
Journal of Automated Reasoning **24:** 165–203, 2000.

Logical Cryptanalysis as a SAT Problem ⋆

Encoding and Analysis of the U.S. Data Encryption Standard

FABIO MASSACCI
Dipart. di Informatica e Sistemistica, Università di Roma I "La Sapienza", Italy and Dipart. di Ingegneria dell'Informazione, Università di Siena, Italy. e-mail: massacci@dii.unisi.it

LAURA MARRARO
Dipart. di Informatica e Sistemistica, Università di Roma I "La Sapienza", Italy

Abstract. Cryptographic algorithms play a key role in computer security and the formal analysis of their robustness is of utmost importance. Yet, logic and automated reasoning tools are seldom used in the analysis of a cipher, and thus one cannot often get the desired formal assurance that the cipher is free from unwanted properties that may weaken its strength.

In this paper, we claim that one can feasibly encode the low-level properties of state-of-the-art cryptographic algorithms as SAT problems and then use efficient automated theorem-proving systems and SAT-solvers for reasoning about them. We call this approach *logical cryptanalysis*.

In this framework, for instance, finding a model for a formula encoding an algorithm is equivalent to finding a key with a cryptanalytic attack. Other important properties, such as cipher integrity or algebraic closure, can also be captured as SAT problems or as quantified boolean formulae. SAT benchmarks based on the encoding of cryptographic algorithms can be used to effectively combine features of "real-world" problems and randomly generated problems.

Here we present a case study on the U.S. Data Encryption Standard (DES) and show how to obtain a manageable encoding of its properties.

We have also tested three SAT provers, TABLEAU by Crawford and Auton, SATO by Zhang, and rel-SAT by Bayardo and Schrag, on the encoding of DES, and we discuss the reasons behind their different performance.

A discussion of open problems and future research concludes the paper.

Key words: cipher verification, Data Encryption Standard, logical cryptanalysis, propositional satisfiability, quantified boolean formulae, SAT benchmarks.

1. Introduction

Providing computer security in large open networks such as the Internet is one of the frontiers of computer science today [2, 38, 35]. Yet, providing security is not so simple, and many technical challenges need to be solved to provide the high assurance that such an enterprise requests. These challenges are sketched in the excellent introduction by Anderson and Needham [2] and require the use of partic-

⋆ This is a longer and revised version of [30]. It contains also some of the experimental data reported in [31] for rel-SAT. Further details, problem instances, and the encoder program can be found at the URL: `http://www.dis.uniromal.it/~massacci/cryptoSAT`.

ular security protocols for communication [1, 19, 45] and advanced cryptographic techniques [45]. Notwithstanding the amount of research and development work, the literature is full of "how to break a secure..." examples [10, 28, 43]. A protocol can be proven formally secure [39] and still be broken because the cipher used for its implementation has unwanted algebraic properties [43].

So, the use of logical encodings and automated reasoning tools for the analysis of cryptographic algorithms seems to be the next step toward greater assurance of security. Yet, in contrast with the large literature on formal verification of cryptographic protocols (see, e.g., [10, 39, 28, 34]), we find little or no use of logics and automated reasoning tools in the proper cryptographic literature. Sophisticated and successful techniques such as linear cryptanalysis [33, 32] or differential cryptanalysis [7] use only statistical tools for solving cryptanalysis problems. The presence or absence of algebraic properties that may substantially weaken a cipher, such as algebraic closure, is often determined by ad hoc methods or huge experiments [12, 26, 45].

In the absence of formal verification, the properties of many cryptographic algorithms are subject to intense debates and speculations.* Cryptographic key search has become THE search problem for many governments and large corporations, and the (lack of) resistance to key search is often the main concern behind the licensing of a cipher [23].

Thus, a new field of potential applications comes to mind:

- Can we encode low-level properties of ciphers into logic?
- Can we do it in such a way that finding a model of the encoded formula is equivalent to finding a key so that AI search techniques can be used to validate the strength of an algorithm?
- Can we do it in such a way that other problems, such as cipher integrity or existence of trapdoors, can also be formally verified by complete automatic reasoning tools?
- Last, but not least, can we do it by using a *feasible encoding*, which might give hard-to-analyze formulae but not overwhelming?

In this paper we claim that propositional logic and automated reasoning tools can be efficiently used to model and verify state-of-the-art cryptographic algorithms such as the (U.S.) Data Encryption Standard.

Among the various possibilities, we have chosen to encode cryptographic properties as SAT problems for a number of factors that are well summarized by Selman, Kautz, and McAllester [47]:

> First new algorithms were discovered, including ones based on stochastic local search as well as systematic search, that have better scaling properties than

* See, e.g., [45, Chapter 12] for a survey of the long-lived debate on whether the Data Encryption Standard has hidden trapdoors that would allow the U.S. National Security Agency to decrypt all traffic.

> the basic Davis–Putnam algorithm. Second, improvement in machine speed, memory size and implementations extended the range of the algorithms. Third, researchers began to develop and solve prepositional encodings of interesting, real-world problems such as planning and diagnoses, with others on the horizon [. . .].
> 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.

As we shall see, a number of interesting and challenging cryptographic problems are now at the border of tractability for SAT-based approach.

1.1. CONTRIBUTIONS OF THIS PAPER

In a nutshell, we show how to encode the abstract, functional properties of a cryptographic algorithm* in a suitable logic so that, for instance, finding a model of the corresponding formulae is equivalent to recovering a key in a cryptanalytic attack. Once the properties of the algorithm are represented as (propositional) formulae, we can use efficient and effective automatic (SAT) reasoning tools for the analysis and the verification of the algorithm. We call this approach *logical cryptanalysis*.

To make our claims concrete, we 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, [36, 45].

The Data Encryption Standard, designed by IBM and NSA in the 1970s, 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 [37], its widespread use and the fact that its security has been the subject of an intense scrutiny since its inception [7, 12, 33, 32] make the case study significant. For many years DES has been (and still is) the algorithm on which cryptanalysts tested the final success of their techniques (see [45, pages 285–294] or Section 2 for a discussion and further references). Even partial successes with AI techniques might be of substantial industrial relevance.

The encoding of the properties of DES that we have been able to generate is at the border of tractability for current search techniques: it is hard but not impossible. For example, the encoding of a cryptographic attack with a known plaintext (where finding a model is equivalent to finding a key) for the commercial version of DES requires slightly more than 60,000 clauses and 10,000 variables (out of this only 56 are independent control variables, the key bits).

* We refer only to the analysis of the algorithm itself and not to the verification of the program, software code, or hardware circuit that implements it. Thus, we are not interested in software verification and in showing that an implementation matches a specification. We are interested in showing that the specification itself does not imply hidden, unwanted properties.

To check the effectiveness of AI techniques on this problem, we have used state-of-the-art SAT provers for cryptographic key search with our encoding. Here we focus on complete algorithms* based on the Davis–Putnam (DPLL) procedure [18, 17] and in particular on

- TABLEAU, by Crawford and Auton [15], a reference implementation of the DPLL procedure,
- SATO, by Zhang [51], which uses a sophisticated trie data structure for speeding up unit propagation,
- rel-SAT, by Bayardo and Schrag [5], a combination of the traditional DPLL procedure with back-jumping and learning.

In the experiments on the Data Encryption Standard, we didn'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 12 workstations for 50 days, which have been used to break DES in the last few years [16, 32]. Still, we were pleasantly surprised by the result: a general-purpose search algorithm such as rel-SAT 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. Yet, there is a lot of research work that needs to be done, since the commercial version is still out of reach for SAT-based systems, and on intermediate versions we cannot fully compete** with the performance of advanced cryptographic techniques based on statistical analysis [3, 7, 32].

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

For the *automated reasoning community*, it provides a set of challenging problems of industrial relevance ranging from satisfiability and validity in propositional logic to validity in quantified boolean logic. Thus we claim that this problem could be one of the reference benchmarks for propositional reasoning and search. We should not see such applications as tools for "electronic criminals". Verification and cryptanalysis of ciphers are the common practice of institutions responsible for international standards and of cryptographic research.‡ After all, before recommending an algorithm for encoding bank transactions, one may want to thoroughly test and verify it.

Moreover, the encoding of Feistel-type ciphers like DES has the same characteristics of challenging hard problems such as the hidden parity bit problem mentioned in [47]: defined and independent variables, a hierarchical and regular

* Local search algorithms such as GSAT and Walk-SAT do not seem to perform well on this problem. See Massacci [31] for further details.

** Note, however, that these cryptographic methods require large amount of data to be effective. See further in Section 2.

‡ Most security firms offer cash prizes for those who are able to break their ciphers.

structure, and large affine subproblems (i.e., formulae with exclusive or). As we shall see, such encodings give 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.*

Thus, we believe that it can be a first step toward an answer to the last SAT challenge proposed by Selman, Kautz, and McAllester [47]. It is also an answer to the problem of generating hard solved instances discussed by Cook and Mitchell in their DIMACS survey [14]. As our experiments shows, our encoding can be used to generate solved instances which are hard to solve, in contrast with the standard generation methods for solved instances. We exploit the fact that cryptographic algorithms are *designed* to be hard to crack. Our proposal follows the spirit of Cook and Mitchell [14], where it was proposed to use RSA for generating hard solved instances.

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 trapdoors or key escrow schemes. 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. Also, to verify the strength of a cipher, one may test it for resistance against SAT solvers and thus possibly avoid the risk of becoming the subject of a "how to break a secure. . ." paper at the next crypto conference.

1.2. PLAN OF THE PAPER

In the rest of the paper we recall at first some basic things about cryptography, how the Data Encryption Standard works, and some of the main approaches used for the cryptanalysis of DES (Section 2).

Then we present the general idea behind logical cryptanalysis, and how different properties can be captured by this approach once we have encoded the functional description of a cipher into propositional logic (Section 3). The details of the encoding of the Data Encryption Standard are presented in Section 4, where we also presents some experimental data on the size and structure of the corresponding SAT-problem.

Third, we present some preliminary experiments on the performance of the automated reasoning tools TABLEAU, SATO, and rel-SAT for the simplest cryptographic problem that we can encode as a SAT problem (cryptographic key search

* This contrast, with the most common benchmarks for propositional reasoning and search, which either are totally random such as the Random 3-SAT CNF [48] or are fixed but have industrial relevance such as the IFIP benchmark for hardware verification [13]. Our proposal follows the line of the work by Gomes et al. on generating structured CSP problems using quasi-group problems [21, 22].

on a limited version of DES) and discuss the reason behind their different performances (Section 5).

We conclude the paper with some topics for future research (Section 6).

2. Basics of Cryptography

At high level (for a comprehensive introduction to the subject, see [45]), a cryptographic algorithm transforms a sequence of bits into another sequence of bits with certain (desirable) properties. So, if we denote three vectors of bits by **P** (the plaintext), **C** (the ciphertext), and **K** (the secret key) we have

$$\mathbf{C} = \mathrm{E}_{\mathbf{K}}(\mathbf{P}).$$

The important property of this transformation, called *encryption*, is that it must be difficult to recover **P** from **C** if one does not know **K**.

In most cases we have another transformation (*decryption*) that maps back **C** into **P** using **K** or another sequence of bits $\mathbf{K}^{-1}$, which is related to **K**.

$$\mathbf{C} = \mathrm{D}_{\mathbf{K}^{-1}}(\mathbf{P}).$$

If **K** can also be used for decryption, we have a *symmetric cipher*.

The first thing a cryptanalyst may wish to do is to *search for a key* which produced a given ciphertext. This can be done by using only the ciphertext, or a number of known pairs of plaintext and ciphertext (known plaintext attack). This latter attack is not so impossible as one might expect: there are lot of standard parts in encrypted messages and files. In some cases it makes sense to assume that the cryptanalyst could choose such pairs (chosen plaintext attack). The cryptographic techniques we sketch in Section 2.2 can be classified along these lines.

There are other, more sophisticated properties that are more interesting though harder to analyze. The basic one regards the *existence of trapdoors* in a cipher. For instance, we may wonder if there is an *universal key* to decrypt any message encrypted with DES. As we already noted, this has been the subject of an intense debate till the recent past [45, Section 12.3].

Almost all algebraic properties of a cipher that distinguish it from a pseudo-random permutation affect (weaken) its strength. A fairly simple property is the existence of *weak keys*, i.e., of keys **K** such that $\mathbf{P} = \mathrm{E}_{\mathbf{K}}(\mathrm{E}_{\mathbf{K}}(\mathbf{P}))$. There is little sense in encrypting things twice with a weak key. For instance, DES has an handful of weak keys [45, p. 280].

Even if no weak key exists, we may wish to know whether *a cipher is closed* [26, 45, 12]; that is, for any plaintext **P** and for any two key $\mathbf{K}^1$ and $\mathbf{K}^2$ we can always find a third key $\mathbf{K}^3$ such that encrypting a plaintext with the first key and then with the second key is perfectly equivalent to encrypting it is just once with the third key. In other words we want to know whether $\mathrm{E}_{\mathbf{K}^2}(\mathrm{E}_{\mathbf{K}^1}(\mathbf{P})) = \mathrm{E}_{\mathbf{K}}(\mathbf{P})$ for a suitable key **K**. In general we may wish to find a key **K** that is independent from the plaintext and just dependent on the other two keys.

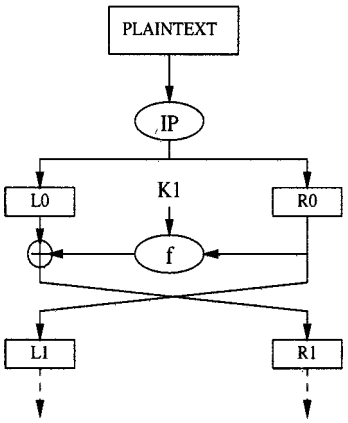

Figure 1. A round of DES.

If a cipher is closed, this means that we cannot improve its security by encrypting things twice with different keys. Even worse, the cipher is vulnerable to particular attacks that make it possible to reduce the search space of brute force attacks to half of the key bits [26]. Proving that a cipher is closed is equivalent to proving that the encryption transformations form a group. For instance, it has been proved with a huge number of cyclic experiments that DES is not a group [12].

Another interesting combination of encryption with different keys is Tuchman's *triple encryption scheme*, which is an encryption of the plaintext with one key, its decryption with a different key, and its final re-encryption with a third key. Again, we wish to known whether we are better off than by using a simple encryption, i.e., whether

$$E_{\mathbf{K}^3}(D_{\mathbf{K}^2}(E_{\mathbf{K}^1}(\mathbf{P}))) \neq E_{\mathbf{K}}(\mathbf{P})$$

for all plaintexts $\mathbf{P}$ and all keys $\mathbf{K}$.

As a final example, we may wish to know whether a cipher is *faithful* [26], i.e., if different keys can be used to generate the same plaintext-ciphertext pair. In symbols we may ask whether $E_{\mathbf{K}}(\mathbf{P}) = E_{\mathbf{K}'}(\mathbf{P})$ implies that $\mathbf{K} = \mathbf{K}'$. This is important if we want to use the data as court evidence.

2.1. THE DATA ENCRYPTION STANDARD

As the reader might now want to know how DES works, we start by saying that DES is a *block cipher*, which enciphers blocks of 64 bits (the plaintext) into blocks of 64 bits (the ciphertext) using a key of 56 bits.* We give here only a simple presentation to make the paper self-contained; the interested reader is referred to [36] or [45, Chap. 12] for more details.

* The key is usually expressed as a 64-bit number, in which every eighth bit is used for parity checks and is ignored by the algorithm.

DES and many other popular symmetric ciphers such as RC5 are built following a common architecture which is due to Feistel and his group [20]. 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 f,
3. XOR the combination with the other half, and
4. swap the two parts.

The combination of these operations is called a *round*. Figure 1 exemplifies the construction.

In the case of DES there are 16 rounds (i.e., the above operations are "repeated" 16 times), which are almost identical except for the keys: for each round a different subset of the 56 key bits is selected and combined with the input of the previous round. The strength of the cipher depends on the way this combination function is designed and on the number of rounds. This design is, to quote Ron Rivest, "part art, part science".

Following the notation introduced at the beginning of the section, we use superscripts to distinguish two different vectors of bits and subscripts to represents the single element within a vector. So, in $\mathbf{L}^1$, $\mathbf{L}^2$, and as or, $\mathbf{L}^i$ is the ith vector and not the ith component. The jth component of the ith vector $\mathbf{L}^i$ is represented as $\mathbf{L}^i_j$.

After an initial permutation, whose purpose is to mix the input bits, the plaintext is divided in two halves, a left one and a right one. The 16 rounds in which the right part of the data is combined with the key can be described as follows:

$$\mathbf{L}^i = \mathbf{R}^{i-1}, \tag{1}$$

$$\mathbf{R}^i = \mathbf{L}^{i-1} \oplus f(\mathbf{R}^{i-1}, \mathbf{K}^i), \tag{2}$$

where we indicate by $\mathbf{A} \oplus \mathbf{B}$ the xor of the jth component of the vector $\mathbf{A}$ with the jth component of the vector $\mathbf{B}$. After the 16th round, the right and left halves are joined, and a final permutation (the inverse of the initial permutation) finishes the algorithm.

For each of the 16 rounds of DES a 48 bits subkey is generated. These subkeys, $\mathbf{K}^i$, are generated according a particular scheduling algorithm. Initially we divide the 56 bits key in two 28 bits blocks, $\mathbf{C}^0$ and $\mathbf{D}^0$, and, during the generic iteration, $\mathbf{C}^{i-1}$ and $\mathbf{D}^{i-1}$ are circularly shifted left of one or two bits based on the round i. This produces two blocks, $\mathbf{C}^i$ and $\mathbf{D}^i$, that are joined and permuted to produce the 48 bits subkey $\mathbf{K}^i$.

The round function f, which DES uses to mix up the key bits with the output of the previous round, takes as arguments the right half $\mathbf{R}^i$ from the previous round and the subkey from the actual round. Then the 32 bits that compose $\mathbf{R}^i$ are permuted and expanded to 48 bits and then xored together with the key bits:

$$\mathbf{X}^i = \mathrm{E}(\mathbf{R}^{i-1}) \oplus \mathbf{K}^i.$$

The 48-bit block $\mathbf{X}^i$ is then divided in eight subblocks of 6 bits each. Each subblock specifies an entry in a *substitution matrix* called *S-box*. Every matrix has six input

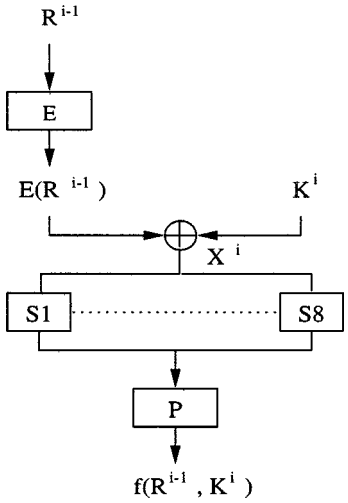

Figure 2. The round function f of DES.

bits and four output bits; therefore it has four rows and sixteen columns. The six input bits specify in which row and column to see to find the four output bits. From the S-boxes we obtain eight 4-bit numbers that form a 32-bit block.

The pictorial representation in Figure 2 should make it clearer.

Substitution and permutation operations realize the properties that Shannon [49] named confusion* and diffusion** and that give a cipher its strength. So the S-boxes represent the most critical step that gives DES its strength [45, pp. 284 and 294].

2.2. EXISTING CRYPTANALYTIC APPROACHES

The simplest approach is *exhaustive search*: try different keys until the right one is found. Testing 2^{55} keys on average is not easy unless one has specialized hardware or a large network of machines. This has proved to be finally successful in 1998: a specialized machine costing 250.000 US$ broke the DES challenge posted by RSA Security in 56 hours [16]. Although the search strategy exploited some of the algebraic properties of DES to cut the size of the search space, we can substantially classify it as exhaustive search.

Differential cryptanalysis was introduced in the early 1990s by Biham and Shamir [7]. It considers ciphertext and plaintext pairs presenting particular fixed differences. Then, it analyzes the evolution of these differences as the plaintexts pass through the rounds of DES. Using the differences appearing in the actual ciphertexts, different probabilities can be assigned to different keys. Considering

* Confusion is created using operations that make an output sequence non-linearly dependent on an input bits sequence.

** With diffusion an output bit is made dependent on a great number of input bits.

an increasing number of ciphertext and plaintext pairs, a key will emerge as the most probable.

Using this approach Biham and Shamir found a chosen plaintext attack that is more efficient than exhaustive search. For a few rounds this approach is still feasible, although it requires larger and larger amounts of data as the number of rounds increase. For instance, DES up to 8 rounds can be broken within few minutes on a PC if a sufficiently large pool of chosen plaintexts (50,000) is available [7, p. 6]. However, for the full DES with 16 rounds, this attack is only theoretical since it requires too many resources. To get the key, it uses 2^{47} chosen plaintext. The time complexity of the data analysis is about 2^{37} DES equivalent operations. If this approach is converted in a known plaintext attack, its efficiency is worse than exhaustive search.

Matsui's *linear cryptanalysis* [32, 33] works better. This method uses linear approximations (xor) to describe the behavior of a block cipher. 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 correct with a certain probability. Using an increasing number of plaintext and related ciphertexts, it is possible to guess the value of some key bits. If more data is analyzed, the guess is more reliable.

To identify a good linear approximation for DES, it is necessary to find good linear approximations for each round, and then join them together. The base attack uses the best linear approximations for 16-rounds DES, and 2^{27} known plaintexts, returning two key bits. A refinement of this method uses the linear approximations of 14 rounds and then guesses the input and output values of the first and last rounds of DES. It can find 26 key bits and for the remaining bits uses exhaustive search. An experimental analysis by Matsui [32] showed that is possible to get the key of the full version of DES in 50 days using a network of workstations.

Davie (unpublished) proposed a potential attack on DES based on the non-uniformity of the distribution of the output of pairs of adjacent S-boxes. This attack is theoretical and requires $2^{56.6}$ known plaintexts to discover two key bits. The improved Davie's attack [6] finds 24 key bits by applying the analysis twice, one considering the odd rounds and the other considering the even rounds. The remaining 32 bits can be found by exhaustive search. The improved version is able to break DES faster than exhaustive search.

3. 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, in which every variable is true when the corresponding bit is 1 and false when it 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 other logics (such as temporal logic) might give a more compact encoding.

The intuition behind logical cryptanalysis is as simple as that. Once we have the formula describing the cipher, the cryptanalysis problems we have described can be easily formalized.

However, before going forward to the examples, we need to notice that if $\mathcal{E}(P, K, C)$ holds there is no guarantee that $\mathcal{E}(C, K, P)$ holds too. $\mathcal{E}(C, K, P)$ does not model the fact that we can obtain **P** by decrypting **C** using **K**. Even for symmetric ciphers, where the same key is used for encryption and decryption, the algorithm used for decryption is slightly different from the algorithm used for encryption (e.g., in DES we need to change the key scheduling). This slight difference is sufficient to make the formula $\mathcal{D}(C, K, P)$ slightly different from $\mathcal{E}(C, K, P)$.

To model *key search*, let v_C be the truth values (true/false) of the available 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 v_P of the propositional variables P. So, we only need to search for a model of $\mathcal{E}(v_P, K, v_C)$.

For reason of space 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. Indeed, this is what we have done in the encoding of DES in Section 4. Still, if the encoding is well made, then K, C, and P should be the only *control 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.

If we have n plaintext and ciphertext pairs, we can constrain the search further by conjoining the corresponding formulae

$$\bigwedge_{i=1}^{n} \mathcal{E}(v_P^i, K, v_C^i).$$

We can thus encode cryptographic key search as a satisfiability problem.

In this way we can generate easily *solved instances* that might be very hard to solve: we generate randomly a key **K** and a plaintext **P**. Then we use the cryptographic algorithm itself to generate $\mathbf{C} = \mathrm{E}_{\mathbf{K}}(\mathbf{P})$. Finally we substitute in $\mathcal{E}(P, K, C)$ the corresponding Boolean values v_P and v_C that we have so far generated. Then the pair $\langle v_K, \mathcal{E}(v_P, K, v_C)\rangle$ gives a solved instance of the satisfiability problem. If the encoding is well designed, one could also use $\mathcal{E}(v_P, v_K, C)$ and unit propagation to obtain v_C. Our encoding satisfies this property, but using the original encryption algorithm is much faster.

Finding a model (v_K or another assignment) is then equivalent to break the cipher with a known plaintext attack that uses only one or few plaintext/ciphertext pair. Since the ciphers are designed to be hard to break, this will provide us with the hard solved instances asked for by Cook and Mitchell [14].

For the analysis of the strength of a cipher, we are often interested in other important properties that can be encoded as SAT problems or at worst as quantified

Boolean formulae (QBF) [9, 11]. For instance, if we use QBF, finding a key would correspond to a constructive proof of

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

The simplest property (but for importance) that can be encoded as a SAT problem is the property that a cipher is *faithful* [26]: for any pair of plaintext and ciphertext there is only one key that could have produced them. This property is captured by the following formula:

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

In the case of DES, by analyzing the algorithm it is possible to detect some (six) key pairs K and K' that do not satisfy this property. They are called *semi-weak keys*.* It is not known whether they are the only ones (see, e.g., [45, p. 281]).

In some cases (e.g., when discussing the value of encrypted evidence in a court case), we may be interested in knowing whether a semi-weak key exists for given plaintexts and ciphertexts. This problem is just a slightly harder variant of the key search problem: we want to know whether $\mathcal{E}(v_P, K, v_C)$ has a unique solution.

Another problem that can be expressed as a SAT problem is the (*non*) *existence of weak keys*: we want to prove that $\mathrm{E}_K(\mathrm{E}_K(\mathbf{P})) \neq \mathbf{P}$ for all values of $\mathbf{P}$. If we denote the result of the first encryption by C, this property can be captured by the propositional formula below:

$$\mathcal{E}(P, K, C) \Rightarrow \neg\mathcal{E}(C, K, P).$$

If the formula above is propositionally valid, then no weak key exists for all values of the plaintext. We can restrict it further by looking at particular instances of v_P or v_K, if we suspect v_K to be a weak key. Since DES has few weak keys, this problem has a known answer. It might be interesting to see whether SAT-based approaches can find other weak keys automatically.

If the strong property above is not valid, we can weaken it by using a 2-QBF formula. Now we just require that for every key there is at least a plaintext that is not mapped into itself.

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

Notice that we do not need to quantify over C: it should be uniquely determined by the values of K and P if the encoding is well designed.

If we want to prove that *a cipher is not closed*, the formula we have to prove is slightly more complicated. Recall that we have to prove that for any pair of keys

* For instance, including the parity bits (every eighth bit), the keys `01FE 01FE 01FE 01FE` and `FE01 FE01 FE01 FE01` are a pair of semi-weak keys.

$\mathbf{K}^1$ and $\mathbf{K}^2$ we can never find a third key $\mathbf{K}$ such that $\mathrm{E}_{\mathbf{K}^2}(\mathrm{E}_{\mathbf{K}^1}(\mathbf{P})) = \mathrm{E}_{\mathbf{K}}(\mathbf{P})$. If we denote by C^1 the result of the first encryption and by C the result of the final encryption, we get the following formula:

$$\big(\mathcal{E}(P, K^1, C^1) \wedge \mathcal{E}(C^1, K^2, C)\big) \Rightarrow \neg\mathcal{E}(P, K, C).$$

If this formula is valid, then the cipher is not closed: encrypting twice (with different keys) is better than encrypting once.

Notice that this property has never been formally proven for DES. There is only an indirect proof based on cyclic experiments [12] of the weaker property that the encryption with DES does not form a group, which is equivalent to the following one:

$$\exists K^1 \exists K^2 . \forall K \exists P . \big(\mathcal{E}(P, K^1, C^1) \wedge \mathcal{E}(C^1, K^2, C)\big) \wedge \neg\mathcal{E}(P, K, C).$$

Proving that Tuchman's *triple encryption is stronger than single encryption* can also be done by proving the following formula:

$$\big(\mathcal{E}(P, K^1, C^1) \wedge \mathcal{D}(C^1, K^2, C^2) \wedge \mathcal{E}(C^2, K^3, C)\big) \rightarrow \neg\mathcal{E}(P, K, C).$$

This property can also be weakened using QBF.

The *existence of universal keys* which can decrypt all traffic can be characterized by a QBF formula. If we denote by C the result of the encryption, we have

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

Of course, we might be more interested in the negation of this formula, i.e., in proving that no universal key exists. It is easy to check that if a cipher is faithful, then no universal key exists.

Among these problems, we believe that the encoding of key search as a SAT problem deserves particular attention as a SAT benchmark. The main advantage behind our proposal is the combination of the seemingly contrasting needs of using "real-world" problems (possibly with a lot of structures) and of generating a huge number of instances which can only be (pseudo) randomly generated.

Real problems are important because solving them is what one expects from SAT solvers. Yet, dealing with them is not simple [5]:

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

Crawford and Auton [15] noted that working with random instances might be preferable because

> [...] 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.

Our main point is that by changing the plaintext and the key we can generate as many *solved instances* as we want. If we encode the full-fledged version of the cryptographic algorithm (such as the Data Encryption Standard), each instance would be identical to an actual plaintext and ciphertext used by a bank, financial institution, or government department. In the case of the Data Encryption Standard we can generate $2^{56} \times 2^{64}$ instances that are fairly enough. Indeed, if we consider the encryption of binary or compressed data, we can substantially span the whole message space. If we restrict ourselves to ASCII plaintexts, the number of different plaintexts shifts from 2^{64} to 2^{56}, since every 8th bit of the 64 message bits will be fixed.

To generate *unsatisfiable instances*, it is sufficient to generate a plaintext and ciphertext pair and then give it to the encoder together with a wrong ciphertext (by randomly flipping few bits). Our experimental analysis (Section 5) on the Data Encryption Standard shows that few rounds or few plaintext and ciphertext pairs are sufficient to constraint the search to only one model. Changing one bit of the ciphertext would make the problem unsatisfiable.

Of course, the hard part is getting a manageable translation. We have done this for the Data Encryption Standard.

4. Encoding the Data Encryption Standard

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

The straightforward approach would be describing the VLSI circuit implementing DES and transforming the circuit into a logical formula. Unfortunately, the resulting formula is too big to be of any use.

Our basic 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 stages of DES should be explicitly represented by a circuit and then 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, to generate the encoding our programs must

- transform into Boolean formulae the fixed matrix operations corresponding to the round function f (Figure 2), and minimize them off-line using state-of-the-art CAD tools;
- encode each bit of the ciphertext, the plaintext and the key as a propositional variable;
- simulate the DES algorithm, and generate formulae corresponding to each DES operation on the way by

  - encoding complex transformations (previously minimized off-line);
  - encoding exclusive or and equivalences;
  - calculating simple operations and permutations;
- if requested, read from files the known values of the plaintext and ciphertext and substitute the corresponding Boolean values in the formula;
- finally, simplify the formula, propagate and eliminate Boolean values, equivalences, duplicate variables, tautological or inessential clauses, and so furth.

This process can be logically divided in three parts, one dealing with the general structure of Feistel-like ciphers, one dealing with the permutations and the key scheduling, one treating the S-boxes.

4.1. ENCODING THE GENERAL FEISTEL ARCHITECTURE

The encoding of the general Feistel-like structure of DES is straightforward. For every round i of the algorithm we generate the formulae

$$\begin{aligned} L_j^i &\Leftrightarrow R_j^{i-1}, && j = 1, \ldots, 32, \\ R_j^i &\Leftrightarrow L_j^{i-1} \oplus F_j^i, && j = 1, \ldots, 32, \\ X_j^i &\Leftrightarrow E(R^{i-1})_j \oplus K_j^i, && j - 1, \ldots, 48, \\ F_j^i &\Leftrightarrow P(S^i)_j, && j = 1, \ldots, 32. \end{aligned}$$

In the previous formulae, the expression $E(R^{i-1})$ represents a permutation and an extension from 32 to 48 bits of the initial vector of 32 propositional variables represented by R^{i-1}. In practice this means that some variables in R^i are duplicated. The expression $P(S^i)$ represents the permutation of the S-boxes outputs. The vector of variables F^i is the output of the f function. The vector of propositional variables K^i represents the ith subset of the initial key that is chosen by the algorithm for each round.

In the future we plan to study some possibility of off-line minimization for these formulae, using the CAD minimization tool we have used for the generation of the S-boxes in Section 4.3, since they are common to many other ciphers and are repeated for many rounds.

4.2. ENCODING PERMUTATIONS AND THE KEY SCHEDULING

The "encoding" of the permutations and the generation of the subkeys K^i have been the subject of the trick we mentioned at the beginning of the section.

So, let M be a permutation matrix and A the vector of propositional variables to be permuted. If m_j is the element j in the matrix M, then the variable M_j representing the element j of the matrix M will be equal to the variable A_{m_j} representing the element m_j of the vector A. Then we can simply consider the permuted variables as the input of the next stage of the algorithm.

To generate the subkey K^i we consider a 64-element vector in which every element is an integer representing the index of the variable that locates a particular key bit. The DES algorithm applies a permutation (the one that eliminates eight parity bits) and reduces these elements to 56. These 56 elements are divided in two halves, and the shift operation is performed for each round. The two shifted parts are then joined and permuted again to give the subkey. To encode the shift operation, suppose that the number of shifts that have to take place at round i is denoted by s_i. Then, for each element in the vector to be shifted we calculate the position p_j with $(j = 1, \ldots, 28)$ that each element occupies in the shifted vector.

The new positions can be obtained using the following algorithm:

1. $p'_j = (j - s_i)$.
2. If $p'_j \leqslant 0$
   then $p_j = 28 - |p'_j|$
   else $p_j = p'_j$.

With the new positions, the shift can be encoded as a permutation.

For instance, suppose that the processed round is the first round. Then s_i is equal to 1, and one of the two vectors containing half of the key bits to be shifted is C_0: [57 49 41 33 ... 52 44 36].

Using the previous criteria it is possible to see that

$$j = 1 \Rightarrow p'_1 = 1 - 1 = 0 \Rightarrow p_1 = 28$$
$$j = 2 \Rightarrow p'_2 = 2 - 1 = 1 \Rightarrow p_2 = 1$$
$$\vdots$$
$$j = 28 \Rightarrow p'_{28} = 28 - 1 = 27 \rightarrow p_{28} = 27$$

In this way we have obtained a permutation vector that has to be applied to C_0. Performing this permutation we obtain the shifted vector C_1: [49 41 33 ... 52 44 36 57]. In the final formula, rather than adding a formula corresponding to the permutation relating the input variables $K_1, K_2, \ldots, K_{64}$ and the output variables of the permutation circuit $K^2_1, K^2_2, \ldots, K^2_{64}$ and then using the output variables $K^2_1, K^i_2, \ldots$ in the rest of the circuit, we use the corresponding permuted variables K_{49}, K_{41}, etc. This is the twist that makes the encoding feasible.

4.3. ENCODING THE S-BOXES

The fundamental problem in processing the S-boxes is to determine a representation that is adequate to translate these matrices into formulae. The simplest approach is to use programmable logic arrays (PLAs) since these matrices are addressed by bits and their entries are bits too. Once every S-box is represented as a PLA, we get formulae of this kind:

$$M^i_{kh} \Leftrightarrow \bigwedge_{j=1}^{48} \pm X^i_{jkh}, \quad k = 1, \ldots, \max\ M_{ih}$$

$$S_h^i \Leftrightarrow \bigvee_{j=k_1}^{\max M_{ih}} M_{k_j h}^i, \quad h = 1, \ldots, 32,$$

where M^i are the minterms generated in the round i, the vector of propositional variables S^i represents the 32 output bits of the S-boxes for the round i, and max M_{ih} is the number of minterms generated in the round i for the hth output bit of the S-boxes.

If we do this translation without using minimization techniques, we introduce a lot of redundant minterms, which make the search harder.

To minimize the PLAs representing the S-boxes, we used a CAD program for PLA minimization [42, 41] named Espresso, a minimization tool for two-level binary functions and multiple-valued logic functions. Espresso accepts as input a two-level description of a Boolean function. This function is represented as a character matrix with keywords embedded in the input to specify the size of the matrix and the logical format of the input function. In our case we automatically generate the inputs for Espresso from the standard S-box descriptions.

We tried two different approaches to minimization. In the first one every S-box was considered as a single PLA with multiple valued outputs. In the second one we modeled each output bit of each S-box as a separate PLA and minimized each PLA independently. The rationale behind the first choice is that we expected to have minterms common to different outputs.

In the first case we have 8 PLAs with six input variables and four output variables. After the minimization, the entries of the matrixes are reduced from 64 to 52 (51 for the eighth S-box). The distribution is presented in the first four columns of Table I. The column represents each S-box outputs.

Considering each output bit of an S-box as a single output we got 32 PLAs with six input variables and one output variable. With this kind of minimization we obtain a further improvement in terms of the number of minterms associated with each output and the average number of variables associated with each minterm. The distribution of the number of minterms is presented in the last four columns of Table I.

As for the number of variables, we note that in the first case we have on average 5.4–5.6 variables per minterm and in the second case 4.8–5.0 variables per minterm.

Clearly, we have chosen the second approach for the final generation.

4.4. BUILDING THE FINAL FORMULA

After every step of DES has been encoded, we obtain a set of formulae whose conjunction is exactly the formula $\mathcal{E}(C, K, P)$ that characterizes DES. To give an intuitive feeling of how it looks we present a sketch in Figure 3. For sake of readability, we have dropped the subscript corresponding to single bits and use only the superscript i to indicate the ith round of DES.

Table I. Number of minterms per output.

S-box	Multiple Outputs				Single Outputs			
	o_1	o_2	o_3	o_4	o_1	o_2	o_3	o_4
S1	20	20	21	22	17	16	19	20
S2	25	23	22	22	19	18	18	14
S3	25	24	25	23	22	17	18	18
S4	23	23	22	23	18	17	17	18
S5	22	24	23	21	17	23	17	17
S6	23	24	25	22	17	22	22	16
S7	23	22	24	26	19	16	18	23
S8	22	22	20	21	19	17	15	18

$$
\begin{aligned}
L^i &= R^{i-1},\\
R^i &= L^{i-1} \oplus F^i,\\
F^i &= P(S^i),\\
S^i &= \bigvee M^i,\\
M^i &= \bigwedge \pm X^i,\\
X^i &= E(R^{i-1}) \oplus K^i.
\end{aligned}
$$

Figure 3. Formulae encoding DES.

At this point, 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^r, and R^r to true or false, where r represents the total number of rounds considered.

Then a simplification phase composed by two operations starts:

1. A substitution of the variables defined by atomic equivalences* with the corresponding values to reduce the number of variables in other formulae, and to introduce the truth values.
2. A propositional simplification of the formulae using the rules listed in Table II.

Propositional simplification may introduce other atomic equivalences, and therefore the overall process is iterated until there are changes.

The formulae in Figure 4 are the result of the elimination of equivalences, and we have given some emphasis to the plaintext and the ciphertext, represented respectively by the values of the first and last rounds of the algorithm.

In a known plaintext-attack this formula can be simplified further, since L^0, R^0, L^r, and R^r are known, and we obtain the results in Figure 5.

* We define an atomic equivalence as a formula of the form $V \Leftrightarrow F$, where V is a variable and F is either another variable or a truth value.

Table II. Propositional simplification rules.

Initial Fml	Simplified	Initial Fml	Generated Fml
$A \Leftrightarrow X \wedge X$	$A \Leftrightarrow X$	$1 \Leftrightarrow A \wedge B$	$A \Leftrightarrow 1; B \Leftrightarrow 1$
$A \Leftrightarrow X \wedge 0$	$A \Leftrightarrow 0$	$0 \Leftrightarrow A \vee B$	$A \Leftrightarrow 0; B \Leftrightarrow 0$
$A \Leftrightarrow X \wedge 1$	$A \Leftrightarrow X$	$0 \Leftrightarrow A \oplus B$	$A \Leftrightarrow B$
$A \Leftrightarrow X \wedge \overline{X}$	$A \Leftrightarrow 0$	$1 \Leftrightarrow A \oplus B$	$A \Leftrightarrow \overline{B}$
$A \Leftrightarrow X \vee X$	$A \Leftrightarrow X$	$A \Leftrightarrow A \oplus B$	$B \Leftrightarrow 0$
$A \Leftrightarrow X \vee 0$	$A \Leftrightarrow X$	$\overline{A} \Leftrightarrow A \oplus B$	$B \Leftrightarrow 1$
$A \Leftrightarrow X \vee 1$	$A \Leftrightarrow 1$		
$A \Leftrightarrow X \vee \overline{X}$	$A \Leftrightarrow 1$		
$A \Leftrightarrow X \oplus X$	$A \Leftrightarrow 0$		
$A \Leftrightarrow X \oplus \overline{X}$	$A \Leftrightarrow 1$		
$A \Leftrightarrow X \oplus 0$	$A \Leftrightarrow X$		
$A \Leftrightarrow X \oplus 1$	$A \Leftrightarrow \overline{X}$		

$$
\begin{aligned}
&R^1 = L^0 \oplus S^1,\\
&R^2 = R^0 \oplus S^2,\\
&R^i = R^{i-2} \oplus S^i, \qquad j = 3, \ldots, r-2,\\
&R^{r-1} = R^{r-3} \oplus S^{r-1},\\
&R^r = R^{r-2} \oplus S^r,\\
&S^i = \bigvee M^i, \qquad i = 1, \ldots, r,\\
&M^i = \bigwedge \pm X^i, \qquad i = 1, \ldots, r,\\
&X^1 = R^0 \oplus K^1,\\
&X^i = R^{i-1} \oplus K^i, \qquad i = 2, \ldots, r-1,\\
&X^r = R^{r-1} \oplus K^r.
\end{aligned}
$$

Figure 4. DES formulae for r rounds.

The final outcome of the encoder is a formula that represents the logical relations between the key bits, the known plaintext, and the known ciphertext. Its structure is shown in Figure 6.

4.5. EXPERIMENTAL DATA

Since it makes no sense to do this encoding by hand, a program⋆ has been designed and implemented to generate formulae encoding DES in an entirely modular and automatic way.

⋆ More details on the encoder can be found in [29] and at the cited Web site.

$$
\begin{aligned}
&R^3 = \pm S^1 \oplus S^3, \\
&R^4 = \pm S^2 \oplus S^4, \\
&R^i = R^{i-2} \oplus S^i, && i = 5, \ldots, r-4, \\
&S^i = \bigvee M^i, && i = 1, \ldots, r, \\
&\pm S^{r-1} = R^{r-5} \oplus S^{r-3}, \\
&\pm S^r = R^{r-4} \oplus S^{r-2}, \\
&M^1 = \bigwedge \pm K^1, \\
&M^i = \bigwedge \pm X^i, && i = 2, \ldots, r-1, \\
&M^r = \bigwedge \pm K^r, \\
&X^2 = \pm S^1 \oplus K^2, \\
&X^3 = \pm S^2 \oplus K^3, \\
&X^i = R^{i-1} \oplus K^i, && i = 4, \ldots, r-3, \\
&X^{r-2} = \pm S^{r-1} \oplus K^{r-1}, \\
&X^{r-1} = \pm S^r \oplus K^r.
\end{aligned}
$$

Figure 5. DES formulae for r rounds with known plaintext and ciphertext.

Definitions

$$
\begin{aligned}
&M^i \Leftrightarrow \bigwedge \pm X^i, && 2 \leqslant i \leqslant r-1, \\
&S^i \Leftrightarrow \bigvee M^i, && 2 \leqslant i \leqslant r, \\
&X^{i+1} \Leftrightarrow S^i \oplus K^i, && 1 \leqslant r-1.
\end{aligned}
$$

Constraints

$$
\begin{aligned}
&M^1 \Leftrightarrow \bigwedge \pm K, \\
&M^r \Leftrightarrow \bigwedge \pm K^r, \\
&\pm S^{r-1} \Leftrightarrow \bigoplus\nolimits_i S^i, && i \text{ even}, \\
&\pm S^r \Leftrightarrow \bigoplus\nolimits_i S^i, && i \text{ odd}.
\end{aligned}
$$

Figure 6. The final encoding of the data encryption standard.

We can easily generate the formulae describing DES-like cryptosystems for any number of rounds up to 16, for any key and any pair of plaintext and ciphertext. All the permutation and substitution matrices (S-boxes) are read from external files so that one can change them according to 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.

The use of Espresso [42] 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 [25], and the TPTP non-clausal format [50].

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 300 MHz using Windows 98 and Linux. Here we report only the data on Solaris and Windows 98, the data using Linux being somehow intermediate between the other two.

Execution times were better on the Pentium. In particular for the full 16-rounds version of DES,

1. to generate the generic circuit are needed almost 25.96 seconds;
2. to generate the simplified circuit are needed almost 71 seconds.

It is also possible to use the algorithm to verify that we are indeed encoding a solved instance; i.e., we generate the simplified circuit, read plaintext, key and ciphertext and verify that everything simplifies to true. This took approximately 197 seconds. Proportionately smaller timings and memory requirements were found for reduced variant of DES. For instance one or two rounds can be done within one second.

Memory usage for the three previous cases is fairly substantial and is substantially smaller for the Unix machines:

1. to generate the generic formulae are needed 130 MB (167 MB for Windows 98);
2. to generate the simplified formulae are needed 136 MB (175 MB for Windows 98);
3. to verify the formulae are needed 137 MB (175 MB for Win 98).

These memory requirements are likely due to a poor (say nonexistent) memory management of our algorithm. With better memory management it could also probably take much less time because it would not have to use the swap space.

Beside memory and time we have collected a substantial amount of quantitative data regarding the size and nature of the final encoded formula. To this extent we have concentrated on the generation of solved instances of the SAT problem, according to the methodology that we have presented in Section 3. Thus our algorithm for the generation of the benchmark suite worked as follows:

1. fix the number of rounds of DES we are interested in;
2. generate randomly a key $\mathbf{K}$ (the solution v_K of the SAT problem);
3. generate randomly 400 blocks of plaintext $\mathbf{P}$ (a block is 64 bits);
4. encrypt the plaintext with the key using DES limited to the requested number of rounds and generate the ciphertext $\mathbf{C} = \mathrm{E}_{\mathbf{K}}(\mathbf{P})$;
5. encode the limited version of DES as a formula $\mathcal{E}(P, K, C)$ and substitute the values of the plaintext v_P and ciphertext v_C.

Table III. Occurrences of formulae per single plaintext/ciphertext pair.

Round	Equiv	Xor	And	Or	Clauses	Vars
1	520	0	504	16	1,645	243
2	1,042	0	1,010	32	3,304	489
3	1,738	48	1,609	80	9,064	1,430
4	2,432	96	2,208	128	14,811	2,368
5	3,096	176	2,760	160	18,738	3,032
6	3,760	256	3,312	192	22,665	3,696
7	4,424	336	3,864	224	26,592	4,360
8	5,088	416	4,416	256	30,519	5,024
9	5,752	496	4,968	288	34,446	5,688
10	6,416	576	5,520	320	38,373	6,352
11	7,080	656	6,072	352	42,300	7,016
12	7,744	736	6,624	384	46,227	7,680
13	8,408	816	7,176	416	50,154	8,344
14	9,072	896	7,728	448	54,081	9,008
15	9,736	976	8,280	480	58,008	9,672
16	10,400	1,056	8,832	512	61,935	10,336

Each block of plaintext and ciphertext gives a (satisfiable) formula $\mathcal{E}(v_P, K, v_C)$. We repeated the process for five randomly generated keys.

Table III reports the arithmetic mean of the number of equivalences, exclusive or, conjunctions, and disjunctions present in a formula $\mathcal{E}(v_P, K, v_C)$ for the various rounds of DES. For instance the fourth row shows that if we limit DES to four rounds (1st column) then $\mathcal{E}(v_P, K, v_C)$ is made (on average) by the conjunction of 2,432 equivalences out of which 96 contain exclusive or, 2,208 contain (multiple) conjunctions and 128 contain disjunctions. See again Figure 5 or Figure 6 for an intuitive idea of their shape.

The last two columns of Table III show the number of variables occurring in the formula that are different from the variables representing the key bits and the number of clauses that is generated when $\mathcal{E}(v_P, K, v_C)$ is translated into clausal normal form.

Note that the data in Table III are given per single pair of plaintext and ciphertext, and therefore the total number of the formulae, clauses, and so forth is obtained by multiplying those numbers by the number of plaintext and ciphertext pairs. For instance, the clauses corresponding to 4 blocks of plain/ciphertext for DES limited to 2 rounds are over 13,000 ($3,304 \times 4$) and the variables over 1,900 (489×4).

The variance of these numbers is not shown here because it is fairly small as we move beyond the fourth round. A simple explanation is that the known values

Table IV. Occurrences of key bits per single plaintext/ciphertext pair.

Round	Avg	Max	Round	Avg	Max
1	40.99	57	9	89.03	115
2	82.01	113	10	90.79	115
3	84.79	112	11	91.64	119
4	85.64	111	12	92.5	118
5	86.50	115	13	93.36	119
6	87.36	112	14	94.21	118
7	88.21	113	15	95.07	118
8	89.07	116	16	95.93	126

of the plaintext and the ciphertext do not propagate through the rounds as soon as exclusive or starts to appear.

The clause length varies quite widely, and therefore the mean clause length is not a good indicator. However, if one looks at the way in which clauses are generated, more information can be obtained:

- for every equivalence containing an *exclusive or* we obtain 8 clauses of length 3;
- for every equivalence containing a *conjunction* we obtain a number of binary clauses (usually less than 5–6) and a large clause with 5–6 literals;
- for every equivalence containing a *disjunction* we obtain a number of binary clauses (as for conjunctions) and one large clause.

So, there is an overwhelming component of binary clauses with ternary clauses starting to appear as soon as exclusive or starts to appear.

For these formulae, the clause over variable ratio (the standard indicator of the hardness of random 3-SAT formulae [48]) does not explain well the hardness of the problem. Indeed, consider the case of the commercial version of DES, whose ratio clauses over variables can be identified by the following formula (where b is the number of blocks, i.e., plaintext/ciphertext pairs, and k the number of key bits):

$$c/v = \frac{61,935 \times b}{k + 10,336 \times b}.$$

The number of key bits is less and less relevant as the number of blocks increases. With an infinite number of blocks, it converges to a fixed value. However, even if we add an infinite number of blocks, after 3 rounds the shift in the clause/variable ratio would be less than 1–2%. This seems to imply that adding more blocks should make the problem neither much easier nor much harder. As we shall see in Section 5, the experimental data contradict this hypothesis.

Table IV contains the average number of variables representing the key bits occurring in the formulae generated for each round.

Let Σ be a set of clauses using variables in V.

– if Σ is empty return SAT;

– if Σ contains an empty clause return UNSAT;

– (Unit Propagation) if Σ contains a unit clause $\{l\}$ then assign l the value true and call recursively DPLL on the simplified set of clauses;

– (Splitting Rule) Select a variable v in V, assign v a truth value and call recursively DPLL on the simplified set of clauses. If DPLL returns SAT then return SAT; otherwise assign v the opposite truth value and return the result of DPLL on the simplified set of clauses.

Figure 7. Davis–Putnam–Longemann–Loveland algorithm.

Not shown here is the fact that there are between 40 (1 round) and 90 (16 rounds) occurrences of each key bit. Moreover, key bits occur in approximately 17% of the total number of clauses and are the most frequently occurring variables, since other variables occur always in less than 100 formulae.

This means that almost all SAT heuristics, which are usually variants of "choose the literal with most occurrences such that X", will select almost only key bits in the search process.

5. Experimental 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 [15], 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 [51, 53], because it uses the trie data structure to boost the speed of the unit propagation phase and because it has been successfully used on the encoding of semi-groups problems;
– rel-SAT by Bayardo and Schrag [5], because it combines the Davis–Putnam algorithm with back-jumping and learning of CSP algorithms and because it has been successfully used on "real-world" problems in scheduling.

The backbone algorithm used by all three provers is the Davis–Putnam–Longemann–Loveland (DPLL) algorithm [18, 17] which we only sketch for reference in Figure 7. For the particular implementation details of each prover we refer to

corresponding paper. We recall that a literal is either a propositional variable or a negated propositional variable and that a clause is a set of literals. A unit clause is a clause with only one literal.

Among the various possibilities described in Section 3 we have focused on key search assuming a known plaintext attack because it is the simplest problem available among the SAT-problems we can generate, and because it generates solved instances. To generate the benchmark suite, we have followed the same methodology that we have presented in Section 4. The only difference here is that we have also grouped together the formulae corresponding to different plaintext using the same key. In other words we have tested the provers on the formulae

$$\mathcal{E}(v_C, K, v_P) \qquad \bigwedge_{i=1}^{2} \mathcal{E}(v_C^i, K, v_P^i) \qquad \ldots \qquad \bigwedge_{i=1}^{n} \mathcal{E}(v_c^i, K, v_P^i)$$

for $n = 1, 2, 4, 8$ to test whether increasing the number of constraints can lead to an increase of performance (i.e., decrease of running time) in the same fashion of what happens in random 3-CNF problems [15]. Indeed, having large quantities of plaintext is necessary for all standard cryptanalytic approaches that do not rely on exhaustive search.

All tested ATP systems admit different settings, and we have tried to stick to the recommended standards, eventually experimenting with heuristics. This was one of the main aims of the experiment: we wanted to see how a generic ATP system performed without being told any information on the problem, i.e., whether SAT-solvers were able to automatically detect and exploit the underlying hidden properties of DES which have been the subject of intense cryptographic research. Indeed the SAT-based approach make sense only if we just need to specialize our SAT-algorithm at the level of heuristics or just add features (such as reasoning with affine subproblems) that are of general interest.

In this framework, the only reasonable re-engineering of the heuristics is the limitation of the branching search variables to the known independent control variables, i.e., to the 56 bits of the key. We have explored this possibility with the various provers but without great successes. This contrasts with other applications of SAT-based encodings, such as planning, where branching on a restricted set of variables, the others being implied, considerably improves the performance [47]. We discuss this issue later in this section.

The tests have been run on SUN UltraSparcs running Solaris with 64 M RAM and on a Pentium II with the same memory and we obtained qualitatively the same results (to avoid normalization problems we only report here the CPU running time on the SUN machines).

For each prover, we report the results as follows: for each class of instances, characterized by the number of rounds of DES and by the number of blocks (i.e., number of $\mathcal{E}(v_C^i, K, v_P^i)$ formulae that are conjoined together) we report the *percentage of success* in finding a solution and the *mean time in seconds* necessary

to find a solution. For mean time we intend the arithmetic mean* over the total number of instances of the class (see Section 4). We also report the mean number of branches explored during the search, with the caveat that each algorithm has a slightly different notion of what constitute a branch. Hence, comparison of number of branches across different provers should only be made w.r.t. scaling rather than w.r.t. absolute values.

Since we didn't know whether the encoding of a plaintext and ciphertext pair would have admitted different models, i.e., keys,† whenever a model was found it was compared with the "original key" used to generate the instance. The average number of matching key bits (arithmetic mean over all instances) is then reported in the tables for each benchmark class.

To be fully complete, our experiments should have also explored another source of randomness: varying the keys, i.e., the solutions of the SAT instances, while keeping the plaintexts fixed (recall that in the bulk of the experiments the key is randomly generated but stay fixed for all the 400 plaintexts we have generated). We have investigated this possibility only for one of the algorithm (i.e., rel-SAT), and we have not found any significant pattern (although our experiments were limited to 50 different keys). Since cryptography research has also investigated key patterns without much results, whereas plaintexts and ciphertexts analysis has led to substantial attacks, we have postponed the analysis of key randomness to future works.

A further general observation concerning the organization of the experiments is the use of normalization: since all chosen ATP systems require clausal normal form we have used the CNF generation facilities provided by the encoder. Before using the clause file as input for the search algorithms, a preprocessing step was necessary to "compact" the integer values used for representing variables** since the encoding was so sparse that we exceeded the maximum number of atoms readable by the ATP systems. The running time of this auxiliary algorithm is not included in the running time of the provers.

The results are reported in separate subsections for each prover, and a final part with general observations concludes this section.

5.1. TABLEAU ON DES

Table V contains the running time of TABLEAU for the limited version of DES for which it could solve some problems. TABLEAU has been run with the standard setting, and eventually with purity disabled.

* We have used the mean time rather than the median time because the running times do not show the big outliers exhibited by random 3-SAT [15, 48].

† Recall that if we have more than one key for the same pair, the cipher is not faithful, and we have semi-weak keys. See again Section 2 for details.

** Our encoder used a redundant and sparse encoding so that one could detect from which round and which pair of ciphertext and plaintext a variable came.

Table V. Performance of TABLEAU.

Rounds	Blocks	% Success	Keybits	Branches	Mean Time (sec)
1	1	100%	33.7	1,8,325	12.96
1	2	100%	48.2	189,170	204.05
1	4	100%	50.7	64,813	117.95
2	1	100%	54.0	61,979	99.14
2	2	100%	55.8	80,133	222.07
2	4	100%	56.0	8,033	36.43

Table VIa. Performance of SATO searching for at least 2 models.

Rounds	Blocks	% Success	Models	Keybits	Branches	Time
1	1	22%	2	32.8	71	0.04
1	2	2%	2	40.5	75	0.07
1	4	48%	1 (75%)	50.8	63	0.12
1	8	100%	1 (100%)	52.0	19	0.10
2	1	72%	1 (30%)	53.9	641	0.54
2	2	88%	1 (84%)	56.0	150	0.25
2	4	98%	1 (98%)	56.0	29	0.16
2	8	100%	1 (100%)	56.0	6	0.15

Note that the problem becomes easier as we add more ciphertext and plaintext pairs. We stopped at 8 blocks because the formulae could not be read by TABLEAU: there were too many variables (see again Section 4 for the size of the problem). The running times had very limited variance, which is not shown for readability.

Yet, it is not all so easy: TABLEAU could not solve 3 rounds of DES in two hours, no matter the number of blocks. Increasing the time limit before timeout has been also tried (up to one day) but to no avail. Increasing the number of rounds did not lead to any solution either.

Re-engineering TABLEAU to branch only on key bits was too hard a task (because of the poor documentation of the program), so we have relied on an alternative implementation using the same heuristics which have been used for finding minimal models [27]. This further experimental analysis was again not conclusive: by restricting to independent variables we do not seem to gain much. In particular we are still not able to crack DES with three rounds.

Table VIb. Performance of SATO searching for 1 model.

Rounds	Blocks	% Success	Keybits	Branches	Time
2	1	76%	53.3	458	0.36
2	2	96%	55.6	118	0.19
2	4	100%	56	18	0.10
2	8	100%	56	9	0.14
3	8	50%	56	269,804	2,192.73

5.2. SATO ON DES

The timings of SATO are shown in Table VIb and Table VIa. At the beginning we have tried SATO with the standard setting and then with different options (such as searching for more than one model) and search heuristics. Table VIa refers to the standard settings, but we stopped the search after the second model has been found (or the search space exhausted). Table VIb is generated by stopping the search after the first model has been found.

Since we have not limited the search to the first model, this explains the greater running time and number of branches in comparison with TABLEAU. In the column "Models" we show between brackets the number of instances on which only one model was found. Also in this case the variance of the running time was limited, and it suggests that the running time distribution of the problem does not exhibit a heavy tail behavior.

Most of SATO's failures are due to virtual memory failure rather than the two hours timeout. Thus, we have decided to run SATO on a Pentium II with 64 M but with a swap space of over 628 M. It turned out that if we let SATO run on three rounds of DES without time limit we could solve all instances within few minutes of actual CPU time. Yet, if we looked at "real" time, it took over 12 hours for a single instance to be solved. In practice, the time is spent by swapping the trie data structure in and out main memory and then performing very few quick operations on each part of the trie. Therefore, we have decided not to include these latest running times into the table: they may give a wrong idea of the computational resources needed by SATO.

For the case of one round and eight blocks, the strange results that we get are for only one model, and yet only 52 out of 56 bits can be explained by the fact that not all key bits occur in the corresponding formulae and SATO has been used with purity on (that is setting to false any bit that does not occur).

With SATO we have also experimented with different heuristics and a new feature of the algorithm that makes it possible to generate lemmata corresponding to closed branches. We have also experimented with a simple list structure rather than the trie data structure. The worst heuristics for this problem seemed to be that

proposed by Jeroslow and Wang. Without the trie data structure, SATO offered the same qualitative performance of TABLEAU.

In this case we also tried to restrict the selection of the branching literals to the 56 independent control variables. The re-engineering of the algorithm was done by Zhang, and again [52] it led to worse results and was abandoned.

5.3. rel-SAT

In DES rel-SAT is a variant of the Davis–Putnam algorithm, enhanced with conflict directed back-jumping and learning [5] and is indeed the only algorithm that solves three rounds of DES within few minutes.

The basic working of the enhanced algorithm is worth repeating:

- 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 has caused the contradiction;
- the clause responsible for the contradiction is resolved with a clause representing the temporary assignment; the resolvent is 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).

More details and references can be found in [5]. The important parameter is the learning factor, i.e., the size of the clauses that can be learned during the backtrack plus resolution stage.

Also in this case, the testing started with the recommend standard: a small learning factor (4), using relevance-based learning (see Table VIIa). For up to two rounds of DES rel-SAT is slightly faster but still comparable with the results of Davis–Putnam-like algorithms such as SATO and TABLEAU or local search algorithm such as Walk-SAT [31]. Again, the performance of the algorithm increases with the number of blocks (plaintext-ciphertext pairs). Adding more constraints makes the search for the only (?) existing solution easier. The success rate (100%) is omitted. The variance of the running times is fairly limited but for the case of 3 rounds of DES and 2 blocks.

Other settings were tried; in particular, we tried both without any learning at all and with larger learning factors. The analysis shows that with no learning we have a decrease in performance (see Table VIIb). With too large a learning factor (order of 20) there is not a big gain in performance. With slightly larger learning factors (order of 5–6) we get a slight but not substantial increase in performance.

Since rel-SAT has the best readable code of the three ATP systems, we experimented with a small re-engineering in the attempt to exploit the knowledge of the domain. In the original algorithm, a first selection of potential branching variables

Table VIIa. Performance of rel-SAT with learning factor 4.

Rounds	Blocks	Keybits	Branches	Time	
		(Mean)	(Mean)	(Mean)	(Variance)
1	1	31.0	28	0.02	0.009
1	2	48.9	105	0.11	0.047
1	4	50.9	104	0.22	0.083
1	8	52.0	83	0.45	0.103
2	1	54.0	231	0.20	0.105
2	2	56.0	111	0.23	0.106
2	4	56.0	68	0.36	0.123
2	8	56.0	57	0.81	0.252
3	1	–	–	$\geqslant$ 1 h	–
3	2	56.0	174,612	983.22	1,034.598
3	4	56.0	19,312	159.13	66.894
3	8	56.0	3,596	75.03	32.783

Table VIIb. Performance of rel-SAT with no learning.

Rounds	Blocks	Keybits	Branches	Time	
		(Mean)	(Mean)	(Mean)	(Variance)
1	4	53.6	172	2.38	1.165
1	8	53.0	185	6.18	1.866
2	4	56.0	157	4.98	3.006
2	8	56.0	103	8.00	6.449
3	4	–	–	$\geqslant$ 1 h	–
3	8	56.0	8,154	822.35	164.792

is done, and then all variables with a value of the heuristic function lower than a threshold were discarded. The modified algorithm didn't check the threshold if the selected variable was a keybit. In this way the algorithm gives preferences to dependent variables with very good properties or independent variables with medium properties. However, the running time of the algorithm didn't improve substantially.

Since this was also the best performing algorithm we have tried, including BDDs [4] and Walk-SAT [31], we have performed more experiments (on a SUN with 256 MB) to determine whether the search space of the problem using rel-SAT was actually smaller than the search space generated by brute force search.⋆ This experiment can show the potential of the CSP/SAT approach to the problem

⋆ This experiment was suggested by an anonymous reviewer and the editors.

Table VIII. Further experiments with rel-SAT with learning factor 5.

Rnds	Blcks	Keybits	Branches	Time		Max Stack Depth	
		(Mean)	(Mean)	(Mean)	(Var)	(Mean)	(Abs)
1	1	31.6	25	0.02	0.010	22	37
1	2	46.1	83	0.09	0.048	17	26
1	4	51.9	97	0.21	0.059	17	20
1	8	52.0	98	0.50	0.068	17	20
2	1	54.7	238	0.22	0.129	16	20
2	2	55.9	108	0.22	0.091	15	18
2	4	56.0	66	0.34	0.116	14	17
2	8	56.0	45	0.66	0.170	14	19
3	2	56.0	99,908	1,314.94	3,078.913	31	34
3	4	56.0	8,883	105.40	37.255	30	33
3	8	56.0	5,491	142.57	23.067	29	31

if constraint propagation substantially reduces the search space, even in absence of specific, problem-dependent information.

The results of this new set of experiments, with a slightly larger learning factor (5), are reported in Table VIII. Beside mean, variance, and number of key bits found, we also report the absolute maximum depth of the stack during the search over all instances and the average maximum depth of the stack over all instances (i.e., for each instance we compute the maximum stack depth and then take the arithmetic mean over all instances). As we can see we are substantially below the 55 choice points that would have been necessary with brute force search. A caveat is that the table also suggests a trend of increasing search depth as the number of rounds increases. This may explain why four rounds cannot be solved by current SAT tools.

In Table IX we have described the results of the experiments when we keep the plaintext fixed and change the keys, i.e., the solution of the SAT-problem. Even if the significance of the data is limited (50 different keys on 10 different plaintexts), we can observe that there is not a big difference between these data and those reported in Table VIII.

Given these promising results, we further engineered the algorithm to accept larger formulae with more variables and tried on 4, 8 and a full 16 rounds of DES, also using 1, 2, and 4 blocks. The algorithm didn't return within one day.

5.4. GENERAL OBSERVATIONS

From a cryptanalyst viewpoint these results are not very impressive: three rounds of DES were cryptanalyzed long ago [3], and linear cryptanalysis [32] or differential

Table IX. Changing the keys for rel-SAT with learning factor 5.

Rnds	Blcks	Keybits	Branches	Time		Max Stack Depth	
		(Mean)	(Mean)	(Mean)	(Var)	(Mean)	(Abs)
3	4	56.0	8,892	285.37	160.922	29	35
3	8	56.0	3,641	260.14	108.360	28	32

cryptanalysis [7] can do much better for up to eight rounds, although they need far larger amount of plaintexts than just eight blocks.

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. They do not know which of the thousands of clauses are just definitions in disguise and which are really constraints. With four blocks of plaintext and two rounds of DES they search in a space of almost $2^{2,000}$ solutions, and still they find the only one that exists. The constraints present in the encoding are sufficient to drastically reduce the search space. This is fairly close to what happens for the DIMACS parity problems, where SAT procedures do manage to find the independent variables from the much larger set of all variables.

What makes these results puzzling is that knowing which are the "variables that count" does not seem to help. Moreover, the hardness of the problem exhibits an abrupt jump: reasonably easy up to two rounds, then moderately hard at three, and then simply unsolvable.

We can explain the good performance by observing that there are many defined variables. Thus, few wrong assignments to control variables are enough to provoke a cascade of assignments to defined variables and then inconsistencies are quickly found. However, this would imply that formulae encoding more rounds should be easier and not harder. Indeed, ciphers are designed to exhibit an "avalanche" effect when the number of rounds is increased: the number of output bits influenced by the key bits increases in a non-linear fashion. With more rounds the propagation of inconsistencies should also be boosted. Since this is not the case, it seems the propagation of assignments is hindered.

We conjecture that the source of the complexity is the particular structure of the Feistel-ciphers, i.e., the xors that are added at each round. What really hinders state-of-art SAT solvers is the inability of handling affine subproblems in an efficient way. Although these problems are themselves polynomially solvable [44], their presence, such as in the hidden parity bit mentioned in [47], makes the difference between hard and unsolvable problems.

This would also explain the abrupt jump in complexity: DES at four rounds is the first problem where exclusive or starts to appear. If we remove those formulae the resulting formula can be shown to be satisfiable by all ATP systems without much effort.

The fact that there is no heavy tail phenomenon as in random 3-SAT can also be explained: the structure of the problem is fairly regular, and its hardness is well controlled by the number of blocks of known plaintext, whose increase makes the problem easier, and the number of rounds of DES, whose increase makes the problem harder.

What remains to be explained is why "knowing the control variables" does not help. We have already noted at the end of Section 4 that the key bits are the variables that occur most often and therefore it may simply be that knowing the control variables does not help because *in this problem the independent variables are already among the variables preferred by the standard heuristics*. An informal analysis of the stack of choice points of rel-SAT revealed that almost all variables that are selected by the standard heuristics are key variables. Thus, by giving the algorithm the full choice on the branching variables we do not give away our bias toward control variables (because the problem is such that the heuristics will likely choose them) and still gives the algorithm the possibility of exploiting shortcuts with defined variables.

A second observation is that branching on control variables alone is what brute force approaches do. In this respect, looking at linear and differential cryptanalysis might be instructive. If we set aside the problem of gathering the massive amount of data they need, we may observe that they can solve the problem better than we do because

1. they exploit the probability distribution of intermediate results to guess their value (this corresponds to splitting on defined variables in the SAT-based approach) without concentrating on key bits only;
2. they make use of the affine subproblems to determine the value of some independent variables out of the dependent variables they have set (whereas we are swamped by them because we perform search on them);
3. only after the first two steps have given an indication of a plausible solution they start searching on control variables alone.

So, we may conclude this point by observing that our ATP systems have been able to do the first step almost correctly by using their sophisticated heuristics, without human intervention, but they have not been able to do the second.

Another interesting point regards the integrity of DES, i.e., the number of keys that are consistent with the same pair of plaintext and ciphertext. If more than one key would be consistent with a plaintext and ciphertext pair (even worse that there was a key consistent with different plaintexts and ciphertexts), we would have discovered a trapdoor. Since only one key exists (at least for three rounds) this offers an independent evidence that DES was not designed to have hidden trapdoors (beside semi-weak keys). Since the number of models actually decreases quickly to one as the number of rounds or the number of plaintexts increase, this is a strong evidence that DES is almost faithful. Note that the outcome of the experiment with

SATO can be easily converted into a *proof* that for each tested pair there is only one key which could have generated it.

It is worth noting that a number of extensive tests with BDDs has been also carried at our Department by Ascione [4], since BDDs are known to perform well on non-clausal problems [8]. Even on our optimized encoding, BDDs could not solve key search problems any better than SAT-based approaches [4]: the tests showed that the BDDs representing the encoded formula reached quickly a million and over nodes and then crashed the systems for virtual memory failure.

The tests with BDDs were, however, useful for confirming the experimental data regarding the integrity of DES: when the number of known plaintexts or the number of rounds increases the final BDD representing $\mathcal{E}(v_P, K, v_C)$ turns out to be a chain (i.e., there is only one model).

6. Open Problems and Conclusions

In this paper we have presented an application of propositional reasoning and search to a key security problem of industrial relevance.

We have shown that the translation of properties of ciphers in logic is doable (although not simple) and that a number of key properties can be modeled.

The effectiveness of general ATP systems in this framework has also been preliminarily tested and looks promising, although ATP search strategies need to be improved to meet the full challenge provided by this benchmark. They seem promising because only one or few blocks of plaintext and ciphertext are necessary to identify a unique solution to the cryptographic problem (rather than the few hundreds or millions required by traditional methods).

Thus, we believe that the whole approach on encoding cryptographic problems as SAT problems can be a step forward toward the development of generators "for problem instances that have computational properties that are more similar to real world instances" [47]. Even if cryptography is a limited application area (although an important one), the structure of the corresponding SAT problems is such that a generic ATP system able to cope with them (e.g., able to handle affine subproblem) will be able to apply its techniques effectively to hard problems from other application areas.

This approach also offers a solution to the problem discussed by Cook and Mitchell [14]: how to randomly generate solved instances that are hard to solve.

Still, there is a lot of work to be done, and a good conclusion of this paper may just be the indication of open problems.

PROBLEM 1. *Find a key for the commercial* 16 *round Data Encryption Standard in less than* 56 *hours using off-the-shelf h/w and s/w but enhanced ATP systems.*

This is the subject of current investigation and, as we have already mentioned, our focus is the enhancement of ATP system with rules and techniques to cope with affine subproblems.

Further experiments are needed to better understand the structure of the problem and which heuristics may be better suited to solve it.

PROBLEM 2. *Prove formally that the commercial* 16 *round Data Encryption Standard is not a group and is not pure, using the encoding and an automated theorem prover.*

This might be an independent way to formally guarantee the experimental evidence obtained in [12] with cycling experiments.

PROBLEM 3. *Prove formally that the commercial* 16 *round Data Encryption Standard DES is* faithful *according to the definition in* [26]*, that is, for any pair of plaintext and ciphertext there is only one key that can generate that pair* (*setting semi-weak keys aside*).

Notice that this, in contrast with the previous two results, is an open problem for cryptography. If a solution of this problem could be found, this would have extreme relevance for the use of a cipher in applications such as electronic commerce where the impossibility of forging data is essential. This could become an established method to guarantee that a cipher has such properties.

Other problems involve QBF theorem proving, and this field is still not mature enough to tackle such hard problems. A preliminary test has been done using the algorithm in [11], but it didn't return in one hour even for reduced version of the algorithm limited to one or two rounds. New tests with an enhanced version of the algorithm that exploits defined variables are in preparation.

PROBLEM 4. *Develop heuristic techniques for propositional reasoning and search that work with every Feistel-type cipher with data-independent permutations like DES.*

The highly regular structure of these ciphers should be exploited by search algorithms in two directions: the identification of no-goods that prune a substantial amount of the search space and the variable selection heuristics. Since the operations are data independent, a certain amount of preprocessing for the internal rounds could be done off-line. This is a case where knowledge compilation [46] may entirely pay off. Another algorithm that may work is Stalmark algorithm, since it uses a data structure that well fits with those of a Feistel-cipher [24].

PROBLEM 5. *Find efficient encodings of Feistel-type ciphers with data-dependent permutations like RC5* [40, 45].

Of course, a straightforward encoding is always possible: just translate the cipher into a circuit and this into propositional logic. Unfortunately, we have noted this is already unworkable for DES.

PROBLEM 6. *Find efficient encoding into propositional* (*or any*) *logic of public-key cryptographic algorithms for digital signatures based on number theory such as RSA* [45].

This problem has been suggested by Cook and Mitchell [14] and might be the hardest, since the way in which the algorithm is expressed (number theory) is fairly remote from propositional logic. Again, one can just encode multipliers and adders into propositional logic but this might be overwhelming. Moreover, factoring is known to be hard (see, e.g., [45, p. 256] for related references), and it may be that SAT-based techniques will be inferior to factoring algorithms.

The coming adoption of a U.S. Advanced Encryption Standard [37] may open new directions of research.

As for all "real-world" problems, there might be a dark side: the measure of success might be the "privilege" (!?) of successful automated reasoning tools being denied export licenses as dangerous weapons.

Acknowledgments

We thank L. Carlucci Aiello for her encouragement and support, M. Ascione for discussing with us his results on the logical cryptanalysis of DES using BDDs, and P. Baumgarten, P. Liberatore, I. Niemela, and H. Zhang for testing our problem using their provers, and for many useful discussions. We also thank all members of the AI group at the Dipartimento of Informatica e Sistemistica in Rome for bearing with us and our machine-swamping experiments. The comments and suggestions from the anonymous reviewers greatly improved the quality of this paper.

F. Massacci acknowledges the support of the CNR fellowship 201-15-9 at the Dipartimento of Informatica and Sistemistica at the Univ. di Roma I "La Sapienza". This work has been partly supported by CNR, MURST and ASI grants of the Dipartimento di Informatica e Sistemistica at the Univ. di Roma I "La Sapienza" and by the AMTEC research grant of the Dipartimento di Ingegneria dell'Informazione at the Univ. di Siena.

References

1. Abadi, M. and Needham, R.: Prudent engineering practice for cryptographic protocols, *IEEE Trans. Software Engng.* **22**(1) (1996), 6–15.
2. Anderson, R. and Needham, R.: Programming Satan's computer, in *Computer Science Today – Recent Trends and Developments*, Lecture Notes in Comput. Sci. 1000, Springer-Verlag, 1996, pp. 426–440.

3. Andleman, D. and Reeds, J.: On the cryptanalysis of rotor machines and substitution-permutations networks, *IEEE Trans. Inform. Theory* **28**(4) (1982), 578–584.
4. Ascione, M.: Validazione e benchmarking dei BDD per la criptanalisi del data encryption standard, Master's thesis, Facoltà di Ingegneria, Univ. di Roma I "La Sapienza", March 1999. In Italian.
5. Bayardo, R. and Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances, in *Proc. of the 14th Nat.* (*US*) *Conf. on Artificial Intelligence* (*AAAI-97*), AAAI Press/The MIT Press, 1997, pp. 203–208.
6. Biham, E. and Biryukov, A.: An improvement of Davies' attack on DES, in *Advances in Cryptology – Eurocrypt 94*, Lecture Notes in Comput. Sci., Springer-Verlag, 1994.
7. Biham, E. and Shamir, A.: Differential cryptanalysis of DES-like cryptosystems, *J. Cryptology* **4**(1) (1991), 3–72.
8. Bryant, R.: Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Computers* **35**(8) (1986), 677–691.
9. Büning, H., Karpinski, M. and Flögel, A.: Resolution for quantified Boolean formulas, *Inform. Comput.* **117**(1) (1995), 12–18.
10. Burrows, M., Abadi, M. and Needham, R.: A logic for authentication, *ACM Trans. Comput. Systems* **8**(1) (1990), 18–36.
11. Cadoli, M., Giovanardi, A. and Schaerf, M.: An algorithm to evaluate quantified Boolean formulae, in *Proc. of the 15th* (*US*) *Nat. Conf. on Artificial Intelligence* (*AAAI-98*), AAAI Press/The MIT Press, 1998, pp. 262–267.
12. Campbell, K. and Weiner, M.: DES is not a group, in *Proc. of Advances in Cryptography* (*CRYPTO-92*), Lecture Notes in Comput. Sci., Springer-Verlag, 1992, pp. 512–520.
13. Claesen, L. (ed.): *Formal VLSI Correctness Verification: VLSI Design Methods*, Vol. II, Elsevier Science Publishers, North-Holland, 1990.
14. Cook, S. and Mitchel, D.: Finding hard instances of the satisfiability problem: A survey, in *Satisfiability Problem: Theory and Applications*, Vol. 35, DIMACS Series in Discrete Math. Theoret. Comput. Sci. Amer. Math. Soc., 1997, pp. 1–17.
15. Crawford, J. and Auton, L.: Experimental results on the crossover point in random 3SAT, *Artif. Intell.* **81**(1–2) (1996), 31–57.
16. Cryptography Research Inc. DES key search project information, Technical report, Cryptography Research Inc., 1998. Available on the web at http://www.cryptography.com/des/.
17. Davis, M., Longemann, G. and Loveland, D.: A machine program for theorem-proving, *Comm. ACM* **5**(7) (1962), 394–397.
18. Davis, M. and Putnam, H.: A computing procedure for quantificational theory, *J. ACM* **7**(3) (1960), 201–215.
19. De Millo, R., Lynch, L. and Merrit, M.: Cryptographic protocols, in *Proc. of the 14th ACM SIGACT Symposium on Theory of Computing* (*STOC-82*), 1982, pp. 383–400.
20. Feistel, H., Notz, W. and Smith, L.: Some cryptographic techniques for machine-to-machine data communication, *Proc. of the IEEE* **63**(11) (1975), 1545–1554.
21. Gomes, C. and Selman, B.: Problem structure in the presence of perturbation, in *Proc. of the 14th Nat.* (*US*) *Conf. on Artificial Intelligence* (*AAAI-97*), AAAI Press/The MIT Press, 1997.
22. Gomes, C., Selman, B. and Crato, N.: Heavy-tailed distributions in combinatorial search, in *Third Internal. Conf. on Principles and Practice of Constraint Programming* (*CP-97*), Lecture Notes in Comput. Sci. 1330, Springer-Verlag, 1997, pp. 121–135.

23. Group of Experts on Information Security and Privacy. Inventory of controls on cryptography technologies, OLIS DSTI/ICCP/REG(98)4/REV3, Organization for Economic Co-operation and Development, Paris, Sep. 1998.
24. Harrison, J.: Stalmarck's algorithm as a HOL derived rule, in *Proc. of the 9th Internal. Conf. on Theorem Proving in Higher Order Logics* (*TPHOLs'96*), Lecture Notes in Comput. Sci. 1125, Springer-Verlag, 1996, pp. 221–234.
25. Johnson, D. and Trick, M. (eds): *Cliques, Coloring, Satisfiability: The Second DIMACS Implementation Challenge*, AMS Series in Discrete Math. and Theoret. Comput. Sci. 26, Amer. Math. Soc., 1996.
26. Kaliski, B., Rivest, R. and Sherman, A.: Is the Data Encryption Standard a group? (preliminary abstract), in *Advances in Cryptology – Eurocrypt 85*, Lecture Notes in Comput. Sci. 219, Springer-Verlag, 1985, pp. 81–95.
27. Liberatore, P.: Algorithms and experiments on finding minimal models, Technical Report 09-99, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999.
28. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR, in *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Comput. Sci. 1055, Springer-Verlag, 1996, pp. 147–166.
29. Marraro, L.: Analisi crittografica del DES mediante logica booleana, Master's thesis, Facolta di Ingegneria, Univ. di Roma I "La Sapienza", December 1998. In Italian.
30. Marraro, L. and Massacci, F.: A new challenge for automated reasoning: Verification and cryptanalysis of cryptographic algorithms, Technical Report 05-99, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999.
31. Massacci, F.: Using walk-SAT and rel-SAT for cryptographic key search, in *Proc. of the 16th Internat. Joint Conf. on Artificial Intelligence* (*IJCAI-99*), Morgan Kaufmann, 1999, pp. 290–295.
32. Matsui, M.: The first experimental cryptanalysis of the Data Encryption Standard, in *Proc. of Advances in Cryptography* (*CRYPTO-94*), Lecture Notes in Comput. Sci. 839, Springer-Verlag, 1994, pp. 1–11.
33. Matsui, M.: Linear cryptanalysis method for DES cipher, in *Advances in Cryptology – Ewocrypt 93*, Lecture Notes in Comput. Sci. 765, Springer-Verlag, 1994, pp. 368–397.
34. Mitchell, J., Mitchell, M. and Stern, U.: Automated analysis of cryptographic protocols using Murphi, in *Proc. of the 16th IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, 1997, pp. 141–151.
35. Organization for Economic Co-operation and Development 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.
36. 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.
37. 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.
38. Committee on Payment, Settlement 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.
39. Paulson, L.: The inductive approach to verifying cryptographic protocols, *J. Comput. Security* (1998).

40. Rivest, R.: The RC5 encryption algorithm, in *Proc. of the Fast Software Encryption Workshop* (*FSE-95*), Lecture Notes in Comput. Sci. 1008, Springer-Veriag, 1995, pp. 86–96.
41. Rudell, R.: Espresso 1OCTTOOLS, January 1988.
42. Rudell, R. and Sangiovanni-Vincentelli, A.: Multiple valued minimization for PLA optimization, *IEEE Trans. Comput. Aided Design.* **6**(5) (1987), 727–750.
43. Ryan, P. and Schneider, S.: An attack on a recurive authentication protocol: A cautionary tale, *Inform. Process. Lett.* **65**(15) (1998), 7–16.
44. Schaefer, T.: The complexity of satisfiability problems, in *Proc. of the 10th ACM Symposium on Theory of Computing* (*STOC-78*), ACM Press and Addison Wesley, 1978, pp. 216–226.
45. Schneier, B.: *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Wiley, 1994.
46. Selman, B. and Kautz, H.: Knowlege compilation and theory approximation, *J. ACM* **43**(2) (1996), 193–224.
47. Selman, B., Kautz, H. and McAllester, D.: Ten challenges in propositional resoning and search, in *Proc. of the 15th Internat. Joint Conf. on Artificial Intelligence* (*IJCAI-97*), Morgan Kaufmann, Los Altos, 1997.
48. Selman, B., Mitchell, D. and Levesque, H.: Generating hard satisfiability problems, *Artif. Intell.* **81**(1–2) (1996), 17–29.
49. Shannon, C.: Communication theory of secrecy systems, *Bell System Technical J.* **28** (1949), 656–715.
50. Suttner, C. and Sutcliffe, G.: The CADE-14 ATP system competition, *J. Automated Reasoning* **21**(1) (1998), 99–134.
51. Zhang, H.: SATO: An efficient propositional prover, in *Proc. of the 14th Internat. Conf. on Automated Deduction* (*CADE-97*), Lecture Notes in Comput. Sci., 1997.
52. Zhang, H.: Personal communication, Nov. 1998.
53. Zhang, H. and Stickel, M.: An efficient algorithm for unit-propagation, in *Proc. of the 4th Internat. Symposium on AI and Mathematics*, 1996.