THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 16, Special Issue 2015, pp. 385-394

TOWARDS A METHODOLOGY FOR RANDOMNESS ASSESSMENT: CHALLENGES AND PITFALLS

Kinga MARTON, Alin SUCIU

Technical University of Cluj Napoca, Computer Science Department, Romania
E-mail: kinga.marton@cs.utcluj.ro

Assessing the randomness of a (true) random number generator is not an easy task. Statistical testing is the most frequently used method, with several software tools being freely available. However, randomness assessment rises significant challenges that need to be overcome and pitfalls that need to be avoided. Unfortunately, a clearly defined and generally accepted methodology for randomness assessment based on statistical testing is not yet available. However, other methods also exist for randomness assessment which may efficiently complement statistical testing. The present paper aims at bringing a contribution to the elaboration of a general methodology for randomness assessment that takes advantage of all the methods available up to date.

Key words: random number generator, randomness assessment, methodology, statistical testing, visual testing.

1. INTRODUCTION

Random numbers are becoming more and more present in our daily lives, although they generally fulfil their essential roles in subtle ways, sometimes without even being noticed. For instance, every time we connect to a secure server on the Web we transparently generate random numbers, without necessarily being aware of that. A more explicit role of random numbers is played by transaction authentication numbers in financial transaction authorizations through online banking. The application domains of random numbers are numerous, and each domain states slightly different requirements on the quality of randomness, with information security and especially the domain of cryptography asserting the most exacting quality requirements.

Therefore the quality of random number generators is of crucial importance, thus it should be analyzed and assessed thoroughly and with great care, or as famously stated by Robert R. Coveyou: "The generation of random numbers is too important to be left to chance". In the last decades several famous applications faced spectacular failures, such as described in [1, 2] and more recently in [3, 4, 5], due to weaknesses in the design of the employed random number generators and hence became public lessons to consider in future designs.

Random number generators can be classified in two main categories: psudo-random number generators – PRNG (some authors call these deterministic or software generators) and true random number generators – TRNG (sometimes called physical or hardware generators). Table 1 presents a brief comparison between PRNGs and TRNGs. Furthermore, there are also intermediateclasses between these two extremes, such as the unpredictable random number generators which rely on the nondeterminism of user-computer interaction or on the complexity of the underlying phenomenon while executing a set of deterministic instructions, or the hybrid generators which combine both TRNGs and PRNGs.

PRNGs are clearly mathematical artefacts and their analysis and assessment is left to the mathematical methods that underlie those generators. For instance, the famous Mersenne Twister PRNG [6] is mathematically proven to be uniform in 623 dimensions and to have a period of $2^{19937}-1$.

Here we will focus on the assessment of generators from the second category, namely TRNGs, which extract randomness by sampling and digitizing natural physical phenomena (like thermal noise, jitter,

radiation decay timings, photon polarization, etc.) and provide the purest form of true entropy, where the unpredictability of the generated values is guaranteed by physical laws. These generators have strange and counterintuitive properties. For instance they do not have a period (which would imply repetition of their output after some time) like PRNGs do. Yet, while TRNGs offer the highest level of nondeterminism and irreproducibility they do not necessarily present perfectly uniform distribution and independence, hence their output needs to be filtered (post processed) in order to reduce possible bias (tendency towards a particular value) or correlation, and make the output more similar to a statistically perfect random sequence.

Table 1

Comparative characteristics of pseudo- and true random number generators

	PRNG	**TRNG**
Principle of operation	Mathematical formulas, algorithms, no entropy	Physical devices, quantum based entropy
Deterministic	Yes	No
Predictable	Yes	No
Periodic	Yes	No
Main field of usage	Simulation, Cryptography (CSPRNGs)	Security/Cryptography, Gambling
Output type	Stream of numbers (int, float)	Stream of bits (boolean)
Speed (output bit rate)	Very fast (aprox. GB/sec)	Slow (KB/sec) / Fast (several MB/sec)

There are currently many vendors and designers of TRNG devices, based on various physical phenomena, from digital clock jitter to quantum particles (photons) and even digital cameras. All these producers make very strong claims about their generators, such as truly quantum, pass any statistical tests, pass any properly designed test for randomness, etc. However, when it comes to providing evidence to support those claims, most producers complain about the lack of certification bodies in the field of true random number generator assessment, and usually perform some in-house testing procedure by applying some statistical tests on very few input sequences, showing the good results - all test passed. However, being physical (hardware) devices subject to fluctuations and aging of the components, these generators may suffer technical problems due to their practical implementation, and may present vulnerabilities which may allow external sources (such as environmental conditions like extreme temperatures) to interfere or alter the generated numbers therefore a constant monitoring is required. Some suppliers have already implemented this feature and are constantly monitoring the output with simple startup and online statistical tests [7].

The assessment of these true random number generators is not a trivial task, and this is the reason why TRNGs are the only cryptographic primitives for which standardization is still missing. In this context, our main goal in this paper is to present the main challenges that we face and the pitfalls that we should avoid in the process of TRNG testing. While we do not have a definitive answer to some of these challenges, we will try to give recommendations and hints based both on theoretical and empirical observations coming from our and others' experience in this field.

2. RANDOMNESS ASSESSMENT

The biggest challenge in assessing the randomness is being able to answer a series of very difficult questions, such as:

- How good is a certain RNG? Bad/ Good/ Very good/ Excellent/Perfect
- Does it have flaws (that compromise the randomness of its output sequences)?

Ideally we should have a metric for randomness, and a methodology for assessing the quality of a RNG, which could lead to answering the above questions. Unfortunately there is no practical method able to determine or prove whether a bit sequence IS random.

However, there are theoretically an infinite number of methods which can identify flaws in a certain random bit sequence. For instance, the sequence should contain approximately the same number of zeroes and ones. If it doesn't (consistently) then we have found a flaw in the generator.

Thus, since we cannot (yet) get a definitive answer, the only way to go is to try finding flaws in the generated sequences using as many relevant methods as we can. If after a certain amount of testing no flaws are found, then our confidence in that generator increases. The more thorough the testing the better. Testing should never end, hence the generators should be constantly or systematically tested for there is always a chance that one of their physical components will fail at some point and then the generator can become considerably or completely biased.

Of course the central means we have to assess a TRNG is by analyzing its output sequences, but we must keep in mind where they come from, we cannot just make abstraction of the generator, the entropy source also needs to be carefully analyzed.

Pitfall: Loosing focus on what we are assessing. A common pitfall here is to lose focus, and to get distracted by a particular sequence being analyzed. One must constantly keep in mind that we are not assessing particular bit sequences, but rather we are assessing generators. Otherwise, the decimals of PI would be quite an attractive sequence to analyze and will pass the majority of statistical tests, yet PI is not a random number generator.

Pitfall: Analyzing too few sequences. Another common pitfall is to stop testing too soon, after analyzing just a few random number sequences produced by the generator. How many sequences should be analyzed is a complex decision and clearly depends on the generator that is subject to analysis, but there are general recommendations as well. NIST recommends the analysis of a number of sequences which are at least on the order of the inverse of the significance level. Hence, if the significance level ? is set to 0.01, than at least 100 distinct bit sequences generated by a certain generator have to be analyzed in order to be able to reliably capture the quality of that generator. The analysis should be repeated at least several times for a different set of 100 sequences, having a different length (same length per set of sequences, though).

Pitfall: Analyzing too short or too long sequences. Most statistical tests impose a minimal size for the input bit sequence being analyzed and constraints are set on a test-by-test basis. For instance several tests from the NIST STS [8] (such as the Frequency test, Cumulative sum test, Runs test) require at least 100 bits, and other test impose a much higher minimal input size for producing reliable results (eg. the Matrix rank test requires at least 38912 bits, the Random excursion, Serial and Linear complexity test require at least 1.000.000 bits), furthermore test can be parameterized by additional values (eg. Block size, template) which also have to be set to meaningful values.

Table 2

Characteristics of the most popular statistical test suites

	NIST STS	**TestU01/Rabbit**	**Diehard**	**ENT**
Author	A. Rukhin et al.	P. LEcuyer and R. Simard	G. Marsaglia	J. Walker
Last update	2014	2009	1996	2008
No. of tests	15	38	15	5
Suggestion of a methodology	yes	no	yes	no
Library/Application	Application	Library	Application	Application
User interface	Text based	none	Text based	Text based
Input sequences	many	one	one (10 MB)	one
P-value based	yes	yes	yes	no
Alpha	0.01	0.001	0.01	-
Proportion of tests passed	yes	no	no	no
Uniformity of p-values per test	yes	no	no	no
Uniformity of p-values overall	no	no	no	no

Clearly there could be a tendency to analyze short sequences to save time, which would be an inadvisable practice, that could lead to misguided results. We suggest that input sequences should be in the interval of 1 MB – 1 GB, and should include sequences of various sizes. The opposite tendency of focusing only on large and very large sequence could lead to loosing focus on smaller sequences that show real defects in the generator. The large sequence could pass the tests but may have subsequences that systematically fail the tests and indicate problems in the generator.

3. METHODS FOR RANDOMNESS ASSESSMENT

The most frequently used method for assessing the randomness of a generator is statistical testing. Conforming to this method the output of the generator, a bit sequence, is subjected to statistical analysis, according to a certain statistical test, in order to verify that the sequence is sound statistically. As a result, the sequence will PASS or FAIL the test.

Obviously one test is not enough so batteries of tests were developed, the most well known being NIST STS [8], Diehard[1] [9], TestU01 [10] and ENT [11].

Table 2 summarizes the main characteristics of the four most popular statistical test suites for randomness assessment.

It is interesting to notice that in theory there could be an infinite number of ways that a sequence may fail statistical testing so there could be an infinite number of statistical tests to apply.

A huge challenge here is to define a battery of statistical tests, because each test takes time to execute and time is a precious commodity. Therefore the selection of tests is critical and has to focus both on assessing independent probabilistic randomness properties in order to provide multiple view points and a large span on the domain, and at the same time on thoroughly evaluating each tested property in order to create a more comprehensive testing process.

For example the authors of TestU01 [10] chose to define a number of parameterized tests and leave it to the users to define their own battery based on these tests (several predefined batteries are provided as well.

Challenge: Defining your own battery of statistical tests.

Pitfall: The user in general does not have the knowledge required to define such a battery. Unless the user is very knowledgeable in the field, we recommend the use of already defined batteries.

Challenge: Selecting among available batteries of tests. Each available battery has its advantages and disadvantages, as shown in Table 2. However, out of the four available batteries we would discard ENT (in its present form) because the integrated five tests although very straightforward, provide just a small coverage on the domain, and furthermore the tests are not based on computing p-values, which make the results difficult to interpret. Also due to the lack of maintenance, poor implementation (translated from Fortran to C), limitation of the input stream, we would not recommend the use of the Diehard test suite either, which is geared towards testing uniform numbers rather than bit sequences.

The selected test suite has to be widely recognized and used in the research community and industry in order to benefit from the research results on the independency and span of the included statistical tests, and at the same time facilitate the comparison between generators.

In our opinion NIST STS is the clear winner here, firstly because of its widespread use which makes the suite a "de facto" standard in the process of randomness assessment, and secondly due to its highly rigorous requirements, large number of independent tests providing a large span on the domain, flexibility of user input and most importantly, because it is the only battery that gives good integration of the results. TestU01 could also be used, as it is increasingly popular, but this involves developing an application for it, with a reasonable user interface, flexibility of input and integration of the results.

Challenge: The limits of statistical testing. The main limit of statistical testing is its intrinsic PASS/FAIL nature. No additional information is provided. In particular it should be very interesting to find out WHY did the analyzed sequence FAIL, in order perhaps to correct the design of the generator.

[1] An updated version called Dieharder was developed in recent years by R.G. Brown at Duke University, but still needs more scrutiny by the scientific community until final acceptance. All tests from the original Diehard were reimplemented, but the author admits some are still unreliable or give suspect results.

Even the simplest statistical test, the frequency (monobit) test, will only tell us that the ratio between the zeroes and ones is statistically wrong. But why? Did we get too many zeroes, or too many ones? Where is the generator biased, towards zero or towards one?

An interesting example from our experience was analyzing the output of a quantum based TRNG that consistently failed statistical tests, but no one knew why. So we resorted to visual inspection of the analyzed sequences, see Fig. 1 depicting the colour image representation of a sequence, and we discovered that it was a defect in the generator that was periodically introducing bias toward zeroes (water ripple-like darker patterns). Based on these simple observations derived from visual inspection of the sequences the authors of the generator were able to correct this behaviour.

Another limit of statistical testing comes from its intrinsic nondeterminism. In other words, even perfect random generators will sometimes generate sequences which contain subsequences that look deterministic and hence may fail some of the tests. If the significance level α is chosen as 0.01 (as in NIST STS) then approximately one sequence in 100 sequences generated by a perfect random number generator will fail the tests.

At the same time nonrandom sequences may contain subsequences which look random and consequently pass the majority of tests, yet the sequence as a whole shows repeating patterns or other traces of lack of randomness. Therefore the sequence to be tested has to be long enough to allow the evaluation process to arrive to the correct conclusion.

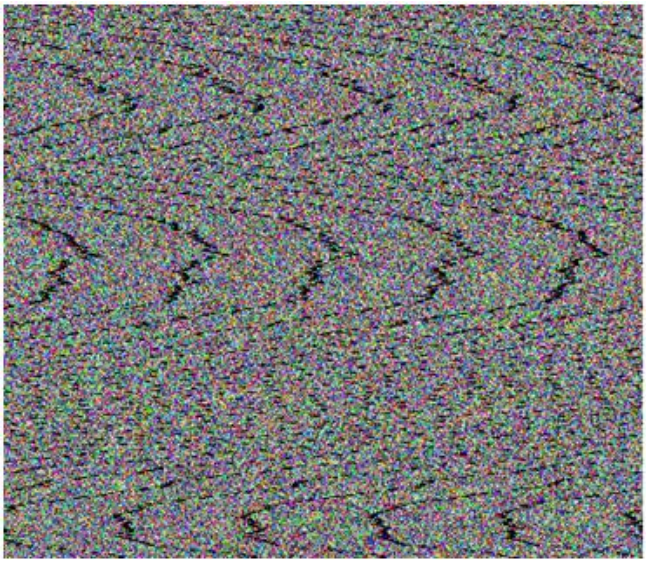

Fig. 1 – Color image representation of a sequence produced by a biased quantum random number generator.

Hence the need to perform many tests on many sequences in order to make sure that detected failures are not simply due to chance in the sense mentioned above, and we should expect a certain number of failures. There is no generator that will pass all statistical tests, although some TRNG producers state this claim, for the very reason mentioned above.

The authors of [12] propose an adaptive testing method in order to identify failures that appear merely by chance. They adaptively modify existing statistical tests so that each time a suspicious p-value (that is very close but under the significance level) is detected, the sample size is automatically increased and the test rerun. The procedure stops when the p-value stabilizes in the acceptable range or in a clearly rejectable range.

Another limit of statistical testing comes from parameterization. As we have already mentioned above, certain tests are very sensitive to input parameters like the sample length, block length or the considered template. But there are other fundamental parameters such as the significance level α, that need to be chosen wisely. If too much freedom is given to users, they may find themselves in danger of wasting valuable computing resources and perform the assessment with irrelevant or perhaps even incorrect tests.

Another limit of statistical testing is ignoring the uniform distribution of p-values requirement. Most users are content to perform several statistical tests and to see that their sequence(s) pass most of those tests. However, the obtained p-values must also be uniformly distributed. The only battery that gives insight into this problem is the NIST STS.

Given the serious limitations of statistical testing mentioned above, it becomes obvious that different methods should also be used when assessing the randomness of a generator.

One of these alternate methods that we already mentioned here, and proved to be very valuable, is visual inspection.

The human visual system is highly trained to extract features of the surrounding world and summarize these with statistical descriptors. Yet, the perceptual evaluation of random number sequences using the human visual system, in a way similar with every other statistical randomness tests, does not provide a method for proving randomness. Instead it takes advantage of our perceptual system to quickly spot tracks of predictability or non-randomness in the sequence and facilitates the understanding of randomness and lack of randomness.

Nevertheless, by representing a sequence of numbers graphically, the human visual system is only capable of determining the degree to which the representation satisfies visual randomness but is unable to tell the difference between real randomness and visual paternlessness. In this context, visual analysis is not to be used exclusively, but rather as an efficient component of a larger randomness testing system which also includes powerful statistical test suites and other approaches to randomness evaluation.

Another alternative method could derive from the work of Kolmogorov, Martin Loef and Chaitin in the field of Algorithmic Information Theory (AIT) [13], and boils down to the property of incompressibility of random sequences regardless of the lossless compression algorithm used. In theory a truly random sequence should be incompressible, and an empirical study that we conducted using several lossless compression algorithms including commercially available tools show promising results. Still, unfortunately we are still a long way ahead from having a ready to use tool based on these theoretical and empirical results.

Pitfall: Using only one method is not enough. We believe that one should use all the available methods for a correct assessment of a random number generator, and we have seen in practice how these methods can complement each other. Therefore we recommend the combined use of as many methods as possible, with the statistical testing being the backbone of the testing methodology.

4. TOOLS FOR RANDOMNESS ASSESSMENT

Challenge: choosing the right tools. Software tools are used in general to assess the randomness of a generator, although we are aware of attempts to implement statistical tests in hardware (FPGA) [14]. Most of these software tools are based on statistical testing using batteries of statistical tests. However, here we are faced with a new and significant challenge regarding the choice of these tools.

Some of these tools may be extremely simple and use non-standard evaluation techniques (lack of p-values computation ENT). Others may be obsolescent, due to the lack of further development, like Diehard.

Pitfall: Using only one tool is not enough. A common pitfall here would be to choose a single tool and ignore all the others. As we already mentioned there are an infinite number of possible statistical tests, so the larger the number of tests the better. This clearly invites to the use of several software tools, instead of focusing excessively on a single one. There isnt any highly superior tool that would justify such an exclusive focus.

We recommend the use of the two prominent statistical testing batteries NIST STS and TestU01/Rabbit, plus a visualization tool like FileSeer [15, 16]. Unfortunately, as far as we know, no tools are available for Kolmogorov complexity (incompressibility) based testing.

Challenge: Implement the tools efficiently. As we mentioned above, the recommended tool for statistical analysis is NIST STS. One major problem with this however, is the lack of efficiency of the original NIST STS implementation. It looks like the authors did not give enough importance to this aspect, as shown in [17, 18, 19] and more recently in [20]. TestU01 on the other hand, does not seem to suffer from this point of view.

Another major problem here is the lack of parallel implementations for these tests, although we now have the hardware resources at hand to run them in parallel. Moreover, as shown in [18][19] both NIST STS tests and TestU01 tests are largely parallelizable, so we believe that more efforts should be done in this direction. Because it is a well known fact that statistical tests are compute-intensive, and therefore it matters if one can completely test a 1GB sequence in one week or just a few hours.

Challenge: Find tools with user friendly GUI. Ideally, one would like to use tools that give easy to understand results, tools that help the user to interpret the results, which is best done with a GUI. For instance, it is a huge difference between reading several pages of p-values (and detecting the ones indicating failure) or taking a look at a picture that represents all these p-values in the unit interval, as in Fig. 2 below.

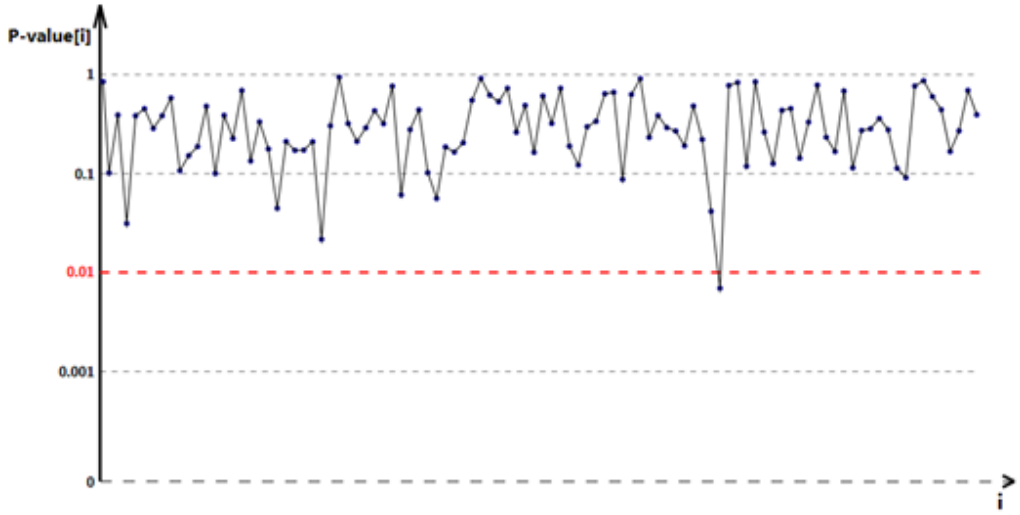

Fig. 2 – Graphical representation of the p-values obtained with the GUI developed for NIST STS [17].

Unfortunately, as Table 2 indicates, none of the available statistical testing batteries offer an easy to use GUI, so we are left with the burden of building one ourselves. As shown by previous work in [17, 18] this looks like a promising direction to follow, which greatly enhances our understanding and interpretation of the results.

5. THE METHODOLOGY

Challenge: Establishing a clearly defined and generally accepted methodology for randomness assessment. While there are efforts to establish such a methodology, like the german standard AIS31 [21] (based on statistical testing), we believe there is still much work to do until a generally accepted methodology, combining several methods, not just statistical testing, will be established. Some methods are very difficult if not impossible to formalize like visual inspection for example, which make this task even harder.

Challenge: Integrating the results. Even for a single method, the statistical testing, this is still an open question. The only tool that gives reasonable integration of results is the NIST STS, but only per each test, and not globally. Two metrics are used for this integration.

The first one is the proportion of passing a specific test, where given a certain number of input sequences tested (m), the threshold values for the proportion of tests passed are given by Eq. (1).

$$T-value = (1-\alpha) \pm 3\sqrt{\frac{\alpha(1-\alpha)}{m}}, \quad (1)$$

where the significance level is α=0.01. For example if 1000 bit sequences were tested the *T–value* is 0.98056072, which means that 981 of these sequences must PASS the test, otherwise the generator is considered flawed.

The second one is the distribution pattern of p-values, where we must also make sure that the p-values obtained after testing several input sequences are uniformly distributed over the interval $[0,1]$. NIST STS implements a simplified version of this metric, where the p-values are distributed in 10 equal ranges $([0,0.1),...,[0.9,1])$ and an overall p-value (POP p-value of p-values) is computed to verify the uniformity of this distribution of individual p-values using Eq. (2), where F_i is the number of p-values in sub interval i and s is the sample size.

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - \frac{s}{10})^2}{\frac{s}{10}}. \quad (2)$$

Both these metrics are used for integrating the results for each of the 15 tests of the NIST STS; however, no attempt is made to integrate results from different tests, as this would rise the difficult question: are all tests equally significant (or powerful)?

The statistical tests integrated in NIST STS may be grouped together according to their focus. In [22] the authors consider four test categories, namely Frequency Tests (NIST tests 1-4), Tests for Repetitive Patterns (NIST tests 5-6), Tests for Pattern Matching (NIST tests 7-12) and Tests based on Random Walk (NIST tests 13-15). Perhaps an integration of the tests from the same category would be the next step, but this still leaves the open question of how to integrate tests globally.

Another important aspect here (in NIST STS) relates to the number of p-values generated by each test; most tests generate one single p-value, but others generate 2, 8, 18 or even more p-values.

So how do we integrate a test that generates one p-value with a test that generates 18 p-values?

The interpretation given by NIST (and we agree with that) is that one test equals one p-value; therefore if one test generates 8 p-values this actually means that we have 8 (sub)tests, and each of them has its own line in the final report (see Fig. 3).

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
11	8	9	8	10	8	14	11	14	7	0.779188	98/100	Frequency
8	14	11	7	10	7	12	10	10	11	0.883171	99/100	BlockFrequency
8	5	12	9	13	12	11	8	12	10	0.779188	100/100	Runs
13	15	10	7	6	14	6	9	11	9	0.401199	97/100	LongestRun
5	8	12	8	13	14	12	11	8	9	0.616305	100/100	Rank
16	14	8	8	12	9	6	11	9	7	0.419021	97/100	FFT
10	8	10	16	4	12	12	7	11	10	0.401199	98/100	NonOverlappingTemplate
												...
5	11	16	17	9	13	5	9	9	6	0.058984	100/100	NonOverlappingTemplate
9	10	14	10	12	6	16	11	9	3	0.191687	99/100	OverlappingTemplate
15	8	7	7	12	7	13	10	13	8	0.514124	99/100	Universal
9	16	10	11	11	6	15	13	3	6	0.080519	97/100	LinearComplexity
17	11	7	8	8	13	11	7	8	10	0.437274	98/100	Serial
10	7	13	6	13	11	10	9	8	13	0.759756	100/100	Serial
15	7	13	8	9	9	10	14	7	8	0.554420	99/100	ApproximateEntropy
10	8	9	5	13	8	15	11	7	14	0.401199	100/100	CumulativeSums
11	9	9	11	9	12	9	8	12	10	0.994250	98/100	CumulativeSums
6	8	8	3	5	8	5	5	5	7	0.888137	59/60	RandomExcursions
7	2	9	6	3	5	7	7	6	8	0.637119	58/60	RandomExcursions
5	6	6	6	5	8	4	7	5	8	0.976060	58/60	RandomExcursions
4	6	11	4	2	6	7	8	7	5	0.407091	60/60	RandomExcursions
7	5	8	5	5	5	6	4	8	7	0.964295	60/60	RandomExcursions
7	4	8	4	11	5	3	5	8	5	0.437274	59/60	RandomExcursions
8	4	11	5	2	9	2	10	5	4	0.066882	59/60	RandomExcursions
3	7	7	3	5	12	6	6	6	5	0.378138	60/60	RandomExcursions
2	10	3	7	2	6	7	4	9	10	0.100508	60/60	RandomExcursionsVariant
												...
6	9	4	5	7	5	4	6	9	5	0.834308	60/60	RandomExcursionsVariant

Fig. 3 – Except of a NIST STS final report.

Furthermore, the non-overlapping template matching test is performed for each aperiodical template of m bits (recommended values for m are 9 or 10), hence for $m = 9$ the test results in 148 p-values, more p-values than all the other tests put together. Is this test more significant than the others? Is it fair to give this test such a large proportion of the final results? These questions need careful investigation.

It is clear that many sequences obtained from a certain generator should be tested statistically to obtain a significant result about the generator. It is less clear how long should each sequence be, although each statistical test specifies a minimal length for its input. However, it is also clear that integrating the results in the NIST STS manner should only be done for sequences of the same length. So there is still an open question regarding the integration of results (p-values) for sequences of different lengths.

Challenge: Are some tests more relevant than others? There is good indication that the answer to this question is "yes", as it transpires from the NIST STS implementation. For example the Frequency (monobit) test is considered more relevant in the sense that it is carried out as a prerequisite for Runs test.

However, establishing a hierarchy among tests is a very difficult task, all approaches in this direction should be thoroughly analyzed. The authors of [7] propose a prioritisation of the NIST tests, defining three tiers of tests. They claim that the tests included in the first tier capture the most relevant types of nonrandomness, tier 2 includes three additional, more computational intensive tests and finally tier three includes all the NIST tests.

6. CONCLUSIONS

In this paper we have shown some major challenges faced during the evaluation of TRNGs and we also presented some common pitfalls to be avoided along this difficult process.

While we do not offer a final answer regarding a definitive methodology for randomness assessment, we believe we answered several crucial questions that we hope will ultimately lead to a generally accepted evaluation methodology for randomness assessment.

We know that we do not have a method to obtain a definitive answer regarding the randomness of a specific (true) random number generator, but we also know that there are methods that might help us in assessing the randomness of a generator, most notably the statistical testing method, which should be accompanied by other methods like visual inspection and Kolmogorov complexity based methods.

We also reviewed the available tools for statistical testing and as a result we recommend the use of NIST STS as the main tool, with TestU01/Rabbit as the secondary tool (needs additional implementation effort). It is also clear that some statistical tests are more important/relevant than others but a concrete hierarchy is difficult to establish yet. However we can say some tests are prerequisites to other tests, such as the Frequency test, which should be performed first, because if a generator consistently fails this test, no further testing is necessary, and the generator should be rejected.

Beyond the multitude of statistical tests to be performed on a wide range of input sequences, an essential problem is that of finding metrics to integrate the results of these statistical tests (p-values). So far two such metrics were implemented in a single statistical testing suite, namely NIST STS. However, other metrics were recently proposed [23] that need to be implemented and combined with the existing ones.

For significant testing results, one should apply statistical tests to many input sequences obtained from the generator under evaluation, that is at least 100 sequences for NIST STS. In general the number of sequences should be at least the inverse of the significance level alpha, which means that for TestU01/Rabbit battery one should test al least 1000 sequences. We recommend the testing of at least 1TB of continuous output of the generator, split into nonoverlapping sequences of the selected size. The selected size for one sequence could vary but it should be at least 1.000.000 bits for NIST STS, as some tests impose this minimum length for their input. However, we recommend the testing of larger sequences as well, in the 10 MB - 100 MB range.

Finally it should be said that while statistical testing benefits from several well designed batteries of tests, the other approaches are still very deficitary; the only tools for visual inspection that we know are FileSeer [15] and FileSeer+[16], and we do not know of any tools based on the Kolmogorov complexity method.

Based on the analysis of various challenges and pitfalls mentioned in the present paper, we aim at developing a general methodology for randomness assessment of true random number generators, as the essential further development of this research.

ACKNOWLEDGMENTS

This paper was supported by the Post-Doctoral Programme POSDRU /159/1.5/S/137516, project co-funded from European Social Fund through the Human Resources Sectorial Operational Program 2007-2013.

7. REFERENCES

1. *DSA-1571-1 openssl -- predictable random number generator*, Debian Security Advisory. 13 May 2008.
2. I. GOLDBERG, D. WAGNER, *Randomness and the Netscape Browser*, Dr. Dobbs Journal, pp. 66-70, January 1996.
3. V. BUTERIN, *Critical Vulnerability Found In Android Wallets*, [Web log post]. Retrieved May 25, 2015, from https://bitcoinmagazine.com/6251/critical-vulnerability-found-in-android-wallets/
4. D. GOODIN, *Google confirms critical Android crypto flaw used in $5,700 Bitcoin heist*, [Web log post]. Retrieved May 25, 2015, from http://arstechnica.com/security/2013/08/google-confirms-critical-android-crypto-flaw-used-in-5700-bitcoin-heist/
5. D. GOODIN, *Crypto flaws in Blockchain Android app sent bitcoins to the wrong address*, [Web log post]. Retrieved May 29, 2015, from http://arstechnica.com/security/2015/05/crypto-flaws-in-blockchain-android-app-sent-bitcoins-to-the-wrong-address/
6. M. MATSUMOTO, T. NISHIMURA, *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation **8**, *1*, pp: 330, 1998.
7. C. KENNY, *Random Number Generators – An Evaluation and Comparison of Random.org and Some Commonly Used Generators*, The Distributed System Group, Trinity College Dublin, April 2005.
8. A. RUKHIN et al., *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, NIST Special Publication 800-22 (with revisions dated April, 2014).
9. G. MARSAGLIA, *DIEHARD*: *A battery of tests of randomness*. http://www.stat.fsu.edu/pub/diehard/, 1996.
10. P. LECUYER, R. SIMARD, *TestU01: A C library for empirical testing of random number generators*, ACM Trans. Math. Softw, **33**, *4*, article 22, 2007.
11. J. WALKER, *ENT – A pseudorandom number sequence test program*, Fourmilab, http://www.fourmilab.ch, 2008.
12. H. HARAMOTO, *Automation of Statistical Tests on Randomness to Obtain Clearer Conclusion*, Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer, pp. 411-421, 2009.
13. M. Lİ, P. M. B. VİTNYİ, *An Introduction to Kolmogorov Complexity and its Applications*, Springer, 3rd ed., 2008.
14. V. SURESH, D. ANTONIOLI, W. BURLESON, *On-chip lightweight implementation of reduced NIST randomness test suite*, IEEE Intl. Symposium on Hardware-Oriented Security and Trust, pp. 93-98, 2013.
15. K. MARTON, I. NAGY, A. SUCIU, *Visual Inspection of Random Number Sequences with FileSeer*, Automation, Computers, Applied Mathematics ACAM, **19**, *1*, pp. 3-10, 2010.
16. K. MARTON, D. PATRASCU, A. SUCIU, *Perceptual Evaluation of Random Number Sequences using FileSeer+*, Studia Universitatis Babes-Bolyai - Series Informatica, **LX**, *1*, pp. 98-110, 2015.
17. A. SUCIU, K. MARTON, I. NAGY, I. PINCA, *Byte-oriented Efficient Implementation of the NIST Statistical Test Suite*, Proc. of 2010 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, Romania, May 28–30, 2010.
18. A. SUCIU, I. NAGY, K. MARTON, I. PINCA, *Parallel Implementation of the NIST Statistical Test Suite*, Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2010), Cluj-Napoca, pp. 363-368, 2010.
19. A. SUCIU, R. TOMA, K. MARTON, *Parallel Object-Oriented Implementation of the TestU01 Statistical Test Suites*, in Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2014), Cluj-Napoca, pp. 311-315, 2014.
20. M. SYS, Z. RIHA, *Faster Randomness Testing with the NIST Statistical Test Suite*, Security, Privacy, and Applied Cryptography Engineering, Lecture Notes in Computer Science, **8804**, pp. 272-284, 2014.
21. W. KILLMANN, W. SCHINDLER, *AIS 31: Functionality classes and evaluation methodology for true (physical) random number generators*, Version 3.1, Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn, 2001.
22. J. K. M. SADIQUE, U. ZAMAN, R. GHOSH, *Review on fifteen Statistical Tests proposed by NIST*, Journal of Theoretical Physics and Cryptography, **1**, pp. 18-31, 2012.
23. A. OPRINA, E. POPESCU, G. SIMION, *Walsh-Hadamard randomness test and new methods of test results integration*, Bulletin of Transilvania University of Brasov, **2**, *51*, pp. 93-106, 2009.