secret key. We saw that the results were even worse compared to the systems with the auxiliary variables. However, when we combined at least two systems with no auxiliary variables, we got much better results, especially for Gröbner bases. Note, for example, that we were able to obtain the secret key for one round of AES-128. We also solved one round of all the other ciphers with the state array reduced.

We showed that a 16-bit version of AES reaches its full diffusion after its third round. We also showed that the polynomial system in the third round has the same properties as the system in the tenth round. From an algebraic cryptanalysis point of view, this might suggest that the original AES has enough spare rounds as well.

We tried reducing the polynomial systems without auxiliary variables by adding similar polynomials so that equal monomials would cancel each other out, and we also tried guessing the most frequent variables. The combination of these two approaches allowed us to obtain the solutions for some of the systems that we could not solve otherwise.

ACKNOWLEDGEMENTS

This work was supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16.019/0000765 "Research Center for Informatics" and by the Grant Agency of the CTU in Prague, grant No. SGS21/142/OHK3/2T/18 funded by the MEYS of the Czech Republic.

REFERENCES

Adams, W. (1994). *An introduction to Gröbner bases*. American Mathematical Society, Providence, R.I.

Aumasson, J.-P. (2019). Too much crypto. *IACR Cryptol. ePrint Arch.*, 2019:1492.

Bard, G. (2009). *Algebraic cryptanalysis*. Springer Science & Business Media.

Becker, T. (1993). *Gröbner bases : a computational approach to commutative algebra*. Springer-Verlag, New York.

Bosma, W., Cannon, J., and Playoust, C. (1997). The

Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265. Computational algebra and NUMBER theory (London, 1993).

Brickenstein, M. and Dreyer, A. (2009). Polybori: A framework for gröbner-basis computations with boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326 – 1345. Effective Methods in Algebraic Geometry.

Buchberger, B. (2006). Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of symbolic computation*, 41(3-4):475–511.

Bulygin, S. and Brickenstein, M. (2010a). Obtaining and solving systems of equations in key variables only for the small variants of aes. *Mathematics in Computer Science*, 3(2):185–200.

Bulygin, S. and Brickenstein, M. (2010b). Obtaining and solving systems of equations in key variables only for the small variants of aes. *Mathematics in Computer Science*, 3(2):185–200.

Cid, C. and Leurent, G. (2005). An analysis of the xsl algorithm. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 333–352. Springer.

Cid, C., Murphy, S., and Robshaw, M. (2006). *Algebraic aspects of the advanced encryption standard*. Springer Science & Business Media.

Cid, C., Murphy, S., and Robshaw, M. J. (2005). Small scale variants of the aes. In *International Workshop on Fast Software Encryption*, pages 145–162. Springer.

Courtois, N. T. and Pieprzyk, J. (2002). Cryptanalysis of block ciphers with overdefined systems of equations. In *International conference on the theory and application of cryptology and information security*, pages 267–287. Springer.

Cox, D. (2015). *Ideals, varieties, and algorithms : an introduction to computational algebraic geometry and commutative algebra*. Springer, Cham.

Faugère, J.-C. (1999). A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88.

Hibi, T. (2013). *Gröbner bases : statistics and software systems*. Springer, Tokyo New York.

Murphy, S. and Robshaw, M. J. (2002). Essential algebraic structure within the aes. In *Annual International Cryptology Conference*, pages 1–16. Springer.

Nover, H. (2005). Algebraic cryptanalysis of aes: an overview. *University of Wisconsin, USA*, pages 1–16.

Pub, N. F. (2001). 197: Advanced encryption standard (aes). *Federal information processing standards publication*, 197(441):0311.

Simmons, S. (2009). Algebraic cryptanalysis of simplified aes*. *Cryptologia*, 33(4):305–314.

Soos, M., Nohl, K., and Castelluccia, C. (2009). Extending sat solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer.

The Sage Developers (2020). *SageMath, the Sage Mathematics Software System (Version 9.1)*.