Evolving Boolean Functions for Fast and Efficient Randomness Testing

Vojtech Mrazek
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Brno, Czech Republic
imrazek@fit.vutbr.cz

Marek Sys
Masaryk University
Faculty of Informatics
Brno, Czech Republic
syso@mail.muni.cz

Zdenek Vasicek
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Brno, Czech Republic
vasicek@fit.vutbr.cz

Lukas Sekanina
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Brno, Czech Republic
sekanina@fit.vutbr.cz

Vashek Matyas
Masaryk University
Faculty of Informatics
Brno, Czech Republic
matyas@fi.muni.cz

ABSTRACT

The security of cryptographic algorithms (such as block ciphers and hash functions) is often evaluated in terms of their output randomness. This paper presents a novel method for the statistical randomness testing of cryptographic primitives, which is based on the evolutionary construction of the so-called randomness distinguisher. Each distinguisher is represented as a Boolean polynomial in the Algebraic Normal Form. The previous approach, in which the distinguishers were developed in two phases by means of the brute-force method, is replaced with a more scalable evolutionary algorithm (EA). On seven complex datasets, this EA provided distinguishers of the same quality as the previous approach, but the execution time was in practice reduced 40 times. This approach allowed us to perform a more efficient search in the space of Boolean distinguishers and to obtain more complex high-quality distinguishers than the previous approach.

CCS CONCEPTS

• **Security and privacy → Information-theoretic techniques**; • **Computing methodologies → Search methodologies**; • **Theory of computation →** *Pseudorandomness and derandomization*;

KEYWORDS

Boolean function, genetic algorithm, statistical randomness testing

ACM Reference Format:
Vojtech Mrazek, Marek Sys, Zdenek Vasicek, Lukas Sekanina, and Vashek Matyas. 2018. Evolving Boolean Functions for Fast and Efficient Randomness Testing. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205518

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205518

1 INTRODUCTION

In past, evolutionary algorithms (EAs) have been applied in the design, optimization and analysis of cryptographic algorithms, such as block ciphers, hash functions and pseudo-random generators [9]. In this paper, we present a new application for evolutionary computation – an evolutionary construction of Boolean functions operating as simple, but high-quality *distinguishers* of statistical (non)randomness in the data generated by cryptographic algorithms.

Cryptographic algorithms always have to go through elaborate testing by skilled experts. History contains many examples of serious flaws in cryptographic algorithms [8, 16, 17], that rise concerns about the design and strength of the algorithms. These boost the research in probing cryptographic algorithms for vulnerabilities, finding design weaknesses or even hidden backdoors. Cryptanalysis conducted by a skilled human cryptanalyst is by far the most successful approach to asses overall security of an algorithm. However, some automation is possible in the first phases of a cryptanalysis, e.g., by using randomness testing suites such as the *NIST Statistical Test Suite* (STS) [12] or *Dieharder* [1]. These tests can be applied to check statistical properties of cryptographic algorithm output, and to look for specific features indicating a deviation from randomness. Such defect signalizes a potential flaw in the algorithm design. Yet such testing suites are limited only to predefined pattern testing for certain statistical defects, therefore others will go unnoticed.

It is crucial from the security point of view that cryptographic algorithms generate unbiased, statistically random data in target applications.

In order to test this property, various cryptanalytic techniques are usually combined with statistical testing. Common statistical test suites such as NIST STS, Dieharder and TestU01 examine the

correlation between the bits produced by a cryptographic primitive. They compare statistics of the bits and expected statistics for random bits using tens of empirical tests of randomness.

A recent paper [14] showed that a carefully constructed Boolean functions can provide the same results, but much faster and using lower data volumes in comparison with commonly used statistical test suites. Boolean functions were represented in an Algebraic Normal Form (ANF), i.e., its logic terms were summed by means of the exclusive-or operation. These Boolean functions were constructed heuristically and optimized by means of a brute force search, where the degree of the polynomial, the number of terms, and literals forming each term were typically optimized. The quality of each distinguisher was measured in terms of the so-called Z-score (see Sect. 3.3) and evaluated by means of several complex datasets coming from real cryptographic algorithms.

In this paper, we propose an evolutionary algorithm to construct a statistical distinguisher (Boolean function) showing desired properties. As suggested by [14], the target Boolean function is represented by an ANF expression and encoded by means of a bit string determining the number of terms and particular literals in each term. We could use a genetic programming approach and encode candidate Boolean functions as syntactic trees, but there are several reasons for using ANF: (i) A simple polynomial is easier to interpret. An understandable solution would help us to identify a real source of the bias present in a concrete cryptographic primitive and possibly to create a better design of the primitive. (ii) We can directly compare our results with [14]. (iii) A similar work dealing with GP led to a significantly more computationally intensive approach [13].

In addition to the evolution of Boolean functions in the ANF, we also combined our EA with the brute force algorithm developed in [14]. All the considered approaches were evaluated using several instances of real-world data streams (10 MB and 100MB files) and compared with commonly used statistical test suites. Finally, we performed a statistical analysis of all tested algorithms to highlight the contributions of employing the proposed evolutionary approach, which primarily lies in shortening the construction time of a distinguisher and improving the quality of results in comparison with [14].

2 PRELIMINARIES

A Boolean function of n variables can be viewed as a map from $\mathbb{F}_2^n$ into $\mathbb{F}_2$, where $\mathbb{F}_2^n$ is the vector space of n-tuples over the field $\mathbb{F}_2$ with the usual addition operation (denoted by $\oplus$). The set of all Boolean functions of n variables $\mathcal{B}_n$ forms a linear space of dimension 2^n over $\mathbb{F}_2$, so $|\mathcal{B}_n| = 2^{2^n}$. Hence every Boolean function $f \in \mathcal{B}_n$ can be represented by its *truth table*, i.e. the binary sequence $(f(\mathbf{0}), f(\mathbf{1}), \ldots, f(\mathbf{2^n - 1}))$ of length 2^n, where $\mathbf{i}$ denotes the binary expansion of the integer i over n bits. Truth table is a well-known canonical representation of the Boolean functions that uniquely identifies a given Boolean function. There exists, however, another canonical representation. Every Boolean function $f \in \mathcal{B}_n$ can be uniquely represented by a polynomial in n variables:

$$f(x_1, \ldots, x_n) = \bigoplus_{\mathbf{I} \in \mathcal{P}(N)} a_I x^I = \bigoplus_{\mathbf{I} \in \mathcal{P}(N)} a_I \prod_{i \in I} x_i,$$

where $\mathcal{P}(N)$ denotes the powerset of $N = \{1, \ldots, n\}$, n denotes the number of variables, and $a_I \in \{0, 1\}$. The polynomial is known as *Algebraic Normal Form* and the product x^I is denoted as *monomial*. The *algebraic degree* of f, denoted by $\deg(f)$, is the maximum of the degrees of the monomials of its ANF. Considering ANF representation and lexicographically ordered powerset $\mathcal{P}(N)$, every Boolean function can uniquely be represented by the binary sequence $(a_0, a_1, \ldots, a_N)$ of length 2^n, where the coefficient a_0 corresponds with the empty set and a_N corresponds to the set N. The polynomial representation helps to easily determine some cryptographic properties. For example, a Boolean function is an affine function if its algebraic degree is at most 1.

Example 1. Let us consider a Boolean function g of three variables:

$$g(x_1, x_2, x_3) = \overline{x_1} x_2 \vee x_2 x_3$$

This function is uniquely represented by truth table (0,0,1,0,0,0,1,1), where the first item corresponds with the output value $g(0, 0, 0)$ and the last item corresponds with $g(1, 1, 1)$. The given Boolean function can be represented also by the ANF

$$g(x_1, x_2, x_3) = x_2 \oplus x_1 x_2 \oplus x_1 x_2 x_3$$

whose algebraic degree is $deg(g) = 3$ and which consists of three monomials ($|I| = 3$). Considering this definition of ANF, each Boolean function of three variables can be represented by a polynomial

$$\begin{aligned} f(x_1, x_2, x_3) \quad = \quad & a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \oplus a_{12} x_1 x_2 \oplus \\ & a_{13} x_1 x_3 \oplus a_{23} x_2 x_3 \oplus a_{123} x_1 x_2 x_3 \end{aligned}$$

To express g, only three coefficients have to be set to one, namely a_2, a_{12}, and a_{123}. Hence, the Boolean function is uniquely determined by $I = \{\{2\}, \{1, 2\}, \{1, 2, 3\}\} \in \mathcal{P}(\{1, 2, 3\})$ and it can be represented as $(a_0, a_1, a_2, a_3, a_{12}, a_{13}, a_{23}, a_{123}) = (0, 0, 1, 0, 1, 0, 0, 1) \in \mathbb{F}_2^8$.

3 RELATED WORK

Related work includes a relevant research on randomness testing and employing evolutionary computation for purposes of cryptography and randomness testing.

3.1 Randomness testing

Each standard statistical test examines randomness of data by looking at a specific feature (the number of ones, number of ones in blocks, etc.). Test is defined by a function F that computes the distribution (histogram) of a given feature within a data. Randomness is evaluated by comparing the observed distribution with a reference expected distribution precomputed for a truly random data. In fact, each standard test checks whether its defining function F forms a distinguisher.

A standard test examines randomness only according to this function (feature). Tests are grouped to testing suites (also called batteries) to provide more complex randomness analysis. NIST STS [12], Dieharder [1] (an extended version of the Diehard) and TestU01 [7] are the most commonly used batteries for statistical randomness testing.

While test batteries consist of a set of tests, their testing ability is quite limited since the function F is fixed for each test. However, there are countless features for which randomness can be examined and countless of distinguishers.

3.2 Evolutionary computation in cryptography

In the context of cryptography, EAs have been applied to solve quite a different set of problems, yet all showing a common property – they can be formulated as a search problem. Picek's recent tutorial [9] provides a list of tasks where EAs proved successful in cryptology, namely: to rapidly check whether some concept (e.g. formula) is correct, to assess the quality of some other method, to produce "good-enough" solutions, and to produce novel and human-competitive solutions. Due to the limited space, we provide only a few examples.

Pseudo-random generators have traditionally been evolved by the EA community (e.g., [15, 18]). One of the requirements for their use for cryptographic purposes is a low implementation cost. Cartesian genetic programming was employed to produce such type of pseudo-random generators [11].

Boolean functions with specific properties (for example, highly nonlinear, balanced, and correlation immune functions) are important in cryptography because they provide a source of non-linearity in building blocks of cryptographic algorithms such as S-Boxes. Genetic algorithms (e.g., for S-Box generation [6]) as well as genetic programming (e.g., for bent function [5] and high correlation immunity function [10] designs) were successfully applied, improving the state of the art results. A general scheme for the design of block ciphers by means of genetic programming was introduced in [4].

Hernandez-Castro and Barrero [3] proposed an approach that is the closest one to our work. Using a genetic algorithm, they generated a large number of pseudo-random numbers with different degrees of randomness and used them to evaluate a popular randomness test suite implemented in a software package named Ent. In total, they evaluated seven statistics (entropy, compression, chisquared, arithmetic mean, pierror, excess and correlation) associated to five tests, resulting in an observation that studied statistics are not completely independent and could be reduced to five statistics. This analysis was based on data streams with a maximum string length of 32 Kb.

3.3 Randomness testing using Boolean functions

Empirical tests of randomness are typically based on the statistical hypothesis testing. These tests evaluate the null hypothesis – "data being tested are random". Each test computes a specific statistic of bits or block of bits. A test checks whether the observed test statistic for analyzed sequence happens to be in the extreme (tail) parts of the null distribution (distribution of test statistic of random data). In such a case, the hypothesis is rejected, and data are considered as non-random. Formally, a test statistic is transformed to a *p-value* (using the null distribution) representing the probability that a perfect random number generator would have produced a sequence "less random" (i.e. more extreme according to the analyzed feature) than the tested sequence [12]. A small *p-value* (below 0.01) is typically interpreted as the tested data not being random.

In general, the empirical tests of randomness are based on the following steps. Firstly, a histogram of patterns for the given dataset is computed by the test. Then the histogram is reduced into a single value representing its "randomness" according to the analyzed feature. Finally, *p-value* is typically calculated from the observed test statistic using the null distribution.

Sys et al. [14] introduced an approach based on the construction of Boolean functions that are able to distinguish a given data from the random data. In particular, they looked for an empirical test of randomness defined by a polynomial representing Boolean function such that the test results in the smallest possible *p-value* (i.e., rejection of the hypothesis). The empirical test was constructed by generalization of the Monobit test that counts the number of ones (#1) and zeros (#0) in the analyzed binary sequences and examines whether the numbers are close to each other as it would be expected for random data. The data to be analyzed are divided to N non-overlapping blocks consisting of n bits. The blocks serve as inputs for function $f(x_1, \cdots, x_n)$. The sum #1 of results of f when applied to the blocks, is computed. The #1 together with the probability of evaluating f to one (denoted as p) is used to compute so called Z-score. Z-score was employed as it normalizes a binomial distribution of #1 [14]. In addition to that, it defines the statistical distance between observed and expected numbers of ones for random data. The *Z-score* is defined as:

$$Z\text{-}score = \frac{\#1 - pN}{\sqrt{p(1-p)N}}. \quad (1)$$

Sys et al. proposed to use the computed test statistic directly as the measure of the strength of distinguishers. A bigger value of observed statistic means stronger distinguisher and conversely.

4 EVOLUTION OF BOOLEAN FUNCTIONS FOR RANDOMNESS TESTING

In this section, we briefly describe the algorithm we used to evolve distinguishers for randomness testing based on Boolean functions represented in ANF.

4.1 Encoding

As we have discussed in Section 2, an ANF of a Boolean function on n variables can be uniquely represented by a binary string of fixed length. Unfortunately, this representation is impractical for large n because the number of bits grows exponentially with the increasing number of variables. Hence, we propose a different encoding.

In this paper, we will consider $deg(f)$ and $|I|$ as input parameters. For better clarity, let us denote these parameters as deg and k. While k determines the number of monomials employed in the ANF, deg limits their maximum acceptable degree. In order to encode a distinguisher satisfying deg and k, we employ a string of integers consisting of $deg \times k$ items. The items are divided into k tuples. Each tuple is associated with a single monomial and defines its inputs. In particular, the tuple contains deg items $(\alpha_1, \alpha_2, \ldots, \alpha_{deg})$, where $1 \leq |\alpha_i| \leq n$. The positive value of α_i determines the index of the input variable x_{α_i}. The negative value means that this item is ignored as it does not contribute to the associated monomial. This scheme enables to have a fixed string of integers (i.e. genotype) on the one hand, but variable phenotypes on the other hand. The monomials can have different degrees and some of them can be even completely switched off depending on the progress of the evolution.

Example 2. Let us suppose the following parameters: $n = 128$, $deg = 2$ and $k = 3$. Then, polynomial $f(x_1, \ldots, x_{128}) = x_{78} \oplus x_5x_{120} \oplus x_{27}x_{63}$ can be encoded using three tuples as *(5,120) (-12,78) (63,27)*. The first tuple encodes monomial x_5x_{120}. The second tuple contains only one positive element. Hence, it produces monomial x_{78}. Similarly to the first case, the last tuple also contains two positive integers that give us $x_{27}x_{63}$.

4.2 Search method

For the search in the search space induced by the chosen representation, we developed an evolutionary algorithm with the following properties. The population consists of a finite number of chromosomes divided into two parts – parents (μ individuals) and offspring (λ individuals). Each chromosome represents a candidate Boolean function using the proposed encoding. At the beginning of evolution, the first population consisting of $\mu + \lambda$ individuals is randomly generated and consequently evaluated. In order to create a new population, the fittest μ chromosomes are selected as new parents. New parents are deterministically selected from both the parents and offspring sets. The new population of λ individuals is then created by applying a *point mutation*. To avoid an additional parameter, we do not consider recombination. The steps of the evolution loop are repeated in the next generations until either a satisfactory solution is found, or a maximal generation count is reached.

The point mutation operator randomly changes up to h genes. It can either invert the sign of a chosen gene or replace its value by a randomly generated, but valid integer. Compared to the latter case, the sign can be inverted with much lower probability. The randomly generated integer must be within interval $[1, n]$, where n is the number of input variables. In addition to that, it is ensured that all genes within a single tuple have different values (each input variable occurs at most once).

4.3 Fitness function

The goal of the evolution is to find a distinguisher showing the highest possible Z-score for a given sequence of data samples $T = (t_1, \ldots, t_N)$, $t_i \in \mathbb{F}_2^n$, produced by a cryptographic function whose randomness is investigated. Let $p(f, D)$ be the ratio of the input samples from a dataset D for which a Boolean function $f(x_1, \ldots, x_n)$ evaluates to one:

$$p(f, D) = \frac{1}{|D|} \sum_{(v_1, \ldots, v_n) \in D} f(v_1, \ldots, v_n) \tag{2}$$

Then, the fitness value of a candidate Boolean function g (which equals to the Z-score) is calculated as follows

$$\text{fitness}(g, T) = \sqrt{|T|} \cdot \frac{|p(g, T) - \rho(g)|}{\sqrt{\rho(g) \cdot (1 - \rho(g))}}, \tag{3}$$

where $\rho(g)$ is the probability that g evaluates to one provided that an indefinite truly random sequence is used as input of g.

Typically, a random sequence $R = (r_1, \ldots, r_N)$, $r_i \in \mathbb{F}_2^n$, consisting of the same number of elements as T is employed to determine $\rho(g)$. It means that the probability is replaced with the relative number of samples and $\rho(g) = p(g, R)$. Unfortunately, this simplification can introduce a bias because the number of samples N is typically significantly lower than 2^n. In addition to that, it is nontrivial to obtain a truly random sequence. Hence, we propose to calculate $\rho(g)$ exactly using Reduced Ordered Binary Decision Diagrams (ROBDDs) [2]. Apart from the canonicity, the main advantage of ROBDDs is the ability to efficiently represent Boolean functions. Every library for ROBDD manipulation is typically equipped with two basic operations – *SatOne* and *SatCount*. *SatOne* determines whether there exists at least one input assignment a for which $f(a) = 1$. *SatCount* computes the number of input assignments $|A|$ for which $f(a_i) = 1$ where $a_i \in A$. Let G denote a ROBDD which represents a Boolean function g. Then, $\rho(g)$ can be determined as $\rho(g) = \frac{1}{2^n}\text{SatCount}(G)$. The effect of the proposed BDD-based approach is twofold. Firstly, many operations over ROBDDs can be performed in linear time with respect to the ROBDD size. This is the case of *SatOne* as well as *SatCount* operation. As a consequence of that, BDDs can significantly speedup the process of evaluation. Secondly, this approach guarantees that all 2^n input combinations are considered. As there is no bias, more accurate results can obtained compared to the approach based on a random sequence R. Example of fitness computation is given in Figure 1.

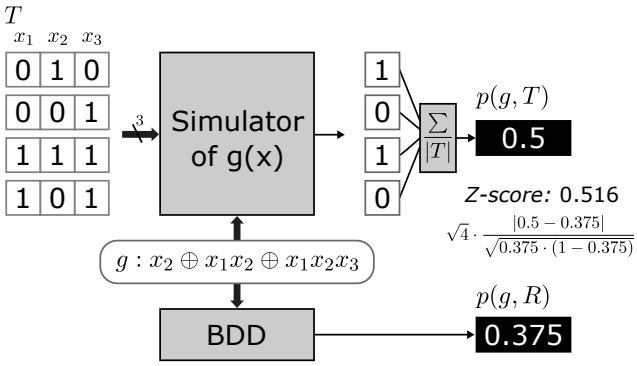

Figure 1: Principle of determining the fitness function (Z-score) of Boolean function g represented using ANF $x_2 \oplus x_1x_2 \oplus x_1x_2x_3$ and sequence T consisting of four samples.

5 EXPERIMENTAL SETUP

In order to fairly evaluate performance of the proposed evolutionary approach, we considered five different strategies that are illustrated in Figure 2.

Firstly, we implemented the heuristic approach based on the brute force (BF) search proposed in [14]. As it is intractable to determine Z-score of all 2^{2^n} existing distinguishers, the authors proposed to employ a two-phase strategy. The goal of the first phase was to enumerate monomials whose degree is lower than four, identify monomials that exhibit the best Z-score and create a set of top 10/100/1000 monomials. In the second phase, more complex distinguishers were constructed from these monomials. The monomials were combined together to obtain a Boolean function $f = b_1 \oplus \cdots \oplus b_k$. Originally, $k \leq 3$ was considered only. In this paper, we extended this limit to $k \leq 20$. Since the brute-force search strategy is employed in both phases, we denote this approach as BF/BF.

Secondly, we implemented the evolutionary method as described in Section 4 (denoted as EVO). Compared to the BF/BF approach, it is

not necessary apply a two-phase approach because EVO can directly encode more complex distinguishers. To define some bounds, we considered ANFs consisting of up to 20 monomials whose degree is less than 10. In addition to that, we implemented another three variants of a two-stage design process which combines BF and EVO. In the first variant, we replaced the second phase of BF/BF by EVO. Compared to BF/BF, this modification denoted as BF/EVO enables to consider more monomials. We employed up to 10000 monomials. In the remaining two variants, we replaced the first phase by EVO. The main advantage is that we can combine monomials of higher degree. In the second phase, we employed either BF or EVO. The difference is in the number of monomials since it is infeasible to enumerate all distinguishers having $k > 10$. Hence, EVO/BF is limited to $k \leq 10$.

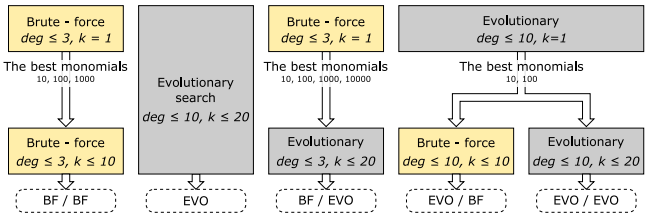

Figure 2: Five different strategies evaluated in this paper. BF stands for Brute-force search, EVO denotes the proposed evolutionary approach.

The method is evaluated on the data generated by means of the stream cipher (RC4), block cipher (AES) and hash functions (SHA-256, MD6, Keccak). In order to be able to find a distinguisher, we intentionally limited the number of rounds of the hash functions (SHA-256 - 3 rounds, MD6 - 8 and 9 rounds, Keccak - 3 rounds) as well as AES (3 rounds). The test data were generated as the standard keystream for RC4. For the remaining cases, a special stream consisting of 128-bit blocks of minimal Hamming weight was employed at the inputs. The Sequence starts with block $B_0 = "00\cdots0"$. Each of next 128 blocks consists of 127 bits "0" and one bit "1" on different positions i.e. $B_1 = "00\cdots01", B_2 = "00\cdots01", \cdots, B_{128} = "100\cdots"$. Next blocks consist of 126 bits "0" and 2 bits of the value "1" etc. The test data were generated as blocks of Sequence processed by one of the functions (SHA-256, etc) separately. In addition to that, a sequence of random data obtained form /dev/urandom is considered. In all cases, 100 MB of data were generated which corresponds with $|T| = 6.25 \cdot 10^6$ test samples.

The parameters were chosen as follows: $\mu = 1$, $\lambda = 4$, up to 5 genes are mutated. The goal was to find a distinguisher for $n = 128$ input variables. The evolution was terminated when the maximal number of evaluations $g_{max} = 10^4$ was exceeded. This setting corresponds with approx. 25 minutes of evolution on Intel Xeon CPU running at 2.4 GHz for the most complex problem (i.e. deg=10, k=20). For each deg and k, 10 independent runs were executed. A highly-optimized 64-bit parallel simulator was employed to determine response of a candidate distinguisher to given input data.

The evolution used in the second phase was executed with a different mutation operator. Instead of mutating a particular gene(s), the whole tuple was replaced by a monomial randomly chosen from the set of available monomials.

6 RESULTS

6.1 Performance of the BDD-based fitness

In the first experiment, we evaluated the performance of the BDD-based fitness with respect to a näive implementation based on a simulator and a set of random vectors R. In order to simplify the problem, we employed Monte-Carlo simulation producing the same number of random vectors as used in T. In both cases, we measured the total time needed to determine the fitness, i.e. Z-score. The results are shown in Figure 3.

As the time required to determine the fitness score grows with increasing k as well as with increasing deg, we can use the product $deg \times k$ as a complexity indicator. Figure 3 shows that the BDD-based method is substantially faster. Compared to the simulation, the time of evaluation increases only slightly with the increasing complexity. In addition to that, the variation in time of BDD evaluation is substantially lower. The time of simulation varies significantly even for the same deg and k.

The proposed BDD-based approach not only scales better but also provides accurate and unbiased results. For the most complex challenging combination $deg \times k$, the average speedup is better than 3x.

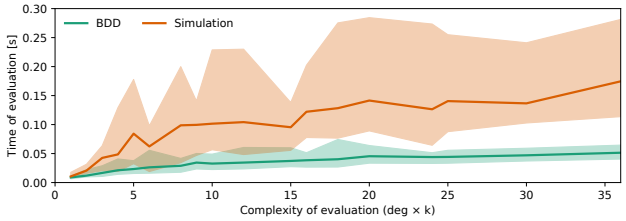

Figure 3: The average time required to evaluate fitness for a) BDD-based fitness and b) simulation obtained from 10 independent evolutionary runs. The maximum as well as minimum time is depicted using the solid area.

6.2 Searching for monomials having $deg \leq 3$

In the second experiment, we investigated the efficiency of the evolutionary approach compared to the brute-force approach employed in the first stage. In order to do that, we took the results discovered by EA executed for $k = 1$ and $deg \in \{1, 2, 3\}$. The results are shown in Figure 4. Due to the limited space, only four data sets are presented.

Since it is possible to enumerate and evaluate all possible distinguishers consisting of exactly one monominal of degree lower than 4 (there exists 349 632 different distinguishers), we can easily identify Z-score of the best distinguisher. In addition to that, we are able to calculate the distribution of Z-score of all distinguishers. Figure 4 is showing that the distribution is very similar independently of the chosen dataset. More than 99.79% of distinguishers exhibit Z-score worse than half of the highest possible Z-score. This indicates a complex and nontrivial search space. Interestingly, the proposed evolutionary approach produces solutions whose Z-score is in the second half of the range (see the bottom part of Figure 4). This is also evident from the convergence curves that are shown in the upper

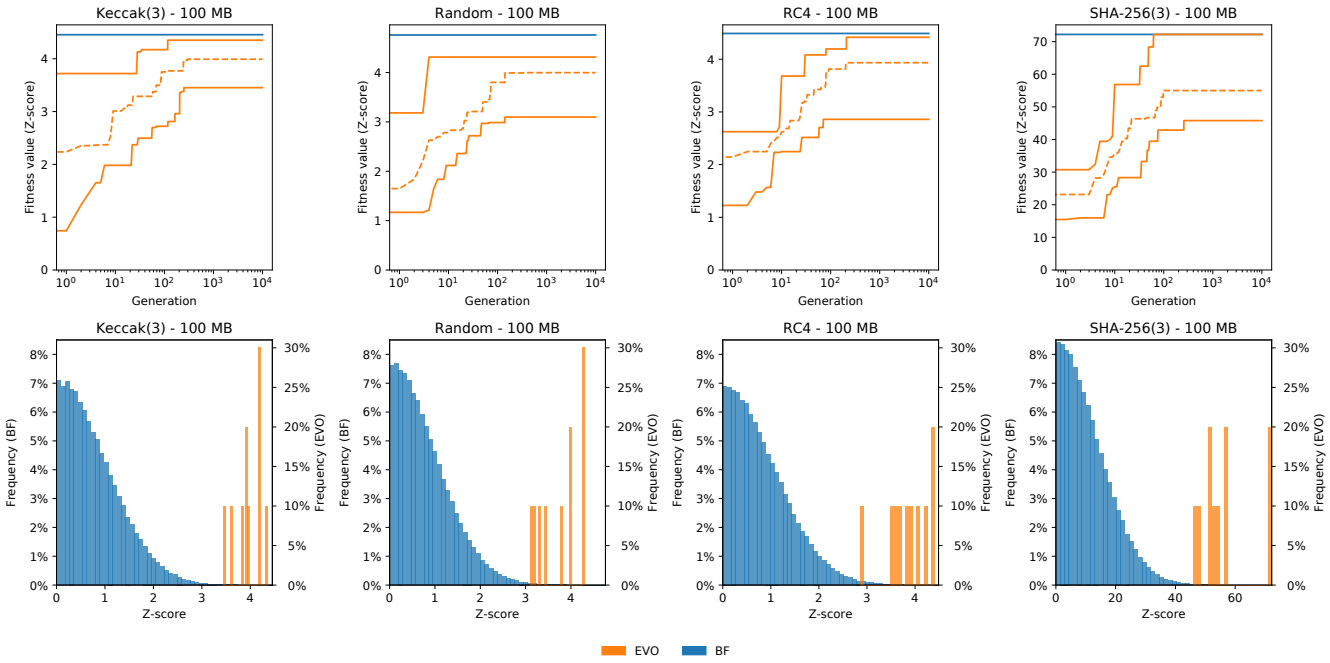

Figure 4: Design of monomials having $deg \leq 3$. The convergence curves of the proposed evolutionary method (top) and the distribution of Z-score (bottom) of the discovered distinguishers. Z-score of the worst/ best candidate solutions is shown using solid line, the average fitness value calculated from 10 independent evolutionary runs is shown using dash line.

part. Z-score of the distinguishers discovered by EA is nearly the same as the best known one. In case of random data and Keccak(3), however, the evolution was unable to find the optimal solution. We analyzed this issue and discovered that there exists only two distinguishers that have higher Z-score compared to the best evolved solution. These results are encouraging because the overall time of computation was reduced 8.7x. Note that the brute-force approach required to evaluate nearly $35 \cdot 10^4$ candidate distinguishers while the EA was terminated after 10^4 generations which corresponds with $4 \cdot 10^4$ evaluations. In most cases, only approx. 10^2 generations were in fact required as visible on convergence curves in Figure 4.

6.3 Searching for distinguishers having $k = 3$

We took the monomials obtained in the previous experiment and identified those exhibiting the best Z-score. In particular, we created three sets: S_{10} consisting of top ten distinguishers, S_{100} and S_{1000} containing the best 100 and 1000 distinguishers respectively. As it is nontrivial to directly compare the quality of distinguishers having different degree and different k, let us restrict our analysis to a single configuration, in particular, $k = 3$ and $deg = 3$. This configuration enables the exhaustive BF search only on S_{10} and S_{100}. There exists 161700 distinguishers consisting of three distinct monomials when we consider S_{100}. Similarly to the previous experiment, we are able to evaluate Z-score of all ANFs and identify the highest Z-score that can be achieved. To fairly compare our approach to the two-stage BF/BF approach, we run the EA for $k = 3$ and $deg \leq 3$. We used exactly the same setup as in the previous case, i.e. 10^4 generations. It means that the evolution is allowed to evaluate only 40000 candidate distinguishers which equals approximately to a quarter of the number of evaluations that have to be considered during the second phase of BF. Please note that the first phase of BF/BF requires itself another $35 \cdot 10^4$ evaluations. Hence, the evolution is forced to use nearly 12.7 times less evaluations. As we have $k > 1$, we can employ even the BF/EVO approach. For simplicity, the same number of generations is utilized. The results are summarized in Figure 5 showing the histograms with the distribution of Z-score of the distinguishers for each method.

For the purpose of our comparison, we also included Z-score of the top 100 monomials from S_{100}. Interestingly, it can be observed that the combination of these monomials does not guarantee that the obtained distinguisher will exhibit higher Z-score. With exception of the SHA-256(3) (intentionally limited to execute only three rounds), the monomials have the Z-score around 4. When we combine three of these monomonials (the result of BF/BF), it might happen that the value of Z-score doubles but the probability is very low (typically 0.005%). In most cases, the Z-score is either slightly better or substantially worse. We can observe two peaks – one peak at Z-score around 4.42 and the second around 1.41. This analysis confirms our previous observation related to the complexity of the search space. There is a very low occurrence of high-quality solutions. For Keccak(3), for example, 85% distinguishers consisting of three monomials have Z-score worse than the best monomial. If we sort the distinguishers according to the Z-score, we can determine that only less than 1.21% of all distinguishers (in average) are in the quartile of Z-score representing best solutions. In the case of SHA-256(3), there is only one peak and the disproportion is even worse.

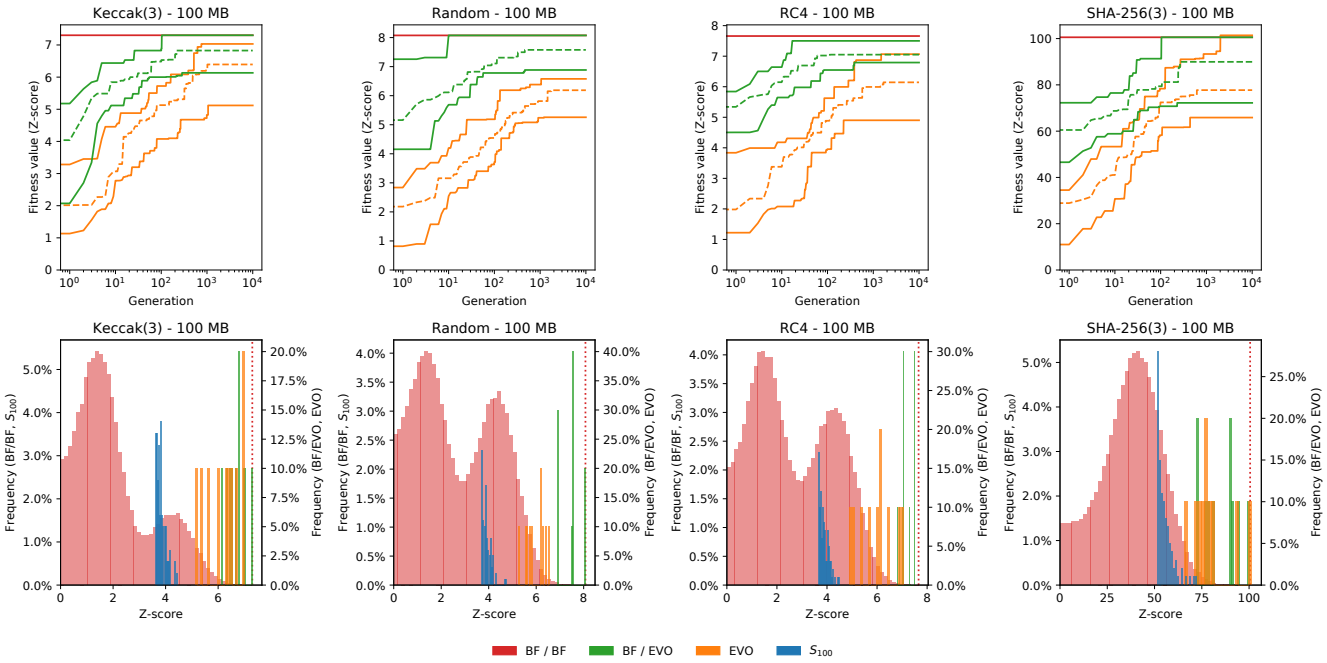

Figure 5: Design of polynomials having $k = deg = 3$. The convergence curves of the proposed evolutionary method as well as two-stage BF/EVO (top) and the distribution of Z-score (bottom) of the discovered distinguishers. The second stage of BF/BF and BF/EVO selected the monomials from S_{100}.

Less than 0.31% of all distinguishers occupy the first quartile. Even in this case the evolution provides very good solutions that occupy the last quartile. It seems that the two-phase approach BF/EVO is able to deliver better results than the single-phase EVO. This is evident even on the convergence curves. However, we cannot directly compare EVO and BF/EVO even though the convergence curves for EVO and BF/EVO are shown in a single plot. It is necessary to consider that BF/EVO is a two-stage process requiring substantially more evaluations. EVO exhibits the worse performance on the random data – there are 211 distinguishers having the Z-score higher than the best-discovered one. In the case of RC4, the best evolved solution is dominated by 29 other solutions. The Z-score converges relative quickly. In average, $1.3 \cdot 10^3$ ($2.3 \cdot 10^3$) generations are required by EVO (BF/EVO) until the evolution sticks and stops to improve the Z-score.

6.4 Evaluation on more complex instances

The goal of this section is to evaluate the quality of the discovered solutions across the considered scenarios. For each scenario and each setting, we determined the polynomials with the best Z-score. As evident from Equation 1, Z-score increases with the increasing number of test samples. This does not represent any problem because we are using the same data sets across all the experiments. We observed, however, that the Z-score also increases with the increasing degree. One possible explanation of this phenomenon is as follows. It seems that the higher degree permits the evolution to search for more complex distinguishers capable of discovering more tricky dependencies among the bits.

Table 1: Z-score of the best discovered distinguishers having degree $deg \leq 3$. The best results are typed in bold.

Dataset T	EVO	BF / EVO				BF / BF		
		S_{10}	S_{100}	S_{1000}	S_{10000}	S_{10}	S_{100}	S_{1000}
k_{max}	20	10	20	20	20	10	3	2
Keccak(3)	**9.5**	5.3	7.8	8.9	8.7	5.3	7.3	6.6
MD6(8)	**43.9**	28.5	31.0	31.0	31.0	31.0	31.0	31.0
SHA-256(3)	**126.3**	91.4	100.6	107.7	111.1	91.4	100.6	93.8
RC4	**9.8**	6.2	8.0	8.9	8.8	6.2	7.7	6.6
MD6(9)	**10.1**	5.9	7.7	8.5	8.5	5.9	7.5	6.7
AES(3)	**22.1**	12.0	17.4	20.9	20.1	12.0	15.6	13.8
random	**9.8**	7.5	8.5	8.9	9.1	7.5	8.1	6.8

To fairly evaluate the method, we divided the results in two parts. The first part consists of polynomials of small degree ($deg \leq 3$) and the second part of more complex ones ($3 < deg \leq 10$). The results are summarized in Table 1 and Table 2 . BF/BF does not scale well even for the polynomials of the small degree. If we increase the number of monomials that can be employed in the second phase, we have to decrease k to be able to enumerate all the existing solutions in a reasonable time. As a consequence of that, we can see that BF/BF for S_{100} produces solutions that are even worse than those for S_{10} even though S_{10} is included in S_{100}. This result indicates that higher k leads to more efficient distinguishers. The same behavior can also be observed for BF/EVO when we look at the results for S_{10} and S_{100}. The key difference between those two settings is that k increased

Table 2: Z-score of the best discovered distinguishers having degree $3 < deg \leq 10$. The best results are typed in bold.

Dataset	EVO	EVO / EVO		EVO / BF	
T		S_{10}	S_{100}	S_{10}	S_{100}
k_{max}	20	10	20	10	3
Keccak(3)	18.19	12.90	**18.77**	12.90	9.90
MD6(8)	**54.08**	38.53	39.75	38.53	39.10
SHA-256(3)	**467.83**	318.75	351.26	318.75	303.35
RC4	**18.24**	13.20	18.06	13.20	10.33
MD6(9)	**18.12**	11.75	17.83	11.75	9.55
AES(3)	**37.66**	27.77	36.92	27.77	24.99
random	**17.95**	11.97	17.10	11.97	9.93

from 10 to 20 since k is bounded by $|S_x|$. When we increase the number of monomials, the Z-score increases but only slightly. In contrast to these results, EVO discovered distinguishers that have substantially higher Z-score. The average relative improvement of Z-score is around 32%.

Similar observations can be made for the results summarized in Table 2 . As there exists more than $2.5 \cdot 10^{14}$ different monomials having its degree equal to ten, it is intractable to perform the exhaustive BF search and identify the best monomials. As a consequence of that, is impossible to run BF/BF. Hence the table contains only results discovered by the proposed evolutionary approach and its two hybrids. Apart from Keccak(3) where the difference is negligible, the proposed evolutionary approach produces the best results. If we compare the results with those given in Table 1 (see column EVO in both tables), it is evident that not only the higher k but also the higher deg implies higher Z-score.

The obtained results are consistent with our expectation. We know that three rounds of SHA-256 generate a very weak sequence that is far from a truly random sequence. This fact was evident even when we applied only monomials of a small degree. The best Z-score on SHA-256 is much higher compared to the other sequences (see Figure 4). Table 2 confirms this evidence. The Z-score is about 25 times higher compared to the Z-score obtained on the random sequence. In addition to that, it is apparent that neither eight rounds of MD6 nor three rounds of AES generate random data of high quality. The remaining sequences have the Z-score comparable with random data.

The average time needed to complete a single run of EA (i.e. 10,000 generations across all settings of deg and k) is 14 min. (64 min. is the worst case) on a common CPU running at 2.4 GHz. However, this setting was used to provide a sufficient time to study the convergence properties of the EA. In fact, the average number of generations to converge to a stable Z-score is only 1,300 generations, i.e. 107 s (the worst case is 8 min.). The BF/BF needed more than 108 minutes for S_{1000} and 53 minutes for S_{100}. In a real practice, the proposed approach is thus capable of providing a good-quality distinguisher 40 times (20 times for S_{100}) faster than BF/BF.

7 CONCLUSIONS

We addressed a challenging problem of the statistical randomness testing of cryptographic primitives. Following the approach in which randomness distinguishers are constructed as Boolean functions in ANF, we developed an EA-based method capable of discovering high-quality distinguishers in a short time. On seven real-world datasets, the EA provided distinguishers of the same quality as the previous brute force approach, but the execution time was reduced by one order in the magnitude. Moreover, the proposed method allowed us to construct more complex distinguishers utilizing higher polynomial degrees which is intractable by the brute force approach. Our future work will be focused on a detailed evaluation of the proposed method on even more challenging cryptography primitives with the ultimate goal of revealing their potential weaknesses.

ACKNOWLEDGMENTS

This work has been supported by the Czech Science Foundation project No. 16-08565S.

REFERENCES

[1] Robert G Brown, Dirk Eddelbuettel, and David Bauer. 2013. Dieharder: A random number test suite 3.31.1. *http://www.phy.duke.edu/~rgb/General/dieharder.php* (2013).
[2] R. Ebendt, G. Fey, and R. Drechsler. 2000. *Advanced BDD Optimization.* Springer.
[3] J. Hernandez-Castro and D. F. Barrero. 2017. Evolutionary generation and degeneration of randomness to assess the indepedence of the Ent test battery. In *2017 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 1420–1427.
[4] J. C. Hernandez-Castro, J. M. Estevez-Tapiador, et al. 2006. Wheedham: An Automatically Designed Block Cipher by means of Genetic Programming. In *2006 IEEE Int. Conf. on Evolutionary Computation.* IEEE, 192–199.
[5] Radek Hrbacek and Vaclav Dvorak. 2014. Bent Function Synthesis by Means of Cartesian Genetic Programming. In *Parallel Problem Solving from Nature - PPSN XIII.* Springer Verlag, 414–423.
[6] Georgi Ivanov, Nikolay Nikolov, and Svetla Nikova. 2016. Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties. *Cryptography and Communications* 8, 2 (2016), 247–276.
[7] Pierre L'Ecuyer and Richard Simard. 2007. TestU01: A C Library for Empirical Testing of Random Number Generators. In *ACM Trans. Math. Softw.*, Vol. 33. ACM, New York, NY, USA, Article 22.
[8] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. 2017. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *Proc. 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS '17).* ACM, New York, NY, USA, 1631–1648.
[9] Stjepan Picek. 2016. Evolutionary Computation and Cryptology. In *Proc. 2016 Genetic and Evolutionary Computation Conference Companion.* ACM, 883–909.
[10] Stjepan Picek, Claude Carlet, Sylvain Guilley, Julian F. Miller, and Domagoj Jakobovic. 2016. Evolutionary Algorithms for Boolean Functions in Diverse Domains of Cryptography. *Evol. Comput.* 24, 4 (2016), 667–694.
[11] Stjepan Picek, Dominik Sisejkovic, et al. 2016. Evolving Cryptographic Pseudo-random Number Generators. In *Proc. 14th Int. Conf. Parallel Problem Solving from Nature.* Springer, 613–622.
[12] A. Rukhin. 2010. A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications, Version STS-2.1. *NIST Special Publication 800-22rev1a.*
[13] Petr Svenda, Martin Ukrop, and Vashek Matyas. 2013. Towards Cryptographic Function Distinguishers with Evolutionary Circuits. In *SECRYPT 2013 - Proc. 10th Int. Conf. on Security and Cryptography.* SciTePress, 135–146.
[14] Marek Sýs, Dusan Klinec, and Petr Svenda. 2017. The Efficient Randomness Testing using Boolean Functions. In *Proc. 14th Int. Joint Conf. on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT.* SciTePress, 92–103.
[15] M. Tomassini, M. Sipper, and M. Perrenoud. 2000. On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Trans. Comput.* 49, 10 (2000), 1146–1151.
[16] Mathy Vanhoef and Frank Piessens. 2015. All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS. In *24th USENIX Security Symposium (USENIX Security 15).* USENIX Association, Washington, D.C., 97–112. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/vanhoef
[17] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. 2005. *Efficient Collision Search Attacks on SHA-0.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–16.
[18] Y. Wang, H. Wang, A. Guan, and H. Zhang. 2009. Evolutionary Design of Random Number Generator. In *2009 International Joint Conference on Artificial Intelligence.* IEEE, 256–259.