Towards a better understanding of SAT translations

Matthew Gwynne and Oliver Kullmann

Computer Science Department
Swansea University, UK
csmg@Swansea.ac.uk, O.Kullmann@Swansea.ac.uk
http://cs.swan.ac.uk/~csmg, http://cs.swan.ac.uk/~csoliver

Abstract. Of fundamental importance for SAT solving is the translation to CNF. One of the basic tasks is to find metrics for determining what are "good" translations, i.e., what makes the resulting SAT problem easy to solve. We introduce a new measure $\mathrm{hd}(F)$, the "hardness", for formulas F in conjunctive normal form (i.e., clause-sets). $\mathrm{hd}(F)$ for unsatisfiable clause-sets has been studied in [12,14]. However, now we treat satisfiable clause-sets differently. We consider the Advanced Encryption Standard (AES) and Data Encryption Standard (DES) as examples. The key discovery for these ciphers present examples of hard problems, and we investigate translations of these problems geared towards minimising $\mathrm{hd}(F)$. We present the *SAT representation hypothesis*: The task of solving a SAT problem efficiently is captured by constructing a representation of the underlying boolean function which is of low hardness.

1 Introduction

Over the past decade, the use of SAT solvers for solving a variety of industrial and crafted problems has greatly increased. In particular, the use of SAT to solve cryptographic problems has become popular, and there have been several translations of the Data Encryption Standard and Advanced Encryption Standard ciphers into SAT. However, the question arises with any such translation: what makes a good translation for some problem P? We investigate this question using AES and DES as examples.

We assume that P is already decomposed, somehow, into "constraints". In our example, a constraint C is a "box" within the cipher; another typical example for C would be a cardinality constraint, such as in [1]. By a "constraint" we mean that we "know" (at least) all satisfying and falsifying assignments. We consider only SAT translations which preserve the decomposition of P. That is, each C gets translated into a clause-set F_C, and all the F_C together yield the translation of P. Despite translating to SAT, we still want to consider $F = F_C$ as some form of constraint; we want to "know" its satisfying and falsifying assignments and we want to let the SAT solver know! A common approach here is to adapt the notion of "hyper-arc consistency" by the use of UCP (unit-clause propagation). This can be considered in two senses. In the weak sense, for every partial assignment φ such that $\varphi * F$ is unsatisfiable, UCP on $\varphi * F$ yields the empty clause. In the stronger sense, one requires that additionally every forced assignment in $\varphi * F$ (for satisfiable $\varphi * F$) is also derived by UCP. We strengthen/generalise this approach in three ways:

- we generalise UCP, allowing arbitrary polynomial time, in some standardised sense, to get to "know" the satisfying and falsifying assignments;
- we consider *all* variables in F_C, not just those in C — that is, we should know *all* satisfying assignments of F;
- instead of ad-hoc methods to construct F (for example those for cardinality constraints) we develop systematic methods.

The Data Encryption Standard and the Advanced Encryption Standard have both received significant attention from the cryptography community, and have been attacked in [2,11,16] using SAT solvers. In both cases, the ciphers encrypt a plaintext P to a ciphertext C using a key K. We consider the key discovery problem for each cipher, i.e., finding K from P and C. The DES can be considered as computing the ciphertext via iterated application, using 16 rounds, of various 10-bit S-boxes (6-bit to 4-bit boolean functions, 8 per round) and addition (XOR) of input and key bits. The AES can be considered in a similar way, using 10 rounds and 16-bit S-boxes (8-bit to 8-bit permutations, 16 per round); the AES also has a linear MixColumns operation in each round, using $4 \cdot 16$ 8-bit to 8-bit multiplication operations, and also a non-trivial key-schedule (unlike the DES), involving further S-boxes and additions. It is precisely these S-boxes, multiplications and additions that we consider as our constraints. That is, we translate the AES and DES via a natural decomposition based on the definition, which considers the S-boxes, additions and multiplications as the boolean functions (constraints) to translate directly to SAT. This differs from the approaches taken in [4,11,16], which apply global algebraic methods and completely "rewrite" the DES/AES constraint system.

In order to obtain solvable problems, we consider the 'small-scale AES" generalisation as introduced in [4], denoted by $\mathrm{aes}(m, r, c, e)$, and parameterised by

- the number m of rounds,
- number r of rows,
- number c of columns,
- and the bit-size e of individual elements of the AES "block".

Key, plaintext and ciphertext have the same number of bits $r \cdot c \cdot e$. The standard 128-bit AES is $\mathrm{aes}(10, 4, 4, 8)$, that is, 10 rounds and the 128-bits are considered as 4×4 matrix of 8-bit elements. All "boxes" (i.e., permutations) have e input bits and e output bits, and thus yield $2e$-bit boolean functions. The MixColumns operation, itself an operation using r^2 many such (multiplication) permutation boxes, has r-vectors of e-bit elements as inputs and outputs, and there are (parallel) c such operations per round.

In Section 2, we introduce a new notion of hardness of CNF formulas where a low hardness should mean a good translation. In Section 3 we investigate the complexity of boolean functions in general (from the hardness-perspective). In Section 4, we present experimental results on using representations with low hardness for SAT translations of AES and DES. All software and results are available within the `OKlibrary` (see [15]), a research platform for hard problems; see [10], a forthcoming technical report.

2 Determining hardness

In [12,14], Kullmann introduced a measure of hardness $\mathrm{hd}_{U,S} : \mathcal{CLS} \to \mathbb{N}_0$ of clause-sets[1)]. For unsatisfiable clause-sets, this notion is closely related to the complexity of tree-resolution. For satisfiable instances it measures the complexity of unsatisfiable sub-instances one has to solve to arrive at a satisfying assignment, using only forced assignments (i.e., the negated assignment yields an unsatisfiable sub-instance) plus guessed assignments. $\mathrm{hd}_{U,S}$ is parameterised by oracles U, S for unsatisfiability/satisfiability detection. For this paper, we restrict our considerations to the simplest cases $U_0 := \{F \in \mathcal{CLS} : \bot \in F\}$ ($\bot$ is the empty clause) and $S_0 := \{\top\}$ ($\top$ is the empty-clause-set). A clause-set is considered trivially unsatisfiable or satisfiable if it is in U_0 or S_0 respectively. Kullmann defines a hierarchy of clause-set classes $G_k(U, S)$. Checking whether $F \in G_k(U, S)$ can be done in time $O(n^{2k})$ using breadth-first search.

Definition 1. *The* ***hardness*** $\mathbf{hd_{U,S}(F)} \in \mathbb{N}_0$ *of a clause-set* $F \in \mathcal{CLS}$ *is the minimum* $k \in \mathbb{N}_0$ *with* $F \in G_k(U, S)$.

We have:

1. $F \in G_0(U_0, S_0)$ iff $\bot \in F$ or $F = \top$.
2. $F \in G_1(U_0, S_0)$ iff after unit-clause propagation either $\bot \in F$ or $F = \top$, where for the latter we can guess one assignment.
3. $F \in G_2(U_0, S_0)$ iff after failed-literal reduction either $\bot \in F$ or $F = \top$, where for the latter we can guess two assignments.

Only using the aspect of forced assignment, without guessing assignments, we obtain the k-level reductions $r_k : \mathcal{CLS} \to \mathcal{CLS}$. r_1 is unit-clause propagation, r_2 is failed-literal reduction, and so on. For unsatisfiable F we have that $\mathrm{hd}_{U_0,S_0}(F) = \mathrm{hd}_{U_0}(F)$ is the smallest $k \in \mathbb{N}_0$ such that $\bot \in r_k(F)$.

2.1 A new notion of hardness

We introduce a new notion of hardness, based on hd_{U_0}. For satisfiable clause-sets, now instead of measuring the resources for finding some satisfying assignment we measure the level of r_k-reductions needed to derive *all* possible conclusions.

Definition 2. *The* ***hardness*** $\mathbf{hd(F)} \in \mathbb{N}_0$ *for* $F \in \mathcal{CLS}$ *is the minimal* $k \in \mathbb{N}_0$ *such that for all clauses* C *with* $F \models C$ *(i.e.,* F *implies* C*) we have* $\bot \in r_k(\varphi_C * F)$. *The partial assignment* φ_C *sets all literals of* C *to* 0, *while "*$*$*" denotes application of partial assignments. If* $\mathrm{hd}(F) \leq k$ *we call* F ***k-soft****, and if* $\mathrm{hd}(F) \geq k$ *we call* F ***k-hard****.*

For unsatisfiable clause-sets F we have $\mathrm{hd}_{U_0}(F) = \mathrm{hd}(F)$. However for $F \in \mathcal{SAT}$, $\mathrm{hd}(F)$ characterises the level of look-ahead needed to determine that the solver has entered a unsatisfiable sub-branch, during *any* depth-first backtracking search. Two simple examples:

[1)] A clause-set is set of set of literals which is then interpreted as either a conjunctive or disjunctive normal form formula. If not stated, we consider CNF clause-sets.

1. If f has a unique CNF representation without new variables (equivalently, every falsifying assignment for f is total), then for this representation F of f not using new variables ($\mathrm{var}(F) = \mathrm{var}(f)$) we have $\mathrm{hd}(F) = 0$ (while $\mathrm{hd}_{U_0,S_0}(F) = n(f)$ (the number of variables in f)). Examples are given by the parity functions.
2. If F is unsatisfiable, then $\mathrm{hd}(F) = 0$ iff $\bot \in F$.

Some polynomial time SAT decision classes have low hardness, for instance:

- If F is (renamable) Horn formula, then $\mathrm{hd}(F) \le 1$.
- If all clauses of F have length at most 2, then $\mathrm{hd}(F) \le 2$.

However the point of the measure $\mathrm{hd}(F)$ in general is not to find classes with poly-time SAT decision, since computing the hardness is NP-hard. A satisfying assignment for F with $\mathrm{hd}(F) = k$ can be found in polynomial time by checking whether $\bot \in r_k(F)$; if $\bot \in r_k(F)$ then F is unsatisfiable; otherwise we find a variable v and assignment $b \in \{0,1\}$ such that $\bot \notin r_k(\langle v \to b\rangle * F)$ and continue recursively on $\langle v \to b\rangle * F$ which must itself be satisfiable.

Now consider a boolean function f with n variables, such that we wish to *represent* f as a CNF clause-set.

Definition 3. *A* ***representation*** $F \in \mathcal{CLS}$ *of a boolean function* f *is a clause-set* F *with* $\mathrm{var}(f) \subseteq \mathrm{var}(F)$ *such that restricting the satisfying assignments of* F *to* $\mathrm{var}(f)$ *we get exactly the satisfying assignments of* f*.*

The two most extreme representations (without using new variables) of F with minimum and maximum hardness are the set of all prime implicates and the canonical CNF for f.

Definition 4. *Let* $\mathbf{prc_0}(f)$ *be the set of all clauses* C *which as CNF are* ***prime implicates*** *of* f*, that is,* $f \models C$ *while* $f \not\models C'$ *for every* $C' \subset C$*. For clause-sets we overload this notion, that is, for* $F \in \mathcal{CLS}$*, we have* $\mathbf{prc_0}(F) = \mathrm{prc}_0(f)$ *for the boolean function* f *given by* F*.*

Let $\mathbf{CNF}(f)$ *be the canonical CNF clause-set for* f*, that is, the CNF clause-set* F *with* $\mathrm{var}(F) = \mathrm{var}(f)$ *and with* $C \in F$ *for all clauses of length* $n(f) := |\mathrm{var}(f)|$ *with* $F \models C$*. For the canonical DNF clause-set, we use* $\mathbf{DNF}(f)$*. In general, we call a clause-set* F ***full*** *if every clause* $C \in F$ *has* $|C| = n(F)$*.*

We have $\mathrm{hd}(F) = 0 \Leftrightarrow \mathrm{prc}_0(F) \subseteq F$, and thus $\mathrm{hd}(\mathrm{prc}_0(f)) = 0$. For all unsatisfiable full clause-sets F we have $\mathrm{hd}(F) = n$, while for arbitrary full F we have $\mathrm{hd}(F) = n(F) - \min_{C \in \mathrm{prc}_0(F)} |C|$.

2.2 Finding representations with low hardness

With $\mathrm{hd}_{U,S}(F)$, there is a polynomial time algorithm for checking "$\mathrm{hd}_{U,S}(F) = k$?". On the contrary, for $\mathrm{hd}(F)$ the idea is that we do not measure $\mathrm{hd}(F)$ for a given F, but *construct* a clause-set F representing a given boolean function f so as to minimise $\mathrm{hd}(F)$. In this paper we consider (only) 1-soft representations of boolean functions, which we will use to translate small sub-functions within AES and DES in Section 4.

The use of new variables is a powerful method for reducing the complexity when solving problems from propositional logic. We introduce a translation for a boolean function f using new variables based on the canonical DNF, introducing a new variable $\mathrm{vct}_f(C) \notin \mathrm{var}(f)$ for each clause $C \in \mathrm{DNF}(f)$ (recall, these clauses represent the total satisfying assignments of f).

Definition 5. *Consider a boolean function $f : \{0,1\}^V \to \{0,1\}$. Let the* ***canonical translation*** $\mathbf{ct}(f) \in \mathcal{CLS}$ *be defined as*

$$\mathrm{ct}(f) := \left\{ \{\mathrm{vct}_f(C)\}_{C \in \mathrm{DNF}(f)} \right\} \cup \bigcup_{C \in \mathrm{DNF}(f)} \mathrm{prc}_0 \left(\mathrm{vct}_f(C) \leftrightarrow \bigwedge_{x \in C} x \right).$$

Lemma 6. *For all $f : \{0,1\}^V \to \{0,1\}$ the canonical translation $\mathrm{ct}(f)$ is 1-soft.*

To construct k-soft representations without new variables, we present a search-based approach. We want then to find representations which are minimal in a sense, and so we introduce the notion of a "base".

Definition 7. *A* ***k-base****, $k \in \mathbb{N}_0$, for a boolean function f is a clause-set F fulfilling the following conditions:*

1. *F is a representation of f;*
2. *F is k-soft;*
3. *F is minimal w.r.t. elimination of clauses and literal occurrences for these two conditions.*

A clause-set F is a ***k-base*** *(for itself) if it is a k-base for the underlying boolean function.*

The basic idea to construct a k-base F is to start with the set $\mathrm{prc}_0(f)$ of all prime implicates, and iteratively removing clauses from F while checking that all $C \in \mathrm{prc}_0(f)$ still follow from F by r_k. This simple scheme takes a very long time, and so we have developed various heuristics for generating small 1-bases. We remark that computing small*est* 1-bases seems not feasible for the examples we considered.

2.3 SAT representation hypothesis

Based on our new notion of hardness, we present the *SAT representation hypothesis*, which roughly states that the notion of "k-soft representations of boolean function" is in principle all what is needed, or, somewhat more precisely:

> The task of finding a "good" representation of a boolean function f, for the purpose of SAT solving in polynomial time or of finding a short refutation, is "fully captured" by constructing a k-soft representation F for f for some appropriate k (depending on the problem).

The smaller k, the lower the exponent for the polynomial in the run-time estimation, but the larger F is, so a balance is to be sought. If f is only some part of a bigger function (like for example the S-box in AES), then f should be made as large as possible.

3 On the complexity of boolean functions

Measuring the complexity of boolean functions in our context happens in the literature in the following two main forms:

1. Based on the Fourier transform in the form of the Walsh transform, many measures have been investigated; see [3] for a first overview, and [6,5,7] for recent research. The strength of these approaches is based on the fact that the Fourier transform behaves well under composition, while the major weakness is that these notions do not tackle computational complexity itself, but only ad-hoc notions possibly related to it.
2. AES has natural formulations in terms of commutative algebra, and this has created much attention, applying these algebraic methods also to other ciphers. A good first introduction is [4]. Again, the strength is that mathematical methods can be applied, while the weakness is (again) that only special methods of "attack" (in effect, solving SAT problems) are considered.

Centered around our notion of hardness, we begin more direct investigations into the boolean functions related to our main examples, DES and AES. Mainly, we investigate the sizes of their various representations, like

- the number of total satisfying assignments (the size of the canonical DNF, determining the size of the canonical translation)
- the number of total falsifying assignments (the size of the canonical CNF)
- the number of prime implicates (the size of the 0-base)
- sizes of 1-bases
- minimum-size representations (the minimum size of ∞-bases).

In Figure 1, we present statistics regarding these representations for the boxes in DES and AES. If only upper bounds are known, then this is indicated by the use of "$\leq$". In Figure 1, the hardness of each of the representations used is given. For most boxes we do not currently know the minimum size of their CNF representations, let alone whether there exist other minimum representations with smaller hardness. However, in the case of the 8-bit multiplication by 02, there exist other minimum representations with hardness 3.

Box (f)	n	DNF	$\mathrm{prc}_0(f)$		can		1-base		min		CNF	
			c	hd	c	hd	c	hd	c	hd	c	hd
DES Sbox 1	10	64	1624	0	705	1	≤ 124	1	≤ 67	3	960	5
DES Sbox 2	10	64	1844	0	705	1	≤ 129	1	≤ 67	3	960	5
DES Sbox 3	10	64	1767	0	705	1	≤ 138	1	≤ 68	3	960	5
DES Sbox 4	10	64	1881	0	705	1	≤ 128	1	≤ 69	3	960	5
DES Sbox 5	10	64	1812	0	705	1	≤ 134	1	≤ 67	2	960	5
DES Sbox 6	10	64	1705	0	705	1	≤ 136	1	≤ 66	3	960	5
DES Sbox 7	10	64	1673	0	705	1	≤ 123	1	≤ 67	3	960	5
DES Sbox 8	10	64	2047	0	705	1	≤ 152	1	≤ 69	3	960	5
AES 4-bit Sbox	8	16	147	0	145	1	≤ 27	1	22	2	240	5
AES 4-bit $\times$02	8	16	14	0	145	1	≤ 10	1	9	2	240	6
AES 4-bit $\times$03	8	16	120	0	145	1	≤ 24	1	16	2	240	5
AES 8-bit Sbox	16	256	136253	0	4353	1	≤ 4398	1	≤ 294	4	65280	11
AES 8-bit $\times$02	16	256	58	0	4353	1	≤ 22	1	20	2	65280	14
AES 8-bit $\times$03	16	256	5048	0	4353	1	≤ 80	1	≤ 36	3	65280	13

Fig. 1. Statistics for DES and AES Boxes

4 Experimental results

The AES and DES ciphers were translated using translations provided in the `OKlibrary`. Additions are translated directly by the set of all prime implicates, while each of the S-boxes and multiplications were translated using:

- "minimum" CNF representations, minimising the number of clauses;
- the canonical representation (see Section 2.2);
- 1-base translations (see Section 2.2);
- CNF representations using the canonical (full) CNF.

In terms of hardness, the canonical and 1-base translation produce 1-soft representations, while using full CNFs maximises $\mathrm{hd}(F)$. We would expect to see the fact, that the canonical and 1-base translations are "easier", is reflected in run times of the solvers, and in particular in the number of conflicts (or nodes) the solver uses to solve the problem.

We ran SAT solvers `minisat-2.2.0` ([8]) and the `OKsolver_2002` ([13]) on key discovery instances of the DES over m rounds, $\mathrm{des}(m)$, and small scale AES variants with m rounds, c columns, r rows and field size e, $\mathrm{aes}(m, r, c, e)$. In general, as each of the parameters increases, the instances become harder. The main results are presented in Figure 2, and were each averaged over 20 different plaintext-ciphertext pairs, and 5 different random shufflings of input clause-set. Results denoted by a "-" timed out after an hour.

In Figure 1, we list the boolean functions that we translate and the number of clauses in each of the translations. The size of the canonical translation $\mathrm{ce}(f)$ and translations using the canonical CNF $\mathrm{CNF}(f)$ are omitted. The number of clauses in $\mathrm{ce}(f)$ and $\mathrm{CNF}(f)$ are given by $l + c + 1$ and $2^n - c$ respectively where l and c are the number of literal occurrences and number of clauses in the canonical DNF for that function. The number of variables for the canonical translation is always $n + c$. In Figure 4, the statistics for the CNF translations of each problem are presented. One sees that in most cases both the minimum CNFs and 1-bases are considerably smaller than the set of prime implicates.

In $\mathrm{aes}(20, 1, 1, 4)$, $\mathrm{aes}(20, 2, 1, 4)$, $\mathrm{aes}(10, 1, 1, 8)$ and $\mathrm{aes}(1, 2, 1, 8)$, we see that the canonical and 1-base translations are solved using far fewer nodes than the minimum, often by an order of magnitude, and similarly the full clause-set translation performs worst in all cases. One sees in Figure 1 that the difference in size between the minimum translations and the 1-bases is much larger for the larger constraints such as the 8-bit AES S-box. This might explain why for instances such as $\mathrm{aes}(10, 1, 1, 8)$ the minimum translation performs so much worse than in the other cases. We see, with the smaller constraints in Figure 1, the hardness of the minimum translations is quite low anyway as there are less variables to start with. For the larger constraints there is more "room" for difference between the hardness of the minimum representation and the 1-soft representations, and so the hardness is much higher. For example, the minimum representation for the AES 8-bit S-box has hardness 4 compared to hardness 2 and 3 for all other boxes.

In the case of $\mathrm{des}(3)$ and $\mathrm{aes}(2, 2, 2, 4)$, concerning the minimum translation, our hypothesis does not seem to hold for the `minisat-2.2.0` results. The minimum translation uses fewer conflicts in both cases (although not by much). In the case of $\mathrm{aes}(2, 2, 2, 4)$, one sees in Figure 1 that the hardness of the 4-bit AES

boxes is 2. This is still low, and in combination with the small size representation, seen in Figure 1, it is possible that this outweighs the (small) increase in hardness.

Variant	$\|K\|$	minisat-2.2.0				OKsolver_2002			
		can	1-base	min	full	can	1-base	min	full
des(3)	64	3987.84	1397.14	3220.98	5946.08	90.17	13477.65	408094.4	-
aes$(20,1,1,4)$	4	27.84	96.53	472.35	3856.75	1.0	6.61	16.07	1521.34
aes$(20,1,2,4)$	8	214.83	126.91	1816.14	15310.35	37.42	64.35	341.98	-
aes$(10,1,1,8)$	8	267.53	2129.09	163743.8	-	1.0	200.75	1083.65	-
aes$(2,2,2,4)$	16	10400.22	11061.98	8606.33	18593.56	179.16	4274.75	12013.04	-
aes$(1,2,1,8)$	16	471.5	335.81	1578.48	-	1.0	620.29	12638.98	-

Fig. 2. Conflicts/nodes used to solve DES/AES variants

Variant	$\|K\|$	minisat-2.2.0				OKsolver_2002			
		can	1-base	min	full	can	1-base	min	full
des(3)	64	0.226	0.020	0.025	0.81	8.746	9.233	161.152	> 3600
aes$(20,1,1,4)$	4	0.005	0.0	0.001	0.113	0.02	0.0	0.0	3.408
aes$(20,1,2,4)$	8	0.0252	0.006	0.025	0.407	0.345	0.019	0.117	> 3600
aes$(10,1,1,8)$	8	0.208	3.794	1.537	> 3600	6.26	51.873	1.269	> 3600
aes$(2,2,2,4)$	16	0.459	0.158	0.110	0.4875	2.711	0.802	1.836	> 3600
aes$(1,2,1,8)$	16	0.143	0.584	0.01	> 3600	2.829	2.572	1.448	> 3600

Fig. 3. Times (in seconds) used to solve DES/AES variants

5 Conclusion

Based on the results so far, the hardness of small sub-functions in translations to SAT seem to offer a good indicator of the performance of SAT solvers on the whole problem. Translations using representations of sub-functions with low hardness were often an order of magnitude better than small or full translations.

In two instances, the results were not as expected, with the minimum translation performing better than the 1-soft translation. However, there are still further questions to be answered including, for instance, how the decomposition of the translation into small boolean functions affects hardness, as well as considering the exact hardness of the minimum translation. Answering these questions should shed light on the reasons for the performance of solvers in such instances.

The next step is to investigate different decompositions of the DES and AES into small boolean functions, considering also translations given in [2,11,16], and to consider how such decompositions affect the hardness of the final translation. Such investigations must also continue at a theoretical level, providing a framework for constructing "easy" representations of boolean functions.

Variant	rep	Before UCP			After UCP				
		n	c	l	n	c	l	clause len	
								min	max
des(3)	1-base	456	4280	20339	293	2867	13582	2	7
	canon	1992	18008	52160	1352	12024	34624	2	64
	small	456	2708	12068	295	1727	7777	2	7
	full	456	24128	233408	296	16352	157696	2	10
aes$(20, 1, 1, 4)$	1-base	492	2136	6528	412	1896	5808	2	4
	canon	1132	6856	19328	1052	6616	18608	2	16
	small	492	1936	5968	412	1696	5248	2	5
	full	492	10656	79488	412	10416	78768	2	8
aes$(20, 1, 2, 4)$	1-base	824	3972	12656	744	3572	10976	2	4
	canon	1784	11052	31856	1704	10652	30176	2	16
	small	824	3672	11816	744	3272	10136	2	5
	full	824	16752	122096	744	16352	120416	2	8
aes$(10, 1, 1, 8)$	1-base	504	89032	604896	424	88792	604176	2	9
	canon	5624	88132	258736	5544	87892	258016	2	256
	small	504	6952	41516	424	6712	40796	2	8
	full	504	1306672	20892336	424	1306432	20891616	2	16
aes$(2, 2, 2, 4)$	1-base	328	1484	4712	320	1444	4544	2	4
	canon	1032	6996	20200	1024	6956	20032	2	16
	small	328	1280	4064	320	1240	3896	2	5
	full	328	11176	86376	320	11136	86208	3	8
aes$(1, 2, 1, 8)$	1-base	192	18296	122840	184	18272	122768	2	9
	canon	3264	52532	154472	3256	52508	154400	2	256
	small	192	1696	9332	184	1672	9260	2	8
	full	192	783656	12534632	184	783632	12534560	2	16

Fig. 4. Statistics for DES/AES instances (without instantiation of P and C)

References

1. Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practices of Constraint Programming - CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.
2. Gregory V. Bard and Nicholas T. Courtois. Algebraic cryptanalysis of the Data Encryption Standard. In Steven D. Galbraith, editor, *Proceedings of the 11th IMA international conference on Cryptography and coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.
3. Jon T. Butler and Tsutomu Sasao. Boolean functions for cryptography. In Tsutomu Sasao and Jon T. Butler, editors, *Progress in Applications of Boolean Functions*, Synthesis Lectures on Digital Circuits and Systems, pages 33–53. Morgan & Claypool Publishers, 2010. ISBN: 9781608451814.
4. Carlos Cid, Sean Murphy, and Matthew Robshaw. *Algebraic Aspects of the Advanced Encryption Standard*. Springer, 2006. ISBN-10 0-387-24363-1.
5. Thomas W. Cusick and Pantelimon Stănică. *Cryptographic Boolean Functions and Applications*. Academic Press, 2009. ISBN 978-0-1237-4890-4.
6. Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer, Berlin, 2001. ISBN 3-540-42580-2; QA76.9.A25 D32 2001.
7. Deepak Kumar Dalai. *On Boolean Functions to Resist Algebraic Attacks: Some Necessary Conditions*. VDM Verlag Dr. Müller, 2010. ISBN: 978-3-639-26685-6; PhD thesis.
8. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, Berlin, 2004. Springer. ISBN 3-540-20851-8.
9. John Franco, Michal Kouril, John Schlipf, Sean Weaver, Michael Dransfield, and W. Mark Vanfleet. Function-complete lookahead in support of efficient SAT search heuristics. *Journal of Universal Computer Science*, 10(12):1655–1692, 2004.
10. Matthew Gwynne and Oliver Kullmann. Attacking DES + AES via SAT: Constraints and boolean functions. Technical Report arXiv:??? [cs.DM], arXiv, April 2011.
11. Philipp Jovanovic and Martin Kreuzer. Algebraic attacks using SAT-solvers. *Groups-Complexity-Cryptology*, 2(2):247–259, December 2010.
12. Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, Electronic Colloquium on Computational Complexity (ECCC), October 1999.
13. Oliver Kullmann. Investigating the behaviour of a SAT solver on random formulas. Technical Report CSR 23-2002, Swansea University, Computer Science Report Series (available from `http://www-compsci.swan.ac.uk/reports/2002.html`), October 2002. 119 pages.
14. Oliver Kullmann. Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 40(3-4):303–352, March 2004.
15. Oliver Kullmann. The `OKlibrary`: Introducing a "holistic" research platform for (generalised) SAT solving. *Studies in Logic*, 2(1):20–53, 2009.
16. Fabio Massacci and Laura Marraro. Logical cryptoanalysis as a SAT problem. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000 Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 343–375. IOS Press, Amsterdam, 2000. ISBN 1 58603 061 2.