Towards the Formal Verification of Ciphers: Logical Cryptanalysis of DES*

Fabio Massacci[†1,2] and **Laura Marraro**[1]

[1] Dip. di Informatica e Sistemistica
Univ. di Roma "La Sapienza" -Italy

[2]Dip. di Ingegneria dell'Informazione
Univ. di Siena - Italy

Abstract

Providing formal assurance of correctness is a key issue for cryptographic algorithms. Yet, automated reasoning tools have only been used for the verification of security protocols, and almost never for the verification and cryptanalysis of the cryptographic algorithms on which those protocols rely.

We claim that one can use logic for encoding the low-level properties of state-of-the-art cryptographic algorithms and then use automated theorem proving for reasoning about them. We call this approach *logical cryptanalysis*. In this framework, finding a model for a formula encoding an algorithm is equivalent to finding a key with a cryptanalytic attack. Other important properties, such as algebraic closure can also be captured.

Here we present a case study on the U.S. Data Encryption Standard (DES) with 56-bits keys and discuss how to obtain a manageable encoding of its properties and how SAT provers perform on our encoding of the DES.

1 Introduction

Providing computer security in large open networks such as the Internet is one of the frontiers of computer science today [Anderson and Needham, 1996; G10, 1996]. Yet, this task is not so simple and, notwithstanding the amount of research and development work, the literature is full of "how to break a secure..." examples [Abadi and Needham, 1996; Lowe, 1996; Ryan and Schneider, 1998].

Thus, the verification of security protocols by automated reasoning tools is becoming a must, also given its notable results with theorem proving [Paulson, 1998] and model checking [Lowe, 1996; Mitchell *et al.*, 1997]. However, such verification only takes a high level view of cryptographic algorithms and it may happen that a protocol can be proven formally secure and still be broken because the cipher chosen for its implementation has unwanted algebraic properties [Ryan and Schneider, 1998].

So, the use of automated reasoning tools for the verification of cryptographic algorithms seems to be the next natural step. Yet, looking at the field of cryptanalysis, we find little or no use of such tools. Sophisticated and successful techniques such as linear cryptanalysis [Matsui, 1994b; 1994a] or differential cryptanalysis [Biham and Shamir, 1991] use only statistical tools for solving cryptanalysis problems. Other properties of ciphers, such as algebraic closure, are determined by ad-hoc methods or experiments [Campbell and Weiner, 1992; Kaliski *et al.*, 1985]. In absence of formal verification, the properties of many cryptographic algorithms are thus subject to intense debates and speculations[1].

Thus, a new field of potential application stems to one's mind:

- can we encode low-level properties of ciphers into logical formulae?
- can we do it in a way so that finding a model of the encoded formula is equivalent to finding a key?
- can other problems, such as cipher integrity, existence of trapdoors, etc. be also automatically verified using this encoding?
- last, but not least, can we get a *feasible encoding*, which might give hard-to-analyze formulae but not overwhelming?

In this paper, we advocate that logic and automatic reasoning tools can be efficiently used to model and verify state-of-the-art cryptographic algorithms. We call this method *logical cryptanalysis*.

*We would like to thank L. Carlucci Aiello, M. Ascione and M. Fabriani for many useful discussions. The first author acknowledges the support of the CNR fellowship 201-15-9 held at the University of Rome I "La Sapienza" Dipartimento di informatica e Sistemistica. This work has been partly supported by the DII-AMTEC convention on analysis and design of high speed cryptographic ciphers.

†Communicating author: Dip. Ing. Informazione, via Roma 56, I-53100 Siena, Italy, email massacci@dii.unisi.it

[1]See e.g. [Schneier, 1994] for a survey of the long lived debate on whether the Data Encryption Standard has hidden trapdoors which would allow the US National Security Agency to decrypt all traffic.

To make this claim concrete, we show that by combining clever reverse engineering, advanced CAD minimization, and propositional simplification, it is possible to give a feasible encoding in propositional logic of the properties of the U.S. Data Encryption Standard, DES for short, [DES, 1997; Schneier, 1994]. The Data Encryption Standard, designed by IBM and NSA in the 70s, is the current U.S. Government standard, has been adopted for many financial and banking institutions, and is the recommended standard of the international banking transaction services. Although DES is currently under review [AES, 1998], its widespread use and the fact that its security has been the subject of an intense scrutiny since its inception [Biham and Shamir, 1991; Campbell and Weiner, 1992; Matsui, 1994b; 1994a; Schneier, 1994] makes the case study significant.

Our encoding of DES is at the border of tractability for current search techniques [Johnson and Trick, 1996]. For example, the encoding of a cryptographic search problem (finding a model is equivalent to finding a key) for the commercial version of DES (the one with 56-bits keys) requires slightly more than 60,000 clauses and 10,000 variables (out of this only 56 were independent search variables).

To check the potential effectiveness of theorem proving techniques on this problem, we have used state-of-the-art SAT provers and BDD tools for cryptographic key search with our encoding. In the experiments on the Data Encryption Standard, we didn't expect to be immediately competitive with twenty years of advanced cryptanalysis techniques. At first 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 12 workstations for 50 days which have been used to break DES in the last few years [Cryptography Research, 1998; Matsui, 1994a]. Second, because we wanted to test the ability of "generic" theorem provers, without giving them problem dependent information. We were pleasantly surprised by the result: general purpose search algorithm using off-the-shelf hardware (Sparcs and Pentium II) can solve limited versions of DES without being told any information on the structure of the problem[2]. Although the results are encouraging, there is a lot of research work that needs to be done since DES beyond 8 rounds is still out of reach for theorem proving methods.

We believe that this approach might be beneficial for both the automated reasoning and the computer security communities.

For the former, it provides a set of challenging problems of industrial relevance ranging from satisfiability and validity in propositional logic and to validity in quantified boolean logic. Thus we claim that this problem should be one of the reference benchmarks for propositional reasoning and search.

[2]The first success on this limited version of DES with standard cryptographic techniques was obtained in 1982 by ad-hoc techniques [Andleman and Reeds, 1982], and modern cryptanalysis techniques [Biham and Shamir, 1991; Matsui, 1994a] can also quickly solve it with a personal computer, although they need more plaintexts.

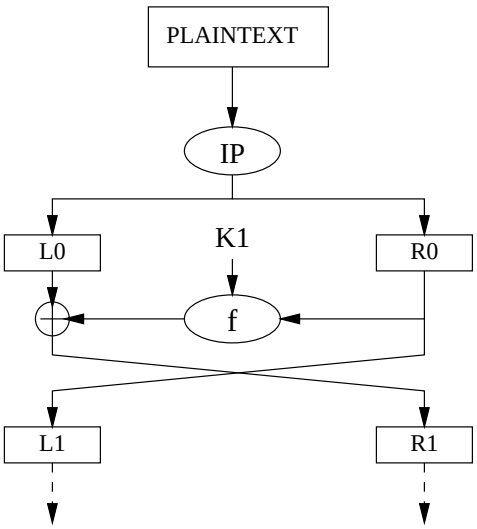

Figure 1: A round of DES

For the security community, the formalization of the property of a cipher using logic might offer a simple solution to many issues regarding the strength of a cryptographic algorithm. Consider the hot debate about the existence of hidden trapdoors. The formal proof that there is no universal key to escrow a cipher might be provided together with the algorithm. The proof might then be quickly machine checked for correctness by suspicious users. Even if finding the proof takes a long time, this can be done off-line once and for all.

In the rest of the paper we present the general idea behind logical cryptanalysis (§2) and how different properties can be captured by this approach once we have encoded the Data Encryption Standard into propositional logic(§3) for which we also presents some quantitative data. Last we discuss the results of preliminary experiments (§4) on the usage of the automated reasoning tools for cryptographic key search and conclude (§5).

Throughout the paper we will assume that the reader has some basic knowledge of cryptography and of the working of DES. For a comprehensive introduction to the subject we refer to [Schneier, 1994] and to [DES, 1997] for the official specification of the Data Encryption Standard. In Fig. 1 we just give a pictorial representation of a round of DES. The input is a block of 64 bits (the plaintext) and the output is another block of 64 bits (the ciphertext), the key is formed by a block of 56 bits. The full DES is obtained by sequentially combining 16 of those rounds in sequence.

2 Logical Cryptanalysis

The main intuition behind logical cryptanalysis is that we should view each bit sequence $\mathbf{P}$, $\mathbf{C}$, $\mathbf{K}$ as a sequence of *propositional variables* P, C, K, which are true when the corresponding bit is 1 and false when the bit is 0.

Then we simply need to *encode the properties of the cryptographic algorithm with a logical formula* $\mathcal{E}(P, K, C)$, which is true if and only if for the corresponding sequences of bits we have that $\mathbf{C} = \mathrm{E}_{\mathbf{K}}(\mathbf{P})$ holds. Propositional logic is the straightforward choice but temporal or first order logic might be more compact.

Note that if $\mathcal{E}(P, K, C)$ holds and the cipher is symmetric (i.e. the same key is used for encryption and decryption), there is no guarantee that $\mathcal{E}(C, K, P)$ holds too. Indeed, even with symmetric ciphers, the functional or operational description of encryption is not completely identical to that of decryption. For instance, for DES decryption we need to invert the key scheduling algorithm [DES, 1997; Schneier, 1994]. Therefore the formula specifying decryption will often be different from the the formula specifying encryption with swapped arguments.

To model *key search*, let v_C be the truth values (true/false) of the desired ciphertext. To find a key with a *ciphertext only-attack*, it is enough to find a model of $\mathcal{E}(P, K, v_C)$. In a *known plaintext attack*, we also know the values of the propositional variables P. So, if v_P are the truth values of P then we have only to search for a model of $\mathcal{E}(v_P, K, v_C)$.

If we have n plaintext and ciphertext pairs $({v_P}^i, {v_C}^i)$ we can constrain the search further by conjoining the corresponding formulae as $\bigwedge_{i=1}^{n} \mathcal{E}({v_P}^i, K, {v_C}^i)$. Finding an assignment v_K which satisfies this formula is then equivalent to break the cipher with a known plaintext attack using those n pairs.

For reason of efficiency we may need to introduce more variables in $\mathcal{E}(P, K, C)$ besides C, P and K to make use of abbreviations and definitions (we have done it in the encoding of DES in §3). However, if the encoding is well made, K, C and P should be the only *independent variables* of the problem, i.e. fixing their values should determine the values of all other variables. Thus, in $\mathcal{E}(v_P, K, v_C)$ the interesting variables are only K. For instance, in the case of the DES encoding we have only 56 control variables.

Other interesting properties can be captured if we use validity or quantified boolean formulae (QBF) for which efficient algorithms also exists [Büning *et al.*, 1995; Cadoli *et al.*, 1998].

If we use QBF, finding a key corresponds to a constructive proof of

$$\exists K. \left(\bigwedge_{j=1}^{n} \mathcal{E}({v_C}^j, K, {v_P}^j) \right)$$

The first property we might wish to prove is the *absence of weak keys*. Recall that a key is weak if for all plaintexts $\mathbf{P}$ one has $\mathrm{E}_{\mathbf{K}}(\mathrm{E}_{\mathbf{K}}(\mathbf{P})) = \mathbf{P}$. In other words a key is weak if we end up with the original plaintext whenever we encrypt it twice. This can be captured by the following formula, where we use C to denote the result of the first encryption:

$$\neg\exists K. \forall P \forall C. \left(\mathcal{E}(P, K, C) \Rightarrow \mathcal{E}(C, K, P)\right)$$

This property can be restricted to particular instances of v_C and v_P and then it simply becomes a problem of validity in propositional logic.

The property that a cipher is *faithful* [Kaliski *et al.*, 1985] can be expressed as a simple propositional formula:

$$\mathcal{E}(P, K, C) \wedge \mathcal{E}(P, K', C) \Rightarrow \left(\bigwedge_i K_i \Leftrightarrow {K'}_i \right)$$

The *existence of an universal key* to decrypt all traffic can be formalized:

$$\exists K^u. \forall K. \forall P \forall C. \left(\mathcal{E}(P, K, C) \Rightarrow \mathcal{E}(P, K^u, C)\right)$$

The negation of this formula is more interesting, i.e. in proving that no universal key exists. Of course, if a cipher is faithful then no universal key exists.

Proving that *a cipher is not closed* [Campbell and Weiner, 1992; Kaliski *et al.*, 1985], is slightly more complicated but can still be captured. In a nutshell we have to express the fact that if we encrypt things twice, once with a key $\mathbf{K^1}$ and then with a key $\mathbf{K^2}$ obtaining $\mathrm{E}_{\mathbf{K^2}}(\mathrm{E}_{\mathbf{K^1}}(\mathbf{P})) = \mathbf{C}$ we obtain a better protection than by using a single encryption, i.e. we cannot obtain $\mathrm{E}_{\mathbf{K^3}}(\mathbf{P}) = \mathbf{C}$ by suitably choosing $\mathbf{K^3}$. If we use C^1 to denote the result of the first encryption with K^1 and C to denote the final result of the second encryption with K^2, then the following formula captures this property:

$$\begin{array}{l} \forall K^1 \forall K^2. \\ \quad \forall P \forall C^1 \forall C. \\ \qquad \left(\mathcal{E}(P, K^1, C^1) \wedge \mathcal{E}(C^1, K^2, C)\right) \\ \qquad\quad \Longrightarrow \neg\exists K^3. \mathcal{E}(P, K^3, C) \end{array}$$

Note that the size of the theorem is not much bigger (3 times) than the size of the formula obtained for the modelling of a key search problem. Moreover, it can also be casted in a simpler problem of propositional validity by some quantifier manipulation.

Proving that Tuchman's *triple encryption is stronger than single encryption* can also be done, but then we need to use a formula characterizing decryption:

$$\begin{array}{l} \forall K^1 \forall K^2 \forall K^3. \\ \quad \forall P \forall C^1 \forall C^2 \forall C. \\ \qquad \left(\mathcal{E}(P, K^1, C^1) \wedge \mathcal{D}(C^1, K^2, C^2) \wedge \mathcal{E}(C^2, K^3, C)\right) \\ \qquad \Longrightarrow \neg\exists K^4. \mathcal{E}(P, K^4, C) \end{array}$$

3 The Encoding of DES

The generation of the formula $\mathcal{E}(P, K, C)$ which describes the logical characteristics of DES has been a substantial operation of reverse "logical" engineering.

The straightforward approach is to describe the VLSI circuit implementing DES and transform the circuit into a logical formula. Unfortunately, the number of resulting formulae is too big to be of any use.

The idea is to walk through the DES algorithm, generating along the way the formulae corresponding to each operation that DES performs, with a clever trick. The

trick to obtain manageable formulae is that not all operations of DES should be explicitly encoded as formulae. Whenever possible, operations should be executed directly on the propositional variables representing the input bits. For instance a permutation is not encoded, rather we execute the permutation of the input bits and provide as output the permuted propositional variables.

Intuitively, the program that generates the encoding works as follows:

- some fixed matrix operations corresponding to the round function f are transformed into boolean formulae and minimized off-line using state-of-the-art CAD tools such as Espresso [Rudell and Sangiovanni-Vincentelli, 1987]);
- each bit of the ciphertext, the plaintext and the key is encoded as a propositional variable;
- then the DES algorithm is simulated, and formulae corresponding to each DES operation are generated on the way by
  - reading the (minimized) complex transformations
  - encoding exclusive or and equivalences;
  - calculating simple operations and permutations;
- if requested, known values of the plaintext and ciphertext are read from file and the corresponding boolean values substituted in the formula;
- finally one simplifies the formula, propagates and eliminates boolean values, equivalences, duplicate variables, tautological or inessential clauses etc.

In summary, the general structure of Feistel-like ciphers is translated on the fly, the permutations and the key scheduling parts are executed, the S-boxes are represented as PLAs, minimized off-line with Espresso, and finally the output of Espresso is used for the encoding of the formulae corresponding to S-Boxes. More details can be found in [Marraro and Massacci, 1999].

If we are interested in a known plaintext attack, it is necessary to introduce the values of particular (plaintext, ciphertext) pairs. This is done by setting each variable in L^0, R^0, $L^{last.round}$, and $R^{last.round}$ to true or false, where *last.round* represents the total number of rounds considered.

The simplification phase is essential to reduce the complexity of the circuits and is repeated until a fixed point is reached. After the simplification, the final outcome of the encoder is a formula which represents the logical relations between the key bits, the (known) plaintext and the (known) ciphertext. Its qualitative structure is shown in Fig. 2.

Since it makes no sense to make this encoding by hand, a program[3] has been designed and implemented to generate formulae encoding DES in an entirely modular and automatic way. We can easily generate the formulae describing DES-like crypto-systems for any number of rounds up to 16, for any key and any pair of plaintext and ciphertext. All permutation and substitution matrices (S-Boxes) are read from external files so that one can change them according one's wishes. In this manner one can evaluate the strength of DES when the number of rounds and the values of the matrices vary.

Definitions

$$\begin{array}{rcll} M_i^r & \Leftrightarrow & \bigwedge_j \pm X_j^r & 2 \le r \le last.round - 1 \\ S_i^r & \Leftrightarrow & \bigvee M^i & 2 \le i \le r \\ X_i^{r+1} & \Leftrightarrow & S_h^r \oplus K_j & 1 \le r \le last.round - 1 \end{array}$$

Constraints

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

Figure 2: The Final Logical Encoding of DES

The use of Espresso [Rudell and Sangiovanni-Vincentelli, 1987] to generate minimal covers for the S-boxes is the only part in our approach where human intervention is necessary: due to interface problems, Espresso must be run on the input files using shell commands. Once Espresso files are generated (and this happens only once for each S-box) our encoder program reads directly from Espresso output files.

As output we can produce a textual representation using AND, OR etc, to be used by BDD-like algorithms, a set of clauses to be used by CNF-based provers, and a non-clausal output in the TPTP non-clausal format.

We have made some tests to get information about execution times and use of memory. These tests were made on a Workstation UltraSparc using Solaris, and on a Pentium II 300Mhz with Windows 98 and Linux.

Execution times were better on the Pentium. For the full 16 rounds version of DES, to generate the generic and the simplified circuits we need approximately 26 seconds and 71 seconds respectively. Proportionately smaller timings and memory requirements were found for reduced variant of DES, e.g. one or two rounds can be done within one second.

Memory is our bottleneck, and this is probably due to a poor (say inexistent) memory management of our program. With better memory management it could also probably take much less time since it would not have to use the swap space. We need from 130MB (SUN) to 175 MB (Windows-98). However note that the final formula is much smaller than that, mostly because of simplification.

Table 1 shows the average number of equivalences, xor, and, or, variables (different from the variables representing the keybits) and clauses generated for each round for

[3] More details on the encoder program can be found in [Marraro and Massacci, 1999] and various encoded formulae are available on the web at `http://www.dis.uniroma1.it/~massacci/cryptoSAT`.

Table 1: Occurrences of Formulae per plaintext/ciphertext block pair

Round	Equiv	Xor	And	Or	Clauses	Vars
1	520	0	504	16	1645	243
2	1042	0	1010	32	3304	489
3	1738	48	1609	80	9064	1430
4	2432	96	2208	128	14811	2368
5	3096	176	2760	160	18738	3032
6	3760	256	3312	192	22665	3696
7	4424	336	3864	224	26592	4360
8	5088	416	4416	256	30519	5024
9	5752	496	4968	288	34446	5688
10	6416	576	5520	320	38373	6352
11	7080	656	6072	352	42300	7016
12	7744	736	6624	384	46227	7680
13	8408	816	7176	416	50154	8344
14	9072	896	7728	448	54081	9008
15	9736	976	8280	480	58008	9672
16	10400	1056	8832	512	61935	10336

one pair of plaintext and ciphertext blocks.

4 Cryptanalysis with Theorem Provers

To test the ability of a *generic* ATP to cope with cryptographic problems, we have chosen three state-of-the-art provers for propositional logic:

- TABLEAU by Crawford and Auton [1996], because it is a reference implementation of the Davis-Putnam procedure and because it has been extensively tested on the random 3-SAT benchmark;
- SATO by Zhang [1997], because it uses a sophisticated data structure based on tries to boost the speed of the unit propagation phase and because it has been successfully used on the encoding of semigroups problems;
- rel-SAT by Bayardo and Schrag [1997], because it combines the Davis-Putnam algorithm with backjumping and learning[4] and because it has been successfully used on large problems in scheduling.

Those three algorithms are all based on variants of Davis-Putnam algorithm. A number of extensive tests with a state-of-the-art BDD package (by G. Janssen) has been also carried [Ascione, 1999], since BDDs are known to perform well on non-clausal problems [Bryant, 1986].

Among the various possibilities described in section 2 we have focussed on key search assuming a known plaintext attack. With the exception of proper QBF problems such as the existence of universal keys, all problems have more or less the same propositional structure and the key search problem has the smallest size. Since we are mainly interested in the verification of the properties of DES (such as closure), then using theorem provers for the smallest problem (key search) should give us an idea on how hard is the full verification task ahead.

[4] If a contradiction is found, then the algorithm backtracks to the literal that has caused the contradiction, and 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.

To generate the propositional formulae we have used for the test, we have worked as follows:

- we have randomly generated few keys (avoiding weak keys [Schneier, 1994]) and some thousands of plaintexts;
- we used the keys to encipher the plaintexts using DES with a variable number of rounds (1,2,3,4,8 and 16) and thus we have generated the corresponding ciphertexts;
- finally we have generated the formulae corresponding to groups of plaintext and ciphertext pairs ranging from one block to 32 blocks.

The final tests have been run on SUN UltraSparcs running Solaris with 64M and 128M RAM and on a Pentium-II running linux with the same memory and we obtained qualitatively the same results. We have obtained the best results with rel-SATand it is indeed the only algorithm which solves 3 rounds of DES within few minutes. The results are reported in Table 2. SATO can also solve 3 round of DES in few minutes of actual CPU time, but requires over 24 hours of "wall-clock" time due to the memory requirements of its data structure: in practice the time is spent simply swapping SATO data back and forth memory and disk.

Given this promising results, the algorithm has been further 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 one day.

Using OBDDs to solve this problem has not led to better results: the extensive tests performed by Ascione [1999] showed that the BDDs representing DES reached quickly a million and over nodes and then crashed the systems for virtual memory failure. Indeed Janssen's OBDD package could not solve key search problems any better than SAT-based approaches such as rel-SAT [Ascione, 1999].

We used SATO to verify how many keys (i.e. models) were consistent with the same pair of plaintext and ciphertext. We have found out that with one round of DES there are many keys that are admissible solutions of the satisfiability problem[5]. With two rounds the number of admissible keys drop sharply and at three rounds the formula has only one model (i.e. only one key is consistent with the given pair).

This result is confirmed by the analysis performed with BDDs [Ascione, 1999]: after three round the final BDD obtained from the processing of $\mathcal{E}(v_C, K, v_P)$ is a simple chain.

We obtained the same result also for two rounds of DES once we conjoined the formulae corresponding to more plaintexts. At three rounds one plaintext/ciphertext pair suffices.

[5] This means that all the different keys that we have found could have been used to generate the ciphertext out of the given plaintext.

Table 2: Performance of CSP with Look-back

		General			No Learning		
R	B	Keybits	Branches	Seconds	Keybits	Branches	Seconds
1	1	31	28	0.02	-	-	-
1	2	48.9	104	0.11	-	-	-
1	4	50.9	104	0.22	53	112	4.44
1	8	52	83	0.45	53	184	6.18
2	1	53.6	20841	32.43	-	-	-
2	2	55.9	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 Conclusions

From a cryptanalyst viewpoint, these results are not very impressive: three rounds of DES were cryptanalyzed long ago [Andleman and Reeds, 1982], and Biham & Shamir [1991] or Matsui [1994a] can do the same and better still on DES with up to 8 rounds (although they need a large number of plaintexts).

What make these results interesting is the fact that the ATP systems do not know at all that the 56 variables of the key are the only "variables that count", the only variables that must be set to find a model, nor they use any heuristics tailored for DES. They do not known which of the thousands of formulae are just definitions in disguise and which are really constraints. With 4 blocks of plaintext and 2 rounds of DES they search in a space of over 2^{1000} solutions, and still they find the only one that exists.

A first explanation is that this happens because there are many defined variables so that few assignments to control variables provoke a cascade of assignments to defined variables and thus inconsistencies are quickly found.

However, this would imply that formulae encoding more rounds, when the defined variables are even more, should be easier and not harder. Indeed, out of the cryptographic literature, we know that the "avalanche" effect increases with the number of rounds, so that a change in a keybit should effect most, if not all, ciphertext bits. Therefore, whenever the search process sets a keybit to the wrong value, this avalanche effect should quickly propagate and generate inconsistencies which allow an early closure of this branch of the search. Since this is not the case, it seems that with more rounds the propagation of assignments is somehow hindered.

We suspect that unit propagation is hindered by the presence of exclusive or at different levels in the encoding of a Feistel-cipher (See Fig. 2 and Fig. 1), since it is well known that exclusive or and equivalences are hard to tame for classical ATPs and BDDs (even if solving them alone can be done in polynomial time). Therefore, it may be that for solving problems in logical cryptanalysis we may need to supply ATP systems with alternative and more effective ways of solving affine subproblems.

The most interesting point regards the faithfulness of DES, i.e. the number of keys which are consistent with the same pair of plaintext and ciphertext. If a key would be consistent with many different plaintext and ciphertext pairs, we would have discovered a trapdoor. Since only one key exists (at least for three rounds) this offers an experimental independent evidence that DES seems not to have hidden trapdoors. Tackling other properties such as proving formally that no such trapdoor exists for the full DES might be a next step.

It is also interesting to notice that, in contrast with standard cryptographic approaches, our solution is exact and not probabilistic (and maybe this is what makes the problem harder for us).

All considered, we think that this is a promising and challenging line of research.

References

[Abadi and Needham, 1996] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[AES, 1998] National Institute of Standards and Technology. Request for comments on candidate algorithms for the Advanced Encryption Standard (AES). *(U.S.) Federal Register*, 63(177), September 1998.

[Anderson and Needham, 1996] R. Anderson and R. Needham. Programming Satan's computer. In *Computer Science Today - Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–440. Springer-Verlag, 1996.

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

[Ascione, 1999] M. Ascione. Validazione e benchmarking dei BDD per la criptanalisi del data encryption stan-

dard. Master's thesis, Facoltà di Ingegneria, Univ. di Roma I "La Sapienza", March 1999. In Italian.

[Bayardo and Schrag, 1997] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 203–208. AAAI Press/The MIT Press, 1997.

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

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

[Büning *et al.*, 1995] H. Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[Cadoli *et al.*, 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 262–267. AAAI Press/The MIT Press, 1998.

[Campbell and Weiner, 1992] K. Campbell and M. Weiner. DES is not a group. In *Proceedings of Advances in Cryptography (CRYPTO-92)*, volume of *Lecture Notes in Computer Science*, pages 512–520. Springer-Verlag, 1992.

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

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

[DES, 1997] National Institute of Standards and Technology. 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.

[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*. Banks for International Settlements, Basle, August 1996.

[Johnson and Trick, 1996] D. Johnson and M. Trick, editors. *Cliques, Coloring, satisfiability: the second DIMACS implementation challenge*, volume 26 of *AMS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[Kaliski *et al.*, 1985] B. Kaliski, R. Rivest, and A. Sherman. Is the Data Encryption Standard a group? (preliminary abstract). In *Advances in Cryptology - Eurocrypt 85*, volume 219 of *Lecture Notes in Computer Science*, pages 81–95. Springer-Verlag, 1985.

[Lowe, 1996] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and Steffen B., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[Marraro and Massacci, 1999] L. Marraro and F. Massacci. A new challenge for automated reasoning: Verification and cryptanalysis of cryptographic algorithms. Technical Report 5, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999.

[Matsui, 1994a] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Proceedings of Advances in Cryptography (CRYPTO-94)*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1994.

[Matsui, 1994b] M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - Eurocrypt 93*, volume 765 of *Lecture Notes in Computer Science*, pages 368–397. Springer-Verlag, 1994.

[Mitchell *et al.*, 1997] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *Proceedings of the 16th IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, 1997.

[Paulson, 1998] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 1998.

[Rudell and Sangiovanni-Vincentelli, 1987] R. Rudell and A. Sangiovanni-Vincentelli. Multiple valued minimization for PLA optimization. *IEEE Transactions on Computer Aided Design*, 6(5):727–750, 1987.

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

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

[Zhang, 1997] H. Zhang. SATO: an efficient propositional prover. In *Proceedings of the 14th International Conference on Automated Deduction (CADE-97)*, volume of *Lecture Notes in Computer Science*, 1997.