Using Walk-SAT and Rel-SAT for Cryptographic Key Search

Fabio Massacci[*]
Dip. di Informatica e Sistemistica
Univ. di Roma I "La Sapienza"
via Salaria 113, I-00198 Roma, Italy
email: `massacci@dis.uniroma1.it`
url: `http://www.dis.uniroma1.it/~massacci`

Abstract

Computer security depends heavily on the strength of cryptographic algorithms. Thus, cryptographic key search is often THE search problem for many governments and corporations.

In the recent years, AI search techniques have achieved notable successes in solving "real world" problems. Following a recent result which showed that the properties of the U.S. Data Encryption Standard can be encoded in propositional logic, this paper advocates the use of cryptographic key search as a benchmark for propositional reasoning and search. Benchmarks based on the encoding of cryptographic algorithms optimally share the features of "real world" and random problems.

In this paper, two state-of-the-art AI search algorithms, Walk-SAT by Kautz & Selman and Rel-SAT by Bayardo & Schrag, have been tested on the encoding of the Data Encryption Standard, to see whether they are up the task, and we discuss what lesson can be learned from the analysis on this benchmark to improve SAT solvers.

New challenges in this field conclude the paper.

1 Introduction

Securing one's data and communication from unauthorized access in large open networks such as the Internet is of the main issues for computer science today [Anderson & Needham, 1996; G10, 1996; OECD, 1998].

Yet security depends heavily on the strength of cryptographic algorithms: security protocols which have been formally proved correct may be broken by the choice of a bad cipher [Ryan & Schneider, 1998]. Thus, cryptographic key search is often the search problem for many government and large corporations; and the ability of law enforcement officers to perform key search becomes the main concern behind the licensing of encryption technology [OECD, 1998].

Search, although in different settings, has also been a problem at the heart of AI research for many years. Recently propositional search has received attention for a number of factors [Selman *et al.*, 1997]:

> First new algorithms were discovered ... based on stochastic local search as well as systematic search [...]. Second, improvements in machine speed, memory size and implementations extended the range of the algorithms. Third, researchers began to develop and solve propositional encodings of interesting, real-world problems [...] Between 1991 and 1996 the size of hard satisfiability problems grew from ones involving less than 100 variables to ones involving over 10,000 variables.

Following a seminal proposal from [Cook and Mitchel, 1997], an application comes to one's mind: *can we encode cryptographic key search as a SAT-problem so that AI search techniques can solve it*?

A recent result in automated reasoning makes this possible. In [Marraro & Massacci, 1999] it has been show that, by combining clever reverse engineering, advanced CAD minimization, and propositional simplification, it is possible to encode in propositional logic the properties of the U.S. Data Encryption Standard, DES for short, [NIST, 1997; Schneier, 1994]. An encoding whose size is within reach of current AI search techniques: the encoding of a cryptographic search problem (finding a model is equivalent to finding a key) for the commercial version of DES requires slightly more than 10,000 variables and 6 times many clauses.

Although DES is currently under review, it is still the most widely used cryptographic algorithm within banks, financial institutions, and governments. It is the algorithm on which cryptanalysts tested the final success of their techniques (see [Schneier, 1994] or Sect. 2 for further references). Even partial successes with AI techniques can be relevant.

In this paper we claim that this problem should be one of the reference SAT-benchmarks. In particular, it gives the possibility of generating as many random instances as one wants and still each instance is as "real-world" as any instance that can be met in commercial cryptographic applications. It provides a neat answer to the last challenge for propositional reasoning and search proposed by Selman, Kautz and McAllester [1997] at IJCAI-97.

[*]Supported by CNR fellowship 201-15-9. This work has been partly supported by ASI, CNR and MURST grants. I would like to thank L. Carlucci Aiello, P. Liberatore, and L. Marraro for useful comments and discussions.

To check the potential effectiveness of AI techniques on this problem, two state-of-the-art SAT solvers have been tested for cryptographic key search using the propositional encoding. The choices are Walk-SAT, a local search algorithm proposed in [Selman *et al.*, 1994] as an improvement of GSAT, and Rel-SAT, a combination of the traditional Davis-Putnam Algorithm with back-jumping and learning proposed in [Bayardo & Schrag, 1997] to solve real-world problems.

In the experiments on the Data Encryption Standard, one shouldn't expect to be immediately competitive with twenty years of advanced cryptanalysis techniques, especially because AI Labs are not equally well funded to afford a specialized hardware machine of 250.000 USD or the exclusive use of a network of workstations for 50 days which have been used to break DES in the last years. Still, general purpose search algorithm using off-the-shelf hardware (Sparcs and Pentium II) can crack limited versions of DES without being told any problem-dependent information. Ad-hoc cryptographic techniques are still better since the first success with the limited version of DES we can solve was obtained in 1982 [Andleman & Reeds, 1982] and modern cryptographic approaches [Biham & Shamir, 1991; Matsui, 1994a] obtain the same results with better scaling properties. Still, the result is promising and points out at weaknesses of AI search algorithms that we need tackle to solve hard problems.

In the next section (§2) we introduce some basic preliminaries on cryptography and the Data Encryption Standard. Then we discuss the features of the encoding (§3). This is followed by the experimental analysis with Walk-SAT (§4) and Rel-SAT (§5). Few lessons for SAT solvers we can learn (§6) and new challenges (§7) conclude the paper.

2 Cryptography and DES

To make the paper self-contained for the non-security expert we sketch some preliminaries about cryptography and DES (for an introduction see [Schneier, 1994]).

Following [Schneier, 1994], we denote vector of bits by P (the plaintext), C (the ciphertext), and K (the secret key). At an abstract level, a cryptographic algorithm is simply a function $\mathrm{C} = \mathrm{E}_{\mathrm{K}}(\mathrm{P})$ that transforms a sequence of bits (the plaintext) into another sequence of bits (the ciphertext) with certain (desirable) properties by using some additional (possibly secret) bits K. To decrypt we use another function that maps back C into P using K (or its inverse).

The important property of the encryption algorithm is that *security of the algorithm must reside in the (secret) key*. If one does not know K, it must be difficult to recover P from C, even if the algorithm has been public for years. In the ideal case, the only way to recover the plaintext must be by brute force "generate-and-test": try out all possible keys and see which yields an acceptable plaintext. The need to hinder brute force attacks has therefore generated hot debates on the minimum size of a key [Schneier, 1994].

Exhaustive search is not so impossible as it seems if one can use (and pay for) specialized hardware: last year a machine costing 250.000 USD broke the Data Encryption Standard finding a 56 bits key in 56 hours [DES Search, 1998a].

Search can be cut down if the cryptanalyst knows a sufficient number of blocks of plaintext with the corresponding ciphertext (known plaintext attack). This is a reasonable hypothesis: almost all messages and files have fixed parts. Using a network of 12 workstation and 2^{47} (randomly generated) plaintexts, Matsui [1994a] broke DES in 50 days.

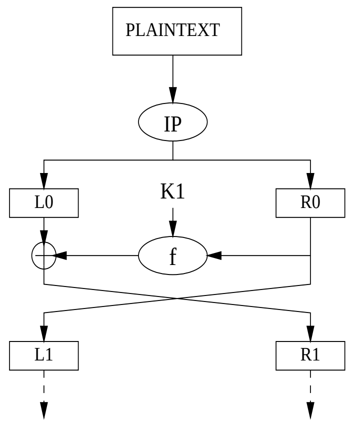

Figure 1: DES Algorithm

As the reader might now want to know how DES works, we start by saying that DES is a block cipher, which encipher *blocks* (sequences) of 64 bits into blocks of 64 bits using a key of 56 bits[1]. DES and almost all symmetric ciphers are built following an architecture which is due to Feistel and his group [Feistel *et al.*, 1975]. After some initial preprocessing, the following operations are executed:

1. break the plaintext in two halves,
2. combine one half with the key using a clever function,
3. XOR the combination with the other half
4. swap the two parts.

These 4 operations constitutes a *round* and are repeated a suitable number of times. Figure 1 exemplifies the idea.

DES has 16 rounds which are almost identical except for the way in which the key is fed into the f function (Fig. 1): for each round a different subset of the 56 keybits is selected and combined with the input of the previous round. The strength of the cipher depends on the number of rounds and on f. Its design is, to quote Ron Rivest, "part art, part science".

As we have mentioned already, the basic way to break DES is by exhaustive search but there are other techniques.

Differential cryptanalysis was introduced by Biham and Shamir [1991]. It assumes that the cryptanalyst can choose ciphertext and plaintext pairs presenting particular fixed differences. Then, it analyzes the evolution of these differences through the rounds of DES. Using the differences resulting from ciphertexts, different probabilities are assigned to different keys. By analyzing a large number of ciphertext and plaintext pairs (2^{47} for the commercial version), a key will emerge as the most probable. This attack is only practical for less than 12 rounds. After that, it requires too many resources.

[1]The key is usually expressed as a 64 bits number, in which every eighth bit is a parity bit ignored by the algorithm.

Matsui's *linear cryptanalysis* [Matsui, 1994a; 1994b] works better. This method uses linear approximations (xor) to describe the behavior of the round function f (Fig. 1). By xoring together some plaintext bits with some ciphertext bits, one can get a bit that is the xor of some key bits. This is a linear approximation of a round that is true with a certain probability. Again, by analyzing a large number of plain/ciphertext pairs (2^{43} are needed for DES at 16 rounds), it is possible to guess the value of some key bits with 80% success rate. A refinement of this method approximates the internal 14-round and then guesses the results of the first and last round. It can find 26 keybits and uses exhaustive search for the rest.

A key aspect of cryptanalytic attacks (beside brute force) is that they are probabilistic. No deterministic method is known.

3 DES as a SAT Problem

Recently, an encoding of the U.S. Data Encryption Standard in propositional logic has been proposed in [Marraro & Massacci, 1999]. Before discussing how the encoding can be used to generate random problems, we sketch its functioning:

- each bit of the ciphertext, the plaintext, and the key is encoded as a propositional variable;
- the operations corresponding to the round function f (Fig. 1) are transformed into boolean formulae and minimized off-line with CAD tools;
- then the encoding algorithm "runs" DES at the meta-level, and generates formulae corresponding to each DES operation on the way;
- since the straightforward application of this method would generate a huge formula, clever optimizations are used so that some operations are encoded as formulae and some operations are computed.

For instance, operations corresponding to permutations of bits are not encoded as formulae; rather the propositional variables describing the inputs are permuted. Further details can be found in [Marraro & Massacci, 1999].

The outcome[2] of the algorithm is a formula $\mathcal{E}(\mathrm{P}, \mathrm{K}, \mathrm{C})$ which represent the logical relations between the key bits K, the plaintext bits P and the ciphertext bits C.

In a traditional plaintext attack we know the value of some plaintext and ciphertext bits so, if we replace the variables by the corresponding truth value, we have a formula whose structure is shown in Fig. 2. The $\mathrm{K_i}$ are the key bits while the other variables $\mathrm{M_i^r}$, $\mathrm{S_i^r}$, $\mathrm{X_i^r}$ are introduced to denote intermediate results and make the formula simpler. We use the superscripts r to denote the results produced at the r-th round and the subscript i to the denote the i-th bit produced at corresponding intermediate stage (i ranges from 1 to 64). Loosely speaking, and looking at Fig. 1, we may say that each X_i^r represents an output of the r-th round and thus an input of the $r + 1$-th round of the algorithm. The value $lastr$ is the number of rounds of DES for which the encoding has been done. The actual formulae have more definitions to ease the subsequent (polynomial) translation into CNF.

[2]The algorithm in [Marraro & Massacci, 1999] takes less than 1 sec (3rounds) up to 25 seconds (16 rounds) to generate the encoding. Memory requires a peak of 135M for the full 16 rounds.

Figure 2: The Encoding for a known-plaintext attack

Definitions

$$\begin{array}{rcll}
\mathrm{M_i^r} & \Leftrightarrow & \bigwedge_j \pm\mathrm{X_j^r} & 2 \le r \le lastr - 1 \\
\mathrm{S_i^r} & \Leftrightarrow & \bigvee_j \mathrm{M_j^r} & 2 \le r \le lastr \\
\mathrm{X_i^{r+1}} & \Leftrightarrow & \mathrm{S_j^r} \oplus \mathrm{K_h} & 1 \le r \le lastr - 1
\end{array}$$

Constraints

$$\begin{array}{rcll}
\mathrm{M_i^1} & \Leftrightarrow & \bigwedge_j \pm\mathrm{K_j} & \\
\mathrm{M_i^{lastr}} & \Leftrightarrow & \bigwedge_j \pm\mathrm{K_j} & \\
\pm\mathrm{S_i^{lastr-1}} & \Leftrightarrow & \bigoplus_r \mathrm{S_j^r} & r \text{ even} \\
\pm\mathrm{S_i^{lastr}} & \Leftrightarrow & \bigoplus_r \mathrm{S^r} & r \text{ odd}
\end{array}$$

Table 1: Formula size per single plain/ciphertext pair

R	$\Leftrightarrow$	$\oplus$	$\wedge$	$\vee$	Clauses	Vars	C/V
1	520	0	504	16	1645	243	6.77
2	1042	0	1010	32	3304	489	6.75
3	1738	48	1609	80	9064	1430	6.34
4	2432	96	2208	128	14811	2368	6.25
5	3096	176	2760	160	18738	3032	6.18
6	3760	256	3312	192	22665	3696	6.13
7	4424	336	3864	224	26592	4360	6.10
8	5088	416	4416	256	30519	5024	6.07
9	5752	496	4968	288	34446	5688	6.06
10	6416	576	5520	320	38373	6352	6.04
11	7080	656	6072	352	42300	7016	6.03
12	7744	736	6624	384	46227	7680	6.02
13	8408	816	7176	416	50154	8344	6.01
14	9072	896	7728	448	54081	9008	6.00
15	9736	976	8280	480	58008	9672	6.00
16	10400	1056	8832	512	61935	10336	5.99

Table 1, taken from [Marraro & Massacci, 1999], shows some quantitative data for the encoding of a single pair constituted by a known block of plaintext and a block of cipher text, for an increasing number of rounds (R).

For random formulae, the ratio of c/v is an indicator of the hardness of the formula. In this case, it is not so. For instance, using the data of Table 1 for 3 rounds or more, we can see that if we use one block or an "infinite" number of blocks, the value of c/v changes by less than 4%. This would seem to imply that adding more blocks should not make the problem neither much easier nor much harder. As we shall see, the experimental results contradict this hypothesis.

In the introduction, it has been claimed that this encoding can be used to combine the contrasting needs of using "real-world" problems (possibly with lot of structure) and of generating a huge number of instances which can only be (pseudo)randomly generated. It might solve the dilemma pointed out in [Bayardo & Schrag, 1997]:

> Care must be taken when experimenting with real world instances because the number of instances available for experimentation is often limited

whereas [Crawford & Auton, 1996] noted that

> [...] random problems are readily available in any given size and virtually inexhaustible numbers. For example, ...[their experiments] required several million problems and it is hard to imagine collecting that many problems any other way.

How do we generate a random SAT problem based on cryptography? At first we generate a random key $\mathrm{v_K}$ and a plaintext block $\mathrm{v_P}$ (just vectors of 0/1). Then we use the cryptographic algorithm itself to get $\mathrm{v_C}$=$\mathrm{E_{v_K}(v_P)}$. Finally we substitute in $\mathcal{E}(\mathrm{P,K,C})$ the corresponding boolean values $\mathrm{v_P}$ and $\mathrm{v_C}$ that we have so far generated. Then the pair $\langle \mathrm{v_K}, \mathcal{E}(\mathrm{v_P}, \mathrm{K}, \mathrm{v_C})\rangle$ is a *solved instance* of the SAT problem. Notice that $\mathcal{E}(\mathrm{v_P}, \mathrm{K}, \mathrm{v_C})$ might contain other variables than K but the latter are the only independent variables. If we have n blocks of plaintext (and the corresponding ciphertext) we can constrain the search further by conjoining the corresponding formulae $\bigwedge_{i=1}^{n} \mathcal{E}(\mathrm{v_P^i}, \mathrm{K}, \mathrm{v_C^i})$.

So we have encoded cryptographic key search, a known plaintext attack to DES, as a SAT problem. Since ciphers are designed to be hard to break, this will provide us with the hard solved instances asked for in [Cook and Mitchel, 1997]. We can generate generic instances by randomly generating both the plaintext and the ciphertext.

The main point here is that by changing the plaintext and the key we can generate an endless stream[3] of different solved instances either with the same solution or with different solutions. At 16 rounds, it would be exactly identical to an actual plaintext and ciphertext used by a bank, financial institution or government department.

4 Walk-SAT on DES

The first tested algorithm is a local search one: Walk-SAT [Selman *et al.*, 1994]. It is only briefly recalled:

- the algorithm starts from a random assignment;
- then it flips the value of some propositional variables (usually one) trying to increase the value of an objective function (here the number of satisfied clauses);
- when a local minimum is reached, the algorithm restart the search with another random assignment.

Variants of this basic approach include the possibility of making random moves from time to time and of continuing the search after a local minimum by using a tabu-list[4]. For more details see [Selman *et al.*, 1994; 1997].

Experiments were run on a Pentium II running Linux (with 64MB) and a Sun Sparc running Solaris (with 64M) with qualitatively and quantitatively similar results. To generate an instance we simply follow the recipe above: generate randomly a key (discarding weak keys [Schneier, 1994]) and then some hundred blocks of plaintext. For each plaintext block we generate the corresponding ciphertext and then substitute the value of the pair in the formula. An instance is finally created by conjoining the formulae corresponding to the desired number of plain/ciphertext pairs (or blocks).

The initial settings of Walk-SAT were the recommend standard: hill-climbing with some random perturbations. The performance improved by using a random flip every 16 moves and the final result is reported in Table 2. R denotes the number of rounds on which DES has been limited and B the number of blocks which have been conjoined to produce the instance (to get the size of an instance, multiply the values of Table 1 for the number of blocks). Sec. is the average running time and Kbits tells on average how many bits of the solution found by Walk-SAT coincided with the known solution. For unsuccessful attempts we also report the lowest number of unsatisfied clauses found.

Table 2: Performance of Walk-SAT

R	B	% Succ	Kbits	Sec	# Bad
1	1	100%	31.4	0.03	-
1	2	100%	46.8	0.53	-
1	4	100%	52	1.67	-
1	8	100%	52.4	7.24	-
2	1	100%	53.8	21.8	-
2	2	100%	55.8	2.63	-
2	4	100%	56	3.16	-
2	8	100%	56	4.29	-
3	1	0%	-	1227.20	15.4
3	2	0%	-	3695.23	35.7

Walk-SAT can crack DES up to 2 rounds, and compares favorably with the results of SATO and TABLEAU reported in [Marraro & Massacci, 1999]. At three rounds Walk-SAT cannot crack any instance, even with a number of flips hundreds times the number of clauses and a few hundreds tries. Moreover, adding more constraints (blocks) makes the search harder and not easier.

Why doesn't Walk-SAT solve the problem well?

The first problem has been already pointed out in [Selman *et al.*, 1997]: the difficulty of local search algorithms to run around *dependent* variables. Recall that here almost all variables are dependent. The dagsat approach proposed in [Kautz *et al.*, 1997] might prove to be more successful.

The second problem is the presence of wide "rugged" plateaus at the bottom of the search space: the number of unsatisfied clauses goes quickly down from thousands to few tens per block and stays there, with Walk-SAT flipping (in vain) a lot of dependent variables and moving from a local minima to the next. The lowest number of bad clauses was decreased by re-engineering Walk-SAT as follows:

- the first time a local minimum is reached, its value is stored as a reference value and the search continues;
- after the search visited n local minima with value higher than the reference value, all keybits were randomly flipped (with a probability $\frac{1}{n}$ all variables were flipped);
- each time a lower minimum was reached, n was reset and that minimum considered the new reference value;

The idea was to escape the plateaus by exploiting the domain knowledge that the keybits were the only independent variables. In this way, the search converges to a much lower value of bad clauses (usually from 40-100 bad clauses per block to less than 10), but we are still stuck there.

[3]For DES we have $2^{56} \times 2^{64}$ instances if we consider the encryption of binary data. If we restrict ourselves to ASCII plaintexts, the number of different plaintexts only shifts from 2^{64} to 2^{56}.

[4]A tabu-list is a list of variables which have just been flipped and which cannot be immediately re-flipped

Table 3: Performance of Rel-SAT

		General			No Learning		
R	B	Kbit	Branch	Sec	Kbit	Branch	Sec
1	1	31	28	0.02	-	-	-
1	2	49	104	0.11	-	-	-
1	4	51	104	0.22	53	112	4.44
1	8	52	83	0.45	53	184	6.18
2	1	54	20841	32.43	-	-	-
2	2	56	40122	111.15	-	-	-
2	4	56	4050	18.39	56	157	4.98
2	8	56	57	0.81	56	103	8.00
3	1	-	-	$\geq$ 1h	-	-	
3	2	56	173075	976.28	-	-	
3	4	56	19311	159.13	56	75538	$\geq$ 1h
3	8	56	3594	75.02	56	8153	822

5 Rel-SAT on DES

The second algorithm is a systematic one: Rel-SAT from [Bayardo & Schrag, 1997]. It is a variant of the Davis-Putnam algorithm, enhanced with conflict directed back-jumping and learning. It works as follows:

- unit propagation is applied to the clause set;
- if no contradiction is found a new literal is selected and added either positively or negatively to the clause set;
- if a contradiction is found then the algorithm backtracks to the literal that caused the contradiction;
- the clause responsible for the contradiction is resolved with a clause representing the temporary assignment; the resolvent is thus learned as a reason to avoid the corresponding assignment;
- the procedure is iterated until all literals have been assigned (SAT) or no backtrack is possible (UNSAT).

For more details see [Bayardo & Schrag, 1997].

The instance generation method, the architecture, and operating systems were the same used for Walk-SAT. Also in this case, the experiment started with the recommend standard: a small learning factor (4), using relevance-based learning.

Up to 2 rounds, Rel-SAT cracks DES only slightly faster than SATO and TABLEAU (see [Marraro & Massacci, 1999]) or Walk-SAT. However, it is the only algorithm which cracks three round of DES in less than ten minutes. Its performance is reported in Table 3. The success rate is omitted since either all instances could be solved or none could (-).

Other settings were tried: no learning at all and learning factors larger than 4. The analysis shows that learning is essential if we have few constraints but it might be skipped if enough constraints are around (Table 3). An intuitive explanation could be that with few blocks the algorithm might split on dependent variables and then discover that this was not necessary. With many constraints, standard heuristics select almost only independent variables and therefore learning contribution to performance is diminished.

Note that the performance of the algorithm improves with the number of blocks composing the instance. Adding more constraints makes the search for the only (?) existing solution easier. This behavior is consistent with standard cryptographic techniques [Biham & Shamir, 1991; Matsui, 1994a] where having more data improves the chances of success.

Given this promising results, the algorithm has been engineered to accept larger formulae with more variables and tried on the full 16-round DES, also using 1, 2, and 4 blocks. The algorithm didn't return within 12 hours.

A small re-engineering of the algorithm was carried to exploit in a limited manner the knowledge of the domain. After a first selection of potential branching variables, a threshold is used in the original algorithm to reduce their number. The modified algorithm didn't check the threshold if the selected variable was a keybit. In this way the algorithm gives preferences to dependent variable with very good properties or independent variables with medium properties. However, the running time of the algorithm didn't improved substantially.

6 Lessons for SAT Solvers

It is a promising result that SAT solvers can crack a limited version of DES without using any problem-dependent heuristics but this is not enough. We want to solve the full DES.

The first temptation is then to dismiss the SAT solvers themselves: this problem has a non-clausal structure, so it has to be expected that CNF provers perform badly; the right tool should have been BDDs [Bryant, 1986]. Surprisingly, an extensive experimentation reported in [Ascione, 1999] shows that the BDDs cannot solve key search problems any better than SAT-based approaches.

The second conclusion might be that the problem is too constrained: at three rounds there is almost only one solution. This makes the problem harder for local search, but should make it easier for complete algorithms. Indeed, the very characteristics of DES with its avalanche effect (all keybits should affect all ciphertext bits, and a flip in the plaintext should affect all ciphertext bits, etc.) should make this problem easy: if the value of few keybits is wrongly chosen, a cascade of unit propagations should immediately generate an inconsistency. This would implies that formulae encoding more rounds (and more defined variables) should be easier and not harder. Since this is not the case, it seems that with more rounds unit propagation is somehow hindered.

Finding an explanation (and a workaround) for this difficulty is important because the structure of the encoding is common in many hard real problems: the natural formulation of a problem is usually structured in layers, makes use of abbreviations and definitions, and often contains modulo-2 arithmetics (xors). For instance see the parity bit problem mentioned in [Selman *et al.*, 1997] and the IFIP benchmark for hardware verification.

If we look again at the structure of the encoding, we may notice that each round is separated by the next round by a level of xors and that most constraints are in form of xors: a large subpart of the problem is an affine problem, which should be polynomially solvable by Schaefer's theorem. It is precisely this affine subproblem that make the problem hard for current AI techniques. Look again at table 1: the problem becomes difficult as soon as xors start to appear.

In contrast, cryptographic techniques exploits this affine subproblem and even approximate the whole problem into an affine problem [Matsui, 1994a]. Therefore, to crack DES or similar real-word problem *a SAT solver needs the ability to solve affine subproblems*.

7 Conclusions and Future Challenges

In this paper we have seen an application of propositional reasoning and search to a key security problem of industrial relevance. We have also discussed how this approach can optimally provide "real-world" problems where many instances can be randomly generated.

Thus we believe that the whole approach on encoding cryptographic key search as propositional search can be a good answer to the final challenge proposed in [Selman *et al.*, 1997]:

> Develop a generator for problem instances that have computational properties that are more similar to real world instances.

Moreover, the preliminary tests on using SAT-solvers to crack the Data Encryption Standard are promising, although SAT-solvers must be improved to meet the full challenge provided by this benchmark. Thus, a good conclusion of this paper may just be the indication of the future challenges. They are listed in order of feasibility.

The first challenge is to find a key for the commercial 16 rounds Data Encryption Standard in less than 56 hours using off-the-shelf h/w and s/w but specialized search heuristics. This might be the simplest and immediately rewarding challenge, assuming that the 10,000 USD prize of RSA Security for breaking its DES challenges will be there in the year 2000.

Then we may wish to design SAT-solvers that work with every Feistel-type cipher with data independent rounds like DES. If we were able to cope with affine subproblems this would not be a big extension. Since the operations are data independent, a certain amount of preprocessing for the internal rounds could be done off-line.

The third challenge is to find efficient encodings into SAT of data-dependent Feistel-type ciphers like RC5 [Schneier, 1994]. A straightforward encoding is always possible: just translate the cipher into a circuit and this into propositional logic. Unfortunately this is already unworkable for DES.

Last but not least, we may wish to find efficient encodings into propositional (or any) logic of public-key algorithms such as RSA. This challenge, firstly proposed in [Cook and Mitchel, 1997], might prove to be the hardest since number theory is fairly remote from propositional logic.

As for all real problems, there might also be a darker side: the final measure of success might well be the "privilege" (!?) of successful SAT algorithms being denied export licenses as dangerous weapons.

References

[Anderson & Needham, 1996] R. Anderson & R. Needham. Programming satan's computer. In *Computer Science Today - Recent Trends and Developments*, LNCS 1000, pp. 426–440. Springer-Verlag, 1996.

[Andleman & Reeds, 1982] D. Andleman & J. Reeds. On the cryptanalysis of rotor machines and substitution-permutations networks. *IEEE Trans. on Inf. Theory*, 28(4):578–584, 1982.

[Ascione, 1999] M. Ascione. Validazione e benchmarking dei BDD per la criptanalisi del DES. Master's thesis, Facoltà di Ingegneria, Univ. di Roma I "La Sapienza", April 1999. In Italian.

[Bayardo & Schrag, 1997] R. Bayardo & R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. in *Proc. of AAAI-97*. pp. 203–208. AAAI Press/The MIT Press, 1997.

[Biham & Shamir, 1991] E. Biham & A. Shamir. Differential cryptanalisis of DES-like cryptosystems. *J. of Cryptology*, 4(1):3–72, 1991.

[Bryant, 1986] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE TOC*, 35(8):677–691, 1986.

[Crawford & Auton, 1996] J. Crawford & L. Auton. Experimental results on the crossover point in random 3SAT. *AIJ*, 81(1-2):31–57, 1996.

[Cook and Mitchel, 1997] S. Cook & D. Mitchel. Finding hard instances of the satisfiability problem: A survey. In *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discr. Math. and TCS, 25:1–17. AMS, 1997.

[DES Search, 1998a] DES key search project information. Technical report, Cryptography Research Inc., 1998. Available on the web at `http://www.cryptography.com/des/`.

[Feistel *et al.*, 1975] H. Feistel, W. Notz, and L. Smith. Some cryptographic techniques for machine-to-machine data communication. *Proc. of the IEEE*, 63(11):1545–1554, 1975.

[G10, 1996] Committee on Payment, Settelment Systems, and the Group of Computer Experts of the Central Banks of the Group of Ten countries. *Security of Electronic Money*. Bank for International Settlements, Basle, August 1996.

[Kautz *et al.*, 1997] H. Kautz, D. McAllester, and & B. Selman. Exploiting Variable Dependency in Local Search Abstract in *Abstracts of the Poster Sessions of IJCAI-97*, 1997.

[Marraro & Massacci, 1999] L. Marraro & F. Massacci. A new challenge for automated reasoning: Verification and cryptanalysis of cryptographic algorithms. Tech. Rep. 5, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza", 1999.

[Matsui, 1994a] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Proc. of CRYPTO-94*, LNCS 839, pp. 1–11. Springer-Verlag, 1994.

[Matsui, 1994b] M. Matsui. Linear cryptanalysis method for des cipher. In *Proc. of Eurocrypt 93*, LNCS 765, pp. 368–397. Springer-Verlag, 1994.

[NIST, 1997] NIST. Data encryption standard. Federal Information Processing Standards Publications FIPS PUB 46-2, National (U.S.) Bureau of Standards, Dec 1997. Supersedes FIPS PUB 46-1 of Jan. 1988 and FIPS PUB 46 of Jan. 1977.

[OECD, 1998] OECD. Emerging market economy forum (EMEF): Report of the ministerial workshop on cryptography policy. OLIS SG/EMEF/ICCP(98)1, Organization for Economic Co-operation and Development, Paris, Feb 1998.

[Ryan & Schneider, 1998] P. Ryan & S. Schneider. An attack on a recurive authentication protocol: a cautionary tale. *IPL*, 65(15):7–16, 1998.

[Schneier, 1994] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, 1994.

[Selman *et al.*, 1994] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. of AAAI-94*, pp. 337–343. AAAI Press/The MIT Press, 1994.

[Selman *et al.*, 1997] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional resoning and search. In *Proc. IJCAI-97*, pp. n50-54. Morgan Kaufmann, 1997.