Solving weakened cryptanalysis problems for the Bivium cipher in the volunteer computing project SAT@home

Oleg Zaikin, Alexander Semenov and Ilya Otpuschennikov

Abstract—**In this paper, a SAT-based cryptanalysis of the Bivium stream cipher is considered. For encoding the initial cryptanalysis problem into SAT a special program Transalg was used. For an obtained SAT instance we use Monte Carlo method to search for a partitioning with good time estimation. Several weakened cryptanalysis instances of the Bivium generator were successfully solved in the volunteer computing project SAT@home using corresponding partitionings found on a computing cluster.**

Keywords—**Boolean satisfiability problem, Transalg, cryptanalysis, Bivium, volunteer computing, SAT@home.**

I. INTRODUCTION

Volunteer computing [1] is a type of distributed computing which uses computational resources of PCs of individuals called volunteers. Each volunteer computing project is designed to solve one or several hard problems. When PC is connected to the project, all the calculations are performed automatically and do not provide any inconvenience to a user since only idle resources of PC are used. Nowadays the most popular platform for organizing volunteer computing projects is BOINC [2] which is being developed in Berkeley since 2002. Today there are about 70 active volunteer projects, the majority of them are based on BOINC. A volunteer computing project consists of the following basic parts: server daemons, database, web site and client application. The daemons include work generator (generates tasks to be processed), validator (checks the correctness of the results received from volunteer's PCs) and assimilator (processes correct results). Client application should have versions for the widespread computing platforms. One of the attractive features of volunteer computing is its low cost – to maintain a project one needs only a dedicated server working 24/7. Main difficulties here are software development and database administration. In addition, it is crucial to provide the feedback to volunteers using the web site of the project and special forums. Another attractive feature of this type of computing is that volunteer project can solve some particular hard problem for months or even years with good average performance.

Wide class of problems from modern computer science can be effectively reduced to Boolean satisfiability problem (SAT) [3]. SAT problems are usually considered as the problems of search for solutions of Boolean equations in the form of CNF=1, where CNF is a conjunctive normal form. There are many works in which various combinatorial problems are reduced to SAT and solved in this form. For example, such problems can be found in areas of verification, cryptography, combinatorics and bioinformatics. Usually if the cryptanalysis is considered as a SAT problem then it is called a *SAT-based cryptanalysis*. In this case to find a secret key it is sufficient to find a solution of corresponding satisfiable SAT instance. For solving corresponding SAT instance one usually needs large computational resources. A volunteer computing project can provide such resources. Below we present a brief outline of our paper. In the second section, we describe how we utilize the TRANSALG tool for reduction of the cryptanalysis problem of the Bivium cipher to SAT. In the third section experimental results on solving weakened cryptanalysis problems for the Bivium generator in the volunteer computing project SAT@home are presented.

II. PROPOSITIONAL ENCODING OF THE CRYPTANALYSIS PROBLEM FOR BIVIUM

The Bivium cipher [4] uses two shift registers of a special kind (see Fig. 1). The first register contains 93 cells and the second contains 84 cells. To initialize the cipher, a secret key of length 80 bit is put to the first register, and a fixed (known) initialization vector of length 80 bit is put to the second register. All remaining cells are filled with zeros. An initialization phase consists of 708 rounds during which keystream output is not released.

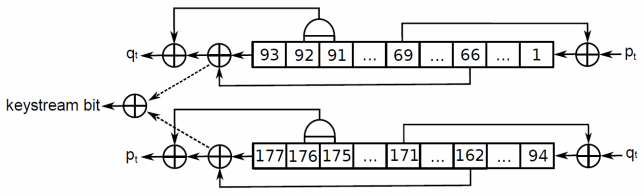

Fig. 1. Scheme of the Bivium cipher

In accordance with [5]–[7] we considered cryptanalysis problem for Bivium in the following formulation. Based on

Manuscript received October 13, 2015. This work was supported in part by the Russian Foundation for Basic Research (grants 14-07-31172-mol-a, 14-07-00403-a and 15-07-07891-a) and by the Council on grants of the President of Russian Federation (grants SP-3667.2013.5, SP-1184.2015.5 and NSH-5007.2014.9).

Oleg Zaikin is a researcher at Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, e-mail: zaikin.icc@gmail.com.

Alexander Semenov is the chief of the laboratory of discrete analysis and applied logic at Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, e-mail: biclop.rambler@yandex.ru.

Ilya Otpuschennikov is a researcher at Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, e-mail: otilya@yandex.ru.

the known fragment of keystream we search for the values of all registers cells at the end of the initialization phase. Therefore, in our experiments we used CNF encodings where initialization phase was omitted. Usually it is believed that to uniquely identify the secret key it is sufficient to consider keystream fragment of length comparable to the total length of shift registers. Here we followed [6], [7] and set the keystream fragment length to 200 bits.

To encode the algorithm of the Bivium cipher into CNF we used TRANSALG tool [8]. As an input TRANSALG takes a source code of a program in the domain specific language named *TA language*. TA language is a procedural programming language with C-like syntax. This source code must describe an algorithm of computing of a discrete function. Output of TRANSALG is a CNF which encodes the considered algorithm.

Translation of a TA-program consists of two main stages. On the first stage a source code of a TA-program is parsed, and concrete syntax tree is constructed. On the second stage of translation, the symbolic execution of TA-program is performed. According to the concept of the symbolic execution, interpreter does not operate on concrete data, but with symbols which encode this data — Boolean variables and Boolean vectors (arrays). The symbolic execution of a TA-program results in a set of Boolean formulas, which is called *propositional encoding* of the algorithm. We will consider constructing of propositional encoding by TRANSALG on the example of TA-program which describes algorithm of the Bivium cipher.

The TA-program for the Bivium cipher starts from the declaration of integer constants (see Listing 1). Here constant `len` is equal to total length of the shift registers (177 bits); constants `lenA` and `lenB` define the lengths of each register (93 and 84 bits respectively); `stream_len` defines the number of bits of a keystream produced by the cipher (200 bits). After this, arrays of variables of type `bit` are declared. Main data type in the TA-language is the `bit` type. TRANSALG uses this type to establish links between variables used in a TA-program and Boolean variables included into corresponding propositional encoding. Below we will refer to variables that appear in a TA-program as *program variables*. All variables included in a propositional encoding are called *encoding variables*.

Global variables of type `bit` can be declared with attribute `__in` or `__out`. Attribute `__in` marks program variables associated with Boolean variables that encode input of an algorithm. In fact, in such a way we declare a set of variables with the following feature: any assignment of these variables derives values of all other variables in the propositional encoding. It should be noted that in every TA-program that describes some algorithm there must be declared variables which encode input of the considered algorithm. Attribute `__out` marks variables which, after TA-program is executed, contain an output of the algorithm.

In the Listing 1 the bit array `reg` with attribute `__in` is declared. This array contains a state of the shift registers of the generator. Attribute `__in` reports to the translator that initial state of the registers is unknown. Encoding variables, linked with program variables marked with attribute `__in`, get first `len` numbers in a propositional encoding.

Also in the Listing 1 the bit array `stream` with attribute `__out` is declared. This array contains the keystream produced by the cipher. Encoding variables, linked with program variables marked with attribute `__out`, get last `stream_len` numbers in a propositional encoding. Thus by using attributes `__in` and `__out` we uniquely identify the sets of Boolean variables in a propositional encoding, which encode input and output of an algorithm respectively.

```
define len  177;
define lenA 93;
define lenB 84;
define stream_len 200;
__in  bit reg[len];
__out bit stream[stream_len];
```

Listing 1. Declarations of global variables and constants

In the considered TA-program shifting of the registers is implemented in the form of the procedure `shift_regs` (see Listing 2). This procedure updates the state of the global bit array `reg`, according to the Bivium algorithm [4]. During interpretation of the procedure `shift_regs`, new encoding variables are created only to represent the result of feedback functions calculation. Operations of copying data between cells add nothing to a propositional encoding — only links between program variables and corresponding Boolean encoding variables are changed.

```
void shift_regs()
{
  bit t1 = reg[65]^reg[90]&reg[91]
           ^reg[92]^reg[170];
  bit t2 = reg[161]^reg[174]&reg[175]
           ^reg[176]^reg[68];
  int i;
  for(i=lenA-1; i > 0; i=i-1){
    reg[i] = reg[i-1];
  }
  reg[0] = t2;
  for(i=lenA+lenB-1; I>lenA; i=i-1){
    reg[i] = reg[i-1];
  }
  reg[lenA] = t1;
}
```

Listing 2. Function that implements one shift of the Bivium registers

Interpretation of a TA-program starts from executing the function `main`. This function is the entry of any TA-program. Presented in the Listing 3 function `main` implements the base loop of the Bivium cipher.

```
void main()
{
  for(int i=0;i<stream_len; i=i+1){
    bit t1 = reg[65] ^ reg[92];
    bit t2 = reg[161] ^ reg[176];
    stream[i] = t1 ^ t2;
    shift_regs();
  }
}
```

Listing 3. Implementation of the base work loop of the Bivium cipher

One iteration of the loop from the Listing 3 outputs one bit of a keystream. For this purpose the current state of the four registers cells are taken — their sum mod 2 gives next

keystream bit. After this, the shifting of the registers is performed by calling function `shift_regs`.

The considered TA-program describes the generation of 200 keystream bits by the Bivium cipher. The translation of this program results in the system of 465 Boolean equations. The transition from this system to the CNF is performed using the Tseitin transformations [9] and procedures of Boolean minimization. The final CNF which encodes generating of 200 keystream bits by Bivium is constructed over the set of 642 Boolean variables, and consists of 9560 clauses and 49920 literals.

III. USING SAT@HOME FOR SOLVING WEAKENED CRYPTANALYSIS PROBLEMS FOR BIVIUM

SAT@home [10] is a volunteer computing BOINC-based project aimed at solving hard combinatorial problems that can be effectively reduced to SAT. It was launched on September 29, 2011 by ISDCT SB RAS and IITP RAS. On February 7, 2012 SAT@home was added to the official list of BOINC projects with alpha status. Recently its status was improved to beta.

The SAT@home server uses a number of standard BOINC daemons responsible for sending and processing tasks (transitioner, feeder, scheduler, etc.). The work generator daemon decomposes the original SAT problem to subproblems according to partitioning found by our Monte Carlo method, implemented as a parallel application [11]. The validator checks the correctness of the results, and the assimilator processes correct results. The client application is based on the SAT solver MINISAT [12].

With respect to the estimation obtained by the Monte Carlo method [11] on a computing cluster, the solving of one instance of cryptanalysis of the Bivium cipher in the SAT@home project with its current performance would take about 2 years (using decomposition set of 50 variables [11]). That is why we decided to solve weakened cryptanalysis problems for this generator. Below we use the notation *BiviumK* to denote a weakened cryptanalysis problem for Bivium with known values of *K* variables (in corresponding SAT instance) encoding last *K* cells of the second shift register. In particular we considered the Bivium9 problem. We used our Monte Carlo method to find a decomposition set with good time estimation for Bivium9. As a result we obtained the decomposition set of 43 variables with time estimation 2 months for solving of one cryptanalysis instance of Bivium9 in SAT@home. From September 2014 to December 2014 with the help of this decomposition set 5 *Bivium*9 instances were solved in SAT@home (1 month in average). During the corresponding experiment for each cryptanalysis instance the search space was divided into 146602 equal tasks, each containing 60 millions subproblems. For processing one such task about 2 hours of 1 core of a modern CPU is needed. The client application was based on a modified version of MINISAT 2.2. Time estimation obtained by the Monte Carlo method shows strong correlation with real solving time of these cryptanalysis instances. During this experiment performance of SAT@home was increased by several computing clusters which were connected with the help of CluBORun tool [13].

IV. RELATED WORK

The authors of [6], [7] presented some estimations of the time required for the SAT-based cryptanalysis of the Bivium cipher. Apparently, [14] became the first paper about the use of a desktop grid based on the BOINC platform for solving SAT, but it did not evolve into a full-fledged volunteer computing project. The predecessor of the SAT@home was the BNB-Grid system [15], that was used to solve some hard SAT-based cryptanalysis problems.

V. CONCLUSION

Obtained results show that SAT@home can be successfully used for solving hard SAT-based cryptanalysis problems. We plan to use SAT@home for solving non-weakened cryptanalysis problem of the Bivium cipher. Also we plan to solve in SAT@home cryptanalysis problems for other stream ciphers.

ACKNOWLEDGMENT

We thank Mikhail Posypkin and Nickolay Khrapov for their help in maintaining the SAT@home project, and all the SAT@home volunteers for their participation.

REFERENCES

[1] M. N. Durrani and J. A. Shamsi, “Review: Volunteer computing: Requirements, challenges, and solutions,” *J. Netw. Comput. Appl.*, vol. 39, pp. 369–380, 2014.

[2] D. P. Anderson and G. Fedak, “The computational and storage potential of volunteer computing,” in *Proc. 6th IEEE International Symposium on Cluster Computing and the Grid,* Singapore, 2006, pp. 73–80.

[3] *Handbook of Satisfiability*, *Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press, 2009.

[4] C. D. Canniere, “Trivium: A stream cipher construction inspired by block cipher design principles,” in *Proc. 9th International Conference ISC*, Samos Island, Greece, 2006, pp. 171–186.

[5] A. Maximov and A. Biryukov, “Two trivial attacks on Trivium,” in *Proc. 14th International Workshop on Selected Areas in Cryptography*, Ottawa, Canada, 2007, pp. 36–55.

[6] T. Eibach, E. Pilz and G. Volkel, “Attacking Bivium Using SAT Solvers,” in *Proc. 11th International Conference on Theory and Applications of Satisfiability Testing*, Guangzhou, China, 2008, pp. 63–76.

[7] M. Soos, “Grain of Salt - an AutomatedWay to Test Stream Ciphers through SAT Solvers,” in *Proc. Workshop on Tools for Cryptanalysis*, London, UK, 2010, pp. 131–144.

[8] I. V. Otpuschennikov, A. A. Semenov and S. E. Kochemazov, “Transalg: a tool for translating procedural descriptions of discrete functions to SAT,” in *Proc. 5th International Workshop on Computer Science and Engineering: Information Processing and Control Engineering*, Moscow, Russia, 2015, pp. 289-294.

[9] G. S. Tseitin, “On the complexity of derivation in propositional calculus,” *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pp. 466–483, 1983.

[10] M. A. Posypkin, A. A. Semenov and O. S. Zaikin, “Using BOINC desktop grid to solve large scale SAT problems,” *Computer Science Journal,* vol. 13, no. 1, pp. 25–34, 2012.

[11] A. A. Semenov and O. S. Zaikin, “Using Monte Carlo method for searching partitionings of hard variants of Boolean satisfiability problem,” in *Proc. 13th International Conference on Parallel Computing Technologies*, Petrozavodsk, Russia, 2015, pp. 222–230.

[12] N. Een and N. Sorensson, “An extensible SAT-solver,” in *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing*, Santa Margherita Ligure, Italy, 2003, pp. 502–518.

[13] A. P. Afanasiev, I. V. Bychkov, M. O. Manzyuk, M. A. Posypkin, A. A. Semenov and O. S. Zaikin, “Technology for Integrating Idle Computing Cluster Resources into Volunteer Computing Projects,” in *Proc. of The 5th International Workshop on Computer Science and Engineering*, Moscow, Russia, 2015, pp. 109-114.

[14] M. Black and G. Bard, “SAT Over BOINC: An Application-Independent Volunteer Grid Project,” in *Proc. 12th IEEE/ACM International Conference on Grid Computing,* Lyon, France, 2011, pp. 226–227.

[15] Y. G. Evtushenko, M. A. Posypkin and I. Kh. Sigal, “A framework for parallel large-scale global optimization,” *Computer Science - Research and Development*, vol. 23(3-4), pp. 211-215, 2009.