Algebraic Attacks Using SAT-Solvers

Philipp Jovanovic and Martin Kreuzer

Fakultät für Informatik und Mathematik
Universität Passau
D-94030 Passau, Germany

Abstract. Algebraic attacks lead to the task of solving polynomial systems over $\mathbb{F}_2$. We study recent suggestions of using SAT-solvers for this task. In particular, we develop several strategies for converting the polynomial system to a set of CNF clauses. This generalizes the approach in [4]. Moreover, we provide a novel way of transforming a system over $\mathbb{F}_{2^e}$ to a (larger) system over $\mathbb{F}_2$. Finally, the efficiency of these methods is examined using standard examples such as CTC, DES, and Small Scale AES.

Key words: algebraic cryptanalysis, SAT solver, AES, polynomial system solving

1 Introduction

The basic idea of algebraic cryptanalysis is to convert the problem of breaking a cypher to the problem of solving a system of polynomial equations over a finite field, usually a field of characteristic 2. A large number of different approaches has been developed to tackle such polynomial systems. For an overview of some approaches see [12].

In this note we examine a recent suggestion, namely to convert the system to a set of propositional logic clauses and then to use a SAT-solver. The first idea in this direction was presented in [16]. In [8], the SAT-solver technique was successfully applied to attack 6 rounds of DES. The first study of efficient methods for converting boolean polynomial systems to CNF clauses was presented in [4] where the following procedure was suggested:

(1) Linearise the system by introducing a new indeterminate for each term in the support of one of the polynomials.
(2) Having written a polynomial as a sum of indeterminates, introduce new indeterminates to cut it after a certain number of terms. (This number is called the *cutting number*.)
(3) Convert the reduced sums into their logical equivalents using a XOR-CNF conversion.

Later this study was extended slightly in [5], [3], and [18] but the procedure was basically unaltered. Our main topic, discussed in Section 2 of this paper,

is to examine different *conversion strategies*, i.e. different ways to convert the polynomial system into a satisfiability problem. The crucial point is that the linearisation phase (1) usually produces too many new indeterminates. Our goal will be to substitute not single terms, but term combinations, in order to save indeterminates and clauses in the CNF output.

For certain cryptosystems such as AES (see [9]) or its small scale variants (see [6]), the polynomial systems arising from an algebraic attack are naturally defined over a finite extension field $\mathbb{F}_{2^e}$ of $\mathbb{F}_2$, for instance over $\mathbb{F}_{16}$ or $\mathbb{F}_{256}$. While it is clear that one can convert a polynomial system over $\mathbb{F}_{2^e}$ to a polynomial system over $\mathbb{F}_2$ by introducing additional indeterminates, it is less clear what the best way is to do this such that the resulting system over $\mathbb{F}_2$ allows a good conversion to a SAT problem. An initial discussion of this question is contained in [2]. Although the algorithm we present in Section 3 is related to the one given there, our method seems to be easier to implement and to allow treatment of larger examples.

In the last section we report on some experiments and timings using the first author's implementation of our strategies in the ApCoCoA system (see [1]). By looking at the Courtois Toy Cipher (CTC), the Data Encryption Standard (DES), and Small Scale AES, we show that a suitably chosen conversion strategy can save a substantial amount of logical variables and clauses in the CNF output. The typical savings are in the order of 10% of the necessary logical variables and up to 25% in the size of the set of clauses. Frequently, the benefit is then a significant speed-up of the SAT-solvers which are applied to these sets of clauses. Here the gain can easily be half of the execution time or even more. We shall also see that, in the cases we examined, a straightforward Gröbner basis approach to solving polynomial systems over $\mathbb{F}_2$ is slower by several orders of magnitude.

This paper is based on the first author's thesis [11]. Unless explicitly noted otherwise, we adhere to the definitions and notation of [13] and [14].

2 Converting Boolean Polynomials to CNF Clauses

In this section we let $\mathbb{F}_2$ be the field with two elements and $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ a polynomial. Usually f will be a *boolean polynomial*, i.e. all terms in the support of f will be squarefree, but this is not an essential hypothesis. Let $M = \{X_1, \ldots, X_n\}$ be a set of *boolean variables* (atomic formulas), and let $\widehat{M}$ be the set of all (propositional) logical formulas that can be constructed from them, i.e. all formulas involving the operations $\neg$, $\wedge$, and $\vee$.

The following definition describes the relation between the zeros of a polynomial and the evaluation of a logical formula.

Definition 1. *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a polynomial. A logical formula $F \in \widehat{M}$ is called a* **logical representation** *of f if $\varphi_a(F) = f(a_1, \ldots, a_n) + 1$ for every $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$. Here φ_a denotes the boolean value of F at the tuple of boolean values a where $1 =$ `true` and $0 =$ `false`.*

The main effect of this definition is that boolean tuples at which F is satisfied correspond uniquely to zeros of f in $\mathbb{F}_2^n$. The following two lemmas contain useful building blocks for conversion strategies.

Lemma 2. *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a polynomial, let $F \in \widehat{M}$ be a logical representation of f, let y be a further indeterminate, and let Y be a further boolean variable. Then $G = (\neg F \Leftrightarrow Y)$ is a logical representation of the polynomial $g = f + y$.*

Proof. Let $\bar{a} = (a_1, \ldots, a_n, b) \in \mathbb{F}_2^{n+1}$. We distinguish two cases.

(1) If $b = 1$ then $g(\bar{a}) = f(a) + 1 = \varphi_a(F)$. Since $\varphi_b(Y) = 1$, we get $\varphi_{\bar{a}}(\neg F \Leftrightarrow Y) = \varphi_a(\neg F) = \varphi_a(F) + 1$.
(2) If $b = 0$ then $g(\bar{a}) = f(a) = \varphi_a(F) + 1$ and $\varphi_b(Y) = 0$ implies $\varphi_{\bar{a}}(\neg F \Longleftrightarrow Y) = \varphi_a(F)$.

In both cases we find $\varphi_{\bar{a}}(\neg F \Longleftrightarrow Y) = g(\bar{a}) + 1$, as claimed. □

The preceding lemma is the work horse for the standard conversion algorithm. The next result extends it in a useful way.

Lemma 3. *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n, y]$ be a polynomial of the form $f = \ell_1 \cdots \ell_s + y$ where $1 \leq s \leq n$ and $\ell_i \in \{x_i, x_i + 1\}$ for $i = 1, \ldots, s$. We define formulas $L_i = X_i$ if $\ell_i = x_i$ and $L_i = \neg X_i$ if $\ell_i = x_i + 1$. Then*

$$F = (\neg Y \vee L_1) \wedge \ldots \wedge (\neg Y \vee L_s) \wedge (Y \vee \neg L_1 \vee \ldots \vee \neg L_s)$$

is a logical representation of f. Notice that F is in conjunctive normal form (CNF) *and has $s + 1$ clauses.*

Proof. Let $a = (a_1, \ldots, a_n, b) \in \mathbb{F}_2^{n+1}$ We will show $\varphi_a(F) = f(a) + 1$ by induction on s. In the case $s = 1$ we have $f = x_1 + y + c$ where $c \in \{0, 1\}$ and $F = (\neg Y \vee L_1) \wedge (Y \vee \neg L_1)$ where $L_1 = X_1$ if $c = 0$ and $L_1 = \neg X_1$ if $c = 1$. The claim $\varphi_a(F) = f(a) + 1$ follows easily with the help of a truth table.

Now we prove the inductive step, assuming that the claim has been shown for $s - 1$ factors ℓ_i, i.e. for $f' = \ell_1 \cdots \ell_{s-1}$ and the corresponding formula F'. To begin with, we assume that $\ell_s = x_s$ and distinguish two sub-cases.

(1) If $a_s = 0$, we have $\varphi_a(F) = \varphi_a(\neg Y \vee L_1) \wedge \ldots \wedge (\neg Y \vee L_{s-1}) \wedge \neg Y = \varphi_b(\neg Y)$ and $f(a) = b$. This shows $\varphi_a(F) = f(a) + 1$.
(2) If $a_s = 1$, we have $f(a) = f'(a)$. Using $\varphi_a(L_s) = 1$, we obtain

$$\varphi_a(F) = \varphi_a(\neg Y \vee L_1) \wedge \ldots \wedge (\neg Y \vee L_{s-1}) \wedge (Y \vee \neg L_1 \vee \ldots \vee \neg L_{s-1}) = \varphi_a(F')$$

Hence the inductive hypothesis yields $\varphi_a(F) = \varphi_a(F') = f'(a) + 1 = f(a) + 1$.

In the case $\ell_s = x_s + 1$, the proof proceeds in exactly the same way. □

Based on these lemmas, we can define three elementary strategies for converting systems of (quadratic) polynomials over $\mathbb{F}_2$ into linear systems and a set of CNF clauses.

Definition 4. *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a polynomial.*

(1) For each non-linear term t in the support of f, introduce a new indeterminate y and a new boolean variable Y. Substitute y for t in f and append the clauses corresponding to $t+y$ in Lemma 3 to the set of clauses. This is called the **standard strategy (SS)**.

(2) Assuming $\deg(f) = 2$, try to find combinations $x_i x_j + x_i$ in the support of f. Introduce a new indeterminate y and a new boolean variable Y. Replace $x_i x_j + x_i$ in f by y and append the clauses corresponding to $x_i(x_j + 1) + y$ in Lemma 3 to the set of clauses. This is called the **linear partner strategy (LPS)**.

(3) Assuming $\deg(f) = 2$, try to find combinations $x_i x_j + x_i + x_j + 1$ in the support of f. Introduce a new indeterminate y and a new boolean variable Y. Replace $x_i x_j + x_i + x_j + 1$ in f by y and append the clauses corresponding to $(x_i + 1)(x_j + 1) + y$ in Lemma 3 to the set of clauses. This is called the **double partner strategy (DPS)**.

Let compare the effect of these strategies in a simple example.

Example 5. Consider the polynomial $f = x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1$ in $\mathbb{F}_2[x_1, x_2, x_3]$. The following table lists the number of additional logical variables (#v) and clauses (#c) each strategy produces during the conversion of this polynomial to a set of CNF clauses.

strategy	SS	LPS	DPS
# v	4	3	3
# c	25	17	13

Even better results can be achieved for quadratic and cubic terms by applying the following two propositions.

Proposition 6 (Quadratic Partner Substitution).
Let $f = x_i x_j + x_i x_k + y \in \mathbb{F}_2[x_1, \ldots, x_n, y]$ be a polynomial such that i, j, k are pairwise distinct. Then

$$\begin{aligned} F = {} & (X_i \vee \neg Y) \wedge (X_j \vee X_k \vee \neg Y) \wedge (\neg X_j \vee \neg X_k \vee \neg Y) \wedge \\ & (\neg X_i \vee \neg X_j \vee X_k \vee Y) \wedge (\neg X_i \vee X_j \vee \neg X_k \vee Y) \end{aligned}$$

is a logical representation of f.

Proof. Using a truth table it is easy to check that the polynomial $g = x_i x_j + x_i x_k \in \mathbb{F}_2[x_1, \ldots, x_n]$ has the logical representation

$$G = (\neg X_i \vee \neg X_j \vee X_k) \wedge (\neg X_i \vee X_j \vee \neg X_k)$$

Now Lemma 2 implies that the formula $F = \neg G \Leftrightarrow Y$ represents f, and after applying some simplifying equivalences we get the claimed formula. □

It is straightforward to formulate a conversion strategy, called the **quadratic partner strategy (QPS)**, for polynomials of degree two based on this proposition. Let us see how this strategy performs in the setting of Example 5.

Example 7. Let $f = x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1 \in \mathbb{F}_2[x_1, x_2, x_3]$. Then QPS introduces 2 new logical variables and produces 16 additional clauses. Although the number of clauses is higher than for DPS, the lower number of new indeterminates is usually more important and provides superior timings.

For cubic terms, e.g. the ones appearing in the equations representing DES, the following substitutions can be used.

Proposition 8 (Cubic Partner Substitution).
Let $f = x_ix_jx_k + x_ix_jx_l + y \in \mathbb{F}_2[x_1, \ldots, x_n, y]$, where i, j, k, l are pairwise distinct. Then

$$\begin{aligned} F = {} & (X_i \vee \neg Y) \wedge (X_j \vee \neg Y) \wedge (X_k \vee X_l \vee \neg Y) \wedge (\neg X_k \vee \neg X_l \vee \neg Y) \wedge \\ & (\neg X_i \vee \neg X_j \vee \neg X_k \vee X_l \vee Y) \wedge (\neg X_i \vee \neg X_j \vee \neg X_k \vee \neg X_l \vee Y) \end{aligned}$$

is a logical representation for f.

Proof. Using a truth table it is easy to check that the polynomial $g = x_ix_jx_k + x_ix_jx_l \in \mathbb{F}_2[x_1, \ldots, x_n]$ has the logical representation

$$G = (\neg X_i \vee \neg X_j \vee \neg X_k \vee X_l) \wedge (\neg X_i \vee \neg X_j \vee X_k \vee \neg X_l)$$

Now Lemma 2 yields the representation $F = \neg G \Leftrightarrow Y$ for f, and straightforward simplification produces the desired result. □

By inserting this substitution method into the conversion algorithm, we get the **cubic partner strategy (CPS)**. In Section 4 we shall see the savings in clauses, indeterminates, and execution time one can achieve by applying this strategy to DES. For cubic terms, it is also possible to pair them if they have just one indeterminate in common. However, this strategy apparently does not result in useful speed-ups and is omitted.

To end this section, we combine the choice of a substitution strategy with the other steps of the conversion algorithm and spell out the version which we implemented and used for the applications and timings in Section 4.

Proposition 9 (Boolean Polynomial System Conversion).
Let $f_1, \ldots, f_m \in \mathbb{F}_2[x_1, \ldots, x_n]$, and let $\ell \geq 3$ be the desired cutting number. Consider the following sequence of instructions.

C1. *Let $G = \emptyset$. Perform the following steps* **C2-C5** *for $i = 1, \ldots, m$.*
C2. *Repeat the following step* **C3** *until no polynomial g can be found anymore.*
C3. *Find a subset of* $\mathrm{Supp}(f_i)$ *which defines a polynomial g of the type required by the chosen conversion strategy. Introduce a new indeterminate y_j, replace f_i by $f_i - g + y_j$, and append $g + y_j$ to G.*

C4. *Perform the following step* **C5** *until* $\#\,\mathrm{Supp}(f_i) \leq \ell$. *Then append* f_i *to* G.
C5. *If* $\#\,\mathrm{Supp}(f_i) > \ell$ *then introduce a new indeterminate* y_j, *let* g *be the sum of the first* $\ell - 1$ *terms of* f_i, *replace* f_i *by* $f_i - g + y_j$, *and append* $g + y_j$ *to* G.
C6. *For each polynomial in* G, *compute a logical representation in CNF. Return the set of all clauses* K *of all these logical representations.*

This is an algorithm which computes (in polynomial time) a set of CNF clauses K *such that the boolean tuples satisfying* K *are in 1-1 correspondence with the solutions of the polynomial system* $f_1 = \cdots = f_m = 0$.

Proof. It is clear that steps **C2-C3** correspond to the linearisation part (1) of the procedure given in the introduction, and that steps **C4-C5** are an explicit version of the cutting part (2) of that procedure. Moreover, step **C6** is based on Lemma 2, Lemma 3, Prop. 6, or Prop. 8 for the polynomials $g + y_j$ from step **C3**, and on the standard XOR-CNF conversion for the linear polynomials from steps **C4-C5**. The claim follows easily from these observations. □

3 Converting Char 2 Polynomials to Boolean Polynomials

In the following we let $e > 0$, and we suppose that we are given polynomials $f_1, \ldots, f_m \in \mathbb{F}_{2^e}[x_1, \ldots, x_n]$. Our goal is to use SAT-solvers to solve the system $f_1(x_1, \ldots, x_n) = \cdots = f_m(x_1, \ldots, x_n) = 0$ over the field $\mathbb{F}_{2^e}$. For this purpose we represent the field $\mathbb{F}_{2^e}$ in the form $\mathbb{F}_{2^e} \cong \mathbb{F}_2[x]/\langle g \rangle$ with an irreducible, unitary polynomial g of degree e.

Notice that every element a of $\mathbb{F}_{2^e}$ has a unique representation $a = a_1 + a_2\bar{x} + a_3\bar{x}^2 + \cdots + a_e\bar{x}^{e-1}$ with $a_i \in \{0, 1\}$. Here $\bar{x}$ denotes the residue class of x in $\mathbb{F}_{2^e}$. Let ε be the homomorphism of $\mathbb{F}_2$-algebras

$$\varepsilon : (\mathbb{F}_2[x]/\langle g \rangle)[x_1, \ldots, x_n] \longrightarrow (\mathbb{F}_2[x]/\langle g \rangle)[y_1, \ldots, y_{en}]$$

given by $x_i \mapsto y_{(i-1)e+1} + y_{(i-1)e+2} \cdot \bar{x} + \cdots + y_{ie} \cdot \bar{x}^{e-1}$ for $i = 1, \ldots, n$.

Proposition 10 (Base Field Transformation).
In the above setting, consider the following sequence of instructions.

F1. *Perform the following steps* **F2-F5** *for* $i = 1, \ldots, m$.
F2. *For each term* $t \in \mathrm{Supp}(f_i)$ *compute a representative* t'' *for* $\varepsilon(t)$ *using the following steps* **F3-F4**. *Recombine the results to get a representative for* $\varepsilon(f_i)$.
F3. *Apply the substitutions* $x_j \mapsto y_{(j-1)e+1} + y_{(j-1)e+2} \cdot \bar{x} + \cdots + y_{je} \cdot \bar{x}^{e-1}$ *and get a polynomial* t'.
F4. *Compute* $t'' = \mathrm{NR}_{Q \cup \{g\}}(t')$. *Here* $Q = \{y_k^2 - y_k \mid k = 1, \ldots, ne\}$ *is the set of* field equations *of* $\mathbb{F}_2$ *and the* normal remainder NR *is computed by the Division Algorithm (see [13], Sect. 1.6).*
F5. *Write* $\varepsilon(f_i) = h_{i1} + h_{i2}\bar{x} + \cdots + h_{ie}\bar{x}^{e-1}$ *with* $h_{ij} \in \mathbb{F}_2[y_1, \ldots, y_{ne}]$.
F6. *Return the set* $H = \{h_{ij}\}$.

This is an algorithm which computes a set of polynomials H in $\mathbb{F}_2[y_1, \ldots, y_{en}]$ such that the $\mathbb{F}_2$-rational common zeros of H correspond 1-1 to the F_{2^e}-rational solutions of $f_1 = \cdots = f_m = 0$.

Proof. Using the isomorphism $\mathbb{F}_{2^e} \cong \mathbb{F}_2[x]/\langle g \rangle$ we represent the elements of $\mathbb{F}_{2^e}$ uniquely as polynomials of degree $\leq e-1$ in the indeterminate x. Let $a = (a_1, \ldots, a_n) \in \mathbb{F}_{2^e}^n$ be a solution of the system $f_1 = \cdots = f_m = 0$. For $k = 1, \ldots, n$, we write $a_k = c_{k1} + c_{k2}\bar{x} + \cdots + c_{ke}\bar{x}^{e-1}$ with $c_{k\ell} \in \{0, 1\}$.

By the definition of ε, we see that $(c_{11}, \ldots, c_{ne})$ is a common zero of the polynomials $\{\varepsilon(f_1), \ldots, \varepsilon(f_m)\}$. Since $\{1, \bar{x}, \ldots, \bar{x}^{e-1}\}$ is a basis of the $\mathbb{F}_2[y_1, \ldots, y_{ne}]$-module $\mathbb{F}_2[\bar{x}][y_1, \ldots, y_{ne}]$, the tuple $(c_{11}, \ldots, c_{1e})$ is actually a common zero of all coefficient polynomials h_{ij} of each $\varepsilon(f_i)$.

In the same way it follows that, conversely, every common zero $(c_{11}, \ldots, c_{ne}) \in \mathbb{F}_2^{ne}$ of H yields a solution $(a_1, \ldots, a_n)$ of the given polynomial system over $\mathbb{F}_{2^e}$ via $a_k = c_{k1} + c_{k2}\bar{x} + \cdots + c_{ke}\bar{x}^{e-1}$. □

In the computations below we used the representations $\mathbb{F}_{16} = \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$ for Small Scale AES and $\mathbb{F}_{256} = \mathbb{F}_2[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle$ for the full AES. They correspond to the specifications in [6] and [9], respectively.

4 Applications and Timings

In this section we report on some experiments with the new conversion strategies and compare them to the standard strategy. Moreover, we compare some of the timings we obtained to the straightforward Gröbner basis approach. For the cryptosystems under consideration, we used the ApCoCoA implementations by J. Limbeck (see [15]).

As in [4], the output of the conversion algorithms are files in the DIMACS format which is used by most SAT-solvers. The only exception is the system CryptoMiniSat which uses a certain XOR-CNF file format (see [18]). The conversion algorithm used the cutting number 4 which turned out to be usually the best. The timings were obtained by running MiniSat 2.1 (see [10]) resp. the XOR extension of CryptoMiniSat 2.6 (see [17]) on a 2.66 GHz Quadcore Xeon PC with 8 GB RAM. The timings for the conversion of the polynomial system to a set of CNF clauses were ignored, since the conversion was not implemented efficiently and should be seen as a preprocessing step.

All experiments followed the same procedure. We randomly produced three keys and plaintext - ciphertext pairs. For each of them, we randomly permuted the clauses produced by the conversion algorithms and performed ≥ 20 SAT solver runs. The stated timings are therefore the averages of ≥ 60 runs each. The reason for the observed variability of the individual timings is that SAT solvers are randomized algorithms which rely heavily on heuristical methods.

4.1 The Courtois Toy Cipher (CTC)

This artificial cryptosystem was described in [7]. Its complexity is configurable. We denote the system having n encryption rounds and b parallel S-boxes by

CTC(n,b). In the following table we collect the savings in logical variables ($\#v$) and clauses ($\#c$) we obtain by using different conversion strategies. The S-boxes were modelled using the full set of 14 equations each.

system	$(\#v_{SS}, \#c_{SS})$	$(\#v_{LPS}, \#c_{LPS})$	$(\#v_{DPS}, \#c_{DPS})$	$(\#v_{QPS}, \#c_{QPS})$
CTC(3,3)	(361, 2250)	(352, 1908)	(334, 1796)	(352, 2246)
CTC(4,4)	(617, 4017)	(601, 3409)	(569, 3153)	(617, 3953)
CTC(5,5)	(956, 6266)	(931, 5316)	(881, 4916)	(956, 6116)
CTC(6,6)	(1369, 8989)	(1333, 7621)	(1261, 7045)	(1369, 8845)

Thus the LPS and QPS conversions do not appear to provide substantial improvements over the standard strategy, but DPS reduces the input for the SAT-solver by about 8% variables and 22% clauses. Let us see whether this results in a meaningful speed-up. To get significant execution times, we consider the system CTC(6,6). We try MiniSat (time t_M in seconds) and the XOR version of CryptoMiniSat (time t_C in seconds).

strategy	# v	# c	t_M	t_C
SS	937	5065	19.2	20.3
LPS	865	4417	15.2	19.9
DPS	793	3841	15.4	20.0
QPS	937	5065	19.1	15.8

LPS and DPS both resulted in a 20% speed-up of MiniSat, whereas QPS there was a 20% speed-up of CryptoMiniSat. By combining these methods with other optimizations, significantly larger CTC examples can be solved.

4.2 The Data Encryption Standard (DES)

Next we examine the application of SAT-solvers to DES. By DES-n we denote the system of equations resulting from an algebraic attack at n rounds of DES. To model the S-boxes, we used optimized sets of 10 or 11 equations that we computed via the technique explained in [12]. Since the S-box equations are mostly composed of terms of degree 3, we compare SS conversion to CPS conversion.

In the following table we provide the number of polynomial indeterminates ($\#i$), the number of polynomial equations ($\#e$), the number of logical variables ($\#v$) and clauses ($\#c$) resulting from the SS and the CPS conversion, together with some timings of MiniSat ($t_{M,SS}$ and $t_{M,CPS}$), as well as the XOR version of CryptoMiniSat (t_C). (The timings are in seconds, except where indicated.)

system	$\#i$	$\#e$	$(\#v_{SS}, \#c_{SS})$	$(\#v_{CPS}, \#c_{CPS})$	$t_{M,SS}$	t_C	$t_{M,CPS}$
DES-3	400	550	(5114, 34075)	(4583, 30175)	0.66	0.23	0.46
DES-4	512	712	(6797, 45428)	(6089, 40228)	420	86	280
DES-5	624	874	(8480, 56775)	(6205, 71361)	27h	38h	9.6h

From this table we see that, for DES-3 and DES-4, the CPS reduces the number of logical variables and clauses by about 11% each, resulting in a 33% speed-up of MiniSat. In the case of DES-5 we achieved a 27% reduction in the number of logical variables and an 65% speed-up.

4.3 Small Scale AES and Full AES

Let us briefly recall the arguments of possible configurations of the Small Scale AES cryptosystem presented in [6]. By AES(n,r,c,e) we denote the system such that

- $n \in \{1, \ldots, 10\}$ is the number of encryption rounds,
- $r \in \{1, 2, 4\}$ is the number of rows in the rectangular input arrangement,
- $c \in \{1, 2, 4\}$ is the number of columns in the rectangular input arrangement,
- $e \in \{4, 8\}$ is the bit size of a word.

The word size e indicates the field $\mathbb{F}_{2^e}$ over which the equations are defined, i.e. $e = 4$ corresponds to $\mathbb{F}_{16}$ and $e = 8$ to $\mathbb{F}_{256}$. By choosing the parameters $r = 4$, $c = 4$ and $w = 8$ one gets a block size of $4 \cdot 4 \cdot 8 = 128$ bits, and Small Scale AES becomes AES.

Let us begin with a table which shows the savings in logical variables and clauses one can achieve by using the QPS conversion method instead of the standard strategy. Notice that AES yields linear polynomials or homogeneous polynomials of degree 2. Thus QPS is the only conversion strategy suitable for minimalizing the number of logical variables and clauses.

In the following table we list the number of indeterminates ($\#i$) and equations ($\#e$) of the original polynomial system, as well as the number of logical variables and clauses of its CNF conversion using the standard strategy ($\#v_{SS}, \#c_{SS}$) and the QPS conversion ($\#v_{QPS}, \#c_{QPS}$).

AES(n,r,c,w)	$\#i$	$\#e$	$(\#v_{SS}, \#c_{SS})$	$(\#v_{QPS}, \#c_{QPS})$
AES(9,1,1,4)	592	1184	(2905, 17769)	(2617, 16905)
AES(4,2,1,4)	544	1088	(3134, 20123)	(2878, 19355)
AES(2,2,2,4)	512	1024	(2663, 17611)	(2471, 17035)
AES(3,1,1,8)	832	1664	(11970, 83509)	(11010, 78469)
AES(1,2,2,8)	1152	2304	(14125, 101249)	(13165, 96209)
AES(2,2,2,8)	2048	4096	(32530, 236669)	(30610, 226589)
AES(1,4,4,8)	4352	8704	(52697, 383849)	(49497, 367049)

Although the QPS conversion reduces the logical indeterminates in the CNF output by only about 6-8% and the clauses by an even meagerer 4-5%, we will see that the speed-up for MiniSat can be substantial, e.g. about 26% for one round of full AES.

Our last table provides some timings for MiniSat with respect to the SS conversion set of clauses ($t_{M,SS}$), with respect to the QPS conversion set of clauses ($t_{M,QPS}$), and of the XOR version of CryptoMiniSat (t_C).

AES(n,r,c,w)	$t_{M,SS}$	t_C	$t_{M,QPS}$
AES(9,1,1,4)	0.07	0.02	0.04
AES(4,2,1,4)	0.89	0.09	0.28
AES(2,2,2,4)	0.89	0.22	0.72
AES(3,1,1,8)	56.3	44.9	42.3
AES(1,2,2,8)	86.4	64	61.3
AES(2,2,2,8)	9h	5.1h	5.3h
AES(1,4,4,8)	8824	3h	6551

Thus the QPS conversion usually yields a sizeable speed-up. Notice that also the XOR version of CryptoMiniSat provides competitive timings, in particular when applied to the clauses obtained from the CPS conversion. As mentioned previously, the timings also depend on the chosen plaintext-ciphertext pairs. Above we give average timings. In extreme cases the gain resulting from our strategies can be striking. For instance, for one plaintext-ciphertext pair in AES(3,1,1,8) we measured 222 sec. for the SS strategy and 0.86 sec. for the QPS strategy.

When we compare these timings to the Gröbner basis approach in [15], we see that SAT-solvers are vastly superior. Of the preceding 7 examples, only the first two finish without exceeding the 8 GB RAM limit. They take 596 sec. and 5381 sec. respectively, compared to fractions of a second for the SAT-solvers.

Finally, we note that the timings seem to depend on the cutting number in a rather subtle and unpredictable way. For instance, the best timing for one full round of AES was obtained by using the QPS conversion and a cutting number of 6. In this case, MiniSat was able to solve that huge set of clauses in 716 seconds, i.e. in less than 12 minutes. Clearly, the use of SAT-solvers in cryptanalysis opens up a wealth of new possibilities.

Acknowledgements. The authors are indebted to Jan Limbeck for the possibility of using his implementations of various cryptosystems in ApCoCoA (see [15]) and for useful advice. They also thank Stefan Schuster for valuable discussions and help with the implementations underlying Sect. 4.

References

1. ApCoCoA team: ApCoCoA: Applied Computations in Commutative Algebra. Available at `http://www.apcocoa.org`
2. Bard, G.: On the rapid solution of systems of polynomial equations over low-degree extension fields of GF(2) via SAT-solvers. In: 8th Central European Conf. on Cryptography (2008)
3. Bard, G.: Algebraic Cryptanalysis. Springer Verlag (2009)
4. Bard, G., Courtois, N., Jefferson, C.: Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. Cryptology ePrint Archive 2007(24) (2007)
5. Chen, B.: Strategies on algebraic attacks using SAT solvers. In: 9th Int. Conf. for Young Computer Scientists. IEEE Press (2008)

6. Cid, C., Murphy, S., Robshaw, M.: Small scale variants of the AES. In: Fast Software Encryption: 12th International Workshop. pp. 145–162. Springer Verlag, Heidelberg (2005)
7. Courtois, N.: How fast can be algebraic attacks on block ciphers. In: Biham, E., Handschuh, H., Lucks, S., Rijmen, V. (eds.) Symmetric Cryptography – Dagstuhl 2007. Dagstuhl Sem. Proc., vol. 7021. Available at `http://drops.dagstuhl.de/opus/volltexte/2007/1013`
8. Courtois, N., Bard, G.: Algebraic cryptanalysis of the data encryption standard. In: Galbraith, S. (ed.) IMA International Conference on Cryptography and Coding Theory. LNCS, vol. 4887, pp. 152–169. Springer Verlag (2007)
9. Daemen, J., Rijmen, V.: The Design of Rijndael. AES – The Advanced Encryption Standard. Springer Verlag, Berlin (2002)
10. Eèn, N., Sörensen, N.: Minisat. Available at `http://minisat.se`
11. Jovanovic, P.: Lösen polynomieller Gleichungssysteme über $\mathbb{F}_2$ mit Hilfe von SAT-Solvern. Universität Passau (2010)
12. Kreuzer, M.: Algebraic attacks galore! Groups - Complexity - Cryptology 1, 231–259 (2009)
13. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra 1. Springer Verlag, Heidelberg (2000)
14. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra 2. Springer Verlag, Heidelberg (2005)
15. Limbeck, J.: Implementation und Optimierung algebraischer Angriffe. Diploma thesis, Universität Passau (2008)
16. Massacci, F., Marraro, L.: Logical cryptanalysis as a SAT problem. J. Automated Reasoning 24, 165–203 (2000)
17. Soos, M., Nohl, K., Castelluccia, C.: CryptoMiniSat. Available at `http://planete.inrialpes.fr/~soos/`
18. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing – SAT 2009. LNCS, vol. 5584. Springer Verlag (2009)