New results on reduced-round Tiny Encryption Algorithm using genetic programming

Karel Kubíček, Jiří Novotný, Petr Švenda, Martin Ukrop
Masaryk University, Brno, Czech Republic
{karel.kubicek, jiri.novotny, xsvenda, mukrop}@mail.muni.cz

***Abstract*—Analysis of cryptoprimitives usually requires extensive work of a skilled cryptanalyst. Some automation is possible, e.g. by using randomness testing batteries such as Statistical Test Suite from NIST (NIST STS) or Diehardar. Such batteries compare the statistical properties of the function's output stream to the theoretical values. A potential drawback is a limitation to predefined tested patterns. However, there is a new approach – EACirc is a genetically inspired randomness testing framework based on finding a dynamically constructed test. This test works as a probabilistic distinguisher separating cipher outputs from truly random data.**

In this work, we use EACirc to analyze the outputs of Tiny Encryption Algorithm (TEA). TEA was selected as a frequently used "benchmark" algorithm for cryptanalytic approaches based on genetic algorithms. In this paper, we provide results of EACirc applied to TEA ciphertext created from differently structured plaintext. We compare the methodology and results with previous approaches for limited-round TEA. A different construction of EACirc tests also allows us to determine which part of cipher's output is relevant to the decision of a well-performing randomness distinguisher.[1]

***Index Terms*—randomness statistical testing, TEA, genetic algorithms, randomness distinguisher, software circuit**

I. Introduction

Automatized randomness testing is useful for checking one of the expected cipher properties – output ciphertext should be indistinguishable from a stream of truly random data. This property alone is not sufficient for a cipher to be secure, but the ability to distinguish ciphertexts from random data constitutes an important hint on potential cipher weaknesses.

The common way to automate testing of randomness is using statistical batteries. NIST STS [1] is a standard battery of tests commonly used for this purpose, together with other batteries such as Diehard [2], Dieharder [3] or TestU01 [4]. The batteries contain sets of fixed tests (usually parameterized to form multiple different subtests) checking expected statistical properties of tested output stream (TEA ciphertext in our case) in comparison to the expected values for truly random data. Empirical tests of randomness fall under the standard statistical model – statistical hypothesis testing. Tests assume the assessed bitstream is random (the null hypothesis) and try to reject it (to show the bitstream is not random). Each randomness test is defined by the test statistic S, which is a real-valued function of a numeric sequence. Tests are evaluated by comparing the p-value (computed from the test statistic) with a chosen significance level α. For the p-value computation, it is necessary to know an exact distribution of the statistic S under a valid null hypothesis or, at least, its close approximation.

The limitation of the standard batteries for randomness testing is the fact they implement a fixed set of tests and can detect only a limited set of patterns and statistical irregularities. If the used set of tests is fixed and known, a sequence of completely deterministic data can be crafted such that no tests will detect statistically significant deviances from truly random data. However, as cryptographic functions have a deterministic output (dependent only on input data and a key), it is a priori expected that the function output cannot pass all possible tests of randomness and so there exist tests that reveal the output sequence as non-random. However, such a test can be very difficult to find.

In this work we use EACirc [5], a novel framework for constructing empirical tests of randomness that can succeed in finding such a test (at least hypothetically). Our goal is to find an empirical test of randomness that indicates if a given sequence is either non-random (with a high probability) or sufficiently indistinguishable from a truly random data stream. In this framework, randomness tests are created iteratively, adapting to the processed sequence. The construction is stochastic and uses genetic programming [6]. The tests are constructed from a predefined pool of operations. A set of these operations, together with a limit on the total number of operations, allows us to control the complexity of the constructed tests. The framework theoretically enables us to build an arbitrary randomness test over a set of chosen operations (in practice, however, the total number of operations used is limited). Therefore, it can be viewed as a general framework for the test construction and it could (hypothetically) provide a better detection ability than standard tests.

TEA has been intensively analyzed, including randomness testing of cipher output with stochastic genetic algorithms. Capabilities of EACirc are compared with previous results as well as conventional statistical batteries.

This paper is organized as follows: section II introduces TEA as a simple encryption algorithm applied nowadays as a benchmark for randomness tests. Subsequently, section III contains information about EACirc with the definition of used settings. Input data structure is also discussed in this section. Results and their interpretation are presented in section IV with analysis of found distinguishers and performance and data usage of EACirc. In section V, we describe the future work.

[1]Paper supplementary material available at http://crcs.cz/papers/infocomm2016

II. Tiny Encryption Algorithm

Tiny Encryption Algorithm (TEA) is a block cipher designed by David Wheeler and Roger Needham [7]. The algorithm was designed to have a simple structure based on the Feistel network with 32 rounds (we count two steps of Feistel network as 1 TEA round). The cipher uses plaintext blocks of 64 bits and keys of 128 bits.

A. TEA distinguishers – state of the art

Nowadays, TEA is not considered secure for regular use as it suffers from multiple weaknesses, most significantly the related-key attack [8]. However, it is frequently used as a benchmark for randomness testing using genetic algorithms. Starting in 2002 with a paper by Julio C. Hernández, José M. Sierra, Pedro Isasi and Arturo Ribagorda [9], statistically significant deviances were found for TEA limited to 1 and 2 rounds. Fixed bitmask with high Hamming weight evolved by genetic algorithms was applied both to the cipher input data and key. The expected distribution of bit patterns of 10 least significant bits of ciphertexts were then evaluated with a χ^2 test. A similar team published new results with the same approach but improved settings of genetic algorithms [10], which also detected deviances for 3 and 4 rounds. Aaron Garrett, John Hamilton and Gerry Dozier [11] extended this work in 2007 with new optimizations of the fitness function, which helped to create masks with a higher weight for 1 and 2 rounds TEA but failed to surpass previous results for 3 and 4 rounds.

Wei Hu et al. [12] in 2010 used quantum inspired genetic algorithm and a similar approach with bitmasks and succeeded with TEA limited to 4 and 5 rounds. Eddie Yee-Tak Ma and Chalie Obimbo [13] realized an attack on TEA limited to 1 round in 2011 utilizing genetic algorithms and harmony search for the derivation of degenerated keys instead of detection of statistical deviances of output.

III. Our approach

A. Randomness testing with genetic programming

As stated in the section I, the common way of automating randomness testing is the use of statistical batteries with predefined tests such as NIST STS. The approach based on genetic algorithms is different, as used tests iteratively evolve and adapt to the presented data.

Firstly, a set of individuals is created with each representing a candidate distinguisher function. Secondly, every individual decides if the provided block of input data is random or non-random. Thirdly, as the correctness of the decision is known, better individuals can be selected. Individuals are randomly mutated or cross-bred to create (hopefully) better descendants. The process follows the principles of biological evolution. I.e. if ciphertexts share a common statistical property (e.g. correlation between i^{th} and j^{th} bit), then an individual capable of expressing this property can potentially be evolved and improved in the process of further evolution.

The use of genetic algorithms also induces a couple of disadvantages. We are affected mainly by these:

- As the representation of the distinguisher functions is not straightforward, there are many possible candidate configurations. This induces a search space that may be artificially and unnecessarily large if the representation is not properly selected.
- Genetic modifications of candidate solutions from the previous iteration (mutation, crossover) are done randomly, and configuration space may not be completely searched. A well-working distinguisher can be missed even if it exists.
- The process of fine-tuning the parameters of genetic algorithms can significantly influence the quality of distinguishers found. E.g. [10] found distinguisher for a higher number of rounds then [9] although using same underlying approach and representation.

For more details about possible problems and their solution in EACirc, refer to the thesis of Martin Ukrop [14], section 3.1.

B. EACirc framework

The constructed distinguisher is a small program that simulates a standard hardware circuit. It consists of logic gates (nodes) grouped into layers. Every gate in a layer is connected to several nodes from the layer above using connectors (see fig. 1). It is crucial that the functionality of the circuit (circuit-like software) can be simply changed by replacing operations in gates or by redirection of connectors. This property is used for an iterative construction of distinguishers. The construction is controlled by genetic programming that uses a fitness function (success rate) based on the ability of a distinguisher to correctly indicate a given bitstream to be non-random with a high confidence. A well-performing distinguisher is able to assign non-random inputs, the outputs with a significantly different distribution than outputs assigned to truly random inputs. The output distribution difference is formalized using the Kolmogorov-Smirnov test [15].

The supposed usage of EACirc is similar to the statistical batteries. The process is fully automatized with statistical results that are simple to interpret. Additionally, EACirc can be used as a tool for showing cipher weaknesses for manual crypt-analysis performed later. For example, skilled cryptologist can see from fig. 1 what part of TEA is causing statistical deviations and how the weakness can be exploited as shown in the result interpretation (section IV-C).

The whole framework is being continuously extended and enhanced by our team and is accessible online with full documentation [5].

C. EACirc parametrization

EACirc can be configured on multiple levels: firstly, the representation of software circuit used to express candidate distinguishers and, secondly, the parameters of genetic programming. General settings are described in the thesis of Martin Ukrop [14], chapter 4 and project's documentation [5]. The following settings were relevant for TEA analysis.

Functions in nodes: Circuit nodes can contain an identity function, constant-producing function, basic logical binary operators, shifts and rotations, integer comparison

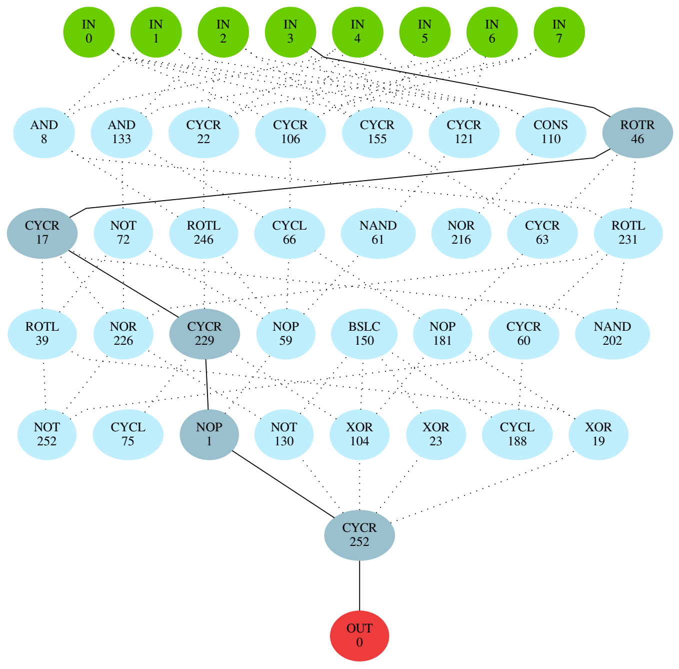

Figure 1. Software circuit with green input nodes, blue inner nodes (operations) and a red output node. Inner nodes and connectors are highlighted if they affect output (dotted edges and lighter nodes are part of evolved circuit, but they are not affecting the output byte in any way). This circuit was evolved in the experiment with 4-round TEA.

functions, masks for bit selection and additional input read operator. The larger diversity of functions means stronger expression capability (within the limited space). However, this vastly increases the space of applicable individuals slowing down the evolution process. Due to this, the set of used function was restricted (integer comparison functions and additional input read operator were not used). All functions are byte-oriented.

Circuit dimensions: In our case, the input layer has the same size (or multiple of) as the TEA block. Other relevant settings include providing more TEA ciphertext blocks as a single input (which would again slow down the evolution considerably). We used 5 internal layers with 8 nodes per layer. The last layer contains a node with 1-byte long output used as the circuit's overall result. See fig. 1 as an example of circuits in our experiments.

Test vectors: Another important setting influencing the success rate of EACirc is the number of test vectors used to evaluate the performance of candidate distinguishers (circuits). In our scenario, the set of test vectors consists of 2 subsets: TEA ciphertexts and data from a quantum random number generator (believed to be completely random), with both subsets of vectors having the same size. More test vectors mean more data for each iteration of evolution to learn from, as well as more precision for the fitness function. On the other hand, more test vectors also need more computation time as every candidate circuit is always evaluated for every separate test vector. In this work, we used two main configurations: for CPU-only version, 1 000 test vectors were used. For nVidia CUDA implementation, 128 000 test vectors were used (see section III-D).

Generations: The number of evolved generations influences the length of searching for the cipher properties. In our case, 30 000 generations were used. The number increased to 300 000 generations provided no observable improvement.

Population size: Number of individuals in a population. We use only one individual for each iteration, which is mutated into two individuals for next generation – an approach similar to hill-climbing heuristics. More individuals may increase the success rate and convergence speed towards a well-performing distinguisher, but may also negatively influence the interpretation of results as different individuals may be correlated. For this reason, the interpretation of results from more individuals is left for future.

D. *Accelerated computation using GPGPU*

The more test vectors are processed, the more computation time to evaluate a circuit is needed. Since the evaluation on a set of test vectors is parallel in nature, data parallelism techniques can be applied. To reduce the runtime and to fully utilize our hardware, the evaluation is optionally computed on GPGPU accelerators using nVidia's CUDA technology. We are running multiple instances of the evaluated circuit on each test vector in the set in parallel.

On used hardware (Intel Core2Duo E8400 and nVidia GeForce GTX 460) the GPU acceleration gives us 229× speedup for circuit evaluation (see fig. 2 for more detailed benchmark results). The execution of EACirc with 1 000 test vectors running only on CPU takes approximately the same time as the GPU-accelerated version with 128 000 test vectors (about 3.5 minutes).

E. *Statistical uniformity testing*

During the process of evolution, distinguishers iteratively adapt to the set of test vectors with a p-value computed in each iteration. We use the fact that for independent samples of truly random data the p-values are uniformly distributed on the interval [0,1]. To leverage this, we intentionally use only p-values from iterations just after the test vectors were regenerated to separate data for training and verification (we regenerate the set every 100$^{\text{th}}$ generation). We can then test their uniformity using the Kolmogorov-Smirnov (KS) test [15] with the assumption that p-values are expected to be uniformly distributed. KS computes its own p-value which is compared with a significance level $\alpha = 5\%$ to draw the conclusion (p-value below the significance level makes us reject the randomness of the assessed data). Since KS gives a probabilistic answer, we repeat the whole process 1 000 times to avoid statistical anomalies. For the random data, it can be expected that about 5% of all runs fail the testing process (since the significance level is set to 5%).

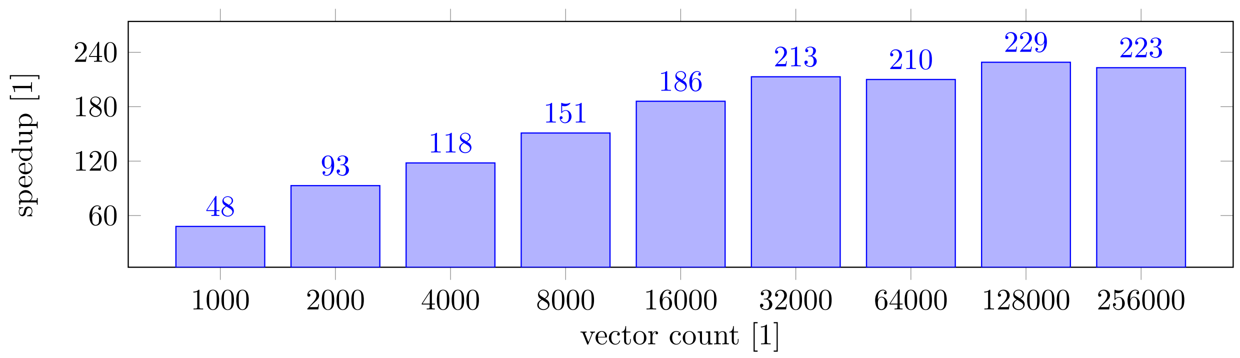

Figure 2. Speedup of GPU-accelerated circuit evaluation against circuit running on CPU, computed as $time_{CPU} \, / \, time_{GPU}$. The performance of a GPU accelerator is generally dependent on work size, in our case the number of test vectors. The benchmark used machines equipped with Intel Core2Duo E8400 at 3 GHz and nVidia GeForce GTX 460 with 336 CUDA Cores on 1550 MHz.

F. Oneclick

As EACirc is randomized in nature, we need to run many tests in parallel. To ease the time-consuming monkey-work of running and post-processing experiments, we use a tool called Oneclick [16], which distributes computations using the BOINC infrastructure [17] on the laboratory computers. This tool reduces both the necessary human work and the running time of the experiment.

G. TEA customization

For complete automation, the tested ciphers are included as plugins into EACirc, which then both generates the test vectors and runs the distinguisher evolution. Since we want to test TEA with a variable number of rounds (not only the recommended 32), we use a slightly changed version of the cipher that is shown below.

Algorithm 1 encrypt(uint32_t *data, const uint32_t *key)

```
const uint32_t delta = 0x9e3779b9;
uint32_t sum = 0;
for int j = 0; j < numRounds; j++ do
    sum += delta;
    data[0] += ((data[1]«4) + key[0]) ˆ (data[1] + sum)
                  ˆ ((data[1]»5) + key[1]);
    data[1] += ((data[0]«4) + key[2]) ˆ (data[0] + sum)
                  ˆ ((data[0]»5) + key[3]);
end for
```

The function input is a plaintext block (64-bits long), stored in the array `data`, and the `key` array of length 128 bits. The output is stored in array `data`. Only changed part of the algorithm is limiting the rounds count to `numRounds`.

H. Design of experiments

There are various settings for generation of output data stream from TEA. The first decision is which cipher mode should be used. We used the electronic codebook (ECB) mode, as this was the case of previous papers on the topic since [9]. This also minimizes the influence of the used mode on the output stream of data (ciphertext) produced by TEA.

An important factor is how the plaintext for TEA is generated. Even a weak cipher will usually provide a strong output if completely random input data are supplied as input. Our framework does not mask the input data with specific bitmask (as was the case in [9]) but instead generates input with some redundancy as described below.

The following ways to generate plaintexts for TEA were implemented:

1) The counter incremented by one for each test vector. This solutions is simple and does not suffer from the problem of repeating plaintexts. It also corresponds to potential usage of the cipher (e.g. if used in the counter mode). On the other side, it has a low Hamming weight (first 40 bits of the plaintext consist only of zeroes). Therefore, this type of plaintext is very difficult to compare with the methodology of previous works.
2) The vectors with 5 randomly placed 0 and other bits set to 1. The number 5 was chosen to create enough unique test vectors (over 10 million). This input also has an extreme (and fixed) Hamming weight, but there are no positions with fixed bits.
3) The vectors with two almost identical parts differing only in a single bit. We used this plaintext type for testing the strict avalanche criterion. The first input block of TEA is fully random and the second is the same with only a single changed bit. In this case, the circuit uses test vectors of 128 bits.

A similar reasoning is relevant for the generation of secret keys used. As we already manipulated input data for the cipher, we used a fixed (but completely random) value as a key for the whole test. For more information about the impact of key reinitialization frequency, please refer to the thesis of Martin Ukrop [14].

Used settings were chosen to simulate TEA usage. Users typically do not change encryption keys during a single session. Input data usually contain some redundancy, as long as meaningful text is processed. Other input types were selected to search for unwanted dependencies inside the cipher.

Table I
Comparison of previous results for reduced rounds TEA.

Rounds	HSIR02 [9]		HI04 [10]		Wei Hu et al. [12]	
	χ^2	(MW)	χ^2	(MW)	χ^2	(MW)
1	8380416	72	522240	153	522.240	153[1]
2	1900	77	736.05	155	602	171
3	(untested)		393.6	116	530.756	117.8
4	(untested)		294.86	50[2]	742.632[1]	67.6[1]
5	(untested)		(untested)		631.74	76

IV. Results

A. Comparison

Results from previous papers can be difficult to compare with ours because the approaches are significantly different (described in section II-A and section III-H). There are no results of statistical batteries from previous works. Therefore, we cannot use them as a common basis for the comparison. Our results can be compared in terms of rounds count, but tested data are different.

All previous works published weights of constructed masks (abbreviated as (MW) in the table), which were used on both the input block and key. This means the mask length is $64 + 128 = 192$ bits, which is a maximum weight for unchanged input. They also presented the average χ^2 statistics of maximal deviation from a random distribution.

Table II, table III and table IV provide a comparison of results from statistical testing batteries NIST STS (version 2.1.1) [1] and Dieharder (version 3.31.1) [3] run in default configuration together with EACirc on the given plaintext type. The result in the cell for Dieharder is the number of passed tests (out of the total 57 tests). From NIST STS, we used all 188 tests. Both batteries used the significance level $\alpha = 1\%$. Results that fail to reject the null hypothesis (are unable to show the non-randomness in data) have gray-colored background. The column for EACirc represents the best results achieved in our experiments. Values from EACirc represent the percentages of runs for which the set of p-values failed the KS test for uniformity with the significance level $\alpha = 5\%$. For the reference random-random distinguishing experiment, the value of 5% is expected (and also measured), so we also mark such results with gray background (data indistinguishable from random). For more detailed explanation of this method, please refer to [18], section 3.2.

We tried different settings of EACirc with the goal of finding the best distinguisher possible. Changes from the default settings (specified in section III-C) are following (EACirc$_{xy}$, where x denotes plaintext type and y stands for EACirc parameters):

- EACirc$_{1a}$ was tested with plaintexts created as a counter incremented by one for each vector (type 1). Besides, this version did not allow shifts and rotations in nodes.

[1]These results are computed as the average of values from tables of the original work [12]. Please note that average value is simplified and for more information refer to the original work.

[2]For this result, a different approach was used. Apart from that, the output mask has very low entropy. For more information, please refer to the original paper [10] (section 2.4).

Table II
Comparison of EACirc and standard statistical batteries with plaintext created as a counter starting from zero (type 1). Gray-colored cells indicate the experiments that failed to reject the randomness of tested data.

Rounds	NIST	Dieharder	EACirc$_{1a}$	EACirc$_{1b}$	EACirc$_{1c}$
	$(x/188)$	$(x/57)$	(%)	(%)	(%)
1	1	0	100	100	100
2	1	1	100	100	100
3	27	3	100	100	100
4	▨ 183	31	▨ 5.0	99.8	100
5	▨ 188	51	▨ 3.0	▨ 5.6	▨ 5.3

Table III
Comparison of EACirc and standard statistical batteries with plaintext with 5 randomly placed zeroes (type 2). Gray-colored cells indicate the experiments that failed to reject the randomness of tested data.

Rounds	NIST	Dieharder	EACirc$_2$
	$(x/188)$	$(x/57)$	(%)
1	24	1	100
2	▨ 183	8	93.3
3	▨ 188	39	▨ 5.6
4	▨ 187	44	▨ 5.6
5	▨ 187	48	▨ 5.5

- EACirc$_{1b}$ used the same settings as EACirc$_{1a}$, except shifts and rotations in nodes were allowed.
- EACirc$_{1c}$ had the same settings as EACirc$_{1b}$ but used the nVidia CUDA implementation, which allows to use 128 000 test vectors as well as increase the evaluator precision.
- EACirc$_2$ was tested with plaintexts of all ones (64b for TEA), with 5 flipped bits to zero on random positions (type 2).
- EACirc$_3$ was tested with twice the input length. The first block is random, and the second is identical to the first but for one bitflip on a random position (plaintext type 3). The total test vector length is 128 bits.

B. Results interpretation

The direct comparison of success with the previous papers is not straightforward due to the different approaches used. In previous approaches, to determine which bits of plaintexts will cause the output of round-reduced TEA to be non-uniform (tested by χ^2 test), the input was changed by applying a bitmask. In this paper, the goal is to find defects in ciphertexts

Table IV
Comparison of EACirc and standard statistical batteries with plaintext suitable for strict avalanche criterion testing (type 3). Gray-colored cells indicate the experiments that failed to reject the randomness of tested data.

Rounds	NIST	Dieharder	EACirc$_3$
	$(x/188)$	$(x/57)$	(%)
1	29	6	100
2	67	7	100
3	▨ 186	24	100
4	▨ 187	39	100
5	▨ 188	▨ 56	▨ 4.5

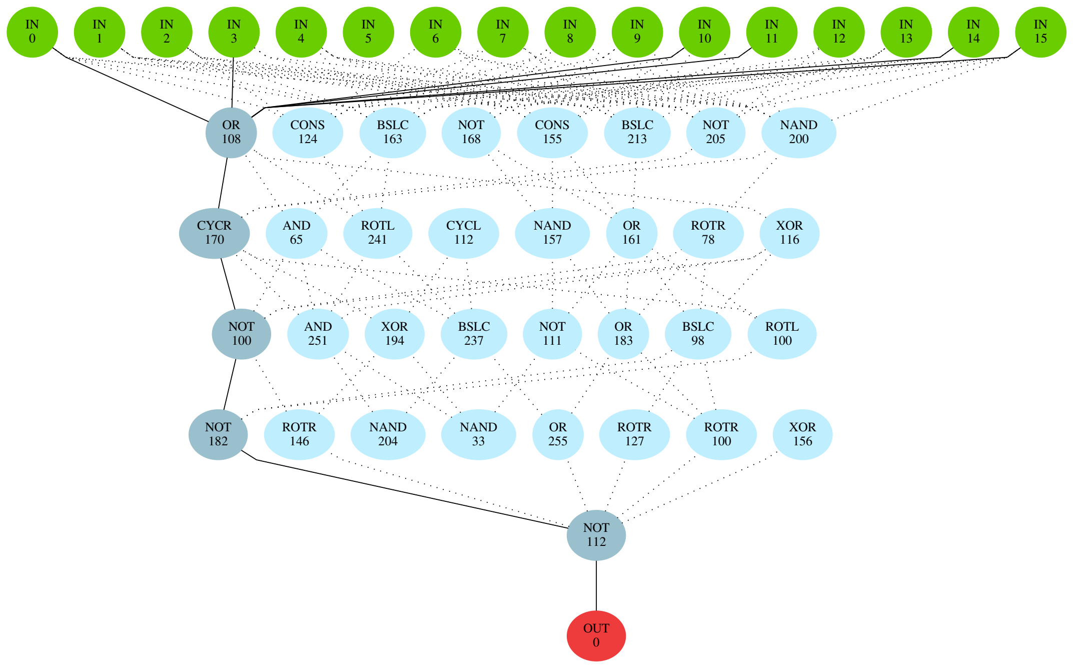

Figure 3. This circuit was evolved in the experiment with 4-round TEA on plaintexts suitable for testing the strict avalanche criterion (type 3). It can be seen that the output byte is mostly dependent on the 4th and the 12th (4th in the second half) byte.

(dependent bits, biased bits, etc.) without directly manipulating plaintexts for the cipher. If we compare only the resulting number of rounds, for which output of the round-reduced TEA can be seen as statistically different from a random bit stream, the best results are provided by [12].

Comparing our approach with the statistical batteries is more straightforward, as we can use the same input data (ciphertext) for EACirc as well for statistical batteries. In all tested combinations (different structures of plaintext), EACirc consistently performs better than NIST STS. Dieharder is able to detect small deviances in one additional round (see table II).

Some information about the cipher can be derived from the results from different plaintext types. For example, statistical batteries perform better than EACirc on plaintexts with just 5 zeroes (type 2). Another observation is based on plaintext types 1 and 3 – EACirc is easily able to detect non-randomness of 4-rounds TEA, but fails to do so for 5 rounds. The same issue may be present for both plaintext types (the 5th round reducing this issue).

The result of each run is single bit fail/pass result (p-value computed by KS uniformity test is smaller/bigger than the significance level). This often leads to the loss of interesting information – what is the quality of the evolved distinguisher and what dependency of output bits was found? Therefore, we decided to perform a deeper analysis of the found distinguishers.

C. *Resulting circuits interpretation*

The outcomes presented in the previous subsection are aggregated results over 1 000 different EACirc runs providing the single p-value for interpretation. Such an approach can provide superior detection of statistical deviances, but will not signalize which concrete bit(s) and dependencies between them cause these deviances – information valuable for the cipher's designer. By analysis of a single well-performing circuit, such an information can be obtained. We analyzed in detail some successful (fitness over 95%) evolved distinguishers. The weaker the distinguisher is, the more noise is present (circuit functionality is not performing as a correct distinguisher for the increasing number of inputs) and the harder it is to reason about the exact bits on which distinguisher's decision is based.

EACirc circuits used in this paper have 4 layers with 8 nodes in each. As a result, the evolved distinguisher can be rather complex and thus difficult to interpret. To provide a better comprehension, reductions of the target circuit can be performed as the distinguisher usually does not use all available nodes and connectors (they do not contribute to the output byte).

First of all, we can prune circuits taking into account the arity of used operations removing the unnecessary connectors. Manual analysis of pruned circuits is considerably simpler.

In the case of 4-round TEA on counter plaintexts (type 1), we analyzed several distinguishers with the fitness over 98%. In all of these circuits (see for example fig. 1) the distinguisher decision is based on the fourth byte of TEA ciphertext. The fourth byte is usually almost unchanged (operations affect only some bits).

We also analyzed 4-round TEA on plaintexts suitable for strict avalanche criterion testing (type 3). In this case, the input layer had 16 input nodes, capable of processing two blocks of TEA ciphertext at once. Analyzed distinguishers (see for example circuit in fig. 3) commonly combine the fourth byte of the first ciphertext block with the fourth byte of the second ciphertext block.

$$\Sigma = 1\,000\ \frac{\text{runs}}{\text{experiment}} \cdot \left(\frac{30\,000\,\frac{\text{generations}}{\text{run}}}{100\,\frac{\text{generations}}{\text{test set}}} \cdot \frac{1}{2} \cdot 1\,000\ \frac{\text{vectors}}{\text{test set}} \cdot 8\ \frac{\text{bytes}}{\text{vector}} \right) \approx 1,2\,\text{GiB per experiment}$$

Figure 4. The amount of data analyzed by EACirc for a single configuration of randomness testing experiment.

D. Performance

The runtime of EACirc with basic settings (1 000 test vectors and 30 000 generations) is around 3.5 minutes on a single core 3 GHz Intel Core2Duo processor. It includes the creation of test vectors and is not measurably affected by the number of TEA rounds executed. Due to the randomized nature of the framework, we replicate every experiment 1 000 times. This gives us a combined computation time of approximately 3 500 minutes on single CPU core.

Since the individual runs are independent, execution can be parallelized and distributed over multiple computers. We used 12 laboratory computers with the 3 GHz Intel Core2Duo processors mentioned above, which resulted in the execution time of about 5 hours for every single tested scenario. Thus, testing TEA limited to 1-8 rounds can be executed within 2 days of computation. For larger sets of tests, we used the national grid infrastructure provided by MetaCentrum [19].

Tests with 128 000 test vectors were executed on GPUs using nVidia CUDA. The running time for each test was around 3.5 minutes. As more GPU cores are available for parallelization of circuit evaluation, a higher amount of test vectors could be evaluated. The runtime was still linearly dependent on the generation count – tests with 300 thousand generations and 128 000 test vectors had a running time of around 105 minutes.

EACirc needs truly random data as a reference stream for a distinguisher evolution phase. We used a pool of 1920 MiB of data pre-loaded from the High Bit Rate Quantum Random Number Generator Service [20].

Regarding the TEA ciphertext, we generated 500 test vectors (half a set) of 64 bits each in 300 test vector sets in 1 000 runs for statistical interpretation. This amounts to 1.2 GiB of ciphertext data for the whole experiment or 1.2 MiB for a single run. See fig. 4 to understand, how we figured out the data usage of EACirc.

V. Discussion and Future work

The EACirc framework is continually developed and extended with different inner approaches and settings with the goal of improving the distinguisher success rates. At the moment, we work on two alternative circuit representations. One with the possibility of executing more complex Java bytecode sequences in circuit nodes. The sequences would be extracted directly from the Java implementation of the tested ciphers. Another alternative would be based on polynomials, which should provide better possibilities not only for creating distinguishers but also for analyzing the importance of isolated parts of tested function's output the distinguisher is based on.

Different heuristics like simulated annealing can be used for the mutation of a single individual, which may provide a better success rate or faster convergence than the currently used hill-climbing technique with a stable mutation probability. We are also working on the interpretation of multi-individual settings to be able to use the full potential of genetic algorithms for TEA analysis.

Acknowledgment

We acknowledge the support of Czech Science Foundation, project GA16-08565S. The access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

References

[1] A. Rukhin *et al.*, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", *NIST Special Publication 800-22rev1a*, 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf (visited on 2015-12-29).

[2] G. Marsaglia. (1995). Diehard battery of tests of randomness, Floridan State University, [Online]. Available: http://stat.fsu.edu/pub/diehard/ (visited on 2015-12-29).

[3] R. G. Brown. (2004). Dieharder, A Random Number Test Suite. version Version 3.31.1, Duke University Physics Department, [Online]. Available: http://www.phy.duke.edu/~rgb/General/dieharder.php (visited on 2015-12-29).

[4] P. L'Ecuyer and R. Simard, "Testu01: a c library for empirical testing of random number generators", *ACM Trans. Math. Softw.*, vol. 33, no. 4, Aug. 2007, ISSN: 0098-3500. DOI: 10.1145/1268776.1268777. [Online]. Available: http://doi.acm.org/10.1145/1268776.1268777 (visited on 2015-12-29).

[5] P. Švenda, M. Ukrop, M. Sýs, *et al.* (2012). Eacirc, Framework for autmatic search for problem solving circuit via evolutionary algorithms, Centre for Research on Cryptography and Security, Masaryk University, [Online]. Available: https://github.com/crocs-muni/EACirc (visited on 2015-12-29).

[6] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, "Genetic Programming: An Introduction, On the Automatic Evolution of Computer Programs and Its Applications", 1997.

[7] D. J. Wheeler and R. M. Needham, "TEA, a tiny encryption algorithm", in *Fast Software Encryption*, Springer, 1995, pp. 363–366.

[8] J. Kelsey, B. Schneier, and D. Wagner, “Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA”, ICICS ’97, pp. 233–246, 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=646277.687180 (visited on 2015-12-29).

[9] J. C. Hernández, J. M. Sierra, P. Isasi, and A. Ribagorda, “Genetic Cryptoanalysis of Two Rounds TEA”, in *Computational Science—ICCS 2002*, Springer, 2002, pp. 1024–1031.

[10] J. C. Hernández and P. Isasi, “Finding Efficient Distinguishers for Cryptographic Mappings, with an Application to the Block Cipher TEA”, *Computational Intelligence*, vol. 20, no. 3, pp. 517–525, 2004.

[11] A. Garrett, J. Hamilton, and G. Dozier, “A Comparison of Genetic Algorithm Techniques for the Cryptanalysis of TEA”, *International journal of intelligent control and systems*, vol. 12, no. 4, pp. 325–330, 2007.

[12] W. Hu *et al.*, “Cryptanalysis of TEA Using Quantum-Inspired Genetic Algorithms”, *Journal of Software Engineering and Applications*, vol. 3, no. 01, p. 50, 2010.

[13] E. Y.-T. Ma and C. Obimbo, “An evolutionary computation attack on one-round tea”, *Procedia Computer Science*, vol. 6, pp. 171–176, 2011.

[14] M. Ukrop, “Usage of evolvable circuit for statistical testing of randomness”, bachelor thesis, Faculty of Informatics Masaryk University, 2013. [Online]. Available: https://is.muni.cz/th/374297/fi_b/ (visited on 2015-12-29).

[15] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*, 3rd ed. CRC Press, 2003, ISBN: 9781420036268.

[16] Ľ. Obrátil, “Automated task management for BOINC infrastructure and EACirc project”, bachelor thesis, Faculty of Informatics Masaryk University, 2015. [Online]. Available: https://is.muni.cz/th/410282/fi_b/ (visited on 2015-12-29).

[17] D. P. Anderson *et al.* (2015). BOINC project, [Online]. Available: https://boinc.berkeley.edu/ (visited on 2015-12-29).

[18] M. Sýs, P. Švenda, M. Ukrop, and V. Matyáš, “Constructing empirical tests of randomness”, 2014. [Online]. Available: http://dx.doi.org/10.5220/0005023902290237 (visited on 2015-12-29).

[19] Team Czech NGI. (2015). Metacentrum – Virtual Organization of the Czech National Grid Organization, [Online]. Available: https://metavo.metacentrum.cz/ (visited on 2015-12-29).

[20] Nano-Optics groups (Department of Physics) and PicoQuant GmbH. (2010). High bit rate quantum random number generator service, Humboldt University of Berlin, [Online]. Available: http://qrng.physik.hu-berlin.de/ (visited on 2015-12-29).

[21] P. Švenda, M. Ukrop, and V. Matyáš, “Towards cryptographic function distinguishers with evolutionary circuits”, in *SECRYPT*, Centre for Research on Cryptography and Security, Masaryk University, 2013, pp. 135–146. [Online]. Available: http://dx.doi.org/10.5220/0004524001350146 (visited on 2015-12-29).