SAT as a programming environment for linear algebra and cryptanalysis

Marian Srebrny
Institute of Computer Science
Polish Academy of Sciences
Warsaw, Poland
marians@ipipan.waw.pl

Mateusz Srebrny
Gadu–Gadu S.A.
Warsaw, Poland
m.srebrny@gadu-gadu.pl

Lidia Stępień
Institute of Mathematics
and Computer Science
Jan Długosz University
Częstochowa, Poland
l.stepien@ajd.czest.pl

Dedicated to Victor Marek
on his 65th birthday

Abstract

In this paper we present an application of the propositional SATisfiability environment to computing some simple orthogonal matrices and some interesting tasks in the area of cryptanalysis. We show how one can code a search for some kind of desired objects as a propositional formulae in such a way that their satisfying valuations code such objects. Some encouraging (and not very encouraging) experimental results are reported for the proposed propositional search procedures using the currently best SAT solvers.

In this paper we pursue a propositional programming paradigm. To solve your problem: (1) translate the problem to SAT (in such a way that a satisfying valuation represents a solution to the problem); (2) run the currently best SAT checker to solve it for you. The propositional encoding formula can be thought of as a declarative program. The hope you can get a solution relatively fast is based on the fact that the SAT solving algorithm is one of the best optimized.

A SAT solving algorithm decides whether a given propositional (Boolean) formula has a satisfying valuation. SAT was the first known NP-complete problem, as proved by Stephen Cook in 1971. Finding a satisfying valuation is infeasible in general, but many SAT instances can be solved surprisingly efficiently. There are many competing algorithms for it and many implementations, most of them were developed over the last two decades as highly optimized versions of the DPLL procedure of (Davis & Putnam 1960) and (Davis, Logeman & Loveland 1962).

In the area of cryptanalysis we apply that idea to try out the power of the SAT solvers in breaking two of the currently most exciting challenges: RSA and SHA-1. Although our experimental results have not turned out to be a success in breaking those cryptosystems, they seem interesting in their own right as reasonable testing benchmarks for the SAT solvers.

Similarly, some formulae encoding search for orthogonal matrices in some linear spaces over Galois field F_2 are presented below.

In the next section we present an overview of our propositional translation of the RSA (factorization) problem and our experimental results on performance of the best SAT solvers on the RSA propositional formula. Section 2 is devoted to another currently most challenging cryptosystem SHA-1, our translation of it into propositional calculus and our experiments with the SAT solvers on it. In section 3 we run the MiniSat solver on a propositional formula encoding orthogonality of square matrices over Galois field F_2. The last section contains some conclusion and open problems.

1 RSA (factorization) and SAT

For the story of RSA, the best known cipher, we refer the interested reader to (Menezes, van Oorschot & Vanstone 2001) and (RSA labs 2007). For the sake of next sections, we only recall here that breaking RSA amounts to integer factorization of a given positive integer n of the form $n = p * q$ with unknown prime factors p and q. Usual requirements are: n large, p and q of similar bit length, p and q cryptographically strong. (See (Menezes, van Oorschot & Vanstone 2001).) No polynomial time factorization algorithm is known, and all non-polynomial time algorithms are not feasible. In other terms, performing over 2^{60} instructions is considered infeasible for today.

We implement RSA as a propositional formula, rsasat: given n, we generate a propositional formula so that its satisfying valuation encodes two integer factors p and q of n.

We represent an l-bit integer p as l propositional variables $P_0, \ldots, P_{l-1}$. E.g., $13 = (1101)_2$ is represented as formula $P = 13$: $P_3 \wedge P_2 \wedge \neg P_1 \wedge P_0$. Formula $R = P$ represents equality $r = p$: $\bigwedge_{i=0}^{l-1}(R_i \wedge P_i) \vee (\neg R_i \wedge \neg P_i)$. Its conjunctive normal form, CNF, is: $\bigwedge_{i=0}^{l-1}(R_i \vee \neg P_i) \wedge (P_i \vee \neg R_i)$.

$R = 2P$ represents $r = 2p$: $\neg R_0 \wedge \bigwedge_{i=1}^{l-1}(R_i \wedge P_{i-1}) \vee (\neg R_i \wedge \neg P_{i-1})$.

We write $R = P + Q$ to represent $r = p + q$ with $(C_0, C_1, \ldots, C_l)$ representing the carry bits. For $i > 0$, we need $(C_i \wedge ((C_{i-1} \wedge P_i) \vee (C_{i-1} \wedge Q_i) \vee (P_i \wedge Q_i))) \vee (\neg C_i \wedge ((\neg C_{i-1} \wedge \neg P_i) \vee (\neg C_{i-1} \wedge \neg Q_i) \vee (\neg P_i \wedge \neg Q_i)))$. Its CNF is:
$(\neg C_i \vee P_i \vee C_{i-1}) \wedge (\neg C_i \vee P_i \vee Q_i) \wedge$
$(\neg C_i \vee Q_i \vee C_{i-1}) \wedge (C_i \vee \neg P_i \vee \neg C_{i-1}) \wedge$
$(C_i \vee \neg P_i \vee \neg Q_i) \wedge (C_i \vee \neg Q_i \vee \neg C_{i-1})$.
It gives the result for $R = P + Q$ as:

$(R_i \wedge ((C_{i-1} \wedge \neg P_i \wedge \neg Q_i) \vee (\neg C_{i-1} \wedge P_i \wedge \neg Q_i) \vee$
$(\neg C_{i-1} \wedge \neg P_i \wedge Q_i) \vee (C_{i-1} \wedge P_i \wedge Q_i))) \vee$
$(\neg R_i \wedge ((C_{i-1} \wedge P_i \wedge \neg Q_i) \vee (\neg C_{i-1} \wedge P_i \wedge Q_i) \vee$
$(C_{i-1} \wedge \neg P_i \wedge Q_i) \vee (\neg C_{i-1} \wedge \neg P_i \wedge \neg Q_i)))$.
Its CNF is: $(R_i \vee Q_i \vee P_i \vee \neg C_{i-1}) \wedge (R_i \vee Q_i \vee \neg P_i \vee C_{i-1}) \wedge$
$(R_i \vee \neg Q_i \vee P_i \vee C_{i-1}) \wedge (R_i \vee \neg Q_i \vee \neg P_i \vee \neg C_{i-1}) \wedge$
$(\neg R_i \vee Q_i \vee P_i \vee C_{i-1}) \wedge (\neg R_i \vee Q_i \vee \neg P_i \vee \neg C_{i-1}) \wedge$
$(\neg R_i \vee \neg Q_i \vee P_i \vee \neg C_{i-1}) \wedge (\neg R_i \vee \neg Q_i \vee \neg P_i \vee C_{i-1})$.
The whole $R = P + Q$ can now be written in the conjunctive normal form as:
$\neg C_0 \wedge \neg C_l \wedge \bigwedge_{i=1}^{l}((\neg C_i \vee P_i \vee C_{i-1}) \wedge (\neg C_i \vee P_i \vee Q_i) \wedge$
$(\neg C_i \vee Q_i \vee C_{i-1}) \wedge (C_i \vee \neg P_i \vee \neg C_{i-1}) \wedge (C_i \vee \neg P_i \vee \neg Q_i) \wedge$
$(C_i \vee \neg Q_i \vee \neg C_{i-1})) \wedge \bigwedge_{i=0}^{l-1}((R_i \vee Q_i \vee P_i \vee \neg C_i) \wedge (R_i \vee$
$Q_i \vee \neg P_i \vee C_i) \wedge (R_i \vee \neg Q_i \vee P_i \vee C_i) \wedge (R_i \vee \neg Q_i \vee \neg P_i \vee$
$\neg C_i) \wedge (\neg R_i \vee Q_i \vee P_i \vee C_i) \wedge (\neg R_i \vee Q_i \vee \neg P_i \vee \neg C_i) \wedge$
$(\neg R_i \vee \neg Q_i \vee P_i \vee \neg C_i) \wedge (\neg R_i \vee \neg Q_i \vee \neg P_i \vee C_i))$.

Similarly, we write $N = PQ$ to represent $n = pq$ via the bit operations. Since

$$pq = q_0p + q_1 2p + q_2 2^2 p + \ldots + q_{l-1} 2^{l-1} p$$

we eventually get:

$$(S^0 = P) \wedge (\bigwedge_{i=1}^{l-1} S^i = 2S^{i-1}) \wedge$$
$$(\bigwedge_{i=0}^{l-1} M^i = Q_i S^i) \wedge$$
$$(R^0 = M^0) \wedge (\bigwedge_{i=1}^{l-1} R^i = R^{i-1} + M^i) \wedge$$
$$(R^{l-1} = N).$$

For n of bit length l, the resulting factorization formula has $4l^2 + 2l$ propositional variables and $19l^2 - 13l - 1$ clauses. One can optimize it to $l^2 + O(l)$ variables.

Our experiments with SAT solver zChaff (zChaff 2007) were carried out on a 2GHz, 2GB RAM IBM PC. The 32-bit RSA was broken in 15 seconds, 46-bit – 3 hours, 47-bit – 3 days wasn't enough, 48-bit – 33 hours, 49-bit – 3 days wasn't enough, 212-bit key chosen at random – 10 seconds (one of the factors was 11). It should be compared with the currently best 640-bit RSA-number factored out in the on-going RSA Factoring Challenge in November 2005 in an effort of 30 2.2GHz-Opteron-CPU years, over five months of calendar time. See (RSA labs 2007).

Taken together, our experimental results show that breaking RSA is unattainable by this method (without any modifications). But one can use the rsasat formula to test what can be done on the computers available today, as an interesting benchmark for performance of the computers and of the SAT solvers.

2 SHA-1 and SAT

SHA-1 is a widely used Secure Hash Algorithm, version 1. See (Menezes, van Oorschot & Vanstone 2001). It receives as input a plain message text in the form of a sequence M of bits of arbitrary length (in our experiments of bit length less than 447) and outputs a hashed sequence h of 160 bits: SHA1(m) = h. In the same notation as in the previous section, we obtained a propositional formula $sha1(M) = H$, with nearly 55 thousand propositional variables and nearly 235 thousand clauses. One can experiment with it in many ways: valuate M and run your favorite SAT solver to find H; valuate H (and the length of plain text — let your solver try to find M; valuate H and 'negate' in any way the original plain text hashed to H beforehand — let your solver try to find a collision (i.e., another plain text with the same hash H) or to prove that no collision of a given length exists.

Our experimental results can be be summarized as follows. Valuated plain text – the solver finds its hash in max. 2 seconds. Valuated hash of a 2-letter message — the solver finds the message in 1 hour. Valuated hash of a 3-letter message — the solver finds the message for the first 22 rounds of SHA1 in 1 hour. Valuated hash of a 4-letter message — the solver finds the message for 19-round SHA1 in 5 hours. Valuated hash of a 5-letter message — the solver finds the message for 19-round SHA1 in 3 hours. No big success here can be interpreted as confirming the required strength of the SHA-1 algorithm. But, first of all it means that straightforward use of the SAT solvers would not break SHA-1.

(Mironov & Zhang 2007) showed an interesting application of zChaff supporting some kind of a not automatic attempt to attack SHA-1.

3 Orthogonal matrices and SAT

In this section we report some encouraging experimental results obtained with SAT on certain simple computational task in elementary algebra. The task is to calculate the number of all the orthogonal square matrices of dimension n over the two-element Galois field F_2. The matrix A of dimension n is called *orthogonal* iff $A \cdot A^T = I_n$, where I_n is the identity matrix and A^T is the transponed matrix. A high level motivation for it came to us a couple of months ago from a friendly algebraist and will be given in some detail in the full version of this paper.

The table below shows the comparison of performance of one of the best SAT solvers — MiniSat v1.14 (Eén & Sörensson 2007) — with that of an algebraic program written by us. Both programs *de facto* do a brute force search through all the n^2 combinations of 0-1 entries and check orthogonality. Due to the exponential complexity 2^{n^2} of the problem the experiments were carried out for the dimensions $n \leq 7$ only. For $n = 7$ we discontinued the algebraic program after 12 hours. Table 1 shows how many orthogonal matrices it had found so far. Table 2 shows the complexity of our propositional formula. The experiments were carried out on an IBM PC Intel Pentium IV 3.2 GHz, 1024 MB RAM, with Linux operation system. The algorithms were implemented in C++.

	algebraic program		SAT	
n	secs	output	secs	output
3	< 1	6	< 1	6
4	< 1	48	< 1	48
5	2.172	720	< 1	720
6	2 482.910	23 040	3.873	23 040
7	43 200.000*	4 896	2247.058	1 451 520

Table 1: SAT performance on a simple algebraic task

n	variables	clauses
3	30	81
4	70	203
5	135	405
6	231	721
7	364	1141

Table 2: Complexity of our propositional formula

4 Conclusion and future work

The experimental results which we have presented in this paper and in (Srebrny & Stępień 2007) seem encouraging to consider SAT as a powerful programming environment for some tasks in linear algebra and cryptography. Probably in our experiments the SAT solvers used brute force search. The solvers have neither specific structural nor number theoretic info about the particular formulae on which we ran them. A question arises how to specialize, optimize, dedicate a SAT solver to one of the tasks we have considered in this paper. A SAT solver dedicated to a single algebraic or cryptanalytic problem? Ideas: grouping clauses, sorting clauses and variables, some specific heuristics, parallelization.

References

Davis, M. and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *Journal of the ACM* 7(1):201–215

Davis, M.; Logemann, G. and Loveland, D.W. 1962. A Machine Program for Theorem Proving. *Communications of the ACM* 5(7):394-397.

Eén, N. and Sörensson, N. 2005. *MiniSat v1.14*. http://www.cs.chalmers.se/∼een.

Menezes, A. J.; van Oorschot, P. C. and Vanstone, S. A. 2001. *Handbook of Applied Cryptography*. CRC Press.

Mironov, I., and Zhang, L. 2006. Applications of SAT Solvers to Cryptanalysis of Hash Functions. In *Nineth International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*. Seattle, Washington.

RSA labs. 2007. http://www.rsa.com/rsalabs/.

Srebrny, M., and Stępień, L. 2007. A propositional programming environment for linear algebra. *Fundamenta Informaticae* 81:325–345.

zChaff. 2007. http://www.princeton.edu/∼chaff/zchaff.html.