Bachelor of Security in Computer Science
June 2019


# Promestra Security compared with other random number generators

**Robin Ringsell**
**Andreas Korsbakke**

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor of Security in Computer Science. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

**Contact Information:**
Author(s):
Robin Ringsell
E-mail: rori16@student.bth.se

Andreas Korsbakke
E-mail: ankk16@student.bth.se

University advisor:
Lecturer Anders Carlsson
Department of Computer Science and Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

# Abstract

**Background.** Being able to trust cryptographic algorithms is a crucial part in the society today, because of all the information that is gathered by companies all over the world. With this thesis we want to help both Promestra AB and potential future customers to evaluate if you can trust their random number generator.
**Objectives.** The main objective for the study is to compare the random number generator in Promestra security with the help of the test suite made by the National Institute of Standards and Technology. The comparison will be made with other random number generators such as Mersenne Twister,Blum-Blum-Schub and more.
**Methods.** The selected method in this study was to gather a total of 100 million bits of each random number generator and use these in the National Institute of Standards and Technology test suite for 100 tests to get a fair evaluation of the algorithms. The test suite provides a statistical summary which was then analyzed.
**Results.** The results shows how many iteration out of 100 that have passed and also the distribution between the results. The obtained results show that there are some random number generators that have been tested that clearly struggles in many of the tests. It also shows that half of the tested generators passed all of the tests.
**Conclusions.** Promestra security and Blum-Blum-Schub is close to passing all the tests, but in the end, they cannot be considered to be the preferable random number generator. The five that passed and seem to have no clear limitations are: Random.org, Micali-Schnorr, Linear-Congruential, CryptGenRandom and Mersenne Twister.

**Keywords:** Randomness, Cryptography, RNG, NIST

# Acknowledgments

We would like to thank our supervisor Anders Carlsson and also Robert Nyqvist for their help with our thesis. A special thanks to Promestra AB for the given opportunity to test their random number generator.

# Contents

# Chapter 1

# Introduction

We live in a society where information plays a huge role for both companies and for individuals. As individuals we don't want our personal information leaked, stolen or seen by people whom should not be trusted or have access to it. We put our trust in many organizations to keep our data inaccessible to people who shouldn't have access[22]. This information could be anything from usernames and passwords to payment details or social security numbers. Companies can use this information in multiple ways, for example, personalized advertisements that could help to increase sales. This means that it is critical that companies can guarantee that personal information is stored in a secure way. That leads us to the question: How does corporations ensure that your personal information is safe? One of the most common solutions to that is using cryptography to obfuscate the information.

Cryptography has been around for a very long time. One of the earliest signs of usage goes back to Julius Caesar time where it is said that he used what is now referred to as Caesar cipher[1]. The Caesar cipher is a type of a substitution cipher where letters are replaced by other letters determined by a specific "shift". Caesar used 3 for example but there are a total of 26 possible shifts in the English alphabet. These types of ciphers are easily decrypted with the use of today's knowledge about cryptography. The encryption algorithms used today are much more complex and sophisticated than just switching letters. Therefore there is a need of a random source to prevent the ability of predicting the previous or upcoming state in the encrypted text. This leads us to the random number generator(RNG), it provides the algorithms with a random factor that should be impossible to predict. Since computers are machines that are supposed to follow given instructions by the user, you have to give it instructions on how to randomize numbers. If you have to give it instructions it can obviously not be considered random. The steps to avoid this problem today, involves computers monitoring unpredictable sources such as atmospheric noise or radioactive decay. Another possible source could be the computers CPU activity or keystroke intervals, these however, cannot be seen as good choices if only one is used. They could be used in combination to create one "random source", the more sources used, the higher entropy.

In order to give a fair and an unbiased assessment and to review if an algorithm can be considered safe, there are multiple tests that are available for different types of cases. This study will use a statistical test suite for random number generators that is created by the National Institute of Standards and Technology(NIST)[19].

In this thesis we will give information about Promestra Security, as there is no official information about Promestra Security, any information we state is from those working at Promestra AB. Promestra security(PS) is a security concept that is being presented as a 40 times faster encryption algorithm than Advanced Encryption Standard(AES). Promestra AB want to test their algorithms in different ways in order to be able to promote their product and for it to see usage in real applications and systems. Promestra has not released this cryptographic publicly and it is not publicly used anywhere, therefore there is not much known about the cryptosystem as a whole nor how they generate random numbers and how well it works.

## 1.1 Purpose

The purpose is to help Promestra AB to get a valid assessment of the random number generator(RNG) in their security concept. By using NISTs framework [19], the study will provide a non-bias comparison between ten random number generators. By performing a statistical analysis and comparison with other random number generators that have been or is currently being used, the results will show if Promestra security's RNG can hold up to the same standards as for example Mersenne twister, CryptGenRandom and a few others.

## 1.2 Scope

The scope of the study is limited to the random number generator of Promestra security, not the full encryption algorithm or key-generation. The study will bring up potential problems with the RNG but won't cover how these problems can be exploited or any potential solutions. Instead of doing tests on the full security concept the study will focus on the RNG, this will help us to evaluate specifically the RNG if it is good enough while other specific parts could be considered as future work.

## 1.3 Research Questions

These are the research questions that we will try to answer in this thesis.

1. Using NIST test framework could PS mathematically be classified as random?
2. Analyzing the results from NIST tests, are there any limitations to the random number generation of PS?

This thesis will use the following objectives to answer the research questions.

## Objectives

1. Gather different random number generators.
2. Produce random number sequences from the algorithms.
3. Run the 15 tests made by NIST using the data produced in step 2.
4. Analyze results from the tests.

# Chapter 2

# Background

Promestra security(PS) is a security concept created by Promestra AB. Their vision is that their system can be used in many different situations such as encrypting files, encrypting communication, usage in General Data Protection Regulation(GDPR)[6] and encryption in Internet of things. They have also stated that it should be possible to use wherever there is a need for security and that it is 40 times faster than Advanced Encryption Standard(AES). Promestra AB has previously combined PS together with biometric as identification with the usage of, for example, fingerprints. They are now looking to reincarnating the project for future use. Further this study will only focus on the RNG part and not how the encryption/decryption functions work.

It is not easy to convince companies and organizations to change crucial parts of their systems like the encryption, therefore it needs to be tested and show a good result in order for companies to even start thinking about switching their systems.

A truly random sequence can be defined using terms of probability, there are multiple ways to detect different patterns in a sequence using statistical tests. Using these statistical tests, we can with great confidence conclude if a sequence is truly random, or if the data isn't random. There will always be a risk that a truly random sequence gets rejected and defined as non-random, just as there is a risk that a sequence that isn't random is defined as random. The goal of these tests is to lower the risk of these errors occurring[19]. By using these tests, this study will present a statistical comparison between Promestra security and 9 other random number generators. By comparing the results with more well known random number generators, a conclusion regarding Promestra security's randomness can be made.

## 2.1 Random number generators

There are different types of Random number generators(RNG), the two most common are called true random number generator(TRNG) and pseudo-random number generator(PRNG). RNGs can be found in many different applications, games, lotteries and more. This could be because that the application or game want to: randomize the teams, randomize the order of who goes first or just to randomize a deck of cards. It can be used in many different ways. In order for a PRNG to work properly, it needs a value to start with, this is called "seed". If a seed is used twice on the same bit-string in a single PRNG, it should produce the same output both times[13].

Most cryptographic algorithms needs a random source to generate its output, this is also assumed by most cryptography practitioners to be perfectly random, that is, a uniformly distributed and unbiased sequence of bits. As a pseudo-random number generator needs at least one initial value to generate a new random number, there are problems of already protecting the state of the pseudo-random number generator so that the next number can't be predicted[18].

### 2.1.1 Pseudo-random number generator

PRNGs are not seen as truly random as the same seed will give the same output and it is being calculated by some mathematical formula. The goal of a PRNG is to simulate a random sequence and is therefore much faster than a TRNG [3]. The German office of information security (Bundesamt für Sicherheit in der Informationstechnik, BSI) have 4 criteria for PRNGs to prove their quality.

- K1 - Should be a high probability that generated sequences are different from each other.
- K2 - A randomized sequence should be impossible to separate from "true random" sequences when performing some of the NIST-tests and also that no subsequence contains information about future sequences.
- K3 - It should be impossible for an attacker to calculate previous or future sequences from a subsequence.
- K4 - It should be impossible for an attacker to calculate previous or future sequences from an inner state of the generator.

[20]

### 2.1.2 True random number generator

A TRNG is called true random because of the usage of seeds that comes from a source that should be impossible to predict such as hardware, "noise-signals" or the current time with nano-seconds together with another source. They generate numbers by physical processes instead of using an algorithm, this is where the prediction is impossible to perform [3]. TRNGs are slower than PRNGs and therefore in some cases a TRNG is used to randomize the seed used in a PRNG instead of letting the TRNG do the whole process.

# Chapter 3

# Related Work

## Statistical Evaluation of Cryptographic Algorithms

This study presents an improvement to the NIST tests [19] using new statistical rules, new decision rules and also an implementation of the five finalists in the AES competition initiated by NIST in 1997[15]. The original NIST test suite (800-22) is using percentage of passed tests and the uniformity of the P-values. This study presents result decision based on statistics maximum values and decision based on the test functions values sum of squares as a better way to show the results [2]. This could be added to future works and to test if the supposed "better" test suite gives somewhat similar results to our study.

## Random Number Generator Based on Hydrogen Gas Sensor for Security Applications

In cryptography it is crucial to have a random factor as stated in the background section. The authors of the study has created a RNG based on a hydrogen gas sensor which is non-deterministic. The seed is therefore unknown and cannot be predicted easily. They use thirty-two data points (four sensors and eight hydrogen concentration) to obtain the random numbers and then uses post-processing with SHA-256 to prevent the bits from being bias since their measurement is limited[8]. Lastly they present a table with results from the NIST test suite[19] where all test are passed. This could be useful to us as a comparison when the evaluation of PS is made.

## Second-level NIST Randomness Tests for Improving Test Reliability

This study presents a way to mitigate and analyze the results from the tests using what they call a second level test. It is a problem that pseudorandom RNGs can pass some of the tests when for us as a human is clearly not random. For example, a periodic generator (A generator with a recurring output) could easily pass the Frequency test if the period contains the same amount of ones and zeros. The second level test checks for uniformity of N amount of P-values gathered from different sequences[21]. This second level test could be considered as future work to our study if the NIST test framework is seen as unreliable in evaluating the algorithms.

### Pseudo-random number generator based on discrete-spacechaotic map

The authors of the study presents a way of obtaining pseudo-random numbers. The method is based on a discrete chaotic map, based on composition of permutations. The method is said to have virtually unlimited key space and has the ability to generate the same number of different sequences as other secure discrete chaotic methods do, but with significantly lower memory requirements. The proposed PRNG can be used in devices that has limited memory. Their PRNG is tested with the same NIST framework as our thesis. The analysis in this study however, is only presenting the proportion as a result from the NIST tests and not the distribution[12].

### A Technique to Test Non-Binary Random Number Generator

The testing of random output from RNGs is basically only created for binary sequences. Because of that, this study proposes a way of testing the randomness in non-binary output using "von Neumann whitening" and other common known statistical tests[7]. Although our study will test the outputs using binary values, it is always good to have more alternatives and it will probably be an easier way for amateurs in the field to be able to test by themselves when using non-binary sequences.

# Chapter 4

# Method

The presented results in this study will come from a statistical analysis using the NIST tests[19]. A comparison between different random number generators will be made. To be able to give a fair evaluation of these RNGs we will produce at least $10^6$ bits of ones and zeros as recommended by NIST. We decided that we would run the NIST tests instead of other test suites like the "DIEHARD test battery"[5]. We based the selection of testing suite on a few different factors. Firstly, it is said that if a test is passed using NIST tests it is said to be passing the industry standard. Secondly, it was designed to be tested with binary numbers and finally, that it was used for the evaluation of the AES competition[15], encryption algorithms that were supposed to protect sensitive information[11].

To be able to run these tests we randomize these bits and then place them in a file that will be the input of the test program. The randomizing part is mostly the same for most of them with the exception of different amount of bits that are being randomized each run.

Some algorithms are randomizing decimal numbers ($x_{10}$) their outputs are then converted to binary numbers ($x_2$) in order to be able to run the tests. The table below shows all different algorithms that will be compared in this study.

| Name | Abbreviation | RNG Type |
|---|---|---|
| Blum Blum Schub | BBS | PRNG |
| CryptGenRandom | CGR | PRNG |
| Linear Congruential | LCG | PRNG |
| Mersenne twister | MT | PRNG |
| Micali-Schnorr | MS | PRNG |
| Modular Exponentiation | MEXP | PRNG |
| Promestra Security | PS | PRNG |
| Quadratic Congruential 1 | QCG 1 | PRNG |
| Quadratic Congruential 2 | QCG 2 | PRNG |
| Random.org | R.ORG | TRNG |

Table 4.1: A list of random number generators with their type and referred abbreviation

### 4.0.1 Promestra Security

Promestra AB provided us with an assembly routine that randomizes 13 000 000 bytes to a binary file, we will scrap the last 500 000 bytes as they are not needed for the tests. We have not modified the file in any way to give a fair assessment. The program needs a seed but that was also provided by Promestra AB.

### 4.0.2 Random.org

Random.org is a website that has a lot of functions such as randomizing strings, numbers, bytes, coin-flip tools and much more. It is a true random generator which means that a seed is not needed as input. Instead Random.org generates data via atmospheric noise[9]. For this study we will be using their service that provides randomized bytes, in total we are getting 12 500 000 bytes which equals to the one hundred million bits we need for the tests. We expect that this generator will pass all the tests, as they regularly test their generator and do so with multiple of the tests that were used in this study[17].

### 4.0.3 Mersenne Twister

As Mersenne twister is a pseudo-random number generator we need a "seed". This has been generated with the help of an inherent random function in the Linux operating system known as /dev/urandom, this is a random number generator using different system timings to produce random numbers[10]. This seed was used to generate 4-byte integers which was converted to their binary representation of 32-bits which we could run the tests on.

### 4.0.4 CryptGenRandom

CryptGenRandom is a random number generator available on Microsoft Windows systems, this is supposed to be cryptographically random according to the documentation of the function[14]. Following the documentation we generated the bits needed for the tests.

### 4.0.5 Blum Blum Schub

Blum Blum Schub is a pseduorandom number generator. It uses two large prime numbers and also a variable (x) that is coprime to $p*q$ in the beginning to be able to start randomizing the numbers. Coprime means that $gcd(p*q, x) = 1$. The variables p, q and the seed are automatically selected within the program and has not been selected in a specific way. Since the prime numbers has to be large its main problem is that it is a bit slow[4].

### 4.0.6 Included random number generators

The test suite comes with some predefined generators that are ready to generate random number sequences that can be tested. We have tested the following included random number generators.

- Linear Congruential
- Modular Exponentiation
- Quadratic Congruential 1
- Quadratic Congruential 2
- Micali-Schnorr

## 4.1 How the NIST framework were used

We used the original test suite (NIST SP 800-22) made by NIST which is a C-program that is available for download on their web page[16]. To ensure that the test suite is working correctly after setup on our machine NIST has included a form of validation by including some predefined datasets and the expected results from the different tests. By comparing our results with the results in "Appendix B" of the NIST user guide[19] we were able to validate that the tests work correctly. This ensures that our test environment is working as intended and should therefore provide us with reliable results.

Following the recommendations stated in the documentation of the test suite, we will be running tests on one hundred sequences of each random number generator. Each sequence contains 1 million bits which is the recommended minimum for multiple tests. All of the following test used the default parameters defined in the documentation[19], except the number of templates used by the non-overlapping template test described in section 4.2.7. This test instead used a maximum number of templates set to 148 as it is all possible aperiodic templates for the specified size.

## 4.2 The NIST tests

All of the 15 tests will present a P-value as a result, this value will tell if the sequence tested is considered random or non-random when depending on the null hypothesis($H_0$). In our case if $H_0$ is true, the result should be considered random. If it is false it is considered non-random.

In this study the P-value will be in the range of $0 - 1$, with a significance level($\alpha$) of 0.01, as recommended by NIST[19]. That means that if $p < 0.01$ it should be considered non-random, otherwise if $p \geq 0.01$ the conclusion is that the sequence is random. As the chosen $\alpha = 0.01$ we can expect that 1% of the sequences will fail, even though they are actually truly random. There is always the possibility of failing more tests than the expected amount of times and therefore being labeled as non-random although it is very small.

All of the test descriptions refers to NISTs documentation[19]. These descriptions are only supposed to give the info necessary to understand how they can help us detect a good or bad random number generator without going into too much depth.

### 4.2.1 The Frequency (Monobit) Test

The purpose of the first test is to find out the proportion of the random bit sequence and how many ones and zeros that exists in the sequence. The proportion should be somewhat equal and therefore be close to $\frac{1}{2}$. If this test fail, it is likely that other tests will fail since this is the most basic evidence for non-randomness. This test detects if there are too many ones or zeros[11].

### 4.2.2 Frequency Test within a Block

This second test checks a block of size $M$ bits and tries to see how many ones there are. A chosen block size of $M = 1$ would make this test equal to the Monobit test in section 4.2.1. The expected outcome of this test would be to find that the number of ones is approximately $\frac{M}{2}$.' This test also detects if there are too many ones or zeros[11].

### 4.2.3 The Runs Test

This test will check the total number of runs in the bit sequence, a run can be explained as an uninterrupted string of the same bits, ones or zeros. A run is therefore the length of identical bits in succession, for example, a run of length n consists of n zeros or n ones. This test sees if the oscillation of ones and zeros are too fast or too slow[11].

### 4.2.4 Tests for the Longest-Run-of-Ones in a Block

The focus of the test is to see how long the longest run of ones are in an $M$-bit block. It tries to determine if the longest run of ones is that of what is expected in a random sequence. Notably an irregularity of the expected length of the longest run of ones will also mean an irregularity of the expected length of the longest run of zeroes. Therefore we only need to run a test for ones. This test also sees if the oscillation of ones and zeros are too fast or too slow[11].

### 4.2.5 The Binary Matrix Rank Test

This test focuses on the rank of disjoint sub-matrices for the entire sequence and has the purpose to examine linear dependence among fixed length substrings from the original sequence. The test will divide the bit string into $N$ matrices of size $MQ$ which is defaulted to $M = 32$ and $Q = 32$, this gives us $N = \frac{n}{MQ}$ amount of matrices where $n$ is equal to the length of the bit string. The test will also discard the last $n - N$ bits as they can not form a full matrix. This test detects deviations from the expected rank distribution[11].

### 4.2.6 The Discrete Fourier Transform (Spectral) Test

This test is also referred to as "FFT" in the result tables. This test has its focus on the peak heights in the Discrete Fourier Transform of the bit sequence, with the purpose

of detecting repetitive patterns that are near each other (i.e., periodic features) which would indicate a deviation from our hypothesis of randomness. The intention of this test is to find out if the peaks that exceed the 95% threshold is significantly different from the expected 5%. This test detects problems with repetitive patterns[11].

### 4.2.7 The Non-overlapping Template Matching Test

This test focus lies on the number of occurrences of target strings which are predefined. This tests purpose is to detect aperiodic patterns which occur too often. For this test as well as the "Overlapping Template Matching Test" in section 4.2.8, a frame of size $m$ is used to search for the defined $m$-bit pattern. If the pattern is found the frames position is set to the bit right after the found pattern to avoid overlapping patterns. However, if the pattern is not found the frame moves its position by one bit.

There are a total of 148 different predefined templates that are used in this test, each of these templates are all the possible non-periodic sequences of the size $m$ which for this study is $m = 9$. This test find irregular occurrences of pre-specified templates[11].

### 4.2.8 The Overlapping Template Matching Test

This test has the same focus and purpose as the "Non-overlapping template matching test" described in section 4.2.7. This test also uses a frame to search for a defined pattern of $m$ bits, and just as in section 4.2.7, it moves the frame by one bit whenever it doesn't find the pattern. If however it does find a pattern the frame still only moves one bit which gives the possibility of finding overlapping patterns, compared to the test in section 4.2.7 where there is no possible overlapping.

The template that is used by this test is $m$ bits where all bits are equal to 1. This test find irregular occurrences of a pre-specified template[11].

### 4.2.9 Maurer's "Universal Statistical" Test

This test focuses on the number of bits found between two matching patterns. As a sequence which is significantly compressible is considered non-random, the purpose is to see if the sequence can be compressed without the loss of information. This test tries to find out if the tested sequence is compressible[11].

### 4.2.10 The Linear Complexity Test

The focus of this test is on the length of a linear feedback shift register(LFSR). Longer LFSR's is considered a characterization of random sequences, a too short LFSR implies non-randomness. This test is performed to find out if the sequence is complex enough to be considered random. This test detects too short LFSR[11].

### 4.2.11 The Serial Test

In this test we focus on the frequency of overlapping $m$-bit patterns within the full sequence. This test has the purpose to find out if the number of occurrences of the $2^m$ $m$-bit overlapping patterns is approximately equal to that of what we expect from a random sequence. With a value of $m = 1$ this test is equal to that of the test in section 4.2.1. This test will compute two different p-values essentially giving us two different tests, both tests are based on the same patterns found and are just calculated differently, which will not be covered exactly in this study but can be found in the NIST documentation[19]. This test detects non-uniformity in the joint distribution[11].

### 4.2.12 The Approximate Entropy Test

This test also focuses on overlapping $m$-bit patterns across the entire sequence, just as the serial test in section 4.2.11. Although this test focuses on the frequency of two adjacent length overlapping blocks($m$ and $m + 1$) and compares with the expected result of a random sequence. This test detects non-uniformity in the joint distribution[11].

### 4.2.13 The Cumulative Sums(Cusums) Test

This test focuses on the random walks defined by a cumulative sum of adjusted digits in the sequence, these adjusted digits are by the formula $X_i = 2\varepsilon_i - 1$ where $\varepsilon$ is the random sequence of bits being tested. This test has two different modes known as forward and backward. These will both be presented in the results where the first of the two is using the forward mode. The difference between these two modes is if the partial sums will start from $X_1$ or if they will start from $X_n$. You then calculate the sum om partial sequences according to the formulas: $S_k = S_{k\ 1} + X_k$ for a forward test and $S_k = S_{k\ 1} + X_{n\ k+1}$ for the backwards test. This test tries to determine if the cumulative sum of the partial sequences found in the tested sequence is too big (or too small) for what we expect from a random sequence. This test detects a problem with too many ones or zeros in the beginning or the end of the sequence[11].

### 4.2.14 The Random Excursions Test

This test has its focus on the number of cycles having exactly $K$ visits in a cumulative sum random walk. This cumulative sum random walk is derived from partial sum after the bit sequence is transferred to the adjusted digit sequence($-1$, 1). A random walk cycle is a sequence of steps of random unit length that begins and returns to the origin, a cycle is basically the number of zero crossings in a random walk. This tests purpose is to determine if the number of visits to a particular state within a cycle deviates from what is the expected result of a random sequence. This test works as a series of eight tests, with one test for each of the states: $-4, -3, -2, -1$ and $+1, +2, +3, +4$. If the number of cycles is $< 500$ the randomness hypothesis is rejected, this means that the test will have a lower total proportion. This test

detects deviations from the distribution of the number of visits of a random walk to a certain state[11].

### 4.2.15 The Random Excursions Variant Test

This tests focus is on the total number of times that a specific state occurs in a cumulative sum random walk. It works very much in a similar way to that of the random excursions test described in section 4.2.14 The purpose of this test is to detect any deviation from the expected number of occurrences to various states in the random walk. This test works as a series of eighteen test, with one test for each of the states: $-9, -8, ..., -1$ and $+1, +2, ..., +9$. This test detects deviations from the distribution of the number of visits across many walks to a certain state[11].

## 4.3 Answering the research questions

The first research question(Using NIST test framework could PS mathematically be classified as random?) will easily be answered if PS fails any of the tests in the NIST framework. However the second research(Analyzing the results from NIST tests, are there any limitations to the random number generation of PS?) question we will have to compare the results with other RNGs to see where Promestra seems to have any limitations, these will still be based around the tests from NIST.

# Chapter 5

# Results

## 5.1 Interpretation of results

From the statistical test suite we get two different results which are both of importance, these are a p-value describing the uniform distribution, and a proportion value, representing how many of the tests succeeded.

By splitting up all the results from each test into ten intervals where each interval is defined as $i1 = 0 \leq p \leq 0.1$ $i2 = 0.1 < p \leq 0.2$ ... $i10 = 0.9 < p \leq 1$. We can calculate the uniform distribution's p-value with the following formula:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - \frac{s}{10})^2}{\frac{s}{10}} \tag{5.1}$$

where $F_i$ is the number of P-values in an interval i, and s is the size of the sample.

If the value is $\geq 0.0001$, the distribution can be considered uniformly distributed. To not confuse this p-value with the p-value from each test described earlier in section 4.2, we will be referring to this as the distribution.

The proportion value shows how many of the test that succeeded, that is, how many tests that had a p-value $\geq \alpha$. The minimal pass rate for every statistical test approximately 0.96, this pass rate comes from calculating the confidence interval, which is defined with the following formula:

$$\hat{\mathrm{p}} \pm 3\sqrt{\frac{\hat{\mathrm{p}}(1-\hat{\mathrm{p}})}{s}} \tag{5.2}$$

where $\hat{\mathrm{p}} = 1 - \alpha$ and s = sample size.

Random excursion and random excursion variant is dependent on the amount of cycles[12] that is possible and can therefore vary a bit but the average number of tests is about 60.
In the next section figures will be presented, in which higher values means better performance.

## 5.2 Obtained results

As there is a total of 188 test results from each random number generator, we have chosen to only show the test results where there were any significant results. Abbreviations used can be found in table 4.1

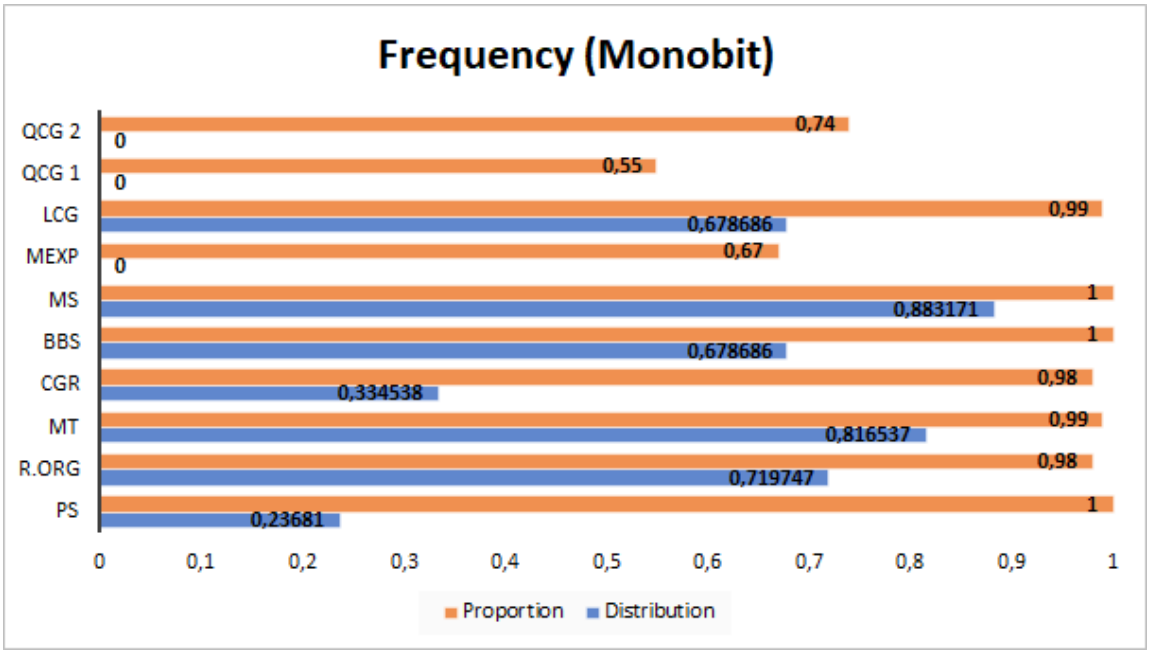


Figure 5.1: Results from the frequency test

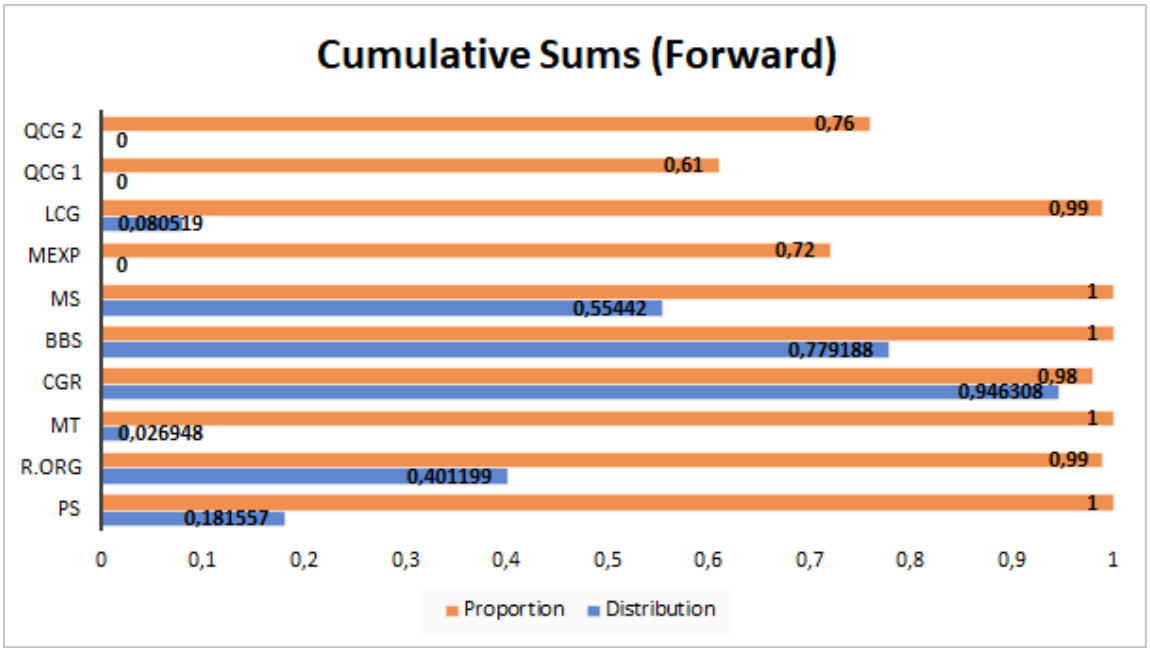


Figure 5.2: Results from the cumulative sums test using the forward mode

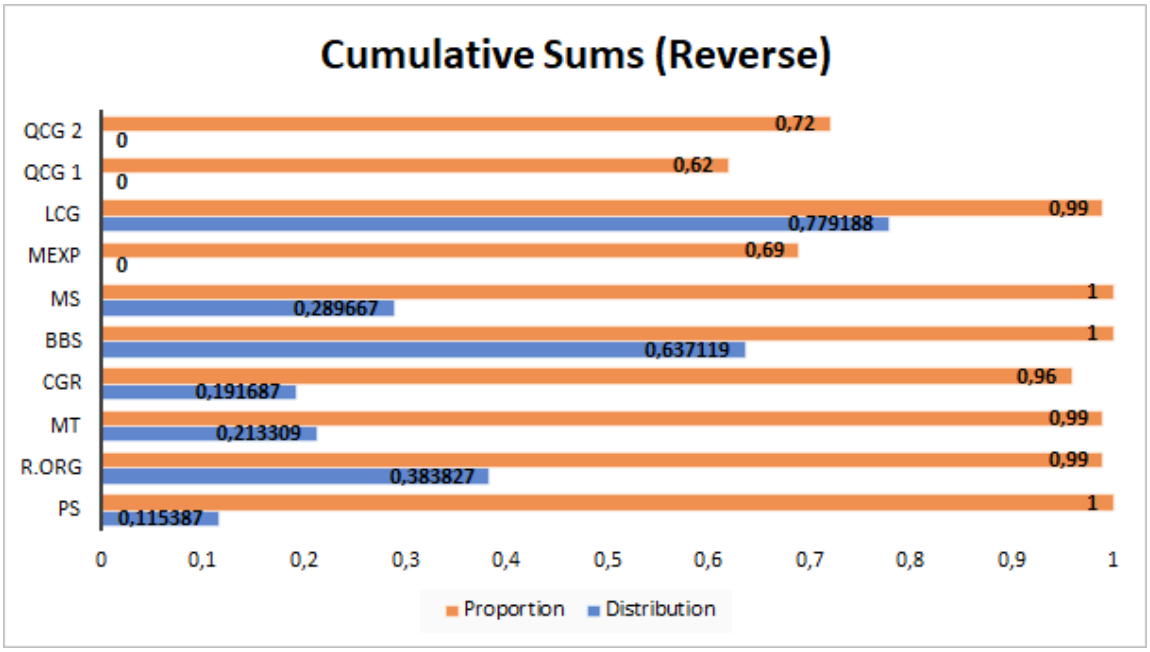


Figure 5.3: Results from the cumulative sums test using the reverse mode

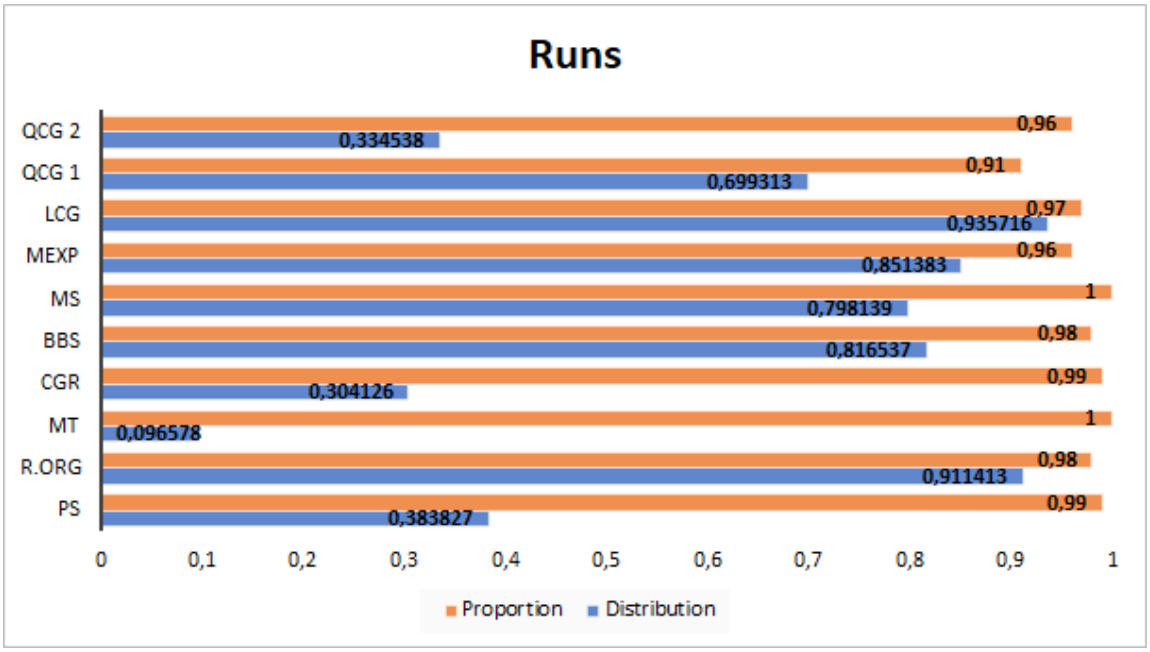


Figure 5.4: Results from the runs test

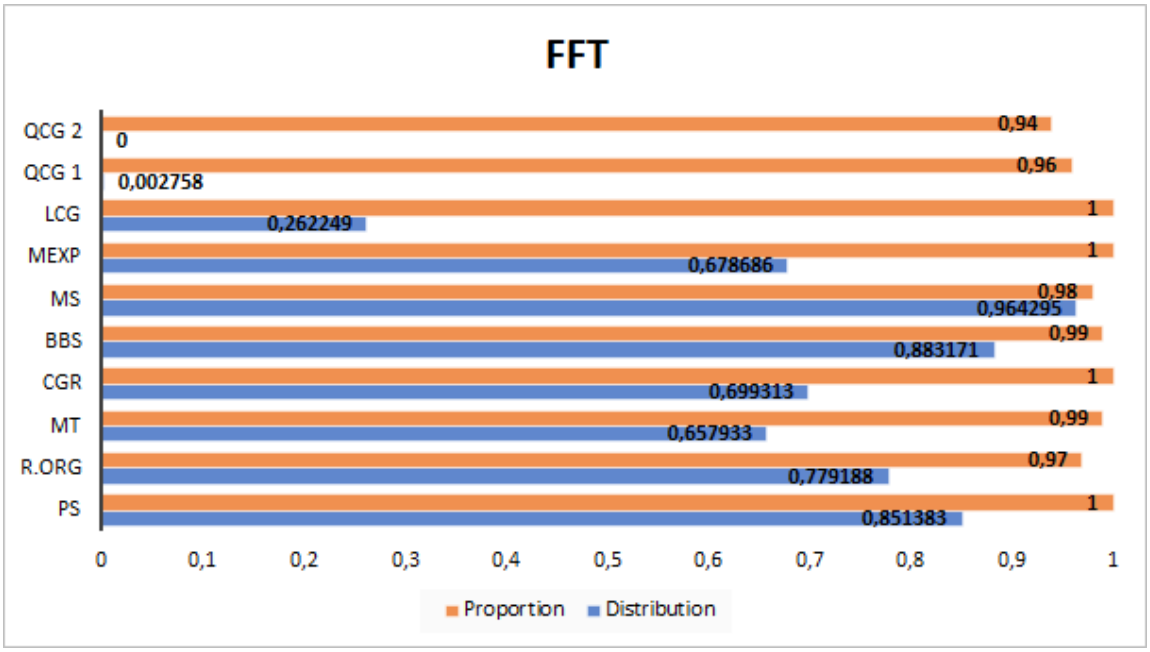


Figure 5.5: Results from the discrete fourier transform test

Failing one of the following template tests, means that the template used occurs an irregular amount of what one would expect in a random sequence.

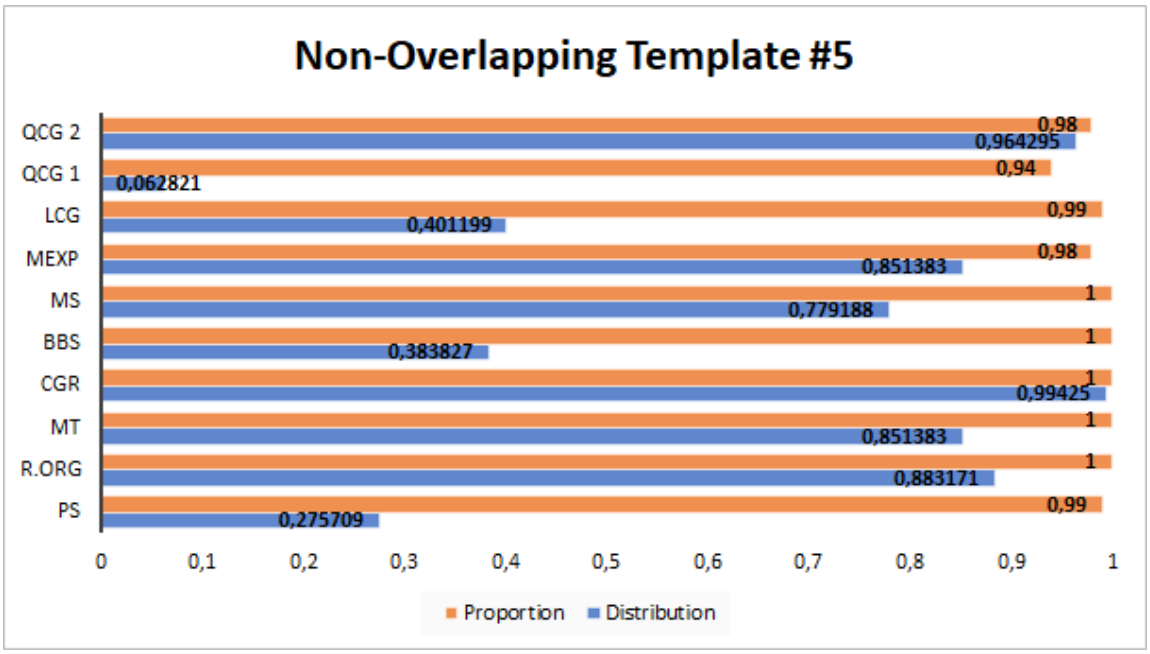


Figure 5.6: Results from the non-overlapping template test using template #5. The template used in this test is "000001001".

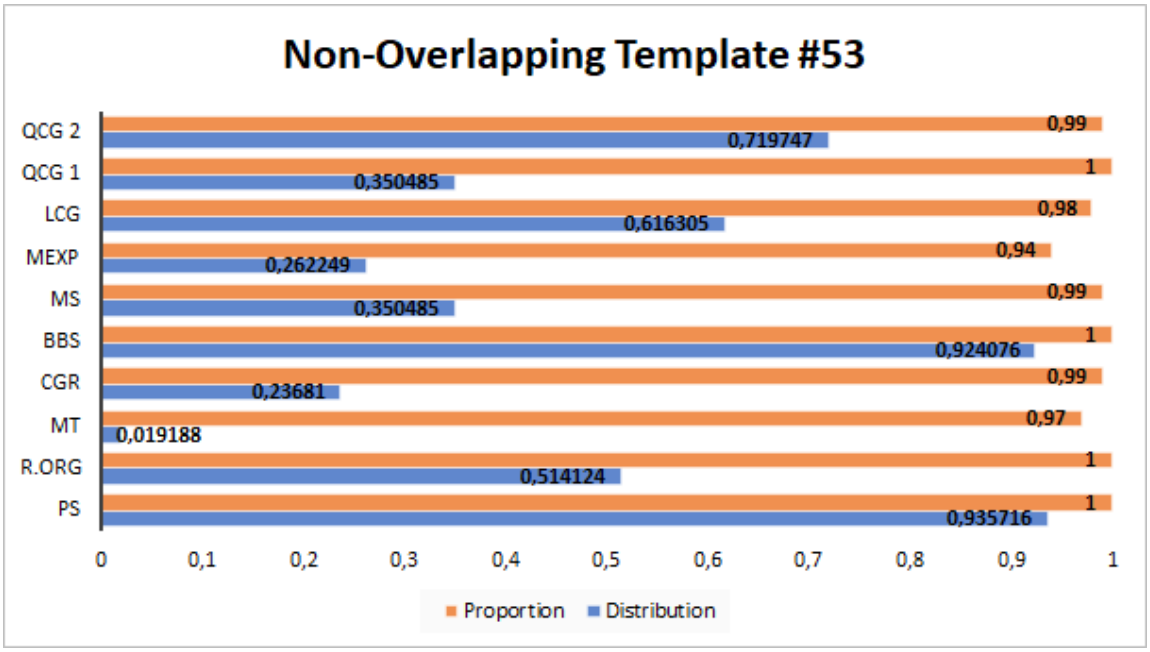


Figure 5.7: Results from the non-overlapping template test using template #53. The template used in this test is "010000011".

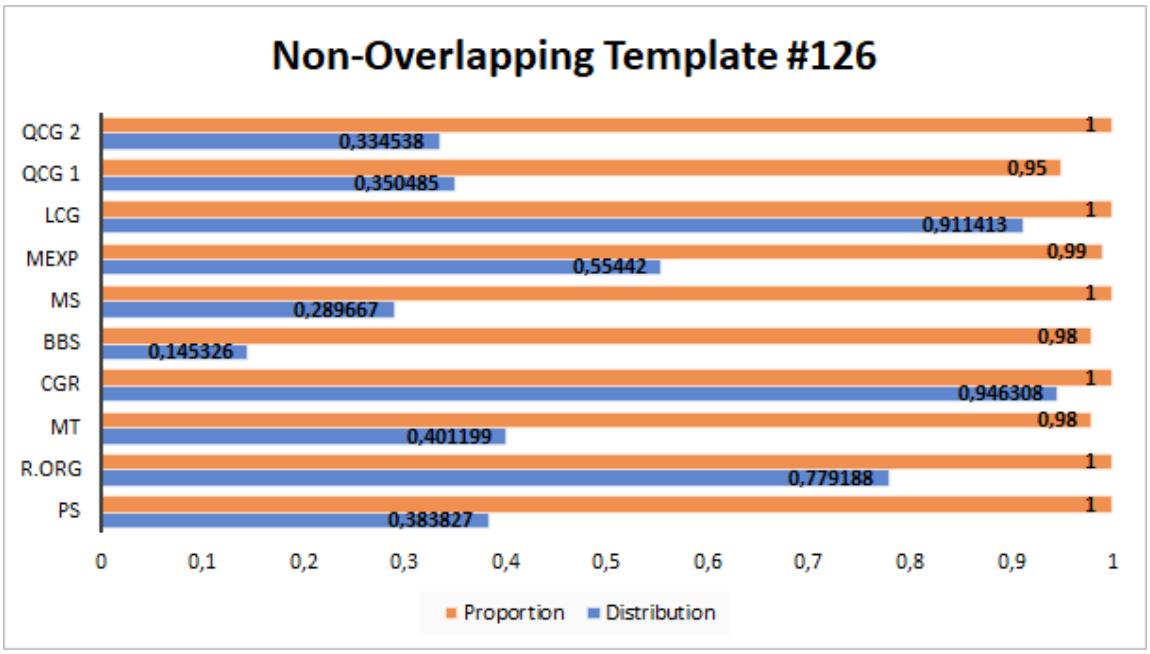


Figure 5.8: Results from the non-overlapping template test using template #126. The template used in this test is "111010000".

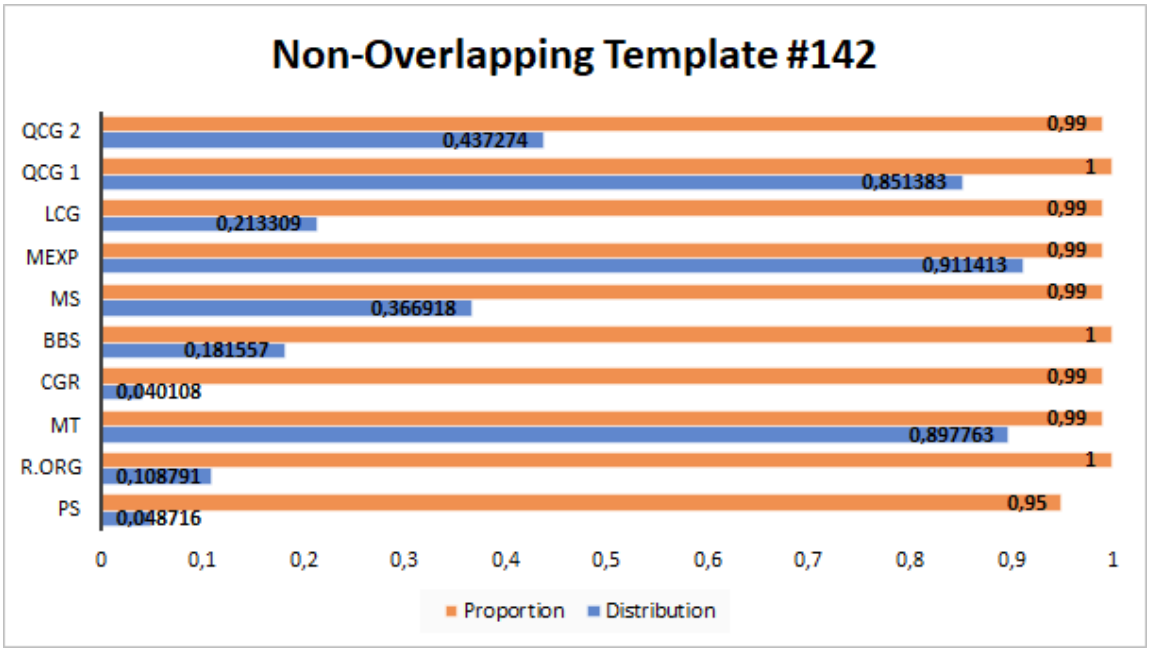


Figure 5.9: Results from the non-overlapping template test using template #142. The template used in this test is "111110010".

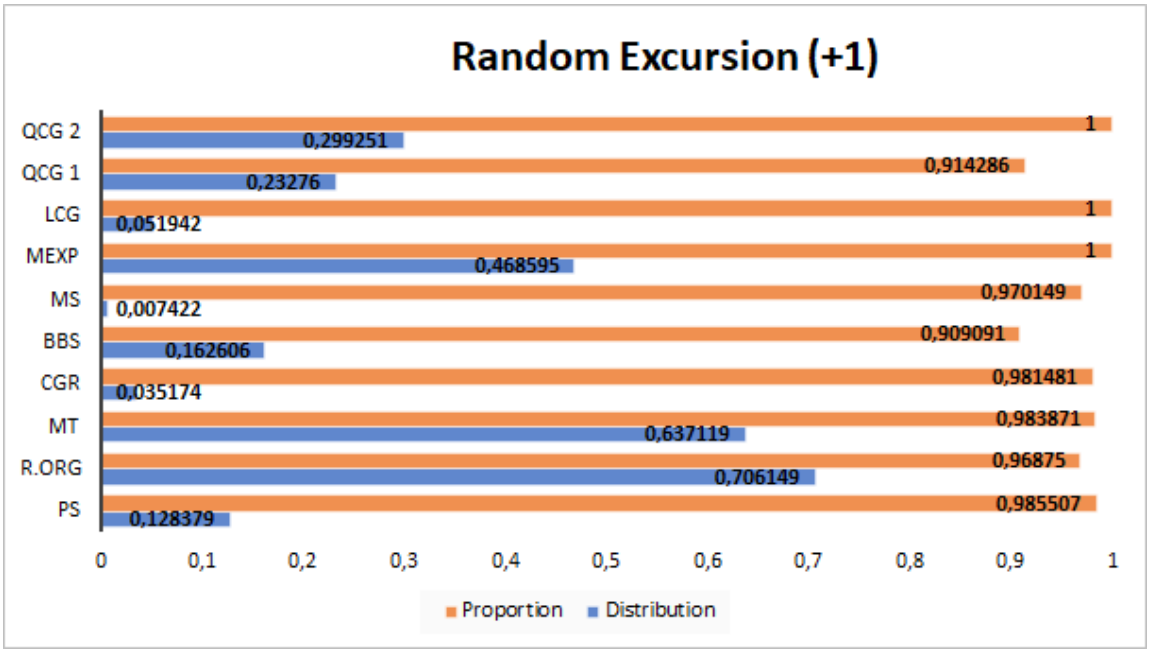


Figure 5.10: Results from one of the random excursion tests checking for states of +1

The amount of tested sequences in Modular Exponentiation, Quadratic Congruential 1 and 2 are only 36, 35 and 30 respectively due to its amount of cycles, while the average number of tested sequences for the other generators is above 50. This means that the test suite has a lower level of acceptance than usual.

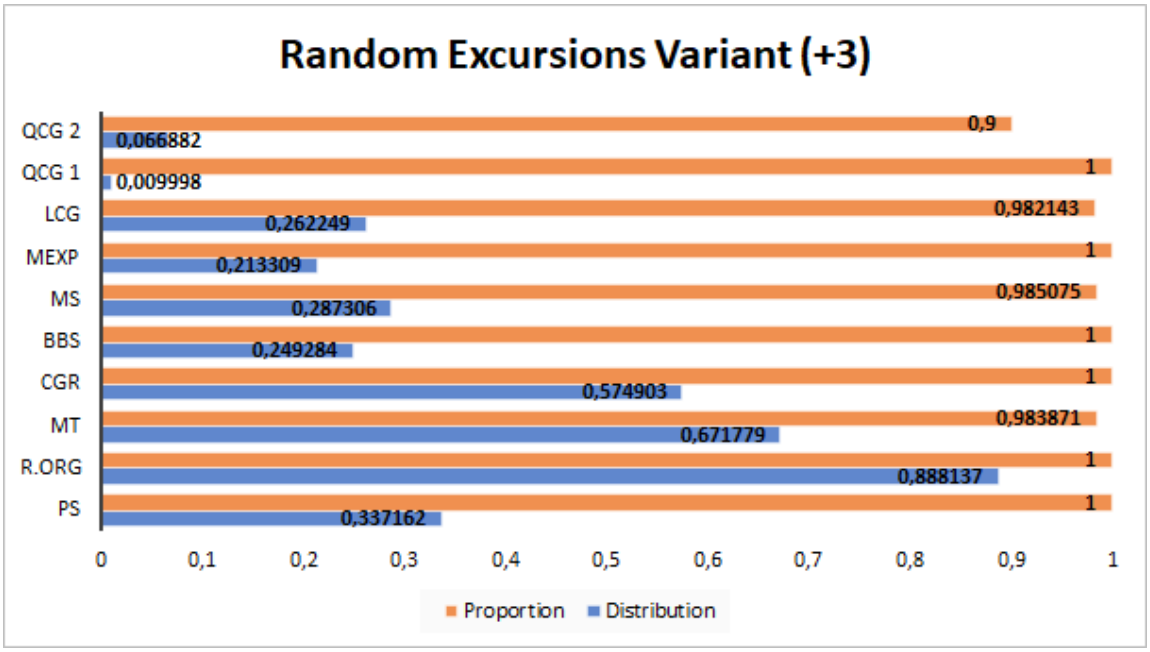


Figure 5.11: Results from one of the random excursions variant tests checking for states of +3

As the random excursion & random excursion variant test gets the same number of tested sequences, the same problem noted for figure 5.10 occurs here, where some of the tested generators will have a little different accuracy.

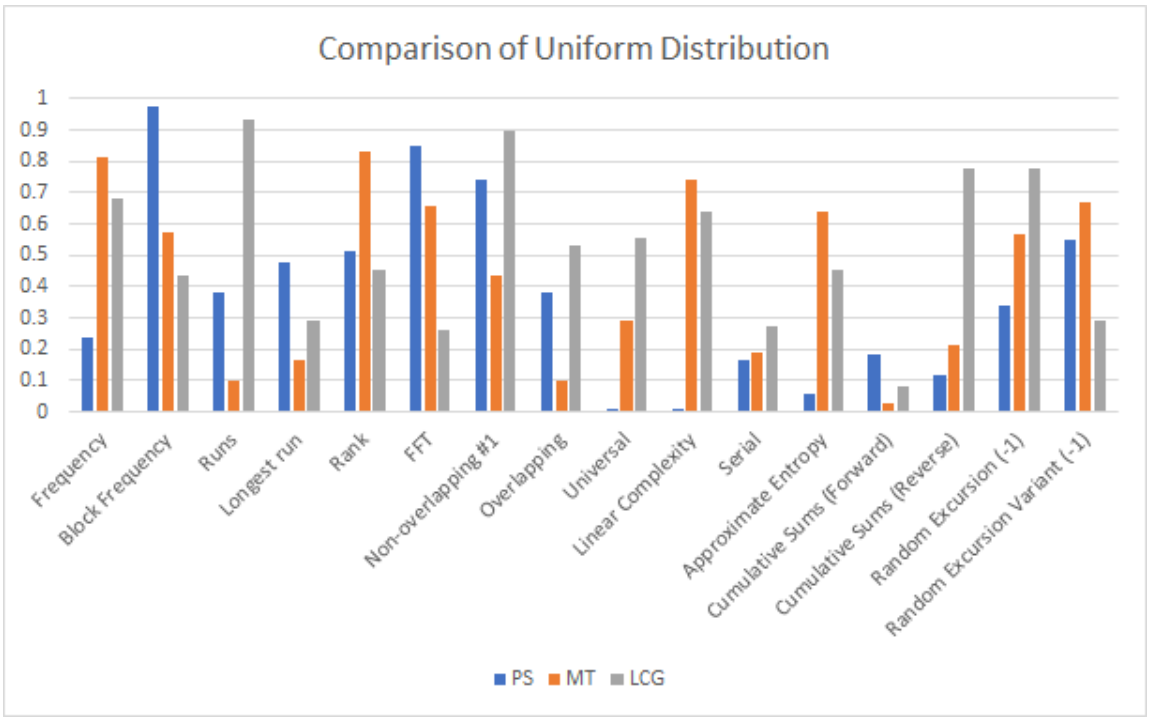


Figure 5.12: A comparison of the uniform distribution for a subset of the tests between PS, MS and LCG

As Mersenne Twister and Linear Congruential are both well-known pseudo-random number generators they are used as a comparison for how well Promestra security performed. The chart is only of a subset of test and does not include any failed tests, even though Promestra security did fail one of the tests seen in figure 5.9.

The results from all the tests can be found in Appendix A. These tables can be seen as a complement to the already presented results above. These tables show exactly how the distribution between the ten intervals are made and also presents the achieved P-Value and proportion. For the distribution to be optimal, all of the intervals C1-C10 should be as close to 10 as possible.

# Chapter 6

# Analysis and Discussion

As we have used a significance level of $\alpha = 0.01$ the expected result is that 1% will fail. Consequently a proportion of 0.99 doesn't necessarily implicate that it is worse than a proportion which is $> 0.99$. Therefore we will consider a proportion of 0.99 or higher, as very good while any other passing proportion is considered good.

If a test has severe issues and the proportion is nowhere close to being acceptable, the calculation of distribution will automatically be very bad because of the amount of tests that gets put in the first interval. Therefore we will not analyze the distribution of a test that clearly isn't random. For the resulting p-value of the uniform distribution to be meaningful a sample size of at least 55 sequences is recommended by NIST[19]. For this reason, some random number generators will not have enough sequences to process, more specifically, during the random excursion and random excursion variant tests.

Analyzing the results it is indisputable what random number generators have failed and which have passed. Although these seem clear-cut, when dealing with randomness there is always the risk, albeit very low, that it was just random failure or success. Therefore those that are right on the edge of pass or fail could need further testing before one write them off completely, or fully trust them to be unpredictably random.

## 6.1 Analysis of each failed test

### 6.1.1 Frequency

Looking at figure 5.1 it is evident that three out of the ten tested generators failed the frequency test in both proportion and in their distribution. On the other hand, Promestra, Blum-Blum-Schub and Micali-Schnorr achieved a score of 1.

### 6.1.2 Cumulative Sums

In the cumulative sums tests (figure 5.2 & 5.3) we have the same three random number generators as those that didn't pass the frequency test, namely, Modular exponentiation as well as Quadratic congruential 1 and 2 not pass the cumulative sums tests. It seems that Mersenne twister is a bit on the lower side when it comes the distribution of the cumulative sums test using the forward mode. While it does

get a very good score on proportion, the uniform distribution is not looking good compared to the others that completed the test.

### 6.1.3 Runs

The results of the runs test (figure 5.4) looks quite good for all of the random number generators tested, except for Quadratic congruential 1 which failed its proportion. Quadratic congruential 2 & Modular Exponentiation which did fail the previous mentioned tests barely passed this test.

### 6.1.4 FFT

Analyzing the results from the FFT test (figure 5.5), Quadratic congruential 2 has clearly failed. Quadratic congruential 1 however did pass both tests but is on the very edge of both proportion and distribution. All other random number generators has a really good result on both proportion and distribution.

### 6.1.5 Non-Overlapping Template

The non-overlapping template test is tested with 148 different templates. The templates that did not pass for all of the generators can be found in section 5.2. Promestra security is below the threshold for passing the proportion in one of the non-overlapping template tests, as seen in figure 5.9, which is the results for template #142. Promestra security is right on the edge with 0.95, while it is possible that this is just an anomaly, the low distribution value suggests that this isn't the case. Either way this is certainly a limiting factor to the randomness of PS. Modular Exponentiation and Quadratic congruential 1 also failed one and two tests respectively, these two failed with a larger margin and as such, it is less likely that these tests are exceptions. The figures 5.6, 5.7 and 5.8 shows the results for the failed templates.

### 6.1.6 Random Excursions & Random Excursions Variant

Analyzing both of these tests can be a bit hard because the amount of accepted iterations varies between the random number generators. Some only has about 30, while others have double the amount. Since the distribution is dependent on the amount of iterated tests we will not be able to reach a conclusion on the distribution of multiple tests. Consequently, it will be hard to compare the results between different generators fairly as some will have a lower accuracy than others.

As the generators affected by this has already failed multiple tests we will not look into this further. NIST recommends that at there needs to be at least 55 tested sequences before the uniform distribution result gets significant[19], as such we can disregard that CryptGenRandom and LCG get quite low as they had just 54 and 56 tested sequences respectively. Micali-Schnorr however did have 67 tested sequences but still a very low distribution, this could possibly be a limiting factor of MS.

## 6.2 Analysis of Promestra Security

As Promestra security has failed one of the tests, specifically the test in figure 5.9 which tested for an irregularity of occurrences of a specific template. This indicates that Promestra security has a problem where it either has too many or too few occurrences of a specific aperiodic template than what is expected out of a random sequence of the tested length. This is clearly a limitation of Promestra security that restrict the possibility of being classified as random. For a random number generator to be concluded as random, all the tests has to pass in both their proportion as well as their distribution. The conclusions made in the studies [12, 8], is that the output of the generators are random, this was based on the fact that both passed all the tests. This is not the case for Promestra security which implies that it is non-random.

### 6.2.1 Comparison between Promestra Security, Mersenne Twister and Linear Congruential

In the first few tests, Promestra security seem to hold up with the standards of the other two compared random number generators. When it comes to the latter part of the tests, it shows that Promestra security is being outnumbered by both generators, especially in the universal, linear complexity and approximate entropy tests. While the tests still passed they don't have a very large margin and are still quite low compared to the other random number generators, we consider this to be a limitation of Promestra security. The limitation more precisely would be their distribution of the results for the following tests:

1. The Non-overlapping Template Matching Test
2. Maurer's "Universal Statistical" Test
3. The Linear Complexity Test
4. The Approximate Entropy Test

# Chapter 7

# Conclusions

By doing an analysis of the results, it is unquestionable what random number generators have failed the tests.

| Passed | Failed |
|---|---|
| Random.org | Quadratic Congruential 1 |
| Mersenne Twister | Quadratic Congruential 2 |
| CryptGenRandom | Blum-Blum-Schub |
| Micali-Schnorr | Promestra Security |
| Linear-Congruential | Modular Exponentiation |

Table 7.1: Number generators that passed or failed

If all the proportions are within the range of the calculation made in the equation 5.2, the result is random [12, 19]. As we are dealing with random numbers, there is always the risk that a random number generator outputs a sequence that looks non-random. The risk of this being the case is very small, albeit it is still a possibility. We recognize that both Promestra security and Blum-Blum-Schub barely failed one of the tests, and could still be a viable random number generator for non-security applications. Even though they could possibly still be viable, we can with great confidence conclude that they cannot hold the same standard as Random.org, Mersenne Twister, CryptGenRandom, Micali-Schnorr and Linear-Congruential generator.

When it comes to the limitation of Promestra security the test failed was one of the non-overlapping template tests, which indicates that there is an irregular amount of occurrences of an aperiodic pattern. This means that the template (111110010) is being searched for in the sequence. If the amount of instances is either too low or too many of what is expected in a sequence, the test will fail. Promestra security also seems to have a low distribution in a few tests, compared to the other random number generators. While it does pass these tests we consider this to be another factor that can limit the performance of Promestra security as a random number generator.

# Chapter 8

# Future Work

In this study we have looked at multiple pseudo-random number generators, these generators are dependent on a seed that is given to the generator before it starts generating numbers. We have merely tested these generators with a seed that at least appears to be random. Further work could involve testing how one or more generators may be affected by different seeds, as well as looking how one could mitigate that difference.

Another interesting test would be to do research and test how different random number generators influence various applications. It could prove beneficial to know what test results affect the performance of a random number generator in different applications and environments. This could help with the choice between multiple random number generators for a specific application.

There are said to be improved test suites available as described in chapter 3, these have not been tested in this study but could be a consideration for future work to test if there are any major differences between the test suites.

# References


[1] *An Introduction to Cryptography*. PGP Corporation, 2002.

[2] Mircea Andraşiu, Adrian Popescu, and Gheorghe Simion. Statistical evaluation of cryptographic algorithms. *2010 8th International Conference on Communications*, pages 473–476, June 2010.

[3] Mohamed Barakat, Christian Eder, and Timo Hanke. *An Introduction to Cryptography*. 2nd edition, 2018.

[4] Lenore Blum, Manuel Blum, and Mike Schub. A simple unpredictable pseudo-random number generator. *Society for Industrial and Applied Mathematics*, 15:364–383, May 1986.

[5] Robert G. Brown. Dieharder: A random number test suite. `http://webhome.phy.duke.edu/~rgb/General/dieharder.php`.

[6] European Comission. 2018 reform of eu data protection rules. `https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en#abouttheregulationanddataprotection`.

[7] Anna Epishkina. A technique to test non-binary random number generator. *Procedia Computer Science*, 145:193 – 198, 2018.

[8] Celal Erbay and Salih Ergïn. Random number generator based on hydrogen gas sensor for security applications. *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 709–712, Aug 2018.

[9] Mads Haahr. Introduction to randomness and random numbers. `https://www.random.org/randomness/`.

[10] David Jones. Good practice in (pseudo) random number generation for bioinformatics applications. 2010.

[11] Charmaine Kenny. Random number generators: An evaluation and comparison of random.org and some commonly used generators. April 2005.

[12] Dragan Lambić and Mladen Nikolić. Pseudo-random number generator based on discrete-space chaotic map. *Nonlinear Dynamics*, 90(1):223–232, Oct 2017.

[13] George Marsaglia. Seeds for random number generators. *Commun. ACM*, 46(5):90–93, 2003.

[14] Microsoft. Cryptgenrandom function. `https://docs.microsoft.com/en-us/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom`.

[15] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, and Edward Roback. Report on the development of the advanced encryption standard (aes). *Journal of Research of the National Institute of Standards and Technology*, 106:511—577, 2001.

[16] National Institute of Standards and Technology. Nist sp 800-22: Documentation and software. `https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software`.

[17] random.org. Real-time statistics. `https://www.random.org/statistics/`.

[18] Sylvain Ruhault. Security analysis for pseudo-random number generators. *Informatique / Cryptographie et sécurité*, 2015.

[19] Andrew Rukhinand, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. National Institute of Standards and Technology, 2010.

[20] Werner Schindler. Functionality classes and evaluation methodologyfor deterministic random number generators. *Anwendungshinweise und Interpretationen (AIS)*, 20:5–11, 1999.

[21] Gianluca Setti, Fabio Pareschi, and Riccardo Rovatti. Second-level nist randomness tests for improving test reliability. *2007 IEEE International Symposium on Circuits and Systems*, pages 1437–1440, 2007.

[22] Simon Singh. *The Code Book: HOW TO MAKE IT,BREAK IT,HACK IT, CRACK IT*. Delacorte Press, 2001.

# Appendix A

# Supplemental Information

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 5 | 12 | 6 | 8 | 14 | 13 | 12 | 7 | 15 | 0.236810 | 100/100 | Promestra Security |
| 12 | 12 | 5 | 8 | 9 | 8 | 13 | 13 | 9 | 11 | 0.719747 | 98/100 | Random.org |
| 8 | 6 | 10 | 12 | 10 | 13 | 13 | 10 | 11 | 7 | 0.816537 | 99/100 | Mersenne Twister |
| 10 | 14 | 11 | 4 | 16 | 10 | 8 | 8 | 11 | 8 | 0.334538 | 98/100 | CryptGenRandom |
| 7 | 16 | 7 | 9 | 11 | 11 | 8 | 9 | 12 | 10 | 0.678686 | 100/100 | Blum-Blum-Schub |
| 10 | 6 | 11 | 8 | 9 | 14 | 11 | 10 | 12 | 9 | 0.883171 | 100/100 | Micali-Schnorr |
| 65 | 21 | 5 | 4 | 2 | 1 | 1 | 0 | 1 | 0 | 0.000000 * | 67/100 * | Modular Exponentiation |
| 6 | 15 | 8 | 9 | 11 | 7 | 10 | 11 | 13 | 10 | 0.678686 | 99/100 | Linear Congruential |
| 71 | 10 | 3 | 6 | 1 | 3 | 4 | 0 | 1 | 1 | 0.000000 * | 55/100 * | Quadratic Congruential 1 |
| 63 | 12 | 7 | 7 | 2 | 3 | 1 | 1 | 2 | 2 | 0.000000 * | 74/100 * | Quadratic Congruential 2 |

Table A.1: Frequency results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 11 | 12 | 8 | 9 | 11 | 12 | 7 | 11 | 0.978072 | 100/100 | Promestra Security |
| 6 | 8 | 9 | 12 | 11 | 9 | 8 | 16 | 10 | 11 | 0.657933 | 100/100 | Random.org |
| 8 | 10 | 15 | 10 | 14 | 9 | 10 | 8 | 11 | 5 | 0.574903 | 99/100 | Mersenne Twister |
| 9 | 7 | 10 | 9 | 13 | 10 | 14 | 10 | 12 | 6 | 0.779188 | 100/100 | CryptGenRandom |
| 11 | 7 | 14 | 12 | 9 | 6 | 10 | 8 | 9 | 14 | 0.657933 | 100/100 | Blum-Blum-Schub |
| 8 | 11 | 8 | 6 | 10 | 8 | 15 | 9 | 11 | 14 | 0.616305 | 100/100 | Micali-Schnorr |
| 10 | 6 | 10 | 10 | 9 | 17 | 10 | 6 | 18 | 4 | 0.032923 | 98/100 | Modular Exponentiation |
| 11 | 7 | 11 | 6 | 10 | 13 | 15 | 13 | 6 | 8 | 0.437274 | 99/100 | Linear-Congruential |
| 8 | 5 | 9 | 13 | 11 | 11 | 16 | 17 | 7 | 3 | 0.030806 | 99/100 | Quadratic Congruential 1 |
| 7 | 7 | 14 | 9 | 11 | 6 | 12 | 12 | 11 | 11 | 0.719747 | 99/100 | Quadratic Congruential 2 |

Table A.2: Block Frequency results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 11 | 3 | 10 | 7 | 13 | 15 | 8 | 12 | 14 | 0.181557 | 100/100 | Promestra Security |
| 14 | 9 | 6 | 6 | 9 | 10 | 12 | 8 | 10 | 16 | 0.401199 | 99/100 | Random.org |
| 11 | 5 | 7 | 6 | 10 | 12 | 20 | 14 | 6 | 9 | 0.026948 | 100/100 | Mersenne Twister |
| 13 | 12 | 12 | 8 | 10 | 11 | 7 | 9 | 9 | 9 | 0.946308 | 98/100 | CryptGenRandom |
| 10 | 12 | 10 | 12 | 12 | 7 | 5 | 13 | 10 | 9 | 0.779188 | 100/100 | Blum-Blum-Schub |
| 7 | 10 | 11 | 7 | 12 | 8 | 9 | 15 | 7 | 14 | 0.554420 | 100/100 | Micali-Schnorr |
| 66 | 15 | 9 | 3 | 1 | 2 | 1 | 1 | 0 | 2 | 0.000000 * | 72/100 * | Modular Exponentiation |
| 6 | 12 | 17 | 10 | 3 | 11 | 10 | 15 | 9 | 7 | 0.080519 | 99/100 | Linear Congruential |
| 71 | 7 | 8 | 4 | 2 | 2 | 1 | 1 | 3 | 1 | 0.000000 * | 61/100 * | Quadratic Congruential 1 |
| 63 | 13 | 6 | 2 | 5 | 5 | 3 | 2 | 0 | 1 | 0.000000 * | 76/100 * | Quadratic Congruential 2 |

Table A.3: Cumulative Sums (Forward) Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 9 | 9 | 12 | 5 | 6 | 9 | 13 | 19 | 10 | 0.115387 | 100/100 | Promestra Security |
| 9 | 12 | 9 | 5 | 11 | 11 | 13 | 7 | 7 | 16 | 0.383827 | 99/100 | Random.org |
| 10 | 6 | 16 | 12 | 5 | 10 | 7 | 15 | 11 | 8 | 0.213309 | 99/100 | Mersenne Twister |
| 10 | 13 | 14 | 8 | 6 | 7 | 6 | 16 | 13 | 7 | 0.191687 | 96/100 | CryptGenRandom |
| 12 | 9 | 9 | 8 | 15 | 11 | 7 | 13 | 10 | 6 | 0.637119 | 100/100 | Blum-Blum-Schub |
| 9 | 7 | 8 | 9 | 8 | 12 | 16 | 16 | 7 | 8 | 0.289667 | 100/100 | Micali-Schnorr |
| 62 | 15 | 13 | 2 | 1 | 2 | 1 | 2 | 0 | 2 | 0.000000 * | 69/100 * | Modular Exponentiation |
| 7 | 13 | 11 | 10 | 9 | 10 | 11 | 13 | 11 | 5 | 0.779188 | 99/100 | Linear Congruential |
| 69 | 13 | 6 | 0 | 3 | 2 | 4 | 3 | 0 | 0 | 0.000000 * | 62/100 * | Quadratic Congruential 1 |
| 59 | 14 | 9 | 1 | 7 | 3 | 3 | 2 | 2 | 0 | 0.000000 * | 72/100 * | Quadratic Congruential 2 |

Table A.4: Cumulative Sums (Reverse) Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 6 | 11 | 14 | 11 | 6 | 15 | 10 | 6 | 9 | 0.383827 | 99/100 | Promestra Security |
| 10 | 11 | 9 | 10 | 12 | 6 | 10 | 14 | 9 | 9 | 0.911413 | 98/100 | Random.org |
| 7 | 8 | 10 | 10 | 12 | 10 | 19 | 5 | 6 | 13 | 0.096578 | 100/100 | Mersenne Twister |
| 13 | 10 | 12 | 7 | 15 | 11 | 13 | 4 | 7 | 8 | 0.304126 | 99/100 | CryptGenRandom |
| 11 | 10 | 12 | 8 | 12 | 13 | 7 | 12 | 6 | 9 | 0.816537 | 98/100 | Blum-Blum-Schub |
| 13 | 11 | 7 | 9 | 15 | 8 | 8 | 10 | 9 | 10 | 0.798139 | 100/100 | Micali-Schnorr |
| 10 | 11 | 8 | 9 | 6 | 8 | 12 | 11 | 14 | 11 | 0.851383 | 96/100 | Modular Exponentiation |
| 12 | 10 | 11 | 8 | 9 | 8 | 14 | 11 | 9 | 8 | 0.935716 | 97/100 | Linear Congruential |
| 14 | 8 | 13 | 10 | 7 | 8 | 14 | 9 | 9 | 8 | 0.699313 | 91/100 * | Quadratic Congruential 1 |
| 9 | 8 | 9 | 14 | 12 | 10 | 3 | 11 | 9 | 15 | 0.334538 | 96/100 | Quadratic Congruential 2 |

Table A.5: Runs Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 9 | 11 | 8 | 12 | 8 | 8 | 7 | 15 | 7 | 0.474986 | 100/100 | Promestra Security |
| 9 | 15 | 12 | 10 | 13 | 7 | 12 | 7 | 9 | 6 | 0.554420 | 98/100 | Random.org |
| 5 | 10 | 6 | 11 | 11 | 16 | 5 | 9 | 13 | 14 | 0.162606 | 99/100 | Mersenne Twister |
| 10 | 8 | 6 | 13 | 14 | 4 | 9 | 14 | 11 | 11 | 0.350485 | 99/100 | CryptGenRandom |
| 10 | 9 | 12 | 15 | 9 | 9 | 7 | 14 | 6 | 9 | 0.595549 | 100/100 | Blum-Blum-Schub |
| 11 | 15 | 6 | 9 | 10 | 10 | 11 | 4 | 8 | 16 | 0.213309 | 100/100 | Micali-Schnorr |
| 8 | 10 | 6 | 7 | 15 | 10 | 9 | 14 | 9 | 12 | 0.574903 | 99/100 | Modular Exponentiation |
| 8 | 8 | 10 | 18 | 7 | 9 | 13 | 10 | 11 | 6 | 0.289667 | 100/100 | Linear Congruential |
| 9 | 12 | 9 | 9 | 9 | 12 | 11 | 12 | 9 | 8 | 0.987896 | 98/100 | Quadratic Congruential 1 |
| 10 | 9 | 7 | 11 | 7 | 10 | 11 | 10 | 13 | 12 | 0.946308 | 99/100 | Quadratic Congruential 2 |

Table A.6: Longest Run Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 9 | 12 | 6 | 11 | 15 | 10 | 9 | 10 | 13 | 0.514124 | 99/100 | Promestra Security |
| 9 | 9 | 10 | 11 | 12 | 7 | 10 | 12 | 10 | 10 | 0.991468 | 98/100 | Random.org |
| 8 | 13 | 8 | 7 | 12 | 12 | 12 | 12 | 8 | 8 | 0.834308 | 99/100 | Mersenne Twister |
| 10 | 15 | 9 | 15 | 13 | 3 | 7 | 13 | 10 | 5 | 0.085587 | 100/100 | CryptGenRandom |
| 10 | 9 | 13 | 10 | 12 | 12 | 5 | 10 | 9 | 10 | 0.883171 | 100/100 | Blum-Blum-Schub |
| 8 | 15 | 11 | 9 | 10 | 5 | 12 | 11 | 8 | 11 | 0.678686 | 99/100 | Micali-Schnorr |
| 8 | 7 | 12 | 10 | 13 | 8 | 9 | 8 | 11 | 14 | 0.816537 | 97/100 | Modular Exponentiation |
| 12 | 9 | 9 | 8 | 8 | 7 | 18 | 10 | 10 | 9 | 0.455937 | 99/100 | Linear Congruential |
| 13 | 5 | 11 | 13 | 11 | 7 | 11 | 11 | 7 | 11 | 0.678686 | 99/100 | Quadratic Congruential 1 |
| 7 | 12 | 12 | 12 | 5 | 13 | 10 | 5 | 12 | 12 | 0.455937 | 100/100 | Quadratic Congruential 2 |

Table A.7: Rank Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 8 | 13 | 13 | 8 | 12 | 8 | 10 | 9 | 7 | 0.851383 | 100/100 | Promestra Security |
| 11 | 10 | 9 | 12 | 10 | 4 | 10 | 9 | 13 | 12 | 0.779188 | 97/100 | Random.org |
| 9 | 16 | 12 | 7 | 11 | 8 | 12 | 8 | 8 | 9 | 0.657933 | 99/100 | Mersenne Twister |
| 6 | 9 | 15 | 8 | 8 | 12 | 9 | 10 | 10 | 13 | 0.699313 | 100/100 | CryptGenRandom |
| 14 | 9 | 12 | 9 | 10 | 10 | 12 | 9 | 6 | 9 | 0.883171 | 99/100 | Blum-Blum-Schub |
| 11 | 9 | 14 | 9 | 9 | 11 | 8 | 10 | 11 | 8 | 0.964295 | 98/100 | Micali-Schnorr |
| 8 | 10 | 14 | 12 | 10 | 12 | 13 | 7 | 8 | 6 | 0.678686 | 100/100 | Modular Exponentiation |
| 5 | 11 | 12 | 11 | 13 | 5 | 15 | 13 | 8 | 7 | 0.262249 | 100/100 | Linear Congruential |
| 17 | 8 | 18 | 15 | 8 | 4 | 8 | 9 | 2 | 11 | 0.002758 | 96/100 | Quadratic Congruential 1 |
| 33 | 17 | 14 | 8 | 4 | 4 | 7 | 6 | 5 | 2 | 0.000000 * | 94/100 * | Quadratic Congruential 2 |

Table A.8: FFT Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 10 | 9 | 6 | 10 | 8 | 11 | 14 | 7 | 13 | 0.739918 | 100/100 | Promestra Security |
| 8 | 7 | 9 | 13 | 17 | 8 | 13 | 10 | 5 | 10 | 0.275709 | 100/100 | Random.org |
| 15 | 9 | 9 | 4 | 10 | 9 | 10 | 15 | 10 | 9 | 0.437274 | 98/100 | Mersenne Twister |
| 14 | 7 | 9 | 7 | 10 | 12 | 7 | 8 | 16 | 10 | 0.455937 | 100/100 | CryptGenRandom |
| 10 | 10 | 9 | 4 | 19 | 7 | 10 | 11 | 11 | 9 | 0.162606 | 99/100 | Blum-Blum-Schub |
| 11 | 7 | 12 | 7 | 14 | 9 | 11 | 7 | 13 | 9 | 0.739918 | 100/100 | Micali-Schnorr |
| 13 | 11 | 11 | 10 | 8 | 4 | 10 | 8 | 11 | 14 | 0.616305 | 99/100 | Modular Exponentiation |
| 13 | 13 | 10 | 9 | 12 | 7 | 11 | 9 | 8 | 8 | 0.897763 | 99/100 | Linear Congruential |
| 10 | 10 | 13 | 5 | 7 | 9 | 5 | 10 | 12 | 19 | 0.080519 | 99/100 | Quadratic Congruential 1 |
| 12 | 14 | 4 | 7 | 15 | 12 | 9 | 6 | 15 | 6 | 0.085587 | 99/100 | Quadratic Congruential 2 |

Table A.9: Non-Overlapping #1 Template Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 8 | 10 | 10 | 10 | 7 | 7 | 7 | 13 | 7 | 0.048716 | 95/100 | Promestra Security |
| 9 | 7 | 11 | 7 | 6 | 16 | 12 | 18 | 4 | 10 | 0.040108 | 97/100 | Random.org |
| 15 | 16 | 11 | 2 | 9 | 6 | 8 | 11 | 14 | 8 | 0.051942 | 96/100 | Mersenne Twister |
| 9 | 15 | 1 | 9 | 7 | 16 | 14 | 14 | 4 | 11 | 0.008266 | 97/100 | CryptGenRandom |
| 16 | 7 | 6 | 13 | 4 | 13 | 9 | 13 | 7 | 12 | 0.129620 | 96/100 | Blum-Blum-Schub |
| 7 | 7 | 15 | 8 | 14 | 11 | 14 | 10 | 4 | 10 | 0.236810 | 96/100 | Micali-Schnorr |
| 14 | 8 | 11 | 15 | 11 | 13 | 8 | 10 | 6 | 4 | 0.262249 | 94/100 | Modular Exponentiation |
| 15 | 7 | 9 | 12 | 14 | 8 | 11 | 10 | 7 | 7 | 0.554420 | 96/100 | Linear Congruential |
| 15 | 15 | 4 | 11 | 11 | 10 | 12 | 4 | 5 | 13 | 0.062821 | 94/100 | Quadratic Congruential 1 |
| 15 | 10 | 10 | 6 | 7 | 9 | 12 | 16 | 10 | 5 | 0.236810 | 96/100 | Quadratic Congruential 2 |

Table A.10: Non-Overlapping Template worst Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 10 | 9 | 12 | 10 | 11 | 7 | 10 | 12 | 0.991468 | 100/100 | Promestra Security |
| 9 | 10 | 13 | 10 | 11 | 10 | 11 | 11 | 8 | 7 | 0.978072 | 100/100 | Random.org |
| 11 | 9 | 11 | 12 | 11 | 9 | 9 | 7 | 10 | 11 | 0.991468 | 100/100 | Mersenne Twister |
| 11 | 12 | 9 | 9 | 10 | 10 | 10 | 11 | 11 | 7 | 0.994250 | 100/100 | CryptGenRandom |
| 12 | 10 | 8 | 9 | 10 | 10 | 11 | 10 | 9 | 11 | 0.998821 | 100/100 | Blum-Blum-Schub |
| 12 | 9 | 7 | 9 | 10 | 10 | 11 | 10 | 10 | 12 | 0.991468 | 100/100 | Micali-Schnorr |
| 8 | 10 | 12 | 11 | 13 | 8 | 11 | 7 | 10 | 10 | 0.955835 | 100/100 | Modular Exponentiation |
| 11 | 11 | 8 | 10 | 10 | 12 | 8 | 11 | 10 | 9 | 0.996335 | 100/100 | Linear Congruential |
| 9 | 7 | 12 | 14 | 10 | 8 | 10 | 10 | 10 | 10 | 0.946308 | 100/100 | Quadratic Congruential 1 |
| 11 | 12 | 9 | 9 | 10 | 8 | 11 | 9 | 10 | 11 | 0.997823 | 100/100 | Quadratic Congruential 2 |

Table A.11: Non-Overlapping Template best Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 13 | 12 | 11 | 6 | 7 | 8 | 16 | 12 | 7 | 0.383827 | 99/100 | Promestra Security |
| 6 | 9 | 13 | 14 | 10 | 7 | 10 | 10 | 10 | 11 | 0.816537 | 99/100 | Random.org |
| 7 | 12 | 10 | 10 | 13 | 19 | 10 | 5 | 8 | 6 | 0.096578 | 99/100 | Mersenne Twister |
| 12 | 7 | 9 | 6 | 7 | 14 | 11 | 12 | 10 | 12 | 0.699313 | 99/100 | CryptGenRandom |
| 7 | 7 | 9 | 10 | 11 | 11 | 12 | 8 | 14 | 11 | 0.867692 | 98/100 | Blum-Blum-Schub |
| 8 | 9 | 10 | 7 | 17 | 16 | 7 | 9 | 9 | 8 | 0.249284 | 99/100 | Micali-Schnorr |
| 12 | 8 | 9 | 11 | 7 | 15 | 6 | 3 | 16 | 13 | 0.080519 | 100/100 | Modular Exponentiation |
| 12 | 16 | 13 | 10 | 10 | 8 | 7 | 10 | 7 | 7 | 0.534146 | 99/100 | Linear Congruential |
| 8 | 11 | 16 | 9 | 8 | 11 | 6 | 15 | 7 | 9 | 0.366918 | 99/100 | Quadratic Congruential 1 |
| 5 | 12 | 13 | 11 | 7 | 12 | 11 | 10 | 9 | 10 | 0.798139 | 100/100 | Quadratic Congruential 2 |

Table A.12: Overlapping Template Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 10 | 16 | 9 | 7 | 6 | 7 | 18 | 16 | 4 | 0.010237 | 100/100 | Promestra Security |
| 9 | 10 | 14 | 8 | 11 | 7 | 10 | 8 | 12 | 11 | 0.911413 | 96/100 | Random.org |
| 11 | 11 | 4 | 14 | 8 | 12 | 11 | 6 | 8 | 15 | 0.289667 | 99/100 | Mersenne Twister |
| 9 | 8 | 11 | 11 | 13 | 10 | 12 | 13 | 5 | 8 | 0.759756 | 99/100 | CryptGenRandom |
| 9 | 5 | 6 | 12 | 9 | 18 | 18 | 5 | 7 | 11 | 0.012650 | 100/100 | Blum-Blum-Schub |
| 12 | 11 | 10 | 8 | 12 | 11 | 12 | 12 | 8 | 4 | 0.719747 | 99/100 | Micali-Schnorr |
| 6 | 17 | 14 | 10 | 11 | 6 | 11 | 12 | 4 | 9 | 0.122325 | 100/100 | Modular Exponentiation |
| 8 | 9 | 8 | 9 | 8 | 12 | 9 | 15 | 15 | 7 | 0.554420 | 99/100 | Linear Congruential |
| 10 | 6 | 12 | 7 | 16 | 8 | 9 | 10 | 12 | 10 | 0.595549 | 99/100 | Quadratic Congruential 1 |
| 18 | 11 | 8 | 10 | 8 | 9 | 10 | 4 | 12 | 10 | 0.249284 | 98/100 | Quadratic Congruential 2 |

Table A.13: Universal Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 12 | 12 | 5 | 17 | 7 | 6 | 11 | 4 | 14 | 0.058984 | 98/100 | Promestra Security |
| 11 | 13 | 14 | 9 | 11 | 4 | 10 | 8 | 7 | 13 | 0.474986 | 99/100 | Random.org |
| 12 | 6 | 8 | 9 | 9 | 7 | 15 | 11 | 10 | 13 | 0.637119 | 98/100 | Mersenne Twister |
| 7 | 15 | 9 | 15 | 9 | 13 | 7 | 9 | 8 | 8 | 0.455937 | 100/100 | CryptGenRandom |
| 16 | 18 | 7 | 6 | 8 | 10 | 6 | 7 | 11 | 11 | 0.075719 | 98/100 | Blum-Blum-Schub |
| 14 | 9 | 11 | 8 | 11 | 11 | 9 | 10 | 8 | 9 | 0.964295 | 100/100 | Micali-Schnorr |
| 9 | 14 | 13 | 10 | 14 | 13 | 8 | 8 | 3 | 8 | 0.262249 | 100/100 | Modular Exponentiation |
| 10 | 9 | 8 | 14 | 14 | 14 | 7 | 9 | 5 | 10 | 0.455937 | 99/100 | Linear Congruential |
| 15 | 12 | 11 | 8 | 12 | 13 | 7 | 9 | 5 | 8 | 0.474986 | 99/100 | Quadratic Congruential 1 |
| 8 | 9 | 11 | 10 | 10 | 10 | 5 | 14 | 8 | 15 | 0.574903 | 99/100 | Quadratic Congruential 2 |

Table A.14: Approximate Entropy Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 11 | 9 | 5 | 11 | 4 | 8 | 5 | 8 | 3 | 0.128379 | 68/69 | Promestra Security |
| 5 | 4 | 6 | 8 | 6 | 9 | 3 | 7 | 9 | 7 | 0.706149 | 62/64 | Random.org |
| 8 | 8 | 8 | 4 | 7 | 6 | 4 | 7 | 2 | 8 | 0.637119 | 61/62 | Mersenne Twister |
| 6 | 6 | 6 | 3 | 2 | 1 | 11 | 8 | 3 | 8 | 0.035174 | 53/54 | CryptGenRandom |
| 10 | 3 | 1 | 7 | 8 | 6 | 4 | 4 | 7 | 5 | 0.162606 | 50/55 * | Blum-Blum-Schub |
| 7 | 6 | 5 | 7 | 3 | 7 | 15 | 10 | 1 | 6 | 0.007422 | 65/67 | Micali-Schnorr |
| 4 | 7 | 3 | 2 | 2 | 4 | 5 | 3 | 2 | 4 | 0.468595 | 36/36 | Modular Exponentiation |
| 3 | 7 | 5 | 4 | 2 | 9 | 4 | 8 | 3 | 11 | 0.051942 | 56/56 | Linear Congruential |
| 4 | 6 | 3 | 2 | 7 | 2 | 1 | 2 | 4 | 4 | 0.232760 | 32/35 | Quadratic Congruential 1 |
| 6 | 0 | 2 | 5 | 2 | 5 | 2 | 2 | 2 | 4 | 0.299251 | 30/30 | Quadratic Congruential 2 |

Table A.15: Random Excursions (+1) Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 10 | 4 | 7 | 5 | 8 | 5 | 9 | 5 | 7 | 0.551026 | 69/69 | Promestra Security |
| 4 | 8 | 8 | 4 | 11 | 4 | 4 | 7 | 7 | 7 | 0.468595 | 64/64 | Random.org |
| 7 | 6 | 8 | 3 | 7 | 8 | 8 | 4 | 8 | 3 | 0.671779 | 62/62 | Mersenne Twister |
| 4 | 8 | 1 | 1 | 6 | 11 | 5 | 4 | 9 | 5 | 0.023545 | 54/54 | CryptGenRandom |
| 8 | 8 | 3 | 1 | 5 | 8 | 7 | 7 | 5 | 3 | 0.224821 | 53/55 | Blum-Blum-Schub |
| 3 | 8 | 13 | 5 | 7 | 5 | 6 | 8 | 2 | 10 | 0.051391 | 66/67 | Micali-Schnorr |
| 2 | 4 | 2 | 4 | 4 | 6 | 6 | 2 | 2 | 4 | 0.468595 | 35/36 | Modular Exponentiation |
| 4 | 2 | 11 | 5 | 5 | 7 | 4 | 6 | 6 | 6 | 0.289667 | 56/56 | Linear Congruential |
| 3 | 5 | 5 | 6 | 4 | 4 | 4 | 2 | 1 | 1 | 0.378138 | 35/35 | Quadratic Congruential 1 |
| 4 | 2 | 1 | 5 | 1 | 4 | 4 | 2 | 4 | 3 | 0.739918 | 30/30 | Quadratic Congruential 2 |

Table A.16: Random Excursions Variant (-1) Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 14 | 12 | 10 | 9 | 4 | 16 | 4 | 11 | 10 | 0.162606 | 99/100 | Promestra Security |
| 12 | 7 | 11 | 9 | 8 | 18 | 11 | 3 | 12 | 9 | 0.129620 | 98/100 | Random.org |
| 5 | 6 | 12 | 8 | 17 | 14 | 7 | 11 | 10 | 10 | 0.191687 | 100/100 | Mersenne Twister |
| 13 | 8 | 6 | 4 | 12 | 9 | 13 | 18 | 9 | 8 | 0.096578 | 97/100 | CryptGenRandom |
| 9 | 13 | 14 | 8 | 8 | 13 | 10 | 3 | 11 | 11 | 0.401199 | 100/100 | Blum-Blum-Schub |
| 8 | 15 | 13 | 9 | 8 | 6 | 7 | 15 | 6 | 13 | 0.224821 | 99/100 | Micali-Schnorr |
| 13 | 18 | 5 | 8 | 11 | 9 | 8 | 12 | 8 | 8 | 0.213309 | 99/100 | Modular Exponentiation |
| 7 | 8 | 11 | 17 | 12 | 9 | 14 | 6 | 9 | 7 | 0.275709 | 100/100 | Linear Congruential |
| 4 | 11 | 16 | 11 | 12 | 4 | 9 | 13 | 10 | 10 | 0.191687 | 100/100 | Quadratic Congruential 1 |
| 8 | 12 | 12 | 5 | 8 | 9 | 9 | 15 | 7 | 15 | 0.334538 | 98/100 | Quadratic Congruential 2 |

Table A.17: Serial Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 9 | 13 | 13 | 4 | 8 | 10 | 8 | 10 | 11 | 0.534146 | 99/100 | Promestra Security |
| 14 | 9 | 12 | 8 | 10 | 9 | 8 | 9 | 4 | 17 | 0.236810 | 99/100 | Random.org |
| 8 | 12 | 10 | 13 | 10 | 3 | 15 | 12 | 11 | 6 | 0.262249 | 100/100 | Mersenne Twister |
| 15 | 6 | 4 | 6 | 11 | 13 | 11 | 11 | 11 | 12 | 0.275709 | 96/100 | CryptGenRandom |
| 9 | 15 | 7 | 14 | 10 | 8 | 8 | 6 | 10 | 13 | 0.494392 | 99/100 | Blum-Blum-Schub |
| 8 | 6 | 10 | 12 | 6 | 10 | 12 | 14 | 10 | 12 | 0.699313 | 99/100 | Micali-Schnorr |
| 17 | 5 | 12 | 15 | 9 | 10 | 3 | 5 | 12 | 12 | 0.028817 | 97/100 | Modular Exponentiation |
| 13 | 8 | 7 | 9 | 17 | 8 | 14 | 8 | 9 | 7 | 0.304126 | 99/100 | Linear Congruential |
| 8 | 10 | 8 | 7 | 11 | 14 | 18 | 5 | 9 | 10 | 0.191687 | 99/100 | Quadratic Congruential 1 |
| 6 | 10 | 6 | 14 | 12 | 6 | 15 | 8 | 10 | 13 | 0.304126 | 99/100 | Quadratic Congruential 2 |

Table A.18: $2_{nd}$ Serial Results

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-Value | Proportion | Random number generator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 12 | 22 | 12 | 5 | 11 | 10 | 7 | 6 | 5 | 0.006661 | 98/100 | Promestra Security |
| 9 | 11 | 13 | 10 | 12 | 7 | 9 | 10 | 12 | 7 | 0.924076 | 100/100 | Random.org |
| 6 | 13 | 11 | 8 | 12 | 13 | 6 | 10 | 10 | 11 | 0.739918 | 100/100 | Mersenne Twister |
| 12 | 8 | 10 | 10 | 14 | 8 | 7 | 8 | 9 | 14 | 0.759756 | 98/100 | CryptGenRandom |
| 9 | 12 | 14 | 7 | 7 | 8 | 9 | 9 | 10 | 15 | 0.637119 | 100/100 | Blum-Blum-Schub |
| 13 | 6 | 15 | 15 | 8 | 10 | 10 | 5 | 9 | 9 | 0.304126 | 98/100 | Micali-Schnorr |
| 8 | 10 | 11 | 16 | 5 | 5 | 10 | 13 | 8 | 14 | 0.213309 | 99/100 | Modular Exponentiation |
| 9 | 12 | 6 | 10 | 15 | 13 | 7 | 11 | 9 | 8 | 0.637119 | 99/100 | Linear Congruential |
| 10 | 8 | 8 | 5 | 9 | 13 | 10 | 13 | 9 | 15 | 0.554420 | 99/100 | Quadratic Congruential 1 |
| 16 | 15 | 9 | 11 | 9 | 11 | 6 | 7 | 8 | 8 | 0.366918 | 98/100 | Quadratic Congruential 2 |

Table A.19: Linear Complexity Results


Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden