Information Processing Letters 113 (2013) 392–397

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A SAT-based preimage analysis of reduced KECCAK hash functions

Paweł Morawiecki [a,b,*,1], Marian Srebrny [a,b]

[a] Kielce University of Commerce, ul. Peryferyjna 15, 25-562 Kielce, Poland
[b] Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland

ARTICLE INFO

Article history:
Received 22 December 2011
Received in revised form 15 November 2012
Accepted 8 March 2013
Available online 20 March 2013
Communicated by A. Tarlecki

Keywords:
Cryptography
Hash functions
KECCAK
Algebraic cryptanalysis
Logical cryptanalysis
SAT solvers

ABSTRACT

In this paper, we present a preimage attack on reduced versions of KECCAK hash functions. We use our recently developed toolkit CryptLogVer for generating the conjunctive normal form, CNF, which is passed to the SAT solver PrecoSAT. We found preimages for some reduced versions of the function and showed that full KECCAK function has a comfortable security margin against this kind of attack.

1. Introduction

Cryptographic hash functions have been employed in a wide variety of security applications and protocols; like password storing/verification, document and file checksums, web certificates, and e-signatures. A hash function takes an arbitrarily long message and returns a fixed-size bit string hash value such that an accidental or intentional change to the message will change the hash value. In this paper we focus on the most significant property that a good hash function should have, called preimage resistance, that it is infeasible to find a message that has a given hash (while it is easy to compute the hash value for any given message).

The most widely used hash function is SHA-1. In [1] security flaws were identified in SHA-1, meaning that SHA-1 does not stand the expected 2^{80} collision resistance, and indicating that a stronger hash function would be desirable. In 2007, the US National Institute of Standards and Technology (NIST) announced a public contest aiming at the selection of a new standard for a cryptographic hash function. In October 2012, after 5 years of intensive competition, the winner has been selected. The new SHA-3 standard will be KECCAK hash function.

KECCAK is a family of cryptographic hash functions selected for the final round of SHA-3 contest. The security of a publicly known cryptographic algorithm, such as KECCAK, is accepted if there has been no known successful attack on it. Often some partial trust is additionally based on some good statistical properties and reported failure of breaking attempts with some known methods, like differential or linear cryptanalysis. The recent hash function MD-6 [2] has also been tested, among other methods, with logical (SAT-based) analysis.

* Corresponding author at: Kielce University of Commerce, ul. Peryferyjna 15, 25-562 Kielce, Poland.
E-mail address: pawelm@wsh-kielce.edu.pl (P. Morawiecki).

[1] The research was cofounded by the European Union from resources of the European Social Fund, Project PO KL Information technologies: Research and their interdisciplinary applications, Agreement UDA-POKL.04.01.01-00-051/10-00.

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2013.03.004

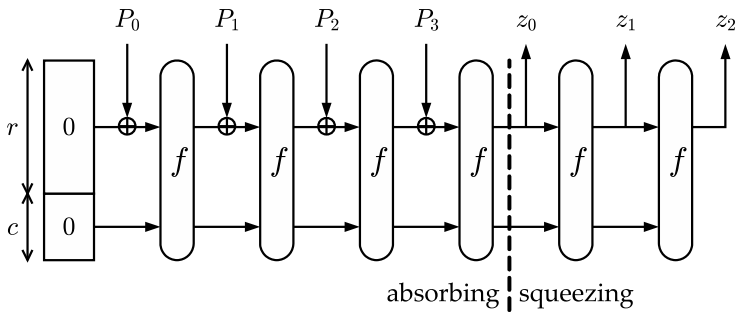

Fig. 1. Sponge construction [8].

SAT solvers accept formulas encoded in Conjunctive Normal Form (CNF) into which many decision problems can be translated. Modern SAT solvers use highly tuned algorithms and data structures to quickly find a solution to a given problem coded in this very simple form. 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 solver to find a solution. The propositional encoding formula can be thought of as a declarative program. One can treat the propositional calculus and the SAT solvers as a powerful programming environment that makes it possible to create and to run the propositional declarative programs for solving the encoded tasks. 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 testing 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 [3]. 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 have been developed over the last two decades as highly optimized versions of the DPLL procedure [4].

In this paper, we present a preimage attack on reduced versions of Keccak hash functions. We also find preimages for reduced versions of Keccak from a given hash and a part of a message (padding bits). Menezes et al. [5] call it a partial preimage attack. In particular we find a preimage for the 3-round Keccak with 40 unknown message bits and this attack is more than 128 times faster than the exhaustive search. To mount the attacks we use our recently developed toolkit CryptLogVer for generating a CNF which is passed to SAT solver PrecoSAT [6].

The paper is organized as follows. In Section 2 we present a short description of Keccak family functions. Then a CNF generation method is given. In Section 3 the detailed attack scenario is described, followed by our experimental results. Comparison to related work is indicated in Section 7. The last section consists of some conclusion and future research.

2. Keccak – a brief description

In this section we bring up only a brief description of Keccak to the extent necessary for understanding the attack described in this paper. For a complete specification we refer the interested reader to [7].

Keccak is a family of hash functions which makes use of the sponge construction. Fig. 1 shows the construction. It has two main parameters r and c which are called bitrate and capacity, respectively. The sum of those two makes the state size which Keccak operates on. Different values for bitrate and capacity give trade-off between speed and security. The higher bitrate gives the faster function but less secure. Keccak proceeds in two phases. In the first phase (absorbing) the r-bit input message blocks are xored into the first r bits of the state, interleaved with applications of the function f (called Keccak-f in the specification). This phase is finished when all message blocks are processed. In the second phase (squeezing) the first r bits of the state are returned as hash bits, interleaved with applications of the function f. The phase is finished when the desired length of hash is produced.

For the SHA-3 proposal, the default values for Keccak are $r = 1024$, $c = 576$ which gives the 1600-bit state. However, Keccak can also operate on smaller states (25, 50, 100, 200, 400 and 800-bit state). Keccak-f[b] is a permutation defined on states of bits of width $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ arranged in a three-dimensional array. The choice of a sequence of steps of these permutations is very different from that in SHA-1 and SHA-2, and in AES. These permutations are iterated through almost identical n_r rounds. The number of rounds n_r depends on the permutation width. All the details are concisely given on two pages 8–10 of [7]. For a default 1600-bit state there are 24 rounds. In the experiments we used reduced versions with smaller number of rounds.

3. CNF formula generation

One of the key steps in attacking cryptographic primitives with SAT solvers is a CNF formula generation. Such a formula completely describes the primitive (or a segment of the primitive) which is the target of the attack. Generating it is a non-trivial task and usually is very laborious. There are many ways to obtain the final CNF and the output results differ in the number of clauses, the average size of clauses and the number of literals. Recently we have developed a new toolkit called CryptLogVer which greatly simplifies the creation of a CNF. Here we describe only the main concepts. The detailed description of CryptLogVer will be published in a separate paper.

Usually a cryptanalyst needs to put a considerable effort into creating the final CNF. It involves writing a separate program dedicated only to a cryptographic primitive under

consideration. To make it efficient, some minimizing algorithms (Karnaugh maps, Quine-McCluskey algorithm or Espresso algorithm) have to be used [9]. These are implemented in the program, or the intermediate results are sent to an external tool (e.g., Espresso minimizer) and then the minimized form is sent back to the main program. Implementing all of these procedures requires a good deal of programming skills, some knowledge of logic synthesis algorithms and careful insight into the details of the primitive's operation. As a result, obtaining a CNF might become the most tedious and error-prone part of any attack. It could be especially discouraging for researchers who start their work from scratch and do not want to spend too much time on writing thousands lines of code.

To avoid those disadvantages we have recently proposed a new toolkit consisting basically of two applications. First of them is Quartus II – a software tool released by Altera for analysis and synthesis of HDL (Hardware Description Language) designs, which enables the developers to compile their designs and configure the target devices (usually FPGAs). We use a free-of-charge version Quartus II Web Edition which provides all the features that we need. The second application, written by us, converts boolean equations (generated by Quartus) to a CNF encoded in DIMACS format (standard format for today's SAT solvers). The complete process of a CNF generation includes the following steps:

1. Code the target cryptographic primitive in HDL;
2. Compile and synthesize the code in Quartus;
3. Generate boolean equations using Quartus inbuilt tool;
4. Convert generated equations to a CNF by a separate application.

Steps 2, 3, 4 are done automatically. The only effort a researcher has to put is to write a code in HDL. Normally programming and 'thinking' in HDL is a bit different from typical high-level languages like Java or C. However it is not the case here. For our needs, programming in HDL looks exactly the same as it would be done in high-level languages. There is no need to care about typical HDL specific issues like proper expressing of concurrency or clocking. It is because we are not going to implement anything in an FPGA device. All we need is to obtain a system of boolean equations which completely describes the primitive we wish to attack.

We are aware that KeccakTools [7] supports the generation of equations, but not in a CNF. This could replace the use of HDL and Quartus to generate the boolean equations in our analysis of Keccak. However, the equations generated by KeccakTools consist of many 'long XOR' equations which would produce an exponential number of clauses (exponential in the number of variables) when converted to an equivalent CNF. To avoid that exponential blowup, one can generate an equisatisfiable CNF with new variables introduced and the equations 'cut' into shorter ones. Thus, an additional processing is essential to make those equations useful for SAT solvers. Besides, our CryptLogVer's in-built generation of equations might turn out useful for a uniform comparison of the kind of analysis of various hash algorithms. Finally, equations generated by KeccakTools are in an Algebraic Normal Form (ANF) and cannot be directly used with a SAT solver. How to efficiently convert an ANF to a CNF is far from trivial and we believe it could be a good point for further research.

4. Description of the attack

We carried out the preimage attack, i.e., for a given hash value h, we tried to find a message m such that $h = f(m)$. We also tried a partial preimage attack where an attacker knows a message length (which was set up to 60 bits). So she can fix the padding bits and the make the resulting formula easier for a SAT solver. Our attacks were mainly focused on Keccak[1024, 576] – a default variant of Keccak hash function with bitrate $r = 1024$ and capacity $c = 576$. Since the full 24-round variant is too hard for any currently realistic SAT-based attack, we experimented with round-reduced variants.

The attack scheme can be divided into three steps:

1. Generate CNF by CryptLogVer toolkit;
2. Fix the output bits (hash) and a part of input bits (the padding bits);
3. Run PrecoSAT on the created CNF.

In case of the preimage attack only the hash bits are fixed in a SAT formula, as an attacker knows nothing about the message. But in the partial preimage attack, the attacker knows the length of the hashed message. (The message from which the hash was calculated.) Thus in that case the padding bits can be fixed. Keccak has a simple padding rule, it adds '1' right after the message, then a certain number of '0s' to fill the whole block and finally ends the block with '1'.

When a message searched for is supposed to be short enough (the number of message bits and required padding bits is less than or equal to bitrate r), it fits into one block P_i. (See Fig. 1.) Consequently, there is only one invocation of Keccak-f function and the CNF used in the attacks can encode only Keccak-f. In general, function f is crucial for the security of the sponge construction and its strength in many cases comes down to the CICO problem (defined by Keccak designers, Section 5.2.4 in Keccak main document [7]). The preimage attacks presented in this paper can be also treated as an attempt of solving the CICO problem. It needs to be emphasized that the claimed security for the CICO problem is higher than for Keccak function. The point of reference for our experiments is the claimed security for Keccak function which is $2^{c/2}$.

5. The experimental results

We mounted the attacks on Keccak[1024, 576] and the experiments were carried out on Intel Core 2 Duo 2.0 GHz. For a SAT solver we chose PrecoSAT. In our preliminary experiments with SHA-1 hash function we tested some other SAT solvers including CryptoMiniSat2. The results were very similar thus we have ruled out the possibility that certain solver somehow favours our encoding and translation to CNF. In Appendix A we give some results obtained

Table 1
Preimage and partial preimage attacks on Keccak[1024, 576].

Input parameters			Attack times [secs]		
Number of rounds	Message size [bits]	Hash size [bits]	SAT solver attack		Exhaustive search[a]
			Average time	Std. deviation	
3	24	1024	$2^{-0,6}$	$2^{-3,6}$	$2^{0,7}$
3	32	1024	$2^{3,6}$	$2^{2,8}$	$2^{8,7}$
3	40	1024	$2^{8,9}$	$2^{7,9}$	$2^{16,7}$
2	60	256	$2^{3,3}$	$2^{2,9}$	$2^{36,1}$
2	–	256	$2^{6,0}$	$2^{6,6}$	$2^{232,17}$

[a] We calculate this value in the following way. On the Core 2 Duo, the speed-optimized implementation of Keccak-f[1600] takes about 200 cycles for 3 rounds. With 2.0 GHz CPU, a time unit is therefore $200/(2.0 * 10^9) = 10^{-7}$ second. So 1 second of CPU time is about $10^7 \approx 2^{23.25}$ evaluations of 3-round Keccak-f[1600]. Finally, the exhaustive search for 24-bit message would take $2^{24-23.25} \approx 2^{0.7}$ seconds.

with zChaff solver, again the results were consistent with PrecoSat's ones.

Table 1 summarizes the results of attacks. For partial preimage attacks a message size is known (second column in Table 1). Each experiment (attack) reported in Table 1 was carried out on a sample of 50 random hash values (in case of the preimage attack) and 50 random messages (in case of the partial preimage attack). For the preimage attack we reached 2 rounds. Additionally, for 2-round variant we found second preimages and collisions. We forced the SAT solver to provide a second preimage/collision by inverting one bit of the preimage, i.e., fixing one message bit (randomly chosen) to the opposite value than in the first preimage. The other message bits we left as unknowns. This way the SAT solver finds different solution than the first preimage previously found and we get a second preimage/collision. We managed to find partial preimages for 2- and 3-round Keccak[1024, 576] with up to 60 unknown bits of a message. The times we obtained are consistent, most groups of the experiments have low standard deviation (beyond 10 seconds). Even if we take the worst-case results they are far better than exhaustive search times shown in Table 1.

We experimented with 4 and 5 rounds, however the SAT solver could not provide any solutions with the 48-hour time limit. The attack against variants with a smaller state size (200- and 50-bit state) turned out to be worse than the exhaustive search.

The exhaustive search time was calculated with reference to C speed-optimized implementation provided by Keccak designers [7]. The exhaustive search time is the time needed for checking all the combinations of the unknown message bits (partial preimage attack) or checking 2^l messages where l is a hash size. (After checking 2^l messages it is expected that one of the message is the preimage.)

6. A note on two other SHA-3 candidates

We have managed to estimate the complexity of a boolean formula in its conjunctive normal form coding the hash function Grøstl. Grøstl uses the AES S-box and [10] announces that the simplest version of Grøstl (with the 256-bit hash values) calls AES S-box 1280 times. AES S-box has been coded by our CryptLogVer toolkit as a formula with about 4800 clauses and 900 variables. So, a straightforward calculation gives at least $1280 * 4800 = 6.144 \times 10^6$ clauses in total. Hence, no SAT-based attack can be feasible (without engaging extraordinary resources such as a cluster of supercomputers), even for reduced versions. From this perspective Grøstl seems to be very strong. For comparison, the AES-128 standard calls S-box 'only' 160 times.

We also have looked closer at Bernstein's CubeHash function [11], encouraged by its simplicity. We have obtained the following estimate of the CNF formula size. For the version originally submitted to the SHA-3 contest, its CNF would have about 1.76×10^6 clauses and 2.7×10^5 variables.

For the sake of comparison, we carried out our SAT-based preimage attack on reduced versions of SHA-1. For full SHA-1, the CNF formula encoding the function has 1.8×10^5 clauses and 3.1×10^4 variables, while full Keccak has 7.75×10^5 clauses and 1.81×10^5 variables. It could be already a sign that Keccak is much stronger, as the CNF formula is over 4 times bigger. We found a short preimage for 27-round SHA-1 (out of its full 80 rounds), but only for the first 3 rounds out of 24 for Keccak.

7. Related work

The designers of Keccak made some experiments to solve the CICO problem. They used SAGE computer algebra software and were able to solve the problems with 12 unknown input bits, up to 8 rounds and with Keccak-f state widths from 25 to 400. As the number of unknown input bits grows, this method quickly becomes infeasible [7]. Courtois and Bard [12] showed that SAT solvers can be a better option for solving cryptographic problems (often comprising of large systems of equations) than computer algebra systems (such as SAGE, MAGMA or Singular) due to their much lower memory requirements.

In [13] the triangulation algorithm was used to solve the CICO problem. They reached 3 rounds for Keccak-f[1600]. They fixed only a few bits which was enough to show non-randomness of the function but did not lead to any real attack. For smaller state sizes of Keccak-f they did not pass 3 rounds.

In [14] partial preimages of 4-round Keccak are found by applying the cube attack. Bernstein [15] outlines a second preimage attack on 8-round Keccak. However it is only a theoretical attack with a complexity of over 2^{500} compression function calls. With the aid of differential analysis, Naya-Plasencia et al. mounted the preimage and collision attacks on 2-round Keccak [16]. The most

successful collision attack was given in [17] where 4-round collisions were presented.

The recent hash function MD-6 of Rivest et al. [2] was also tested with logical (SAT-based) analysis, among other methods. They found collisions only for much reduced versions of the function, with 11 rounds as the best result. They observed that after the first 7 rounds the attack running time grows superexponentially in the number of rounds. Therefore, it makes this method inefficient against the full MD-6 algorithm.

The first connection between SAT and crypto dates back to [18], where a suggestion appeared to use cryptoformulae as hard benchmarks for propositional satisfiability checkers. The first application of SAT solvers in cryptanalysis was due to Massacci et al. [19]. They ran a SAT solver on DES key search, and then also for faking an RSA signature for a given message by finding the e-th roots of a (digitalized) message m modulo n, in [20]. They called it logical cryptanalysis. Courtois and Pieprzyk [21] presented an approach to code in SAT their algebraic cryptanalysis with some gigantic systems of low degree equations designed as potential procedures for breaking some ciphers. [22] proposed enhancing a SAT solver with some special-purpose algebraic procedures, such as Gaussian elimination. Mironov and Zhang [23] showed an application of a SAT solver supporting a non-automatic part of the attack [1] on SHA-1.

8. Conclusion

We have carried out the SAT-based preimage attack on reduced Keccak hash functions. We have found preimages only for much reduced versions of Keccak; e.g., a preimage for the 3-round Keccak with 40 unknown message bits. We also have found collisions for the 2-round variant. When considering practical (experimentally verified) preimage attacks, our findings are among state-of-the-art results, according to the reports from SHA-3 zoo website [24].

The results suggest the strength of the function against this kind of attack. It has a comfortable security margin of over 20 rounds to resist the SAT-based attack. Also Grøstl and CubeHash seem to be very strong against SAT-based cryptanalysis. For future research one can try extrapolating our results for the full Keccak function. Such a technique was used in [22]. SAT-based analysis of the other SHA-3 candidates also might be interesting.

Acknowledgements

The authors gratefully acknowledge Mate Soos for his cooperation in carrying out some of the experiments on Grid'5000 [25]. The authors also thank Gilles Van Assche of the Keccak team, as well as Mateusz Srebrny, Rene Peralta, Stanislaw Radziszowski, and Praveen Gauravaram (Technical University of Denmark) for their valuable contributions and remarks at various stages of development of our SAT-based cryptanalytic technique and on earlier versions of this paper. Finally, authors thank the anonymous reviewers for many helpful comments and corrections.

Table 2
Best results of both methods for factorization problem and partial preimage attack on SHA-1 hash function.

Method	Problem solved
Hand-written	50-bit RSA number factorization
CryptLogVer	60-bit RSA number factorization
Hand-written	22-round SHA-1, partial preimage attack
CryptLogVer	27-round SHA-1, partial preimage attack

Appendix A

We present here a short comparison with a 'hand-written' SAT instances of factorization problem (which RSA cryptosystem is based on) and a hash function SHA-1 [26]. This work could be a point of reference for our automated approach and it turns out that CryptLogVer is competitive or even better than the 'hand-written', dedicated solution. Table 2 shows the best results for each method. The experiments were carried out on a PC 2.0 GHz CPU with the time limit set to 12 hours. PrecoSAT solver was used for factorization problem and zChaff solver for attacking SHA-1. The number of unknown message bits in a partial preimage attack was set to 40.

In both tasks CryptLogVer had better results. After a more careful examination of 'hand-written' formulas it turned out that they did not used a minimized equation for a full adder. CryptLogVer toolkit takes advantage of minimized form of that equation and consequently makes the SAT formulas more compact. For the cryptographic problems we examined, smaller formulas were easier to handle by the solvers.

A detailed and robust comparison with other encoding techniques (such as efficient Tseitin encoding or And-Inverter Graphs with minimization) is out of the scope of this paper. However, it could be a good point for further research and consideration of incorporating other techniques into our toolkit.

References

[1] X. Wang, Y.L. Yin, H. Yu, Finding collisions in the full SHA-1, in: Crypto, in: Lecture Notes in Computer Science, vol. 3621, Springer, Berlin, Heidelberg, 2005, pp. 17–36.
[2] R. Rivest, et al., The MD6 hash function, http://groups.csail.mit.edu/cis/md6/.
[3] S.A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, ACM, New York, NY, USA, 1971, pp. 151–158.
[4] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Communications of the ACM 7 (5) (1962) 394–397.
[5] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 2001.
[6] PrecoSAT, http://fmv.jku.at/precosat/.
[7] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Keccak sponge function family main document, http://keccak.noekeon.org/Keccak-main-2.1.pdf.
[8] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Cryptographic sponges, http://sponge.noekeon.org.
[9] P.K. Lala, Principles of Modern Digital Design, Wiley-Interscience, 2007.
[10] P. Gauravaram, L.R. Knudsen, K. Matusiewicz, F. Mendel, Ch. Rechberger, M. Schläffer, S.S. Thomsen, Groestl – SHA3 candidate, original specification, October 31, 2008, http://www.groestl.info/Groestl-0.pdf.
[11] D.J. Bernstein, CubeHash, http://cubehash.cr.yp.to.

[12] N.T. Courtois, G.V. Bard, Algebraic cryptanalysis of the data encryption standard, Cryptology ePrint Archive, Report 2006/402, http://eprint.iacr.org/2006/402, 2006.

[13] J.-P. Aumasson, D. Khovratovich, First analysis of Keccak, http://131002.net/data/papers/AK09.pdf.

[14] J. Lathrop, Cube attacks on cryptographic hash functions, Master's thesis, Rochester Institute of Technology, 2009.

[15] D.J. Bernstein, Second preimages for 6 (7? (8??)) rounds of Keccak?, http://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt.

[16] M. Naya-Plasencia, A. Röck, W. Meier, Practical analysis of reduced-round keccak, in: D. Bernstein, S. Chatterjee (Eds.), Progress in Cryptology – INDOCRYPT 2011, in: Lecture Notes in Computer Science, vol. 7107, Springer, Berlin, Heidelberg, 2011, pp. 236–254.

[17] I. Dinur, O. Dunkelman, A. Shamir, New attacks on Keccak-224 and Keccak-256, in: A. Canteaut (Ed.), Fast Software Encryption, in: Lecture Notes in Computer Science, vol. 7549, Springer, Berlin, Heidelberg, 2012, pp. 442–461, http://dx.doi.org/10.1007/978-3-642-34047-5_25.

[18] S.A. Cook, D.G. Mitchell, Finding Hard Instances of the Satisfiability Problem: A Survey, American Mathematical Society, 1997, pp. 1–17.

[19] F. Massacci, Using Walk-SAT and Rel-SAT for cryptographic key search, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1999, pp. 290–295.

[20] C. Fiorini, E. Martinelli, F. Massacci, How to fake an RSA signature by encoding modular root finding as a SAT problem, Discrete Applied Mathematics 130 (2003) 101–127.

[21] N. Courtois, J. Pieprzyk, Cryptanalysis of block ciphers with overdefined systems of equations, in: Y. Zheng (Ed.), Advances in Cryptology – ASIACRYPT 2002, in: Lecture Notes in Computer Science, vol. 2501, Springer, Berlin, Heidelberg, 2002, pp. 267–287.

[22] M. Soos, K. Nohl, C. Castelluccia, Extending SAT solvers to cryptographic problems, 2009, pp. 244–257.

[23] I. Mironov, L. Zhang, Applications of SAT solvers to cryptanalysis of hash functions, in: A. Biere, C. Gomes (Eds.), Theory and Applications of Satisfiability Testing – SAT 2006, in: Lecture Notes in Computer Science, vol. 4121, Springer, Berlin, Heidelberg, 2006, pp. 102–115.

[24] The SHA-3 zoo, http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo.

[25] Grid'5000, http://www.grid5000.fr.

[26] M. Srebrny, M. Srebrny, L. Stępień, SAT as a programming environment for linear algebra and cryptanalysis, in: Workshop on Concurrency, Specification, and Programming, 2009.