ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1594

Encoding Cryptographic Functions to SAT Using TRANSALG System

Ilya Otpuschennikov[1], Alexander Semenov[1], Irina Gribanova[1], Oleg Zaikin[1], Stepan Kochemazov[1]

Abstract. In this paper we propose the technology for constructing propositional encodings of discrete functions. It is aimed at solving inversion problems of considered functions using state-of-the-art SAT solvers. We implemented this technology in the form of the software system called TRANSALG, and used it to construct SAT encodings for a number of cryptanalysis problems. By applying SAT solvers to these encodings we managed to invert several cryptographic functions. In particular, we used the SAT encodings produced by TRANSALG to construct the family of two-block MD5 collisions in which the first 10 bytes are zeros. In addition to that we used TRANSALG encoding for the widely known A5/1 keystream generator to solve several dozen of its cryptanalysis instances in a distributed computing environment. Also in the present paper we compare the functionality of TRANSALG with that of similar software systems.

1 FOUNDATIONS OF SAT-BASED CRYPTANALYSIS

By $\{0,1\}^*$ we denote the set of all binary words of an arbitrary finite length. By discrete functions we mean arbitrary (possibly, partial) functions of the kind: $f : \{0,1\}^* \to \{0,1\}^*$. Hereinafter we consider only total computable discrete functions. In other words we assume that f is specified by some program (algorithm) A_f, that has finite runtime on each word from $\{0,1\}^*$. The program A_f specifies a family of functions of the kind $f_n : \{0,1\}^n \to \{0,1\}^*$, $n \in N_1$. Below we consider the problem of inversion of an arbitrary function f_n as follows: based on the known $y \in Range f_n$ and the known algorithm A_f, find such $x \in \{0,1\}^n$ that $f(x) = y$. Many cryptanalysis problems can be formulated as inversion problems of discrete functions. For example, suppose that given a secret key $x \in \{0,1\}^n$, f_n generates a pseudorandom sequence (in general, of an arbitrary length), that is later used to cipher some plaintext via bit-wise XOR. Such a sequence is called a *keystream*. Knowing some fragment of plaintext lets us know the corresponding fragment of keystream, i.e. some word y for which we can consider the problem of finding such $x \in \{0,1\}^n$, that $f_n(x) = y$. Regarding cryptographic keystream generators this corresponds to the so called known plaintext attack. Let us give another example. Total functions of the kind $f : \{0,1\}^* \to \{0,1\}^c$, where c is some constant, are called *hash functions*. If n is the length of the input message and $n > c$, then there exist such $x_1, x_2, x_1 \neq x_2$, that $f_n(x_1) = f_n(x_2)$. Such a pair x_1, x_2 is called a collision of a hash function f. A cryptographic hash function is considered compromised if one is able to find collisions of that function in reasonable time.

SAT-based cryptanalysis is a quite new direction in cryptanalysis and its basic paradigm suggests that specific inversion problem of a considered function is reduced to a SAT instance. Recall, that Boolean Satisfiability Problem (SAT) consists in the following: for an arbitrary Boolean formula to decide whether it is satisfiable or not. Using the ideas by S.A. Cook [2] and J.C. King [6] it can be shown that for an arbitrary function f_n from the class described above, the corresponding inversion problem can be effectively reduced to SAT for some satisfiable CNF.

The main result of our paper is the TRANSALG software system designed specifically to construct SAT encodings for cryptographic functions and apply to constructed encodings state-of-the-art SAT solvers. It is based on the concept of symbolic execution [6]. The features of TRANSALG translation procedures were described in [9]. In the present paper we show how TRANSALG can be applied to solving inversion problems of several cryptographic functions and compare its effectiveness with that of similar systems: GRAIN OF SALT [11], URSA [5], and CRYPTOL+SAW [4].

Let us now point out key differences between TRANSALG and aforementioned systems. The distinctive feature of TRANSALG is that it can construct and explicitly output the so-called *template CNF* $C(f_n)$. When it constructs $C(f_n)$ it employs the concept of symbolic execution of program A_f fully reflecting this process in the memory of abstract computing machine. As a result, in a template CNF $C(f_n)$ all elementary operations with the memory of abstract machine are represented in the form of Boolean equations over sets of Boolean variables. TRANSALG makes it possible to work with these variables directly, thus providing a number of useful features for cryptanalysis. In particular, we can quickly generate families of cryptographic instances: to make one SAT instance for function inversion it is sufficient to add to a template CNF unit clauses encoding the corresponding output. That is why template CNFs are very handy when one uses partitioning strategy to solve some hard SAT instance in a distributed computing environment. Also TRANSALG can identify variables corresponding to inputs and outputs of considered function, so external tools can be used to check correctness of SAT encodings and to analyze the results of solving SAT. In particular, thanks to this we can use any SAT solvers and preprocessors. TRANSALG allows to monitor the values of program variables inside program A_f at any step of computing, and, therefore, to assert any conditions on these variables. For example, thanks to this it is easy to write in a program A_f the conditions specifying the differential path for finding collisions of cryptographic hash functions. In other considered systems (URSA, Cryptol) there arise significant difficulties when writing such conditions. Finally, let us note that the connec-

[1] Matrosov Institute for System Dynamics and Control Theory of SB RAS, Irkutsk, Russia, email: otilya@yandex.ru, biclop.rambler@yandex.ru, the42dimension@gmail.com, zaikin.icc@gmail.com, veinamond@gmail.com

tion between the structure of CNF $C(f_n)$ and an original algorithm, reflected by TRANSALG, can play an important role in implementation of several cryptographic attacks (such as guess-and-determine attacks [1]) in parallel.

2 SAT-BASED CRYPTANALYSIS OF SEVERAL CRYPTOGRAPHIC SYSTEMS USING TRANSALG

In the first series of experiments we considered SAT-based cryptanalysis of the Bivium, Trivium and Grain keystream generators. Note that similar problems were studied earlier in [3, 11, 12]. In accordance with these papers we considered the inversion problems for the following functions: $f^{Bivium} : \{0,1\}^{177} \to \{0,1\}^{200}$, $f^{Grain} : \{0,1\}^{160} \to \{0,1\}^{160}$, $f^{Trivium} : \{0,1\}^{288} \to \{0,1\}^{300}$.

In our experiments we assumed that a number of bits from the secret key are known. To these bits we will below refer as *guessing bits* [1]. In other words, assume that we consider the inversion problem of function $f_n : \{0,1\}^n \to \{0,1\}^m$ in some point $y \in Range f_n$. Let $C(f_n)$ be the template CNF for f_n and let X^{in} be the set of variables encoding the input of f_n. Let us choose as the set of guessing bits some set X', $X' \subseteq X^{in}$. By *GeneratorK* we mean the SAT instances which encode cryptanalysis of the corresponding generator, modified by assigning values to variables from set X',$|X'| = K$. Essentially, *GeneratorK* means a series of SAT instances that differ in values of variables from X'. We considered such series of 100 instances each. On instances from each series we ran state-of-the-art CDCL SAT solvers that rated high in SAT competition 2014. In case of the encodings produced by URSA we were forced to use only the solvers CLASP and ARGOSAT embedded into this system. On average all considered tools constructed SAT encodings with more or less similar solving time (for each system we tracked the best results using all solvers). In terms of SAT instances solved within the time limit of one hour, TRANSALG lost to competition at most 6% on Bivium30 and Trivium142, but won at least 30% on Grain102 .

Earlier we applied TRANSALG to perform the SAT-based cryptanalysis of the widely known A5/1 keystream generator, that is still used to cipher GSM traffic in many countries. In detail the corresponding computational experiment was described in [10]. We managed to solve non-weakened cryptanalysis instances for this generator in a specially constructed distributed environment. Note, that it was possible to effectively parallelize this problem thanks to the functional features of TRANSALG outlined above.

The next cryptanalysis problem we considered was the problem of finding collisions of cryptographic hash functions MD4 and MD5. The first successful application of SAT-solvers to this problem was described in [8]. The authors of [8] note that to find one MD4 collision it took them about 10 minutes (500 seconds), while finding MD5 collisions proved to be more difficult. The SAT encodings for the corresponding problems constructed using TRANSALG system were much more compact than that from [8]. In our experiments in less than 500 seconds it was possible to generate 1000 MD4 collisions using one mainstream processor core (with the help of CRYPTOMINISAT SAT solver [12]). To find two-block MD5 collisions we employed PLINGELING and TREENGELING SAT solvers. With their help we found several dozens two-block MD5 collisions with first 10 zero bytes. Note, that we implemented a SAT variant of differential attack by X.Wang et al. [13]. With this purpose we added to template CNF additional constraints encoding the differential path. In the TRANSALG system thanks to its translation concept this step can be performed effectively, while in other systems it requires significant amount work to be implemented.

Finally we compared the effectiveness of SAT and SMT approaches to inversion of cryptographic functions. We performed SAT-based cryptanalysis of Geffe generator [7], where the mixing function was $majority(x_1, x_2, x_3)$. We considered the cryptanalysis problem in the following form: to find 96-bit secret key using the known keystream fragment of length 200 bits. We constructed a series of 100 instances of this kind. Each instance was considered both as SAT and as SMT. The SAT encodings were constructed using TRANSALG system and were solved using MINISAT solver. The SMT encoding were constructed using CRYPTOL+SAW and solved using SMT-solvers BOOLECTOR, YICES, CVC4 and Z3. In the considered series of experiments SMT approach lost to SAT in both the number of problems solved within 1 minute (83 vs 100) and in average time on solved instances (35 seconds vs 7).

The extended version of this paper can be found online [2].

ACKNOWLEDGEMENTS

The research was funded by Russian Science Foundation (project no 16-11-10046).

REFERENCES

[1] Gregory V. Bard, *Algebraic Cryptanalysis*, Springer Publishing Company, Incorporated, 1st edn., 2009.

[2] Stephen A. Cook, 'The complexity of theorem-proving procedures', in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pp. 151–158, (1971).

[3] Tobias Eibach, Enrico Pilz, and Gunnar Völkel, 'Attacking Bivium using SAT solvers', in *SAT*, eds., Hans Kleine Büning and Xishun Zhao, volume 4996 of *Lecture Notes in Computer Science*, pp. 63–76. Springer, (2008).

[4] Levent Erkök and John Matthews, 'High assurance programming in cryptol', in *Fifth Cyber Security and Information Intelligence Research Workshop, CSIIRW '09, Knoxville, TN, USA, April 13-15, 2009*, eds., Frederick T. Sheldon, Greg Peterson, Axel W. Krings, Robert K. Abercrombie, and Ali Mili, p. 60. ACM, (2009).

[5] Predrag Janicic, 'URSA: a System for Uniform Reduction to SAT', *Logical Methods in Computer Science*, **8**(3), 1–39, (2012).

[6] James C. King, 'Symbolic execution and program testing', *Commun. ACM*, **19**(7), 385–394, (July 1976).

[7] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot, *Handbook of Applied Cryptography*, CRC Press, Inc., Boca Raton, FL, USA, 1st edn., 1996.

[8] Ilya Mironov and Lintao Zhang, 'Applications of SAT solvers to cryptanalysis of hash functions', in *SAT*, eds., Armin Biere and Carla P. Gomes, volume 4121 of *Lecture Notes in Computer Science*, pp. 102–115. Springer, (2006).

[9] Ilya Otpuschemmikov, Alexander Semenov, and Stepan Kochemazov, 'Transalg: a tool for translating procedural descriptions of discrete functions to SAT', in *WCSE 2015-IPCE: Proceedings of The 5th International Workshop on Computer Science and Engineering: Information Processing and Control Engineering*, pp. 289–294, (2015).

[10] Alexander Semenov and Oleg Zaikin, 'Algorithm for finding partitionings of hard variants of boolean satisfiability problem with application to inversion of some cryptographic functions', *SpringerPlus*, **5**(1), 1–16, (2016).

[11] Mate Soos, 'Grain of Salt - an automated way to test stream ciphers through SAT solvers', in *Tools'10: Proceedings of the Workshop on Tools for Cryptanalysis*, pp. 131–144, (2010).

[12] Mate Soos, Karsten Nohl, and Claude Castelluccia, 'Extending SAT solvers to cryptographic problems', in *SAT*, ed., Oliver Kullmann, volume 5584 of *Lecture Notes in Computer Science*, pp. 244–257. Springer, (2009).

[13] Xiaoyun Wang and Hongbo Yu, 'How to break MD5 and other hash functions', in *Advances in Cryptology - EUROCRYPT 2005, Proceedings*, pp. 19–35, (2005).

[2] http://arxiv.org/abs/1607.00888