Parallel Framework for Evolutionary Black-box Optimization with Application to Algebraic Cryptanalysis

A. Pavlenko*, A. Semenov**, V. Ulyantsev* and O. Zaikin**
* ITMO University, Saint-Petersburg, Russia
** ISDCT SB RAS, Irkutsk, Russia
alpavlenko@corp.ifmo.ru

***Abstract* - In this study, a framework aimed at implementing pseudo-Boolean optimization algorithms is presented. In this framework, called EvoGuess objective functions are calculated using the Monte Carlo random sampling method. The framework is used to construct algebraic attacks on several cryptographic keystream generators. An original cryptanalysis problem is first reduced to the Boolean satisfiability problem (SAT). The obtained problem can be simplified by assigning guessed bits values from some set of variables. A problem of finding a guessed bits set, which corresponds to a SAT-based attack with relatively low runtime estimation, is studied. This problem is considered as an optimization problem of a special black-box fitness function on the Boolean hypercube. To solve this optimization problem we develop EvoGuess, in which several evolutionary strategies are implemented. Also, the framework allows to use an arbitrary CDCL-based SAT solver. In computational experiments, SAT-based guess-and-determine attacks on several well-known keystream generators are constructed.**

Keywords - algebraic cryptanalysis; guess-and-determine attack; SAT; black-box optimization; evolutionary computations

I. Introduction

According to the algebraic cryptanalysis [1], a cryptanalysis problem is reduced to solving algebraic or Boolean equations. In the last two decades, such equations were usually solved by algorithms for solving the Boolean satisfiability problem (SAT) [1-9]. In almost all cases, SAT instances which encode cryptanalysis problems, are hard for all state-of-the-art SAT solving algorithms. Moreover, for any SAT solver it is problematic to estimate the runtime required to solve such an instance. The unpredictability of state-of-the-art SAT solvers on hard instances is known as 'Heavy-tailed phenomena', or 'Heavy-tailed behavior' [10]. Heavy-tailed behavior usually occurs when SAT solvers are used in cryptanalysis. Sometimes guessing one bit in a Boolean formula, which encodes a cryptanalysis problem, can decrease tenfold the runtime on the corresponding SAT instance.

In order to estimate the runtime, an original system of Boolean equations can be simplified by substituting some guessed information. Usually in the role of such information some bits of a secret key are used. It is always possible to simplify an original SAT instance in such a way, that the corresponding subinstances can be solved in reasonable time. However, the amount of such subinstances can be huge, while to perform an attack all or almost all of them must be solved. In practice, to show that a certain cipher is not resistant, it is not required to perform an attack – it is enough to show, that the complexity of the attack is less than that for all other known attacks. As a rule, such an attack should be significantly more effective than the brute-force attack. Attacks, in which cryptanalysis equations are simplified by substituting some guessed bits, are call *guess-and-determine attacks*.

In [9], an automatic method of constructing guess-and-determine attacks with good runtime estimations was proposed. According to this method, a problem of searching for guessed bits set with good runtime estimation is considered as a pseudo-Boolean black-box optimization problem. In [9], for this purpose simple metaheuristics, namely Tabu Search and Hill Climbing, were used. In the present paper, we describe the structure and abilities of a framework designed especially for constructing guess-and-determine attacks based on minimization of the function from [9]. Also, we compare our tool with several known competitors, which can be used for the same purpose.

The brief outline of the paper is as follows. In the next section, the problem of constructing guess-and-determine attacks is considered as a pseudo-Boolean black-box optimization problem. The section is mainly based on papers [9] and [11]. In Section 3, the framework called EvoGuess for solving the mentioned optimization problem is proposed. This framework uses SAT solvers to estimate the runtime of attacks. To minimize the objective function, the framework employs evolutionary algorithms. Section 4 describes the results of computational experiments. In particular, the developed framework EvoGuess was compared with the ALIAS tool [12]. In the last sections, the related work is discussed, and conclusions are drawn.

II. Constructing Guess-and-Determine Attacks as a Pseudo-Boolean Optimization Problem

Further we consider problems of searching for preimages of cryptographic functions and construct nontrivial runtime estimations for these problems. As it was

mentioned above, we reduce cryptanalysis problems to solving algebraic equations over a finite field (as a rule, over GF(2)). To solve such equations, we use state-of-the-art SAT solvers. This direction is known as algebraic cryptanalysis [1]. In the present paper, we study the cryptographic resistance of keystream generators to algebraic attacks.

Let us remind (see [13]), that a *keystream generator* - is a function of the type $\{0,1\}^n \to \{0,1\}^*$, which is determined everywhere in $\{0,1\}^n$. Here $\{0,1\}^n$ stands for the set of all binary strings (Boolean vectors) of length n, $\{0,1\}^*$ – is the set of all Boolean vectors of an arbitrary finite length. Any element from $\{0,1\}^n$ is called *secret key*. Any value of this function is called *keystream*, which is produced by the considered generator. If the length of the keystream is limited by some finite number m, then the following function is obtained:

$$g: \{0,1\}^n \to \{0,1\}^m. \qquad (1)$$

Values of function (1) are called *keystream fragments of length* m. The following problem is considered: given a known $\gamma \in Range\ g$ to find such $\alpha \in \{0,1\}^n$, that $g(\alpha) = \gamma$. This problem is further called *problem of searching preimages of function* g *given known image* (or *inversion problem* for g). As for keystream generators, this problem implies finding a secret key given a keystream fragment. An arbitrary keystream generator is usually defined by some fast algorithm. This feature is required for high speed of the encryption. However, the inversion of a function of the type (1) for $m \geq n$ must be computationally hard.

In algebraic cryptanalysis, a system of algebraic equations over GF(2) encodes some algorithm A_g, which in turn implements function (1). In this system, all steps of the algorithm A_g are connected with each other by variables with values from $\{0,1\}$ and operations $\wedge, \oplus$. The substitution of an arbitrary $\gamma \in Range\ g$ in this system results in a new system, that has a solution. Moreover, from this solution such $\alpha \in \{0,1\}^n$ can be effectively derived, that $g(\alpha) = \gamma$. The problem of solving an algebraic equation over $GF(2)$ can be effectively reduced to the Boolean satisfiability problem (SAT). Several software tools can be used for this purpose. In the present paper, a software tool Transalg [14] is used to construct SAT encodings of all studied keystream generators.

Thus, cryptanalysis of a keystream generator is considered as an inversion problem for a function of the type (1). Let's construct a template conjunctive normal form (template CNF, see [14]) C_g that implements A_g. Let X be a set of variables from C_g. There are two special subsets of this set: $X^{in}, |X^{in}| = n$ – variables, which encode an input of g, and $X^{out}, |X^{out}|$ – variables, which encode an output of g. For an arbitrary $\gamma \in Range\ g$, let $C_g(\gamma)$ be a CNF constructed from C_g by assigning values from γ to variables from X^{out}. The problem is to find an assignment, that satisfies $C_g(\gamma)$. Once it is found, then such $\alpha \in \{0,1\}^n$, that $g(\alpha) = \gamma$, can be easily found too.

As it has been already mentioned above, for resistant ciphers the problem of finding a satisfying assignment for $C_g(\gamma)$ is hard for all state-of-the-art SAT solvers. Moreover, for an arbitrary SAT solver it is even impossible to estimate how much time it would take to cope with $C_g(\gamma)$. However, one can simplify a SAT instance for $C_g(\gamma)$ by guessing values of some variables from $X \backslash X^{out}$. An arbitrary set of this kind $B, B \subseteq X \backslash X^{out}$ is further called *guessed bits set*. Guess-and-determine attack on a cipher implies solving SAT instances which are produced by varying some guessed bits set B [1]. Many guess-and-determine attacks, based on analysis of ciphers features, are known. In [9, 11] the problem of searching for guessed bits set B is considered as a minimization problem of pseudo-Boolean functions.

The mentioned functions make it possible to estimate the runtime of guess-and-determine attacks, based on certain sets B. The functions described in [9, 11] differ from each other. While the function from [11] is aimed mainly at proving the unsatisfiability of CNFs, the function from [9] suits better for proving the satisfiability of a satisfiable CNF in time, that does not exceed some limit. In [12, 15] the minimization of the function of the same type that in [11] was considered.

Let's briefly remind the main ideas of these functions. The set $B, B \subseteq X \backslash X^{out}$ is represented by a Boolean vector χ_B, in which 1 means that the corresponding variable is present in B, otherwise it is 0. Random variables of two types are connected with an arbitrary set B. In the first case, such random variable is denoted for CNF, that encodes the inversion problem for function g in a certain point $\gamma \in Range\ g$. Let $C_g(\gamma)$ be a CNF that encodes an inversion problem for the function g in the point γ. Let's connect a random variable ζ_B with an arbitrary set $B \subseteq X \backslash X^{out}$. In particular, $\beta \in \{0,1\}^{|B|}$, which is chosen in accordance to the uniform distribution, denotes to the time required to solve SAT for $C_g(\gamma, \beta)$. CNF $C_g(\gamma, \beta)$ is obtained from $C_g(\gamma)$ as a result of assigning β variables from B. A set of CNFs of the type $C_g(\gamma, \beta)$ for all $\beta \in \{0,1\}^{|B|}$ is called *SAT-partitioning* of CNF $C_g(\gamma)$. Suppose, that some determined complete SAT solving algorithm is used. Let $T\big(C_g(\gamma), B\big)$ be a summarized SAT solving time for all CNFs of the type $C_g(\gamma, \beta)$. It is easy to show (see [11]), that

$$T\big(C_g(\gamma), B\big) = 2^{|B|} \cdot \mathrm{E}[\zeta_B], \qquad (2)$$

where $\mathrm{E}[\zeta_B]$ – the expected value of the random variable ζ_B. The value $T\big(C_g(\gamma), B\big)$ can be considered as a runtime of a guess-and-determine attack, which is aimed at finding a secret key given a keystream fragment. To estimate $\mathrm{E}[\zeta_B]$, the Monte Carlo method [16] can be used. In particular, $\mathrm{E}[\zeta_B]$ can be estimated by sample mean using a random sample of large size M:

$$\Psi(\chi_B) = 2^{|B|} \cdot \frac{1}{M} \cdot \sum_{j=1}^{M} \zeta_B^j. \qquad (3)$$

In (3), $\zeta_B^j, j \in \{1, \ldots, M\}$ is a value of ζ_B, observed in M random experiments. According to the Monte Carlo method, if $\mathrm{E}[\zeta_B]$ and $Var(\zeta_B)$ are finite, then the

following holds true: the larger M, the better (3) approximates (2). The function (3) is a pseudo Boolean fitness function from [11,12,15].

The fitness function from [9], that estimates the runtime of guess-and-determine attacks based on the guessed bits set B, looks as follows:

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

In (4), ξ_B^j, $j \in \{1, \dots, M\}$ is the values of an observed random variable ξ, which is denoted as follows: $\xi = 1$, if for a CNF, constructed from C_g by a random experiment, a satisfying assignment is found by some SAT solving algorithm in time, that does not exceed t. Otherwise, $\xi = 0$. Thus, M random experiments are performed for each B. Each such experiment requires $\leq t$ time. The value $\frac{1}{M}\sum_{j=1}^{M} \xi_M^j$ is a Monte Carlo estimation [16] of the expected value of ξ.

In [9, 11, 12, 15] to minimize functions (3) and (4), Tabu Search and Hill Climbing metaheuristics were used. In the next section, a new framework is described, in which evolutionary and genetic algorithm are employed for this purpose.

III. Framework for minimizing fitness functions, which are aimed at estimating the runtime of algebraic attacks on keystream generators

For minimizing fitness functions, which were described in the previous section, a software framework EvoGuess was developed. This framework is designed for operating on a computing cluster. It was implemented in Python 2.7. The scheme of the framework is shown in Fig. 1.

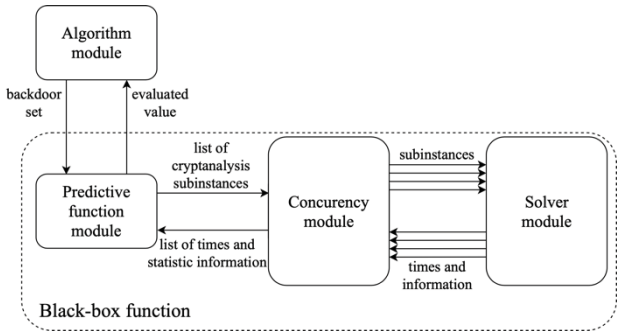

Fig. 1. The EvoGuess framework scheme

The framework has a module structure. Each module is briefly described below.

Algorithm module. This module is the entry point of the framework. It manages all other modules. The minimization of fitness functions is performed in the form of iterative procedures, which are based on evolutionary and genetic algorithms. At each iteration, a set of points is constructed, where some backdoor set corresponds to each such point. These points form a population, that consists of individuals (in accordance with the terminology from the evolutionary computation, see [17]). At each point, the value of the considered fitness function is calculated. This is done by *Predictive function module.* To form the next population, various metaheuristics are used: simple (1+1)-Evolutionary algorithm, a special version of the Genetic algorithm, etc. All processed points are stored to exclude the redundant calculations. If, when trying to exit a certain point, the permissible number of stagnations is exceeded, then a restart is done. The algorithm module has the following parameters: base algorithm; mutation function; crossover function (for the genetic algorithm); the permissible number of stagnations; search stop criterion.

Predictive function module. This module is aimed at estimating the runtime of guess-and-determine attacks based on given backdoor set. To calculate the fitness function, the Monte Carlo method is used. For each backdoor set a random sample is generated, which consists of simplified instances of the original SAT instance. This sample is given to the concurrency module, which solves all the subinstances using given computational resources. The corresponding backdoor set is estimated based on the obtained results and the chosen fitness function. The framework maintains fitness functions (3) and (4), described in the previous section. Most calculations were performed for function (4). In this case, an additional heuristic is used to obtain the final estimation. First, all subinstances from the sample are solved with the time limit t. Then such time limit $< t$ is chosen, that the value of function (4) is smaller. Note, that all subinstances, which were not solved in time t, are considered as unsolved.

Concurrency module. This module parallelizes all the computations to perform them on a computing cluster. In particular, subinstances are uniformly distributed between free threads. The number of such threads can be changed. The module was implemented using the standard Python library Pool.

Solver module. This module solves subinstances by the chosen SAT solver. The communication with a SAT solver is organized by a special wrapping, which uses console commands for launching this solver with given parameters. Also, this wrapping is able to output the results: found backdoor set; execution time; statistics information. At the moment, the framework maintains the following SAT solvers: ROKK, Lingeling, Plingeling, Treengeling, MiniSat, CryptoMiniSat, PaInleSS.

IV. Computational experiments

The framework described in the previous section was applied to constructing guess-and-determine attacks on several keystream generators. In particular, the following stream ciphers were considered: Grain, Trivium, Mickey. They were the winners of the eSTREAM competition. For minimizing fitness function, we used the evolution algorithm with strategy (1+1) ((1+1-EA) and a special version of the genetic algorithm know as Elitism (GA) with population size $N = 10$. In our experiments with metaheuristics (1+1)-EA we used: standard bit mutation, permissible number of stagnations equaled 300 and the 12-hour search wall-clock time budget. For GA we also used uniform crossover with probability $p = 0.2$. The sample size for one individual was $M = 1000$. For a SAT

solver, we used the ROKK solver. The time limit imposed on the SAT solver was set to 10 seconds.

All the computational experiments were held on the computing cluster 'Academician V.M. Matrosov' of Irkutsk Supercomputer Center SB RAS [18]. For each run we used five identical nodes of a cluster. Each node contained one Intel®Xeon®E5-2695 v4 processor, clocked at 2.1 GHz, having 18 physical cores and 36 threads.

Two modifications of Grain were considered: Grain v0 and Grain v1. There exist state recovery attacks on Grain v0, which are significantly more efficient than the brute force over the key space $\{0,1\}^{80}$ [19]. The similar attacks on Grain v1 are unknown. Thus, it is expected, that the resistance of Grain v0 and Grain v1 to algebraic attacks, based on minimizing pseudo-Boolean functions described in Section 2, are different. This hypothesis has been confirmed during the computational experiments. In Table 1, the corresponding runtime estimations calculated using function (4) are presented. 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.

Table 1. The runtime estimations (in seconds) of guess-and-determine attacks on Grain v0 and Grain v1, obtained as a result of minimizing function (4).

	(1+1)-EA		GA	
	|B|	Attack time	|B|	Attack time
Grain v0 160/160	73	4.419e+23	73	5.326e+23
Grain v1 160/160	104	4.712e+32	102	7.226e+31

According to the results, the runtime estimation of the best attack on Grain v1 is 10^8 times higher than that for the best attack on Grain v0.

In Table 2, the proposed framework is compared with its competitor, ALIAS [12] on Trivium, Grain and Mickey. ALIAS minimizes a function of the type (3), while the developed framework is able to minimize both (3) and (4). The results of ALIAS were taken from [15]. It should be noticed that for now our framework isn't outperforming ALIAS tool on function (3) minimization. We think that it is connected with the simplicity of our algorithms which are not boosted for cryptanalysis. We are currently implementing advanced evolutionary techniques in EvoGuess. Also, ALIAS is not supporting function (4) minimization.

Table 3 shows the best-constructed guesses bit sets for each considered cipher applying (1+1)-EA and GA metaheuristics. Also for Grain v1, Trivium and Mickey we presented guessed bit sets which found using the function of the type (3).

V. Related work

As it was mentioned above, the algebraic cryptanalysis consists in solving systems of algebraic equations, which describe a considered cipher. There exist several types of algebraic cryptanalysis: Buchberger algorithm [20] and its further modifications (see, e.g., [21]); ElimLin algorithm [2, 22]; characteristic set method [23], etc. Also, SAT solvers can be used for solving algebraic equations. In the book [1] it is noted, that SAT is one of the most effective and promising technique of the algebraic cryptanalysis.
Guess-and-determine attacks forms one of the largest classes of cryptographic attacks. They are based on the fact, that the considered cryptographic function can be inversed significantly faster than by a brute-force attack. Guess-and-determine attacks are based on the usage of a guessed bits set. Variables values from such a set are substituted to a system of equations and, as a result, simplify it. Such simplified systems can be solved by various algorithms in relatively small time. In [9, 11] it was suggested to consider problems of constructing guess-and-determine attacks in the form of an optimization problem for special pseudo-Boolean black-box functions. The software implementation of the corresponding procedures results in tools which are able to automatically construct guess-and-determine attacks. This, in turn, makes it possible to estimate the resistance of wide class of ciphers. Of course, such estimations are rough, but they are usually better than all known state-of-the-art estimations. PDSAT [11] and ALIAS [12] are examples of such tools. As it was mentioned above, in these tools Tabu Search and Hill Climbing algorithms were used to minimize functions which construct runtime estimations for guess-and-determine attacks. In the present paper, a framework is

Table 2. The comparison of runtime estimations (in seconds) of guess-and-determine attacks, constructed by ALIAS and the developed framework on winners of the eSTREAM competition. Results for ALIAS were taken from [15].

	ALIAS function (3)		EvoGuess (1+1)-EA function (3)		EvoGuess (1+1)-EA function (4)		EvoGuess GA function (4)	
	|B|	Attack time (s)	|B|	Attack time (s)	|B|	Attack time (s)	|B|	Attack time (s)
Grain v1 160/160	109	4.04e+30	100	7.51e+30	104	4.71e+32	102	7.23e+31
Trivium 288/300	144	1.40e+41	143	3.51e+43	136	1.52e+43	146	5.08e+43
Mickey 200/250	158	1.56e+48	169	1.77e+51	159	9.73e+50	152	8.18e+50

Table 3. Differences between constructed guessed bit sets by EvoGuess. Best times are typeset in bold

Strategy	|B|	Attack time (s)	Guessed bit set
Cipher: Grain v0 160/160			
(1+1)-EA	73	**4.42e+23**	2 4 7 13 14 16 17 18 20 21 22 25 33 38 42 49 50 51 57 60 62 63 65 70 72 74 77 80 82 84 87 88 93 95 96 [103..106] 109 114 116 [118..122] [124..135] 138 141 142 143 146 147 149 151 152 153 155 157 159 160
GA	73	5.33e+23	5 6 9 16 19 22 27 32 35 38 39 42 45 50 55 56 59 60 [66..72] 83 84 85 87 89 93 94 96 [98..110] 114 116 117 120 123 126 127 [129..133] [137..141] [143..148] 151 152 153 157
Cipher: Grain v1 160/160			
(1+1)-EA	100	**7.51e+30**	[2..6] 8 10 12 13 14 [16..19] 21 [23..27] 29 30 31 [35..43] 45 46 48 50 51 [53..58] [60..71] 75 76 78 79 80 85 86 87 90 91 94 95 97 99 101 [104..108] [115..119] 122 123 127 128 130 132 133 135 136 139 142 145 [147..150] 153 154 157 158
(1+1)-EA	104	4.71e+32	[2..5] 7 8 [10..13] 16 18 19 20 [22..29] 32 33 34 36 37 39 41 42 43 [47..50] 52 [54..57] [59..63] 65 66 68 70 72 74 77 79 80 81 [84..87] 89 91 92 94 96 99 100 102 103 [105..109] [111..114] 117 120 122 123 125 127 128 130 131 133 135 [137..141] 143 145 147 150 [153..156] 158 159 160
GA	102	7.23e+31	[4..11] 13 [16..23] 26 [28..33] [35..38] 40 41 43 44 47 50 [52..62] 64 66 67 [69..74] [76..81] 88 91 92 [94..98] [100..104] 107 108 110 111 114 115 116 122 123 125 128 129 131 132 135 138 139 140 142 144 148 149 [151..154] 157 158 159
Cipher: Trivium 288/300			
(1+1)-EA	143	3.51e+43	1 2 5 12 14 19 20 22 25 28 29 32 34 35 37 39 40 41 43 44 [49..52] 55 56 58 59 61 62 64 65 67 70 73 74 77 78 79 [82..86] 89 90 92 95 96 100 103 105 106 107 112 113 117 [119..122] 127 128 130 131 137 139 140 142 143 146 147 149 152 154 158 159 [163..166] 168 171 173 176 178 179 182 185 187 189 [191..197] 199 202 [206..209] 211 212 213 217 223 225 229 230 231 236 238 239 242 244 247 248 249 251 253 254 257 259 260 262 264 265 266 268 269 271 272 275 280 281 283 284 285 287 288
(1+1)-EA	136	**1.52e+43**	2 3 5 6 9 11 [15..21] 24 30 31 35 37 38 40 43 44 45 47 51 55 56 58 60 63 64 66 68 71 74 [77..80] 84 86 88 90 91 92 98 100 103 105 107 112 113 116 119 [123..126] 128 131 134 139 140 [142..147] 149 151 153 154 157 158 159 161 162 164 166 169 170 171 175 181 182 183 [186..190] 193 197 199 200 203 204 210 211 [213..216] 220 226 229 230 233 234 235 238 241 [243..247] 249 250 251 253 255 256 257 262 269 273 276 277 280 281 284 285 286 288
GA	146	5.08e+43	6 8 11 12 17 18 20 21 25 27 29 30 31 34 42 44 46 47 49 51 54 57 59 60 61 65 67 69 70 72 74 76 [78..82] [84..91] 93 94 97 99 105 107 109 111 113 114 117 119 120 122 [126..130] 134 135 137 139 140 145 147 149 151 152 154 156 157 161 163 164 167 168 169 172 176 178 183 185 187 188 191 193 [198..201] 204 205 [207..211] 214 215 217 220 221 223 225 226 229 230 231 234 235 236 241 242 248 249 250 255 257 259 261 265 267 270 273 278 281 283 284 286 288
Cipher: Mickey 200/250			
(1+1)-EA	169	1.77e+51	[1..11] 13 14 15 [17..27] 29 30 31 [34..40] 42 [44..53] [55..63] 65 66 67 [72..79] 81 [83..87] [89..101] 103 104 105 107 108 110 111 [113..118] [120..131] [133..156] [158..163] [165..171] 174 [176..184] [186..191] 193 [196..200]
(1+1)-EA	159	9.73e+50	3 4 5 7 [9..24] 26 28 29 30 32 33 [35..40] 43 [45..48] 50 51 52 [55..72] 74 75 77 78 79 81 [85..98] 101 103 104 106 108 110 112 [114..121] [123..134] 138 [140..146] 148 149 [151..157] 159 160 [162..178] [180..184] 186 187 189 190 [192..200]
GA	152	**8.18e+50**	2 3 5 [7..17] 19 20 [22..29] 32 35 [37..54] 56 [58..63] [66..71] 73 74 75 77 78 80 81 [86..93] 95 96 97 [101..106] 109 110 [113..116] [119..129] 131 132 133 [135..142] [144..148] [150..158] 160 162 163 [165..174] 177 [180..186] 188 [192..195] 197 198 200

proposed that uses evolutionary and genetic algorithms for the same purpose.

VI. Conlusion and future work

In the present paper, a new framework for algebraic cryptanalysis was proposed. This framework is designed

for operating on parallel computing systems. In particular, the framework constructs guess-and-determine attacks on symmetric ciphers. It is done by minimizing pseudo-Boolean functions via evolutionary and genetic algorithms on a computing cluster. Using the framework, runtime estimations of guess-and-determine attacks on several stream ciphers were calculated. These ciphers were winners of the eSTREAM competition.

In the nearest future, we are planning to significantly extend the framework's spectrum of pseudo-Boolean optimization algorithms. Also, we are planning to improve the search for guessed bits via tuning parameters of the used SAT solvers.

Acknowledgment

The study was funded by a grant from the Russian Science Foundation (project No 18-71-00150). The authors would like to thank the anonymous reviewer for useful comments.

References

[1] Bard, G.V.: Algebraic Cryptanalysis. Springer (2009)
[2] Courtois, N.T., Bard, G.V.: Algebraic Cryptanalysis of the Data Encryption Standard. In: Cryptography and Coding. LNCS, vol. 4887, pp. 152–169 (2007)
[3] Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: SAT'06. LNCS, vol. 4121, pp. 102–115 (2006)
[4] De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion Attacks on Secure Hash Functions Using SAT Solvers. In: SAT'07. LNCS, vol. 4501, pp. 377–382 (2007)
[5] Soos, M., Nohl, K., Castelluccia, C.: Extending SAT Solvers to Cryptographic Problems. In: SAT 2009. LNCS, vol. 5584, pp. 244–257 (2009)
[6] Semenov A., Zaikin O., Bespalov D., Posypkin M.: Parallel Logical Cryptanalysis of the Generator A5/1 in BNB-Grid System. In: International Conference on Parallel Computing Technologies. pp. 473-483. Springer (2011)
[7] Courtois, N.T., Gawinecki, J.A., Song, G.: Contradiction Immunity and Guess-then-Determine Attacks on GOST. Tatra Mountains Mathematical Publications, vol. 53, pp. 65–79 (2012)
[8] Yeo, S.L., Li, Z., Khoo, K., Low, Y.B.: An Enhanced Binary Characteristic Set Algorithm and Its Applications to Algebraic Cryptanalysis. In: Applied Cryptography and Network Security. LNCS, vol. 10355, pp. 518–536 (2017)
[9] Semenov A., Zaikin O., Otpuschennikov I., Kochemazov S., Ignatiev A. On Cryptographic Attacks Using Backdoors for SAT. In: Proc. of AAAI 2018. pp. 6641-6648 (2018)
[10] Gomes C., Sabharwal A. Exploiting Runtime Variation in Complete Solvers. In Biere A., Heule M., van Maaren H., Walsh T. edt. Handbook on Satisfiability, Ch. 9, pp. 271-288.
[11] 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)
[12] Kochemazov S., Zaikin O. ALIAS: A Modular Tool for Finding Backdoors for SAT. In SAT 2018. LNCS, vol. 10929, pp. 419-427 (2018).

[13] Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1st edn. (1996)
[14] 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)
[15] Zaikin O., Kochemazov S. Pseudo-Boolean Black-Box Optimization Methods in the Context of Divide-and-Conquer Approach to Solving Hard SAT Instances. In DEStech Transactions on Computer Science and Engineering, pp. 76-87 (2018)
[16] Metropolis N., Ulam S.: The Monte Carlo Method. J. Amer. Statistical Assoc. **44**(247), pp. 335-341 (1949)
[17] Luke, S.: Essential of metaheuristics. 2nd Edition. George Mason University (2015)
[18] Irkutsk Supercomputing Center, Siberian Branch of the Russian Academy of Sciences. URL: http://hpc.icc.ru. Cited February 15, 2019.
[19] Berbain C., Gilbert H., Maximov A.: Cryptanalysis of Grain. In: International Workshop on Fast Software Encryption. Vol. 4047. pp. 15-29. Springer (2006)
[20] Buchberger B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal, PhD thesis, Innsbruck (1965)
[21] Faugére J. C.: A New Efficient Algorithm for Computing Grobner Bases (F4). Journal of Pure and Applied Algebra. Vol. 139. pp. 61-88 (1999)
[22] Courtois N.T., Sepehrdad P., Susil P., Vaudenay S.: ElimLin Algorithm Revisited. In: International Workshop on Fast Software Encryption. LNCS, vol. 7549, pp. 306–325. Springer (2012)
[23] Fengjuan C., Xiao-Shan G., Chunming Y.: A Characteristic Set Method for Solving Boolean Equations and Applications in Cryptanalysis of Stream Ciphers. Journal of Systems Science and Complexity. 21(2), pp. 191-208 (2008)