Randomness analysis in authenticated encryption systems

Master thesis

Martin Ukrop

Brno, autumn 2015
Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Martin Ukrop

Advisor: RNDr. Petr Švenda, Ph.D.
Acknowledgement

Many thanks to you all.

There would be much less algebra, board gaming, curiosity, drama, experience, functional programming, geekiness, honesty, inspiration, joy, knowledge, learning, magic, nighttime walks, OpenLabs, puzzle hunts, quiet, respect, surprises, trust, unpredictability, vigilance, Wachumba, xylophone, yummies and zeal in the world for me without you.

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.
Abstract

This thesis explores the randomness of outputs created by authenticated encryption schemes submitted to the CAESAR competition. Tested scenarios included three different modes of public message numbers. For the assessment, four different software tools were used: three common statistical batteries (NIST STS, Dieharder, TestU01) and a novel genetically inspired framework (EACirc). The obtained results are interpreted in two ways: Firstly, to gain insights into the quality of the proposed CAESAR candidates. Secondly, to compare and contrast the used randomness testing tools. Directions for future research are proposed based on the obtained conclusions.
Keywords

statistical randomness, authenticated encryption, CAESAR, evolutionary algorithms, genetic programming, EACirc, NIST STS, Dieharder, TestU01
Contents

1 Introduction .................................................. 2
2 Previous works .................................................. 3
   2.1 Cryptoprimitives assessment ............................. 3
   2.2 Genetic algorithms in cryptography ..................... 4
   2.3 EACirc framework ....................................... 4
3 Authenticated encryption ...................................... 6
   3.1 CAESAR competition ..................................... 6
      3.1.1 Candidates requirements ............................ 7
      3.1.2 Submissions ......................................... 8
   3.2 Tested ciphers .......................................... 8
4 Experiment methodology ...................................... 11
   4.1 Statistical batteries .................................... 11
      4.1.1 NIST STS ........................................... 13
      4.1.2 Dieharder ........................................... 14
      4.1.3 TestU01 ............................................. 15
   4.2 EACirc .................................................. 15
      4.2.1 Workflow ........................................... 16
      4.2.2 Implementation and settings ....................... 17
      4.2.3 Results interpretation ............................. 21
   4.3 Reference experiments .................................. 22
5 Experiment results ........................................... 24
   5.1 Experiment settings .................................... 24
   5.2 Interpretation of results ................................ 26
   5.3 Conclusions for CAESAR candidates .................... 27
   5.4 Conclusions for randomness testing tools .............. 28
6 Summary ....................................................... 37
   6.1 High-level conclusions .................................. 37
   6.2 Proposed future work .................................... 38
A Data attachment ............................................. 44
1 Introduction

Nowadays, cryptography interferes with almost every aspect of our lives. Ongoing research of cryptoprimitives is, therefore, essential to ensure they provide the assurances required. The assessment can be done in a multitude of ways – this thesis concentrates on randomness testing. Even though finding patterns in the produced outputs does not prove the design insecure, it significantly hints at its potential weaknesses.

Historically, randomness was assessed primarily by statistical tests. Over time, multiple tests were grouped into suites, such as Statistical Test Suite by the National Institute of Standards and Technology (NIST STS) [Nat97b]. However, creating the tests required an enormous amount of human analytical work. Furthermore, the tests were limited to pre-defined data characteristics manually inspected by the analysts. Recently, other approaches supplementing statistical batteries started to emerge [Š+12; Kam13]. Although they offer potential to use novel and/or unusual data characteristics, it is wise to complement them with results obtained by standard means.

In this thesis, we chose to scrutinize submissions of the ongoing CAESAR competition [CAE13] (Competition for Authenticated Encryption: Security, Applicability, and Robustness). The authentication tags produced by all candidates are examined using four different software tools: a novel genetically-inspired framework (EACirc [Š+12]) and three standard statistical batteries (NIST STS [Nat97b], Dieharder [Bro04] and TestU01 [LS07]). The research is a natural continuation of previously published works [Ukr13; Sýs+14].

Following this introduction, in chapter 2, accounts of related work are given. Next, in chapter 3, we summarize the relevant details of the CAESAR competition, its submissions and the procedures necessary to use them. Chapter 4 describes the methodology of the performed experiments, introducing the used tools, their fundamental operating principles and settings. The tests we carried out are reported in chapter 5 along with the interpretation of the measured values. Chapter 6 concludes the thesis by summarizing the main conclusions and proposing directions for the future.

Although the research presented in this thesis was done mostly by myself, plural is used in the thesis text. EACirc and the related research is a result of a wider team\footnote{The most notable people involved in the creation of this thesis include Petr Švenda, Marek Sýs, Karel Kubíček, Jiří Novotný and Lubomír Obrátil.} at the Centre for Research on Cryptography and Security, Masaryk University, where all problems and ideas are discussed together. Parts primarily done by others are properly attributed when mentioned.

The thesis text was typeset in \LaTeX\ using the \texttt{fitthesis2} package created by S. Filipčík [Fil09]. The text of this thesis is licensed under a Creative Commons Attribution 4.0 International License.\footnote{Licence details can be found at \url{https://creativecommons.org/licenses/by/4.0/}.} The icons used in the diagrams were taken from \texttt{The Noun Project}\footnote{Project homepage: \url{https://thenounproject.com}}. The text of this thesis are in public domain or licensed under Creative Commons Attribution 3.0 United States.\footnote{Licence details can be found at \url{https://creativecommons.org/licenses/by/3.0/us/}.}
2 Previous works

The research presented in this thesis combines several tools and ideas. Its primary goal is to assess the output produced by authenticated encryption systems by trying to find a working distinguisher. The summary of this approach previously applied to other categories of cryptoprimitives as well as other research on authenticated encryption schemes is summarized in section 2.1.

Apart from statistical batteries that are commonly used for randomness assessment, we use a novel genetically inspired framework EACirc. Previous research done using this framework can be found in section 2.3 and the explanation of its principles and settings in section 4.2. EACirc is based on genetic programming [Ban+97] – an evolutionary algorithm-based stochastic methodology inspired by biological evolution to find computer programs that perform a user-defined task. Such methodologies have been used in cryptography before – a summary of the most relevant research can be seen in section 2.2.

2.1 Cryptoprimitives assessment

Numerous works tackled the problem of assessing randomness of outputs from cryptoprimitives before. E. Simion [Sim15] gave a nice and readable overview of statistical requirements for cryptographic primitives and the relevance of statistical testing. Usually, statistical testing with standard batteries of tests is performed. The Ph.D. thesis of K. Jakobsson [Jak14] gives both a good theoretical background and a comparison of commonly available tools for random number testing. Its results are based on assessing a variety of pseudo-random and quantum random number generators.

Cryptographic competitions are often the target of these analyses since the unified function API allows for effortless evaluation of a high number of schemes. M. Turan et al. [TDÇ08] performed a detailed examination of eStream phase 2 candidates (both full and reduced-round) with NIST STS and structural randomness tests, finding six ciphers deviating from expected values. In 2010, Doganaksoy et al. [Dog+10] applied the same battery, but only a subset of tests to SHA-3 candidates with a reduced number of rounds as well as only to their compression functions. 256-bit versions of SHA-3 finalists were then subjected to statistical tests using a GPU-accelerated evaluation by A. Kaminsky [Kam12] detecting some deviations in all but the Grøstl algorithm.

CryptoStat [Kam13] constitutes a different view of the problem, using the Bayesian model selection to evaluate the randomness of block ciphers and MACs.

As CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) [CAE13] is an on-going initiative with many submissions (details in chapter 3), there are still not many publications thoroughly examining the security of all the proposed algorithms. F. Abed et al. [AFL14] give an excellent overview of the candidates along with a classification with regard to their core primitives. K. Hakju and K. Kwangjo [HK14] discuss the features of authenticated encryption and predict the essential characteristics of the submissions to survive the CAESAR competition.
2. Previous works

No deeper competition-wide comparison has been done so far – more detailed analysis was performed only on a per-candidate basis. For example, R. Ankele in his Ph.D. thesis [Ank15] analyses the COPA authenticated encryption composition scheme used in several CAESAR candidates. M. Nandi in his 2014 paper [Nan14] demonstrates a forging attack on COBRA and POET designs.

2.2 Genetic algorithms in cryptography

Genetic algorithms were previously applied also in cryptography to some extent. A comprehensive review of the usage up to the year 2004 can be found in B. Delman’s Ph.D. thesis [Del04]. A more recent review is provided in the Ph.D. thesis of S. Picek [Pic15].

For testing randomness of outputs from cryptoprimitives using genetic algorithms, Tiny Encryption Algorithm (TEA) is frequently used. TEA, a simple block cipher designed by D. Wheeler and R. Needham [WN95], constitutes a useful benchmark due to its simple design with multiple repeated rounds.

Starting in 2002 with a paper by J. Hernández et al. [Her+02], statistically significant deviations were found for TEA limited to 1 and 2 rounds. A fixed bitmask with a 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 $\chi^2$ test. Two years later, using the same approach, a similar team published improved results [HI04] detecting deviations for 3 and 4 rounds as well. Subsequent work by W. Hu [Hu10] in 2010 improves an earlier attack with quantum-inspired genetic algorithms, succeeding for TEA reduced for 5 rounds. However, up to the publication of this thesis at the beginning of 2016, no distinguisher for a higher number of rounds was found.

Using a different technique, E. Ma and Ch. Obimbo [MO11] realized an attack on TEA limited to 1 round in 2011. They utilized genetic algorithms and harmony search for the derivation of degenerated keys instead of detection of statistical deviations of output.

2.3 EACirc framework

EACirc [Ś+12] is also based on the techniques of genetic programming but constructs a different type of results when compared to the research presented in section 2.2. Instead of bitmasks, it searches for a program (in the form of a software circuit) working as a randomness distinguisher. Furthermore, EACirc tries to find defects in outputs of cryptoprimitives (such as dependent or biased bits) without directly manipulating plaintexts for the cipher (unlike the case with the evolved bitmasks).

Previously, we used the framework for assessing the randomness of output produced by the round-limited eSTREAM and SHA-3 candidates [Ukr13; ŠUM13; ŠUM14]. To improve the results and increase the number of successfully distinguished rounds, an improved evaluator module based on $\chi^2$-test was developed [Sýs+14]. Although still falling behind in some cases, this improvement enabled us to surpass NIST STS in a few instances.
To be able to compare EACirc’s capabilities with other used methodologies, round-limited TEA was inspected as well [Kub+16]. As previously, our achievements were comparable to the standard statistical batteries, being able to distinguish TEA with 4 or fewer rounds.

During the research, a need arose for supporting tools speeding up and simplifying the performed experiments. On the one hand, implementing crucial parts of the framework in nVidia CUDA suitable for GPU acceleration [Nov15] enabled us to have significantly more test vectors. On the other hand, automating the creation, distribution and evaluation of multiple jobs in a suitable environment with the OneClick tool [Obr15] made quick benchmarking possible and convenient.
3 Authenticated encryption

A cryptosystem for authenticated encryption simultaneously provides confidentiality, integrity, and authenticity assurances on data – decryption is combined in a single step with integrity verification. Authenticated ciphers are often built as various combinations of block ciphers, stream ciphers, message authentication codes, and hash functions. There are many examples commonly used today, such as the Offset codebook mode (OCB) [Rog+01] or Galois/counter mode (GCM) [MV04] based on block ciphers.

Combining confidentiality and integrity assurances into a single scheme has tremendous advantages as combining a confidentiality mode with an authentication mode could be error prone and difficult\textsuperscript{1}. Therefore, following a long tradition of cryptography competitions, CAESAR [CAE13] aims to create a portfolio of authenticated encryption systems intended for wide public adoption.

The object of this thesis is to assess several authenticated encryption schemes with regard to the randomness of produced outputs. To ease the testing of a multitude of systems, we decided to restrict ourselves to just to the CAESAR submissions. This allows us to take advantage of the unified API prescribed by the competition.

The rest of the chapter contains details on the ciphers tested. In section 3.1, CAESAR competition details are given along with general requirements and statistics of all its submissions. In section 3.2, we describe the set of tested functions and their necessary modifications.

3.1 CAESAR competition

CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) [CAE13] is an effort to identify a portfolio of authenticated ciphers that are suitable for widespread adoption and offer an advantage over AES used in Galois/counter mode [MV04].

The contest builds on a strong tradition of focused cryptography competitions believed to have boosted the cryptographic research and enhanced the understanding of the underlying primitives. The first and most well-known was an open competition for a new Advanced Encryption Standard [Nat97a] held in 1997 by the United States National Institute of Standards and Technology (NIST). In 2004, ECRYPT (Network of Excellence funded by the European Union) announced eSTREAM, the ECRYPT Stream Cipher Project [Eur05] calling for a new stream ciphers suitable for widespread adoption. In 2007, NIST announced an open competition for a new hash standard, SHA-3 [Nat07]. Most recently (2013) the crypto community’s efforts were focused on password processing in the Password Hashing Competition [PHC13].

\textsuperscript{1} “It is very easy to accidentally combine secure encryption schemes with secure MACs and still get insecure authenticated encryption schemes.“ [KVW03]
The final deadline for CAESAR submissions was on March 15th, 2014. All 56 proposals were published for detailed evaluation and wider scrutiny. The organizing committee expects three regular rounds and one final before announcing the final portfolio. The authors of the original submissions are allowed to perform further tweaks in the subsequent rounds. The tentative submission deadlines are in mid-2015, the first quarter of 2016 and the beginning of 2017 respectively.

All submissions should be usable in both software and hardware version. Although the first round requires only software implementations, each candidate selected for the second round will also be required to include a reference hardware design. Submitters are free to choose the intellectual property status of their designs (i.e. patented submissions are allowed), but the committee states that patenting a cipher is likely to be considered as a downside.

3.1.1 Candidates requirements

Authenticated ciphers (as required by the CAESAR submission call) take five byte-string inputs and one byte-string output with different security purposes. The inputs are as follows:

- **Key**
  A mandatory byte-string with length fixed to an arbitrary number (nevertheless, it is recommended to support 80, 128 and 256-bit keys).

- **Plaintext**
  A mandatory variable-length input. Designers are permitted to specify the maximum length (but not smaller than 65,536 bytes). The proposed cryptosystem should ensure both integrity and confidentiality for the plaintext.

- **Associated data**
  A mandatory variable-length input, the integrity of which must be preserved by the cipher. The maximum length may be set, but it must not be less than 65,536 bytes.

- **Public message number**
  An optional fixed-length field, the integrity of which must be preserved. Designers may impose single-use limits, see discussion for the secret message number.

- **Secret message number**
  An optional byte-string with fixed length. Both integrity and confidentiality must be retained. The call advises that existing solutions often avoid using secret message numbers. Candidates are expected to maintain security regardless of the way the users choose message numbers. However, ciphers are permitted to lose all security if a single (secret message number, public message number)-pair is used for two encryptions with the same key.

All submissions must accept all byte-strings meeting the specified lengths. Any length limits must be thoroughly justified in the submitted documentation. It is permitted to leak the plaintext length via the ciphertext length (e.g. by having the ciphertext and plaintext length difference be constant).
3. Authenticated encryption

Each submission specifies a family of authenticated ciphers. Family members differ only in parameters (e.g. key length, the number of „rounds“). The list of recommended parameter sets must be prioritized and have at most 10 items with justification for each recommendation. The presented documentation should include the authors, full specification, security goals for each parameter set, security analysis, feature list, design rationale and intellectual property status.

All candidates are required to be self-contained (i.e. the documentation should include all information necessary to implement the cipher from scratch), except for AES encryption and decryption utilities with the key lengths of 128, 192 and 256 bits.

3.1.2 Submissions

There were 56 different designs submitted to the first round. Taking into account all possible parameter sets, this amounts to 172 independent schemes. Till the announcement of the second-round candidates, 9 designs were withdrawn by their authors. On July 7th, 2015, 29 designs were chosen for the second round.

Each first-round candidate is accompanied by a portable reference software implementation. This enables extensive public scrutiny of the design and verification of subsequent implementations. This implementation must cover all recommended parameter sets, and must compute exactly the function specified in the submission. All submissions are available in eBACS, the ECRYPT Benchmarking of Cryptographic Systems [Vir08].

3.2 Tested ciphers

Our goal was to test as many authenticated encryption schemes as possible. Using CAESAR candidates enabled us to test many designs and many configurations automatically due to the shared API. All the candidate codes were taken from the SUPERCOP repository managed by eBACS [Vir08].

In the end, there were 168 different ciphers tested in all performed experiments. From 172 submitted independent schemes (56 designs with different parameter sets), 6 were not tested. Firstly, we could not get the AVALANCHE candidates working properly (segmentation fault while running). Secondly, Julius did not compile due to problems with the inclusion of the external AES routines provider. Thirdly, POLAWIS seemed not to have followed the prescribed API. Lastly, the implementation of PAES is probably faulty, since it did not pass our encrypt-decrypt sanity test. We might have been able to fix most of these cases, but doing so would require extensive interventions in the code increasing the possibility of error. Apart from the submitted candidates, we tested 2 versions of AES/GCM referenced by the CAESAR committee as a design baseline.

On the one hand, some modifications of the source code were needed to compile the ciphers successfully within the EACirc framework. On the other hand, any changes to the implementations increased the possibility of error potentially causing meaningless results. Therefore, an highly cautious approach was taken: All used ciphers were accompanied by a
metadata file and the necessary changes to the source code were thoroughly tracked. The basic metadata file consists of the following:

- Unique identifier of the candidate, the submission family it belongs to and its authors.
- The used implementation type and version along with the exact URL and date of download.
- The summary of necessary changes performed in the scheme’s codebase.

Apart from this file, any change in the code is prefixed with a commentary line starting with a keyword `CHANGE` and a reason for the adjustment (to ease the subsequent localization of changes). Furthermore, in the case of nontrivial modifications, the commentary is followed by the commented original version of the code section.

Due to the high volume of the tested ciphers, the basic adjustments were done automatically using custom-made scripts. This made the subsequent manual inspection of all source files a must. In the process, the following modifications were made (not exhaustive):

- **File renaming and creation of the folder structure**
  Many designers shared the names of the main source files (such as `encrypt.c`). To uniquely identify each source file both for comprehension and the ease of compilation, all files were prefixed by the unique name of the design and its parameter set. The codebase was then hierarchically organized to allow simple work with cipher families.

- **Conversion to C++, header includes resolution**
  To simplify and unify the linking process, all source files written in C were treated as C++ (and therefore renamed to have a `.cpp` suffix). To allow for the file renaming in this and the previous point, all the header file inclusion had to be appropriately adjusted.

- **Namespacing, object interface creation**
  Since all the ciphers were defining the same functions (prescribed by the competition API), the individual implementations needed to be separated. For this, we used C++ namespaces, enclosing each cipher into a distinct virtual space. During the computation, the implementations are accessed via a virtual CAESAR cipher interface with a generated object for each candidate.

- **Dependency resolution**
  As the submission requirements permit the usage of AES routines without implementation, several designs relied on external libraries such as OpenSSL [Ope98]. In other cases, the ciphers depended on other routines available in the SUPERCOP repository [Vir08]. These were separated out and provided to all the candidates in question.

- **Preparation for round limitation**
  To aid the foreseeable future work, an extra variable was added to each namespace via the generated object interface. If realized, it will be used to weaken the cipher design by limiting the number of its internal rounds that would enable us to analyze its security properties more precisely and have a finer comparison of the used tools.
Although it is not used for the presented work at all, the modifications had been done since they imposed a negligible overhead during the automatized mass adjustment phase.

- **Compiler issues resolution**
  During experiments, EACirc is run on both Windows and Linux. Thus, the framework is kept compilable using both *GNU Compiler Collection* and *Microsoft Visual Studio C++ Compiler*. Several changes were, therefore, necessary to ensure a clean build in both these environments, as not all the submissions were perfectly portable. These included, for instance, swapping the variable length arrays allocations for standard dynamic memory management, consolidating data types or enforcing a stricter object management using static casts.

- **Other specific issues**
  For some designs, other minor issues needed to be resolved. To name at least one, *CLOC* and *SILC* candidates lacked accompanying calls for dynamic memory cleanup causing memory leaks. These had to be added since a repeated call for the encryption routine during the generation of the stream intended for statistical batteries (see chapter 5 for details on performed experiments) caused a massive memory consumption increase.
4 Experiment methodology

The main goal of this thesis is to assess randomness of outputs produced by different authenticated encryption schemes. For the ease of implementation, only submissions for the recent CAESAR competition were evaluated (more information on the competition can be found in chapter 3).

The high-level overview of the assessment procedure is summarized below as well as in figure 4.1. The data stream produced by the particular cipher is independently assessed by several statistical batteries. The usage details, as well as used settings, are described in section 4.1. Furthermore, a novel genetically inspired framework EACirc is used to replicate the assessment and provide a different approach to the problem solution. The overview of EACirc, its settings and interpretation of its results is given in section 4.2. The details on CAESAR candidates settings, numerical results and their interpretation can be found in chapter 5. Section 4.3 provides reference experiments for all the used tools.

4.1 Statistical batteries

Evaluating the quality of randomness of a given data stream is a difficult task. In practice, randomness assessment heavily relies on empirical tests of randomness. Each test examines the data from a particular point of view, testing certain statistical features (e.g. the ratio of zeros to ones or the frequency of ones in m-bit blocks). The majority of randomness tests are based on statistical hypothesis testing. The observed characteristics of data are compared with the expected test statistic precomputed for infinite random sequences. However, even a good random number generator sometimes produces sequences (for instance a sequence of many consecutive ones) with characteristics significantly different from the values expected in tests. Therefore, we are unable to distinguish with certainty whether a given sequence with „bad“ characteristics was produced by a defective generator or by a sound generator in a rare case. Thus, the randomness is expressed as a probability, usually in terms of $p$-values. To draw conclusions, we choose a significance level $\alpha$ (the type I error) indicating the likelihood of error when rejecting the randomness of a given sequence.

Since statistical randomness can be tested from many points of view, tests are usually grouped into testing suites to provide more comprehensive randomness analysis. One of the first compact sets of randomness tests was the Diehard Battery of Tests of Randomness by G. Marsaglia [Mar95]. Soon after, the Statistical Test Suite by the National Institute of Standards and Technology (NIST STS) [Nat97b] and the theoretical summary by D. Knuth [Knu97] followed. The NIST STS has a special importance since it was published as a NIST standard and is still used for the preparation of many formal certifications or approvals (for example for selecting AES). As these suites ceased to be maintained and improved, new efforts for a modular and extensible testing frameworks arose. These included Dieharder [Bro04], TestU01 [LS07] and ENT [Wal08] to name just a few.
For the ease of usage (yet still keeping a wider variety of suites) we tested out sequences with the following three statistical testing suites: NIST STS (older, yet still commonly used and a valid NIST standard), Dieharder (modern framework reimplementing other suites as well as adding brand new tests) and TestU01 (another modular framework implementing many tests).

Although the $p$-value of a randomness test focusing on a single characteristic has a clear statistical interpretation, the interpretation of results produced by testing suites is somewhat problematic. We need to determine what number of failed tests allows us to reject randomness of the assessed sequence while respecting the chosen significance level. For this, we use the methods proposed by M. Sýs et al. [Sýs+15]. The resulting threshold for all batteries can be found in the following subsections.

For all experiments, we chose the significance level of $\alpha = 1\%$. This keeps the type I. error (false positives) reasonably low while preventing the type II. error (false negatives) to reach too high values.
4. Experiment methodology

4.1.1 NIST STS

A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [Nat97b] is a battery of statistical tests implemented by National Institute of Standards and Technology. It consists of 15 independent tests. Five of these tests (the cumulative sums test, non-overlapping template matching test, serial test, random excursions test and random excursions variant test) are performed in more variants. These can be seen as separate sub-tests, the whole battery, therefore, amounting to 188 tests altogether. It should be noted that some tests (runs, random excursions and random excursions variant) are not always applicable. These tests are applied only if the sequence meets certain criteria (frequency test passes, the number of cycles is greater than 500). Excluding these, there are 162 tests altogether.

We used NIST STS version 2.1.1. When running the tests, we retained the default parameters (block lengths) for all tests. The confidence level was also left unchanged at the value of $\alpha = 1\%$. To comply with the minimal required stream length for individual tests [Ruk+00], we tested 100 independent 1000 000 bit long sequences for each candidate. In summary, NIST STS used about 12 MiB of input data for each test. For more details on the assessed data stream, see section 5.1.

Regarding the interpretation of a single test, NIST STS adopted the following methods:

- **Proportion of sequences passing the test**
  The relative number of sequences passing the test should lie within a pre-computed interval (the width of the interval depends on the chosen significance level).

- **The uniformity of resulting p-values**
  $p$-values computed for random sequences should be uniformly distributed on the interval $[0, 1]$. Uniformity of $p$-values can be tested using a second-level statistical tests.

For us, the test is considered failed if either of these two methods label the test as failed. We conclude the assessed data stream was not random (i.e. we reject the randomness hypothesis on the significance level of $\alpha = 1\%$) if at least 7 tests fail. This interpretation is based on the research by M. Sýs et al. [Sýs+15].

The main point of the cited paper can be summarized as follows: The chosen significance level (in our case $\alpha = 1\%$) is the type I error for a single test. So assuming the null hypothesis holds, and the tested data is random, each test has $1 - \alpha = 99\%$ chance of passing. That means the probability of all 188 tests passing is $0.99^{188} \approx 0.151 = 15.1\%$ that is far from the chosen significance level. To achieve the required type I error of $1\%$, we must allow some tests to fail. Computing the exact probabilities we get to the chosen significance level for at most 6 failing tests:

$$
\sum_{i=0}^{6} \binom{i}{188} 0.99^{188-i} 0.01^i \approx 0.997 = 99.7\% > 1 - \alpha
$$

This, however, assumes the independence of all tests in the battery. M. Sýs et al. therefore confronted the expected distribution with the distribution for a huge amount of quantum
random data. For NIST STS, the difference is small. Nevertheless, the assumption probably is more violated for other test batteries, as can be seen in our reference experiments (section 4.3).

4.1.2 Dieharder

This testing suite was developed by R. G. Brown at the Duke University [Bro04]. Its main aim is to make the process of testing randomness of bit streams easy while it is still possible for researchers to control tests on a low level. The suite is not just a descendant of Diehard [Mar95], although it mainly includes tests from this suite in an enhanced way. Tests from NIST STS are being incorporated into the battery as well as entirely new tests developed by its authors and other users. This tool features many improvements over Diehard such as full extensibility, simple user interface or open source code.

The complete suite has 26 different tests (18 based on the original Diehard tests, 3 reimplemented from NIST STS and 5 from other sources) as of 2015. Two tests out of these were not run – the Diehard sums test (since its usage is strongly discouraged by the authors) and the Marsaglia and Tsang GCD test (because it requires impractically long input streams for our scenario). Excluding these two and taking into account all the variants, there are 55 tests altogether.

We used Dieharder version 3.31.1. The two parametrizable tests were configured with recommended values (12-tuples for RGB bit distribution test, 8000 points in 2 dimensions for RGB generalized minimum distance test). The default significance level for Dieharder is 0.0001% (strong reject) and 0.5% (weak reject). In order to ease the comparison with the other used batteries, we decided to reset these to the value of 1%. Although this causes a loss of information (which rejects were weak and which were strong), we prefer such setting to improve the comprehension of results presented alongside the outcomes from other tools.

The length of the input stream processed by Dieharder varies from test to test. The humblest required about 48 kiB, while the greediest one of the run tests takes about 9.2 MiB. To ensure the best possible comparability with the other test suites, we again analysed 100 independent samples of the input. In summary, Dieharder tests used between 4.7 MiB and 916 MiB of input data (depending on the particular test). For more details on the assessed data stream, see section 5.1.

The interpretation of the battery results is similar to the case of NIST STS (see section 4.1.1). The situation is slightly simpler as there is only a single pass/fail output for each test (Dieharder does not assess the proportion of the sequences on which the test passed, only the uniformity of the resulting p-values). Using the same methodology as before, we expect the assessed data stream is not random (i.e. we reject the randomness hypothesis on the significance level of $\alpha = 1\%$) if at least 4 tests fail. However, tests in Dieharder does not seem to be completely independent of each other, see results in section 4.3.
4. Experiment methodology

4.1.3 TestU01

TestU01 is a library for empirical testing of random number generators. It was developed at Université de Montréal mainly by Pierre L’Ecuyer [LS07]. The library implements several types of random number generators in a generic form, as well as many specific generators proposed in the literature or found in widely-used software. It provides general implementations of the classical statistical tests for random number generators, as well as several others proposed in the literature, and some original ones.

It implements various sub-batteries intended for different purposes and has a different set of tests. The most relevant sub-batteries are Rabbit, Alphabit and BlockAlphabit. These are intended for testing finite binary sequences. Rabbit and Alphabit apply 38 and 17 different statistical tests respectively. BlockAlphabit applies the Alphabit battery repeatedly after reordering the bits by blocks of different sizes (2, 4, 8, 16 and 32 bits). Including all the applicable sub-tests there are 159 tests. We used TestU01 version 1.2.3.

The length of the input stream taken by TestU01 can be set arbitrarily. To have an amount of data comparable with the other used batteries, we chose to process $2^{30}$ bits for each test. In summary, TestU01 thus used about 128 MiB of input data for each test. For more details on the assessed data stream, see section 5.1.

In order to be as close as possible to the other statistical testing suites, we changed the default value of the significance level (0.1%) to the common level of $\alpha = 1\%$. The interpretation of the battery results is similar to the previous cases (see section 4.1.1, section 4.1.2). Using the same methodology as before, we expect the assessed data stream is not random (i.e. we reject the randomness hypothesis on the significance level of $\alpha = 1\%$) if at least 6 tests fail. However, tests in TestU01 are not independent of each other. The interdependence could have been foreseen, considering the principle of BlockAlphabit. For more details see section 4.3.

4.2 EACirc

In this section, we try to describe the ideas and workings of EACirc [S+12], a novel framework for automatically generating statistical randomness tests. Compared to the standard (manual) way of test creation (as was the case with all the tests used in statistical batteries), our approach has a couple of advantages:

- no prior knowledge of statistical properties of random data is needed;
- test creation does not require excessive human analytical labour;
- tests are dynamically adapting to the tested data;
- atypical and/or yet unknown input data properties may be used to distinguish them from the reference random data.

The main idea is to use supervised learning techniques based on evolutionary algorithms to design and further optimize a successful distinguisher – a test determining whether its input comes from a truly random source or not. The distinguisher will be represented as a
4. Experiment methodology

Figure 4.2: Example of software-emulated circuit as used within EACirc in our experiments.

hardware-like circuit consisting of simple interconnected functions. The evolution will use the principles of genetic programming.

The framework was previously used for assessing randomness of outputs produced by stream ciphers and hash functions [ŠUM13; Sýs+14]. Although some parts of the design have since evolved, most of this section is based on the detailed description published previously [Ukr13; ŠUM14]. The overview of further research based on this tool can be found in section 2.3.

4.2.1 Workflow

EACirc works with a notion of a circuit – a software representation of a hardware-like circuit with nodes (responsible for computation of simple functions, e.g. AND, OR) and connectors (linking node inputs and outputs). A circuit is formed by several layers of such nodes. A node may be connected to any number of nodes from the previous layer – to all, only some of them or none at all. A simple circuit overview can be seen in figure 4.2. Contrary to real single-layer hardware circuits, connectors may also cross each other.

Circuit usage is versatile – from Boolean circuits where functions computed in nodes are limited to logical operators to artificial neural networks where nodes compute the weighted sum of the inputs. Besides studying complexity problems, these circuits were used in various applications such as the design of efficient image filters. Circuit evaluation can be performed by a software emulator or directly in hardware when FPGAs are used.
4. Experiment methodology

EACirc’s main goal is to find a circuit that will reveal an unwanted defect in the inspected cryptographic function. For example, if a circuit can correctly predict the \( n^{th} \) bit of a stream cipher output just by observing the previous \((n-1)\) bits, then this circuit serves as a next-bit predictor [Yao82], breaking the security of the given stream cipher. When a circuit can distinguish the output of the tested function from a truly random sequence, it serves as a random distinguisher [EHJ07] providing a warning sign of function weakness. Note that a circuit does not have to provide correct answers for all inputs – it is sufficient if a correct answer is provided with a probability significantly higher than random guessing.

The greatest challenge is the precise circuit design. It can be laid out by an experienced human analyst (representing a known test, e.g. monobit test) or created and further optimized automatically. We use the latter approach and combine a software circuit evaluated on a CPU/GPU with evolutionary algorithms. The whole process of circuit design, as also depicted in figure 4.3, is as follows:

0. A set of circuits (possible solutions) is initialized by randomly selecting both functions in nodes and connectors in between them. Note that such a random circuit will, most probably, not provide any meaningful output for given inputs and can even have disconnected layers.

1. If necessary, new test vectors used for success evaluation are generated. Half of these is taken from the pseudorandom stream of assessed data (outputs of authenticated encryption schemes in our case) while the other half constitutes a reference sample generated by a truly random generator.

2. Every individual (circuit) in the population is evaluated on all test inputs. The fitness function assigns each circuit a rating based on the obtained outputs (e.g. what fraction of inputs were correctly recognized as being outputs of a stream cipher rather than completely random sequences, see section 4.2.2 for details).

3. What follows is the survival phase, in which worse individuals (the ones with lower fitness value) are removed from the process.

4. Based on the evaluation provided by the fitness function, a potentially improved population is generated from the existing individuals by mutation and sexual crossover. Every individual (circuit) may be changed by altering operations computed in nodes and/or adding/removing connectors between nodes in subsequent layers.

5. The process is repeated from step 2. Usually, hundreds of thousands or more repeats are necessary until the desired success rate of the distinguisher is achieved.

4.2.2 Implementation and settings

EACirc can be configured in many different ways. The most important factors are inner workings of the used genetic operators (initializer, fitness assessment, mutation, crossover). For the genetic programming routines, we used the GAlib genetic algorithm package [Wal95], written by Matthew Wall at the Massachusetts Institute of Technology. However, as our individuals are in a form of software circuits, we had to implement the genetic operators ourselves. The initializer generates the initial circuits at random. The mutator changes
4. Experiment methodology

4. Mutation and sexual crossover
Small random changes in nodes and connectors happen (mutation). Pairs of individuals are crossbred to form offspring, potentially better than parents.

3. Survival of the fittest
Worse individuals are discarded, better survive to the next generation. The higher the fitness, the bigger is the chance of survival.

0. Population
A set of currently considered distinguishers (solutions).

1. Test vector generation
Testing data streams (vectors) are generated from the inspected cryptoprimitive. The same amount of reference streams are sampled from a truly random source.

2. Fitness assessment
Each circuit from the population is evaluated on all test vectors from the current set. Based on the outputs, it is assigned a fitness value from the interval [0,1].

Figure 4.3: A simplified work-flow of the genetic processes in EACirc. The evolution cycle repeats many times. The fitness data from all generations is later analyzed to assess the success of the particular EACirc run.
4. Experiment methodology

every connector and every node with a small (but non-zero) probability. The crossover is performed by cutting 2 parent circuits vertically in two parts and joining a part of each parent to create the offspring. For more details, consult the project codebase [Š+12].

The crucial operator turned out to be the fitness function. Previously [Ukr13; ŠUM13], we used the proportion of correctly identified test vectors (random vs. non-random) as the fitness measure. This proved insufficient, so the assessment was improved [Sýs+14] as demonstrated in figure 4.4. The idea was to take into account the entire distribution of the circuit outputs in the form of a histogram. For example, if the circuit output is a single byte, the observed distribution of its 256 possible values produced by processing all pseudorandom (i.e. cipher-produced) test vectors was to be compared with the expected distribution using a standard χ² test. We approximated the expected distribution by processing the same number of truly random reference data. To solve problems occurring from an inaccurate approximation of expected frequencies, obtained and expected frequencies can be compared using a two-sample test. In our approach, we use the two-sample χ² test [Nat93] since the distribution of test statistic values for two-sample χ² tests is also the χ² distribution. The resulting p-value (expressing the divergence of the distributions produced by truly random and pseudorandom data) is then used as the fitness measure for the circuit’s distinguishing capabilities.

In all experiments presented in this thesis, we used the following settings:

- In each computation, 30 000 generations were evolved.
- Each test vector set consisted of 1000 bitstreams, half of which was produced by the assessed cryptoprimitive while the other reference half was taken from a truly random data source. The test set was renewed in every 100th generation.
- The generation consisted only of a single individual. Although more individuals may increase the success rate and convergence speed towards a well-performing distinguisher, larger populations also cause problems with the interpretation of results (details in section 4.2.3). Therefore, the crossover operator was disabled for now. For each generation, a single new circuit was generated by mutation and the better of the two was passed into the subsequent evolution – an approach similar to hill-climbing heuristics.
- The circuit nodes operate on bytes. The functions applicable are the basic byte manipulation functions (AND, NAND, OR, XOR, NOR, NOT, left and right shifts and rotations, identity, constant function and function selecting only some bits of the input).
- The evolved circuits are 5 layers with 8 nodes in each intermediate layer. The input layer size is of variable width (details in section 5.1); the output layer is a single byte. The inspected output distribution has 8 categories produced by taking the 3 right-most bits of the output byte.

Unfortunately, there are too many variables in the experiments to list the complete settings

---

1. To be precise, the fitness value is \((1 − p\text{-value})\), since more successful individuals (divergent histograms, low p-values) need to have a higher fitness value than less successful ones (similar histograms, higher p-values).
4. Experiment methodology

![Diagram of the summary of fitness computation in EACirc](image)

**Data sources**
Testing data come from two sources:
1) Data produced by the tested cryptoprimitive.
2) Reference data from a random generator.

**Test vector set**
We sample equal amounts of both types of data.

**Circuit evaluation**
All test vectors are fed into the evaluated circuit. Outputs for both input types are recorded.

**Circuit output histograms**
A histogram (frequency of particular byte values) is created from outputs for each input type.

**Output histogram comparison**
The similarity of produced histograms is computed using two-sample \( \chi^2 \)-test.

**Final fitness of the circuit**
The similarity test outputs a \( p \)-value in \([0,1]\). This directly determines the circuit fitness.

Figure 4.4: The summary of fitness computation in EACirc. Each circuit in each generation is evaluated in this way to assign it a fitness value.

Here. Since the experiments are based on previously published works, for detailed settings the reader should refer to papers mentioned in section 2.3 (most notably [SUM14]) or to the data logs in the attachment.

The used settings cause EACirc to process approximately 2.24 MiB of data produced by the tested cryptoprimitive for a single EACirc run assuming 16-byte long test vectors (for details about other used test vector lengths, refer to chapter 5). This amounts to about 2.24 GiB of data for a single experiment. See figure 4.5 to see the reasoning behind this number.

The quality of the reference random data is crucial for the good approximation of the expected frequencies and therefore for the entire fitness assessment. We used a stream of 1.2 GiB obtained from the *High Bit Rate Quantum Random Number Generator Service* [Nan10]. It is a joint research effort of PicoQuant GmbH and the Nano-Optics groups at the Department of Physics of Humboldt University providing random bitstreams based on the quantum randomness of photon arrival times.
4. Experiment methodology

$$\Sigma = 1000 \frac{\text{runs}}{\text{experiment}} \cdot \left( \frac{30000 \text{ generations}}{\text{run}} \cdot \frac{1}{2} \cdot 1000 \frac{\text{vectors}}{\text{test set}} \cdot 16 \frac{\text{bytes}}{\text{vector}} \right) \approx 2.24 \text{ GiB}$$

Figure 4.5: The amount of data analyzed by EACirc for a single experiment assuming the test vectors of 16 bytes.

To leverage the massive computing, the individual evaluation was parallelized [Nov15] using nVidia CUDA. The computations were run in a highly distributed fashion on the computers of the Centre for Research on Cryptography and Security at Masaryk University and the National Grid Infrastructure operated by MetaCentrum [Tea15]. The job management was automated with the Oneclick tool [Obr15] to allow for easy replication and experiment settings improvement.

4.2.3 Results interpretation

To interpret the results of a single EACirc run, we inspect the fitness of individual partial solutions (circuits) in the generations just after the test set change. That is, we are interested in fitness values produced on test vectors never-before-seen by the particular circuit. This mitigates the effect of over-learning (the circuit adapting to a particular set of test vectors, not the general characteristics of the assessed stream).

Provided the assessed data be random (our null hypothesis), the fitness values from these selected generations should be uniformly distributed on the interval $[0, 1]$. If, however, the evolution was able to produce a circuit successfully distinguishing the pseudo-random cipher output from the reference truly random stream, the fitness value distribution will be biased towards the high end of the interval (lower $p$-values).

Therefore, at the end of the computation, we perform a Kolmogorov-Smirnov uniformity test [She03] on the vector of fitness values from the selected generations. The run is considered to have found non-randomness in the data (the uniformity hypothesis is rejected) if the $p$-value resulting from the Kolmogorov-Smirnov test is above the critical value computed for the significance level of $\alpha = 1\%$.

However, due to the randomized nature of generic algorithms, having a single run is insufficient. All EACirc experiments were therefore replicated 1000 times to eliminate the possible statistical anomalies. Provided the underlying assessed data be random, the uniformly distributed fitness values in selected generations imply the uniformity of the $p$-values of the Kolmogorov-Smirnov uniformity tests. Thus, to evaluate the set of 1000 EACirc runs, we inspect the proportion of runs rejecting the null hypothesis (uniformity of the assessed data). If this proportion fluctuates around the set significance level of $\alpha = 1\%$, we cannot reject the hypothesis. If, on the other hand, the proportion wildly deviates from this, we conclude the underlying data was not random with a very high certainty.

To recapitulate the complex interpretation process, see the diagram in figure 4.6.
4. Experiment methodology

To verify at least a basic sanity of the implementation and proposed experiments, we performed a series of reference tests. These tests use the methodology of real experiments (see section 5.1 for details), but use truly random data instead of the pseudo-random cipher stream.

To verify the outputs of statistical batteries, we generated a stream of random data by EACirc. In order to have the process as similar to the real experiments as possible, a mock-up cipher was created and used in place of the CAESAR candidate. This cipher directly outputs the plaintext as a valid ciphertext, prolonged by a 128-bit tag, sampled from the random generator. Thanks to this, the process of generating the random stream used most of the CAESAR-handling routines.

To test the sanity of EACirc, we run the experiments trying to distinguish tags from this random mock-up cipher from truly random data. In summary, we are trying to distinguish one set of truly random data from another set of truly random data. Obviously, we expected to fail at this. As most of these computations are randomized in nature, the testing was replicated 10 times. The results are summarized in table 4.1.

The presented EACirc results confirm our inability of distinguishing two sets of truly random data from each other – the proportion of runs rejecting the null hypothesis oscillates around the significance level of $\alpha = 1\%$. Nevertheless, it can be seen that the proportion dropped below the value of 0.010 only once in 10 replicated experiments. This may have numerous causes: It may just be a rare case. There may be too little random data for such reference experiments (a single experiment processed $1000 \times 2.24 \text{ MiB}$; we have a total
4. Experiment methodology

<table>
<thead>
<tr>
<th>run (id)</th>
<th>EACirc (proportion of rejected)</th>
<th>NIST STS (x/188)</th>
<th>Dieharder (x/55)</th>
<th>TestU01 (x/159)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.011</td>
<td>187</td>
<td>52</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>0.014</td>
<td>187</td>
<td>53</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>0.019</td>
<td>188</td>
<td>52</td>
<td>148</td>
</tr>
<tr>
<td>4</td>
<td>0.012</td>
<td>188</td>
<td>54</td>
<td>150</td>
</tr>
<tr>
<td>5</td>
<td>0.010</td>
<td>188</td>
<td>54</td>
<td>151</td>
</tr>
<tr>
<td>6</td>
<td>0.016</td>
<td>185</td>
<td>53</td>
<td>154</td>
</tr>
<tr>
<td>7</td>
<td>0.008</td>
<td>187</td>
<td>53</td>
<td>154</td>
</tr>
<tr>
<td>8</td>
<td>0.011</td>
<td>188</td>
<td>53</td>
<td>153</td>
</tr>
<tr>
<td>9</td>
<td>0.015</td>
<td>188</td>
<td>51</td>
<td>152</td>
</tr>
</tbody>
</table>

Table 4.1: The results of reference experiments running on truly random data. Columns correspond to four different tools used for the analysis. The cells representing results rejecting the null hypothesis using the theoretical thresholds from section 4.1 are coloured in gray. For further discussion of the results, see section 4.3.

of 1.92GiB of quantum random data). There may be an error in $p$-value evaluation or CAESAR processing functions. To be conservative in all the following experiments, only proportions higher than 2.5% (based on the numbers in the reference experiments) will be considered as outliers signifying the non-randomness of the assessed data.

The outcomes for NIST STS also confirm the previous findings [Sýs+15] – almost all the tests pass. Therefore, in all subsequent experiments, the expected level of 7 or more failed tests for the conclusion of rejected null hypothesis is used.

The case is different for Dieharder and TestU01. The expected thresholds were 4 and 6 failed tests respectively. For Dieharder, this threshold was crossed in one case. This either is a rare case, or it hints at a small interdependence of the tests. To be on the safe side, we advance the limit for the subsequent experiments to 6 or more failed tests. TestU01 exhibits even more surprising behaviour, breaching the threshold in all but two cases. Taking into account also the workings of the battery (BlockAlphabit running the same set of tests 5 times over), the test independence assumption is rather improbable. Nevertheless, as we need a threshold to be able to evaluate the tests in a comprehensible manner, we choose (somewhat arbitrarily) the limit of at least 18 failed tests (three times the expected number) to indicate found non-randomness.

Although the interpretation of results is not ideal, a much deeper analysis of the used test suites would need to be undertaken to achieve more rigorous bounds. That, however, is not in the scope of this thesis and may be performed in subsequent research.
5 Experiment results

In this section, we describe and evaluate several experiments and try to provide conclusions based on the measured results. In a high-level view, we assess the randomness of authentication tags produced by CAESAR candidate ciphers. In total, we tested 168 different ciphers (53 distinct designs) – for details about submissions not tested, see section 3.2. We differentiate three distinct modes of public message numbers (fixed, counter-based and random). Each case is investigated with four separate tools: a novel problem-solving framework based on genetic programming (EACirc) and three statistical testing batteries (NIST STS, Dieharder and TestU01).

Firstly, in section 5.1, we describe the used CAESAR cipher settings and the method of creating the binary streams for statistical test suites. Secondly, in section 5.2, we summarize the interpretation of all numbers presented in the result tables throughout the chapter. In section 5.3, some conclusions from the measured outcomes regarding the particular CAESAR candidates are drawn. Lastly, in section 5.4, we use the obtained results to reason about the randomness testing tools themselves.

5.1 Experiment settings

The aim of the performed experiments is to assess randomness of authentication tags produced by many authenticated encryption systems. In particular, we inspect tags provided by CAESAR candidates initialized as stated below. An outline of tag generation is also given in figure 5.1.

- **Key**
  The key length is determined by the design or the particular parameter set. The key value was taken randomly but was fixed. For EACirc, the 1000 independent runs used different keys to allow for variation (otherwise, the same numerical results would be produced).

- **Associated data, secret message number**
  We used two bytes of associated data; the length of the secret message number was determined by the design or the parameter set. Both fields’ values were fixed to binary zeros. Note that only three designs used secret message numbers.

- **Plaintext**
  The plaintext used is 16 bytes long, formatted as a single counter starting from zero. We could not use fixed-value plaintext, because, in the case of fixed-value public message numbers, the produced tags would be identical (considering settings of the other arguments). A plaintext of binary zeros would have been possible in the other two modes for public message numbers, but we refrained from doing so to keep the experiments as comparable as possible (with as similar settings as possible).

- **Tag length**
  The length of the produced tag (extra ciphertext bytes when compared to the plaintext length) is determined by the cipher design. However, to normalize input width
Figure 5.1: The process of creating the tested bitstream – authentication tags produced by the CAESAR candidate are concatenated together. For detailed explanation of the cipher settings, refer to section 5.1.

For the EACirc circuits, we took only the first 128 bits (16 bytes) from the tag. In the case of shorter tags, the test vectors had 96, 64, 32 or 16 bits (the longest applicable). Shorter test vectors mean EACirc inspected proportionally less data for the candidates producing shorter tags. Since these were the tag lengths advised by the submission call, no bits were dropped from the generated tags in most cases.

- **Public message number**
  
  This was the only parameter explored in different settings. From the nature of the arguments prescribed by the CAESAR API, public message number is probably the argument to be most easily (unintentionally) misused. Security requirements for keys are well known, secret message numbers are usually not used, plaintext and associated data are mostly self-explanatory. Public message numbers are sometimes required to be unique (to have the property of nonces), but sometimes this is not necessary. In a way, we deem testing different modes of public message numbers as examining the robustness of the cipher design.

For testing purposes, we set public message numbers in three distinct ways – each of which was tested separately with all the tools:

1. Fixed to a string of binary zeros for the whole time.
2. Increasing as a counter – each value is unique (but all have a low Hamming weight).
3. Having each value completely random.

EACirc produced the necessary tags on-the-fly. For statistical batteries, a standalone file of 1 GiB was generated using EACirc with identical settings by concatenating the tags. The same file was used for all three suites. To ensure repeatability, the file was generated from
a fixed seed that can be used to re-generate the same stream again, if necessary. Note that
the file for statistical batteries was not composed of the bit-to-bit identical tags as those
generated by EACirc on-the-fly for its own use. The difference was due to other uses of
the randomness generator in the framework during an ordinary run (as opposed to the run
dedicated only to stream generation). Furthermore, EACirc requires multiple randomized
runs to statistically evaluate the experiment. The most notable difference in the produced
tags was the value of the key – fixed to a single value for the file generation (and thus all
the statistical tests) but set to other values during the independent 1 000 runs of EACirc.

Using different keys in independent EACirc runs should not be a problem since we
aim at assessing the cipher’s global behaviour. Although results produced from statistical
batteries (with a single key) may reflect the weakness of the particular key, the chance
of hitting a weak key is minuscule. As a comparison, DES has 64 keys that should be
avoided [Wil04]. Out of the total $2^{56}$ possibilities, this is a negligible amount.

5.2 Interpretation of results

The goal of this section is to comprehensibly describe the meaning of all the numbers
presented in tables 5.1 to 5.8. Rows have all the tested CAESAR candidates, and columns
represent the used randomness testing tools grouped into three column-sets according to
the used public message number setting. Each candidate also states the length of the
test vectors used. The three modes and the reasons for varying test vector lengths are
characterized in section 5.1.

For the ease of high-level comprehension, cells rejecting the null hypothesis (claiming
the tested data is not random with a chance of error $\alpha = 1\%$) are coloured gray. Note the
rejection criterion is different for each tool as recounted below.

The tables list most CAESAR candidates assembled into four sections: the reference
AES/GCM scheme (table 5.1), 2 schemes withdrawn by the authors (table 5.1), 16 schemes
that ended in the first round (tables 5.1 to 5.3) and 27 schemes selected for the second
round (tables 5.4 to 5.8). For details on the candidates not tested, refer to section 3.2.
Within sections, the submissions are ordered alphabetically to correspond with the CAE-
SAR website. The stated folder ID serves as a unique identification of the submission and
its parametrization, used as a folder name in the SUPERCOP repository [Vir08], from
where the source codes were downloaded.

As for the statistical tools themselves, the presented numbers are not directly com-
parable – the interpretation is summarized below. Furthermore, let us note that all the
tools inspected a different amount of data. For settings, interpretation reasoning and other
particulars, see chapter 4.

- EACirc
  The displayed number expresses a proportion of runs rejecting the null hypothesis
  (the uniformity of assessed data) out of 1 000 independent runs. For valid null hy-
  pothesis, it should oscillate around 0.010 (1%). We interpret the value as rejecting
  randomness of the tested data if the proportion is above 0.025.
5. Experiment results

- **NIST STS**
  A number of passed tests is displayed. If 7 or more tests out of the total 188 failed, we would reject the null hypothesis. In cases denoted by a small star (⋆) fewer tests (though, always at least 162) were applicable. The rejection threshold should be adjusted in these cases, but the particular value is not stated since almost the entire test suite failed in all such situations.

- **Dieharder**
  Again, we show the number of passed tests. The total is 55 and the threshold for null hypothesis rejection is set to 6 or more failing.

- **TestU01**
  Once more, we present the number of passed tests out of the total 159. The null hypothesis is rejected if at least 18 tests fail.

5.3 Conclusions for CAESAR candidates

Firstly, let’s compare the outcomes for the three inspected public message number modes. We expected the random-valued to perform the best, followed by counter-based and then by zero-fixed public message numbers. We reasoned that the more differences there will be among the used values, the „easier“ it will be for the cipher to produce a random-looking tag (since it has more entropy to start from). As stated in the submission call, the ciphers were allowed to lose all security in case of reused (public message number, private message number)-pair under the same key. Nevertheless, we expected some (albeit not many) ciphers will be able to retain the apparent randomness of the produced tag – even though it would require an adamant avalanche effect (all arguments are identical apart from a single bit change in plaintext).

From the conducted experiments we see that the primary hypothesis (random values performing better than a counter and much better than zeros) was confirmed. However, none out of the tested candidates passed with the public message numbers fixed to zero. The single bit change in plaintext with all other arguments fixed might not have been enough to cause the avalanche effect needed to produce a tag looking sufficiently random.

Secondly, let’s inspect the results for the individual candidates. Tags of just five designs (AES/GCM, Marble, AEC-CMCC, AES-CPFB, Raviyoyla) were distinguishable from random streams with counter-valued public message numbers. Three of these designs (Marble, AES-CMCC, AES-CPFB) also failed in the random-valued scenario. The evidence is still too weak to deem the designs not secure – it may merely be the case they produce a constant delimiter between the ciphertext and tag, violating the statistical randomness of the created tag. To draw any conclusions, a detailed inspection of the designs (starting with the supporting documentation) would need to be performed. It is, however, worth mentioning that no candidates failing in either counter- or random-valued scenario were selected by the CAESAR committee to the second round of the competition.

There is one more observation to be brought to attention: In two cases (the reference AES/GCM and the first-round AES-CPFB) the 256-bit version passed while the 128-bit
version was found as non-random. This is highly unexpected and deserves further inspection – it may turn out to be a bug in either the implementation of the candidate submission or in our testing methodology.

5.4 Conclusions for randomness testing tools

Apart from the findings for the CAESAR candidates, the results allow us to gain insights into the capabilities of the used randomness testing tools. Based on the previous works [Ukr13; Sýs+14], we expected the randomness distinguishing abilities of EACirc and NIST STS will be similar while both will be surpassed by Dieharder and TestU01. On the one hand, the observed results showed many deficiencies of EACirc – it performed worse than NIST STS in given tested scenarios. On the other, all three statistical batteries achieved comparable results. However, before any conclusions on the quality of the batteries are drawn, one has to be aware there are many domains in which these tools remain incomparable. They inspect different amounts of data and have different modes of operation (batteries see the stream as a whole, EACirc processes short, distinct test vectors).

There is one case contrary to the general behaviour observed above: Raviyoyla with randomly initialized public message numbers for each test vector seems to be successfully rejected from the random stream by EACirc although none of the statistical batteries support such result. It appears very promising but also requires additional inspection and enhanced testing to announce a case of EACirc surpassing all tested statistical batteries.

We can also revisit the set rejection thresholds (although based on the previous research for NIST STS, the set values were adjusted according to the reference experiments in a bit arbitrary way, see section 4.3). The rejection margin for Dieharder (49/55 or less passed tests) was probably too strict. Dieharder’s results were just slightly below the threshold in 11 cases in which no other tool hinted at non-randomness in the data. The tests may be more interdependent than was expected.

The case was the other way around for TestU01 – the rejection threshold (141/159 or less passed tests) may have been too benevolent. All seemingly random streams (based on results from the other tools) have above 150 passed tests while all non-random have below 20. In the end, the tests may have been less interdependent than the reference experiments suggested. The only case that would be influenced by the threshold raise is Raviyoyla – the only one where EACirc alone rejected the randomness hypothesis. After the adjustment, TestU01 would support the non-randomness claim of EACirc.

To determine the most appropriate rejection levels for the used tools, a wider study similar to the one done by M. Sýs for NIST STS [Sýs+15] would need to be carried out.

Regardless of the exact thresholds, it may seem that Dieharder and TestU01 present more reliable conclusions since proportionally fewer tests passed in non-random cases in these batteries when compared to NIST STS. This is a very delicate matter highly dependent on the process of results interpretation. Therefore, we refrain from drawing such conclusion solely based on the results obtained from experiments performed in this thesis.
Table 5.1: The randomness assessment of the reference AES/GCM, two withdrawn and some first-round CAESAR candidates (part 1/3) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
<table>
<thead>
<tr>
<th>Cipher</th>
<th>Folder ID</th>
<th>TV length (bits)</th>
<th>EACirc (proportion)</th>
<th>NIST STS (x/188)</th>
<th>Dielharder (x/55)</th>
<th>TestU01 (x/159)</th>
<th>EACirc (proportion)</th>
<th>NIST STS (x/188)</th>
<th>Dielharder (x/55)</th>
<th>TestU01 (x/159)</th>
<th>EACirc (proportion)</th>
<th>NIST STS (x/188)</th>
<th>Dielharder (x/55)</th>
<th>TestU01 (x/159)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBA cba1</td>
<td></td>
<td>32</td>
<td>0.011 103 3 3 3</td>
<td>0.010 187 54 155</td>
<td>0.011 188 49 157</td>
<td></td>
<td></td>
<td>0.008 187 52 157</td>
<td></td>
<td></td>
<td>0.008 187 52 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba2</td>
<td></td>
<td>32</td>
<td>0.008 93 3 3 3</td>
<td>0.012 187 52 157</td>
<td>0.011 188 52 159</td>
<td></td>
<td></td>
<td>0.011 188 52 159</td>
<td></td>
<td></td>
<td>0.011 188 52 159</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba3</td>
<td></td>
<td>64</td>
<td>0.008 124 6 5 5</td>
<td>0.006 186 54 156</td>
<td>0.009 188 52 158</td>
<td></td>
<td></td>
<td>0.011 188 54 157</td>
<td></td>
<td></td>
<td>0.009 188 54 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba4</td>
<td></td>
<td>64</td>
<td>0.013 140 7 4 4</td>
<td>0.009 188 52 158</td>
<td>0.011 188 52 158</td>
<td></td>
<td></td>
<td>0.009 188 54 159</td>
<td></td>
<td></td>
<td>0.009 188 54 159</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba5</td>
<td></td>
<td>64</td>
<td>0.013 131 5 5 6</td>
<td>0.011 186 52 158</td>
<td>0.011 186 52 158</td>
<td></td>
<td></td>
<td>0.009 188 54 157</td>
<td></td>
<td></td>
<td>0.009 188 54 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba6</td>
<td></td>
<td>96</td>
<td>0.011 137 10 5 10</td>
<td>0.012 187 54 156</td>
<td>0.010 186 55 157</td>
<td></td>
<td></td>
<td>0.013 188 50 156</td>
<td></td>
<td></td>
<td>0.013 188 50 156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba7</td>
<td></td>
<td>96</td>
<td>0.008 139 13 6 6</td>
<td>0.010 186 55 157</td>
<td>0.013 188 50 157</td>
<td></td>
<td></td>
<td>0.006 188 52 156</td>
<td></td>
<td></td>
<td>0.006 188 52 156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba8</td>
<td></td>
<td>96</td>
<td>0.009 147 7 7 7</td>
<td>0.013 188 50 158</td>
<td>0.018 188 53 157</td>
<td></td>
<td></td>
<td>0.006 188 52 156</td>
<td></td>
<td></td>
<td>0.006 188 52 156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba9</td>
<td></td>
<td>64</td>
<td>0.013 131 5 5 7</td>
<td>0.005 186 51 152</td>
<td>0.008 188 53 157</td>
<td></td>
<td></td>
<td>0.008 188 53 157</td>
<td></td>
<td></td>
<td>0.008 188 53 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA cba10</td>
<td></td>
<td>96</td>
<td>0.010 145 11 7 7</td>
<td>0.013 188 53 155</td>
<td>0.013 187 52 158</td>
<td></td>
<td></td>
<td>0.013 187 52 158</td>
<td></td>
<td></td>
<td>0.013 187 52 158</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enchilada</td>
<td>enchilada128v1</td>
<td>128</td>
<td>1.000 71 2 15 15</td>
<td>0.017 187 53 157</td>
<td>0.010 186 52 155</td>
<td></td>
<td></td>
<td>0.016 186 52 155</td>
<td></td>
<td></td>
<td>0.016 186 52 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enchilada</td>
<td>enchilada256v1</td>
<td>128</td>
<td>1.000 77 1 11 11</td>
<td>0.013 188 54 156</td>
<td>0.016 188 53 155</td>
<td></td>
<td></td>
<td>0.007 187 53 155</td>
<td></td>
<td></td>
<td>0.007 187 53 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iFeed[AES]</td>
<td>ifeedaes128n104v1</td>
<td>128</td>
<td>0.017 157 14 11 11</td>
<td>0.017 188 52 157</td>
<td>0.017 187 55 154</td>
<td></td>
<td></td>
<td>0.017 187 55 154</td>
<td></td>
<td></td>
<td>0.017 187 55 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iFeed[AES]</td>
<td>ifeedaes128n96v1</td>
<td>128</td>
<td>0.016 163 10 6 10</td>
<td>0.011 188 53 153</td>
<td>0.012 188 53 155</td>
<td></td>
<td></td>
<td>0.007 187 53 155</td>
<td></td>
<td></td>
<td>0.007 187 53 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCREAM</td>
<td>iscream12v1</td>
<td>128</td>
<td>0.013 163 19 6 6</td>
<td>0.008 187 54 157</td>
<td>0.012 188 52 158</td>
<td></td>
<td></td>
<td>0.012 188 52 158</td>
<td></td>
<td></td>
<td>0.012 188 52 158</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCREAM</td>
<td>iscream12v2</td>
<td>128</td>
<td>0.010 166 14 7 7</td>
<td>0.010 188 54 157</td>
<td>0.018 188 54 157</td>
<td></td>
<td></td>
<td>0.018 188 54 157</td>
<td></td>
<td></td>
<td>0.018 188 54 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCREAM</td>
<td>iscream14v1</td>
<td>128</td>
<td>0.011 172 13 9 9</td>
<td>0.008 186 53 153</td>
<td>0.013 186 54 156</td>
<td></td>
<td></td>
<td>0.013 186 54 156</td>
<td></td>
<td></td>
<td>0.013 186 54 156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCREAM</td>
<td>iscream14v2</td>
<td>128</td>
<td>0.011 159 13 8 8</td>
<td>0.013 188 54 154</td>
<td>0.009 188 54 157</td>
<td></td>
<td></td>
<td>0.009 188 54 157</td>
<td></td>
<td></td>
<td>0.009 188 54 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIASU</td>
<td>kiasueq128v1</td>
<td>128</td>
<td>0.017 164 19 6 6</td>
<td>0.014 188 53 154</td>
<td>0.014 188 55 155</td>
<td></td>
<td></td>
<td>0.014 188 55 155</td>
<td></td>
<td></td>
<td>0.014 188 55 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIASU</td>
<td>kiasuneq128v1</td>
<td>128</td>
<td>0.013 169 18 10 10</td>
<td>0.006 188 53 154</td>
<td>0.014 188 51 157</td>
<td></td>
<td></td>
<td>0.014 188 51 157</td>
<td></td>
<td></td>
<td>0.014 188 51 157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAC</td>
<td>lacv1</td>
<td>64</td>
<td>0.015 139 6 3 3</td>
<td>0.008 187 55 156</td>
<td>0.013 187 53 153</td>
<td></td>
<td></td>
<td>0.013 187 53 153</td>
<td></td>
<td></td>
<td>0.013 187 53 153</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: The assessment of first-round CAESAR candidates (part 2/3) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
<table>
<thead>
<tr>
<th>Cipher</th>
<th>Folder ID</th>
<th>TV length (bits)</th>
<th>PMN fixed to zero</th>
<th>PMN counter-based</th>
<th>PMN truly random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EACirc (proportion)</td>
<td>NIST STS (x/188)</td>
<td>Dieharder (x/185)</td>
</tr>
<tr>
<td>Prost</td>
<td>proest128apev1</td>
<td>128</td>
<td>0.005</td>
<td>161</td>
<td>22</td>
</tr>
<tr>
<td>Prost</td>
<td>proest128copav1</td>
<td>128</td>
<td>0.011</td>
<td>160</td>
<td>15</td>
</tr>
<tr>
<td>Prost</td>
<td>proest128otrv1</td>
<td>128</td>
<td>1.000</td>
<td>*0</td>
<td>0</td>
</tr>
<tr>
<td>Prost</td>
<td>proest256apev1</td>
<td>128</td>
<td>0.017</td>
<td>159</td>
<td>13</td>
</tr>
<tr>
<td>Prost</td>
<td>proest256copav1</td>
<td>128</td>
<td>0.012</td>
<td>158</td>
<td>18</td>
</tr>
<tr>
<td>Prost</td>
<td>proest256otrv1</td>
<td>128</td>
<td>1.000</td>
<td>*0</td>
<td>0</td>
</tr>
<tr>
<td>Raviyoyla</td>
<td>raviyoylav1</td>
<td>128</td>
<td>1.000</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>Sablier</td>
<td>sablierv1</td>
<td>32</td>
<td>1.000</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Silver</td>
<td>silverv1</td>
<td>128</td>
<td>0.017</td>
<td>160</td>
<td>13</td>
</tr>
<tr>
<td>Wheesht</td>
<td>wheeshtv1mr3fr1t128</td>
<td>128</td>
<td>0.016</td>
<td>166</td>
<td>9</td>
</tr>
<tr>
<td>Wheesht</td>
<td>wheeshtv1mr3fr1t256</td>
<td>128</td>
<td>0.018</td>
<td>159</td>
<td>16</td>
</tr>
<tr>
<td>Wheesht</td>
<td>wheeshtv1mr3fr3t256</td>
<td>128</td>
<td>0.014</td>
<td>166</td>
<td>18</td>
</tr>
<tr>
<td>Wheesht</td>
<td>wheeshtv1mr5fr7t256</td>
<td>128</td>
<td>0.012</td>
<td>172</td>
<td>16</td>
</tr>
<tr>
<td>YAES</td>
<td>yaes128v2</td>
<td>128</td>
<td>0.014</td>
<td>160</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 5.3: The assessment of first-round CAESAR candidates (part 3/3) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
<table>
<thead>
<tr>
<th>Cipher</th>
<th>Folder ID</th>
<th>TV length (bits)</th>
<th>PMN fixed to zero</th>
<th>PMN counter-based</th>
<th>PMN truly random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EACirc (proportion)</td>
<td>NIST STS (x/188)</td>
<td>Dieharder (x/55)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACORN acorn128</td>
<td>128</td>
<td>0.009</td>
<td>159</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>AEGIS aegis128</td>
<td>128</td>
<td>0.012</td>
<td>158</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>AEGIS aegis128l</td>
<td>128</td>
<td>0.009</td>
<td>160</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>AEGIS aegis256</td>
<td>128</td>
<td>0.007</td>
<td>155</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>AES-COPA aescopav1</td>
<td>128</td>
<td>0.007</td>
<td>165</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>AES-JAMBU aesjambuv1</td>
<td>64</td>
<td>0.013</td>
<td>132</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>AES-OTR aes128otrpv1</td>
<td>128</td>
<td>0.014</td>
<td>170</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>AES-OTR aes128otrsv1</td>
<td>128</td>
<td>0.011</td>
<td>160</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>AES-OTR aes256otrpv1</td>
<td>128</td>
<td>0.013</td>
<td>160</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>AES-OTR aes256otrsv1</td>
<td>128</td>
<td>0.013</td>
<td>164</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>AEZ aezv1</td>
<td>128</td>
<td>0.014</td>
<td>169</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>AEZ aezv3</td>
<td>128</td>
<td>0.016</td>
<td>164</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Ascon ascon128v1</td>
<td>128</td>
<td>0.011</td>
<td>175</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>Ascon ascon96v1</td>
<td>96</td>
<td>0.009</td>
<td>150</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>CLOC aes128n12clocv1</td>
<td>64</td>
<td>0.009</td>
<td>124</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>CLOC aes128n12clocv1</td>
<td>64</td>
<td>0.010</td>
<td>129</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>CLOC twine80n6clocv1</td>
<td>32</td>
<td>0.010</td>
<td>87</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>SILC aes128n12slicv1</td>
<td>64</td>
<td>0.011</td>
<td>132</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>SILC aes128n6slicv1</td>
<td>64</td>
<td>0.007</td>
<td>132</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>SILC led80n6slicv1</td>
<td>32</td>
<td>0.004</td>
<td>93</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>SILC present80n6slicv1</td>
<td>32</td>
<td>0.010</td>
<td>92</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Deoxys deoxyseq128128v1</td>
<td>128</td>
<td>0.009</td>
<td>168</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Deoxys deoxyseq256128v1</td>
<td>128</td>
<td>0.014</td>
<td>160</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Deoxys deoxysneq128128v1</td>
<td>128</td>
<td>0.016</td>
<td>170</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>Deoxys deoxysneq256128v1</td>
<td>128</td>
<td>0.011</td>
<td>162</td>
<td>18</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5.4: The assessment of second-round CAESAR candidates (part 1/5) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
<table>
<thead>
<tr>
<th>Cipher</th>
<th>Folder ID</th>
<th>TV length</th>
<th>PMN fixed to zero</th>
<th>PMN counter-based</th>
<th>PMN truly random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EACirc (proportion)</td>
<td>NIST STS (x/188)</td>
<td>Dieharder (x/55)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELmD</td>
<td>elmd1000v1</td>
<td>128</td>
<td>0.012</td>
<td>161</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>elmd1001v1</td>
<td>128</td>
<td>0.009</td>
<td>166</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>elmd101270v1</td>
<td>128</td>
<td>0.009</td>
<td>163</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>elmd101271v1</td>
<td>128</td>
<td>0.013</td>
<td>162</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>elmd500v1</td>
<td>128</td>
<td>0.009</td>
<td>172</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>elmd501v1</td>
<td>128</td>
<td>0.012</td>
<td>166</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>elmd51270v1</td>
<td>128</td>
<td>0.013</td>
<td>165</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>elmd51271v1</td>
<td>128</td>
<td>0.016</td>
<td>163</td>
<td>15</td>
</tr>
<tr>
<td>HSI-SIV</td>
<td>hs1sivhiv1</td>
<td>128</td>
<td>0.016</td>
<td>172</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>hs1sivlov1</td>
<td>64</td>
<td>0.012</td>
<td>131</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>hs1sivv1</td>
<td>128</td>
<td>0.010</td>
<td>165</td>
<td>14</td>
</tr>
<tr>
<td>ICEPOLE</td>
<td>icepole128av1</td>
<td>128</td>
<td>0.012</td>
<td>166</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>icepole128v1</td>
<td>128</td>
<td>1.000</td>
<td>*1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>icepole256av1</td>
<td>128</td>
<td>0.013</td>
<td>169</td>
<td>17</td>
</tr>
<tr>
<td>Joltik</td>
<td>joltikeq12864v1</td>
<td>64</td>
<td>0.009</td>
<td>142</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>joltikeq6464v1</td>
<td>64</td>
<td>0.010</td>
<td>140</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>joltikeq8048v1</td>
<td>64</td>
<td>0.009</td>
<td>137</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>joltikeq9696v1</td>
<td>64</td>
<td>0.011</td>
<td>119</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>joltiknq12864v1</td>
<td>64</td>
<td>0.017</td>
<td>141</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>joltiknq6464v1</td>
<td>64</td>
<td>0.012</td>
<td>130</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>joltiknq8048v1</td>
<td>64</td>
<td>0.001</td>
<td>134</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>joltiknq9696v1</td>
<td>64</td>
<td>0.004</td>
<td>133</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5.5: The assessment of second-round CAESAR candidates (part 2/5) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
<table>
<thead>
<tr>
<th>Cipher (official name)</th>
<th>Folder ID (as used in SUPERCOP [Vir08])</th>
<th>TV length (bits)</th>
<th>PMN fixed to zero</th>
<th>PMN counter-based</th>
<th>PMN truly random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ketje</td>
<td>ketjejrv1</td>
<td>96</td>
<td>0.014 146 8 10</td>
<td>0.012 188 54 155</td>
<td>0.008 187 55 156</td>
</tr>
<tr>
<td>Ketje</td>
<td>ketjesrv1</td>
<td>128</td>
<td>0.016 170 18 7</td>
<td>0.020 188 54 156</td>
<td>0.016 187 49 157</td>
</tr>
<tr>
<td>Keyak</td>
<td>lakekeyakv1</td>
<td>128</td>
<td>0.018 165 14 6</td>
<td>0.010 187 54 159</td>
<td>0.007 188 54 156</td>
</tr>
<tr>
<td>Keyak</td>
<td>oceanekeyakv1</td>
<td>128</td>
<td>0.009 160 7 4</td>
<td>0.009 187 54 155</td>
<td>0.013 188 54 156</td>
</tr>
<tr>
<td>Keyak</td>
<td>riverkeyakv1</td>
<td>128</td>
<td>0.010 163 9 8</td>
<td>0.023 188 54 151</td>
<td>0.015 188 52 155</td>
</tr>
<tr>
<td>Keyak</td>
<td>seakeyakv1</td>
<td>128</td>
<td>0.021 163 18 8</td>
<td>0.011 188 53 157</td>
<td>0.011 188 52 156</td>
</tr>
<tr>
<td>Minalpher</td>
<td>minalpherv1</td>
<td>128</td>
<td>0.010 159 20 9</td>
<td>0.008 187 52 158</td>
<td>0.013 188 54 155</td>
</tr>
<tr>
<td>MORUS</td>
<td>morus1280128v1</td>
<td>128</td>
<td>0.005 160 9 8</td>
<td>0.017 188 51 153</td>
<td>0.011 188 51 156</td>
</tr>
<tr>
<td>MORUS</td>
<td>morus1280256v1</td>
<td>128</td>
<td>0.017 161 8 9</td>
<td>0.009 187 55 154</td>
<td>0.009 188 54 156</td>
</tr>
<tr>
<td>MORUS</td>
<td>morus640128v1</td>
<td>128</td>
<td>0.009 165 15 5</td>
<td>0.018 188 54 157</td>
<td>0.014 188 52 156</td>
</tr>
<tr>
<td>NORX</td>
<td>norx3241v1</td>
<td>128</td>
<td>0.014 165 14 9</td>
<td>0.008 188 52 159</td>
<td>0.011 186 55 158</td>
</tr>
<tr>
<td>NORX</td>
<td>norx3261v1</td>
<td>128</td>
<td>0.013 166 18 6</td>
<td>0.008 188 53 157</td>
<td>0.016 187 50 156</td>
</tr>
<tr>
<td>NORX</td>
<td>norx6441v1</td>
<td>128</td>
<td>0.014 156 16 12</td>
<td>0.009 187 51 155</td>
<td>0.018 187 51 154</td>
</tr>
<tr>
<td>NORX</td>
<td>norx6444v1</td>
<td>128</td>
<td>0.013 167 16 8</td>
<td>0.013 186 53 156</td>
<td>0.011 188 54 153</td>
</tr>
<tr>
<td>NORX</td>
<td>norx6461v1</td>
<td>128</td>
<td>0.016 162 10 7</td>
<td>0.010 187 52 156</td>
<td>0.014 187 54 156</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha256k128n96tau128v1</td>
<td>128</td>
<td>0.012 160 15 5</td>
<td>0.012 187 53 156</td>
<td>0.017 188 55 155</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha256k128n96tau64v1</td>
<td>64</td>
<td>0.013 135 10 6</td>
<td>0.008 185 54 156</td>
<td>0.009 187 53 154</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha256k128n96tau96v1</td>
<td>96</td>
<td>0.010 157 19 9</td>
<td>0.014 188 54 157</td>
<td>0.013 187 53 156</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha256k192n104tau128v1</td>
<td>64</td>
<td>0.010 138 6 6</td>
<td>0.008 187 50 156</td>
<td>0.008 187 51 156</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha256k256n104tau160v1</td>
<td>128</td>
<td>0.014 170 16 9</td>
<td>0.015 187 53 157</td>
<td>0.012 188 53 155</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha256k256n248tau256v1</td>
<td>128</td>
<td>0.010 160 15 8</td>
<td>0.011 186 51 155</td>
<td>0.011 187 53 155</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha512k128n128tau128v1</td>
<td>128</td>
<td>0.011 171 13 7</td>
<td>0.010 188 51 157</td>
<td>0.005 188 52 155</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha512k256n256tau256v1</td>
<td>128</td>
<td>0.017 167 10 6</td>
<td>0.010 188 52 156</td>
<td>0.009 188 52 158</td>
</tr>
<tr>
<td>OMD</td>
<td>omdsha512k512n256tau256v1</td>
<td>128</td>
<td>0.017 159 12 7</td>
<td>0.017 187 55 156</td>
<td>0.015 188 50 156</td>
</tr>
</tbody>
</table>

Table 5.6: The assessment of second-round CAESAR candidates (part 3/5) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
<table>
<thead>
<tr>
<th>Cipher</th>
<th>Folder ID (official name)</th>
<th>TV length (bits)</th>
<th>PMN fixed to zero</th>
<th>PMN counter-based</th>
<th>PMN truly random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EACirc (proportion)</td>
<td>NIST STS (x/188)</td>
<td>Dieharder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(x/55)</td>
<td>TestU01 (x/159)</td>
<td>EACirc (proportion)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(x/55)</td>
<td>TestU01 (x/159)</td>
<td>EACirc (proportion)</td>
</tr>
<tr>
<td>PAEQ</td>
<td>paeq128</td>
<td>128</td>
<td>0.010 164.16 7</td>
<td>0.012 188.52 155</td>
<td>0.015 187.50 157</td>
</tr>
<tr>
<td>PAEQ</td>
<td>paeq128t</td>
<td>128</td>
<td>0.017 168.7 8</td>
<td>0.009 188.55 158</td>
<td>0.011 188.50 158</td>
</tr>
<tr>
<td>PAEQ</td>
<td>paeq128tnm</td>
<td>128</td>
<td>0.020 158.20 9</td>
<td>0.014 188.54 155</td>
<td>0.008 186.54 158</td>
</tr>
<tr>
<td>PAEQ</td>
<td>paeq160</td>
<td>128</td>
<td>0.013 158.11 5</td>
<td>0.013 187.53 155</td>
<td>0.013 187.54 157</td>
</tr>
<tr>
<td>PAEQ</td>
<td>paeq64</td>
<td>64</td>
<td>0.016 142.12 9</td>
<td>0.016 188.49 157</td>
<td>0.008 188.52 156</td>
</tr>
<tr>
<td>PAEQ</td>
<td>paeq80</td>
<td>64</td>
<td>0.016 130.6 5</td>
<td>0.010 187.52 156</td>
<td>0.017 188.54 152</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi16cipher096v1</td>
<td>128</td>
<td>1.000 *1 0 3</td>
<td>0.012 187.53 159</td>
<td>0.014 188.52 153</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi16cipher128v1</td>
<td>128</td>
<td>1.000 *1 0 2</td>
<td>0.015 188.55 157</td>
<td>0.009 188.53 153</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi32cipher128v1</td>
<td>128</td>
<td>1.000 *0 0 0</td>
<td>0.012 188.52 158</td>
<td>0.008 187.53 155</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi32cipher256v1</td>
<td>128</td>
<td>1.000 *0 0 2</td>
<td>0.011 188.54 156</td>
<td>0.015 188.53 152</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi64cipher128v1</td>
<td>128</td>
<td>1.000 *0 0 2</td>
<td>0.011 187.52 156</td>
<td>0.010 186.54 156</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi64cipher256v1</td>
<td>128</td>
<td>1.000 *0 0 4</td>
<td>0.014 188 48 156</td>
<td>0.008 185.53 154</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi64cipher256v1onerround</td>
<td>128</td>
<td>1.000 *0 0 8</td>
<td>0.012 187.52 155</td>
<td>0.011 187.53 156</td>
</tr>
<tr>
<td>π-Cipher</td>
<td>pi64cipher256v1tworounds</td>
<td>128</td>
<td>1.000 *0 0 2</td>
<td>0.010 187.53 155</td>
<td>0.013 187.53 155</td>
</tr>
<tr>
<td>POET</td>
<td>aes128poetv1aes128</td>
<td>128</td>
<td>0.009 169 14 4</td>
<td>0.014 188.52 156</td>
<td>0.015 188.54 154</td>
</tr>
<tr>
<td>POET</td>
<td>aes128poetv1aes4</td>
<td>128</td>
<td>0.010 172 11 8</td>
<td>0.013 187.54 157</td>
<td>0.016 188.52 157</td>
</tr>
<tr>
<td>PRIMATEs</td>
<td>primatesv1ape120</td>
<td>128</td>
<td>0.013 169 17 11</td>
<td>0.013 187.55 157</td>
<td>0.013 187.51 156</td>
</tr>
<tr>
<td>PRIMATEs</td>
<td>primatesv1ape80</td>
<td>128</td>
<td>0.019 159 16 8</td>
<td>0.013 188.52 157</td>
<td>0.013 187.54 156</td>
</tr>
<tr>
<td>PRIMATEs</td>
<td>primatesv1gibbon120</td>
<td>96</td>
<td>0.007 154 12 5</td>
<td>0.014 188.55 154</td>
<td>0.015 188.54 157</td>
</tr>
<tr>
<td>PRIMATEs</td>
<td>primatesv1gibbon80</td>
<td>64</td>
<td>0.010 128 4 5</td>
<td>0.013 186.52 156</td>
<td>0.006 187.53 156</td>
</tr>
<tr>
<td>PRIMATEs</td>
<td>primatesv1hanuman120</td>
<td>96</td>
<td>0.009 130 12 7</td>
<td>0.009 188.53 158</td>
<td>0.016 188.53 156</td>
</tr>
<tr>
<td>PRIMATEs</td>
<td>primatesv1hanuman80</td>
<td>64</td>
<td>0.012 148 13 3</td>
<td>0.007 188.53 153</td>
<td>0.011 186.54 157</td>
</tr>
</tbody>
</table>

Table 5.7: The assessment of second-round CAESAR candidates (part 4/5) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.
### Table 5.8: The assessment of second-round CAESAR candidates (part 5/5) in three different public message number modes. For interpretation of displayed numbers and signs, see section 5.2. For drawn conclusions, see section 5.3 and section 5.4.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Folder ID</th>
<th>TV length</th>
<th>PMN fixed to zero</th>
<th>PMN counter-based</th>
<th>PMN truly random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EACirc (proportion)</td>
<td>NIST STS (x/188)</td>
<td>EACirc (proportion)</td>
</tr>
<tr>
<td>SCREAM</td>
<td>scream10v1</td>
<td>128</td>
<td>0.015 156 18 4</td>
<td>0.014 187 55 155</td>
<td>0.015 188 54 156</td>
</tr>
<tr>
<td>SCREAM</td>
<td>scream10v2</td>
<td>128</td>
<td>0.012 163 17 11</td>
<td>0.010 188 52 157</td>
<td>0.013 188 53 154</td>
</tr>
<tr>
<td>SCREAM</td>
<td>scream12v1</td>
<td>128</td>
<td>0.018 165 13 6</td>
<td>0.011 187 55 158</td>
<td>0.013 188 53 157</td>
</tr>
<tr>
<td>SCREAM</td>
<td>scream12v2</td>
<td>128</td>
<td>0.017 159 16 12</td>
<td>0.010 186 53 154</td>
<td>0.015 188 52 153</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d4n64</td>
<td>128</td>
<td>0.009 169 18 4</td>
<td>0.010 187 53 156</td>
<td>0.019 187 53 153</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d4n80</td>
<td>128</td>
<td>0.017 169 18 4</td>
<td>0.007 187 53 156</td>
<td>0.015 188 52 153</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d5n64</td>
<td>128</td>
<td>0.011 164 22 10</td>
<td>0.011 188 49 157</td>
<td>0.007 188 53 154</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d5n80</td>
<td>128</td>
<td>0.009 164 22 10</td>
<td>0.013 188 49 157</td>
<td>0.017 187 55 156</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d6n64</td>
<td>128</td>
<td>0.014 160 24 13</td>
<td>0.013 186 54 155</td>
<td>0.013 187 51 159</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d6n80</td>
<td>128</td>
<td>0.012 160 24 13</td>
<td>0.013 186 54 155</td>
<td>0.010 188 52 157</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d7n64</td>
<td>128</td>
<td>0.019 163 17 9</td>
<td>0.018 186 54 156</td>
<td>0.008 187 53 159</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d7n80</td>
<td>128</td>
<td>0.019 163 17 9</td>
<td>0.018 186 54 155</td>
<td>0.014 186 53 155</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d8n64</td>
<td>128</td>
<td>0.015 176 15 8</td>
<td>0.010 188 51 157</td>
<td>0.016 187 53 157</td>
</tr>
<tr>
<td>SHELL</td>
<td>shellaes128v1d8n80</td>
<td>128</td>
<td>0.013 176 15 12</td>
<td>0.010 188 51 157</td>
<td>0.016 187 53 155</td>
</tr>
<tr>
<td>STRIBOB</td>
<td>stribob192v1</td>
<td>128</td>
<td>0.016 158 18 12</td>
<td>0.011 187 54 154</td>
<td>0.014 188 53 156</td>
</tr>
<tr>
<td>Tiaoxin</td>
<td>tiaoxinv1</td>
<td>128</td>
<td>0.009 161 24 8</td>
<td>0.012 188 53 158</td>
<td>0.018 188 53 153</td>
</tr>
<tr>
<td>TriviA-ck</td>
<td>trivia0v1</td>
<td>128</td>
<td>0.999 140 5 8</td>
<td>0.015 186 52 157</td>
<td>0.005 187 55 154</td>
</tr>
<tr>
<td>TriviA-ck</td>
<td>trivi128v1</td>
<td>128</td>
<td>0.993 158 12 8</td>
<td>0.017 188 53 158</td>
<td>0.009 188 54 157</td>
</tr>
</tbody>
</table>
6 Summary

The aim of this thesis was to analyze randomness of multiple authenticated encryption systems. In the end, we assessed outputs from 168 distinct schemes (all but six CAESAR submissions) in three different settings (public message number modes) using four different software tools (EACirc, NIST STS, Dieharder and TestU01).

There is a new module for EACirc enabling the randomness assessment of tags produced by authenticated encryption systems. Within this module, there is a codebase of all CAESAR submissions coherently separated as C++ objects compiling in both Windows and Unix environments. The codebase has a comprehensive change tracking system enabling additional detailed inspection of performed changes. There is a set of scripts allowing effortless running of statistical tests.

The conducted tests, taking together thousands of CPU days, were distributed on tens of cores in two facilities. All necessary metadata of all the performed experiments is retained to allow for easy replication if necessary.

6.1 High-level conclusions

The obtained results can be understood in two fundamentally different ways. Looking at the tables row-wise, conclusions can be made for individual CAESAR candidates. Inspecting them column-wise gives us insights into the used randomness testing tools and their relative qualities.

We examined a scenario with random (but fixed) keys, counter-based plaintext and three different settings of public message numbers. As expected, tags produced in configurations with random public message numbers fared better than the ones from counter-based configurations. Both did better than tags from fixed-value public message numbers – no submission had an avalanche effect strong enough to produce random-looking tags in the scenario where all test vectors had the same public message numbers.

Only five CAESAR submissions (AES/GCM, Marble, AEC-CMCC, AES-CPFB, Raviyoyla) failed to produce seemingly random tags with counter-based public message numbers. For entirely random public message numbers, three candidates failed (Marble, AES-CMCC, AES-CPFB). Importantly, none of these candidates made it to the second round of the competition. Interestingly, in two designs the 256-bit versions significantly outperformed the 128-bit versions.

Regarding the tools used for tag evaluation, EACirc seemed to be the least suitable for the given task, being beaten by all the statistical batteries. The batteries themselves (NIST STS, Dieharder and TestU01) produced comparable results. The only exception is the case of Raviyoyla, in which EACirc seems to have outperformed all the other tools. However, when making comparisons, one has to take into account the amount of data inspected by each tool and their different modes of operation.
It turned out that interpreting the results of sets of statistical tests is far from straight-forward. The set rejection thresholds influence the conclusions about battery capabilities a lot. To reliably assign such thresholds for Dieharder and TestU01, a thorough inspection of the test interdependence, similar to the one done for NIST STS [Sýs+15], would need to be carried out.

### 6.2 Proposed future work

The results lead us to several interesting hypotheses requiring further inspection. The candidates exhibiting differences in versions with different parameter sets, as well as those failing in randomness tests, would deserve a deeper manual inspection to prove their potential (in)security.

The used statistical testing suites themselves would be an interesting target for further research. It turned out that interpretation of test suites is quite difficult, and thorough research on test interdependence is necessary. Another prospective direction would be weakening the cipher designs (e.g. by limiting the number of internal rounds) to achieve a fine-grained comparison of the used tools.

A new area of available research concerns the novel EACirc framework. Probably the most beneficial for the cryptographic community would be an analysis of the evolved distinguisher circuits as this could hint at particular cipher weaknesses. Modifications enabling EACirc to process longer inputs or keep custom information from the previously tested vectors might also be helpful. The back-end features of the framework could make advantage of other existing heuristics – we consider superseding the circuits by individuals in the form of simple polynomials in hopes of speeding the evaluation, easing the interpretation and enabling the use of existing analytical tools.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


A Data attachment

The data attachment available in the thesis repository\(^1\) contains source codes and most experimental results organized in the following structure:

- **eacirc**
  Source codes of EACirc (copy of entire project repository with master commit 35c0c0a from 2016-01-05).

- **eacirc-wiki**
  Project’s wiki-based documentation (copy of entire wiki repository with master commit 34cec22 from 2016-01-05).

- **data-eacirc**
  Underlying data for EACirc results presented in chapter 5 further divided into subdirectories according to the public message number mode (pmn-zero, pmn-counter and pmn-random). Only a sample of 10 EACirc runs from 1000 is provided for each case due to size constraints.

- **data-statistical-batteries**
  Underlying data for results of statistical batteries presented in chapter 5 further divided into subdirectories according to the public message number mode (pmn-zero, pmn-counter and pmn-random). The tested binary files are omitted due to their size.

- **data-reference**
  Underlying data for reference results presented in section 4.3 for all four statistical testing tools. The tested binary files are omitted due to their size. Only a sample of 10 EACirc runs from 1000 is provided for each case (again, due to size constraints).

- **thesis-src**
  Thesis text source files including bibliography and used images (repository commit 178f932 from 2016-01-09).

\(^1\) [http://is.muni.cz/th/374297/fi_m/](http://is.muni.cz/th/374297/fi_m/)