Statistical Tests for Random and Quantum Random Number Generators

Gabriela MOGOS*

International College, Hunan University of Arts and Science, 3150 Dongting Avenue, Changde, 415000, Hunan, China

*Corresponding author

Keywords: Cryptography, P-value, Random number generator, Quantum random number generator.

Abstract. This paper presents a few aspects regarding the testing of two random number generators. The importance of the quality of a generator is according to the type of application used. The outputs of these generators may be used in cryptographic applications to generate encryption keys. Random number generators used in cryptographic applications must meet certain requirements, and the most important is that their outputs must be unpredictable in the absence of knowledge of the inputs. This paper presents the analysis of a few statistical tests performed on two random number generators: a classic-software, and a quantum-hardware. These tests may be considered to determine if a generator is adequate to be used in particular cryptographic applications.

Introduction

Almost all encryption systems and protocols used in cryptography have a central idea: the selection of random, previously unknown, unpredictable numbers; the standard name is random numbers, or randomly generated numbers. The use of the computer reduces the term of random numbers to a row of randomly generated bits, grouped according to a certain rule. Mathematically, there is no shorter modality to specify the row except the sequence itself.

The statistics offer rather little information on the randomly generated bits [1]. For example, it is known that 0 must appear as often as 1, that 00 must appear half as often as 0 (or 1), and as often as 11; 10; 01. There are also statistical tests (Kolmogorov) showing how random the numbers in a row are. In cryptography, it is essential that a random number cannot be found. A perfectly random number is the one which cannot be guessed except by using brute-force.

A purely random number generation is performed by collecting and processing data obtained from a source of entropy outside the computer. The source of entropy may be very simple, e.g. the variations of mouse movements, or the time between pressing two keys. Very good sources of entropy may be the radioactive ones, or those using noises from the atmosphere. The property of being random was introduced in computers with the help of pseudo-random number generators.

Random numbers are important in fields like cryptography, Monte Carlo simulations, VLSI chip tests, and in probabilistic calculation methods, like genetic algorithms and neural networks.

The values generated by a generator must be different from one experiment to another. To solve this paradox, most generators need a starting value, called "seed". This value initializes the random number generator to provide a certain row of values.

Randomly generated numbers for cryptographic applications must be unpredictable. A good generator must be rigorously tested, however even in these conditions there may be applications for which it is not usable. There are two methods of random number generation: software solutions, or hardware solutions.

This paper presents the results obtained in a series of statistical tests performed on two types of random number generators: a classic-software generator, and a quantum-hardware generator. The design and the cryptanalysis of these random number generators are not the object of this paper.
Statistical Tests

In general, all the methods of random number generation may be classified in hardware and software. The hardware devices of random number generation are devices digitalizing any parameter of the environment or of a physical process. Regarding the software of random number generation, we may consider different algorithms to generate a sequence of random numbers.

The purpose of these tests is to determine if a generator is suitable for a particular cryptographic application. The two random number generators (classical and quantum) are integrated in software applications simulating asymmetrical cryptographic systems.

The outputs of the generators are used by Diffie–Hellman key distribution protocol, such as the generation of key material. In the case of the quantum random number generator, programming and collection date were done via USB port, and, the classic random number generator being a software one.

![Figure 1. Diffie—Hellman key distribution protocol.](image)

Random Number Generator

A random number generator, contrary to what one might think at first, does not generate random numbers, but "pseudo"- random numbers, which only give the impression that they are random. Number generation means the performance of an algorithm, each new number depending more or less on the last generated one. Consequently, any such algorithm needs an initial number, called a "seed".

An algorithm initialized with the same seed generates the same sequence of numbers. This is an important aspect, because in practice the initialization of the generator is often forgotten. Being a well determined mathematical algorithm, the generated numbers are also determinist.

The generator of random numbers used in this study was implemented in C#.

![Figure 2. Random Number Generator – interface.](image)

In our application has been used an integer number generator, allowing the generation of a number sequence with a user-defined length:

```csharp
var r = new Random();
```

In addition, the application may calculate Shannon entropy value, the time of response, and a statistical representation.
Random Number Generator

The only way to generate true random numbers, which are impossible to predict or determine, is by using a hardware device based on the measurement of different physical phenomena: sound intensity, temperature variations, etc. However, even these apparently complex physical phenomena hide determinist processes.

In modern physics, the concept of “random” is incorporated only in the branch of quantum physics, and this branch is at the basis of Quantis device (Fig. 3).

![Quantis device](image)

Figure 3. Quantum Random Number Generator – Quantis.

From the perspective of quantum physics, light is made of elementary particles called photons. At their passage through a semi-transparent mirror, these particles can continue their passage, or they can be reflected, and this behavior is intrinsically random, and cannot be influenced by any external factor. A detector placed on each of the two possible directions register this behavior, generating random sequences of 1 and 0.

Quantis is a quantum random number generator exploiting an optical quantum process as source of randomness. Quantis produces a high bit rate of 4 to 16 Mbits/sec of truly random bits.

Quantis is a second-generation quantum random number generator exploiting an optical quantum process as source of randomness, produced by ID Quantique [2].

![Quantis interface](image)

Figure 4. Quantum Random Number generator – interface.

This application allows the user to decide on the number of bytes generated, and it calculates the generation time, the metric entropy, and Shannon entropy.

Testing

Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to a truly random sequence. Randomness is a probabilistic property; that is, the properties of a random sequence can be characterized and described in terms of probability. The likely outcome of statistical tests, when applied to a truly random sequence, is known a priori and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of a “pattern” which, if detected, would indicate that the sequence is nonrandom.
The RaBiGeTe_MT 32 bits – v2.0.0 [3] is a statistical package consisting of 20 tests that were developed to test the randomness of sequences produced by either hardware or software random generators. These tests focus on a variety of different types of non-randomness that could exist in a sequence.

RaBiGeTe package includes NIST [4], FIPS-140 [5] and DIEHARD [6] statistical tests and allows saving p-values and the overall results in a file named “RaBiGeTe.txt”.

Many tests are based on testing the hypotheses:
1. The null hypothesis is formulated. Suppose that the binary row generated is random.
2. Obtain a test sequence. The testing is performed at bit level.
3. Calculate p-value from the interval [0, 1].
4. Compare p-value with $\alpha$ threshold, where $\alpha = 0.01$. The test is successfully passed if p-value is higher than $\alpha$ threshold.

In addition, RaBiGeTe allows obtaining a graphical representation of the distribution of the p-values using three methods: Kolmogorov-Smirnov (K-S) test, Anderson-Darling (A-D) test and Straight Line (SL) test. The Straight-Line test is used to see how much the distribution of the p-values differs from the uniform distribution.

**Results**

The testing of the two random number generators was performed on a battery of RaBiGeTe statistical tests, on a Windows 8 Acer Aspire V5 notebook equipped with AMD C-70 APU and 8 GB of memory. The code was compiled using MS Visual Studio 2012. Although the speed measurements were performed on a 64-bit operating system.

For this study, were used sequences of date generated by the classic random number generator RNG, and by the quantum random number generator QRNG.

This paper analyses the case of the sequences of hexadecimals date obtained by each of the two random number generators.

Figure 5a and 5b present the p-values related to the Pearson chi-square test. The test compares the frequencies of a (variable) distribution with the corresponding frequencies of another (variable) distribution, both measured on a categorical scale, with the purpose to see if there is any association between the two variables. The chi-square test analyses frequencies.

The number of distributions is calculated as the square root of the number of testing sequences. The distributions are numbered from 1 to n, and they appear in the columns 1, 2, ..., n. Each distribution contains the p-values in the interval $[(b-1)/n, b/n)$, where b is the distribution number.

The estimated number of the p-value for each distribution is presented in the column "Exp", while the number of the p-value from each distribution is presented in the column 1, 2, ..., n.

The p-value of each test is the probability for the calculated chi-square to be higher than a uniformly distributed random variant.

Consequently, are three situations [7]:
- If $pv > 0.01$, then the generator is good;
- If $pv < 0.001$, then the generator is certainly bad;
- If $0.001 < pv < 0.01$, the test must be repeated.
As we can see in the tables in figure 5a and 5b for each of the tests performed, only for Quantum Random Number Generator QRNG the obtained values are higher than 0.01. The meaning is there generator can be used for applications in cryptography, having unpredictable results.

The most of classic random generator RNG are lower than 0.01, this generator is certainly bad for cryptography.

Comparing the values between them, there can see that those obtained from the sequences of quantum generator QRNG are better in all the 20 tests, which means that the output data appears more randomly than the sequences of classic generator RNG.

Figures 6a and 6b present graphically the distributions of the ordered p-values obtained for each of the tests, showing for each Kolmogorov-Smirnov KS and Anderson-Darling AD values, in agreement with Pearson table.

In these graphs, the points are the p-values and the bold line is the ideal position of the p-values; it starts from (1; 0) and ends at (n; 1).

The values of a good generator are as close as possible to the ideal black line in the graph.
Comparing the two graphs, we can see that the number of the values which are closest to the ideal line is higher for Quantum Random Number Generator (QRNG) than classic Random Number Generator (RNG). Once again it can be concluded that the RNG’s sequences are predictable, therefore, this one cannot be used in cryptographic applications.

**Conclusions**

The only way to generate true random numbers, which are impossible to predict or to determine, is by using a hardware device based on the measurement of different physical phenomena: sound intensity, temperature variations, etc. However, even these apparently complex physical phenomena hide determinist processes.

It is noteworthy that the importance of the quality of a generator depends on the type of application used: it is less important in applications using random numbers only for graphical effect illustrations, and it becomes very important in other applications, for instance in the approximation of integrals by Monte-Carlo method.

In addition, the production of high-quality random numbers may be too time consuming, making such production undesirable when a large quantity of random numbers is needed. To produce large quantities of random numbers, pseudo-random number generators may be preferable.

**Reference**


