Development and Testing of a Hardware Random Number Generator

Lidiia Lazebnikova

"People believe the only alternative to randomness is intelligent design." – Richard Dawkins

"The only alternative to randomness is bad randomness." – Me

1. Introduction

Almost everyone has a vague idea of what a random number is. It is essentially a number that cannot be reproduced by an obvious rule, e.g. if I asked you to generate a number in your head and you would use your mother's birthday multiplied by the number of your former pets, it would make a good random number because that information is not available to me. However, if I am holding up three fingers while asking you to perform the exercise and coincidentally, the number you come up with is, in fact, three, nothing about this interaction had been random.
Computers work in a similar way. Consider these two series of binary digits, each one representing one bit of information:

01010101010101010101
01101100110111100010

The first sequence seems neat to a casual observer and the rule for its creation is simple: it is the sequence "01" repeated ten times. If asked how the series might continue, anyone could surmise that the next two digits would be zero and one. The second sequence does not allow for a similar comprehensive pattern, the arrangement seems hap-hazard and the sequence appears to be a random assortment of zeroes and ones.
It was created by something as trivial as flipping a coin, the sides of which are assigned with binary values.
A random binary sequence could be represented by the flips of a "fair coin" (the probability of heads or tails is exactly the same) with sides that are assigned the values of "O" and "1". Each turn has to be independent from another. If these conditions are met, the coin is the perfect random bit stream generator. [1]
Humans tend to be burdened by thought processes that lead to specific conclusions, while the coin is not, thus making it, despite its imperfections, a better entropy source – a controlled chaotic environment that creates a steady stream of random values.
Let us return to my request to think of a random number, this time ranging between 0 and 10. If the number you thought of is seven, it is perfectly predictable using statistics. Seven is one more than the uncomfortable six and one less than the disconcerting eight, making it just right, or so it seems, for most people. However, if you did use an algorithm of your own choosing I did not think or know about, such as the number of hours you have exercised this week, the result would surprise me, making the outcome not guessable. This property is called **independence** — no part of the sequence can be inferred from others parts of the sequence.
But suppose I have asked a lot in the past and my memory is excellent. Exercise, after all, is a habit. If I discover a pattern with these numbers, the probability that I might get the next one right rises. Suppose, that you do not like exercise and each time the

output is zero. That would make for terrible randomness, even if I do not know the function that creates it. The property of randomness I am implying here is **uniformity** — the frequency of occurrence of 0s and 1s has to be approximately equal.

1.1. Utilization of random numbers

In the following chapter, I explore:

- Uses outside of cryptography (gambling, simulations, Monte Carlo)
- Uses in cryptography and why
- Encryption
- Digital signatures
- One-time pad
- Authentication systems
- TLS
- Nonces and their use

The question is, why do we need random numbers?
How is unpredictable input valuable for our day-to-day life?
Intuition's first notion is **gambling**. Games of chance live off unpredictability, especially when it is not "hackable". Counting cards, meddling with slot-machines and exposing underlying algorithms of online poker are not illegal, yet highly frowned upon and cost the gambling industry millions. Sometimes casinos resort to violence in order to prevent such acts of "unfair use". The distrust is mutual, many gamblers are convinced that the slot machines are rigged in favor of the house and they may be right.

A system that generates reliable randomness and is resilient against attackers would redefine the word "chance" and renew the trust, should both parties be informed of it, of course.
In the natural sciences, randomness helps simulate volatile physical processes, like virus epidemics, beta decay of nuclei and insect reproduction. Simulation experiments often extend to trying to predict the political landscape in any given country. [1]

Theorists also use randomness in order to solve problems that are considered either too difficult or impossible to solve using standard methods. The tool set they use is called the **Monte Carlo method** or **Monte Carlo experiments.** There are three main problem classes that rely on Monte Carlo: optimization, numerical integration and drawing from a probability distribution. The method itself and the problems it helps solving are widely considered incredibly complex. However, it is far less mystifying than one would think. Eventually, it all comes down to four concise steps:
1. Identify the problem and build a model.
2. Define the parameters for each factor of the model.
3. Create random data for every parameter.
4. Simulate and analyze.

There is specialized software for this method and its practical application, one of the examples is the Minitab Statistical Software.

The largest randomness-dependent party is computer science. There is research being done on sorting algorithms that has delivered promising results. After all, before order can be brought into chaos, chaos must first be created.
The most dominant field among those who use random numbers is cryptography. They are so crucial to secure computer systems that without them, any form of secure encryption stops being secure.
These systems offer protection against snooping and spoofing. Snooping means the intercept of information either being exchanged between two participants or stored somewhere, whereas spoofing is communications deception. [2]

Cryptography is a very old subset of computer science, thought to be originated in the time of war. Until the late eighties, because cryptographic devices were expensive, they were almost only used in the military, the governments and some financial transactions. With the rise of faster CPUs, cryptography began touching software. Nowadays, cryptography is everywhere. The largest volumes of cryptographic devices are for communications such as WhatsApp and Skype, the web, access cards, debit and credit cards, DRM for media copyright and hard disk encryption. Very few of these offer end-to-end confidentiality or open their source code, which makes the systems brittle. However, through massive interception with trailblazing search techniques, intelligence agencies created a world-wide mass surveillance grid and the need for good privacy is dire, despite of business models, which gain from selling user data, exploiting Big Data in a massive way. Cyber security professions now have the responsibility to restore the balance between citizens and the union of governments and corporations. [3]

At the heart of every cryptographic system is the generation of unpredictable (i.e., random) numbers.
Let us start with basic cryptography. Even though it is impractical, ideally, a user should choose a random password every certain period of time. It needs to be a simple character string and if it is reusable, it also needs to be memorable. In this case, the requirement for such a password is only that it is not guessable. Although, for keys of fixed length, such as PINs, you would want the appearance of true randomness, something that could pass statistical randomness tests.

There are generally two types of encryption: symmetric and asymmetric.
Symmetric encryption like one-time pads or the US Advanced Encryption Standard (AES) means that two people or machines communicating privately know the same secret key.

Asymmetric encryption suggests that the keys each person or machine has, comes in pairs. One key is kept private and another is published to everyone who wants it. Knowing the "public key" does not help to find out the one kept private and is also of no use to someone who wants to intercept the communication.

One of the simpler asymmetric algorithms is RSA and it does need random data to create every pair of keys. However, any number of messages can be encrypted with the same key, so the need for randomness is less acute here.

In a situation where a key was used, an attacker would have to guess the key through trial and error. Thus the probability of them coming up with the right key should be acceptably low. This probability depends on the application.

The algorithm proposed by the US National Institute of Standards and Technology needs good random numbers for each signature in their Digital Signature Standard (DSS) [4].
Digital signatures notify the receiver that the message was written by none other than the sender.

Encrypting with a one-time pad, which is theoretically the strongest technique in encryption, needs equal randomness for each message. A plaintext is paired with a secret key completely comprised of randomness. Then each bit of the plaintext is encrypted by adding the corresponding character in the key to it in a modular way. Now, what needs to be mentioned is, that this type of encryption is far from new. However, it is incredibly effective and thorough.

Despite of that, it is not the standard practice. The reason why is because it costs a lot of resources. In order to be secure, one-time pad needs to have just that – a one-time key, never to be used again. That creates a steady need for randomness that cannot be quite satisfied.

The following example is taken from a fairly recently de-classified document, which consists of a series of lectures given by David G. Boak in 1966 to the interns and employees of the National Security Agency. Boak had participated in the development of U.S. Communications Security for over 20 years. [5]

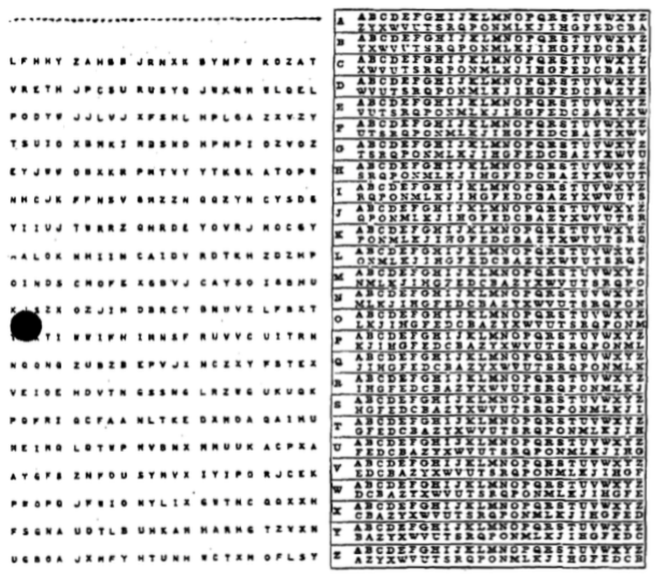


Figure 1: DIANA [5]

One of the oldest types of one-time pads is depicted in Figure 1. Widely used by the NSA in the 70s, DIANA is a cipher system that consists of pages after pages of random numbers or letters. Ever since the one-time pad has undergone a lot improvement. The version after DIANA used by the NSA was ORION and was three times as fast. The system uses carbon paper and two pages: one with the key and one with the cipher, the first over the latter. The key is simply the alphabet repeated in every line in a row. To encipher, you circle each letter (only one per line) on the key and the mark appears on the cipher. However, 100 words encrypted with DIANA turned to 10 and on account of gaining speed, the NSA also had to print more paper. It was efficient for short messages and when no machines were available.

The next type of manual systems that use randomness are authentication systems. They establish that any received communication is genuine and not "spoofed." It is timeless in its application and centuries-old.
It has two phases: challenge and response, where sender and recipient can interrogate each other and establish that both are who they say they are before they would communicate. It is incredibly important in communication where either not each or none at all messages are encrypted.

Any HTTPS session starts the following way: first, the web browser introduces itself to the server. It offers a package that includes information about which version of SSL it wants to use. Then the server responds with a similar package, which includes its SSL certificate. Finally, the web browser checks if the certificate is valid and generates a so-called "pre-master secret". This key secures the following transfers. It is very important that it is unpredictable for the connection to be secure. Every browser generates its own pre-master secret for each session with each server. Considering

that a session only lasts until the user becomes bored and switches to another webpage, there are a lot of sessions.
The pre-master secret is a 48-byte sequence. Of course, the first two bytes are, by convention, the TLS version. It still leaves 46 bytes of randomness, but it is not the end. A Pseudo-Random Function (PRF) is used to generate a different key combining the secret, the ASCII label and the seed data from the Message Authentication Code. This function may be defined, but by our own definition, it is still unpredictable to any attacker and therefore generates even more randomness. The key that results from it is the 48-byte "master secret". [6]
According to statista.com, the rough estimate of unique monthly visitors to amazon.com is a number close to 200 million. Under the assumption that users tend to log in more than once even during one day, the estimated number of TLS handshakes that day is in the millions. This means that for Amazon, somewhere between 1 GB and 10 GB of randomness a day in the US alone are needed. There is a million top sites that use the TLS protocol, assuming they are visited as frequently as Amazon, ten million GB of randomness are needed in 24 hours, that still makes roughly 120 GB per second of world randomness demand.
That reveals the sheer number of random bits that need to be generated every day in order to secure the connection between the user who only intends to purchase something and the server, which is willing to sell. Facebook quantifies this number by a yet unknown factor.
Other authentication protocols also use randomness.

A **nonce** is a number that is only allowed to be used once. The sender generates a string of randomness devoid of information and appends it to a message. Each message gets its own randomness. The receiver recognizes the validity of the message because at each turn, the "randomness signature" is known to them. This prevents something called a **replay attack**, when a signature is extracted from an old message by the attacker, stamped on a falsified message and accepted by the receiver.
Of course, time-stamping can prevent such an attack, but it is not a very secure process because of how transparent it is. [2]

1.2. Problems generating random numbers and getting entropy out of a deterministic computer

This chapter contains:

- Determinism and nondeterminism
- How a computer can contain nondeterminism?
- Seeds and pseudorandom functions

The oldest philosophical problem humanity has dealt with is the notion of **free will** versus **destiny**. When I choose a complex and particular coffee order, is it completely and utterly my decision or was it whispered to me by a combination of a higher power and my body's chemical processes. Some argue the former does not exist, some argue — the latter. Some think that both are the same. Some choose to refrain from talking about it.

Computers do not have free will. According to Microsoft, not yet. They use a sequence of actions that was programmed into them — an **algorithm**. Therefore, in theory, every algorithm inside a computer, provided they use the same input, should follow the **same steps** and return the **same output**. It is called **determinism** and algorithms that match this description are called deterministic algorithms. All computer algorithms are **deterministic**, unless they:

A) use an external kind of input, something that does not repeat itself.
B) are timing-sensitive. Like in the case of multiple processors that access the same data at the same time. If they edit it independently from one another, the order in which they have access will influence the data itself.
Or C) a hardware error changes them unexpectedly.

So how does a computer achieve simulated free will? Mostly A).
Option B is not very convenient. Relying on two processors "crossing swords" would not produce a lot of randomness. Using the time (and/or date) of the starting process would not be non-deterministic because the time never is. Option C is simply dangerous.

In order to practically implement option A, a starting value is used and it is called a **seed**. Since computers are numbers-based machines, this seed is usually a number picked out of a pool of unrelated numbers. Then a deterministic algorithm derives an entire series of *seemingly unrelated* numbers from this **seed**. Because this algorithm is deterministic in nature but produces output that only simulates non-determinism, it is called a **pseudorandom function**. The numbers generated by it are called pseudorandom numbers, because they look random, but are not, as opposed to **true random number**s that are not generated inside a computer, but are merely collected inside one.

The difference between these is that the pseudorandom number sequence would eventually repeat. It is periodic in nature because the assortment of seeds it has is finite. The number of seeds depends on the length of each seed. So for a pseudorandom sequence to be virtually indistinguishable from a true random sequence the seed length would have to be significant.

1.3. Structure of the thesis

This thesis explores the nature of randomness through the scope of statistical properties and explores the battery of tests suggested by the National Institute of Standards and Technology. Furthermore, I look at different physical phenomena that can be used as sources for random numbers and explain the difference between true random and pseudorandom methods of number generation, finishing with hands-on testing of several hardware and software solutions.

2. State of the art

2.1. Requirements and tests

In this chapter, I explore:

- The probabilistic method
- Chi-Squared Test
- Null hypothesis
- Confidence interval
- Critical values
- P-values
- Frequency Test
- Runs Test
- Binary Matrix Rank Test
- Tests with Random Walks
- Data Compression

So how does one evaluate a sequence of random numbers?
By ways of statistics, the means most frequently used is the probabilistic method.
Suppose, we use a different method of random number generation but quite similar to a coin — a set of dice. There are six possible events: rolling a one, two, three, four, five or six. And the probability of achieving each event is the same — 1/6. The distribution of this probability is shown in Figure 2 under "even distribution".

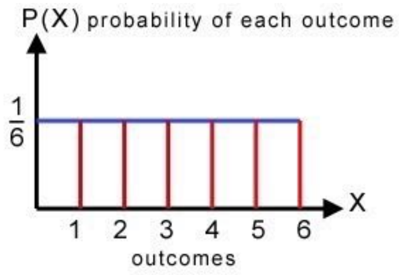


Figure 2: Even distribution [7]

If we have two dice instead of one, everything changes. The number of events is increased. The lowest sum achievable is now two and the highest — twelve. Also seeing that we do not distinguish between the two die, we have many different combinations for the same sum, e.g. the number 7 can be achieved as follows: (1,6), (2,5), (3,4), (4,3), (5,2), (6,1). Now that we have many ways of achieving the 7 with the

same probability, it becomes our most likely event. The distribution would look like a triangle, the value for 7 being at the top.
The more dice we roll simultaneously the more discrete events we have. According to the de-Moivre-Laplace theorem, the triangle becomes a graph presented in Figure 3, when the number of discrete events reaches infinity. This Bell curve distribution is a common name given to a normal distribution. And it is known as "normal" because it is very frequent in nature.

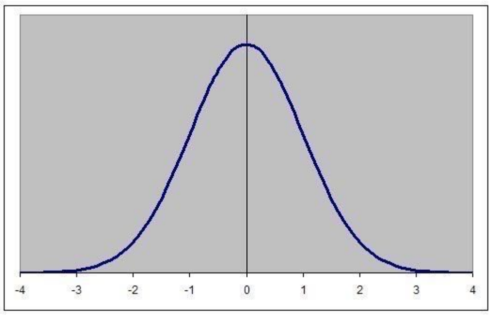


Figure 3: The Bell Curve [8]

Since nature is the number one generator for randomness, the probabilistic method's principle lies in comparing the occurrence of events inside a random sequence to the normal distribution. It is not the most exact method and the accuracy is reduced with the sample size, but it is a good way to determine whether or not the randomness source is flawed.
The test that determines that is known as the Chi-Squared test. It is represented through a simple equation:

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

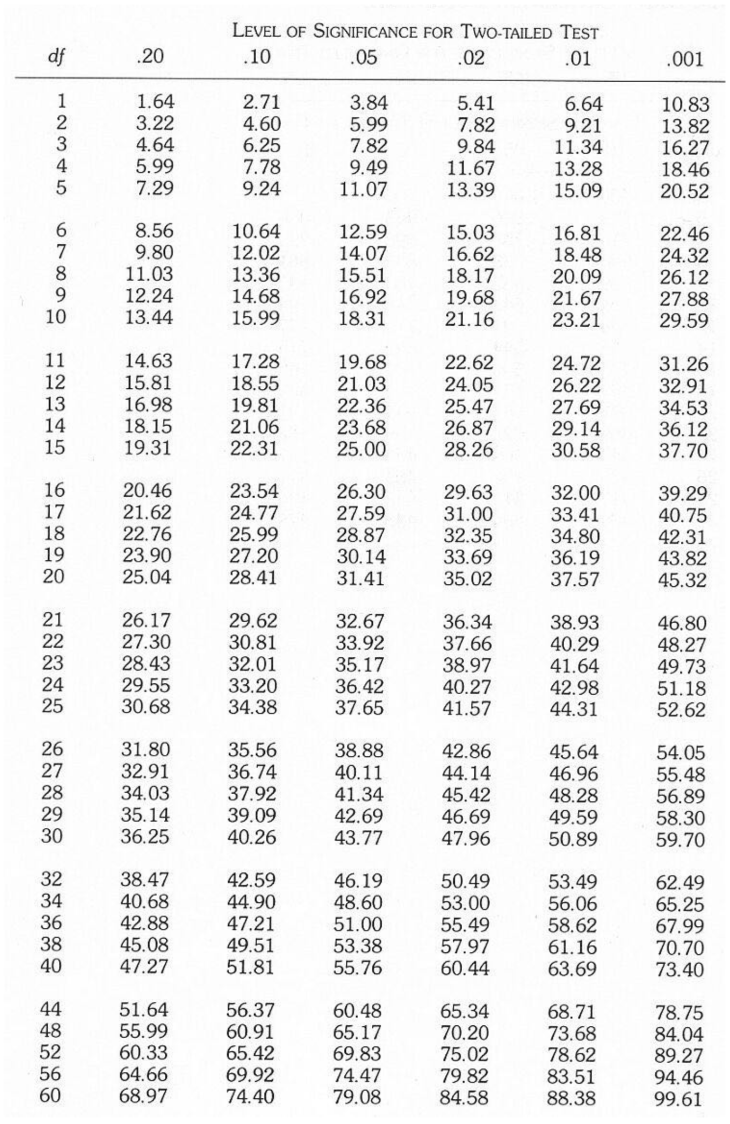
Developed by Karl Pearson in 1900, this test is applied to sets of categorical data and it establishes whether or not the difference between the separate sets arose by chance or if it was influenced by a variable.
The equation is constructed in the following way:

- χ^2 is Pearson's cumulative test statistic. It is the result to be compared to an acceptable value.
- O_i is the number of observations of a certain type i. Essentially, it is the number of occurrences of a certain event that were recorded

- E_i Is the expected number of occurrences of the same event, derived from its probability

Suppose we flip a fair coin a hundred times. There are two possible events in this case — heads and tails. The probability of each occurring is exactly $\frac{1}{2}$. Should I observe that I got tails 38 times, the probability seems to not match the observed data entirely. In order to accept or reject the coin itself, I could use this test to determine if it is skewed.
First, I would use a **null hypothesis**. Due to the simplicity of the given example, this step seems to be trivial, but in multivariable problems, it is not. The null hypothesis is a statement that the test confirms or denies. Usually, it is that there is no significant difference between the observed and expected frequencies of the event. In other words, the coin is fair and I am flipping it correctly.
Before the calculations are made, two significant factors need to be defined — the **degree of freedom** and the **confidence interval**.
In terms of the degree of freedom, one has to have at least two outcomes. In this case, we have exactly two — heads and tails. The degree of freedom is one less than the number of possible outcomes and would equal 1.
The confidence interval, also known as alpha, is a lot simpler to determine, it is simply how sure you wish to be. In case 95% certainty is enough, the confidence interval equals 0.05.
From these two values, using the Student's t-distribution, the **critical value** is calculated. With the confidence interval of 0.05 and one degree of freedom, the value is 3.84, as found in Table 1, which is known as the T table.

Table 1: Student's T-table [9]



	Level of Significance for Two-tailed Test					
df	.20	.10	.05	.02	.01	.001
1	1.64	2.71	3.84	5.41	6.64	10.83
2	3.22	4.60	5.99	7.82	9.21	13.82
3	4.64	6.25	7.82	9.84	11.34	16.27
4	5.99	7.78	9.49	11.67	13.28	18.46
5	7.29	9.24	11.07	13.39	15.09	20.52
6	8.56	10.64	12.59	15.03	16.81	22.46
7	9.80	12.02	14.07	16.62	18.48	24.32
8	11.03	13.36	15.51	18.17	20.09	26.12
9	12.24	14.68	16.92	19.68	21.67	27.88
10	13.44	15.99	18.31	21.16	23.21	29.59
11	14.63	17.28	19.68	22.62	24.72	31.26
12	15.81	18.55	21.03	24.05	26.22	32.91
13	16.98	19.81	22.36	25.47	27.69	34.53
14	18.15	21.06	23.68	26.87	29.14	36.12
15	19.31	22.31	25.00	28.26	30.58	37.70
16	20.46	23.54	26.30	29.63	32.00	39.29
17	21.62	24.77	27.59	31.00	33.41	40.75
18	22.76	25.99	28.87	32.35	34.80	42.31
19	23.90	27.20	30.14	33.69	36.19	43.82
20	25.04	28.41	31.41	35.02	37.57	45.32
21	26.17	29.62	32.67	36.34	38.93	46.80
22	27.30	30.81	33.92	37.66	40.29	48.27
23	28.43	32.01	35.17	38.97	41.64	49.73
24	29.55	33.20	36.42	40.27	42.98	51.18
25	30.68	34.38	37.65	41.57	44.31	52.62
26	31.80	35.56	38.88	42.86	45.64	54.05
27	32.91	36.74	40.11	44.14	46.96	55.48
28	34.03	37.92	41.34	45.42	48.28	56.89
29	35.14	39.09	42.69	46.69	49.59	58.30
30	36.25	40.26	43.77	47.96	50.89	59.70
32	38.47	42.59	46.19	50.49	53.49	62.49
34	40.68	44.90	48.60	53.00	56.06	65.25
36	42.88	47.21	51.00	55.49	58.62	67.99
38	45.08	49.51	53.38	57.97	61.16	70.70
40	47.27	51.81	55.76	60.44	63.69	73.40
44	51.64	56.37	60.48	65.34	68.71	78.75
48	55.99	60.91	65.17	70.20	73.68	84.04
52	60.33	65.42	69.83	75.02	78.62	89.27
56	64.66	69.92	74.47	79.82	83.51	94.46
60	68.97	74.40	79.08	84.58	88.38	99.61

Should the calculated χ^2 be higher than 3.84, we would be forced to reject the null hypothesis. [7]

So 38 heads and 62 tails from 100 coin flips would give χ^2 the value of 5.76 using this calculation:

$$\frac{1}{50}(38-50)^2+\frac{1}{50}(62-50)^2$$

Ergo, the null hypothesis is rejected and there is either something wrong with the coin or the way I flip it and we are 95% sure of it.
In randomness testing, the null hypothesis is that the data is random and the confidence interval is typically 0.05. It does not mean that the statement of the null hypothesis is 95% probable to be true, but it does mean that 95% of the data can be considered random.

An important aspect to know of is the distinction between one- and two-tailed tests. In the table shown above, I have taken the value meant for a two-tailed test and the Chi-Squared test introduced by Pearson examines one-tailed events.
The distinction, while important, only needs to be made once in randomness testing.
The Bell Curve depicted in Figure 3 represents the probability distribution for most natural occurring events. The events far left and far right to the curve are the events least likely to occur. In terms of randomness, these are the events of the given generator either being **perfectly random** or **completely useless**. As we see, testing is not about these binary decisions, but how **probable** it is for one of these **extreme events** to occur with the null hypothesis in mind. Since both of these extreme events are of interest to us, the tests are called two-tailed.
The value for this probability is called the p-value. For random number generators, the p-value is chosen to be the probability that the sequence being tested appears to be perfectly random. P-values range from 0 to 1. [8]
These are the notions with which the NIST statistical suite operates. Developed to select and test random and pseudorandom number generators, this test suite useful in determining whether or not a generator can be used for cryptographic application. It is stressed, however, that no amount of statistical testing can serve as a substitute for **cryptanalysis**. Since sample size is directly proportional to the accuracy, this test suite also demands a large sequence length (of the order of thousands to millions of bits). Some of these tests can be used for smaller values, but it is discouraged for the sake of accuracy.
On the one hand, some of the 15 tests provided by NIST test for **uniformity**. These are the **Frequency Test**, the **Runs Test** and the **Binary Matrix Rank Test**.

- The Frequency (also known as Monobit) Test focuses on the numbers of zeroes and ones for the entire sequence. Test No. 2 applies the same test within M-bit blocks, where the value of M can be changed in the settings. This is a more or less direct application of the Chi-Squared test explained above.
- The Runs Test and its immediate follower — No. 4 — count the runs within a sequence, while No. 4 also divides that sequence into M-bit blocks. A run is an uninterrupted stream of identical bits. It can either be a sequence of just ones, or just zeroes. The number and length of these runs have to match the expectancy within a random sequence. The prerequisite for this test is the Frequency test. The principle is that the value 0 is added when two neighboring bits are the same and the value 1 is added if they are not. The calculated sum is used to determine the p-value.
- The Binary Matrix Rank Test focuses on the rank of matrices in the sequence, it is also a fairly old test that was present in the DIEHARD suite computed by George Marsaglia in 1995. [9]

Testing for uniformity is not difficult. But the most crucial property of random sequences is the lack of patterns, or **uniformity**. Predicting periodic features such as that is more complex. Human beings are great at it and that is why it is so easy for us to tell when something genuinely appears random. For instance, just by looking at a bitmap made up of binary values (black dots representing ones and white dots representing zeroes, or vice versa), we can accurately surmise randomness.

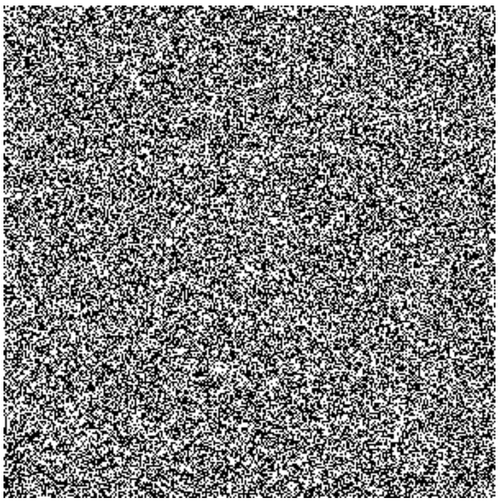


Figure 4: A bitmap generated from randomness that comes from atmospheric noise [10]

Figure 4 shows a random sequence converted into a bitmap, and at first glance, it looks sufficiently random. If I had said that it is a pseudorandom sequence, the more you would look at it, the more patterns you would begin to discover. This demonstrates the problem with the visual method: human beings are incredibly slow at discovering patterns; it can be proven by watching someone do a word search puzzle.
Unfortunately, there are no computer algorithms that can accurately determine what feels random. But fortunately, patterns can be detected using **random walks** — a mathematical way of describing a path that consists of random steps. E.g., a molecule's way through a liquid of a gas can be considered a random walk, or in another example, an animal looking for food. This term has been introduced by the very same Karl Pearson in 1905, who must have accomplished everything in the field of randomness. Often, random walks are assumed to be either Markov chains or Markov processes and sometimes, they are depicted as graphs, the properties of which can be studied using computer algorithms.
Looking at randomness from a purely statistical point of view has its merits. However, it limits you in seeing the purpose of randomness. The desired property of random numbers is that their occurrence is unpredictable, and what better way of assessing it than from an information technology point of view?

Data compression is the process of converting a data stream into another of a smaller size. Today, our life without it is unthinkable. Not anymore because we have limited storage — since the limit to the storage we can have is now only defined by our understanding of it. Data compression has made communication a lot easier. The idea of judging how random something is with it lies in trying to compress it. There are many known methods of data compression. Based on different ideas and suitable for different types of data, they also produce different results. Nonetheless, data compression operates under the same principle — removing redundancy. Any nonrandom data has some structure and it can be used to create a smaller representation of the data to be compressed.

For example, the English language uses the letter 'e' a lot. The letter 'z', on the other hand, not so much. This is called *alphabetic redundancy*. It suggests, if you would assign variable-sized code to each letter of the alphabet, that 'e' would get the shortest code and 'z' would get a longer one. There is another kind of redundancy called the *contextual redundancy*, in our example it is represented by the fact that the letter 'q' is almost always followed by 'u'.[11] The English language has a method of compression called *the textspeak*. Since the invention of Short Message Service (SMS), communicating became instant and in order to speed up the process of composing a message, people created acronyms for most common words, often eliminating the vowels.

In 2003 a rumor began that a teenager had written an essay entirely in textspeak. The reported extract began like this:

MY SMMR HOLS WR CWOT. B4, WE USED 2GO2 NY 2C MY BRO, HIS GF & THR 3 :-@ KIDS FTF. ILNY, IT'S A GR8 PLC.

It can be translated as follows:

"My summer holidays were a complete waste of time, Before, we used to go to New York to see my brother, his girlfriend and their three screaming kids face-to-face. I love New York. It's a great place."

If a binary sequence needs to be evaluated for randomness, one can just simply ask if it can be compressed through conventional algorithms. The answer is usually yes, or no. The answer depends on whether or not random data has structure, which it by definition has not. Thus, we cannot have a margin for acceptable non-randomness with perfection squarely out of our grasp.

2.2. Physical sources of random numbers

In this chapter, I introduce:

- Thermal noise
- Avalanche noise
- Nuclear decay
- User input
- Air turbulence

Entropy sources largely depend on their application. Once enough randomness is collected, it can be used to produce even more pseudo-randomness, after it is de-skewed or mixed. [2]

Particularly desirable are sources of entropy rooted in physical phenomena. The emission of radiation was the one of the first of such phenomena that were proposed. Though reliable in randomness, it does use specialized hardware not integrated in personal computers. [12]

There are entropy sources that do not have quantum-random properties and are therefore easier to detect. They are sometimes skewed when the temperature of the system is lowered, though most systems have countermeasures in place. Such systems use, e.g. thermal noise.

Thermal noise or Johnson-Nyquist noise exists in electronic circuits. It was first recognized by John B. Johnson in 1926, but the one who explained its origin was Harry Nyquist. This noise is created when particles (typically electrons) that carry charge are exposed to thermal energy, i.e. Heat. These begin to vibrate and the higher the temperature they are exposed to, the higher the thermal noise level. In its nature, it is completely random, which is excellent for conversion into random numeric values. The other excellent thing about it is that it exists in any electrical circuit, no matter the quality of equipment. The higher the resistance, the higher the temperature, the more thermal noise. [13]

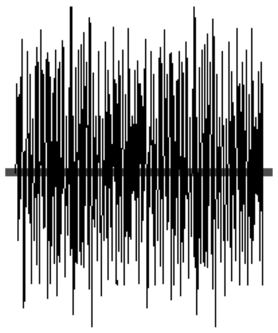


Figure 4: Noise signal visible with an oscilloscope [12]

The advantage of thermal noise as an entropy source is that there is not a lot of hardware components needed. That means that it could easily be integrated into computer hardware.

Noise generated from an Avalanche diode is also common, i.e. Avalanche noise. It is a form of electric current that collects itself in semiconductors or insulators through electric fields. A designated avalanche diode is built to prevent hot spots from current concentration and still experience avalanche breakdown in a controlled environment.

Atmospheric noise and waves from radio transmissions detected by a receiver attached to a PC is another source of entropy that is quite rich. Lightening noise, static, VHF waves are captured and transferred into numbers. This source can be vulnerable to interference, but usually, it is very reliable. [14]

Entropy sources with quantum-random properties focus on atomic and sub-atomic effects, relying on the theory of quantum mechanics — that such physical phenomena as the nuclear decay of atoms are random and cannot be predicted. For that reason, outcome of such generators is model for all other generators.

Nuclear decay, of course not of any hazardous isotopes, can be detected by a Geiger counter attached to a PC. Some commercial smoke detectors carry a very small quantity of americium-241 — a radioisotope, which ionizes air. This isotope is produced by nuclear reactors when plutonium is bombarded with neutrons. In turn, it emits alpha particles and gamma radiation in order to become neptunium-237. The principle of such smoke detectors is that alpha particles from the isotope collide with oxygen and nitrogen inside of a specifically designed small chamber inside the smoke detector, creating electric current through ionization. Smoke neutralizes ions, decreasing the electricity, at which point the alarm goes off.

Often user input is used as an entropy source, e.g. SecurPC — an RSA toolkit running under Microsoft Windows 95 — generated random seeds based on keyboard and mouse timings. Every now and then, the user was animated to press "random" keys on the keyboard for a few minutes. This sort of approach might not have been simply annoying, but also questionable. Reliance on user-generated randomness should be avoided, since it vulnerable to attacks, providing a false sense of safety.

Some entropy algorithms use computer hardware but also rely on physical phenomena. For example, AT&T's truerand that exploits the CPU clock and the real-time clock and the fact that they are not connected. It uses a counter switching from Busy to Waiting until a timer interrupt terminates the loop, thereby pitting the real-time clock signal against the CPU signal, using their differences to its advantage.

Air turbulence in hard drives is another entropy source that is based on computer hardware, unfortunately, it is becoming outdated. Using the disk's rotation and the high-speed turbulence in HDD drives, one of the ways of generating randomness is measuring the read/write times.

It is important for software creation to avoid specialized hardware. Suppose, a virus is transferred unwittingly from a machine with specialized randomness hardware to one without, the software in play cannot assume that such hardware is present, much like human beings carrying a virus they have been vaccinated against should still exercise caution.

2.3. Hardware random number generators and getting physical values into bits

In this chapter, I investigate:

- Random processes
- Early hardware random number generators
- OneRNG
- XR232
- Atmospheric noise, radio frequencies

As we have previously established, nature is full of **random processes**. In urban culture, we surround ourselves with predictability: a bus arriving according to schedule, a clock striking every hour, coffee maker whirring at preset time, etc. However, hard as we try to make our environment predictable, unpredictability breaks through in weather changes, air currents, grass growing in most unwelcome places, even natural disasters. It is inconvenient, but it is also something we have never lived without.

Random processes repeat themselves and it is difficult to find a deterministic pattern in their output. [15]

Number generators based on random processes are known as hardware random number generators or **TRNGs** (True Random Number Generators). There is no randomization algorithm used, since the data extracted from the generator is already **random in theory**. Practicality and security usually do not rely on theoretical randomness, though.

One of the earlier ways of producing randomness was, of course, based on gambling. The machines that used numbered ping-pong balls with blown air that were withdrawn from a mixing chamber were built similarly to those used to play keno or select lottery numbers.

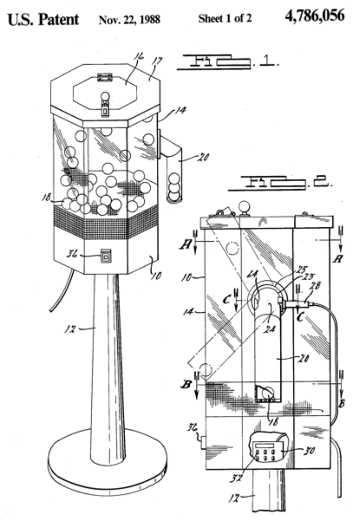


Figure 5: Random number generator proposed by Richard P. Dinnigan in 1987 to the US Patent Office [14]

The method described and depicted in Figure 5 offers reasonable results in terms of randomness but unfortunately, is very slow.
Modern approaches mostly use specialized hardware that can be connected to a computer processor. One such generator was funded on December 21st, 2014 by a very successful campaign (less than a week) on Kickstarter. While being open source in terms of hardware design and firmware, the small generator about the size of a USB data stick also possesses a removable radio frequency noise shield to protect from physical attacks. As another security measure, OneRNG's board cannot be reprogrammed over USB by errant software. However, the suite it was created with is included in the shipping parcel, as well as a suitable cable. One unit generates 320 000 bits per second. [16]
The data can be used directly, but it is recommended to feed it into a kernel random number generator's entropy pool, such as the Linux/Unix implementation under /dev/random or /dev/urandom, which will be discussed in the following chapter.
OneRNG uses two sources to generate randomness — an avalanche diode and a radio frequency receiver. Data is sourced one byte at a time from both sources and put into a so-called "whitener": a CRC16 generator that performs cyclic redundancy checks. It means that bits are merged and XOR'ed together to improve uniformity. While OneRNG is fairly inexpensive, it is calibrated in such a specific way, it cannot be used for any other purpose but to generate random numbers. Since computers are growing exceedingly small, there is no room for specialized hardware, especially because using it as a selling point does not attract the average consumer. And what does not attract the average consumer is missing in most computers using a specific

kind of cryptographic software that relies on said specialized hardware. Without it, this software is in great peril with so many weak links in its chain.
So this brings about an interesting question: can one recycle **cheap** ubiquitous components and make a true random hardware generator accessible to any computer system no matter the budget?
The answer is yes, there is such a solution made out of very cheap components, the reliability of which will be tested in the following chapters.
XR232 is a true random number generator that delivers independent random data to a host computer. It uses open-source software and has a very transparent and well-tested circuitry. [17]

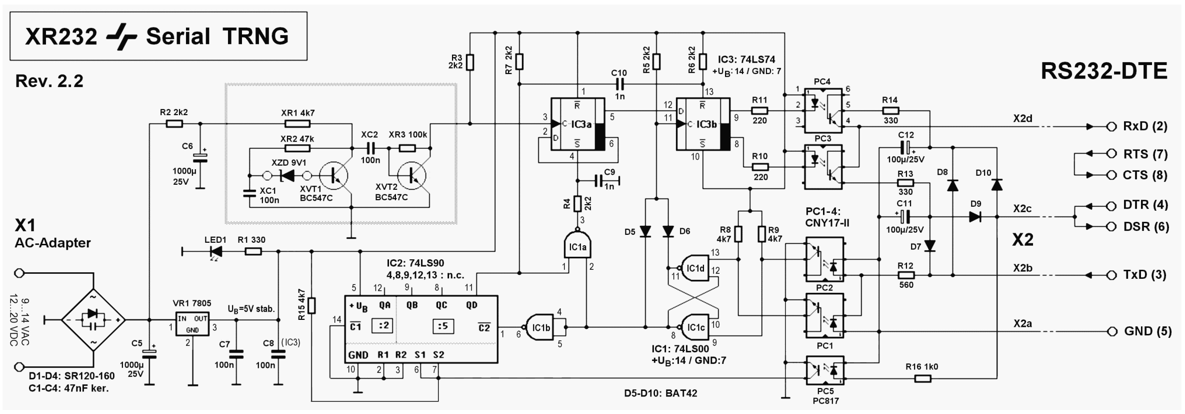


Figure 6: The schematics for the XR232 zener-powered random number generator by Julien Thomas [16]

Avalanche and Zener breakdown are essentially the same effect happening in two different components: the avalanche diode and the the Zener diode. However, one of them lasts longer and is considered more stable — the Zener diode.
The Zener diode (XZD in Figure 6) is sometimes called the breakdown diode because it is designed to operate mostly in the breakdown region. So this effect is not only useful, it is desirable. To obtain the Zener effect, the diode is used in a reverse bias mode. That means that unlike the current traveling in a normal fashion — from component A to component B, the current is traveling through the Zener diode in a B to A fashion. The diode thereby connects to the negative supply. In this mode, no matter how much voltage is at the diode, it only lets a stable low voltage pass through. This saves the circuit from damage.
Zener noise is produced at the beginning of the Zener effect, where electrons and protons collide with the atoms in the crystal lattice of the Zener diode creating an electric field. If this process happens to spill over into an avalanche effect, random noise spikes may be observed.

The noise serves as a bit input into the semiconductor (IC3). The job of this component is to digitize the noise. The now digitized noise is forwarded to the converter (marked with IC2 in Figure 6) it formats the signal into a readable protocol — RS232. To ensure the safety of the recipient device, this circuit contains the optocouplers, also called the optical oscillators. They are used to prevent high voltage from affecting the system. [18]
Another way of using inexpensive components efficiently, is the ability to utilize them for a different purpose. That is the case with software defined radio.
An rtl-sdr dongle receives radio frequency signals and can be routed to your computer with a software interface. It allows the user to listen to amateur radio, watch analog television, listen to FM broadcasts and a number of other things, such as monitoring first responder and aircraft frequencies. But the radio frequency noise is also caused by natural occurrences, such as weak galactic radiation and stronger local as well as remote lightning strikes. This provides a great deal of entropy for a random number generator.
Of course, the dongle cannot get down to the frequencies necessary for sampling atmospheric noise, which is about 100 KHz to 10 MHz and above 10 GHz. But in turn, it can sample cosmic noise as well as urban and suburban noise that is man-made, solar noise, thermal noise and other terrestrial noises within the range of the dongle.
These dongles are quite cheap, you can connect them via USB and they use an antenna. With a standard telescoping antenna about 3 Mbps of true random data can be observed. [19]
The data will be tested in the following chapters. But the preemptive visual analysis looks good so far, and can be looked at in Figure 7.

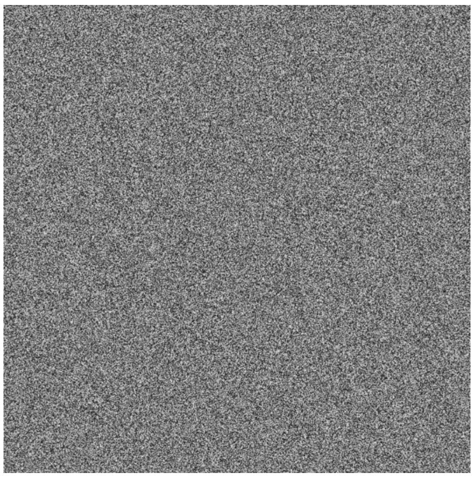


Figure 7: Visual analysis of rtl_entropy generated data [17]

2.4. Pseudorandom number generators and Linux implementation of dev/random and dev/urandom

In this chapter, I explore the pseudorandom number generators and the following keywords:

- Principle
- Obvious disadvantages
- Lehmer (RANDU)
- ENIAC (von Neumann)
- Xorshift
- Cryptographically secure PRNGs
- LRNG

Suppose there is a relatively small number of truly random numbers that were not generated using the following method. These random numbers already exist as a base upon which further calculations are made. One value, at a time, is extracted from the pool and used as a **seed**.
A deterministic algorithm accepts the seed as its initial state and generates a sequence of numbers that have to look like they are random. The same seed always generates the same sequence.
If it does not sound like a good idea for cryptography, that is because it is not. Simple PRNGs are not cryptographically secure and fail miserably when tested for patterns. They are also periodic, which means that sequences tend to repeat themselves after a certain time because the pool is relatively small. Large generated sequences tend to be less uniform. In other words, small sequences appear random while certain parts of larger ones do not. Another disadvantage is that there are correlations in neighboring values.
One bad example of a horrible bunch is RANDU. It is in its principle a Lehmer RNG, also referred to as Park-Miller RNG. It is a type of linear congruential generator that uses the following general formula:

$$X_{k+1} = g \cdot X_k \mod n$$

RANDU's specifications are:
g = 65539
n = 2^{31}
The seed numbers for this generator are always odd. And the sequences generated fall in the range between 1 and $2^{(31-1)}$.
The problem with this generator is depicted in Figure 8.

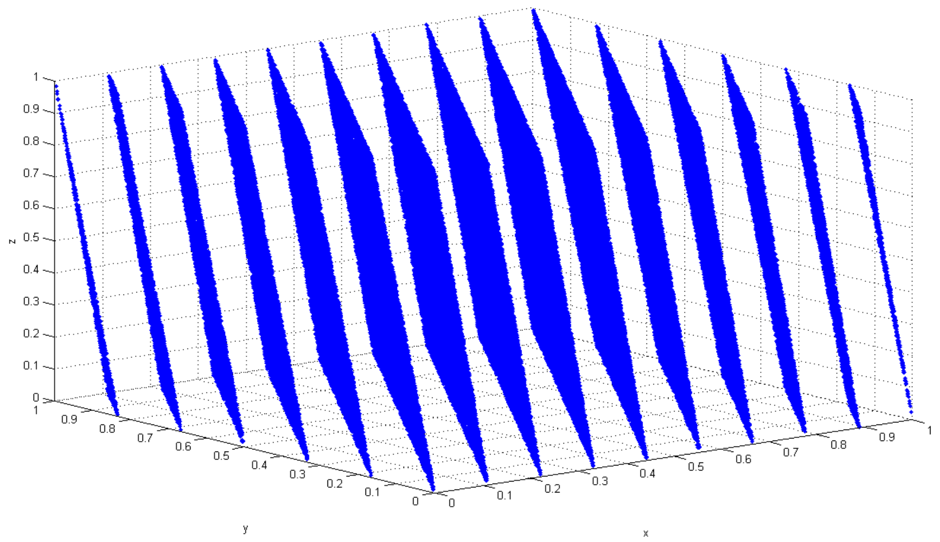


Figure 8: Three-dimensional plot of 100,000 values generated by RANDU. Each point represents 3 neighboring values. Uploaded by Luis Sanchez. [20]

The flaw clearly lies in the recurrence. For each three consecutive values, provided we expand the quadratic factor, the recurrence only comes down to:

$$x_{k+2} = 6x_{k+1} - 9x_k$$

So the results fall in 15 planes. [20]

Lehrer RNGs were important for over fifty years because none other were available. At the end of the 20th century, seeing that linear congruential generators achieved dubious results, the scientific community distanced itself from them with Mersenne Twister and the WELL family generators, only to be brought back to them by George Marsaglia in 2003 with his Xorshift generators.

Xorshift means that each next number in the sequence is the repeated taking the "exclusive or" of a number with a bit-shifted version of itself. This PRNG class is incredibly fast even on slowest processors (thanks to pipelining), but it is not cryptographically secure either.

For CSPRNGs or cryptographically-secure PRNGs, ideally, one would need a high-quality entropy source. In other words, a TRNG-generated source of random numbers. Plus, in order to be cryptographically secure, they have more requirements than PRNGs.

Firstly, the **next-bit test**. Proven by Andrew Yao in 1982, should a generator pass the next-bit test, all other polynomial-time statistical tests for randomness would not be a problem. The test establishes, whether or not, derived from the indefinite number of known bits, the next one could be guessed with the probability of success better than 50%.

Secondly, "**state compromise extensions**". Suppose the seed used in the running sequence is a bit of pi. Suppose it is four. Should the attacker know the running

sequence and the seed, they should not be able to figure out the previous sequences and their respective seeds. With pi, that is not a given. A PRNG deriving sequences from pi may pass the next-bit test, because pi appears to be a random sequence, but the algorithm still is not cryptographically secure.
Most PRNGs indeed are not. Despite offering seemingly random sequences, they can be reverse engineered, allowing the attacker to see all past random numbers. CSPRNGs are specifically engineered to prevent that. [21]
Often, pseudorandom numbers appear to be more random than those from physical sources. The construct of a pseudorandom number generator is more complex and has multiple layers, each introducing more randomness. Each layer contains a transformation that can eliminate statistical auto-correlations between starting and end values. This way the output of a PRNG may have better statistical properties and be produced faster than a TRNG.
One implementation widely used in Linux- and Unix-based operating systems is the Linux pseudorandom number generator (LRNG). This random number generator resides in the kernel of most UNIX-based systems under /dev/random or /dev/urandom.
E.g. Some versions of Linux have this generator with an entropy pool of 512 bytes (128 different words of 4 bytes each). This pool is disturbed whenever an event, such as disk drive interrupt, occurs and the time of the event is added to the pool via a XOR operation, after which the pool is stirred with a primitive polynomial of degree 128.
Each time entropy is added to the pool in this way estimates the amount of true randomness of the input. There is an accumulator in the pool itself that estimates the randomness contained there.
These events come from several sources, such as:

- Keyboard interrupts. Not the content of the keystrokes adds entropy to the pool, but the inter-keystroke time.
- The mentioned above disk activity, among other things — disk completion, because a system accessed by a human being is statistically likely to have an unpredictable pattern of disk activity.
- Mouse motion, where both mouse position and timing are added in.

When random data is taken out of the generator, the pool is hashed with the SHA-1 algorithm. The output of this algorithm is 20 bytes long, but if more is required, then the output is stirred back into the pool and new hash is performed to get the next 20 bytes. The more bytes are taken from the pool, the less entropy is expected.
To make sure that the pool is random at system startup, the scripts of starting and shutting down the generator save the pool to the disk and then read this file at system startup.
Now, what is the difference between /dev/random and /dev/urandom, since we are clearly dealing with the same random number generator?

/dev/random blocks when the pool accumulator shows that the estimated randomness inside the pool is zero. The more events occur the more data becomes available at /dev/random. So it is better for generation of long term keys, provided that there is enough data in the pool.
/dev/urandom works exactly like /dev/random, but it does not block. It provides data even if the entropy estimate equals zero. This is okay if used for session keys or when it is not possible to wait for more random bits. The risk of continuing to take data from the pool's entropy when the estimate of randomness is small because even if the attacker can get past output, reversing SHA-1 to get the seeds is an issue. The algorithm was designed to be non-invertible. [2]
So which is better?
There is no definitive answer to this question. Since /dev/urandom does not block when it deems the entropy to be insufficient, it seems to be insecure to a casual observer. It is not. Both use a generator that is cryptographically secure. It is recommended to use /dev/urandom for "normal" cryptographic purposes by quite a few experts in the field. But why?
The gold standard in true randomness is the quantum effect, but the purpose of this randomness is mostly cryptography, where these perfectly random numbers are fed into algorithms that can only offer computational security at best and **not information-theoretic security**, except for Shamir's Secret Sharing and the One-time pad. The latter is simply impractical.
The rest, being AES, RSA, Diffie-Hellman and so forth, do not protect from an adversary that disposes of unlimited computational power. They are still used because it is not presumed that in order to break a key, all computers in the world would be used together for a period of time longer than the existence of the universe.
There is no way to predict the development of computer systems in the future. But for the sake of cryptography right now, computational security is **good enough** and so is /dev/urandom. [22]

3. Testing random number generators

3.1. Compatibility of random number generators, which one is better and the criteria

The following chapter will contain data on:

- Unpredictability and goodness-of-fit
- Methods for improvement
- Data rate efficiency
- Price value
- Security
- Compatibility of random number generators

The previous chapters show that there is a wide variety of random number generators. But not all are created equal.

It is self-evident that only random number generators with solid properties in theory should be used, also they should have successfully passed a battery of tests. Bias within the generated sequences can render a random number generator useless for most applications.

In order to empirically test random number generators, it is best to generate sequences from each of them and then apply various distributional and randomness tests on the sequences. It is not an in-depth approach on the properties of each generator, but it is adequate to evaluate the overall performance of RNGs. It is important to understand that what may apply for one sequence does not always apply for all sequences. [23]

Many empirical statistical tests have been developed in an attempt to determine whether or not there are any short-time or long-time correlations between the numbers or their distribution. These tests are not enough. The ultimate test requires to be run on many generators and if the results agree within a certain error margin, the results are only likely to be correct. [24]

As was already mentioned in the previous chapter, cryptography does not require random number generators to be information-theory secure, but computationally secure.

Computational security, just like any type of security, cannot be absolute. In the case of cryptography, there are a lot of links in a chain: the application, end-to-end encryption (the algorithm and the randomness generated for it) in Alice and Bob respectively and what happens during transmission, e.g. server communication), hard drive encryption and so forth.

The basic requirement for session keys, passwords and nonces throughout this chain is **unpredictability**.

Unfortunately, you cannot measure it. As was already established in the previous section, there is an expected distribution and the distribution of the generated values is being compared to it. There is only a certain margin that can be reached for random numbers to be unacceptable and it is very small.

The goodness-of-fit test provided by National Institute of Standards and Technology of the U.S. Department of Commerce (NIST) consists of 15 individual tests and operates under the tentative presumption of randomness.

Each statistical test is written to test a **null hypothesis** (H0), that the sequence being tested is random. Derived from this hypothesis is the **alternative hypothesis** (Ha), that the sequence is not random. For each test, the null hypothesis is either rejected (and the alternative hypothesis is accepted) or accepted. In order to either accept or reject the hypothesis, a **test statistic** needs to be calculated and then compared to the **critical value**, which was already discussed in the chapter 2.1. In the NIST suite, the critical value is chosen to be 0.01. If the test statistic value is smaller than the critical value, the null hypothesis is rejected. Otherwise the null hypothesis is accepted.

In truth, the reason why the statistical testing works is that if the randomness assumption is true for the data, then the test statistic value on the data will have a very low probability of being lower than the critical value. If it, in fact is, then there is a low probability that this event would occur naturally. So if it happens, the original hypothesis is immediately rejected. [8]

So how does the suite evaluate **how random** a generator is? The truth is, it does no such thing. The statistical suite makes a binary decision: "random" or "non-random" and then generates a **p-value** for each separate test. This p-value means how probable it is that the generated sequence is perfectly random. And having a higher p-value for a tested sequence from one generator than a tested sequence from another does not immediately mean that the first generator is better than the latter. If the p-value for most of the tests is over 0.5, then the sequence is good and can be used for encryption. [15]

But even if the NIST suite's results for a sequence are unfavorable, it is not a reason to completely abandon a random number generator. In order to achieve higher p-values, the statistical properties of the output can be improved through different means:

Encryption: by adding a layer of encryption randomness can be added to a weak random sequence

De-skewing. Suppose the sequence is not perfectly uniform, however, it is known how much it diverges from it, then the sequence's uniformity can be improved. One of the de-skewing techniques, developed by von Neumann, uses Transition Mappings. It examines a bitstream as a sequence of non-overlapping consecutive pairs. First, one discards any same-bit pairs, then, one interprets 01 as 0 and 10 as 1. That eliminates bias but requires a number of input bits that is indeterminable if the desired number of output bits is known. Another method of de-skewing is the Fourier transform of data. FFT discards strong correlations. [2]

Mixing: employing a good mixing function can be helpful to distort the output of an unreliable random number generator. The principle of mixing is employed in pseudorandom number generators. For example, the AES algorithm is an extremely reliable, well-documented and thoroughly tested method that has the advantage of being non-invertible. [2]

3.2. Requirements per purpose

- Uses outside of cryptography (gambling, simulations, Monte Carlo)
- Uses in cryptography

True random number generators are by definition unpredictable and pseudorandom variants would produce the same sequence if they use the very same seed, which leads to the conclusion that we do not necessarily need the latter. It is a false conclusion.

In fact, pseudorandom numbers often appear to be more random than random numbers obtained from physical sources. It happens because if a pseudorandom sequence is properly constructed, each value in the sequence is used in transformations to produce seemingly additional randomness in the following value. A series of these transformations can dispel statistical auto-correlations between input and output. Because of that, the outputs of PRNGs sometimes have **better statistical properties**, not to mention produce sequences **faster** than an RNG. [8]

Plus, for simulations and Monte Carlo purposes, the pseudorandom generators have one serious perk: **repeatability**. Namely, the fact that if the seed remains the same, they generate the same sequence of randomness. It serves a significant purpose in recreating an experiment.

Gambling does not benefit from speed and repeatability, in fact, most slot machines are built with longer sequences to build suspense. A perfect gambling system would need to have the structure of a black box, **impenetrable** to an attacker. So a guard from interference is key.

Cryptography demands speed because users rely on almost instant communication and an impenetrable system is a prerequisite. A reliable random number generator is very important because the secret key used in encryption and handshake protocols needs to be not guessable.

For speed, cryptography relies on pseudorandom number generators, but in order to be useful, they need to be **cryptographically secure**.

Firstly, the requirement for a PRNG to be secure is **pseudo randomness**. You may think that since this word is in the title, this requirement is trivial, but it is not. It merely means that the output of a deterministic generator needs to be indistinguishable from that of a true random number generator. That is not to say it needs to be exactly the same. But it needs to be impossible to predict, just like the output of a — by definition — unpredictable generator. The question is: how far do you go, checking this requirement, and the answer usually presents itself as a theoretically vague response of any information scientist — polynomial time. What it means is that depending on the length of the tested sequence (n), in the worst-case scenario the testing algorithm should have n^k steps, of which k is some constant. In computer science, problems of this complexity are considered feasible. In order to simplify the understand, I will put it in terms of a malicious attacker.

If there is encrypted communication and the key to that communication is a randomly generated sequence, it should be impossible for the attacker using modern technology to guess this key, regardless of how fast their system is.

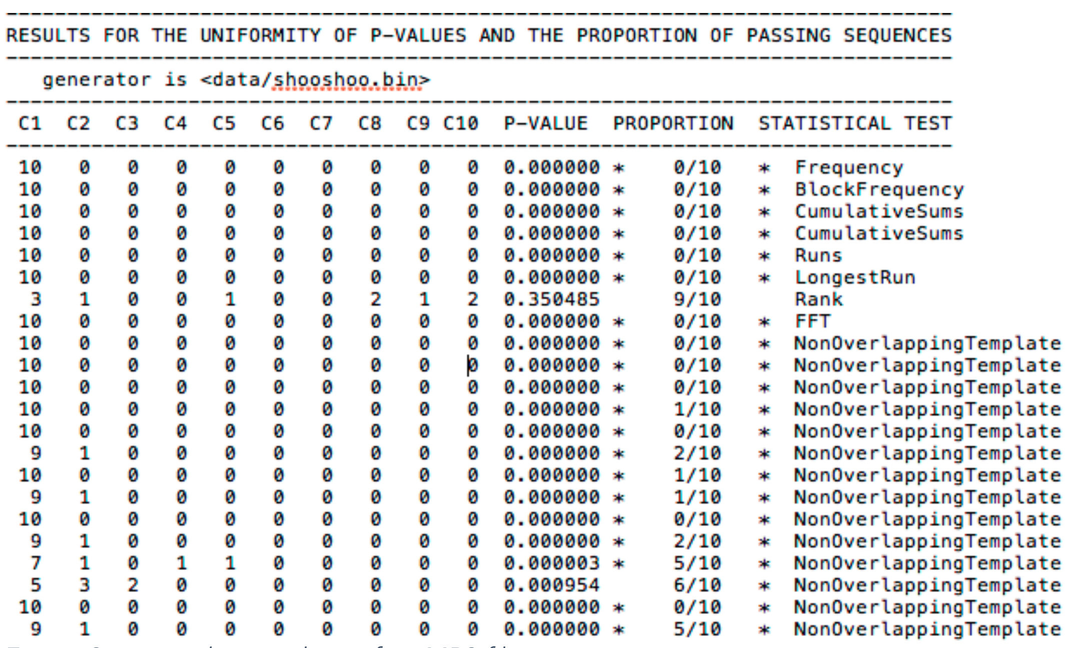
A test for pseudo randomness was developed by Andrew C. Yao and is called the **next-bit test**. The principle of it is also very simple: a bit-guessing algorithm or „distinguisher" should not be able to guess the next bit of the sequence with a probability higher than 50%. [25]

The last two requirements for a cryptographically secure generator are **forward and backward security**. Forward security means that knowing the generator's current state, the attacker should not be able to elicit its past output. Backward security signifies that knowing the current state of the generator as well, the attacker should not be able to predict the future output. [26]

4. Results

4.1. Statistics over several runs and repeatability

In the testing of data using the NIST STS, I decided to start with nonrandom data, to see how the suite would perform. I have also chosen 10 bitstreams of 512.000 bits each, while testing an MP3 file in binary, which revealed that the data is very much nonrandom and the tests failed systematically. It was expected. You can see in Figure 9, the complete results for the first seven tests and only part of the results for the eighth — Non-Overlapping Template test. The star immediately after the proportion column indicates that the minimal pass rate of eight sequences out of ten was not reached and the star before it indicates that the p-value for every single bitstream is below the critical value of 0.01, as was discussed in the Chapter „Requirements and Tests" of the State of the Art section, it means that the data is non-random.



```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <data/shooshoo.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  Frequency
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  BlockFrequency
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  CumulativeSums
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  CumulativeSums
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  Runs
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  LongestRun
  3   1   0   0   1   0   0   2   1   2  0.350485      9/10       Rank
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  FFT
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    1/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  NonOverlappingTemplate
  9   1   0   0   0   0   0   0   0   0  0.000000 *    2/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    1/10    *  NonOverlappingTemplate
  9   1   0   0   0   0   0   0   0   0  0.000000 *    1/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  NonOverlappingTemplate
  9   1   0   0   0   0   0   0   0   0  0.000000 *    2/10    *  NonOverlappingTemplate
  7   1   0   1   1   0   0   0   0   0  0.000003 *    5/10    *  NonOverlappingTemplate
  5   3   2   0   0   0   0   0   0   0  0.000954      6/10    *  NonOverlappingTemplate
 10   0   0   0   0   0   0   0   0   0  0.000000 *    0/10    *  NonOverlappingTemplate
  9   1   0   0   0   0   0   0   0   0  0.000000 *    5/10    *  NonOverlappingTemplate
```

Figure 9: incomplete analysis of an MP3 file

Interestingly, the only test that the sequence has passed was the Rank test, which involves the rank of binary matrices. It can be explained by the lack of uniformity within the sequence, whereby less patterns in linear dependence between substrings of fixed length can be detected. This test also appeared on the DIEHARD battery of tests and these results demonstrate that merely performing one test could never prove randomness beyond reasonable doubt.

After the construction of XR232, we have discovered that the output of the device is skewed towards zero. This anomaly could be explained by a malfunction in digitization of the noise that comes from the Z-diode. The schematics for this generator can be viewed in Figure 6. Figure 10 shows the generator I constructed.

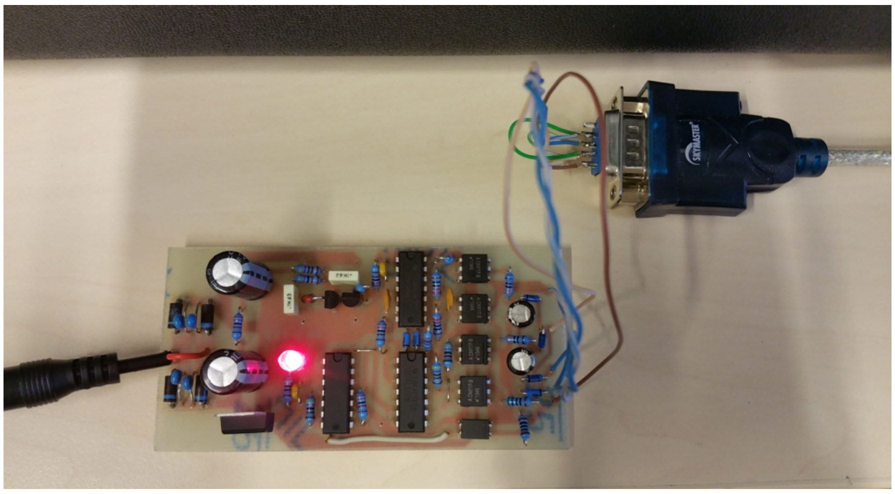


Figure 10: XR232

The zero bias could stem from the voltage only barely reaching 5 volts and that still being interpreted as a zero signal. Furthermore, the generator was incredibly slow.

These results were evaluated using the NIST STS and our fears were confirmed. Initially, the raw results were so bad, the suite's tests failed systematically. In an attempt to de-skew the output of XR232, I encrypted it using GPGTools. The suite revealed that the sequence is still unusable for cryptographic purposes, however only barely not passing the lowest performance margin in two instances of the Non-Overlapping Templates test. The results can be found in Table 2.

Much better results were achieved by a generator using atmospheric noise. The generation was fast and the tests show green across the board. They can be seen in Table 3.

The best results so far were recorded by the STS when testing the output of dev/random. Most of the tests generated a p-value over 0.5, which is remarkably good. First, I tested 10 bitstreams (512.000 bits each) and the proportion was nearly perfect. The complete set of results can be seen in Table 4. I decided to perform another round of testing, this time encompassing 100 bitstreams of the same size, as seen in Table 5. My fears that the results were somehow skewed were not confirmed. The excellent statistical properties of the dev/random-generated sequences can be explained by the generator having several de-skewing and mixing measures built in. It is safe to say that the generator is indeed cryptographically secure and the usage of it can be encouraged.

The most surprising experience was testing the output of the ANU Quantum Random Numbers Server (http://qrng.anu.edu.au/). Generating the sequence was not fast, and I was hoping that the golden standard of randomness would trump the results of an albeit cryptographically secure but still a pseudorandom generator. The above mentioned webpage contains the results of continuously running NIST STS tests. Though the p-values are high on average, they sometimes fall below the acceptable margin, which lead me to believe that the quantum effect was not worth the wait. Table 6 shows that only once the minimal margin was not reached. While that does not mean that the golden standard of randomness disappointed completely, the slowness in generation was a bitter pill to swallow.

4.2. Graphics and charts

Table 2: testing a XR323 sequence with a layer of encryption, length of 100 bitstreams per 512.000 bits

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <data/zrandom.bin.gpg>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  8   6   8   8  12   8  12  16  10  12  0.534146     98/100     Frequency
  8  10  10  14   8   7  11   9  12  11  0.911413     97/100     BlockFrequency
  8   7   9   8  12   7  14  13  10  12  0.739918     98/100     CumulativeSums
  7   5  12  10   6   8  15  15   9  13  0.224821     99/100     CumulativeSums
  9   7  15   9   7  14  17   8   7   7  0.153763    100/100     Runs
  9  16   9   9   7   8  10  12  11   9  0.759756     99/100     LongestRun
 14  12  13  11   6   8   8  12  10   6  0.595549    100/100     Rank
  9  13   5  13   9   9   8   8  13  13  0.616305     97/100     FFT
 13   8  11  14   6   5  12  13   8  10  0.455937     98/100     NonOverlappingTemplate
 16   7  12  13  13   8   8   5   9   9  0.334538     98/100     NonOverlappingTemplate
  9  10   6  12  11  11  14   6  16   5  0.236810     99/100     NonOverlappingTemplate
  7   7  12  13   8   8  13  15   8   9  0.554420     99/100     NonOverlappingTemplate
  7   9  11  12  11   6  15  12   8   9  0.678686    100/100     NonOverlappingTemplate
  4  10   3  15   8  10  13  11  14  12  0.108791    100/100     NonOverlappingTemplate
  9   9  13   9  16  10   8  14   6   6  0.350485     99/100     NonOverlappingTemplate
  5   9  14   9  16   6  12  10  11   8  0.319084    100/100     NonOverlappingTemplate
 10  10  10   5  14  11   9   9   8  14  0.699313    100/100     NonOverlappingTemplate
 13  12  12  10  12  10   7   9   8   7  0.883171    100/100     NonOverlappingTemplate
 11  12  11  11  13   9   7  11   9   6  0.883171    100/100     NonOverlappingTemplate
 11  11  10   6  14   5   7  18  10   8  0.137282     99/100     NonOverlappingTemplate
  6  10   9  10  12   8  11  14  11   9  0.883171     98/100     NonOverlappingTemplate
 12   6  13   8  12   9  16  11   7   6  0.350485     98/100     NonOverlappingTemplate
 12  13   8  12   9   6   9  11  17   3  0.129620     99/100     NonOverlappingTemplate
  9  10  10  12   9   9  13  11  10   7  0.978072     97/100     NonOverlappingTemplate
  8   6   9   9  12   9   7  21  10   9  0.071177     99/100     NonOverlappingTemplate
  7  14   8   4  10  13   7  10  13  14  0.289667     99/100     NonOverlappingTemplate
 15  12   9   8  10   7   8  12  13   6  0.574903     99/100     NonOverlappingTemplate
 11   7   7  13  15  11  10  11   7   8  0.657933    100/100     NonOverlappingTemplate
  8  11  11   7   7  10  10  14  13   9  0.834308     99/100     NonOverlappingTemplate
  5  12   7   7  15  10   7  11  13  13  0.350485     99/100     NonOverlappingTemplate
 16   7  12  10   7   9   6   9  12  12  0.494392     99/100     NonOverlappingTemplate
 10   7   9   9  14   8   8  14   5  16  0.262249    100/100     NonOverlappingTemplate
 11  11  12  11   4   5  11  12  13  10  0.514124     98/100     NonOverlappingTemplate
 11  11  10   8   7  13   9   8  13  10  0.924076     97/100     NonOverlappingTemplate
  9  15   9  10  13  12   3   7  11  11  0.350485    100/100     NonOverlappingTemplate
  5  14  12   5   8   9   9   9  13  16  0.202268     99/100     NonOverlappingTemplate
  6  13   8   7  15  10   5  13  15   8  0.181557    100/100     NonOverlappingTemplate
 11   9   9  14  10   8  12   8   7  12  0.883171     98/100     NonOverlappingTemplate
 10  13  12  10   6  10  11   5  12  11  0.739918     99/100     NonOverlappingTemplate
 14  15  13   7  12   9   8   6   6  10  0.350485     98/100     NonOverlappingTemplate
 11   9  10   6  14  10  10   9   9  12  0.911413    100/100     NonOverlappingTemplate
 12   8   8  13   7  15  10  13   6   8  0.494392     99/100     NonOverlappingTemplate
 10  12  15  10  10   9  10   4   9  11  0.657933     99/100     NonOverlappingTemplate
 11   9  12   5   7  16  14   9   5  12  0.202268    100/100     NonOverlappingTemplate
 14  13  11   7   8   5  11  12  11   8  0.595549     96/100     NonOverlappingTemplate
  7   8  13  13  11   9   8  12   9  10  0.897763     99/100     NonOverlappingTemplate
  7   8  16  10   7   7  12   8  11  14  0.419021     97/100     NonOverlappingTemplate
  8   7   8   9  18   9   9  12   9  11  0.437274     99/100     NonOverlappingTemplate
 12  12  12  11   7  14   8  12   5   7  0.534146    100/100     NonOverlappingTemplate
 11   8   6  15  12   9   9   6  11  13  0.554420     99/100     NonOverlappingTemplate
 11   9  17   9  10   7   5   9  11  12  0.419021    100/100     NonOverlappingTemplate
  8   9  16  12   8  10  10  10   9   8  0.798139    100/100     NonOverlappingTemplate
  9   6  10  11   5   8  11   9  12  19  0.145326    100/100     NonOverlappingTemplate
  9   8  15   8   4  14   7  10  17   8  0.096578     99/100     NonOverlappingTemplate
 15   9   9  16  14   7   7  11   5   7  0.153763     99/100     NonOverlappingTemplate
 11  10   7  11   7   9   9  13  11  12  0.935716     99/100     NonOverlappingTemplate
  6   8  12  14   9  11   5  12   8  15  0.350485     99/100     NonOverlappingTemplate
 10  10   7   7  10  11  13  11  11  10  0.964295     99/100     NonOverlappingTemplate
 11  11   7   9  14   9  12   9  11   7  0.883171     99/100     NonOverlappingTemplate
  8  12   6  15  10  12   7  11  12   7  0.574903    100/100     NonOverlappingTemplate
  6   9   9  13  12  14  10  10   9   8  0.816537     99/100     NonOverlappingTemplate
 15   9   8   6  14   6   8  10  15   9  0.289667    100/100     NonOverlappingTemplate
 10   7   9  13   9  12  10   6  13  11  0.834308     99/100     NonOverlappingTemplate
  6  11  13  17  12  11   9   7   8   6  0.275709     99/100     NonOverlappingTemplate
 12   7  10  14   6  14   5   9   8  15  0.236810     99/100     NonOverlappingTemplate
  6  10  10   7  15  11  13  12   7   9  0.595549     99/100     NonOverlappingTemplate
  7  13  11  13  12  11   8   5  10  10  0.719747     99/100     NonOverlappingTemplate
  8  12   6  12  11   4  12  11  11  13  0.534146    100/100     NonOverlappingTemplate
 12   5  11   7   7  16  10   6  15  11  0.181557     99/100     NonOverlappingTemplate
 10   5  13   8  13   3  17  11  10  10  0.102526    100/100     NonOverlappingTemplate
  7  13   9  11   9  12   8   8  10  13  0.897763     99/100     NonOverlappingTemplate
  9   8   9  12  12   7  12  16   7   8  0.574903    100/100     NonOverlappingTemplate
 15  13   8   6  10   8   6  12   9  13  0.455937     95/100  *  NonOverlappingTemplate
 10   6  10   8  12   7   7   7  18  15  0.122325    100/100     NonOverlappingTemplate
 14  11  12  10   8   7   6  14  11   7  0.574903     99/100     NonOverlappingTemplate
 13  19  12   8   6   3   8  12  12   7  0.030806     98/100     NonOverlappingTemplate
  3  14   9  11   4  16  10  10  17   6  0.015598     99/100     NonOverlappingTemplate
 12  11  13   6   9   9   9   7  11  13  0.816537    100/100     NonOverlappingTemplate
  9   8   5   9  10  11  13   8  17  10  0.401199     98/100     NonOverlappingTemplate
  9  12  11   7  13   9  13   9  13   4  0.534146     95/100  *  NonOverlappingTemplate
  6  12   7   8  10  10  11  10   6  20  0.090936     100/100      NonOverlappingTemplate
  7   9  10  14  10   9  13   5  14   9  0.554420     100/100      NonOverlappingTemplate
 13   8  11  14   6   5  12  13   8  10  0.455937      98/100      NonOverlappingTemplate
  8   9   5   6  13  14   9  14  13   9  0.366918      99/100      NonOverlappingTemplate
 14   5   4  17   9   4   7  16  15   9  0.005358     100/100      NonOverlappingTemplate
  8   7  13  12  11   5  13   8  13  10  0.595549      97/100      NonOverlappingTemplate
 16   7  11  12   9  10  10   5   8  12  0.494392     100/100      NonOverlappingTemplate
  9  12   9  11  10   8  12   9   8  12  0.983453     100/100      NonOverlappingTemplate
 13   5   8  12   6  14  10  10  12  10  0.554420      98/100      NonOverlappingTemplate
  3   7   8  17  12  10   7  17   7  12  0.028817     100/100      NonOverlappingTemplate
 11   8  13   8   7  13  15   9   8   8  0.637119      97/100      NonOverlappingTemplate
 13   8  10  14  10   8  10   6   8  13  0.719747      99/100      NonOverlappingTemplate
 18  12   6   9   8   6  10  11  11   9  0.289667      97/100      NonOverlappingTemplate
 10  11   8  16  10   5   9  10  13   8  0.534146     100/100      NonOverlappingTemplate
 14  10  11  13   9   9   9  12   5   8  0.719747     100/100      NonOverlappingTemplate
 11  12   7  12  13  10   5   8  12  10  0.739918     100/100      NonOverlappingTemplate
 10   4   9  11   6  14  10  15  14   7  0.213309      99/100      NonOverlappingTemplate
 10  10  11   9  10  10   5  12  15   8  0.739918      99/100      NonOverlappingTemplate
 14  16   7   9   8   9  11   9   7  10  0.554420      97/100      NonOverlappingTemplate
 10   8  13  11  12  11   8   3  16   8  0.262249      97/100      NonOverlappingTemplate
  6  10  13  12  10  11  12   6   7  13  0.657933     100/100      NonOverlappingTemplate
 12  10  11   6   9   9  11  10   7  15  0.759756      99/100      NonOverlappingTemplate
 12  11  12  13   7  15  10   5   5  10  0.334538      96/100      NonOverlappingTemplate
 12   7   8   6   7  11  19  10   7  13  0.115387      98/100      NonOverlappingTemplate
  9  15   8  10   7   8  11  10  12  10  0.851383     100/100      NonOverlappingTemplate
 10   9  11  11   9   9   7  11   9  14  0.955835      99/100      NonOverlappingTemplate
  7  10  13  11   8  11   9  12  11   8  0.946308     100/100      NonOverlappingTemplate
  8  10  15  12   9  11   9   3  16   7  0.162606     100/100      NonOverlappingTemplate
  7  13  13  15   7  13   6   8  11   7  0.350485      99/100      NonOverlappingTemplate
  8  14   8   4  14   9  12  13   4  14  0.115387      99/100      NonOverlappingTemplate
  7  16   9   8   9  11  11  11   7  11  0.699313      99/100      NonOverlappingTemplate
 14  10   9  10   5   5  14   5  15  13  0.115387      99/100      NonOverlappingTemplate
  6   5   8  11  12  11  14  12  10  11  0.616305      99/100      NonOverlappingTemplate
 10  14  13   9  13   8   8   9   5  11  0.637119      99/100      NonOverlappingTemplate
 17   5  15  11  11  16   9   7   4   5  0.013569     100/100      NonOverlappingTemplate
 15  13   6   9  10  10   7  11   8  11  0.678686      99/100      NonOverlappingTemplate
 10   9   9   8  10   8  11   9  11  15  0.924076      99/100      NonOverlappingTemplate
 18  10  12   7  14  13   6   9   3   8  0.045675     100/100      NonOverlappingTemplate
  5  10   9  15  10   7  10   9  13  12  0.595549      99/100      NonOverlappingTemplate
  9  13   8  13  11   5   8  14  10   9  0.637119      99/100      NonOverlappingTemplate
  8  10   7  20  13   6  11   6  13   6  0.035174      97/100      NonOverlappingTemplate
  9  13   6   7   8  12   9   5  19  12  0.080519      96/100      NonOverlappingTemplate
 16   8   8  10   9  10   9  12   7  11  0.739918      98/100      NonOverlappingTemplate
 10   8   9   7  13  13  17  10   5   8  0.275709      99/100      NonOverlappingTemplate
 10   8   8  12  13   5  16   8   8  12  0.401199     100/100      NonOverlappingTemplate
  9  10   9  14   6   9   9  11  12  11  0.897763      99/100      NonOverlappingTemplate
  8  10   9   9   9  13  15   7   7  13  0.657933      98/100      NonOverlappingTemplate
 11  11  13   8  11   6   8   9  13  10  0.867692      99/100      NonOverlappingTemplate
 10  14   8  12   9   9   8   8  11  11  0.935716      99/100      NonOverlappingTemplate
 10  14  11  12   4  11   7  15  10   6  0.289667      98/100      NonOverlappingTemplate
  5   9   8   9  13  10  15  10   8  13  0.554420     100/100      NonOverlappingTemplate
  6  15   7  11  12  14   9   9   9   8  0.554420     100/100      NonOverlappingTemplate
 16   9   9  14   5   9   9   8   8  13  0.366918     100/100      NonOverlappingTemplate
 10   8  18  10  11   8  10   6   8  11  0.401199      99/100      NonOverlappingTemplate
 11  10   5  18  11  11   6   9  11   8  0.249284      98/100      NonOverlappingTemplate
  8  10   6  16   7   8  13  14   9   9  0.383827      98/100      NonOverlappingTemplate
  6  12  10   7  10  10  12  13   3  17  0.122325      98/100      NonOverlappingTemplate
  6  15   9  10   9   8  18   6  11   8  0.153763     100/100      NonOverlappingTemplate
 12  10  15  14   7   7  10   6   8  11  0.494392     100/100      NonOverlappingTemplate
 11   9  11  10  10  12   8  10  10   9  0.998821     100/100      NonOverlappingTemplate
 10  14   8   7   8   9  11   7  11  15  0.637119      99/100      NonOverlappingTemplate
 12  11  11   8   6  10   9  12  12   9  0.935716      98/100      NonOverlappingTemplate
  6   8   8  12   5   5  13  16   9  18  0.026948     100/100      NonOverlappingTemplate
 10  12   9  13   5  11  11  12  10   7  0.798139      98/100      NonOverlappingTemplate
 10  10  12  11  12  10  10   9   3  13  0.657933     100/100      NonOverlappingTemplate
  9   8  11  11  12   8   8  12  11  10  0.983453     100/100      NonOverlappingTemplate
  7  10   7  16  10  13   6  12   7  12  0.383827     100/100      NonOverlappingTemplate
 11  11   9   9  13   4   9  14  12   8  0.595549      98/100      NonOverlappingTemplate
  8  10   6  10   8  12  15   9  13   9  0.699313      98/100      NonOverlappingTemplate
  9   5  11  12  17   4   9   8  14  11  0.129620     100/100      NonOverlappingTemplate
 10  12  11  11  13   9  11  10   8   5  0.867692      99/100      NonOverlappingTemplate
  3   7  18  16   4   9   9  11  13  10  0.014550     100/100      NonOverlappingTemplate
  9   7  12   5  12  15  13   6   7  14  0.224821      98/100      NonOverlappingTemplate
  9  17   7   9  11   9  13   9  12   4  0.262249      97/100      NonOverlappingTemplate
  5   5  15  10   7  14   7  15   6  16  0.028817      99/100      NonOverlappingTemplate
  6  10  10  14  10   9  13   5  14   9  0.494392     100/100      NonOverlappingTemplate
 13   9  10   7  12   9   8   9  12  11  0.946308      98/100      OverlappingTemplate
 11  10   8  11  10  17   6  11  11   5  0.366918     100/100      Universal
 13  10  13   7   5  16   6  14   8   8  0.171867      97/100      ApproximateEntropy
  6   4   2   3   6   6   4   2   3   6  0.689019      39/42       RandomExcursions
  7   5   1   7   2   4   1   6   3   6  0.186566      41/42       RandomExcursions
  7   5   7   3   6   1   6   6   0   1  0.057146      40/42       RandomExcursions
  6   3   3   5   6   1   3   8   5   2  0.311542      41/42       RandomExcursions
  9   2   2   4   5   7   2   2   5   4  0.162606      40/42       RandomExcursions
  4   5   4   5   6   1   7   6   2   2  0.437274      42/42       RandomExcursions
  2   4   3   5   5   6   4   3   6   4  0.911413      42/42       RandomExcursions
  5   5   4   5   3   6   2   4   6   2  0.834308      41/42       RandomExcursions
  3   4   6   8   6   2   5   1   4   3  0.350485      42/42       RandomExcursionsVariant
  4   6   7   9   3   1   1   3   2   6  0.057146      42/42       RandomExcursionsVariant
  4   9   6   4   2   7   1   3   2   4  0.122325      42/42       RandomExcursionsVariant
  4   8   5   5   1   8   2   3   2   4  0.162606      42/42       RandomExcursionsVariant
  4   8   4   3   5   3   2   6   4   3  0.637119      42/42       RandomExcursionsVariant
  6   7   5   4   2   2   5   4   5   2  0.637119      42/42       RandomExcursionsVariant
  7   5   6   4   3   4   3   1   3   6  0.585209      42/42       RandomExcursionsVariant
  5   3   6   3   8   4   3   3   6   1  0.392456      42/42       RandomExcursionsVariant
  2   3   3  10   4   5   2   5   5   3  0.186566      42/42       RandomExcursionsVariant
  6   2   2   2   4   4   5   5   5   7  0.637119      42/42       RandomExcursionsVariant
```

```
   4   8   4   0   4   8   2   6   3   3  0.105618     41/42      RandomExcursionsVariant
   7   3   4   7   3   1   2   8   5   2  0.141256     42/42      RandomExcursionsVariant
   2   8   3   6   6   3   2   2   7   3  0.213309     42/42      RandomExcursionsVariant
   3   6   3   3   6   7   8   2   0   4  0.122325     42/42      RandomExcursionsVariant
   3   5   5   6   3   3   4   5   4   4  0.980883     42/42      RandomExcursionsVariant
   3   6   5   6   1   7   5   4   3   2  0.484646     42/42      RandomExcursionsVariant
   3   8   1   5   5   7   2   5   2   4  0.242986     42/42      RandomExcursionsVariant
   3   7   4   4   0   5   2   7   6   4  0.275709     42/42      RandomExcursionsVariant
   7  11  12  10   7  12  10   8  10  13  0.911413     99/100     Serial
   8   9   9   8  10   8   9  12  17  10  0.657933     99/100     Serial
   4  13   8  11  11  12  10   7  13  11  0.595549     99/100     LinearComplexity


 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 The minimum pass rate for each statistical test with the exception of the
 random excursion (variant) test is approximately = 96 for a
 sample size = 100 binary sequences.

 The minimum pass rate for the random excursion (variant) test
 is approximately = 39 for a sample size = 42 binary sequences.

 For further guidelines construct a probability table using the MAPLE program
 provided in the addendum section of the documentation.
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Table 3: testing a sequence of 100 bitstreams per 512.000 bits generated by a radio frequency dongle and converted into bits by rtl_entropy

```
 ------------------------------------------------------------------------------
 RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
 ------------------------------------------------------------------------------
    generator is <data/radioentropy.bin>
 ------------------------------------------------------------------------------
  C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
 ------------------------------------------------------------------------------
   9  17   9  14   8  11   6  10  13   3  0.102526    100/100     Frequency
   9   6  14  14   8  12  12   6   8  11  0.514124     98/100     BlockFrequency
  13  12  14  10   7   6  10  10  13   5  0.455937    100/100     CumulativeSums
  14  10  10  10   9  11  13   6   9   8  0.851383    100/100     CumulativeSums
   8   5  14   6  10   8   9  11  13  16  0.262249     99/100     Runs
   6  12   8  12  13   9  15   8  10   7  0.574903     99/100     LongestRun
  10  10  14  10  12  11   8   8   9   8  0.946308    100/100     Rank
   9   8  10   9   8   9  10  11  12  14  0.955835     99/100     FFT
   8  13   6  13  11  15  10   9   8   7  0.554420     99/100     NonOverlappingTemplate
  15   9  10   9   5  13   7  15   5  12  0.191687     99/100     NonOverlappingTemplate
  11   5  18   4   8   6  13  14   6  15  0.011791     99/100     NonOverlappingTemplate
  14   8   7  14  11  10   9  10  11   6  0.699313     99/100     NonOverlappingTemplate
   8   7   9  11   9  11   8  16   9  12  0.719747     99/100     NonOverlappingTemplate
  11   9  11   5   9  11   8  11  16   9  0.616305    100/100     NonOverlappingTemplate
   8  10   7  13   9  11  11   7  12  12  0.897763     99/100     NonOverlappingTemplate
  13   6  11   8  10  11  16  11   5   9  0.401199    100/100     NonOverlappingTemplate
   7  10  12   9  10   8  12  11  12   9  0.971699     99/100     NonOverlappingTemplate
  11  10  12   8   9  17   7   5   9  12  0.366918     99/100     NonOverlappingTemplate
   9  13  13   7   7   9  13  10   9  10  0.851383    100/100     NonOverlappingTemplate
  13  13   4  10   7  14  12  13   5   9  0.224821     97/100     NonOverlappingTemplate
  17   7   6   9  12   8  10   9   8  14  0.319084     97/100     NonOverlappingTemplate
  16   2   6   9  11  12   7  16  14   7  0.023545    100/100     NonOverlappingTemplate
   6  13  10  13  14  10   7   6  11  10  0.574903    100/100     NonOverlappingTemplate
  14   7   7  11  16  11  10  10   5   9  0.366918     98/100     NonOverlappingTemplate
   7   8  15  12  10  13   9  10   9   7  0.719747     99/100     NonOverlappingTemplate
  11  14   6  10  11  11  16   2   9  10  0.137282     98/100     NonOverlappingTemplate
  10  12  10  12  15   8   8  10   7   8  0.798139    100/100     NonOverlappingTemplate
   9  11  10   7  13   9   7   9  15  10  0.779188     99/100     NonOverlappingTemplate
  13   9  12  11   6  10  13   9   4  13  0.474986    100/100     NonOverlappingTemplate
   8  12  16   7   9   9   8   8  16   7  0.289667     98/100     NonOverlappingTemplate
  10   7  10   9  13   9  11   9   6  16  0.595549    100/100     NonOverlappingTemplate
  14  13  12  11   8   1  11   9   7  14  0.115387    100/100     NonOverlappingTemplate
   7  11   7  14   6   8  15  10  16   6  0.153763     98/100     NonOverlappingTemplate
  10   6   4  12  11  12  11   9  14  11  0.534146    100/100     NonOverlappingTemplate
   7   5   6  14  10   4  16  12  12  14  0.062821    100/100     NonOverlappingTemplate
   8  10  11   7  11  19   8  11   7   8  0.249284     99/100     NonOverlappingTemplate
   6  14   9  10  10   9   6  12  12  12  0.719747    100/100     NonOverlappingTemplate
  10   4  16  12   9  15   9  11   6   8  0.191687     98/100     NonOverlappingTemplate
  10   6   9  12  11  10  13  12   8   9  0.911413     97/100     NonOverlappingTemplate
   3  17   9  12  13  11   7  11   7  10  0.153763    100/100     NonOverlappingTemplate
  13  11  10  11  10  12  10   7   9   7  0.946308     97/100     NonOverlappingTemplate
   8  12   6   9  11  13   8  12   9  12  0.851383     99/100     NonOverlappingTemplate
   7  14  14  11  10  12   6   8   6  12  0.474986    100/100     NonOverlappingTemplate
   7   9   7  10  15  13   5   8  15  11  0.289667     99/100     NonOverlappingTemplate
  10  13   6   4   9  13  12   9  13  11  0.474986    100/100     NonOverlappingTemplate
   6  19  11   5  13   6   9   9  13   9  0.066882     98/100     NonOverlappingTemplate
  14  12  17   6   8  12   8   9  10   4  0.145326     99/100     NonOverlappingTemplate
   5  11   8  14   8   6   8  16  11  13  0.236810    100/100     NonOverlappingTemplate
  10  14  15  11  11  10   3   8  10   8  0.350485     99/100     NonOverlappingTemplate
  10  13  12   8  13   9  11   7   8   9  0.897763     99/100     NonOverlappingTemplate
   6   8  11   7  13  11   6  12  17   9  0.275709     99/100     NonOverlappingTemplate
   9  13   8  11  10  12   7   9  11  10  0.964295    100/100     NonOverlappingTemplate
   9  10   9   8  11  12  14  11   9   7  0.924076     98/100     NonOverlappingTemplate
  10   4  10   7  12  11  12  13   8  13  0.574903    100/100     NonOverlappingTemplate
   4  10  13   5  13   8  11   9  16  11  0.202268     99/100     NonOverlappingTemplate
  14  10  13  10   5  14   7   3  10  14  0.122325    100/100     NonOverlappingTemplate
  12   9  14   6  11  11  11   9   9   8  0.867692    100/100     NonOverlappingTemplate
  10  11  15   8  16   9   6   5  11   9  0.275709     98/100     NonOverlappingTemplate
   8   8  14  12  13   5   7   9   8  16  0.262249     98/100     NonOverlappingTemplate
  13   8   8   1  11  14   9  12  14  10  0.137282    100/100     NonOverlappingTemplate
 11  10  12   9  11  11   8  12   6  10  0.955835     99/100      NonOverlappingTemplate
  8  11  10  11   5   9   8  11  16  11  0.595549     99/100      NonOverlappingTemplate
  9  12   8   8  10   7  12  11  12  11  0.955835     98/100      NonOverlappingTemplate
  9  13  11   6   6  15   9  11   9  11  0.616305     97/100      NonOverlappingTemplate
 11  11  16   5  11   9   9   6  11  11  0.494392     99/100      NonOverlappingTemplate
  6  11  12   8  10  10  11   7  12  13  0.851383    100/100      NonOverlappingTemplate
  7   9  13  15   8   8  13  10   8   9  0.678686     99/100      NonOverlappingTemplate
  7   8  10  14  11  11  11   8   7  13  0.798139    100/100      NonOverlappingTemplate
 10  11  13  14  10   9   7  10   7   9  0.867692     98/100      NonOverlappingTemplate
 10   9   7  12  10   3  12   7  12  18  0.108791     99/100      NonOverlappingTemplate
 15  14   5  11   8   7   9   9  13   9  0.419021     99/100      NonOverlappingTemplate
  6  12   8  10  13   8  15  10   7  11  0.616305    100/100      NonOverlappingTemplate
 11  10  12  12   9  12  12   6   5  11  0.739918    100/100      NonOverlappingTemplate
 11   7  13   7   9   9   9  11  16   8  0.616305     98/100      NonOverlappingTemplate
 15   9   9  12   9   7   7   8   9  15  0.534146     99/100      NonOverlappingTemplate
  7   9  14   7  17   4  12  10  13   7  0.115387    100/100      NonOverlappingTemplate
 10   8   9   5  15  12  11  10  12   8  0.657933     99/100      NonOverlappingTemplate
  9  15  11   8   9   9   9   9  10  11  0.935716     98/100      NonOverlappingTemplate
 16   8   8  11   8   7  15  11   8   8  0.419021     99/100      NonOverlappingTemplate
 12  15   8   4  11  10   6  11  13  10  0.383827    100/100      NonOverlappingTemplate
  8  11   7   8  16  12  13   9  10   6  0.494392    100/100      NonOverlappingTemplate
 10  11   8  13  13  10   9  11   9   6  0.897763    100/100      NonOverlappingTemplate
  8  13   6  13  11  15  10   9   8   7  0.554420     99/100      NonOverlappingTemplate
 10  14  10   6  13  15   5  11   5  11  0.224821    100/100      NonOverlappingTemplate
  7  11  11  14  10   5  10  13  10   9  0.719747     98/100      NonOverlappingTemplate
 11   8   9   6   8   8  13   7  13  17  0.304126     98/100      NonOverlappingTemplate
  7  14  11  12  19   7   7   5  11   7  0.058984     97/100      NonOverlappingTemplate
  8   9  11  10  12   9  11  11  11   8  0.994250    100/100      NonOverlappingTemplate
 12  12   2  13  10   8  12  11  12   8  0.366918     99/100      NonOverlappingTemplate
 10  12   9   9   8  12   4  14   9  13  0.574903     97/100      NonOverlappingTemplate
 16  12  11   6   8   9  11  10   8   9  0.657933    100/100      NonOverlappingTemplate
 16  14   7   7  12   9   7  11   5  12  0.249284     99/100      NonOverlappingTemplate
 12  10  10  10   8  11   8  12  11   8  0.987896     98/100      NonOverlappingTemplate
 17   3  12  13  10  10   9  10   9   7  0.202268     98/100      NonOverlappingTemplate
 16  11  10   7  10   9   6   6  12  13  0.419021     97/100      NonOverlappingTemplate
  9   9   7   9  15   8  10  11  15   7  0.574903    100/100      NonOverlappingTemplate
  6  10  12  17  14   5   8  11  10   7  0.191687    100/100      NonOverlappingTemplate
 10  11  12   9  11  10  10   8  11   8  0.996335    100/100      NonOverlappingTemplate
 15  15  16  10   6   8   6  12   5   7  0.066882     99/100      NonOverlappingTemplate
 17  16   9  11   8  12  11   5   6   5  0.062821     98/100      NonOverlappingTemplate
 11  12   6  12   3   6  11  10  11  18  0.075719     98/100      NonOverlappingTemplate
 12  11  11  15   6  12  10   9   7   7  0.637119     98/100      NonOverlappingTemplate
  6  16   9   6  15  10  15   5  12   6  0.058984    100/100      NonOverlappingTemplate
 14  14  10  11   6   8   9  12   7   9  0.657933     99/100      NonOverlappingTemplate
  9  11   8  10   9  15   7   8  13  10  0.798139     99/100      NonOverlappingTemplate
 11  10  12   7   6   8  15   9  15   7  0.401199     99/100      NonOverlappingTemplate
 10  18  12  10   6   8   8   6  10  12  0.262249    100/100      NonOverlappingTemplate
 10  10  12  11  13   9   9   7  10   9  0.978072     99/100      NonOverlappingTemplate
  9   6  13  11  15   9   6   7   7  17  0.137282    100/100      NonOverlappingTemplate
  7  13   7  13   8  11  11   6   6  18  0.129620     99/100      NonOverlappingTemplate
 11  13  12   8  12   7  14   5  14   4  0.191687    100/100      NonOverlappingTemplate
  7  11  15   7   6  14  10  11  12   7  0.437274    100/100      NonOverlappingTemplate
 14  11  12   7   9  14  10   3  12   8  0.319084     99/100      NonOverlappingTemplate
  7  15   6  18  12   7  10   6  11   8  0.096578    100/100      NonOverlappingTemplate
 10  11   5  13  11  13  12   7   7  11  0.657933     98/100      NonOverlappingTemplate
  6  11   9  11  12   8  10  13  11   9  0.924076    100/100      NonOverlappingTemplate
 12   6  14  11  11  12   9   9   6  10  0.739918     96/100      NonOverlappingTemplate
  8  11   8   9  12   8  13   7   7  17  0.401199     98/100      NonOverlappingTemplate
  8  12  10   8   6   9  13  13   9  12  0.816537    100/100      NonOverlappingTemplate
 10   7   4  11  11  11  12  13   6  15  0.334538     98/100      NonOverlappingTemplate
  7  11  11   9  10  15  12   8  10   7  0.798139     99/100      NonOverlappingTemplate
  3  10  14   6  10  10  11   7  16  13  0.137282     98/100      NonOverlappingTemplate
 11   9   8  11  16   8   7  11   7  12  0.637119     99/100      NonOverlappingTemplate
  7   9   7  12  11  12  13  11   7  11  0.851383    100/100      NonOverlappingTemplate
 11  15   8   6   6  11  15  10   8  10  0.419021     97/100      NonOverlappingTemplate
 10  12  10  10  11   7   7   9  11  13  0.946308     98/100      NonOverlappingTemplate
 14   4   9   5  11  13  11   9   9  15  0.236810     99/100      NonOverlappingTemplate
 10  11  11  11   8  12  10   9  10   8  0.996335    100/100      NonOverlappingTemplate
 12  14   7   7  11  10  11  12   9   7  0.798139     99/100      NonOverlappingTemplate
 10  11   8  13   6  16  10   7   8  11  0.534146    100/100      NonOverlappingTemplate
  5   5   9  11  11  13  17   8  12   9  0.213309    100/100      NonOverlappingTemplate
 11  12   6   7   9   7  14  12  12  10  0.699313     98/100      NonOverlappingTemplate
 12  11  12   7  15  11   8  12   7   5  0.474986    100/100      NonOverlappingTemplate
  5  10   9  14   7  10   5  10  19  11  0.071177    100/100      NonOverlappingTemplate
  7  13  11  11   9  13   5  11  13   7  0.595549    100/100      NonOverlappingTemplate
  8   5   6  10  13  10  15  13   9  11  0.437274    100/100      NonOverlappingTemplate
 11   9   8   9  10  11   8  16   5  13  0.514124     98/100      NonOverlappingTemplate
  6  10   9  12   9  10   6  15  13  10  0.616305    100/100      NonOverlappingTemplate
 12   8   8  12   3  14   9   8  10  16  0.202268    100/100      NonOverlappingTemplate
  6  15   7  11  13  10  11   8  11   8  0.637119     99/100      NonOverlappingTemplate
 10   6   9  12  11  11   6  15  11   9  0.678686    100/100      NonOverlappingTemplate
 12  14   8   8  10  12   7   8  11  10  0.867692     99/100      NonOverlappingTemplate
 14   4   7   6  13  12   9  10  10  15  0.236810    100/100      NonOverlappingTemplate
 12   9  12  13  13  10   9  11   4   7  0.595549     99/100      NonOverlappingTemplate
 13   7   7   4  14  10  12  15   5  13  0.115387     97/100      NonOverlappingTemplate
  7   9  16   6  14  11   7  12  11   7  0.334538    100/100      NonOverlappingTemplate
 10   7  11   9  10  11  11   5  11  15  0.699313    100/100      NonOverlappingTemplate
  8  10   8   6   6  12   6  18  15  11  0.090936    100/100      NonOverlappingTemplate
 12   6   9  16   9  10  15   7   9   7  0.334538    100/100      NonOverlappingTemplate
  9   9  14   6   8   9  16   7  13   9  0.401199     99/100      NonOverlappingTemplate
  9  11  11  11  14  10   8   8   8  10  0.955835    100/100      NonOverlappingTemplate
  6  11   6  10  16   5  12   7  13  14  0.153763     98/100      NonOverlappingTemplate
 12  14   9  15  10   6   9  10   6   9  0.534146     99/100      NonOverlappingTemplate
 11  10   5  11  12   7  12   5  15  12  0.366918     99/100      NonOverlappingTemplate
  6  14  11   8   9  10  15   6   9  12  0.494392    100/100      NonOverlappingTemplate
 10  11   8  13  13  10   9  11   9   6  0.897763    100/100      NonOverlappingTemplate
  9   9  11  12  13  11   8  12   6   9  0.897763    100/100      OverlappingTemplate
 11   7   2  14  15  11  13  10   8   9  0.162606    100/100     Universal
 12  10   9   8  12   9   7  16   7  10  0.657933     99/100     ApproximateEntropy
  4   2   7   1   3   9   3   7   6   3  0.072289     45/45      RandomExcursions
 11   4   6   3   4   3   5   2   4   3  0.084294     44/45      RandomExcursions
  7   4   7   3   2   7   4   1   6   4  0.258961     44/45      RandomExcursions
  5   3   3   3   4   4   9   4   5   5  0.559523     44/45      RandomExcursions
  8   6   2   8   3   3   6   4   3   2  0.174249     44/45      RandomExcursions
  5   2   4   4   2   7   3   7   5   6  0.509162     45/45      RandomExcursions
  4   4   5   3   5   8   4   6   2   4  0.663130     45/45      RandomExcursions
  4   2   6   5   8   3   3   6   3   5  0.509162     44/45      RandomExcursions
  5   4   5   7   6   5   7   2   1   3  0.371101     45/45      RandomExcursionsVariant
  4   5   3   7  10   4   3   2   4   3  0.151616     45/45      RandomExcursionsVariant
  4   5   8   8   5   1   1   4   4   5  0.151616     45/45      RandomExcursionsVariant
  2   6  12   5   2   4   3   4   1   6  0.006783     45/45      RandomExcursionsVariant
  1   6   9   6   3   4   4   3   1   8  0.044942     45/45      RandomExcursionsVariant
  2   6   7   4   2   5   7   5   3   4  0.509162     45/45      RandomExcursionsVariant
  1   5   3   5   5   8   2   9   4   3  0.098036     45/45      RandomExcursionsVariant
  1   4   5   4   5   5   4  10   3   4  0.199580     45/45      RandomExcursionsVariant
  3   5   1   7   2   3   8   6   6   4  0.199580     45/45      RandomExcursionsVariant
  7   5   4   3   7   4   3   4   2   6  0.611108     44/45      RandomExcursionsVariant
  3   6   6   4   5   7   5   3   5   1  0.559523     44/45      RandomExcursionsVariant
  1   5   3   7   4   4   5   3   5   8  0.371101     45/45      RandomExcursionsVariant
  3   5   2   3   4   4   6   7   5   6  0.714660     45/45      RandomExcursionsVariant
  6   4   3   1   5   6   4   6   7   3  0.509162     45/45      RandomExcursionsVariant
  7   3   6   4   3   6   3   3   3   7  0.559523     45/45      RandomExcursionsVariant
  8   5   3   5   4   3   3   6   3   5  0.663130     45/45      RandomExcursionsVariant
  8   4   5   3   5   3   4   3   5   5  0.764655     45/45      RandomExcursionsVariant
  9   4   4   3   4   4   4   5   3   5  0.611108     44/45      RandomExcursionsVariant
 10   6  10  13   9  13   8  10   8  13  0.816537     99/100     Serial
  8  10   8  10   7  13  13   6  12  13  0.699313    100/100     Serial
  9  13  16   7  10   3  12  13  11   6  0.145326    100/100     LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 42 for a sample size = 45 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Table 4: first time testing of dev/random

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <data/random.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  2   1   1   1   0   1   3   1   0   0  0.534146      9/10      Frequency
  1   0   0   1   1   1   2   4   0   0  0.122325     10/10      BlockFrequency
  2   1   0   1   1   1   0   1   2   1  0.911413      9/10      CumulativeSums
  2   1   0   1   2   0   0   2   1   1  0.739918      9/10      CumulativeSums
  1   1   1   2   0   2   0   1   1   1  0.911413     10/10      Runs
  0   2   0   2   0   3   1   0   1   1  0.350485     10/10      LongestRun
  0   0   2   1   0   2   1   0   3   1  0.350485     10/10      Rank
  1   2   0   0   2   1   1   1   2   0  0.739918     10/10      FFT
  1   2   1   2   0   0   2   1   0   1  0.739918     10/10      NonOverlappingTemplate
  0   0   0   1   0   1   3   3   0   2  0.122325     10/10      NonOverlappingTemplate
  1   0   1   0   0   1   3   2   2   0  0.350485     10/10      NonOverlappingTemplate
  1   2   0   0   0   1   1   0   2   3  0.350485     10/10      NonOverlappingTemplate
  0   1   0   0   3   1   1   1   3   0  0.213309     10/10      NonOverlappingTemplate
  3   0   1   0   0   2   1   2   1   0  0.350485      9/10      NonOverlappingTemplate
  2   1   1   0   1   1   1   1   1   1  0.991468     10/10      NonOverlappingTemplate
  1   0   0   0   2   0   3   3   0   1  0.122325     10/10      NonOverlappingTemplate
  1   2   0   1   0   2   0   0   1   3  0.350485     10/10      NonOverlappingTemplate
  0   3   1   0   1   1   1   0   2   1  0.534146     10/10      NonOverlappingTemplate
  4   1   0   1   0   2   1   0   0   1  0.122325     10/10      NonOverlappingTemplate
  2   1   0   1   1   2   1   2   0   0  0.739918     10/10      NonOverlappingTemplate
  1   0   2   1   1   0   1   0   2   2  0.739918     10/10      NonOverlappingTemplate
  1   0   1   2   1   2   0   1   1   1  0.911413     10/10      NonOverlappingTemplate
  2   0   0   2   0   2   0   3   1   0  0.213309     10/10      NonOverlappingTemplate
  0   0   1   0   0   2   1   2   2   2  0.534146     10/10      NonOverlappingTemplate
  1   1   3   0   1   1   2   0   0   1  0.534146     10/10      NonOverlappingTemplate
  1   2   0   2   1   1   2   0   1   0  0.739918     10/10      NonOverlappingTemplate
  2   1   0   1   2   0   1   0   0   3  0.350485      8/10      NonOverlappingTemplate
  0   0   0   1   1   1   2   0   3   2  0.350485     10/10      NonOverlappingTemplate
  2   1   2   1   1   1   0   1   1   0  0.911413     10/10      NonOverlappingTemplate
  0   1   3   1   0   0   2   0   1   2  0.350485     10/10      NonOverlappingTemplate
  1   0   0   0   0   2   1   4   0   2  0.066882     10/10      NonOverlappingTemplate
  2   1   2   0   2   0   0   1   2   0  0.534146     10/10      NonOverlappingTemplate
  0   2   3   0   0   1   1   0   1   2  0.350485     10/10      NonOverlappingTemplate
  0   2   1   1   0   1   1   0   3   1  0.534146     10/10      NonOverlappingTemplate
  0   0   1   2   2   0   0   3   1   1  0.350485     10/10      NonOverlappingTemplate
  1   2   1   1   1   1   0   2   1   0  0.911413      9/10      NonOverlappingTemplate
  3   1   1   0   1   0   2   0   1   1  0.534146     10/10      NonOverlappingTemplate
  3   0   1   1   1   0   0   1   0   3  0.213309     10/10      NonOverlappingTemplate
  2   2   0   0   0   1   1   0   1   3  0.350485     10/10      NonOverlappingTemplate
  1   2   3   0   1   2   1   0   0   0  0.350485     10/10      NonOverlappingTemplate
```

```
   1    0    2    2    0    0    2    1    0    2  0.534146     10/10      NonOverlappingTemplate
   4    0    0    0    1    0    0    4    0    1  0.004301      9/10      NonOverlappingTemplate
   0    1    2    2    1    0    2    1    0    1  0.739918     10/10      NonOverlappingTemplate
   1    2    1    2    0    0    1    1    2    0  0.739918      9/10      NonOverlappingTemplate
   0    0    0    2    2    0    2    1    0    3  0.213309     10/10      NonOverlappingTemplate
   0    0    2    1    2    1    2    0    1    1  0.739918     10/10      NonOverlappingTemplate
   0    1    1    2    0    1    1    3    0    1  0.534146     10/10      NonOverlappingTemplate
   3    0    1    0    3    0    1    1    0    1  0.213309      9/10      NonOverlappingTemplate
   1    1    1    0    1    3    0    2    1    0  0.534146     10/10      NonOverlappingTemplate
   0    0    2    0    3    2    0    1    1    1  0.350485     10/10      NonOverlappingTemplate
   0    0    3    1    1    0    1    1    1    2  0.534146     10/10      NonOverlappingTemplate
   0    1    1    0    0    1    4    1    2    0  0.122325     10/10      NonOverlappingTemplate
   0    0    0    2    0    0    3    1    2    2  0.213309     10/10      NonOverlappingTemplate
   1    1    0    2    2    0    1    1    1    1  0.911413     10/10      NonOverlappingTemplate
   0    0    2    1    2    2    0    1    2    0  0.534146     10/10      NonOverlappingTemplate
   1    1    2    1    1    0    0    1    1    2  0.911413     10/10      NonOverlappingTemplate
   2    3    0    2    0    0    2    0    0    1  0.213309     10/10      NonOverlappingTemplate
   0    3    1    1    1    0    1    1    1    1  0.739918     10/10      NonOverlappingTemplate
   1    0    2    0    1    1    2    0    2    1  0.739918     10/10      NonOverlappingTemplate
   0    2    1    0    0    2    1    2    1    1  0.739918     10/10      NonOverlappingTemplate
   1    3    0    0    1    1    0    0    1    3  0.213309     10/10      NonOverlappingTemplate
   2    0    1    0    1    2    0    2    1    1  0.739918     10/10      NonOverlappingTemplate
   1    0    0    0    3    1    2    1    0    2  0.350485      9/10      NonOverlappingTemplate
   2    0    0    0    1    2    3    1    0    1  0.350485     10/10      NonOverlappingTemplate
   2    1    2    1    0    0    2    1    1    0  0.739918     10/10      NonOverlappingTemplate
   3    1    0    0    2    0    1    1    0    2  0.350485      9/10      NonOverlappingTemplate
   1    1    0    0    0    4    0    0    3    1  0.035174     10/10      NonOverlappingTemplate
   1    2    0    1    0    0    2    2    2    0  0.534146     10/10      NonOverlappingTemplate
   0    1    1    1    2    1    1    0    3    0  0.534146     10/10      NonOverlappingTemplate
   1    1    1    2    1    1    0    2    0    1  0.911413     10/10      NonOverlappingTemplate
   0    0    1    2    2    0    2    1    1    1  0.739918     10/10      NonOverlappingTemplate
   0    0    1    0    0    0    2    2    3    2  0.213309     10/10      NonOverlappingTemplate
   0    2    0    2    2    2    1    1    0    0  0.534146     10/10      NonOverlappingTemplate
   1    2    2    1    1    0    0    0    3    0  0.350485     10/10      NonOverlappingTemplate
   1    0    3    0    2    2    0    1    1    0  0.350485     10/10      NonOverlappingTemplate
   0    0    0    0    1    2    3    1    1    2  0.350485     10/10      NonOverlappingTemplate
   2    0    0    1    1    2    2    1    0    1  0.739918     10/10      NonOverlappingTemplate
   3    0    0    1    0    0    1    1    2    2  0.350485     10/10      NonOverlappingTemplate
   0    0    0    1    1    1    3    2    2    0  0.350485     10/10      NonOverlappingTemplate
   1    1    2    2    0    1    0    1    2    0  0.739918     10/10      NonOverlappingTemplate
   1    0    2    0    2    0    1    2    1    1  0.739918     10/10      NonOverlappingTemplate
   0    0    2    0    1    1    2    0    2    2  0.534146     10/10      NonOverlappingTemplate
   1    2    1    2    0    0    2    1    0    1  0.739918     10/10      NonOverlappingTemplate
   2    1    0    2    0    1    1    2    1    0  0.739918     10/10      NonOverlappingTemplate
   0    2    1    0    2    1    1    1    1    1  0.911413     10/10      NonOverlappingTemplate
   1    3    1    2    0    0    2    1    0    0  0.350485     10/10      NonOverlappingTemplate
   2    1    2    1    2    0    0    2    0    0  0.534146     10/10      NonOverlappingTemplate
   0    0    1    2    0    1    1    1    2    2  0.739918     10/10      NonOverlappingTemplate
   0    0    1    1    2    2    0    3    1    0  0.350485     10/10      NonOverlappingTemplate
   2    1    2    0    2    1    0    0    1    1  0.739918     10/10      NonOverlappingTemplate
   1    2    1    2    2    0    0    1    1    0  0.739918     10/10      NonOverlappingTemplate
   0    1    1    3    1    0    2    0    2    0  0.350485     10/10      NonOverlappingTemplate
   0    2    1    0    1    0    4    0    1    1  0.122325     10/10      NonOverlappingTemplate
   3    2    0    1    0    2    1    0    1    0  0.350485      9/10      NonOverlappingTemplate
   0    2    2    1    1    0    1    0    2    1  0.739918     10/10      NonOverlappingTemplate
   0    2    2    0    2    1    0    0    1    2  0.534146     10/10      NonOverlappingTemplate
   0    1    1    2    1    1    1    2    1    0  0.911413     10/10      NonOverlappingTemplate
   0    0    2    0    1    2    2    1    0    2  0.534146     10/10      NonOverlappingTemplate
   0    1    1    2    1    1    0    1    1    2  0.911413     10/10      NonOverlappingTemplate
   2    1    2    2    1    1    0    0    1    0  0.739918     10/10      NonOverlappingTemplate
   2    3    1    1    1    0    1    0    0    1  0.534146     10/10      NonOverlappingTemplate
   1    2    1    0    0    2    1    2    1    0  0.739918     10/10      NonOverlappingTemplate
   1    1    3    1    2    0    0    0    2    0  0.350485     10/10      NonOverlappingTemplate
   2    1    1    1    0    1    0    3    0    1  0.534146     10/10      NonOverlappingTemplate
   1    2    0    1    2    1    1    0    0    2  0.739918     10/10      NonOverlappingTemplate
   0    2    0    0    1    1    1    2    2    1  0.739918     10/10      NonOverlappingTemplate
   0    1    2    0    4    1    0    0    1    1  0.122325     10/10      NonOverlappingTemplate
   1    0    1    2    2    1    2    0    0    1  0.739918     10/10      NonOverlappingTemplate
   0    1    2    2    1    2    0    1    0    1  0.739918     10/10      NonOverlappingTemplate
   1    1    1    3    1    0    0    0    1    2  0.534146     10/10      NonOverlappingTemplate
   0    0    2    1    2    2    0    1    1    1  0.739918     10/10      NonOverlappingTemplate
   1    0    1    0    0    1    2    2    2    1  0.739918     10/10      NonOverlappingTemplate
   3    3    0    0    0    2    0    1    0    1  0.122325     10/10      NonOverlappingTemplate
   2    1    0    2    1    1    1    0    2    0  0.739918     10/10      NonOverlappingTemplate
   1    2    0    0    2    2    0    3    0    0  0.213309     10/10      NonOverlappingTemplate
   2    1    0    1    2    2    0    0    2    0  0.534146      9/10      NonOverlappingTemplate
   0    1    0    1    2    1    2    0    0    3  0.350485     10/10      NonOverlappingTemplate
   0    1    1    1    2    1    2    1    1    0  0.911413     10/10      NonOverlappingTemplate
   2    0    1    1    0    0    1    2    3    0  0.350485     10/10      NonOverlappingTemplate
   1    0    0    0    1    2    1    1    3    1  0.534146     10/10      NonOverlappingTemplate
   0    1    1    1    1    1    0    0    3    2  0.534146     10/10      NonOverlappingTemplate
   1    2    0    0    1    2    0    1    2    1  0.739918     10/10      NonOverlappingTemplate
   0    0    1    1    0    2    2    2    0    2  0.534146     10/10      NonOverlappingTemplate
   2    1    1    0    1    2    0    0    1    2  0.739918     10/10      NonOverlappingTemplate
   0    1    2    2    2    0    2    0    0    1  0.534146     10/10      NonOverlappingTemplate
   1    1    1    0    0    1    3    2    0    1  0.534146     10/10      NonOverlappingTemplate
   2    2    0    0    1    1    0    1    2    1  0.739918      9/10      NonOverlappingTemplate
   2    1    1    0    2    1    1    0    0    2  0.739918     10/10      NonOverlappingTemplate
   0    1    0    2    1    3    1    0    2    0  0.350485     10/10      NonOverlappingTemplate
   0    2    0    2    1    0    1    2    1    1  0.739918     10/10      NonOverlappingTemplate
   0    2    0    3    0    1    1    0    2    1  0.350485     10/10      NonOverlappingTemplate
   0    1    0    1    1    2    1    2    1    1  0.911413     10/10      NonOverlappingTemplate
   0    0    3    0    2    0    0    2    2    1  0.213309     10/10      NonOverlappingTemplate
   2    1    0    1    1    2    1    0    0    2  0.739918     10/10      NonOverlappingTemplate
   2    0    0    2    1    1    1    1    1    1  0.911413     10/10      NonOverlappingTemplate
   1    1    0    1    3    0    0    0    3    1  0.213309     10/10      NonOverlappingTemplate
   1    0    0    2    0    2    3    1    1    0  0.350485     10/10      NonOverlappingTemplate
   1    0    0    3    2    1    0    1    1    1  0.534146      10/10      NonOverlappingTemplate
   1    0    0    1    1    0    2    3    1    1  0.534146      10/10      NonOverlappingTemplate
   2    0    2    0    0    1    1    1    1    2  0.739918       9/10      NonOverlappingTemplate
   0    0    3    2    0    2    1    2    0    0  0.213309      10/10      NonOverlappingTemplate
   1    0    2    1    1    2    1    2    0    0  0.739918      10/10      NonOverlappingTemplate
   0    0    0    2    1    1    2    2    0    2  0.534146      10/10      NonOverlappingTemplate
   1    2    0    1    2    1    1    1    1    0  0.911413      10/10      NonOverlappingTemplate
   2    2    1    1    0    1    1    1    1    0  0.911413      10/10      NonOverlappingTemplate
   1    0    1    0    0    4    0    2    1    1  0.122325      10/10      NonOverlappingTemplate
   1    1    0    3    0    1    2    1    0    1  0.534146      10/10      NonOverlappingTemplate
   1    0    1    1    0    2    0    0    0    5  0.008879      10/10      NonOverlappingTemplate
   2    2    0    1    0    1    2    0    1    1  0.739918       9/10      NonOverlappingTemplate
   1    1    0    1    1    1    1    0    1    3  0.739918      10/10      NonOverlappingTemplate
   1    0    2    2    1    2    0    0    1    1  0.739918      10/10      NonOverlappingTemplate
   0    1    0    2    2    0    0    3    1    1  0.350485      10/10      NonOverlappingTemplate
   1    0    3    1    0    1    0    0    2    2  0.350485      10/10      NonOverlappingTemplate
   1    2    1    0    0    1    2    0    2    1  0.739918      10/10      NonOverlappingTemplate
   0    2    1    1    1    1    3    0    0    1  0.534146      10/10      NonOverlappingTemplate
   0    0    2    0    1    1    2    0    2    2  0.534146      10/10      NonOverlappingTemplate
   2    1    2    1    1    2    0    0    1    0  0.739918      10/10      OverlappingTemplate
   1    0    2    2    0    1    1    1    1    1  0.911413      10/10      Universal
   1    1    2    1    1    1    0    2    0    1  0.911413      10/10      ApproximateEntropy
   1    0    0    1    0    0    2    0    1    0     ----        5/5       RandomExcursions
   1    0    1    2    0    0    0    1    0    0     ----        4/5       RandomExcursions
   2    0    0    1    0    0    1    1    0    0     ----        5/5       RandomExcursions
   0    2    2    0    0    0    1    0    0    0     ----        5/5       RandomExcursions
   0    1    1    0    0    0    1    1    1    0     ----        5/5       RandomExcursions
   0    1    0    1    1    0    0    0    1    1     ----        5/5       RandomExcursions
   0    0    1    0    2    0    0    0    0    2     ----        5/5       RandomExcursions
   0    1    0    1    1    0    1    0    0    1     ----        5/5       RandomExcursions
   0    0    1    0    1    0    1    0    1    1     ----        5/5       RandomExcursionsVariant
   0    0    1    0    0    1    0    1    0    2     ----        5/5       RandomExcursionsVariant
   0    0    1    0    0    0    1    0    3    0     ----        5/5       RandomExcursionsVariant
   0    0    0    1    1    1    1    0    0    1     ----        5/5       RandomExcursionsVariant
   0    0    0    1    1    2    0    1    0    0     ----        5/5       RandomExcursionsVariant
   0    0    0    1    1    2    0    0    1    0     ----        5/5       RandomExcursionsVariant
   1    0    0    1    0    0    2    1    0    0     ----        5/5       RandomExcursionsVariant
   1    0    0    0    1    1    1    0    1    0     ----        5/5       RandomExcursionsVariant
   0    2    0    1    0    1    0    0    0    1     ----        5/5       RandomExcursionsVariant
   1    2    0    1    0    0    0    0    1    0     ----        5/5       RandomExcursionsVariant
   0    1    1    0    0    1    0    1    0    1     ----        5/5       RandomExcursionsVariant
   0    0    0    0    0    0    1    1    2    1     ----        5/5       RandomExcursionsVariant
   0    0    0    0    0    1    1    0    2    1     ----        5/5       RandomExcursionsVariant
   0    0    0    0    1    0    1    1    1    1     ----        5/5       RandomExcursionsVariant
   0    0    1    0    0    0    2    1    1    0     ----        5/5       RandomExcursionsVariant
   0    0    0    1    1    2    0    0    1    0     ----        5/5       RandomExcursionsVariant
   0    0    0    1    0    3    0    1    0    0     ----        5/5       RandomExcursionsVariant
   0    0    0    0    0    2    1    1    1    0     ----        5/5       RandomExcursionsVariant
   1    1    1    0    2    1    1    0    1    2  0.911413      10/10      Serial
   0    1    1    2    1    1    0    1    3    0  0.534146      10/10      Serial
   0    2    2    0    3    0    0    1    1    1  0.350485      10/10      LinearComplexity


 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 The minimum pass rate for each statistical test with the exception of the
 random excursion (variant) test is approximately = 8 for a
 sample size = 10 binary sequences.

 The minimum pass rate for the random excursion (variant) test
 is approximately = 4 for a sample size = 5 binary sequences.

 For further guidelines construct a probability table using the MAPLE program
 provided in the addendum section of the documentation.
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Table 5: second time testing /dev/random

```
 ------------------------------------------------------------------------------
 RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
 ------------------------------------------------------------------------------
    generator is <data/devrandom2.bin>
 ------------------------------------------------------------------------------
  C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
 ------------------------------------------------------------------------------
   8  13  12   9   8  15   4  11  11   9  0.474986    100/100     Frequency
   8  15  10  15  12   8   9   9   7   7  0.514124    100/100     BlockFrequency
   7  10  10  16   7  13   8   9   8  12  0.574903    100/100     CumulativeSums
   9   8  14   7   9  13  12  12   5  11  0.595549    100/100     CumulativeSums
  10  14   7  10   8   8   9   6  12  16  0.437274     99/100     Runs
  10   8   9  12  11  12   7  10  12   9  0.971699     99/100     LongestRun
   8  10  12  10  14  11   9   3  12  11  0.534146     98/100     Rank
  12  11   7  10   6  11   5   9  13  16  0.334538     99/100     FFT
  13  10  14   7  12   8  12  11   9   4  0.494392     99/100     NonOverlappingTemplate
  13  12  11   7  10   6  11  10  12   8  0.851383    100/100     NonOverlappingTemplate
   9  10  10  11   9   9  11  13   9   9  0.996335     99/100     NonOverlappingTemplate
  10   8   7   9  12  14   9  14   7  10  0.739918     97/100     NonOverlappingTemplate
   9  13   9   7  11  11  11  13  14   2  0.262249    100/100     NonOverlappingTemplate
   8   9   5   7  15  11  11  13  10  11  0.574903    100/100     NonOverlappingTemplate
  10  13  11   7   8   7   8   7  17  12  0.366918     99/100     NonOverlappingTemplate
   8   8   8  10   9   7  12   8  14  16  0.514124     99/100     NonOverlappingTemplate
   6  10   8   8  15  13   9   9  12  10  0.699313     99/100     NonOverlappingTemplate
  13  10  11  10  10  11   9   8   5  13  0.834308     96/100     NonOverlappingTemplate
   6   6  11   9  16   8  11   6  15  12  0.213309     99/100     NonOverlappingTemplate
  7   6   7  14  14  10  10  11  10  11  0.657933     99/100      NonOverlappingTemplate
 12   6  11  14  10  11  11  10   6   9  0.779188     99/100      NonOverlappingTemplate
 10  11   5   6   9   9  12  13  11  14  0.595549     97/100      NonOverlappingTemplate
 12  13  17   7  11  10   7   6  10   7  0.304126     99/100      NonOverlappingTemplate
  7   9  13  12   4  15   8  14   5  13  0.129620     99/100      NonOverlappingTemplate
 14   8   7   6  15   8  11   7  10  14  0.350485     98/100      NonOverlappingTemplate
 19   7   8   3  15  10   9   9  12   8  0.037566     97/100      NonOverlappingTemplate
  9  12   9   7   7  13  14  11   6  12  0.637119     99/100      NonOverlappingTemplate
 16   4   9   8  12  17   5   9  11   9  0.071177     99/100      NonOverlappingTemplate
  9   8   9   8  10   9  17   8  13   9  0.595549    100/100      NonOverlappingTemplate
  9  13  10   9  12   9   9  10  12   7  0.964295    100/100      NonOverlappingTemplate
 13  14  12  13   7  11   6  10   6   8  0.494392    100/100      NonOverlappingTemplate
 11   7   7  12  12   8   9  13  11  10  0.897763    100/100      NonOverlappingTemplate
  9  13   7  10  13   8  12  12   8   8  0.851383    100/100      NonOverlappingTemplate
  7   6  11   7  13   9   9  12   9  17  0.350485     97/100      NonOverlappingTemplate
 16  15  10  10   7  13   8  10   6   5  0.191687    100/100      NonOverlappingTemplate
 14  12   9  14  15   7  11   6   5   7  0.202268    100/100      NonOverlappingTemplate
 13   6  12  15   7  12   9  13   5   8  0.304126     99/100      NonOverlappingTemplate
  8   9  14  11   7  10  19   7   5  10  0.102526     99/100      NonOverlappingTemplate
  8  11   7  11   9  15  11   5  15   8  0.383827    100/100      NonOverlappingTemplate
  8   8  16  11   7  10   9  11   9  11  0.759756     99/100      NonOverlappingTemplate
 16  11  13  10   9  10   9   8   6   8  0.616305     98/100      NonOverlappingTemplate
 14   7  10  10  16  11   7   9   8   8  0.534146     98/100      NonOverlappingTemplate
  9  11   7  12   9   9  10   6  14  13  0.759756    100/100      NonOverlappingTemplate
 13  14   7  11  11   8  15   4   6  11  0.224821     98/100      NonOverlappingTemplate
  6   7   9  13   9  15  16  14   6   5  0.080519    100/100      NonOverlappingTemplate
  9  16   3  11   8   6  14  14   6  13  0.058984     99/100      NonOverlappingTemplate
 10  11   9  10  10  14  10  11   5  10  0.883171    100/100      NonOverlappingTemplate
  9  14  11  12  10  11   7  11   6   9  0.834308     99/100      NonOverlappingTemplate
 15   6  10   7  11   7   9   9  13  13  0.534146     97/100      NonOverlappingTemplate
 10  12  10  11  11   7  13   8   9   9  0.964295     98/100      NonOverlappingTemplate
 10   5  19   9   8  13   9   7  11   9  0.153763     98/100      NonOverlappingTemplate
  6   6  15   5   7  14  13   6  15  13  0.055361    100/100      NonOverlappingTemplate
  9  12  14   7   7   7  10  13  10  11  0.759756     99/100      NonOverlappingTemplate
 12   7  12  14  11  10  13   7   6   8  0.616305     99/100      NonOverlappingTemplate
 16   6  12  10  12   8   9  10   6  11  0.514124     98/100      NonOverlappingTemplate
  8  10  13   9   9   8  13   8   7  15  0.678686    100/100      NonOverlappingTemplate
 17  11   8  16   7   7  11  11   6   6  0.115387     99/100      NonOverlappingTemplate
 11  16  10   8  11  10  11   8   7   8  0.739918    100/100      NonOverlappingTemplate
  8  14  11   8  13   8   6  10  10  12  0.759756    100/100      NonOverlappingTemplate
  8  12   8  16   6  12  11   6  10  11  0.474986     98/100      NonOverlappingTemplate
 15   9  10   8   9   7   9  13   9  11  0.816537     98/100      NonOverlappingTemplate
 11  12  11   7  14  12   8   8   8   9  0.851383     99/100      NonOverlappingTemplate
 13  12  11  12   1  16  11   7   7  10  0.080519    100/100      NonOverlappingTemplate
 10  10  10  12  13   4  11  14   6  10  0.514124     98/100      NonOverlappingTemplate
  4   7  13   9  14  11  12  10  10  10  0.574903     99/100      NonOverlappingTemplate
 11   8  11   5  11  13  10   7  14  10  0.678686     99/100      NonOverlappingTemplate
 11  10  12   6   7  11  15  10  11   7  0.678686     99/100      NonOverlappingTemplate
 10   6  14  12  11  14   8   9   4  12  0.366918     98/100      NonOverlappingTemplate
  8   8  10  12  14  10   7   8   7  16  0.474986     99/100      NonOverlappingTemplate
 14  12  12  14  12   8   8   7   6   7  0.474986     98/100      NonOverlappingTemplate
  5   8  14   6   8   7  16  13  13  10  0.171867    100/100      NonOverlappingTemplate
  5  11  14   9  14  13   7   8   8  11  0.474986    100/100      NonOverlappingTemplate
 13  13  13  14   4   5   5  11  11  11  0.153763     97/100      NonOverlappingTemplate
  8  12   7  13   8   9   7  13  14   9  0.678686    100/100      NonOverlappingTemplate
  4  10  13  11  12   8  11  15   9   7  0.437274    100/100      NonOverlappingTemplate
 19  17  11   6   7   7   9   8   6  10  0.028817    100/100      NonOverlappingTemplate
 11   7   8   9  15  10   8  15   9   8  0.595549    100/100      NonOverlappingTemplate
  8  10  15   7  12   8  10  11   6  13  0.616305     99/100      NonOverlappingTemplate
  9  10  18   7   9  15   8   8   6  10  0.191687     99/100      NonOverlappingTemplate
 12  10  11  11  11   9   7  12   9   8  0.978072    100/100      NonOverlappingTemplate
 12  10   6  12  15  11   9   8   9   8  0.739918     99/100      NonOverlappingTemplate
  6  11   5  13   9  13  11  12   7  13  0.494392    100/100      NonOverlappingTemplate
 13  10  14   7  12   9  11  11   9   4  0.554420     99/100      NonOverlappingTemplate
 14  10  10  12   9   6   8  11  10  10  0.897763     99/100      NonOverlappingTemplate
 17  11   8  12  11   4   9   7  10  11  0.304126     99/100      NonOverlappingTemplate
  9  16  10  11   8  10   8  14   6   8  0.514124     98/100      NonOverlappingTemplate
  9   5  14  13   8  14  10  12   8   7  0.455937     99/100      NonOverlappingTemplate
  7   7  10  10  11   7  12  13  13  10  0.834308     99/100      NonOverlappingTemplate
 10   8   2  15   9   6  10  16  13  11  0.075719    100/100      NonOverlappingTemplate
 10  20   7  14  10   6   9  12   7   5  0.035174     99/100      NonOverlappingTemplate
 11  16   9  11   5  11  10  12  10   5  0.401199     98/100      NonOverlappingTemplate
  7  10  10   4   8  17  18   8   8  10  0.048716    100/100      NonOverlappingTemplate
  7  10  17   8  13   9   7   8   9  12  0.437274     98/100      NonOverlappingTemplate
  7   8  17  12  14   5   9   7   7  14  0.115387     98/100      NonOverlappingTemplate
  9  14   6  14   6  10  11  10   8  12  0.595549    100/100      NonOverlappingTemplate
 10   5  14  11  10   9   6  12  15   8  0.419021     98/100      NonOverlappingTemplate
 18   9   7   6   9   9  11  10  11  10  0.401199     98/100      NonOverlappingTemplate
 11   8  13  13  10   7   8  12  11   7  0.834308     97/100      NonOverlappingTemplate
  8   9   8   7   9  12  10  12  12  13  0.911413    100/100      NonOverlappingTemplate
 16  12   5  11   9  12   9  10   5  11  0.366918     98/100      NonOverlappingTemplate
  7  11   5  11  12   9  11  11  14   9  0.739918    100/100      NonOverlappingTemplate
  6  11   7  10  10  10  15   8   8  15  0.494392     97/100      NonOverlappingTemplate
 15  16   8   7   9   6  12   8   9  10  0.350485    100/100      NonOverlappingTemplate
  9  12  11  12   7  10  11   6  14   8  0.779188     99/100      NonOverlappingTemplate
 12  10  10   8   9   9  11  11  10  10  0.998821     99/100      NonOverlappingTemplate
  8  12   6  12  16  10  11  10   6   9  0.514124     98/100      NonOverlappingTemplate
  8   9   6   9  17   9  12  10   8  12  0.494392     98/100      NonOverlappingTemplate
  8  15   9  11  14  11  10   7   7   8  0.637119     99/100      NonOverlappingTemplate
 11  11  11   7  13   8  12   8   9  10  0.946308    100/100      NonOverlappingTemplate
 13   9  11   9  10   9   9  10  13   7  0.955835     99/100      NonOverlappingTemplate
  6  14  10   8   7   9  11  11  16   8  0.455937    100/100      NonOverlappingTemplate
 12  15   9   8   8  10   7   9   7  15  0.514124     99/100      NonOverlappingTemplate
 13   7   6  12  10   8  11  11   5  17  0.224821     99/100      NonOverlappingTemplate
 15  16   7   6   9  14   5  11  14   3  0.021999    100/100      NonOverlappingTemplate
  9  11   8   8  12   9  12   6  10  15  0.739918    100/100      NonOverlappingTemplate
 15  19  13  13   5   9   3   7   7   9  0.009535    100/100      NonOverlappingTemplate
```

```
  12  10  10  11   7   9  10  12   7  12  0.955835     97/100     NonOverlappingTemplate
  10   8  13   8  10  12   9   6  12  12  0.867692     98/100     NonOverlappingTemplate
   7  11   6  10  11   9  10  16   9  11  0.678686     99/100     NonOverlappingTemplate
  15   7   6   6  14  13   9  11  12   7  0.304126    100/100     NonOverlappingTemplate
  10   5  16  11  12   6  10  12   9   9  0.455937     98/100     NonOverlappingTemplate
  10   7  11   6  12  11  11   8  14  10  0.816537     99/100     NonOverlappingTemplate
   8  15  10   8  10   5   9   7  10  18  0.153763     96/100     NonOverlappingTemplate
  11  15  14  10   6   8   8   7   6  15  0.236810     98/100     NonOverlappingTemplate
  10  12   6  14  12   9   7  12   6  12  0.595549     98/100     NonOverlappingTemplate
   6  10   9  10  16  14  10   8  11   6  0.437274    100/100     NonOverlappingTemplate
   7   5  11  16  12  12   9  11   7  10  0.437274    100/100     NonOverlappingTemplate
  11   4  16  11   9   8  12  10   9  10  0.494392     98/100     NonOverlappingTemplate
   6  10  14  11   8   8  11   5  11  16  0.319084    100/100     NonOverlappingTemplate
  14   9   9   9  10  14   6  10  10   9  0.816537     99/100     NonOverlappingTemplate
  10   9  12   5  13  12   4  14  11  10  0.383827     99/100     NonOverlappingTemplate
  14   8  10  11   9   8  10   8  13   9  0.911413     99/100     NonOverlappingTemplate
  10  12   7   6  13  10  14  10   6  12  0.595549    100/100     NonOverlappingTemplate
   7  10   7   9   9  14  13   7   8  16  0.401199    100/100     NonOverlappingTemplate
  10   7  10  14   8  11  12   9  13   6  0.739918    100/100     NonOverlappingTemplate
  10  10  10  10   6  16  12   7   8  11  0.637119     99/100     NonOverlappingTemplate
   8  17   7  10  10  14   5  13   8   8  0.213309     99/100     NonOverlappingTemplate
   6  11  13   3  15  13  10   5  14  10  0.090936     97/100     NonOverlappingTemplate
  18   9   7  11   4  14   8   4  12  13  0.035174     99/100     NonOverlappingTemplate
  10  11   6  13  11   8  11   9  12   9  0.924076     99/100     NonOverlappingTemplate
   8   7   8  12  16  15   6   7  10  11  0.289667     99/100     NonOverlappingTemplate
  10   8  12   6   8   5  12  12  11  16  0.366918    100/100     NonOverlappingTemplate
   9  11  15  15   9   3  14   6   9   9  0.137282     99/100     NonOverlappingTemplate
   4  11  12  13   8  11   6  12  10  13  0.494392     99/100     NonOverlappingTemplate
   9  11  13   5  12  12   7   5  17   9  0.171867    100/100     NonOverlappingTemplate
  17   8  10  16   5   3  15  12   7   7  0.012650     96/100     NonOverlappingTemplate
   3  10  10   8   8   9   8  16  16  12  0.129620    100/100     NonOverlappingTemplate
  16   7  14  10   8   8  11   9   7  10  0.534146     97/100     NonOverlappingTemplate
   8  12  12  10   5  12  11  19   5   6  0.058984     99/100     NonOverlappingTemplate
  12  11   6   9   9  13  12   8   7  13  0.759756    100/100     NonOverlappingTemplate
  10  11  16  12   4  11  11   9   8   8  0.455937     99/100     NonOverlappingTemplate
   5  11   6  16   9   8  13  16   7   9  0.129620    100/100     NonOverlappingTemplate
   8   7  13  15  12  11   7   6  12   9  0.514124    100/100     NonOverlappingTemplate
  11   9  12  19  13  13   6   7   2   8  0.019188     99/100     NonOverlappingTemplate
  12  15  11  13  10  14   7   2   9   7  0.129620     97/100     NonOverlappingTemplate
   6  11   5  13   9  12  12  12   7  13  0.514124    100/100     NonOverlappingTemplate
  12   5  11  13   8   9  12   6   9  15  0.437274     99/100     OverlappingTemplate
  18   9   6   9   9  10  10  10   9  10  0.494392     99/100     Universal
  14  10  16  10   4   6  10   8  12  10  0.262249     98/100     ApproximateEntropy
   4   7   6   6   1   6   4   5   4   3  0.534146     46/46      RandomExcursions
   7   6   6   3   6   3   2   6   3   4  0.534146     45/46      RandomExcursions
   2   5   6   6   6   8   4   3   4   2  0.392456     46/46      RandomExcursions
   7   2   4   3   4   5   4   7   7   3  0.484646     46/46      RandomExcursions
   5   1   1   7   5   6   6   9   4   2  0.057146     46/46      RandomExcursions
   7   7   4   4   3   5   6   4   2   4  0.637119     46/46      RandomExcursions
   3   5   1   4   4   6   5   4   5   9  0.311542     45/46      RandomExcursions
   3   3   2   4   7   7   3   5   8   4  0.311542     45/46      RandomExcursions
   8   3   3   6   5   7   3   6   4   1  0.242986     45/46      RandomExcursionsVariant
   7   3   5   5   5   4   3   5   6   3  0.834308     45/46      RandomExcursionsVariant
   7   3   2   6   3   5   5   4   8   3  0.392456     45/46      RandomExcursionsVariant
   6   3   6   4   1   2   5   9   5   5  0.186566     46/46      RandomExcursionsVariant
   7   5   2   6   1   3   3   7   8   4  0.141256     46/46      RandomExcursionsVariant
   7   6   3   3   4   4   5   3   6   5  0.788728     46/46      RandomExcursionsVariant
   6   5   7   3   6   5   1   3   6   4  0.484646     46/46      RandomExcursionsVariant
   3   4   3   5  10   6   1   4   5   5  0.141256     45/46      RandomExcursionsVariant
   3   2   5   6   3   5   5   4   5   8  0.585209     45/46      RandomExcursionsVariant
   2   6   9   3   5   2   2   5   7   5  0.141256     46/46      RandomExcursionsVariant
   3   5   9   2   8   2   5   4   4   4  0.162606     46/46      RandomExcursionsVariant
   3   7   6   6   3   6   4   6   1   4  0.437274     46/46      RandomExcursionsVariant
   3   8   4   6   2   5   5   4   5   4  0.637119     45/46      RandomExcursionsVariant
   4   5   7   4   8   3   2   6   6   1  0.213309     46/46      RandomExcursionsVariant
   4   5   7   5   5   3   4   4   2   7  0.689019     45/46      RandomExcursionsVariant
   5   5   3   4   4  10   3   4   4   4  0.350485     45/46      RandomExcursionsVariant
   5   3   6   4   6   6   1   4   7   4  0.534146     46/46      RandomExcursionsVariant
   5   5   6   6   1   3   6   0   7   7  0.105618     45/46      RandomExcursionsVariant
   6   6  14  17   7  14  10   8   7  11  0.137282     99/100     Serial
   9  11  13   8   9  11   9   7  14   9  0.883171     99/100     Serial
   7  15   8   9   7   8   7  15  12  12  0.401199     99/100     LinearComplexity


 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 The minimum pass rate for each statistical test with the exception of the
 random excursion (variant) test is approximately = 96 for a
 sample size = 100 binary sequences.

 The minimum pass rate for the random excursion (variant) test
 is approximately = 43 for a sample size = 46 binary sequences.

 For further guidelines construct a probability table using the MAPLE program
 provided in the addendum section of the documentation.
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Table 6: the results of testing the output of the ANU Quantum Random Numbers Server

```
 ------------------------------------------------------------------------------
 RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
 ------------------------------------------------------------------------------
    generator is <data/quantumrandom.bin>
 ------------------------------------------------------------------------------
  C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
 ------------------------------------------------------------------------------
```

```
   9   4   8  14  15  11  11   9   6  13  0.275709     99/100      Frequency
   5   7  12  13  14  10  10   9   8  12  0.616305     98/100      BlockFrequency
   8   8  10  11  10  12  12  12   9   8  0.978072     99/100      CumulativeSums
   7   9   6   9  14  13  15  10   8   9  0.514124    100/100      CumulativeSums
  10   6   8  11  14   5  14   9  13  10  0.455937     99/100      Runs
  14  12   6   7   7  11   9  12  11  11  0.719747     99/100      LongestRun
  14  11   6   8   9   8  10  12  13   9  0.779188     99/100      Rank
  12  15   8   8   8   7   8   9  11  14  0.616305     99/100      FFT
   7  10  16  13   9  13   6   7  13   6  0.249284     98/100      NonOverlappingTemplate
   7  11  11  11  15  14   8   5   7  11  0.419021     98/100      NonOverlappingTemplate
   4  13   8  14  10   5   7  10  15  14  0.122325    100/100      NonOverlappingTemplate
   4  10  11   9   4   9  12  12  12  17  0.137282    100/100      NonOverlappingTemplate
  14   7  10   8   7   5  12  10  10  17  0.236810     99/100      NonOverlappingTemplate
   8   6  12  20   7   8  13   9  10   7  0.075719     99/100      NonOverlappingTemplate
   9  16   8   9   7   9  16  11   7   8  0.334538    100/100      NonOverlappingTemplate
   7   8   9   7  14  12  14  16   5   8  0.191687    100/100      NonOverlappingTemplate
   9   8  11   7  14   9  17   5   9  11  0.289667    100/100      NonOverlappingTemplate
   7  12  13   6  11  12  10   8   9  12  0.816537    100/100      NonOverlappingTemplate
  14  11   6  11   9   7   6  14  12  10  0.534146     96/100      NonOverlappingTemplate
  10   7  17   3  11   9   8  11  10  14  0.162606     99/100      NonOverlappingTemplate
   7  11   7  10  12   9  13  11  10  10  0.946308     98/100      NonOverlappingTemplate
  11  13   9  14  13  13  11   8   3   5  0.191687    100/100      NonOverlappingTemplate
   7  13   6  12  12  16   9   5  10  10  0.319084    100/100      NonOverlappingTemplate
  19   6   6   8   7  11   9  12  13   9  0.115387     98/100      NonOverlappingTemplate
  11  10  10  10  10   9  15   9   4  12  0.657933     99/100      NonOverlappingTemplate
   5   8  11  12  16   9   9  13   4  13  0.181557    100/100      NonOverlappingTemplate
   8  16   9   9  13  12   7   9   8   9  0.637119     97/100      NonOverlappingTemplate
   4  18  10  10   7   6  15   6  10  14  0.032923    100/100      NonOverlappingTemplate
  10   8  14   6  11   7  10   5  22   7  0.007694    100/100      NonOverlappingTemplate
  13   8  11  11  11   9   9  11   9   8  0.983453     98/100      NonOverlappingTemplate
  12  13   9   8   7   7   9  12  15   8  0.637119     98/100      NonOverlappingTemplate
   7   6  11   9  13  14  11  12   6  11  0.595549     99/100      NonOverlappingTemplate
   9  12  13   9  10   7  13   8   8  11  0.897763     98/100      NonOverlappingTemplate
  12  11  10  18   8   8   6   6   8  13  0.202268    100/100      NonOverlappingTemplate
  12  13   9   6  10  10   6  15   9  10  0.616305     97/100      NonOverlappingTemplate
  14   8  10  10   6  11   8  11   7  15  0.574903     97/100      NonOverlappingTemplate
   7   8   8  16   6  10  13  11  10  11  0.534146    100/100      NonOverlappingTemplate
  12  13   5  16   8  13   9   9  10   5  0.249284    100/100      NonOverlappingTemplate
  12  11   8  12  10  12  11   9   7   8  0.955835     99/100      NonOverlappingTemplate
   9   8   7   9  14   9  11  13   8  12  0.834308     99/100      NonOverlappingTemplate
   7   7  10   5  12  13  13   9  14  10  0.514124     98/100      NonOverlappingTemplate
  10  13   6   9  13  16   5  12  10   6  0.236810    100/100      NonOverlappingTemplate
  11   9   9  11   9  10   9  13   7  12  0.971699     98/100      NonOverlappingTemplate
  10  15  12  13   8   8  12   8   6   8  0.595549     95/100    * NonOverlappingTemplate
  12  13   8   9   9  11   7   7  12  12  0.867692    100/100      NonOverlappingTemplate
  11  14   8   8   7   5  14  15   9   9  0.334538    100/100      NonOverlappingTemplate
  10   6   8  10   8   5  12  14  14  13  0.401199     99/100      NonOverlappingTemplate
   9   8   7   8  12  11   9  14  12  10  0.883171     99/100      NonOverlappingTemplate
  12   9  11   9   9   6  12   5  12  15  0.514124    100/100      NonOverlappingTemplate
   8   9  12  11  13  13  10   8   6  10  0.851383     99/100      NonOverlappingTemplate
  11   9  13  13  14   5   7  10   8  10  0.595549     99/100      NonOverlappingTemplate
   9  10  10   9   9   6  13  13  13   8  0.834308     99/100      NonOverlappingTemplate
  11  12  13   5  13   9  11   8   6  12  0.595549    100/100      NonOverlappingTemplate
  10   9  11   6   9  11   6  13  11  14  0.719747    100/100      NonOverlappingTemplate
   7  13   7   6  10  12   9  10  13  13  0.678686     98/100      NonOverlappingTemplate
   7  15  13   6  13   8   7   9   8  14  0.334538    100/100      NonOverlappingTemplate
  11   8   7   9  12   8  12   9  13  11  0.924076     99/100      NonOverlappingTemplate
  12   9   9  15   9  10   7   7  11  11  0.816537    100/100      NonOverlappingTemplate
   9   8   8  14   8  11   5   4  16  17  0.040108     99/100      NonOverlappingTemplate
   7   8  12   7  18   6  13   9  10  10  0.236810     98/100      NonOverlappingTemplate
   9  14   8   9  16   7   8   8   8  13  0.455937     97/100      NonOverlappingTemplate
  11  13  11  11  12   7   8  10   6  11  0.867692     99/100      NonOverlappingTemplate
   6   7  13  10  13   5  12  11  12  11  0.554420    100/100      NonOverlappingTemplate
  12  12  10   9   9   9   9  12   9   9  0.994250     99/100      NonOverlappingTemplate
   6   9  13  11  13  12  10   7   9  10  0.834308    100/100      NonOverlappingTemplate
  17  10  12   7   7  14   9   6  10   8  0.289667     98/100      NonOverlappingTemplate
   9  17   6   8  10   9  10   6  14  11  0.319084     98/100      NonOverlappingTemplate
  10   7   7   9  14   9  13  10  10  11  0.867692     98/100      NonOverlappingTemplate
   9  14   9  14   8   9   9   7   5  16  0.275709    100/100      NonOverlappingTemplate
  13   8   9  11  11   8   8   8  15   9  0.798139    100/100      NonOverlappingTemplate
  10  19  10  10   9   8   2   8  11  13  0.058984    100/100      NonOverlappingTemplate
   8  10  12  12  12  11   7   7  11  10  0.935716    100/100      NonOverlappingTemplate
   7  10  16  12  10  12   8  13   4   8  0.304126     99/100      NonOverlappingTemplate
  14   6  10   6   9  14   9  11  12   9  0.616305    100/100      NonOverlappingTemplate
   5  10  14   7  13   4  10  14  13  10  0.213309     99/100      NonOverlappingTemplate
  10   8  12   7   7  12   9   9  12  14  0.816537     99/100      NonOverlappingTemplate
  17   7   8   9  15   7   5  11  10  11  0.191687     98/100      NonOverlappingTemplate
  12   6  11  14  12  10  10   7  12   6  0.637119     99/100      NonOverlappingTemplate
  10   8   6  14   9   9  13   6  12  13  0.574903    100/100      NonOverlappingTemplate
  10   8  13   8  11  13  14   8   5  10  0.616305    100/100      NonOverlappingTemplate
   7   8  15   8  12  11  11  11   8   9  0.798139     99/100      NonOverlappingTemplate
  10   4  10  10  12  16  11   9   8  10  0.514124     98/100      NonOverlappingTemplate
   7  10  16  13   9  13   6   8  12   6  0.319084     98/100      NonOverlappingTemplate
   6   9  12  11  10   9  12   9  15   7  0.719747     99/100      NonOverlappingTemplate
   8  12   7  12  11  10  15   8  10   7  0.739918     99/100      NonOverlappingTemplate
  10  10   6  13   9  11   9  16   8   8  0.616305    100/100      NonOverlappingTemplate
  13   6  12  13   9   8  11   8  11   9  0.834308     99/100      NonOverlappingTemplate
  13   8   8  12   5  10  11  13   5  15  0.304126     98/100      NonOverlappingTemplate
  11   5   8  16  12   7  17  11   8   5  0.071177     99/100      NonOverlappingTemplate
  13  10   9  15  12   8  11   6   5  11  0.474986     99/100      NonOverlappingTemplate
  10  13  12   4  17   8  10   9   6  11  0.213309     99/100      NonOverlappingTemplate
   7  12  11  13   9   8  14   6   8  12  0.657933    100/100      NonOverlappingTemplate
   6  13  10  12   5  16   3  16   9  10  0.040108    100/100      NonOverlappingTemplate
   5  13   9  15  12   5  12   7  12  10  0.304126    100/100      NonOverlappingTemplate
  11  11   9  12   5  15   8   9   8  12  0.637119     99/100      NonOverlappingTemplate
  11  17   9   5   7   9  15   8  10   9  0.236810     99/100      NonOverlappingTemplate
   7  13   8  11  10  12   7   8  12  12  0.851383    100/100      NonOverlappingTemplate
  8   8   9  10  14   7  12  11  11  10  0.911413    100/100     NonOverlappingTemplate
  8   7  13  14   7  10   9   9  10  13  0.759756     99/100     NonOverlappingTemplate
  9  12   6  13  13   7   8   8  11  13  0.678686     98/100     NonOverlappingTemplate
 10   7   8  17   5   8   7  13  11  14  0.181557     98/100     NonOverlappingTemplate
  9  18  14  13   6   8   8   6   8  10  0.145326     99/100     NonOverlappingTemplate
  8   9   9  13   4  12  15   6  10  14  0.262249     98/100     NonOverlappingTemplate
  7   8  14   5  11   7   7  15  12  14  0.224821    100/100     NonOverlappingTemplate
 14   3   9  13  10  15  12   6   7  11  0.162606    100/100     NonOverlappingTemplate
  7   3  13  14  10  11   9   9  11  13  0.383827    100/100     NonOverlappingTemplate
 10   9  10  17   6   9  15   5   7  12  0.162606    100/100     NonOverlappingTemplate
  8  11   7  11  11   6  14  15  10   7  0.514124    100/100     NonOverlappingTemplate
  8  15  11  10  10  10   9   7  12   8  0.851383    100/100     NonOverlappingTemplate
  9  12   8  10  12  15  11   6   8   9  0.739918     99/100     NonOverlappingTemplate
  9  11   7  10   7  12  13  12   7  12  0.834308     98/100     NonOverlappingTemplate
  9   7  18  12   9  10   8   7   7  13  0.275709     99/100     NonOverlappingTemplate
  5   9  10  14   7  13  13   5  15   9  0.213309    100/100     NonOverlappingTemplate
 10  12  12   6   9  11   4  13  13  10  0.534146     99/100     NonOverlappingTemplate
  8  19  14   0   8   7   9  16  12   7  0.001757     99/100     NonOverlappingTemplate
 12   9   9  11   3   8  10  12  15  11  0.437274    100/100     NonOverlappingTemplate
 12  10  11  10  10   9   7  12   9  10  0.991468     99/100     NonOverlappingTemplate
  7   5  12   9  15  14   7  12   8  11  0.366918     98/100     NonOverlappingTemplate
  9  11  10  11   8  10   9  12  11   9  0.997823    100/100     NonOverlappingTemplate
 13  12  11   5   9  10   7  10  14   9  0.678686    100/100     NonOverlappingTemplate
  9  11  11  11   9  18   8   4  13   6  0.145326     99/100     NonOverlappingTemplate
  8   9   9   8  11  11  14   8  12  10  0.935716    100/100     NonOverlappingTemplate
  8  12  11   8  12   6  13  12  10   8  0.834308    100/100     NonOverlappingTemplate
  9  12   9  12  15   8   8   9  12   6  0.699313     99/100     NonOverlappingTemplate
 13  10   8   9  15  14   4   9   7  11  0.334538     99/100     NonOverlappingTemplate
 10  13   7   9  10  11   8  10  11  11  0.978072     98/100     NonOverlappingTemplate
 12   7   8   6  13  14  12  11   9   8  0.657933     99/100     NonOverlappingTemplate
  7   9   7  12  16   8   9  11   6  15  0.304126    100/100     NonOverlappingTemplate
  8  11   9  11   5  11  14  15   7   9  0.494392    100/100     NonOverlappingTemplate
 11   7  12  17   5  13   5   7  12  11  0.137282    100/100     NonOverlappingTemplate
 11  11  13  13  10   7   9   7   9  10  0.911413    100/100     NonOverlappingTemplate
 11  12   8  11  12  12  13  10   9   2  0.419021     98/100     NonOverlappingTemplate
 17   8  11   3  11  10   5  13  14   8  0.071177     99/100     NonOverlappingTemplate
 15   9  13  11  12   5  12   9   8   6  0.437274     99/100     NonOverlappingTemplate
 11  11  13  14  10   8  10   6   8   9  0.816537     98/100     NonOverlappingTemplate
 10   4   7  19  10   8   8  12  11  11  0.122325     99/100     NonOverlappingTemplate
 13   9  10  13   5  14   9   8   9  10  0.678686    100/100     NonOverlappingTemplate
 12  13  10  14   4   6  15  10   9   7  0.236810     96/100     NonOverlappingTemplate
 12   7   9   6  16  10   8   4   9  19  0.026948     99/100     NonOverlappingTemplate
  8  12   6   8  14   9   9   6  15  13  0.383827    100/100     NonOverlappingTemplate
 12  10  10   7   9   7   8  12  16   9  0.657933    100/100     NonOverlappingTemplate
  6  14  14   7   8   9   9   9   7  17  0.202268     99/100     NonOverlappingTemplate
  9   9  10  14  11   8   7   9   6  17  0.366918     97/100     NonOverlappingTemplate
 13   7  14   6  12   9   9  11   9  10  0.759756    100/100     NonOverlappingTemplate
  8  13  18  11  10   5   3  11   9  12  0.071177     99/100     NonOverlappingTemplate
  9  12  14  12   8   7   6  12   8  12  0.678686     98/100     NonOverlappingTemplate
  5   9   5  11  19  12   8  13   7  11  0.066882    100/100     NonOverlappingTemplate
  6  13   8  11  10   9  12  11   7  13  0.798139    100/100     NonOverlappingTemplate
  9   9   7  12  11  12  12   9   8  11  0.964295     99/100     NonOverlappingTemplate
  6  11   9   8  11  10  15  15   7   8  0.474986    100/100     NonOverlappingTemplate
 11  10   8   7   9  11  14  13   5  12  0.637119    100/100     NonOverlappingTemplate
  5  13  13   7   9   8  12   7   7  19  0.066882     99/100     NonOverlappingTemplate
  3  13  13   7  12   8  14  10  11   9  0.334538    100/100     NonOverlappingTemplate
 14  11  10   6  15   9   4  12  11   8  0.319084     99/100     NonOverlappingTemplate
 13   5   9  15  10   9   5  12  10  12  0.401199     98/100     NonOverlappingTemplate
 10   4  10  10  12  16  11   9   8  10  0.514124     98/100     NonOverlappingTemplate
  6  12   9  10  12   7  10  10   9  15  0.739918    100/100     OverlappingTemplate
 11   8  12  19   8   9  12   7   5   9  0.145326     99/100     Universal
  9   8   4  10  13  12   9  15   9  11  0.514124    100/100     ApproximateEntropy
 10   5   6   7   3   3   5   3  11   5  0.075719     57/58      RandomExcursions
  9   5   7   6   7   7   3   3   3   8  0.350485     58/58      RandomExcursions
  7   6   4   8   4   5   4   4   9   7  0.574903     57/58      RandomExcursions
  4   6   6   9   6  10   3   3   7   4  0.236810     58/58      RandomExcursions
  5   4  10   8   7   2   2   5  10   5  0.058984     58/58      RandomExcursions
  3   7   2   6   6   9   7   6   8   4  0.350485     58/58      RandomExcursions
  5   5   4   7   3   7   4   4   4  15  0.005762     57/58      RandomExcursions
  6   1  11   4   5   7   5   9   5   5  0.096578     57/58      RandomExcursions
  9   5   2   8   7   7   4   7   5   4  0.383827     58/58      RandomExcursionsVariant
  4   5  13   5   6   4  10   2   7   2  0.006661     58/58      RandomExcursionsVariant
  3   8  10   7   5   5   3   7   2   8  0.137282     58/58      RandomExcursionsVariant
  5   8   6   6   6   4   6   6   4   7  0.911413     58/58      RandomExcursionsVariant
  5   9   5   8   5   3   4   5   5   9  0.419021     57/58      RandomExcursionsVariant
  5   6   7   7   7   8   5   4   6   3  0.779188     58/58      RandomExcursionsVariant
  3   6  10   6   6   4   8   6   3   6  0.383827     58/58      RandomExcursionsVariant
  4   6  12   5   7   5   4   4   5   6  0.236810     58/58      RandomExcursionsVariant
 11   3   5   6   5   3   5   8   4   8  0.171867     58/58      RandomExcursionsVariant
  4   7   8   3  12   4   4   7   6   3  0.075719     58/58      RandomExcursionsVariant
  5   6  11   1   3   9  10   3   5   5  0.015598     58/58      RandomExcursionsVariant
  7   6   6   6   3   7   6   5   5   7  0.911413     58/58      RandomExcursionsVariant
  8   5   6   1   8   6   6   5   7   6  0.494392     57/58      RandomExcursionsVariant
  8   5   3   3   5   6  10   6   7   5  0.383827     56/58      RandomExcursionsVariant
  6   5   9   4   7   3   5   8   7   4  0.534146     56/58      RandomExcursionsVariant
  6   5  11   4   7   5   6   4   4   6  0.419021     57/58      RandomExcursionsVariant
  4   5   9  10   4   5   7   5   5   4  0.383827     57/58      RandomExcursionsVariant
  5   7   3  10   9   7   4   4   6   3  0.213309     56/58      RandomExcursionsVariant
  8   8   9  10  15   6   5  19   8  12  0.058984     99/100     Serial
  7  14   6  12   9  10   8  14  10  10  0.678686    100/100     Serial
  9   9  12   7  12  10   9  13  10   9  0.964295    100/100     LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.
```

```

The minimum pass rate for the random excursion (variant) test
is approximately = 55 for a sample size = 58 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

5. Conclusion

Everyone has a vague idea of what a random number is and how it can be achieved. It is a number that cannot be reproduced by an obvious rule, which means that it either results from an unpredictable physical process or from a complex, non-trivial calculation. Unpredictable physical processes are the golden standard of random number generation and either have or do not have quantum properties. Random number generators that are based on them are called "true". Quantum randomness has two important advantages: 1) randomness is in its nature; it does not merely appear to be random because of our lack of knowledge about it, it is intrinsically unpredictable and 2) security is guaranteed – previous output bears no resemblance to the future output and vice versa. The disadvantages of quantum randomness are that it is generated by specialized hardware, which, for the sake of software creation, needs to be avoided. There are physical sources that do not have quantum properties but can be just as good, and most of them are based on noise. There have been attempts to use computer hardware or user input to generate randomness, unfortunately, they end up being either impractical or insecure.

The security of random number generators can be tested through hands-on brute force approaches, which takes a vast amount of resources, including time. It is the only way to assess the generator and not simply its output and is known under a broad name – cryptanalysis. Other, less resource-consuming methods use the probabilistic method and data compression. The probabilistic method involves assessing the probability with which the generated sequences occur naturally. The data compression approach assesses the rate at which the generated sequence can be compressed to a smaller bulk of data.

Apart from security, random number generators must fulfill other requirements mostly defined by the purpose of their utilization, e.g. the speed with which the output is produced, repeatability of sequences and distribution of values.

The decision between the two methods of random number generation (hardware and algorithmic) is not a trivial one. While the hardware random number generators do not possess a source code that can be utilized by an attacker, they are oftentimes slow and sometimes unreliable due to interference, hardware defects and flaws in digitalization and subsequent de-skewing of data. The algorithmic approach, albeit faster, survives under the principle of computational security, i.e. that an attacker could not predict the future or decipher the past output in realistic time.

Thus, it is best to combine both approaches in a mutually beneficial manner to balance out their deficiencies.

The hardware solutions I tested showed promise but were in need of improvement.

The circuit around the Zener diode appeared to have a flaw in registering the voltage flowing through the diode and thereby, skewing the output in favor of zero values. This method also appeared to be slow, but with price-effective components and provided sufficient de-skewing, it could be used to fill an entropy pool for a pseudorandom generator, e.g. the LRNG at /dev/random, which in itself possesses good statistical properties and a lot of security measures, like the mixing of the entropy pool after each generated sequence.

The atmospheric noise generator I tested was significantly faster than the Z-breakdown generator and showed good statistical properties, but I see potential risks for its utilization because of interference. Since there is an on-going trend towards mobile devices, I would be wary of using an atmospheric-noise generator for portable appliances, especially because there was, as of yet, not a lot of research exists on portable generators.

There is a lot of research yet to be done on random number generation and testing with ever-advancing computational power, but for now, cryptographically secure pseudorandom generators coupled with true random entropy pools are a universally friendly source of randomness.

Bibliography

[1] G. J. Chaitin, *Information, Randomness & Incompleteness*. World Scientific, 1990.

[2] J. Schiller and S. Crocker, "Standards Track," D. Eastlake, 3rd, Motorola Laboratories, Jun. 2005.

[3] M. Fischlin and J.-S. Coron, *Advances in Cryptology – EUROCRYPT 2016*, vol. 9665. Berlin, Heidelberg: Springer, 2016.

[4] "Digital Signature Algorithm." Wikipedia.

[5] D. G. Boak, "A History of U.S. Communications Security (Volumes I and II)," *governmentattic.org*, 24-Dec-2008.

[6] J. Moser, "The First Few Milliseconds of an HTTPS Connection," *moserware.com*, 10-Jun-2009. [Online]. Available: http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html. [Accessed: 15-May-2016].

[7] *Chi-squared Test*, vol. 11. YouTube, 2011.

[8] L. E. I. Bassham, "NIST SP 800-22, Revision1a, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," Apr. 2010.

[9] "Diehard tests." [Online]. Available: http://en.wikipedia.org/wiki/Diehard%20tests. [Accessed: 15-May-16AD].

[10] "Random Bitmap Generator," *random.org*. [Online]. Available: https://www.random.org/bitmaps/. [Accessed: 01-Jul-2016].

[11] D. Salomon, *Data Compression*. Springer, 2012.

[12] A. Young and M. Yung, *Malicious Cryptography*. John Wiley & Sons, 2004.

[13] I. Poole, "RF Thermal Noise | Johnson-Nyquist Noise | Tutorial," *radio-electronics.com*. [Online]. Available: http://www.radio-electronics.com/info/rf-technology-design/noise/thermal-johnson-nyquist-basics-tutorial.php. [Accessed: 31-May-2016].

[14] D. G. Marangon, G. Vallone, and P. Villoresi, "Random bits, true and unbiased, from atmospheric turbulence," *Scientific Reports*, vol. 4, pp. 5490 EP –, Jun. 2014.

[15] R. Soorat, M. K, and A. Vudayagiri, "Hardware Random number Generator for cryptography," *arXiv.org*, vol. physics.comp-ph. 05-Oct-2015.

[16] "onerng.info," *onerng.info*. [Online]. Available: http://onerng.info. [Accessed: 02-May-2016].

[17] J. Thomas, "xr232usb," *jtxp.org*, Sep-2011. [Online]. Available: http://www.jtxp.org/tech/xr232usb_en.htm. [Accessed: 28-Jul-2016].

[18] J. Thomas, "***XR232 - Echter Zufall und echtes RS232-Protokoll***," *jtxp.org*. [Online]. Available: http://www.jtxp.org/tech/xr232web.htm. [Accessed: 02-May-2016].

[19] A. Toponce, “Hardware RNG Through an rtl-sdr Dongle,” *pthree.org*, 16-Jun-2015. [Online]. Available: https://pthree.org/2015/06/16/hardware-rng-through-an-rtl-sdr-dongle/. [Accessed: 29-Jul-2016].

[20] “RANDU,” *en.wikipedia.org*. [Online]. Available: http://en.wikipedia.org/wiki/RANDU. [Accessed: 09-Jun-2016].

[21] “Cryptographically secure pseudorandom number generator.” Wikipedia.

[22] T. Hühn, “Myths about /dev/urandom,” *2uo.de*. [Online]. Available: http://www.2uo.de/myths-about-urandom/. [Accessed: 15-May-2016].

[23] R. O. Gilbert, “Evaluation of four pseudo-random number generators,” May 1973.

[24] H. G. Katzgraber, “Random Numbers in Scientific Computing: An Introduction,” *arXiv.org*, vol. physics.comp-ph. 22-May-2010.

[25] R. Oppliger, *Contemporary Cryptography, Second Edition*. Artech House, 2011.

[26] Z. Gutterman, B. Pinkas, and T. Reinman, “Analysis of the Linux random number generator,” presented at the 2006 IEEE Symposium on Security and Privacy (S&P'06, 2006, pp. 15 pp.–385.