Fitness Comparison by Statistical Testing in Construction of SAT-Based Guess-and-Determine Cryptographic Attacks

Artem Pavlenko
ITMO University
Saint Petersburg, Russia
alpavlenko@corp.ifmo.ru

Maxim Buzdalov
ITMO University
Saint Petersburg, Russia
mbuzdalov@corp.ifmo.ru

Vladimir Ulyantsev
ITMO University
Saint Petersburg, Russia
ulyantsev@corp.ifmo.ru

ABSTRACT

Algebraic cryptanalysis studies breaking ciphers by solving algebraic equations. Some of the promising approaches use SAT solvers for this purpose. Although the corresponding satisfiability problems are hard, their difficulty can often be lowered by choosing a set of variables to brute force over, and by solving each of the corresponding reduced problems using a SAT solver, which is called the guess-and-determine attack. In many successful cipher breaking attempts this set was chosen analytically, however, the nature of the problem makes evolutionary computation a good choice.

We investigate one particular method for constructing guess-and-determine attacks based on evolutionary algorithms. This method estimates the fitness of a particular guessed bit set by Monte-Carlo simulations. We show that using statistical tests within the comparator of fitness values, which can be used to reduce the necessary number of samples, together with a dynamic strategy for the upper limit on the number of samples, speeds up the attack by a factor of 1.5 to 4.3 even on a distributed cluster.

CCS CONCEPTS

• **Security and privacy → Block and stream ciphers**; **Cryptanalysis and other attacks**; • **Theory of computation → Evolutionary algorithms**; *Constraint and logic programming*.

KEYWORDS

Algebraic cryptanalysis, satisfiability problems, approximate fitness evaluation.

ACM Reference Format:
Artem Pavlenko, Maxim Buzdalov, and Vladimir Ulyantsev. 2019. Fitness Comparison by Statistical Testing in Construction of SAT-Based Guess-and-Determine Cryptographic Attacks. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321847

1 INTRODUCTION

Cryptanalysis is a field of cryptology that studies how hard it is to break cryptographic systems. The main desire of cryptanalysis is to find an easy way to break a given code, however, as most modern cryptographic techniques are very strong, even a small improvement compared to brute force is sometimes a good result.

One of its subfields, algebraic cryptanalysis, aims at reducing problems of breaking codes to solving algebraic equations. For some classes of such equations, very powerful tools have been developed, which are of course used not only for cryptanalysis; the most prominent example is the Boolean satisfiability problem (SAT) and numerous efficient tools called SAT solvers. However, problems derived from cryptography are still very hard and often require some effort to make the existing tools usable.

One of the well-established ways to help SAT solvers with the cryptography problems is called the *guess-and-determine attack*, and is effectively a combination of brute force search on the carefully chosen set of Boolean variables (the *guessed bit set*) and running a SAT solver to determine the values of other variables. A good choice of the guessed bit set can significantly reduce the time of solving the equation, and thus to complete the attack. However, finding such a set is a hard problem, and most known attacks of this sort were designed analytically.

We continue a line of the research that aims at finding a good guessed bit set using evolutionary algorithms. Our main contribution is towards improvement of evaluation and comparison of fitness values. As this particular application domain is quite far away from a typical GECCO contribution, we first introduce it on a more detailed level, which will hopefully make the reader comfortable with the problem, and then formulate our contribution.

1.1 Basics of Cryptanalysis

A typical cryptographic scenario is to take some message, or *plaintext*, which needs to be secretly passed to the receiving side, and turn it into the *ciphertext*, that can be easily restored at the receiving side without losses, but is very hard to be understood when eavesdropped. The goal of cryptanalysis is to restore as much information from the plaintext data, given the ciphertext. Apart from guessing the exact plaintext, partial breaks (guessing the algorithm, deducing information about the plaintext) are often useful as well.

The attacks can be classified into the following categories [36]:

- Ciphertext-only: the only available data is the ciphertext (or a collection of ciphertexts).
- Known-plaintext: there are also plaintexts, for each of which the corresponding ciphertext is known.
- Chosen-plaintext (or chosen-ciphertext): one can obtain ciphertexts for as many plaintexts as one wishes (or vice versa), but such ability is granted only once.
- Adaptive chosen-plaintext (or adaptive chosen-ciphertext): same as above, but subsequent plaintexts (or ciphertexts) can be chosen based on information learned from previous ones.

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321847

- Related-key attack: one can obtain ciphertexts from the given plaintexts using two keys which are related in a known way.

An attack can require different resources in different quantities, such as time (the amount of calculations needed to complete the attack, or an equivalent measure), memory (how much of a computer memory is needed to make the attack possible), or data (the lengths of plaintexts or ciphertexts).

It is often the case that a particular research did not lead to a successful discovery of a plaintext from the given ciphertext (or to a similar success in terms of other possible types of attacks), but rather estimated the amount of the resources needed to do this, even if this amount is astronomically large. An example of such statement is "SHA-1 collisions now 2^{52}" [27], which basically means that a way is found to find two texts with the same value of the SHA-1 hash function in roughly 2^{52} elementary operations. The presented amount of resources need not be practical to treat the attack as a successful one; in fact, numbers much smaller than the ones required to perform a brute force attack are to be interpreted as a signal to move away from the particular compromised cryptosystem long before such an attack becomes physically possible.

1.2 Symmetric and Public-Key Cryptography

Modern cryptographic algorithms, as well as the corresponding fields of study, can be roughly split in two categories: the *symmetric-key cryptography* and the *public-key*, or *asymmetric cryptography*. Cryptographic pseudorandom number generators is another related field of study, which is aimed at constructing cryptographically strong pseudorandom number generators, as well as finding and exploiting their weaknesses.

The symmetric-key cryptographic algorithms encrypt and decrypt the messages using the same secret key. An example of this approach is as follows: assume the plaintext message is a bit string A of length n, and there is a secret key K, also a bit string of length n, that is available both to the sender and the receiver. Then the ciphertext is $B = A \oplus K$, where the $\oplus$ is the bitwise exclusive-OR operation. The receiver can restore the plaintext by applying the same operation $B \oplus K = (A \oplus K) \oplus K = A$. The cryptographic strength in all such algorithms is determined by the properties of K, more precisely, of the algorithms that generate such secret keys. An essential practice is to avoid using the same K for more than one message, which leads to considering *stream ciphers*, also known as *keystreams*, instead of single secret keys K, and the necessity of *keystream generators*. Many cryptographic attacks on symmetric-key algorithms try to predict the keystream bits following the eavesdropped sequence.

On the other hand, the public-key cryptography requires two essentially different keys: the *public key*, which is known to everyone and is used to generate ciphertexts intended to be read by the receiver, and the *private key*, which is known only to the receiver and is used to decrypt the incoming messages. The cryptographic strength is based on the difficulty of restoring the private key using the available information, such as the public key. The RSA algorithm [35] is a well-known example of this approach. Note that *digital signatures* are also based on similar principles where the roles of the keys are essentially reversed: a private key is used to compute the signature, while a public key is used to ensure to verify the sender's identity and that the message is not altered.

In this paper, we restrict ourselves with symmetric-key cryptography and stream ciphers.

1.3 Algebraic Cryptanalysis

Algebraic cryptanalysis performs attacks by reducing the problem of conducting a particular attack to a system of polynomial equations, typically over some finite field, and then solving this system [2]. The attack generally follows three phases.

First, an algorithm is translated to a system of equations, where the set of variables generally consists of the input and the output bits, or, in the case of keystream generators, the output bits and the internal state of the algorithm. The system is intended to be solved for concrete values of the output bits, and when it is solved, values of all other variables become known, thus revealing either the plaintext or the internal state of the algorithm, where the latter enables predicting the rest of the keystream. This translation can be manual, which allows for some additional considerations that can simplify the system, or automated [20, 30].

Second, the constructed system is analyzed, and the efficient way of solving it is chosen. A SAT solver can be used [8, 9, 37, 38], some of which are designed to work well with cryptography problems [39]. Other means are also of use, such as solvers for constraint satisfaction problems [15] or less-known approaches such as the Characteristic Set method [18, 19]. On this stage, the bounds on time and data can be estimated. Finally, the actual attack can be performed when the data becomes available.

1.4 Guess-and-Determine Attacks

Existing solvers of equations over finite fields, such as SAT solvers, typically find cryptography-derived equations to be very hard. However, sometimes it is possible to hybridize such a solver with plain brute force over a subset of variables, letting the solver do its work for the rest of variables once the chosen ones are fixed to certain values. Of course, it would be a surprise if a randomly chosen subset of variables worked, but a good choice can result in significant reduction of the attack time. This technique is called the *guess-and-determine attack*, and the set of variables to brute force over is called the *guessed bit set*.

In fact, such an attack can use a much simpler tool for solving the remaining equations. One of the simplest examples of the guess-and-determine attack was presented in [1] for the A5/1 cipher, which served as a standard for encrypting the GSM traffic in cellular telephony. The internal state of this cipher is completely described by 64 bits. It was found that a carefully chosen set of 53 bits enables finding the remaining 11 bits using a triangular system of equations.

Most guess-and-determine attacks use more powerful tools to solve reduced systems of equations, however the choice of the guessed bit set is still largely based on analysis of the properties of the cipher [9, 18]. Some of the recent papers tend to automate this process by using e.g. tabu search or genetic algorithms [37, 38].

1.5 Evolutionary Computation Applied to Cryptography

Applications of evolutionary computation to cryptanalysis, and to cryptography in general, are wide and diverse and can be both destructive and constructive.

The survey [22] references works using evolutionary computation methods for finding short addition chains that are used to speed up modular exponentiation of the form $B = A^c \bmod P$ compared to the classical binary method, breaking Merkle-Hellman knapsack-based cryptosystems by a constant factor faster than previously known methods, constructing cryptographic functions that possess certain properties (having high nonlinearity, being balanced) or their building blocks called the S-boxes, evolving cryptographic pseudorandom number generators. A lot of work towards constructing cryptographically sound functions was also done in [7].

An earlier survey [31] also mentions more exotic works connected with evolutionary computation, such as the ICIGA system that uses the building blocks of evolutionary algorithms (mutation and crossover) as instruments for encryption and decryption [41], or genetic programming to evolve hardware for RSA cryptosystems that is fast, consumes small enough space on a chip, and is resistant to side-channel leakage, all in the same time [28].

There is also some recent interest for analysing quantum cryptography using evolutionary algorithms. For instance, [23] proposes a genetic algorithm to measure the tolerance of a quantum key distribution algorithm to adversaries that are not all-powerful but rather having access only to physically possible devices.

Genetic algorithms working on binary strings are often seen as natural candidates for cracking ciphers. For instance, [13] attempts to crack the RC4 keystream generator using the genetic algorithm, where the individuals are the descriptions of the initial state of the cipher (the state is a permutation of size 256). The fitness function reflects how much of the keystream produced from the guessed state matches the required keystream. The authors derive the attack time to be $2^{121.5}$ generations, however, this estimation is based on the extrapolation of the observed progress, and thus is rather questionable. A particle swarm optimizer and simulated annealing were also tested, but were found to be inferior to the genetic algorithm.

The work [32] also attacks the RC4 cipher, but in a different way: here, the individual is the description of a linear feedback shift register, with which one tries to approximate the cipher. The fitness is as follows: the register is initialized with the first bits of the keystream, and the rest of the real keystream is compared with the keystream produced by the individual, where the needed amount of first bits is not considered. The paper reports a maximum similarity of the fitted linear feedback shift register to be 80%.

1.6 Contribution and Structure of the Paper

This paper continues a line of the research which:

- performs cryptanalysis of keystream generators, thus staying within the realm of symmetric cryptography;
- uses an automated conversion of their definition to SAT formulas using tools such as [30];
- implements guess-and-determine attacks over these SAT formulas using existing SAT solvers to do the rest [37];
- chooses guessed bit sets using black-box optimizers (evolutionary algorithms in this paper, tabu search in [38]).

In this framework, measuring the fitness of a guessed bit set relates on Monte-Carlo sampling and becomes more precise with more samples, which will be detailed in subsequent sections. We aim at reducing the overall computing time, without changing the landscape of the results, by improving the detection of whether one fitness is smaller than, greater than, or equal to another fitness using statistical testing. This improved comparison allows reducing the number of samples, needed to perform before the decision is made with enough confidence. As a result, the running times needed to find a good guessed bit set reduced by up to 4.3 times, even when running on a computing cluster with all the scheduling overheads.

The rest of the paper is structured as follows. Section 2 explains the pipeline in more detail, including the description of the fitness function. Section 3 elaborates on the proposed usage of statistical testing to improve comparison of fitness values, taking into account that the entire algorithm is typically run in a distributed environment. Section 4 presents results of experiments with and without the proposed technique, and also discusses the consequences of applying certain statistical tests. Section 5 concludes the paper and gives the final remarks.

2 PRELIMINARIES

2.1 Boolean Satisfiability and SAT Solvers

The Boolean satisfiability problem (SAT) is defined as follows. There are n Boolean variables $x_1, \ldots, x_n$ which take values from $\{\texttt{false}, \texttt{true}\}$, or, alternatively written, from $\{0; 1\}$. An expression $F(x_1, \ldots, x_n)$ is given, which evaluates to a Boolean value and has a known structure. The problem is to find a *satisfying assignment*, which is a mapping from a variable x_i to the value v_i it needs to take, such that $F(v_1, \ldots, v_n) = 1$, or to show that there is no satisfying assignment.

The expressions are usually presented using a small set of small Boolean functions, or a *basis*. If any Boolean function can be expressed in a certain basis, such basis is called *complete*. The most popular complete basis consists of *negation* (a function that takes one argument x and returns $1 - x$, denoted as $\neg x$), *conjunction* (takes two arguments and returns 1 iff both arguments are 1, denoted as $\wedge$) and *disjunction* (takes two arguments and returns 0 iff both arguments are 0, denoted as $\vee$). As both conjunction and disjunction are associative and commutative, an expression of the form $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ is called a conjunction of variables $x_1 \ldots x_n$, and similarly $x_1 \vee x_2 \vee \ldots \vee x_n$ is a disjunction of these variables.

Practical satisfiability problems are usually presented in the *conjunctive normal form*, or CNF. In this form, the expression F is a conjunction of m subexpressions $F_1 \wedge \ldots \wedge F_m$, where each F_i is a disjunction of either a variable or its negation. Many classes of problems can be expressed in CNF in polynomial time, however, finding a satisfying assignment is a well-known NP-complete problem, apart from a few special cases.

A large demand to solve satisfiability problems appearing in practice has led to development of many highly efficient software tools called SAT solvers [5]. There are yearly competitions dedicated to comparison of SAT solvers on problems from different areas [4].

2.2 SAT-Based Keystream Breaking

A keystream generator can be described as a collection of bits defining its internal state, together with a procedure that determines the changes happening with the internal state, as well as the produced output bits, on each iteration.

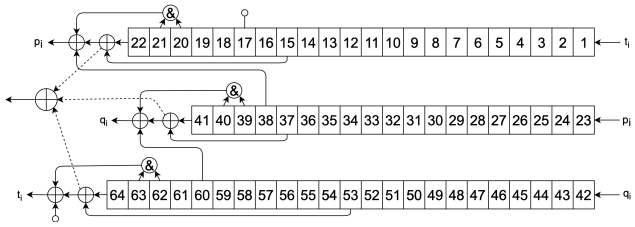

Figure 1: Trivium-64, a keystream generator

As an example of a stream cipher we consider a weaker version of Trivium [10], called Trivium-64, that was created specially for studies in cryptanalysis. Its structure is presented on Fig. 1.

It contains three registers of sizes 22, 19, and 23. On each iteration, the bits in these registers are shifted (right to left on Fig. 1), and the new values for the bits on the right are determined using exclusive-OR and AND operations on several chosen bits. Some of the intermediate values during these computations also determine the value of the output bit. For instance, the following happens with the middle register: the old values of bits 41 and 37 get XOR-ed and serve as one of the inputs for the output bit; additionally, this value is XOR-ed with the result of AND on bits 39 and 40, as well as with bit 60 coming from the bottom register, and this new value goes to the rightmost bit of the bottom register.

When the internal state of such a cipher is initialized with some values, its subsequent output is fully determined by these values. If an attacker has a plaintext and a corresponding ciphertext, she can apply bitwise exclusive-OR and get a part of the keystream. If this part is large enough, she can uniquely restore the internal state as it was just before generating this keystream, and by this she can predict arbitrarily many bits from the rest of the keystream.

To do that, it is possible to create a Boolean formula over variables coming from three sets: the *input variables* X_{in} which correspond to the initial values of the internal state bits, the *output variables* X_{out} which correspond to the generated keystream, and the *intermediate variables* X_{mid} which might be necessary to encode the values of the internal state bits in the meantime. Such a formula $F(X_{\text{in}} \cup X_{\text{out}} \cup X_{\text{mid}})$ is satisfied only by valid combinations of the variable values. Note that it is trivially computed by unit propagation once the input variables X_{in} are substituted by their concrete values [43]. The attack is performed using the other direction: the output variables X_{out} are substituted by the eavesdropped keystream, and the solution would tell which X_{in} values were needed to generate such a keystream.

To get an impression for how such a formula can be constructed, we again use Trivium (Fig. 1) as an example. For each output variable $o^i \in X_{\text{out}}$ we declare 70 auxiliary variables a^i_j, 64 of which correspond to internal state variables, and six more correspond to the nodes of intermediate computations. For all internal variables with the lower index j less than 65 and not equal to 1, 23 and 42, we know that $a^i_j = a^{i-1}_{j-1}$, so we add two CNF clauses $(a^i_j \vee \neg a^{i-1}_{j-1})$, $(\neg a^i_j \vee a^{i-1}_{j-1})$ to our formula. All other variables, including o^i, are defined as results of XOR-ing or AND-ing several other variables, for which we can also add the corresponding clauses to our formula.

Note that each input variable $n_j \in X_{\text{in}}$ is essentially the same as a^0_j, so we can either merge them or add some more clauses.

The resulting formula looks very large, however, as many variables are just copies of other variables, they can be eliminated by some formula preprocessing, and many of the variables representing the intermediate results can also be eliminated, which results in a much smaller formula. Tools such as [20, 30] can take care of most of the steps automatically.

Generally, the number of output variables (which is the length of the eavesdropped keystream) needs to be large enough to determine the input variables uniquely (at least as many as input variables).

2.3 Guess-and-Determine Attacks

As explained in Section 1.4, guess-and-determine attacks are a viable way to solve the satisfiability problems constructed from keystream generators as described in the previous section. To perform such an attack, one needs to choose a guessed bit set B from the non-output variables: $B \subset X_{\text{in}} \cup X_{\text{mid}}$. Then one needs to iterate over all possible assignments to the variables from B (there are $2^{|B|}$ of them), substitute these assignments, together with the values of the output variables, into the formula F, and solve the reduced formula on variables $(X_{\text{in}} \cup X_{\text{mid}}) \setminus B$.

One useful property of the family of satisfiability subproblems imposed by introducing a guessed bit set is that *exactly one* of these problems is satisfiable, and all others are not. Indeed, if the values of the output variables uniquely determine the values of all other variables, then only one of our guesses for values of B matches the right values. This means that if one of the subproblems is solved, the attack is complete, and solvers that are busy with other subproblems can be safely terminated. This property can significantly reduce the time needed to design such an attack.

2.4 The Quality Measure of a Guessed Bit Set

In this section, we describe the way the quality of a guessed bit set B, which is essentially the time required for the corresponding attack, is evaluated, as proposed in [38] and similar to [12].

The attack is parameterized with a time limit T, which is imposed on the SAT solver that would solve all the satisfiability problems. We assume that we have an unlimited keystream, which we split into blocks of size $|X_{\text{out}}|$. Once we found the initial state of the cipher that generated any of these blocks, the attack is complete. When working with one of these blocks, $2^{|B|}$ satisfiability problems are created, which are all submitted to the SAT solver, in turn. If one of these problems is solved within the time limit T, and the satisfying assignment is found, the attack is complete. Otherwise, it takes at most $T \cdot 2^{|B|}$ seconds to process such a block.

As the cipher is presumably rather strong, we can safely assume that our keystream blocks come from a uniform distribution over all possible blocks of the required size. If we estimate the probability p that such a block is solved, then the expected time until the attack is complete is $T \cdot 2^{|B|}/p$, and the attack will complete with probability 0.95 in time $3 \cdot T \cdot 2^{|B|}/p$.

In turn, the probability p can be estimated using the Monte-Carlo principle as follows. We sample the values of *input* variables uniformly at random, substitute them into the formula and derive the values of all other variables (using unit propagation in linear

time). By this we obtain not only an assignment for output variables, but an assignment for the guessed bit set that corresponds to the unique solvable subproblem. Then we run the SAT solver with the time limit T on this subproblem, and record whether it managed to solve it. If this procedure is repeated m times, and in m_1 cases the subproblem is solved, the probability can be estimated as $p = m_1/m$.

Note that such a procedure is roughly $2^{|B|}$ times faster than measuring the expected time of the attack experimentally, and, in the same time, the precision of the result is not sacrificed. Note that p depends on T, and the true attack time is a minimum over all T of the values produced by the described procedure. However, for practical reasons, only few values for T are usually tested.

One can additionally optimize the time limit value T for each of the guessed bit sets in order to decrease the estimated time of the attack. For example, this value can be safely reduced to be just a little bit greater than the maximum of the times for successful measurements. If most of the successful measurements are concentrated at small times, the time limit can be reduced further.

The process of designing the attack, as well as the attack itself, can enjoy massive parallelism if available [37]. Indeed, the attack can solve the subproblems induced by the guessed bit set B in parallel. During the design of the attack multiple assignments of input variables, that estimate the quality of a single guessed bit set, can also be processed in parallel, let alone independent estimation of multiple guessed bit sets.

2.5 Evolutionary Techniques for Automated Discovery of Guessed Bit Sets

The paper [37] explains guess-and-determine attacks, where the subproblems are intended to be solved by SAT solvers, as a special case of SAT partitioning. For evaluation of quality of a guessed bit set, which is used as a fitness function, they used an approach similar in spirit to the one described in Section 2.4, but no explicit time limit was used: fitness evaluation waited for all subproblems to be solved. Tabu search [16] and simulated annealing [21] were used as optimizers. MiniSAT [11] was used to solve the subproblems, and a special manager called PDSAT was used to control its instances within a computational cluster, as well as within the SAT@home project. The following ciphers were considered in that paper: A5/1, Bivium [10] and Grain [17]. The number of Monte-Carlo samples during fitness evaluation was set to 10^4 for A5/1 and to 10^5 for Bivium and Grain. Unfortunately, we cannot compare our results directly with that paper, as fitness functions differ significantly.

The paper [38] uses fitness function (called the *resistance function*) of the same type as used in this study. The sample size was fixed and set to 1000. The ROKK SAT solver was used [44]. As an optimizer, a tailored tabu search algorithm was used. This paper reports results on weakened versions of Trivium, which are not better than the ones in [18], as well as improvements on the 2.5-round version of AES-128 and on the 8-round GOST 28147-89 cipher.

2.6 Ciphers Used in This Study

A stream cipher called Trivium was proposed in [10] and attracted significant interest in the cryptoanalysis community. It has 288 bits of internal state, which is initialized with a 80-bit secret key. There are no known attacks targeting this secret key which are more efficient than the brute force search. However, the problem of restoring the internal state was found to be easier than brute force by several authors [26, 34]. Most of attention is currently attracted to simplified Trivium-like ciphers. One of the, Bivium, was also proposed in [10], which has only 177 bits of internal state and only two registers, instead of three, and is much weaker than Trivium.

Papers [6, 40] 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. In this study, we use the Trivium-like ciphers from [6]. By Trivium-L we denote a cipher from this family, in which L is a total size of state that should be recovered. We consider attacks aimed at recovering the internal state of Trivium-96 (as described in [6]), Trivium-64 and Bivium. We also consider the A5/1 cipher, which is known to be completely solved as rainbow tables exist [29], however, it is still of interest when testing approaches for cryptanalysis.

3 PROPOSED IMPROVEMENTS

Recall that the fitness value in designing guess-and-determine attacks following [38] is a sequence of events of two types:

- the SAT solver was unable to restore the input variables and exceeded the time limit T;
- the SAT solver was able to restore these in time $t_i < T$.

Note that all events of the first type are indistinguishable. Assume the sequence S consists of N events, N_+ of them are of the second type and have associated times $t_1, t_2, \ldots, t_{N_+}$, and $N - N_+$ are of the first type. Based on this measure, the approximation of the time to restore the input variables for the first time (assuming on reaching the time limit we move on to another keystream sample) is:

$$E'(S) = \frac{\sum_{i=1}^{N_+} t_i}{N_+} + \frac{N - N_+}{N_+} \cdot T,$$

assuming $N_+ > 0$, otherwise $E(S) = \infty$. In the attack itself, each unsuccessful attempt requires not only time T for the correctly guessed values of the variables from the guessed bit set B, but at most $T \cdot (2^{|B|} - 1)$ more time for all incorrectly guessed values, and at most $T \cdot 2^{|B|}/2$ more time, on expectation, required to process some of the incorrectly guessed values on the successful round. This results in the following estimation for the attack time:

$$E_A(S) = \frac{\sum_{i=1}^{N_+} t_i}{N_+} + \left(\frac{N}{N_+} - \frac{1}{2}\right) \cdot (T \cdot 2^{|B|}). \quad (1)$$

Note that the first addend is typically much smaller than the second one, as it cannot exceed T. However, this is an important term when this estimation is used as a fitness function in evolutionary algorithms, as it is able to guide optimization towards smaller values of t_i when N_+ is fixed, and eventually leads to an increase of N_+. The existing approaches [38] use expressions like (1) with the fixed value of N.

A drawback of this approach is that N shall be quite large so that the error in (1) is relatively small, and such fitness values can be directly compared. On the other hand, some of the moves in the search space result in an inferior individual. Such an individual still takes considerable time to be evaluated, but its inferiority can often be detected at early stages. In this paper, we propose to use statistical testing in fitness comparison, which has chances to detect

Algorithm 1 (1 + 1) EA with Statistical Testing

```
– f : {0;1}^n → R ∪ {∞}                          ▷ function to make one measure
– Q ∈ N                                          ▷ batch size for measurements
– N_max ∈ N                                      ▷ max measures for one individual
– S : [R ∪ {∞}] × [R ∪ {∞}] → {−1; 0; 1}         ▷ statistical test
x ← UniformRandom({0;1}^n)                       ▷ the best individual
f_x ← [f(x) for Q times]                         ▷ initial fitness evaluation
while x is not good do
    y ← Mutate(x)                                ▷ the offspring
    f_y ← [f(y) for Q times]                     ▷ the offspring's fitness
    δ ← S(f_x, f_y)                              ▷ first statistical comparison
    while δ = 0 do                               ▷ not different → more evaluations
        if |f_y| < |f_x| then
            f_y ← f_y ∪ [f(y) for Q times]       ▷ more to the offspring
        else if |f_x| < N_max then
            f_x ← f_x ∪ [f(x) for Q times]       ▷ more to the parent
        else
            break
        end if
        δ ← S(f_x, f_y)
    end while
    if δ ≥ 0 then                                ▷ offspring not worse → replace
        x ← y
        f_x ← f_y
    end if
end while
```

statistically significant differences between two individuals early enough and save computational resources.

3.1 Example: (1 + 1) EA with Statistical Testing

The basic idea is presented as Algorithm 1. We lay out this algorithm in a quite generic form, e.g. we do not care yet what are the mutations, or which exactly statistical test we employ. There are two constants. $N_{\max}$ controls the maximum number of measurements conducted on a single individual, as otherwise two individuals that are identical in behavior will never be distinguished by a statistical test (or it will take too much time to occasionally fall below the statistical significance threshold). Q is the minimum number of measurements performed in a single run. On a single-threaded machine, $Q = 1$ is probably a viable choice, however when the measurements are performed on a cluster, Q shall be large enough in order to decrease the scheduling overheads.

On producing an offspring, the algorithm attempts to run as few measurements for the offspring as possible to determine whether it is better or not. However, if the parent and the offspring are too similar, it continues adding measurements to the offspring (or to the parent if needed) until either the parent and the offspring are found to be different, or the measurement limit is reached. When the offspring replaces the parent, all the measurements are kept with it for the future comparisons.

3.2 The Choice of the Statistical Test

For the considered problem, the measurements, which are the elementary inputs for a statistical test, seem to disobey common requirements of the most commonly used statistical tests, such as normality. Indeed, our data is right-censored, which means we cannot distinguish measurements if their true values are above the threshold T. Additionally, even without this feature, the measurements cannot expected to be normally distributed, as they depend on the interaction between the heuristics of a particular SAT solver and the structure of the satisfiability problem.

Our choice in these conditions is the Wilcoxon rank sum test [42], also known as the Mann-Whitney U test [25]. Opposed to more widely used tests assuming normality of distributions [24], this is a non-parametric test, as it relies only on results of comparisons between the measurements, but not on their actual values, so it does not require the data to be normally distributed. Although the original formulation requires the measurements to come from a continuous distribution, many implementations, such as the one from R [33], can correctly process distibutions which are discrete or partially discrete. This also enables producing correct results in the presence of censored measurements.

A slight disadvantage of the Wilcoxon rank sum test is that its alternative hypothesis is about medians, not the average values, so it does not reflect the spirit of the estimations in its entirety. However, we leave the development of the statistical test that captures this essence more precisely for the future work.

In the case where one needs to compare fitness values of the guessed bit sets of equal size, the Wilcoxon rank sum test can be applied immediately to the raw measurements. However, fitness values may also come from guessed bit sets of different sizes, for instance, $|B_1| < |B_2|$. Our solution in this case is to multiply the times of the successful attempts for the smaller guessed bit set by $|B_2|/|B_1|$. After that, some of these values may exceed the time limit T, in which case they are treated as if the solver did not manage to find the solution. By this we lose some of the information collected for B_1, however, the reverse direction of actions (e.g. dividing the results for the bigger guessed bit set by the same value) would underestimate the quality of the bigger set. Indeed, some of the original measurements might be just a little bit over the time limit, and these might get below the time limit once divided by $|B_2|/|B_1|$.

To evaluate the impact introduced by the choice of the statistical test, we used the Barnard's test [3], an improved version of the Fisher's exact test [14]. Both these tests are developed to compare two binary random variables, and so they ignore the times, reported when the SAT solver successfuly restores the cipher's state. Although the search process used the Wilcoxon rank sum test, we also recorded the results produced by the Barnard's test.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

For experimental evaluation, we considered the ciphers listed in Section 2.6 (A5/1, Bivium, Trivium-64, Trivium-96). For an optimizer, we used a genetic algorithm with population size $N = 10$ which works as follows:

- the elitist step: take two best individuals from the previous generation to the next one;
- the mutation step: choose, using roulette wheel selection, four individuals from the previous generation, apply mutation to them and add the results to the new generation;

Table 1: Overview of the best results. *Time limit* and *Attack time* refer to improved values explained at the end of Sect. 4.1

Cipher	$\|B\|$	Time limit	Attack time	#evals w/ stats	#evals w/o stats	Guessed bit set
A5/1	35	0.278	$2.19 \cdot 10^{12}$	1471	341	1 [3..5] 7 9 10 13 16 [18..27] 30 31 34 37 41 43 44 47 48 [50..52] 56 60 61 64
Bivium	28	2.715	$1.15 \cdot 10^{12}$	3616	2439	15 16 18 33 42 43 56 61 70 72 74 85 96 97 100 108 109 115 124 126 127 130 135 138 141 144 145 154
Trivium-64	21	2.373	$3.23 \cdot 10^{7}$	3398	1323	2 4 6 9 11 16 20 [23..28] 30 32 36 38 [46..48] 51
Trivium-96	35	2.485	$1.24 \cdot 10^{12}$	2494	1299	6 9 11 15 [18..21] 27 29 [32..34] 38 39 41 42 45 47 49 53 54 61 63 [69..71] 75 77 78 84 [90..93]

- the crossover step: choose, using roulette wheel selection, two pairs of individuals from the previous generation, apply crossover to each pair, add offspring to the new generation.

Note that the roulette wheel selector is based directly on the fitness function described in Section 2.4. However, during fitness evaluation, if the individual is found to be statistically worse than the best found individual, it is discarded by the selection process.

The initial population consists of copies of an individual that has all possible input bits set to 1, which corresponds to the guessed bit set consisting of all input bits, and which corresponds to the easiest problem for the SAT solver. Standard bit mutation is used. For the crossover, a uniform crossover with probability 0.2 is used. The algorithm performs a restart when there was no fitness improvement in 300 consecutive iterations.

The batch size for the measurements was chosen to be $Q = 100$. The maximum number of measurements for one individual was $N_{\max} = 500$. In the statistical test, the p-value threshold of 0.05 was used to determine statistical significance of a comparison. Note that the usual criticism of not having corrections for multiple comparisons does not really apply here, since statistical tests are used here as a heuristic to reduce the number of evaluations, mostly to cut off the individuals that are too bad.

For a SAT solver, we used the ROKK solver [44]. The time limit imposed on the SAT solver was set to 10 seconds.

All the evaluations were run on 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. For each cipher, five independent experiments has been performed, and for each experiment a wall-clock time budget of 12 CPU-hours was allocated.

The generated guessed bit sets were subsequently re-evaluated with the larger number of measurements ($N_{\text{big}} = 10^4$). Once this is done, the time limit is optimized for each guessed bit set separately to improve the attack time, as motivated in Section 2.4.

4.2 Results of Experiments: Big Picture

The overview of the best results achieved by our approach is presented in Table 1. Along with the attack times, we report the sizes of the guessed bit sets, the sets themselves, the time limit (in seconds) for the SAT solver used to achieve the attack time, as well as the number of evaluated individuals with and without applying statistical tests. Our approach reached the same magnitude of the attack quality as the previous approaches, and in particular produced the best known result for Trivium-96.

Note that the number of evaluated points, within the same computation time budget, is much larger with the proposed approach. Indeed, for A5/1 there were 4.31× more points tested, whereas for Bivium the factor is 1.48×, for Trivium-64 it is 2.56×, and for Trivium-96 it is 1.92×,

4.3 Assessing the Impact of a Choice of a Particular Statistical Test

During the experiments, not only we used the Wilcoxon rank sum test to perform comparison between the individuals, but we also ran the Barnard's test on the same inputs. It could be that both of the tests rejected their null hypothesis, or none of them did, or only one of them rejected its null hypothesis.

Table 2 reports the number of cases for each of these cases across all runs. One can see that in most of cases the tests agree in their decisions (95.1% for A5/1, 73.8% for Bivium, 85.2% for Trivium-64, 91.5% for Trivium-96). However, the resolution power of the Wilcoxon rank sum test is for all the cases better than of the Barnard's test (18.99% vs 18.06% for A5/1, 72.80% vs 57.10% for Bivium, 46.63% vs 38.46% for Trivium-64, 32.56% vs 29.19% for Trivium-96). This justifies the usage of the Wilcoxon rank sum test in the realm of the guess-and-determine attacks driven by evolutionary algorithms.

4.4 Results of Experiments: Diversity Analysis

Five independent evaluations on each cipher enables us to perform a rudimentary diversity analysis. Table 3 presents the best and near-best individuals for every run (and every restart within a run, if any), along with their finalized fitness values.

The first insight is that the best results from each run typically do not vary by a very large margin: the maximum of the worst-to-best ratio, 20.13 is observed for A5/1, while others are smaller (2.23 for Bivium, 6.13 for Trivium-64, 5.7 for Trivium-96).

All ciphers feature results which are close in fitness, but significantly deviate in genotype (in particular, occasionally, in the guessed bit set size $|B|$). This signifies a sufficiently multimodal landscape of the problem, as well as the fact that our algorithm is able to reach basins of different local optima. On the other hand, the very similar individuals which are different in the location of only one bit (two last lines) can be different in their fitness by roughly 2×, which indicates that the problem landscape is not very smooth.

Table 2: Comparison of strength of the used statistical tests (# of measurements below the $p = 0.05$ level)

Cipher	Wilcoxon only	Barnard only	Both	None
A5/1	215	146	1182	5812
Bivium	3786	946	9381	3974
Trivium-64	1943	560	5951	8476
Trivium-96	738	318	3322	8092

Table 3: Differences between multiple runs; best individuals are typeset in bold

Run/ Restart	$\|B\|$	Time limit	Attack time	Guessed bit set
				Cipher: A5/1
1/1	28	1.162	$2.34 \cdot 10^{12}$	[1..5] 9 12 16 [19..25] 27 28 30 36 [41..43] [47..50] 52 60
2/1	28	2.605	$1.05 \cdot 10^{13}$	1 5 7 9 10 13 16 18 [19..26] 28 30 34 37 41 43 [50..52] 60 61 64
2/2	35	0.278	$2.19 \cdot 10^{12}$	**1 [3..5] 7 9 10 13 16 [18..27] 30 31 34 37 41 43 44 47 48 [50..52] 56 60 61 64**
3/1	28	1.572	$3.16 \cdot 10^{12}$	[1..3] [5..9] [19..24] [26..28] 33 34 36 38 [42..45] 49 59 61
	29	3.760	$7.57 \cdot 10^{12}$	[1..3] [5..8] 12 14 [19..24] [26..28] 34 36 38 [42..45] 48 49 59 61
4/1	37	0.173	$2.03 \cdot 10^{13}$	1 4 5 [7..9] 13 [15..24] 29 31 32 34 35 37 38 40 41 45 46 [49..54] 57 58 61
	39	0.155	$4.41 \cdot 10^{13}$	1 4 5 [7..9] 11 [15..24] 29 31 32 34 35 37 38 40 41 45 48 [49..54] 56 57 58 61 64
5/1	39	0.088	$5.20 \cdot 10^{12}$	[2..14] 16 19 [22..30] 33 37 39 43 46 [48..54] 56 58 62
	41	0.078	$6.23 \cdot 10^{12}$	[2..14] 16 19 [22..30] 33 37 39 43 [45..54] 56 58 62
				Cipher: Bivium
1/1	38	0.328	$2.67 \cdot 10^{12}$	4 7 18 21 34 36 37 39 45 58 60 61 65 66 72 74 76 87 99 105 120 127 132 135 139 143 145 148 [152..154] 157 159 160 165 169 171 172
2/1	28	2.715	$1.15 \cdot 10^{12}$	**15 16 18 33 42 43 56 61 70 72 74 85 96 97 100 108 109 115 124 126 127 130 135 138 141 144 145 154**
	35	0.121	$1.31 \cdot 10^{12}$	3 4 15 16 18 33 42 43 46 56 61 70 72 74 85 96 100 102 108 109 115 124 126 127 130 135 138 140 141 144 145 148 153 154 160
3/1	38	0.273	$1.17 \cdot 10^{12}$	1 4 5 17 20 32 35 41 [43..45] 56 [60..62] 68 [83..85] 97 101 112 113 122 131 137 139 140 143 145 148 149 [153..155] 164 166 169
	36	0.296	$1.39 \cdot 10^{12}$	1 4 5 17 20 32 35 43 44 56 61 62 68 [83..85] 97 112 113 115 121 122 131 137 139 140 143 148 149 153 155 162 164 166 168 169
4/1	38	0.316	$1.79 \cdot 10^{12}$	4 5 17 22 33 38 43 45 47 49 57 65 70 71 74 87 89 98 100 104 112 115 119 127 128 136 137 [141..143] 145 149 152 154 156 164 166 171
5/1	38	0.720	$2.03 \cdot 10^{12}$	1 3 14 17 19 28 43 [44..46] 54 55 57 59 60 68 [70..72] 104 108 110 [121..123] 126 128 129 131 140 144 146 [149..151] 154 156 158
				Cipher: Trivium 64
1/1	21	1.982	$4.36 \cdot 10^{7}$	1 8 11 15 18 23 26 [28..31] [33..35] 38 39 49 [56..58] 60
2/1	21	2.373	$3.23 \cdot 10^{7}$	**2 4 6 9 11 16 20 [23..28] 30 32 36 38 [46..48] 51**
	22	2.387	$4.69 \cdot 10^{7}$	4 6 9 11 16 20 23 24 [26..28] 30 32 33 38 46 47 [50..52] 58 60
	23	0.643	$4.80 \cdot 10^{7}$	4 6 9 11 16 20 [23..30] 32 33 36 46 47 [50..52] 58
3/1	20	2.447	$3.25 \cdot 10^{7}$	1 8 11 18 23 [25..31] 34 35 37 46 49 54 56 60
	21	2.489	$3.44 \cdot 10^{7}$	1 8 11 18 23 [25..31] 34 35 46 48 49 51 54 56 60
	23	0.785	$3.65 \cdot 10^{7}$	1 3 8 11 18 23 [25..31] 34 35 37 46 48 49 51 54 56 60
4/1	23	0.808	$3.94 \cdot 10^{7}$	2 3 9 10 16 [24..31] 37 42 45 [48..50] 55 56 59 60
	19	2.489	$4.25 \cdot 10^{7}$	2 6 7 [24..31] 33 37 45 [48..50] 55 56
4/2	18	8.658	$1.98 \cdot 10^{8}$	2 7 11 [23..25] [27..31] [33..35] 45 49 50 52
5/1	21	2.417	$4.26 \cdot 10^{7}$	3 4 10 [24..29] 32 34 37 42 [47..49] [51..53] 55 59
	22	1.972	$3.90 \cdot 10^{7}$	4 5 10 17 [24..30] 32 34 42 44 [47..49] 51 53 55 59
				Cipher: Trivium 96
1/1	35	1.957	$1.73 \cdot 10^{12}$	4 5 8 [14..16] 20 23 26 29 36 39 40 43 44 46 49 50 52 53 56 57 66 [71..74] [78..81] [87..89] 91
2/1	35	2.485	$1.24 \cdot 10^{12}$	**6 9 11 15 [18..21] 27 29 [32..34] 38 39 41 42 45 47 49 53 54 61 63 [69..71] 75 77 78 84 [90..93]**
	32	2.184	$2.49 \cdot 10^{12}$	6 9 10 15 [18..21] [27..29] 33 38 39 41 45 [47..49] 53 54 61 69 70 77 78 84 [90..94]
3/1	37	0.743	$1.40 \cdot 10^{12}$	4 [6..8] [13..17] 20 22 23 28 31 36 38 39 41 43 45 [47..49] 55 56 63 64 77 [79..81] 84 87 92 [94..96]
	35	2.426	$1.43 \cdot 10^{12}$	4 [6..8] 13 [15..17] 22 23 28 38 39 41 [43..45] [47..51] 55 56 63 64 70 [79..81] 84 87 [94..96]
4/1	35	2.472	$1.75 \cdot 10^{12}$	7 8 [15..17] [21..23] 27 30 [34..36] 40 44 45 47 49 50 52 53 55 56 64 65 69 73 77 79 80 [84..86] 88 94
	36	2.385	$1.81 \cdot 10^{12}$	8 [14..17] [21..23] 27 29 [34..36] 40 [43..45] [47..50] 52 53 [55..57] 65 69 77 79 80 [84..86] 88 94
	33	2.284	$1.66 \cdot 10^{12}$	4 7 8 14 16 17 21 27 30 34 36 40 44 45 47 48 50 52 53 55 56 57 59 64 65 77 79 80 [84..86] 88 94
5/1	32	2.112	$1.46 \cdot 10^{12}$	3 4 [10..12] 18 19 21 22 26 27 29 33 36 39 41 43 [47..49] 54 62 63 [69..71] 75 78 83 84 86 92
	33	2.502	$2.33 \cdot 10^{12}$	3 4 [10..12] 16 18 22 29 32 34 36 [40..43] 45 [47..50] 54 63 69 70 75 78 [82..84] 86 90 92
	30	2.456	$3.77 \cdot 10^{12}$	4 [10..12] 18 19 22 26 27 29 31 33 36 41 43 [47..49] 54 62 [69..71] 75 78 83 84 86 91 92
	30	5.045	$7.07 \cdot 10^{12}$	4 [10..12] 18 19 22 26 27 29 32 33 36 41 43 [47..49] 54 62 [69..71] 75 78 83 84 86 91 92

The results also indicate that the problem cannot be significantly simplified by noticing certain features and improving the algorithm by taking them into account. None of the ciphers produced the evidence of certain bits that should be set in any optimal solution. There is also no evidence that contiguous groups of bits shall be selected together in a good solution, as observed bit groups are not longer than those in uniformly sampled bit sets of the same size.

5 CONCLUSION

We introduced statistical tests as a viable tool to save computational resources in cracking stream ciphers by guess-and-determine attacks assessed by SAT solvers, where the guessed bit sets are evolved using evolutionary algorithms. It enabled to compute 1.5× to 4.3× more individuals using the same computational budget by quickly deciding which individuals are too bad to be considered in full, or which are too good to continue comparison with. We have investigated some basic properties of the landscapes of the problem, from the evolutionary computation perspective, on different ciphers, which indeed indicate that the problem is not trivially solvable by local search techniques.

Our future work is to understand better the statistical properties when comparing individuals whose fitness is a sequence of right-censored measurements with possibly different thresholds. Although we showed that the Wilcoxon rank sum test performs rather well, and in particular better than the Barnard's test, a statistical test that suits our needs better is worth developing.

Acknowledgment: This research was supported by the Russian Scientific Foundation, agreement No. 18-71-00150. The authors would like to thank Alexander Semenov and anonymous reviewers for useful comments.

REFERENCES

[1] Ross Anderson. 1994. A5 (was: Hacking digital phones). http://yarchive.net/phone/gsmcipher.html

[2] Gregory V. Bard. 2009. *Algebraic Cryptanalysis*. Springer.

[3] George Alfred Barnard. 1945. A New Test for 2 × 2 Tables. *Nature* 156 (1945), 177. https://doi.org/10.1038/156177a0

[4] Armin Biere. 2016. Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016. In *Proceedings of SAT Competition 2016*, Vol. B-2016-1. 44–45.

[5] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). 2009. *Handbook of Satisfiability*. Number 185 in Frontiers in Artificial Intelligence and Applications. IOS Press.

[6] Antonio Castro Lechtaler, Marcelo Cipriano, Edith García, Julio Liporace, Ariel Maiorano, and Eduardo Malvacio. 2014. Model Design for a Reduced Variant of a Trivium Type Stream Cipher. *Journal of Computer Science & Technology* 14, 01 (2014), 55–58.

[7] Andrew John Clark. 1998. *Optimisation Heuristics for Cryptology*. Ph.D. Dissertation. Queensland University of Technology.

[8] Nicolas T. Courtois and Gregory V. Bard. 2007. Algebraic Cryptanalysis of the Data Encryption Standard. In *Cryptography and Coding*. Number 4887 in Lecture Notes in Computer Science. 152–169.

[9] Nicolas T. Courtois, Jerzy A. Gawinecki, and Guangyan Song. 2012. Contradiction Immunity and Guess-then-Determine Attacks on GOST. *Tatra Mountains Mathematical Publications* 53 (2012), 65–79.

[10] Christophe De Cannière. 2006. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *Information Security*. Number 4176 in Lecture Notes in Computer Science. 171–186.

[11] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *SAT 2003: Theory and Applications of Satisfiability Testing*. Number 2919 in Lecture Notes in Computer Science. 502–518.

[12] T. Eibach, E. Pilz, and G. Völkel. 2008. Attacking Bivium Using SAT Solvers. In *SAT 2008*. Number 4996 in Lecture Notes in Computer Science. 63–76.

[13] Benjamin Ferriman. 2013. *Cryptanalysis of the RC4 Stream Cipher using Evolutionary Computation Methods*. Ph.D. Dissertation. University of Guelph. http://hdl.handle.net/10214/7770

[14] Ronald A. Fisher. 1922. On the interpretation of $\chi 2$ from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society* 85, 1 (1922), 87–94.

[15] David Gerault, Marine Minier, and Christine Solnon. 2017. Using Constraint Programming to Solve a Cryptanalytic Problem. In *Proceedings of International Joint Conference on Artificial Intelligence*. 4844–4848.

[16] F. Glover. 1998. Tabu search methods for optimization. *Feature Issue of European Journal on Operations Research* 106, 2 (1998), 110–115.

[17] M. Hell, T. Johansson, and W. Meier. 2007. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing* 2, 1 (2007), 86–93.

[18] Zhenyu Huang and Dongdai Lin. 2011. Attacking Bivium and Trivium with the Characteristic Set Method. In *AFRICACRYPT*. Number 6737 in Lecture Notes in Computer Science. 77–91.

[19] Zhenyu Huang, Yao Sun, and Dongdai Lin. 2016. On the Efficiency of Solving Boolean Polynomial Systems with the Characteristic Set Method. https://arxiv.org/abs/1405.4596v3

[20] Predrag Janicic. 2012. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science* 8, 3 (2012), 1–39.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680.

[22] Karlo Knežević. 2017. Combinatorial Optimization in Cryptography. In *Proceedings of 40th International Convention on Information and Communication Technology, Electronics and Microelectronics*. 1324–1330.

[23] Walter O. Krawiec and Sam A. Markelon. 2018. Genetic Algorithm to Study Practical Quantum Adversaries. In *Proceedings of Genetic and Evolutionary Computation Conference*. 1270–1277.

[24] W. Kruskal and W. Wallis. 1952. Use of ranks in one-criterion variance analysis. *J. Amer. Statist. Assoc.* 47 (1952), 583–621.

[25] Henry B. Mann and Donald R Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics* 18, 1 (1947), 50–60.

[26] Alexander Maximov and Alex Biryukov. 2007. Two Trivial Attacks on Trivium. In *Selected Areas in Cryptography (Lecture Notes in Computer Science)*. 36–55.

[27] Cameron McDonald, Josef Pieprzyk, and Phil Hawkes. 2009. Automatic Differential Path Searching for SHA-1. http://eurocrypt2009rump.cr.yp.to/837a0a8086fa6ca714249409ddfae43d.pdf

[28] N. Nedjah and L. de Macedo Mourelle. 2004. Multi-Objective Evolutionary Hardware for RSA-Based Cryptosystems. In *Proceedings of International Conference on Information Technology: Coding and Computing*, Vol. 2. 503–507.

[29] K. Nohl. 2010. Attacking Phone Privacy. https://media.blackhat.com/bh-us-10/whitepapers/Nohl/BlackHat-USA-2010-Nohl-Attacking.Phone.Privacy-wp.pdf

[30] Ilya Otpuschennikov, Alexander Semenov, Irina Gribanova, Oleg Zaikin, and Stepan Kochemazov. 2016. Encoding Cryptographic Functions to SAT Using TRANSALG System. In *ECAI 2016*. Number 285 in Frontiers in Artificial Intelligence and Applications. 1594–1595. https://doi.org/10.3233/978-1-61499-672-9-1594

[31] Stjepan Picek and Marin Golub. 2011. On Evolutionary Computation Methods in Cryptography. In *Proceedings of 34th International Convention on Information and Communication Technology, Electronics and Microelectronics*. 1496–1501.

[32] Iwona Polak and Mariusz Boryczka. 2015. Genetic Algorithm in Stream Cipher Cryptanalysis. In *Computational Collective Intelligence: Proceedings of ICCCI 2015, Part II*. Number 9330 in Lecture Notes in Computer Science. 149–158.

[33] R Core Team. 2013. R: A Language and Environment for Statistical Computing. http://www.R-project.org/. http://www.R-project.org/

[34] Havard Raddum. 2006. Cryptanalytic Results on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039.

[35] Ronald L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126. https://doi.org/10.1145/359340.359342

[36] Klaus Schmeh. 2003. *Cryptography and public key infrastructure on the Internet*. John Wiley & Sons.

[37] Alexander Semenov and Oleg Zaikin. 2016. Algorithm for finding partitionings of hard variants of boolean satisfiability problem with application to inversion of some cryptographic functions. *SpringerPlus* 5:554 (2016). https://doi.org/10.1186/s40064-016-2187-4

[38] Alexander Semenov, Oleg Zaikin, Ilya Otpuschennikov, Stepan Kochemazov, and Alexey Ignatiev. 2018. On Cryptographic Attacks Using Backdoors for SAT. In *Proceedings of the AAAI Conference*. 6641–6648.

[39] Mate Soos, Karsten Nohl, and Claude Castelluccia. 2009. Extending SAT Solvers to Cryptographic Problems. In *Theory and Applications of Satisfiability Testing — SAT 2009 (Lecture Notes in Computer Science)*. 244–257.

[40] Sui-Guan Teo, Kenneth Koon-Ho Wong, Harry Bartlett, Leonie Simpson, and Ed Dawson. 2014. Algebraic analysis of Trivium-like ciphers. In *Proceedings of Australasian Information Security Conference*, Vol. 149. Australian Computer Society, 77–81.

[41] A. Tragha, F. Omary, and A. Mouloudi. 2006. ICIGA: Improved Cryptography Inspired by Genetic Algorithms. In *Proceedings of International Conference on Hybrid Information Technology*. 335–341.

[42] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.

[43] Ryan Williams, Carla P. Gomes, and Bart Selman. 2003. Backdoors to Typical Case Complexity. In *Proceedings of International Joint Conference on Artificial Intelligence*. 1173–1178.

[44] Takeru Yasumoto and Takumi Okuwaga. 2014. ROKK 1.0.1. In *SAT Competition 2014*, Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo (Eds.). 70.