INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2020, VOL. 66, NO. 3, PP. 443-448
Manuscript received February 28, 2020; revised July, 2020. DOI: 10.24425/ijet.2020.131897

Using SAT Solvers to Finding Short Cycles in Cryptographic Algorithms

Władysław Dudzic, and Krzysztof Kanciak

Abstract—**A desirable property of iterated cryptographic algorithms, such as stream ciphers or pseudo-random generators, is the lack of short cycles. Many of the previously mentioned algorithms are based on the use of linear feedback shift registers (LFSR) and nonlinear feedback shift registers (NLFSR) and their combination. It is currently known how to construct LFSR to generate a bit sequence with a maximum period, but there is no such knowledge in the case of NLFSR. The latter would be useful in cryptography application (to have a few taps and relatively low algebraic degree). In this article, we propose a simple method based on the generation of algebraic equations to describe iterated cryptographic algorithms and find their solutions using an SAT solver to exclude short cycles in algorithms such as stream ciphers or nonlinear feedback shift register (NLFSR). Thanks to the use of AIG graphs, it is also possible to fully automate our algorithm, and the results of its operation are comparable to the results obtained by manual generation of equations. We present also the results of experiments in which we successfully found short cycles in the NLFSRs used in Grain-80, Grain-128 and Grain-128a stream ciphers and also in stream ciphers Bivium and Trivium (without constants used in the initialization step).**

Keywords—**NLFSR, short cycles, stream ciphers, Trivium, Bivium, Grain-80, Grain-128**

I. Introduction

The phrase SAT solver is commonly used to refer to software that solves the boolean satisfiability problem (sometimes called propositional satisfiability problem and abbreviated SATISFIABILITY or SAT). It finds the evaluation of variables (0 or 1) for which all logical formulas of a given problem are met. This problem is an NP-complete problem as demonstrated by Stephan Cook in [17]. Currently there is no known algorithm which would effectively solve every SAT problem, and it is believed that such an algorithm does not exist. However, proof of this hypothesis has not been carried out. There are many recognized SAT solvers which use heuristic methods of solving the SAT problem. These can be grouped according to technique, e.g. DPLL (Davis-Putnam-Logemann-Loveland [11]) and CDCL (Conflict Driven Clause Learning [12]).

In cryptology, SAT solvers are successfully used among other methods in issues related to cryptanalysis of block and stream ciphers [14], hash functions [7], and in methods related to formal verification, automatic test pattern generation and logic synthesis [18]. The idea of using SAT solvers to search for short cycles of length n in iterated cryptographic algorithms (like stream ciphers) or primitives (like LFSR) is based on describing the n-iteration of an algorithm by an algebraic system of equations, adding equations where initial internal state equals final internal state, and then solving this system using an SAT solver. When the problem is unsatisfiable (unsat) the cycle of length n does not exist, but when it is satisfiable (sat), the n-cycle exists, and we get the initial value of the internal state. No known algorithm can check whether the NLFSR or stream cipher has a full period with at least polynomial complexity.

II. Previous work

Using SAT solvers to find short cycles in cryptographic algorithms is a relatively new approach, and the most promising results can be found in [6] and [5]. Table I shows the results of searching for short cycles in the NLFSRs used in Grain-80 and Grain-128 stream ciphers, and also in stream ciphers Bivium and Trivium (without constants used in the initialization step) presented in [5]. Cycles of length 1 consist of all 0. The experiments were run on a PC with Intel Core i7-4600U CPU at 2.1 GHz with 8 GB RAM running under Ubuntu 14.04 LTS.

Due to the addition of further conditions during calculations, the algorithm proposed in [5] makes it possible to find all cycles; however, its operation time is not satisfactory.

Currently, the search of NLFSRs with a maximum period is also an important issue. The articles [15] and [21] show results of searching such primitives using FPGA and [20] using GPGPU. However, a state of found registers is not to large.

It can be proved [19] that only nonsingular NLFSR may have a maximum period. The register is nosingilar if his feedback function has the form:

$$f(x_0, x_1, \cdots, x_{n-1}) = x_0 + g(x_1, \cdots, x_{n-1})$$

when we rotate register in left or:

$$f(x_0, x_1, \cdots, x_{n-1}) = x_{n-1} + g(x_0, \cdots, x_{n-2})$$

when we rotate register in right.

In other case is called singular [15]. However in general, it is not know how to construct NLFSR with large state and maximum period witch is cryptographically applicable.

III. Experiment

The proposed algorithm to find a cycle of length n consists of three steps:

This work was presented at the International Scientific Conference Mathematical Cryptology & Cybersecurity (MC&C 2020), Warsaw, 16-17.01.2020.

W. Dudzic is with Military University of Technology, Warsaw, Poland (e-mail: wladyslaw.dudzic@wat.edu.pl).

K. Kanciak is with Military University of Technologycity, Warsaw, Poland (e-mail: krzysztof.kanciak@wat.edu.pl).

TABLE I
RUNTIME, T, AND PEAK MEMORY CONSUMPTION, M (IN KBYTES), USED BY THE ALGORITHM TO FIND N CYCLES OF LENGTH K. [5]

	Trivium			Bivium			Grain-80 (NLFSR part)			Grain-128 (NLFSR part)		
k	*N*	*t*	*m*	*N*	*t*	*m*	*N*	*t*	*m*	*N*	*t*	*m*
1	1	0m 0.041s	0	1	0m 0.056s	0	1	0m 0.037s	0	1	0m 0.206s	12740
2		0m 0.027s	0		0m 0.045s	0		0m 0.035s	0		0m 1.205s	24704
3	21	0m 0.038s	0	5	0m 0.039s	0	1	0m 0.042s	0		0m 1.883s	32672
4		0m 0.038s	1284		0m 0.049s	0		0m 0.037s	0		0m 2.445s	41868
5		0m 0.043s	1288		0m 0.059s	0		0m 0.043s	1568		0m 2.817s	49816
6		0m 0.044s	1548		0m 0.052s	0		0m 0.037s	1572		0m 3.045s	59588
7		0m 0.047s	1544		0m 0.048s	1428		0m 0.038s	1576	1	0m 2.646s	70416
8		0m 0.094s	1548		0m 0.049s	1276		0m 0.047s	1568	1	0m 3.231s	78092
9		0m 0.101s	1548		0m 0.066s	1280		0m 0.048s	1572		0m 4.113s	86804
10	1	0m 0.315s	1808		0m 0.079s	1484		0m 0.044s	1576		0m 4.132s	97580
11	1	0m 0.224s	1808		0m 0.111s	1540		0m 0.059s	1568		0m 4.965s	110812
12	2	0m 0.815s	2600		0m 0.106s	1540	1	0m 0.062s	1576		0m 5.582s	117652
13		0m 0.047s	1804		0m 0.048s	1536		0m 0.042s	1572		0m 6.030s	128436
14		0m 0.678s	2340		0m 0.090s	1532		0m 0.081s	1576		0m 9.352s	144728
15	1	0m 0.350s	2072		0m 0.155s	1676		0m 0.083s	1572		0m 16.012s	157388
16		0m 6.446s	4004		0m 0.197s	1796		0m 0.103s	1840		0m 15.262s	164444
17		0m 7.896s	3956		0m 0.895s	2480		0m 0.154s	3156		0m 32.113s	180744
18		0m 41.498s	6736		0m 0.912s	2332		0m 0.217s	3420		1m 4.908s	201104
19		0m 56.107s	8696		0m 2.371s	2596		0m 0.397s	3688		2m 2.071s	217204
20		2m 19.115s	11844		0m 6.262s	3452		0m 0.230s	3684		2m 4.122s	220528
21		1m 59.727s	9888		0m 1.711s	2632		0m 0.772s	4480		3m 52.535s	243900
22		0m 47.351s	8028		0m 11.121s	3976		0m 1.010s	4852		4m 47.395s	250860
23		0m 0.202s	2324		0m 0.114s	1668		0m 1.422s	5196		6m 30.634s	278004
24		9m 3.292s	19916		0m 8.439s	3728		0m 0.492s	4328		6m 9.751s	270916
25		8m 5.652s	20684		0m 7.581s	3472		0m 4.458s	7704		19m 8.558s	318784
26		0m 0.119s	2060		0m 0.098s	1672		0m 0.905s	5128		17m 9.791s	315800
27		8m 47.293s	19708		0m 0.599s	2192		0m 7.466s	8948		50m 27.868s	381832
28		91m 19.361s	61840		0m 11.522s	4304		0m 1.289s	5344		30m 3.206s	364652
29		0m 0.184s	2592		1m 18.495s	9104		0m 19.940s	11204		70m 20.679s	422768
30		290m 9.726s	94608		1m 38.653s	9432		0m 7.175s	8380		91m 44.888s	463004
31		-	-		2m 54.762s	13124		0m 58.225s	13608		227m 24.577s	558228
32		-	-		0m 40.476s	7904		0m 23.957s	11876		182m 19.017s	580328
33		-	-		0m 44.421s	8424		1m 38.085s	15012		-	-
34		-	-		16m 28.714s	27116		5m 34.866s	26660		-	-
35		-	-		3m 30.944s	15096		0m 58.563s	15872		-	-
36		-	-		23m 58.093s	29756		0m 56.514s	14948		-	-
37		-	-		134m 14.915s	66292		1m 3.641s	16044		-	-
38		-	-		99m 36.861s	65756		24m 0.992s	49916		-	-
39		-	-		0m 0.224s	2324		0m 40.753s	16024		-	-
40		-	-		1m 15.837s	9304		3m 58.102s	27276		-	-
41		-	-		0m 12.274s	4992		6m 41.412s	30504		-	-
42		-	-		33m 8.211s	43660		32m 25.806s	54752		-	-
43		-	-		503m 28.785s	124964		10m 54.320s	38092		-	-
44		-	-		-	-		30m 11.795s	50580		-	-
45		-	-		-	-		10m11.411s	32060		-	-
46		-	-		-	-		683m36.550s	213228		-	-
47		-	-		-	-		-	-		-	-

1) generation of algebraic equations describing n iterations of cryptographic algorithms in algebraic normal form (ANF),
2) converting ANF to conjunctive normal form (CNF),
3) solving CNF problem using SAT solver.

In our experiments, we used two methods to generate algebraic equations that describe iterated cryptographic algorithms. The first is based on manually generated equations. Handwritten equations seem to be the most natural and readable for humans. For example, if the feedback function of NLFSR is:

$$f(x_0, x_1, x_2, x_3) = x_0 \oplus x_1 \oplus x_2 \oplus x_1 \cdot x_3$$

we can describe 2 iterations of register using the equations:

$$\begin{aligned}
x_4 &= x_0 \oplus x_1 \oplus x_2 \oplus x_1 \cdot x_3\\
x_5 &= x_0\\
x_6 &= x_1\\
x_7 &= x_2\\
x_8 &= x_4 \oplus x_5 \oplus x_6 \oplus x_5 \cdot x_7\\
x_9 &= x_4\\
x_{10} &= x_5\\
x_{11} &= x_6
\end{aligned}$$

and add conditions where the initial internal state is equal to the final internal state:

$$\begin{aligned}
x_0 &= x_8\\
x_1 &= x_9\\
x_2 &= x_{10}\\
x_3 &= x_{11}
\end{aligned}$$

where:

- x_0 x_1 x_2 x_3 - initial internal state
- x_4 x_5 x_6 x_7 - state after first iteration
- x_8 x_9 x_{10} x_{11} - state after second iteration

The second method uses automatically generated equations (based on Cryptol implementation which is translated to the and-inverted-graphs (AIG [2])) and converts them into ANF. In this process we use SAW [9] and ABC [22] from UC Berkeley. The idea of using an equation taken from implementation was earlier explored by Courtois et al. [14] to conduct an SAT attack on DES block cipher. In 2012, during SHA-3 competition, Homsirikamol et al. [7] developed a similar tool to obtain hardware equations that described SHA-3 final candidates and evaluated their security margin.

The conversion of ANF to CNF is performed using a modification of US open-source software available at https: //www.lukbettale.ze.cx/anf2cnf. During conversion, the *CNF CUT* parameter is always set to 3. It is possible that other conversion methods may affect the effectiveness of solving CNF problems. However, we have not yet conducted research in this area.

To solve a CNF problem, we used a Plingeling SAT solver on 44 cores Intel(R) Xeon(R) CPU E5-2699 v4 2.20 Ghz. These results are presented in this article. We also used SAT solvers CaDiCaL, Treengeling and Lingeling. A description of the solvers we used and their benchmarks is presented in article [1]. The maximum time limit for solving a task was set at 3600 seconds. During testing, it was assumed that after finding a cycle with the length n, there would be no further search for cycles with the length kn, where k is an integer greater than 0.

A. Analysis of 80-bit NLFSR from stream cipher Grain-80

The Grain-80 stream cipher was proposed in [13]. It has been selected for the final eSTREAM portfolio for profile 2 by the eSTREAM project. Grain stream ciphers are designed primarily for restricted hardware environments. The key stream generator contains 80-bit NLFSR and an 80-bit LFSR. The LFSR is known to have the maximum period of $2^{80} - 1$ since it uses a primitive generator polynomial of degree 80. The period of NLFSR is unknown and its feedback function F is:

$$\begin{aligned}
&F(x_0, ..., x_{79}) = 1 + x_{16} + x_{19} + x_{27} + x_{34} + x_{42} + x_{46} + x_{51} +\\
&x_{58} + x_{64} + x_{70} + x_{79} + x_{16}x_{19} + x_{42}x_{46} + x_{64}x_{70} + x_{19}x_{27}x_{34} +\\
&x_{46}x_{51}x_{58} + x_{16}x_{34}x_{51}x_{70} + x_{19}x_{27}x_{42}x_{46} + x_{16}x_{19}x_{58}x_{64} +\\
&x_{16}x_{19}x_{27}x_{34}x_{42} + x_{46}x_{51}x_{58}x_{64}x_{70} + x_{27}x_{34}x_{42}x_{46}x_{51}x_{58}
\end{aligned}$$

During testing of 80-bit NLFSR from a Grain-80 cipher (results in table II) we rotated register in right, periods of length 2 and 3 were detected. Furthermore, it was found that the examined register does not include periods from length 5 to 55 (excluding cycles of length $2k$ and $3k$, where k is an integer greater than 0 -– those lengths were omitted from the calculation). Found cycles we present in table III. In article [5] and results show in table I the cycle of length 2 was not found. Probably authors not consider of 1 which is add to NLFSR feedback function in original specification of Grain80.

B. Analysis of 128-bit NLFSR from stream cipher Grain-128

The Grain-128 stream cipher was proposed in [3]. The design is very small in hardware, and targets environments

TABLE II
ANALYSIS OF 80-BIT NLFSR FROM GRAIN-80.

n	result manual method	result automated method	time manual method	time automated method
2	sat	sat	0m 00.08s	0m 00.05s
3	sat	sat	0m 00.07s	0m 00.06s
5	unsat	unsat	0m 00.24s	0m 00.07s
7	unsat	unsat	0m 00.24s	0m 00.09s
11	unsat	unsat	0m 00.26s	0m 00.15s
13	unsat	unsat	0m 00.34s	0m 00.18s
17	unsat	unsat	0m 00.45s	0m 00.69s
19	unsat	unsat	0m 00.81s	0m 01.02s
23	unsat	unsat	0m 01.37s	0m 01.96s
25	unsat	unsat	0m 03.22s	0m 04.19s
29	unsat	unsat	0m 08.95s	0m 11.50s
31	unsat	unsat	0m 10.03s	0m 10.92s
35	unsat	unsat	0m 08.79s	0m 12.33s
37	unsat	unsat	0m 11.57s	0m 10.64s
41	unsat	unsat	1m 03.11s	0m 48.29s
43	unsat	unsat	0m 42.09s	0m 33.52s
47	unsat	unsat	7m 05.38s	3m 25.04s
49	unsat	unsat	16m 24.07s	26m 15.56s
53	unsat	timeout	58m 27.69s	-------
55	unsat	timeout	55m 26.48s	-------
59	timeout	timeout	-------	-------

TABLE III
CYCLES FOUND IN NLFSR FROM GRAIN-80.

n	state [hex]
2	AAAAAAAAAAAAAAAAAAAA
3	92492492492492492492

with very limited resources in gate count, power consumption, and chip area. Similar to Grain-80, it contains 128-bit NLFSR and 128-bit LFSR.

The LFSR is known to have the maximum period of $2^{128}-1$ since it uses a primitive generator polynomial of degree 128. The period of NLFSR is unknown, and its feedback function F is:

$$F(x_0, ..., x_{127}) = x_{127} + x_{101} + x_{71} + x_{36} + x_{31} + x_{124}x_{60} + x_{116}x_{114} + x_{110}x_{109} + x_{100}x_{68} + x_{87}x_{79} + x_{66}x_{62} + x_{59}x_{43}$$

During testing of 128-bit NLFSR from a Grain-128 cipher (results in table IV) we rotated register in right, periods of length 7, 8 and 59 were detected. Furthermore, it was found that the examined register does not include periods from length 2 to 44 (excluding cycles of length $7k$ and $8k$, where k is an integer greater than 0 — those lengths were omitted from the calculation). Found cycles we present in table V.

C. *Analysis of 128-bit NLFSR from stream cipher Grain-128a*

The Grain-128a stream cipher was proposed in [10]. The algorithm is a new version of Grain-128 and is strengthened against all known attacks and observations, with built-in support for optional authentication. The period of the new 128-bit NLFSR is also unknown, and its feedback function F is:

TABLE IV
ANALYSIS OF 128-BIT NLFSR FROM GRAIN-128.

n	result manual method	result automated method	time manual method	time automated method
2	unsat	unsat	0m 00.04s	0m 00.04s
3	unsat	unsat	0m 00.05s	0m 00.04s
4	unsat	unsat	0m 00.05s	0m 00.05s
5	unsat	unsat	0m 00.05s	0m 00.07s
6	unsat	unsat	0m 00.05s	0m 00.05s
7	sat	sat	0m 00.06s	0m 00.06s
8	sat	sat	0m 00.06s	0m 00.06s
9	unsat	unsat	0m 00.07s	0m 00.11s
10	unsat	unsat	0m 00.06s	0m 00.11s
11	unsat	unsat	0m 00.09s	0m 00.10s
12	unsat	unsat	0m 00.08s	0m 00.13s
13	unsat	unsat	0m 00.09s	0m 00.12s
15	unsat	unsat	0m 00.16s	0m 00.49s
17	unsat	unsat	0m 00.17s	0m 00.64s
18	unsat	unsat	0m 00.60s	0m 00.81s
19	unsat	unsat	0m 00.93s	0m 01.42s
20	unsat	unsat	0m 00.86s	0m 00.77s
22	unsat	unsat	0m 01.07s	0m 01.34s
23	unsat	unsat	0m 02.31s	0m 01.42s
25	unsat	unsat	0m 02.23s	0m 02.47s
26	unsat	unsat	0m 02.80s	0m 03.43s
27	unsat	unsat	0m 08.94s	0m 08.04s
29	unsat	unsat	0m 07.95s	0m 12.42s
30	unsat	unsat	0m 08.42s	0m 06.53s
31	unsat	unsat	0m 24.80s	0m 27.63s
33	unsat	unsat	1m 07.11s	1m 08.25s
34	unsat	unsat	0m 16.37s	0m 31.43s
36	unsat	unsat	1m 15.63s	2m 20.12s
37	unsat	unsat	0m 58.69s	1m 01.83s
38	unsat	unsat	3m 16.68s	3m 33.52s
39	unsat	unsat	0m 51.58s	1m 06.15s
41	unsat	unsat	4m 39.11s	4m 46.36s
43	unsat	unsat	17m 48.66s	15m 15.22s
44	unsat	unsat	25m 56.06s	20m 06.97s
45	timeout	timeout	-------	-------
50	timeout	timeout	-------	-------
51	timeout	timeout	-------	-------
52	timeout	timeout	-------	-------
53	timeout	timeout	-------	-------
54	timeout	timeout	-------	-------
55	timeout	timeout	-------	-------
57	timeout	timeout	-------	-------
58	timeout	timeout	-------	-------
59	sat	timeout	31m 58.28s	-------
60	timeout	timeout	-------	-------

$$F(x_0, ..., x_{127}) = 1 + x_{31} + x_{36} + x_{71} + x_{101} + x_{127} + x_{43}x_{59} + x_{60}x_{124} + x_{62}x_{66}x_{68}x_{100} + x_{79}x_{87} + x_{109}x_{110} + x_{114}x_{116} + x_{45}x_{49}x_{57} + x_{102}x_{103}x_{105} + x_{32}x_{34}x_{35}x_{39}$$

During testing of 128-bit NLFSR from a Grain-128a cipher (results in table VI) we rotated register in right, periods of length 3, 31, 37 and 65 were detected. Furthermore, it was found that the examined register does not include periods from length 2 to 41 and length 44 (excluding cycles of length $3k$, $31k$ and $37k$, where k is an integer greater than 0 -– those

TABLE V
Cycles found in NLFSR from Grain-128.

n	state [hex]
7	3A74E9D3A74E9D3A74E9D3A74E9D3A74
8	2F2F2F2F2F2F2F2F2F2F2F2F2F2F2F2F
59	C83BBB78A6B74C1907776F14D6E98320

lengths were omitted from the calculation). Found cycles we present in table VII.

D. Analysis of Bivium and Trivium

We also applied the presented algorithm to Trivium [4] stream ciphers and his simpler version Bivium [8]. Referring to figure 1 in Bivium we can distinguish two LFSR's: 93-bit register A and 84-bit register B. The Internal state of Bivium consists 177 bits.

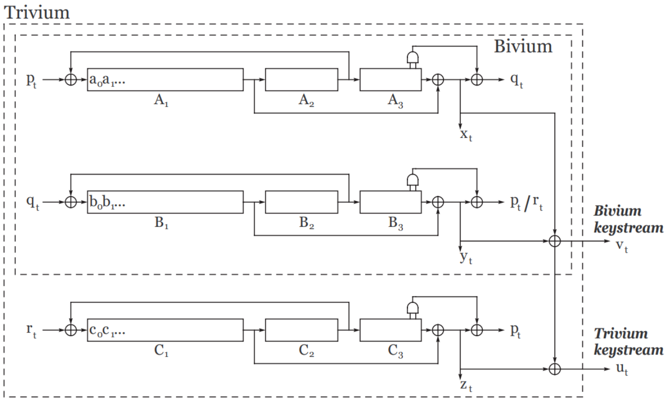

Fig. 1. Bivium and Trivium stream ciphers

The Trivium stream cipher includes an additional 111-bit register C. All used LFSRs are known to have the maximum period, since Trivium uses a primitive generator polynomials of degree 93, 84, 111. The internal state of Trivium consists of 288 bits. Trivium was submitted to the profile 2 (hardware) of the eSTREAM competition and has been selected as part of the portfolio for low area hardware ciphers profile 2 by the *eSTREAM* project [16]. It is not patented and has been specified as an International Standard under ISO/IE. The algorithm generates up to 2^{64} bits of output keystream from an 80-bit key and an 80-bit IV vector.

During testing Bivium key stream generator (results in table VIII) were not included constants used in the initialization step. A period of length 3 was detected. Furthermore, it was found that the examined construction does not include periods from length 2 to 46 or from length 52 to 55 (excluding cycles of length $3k$, where k is an integer greater than 0 — those lengths were omitted from the calculation). Found cycle we present in table IX (|| is bitwise concatenation).

During testing Trivium key stream generator (results in table X) were not included constants used in the initialization step of Trivium. Periods of length 3, 10 and 11 were detected. Furthermore, it was found that the examined construction does not include periods from length 2 to 31 (excluding cycles of length $3k$, $10k$ and $11k$, where k is an integer greater than 0 -– those lengths were omitted from the calculation). We also examined Trivium with key stream generators as included constants used in the initialization step, but no cycles were found from length 2 to 177. Found cycles we present in table XI.

TABLE VI
Analysis of 128-bit NLFSR from Grain-128a.

n	result manual method	result automated method	time manual method	time automated method
2	unsat	unsat	0m 00.05s	0m 00.29s
3	sat	sat	0m 00.04s	0m 00.33s
4	unsat	unsat	0m 00.05s	0m 00.39s
5	unsat	unsat	0m 00.05s	0m 00.27s
7	unsat	unsat	0m 00.06s	0m 00.33s
8	unsat	unsat	0m 00.05s	0m 00.30s
10	unsat	unsat	0m 00.07s	0m 00.36s
11	unsat	unsat	0m 00.07s	0m 00.41s
13	unsat	unsat	0m 00.10s	0m 00.45s
14	unsat	unsat	0m 00.09s	0m 00.51s
16	unsat	unsat	0m 00.39s	0m 00.79s
17	unsat	unsat	0m 00.17s	0m 01.00s
19	unsat	unsat	0m 01.38s	0m 02.01s
20	unsat	unsat	0m 01.00s	0m 01.68s
22	unsat	unsat	0m 01.16s	0m 02.15s
23	unsat	unsat	0m 02.11s	0m 02.55s
25	unsat	unsat	0m 05.51s	0m 06.29s
26	unsat	unsat	0m 04.11s	0m 04.49s
28	unsat	unsat	0m 04.00s	0m 05.18s
29	unsat	unsat	0m 11.42s	0m 18.47s
31	sat	sat	0m 01.11s	0m 06.75s
32	unsat	unsat	0m 30.62s	0m 54.72s
34	unsat	unsat	0m 30.00s	1m 22.09s
35	unsat	unsat	0m 35.77s	0m 46.86s
37	sat	sat	0m 03.84s	0m 10.74s
38	unsat	timeout	19m 19.39s	-------
40	unsat	unsat	2m 05.77s	5m 17.70s
41	unsat	timeout	29m 58.56s	-------
43	timeout	timeout	-------	-------
44	unsat	unsat	17m 45.83s	47m 14.03s
46	timeout	timeout	-------	-------
47	timeout	timeout	-------	-------
49	timeout	timeout	-------	-------
50	timeout	timeout	-------	-------
52	timeout	timeout	-------	-------
53	timeout	timeout	-------	-------
55	timeout	timeout	-------	-------
56	timeout	timeout	-------	-------
58	timeout	timeout	-------	-------
59	timeout	timeout	-------	-------
61	timeout	timeout	-------	-------
64	timeout	timeout	-------	-------
65	timeout	sat	-------	57m 56.46s

E. Manualy generation of equantions versus automated generation of equantions

In our experiments we used two methods to generate equations in algebraic normal form: manual (based on handwritten equations) and automated (based on AIG graphs). The main

TABLE VII
CYCLES FOUND IN NLFSR FROM GRAIN-128A.

n	state [hex]
3	6DB6DB6DB6DB6DB6DB6DB6DB6DB6DB6D
31	0E739A721CE734E439CE69C8739CD390
37	FBA40E3E6FDD2071F37EE9038F9BF748
65	049D61EB869158AE824EB0F5C348AC57

TABLE VIII
ANALYSIS OF BIVIUM
(WHITHOUT CONSTANTS USING IN INITIALIZATION STEP)

n	result automated method	time automated method
2	unsat	0m 00.05s
3	sat	0m 00.03s
4	unsat	0m 00.03s
5	unsat	0m 00.03s
7	unsat	0m 00.03s
8	unsat	0m 00.04s
10	unsat	0m 00.05s
11	unsat	0m 00.08s
13	unsat	0m 00.05s
14	unsat	0m 00.07s
16	unsat	0m 00.10s
17	unsat	0m 00.17s
19	unsat	0m 00.24s
20	unsat	0m 00.33s
22	unsat	0m 00.36s
23	unsat	0m 00.08s
25	unsat	0m 00.32s
26	unsat	0m 00.06s
28	unsat	0m 00.44s
29	unsat	0m 01.67s
31	unsat	0m 05.69s
32	unsat	0m 01.01s
34	unsat	0m 28.18s
35	unsat	0m 04.36s
37	unsat	3m 19.98s
38	unsat	2m 22.31s
40	unsat	0m 01.78s
41	unsat	0m 00.37s
43	unsat	53m 20.32s
44	unsat	39m 57.73s
46	unsat	2m 33.90s
47	timeout	-------
49	timeout	-------
50	timeout	-------
52	unsat	0m 03.90s
53	unsat	58m 16.13s
55	unsat	0m 40.02s
56	timeout	-------

differences between the two sets are the number of equations and the maximal algebraic degree. For the method based on handwritten equations, the maximal algebraic degree depends on the form of the NLFSR feedback function, and for the method based on AIG graphs, the maximal algebraic degree is always equal to 2. This is a natural consequence of the construction of AIG graphs. For this reason, the number of equations is higher when the second method is used.

TABLE IX
CYCLES FOUND IN BIVIUM
(WITHOUT CONSTANTSUSED IN THE INITIALIZATION STEP).

n	state [hex || bit]
3	00000000000000000000000124 924924924924924924 || True

TABLE X
ANALYSIS OF TRIVIUM
(WHITHOUT CONSTANTS USING IN INITIALIZATION STEP)

n	result automated method	time automated method
2	unsat	0m 00,05s
3	sat	0m 00,06s
4	unsat	0m 00,07s
5	unsat	0m 00,07s
7	unsat	0m 00,09s
8	unsat	0m 00,09s
10	sat	0m 00,14s
11	sat	0m 00,09s
13	unsat	0m 00,07s
14	unsat	0m 00,14s
16	unsat	0m 00,31s
17	unsat	0m 00,37s
19	unsat	0m 01,14s
23	unsat	0m 00,11s
25	unsat	0m 14,38s
26	unsat	0m 00,11s
28	unsat	2m 49,88s
29	unsat	0m 00,11s
31	unsat	51m 54,39s
32	timeout	-------

The task in the CNF can be characterized by the number of variables and the number of clauses. Due to the ANF systems from which our CNF tasks were generated, it was typical that CNF systems generated from the AIG based method had many more variables and clauses than those generated from handwritten equations. In both cases, it is clear that the increase in both indicators is linear. The relevant dependencies are shown in figure 2 and 3.

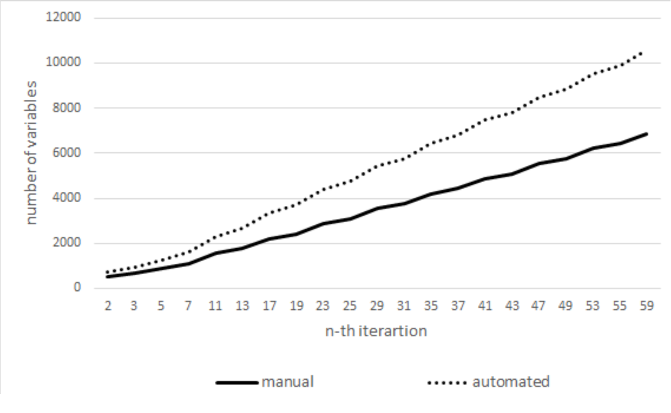

Fig. 2. Number of variables in CNF on n-th iterationfor Grain-80

Despite the significant differences between CNF generated by the handwritten equations method and the AIG graph

TABLE XI
CYCLES FOUND IN TRIVIUM
(WITHOUT CONSTANTSUSED IN THE INITIALIZATION STEP).

n	state [hex]
3	000000000000000000000004924924924924 924924920000000000000000000000000000
10	94E5394E5394E5394E5394E733CCF33CCF33 CCF33CCF6F5BD6F5BD6F5BD6F5BD6F5BD6F5
11	4148290520A4148290520A413E27C4F89F13 E27C4F899BC3786F0DE1BC3786F0DE1BC378

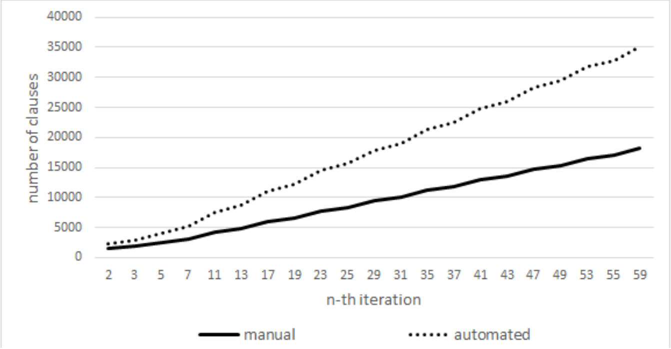

Fig. 3. Number of clauses in CNF on n-th iterationfor Grain-80

method, the time to find solutions is similar. The time difference is due to heuristic methods of solving the CNF task by Plingeling SAT solver. Considering this fact, it is difficult to assess which method of generating equations is better. Both methods allow the occurrence of short cycles in tested NLFSRs and stream ciphers to be easily checked.

IV. CONCLUSION

In summary, our experiments discovered short cycles in the NLFSRs used in Grain-80, Grain-128 and Grain-128a stream ciphers (all examineted registers are nonsingular in case when they will be rotate in right) and also in stream ciphers Bivium and Trivium (without constants used in the initialization step). Furthermore, by obtaining proof of the contradiction of the SAT problem, we also determined the number of iterations for which such cycles do not exist.

The time needed to find a cycle or proof of its absence is better than that of the algorithm used in [5]. This is clearly shown as the iteration of the given transformation increases. This fact allowed us to evaluate a larger range of iterations than was tested in [5]. Unfortunately, due to the nature of the SAT problem, it we did not estimated the computational and memory complexity. This is the main disadvantage with respect to the method proposed in [5].

In the future, we want to use the divide-and-conquer strategy in solving SAT. We believe that this approach can significantly reduce the calculation time, which will allow for evaluation of a larger range of iterations.

In our opinion, the presented method may prove to be a good approach to check whether a given iterated cryptographic algorithm has short cycles. Certainly, it is useful when no other algorithms exist (except brute force) to check this property (i.e. in the case of NLFSR). It can also be useful in determining the distribution of cycles of tested transformation.

From the cryptanalysis point of view, it will be interesting to check how the cycles found affect the security of the examined algorithms. The occurrence of short cycles in the key stream generator in practice discredits such an algorithm

REFERENCES

[1] A. Biere, "CADICAL, LINGELING, PLINGELING, TREENGELING and YALSAT Entering the SAT Competition 2017".

[2] A. Biere, "The AIGER And-Inverter Graph (AIG) Format", 2007.

[3] A. Maximov, M. Hell, T. Johansson and W. Meier, "A Stream Cipher Proposal: Grain-128", *IEEE International Symposium on Information Theory*, 2006, doi: 10.1109/ISIT.2006.261549.

[4] C. De Canniere and B. Preneel, "Trivium Specifications", *Springer Berlin Heidelberg*, pages 244-266, 2008, doi: 10.1007/978-3-540-68351-3_18.

[5] E. Dubrova and M. Teslenko, "An efficient SAT-based algorithm for finding short cycles in cryptographic algorithms", *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 2018, doi: 10.1109/HST.2018.8383892.

[6] E. Dubrova and M. Teslenko, "On Finding Short Cycles in Cryptographic Algorithms", *Cryptology ePrint Archive, Report 2016/1068*, 2016.

[7] E. Homsirikamol, P. Morawiecki, M. Rogawski and M. Srebrny, "Security Margin Evaluation of SHA-3 Contest Finalists through SAT-Based Attacks", *Computer Information Systems and Industrial Management*, pages 56–67, 2012, doi: 10.1007/978-3-642-33260-9_4.

[8] J. Borghoff, L. R. Knudsen and M. Stolpe, "Bivium as a Mixed-Integer Linear Programming Problem", *Lecture Notes in Computer Science (LNCS)*, Vol. 5921, pp 133-152, 2009, doi: 10.1007/978-3-642-10868-6_9.

[9] K. Carter, A. Foltzer, J. Hendrix, B. Huffman, A. Tomb, "SAW: the software analysis workbench", *Proceedings of the 2013 ACM SIGAda annual conference on High integrity language technology*, pages 15-18, 2013, doi: 10.1145/2527269.2527277.

[10] M. Ågren, M. Hell, T. Johansson and W. Meier, "Grain-128a: a new version of Grain-128 with optional authentication", *International Journal of Wireless and Mobile Computing*, 2011, doi: 10.1504/IJWMC.2011.044106.

[11] M. Davis and H. Putnam, "A Computing Procedere for Quantification Theory", *Journal of the ACM*, 1960, doi: 10.1145/321033.321034.

[12] M. Davis, G. Logemann, D. Loveland, "A Machine Program for Theorem Proving", *Communications of the ACM*, 1962, doi: 10.1145/368273.368557.

[13] M. Hell, T. Johansson and W. Meier, "Grain: a stream cipher for constrained environments", *International Journal of Wireless and Mobile Computing*, 2007, doi: 10.1504/IJWMC.2007.013798.

[14] N. T. Courtois and G. V. Bard, "Algebraic cryptanalysis of thedata encryption standard", *Springer Berlin Heidelberg*, pages 152–169, 2007, doi: 10.1007/978-3-540-77272-9_10.

[15] T. Rachwalik, J. Szmidt, R. Wicik and J. Zabłocki, "Generation of Nonlinear Feedback Shift Registers with Special-Purpose Hardware", *2012 Military Communications and Information Systems Conference, MCC 2012*, 2012.

[16] S. Babbage, J. Borghoff and V. Velichkov, "The eSTREAM Portfolio in 2012", *ICT-2007-216676. ECRYPT II*, 2012.

[17] S. Cook, "The complexity of theorem proving procedures", *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 1971, doi: 10.1145/800157.805047.

[18] S. Hassoun and S. Tsutomu, "Logic Synthesis and Verification", *Kluwer Academic Publishers*, 2002, doi: 10.1007/978-1-4615-0817-5.

[19] S. W. Golomb, "Shift Register Sequences", *Aegean Park Press*, 1981, doi: 10.5555/578271.

[20] P. Augustynowicz and K. Kanciak, "Scalable Method of Searching for Full-period Nonlinear Feedback Shift Registers with GPGPU. New List of Maximum Period NLFSRs" *International Journal of Electronics and Telecommunications*, 2018, doi: 10.24425/119365.

[21] P. Dąbrowski, G. Łabuzek, T. Rachwalik and J. Szmidt "Searching for Nonlinear Feedback Shift Registers with Parallel Computing", *Information Processing Letters*, 2013, 10.1016/j.ipl.2013.12.002.

[22] R. Brayton, A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", *Springer Berlin Heidelberg*, pages 24-40, 2010, doi: 10.1007/978-3-642-14295-6_5.