Evolutionary Computation Techniques for Constructing SAT-Based Attacks in Algebraic Cryptanalysis

Artem Pavlenko[1](✉), Alexander Semenov[2], and Vladimir Ulyantsev[1]

[1] ITMO University, St. Petersburg, Russia
{alpavlenko,ulyantsev}@corp.ifmo.ru

[2] Matrosov Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
biclop.rambler@yandex.ru

Abstract. In this paper we present the results on applying evolutionary computation techniques to construction of several cryptographic attacks. In particular, SAT-based guess-and-determine attacks studied in the context of algebraic cryptanalysis. Each of these attacks is built upon some set of Boolean variables, thus it can be specified by a Boolean vector. We use two general evolutionary strategies to find an optimal vector: (1+1)-EA and GA. Based on these strategies parallel algorithms (based on modern SAT-solvers) for solving the problem of minimization of a special pseudo-Boolean function are implemented. This function is a fitness function used to evaluate the runtime of a guess-and-determine attack. We compare the efficiency of (1+1)-EA and GA with the algorithm from the Tabu search class, that was earlier used to solve related problems. Our GA-based solution showed the best results on a number of test instances, namely, cryptanalysis problems of several stream ciphers (cryptographic keystream generators).

Keywords: Algebraic cryptanalysis · Guess-and-determine attack · SAT · Evolutionary computation

1 Introduction

Algebraic Cryptanalysis (see [1]) is a way of breaking ciphers through solving systems of algebraic equations over finite fields. The corresponding attacks are called algebraic. Systems of algebraic equations constructed for strong ciphers are usually difficult for all known state-of-the-art algorithms. The resulting system of equations can be simplified by guessing the values of some of its variables. Then we can try all possible assignments of such variables, every time obtaining some simplified system. It might happen that the time spent by some algorithm on

The study was funded by a grant from the Russian Science Foundation (project No. 18-71-00150).

solving all such systems will be significantly smaller than the brute force attack time (e.g., if we test all possible keys of the cipher in question). An algebraic attack that uses some guessed bit set to simplify the system of cryptanalysis equations is called a guess-and-determine attack.

Over the last 20 years a large number of guess-and-determine attacks have been designed. In the vast majority of cases, a guess-and-determine attack is based on the analysis of the cipher features (see Fig. 1 with Trivium-Toy 64 cipher example). Such an analysis usually requires a lot of manual work. The recent papers [2,3] describe an automatic method for constructing guess-and-determine attacks. In the framework of this method, the weakened equations of cryptanalysis are solved using modern Boolean SATisfiability (SAT) solvers. Each guessed bit set is represented as a point in the Boolean hypercube. An arbitrary point is associated with the value of a special function that evaluates the complexity of the corresponding guess-and-determine attack. This function is a black-box pseudo-Boolean function. Thus, the construction of a guess-and-determine attack is reduced to the pseudo-Boolean black-box optimization problem. Various metaheuristic algorithms can be used to solve it and the papers [2,3] employ the simplest local search schemes, such as Simulated Annealing and Tabu Search. The main purpose of this paper is to demonstrate capabilities of evolutionary computation in application to the problem of automatic construction of guess-and-determine attacks in algebraic cryptanalysis. Below is the brief outline of the present work.

In Sect. 2 we introduce basic notations and facts of the presented paper. In particular, we briefly describe construction the known reduction of the problem of cryptographic attacks to the problem of pseudo-Boolean optimization. The corresponding pseudo-Boolean function Φ is not specified analytically and to minimize it metaheuristic algorithms related to local search methods were previously used [2,3]. In the present paper in order to solve this problem we apply two common strategies: (1+1)-Evolutionary Algorithm and Genetic Algorithm. The corresponding algorithms and techniques are described in Sect. 3. Section 4 contains results of computational experiments. In Sect. 5 we summarize the obtained results and outline future research.

2 Preliminaries

In this section, we give some auxiliary information from the Boolean functions theory and cryptanalysis.

2.1 Boolean Functions, Formulas and Boolean Satisfiability Problem (SAT)

Let $\{0,1\}^k$, $k \in \mathbf{N}$ denote a set of all binary words of length k ($\{0,1\}^0$ corresponds to an empty word). The words from $\{0,1\}^k$, $k \geq 1$ are sometimes called Boolean vectors of length k, whereas the set $\{0,1\}^k$, ($k \geq 1$) is referred to as a Boolean hypercube of dimension k. An arbitrary total function of the form

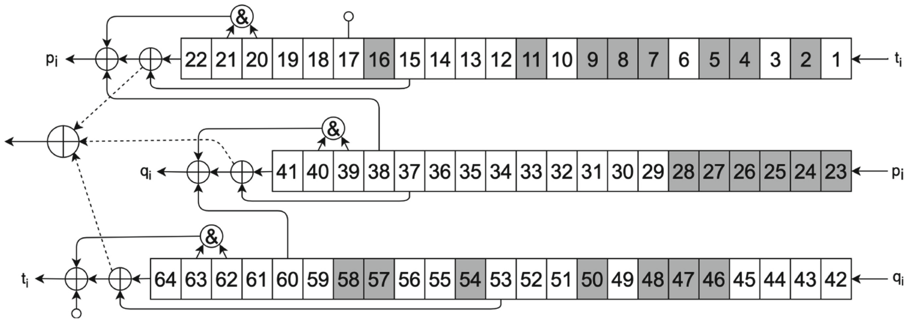

Fig. 1. Visualization of the key stream generator Trivium-Toy 64 with three registers. The guessed bit set, which is considered as an individual of evolutionary algorithms in the present work, is colored grey

$f : \{0,1\}^k \to \{0,1\}$ is called a Boolean function of arity k. An arbitrary function of the form $h : \{0,1\}^* \to \{0,1\}^*$, where

$$\{0,1\}^* = \bigcup_{k=0}^{\infty} \{0,1\}^k$$

is called a discrete function.

Boolean variables are the variables that take values from $\{0,1\}$. A Boolean formula with respect to k variables is an expression built by special rules over the alphabet comprising k Boolean variables $x_1, \ldots, x_k$ and special symbols called Boolean connectives. An arbitrary Boolean formula with respect to k variables defines a Boolean function of the kind $f_k : \{0,1\}^k \to \{0,1\}$. The set of Boolean connectives is called complete if they can be used to create any Boolean function of arbitrary arity. Such a set is called a complete system of connectives or a complete basis. The following set is a complete basis: $\{\wedge, \vee, \neg\}$, where $\wedge$ is conjunction, $\vee$ is disjunction and $\neg$ is negation.

Assume that F is an arbitrary Boolean formula, X is a set of variables that can be found in F, B is an arbitrary subset of X $(B \subseteq X)$. By $\{0,1\}^{|B|}$ we denote a set of all assignments of variables from B.

Let x be a Boolean variable. The formula that consists of a single variable or a negation is called a literal. Let x be an arbitrary Boolean variable. A pair of literals $(x, \neg x)$ is called a complementary pair. An arbitrary disjunction of different literals, which do not have any complementary pairs among them, is called a clause. An arbitrary conjunction of different clauses is called a Conjunctive Normal Form (CNF). If C is a CNF and $X = \{x_1, \ldots, x_n\}$ is a set of all Boolean variables that can be found in C, then we can say that C is a CNF over the set of variables X.

Let C be a CNF over X, $|X| = k$. Denote $f_C : \{0,1\}^k \to \{0,1\}$ a Boolean function defined by the CNF C. The CNF C is called satisfiable if there exists such $\alpha \in \{0,1\}^k$ (i.e. an assignment of variables from X), that $f_C(\alpha) = 1$ holds.

If such α exists, then α is called the satisfying assignment of C. If such α does not exist, then C is called unsatisfiable. The Boolean Satisfiability Problem, shortly denoted as SAT, has the following formulation: for any given CNF C, find if C is satisfiable. SAT is a classic NP-complete problem [4].

Recently, SAT has been targeted by algorithms that demonstrate a high efficiency for a wide class of applied problems [5]. Application of SAT solvers proved to be very successful in the following areas: symbolic verification, bioinformatics, combinatorics and Ramsey's theory, cryptanalysis.

2.2 Guess-and-Determine Attacks in Algebraic Cryptanalysis

As has been mentioned above, the algebraic cryptanalysis implies solution of systems of algebraic equations (usually over the field $GF(2)$) that describe some cipher.

Any cipher can be considered as a total discrete function of the kind

$$f : \{0,1\}^n \to \{0,1\}^m. \tag{1}$$

Then the cryptanalysis problem can be considered in the context of the problem of finding a preimage for some known value of the function (1): using the known $\gamma \in \mathit{Range}\ f$, $\mathit{Range}\ f \subseteq \{0,1\}^m$, find such $\alpha \in \{0,1\}^n$ that $f(\alpha) = \gamma$. We will call it the inversion problem for the function (1).

It is well-known (see, e.g. [1]) that the algorithm for calculating function (1) can be effectively described by a system of algebraic equations over the field $GF(2)$. Denote this system by $E(f)$. Roughly speaking, it describes the process of finding an output of the function f that corresponds to an arbitrary input. Let X be a set of all variables that can be found in $E(f)$. Denote by X^{in} and X^{out} sets of variables that were assigned to inputs and outputs of the function f, respectively.

Substitution of the values into the system $E(f)$ is determined in a standard way. It can be shown that if we substitute an arbitrary assignment $\alpha \in \{0,1\}^n$ for some variables from X^{in} into $E(f)$, we can derive assignments for all other variables from $E(f)$ by the following simple rules.

Substitute an arbitrary $\gamma \in \mathit{Range}\ f$ into $E(f)$, and let $E(f,\gamma)$ be the resulting system. If we manage to solve $E(f,\gamma)$ then we can extract such $\alpha \in \{0,1\}^n$ from the solution of $E(f,\gamma)$ that $f(\alpha) = \gamma$. However, this is a difficult problem for strong ciphers.

The simplest and most efficient way of solving $E(f,\gamma)$ is a sequential substitution of all possible $\alpha \in \{0,1\}^n$ into $E(f,\gamma)$. We denote the resulting system as $E(f,\alpha,\gamma)$. In line with the above, an arbitrary system $E(f,\alpha,\gamma)$ can be easily solved. If $f(\alpha) \neq \gamma$, then by applying the simple rules mentioned above, we derive a contradiction from $E(f,\alpha,\gamma)$. If $f(\alpha) = \gamma$, the contradiction does not occur and each variable from X will get a certain value. This scheme of trying all possible inputs corresponds to the method of cryptanalysis called the *brute force attack*.

For some ciphers and for functions of the form (1) that describe them, we can select a subset B of the set X with the following properties:

1. $|B| \ll n$;
2. the problem of finding a solution of $E(f, \gamma, \beta)$ or proving its inconsistency can be solved relatively quickly using some algorithm A; here β is an assignment for variables from B;
3. the total time required for finding $\alpha : f(\alpha) = \gamma$ by trying various $\beta \in \{0, 1\}^{|B|}$ is considerably smaller than the time of the brute force attack.

If the requirements listed above are fulfilled, then we can talk about a guess-and-determine attack based on the guessed bit set B.

Over the last 15–20 years a substantial number of different guess-and-determine attacks have been proposed. Some of them proved to be fatal for the corresponding ciphers. One of the simplest examples of the guess-and-determine attack is the attack on the A5/1 cipher described by Ross Andersen in 1994 [6]. For a long time, A5/1 served as a standard for encrypting the GSM traffic in cellular telephony. This cipher uses a 64-bit secret key. R. Anderson noted that if we choose in a certain way 53 bits of the internal state of the A5/1 registers and we know a certain number of bits of the key flow, then we can recover 11 unknown bits of the state of the registers by solving a trivial linear system over the field $GF(2)$ (in fact, we just have to solve a triangular system). Therefore, the Anderson attack is based on the guessed bit set $B : |B| = 53$, the role of the A algorithm is played by the algorithm for solving systems of linear equations. The form of these equations enables implementation of this attack on specialized computational architectures, for instance, in [7] the Anderson attack was performed on an FPGA device. The runtime of the corresponding attacks is up to 10 h for one cryptanalysis problem.

A series of works on algebraic cryptanalysis (see e.g., [1,8–10]) implies that algorithm A does not have to be polynomial. The problem of solving systems of the form $E(f, \gamma)$ can be efficiently reduced to combinatorial problems that are difficult in the worst case (NP-hard), but not difficult for most of their particular cases. One of the most computationally attractive problems here is SAT. Accordingly, any SAT-solving algorithm can play the role of A. In all the articles listed above, as well as in a number of other papers, the problem of finding a solution to an arbitrary system of the form $E(f, \gamma)$ was reduced to SAT for CNF $C(f, \gamma)$. To solve the resulting SAT instances, CDCL-based SAT solvers were used [11].

Some of those works describe guess-and-determine attacks where weakened cryptanalysis equations are solved using SAT-solvers. In [8] SAT-solvers were used to build a guess-and-determine attack on the truncated variants of the DES cipher, in [9] similar attacks targeted truncated variants of the GOST 28147-89 cipher. The paper [10] described a SAT-based guess-and-determine attack on the A5/1 cipher that used the guessed bit set $B : |B| = 31$.

2.3 Automatic Methods for Constructing Guess-and-Determine Attacks

In all of the above attacks, the guessed bit set was selected while analyzing the characteristics of the cipher in question. In other words, the corresponding attacks were constructed manually. As mentioned above, papers [2,3] propose approaches to the automatic construction of guess-and-determine attacks. To this end, special functions were introduced: they evaluate the efficiency of the attack and are calculated using a probabilistic experiment. Such functions are black-box pseudo-Boolean functions [12]. In [2,3] the problem of constructing a guess-and-determine attack with the lowest time was reduced to the pseudo-Boolean black-box optimization problem. Next, we briefly describe the corresponding techniques presented in [3].

Consider the problem of finding the preimage for the function $f : \{0,1\}^n \to \{0,1\}^m$, given by some efficient algorithm. Following the ideology of symbolic execution [4,13], we can build the CNF $C(f)$ that possesses some important properties using the algorithm that computes f (for more details see [3]). Let $\gamma \in \{0,1\}^m$ be an output of the function f and let B be an arbitrary set of variables in $C(f)$ that does not include variables from X^{out}. Consider B as a guessed bit set and denote by β an arbitrary assignment for variables from B. Let $C(f, \gamma, \beta)$ denote the CNF resulting from the substitution of the assignments γ and β into $C(f)$.

Let A be some algorithm for solving SAT. Fix some positive number t. For each $\beta \in \{0,1\}^{|B|}$ build a CNF $C(f, \gamma, \beta)$ and apply the algorithm A to it, limiting the algorithm runtime to t. If A fails to solve the corresponding SAT instance in time t, then we terminate A and move to the next β. If for some $\beta \in \{0,1\}^{|B|}$ the algorithm A finds a satisfying assignment for $C(f, \gamma, \beta)$ in time $\le t$, then thereby it will find $\alpha \in \{0,1\}^n : f(\alpha) = \gamma$.

It was shown in [3] that if α was randomly chosen in correspondence with a given on $\{0,1\}^n$ uniform distribution and if $\gamma : \gamma = f(\alpha)$ is known, then we can determine the probability of the following event: by applying the brute force strategy described above to the set $\{0,1\}^{|B|}$ we will find $\alpha : f(\alpha) = \gamma$. Denote this probability by ρ_B. It might be very small. Then we can repeat the strategy considering different outputs $\gamma_1, \ldots, \gamma_r$ of the function f (for many ciphers, it is enough to solve this problem at least for one such output to find the secret key). Then the probability of finding the preimage for at least one output $\gamma_1, \ldots, \gamma_r$ is

$$P_B^* = 1 - (1 - \rho_B)^r.$$

It is obvious that for a fixed $\rho_B > 0$ the probability P_B^* tends to 1 as r increases.

Note that the strategy described above takes time $2^s \cdot t$, $(s = |B|)$ to process one output γ_i, $i \in \{1, \ldots, r\}$, therefore, the upper bound for the total runtime of the corresponding attack is $2^s \cdot t \cdot r$. It was shown in [3] that if $r \approx \frac{3}{\rho_B}$, then $P_B^* > 0.95$, which means that the inversion of at least one of the r outputs of the function under consideration is an almost certain event. For this reason, it is desirable to find such B, for which the value

$$2^s \cdot t \cdot \frac{3}{\rho_B}$$

reaches the minimum on a set of possible alternatives of guessed bit sets. Unfortunately, if we want to calculate precisely the probability ρ_B, we have to search through the entire set $\{0,1\}^n$, which is infeasible. Therefore, the probability ρ_B takes the form of the expected value $E[\xi_B]$ of the random variable ξ_B of a special kind. Thus, it is required to minimize the function

$$2^s \cdot t \cdot \frac{3}{E[\xi_B]} \tag{2}$$

over all possible sets B. The variable ξ_B is derived from simple probability experiments (see details in [3]), whereas we use the Monte Carlo method [14] to evaluate $E[\xi_B]$.

The paper [3] considers the problem of constructing an efficient SAT-based guess-and-determine attack in the context of the minimization problem of a black-box pseudo-Boolean function with the vector χ_B as the function's input. The unit components of χ_B select the set B in the set of variables found in $C(f)$. Then a probability experiment for calculating the function (2) is set up for this set B. The obtained number is an estimate of the time for the guess-and-determine attack, where B is used as a guessed bit set. The goal is to find B with the smallest value of the estimate for the function (2).

In [2,3], pseudo-Boolean optimization problems were solved by simple metaheuristics such as Simulated Annealing and Tabu Search. The main results of this paper deal with application of evolutionary algorithms to these problems.

3 Applying Evolutionary Computations to Construction of Guess-and-Determine Attacks

In this section we formulate the problem of constructing an efficient guess-and-determine attack as the pseudo-Boolean optimization problem. We also describe the basic techniques of evolutionary computing that we employ.

3.1 Construction of an Efficient Guess-and-Determine Attack as the Problem of Pseudo-Boolean Function Minimization

We consider the cryptanalysis problem as the problem of inversion of the function (1). Construct the CNF $C(f)$. Let X be a set of variables found in $C(f)$. Let $W = X \setminus X^{\text{out}}$ and let B be an arbitrary subset of W. We can present an arbitrary B with the help of the Boolean vector χ_B of length $q = |W|$. Ones in χ_B will indicate those variables from W that were included into B.

Following the ideas from [3], define the pseudo-Boolean function

$$\Phi : \{0,1\}^q \rightarrow \mathbf{R}. \tag{3}$$

The arbitrary vector $\chi_B \in \{0,1\}^q$ is an input of the function (3). We use this vector to build the set B, $B \subseteq W$.

Recall that X^{in}, $|X^{\text{in}}| = n$ is the set formed by the variables from X corresponding to the input of the function (1). Define uniform distribution on $\{0,1\}^n$ and choose the corresponding Boolean vectors $\alpha_1, \ldots, \alpha_M$ ($\alpha_j \in \{0,1\}^n$, $j \in \{1, \ldots, M\}$). We will refer to this set of vectors as a random sample of size M. In view of the above, substitution of an arbitrary α_j into $C(f)$ and application of simple rules yields a derivation of assignments for all variables from X. Let γ_j be an assignment for variables from X^{out} obtained as a result of this derivation. By β_j we denote the assignment derived for variables from B, $j \in \{1, \ldots, M\}$.

Let us construct the CNFs $C(f, \beta_1, \gamma_1), \ldots, C(f, \beta_M, \gamma_M)$. Let each of these CNFs be an input of the SAT-solver A and set the solving time of each corresponding SAT instance as t. For each $j \in \{1, \ldots, M\}$ consider a random variable ξ_B^j, that evaluates to 1 if SAT instance for the CNF $C(f, \beta_j, \gamma_j)$ was solved in time $\leq t$, and $\xi_B^j = 0$ otherwise. The value of the function (3) at arbitrary point $\chi_B \in \{0,1\}^s$ is defined as follows:

$$\Phi(\chi_B) = 2^{|B|} \cdot t \cdot \frac{3M}{\sum_{j=1}^{M} \xi_B^j}, \tag{4}$$

where t and M are the parameters of this function.

In accordance with the Monte Carlo method, the bigger the size of the random sample M, the better approximation of (2) is given by the function (4).

Now consider the problem of finding the minimum of the function (4) over the Boolean hypercube $\{0,1\}^q$. To solve this problem we will use evolutionary algorithms.

3.2 Evolutionary Computation Techniques Used for Minimization of the Suggested Pseudo-Boolean Function

In this section we present some techniques that complement such strategies as the (1+1)-Evolutionary Algorithm [15] ((1+1)-EA) and one variant of the Genetic Algorithm (GA). We used these techniques to solve the minimization problem for functions of type (4) for several stream ciphers.

Basic Schemes of Evolutionary Computation. Algorithm 1, which is given below, is a common outline of an evolutionary algorithm for solving the minimization problem of an arbitrary function of type (4).

As an input, the algorithm takes the CNF $C(f)$, a random sample size M, the time limit t, the minimization strategy S, and the initial guessed bit set represented by the Boolean vector χ_{start}.

Algorithm 1. Evolutionary algorithm

```
Input: CNF formula $C(f)$, initial sample size $M$, time limit $t$, strategy $S$, initial
    guessed bits $\chi_{\text{start}}$
 1: $P \leftarrow$ InitPopulation($S, \chi_{\text{start}}$)
 2: $\langle \chi_{\text{best}}, v_{\text{best}} \rangle \leftarrow \langle \chi_{\text{start}}, \Phi(\chi_{\text{start}}, M, t) \rangle$
 3: $N_{\text{stag}} \leftarrow 0$                                    ▷ stagnation count
 4: while not StopCondition( ) do
 5:     $N_{\text{stag}} \leftarrow N_{\text{stag}} + 1$
 6:     for $\chi$ in $P$ do
 7:         $v \leftarrow \Phi(\chi, M, t)$
 8:         if $v < v_{\text{best}}$ then
 9:             $\langle \chi_{\text{best}}, v_{\text{best}} \rangle \leftarrow \langle \chi, v \rangle$
10:             $N_{\text{stag}} \leftarrow 0$
11:         end if
12:         $M \leftarrow$ SelectM()
13:     end for
14:     if $N_{\text{stag}} < N_{\text{stag}}^{\max}$ then
15:         $P \leftarrow$ GetNextPopulation($S, P$)
16:     else
17:         $P \leftarrow$ Restart($S, \chi_{\text{start}}$)
18:         $N_{\text{stag}} \leftarrow 0$
19:     end if
20: end while
21: return $\langle \chi_{\text{best}}, v_{\text{best}} \rangle$
```

Each Boolean vector χ from $\{0,1\}^q$ is considered as a population in the general concept of evolutionary computation. The value of the function Φ of type (4) at point χ is considered as a value of the fitness function for the corresponding population.

The function InitPopulation(S, χ_{start}) forms the initial population according to the input vector χ_{start} within the frames of the chosen evolutionary strategy S. The pair $\langle \chi_{\text{best}}, v_{\text{best}} \rangle$ corresponds to point in $\{0,1\}^q$ with the current Best Known Value (BKV) of the function Φ.

The value of Φ at each specific point $\chi_B \in \{0,1\}^q$ is computed using the scheme described in the previous section: construct the set B defined by χ, construct the random sample $\{\alpha_1, \ldots, \alpha_M\}$, find Boolean vectors $\gamma_1, \ldots, \gamma_M$ and $\beta_1, \ldots, \beta_M$ induced by this random sample, and finally build the SAT instances $C(f, \beta_1, \gamma_1), \ldots, C(f, \beta_M, \gamma_M)$. We use the SAT-solver A to solve all these SAT instances, each SAT-instance should be solved in time t. If the satisfying assignment is found or the time limit t is exceeded, the corresponding solving process is terminated. In the first case we set $\xi_B^j = 1$, in the second case $\xi_B^j = 0$ ($j = 1, \ldots, M$). Then we compute (4).

In Algorithm 1, the transition to the next population is performed via the function GetNextPopulation(S, P). The situation, when the algorithm fails to improve the current BKV during one iteration of the population change is called stagnation. If the number of stagnations exceeds some given limit $N_{\text{stag}}^{\max}$, then we perform a restart: a new starting population is formed using the function Restart(S, χ_{start}). The algorithm stops if the given general limit (for example, 12 h of operation of a computing cluster) is exceeded.

Techniques of Improvement. Note that the value of the observed random variable becomes known only after algorithm A has run for some time (not greater than t). On the other hand, the greater M is, the more accurate estimate of the time of the guess-and-determine attack is given by the value of function (4). Accordingly, the computation time required to find the value of function (4) is critically dependent on the random sample size M. Therefore, the greater M is, the more computation is required to find the value of (4). When $M > 500$ we have to use a computing cluster to minimize (4). As we will see further from computational experiments, reduction of the random sample size allows significantly increasing the algorithm's speed (the speed correlated with the number of hypercube points in which the objective function value was calculated during optimization).

Next, we describe a special technique that helps significantly increase the number of points that the algorithm processes during a fixed time limit (function SelectM). This technique is based on the dynamic change of the random sample size. However, before we proceed, we will focus on some important details.

Note that if we consider functions of type (4), we can significantly reduce the dimension of the search space by taking into account the features of the original cryptanalysis problem. It was shown in [2,3] that we can consider $\{0,1\}^n$, $n = |X^{\text{in}}|$ as a hypercube over which we perform minimization. In other words, the set B can be searched for as some subset in the set of all variables of the secret key. This can explained by the fact that we have already mentioned above: substitution of arbitrary assignments of all variables from X^{in} into $C(f)$ derives the assignment for all variables from $X \setminus X^{\text{in}}$. The SAT-solver A performs this derivation very quickly (essentially, the running time is linearly dependent on $|C(f)|$, because in this case A does not have to solve a combinatorial problem). This happens because X^{in} is a Strong Unit Propagation Backdoor Set (SUPBS) [16]. Therefore, $\chi_{\text{start}} = 1^n$ (the Boolean vector that consists of n ones) can always be chosen as a starting population of our minimization procedure. This point corresponds to the situation $B_{\text{start}} = X^{\text{in}}$. Any subsequent point will define some subset of X^{in}.

Thus, values of the random variable ξ_B in point $\chi_{\text{start}} = 1^n$ are computed very quickly. This property holds for several subsequent points and can be observed in Fig. 2, where we present the results of two runs of the algorithm described above.

Since the time spent on solving a single SAT instance in the Random Sample during the Fast Descent phase is already small, the spectrum of the function (4) on such random samples contains close values. This means that there is no principal need to use random samples of big size. The general idea is to change M while the algorithm runs, depending on the "homogeneity" of the sample in terms of the values of the observed random variable. After a whole range of experiments,

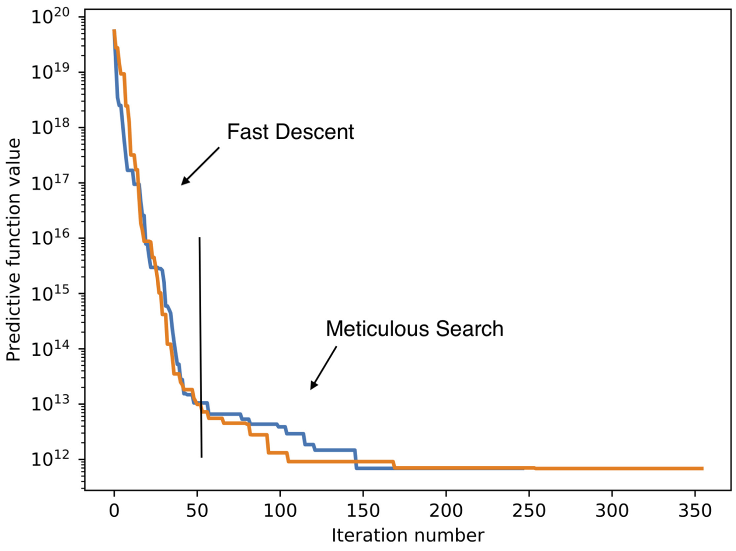

Fig. 2. Typical minimization of function (4). The process can be tentatively divided into two phases: Fast Descent and Meticulous Search.

we chose the following scheme: at the initial stage we use $M = 10$, then, passing through special checkpoints, M changes taking values $50, 100, 300, 500, 800$. The decision to change the sample size is made heuristically depending on the portion of $j \in \{1, \ldots, M\}$, for which $\xi_B^j = 1$ at the current value of M. General considerations here are as follows. Suppose that we consider two random samples R_1 and R_2 of size $M = 1000$ and that in 495 cases out of 1000 $\xi_B^j = 1$ for the sample R_1, and $\xi_B^j = 1$ for the sample R_2 in 510 cases out of 1000. The difference between these two values is ≈3%. Now let $M = 100$, and again we consider two random samples R_1, R_2. Suppose that $\xi_B^j = 1$ in 47 cases out of 100 for R_1 and $\xi_B^j = 1$ in 51 cases out of 100 for R_2. Here we have the difference ≈8%. Therefore, if in this situation we take $M = 100$, we will loose ≈5% of accuracy, but we will spend 10 times less resources on computing the value of our function. Thus, in the initial search stage, when BKV often improves at each iteration, minor loss of accuracy is not an issue. In final iterations, when the algorithm takes too much time to improve some BKV, the accuracy of $E[\xi_B]$ is very important, which is why it is reasonable to considerably increase the sample size in later stages of the algorithm for minimization of (4).

When traversing the hypercube we use hash tables to store the passed points. If the mutation results in a point where the value of the function (4) was calculated earlier, then, after finding this point in the hash table, recalculation is not performed.

Finally, here we describe a special variant of the Genetic Algorithm (GA), which we used for minimization of (4). This algorithm employs the technique known as elitism. Our algorithm works with populations that consist of N

individuals. As in the case of (1+1)-EA, the GA starts with the point $\chi_{\text{start}} = 1^n$ for which it constructs N replicas. Each such replica is an individual. Let us describe an arbitrary iteration of the algorithm. Let $P_{\text{curr}} = \{I_1, \ldots, I_N\}$ and P_{next}, $|P_{\text{next}}| = N$ be the current and the new populations, respectively. Choose from P_{curr} L individuals with the best values of the function (4) and move them to P_{next} (elitism). Let $v_1, \ldots, v_N$ be the values of (4) for all individuals from P_{curr} and consider the set of numbers $U = \{u_1, \ldots, u_N\}$, where $u_j = \frac{1}{v_j}$. We associate with any individual $I_j \in P_{\text{curr}}$ a number p_j defined as $p_j = \frac{u_j}{\sum_{i=1}^{N} u_i}$. For individuals from P_{curr}, consider the set U with the probability distribution $X_U = \{p_1, \ldots, p_N\}$. Choose randomly H individuals from P_{curr} with respect to the distribution X_U and apply to each of them the standard mutation, flipping each bit with the probability $\frac{1}{n}$. Move the resulting individuals to P_{new}. Finally, choose from P_{curr} individuals with respect to the distribution X_U and perform over them one of the known crossover operations. Assume that G individuals were obtained as the result of crossover and move them to P_{new}.

Thus the individuals with smaller value of function (4) are more likely to be selected for mutation or crossover. Finally, we require the following condition to be satisfied: $L + H + G = N$. In our experiments, we used a variant of the described algorithm with $N = 10$, $L = 2$, $H = G = 4$.

4 Computational Experiments

We applied the algorithms and techniques described in the previous section to the cryptanalysis problems of keystream generators.

Stream cipher or keystream generator (see e.g. [17]) is a discrete function of kind (1), such that $m \gg n$. This function, taking an arbitrary n-bit word as input, generates a word of length m that behaves as a random sequence. The generator's input is called the *secret key*, the output is called the *keystream*. The practical implication of the keystream generator is very quick generation of a long keystream for a given random key. Short keys can be exchanged by participants via, for example, nonsymmetrical cryptography that provides substantial guarantees for resistance, but is extremely slow.

Suppose that for a common secret key α the participants **A** and **B** simultaneously generate the same keystream $\delta = f(\alpha)$. If **A** wants to send **B** a secret message $x \in \{0,1\}^m$, it creates a ciphertext $x \oplus \delta$ (a componentwise addition modulo 2), which is sent to the public channel. Upon receiving the ciphertext and knowing δ, **B** easily finds x. It often happens that a malefactor (or adversary) **M** knows a fragment of the message x – this can be, for example, some known proprietary information which is ciphered together with the secret message. This very system vulnerability for a long time had been present in the traffic encryption protocol of cellular telephony (see [18]). In this situation, **M** can use a ciphertext to find the fragment δ, denoted as γ, and then try to find α as an preimage of γ for a mapping f. To do that, **M** has to solve a system of algebraic equations that describes construction of γ from α, or he has to solve a corresponding SAT instance. For a correct definition of α, the length of the keystream fragment γ should not be smaller than the length of α (usually it is slightly larger). Such cryptanalysis of a keystream fragment in order to find a secret key is called the *known plaintext scenario* [17].

In view of the above, consider the inversion problem for the function $f : \{0,1\}^n \rightarrow \{0,1\}^k$, $k \approx n$, where for the known $\gamma \in$ *Range* f one needs to find such $\alpha \in \{0,1\}^n$ that $f(\alpha) = \gamma$. We analyzed several functions that can be used as keystream generators:

1. Stream ciphers of the family Trivium (Trivium-Toy 64, Trivium-Toy 96, Bivium).
2. Alternating Step Generator (ASG).

Below we give a brief description of these ciphers.

The Trivium stream cipher was proposed in [19]. Being one of the eSTREAM project winners, this cipher attracts a lot of attention from cryptanalysts. In several papers (see e.g. [20,21]) it was shown that there are guess-and-determine attacks for Trivium which are more efficient than brute force attacks in the context of the so-called state recovery problem.

The Bivium cipher is a weakened version of Trivium (it uses only two original registers out of three). This cipher, as well as Trivium, was described in [19], where the author states that it is of mainly a research interest. A series of attacks on Bivium (including the algebraic ones that use SAT) can be found in [21–25].

Papers [26,27] propose a general approach to the construction of Trivium-like ciphers with a smaller total size of registers which preserves the algebraic properties of the original Trivium. Below we follow [26] and refer to this family as Trivium-Toy. In particular, by Trivium-Toy L we denote a cipher from this family, in which L is a total size of state that should be recovered. Hereinafter we consider state recovery attacks on Trivium-Toy 96 (this cipher is described in [26]) and Trivium-Toy 64.

The Alternating Step Generator (ASG) was suggested in [28] and actually it is a common design of keystream generators that can be used for construction of stream ciphers with different lengths of the secret key. The most attractive property of ASG is ease of implementation and high speed of keystream generation. ASG was targeted by several attacks, the analysis of which is presented in [29]. Some of these attacks employ a significant amount of keystream. The original paper [28] describes an attack that tried all possible ways to fill the control register. Essentially, this is a guess-and-determine attack which uses the so-called Linear Consistency Test [30] as an algorithm for solving weakened systems of cryptanalysis equations. The paper [31] presents a SAT-based guess-and-determine attack for ASG based on building SAT Partitionings of hard variants of SAT described in [2].

We implemented the strategies (1+1)-EA and GA presented above and Tabu Search [32] in the same manner as it is described in [2,3]. The created program is an MPI application that uses the SAT solver ROKK [33] as a computational core.

Each experiment was conducted on 180 cores of a computing cluster equipped with Intel Xeon E5-2695 processors. The duration of one experiment was 12 h. In the minimization process of functions of the kind (4) we used a random sample of size $M = 500$. For each point taken as a solution we recalculated the value of the objective function using a sample of size $M = 10^4$. The resulting value was taken as final.

As test problems we considered cryptanalysis problems for keystream generators from the Trivium and the ASG families. The results of computational experiments are presented in Table 1. For a specific generator G notation G n/m means that one needs to find an n-bit secret key by analyzing m bits of a keystream. SAT instances encoding the corresponding cryptanalysis problems were constructed using the TRANSALG system [34] which translates algorithms for calculating discrete functions of the kind (1) into SAT.

Let us note that GA-elitism showed the best results among the considered strategies for minimization of functions of the kind (4), overtaking competitors in three test problems out of six.

Table 1. Experimental results for six cryptographic algorithms. The leftmost column contains the name of a keystream generator for which the cryptanalysis problem is considered. The remaining table is divided into three sections corresponding to strategies (1+1)-EA, GA and Tabu Search. The first column of each section contains the power of the guessed bit set which corresponds to the best guess-and-determine attack found. The second column contains a time estimation (in seconds) for the attack

	(1+1)-EA		GA		Tabu Search	
	Power of guessed bit set	G&D attack (seconds)	Power of guessed bits set	G&D attack (seconds)	Power of guessed bits set	G&D attack (seconds)
Trivium-Toy 64/75	21	**3.19e+07**	22	5.36e+07	17	4.30e+07
Trivium-Toy 96/100	33	1.28e+13	40	**2.09e+12**	34	3.14e+12
Bivium 177/200	32	2.60e+12	39	**1.49e+12**	40	4.29e+12
ASG 72/76	9	5604.8	8	6155.19	8	**5601.33**
ASG 96/112	13	6.76e+06	16	**3.72e+06**	14	3.95e+06
ASG 192/200	47	2.27e+18	44	2.84e+17	47	**1.14e+16**

5 Conclusion and Future Work

In the presented paper we applied evolutionary computation strategies to construct guess-and-determine attacks arising in algebraic cryptanalysis. Each of these attacks implies solving a family of algebraic equations over a finite field (usually, $GF(2)$). Each equation from such family is the result of substituting the values of some bits into a general equation describing how a considered cipher works. The substituted bits are called the guessed bits. To solve the equations we use SAT solvers, similar to a number of other works. It is clear that different sets of guessed bits correspond to guess-and-determine attacks with different runtime. We consider the problem of finding a set of guessed bits that yields an attack with the lowest runtime as the problem of minimization of a pseudo-Boolean function. To solve it we used two general strategies employed in evolutionary computation: (1+1)-EA and GA. The proposed algorithms were implemented in the form of a parallel MPI program for a computing cluster. Using these algorithms we constructed guess-and-determine attacks for several keystream ciphers. The obtained attacks are better than the ones constructed using the Tabu Search algorithm, but not dramatically better. However, from our point of view, we have only touched the potential of evolutionary computation in this area. In the nearest future we plan to significantly extend the spectrum of employed evolutionary computation techniques in application to problems of constructing Algebraic cryptanalysis attacks.

Acknowledgements. The authors would like to thank Daniil Chivilikhin, Maxim Buzdalov and anonymous reviewers for useful comments.

References

1. Bard, G.V.: Algebraic Cryptanalysis. Springer, New York (2009). https://doi.org/10.1007/978-0-387-88757-9
2. 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), 554 (2016)
3. Semenov, A., Zaikin, O., Otpuschennikov, I., Kochemazov, S., Ignatiev, A.: On cryptographic attacks using backdoors for SAT. In: Proceedings of AAAI 2018, pp. 6641–6648 (2018)
4. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151–158. ACM (1971)
5. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
6. Anderson, R.: A5 (was: hacking digital phones). Newsgroup Communication (1994). http://yarchive.net/phone/gsmcipher.html
7. Gendrullis, T., Novotný, M., Rupp, A.: A real-world attack breaking A5/1 within hours. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 266–282. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_17
8. Courtois, N.T., Bard, G.V.: Algebraic cryptanalysis of the data encryption standard. In: Galbraith, S.D. (ed.) Cryptography and Coding 2007. LNCS, vol. 4887, pp. 152–169. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77272-9_10
9. Courtois, N.T., Gawinecki, J.A., Song, G.: Contradiction immunity and guess-then-determine attacks on GOST. Tatra Mountains Math. Publ. **53**, 65–79 (2012)
10. Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel logical cryptanalysis of the generator A5/1 in BNB-grid system. In: Malyshkin, V. (ed.) PaCT 2011. LNCS, vol. 6873, pp. 473–483. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23178-0_43

11. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Frontiers in Artificial Intelligence and Applications, vol. 85, pp. 131–153 (2009)
12. Boros, E., Hammer, P.L.: Pseudo-Boolean optimization. Discrete Appl. Math. **123**(1–3), 155–225 (2002)
13. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976)
14. Metropolis, N., Ulam, S.: The Monte Carlo method. J. Am. Stat. Assoc. **44**(247), 335–341 (1949)
15. Rudolph, G.: Convergence Properties of Evolutionary Algorithms. Verlag Dr. Kovac, Hamburg (1997)
16. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: IJCAI 2003, pp. 1173–1178 (2003)
17. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography, 1st edn. CRC Press Inc., Boca Raton (1996)
18. Nohl, K.: Attacking Phone Privacy, pp. 1–6. Black Hat, Las Vegas (2010)
19. Cannière, C.: Trivium: a stream cipher construction inspired by block cipher design principles. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 171–186. Springer, Heidelberg (2006). https://doi.org/10.1007/11836810_13
20. Raddum, H.: Cryptanalytic Results on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039 (2006)
21. Maximov, A., Biryukov, A.: Two trivial attacks on Trivium. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 36–55. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77360-3_3
22. Eibach, T., Pilz, E., Völkel, G.: Attacking Bivium using SAT solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 63–76. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79719-7_7
23. Eibach, T., Völkel, G., Pilz, E.: Optimising Gröbner bases on Bivium. Math. Comput. Sci. **3**(2), 159–172 (2010)
24. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_24
25. Huang, Z., Lin, D.: Attacking Bivium and Trivium with the characteristic set method. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 77–91. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21969-6_5
26. Castro Lechtaler, A., Cipriano, M., García, E., Liporace, J., Maiorano, A., Malvacio, E.: Model design for a reduced variant of a Trivium type stream cipher. J. Comput. Sci. Technol. **14**(01), 55–58 (2014)
27. Teo, S.G., Wong, K.K.H., Bartlett, H., Simpson, L., Dawson, E.: Algebraic analysis of Trivium-like ciphers. In: Australasian Information Security Conference (ACSW-AISC 2014), vol. 149, pp. 77–81. Australian Computer Society (2014)
28. Günther, C.G.: Alternating step generators controlled by De Bruijn sequences. In: Chaum, D., Price, W.L. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 5–14. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-39118-5_2
29. Khazaei, S., Fischer, S., Meier, W.: Reduced complexity attacks on the alternating step generator. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 1–16. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77360-3_1

30. Zeng, K., Yang, C.H., Rao, T.R.N.: On the Linear Consistency Test (LCT) in cryptanalysis with applications. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 164–174. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_16
31. Zaikin, O., Kochemazov, S.: An improved SAT-based guess-and-determine attack on the alternating step generator. In: Nguyen, P., Zhou, J. (eds.) ISC 2017. LNCS, vol. 10599, pp. 21–38. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69659-1_2
32. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
33. Yasumoto, T., Okuwaga, T.: ROKK 1.0.1. In: Belov, A., Diepold, D., Heule, M., Järvisalo, M. (eds.) SAT Competition 2014, p. 70 (2014)
34. Otpuschennikov, I., Semenov, A., Gribanova, I., Zaikin, O., Kochemazov, S.: Encoding cryptographic functions to SAT using TRANSALG system. In: ECAI 2016. FAIA, vol. 285, pp. 1594–1595 (2016)