An Improved SAT-based Guess-and-Determine Attack on the Alternating Step Generator

Oleg Zaikin and Stepan Kochemazov

Matrosov Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
zaikin.icc@gmail.com, veinamond@gmail.com

Abstract. In this paper, we propose an algorithm for constructing guess-and-determine attacks on keystream generators and apply it to the cryptanalysis of the alternating step generator (ASG) and two its modifications (MASG and MASG0). In a guess-and-determine attack, we first "guess" some part of an initial state and then apply some procedure to determine, if the guess was correct and we can use the guessed information to solve the problem, thus performing an exhaustive search over all possible assignments of bits forming a chosen part of an initial state. We propose to use in the "determine" part the algorithms for solving Boolean satisfiability problem (SAT). It allows us to consider sets of bits with nontrivial structure. For each such set it is possible to estimate the runtime of a corresponding guess-and-determine attack via the Monte-Carlo method, so we can search for a set of bits yielding the best attack via a black-box optimization algorithm augmented with several SAT-specific features. We constructed and implemented such attacks on ASG, MASG, and MASG0 to prove that the constructed runtime estimations are reliable. We show, that the constructed attacks are better than the trivial ones, which imply exhaustive search over all possible states of the control register, and present the results of experiments on cryptanalysis of ASG and MASG/MASG0 with total registers length of 72 and 96, which have not been previously published in the literature.

Keywords: keystream generator, alternating step generator, cryptanalysis, guess-and-determine attack, SAT, Monte Carlo

1 Introduction

The alternating step generator (ASG) was proposed in [16]. It consists of two stop/go clocked binary Linear Feedback Shift Registers (LFSRs), $\mathrm{LFSR_X}$ and $\mathrm{LFSR_Y}$, and a regularly clocked binary LFSR, $\mathrm{LFSR_C}$. The clock-control bit defines which of the two stop/go LFSRs is clocked, and the keystream bit is obtained as the bitwise sum of stop/go LFSRs' output bits. There exist many attacks on ASG. The majority of them (e.g., [14, 15, 19, 20]) follow the divide-and-conquer approach, where a correlation attack is performed on stop/go LFSRs.

There is a number of ASG modifications. In [32] two of its modifications (MASG and MASG0) were proposed. They are based on replacing stop/go LFSRs by Nonlinear Feedback Shift Registers (NLFSRs). Because of the nonlinearity of the controlled registers, it is unlikely that most attacks on ASG can be easily extended to them.

In the present paper we develop the guess-and-determine approach to ASG, MASG and MASG0 cryptanalysis. The most simple variant of a guess-and-determine attack on ASG looks as follows. First, we "guess" the initial state of the control register (e.g., see [16, 34]). By guessing we mean assigning values to corresponding bits. After this we write a system of equations over bits corresponding to states of controlled LFSRs and "determine" using appropriate methods if the system is consistent and has a solution. It is clear that to find a correct "guess" we need to perform an exhaustive search over all possible states of the control register. An interesting question is whether there exist less trivial sets of bits than that comprising the control register, and if they do, how can one solve the systems of (in a general case) nonlinear equations produced by assigning values to the corresponding bits? In the present paper we positively answer the former question thanks to applying algorithms for solving Boolean satisfiability problem (SAT) [4] to the latter.

SAT is formulated as follows: for a given propositional formula to either find its satisfying assignment (the assignment of all its variables that makes formula True), or to prove that it is unsatisfiable. Because SAT is an NP-hard problem, it means that even if our simplified system of equations contains nonlinear entries, we can still reduce it to SAT and solve it in such form. It is important to notice, that while state-of-the-art SAT solving algorithms (usually referred to as SAT solvers) show remarkable performance on a huge variety of test samples, it is impossible to know in advance how long will it take to solve each particular SAT instance. Nevertheless, following a number of papers [10, 30] we show that it is possible to construct a runtime estimation of cryptanalysis of a keystream generator for each chosen set of bits to guess, SAT solver and keystream fragment size. This runtime estimation is constructed computationally via the Monte Carlo method [12] and can not be expressed by formula.

Thus, we can construct a guess-and-determine attack for an arbitrary subset of a set of bits, corresponding to an initial state of a keystream generator and estimate its runtime. It means, that using black-box optimization algorithms we can in fact organize an automatic procedure for finding good subsets of bits that yield better attacks. It was done before in application to several generators [10, 28–30], but the previous papers did not take into account a number of important SAT-related issues, thus the approach presented in our paper simply works better in one or the other aspect.

Let us present a brief outline of the paper. In Section 2 we briefly describe ASG and its modifications studied in the paper, and focus on particular configurations of ASG, MASG and MASG0 (as well as their SAT encodings). In Section 3 we suggest a new Monte-Carlo based algorithm, which for a given generator allows to construct a SAT-based guess-and-determine attack with a good runtime estimation and discuss why the runtime estimations constructed can be believed to be reliable. In Section 4 we construct such attacks on ASG (with 72-bit, 96-bit, and 192-bit initial states), MASG and MASG0 (both of them with 72-bit initial states). For each considered generator configuration (except the 192-bit ASG version) we prove that our runtime estimations are correct by solving 20 cryptanalysis instances. We also show that the constructed SAT-based guess-and-determine attacks are better than the trivial SAT-based guess-and-determine attacks in all cases. In the rest of the paper we observe the related work and draw conclusions.

2 Considered Cryptanalysis Problems

As it was outlined above, unlike most cryptanalytic attacks our approach does not make it possible to construct a general formula that would express its complexity. Rather, we can construct runtime estimation for each particular cryptanalysis problem. As such, hereinafter we consider cryptanalysis problems for three configurations of ASG – with total length of registers equal to 72, 96 and 192 (further we will refer to them as ASG-72, ASG-96 and ASG-192). Below we show the primitive polynomials used in each version.

ASG-72:

- $\mathrm{LFSR_C}$ (23 bits): $X^{23} \oplus X^{22} \oplus X^{20} \oplus X^{18} \oplus 1$;
- $\mathrm{LFSR_X}$ (24 bits): $X^{24} \oplus X^{23} \oplus X^{22} \oplus X^{17} \oplus 1$;
- $\mathrm{LFSR_Y}$ (25 bits): $X^{25} \oplus X^{24} \oplus X^{23} \oplus X^{22} \oplus 1$.

ASG-96:

- $\mathrm{LFSR_C}$ (31 bits): $X^{31} \oplus X^{7} \oplus 1$;
- $\mathrm{LFSR_X}$ (32 bits): $X^{32} \oplus X^{7} \oplus X^{5} \oplus X^{3} \oplus X^{2} \oplus X \oplus 1$;
- $\mathrm{LFSR_Y}$ (33 bits): $X^{33} \oplus X^{16} \oplus X^{4} \oplus X \oplus 1$.

ASG-192:

- $\mathrm{LFSR_C}$ (61 bits): $X^{61} \oplus X^{60} \oplus X^{46} \oplus X^{45} \oplus 1$;
- $\mathrm{LFSR_X}$ (64 bits): $X^{64} \oplus X^{63} \oplus X^{61} \oplus X^{60} \oplus 1$;
- $\mathrm{LFSR_Y}$ (67 bits): $X^{67} \oplus X^{66} \oplus X^{58} \oplus X^{57} \oplus 1$.

We also consider cryptanalysis problems for MASG and MASG0, which were proposed in [32]. In these modifications $\mathrm{LFSR_X}$ and $\mathrm{LFSR_Y}$ are replaced by NLFSRs, to which we refer below as $\mathrm{NLFSR_X}$ and $\mathrm{NLFSR_Y}$. In MASG a keystream bit is produced similarly to the original ASG: as a bitwise sum of output bits of $\mathrm{NLFSR_X}$ and $\mathrm{NLFSR_Y}$. In MASG0 a keystream bit is produced as a bitwise sum of outputs of all three registers ($\mathrm{LFSR_C}$, $\mathrm{NLFSR_X}$ and $\mathrm{NLFSR_Y}$). For both MASG and MASG0 the following feedback polynomials were used:

- $\mathrm{LFSR_C}$ (23 bits): $X^{23} \oplus X^{22} \oplus X^{20} \oplus X^{18} \oplus 1$;
- $\mathrm{NLFSR_X}$ (24 bits): $X^{19} \cdot X^{8} \oplus X^{16} \oplus X^{10} \oplus X^{9} \oplus X^{2} \oplus X$;
- $\mathrm{NLFSR_Y}$ (25 bits): $X^{24} \cdot X^{22} \cdot X^{2} \oplus X^{17} \oplus X^{5} \oplus X$.

It should be noted, that here we used the same $\mathrm{LFSR_C}$, as in ASG-72. The polynomials for NLFSRs were taken from [7, 25]. So, we consider MASG and MASG0 configurations with total length of registers equal to 72 (further we will refer to them as MASG-72 and MASG0-72).

The transition from an original problem to SAT is usually quite nontrivial (see survey [27]). There exist several openly available automatic tools that make it possible to reduce cryptanalysis problems to SAT [11, 18, 26, 31]. These tools produce relatively similar encodings, thus we applied the `Transalg` tool [26] to construct the SAT encodings for considered configurations of generators. In particular, for each considered configuration we obtained a Conjunctive Normal Form (CNF). In Table 1 we present the size, number of clauses, number of variables and keystream fragment size for the constructed CNFs. In Section 4 we will describe, why exactly these keystream fragment sizes were used.

Table 1: Characteristics of CNFs encoding the considered keystream generators.

Generator	Size, Mb	Variables	Clauses	Keystream fragment size
ASG-72	0.3	3 426	15 382	76
MASG-72	0.5	3 426	20 454	76
MASG0-72	0.5	3 426	20 758	76
ASG-96	0.7	6 658	32 166	112
ASG-192	1.9	22 705	95 326	200

3 Algorithm for Constructing SAT-based Guess-and-Determine Attacks

Let C be a CNF encoding a cryptanalysis problem for some keystream generator. Assume that X^{in} is a set of Boolean variables corresponding to an initial state of generator registers. In the case of ASG-96 (see Section 2), $|X^{in}| = 96$ (while there are 6658 Boolean variables in the corresponding CNF in total). We can choose some subset $X^* \subset X^{in}$ and consider all possible assignments of variables from X^*. Below let us refer to X^* as to *set of partitioning variables* and to a family of subproblems, formed by adding information about a particular assignment of variables from X^* to an original CNF for a considered problem, as to a *partitioning* [17].

It is easy to see, that on the one hand any subproblem from a partitioning should most likely be much easier to solve compared to an original problem (since we "know" a sizable chunk of information we need), and on the other hand by processing all such subproblems we will be able to obtain a solution of a considered hard problem. Of course, there exists some trade-off between the size and contents of X^* and the difficulty rate of constructed subproblems. It is not always possible to evaluate this trade-off analytically, so in a number of papers [5, 10, 29, 30] there were studied several ways how it can be achieved automatically or at least semi-automatically. Basically it all boils down to the problem of how to choose the best X^*.

It is clear, that any X^* corresponds to some guess-and-determine attack on a considered keystream generator. The nontrivial fact consists in the fact that for a given X^* it is possible to estimate a runtime of a corresponding attack. Essentially, the estimation can be done by means of the Monte Carlo method [12]: we choose relatively small random sample of subproblems from our partitioning, solve them, compute the average time required to solve one subproblem and scale it to the number of subproblems. However, in reality, there are many important nuances.

Let us describe the basic Monte-Carlo-based procedure, which is usually used to obtain the runtime estimation for a set of partitioning variables. The procedure takes as an input a CNF C, a known keystream fragment F, a set of partitioning variables X^*, and the number N, representing the size of a random sample. The procedure works as follows.

1. Construct a random sample S by choosing N binary words from $\{0, 1\}^{|X^*|}$ according to the uniform distribution.

2. Launch Conflict-Driven Clause Learning (CDCL, [21]) solver on N SAT instances formed by appending information from F and $s_i \in S$ to C and record the runtime of the solver on this instance to t_i.
3. Compute the runtime estimation by averaging t_i over S and multiplying the constructed value by the size of a partitioning: $R = 2^{|X^*|} \times \frac{\sum_{i=1}^{N} s_i}{N}$.

The described procedure defines an objective function – using some optimization algorithm one can try to find a set of partitioning variables with minimal value of this function. For this purpose it is natural to first construct a search space of all possible sets of partitioning variables (i.e. all possible subsets of a set of Boolean variables corresponding to an initial state of a considered keystream generator). Each point in this search space corresponds to some guess-and-determine attack. For every point we can calculate a runtime estimation using the objective function defined above. By traversing a search space via some optimization algorithm we can find a set of partitioning variables with a good runtime estimation. In our experiments in the role of such algorithm we use a simple tabu-based local search algorithm. As its starting point we always choose a set X^{in}. The optimization algorithm stops after reaching a given time limit. The output of this algorithm is a best found attack (compared to that, processed by the algorithm during its work).

We implemented the procedure described above and applied it to construct guess-and-determine attacks on several ASG configurations. However, it turned out that it could only find sets of partitioning variables for which the runtime estimations were very inaccurate: sometimes the solving time was several times larger (and sometimes lower) than runtime estimation. Also, in most experiments the found guess-and-determine attacks were worse than the trivial attack that implies guessing the bits corresponding to the initial state of the control register.

When we studied, why the described procedure gives very inaccurate estimations for the considered problems, we found out that it often gives overly optimistic or overly pessimistic estimations for a given set of partitioning variables because of the way the random sample is constructed. For cryptographic instances it is a common situation when for some set of partitioning variables the percentage of very simple subproblems in a partitioning is very large. By simple problems we mean here the ones that can be solved effectively – by means of Unit Propagation algorithm [6]. Meanwhile, the rest of the subproblems (not solved by Unit Propagation) can be exceptionally hard, such that one hard subproblem is solved many times longer than a whole random sample of simple subproblems. In other words, if we generate a random sample in the most simple way possible without additional consideration, it is often the case that a constructed sample does not adequately represent a partitioning, and even increasing its size has little to no effect.

Thus, the problem with the outlined scheme lies mainly in the first step of the procedure — how a random sample is constructed. So we decided to modify the procedure in such a way that it works well on the considered problems. Basically, on the one hand, we want the new procedure to construct random samples which contain subproblems that are not all solved by Unit Propagation. For this purpose we need to introduce some filtering procedure that determines if a problem can be solved by unit propagation or not. This procedure can be constructed by stripping a SAT solver down. On the other hand, we do not want to just neglect unit propagation stage at all – it can provide a sizable chunk of runtime.

New procedure takes as an input several parameters: X^* – the set of partitioning variables, D – a number of diapasons to be processed, s – a diapason size, K – a number of problems that have to be constructed within the diapason and not be solved by unit propagation. It works as follows.

1. Construct D binary words chosen randomly according to the uniform distribution from $\{0,1\}^{|X^*|}$. These D points serve as diapason starting values.
2. Process each constructed diapason beginning from a starting value. Attempt to construct K problems that are not solved by unit propagation, by sequentially applying the filtering procedure to each next word taken from a diapason in a lexicographic order.
3. If K such words were constructed, while not exceeding the diapason size, then the corresponding K words are returned as a result, along with the number of words P that did not pass filtering.
4. Solve K corresponding subproblems by a CDCL solver (without any limitations) in the incremental mode [9] (this mode prevents runtime estimation from being too pessimistic).
5. Calculate an average runtime for each diapason, taking into account both runtime on subproblems solved by unit propagation and that on the subproblems solved by a CDCL solver in the incremental-based loop.
6. Compute the runtime estimation for X^* by averaging t_i over D and multiplying the constructed value by the size of a partitioning.

The suggested procedure, augmented by the aforementioned black-box optimization algorithm, was implemented in the form of a parallel program, which is based on Message Passing Interface (MPI) [13]. To solve subproblems we employ the `ROKK` CDCL-solver, which is a slightly modified version of `MiniSat` 2.2 [8]. According to our experience [26], it shows good results in cryptanalysis of keystream generators.

One thread of our program is a control thread, while the others are computing threads. Each computing thread receives tasks from the control thread, performs the corresponding calculations and sends obtained results. This program works in two modes – the estimating mode and the solving mode. In the estimating mode, in order to calculate a runtime estimation for a particular X^*, the control thread first randomly generates D binary words of size $|X^*|$ and forms D computing tasks containing X^* and one of D words. Then every computing thread works with one task per process at a time. After performing the processing of a corresponding diapason according to the procedure outlined above, a computed average runtime for a diapason is sent to the control thread, which then takes all D such values and based on them computes a runtime estimation for X^*.

In the solving mode, our program takes as an input a set of partitioning variables. This set can be found in the estimating mode, or it can be constructed manually. For example, one can use the set of variables, which encode the initial state of a clock control register of a generator. Given a set of partitioning variables, the program solves all subproblems from a corresponding partitioning.

4 Computational Experiments

Using the algorithm, described in Section 3, we constructed guess-and-determine attacks on ASG-72, ASG-96, ASG-192, MASG-72 and MASG0-72. For each of them (excluding ASG-192) 20 cryptanalysis instances were constructed by randomly generating 20 initial states values. For each configuration the size of the corresponding keystream fragment is discussed below.

Hereinafter by *total solving time* we mean the time required to solve all subproblems from a partitioning. Of course, for the majority of satisfiable SAT instances we find a satisfying assignment faster. In particular, each considered SAT instance has exactly 1 satisfying assignment, so on average it usually takes twice less time. However, we compare our estimations with total solving time for all subproblems, because in fact it is this runtime that we estimate.

It should be noted, that we applied our program to construct a set of partitioning variables only for 1 instance out of 20 in every case (in particular, for the first one from a series). After this the constructed guess-and-determine attack was performed on all 20 instances from a series (including the one, which was used to find a set of partitioning variables). Our empirical evaluations and the results of computational experiments show that the SAT-based guess-and-determine attack for a particular cryptanalysis instance with fixed keystream fragment can be extended to cryptanalysis instances that have different keystream fragments. This fact allows us to say, that by finding a set of partitioning variables for a considered generator configuration, we construct a guess-and-determine attack not only on this particular instance, but on the generator itself.

All calculations were performed on the HPC-cluster "Academician V.M. Matrosov" [23]. Each computing node of this cluster is equipped with two 18-core CPUs Intel Xeon E5-2695 (36 CPU cores in total) and 128 gigabytes of RAM. In order to automatically construct guess-and-determine attacks, we used the following values of parameters for the procedure described in Section 3: $D = 1000, s = 1000000, K = 1000$.

For each generator configuration we compared the automatically constructed guess-and-determine attack with the trivial one, based on guessing the bits of the control register. We also compared it with two multithreaded CDCL solvers: `plingeling` and `treengeling` [3]. In the SAT competition 2016 they won the first two prizes in the parallel category [1]. We chose these standard solvers in order to check, if the high-ranked CDCL-based parallel SAT solvers can efficiently solve the considered problems directly, without constructing a guess-and-determine attack. It should be noted, that in the solving mode we employed exactly 1 computing node of the cluster in all cases, because the mentioned multithreaded solvers can work only within 1 workstation (i.e. they can not be launched on a HPC cluster using MPI). In the following subsection we will present the results of computational experiments for the considered generators configurations.

4.1 Additional Optimization: Choosing the Right Keystream Fragment Size

In the case of ASG-72, we first considered cryptanalysis problem for the keystream fragment length of 100 bits (this value is four times greater, than the length of the largest employed LFSR). We constructed 1 CNF encoding randomly formed cryptanalysis problem, and on this CNF we launched our parallel program (see Section 3) in the estimating mode for 2 hours to find a set of partitioning variables (as a subset of a set of 72 variables corresponding to the initial state). In this case (as well as in all other launches in estimating mode) our program used 10 computing nodes (360 CPU cores in total). As a result, for ASG-72 we found the set, consisting of 21 variables with runtime estimation equal to 32 seconds (if running on the same workstation). This set contains the following variables: 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 ($\mathrm{LFSR_X}$); 45 ($\mathrm{LFSR_X}$); 69 70 71 ($\mathrm{LFSR_Y}$). Here we use end-to-end numbering – variables of the control register $\mathrm{LFSR_C}$ have numbers from 1 to 23, for the controlled register $\mathrm{LFSR_X}$ – from 24 to 47, for the controlled register $\mathrm{LFSR_Y}$ – from 48 to 72. In Table 2 the set is depicted – here "+" denotes that the corresponding variable belongs to the set.

Table 2: The set of partitioning variables, found for ASG-72.

$\mathrm{LFSR_C}$	$- - - - + + + + + + + + + + + + + + + + + - -$
$\mathrm{LFSR_X}$	$- + - -$
$\mathrm{LFSR_Y}$	$- + + + -$

In [20] it was stated, that the average number of ASG preimages for any keystream fragment with length m is about $2^{3L-m}, m \leq 3L$, where L is the length of the controlled stop/go register. In [20] an ASG with the controlled registers of equal lengths was considered. In our case (with controlled registers of different lengths) as L we used a length of the largest controlled register. So, for ASG-72 $L = 25$, and about 75 bits of a given keystream fragment should be enough to get only 1 preimage. We decided to find the length of a keystream fragment, which yields the best runtime estimation for the considered cryptanalysis problem when the set of partitioning variables is fixed. We randomly constructed 7 more cryptanalysis instances for ASG-72 – with keystream fragment lengths from 72 to 96 with the step of 4. We then solved each of them using the constructed guess-and-determine attack. In order to compare the total solving time (in seconds), all subproblems from each partitioning were solved. It turned out, that on 72-bit fragment two preimages were found, on the other variants there was only 1 preimage. The obtained results are presented in Table 3. Along with the total solving time, for each variant we show the runtime estimation (calculated for the set of 21 variables, found on the 100-bit variant). We can conclude, that the total solving time agrees well with the estimation – the difference is about 18 $\%$. As it was mentioned before, in the estimating mode our program uses 10 computing nodes, while in the solving mode it uses 1 node. So, further all runtime estimations are given for 1 computing node.

According to the table, the fragments of sizes 72, 76, 80 and 84 bits provide the best efficiency. We chose the least value, for which only 1 preimage was found. So we used a fragment of size 76 in all our further experiments for ASG-72 (as well, as for MASG-72 and MASG0-72).

We did the similar calculations for ASG-96. We first considered the cryptanalysis problem for the keystream fragment length of size 132 (this value is four times greater

Table 3: The comparison of ASG-72 total solving time (in seconds) with different sizes of keystream fragment

Keystream length	72	76	80	84	88	92	96	100
Estimation	31	31	32	31	32	32	32	32
Total solving time	35	35	35	36	37	37	41	38
Preimage number	2	1	1	1	1	1	1	1

than the length of the largest employed LFSR). We constructed 1 CNF encoding randomly formed cryptanalysis problem, and on this CNF we launched our parallel program (see Section 3) in the estimating mode for 12 hours to find a set of partitioning variables (as a subset of a set of 96 variables corresponding to the initial state). As a result, we found the set, consisting of 30 variables with runtime estimation equal to 29 497 seconds (8 hours and 12 minutes). The found set consists of the following 30 variables: 2 3 4 5 6 12 13 14 15 17 19 20 22 23 25 26 27 29 30 ($\mathrm{LFSR_C}$); 56 57 58 59 60 62 ($\mathrm{LFSR_X}$); 89 91 92 93 94 ($\mathrm{LFSR_Y}$). This set is depicted in Table 4.

Table 4: The set of partitioning variables, found for ASG-96.

$\mathrm{LFSR_C}$	$-+++++-----++++-+-++-++-+++-++-$
$\mathrm{LFSR_X}$	$------------------------+++++-+-$
$\mathrm{LFSR_Y}$	$-------------------------+-++++--$

We then randomly constructed 8 more cryptanalysis instances for ASG-96 – with keystream fragment lengths from 100 to 128 with the step of 4. We solved each of 9 SAT instances using the constructed guess-and-determine attack. As in the case of ASG-72, all subproblems of each partitioning were solved. As a result for a 100-bit fragment 2 preimages were found, on the other variants there was only 1 preimage. The obtained results are presented in Table 5. We can conclude, that the total solving time agrees well with the estimation – the difference is about 7 %.

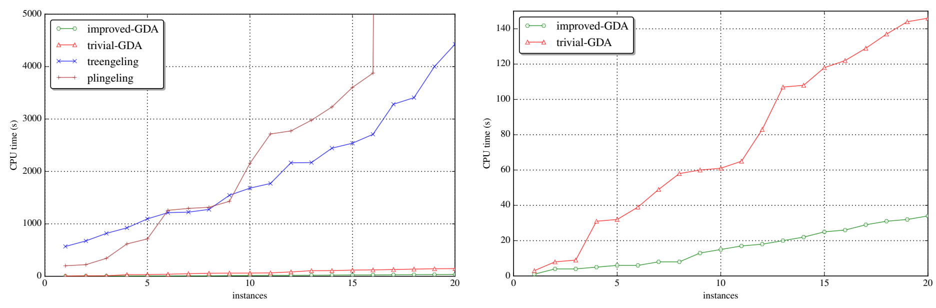
Table 5: The comparison of ASG-96 total solving time with different keystream fragment lengths

Keystream length	100	104	108	112	116	120	124	128	132
Total solving time	30 905	32 195	30 608	30 292	31 132	32 627	31 311	31 558	31 566
Estimation	31 671	33 137	31 571	30 946	31 931	33 763	32 427	31 971	29 497
Preimage number	2	1	1	1	1	1	1	1	1

According to the table, the fragment length of 112 bits provides the best efficiency. So we used the fragment of size 112 bits in our further experiments for ASG-96.

4.2 ASG-72

We used the found set of 21 variables (Table 2) to solve 20 cryptanalysis instances for ASG-72 (in each instance 76 bits of a keystream were known). The average time required to solve them turned out to be 16 seconds (we stopped processing of each partitioning when a correct initial state value was found). We can conclude, that this average solving time agrees well with the constructed estimation (remind that it is equal to 31 seconds). We also tried to solve all these instances by `plingeling`, `treengeling`, and by our program using the trivial set – formed by 23 variables corresponding to the initial state of the control register. The results of the comparison are depicted in Fig. 1. Here GDA is an abbreviation for “guess-and-determine attack”. The runtime was limited by 5000 seconds for every launch. On the figure we used the so-called cactus plots. On such plot the values are sorted in the ascending order. From these figures it follows, that `plingeling` and `treengeling` work much worse, than other two variants. It also follows, that the constructed guess-and-determine attack is better, than the trivial one.



(a) All considered attacks (b) The trivial and the improved GDA attacks

Fig. 1: Comparison of the considered SAT-based attacks on ASG-72

In Table 6 for each program the number of solved instances and the average time (in seconds) on solved instances are shown. Our improved guess-and-determine attack turned out to be about 4.7 times better, than the trivial one. We would like to emphasize, that in these experiments an estimation is considered as accurate, if it is about 2 times greater, than the average solving time. As it was said above, on average one needs to process half of a partitioning to find a solution.

4.3 ASG-96

We used the found set of 30 variables (it was described above) to solve 20 randomly constructed cryptanalysis instances for ASG-96 (in each instance 112 bits of keystream were known). The results of the comparison are depicted in Fig. 2. The runtime was limited by 12 hours (43 200 seconds) for every launch. As a result, `plingeling` and `treengeling` could not solve any instance in time, while both guess-and-determine

Table 6: The comparison of different SAT-based attacks on ASG-72.

Attack	Solved	Avg. time on solved	Estimation
`plingeling`	16	1 795	-
`treengeling`	20	1 997	-
Trivial GDA	20	75	121
Improved GDA	20	16	31

attacks solved all of them. Here in the trivial attack the set of 31 variables, corresponding to the control register, was used. From the figure it follows, that the constructed guess-and-determine attack is better, than the trivial one.

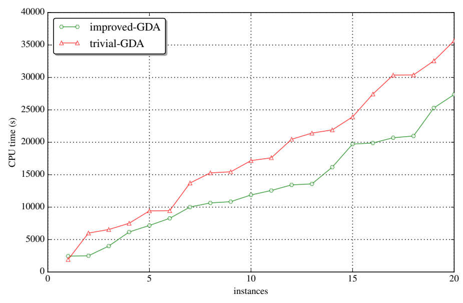


Fig. 2: Comparison of the trivial and the improved guess-and-determine attacks on ASG-96

In Table 7 for each SAT-based attack the number of solved instances and the average time (in seconds) on solved instances are shown. Our improved guess-and-determine attack turned out to be about 38 % better, than the trivial one.

Table 7: The comparison of different SAT-based attacks on ASG-96.

Attack	Solved	Avg. time on solved	Estimation
`plingeling`	0	-	-
`treengeling`	0	-	-
Trivial GDA	20	18 211	40 357
Improved GDA	20	13 181	30 946

4.4 MASG-72 and MASG0-72

In the cases of MASG-72 and MASG0-72 the keystream length of 76 was used (similar to ASG-72). For both generators our program was launched in the estimating mode for 2 hours on the cluster.

As a result for MASG-72 we found the set of partitioning variables consisting of 22 variables with runtime estimation equal to 71 seconds. The set consists of the following variables: 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 ($\mathrm{LFSR_C}$); 40 41 42 45 ($\mathrm{LFSR_X}$); 64 67 68 ($\mathrm{LFSR_Y}$). This set is also presented in Table 8.

Table 8: The set of partitioning variables, found for MASG-72.

$\mathrm{LFSR_C}$	− − − − − − + + + + + + + + + + + + + + + − −
$\mathrm{LFSR_X}$	− − − − − − − − − − − − − − − − + + + − − + − −
$\mathrm{LFSR_Y}$	− − − − − − − − − − − − − − − − + − − + + − − − −

We used this set to solve 20 randomly generated cryptanalysis instances for MASG-72. We also launched `plingeling`, `treengeling` and trivial guess-and-determine attack on them. In Table 9 for each SAT-based attack the number of solved instances and the average time (in seconds) on solved instances are shown. Our improved guess-and-determine attack turned out to be about 29 % better, than the trivial one. The results of experiments are also presented in Fig. 3.

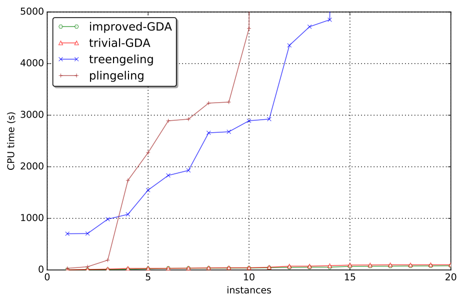
Table 9: The comparison of different SAT-based attacks on MASG-72.

Attack	Solved	Avg. time on solved	Estimation
`plingeling`	10	1 935	-
`treengeling`	14	2 418	-
Trivial GDA	20	58	89
Improved GDA	20	45	71

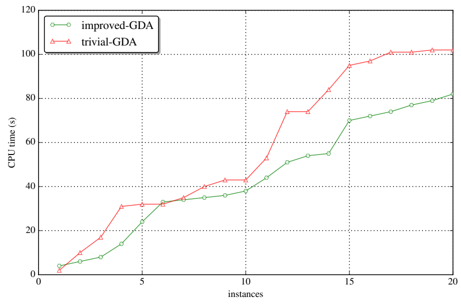
For MASG0-72 in the same conditions we found the set of 22 variables with runtime estimation of 74 seconds. The set consists of the following variables: 3 7 8 9 10 11 12 13 14 15 16 17 18 20 21 22 ($\mathrm{LFSR_C}$); 41 42 46 ($\mathrm{LFSR_X}$); 65 67 68 ($\mathrm{LFSR_Y}$). This set is also presented in Table 10.

Table 10: The set of partitioning variables, found for MASG0-72.

$\mathrm{LFSR_C}$	− − + − − − + + + + + + + + + + + + − + + +
$\mathrm{LFSR_X}$	− − − − − − − − − − − − − − − − − + + − − − + −
$\mathrm{LFSR_Y}$	− − − − − − − − − − − − − − − − − + − + + − − − −



(a) All considered attacks



(b) The trivial and the improved GDA attacks

Fig. 3: Comparison of the considered SAT-based attacks on MASG-72

In Table 11 for each SAT-based attack the number of solved instances and the average time (in seconds) on solved instances are shown. Our improved guess-and-determine attack turned out to be about 20 % better, than the trivial one. The results of experiments are also presented in Fig. 4.

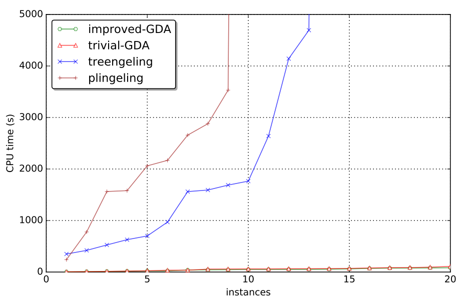
Table 11: The comparison of different SAT-based attacks on MASG-72.

Attack	Solved	Avg. time on solved	Estimation
`plingeling`	9	1 746	-
`treengeling`	13	1 667	-
Trivial GDA	20	55	92
Improved GDA	20	46	74

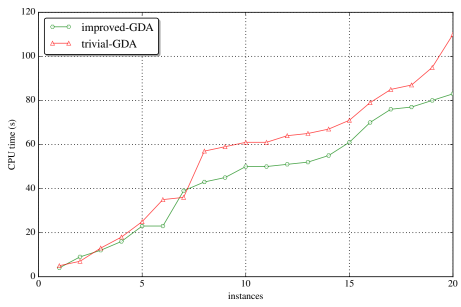
4.5 ASG-192

We launched our program for 24 hours in order to construct a guess-and-determine attack on ASG-192. As a result we found the set of 63 variables with the runtime estimation of $7.55e{+}13$ seconds. This set is presented in Table 12. The set consists of the following variables: 6 7 8 9 11 13 14 15 16 17 18 19 20 21 22 25 29 30 31 33 34 37 38 39 40 42 43 44 45 46 47 48 50 51 52 55 57 60 ($\mathrm{LFSR_C}$); 94 97 100 101 105 106 113 114 115 116 117 118 119 121 124 ($\mathrm{LFSR_X}$); 160 161 162 166 167 168 172 174 175 186 ($\mathrm{LFSR_Y}$).

We also used our program to estimate the trivial set (61 variables, corresponding to the control register). The corresponding estimation turned out to be $3.60e{+}14$ seconds, i.e. our attack is about 4.77 times better (by estimation). According to the obtained estimations, we decided not to perform the constructed improved attack in practice. This example shows, that using our approach for a given guess-and-determine attack, one can determine, if this attack can be performed in reasonable time in practice.



(a) All considered attacks



(b) The trivial and the improved GDA attacks

Fig. 4: Comparison of the considered SAT-based attacks on MASG0-72

Table 12: The set of partitioning variables, found for ASG-192.

$\mathrm{LFSR_C}$	− − − − − + + + + − + − + + + + + + + + + + − − + − − − + + + − + +
	− − + + + + − + + + + + + + − + + + − − + − + − − + −
$\mathrm{LFSR_X}$	− + −
	− + − − + + − − − + + − − − − − − + + + + + + + − + − − + −
$\mathrm{LFSR_Y}$	− −
	+ + + − − − + + + − − − + − + + − − − − − − − − − − + − − − − − −

5 Related work

The cryptographic resistance of the alternating step generator was analyzed in a number of papers [14–16, 20, 34]. The majority of these attacks implement different variants of correlation attacks on one or both controlled LFSRs [14, 15, 20]. A good overview of these attacks can be found in [20]. Hereinafter, assume that l is the length of the control LFSR, and m and n are the lengths of two controlled stop/go LFSRs. The attack with lowest time complexity was proposed in [19]: $O(m^2 \times 2^{2m/3})$, but it requires a lot of keystream ($O(2^{2m/3})$) and has a number of specific requirements regarding the keystream fragment. The same can be said about the attack from [20].

Since our attack does not have such requirements and uses a fragment of keystream of relatively small size, we compare it with the best attacks with similar properties. From this point of view the best attack among previously published results is the divide-and-conquer attack from [16], because in all configurations considered in our paper the control register is the smallest.

The attack from [16] has the time complexity of $O(min(m, n) \times 2^l)$, however, since we can not express the complexity of our attack analytically, it is necessary to get into details. In the attack from [16] we perform an exhaustive search over all possible variants of the initial value of control register (it corresponds to 2^l component in $O(\cdot)$). Intuitively, after guessing the value we derive a system of linear equations over bits corresponding to initial values of controlled registers and apply to it the so-called Linear Consistency Test (LCT) [33]. Essentially, LCT consists in solving the constructed system by means of Gaussian elimination or more state-of-the-art algorithm [2] and simultaneously checking if it is consistent. If the system yields a solution then with overwhelming probability it is the solution of our cryptanalysis problem. Now, for ASG-72 (for which $l = 23$, $m = 24$ and $n = 25$) the average runtime of our attack is 16 seconds on 36 cores, so about 576 seconds on one core of Intel Xeon E5 2695v4. It means, that in order for attack from [16] to be equally fast as our attack, it would need to be able to process about $2^{23}/576 = 14563.5$ states of control register per second on the same processor core. For ASG-96 the corresponding number of states per second is $2^{31}/(13181 \times 36) = 4525$. It is very hard to say what will be the performance of this attack if implemented properly without actually implementing it. We could not find ready implementations and implementing attack ourselves is out of the scope of the present research. Our guess is that if programmed properly it would be in the general vicinity of our approach. The important consideration here is that we present the results of a practical attack – it involves a lot of auxiliary work, such as actual decomposition of the problem into partitioning, sending commands to computing processes, processing the results, etc. Meanwhile the attack in [16] has only general outline.

We are not aware of any SAT-based and/or guess-and-determine attacks on ASG. Meanwhile, the corresponding approach works quite well in other areas of cryptanalysis. The overview of possible applications of SAT in algebraic cryptanalysis can be found in [2]. In [22] a SAT-based attack on a reduced variant of DES was proposed. In [24] there were studied several applications of SAT solvers to finding collisions of cryptographic hash functions.

In [10, 29, 30] using a relatively similar way to our approach, the Monte Carlo algorithms were applied to construct SAT-based guess-and-determine attacks on several keystream generators. However, we suggest a Monte Carlo-based algorithm with the new significantly improved functionality that takes into account several previously ignored issues, that greatly improve its accuracy.

Another relatively similar approach to cryptanalysis of ASG and other generators was proposed in [34]. In that paper it was suggested to use a straightforward backtracking algorithm to determine if a system of equations, specifying the cryptanalysis instance, can be solved. In a way, our work can be considered as a development in this direction, however we replace simple backtracking algorithm by the accumulated experience and methods from the area of SAT solving in the form of state-of-the-art CDCL algorithms.

As for MASG/MASG0, we have not found any papers considering the cryptanalysis of these generator modifications. Since we replace controlled LFSRs by NLFSRs, it means that the vast majority of correlation attacks or their variants, that work well for ASG, can not be applied to MASG/MASG0. The same can be said about the attack from [16]. Theoretically, the attack employing backtracking scheme proposed in [34], can be extended to considered modifications, but evaluating its complexity is a nontrivial task.

Overall, from our point of view, the method for constructing guess-and-determine attacks presented in our paper is interesting because despite relying on black-box optimization algorithms and algorithms for solving Boolean satisfiability problem (which is NP-hard) it shows competitive results on cryptanalysis of ASG/MASG/MASG0, and makes it possible to extend the paradigm of guess-and-determine attacks by considering non-trivial sets of bits to guess.

6 Conclusions

In the present paper, we proposed a new algorithm for constructing a SAT-based guess-and-determine attack on ASG and two its modifications (MASG and MASG0). Using this algorithm we obtained new guess-and-determine attacks that are better than the trivial ones (where we guess an initial state of the control clock register). The constructed attacks were used to perform in practice the cryptanalysis of the considered generators (with the initial states of size up to 96 bits).

Acknowledgments

Authors thank all anonymous reviewers for valuable comments.
The research was funded by Russian Science Foundation (project No. 16-11-10046) and by Council for Grants of the President of the Russian Federation (stipends no. SP-1184.2015.5 and SP-1829.2016.5).

References

1. Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT competition 2016: Recent developments. In: Singh, S.P., Markovitch, S. (eds.) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 5061–5063. AAAI Press (2017)
2. Bard, G.V.: Algebraic Cryptanalysis. Springer Publishing Company, Incorporated, 1st edn. (2009)
3. Biere, A.: Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016. In: Balyo, T., Heule, M., Järvisalo, M. (eds.) Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2016-1, pp. 44–45. University of Helsinki (2016)
4. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
5. Courtois, N.: Low-complexity key recovery attacks on GOST block cipher. Cryptologia 37(1), 1–10 (Jan 2013)
6. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
7. Dubrova, E.: A list of maximum period NLFSRs. IACR Cryptology ePrint Archive 2012, 166 (2012), informal publication
8. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003)
9. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. 89(4), 543–560 (2003)

10. Eibach, T., Pilz, E., Völkel, G.: Attacking Bivium using SAT solvers. In: Bning, H.K., Zhao, X. (eds.) Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing: 12-15 May 2008; Guangzhou, China. pp. 63–76 (2008)
11. Erkök, L., Matthews, J.: High assurance programming in Cryptol. In: Sheldon, F.T., Peterson, G., Krings, A.W., Abercrombie, R.K., Mili, A. (eds.) Fifth Cyber Security and Information Intelligence Research Workshop, CSIIRW '09, Knoxville, TN, USA, April 13-15, 2009. p. 60. ACM (2009)
12. Fishman, G.S.: Monte Carlo: Concepts, algorithms, and applications. Springer Series in Operations Research, Springer-Verlag, New York (1996)
13. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
14. Golic, J.D., Menicocci, R.: Edit distance correlation attack on the alternating step generator. In: Jr., B.S.K. (ed.) Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1294, pp. 499–512. Springer (1997)
15. Golic, J.D., Menicocci, R.: Correlation analysis of the alternating step generator. Des. Codes Cryptography 31(1), 51–74 (2004)
16. Günther, C.G.: Alternating Step Generators Controlled by De Bruijn Sequences, vol. 304, pp. 5–14. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
17. Hyvärinen, A.E.J.: Grid Based Propositional Satisfiability Solving. Ph.D. thesis, Aalto University (2011)
18. Janicic, P.: URSA: a system for uniform reduction to SAT. Logical Methods in Computer Science 8(3), 1–39 (2012)
19. Johansson, T.: Reduced complexity correlation attacks on two clock-controlled generators. In: Ohta, K., Pei, D. (eds.) Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1514, pp. 342–356. Springer (1998)
20. Khazaei, S., Fischer, S., Meier, W.: Reduced complexity attacks on the alternating step generator. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4876, pp. 1–16. Springer (2007)
21. Marques-Silva, J.P., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere et al. [4], pp. 131–153
22. Massacci, F., Marraro, L.: Logical cryptanalysis as a SAT problem. J. Autom. Reasoning 24(1/2), 165–203 (2000)
23. Irkutsk Supercomputer Center of SB RAS, http://hpc.icc.ru
24. Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing. pp. 102–115. SAT'06, Springer-Verlag, Berlin, Heidelberg (2006)
25. Maximum period NLFSRs, https://people.kth.se/ dubrova/nlfsr.html
26. Otpuschennikov, I., Semenov, A., Gribanova, I., Zaikin, O., Kochemazov, S.: Encoding cryptographic functions to SAT using TRANSALG system. In: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1594–1595. IOS Press (2016)
27. Prestwich, S.D.: CNF encodings. In: Biere et al. [4], pp. 75–97
28. Semenov, A.A., Zaikin, O.S.: Using Monte Carlo method for searching partitionings of hard variants of Boolean satisfiability problem. In: Malyshkin, V. (ed.) Proceedings of the 13th International Conference on Parallel Computing Technologies: 31 August - 4 September; Petrozavodsk, Russia. pp. 222–230 (2015)

29. Semenov, A., Zaikin, O.: 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)
30. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing: 30 June - 3 July, 2009; Swansea, UK. pp. 244–257 (2009)
31. Soos, M.: 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)
32. Wicik, R., Rachwalik, T.: Modified alternating step generators. IACR Cryptology ePrint Archive 2013, 728 (2013)
33. Zeng, K., Yang, C.H., Rao, T.R.N.: On the Linear Consistency Test (LCT) in Cryptanalysis with Applications, pp. 164–174. Springer New York, New York, NY (1990)
34. Zenner, E.: On the efficiency of the clock control guessing attack. In: Lee, P.J., Lim, C.H. (eds.) Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2587, pp. 200–212. Springer (2002)