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

Karel Kubíček

karel.kubicek@mail.muni.cz

Masaryk University, Brno, Czech Republic

Abstract

Analysing cryptographic functions usually requires extensive work of a skilled cryptanalyst. However, some automation is possible, e.g., by using randomness testing batteries such as NIST STS or Dieharder. Yet such tests are limited to predefined test patterns. However, there is a new approach – EACirc is a novel randomness testing framework based on finding a distinguisher for a given cipher output. In this work, we use EACirc to analyse the outputs of Tiny Encryption Algorithm (TEA). TEA was previously used with genetic algorithms for evolution of bit masks used for restriction of cipher input. In this paper, we compare the methodology and results of previous works with EACirc. Instead of evolving bit masks, we create a software circuit applicable as a distinguisher for limited-round TEA (up to 4 rounds). Results of EACirc are also compared to the standard statistical batteries.

Keywords: randomness statistical testing, TEA, genetic algorithms, randomness distinguisher, software circuit

1 Introduction

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

The common way to automate testing of randomness is using statistical batteries. NIST STS [Ruk10] is a standard battery of tests, that are commonly used for this purpose, together with other batteries such as Diehard [Mar95], Dieharder [BEB09] or TestU01 [LS07]. The batteries contain sets of fixed tests (usually parametrized to form multiple different subtests) checking expected statistics of tested output stream (TEA ciphertext in our case) in comparison to truly random data.

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 deviances from truly random data. But as cryptographic functions have a deterministic output (dependent only on input data and a key), it is a priori expected that they cannot pass all possible tests of randomness and there are tests of randomness that reveal the sequence as non-random. However, such a test can be very difficult to find.

In this work we use EACirc [ŠU+15], 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 truly random data stream. In the framework, randomness tests are computed iteratively, adapting to the processed sequence. The construction is stochastic and uses genetic programming. Tests are constructed from a predefined pool of operations (building blocks). Set of operations, together with a limit for the number of operations, allows us to control the complexity of the tests. The framework theoretically allows us to construct 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 should (hypothetically) provide a better detection ability than standard tests.

We performed several experiments on Tiny Encryption Algorithm. This cipher has been used by other teams, that used genetic algorithms for testing randomness. Capabilities of our framework are tested on this cipher and compared with both statistical batteries and other papers.

This paper is organized as follows: Section 2 introduces TEA as a simple encryption algorithm used nowadays as a benchmark for randomness tests. Section 3 contains information about our framework EACirc with definition of used settings. Also TEA customization and input data structure is discussed in this section. Results and their interpretation are presented in section 4 with performance and data usage of EACirc. In Section 5 we describe future work.

2 Tiny Encryption Algorithm

Tiny Encryption Algorithm (TEA) is a block cipher designed by David Wheeler and Roger Needham of the Cambridge Computer Laboratory [WN95]. The algorithm was designed to have a simple structure based on the Feistel network with 32 rounds (we count 2 steps of Feistel network as 1 TEA round). The cipher uses 64-bit long blocks and 128-bit long keys.

2.1 TEA distinguishers – state of the art

Nowadays, the cipher is not considered to be secure for regular use as it suffers from multiple weaknesses, most significantly the related-key attack [KSW97]. However, it was used as a benchmark for randomness testing using genetic algorithms starting with a paper in 2002 by Julio C. Hernández, José M. Sierra, Pedro Isasi and Arturo Ribagorda [HSIR02], who were successful with TEA limited to 1 and 2 rounds. 2 years later, a similar team published new results with improved settings [HI04], which resulted in 3 to 4 rounds. The newest results from 2010 by Wei Hu [Hu+10], using quantum inspired genetic algorithm, succeeded with 4 to 5 rounds TEA. There were more papers on this topic by other teams too, but they came later with distinguisher efficient for less rounds.

These works were using a different approach to the distinguisher evolution than us. In [HSIR02], a genetic algorithms with χ^2 statistic for customized fitness functions was used. The paper focuses on bit patterns of the least significant 10 bits of ciphertext from reduced-round TEA. The plaintext and keys were generated using a bit-mask, which was iteratively evolved. Following works then followed this methodology with only difference of evolution settings, not the approach itself.

3 Our approach

3.1 Randomness testing with genetic programming

As stated in the Introduction, the common way of automate testing of randomness is using statistical batteries. The approach of genetic algorithms is different then running predefined sets of tests from a statistical battery. Firstly, a set of individuals is created with each individual representing a candidate distinguisher function. Secondly, every individual decides if the provided block of input data is random or non-random. Thirdly, better individuals are randomly mutated or cross-bred to create better descendants (the ratio of correct guesses constitutes a fitness function). The process follows the principles of biological evolution. Therefore, if ciphertexts have a common property expressible as a distinguisher function, then an individual representing this function can be potentially evolved and improved in the process of further evolution.

Use of genetic algorithms induce a couple of disadvantages. The evolution can be computationally very expensive, since finding sufficient changes of individual requires up to search over all possible changes to individual, which is done randomly. This requires enough generations to be executed with no guarantee that such individual will be found. Also the process of fine-tune of the parameters can be difficult to achieve good working evolution. For more details of possible problems and their solution in EACirc, follow to thesis of Martin Ukrop [Ukr13], section 3.1.

3.2 EACirc framework

In case of EACirc, the individual is a hardware-like circuit. It consists of gates and interconnecting wires, transforming input data into desired output data. Usually, the input is as long as the block of ciphertext (64 bits for TEA), and the output is 1 byte. The fitness function is a distance statistics over output bytes produced for the assessed stream and truly random data. Further information can be found in [SŠUM+14].

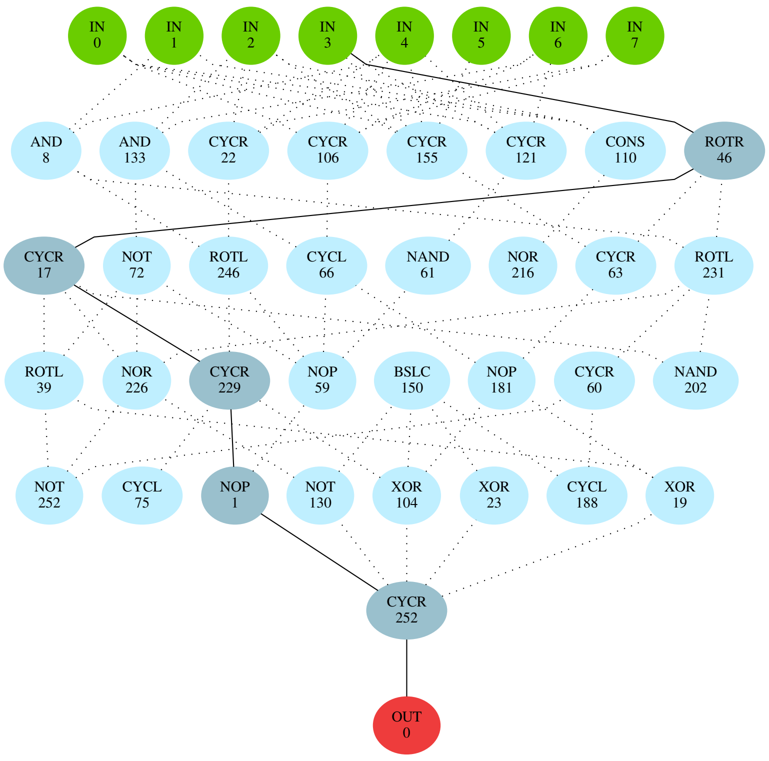

Figure 1: Software circuit with green input nodes, blue inner nodes and red output node. Inner nodes and connections are highlighted if they affect output. The circuit is result of experiment with 4 rounds TEA.

EACirc supposed usage is similar as statistical batteries. The process have to be fully automatized and the results are simple to interpret. But a top of automatized way of testing, EACirc can be used as tool for showing weaknesses of cipher for manual cryptoanalysis. For example, from Figure 1, skilled cryptologist can discover, what part of TEA is weak and maybe also how the weakness can be exploited.

The whole framework is being continuously extended and enhanced by team from Faculty of Informatics of Masaryk University. Current target is to make it fully automatized and simple to use for extending statistical batteries as a randomness testing tool. Another target is increasing ability of the distinguisher. The whole project is accessible on github page with full documentation [ŠU+15].

3.3 EACirc parametrization

EACirc framework has very complex configuration, because both evolution and the circuit can be set up in various ways. General settings are described in thesis of Martin Ukrop [Ukr13], chapter 4 and github documentation [ŠU+15]. Following settings are described, because they were changed 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 functions, masks for bit selection, input read connector and external C function call. 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 function, input read and the external function call were not used).

Circuit dimensions In our case, the input layer has the same size as the TEA working block (8 nodes with 8 bits each). Other applicable 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 Figure 1 for used circuit dimension of 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, 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. On the one hand, 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 one candidate circuit is 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, 32 000 test vectors were used. Using these settings, run time for CPU (Intel Core2Duo E8400 at 3 GHz) was taking about the same time as for CUDA (nVidia GeForce GTX 460 with 336 CUDA Cores on 1550 MHz).

Generations The number of evolved generations influences the length of searching for the cipher properties. In our case, 30 000 generations were used.

Genome size The last setting discussed in this work is count of individuals in the generation. The current approach to fitness evaluation is unable to interpret more individuals, because they can be in correlation, as it uses the Kolmogorov-Smirnov test [She03] for an overall statistic. For this reason, we currently use only one individual for each iteration, which is mutated into 2 individuals for evolution. This approach is more similar to hill climbing heuristics than to evolution. Interpretation of more individuals is a planned future work.

Categories-based evaluator Results of distinguisher consist of series of P-values computed during the whole evolution. We store only P-values of generations after regeneration of test vector (reason for this is independence of P-values). For testing uniformity of P-values, we use the Kolmogorov-Smirnov (KS) test. KS computes its own P-value that can be compared to the significance level (chosen as $\alpha = 5\%$) to evaluate the KS test. Since P-values computed by the KS test could be smaller than α even for uniformly distributed P-values, we repeat the whole process 1000 times. For a proper and clear statistical interpretation we test whether 5% of P-values computed by the KS test are smaller than the chosen significance level $\alpha = 5\%$.

Oneclick As we use statistics to interpret the outputs of EACirc, 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 [vOb15], which distributes computations using BOINC infrastructure **[citation]** on the laboratory computers. This tool reduces both the necessary human work and also running time of the experiment.

3.4 TEA customization

For complete automation, the tested ciphers are implemented directly into EACirc, which then both generates the test vectors and runs the genetic programming of distinguisher. 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.

```
static const u32 delta = 0x9e3779b9; //u32 typedefed for 32-bits long unsigned int
void encrypt(u32 *data, const u32 *key) {
    u32 sum = 0;
    for (int j = 0; j < numRounds; j++) { //limited rounds by numRounds
        sum += delta;
        data[0] += ((data[1]<<4) + key[0])
           ^ (data[1] + sum)
           ^ ((data[1]>>5) + key[1]);

        data[1] += ((data[0]<<4) + key[2])
            ^ (data[0] + sum)
            ^ ((data[0]>>5) + key[3]);
    }
}
```

Figure 2: Customized TEA code for EACirc interface with limited number of rounds

Input of the function is one block (64-bits long), stored in data array `data` with length 2 and the `key` with length 128-bits (array of unsigned int of length 4). Output is stored in array `data`. Only changed part of the algorithm is limited rounds count to tested `numRounds`.

3.5 Testing input data

There are various possible settings for generation of output data stream from TEA. The first decision is therefore which cipher mode should be used. We used the electronic code book (ECB) mode, as this was the case for previous papers on the topic since the first paper [HSIR02]. 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 input plaintext data for TEA are generated. Our framework does not change the input data (as was the case in [HSIR02]). Note that even a weak cipher will provide strong output if completely random input data are supplied as input.

Initially, we chose a biased input stream with a bias towards value 1 of bits of 95 %, (each individual bit was much more likely to be 1 than 0). This decision provided incorrect results as the probability of generating a plaintext full of ones was $0.95^{64} = 0.038$. Resulting in around 19 identical test vectors from the set of 500 (the remaining 500 vectors were taken from a truly random data source). The distinguisher quickly learned to find these vectors created from the plaintexts of ones even for TEA without artificial round limitations. The same data stream tested by Dieharder also failed with only about 2 out of 20 tests passing.

To mitigate the described problem, we implemented three different ways to generate plaintext data for TEA:

1. The counter incremented by one for each test vector,
2. the vectors with 5 randomly placed 0 and other bits set to 1,
3. the vectors with two almost identical parts differing only in a single bit (useful for the test vectors twice the length of the TEA block).

These methods do not suffer repeating plaintextes problem as biased input.

A similar reasoning is relevant for generation of the secret key used by cipher. As we already manipulate input data for the cipher, we used a fixed but completely random key for the whole test. For more information about impact of key reinitialization frequency, please refer to thesis by Martin Ukrop [Ukr13].

Used settings was chosen to simulate TEA usage. User does not change encryption key for used session (communication, disc encryption, etc.). Also input data are not fully random, as long as meaningful text does not seem random. Other input types were developed for searching for inside dependency of the cipher.

4 Results

4.1 Comparison

Results from previous papers can be difficult to compare with ours, because the approaches are significantly different (described in subsection 2.1 and subsection 3.5). There are no results of statistical batteries from previous works. Therefore, we cannot use them as common basis for the comparison. Our results can be compared in terms of rounds count, but tested data are completely different.

Although more papers used genetic algorithms on TEA randomness analysis, we compare only these, that came with best results in their time.

All of previous works published weights of constructed mask, which were used on pair of both input block and key. This means the mask length is $64 + 128 = 192$ bits, which is a maximum weight for unchanged input. They published the mask weights together with average χ^2 statistics of maximal deviation from random distribution.

Rounds	J.C. Hernández et al. [HSIR02]		J.C. Hernández et al. [HI04]		Wei Hu [Hu+10]	
	χ^2	Mask weight	χ^2	Mask weight	χ^2	Mask weight
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

Table 1: Comparison of previous results for reduced rounds TEA.

The next table provides results from statistical test batteries NIST STS (version 2.1.1) [Ruk10] and Dieharder (version 3.31.1) [BEB09]. Dieharder provides three levels of evaluation (pass, weak, fail), we assigned to these levels values 1, 0.5 and 0 respectively. The result in the cell is sum of the 20 tests. From NIST STS we were using 162 tests (with pass or fail level). Both batteries were tested with significance level $\alpha = 1\,\%$. Result with 0 passed tests is in parenthesis as mark of indistinguishable from random result. The column for EACirc represents the best achieved results from our measurements. Values from EACirc represents percentage of runs for which the set of P-values failed the KS test for uniformity with significance level $\alpha = 5\%$. For the reference random-random distinguishing experiment, the value of 5% is expected (and also measured), so we mark this value with parenthesis as indistinguishable from random. For more detailed explanation of this method with proof of correctness interpretation, please refer to [SŠUM+14], section 3.2.

Rounds	NIST	Dieharder	EACirc$_5$
1	162	20	100
2	162	18.5	100
3	162	16.5	100
4	6	11.0	100
5	(0)	(0)	(5)

Table 2: Results of statistical batteries in comparison with best result of EACirc.

We tried different settings of EACirc with the goal of finding the best distinguisher possible. Changes from default settings (specified in subsection 3.3) are following:

1. EACirc$_1$ column was tested with input type 2. This means, that plaintexts are all ones (64b for TEA), with 5 flipped bits to zero on random positions.
2. EACirc$_2$ column was tested with input type 3. The first half of plaintext is random, second is identical to the first but for one bitflip on random position.
3. EACirc$_3$ column was tested with input type 1. Plaintexts for test vectors were created by counter incremented by one for each vector. Also this experiment does not use shifts and rotations in nodes.
4. EACirc$_4$ uses same settings as EACirc$_3$, except shift and rotations in nodes were allowed.
5. EACirc$_5$ has same settings as EACirc$_4$, but uses nVidia CUDA implementation, which allows to use 32 000 test vectors and also to increase of evaluator precision.

Rounds	EACirc$_1$	EACirc$_2$	EACirc$_3$	EACirc$_4$	EACirc$_5$
1	100	(5.6)	100	100	100
2	93.3	(4.8)	100	100	100
3	(5.6)	(5.4)	100	100	100
4	(5.6)	(5.4)	(5.0)	99.8	100
5	(5.5)	(5.2)	(3.0)	(5.6)	(5.3)

Table 3: EACirc results with different settings. The results were obtained from 1000 runs for every round-limited TEA with basic settings from subsection 3.5.

[1]These results are computed as average of values from tables of the original work [Hu+10]. Please, note that average value is simplified and for more information, refer to the original work.

[2]For this result, different approach was used. Also the output mask has very low entropy. For more information, please refer to the original paper [HI04] (section 2.4).

4.2 Results interpretation

Results on TEA of EACirc are on similar level as results from statistical batteries. EACirc is unable to find distinguisher for more rounds then NIST STS or Dieharder. EACirc give a little more information, since we can analyze the successful circuits. As showed on example in Figure 1, we can find which bits are correlated.

Comparison with previous works is more difficult. If we input random data (or only little changed data), EACirc cannot evolve sufficient distinguisher. As we do not use evolution to change input, but we evolve candidate circuit, which works with ciphertext, we have to input nonrandom data. And the level of nonrandom differs, for how much rounds we can evolve distinguisher. This may be compared with mask weight in works of other teams. Compared to this, we can use only vectors with very little weight (but unique) for distinguisher for 4 rounds TEA.

Last, but not least, additional information can be obtained from used settings and corresponding results. If we allow rotations and bit shifts as primitive operation in nodes, we can evolve distinguisher for four rounds. Without these operations, distinguisher for only three rounds was found. The final distinguisher from Figure 1 contains mentioned function repeatedly. As TEA also uses shifts in its code, value shifting seems to be important operation for TEA distinguisher.

From more runs, we can analyse other circuits. For 4 rounds TEA, we found interesting dependency on 4. byte of the input (please, see circuits from Figure 1 and appendix). Our hypothesis from this measurements is, that fourth byte of 4 rounds TEA can be distinguished from random (with our used key). To confirm this hypothesis, we should use evolved distinguisher on more data and analyze the given output, which may be part of future work.

4.3 Performance

The runtime of EACirc with basic settings (1 000 test vectors and 30 000 generations) is around 1 minute on single core of 3 GHz Intel Core2Duo processor. It includes generation of the test vectors and is not measurably affected by the number of TEA rounds executed. For the interpretation of the statistics, we execute every test 1 000 times, which gives us a combined computation time of approximately 1 000 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 above-mentioned 3 GHz Intel Core2Duo processors, which results in the execution time of about 45 minutes for every single tested scenario. Thus, testing TEA limited to 1-8 rounds can be executed within a day of computation. For larger sets of tests, we used the national grid infrastructure provided by Metacentrum [CN15].

Tests with 32 000 test vectors were executed on GPUs using nVidia CUDA. The running time for each test is around 1 minute. Due to parallelization of candidate circuit evaluation, a higher amount of test vectors are evaluated as more GPU cores are available. The runtime is still linearly dependent on the generation count – tests with 300 thousands generations and 128 000 test vectors have a running time of around 30 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 Quantum Random Bit Generator Service [Ste07].

Regarding the TEA ciphertext, we generated 1 000 test vectors 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 just single run.

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

5 Discussion and Future work

The EACirc framework is continually developed and extended with different inner settings with the goal of improving distinguisher success rate. At the moment, we work on 2 alternative circuit representations. One with the possibility of executing more complex Java byte-code sequences in circuit nodes. The sequences would be extracted directly from the Java implementation of the tested ciphers. Another alternative representation is 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 better success rate or faster convergence than the currently used hill climbing technique with a stable mutation probability.

We will also work on interpretation of multi-individuals settings, so we can use full potential of genetic algorithms for TEA analysis.

We can also work more with input data for the experiments. One of interesting ideas is using masks evolved in [Hu+10] to modify the input. Using evolved mask together with circuit, which is able to modify the data, can be very strong combination. Or even EACirc can be altered to change not only the output data, but even to work with the plaintext used for the test vectors.

It seems, that current implementation of EACirc with input generation found its limits with even by order more test vectors and generations. But changes suggested in previous paragraphs can lead to creating successful distinguisher for more-rounds TEA, which we was unable to found with more computations. Our future aim will be adding these new methods.

Acknowledgements: 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

[BEB09] R. G. Brown, D. Eddelbuettel, and D. Bauer, “Dieharder: a random number test suite”, *Duke University Physics Department*, 2009.

[CN15] Team Czech NGI. (2015). Metacentrum – Virtual Organization of the Czech National Grid Organization, [Online]. Available: https://metavo.metacentrum.cz/ (visited on 2015-09-26).

[HI04] 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.

[HSIR02] 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.

[Hu+10] 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.

[KSW97] J. Kelsey, B. Schneier, and D. Wagner, “Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X NewDES, RC2, and TEA”, 1997.

[LS07] 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.

[Mar95] G. Marsaglia, *The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness*, http://www.stat.fsu.edu/pub/diehard/, 1995.

[Ruk10] A. Rukhin, “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*, 2010.

[She03] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures.* crc Press, 2003.

[Ste07] R. Stevanović, *Quantum random bit generator service*, http://random.irb.hr/, 2007.

[SŠUM+14] M. Sýs, P. Švenda, M. Ukrop, V. Matyáš, *et al.*, “Constructing empirical tests of randomness”, *SCITEPRESS–Science and Technology Publications*, 2014.

[Ukr13] M. Ukrop, “Usage of evolvable circuit for statistical testing of randomness”, *Bachelor thesis, Masaryk university*, 2013. [Online]. Available: http://is.muni.cz/th/374297/fi_b/thesis.pdf.

[vOb15] Ľubomír Obrátil, “Automated task management for BOINC infrastructure and EACirc project”, 2015. [Online]. Available: https://is.muni.cz/auth/th/410282/fi_b/thesis.pdf.

[vUM13] P. Švenda, M. Ukrop, and V. Matyáš, “Towards Cryptographic Function Distinguishers with Evolutionary Circuits.”, in *SECRYPT*, SciTePress, 2013, pp. 135–146, ISBN: 978-989-8565-73-0.

[WN95] D. J. Wheeler and R. M. Needham, “TEA, a tiny encryption algorithm”, in *Fast Software Encryption*, Springer, 1995, pp. 363–366.

[ŠU+15] P. Švenda, M. Ukrop, *et al.* (2015). EACirc project, [Online]. Available: https://github.com/petrs/EACirc.

A Analysis of output circuits

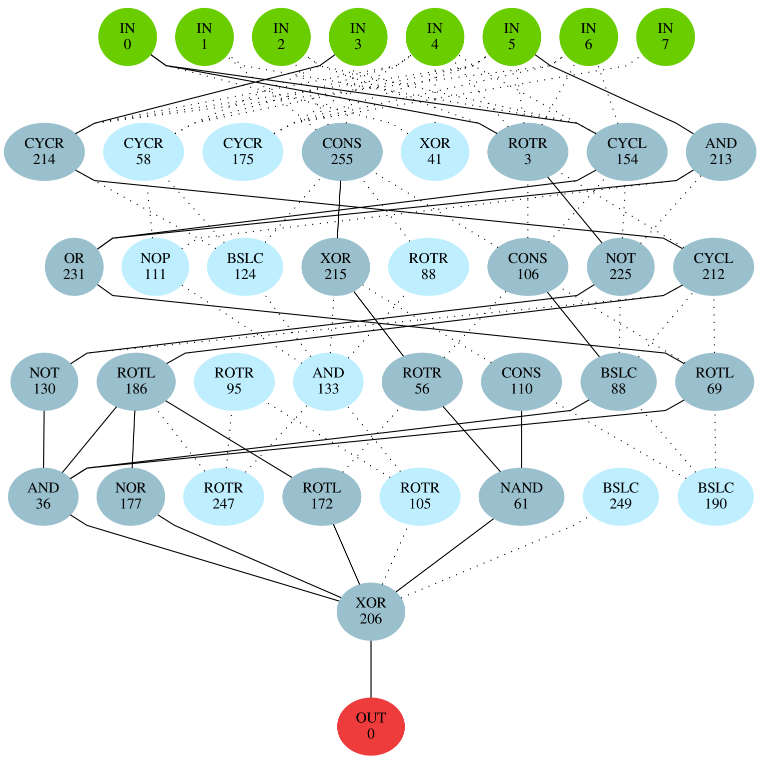

Figure 3: Circuit from 4 rounds TEA analysis.

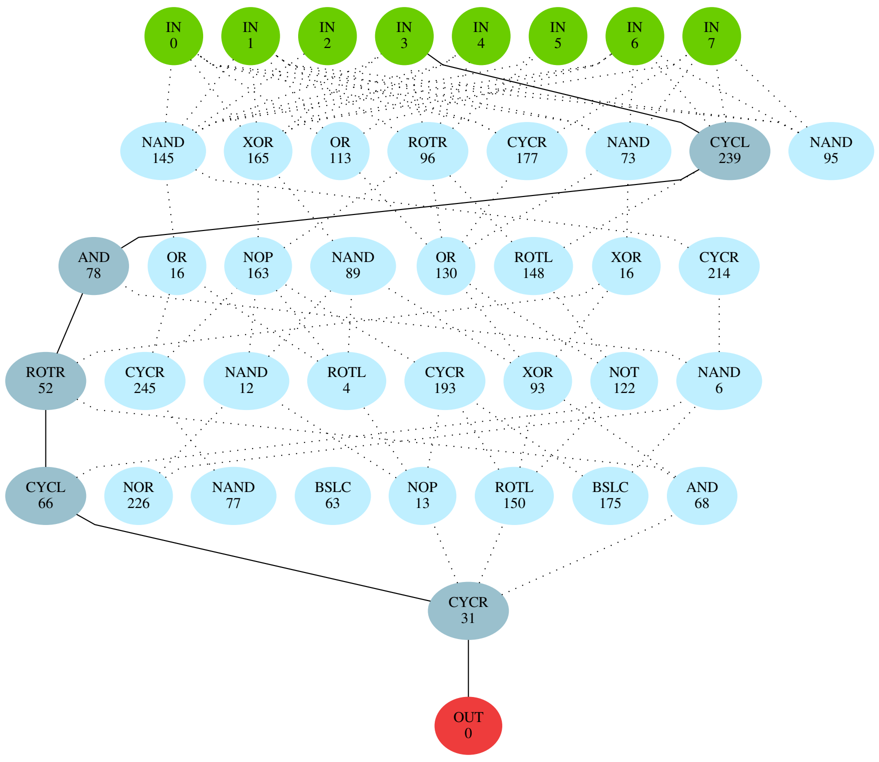

Figure 4: Circuit from 4 rounds TEA analysis.

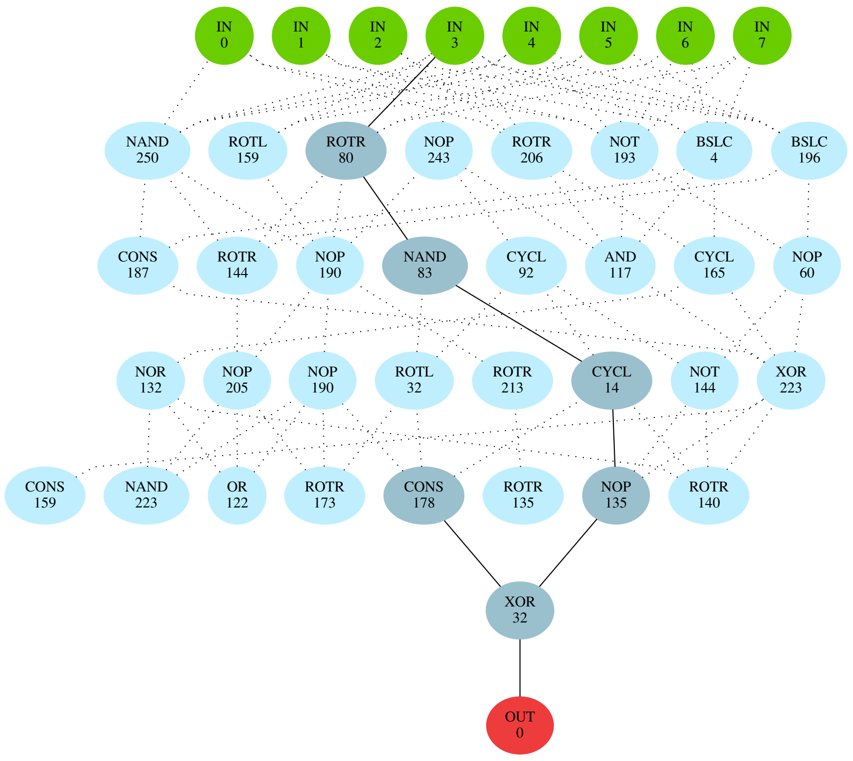

Figure 5: Circuit from 4 rounds TEA analysis.