Optimizing the AES S-Box using SAT*

Carsten Fuhs
fuhs@informatik.rwth-aachen.de
LuFG Informatik 2, RWTH Aachen University, Germany
and Peter Schneider-Kamp
petersk@imada.sdu.dk
IMADA, University of Southern Denmark, Denmark

1 Introduction

In this paper we describe the implementation of a technique for minimizing XOR circuits used in cryptographic algorithms. More precisely, we present our work from [4] for encoding this synthesis problem to SAT with a focus on the case study of optimizing an important component of the Advanced Encryption Standard (AES) [8]. In addition to these previously published contributions, we report on novel encouraging experimental results that allow us to actually prove optimality of the results obtained.

The AES algorithm consists of the (repeated) application of four steps. The main step for introducing non-linearity is the SubBytes step that is based on a so-called S-box. This S-box is a transformation based on multiplicative inverses in $\mathrm{GF}(2^8)$ combined with an invertible affine transformation. This step can be decomposed into two linear parts and a minimal non-linear part. We focus on the optimization of the linear parts, in particular the first one (called the "top matrix" in [2]).

In this paper, we assume that we have n inputs $x_1, \ldots, x_n$ and m outputs $y_1, \ldots, y_m$. Then the linear function to be computed can be specified by m equations of the form $y_\ell = a_{\ell,1} \cdot x_1 \oplus a_{\ell,2} \cdot x_2 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ for $1 \leq \ell \leq m$. We call each equation a *linear form*. Note that each $a_{\ell,j}$ is a constant from $\mathrm{GF}(2) = \{0, 1\}$, each x_j is a variable over $\mathrm{GF}(2)$, and $\oplus$ and $\cdot$ denote standard addition and multiplication on $\mathrm{GF}(2)$.

Our goal is to come up with an algorithm that computes these linear forms given $x_1, \ldots, x_n$ as inputs. More specifically, we would like to express this algorithm via a *linear straight-line program* (or, for brevity, just *program*). Here, every line of the program has the shape $u := e \cdot v \oplus f \cdot w$ with $e, f \in \mathrm{GF}(2)$ and v, w variables. Some lines of the program will contain the output, i.e., assign the value of one of the desired linear forms to a variable. The *length* of a program is the number of lines the program contains. A program is *optimal* if there is no shorter program that computes the same linear forms.

Example 1. *Consider the following linear forms:*

$$y_1 = x_1 \oplus x_2 \oplus x_3 \qquad y_2 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \qquad y_3 = x_1 \oplus x_2 \oplus x_4$$

A shortest linear program for computing these linear forms has length 4. The following linear program is an optimal solution for this example (and demonstrates cancellation).

$$\begin{array}{llll} v_1 := x_1 \oplus x_2 & & v_3 := x_4 \oplus v_2 & [y_2] \\ v_2 := x_3 \oplus v_1 & [y_1] & v_4 := x_3 \oplus v_3 & [y_3] \end{array}$$

For each output y_ℓ there is a variable v_i that contains the linear form for y_ℓ. This mapping from variables to outputs is given by annotating the program lines with the associated output in square brackets.

*Supported by the G.I.F. grant 966-116.6 and the Danish Natural Science Research Council.

The goal is to find an optimal linear straight-line program for a given set of linear forms both automatically and efficiently. In [4] we first present an encoding of the associated decision problem to SAT:

> Given n variables $x_1, \ldots, x_n$ over GF(2), m linear forms $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and a natural number k, decide if there exists a linear program of length k that computes all y_ℓ.

Of course, if the answer to this question is "Yes", we do not only wish to get this answer, but we would also like to obtain a corresponding program of length (at most) k. To this end, we ensure that each model of the propositional formula describes a program of lenth (at most) k.

Matrix Representation

To facilitate the description of our encoding, we reformulate the problem via matrices over GF(2). Here, given a natural number k, we represent the given coefficients of the m linear forms over n inputs with $y_\ell = a_{\ell,1} \cdot x_1 \oplus a_{\ell,2} \cdot x_2 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ $(1 \leq \ell \leq m)$ as rows of an $m \times n$-matrix A. The ℓ-th row thus consists of the entries $a_{\ell,1} a_{\ell,2} \ldots a_{\ell,n}$ from GF(2). Likewise, we can also express the resulting program via two matrices over GF(2): $B = (b_{i,j})_{k \times n}$, where $b_{i,j} = 1$ iff in line i of the program the input variable x_j is read, and $C = (c_{i,j})_{k \times k}$, where $c_{i,j} = 1$ iff in line i of the program the intermediate variable v_j is read. To represent for example the program line $v_3 = x_4 \oplus v_2$ from Example 1, all $b_{3,j}$ except for $b_{3,4}$ and all $c_{3,j}$ except for $c_{3,2}$ have to be 0. Thus, the third row in B is $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$ while in C it is $\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$. Now, for the matrices B and C to actually represent a legal linear straight-line program, for any row i there must be exactly two non-zero entries in the combined i-th row of B and C. That is, the vector $\begin{pmatrix} b_{i,1} & \ldots & b_{i,n} & c_{i,1} & \ldots & c_{i,k} \end{pmatrix}$ must contain exactly two 1s.

Furthermore, for the represented program to actually compute our linear forms, we have to demand that for each desired output y_ℓ, there is a line i in the program (and the matrices) such that $v_i = y_\ell$ where $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and $v_i = b_{i,1} \cdot x_1 \oplus \ldots \oplus b_{i,n} \cdot x_n \oplus c_{i,1} \cdot v_1 \oplus \ldots \oplus c_{i,i-1} \cdot v_{i-1}$. Finally, to represent the mapping of intermediate variables to outputs, we use a function $f : \{1, \ldots, m\} \to \{1, \ldots, k\}$.

Example 2. *Consider again the linear forms from Example 1. They are represented by the following matrix A. Likewise, the program is represented by the matrices B and C and the function f.*

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad f = \begin{cases} 1 & \mapsto & 2 \\ 2 & \mapsto & 3 \\ 3 & \mapsto & 4 \end{cases}$$

Obviously, all combined rows of B and C contain exactly two non-zero elements. Moreover, by computing the v_i and the y_ℓ, we see that each of the linear forms from A is computed by the program in B and C.

Encoding to Propositional Logic

In [4] we give an encoding of the decision problem as a logical formula in second order logic and perform a stepwise refinement of that encoding such that we can finally reduce it to satisfiability of propositional logic. For the sake of brevity, we make use of cardinality constraints represented by predicates like exactly_k that take a list of variables and check that the number of variables that are assigned 1 is exactly k. The final encoding to SAT can be performed by using [3,

1]. With these constraints, we can ensure that B and C represent a legal linear straight-line program. This is encoded by the following formula β_1:

$$\beta_1 = \bigwedge_{1 \leq i \leq k} \text{exactly}_2(b_{i,1}, \ldots, b_{i,n}, c_{i,1}, \ldots, c_{i,i-1})$$

For $1 \leq j \leq n$ and $1 \leq i \leq k$, we introduce the auxiliary formulae $\psi(j,i)$, which denotes the dependence of the value for v_i with respect to x_j (i.e., whether the value of v_i toggles if x_j changes or not):

$$\psi(j,i) = b_{i,j} \oplus \bigoplus_{1 \leq p < i} c_{i,p} \wedge \psi(j,p)$$

We get the following encoding δ which expresses that f is a function and the i-th intermediate variable indicated by $f(\ell)$ actually computes the ℓ-th linear form.

$$\delta = \beta_1 \ \wedge \bigwedge_{1 \leq \ell \leq m} \left(\bigwedge_{1 \leq i \leq k} \left(f_{\ell,i} \rightarrow \bigwedge_{1 \leq j \leq n} (\psi(j,i) \leftrightarrow a_{\ell,j}) \right) \ \wedge \ \text{exactly}_1(f_{\ell,1}, \ldots, f_{\ell,k}) \right)$$

For the implementation of δ we use the SAT framework in the verification environment AProVE [5] and the Tseitin implementation from SAT4J [7]. As shown in [4], given a decision problem with an $m \times n$ matrix and a natural number k (where w.l.o.g. $m \leq k$ holds since for $m > k$, we could just set $\delta = 0$), our encoding δ has size $\mathcal{O}(n \cdot k^2)$ if the cardinality constraints are encoded in linear space [3].

2 Case Study: Advanced Encryption Standard

As mentioned in the introduction, a major motivation for our work is the minimization of circuits for implementing cryptographic algorithms. For our case study, we consider the first of the linear parts (called the "top matrix" in [2]) which is represented by the 21×8 matrix A given in Figure 1.

Here, the matrices B and C represent a solution with length $k = 23$. This solution was found in less than one minute using our decision procedure from Section 1 with MiniSAT v2.1 as backend on a 2.67 GHz Intel Core i7. We now know that $k_{\min} = 23$ and, indeed, the shortest previously known linear straight-line program for the linear forms described by the matrix A has length $k = 23$ [2]. This shows that our SAT-based optimization method is able to find very good solutions in reasonable time. Unfortunately, proving the unsatisfiability for $k = 22$ proves to be much more challenging. Indeed, we have run many different SAT solvers (including but not limited to glucose, ManySat, MiniSat, MiraXT with 8 threads, OKsolver, PicoSAT, PrecoSAT, RSat, SAT4J) on the CNF file for this instance of our decision problem.

Some promising solvers were run for more than 40 days without terminating.

In an effort to prove unsatisfiability of this instance and thereby prove optimality of the solution with $k = 23$, we have asked for and received a lot of support and good advice from the SAT community (see the Acknowledgements at the end of this paper). At SAT'10, we were pointed to CryptoMiniSat [9], a solver featuring special support for dealing with (implicit) XOR gates, which are often used for cryptographic functions. Using version 2.5.1 of this solver, we were able to prove optimality of the corresponding instance with the improvements of Section 3 on an Opteron 848 at 2.2 GHz within less than 106 hours. Using pre-processing techniques, the

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

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 1:

k	result	time
8	UNSAT	0.4
9	UNSAT	0.5
10	UNSAT	1.2
11	UNSAT	5.0
12	UNSAT	76.8
13	SAT	1.0
14	SAT	3.4
15	SAT	2.8
16	SAT	1.5
17	SAT	4.3
18	SAT	2.7
19	SAT	2.5

Figure 2:

number of variables of this instance can be reduced from more than 45000 to less than 5000 in a matter of minutes.[1]

To analyze how difficult these problems really are, we consider a small subset of the linear forms to be computed for the top matrix. The table to the right shows how the runtimes in seconds of the SAT solver are affected by the choice of k for the case that we consider only the first 8 out of 21 linear forms from A. In order to keep runtimes manageable we already incorporated the symmetry breaking improvement described in Section 3. Note that unsatisfiability for $k = 12$ is still much harder to show than satisfiability for $k_{min} = 13$.

To conclude this case study, we see that while finding (potentially) minimal solutions is obviously feasible, proving their optimality (i.e., unsatisfiability of the associated decision problem for $k = k_{min} - 1$) is challenging. This observation confirms observations made in [6]. In the following section we present some of our attempts to improve the efficiency of our encoding for the UNSAT case.

3 Symmetry Breaking

Satisfiability of propositional logic is an NP-complete problem and, thus, we can expect that at least some instances are computationally expensive. While SAT solvers have proven to be a Swiss army knife for solving practically relevant instances of many different NP-complete problems, our kind of program synthesis problems seems to be a major challenge for today's SAT solvers even on instances with "just" 1500 variables. In this section we discuss an approach based on symmetry breaking.

In general, having many solutions is considered good for SAT instances as the SAT solver is more likely to "stumble" upon one of them. For UNSAT instances, though, having many potential solutions usually means that the search space to exhaust is very large.

One of the main reasons for having many solutions is symmetry. For example, it does not matter if we first compute $v_1 = x_1 \oplus x_2$ and then $v_2 = x_3 \oplus x_4$ or the other way around. Limiting these kinds of symmetries can be expected to significantly reduce the runtimes for

[1] The reader is cordially invited to try his favorite SAT solver on one of the instances available from: `http://aprove.informatik.rwth-aachen.de/eval/slp.zip`

UNSAT instances. In our concrete setting, being able to reorder independent program lines is one of the major sources of symmetry. Two outputs in a straight-line programs are said to be *independent* if neither of them depends on the other (directly through the matrix C or indirectly).

Now, the idea for breaking symmetry is to impose an order on these lines: the line which computes the "smaller" linear form (w.r.t. a total order on linear forms, which can e.g. be obtained by lexicographic comparison of the coefficient vectors) must occur before the line which computes the greater linear form.

We can encode the direct dependence of v_i on v_p:

$$\bigwedge_{1\leq i\leq k} \bigwedge_{1\leq p<i} c_{i,p} \rightarrow dep_{i,p}$$

Likewise, the indirect dependence of v_i on v_p can be encoded by transitivity:

$$\bigwedge_{1\leq i\leq k} \bigwedge_{1\leq p<i} \bigwedge_{p<q<i} c_{i,q} \wedge dep_{q,p} \rightarrow dep_{i,p}$$

We also need to encode the reverse direction, i.e.:

$$\bigwedge_{1\leq i\leq k} \bigwedge_{1\leq p<i} \left(dep_{i,p} \rightarrow \left(c_{i,p} \ \vee \bigvee_{p<q<i} (c_{i,q} \wedge dep_{q,p}) \right) \right)$$

Now for $i > p$, the output v_i should depend on the output v_p or encode a greater linear form than v_p:

$$\bigwedge_{1\leq i\leq k} \bigwedge_{1\leq p<i} (dep_{i,p} \ \vee \ [\psi(1,i),\ldots,\psi(n,i)] >_{lex} [\psi(1,p),\ldots,\psi(n,p)])$$

Here lexicographic comparison of formula tuples is encoded in the usual way (cf. the encoding in [3]).

While this approach eliminates some otherwise valid solutions of length k and thus reduces the set of admissible solutions, obviously there is at least one solution of length k which satisfies our constraints whenever solutions of length k exist at all. This way, we greatly reduce the search space by breaking symmetries that are not relevant for the result, but may slow down the search considerably.

Consider again the restriction of our S-box top matrix to the first 8 linear forms. With symmetry breaking, we can show unsatisfiability for the "hard" case $k = 12$ in 76.8 seconds. In contrast, without symmetry breaking, we cannot show unsatisfiability within 4 days.

4 Conclusion

In this paper we have shortly reiterated how shortest linear straight-line programs for given linear forms can be synthesized using SAT solvers. Then we have evaluated the feasibility of this approach by a case study where we minimize an important part of the S-box for the Advanced Encryption Standard. This study shows that our SAT-based approach is indeed able to synthesize shortest programs for realistic problem settings within reasonable time.

Proving optimality of the programs found by showing unsatisfiability of the associated decision problem leads to very challenging SAT problems. We have shown that breaking some

symmetries of our problem significantly reduces runtimes in the UNSAT case, and using CryptoMiniSat recently we showed optimality in our case study. In future work, we consider to apply our method to other problems from cryptography. Also, we plan to further enhance our encoding and specialize existing SAT solvers to further improve performance in the UNSAT case.

Acknowledgements

Our sincere thanks go to Erika Ábrahám, Daniel Le Berre, Armin Biere, Youssef Hamadi, Marijn Heule, Oliver Kullmann, Matthew Lewis, Lakhdar Saïs, Laurent Simon, and Mate Soos for input on and help with the experiments. We also want to thank the anonymous referees for helpful comments.

Furthermore, we thank Joan Boyar and René Peralta for providing us with information on their work and Michael Codish for pointing out similarities to common subexpression elimination.

References

[1] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In *Proc. SAT '09*, volume 5584 of *LNCS*, pages 167–180, 2009.

[2] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. In *Proc. SEA '10*, volume 6049 of *LNCS*, pages 178–189, 2010.

[3] M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 5:193–215, 2008.

[4] C. Fuhs and P. Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT. In *Proc. SAT '10*, volume 6175 of *LNCS*, pages 71–84, 2010.

[5] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, volume 4130 of *LNAI*, pages 281–286, 2006.

[6] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev. Finding efficient circuits using SAT-solvers. In *Proc. SAT '09*, volume 5584 of *LNCS*, pages 32–44, 2009.

[7] D. Le Berre and A. Parrain. The SAT4J library, release 2.2. *Journal on Satisfiability, Boolean Modelling and Computation (JSAT)*, 7:59–64, 2010.

[8] Federal Information Processing Standard 197. The advanced encryption standard. Technical report, National Institute of Standards and Technology, 2001.

[9] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. *Proc. SAT '09*, volume 5584 of *LNCS*, pages 244–257, 2009.