Avalanche Effect in Improperly Initialized CAESAR Candidates

Martin Ukrop
Centre for Research on Cryptography and Security,
Faculty of Informatics,
Masaryk University, Brno, Czech Republic
mukrop@mail.muni.cz

Petr Švenda
Centre for Research on Cryptography and Security,
Faculty of Informatics,
Masaryk University, Brno, Czech Republic
svenda@fi.muni.cz

Cryptoprimitives rely on thorough theoretical background, but often lack basic usability features making them prone to unintentional misuse by developers. We argue that this is true even for the state-of-the-art designs. Analyzing 52 candidates of the current CAESAR competition has shown none of them have avalanche effect in authentication tag strong enough to work properly when partially misconfigured. Although not directly decreasing their security profile, this hints at their security usability being less than perfect.[1]

1 Introduction

Nowadays, experts realize that having cryptography attack-resistant from the theoretical point of view is not sufficient since many attacks are caused by improper use of otherwise sound cryptographic primitives. Developers routinely produce horrendous implementations (at least from the point of security) when they neglect to properly set initialization vectors or ignore the requirement of unique sequence numbers. Recently, Cairns and Steel outlined their vision for developer-resistant cryptography [5] with designs that cannot be misused by the programmer.

The question the security-optimist would ask is: *Is that not the case only for old primitives, old protocols and old designs? Are new designs also prone to developer misuse?* We argue the problem is still open – we tested 52 participants of the current state-of-the-art cryptographic competition by checking the avalanche effect of the candidates in settings simulating partial misconfiguration.

It is long known that cryptographic primitives such as ciphers, hash functions and message authentication codes should produce seemingly random outputs. Further requirements ask for outputs to change unpredictably with respect to changes in the input. The strict avalanche criterion, as introduced by Webster and Tavares in 1985 [23], is one way to formalize this. It is satisfied if, whenever a single input bit is complemented, each of the output bits changes with a 50% probability. It is commonly used for assessing the security of hash functions, though using it as a randomness test has also been done before [6].

In this paper, we scrutinize submissions of the ongoing CAESAR competition (*Competition for Authenticated Encryption: Security, Applicability, and Robustness*) [4]. The authentication tags produced by all candidates are examined using four different software tools: three standard statistical batteries (NIST STS [14], Dieharder [3] and TestU01 [11]) and a novel genetically-inspired framework (EACirc [22]). The overview of the main experiment idea is depicted in Figure 1.

The analysis was done separately for three different settings of the public message number (fixing it to zero, using a counter and generating unique random value each time). It turned out that none of the tested CAESAR candidates had an avalanche effect strong enough to produce random-looking tags in

[1] Paper details available at crcs.cz/papers/memics2016

Submitted to:
MEMICS 2016

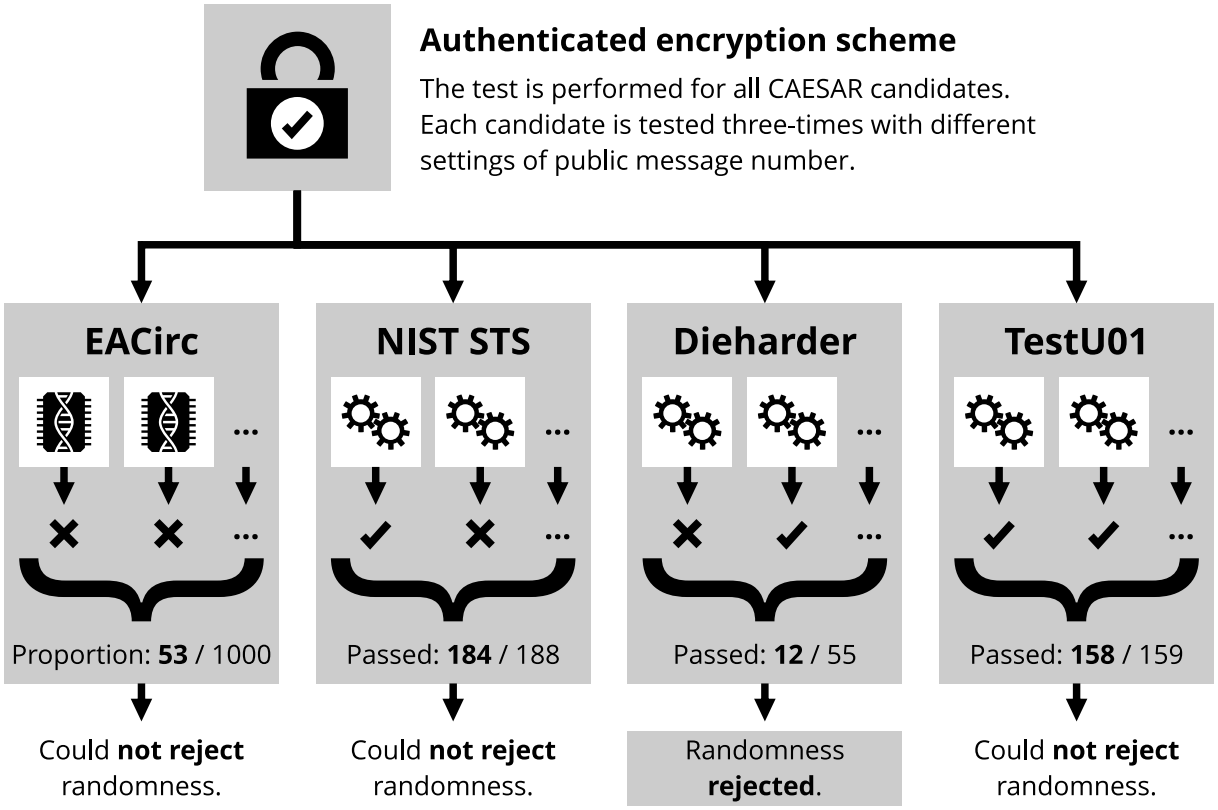

Figure 1: High-level overview of the performed experiments. Firstly, a CAESAR cipher is used to generate a stream of authentication tags. The randomness of this stream is then assessed by 4 tools (EACirc and 3 statistical testing suites). If the design is good, it should exhibit an avalanche effect strong enough for the stream to look random.

the most seriously misconfigured case with zero public message numbers (thus avalanching from only a very few changed bits in plaintext). In the case of counter-based and random public message numbers, the ciphers fared much better.

Firstly, in Section 2, the paper gives an overview of the related research. Then the basics of authenticated encryption are explained along with the essentials of CAESAR competition (Section 3). The following sections summarize the way of generating the tested data (Section 4) and the tools used for the analysis (Section 5). Lastly, the results and their interpretation are given in Section 6.

2 Related research

As CAESAR competition is an on-going initiative with many submissions, there are still not many publications thoroughly examining the security of all the proposed algorithms. F. Abed et al. [1] give an excellent overview of the candidates along with a classification with regard to their core primitives. K. Hakju and K. Kwangjo [8] discuss the features of authenticated encryption and predict the essential characteristics of the submissions to survive the CAESAR competition.

Probably the most comprehensive competition-wide analysis so far has been done by M. Saari-

nen [15] using the BRUTUS automatic cryptanalytic framework. Deeper analysis exists only on a per-candidate basis. For example, R. Ankele in his Ph.D. thesis [2] analyses the *COPA* authenticated encryption composition scheme used in several CAESAR candidates. M. Nandi in his 2014 paper [13] demonstrates a forging attack on *COBRA* and *POET* ciphers.

Numerous works tackled the problem of assessing randomness of outputs from other cryptoprimitives. E. Simion [16] gave a nice overview of statistical requirements for cryptographic primitives in his work. The Ph.D. thesis of K. Jakobsson [9] 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. [19] 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. [7] 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.

A different strategy is employed in the EACirc framework – it uses a genetically-inspired process to find a successful distinguisher (function capable for differencing between cipher output and random stream). The framework has been used for assessing the randomness of outputs produced by the round-limited eSTREAM and SHA-3 candidates [22, 18]. Although still falling behind in some cases, this approach surpasses NIST STS in a few instances.

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 *Galois/counter mode* (GCM) [12] 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[2]. Therefore, following a long tradition of cryptography competitions, CAESAR [4] aims to create a portfolio of authenticated encryption systems intended for wide public adoption.

Each submission in CAESAR specifies a family of authenticated ciphers. Family members differ only in parameters (e.g. key length, the number of internal rounds). 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 ciphers were withdrawn by their authors. On July 7th 2015, 29 ciphers were chosen for the second round. Later, on August 15th 2016, 15 ciphers out of these were selected for the third round.

Our goal was to test as many authenticated encryption schemes as possible. Using CAESAR candidates enabled us to test many ciphers and many configurations automatically due to the shared API. All the candidate source codes were taken from the 1st-round SUPERCOP repository managed by eBACS [21].

[2] *"It is very easy to accidentally combine secure encryption schemes with secure MACs and still get insecure authenticated encryption schemes."* [10]

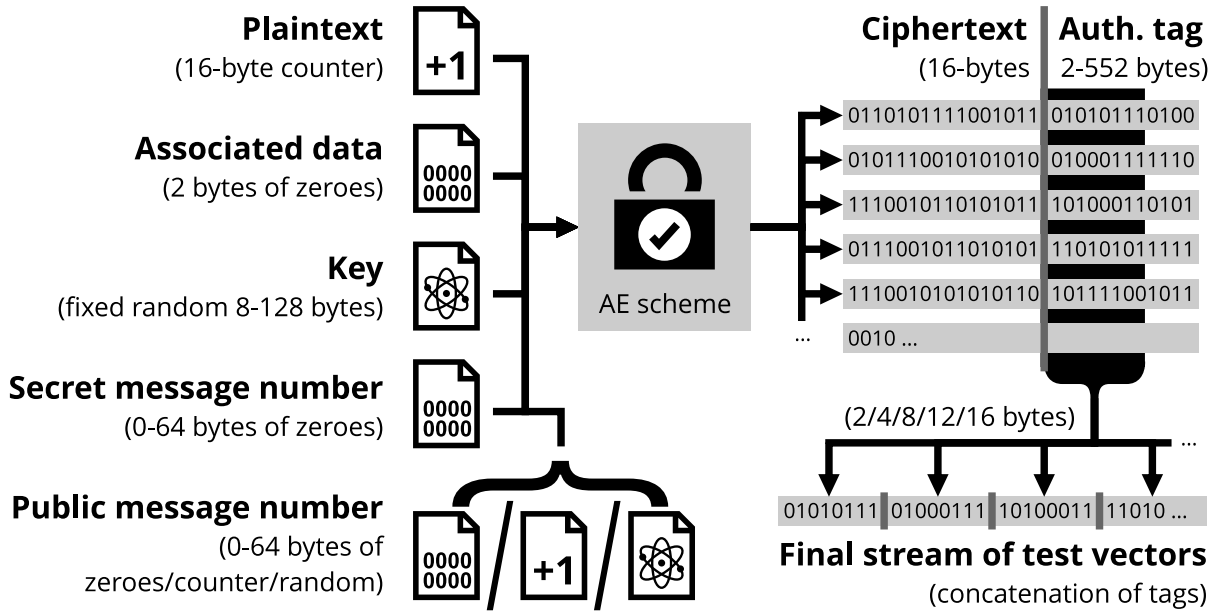

Figure 2: The process of creating the tested data streams by individual CAESAR candidates. The cipher is initialized as depicted in the diagram. The produced authentication tags are then concatenated to form a continuous stream suitable for randomness analysis.

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.

4 Tested data streams

The aim of the performed experiments is to assess randomness of authentication tags produced by many authenticated encryption schemes. The same analysis could also be performed on produced ciphertext, but that it out of scope of this paper. In particular, we inspect tags provided by CAESAR candidates in three independent scenarios differing in public message number setting. An overview of tag generation is given below and in Figure 2.

The cipher has 5 inputs: plaintext (encrypted and authenticated user input), associated data (authenticated user input), key, secret message number (secret nonce) and public message number (public nonce). The produced tag (extra ciphertext bytes when compared to the plaintext length) is determined by the cipher design. In the majority of the cases, this means 128 bits (16 bytes), but some candidates produce shorter tags (2, 4, 8 or 12 bytes). These tags were concatenated to form a continuous stream suitable for randomness assessment.

From the nature of the arguments prescribed by the CAESAR API, the 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 properties 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. The fields were initialized as follows:

- **Key**
  The key value was taken randomly but was fixed. For EACirc (one of the used tools), 1 000 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 cipher or the parameter set. Both fields' values were fixed to binary zeros. Note that only three ciphers used secret message numbers.
- **Public message number**
  This was the only parameter explored in different settings:
  - Fixed to a string of binary zeros for the whole time.
  - Increasing as a counter – each value unique but similar to others.
  - Having each value completely random.
- **Plaintext**
  The plaintext was 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).

In summary, if we denote the cipher as a function $F(plain, adata, key, smn, pmn)$ producing the authentication tag (the ciphertext is not used in our analysis, but inspecting it would also be interesting), the final analyzed stream in the scenario with random public message numbers looks as follows:

$$\begin{aligned}\text{Stream} = {} & F(0,0,rand_A,0,rand_1)\,||\,F(1,0,rand_A,0,rand_2)\,||\\ & F(2,0,rand_A,0,rand_3)\,||\,F(3,0,rand_A,0,rand_4)\,||\,\ldots\end{aligned}$$

5 Randomness testing tools

The most common way of testing randomness is using statistical testing. From the multitude of available batteries, we used the following three: 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 in 2015 by M. Sýs et al. [17].

For all experiments, we chose the significance level of $\alpha = 1\%$, which is the default value for NIST STS [14]. This keeps the type I. error (false positives) reasonably low while preventing the type II. error (false negatives) to reach too high values.

We used NIST STS version 2.1.1 with the default parameters (block lengths) for all tests. To comply with the minimal required stream length for individual tests [14], we tested 100 independent 1 000 000 bit long sequences for each candidate. In summary, NIST STS used about 12 MiB (about 700 000 tags) of data from each candidate for each test.

Dieharder version 3.31.1 was used. The two parametrizable tests were configured with recommended values. The length of the input stream processed by Dieharder varies from test to test. The humblest (Diehard 3D-sphere test) required about 48 kiB, while the greediest one (Bit distribution test) took about 9.2 MiB. To ensure the best possible comparability with the other test suites, we again analyzed 100 independent samples of the input. In summary, Dieharder tests used between 4.7 MiB (about 300 000 tags) and 916 MiB (about 60 000 000 tags) of input data for each candidate (depending on the test).

TestU01 was used in version 1.2.3. The most relevant sub-batteries are Rabbit, Alphabit and Block-Alphabit. These are intended for testing finite binary sequences. The length of the input stream taken by TestU01 can be set arbitrarily. To have an amount of data comparable to the other used batteries, we chose to process 2^{30} bits for each test. In summary, TestU01 thus used about 128 MiB (about 8 400 000 tags) of input data for each test.

EACirc represents a completely different approach to testing data randomness: 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 is represented as a hardware-like circuit consisting of simple interconnected functions. The used settings cause EACirc to process approximately 2.24 MiB of data produced by the tested cryptoprimitive for a single EACirc run. This amounts to about 2.24 GiB (about 150 000 000 tags) of data for a single experiment with 1 000 runs.

6 Results and interpretation

A selection of the numerical results can be seen in Table 1. The table aims for a representative selection of the interesting cases including all categories from the reference schemes to the algorithms that passed to the third (currently last) round. For the complete numerical results and detailed reasoning, see [20].

Firstly, let us 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 few bits 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.

Category (CAESAR round)	**Cipher** (official name)	**Candidate ID** (as used in SUPERCOP [21])	PMN fixed to zero				PMN counter-based				PMN truly random			
			EACirc (proportion)	**NIST STS** (x/188)	**Dieharder** (x/55)	**TestU01** (x/159)	**EACirc** (proportion)	**NIST STS** (x/188)	**Dieharder** (x/55)	**TestU01** (x/159)	**EACirc** (proportion)	**NIST STS** (x/188)	**Dieharder** (x/55)	**TestU01** (x/159)
Reference candidates	AES-GCM	aes128gcmv1	1.000	23	1	11	0.014	187	52	157	0.019	187	52	158
	AES-GCM	aes256gcmv1	1.000	71	1	13	0.013	188	51	156	0.007	188	54	158
Withdrawn candidates	Calico	calicov8	0.013	128	3	8	0.009	186	55	156	0.015	188	53	158
	Marble	aes128marble4rv1	0.016	160	16	6	0.010	168	14	8	0.010	160	16	6
First-round candidates	AES-CMCC	cmcc22v1	0.008	53	1	4	0.011	46	2	8	0.008	187	54	156
	AES-CMCC	cmcc24v1	0.005	43	3	3	0.008	188	50	155	0.023	186	50	152
	AES-CMCC	cmcc42v1	0.011	87	2	3	0.008	86	2	4	0.015	182	54	154
	AES-CMCC	cmcc44v1	0.008	85	4	5	0.013	183	52	155	0.007	188	53	152
	AES-CMCC	cmcc84v1	0.014	147	7	4	0.009	184	53	156	0.007	182	48	158
	Enchilada	enchilada128v1	1.000	71	2	15	0.017	187	53	157	0.010	186	52	155
	Enchilada	enchilada256v1	1.000	77	1	11	0.013	188	54	156	0.016	188	53	155
	Raviyoyla	raviyoylav1	1.000	22	2	7	1.000	148	28	24	0.295	186	51	144
Second-round candidates	TriviA-ck	trivia0v1	0.999	140	5	8	0.015	186	52	157	0.005	187	55	154
	TriviA-ck	trivia128v1	0.993	158	12	8	0.017	188	53	158	0.009	188	54	157
Third-round candidates	AEZ	aezv1	0.014	169	15	9	0.015	187	52	155	0.010	188	52	157
	AEZ	aezv3	0.016	164	13	6	0.011	188	53	157	0.009	185	50	156

Table 1: The selection of ciphers with interesting results from different categories (from reference schemes to 3rd round candidates). The numbers in columns of statistical batteries represent the number of passed tests (should be close to all for random stream), while EACirc displays the ratio of runs rejecting randomness (should be around 0.010 for random stream). For the ease of comprehension, if the result rejects randomness of the particular stream, the cell is gray-colored. Note that this is only a subset of the tested candidates (most of the omitted ones have results similar to those of *AEZ*). For complete numerical results and threshold values, see [20].

Secondly, let us inspect the results for the individual candidates (see Table 1). Tags of just five ciphers (*AES/GCM*, *Marble*, *AEC-CMCC*, *AES-CPFB*, *Raviyoyla*) were distinguishable from random streams with counter-valued public message numbers. Three of these ciphers (*Marble*, *AES-CMCC*, *AES-CPFB*) also failed in the random-valued scenario. The evidence is still too weak to deem the designs insecure – 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 ciphers 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.

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 [18], 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 behavior observed above (see Table 1): *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.

The results lead us to several interesting hypotheses requiring further inspection. The candidates 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 perspective 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.

7 Summary

We have set off to examine modern authenticated encryption systems from the point of resistance against common developer misconfiguration. In the end, we assessed outputs from 168 distinct schemes (all but six CAESAR submissions) in three different configurations using multiple software tools (NIST STS, Dieharder, TestU01 and EACirc).

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 three CAESAR submissions (*Marble*, *AEC-CMCC*, *Raviyoyla*) failed to produce seemingly random tags with counter-based public message numbers. For entirely random public message numbers, only *Marble* failed convincingly. *AEC-CMCC* achieved a borderline value in Dieharder and passed in other tools. *Raviyoyla* seems to have failed according to EACirc – this case is suspicious and worth of

further investigation, since it is the only case where EACirc surpassed the other tools. Importantly, none of these candidates made it to the second round of the competition (indirectly supporting our results).

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.

All in all, not even the state-of-the-art authenticated encryption designs do not have avalanche effect strong enough in the case with zero-fixed public message numbers. Although not forming a direct practical attack on the ciphers, it breaks semantic security of the scheme, since the attacker is able to distinguish two messages based on the leakage present in inspected scenarios. A security-optimist from the introduction may see this as an interesting area for further improvements.

Acknowledgments

We acknowledge the support of Czech Science Foundation, project GA16-08565S. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme Projects of Projects of Large Research, Development, and Innovations Infrastructures (CESNET LM2015042), is greatly appreciated.

References

[1] Farzaneh Abed, Christian Forler & Stefan Lucks (2014): *General Overview of the Authenticated Schemes for the First Round of the CAESAR Competition*. Cryptology ePrint Archive. Available at `http://ia.cr/2014/792`.

[2] Robin Ankele (2015): *Provable Security of Submissions to the CAESAR Cryptographic Competition*. Master thesis, Graz University of Technology. Available at `https://securewww.esat.kuleuven.be/cosic/publications/thesis-263.pdf`.

[3] Robert G. Brown (2004): *Dieharder: A Random Number Test Suite*. `http://www.phy.duke.edu/%7Ergb/General/dieharder.php`.

[4] CAESAR committee (2013): *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*. http://competitions.cr.yp.to/caesar-call.html.

[5] Kelsey Cairns & Graham Steel (2014): *Developer-resistant cryptography*. In: *A W3C/IAB workshop on Strengthening the Internet Against Pervasive Monitoring (STRINT)*.

[6] Julio Cesar Hernandez Castro, José María Sierra, Andre Seznec, Antonio Izquierdo & Arturo Ribagorda (2005): *The strict avalanche criterion randomness test*. Mathematics and Computers in Simulation 68(1), pp. 1–7, doi:10.1016/j.matcom.2004.09.001.

[7] Ali Doganaksoy, Baris Ege, Onur Koçak & Fatih Sulak (2010): *Statistical Analysis of Reduced Round Compression Functions of SHA-3 Second Round Candidates*. IACR Cryptology ePrint Archive. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.187.814&rep=rep1&type=pdf`.

[8] Kim Hakju & Kim Kwangjo (2014): *Who can survive in CAESAR competition at round-zero*. In: *The 31th Symposium on Cryptography and Information Security Kagoshima*, pp. 21–24. Available at `http://caislab.kaist.ac.kr/publication/paper_files/2014/SCIS2014_HJ.pdf`.

[9] Krister Sune Jakobsson (2014): *Theory, Methods and Tools for Statistical Testing of Pseudo and Quantum Random Number Generators*. Ph.D. thesis, Linköpings universitet, Sweden. Available at `http://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A740158&dswid=9282`.

[10] Tadayoshi Kohno, John Viega & Doug Whiting (2003): *The CWC authenticated encryption (associated data) mode*. *ePrint Archives*. Available at `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/cwc/cwc-spec.pdf`.

[11] Pierre L'Ecuyer & Richard Simard (2007): *TestU01: A C Library for Empirical Testing of Random Number Generators*. *ACM Transactions on Mathematical Software* 33(4), doi:10.1145/1268776.1268777.

[12] David McGrew & John Viega (2004): *The Galois/Counter Mode of Operation (GCM)*. *Submission to NIST*. Available at `http://siswg.net/docs/gcm_spec.pdf`.

[13] Mridul Nandi (2014): *Forging Attacks on Two Authenticated Encryption Schemes COBRA and POET*. In: *Advances in Cryptology – ASIACRYPT 2014*, 8873, Springer Berlin Heidelberg, pp. 126–140, doi:10.1007/978-3-662-45611-8_7.

[14] Andrew Rukhin et al. (2000): *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Technical Report, National Institute of Standards and Technology (NIST). Available at `http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf`.

[15] Markku-Juhani O. Saarinen (2015): *The BRUTUS automatic cryptanalytic framework*. *Journal of Cryptographic Engineering* 6(1), pp. 75–82, doi:10.1007/s13389-015-0114-1.

[16] Emil Simion (2015): *The Relevance of Statistical Tests in Cryptography*. *IEEE Security & Privacy*, pp. 66–70, doi:10.1109/MSP.2015.16.

[17] Marek Sýs, Zdenk Říha, Václav Matyáš, Kinga Márton & Alin Suciu (2015): *On the Interpretation of Results from the NIST Statistical Test Suite*. *Romanian Journal of Information Science and Technology* 18(1), pp. 18–32.

[18] Marek Sýs, Petr Švenda, Martin Ukrop & Vashek Matyáš (2014): *Constructing empirical tests of randomness*. In: *SECRYPT 2014 Proceedings of the 11th International Conference on Security and Cryptography*, SCITEPRESS Science and Technology Publications, pp. 229–237, doi:10.5220/0005023902290237.

[19] Meltem Sonmez Turan, Ali Doganaksoy & Çagdas Çalik (2008): *On Statistical Analysis of Synchronous Stream Ciphers*. Ph.D. thesis, The Middle East Technical University. Available at `http://etd.lib.metu.edu.tr/upload/12609581/index.pdf`.

[20] Martin Ukrop (2016): *Randomness analysis in authenticated encryption systems*. Master thesis, Faculty of Informatics, Masaryk University. Available at `http://is.muni.cz/th/374297/fi_m/`.

[21] Virtual Applications and Implementations Research Lab (2008): *SUPERCOP: System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives*. Available at `http://bench.cr.yp.to/supercop.html`.

[22] Petr Švenda, Martin Ukrop & Vashek Matyáš (2014): *Determining cryptographic distinguishers for eStream and SHA-3 candidate functions with evolutionary circuits*. In: *E-Business and Telecommunications*, 456, Springer Berlin Heidelberg, pp. 290–305, doi:10.1007/978-3-662-44788-8_17.

[23] A. F. Webster & S. E. Tavares (1986): *On the Design of S-Boxes*, pp. 523–534. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/3-540-39799-X_41.