A Concise Introduction to Random Number Generators

Peter Hellekalek

November 15, 2004

1 Introduction

Since John von Neumann we know that anyone who generates random numbers (or random bits) on a deterministic machine "is in a state of sin" (see Knuth[7]).

It is difficult to define or measure the extent of this sin by mathematical means.

History shows that the generation and the application of random numbers is more difficult than one might expect. We refer to the story on randomness and the Netscape browser, (see `http://www.cs.berkeley.edu/~daw/rnd` for details), and to a tragic historical example, called the Rosenberg story and the VENONA files, see Kahn[6] and `http://www.nsa.gov/venona` .

1.1 What is our goal?

We want a device (hardware or algorithm) whose output is computationally indistinguishable from truly random sources. We would like to generate bits (or numbers) that *appear* like being sampled from a uniform distribution on $\{0, 1\}$ (or $[0, 1[$), independently of each other.

What we get in practice from algorithms like AES, TT800, ... and hardware generators are finite bit strings that pass many tests of randomness.

So, what is *randomness?*

1.2 What is this talk about?

In this talk, we will discuss criteria for random number generators. We will present some of the main theoretical concepts and we will also have a closer look at statistical testing.

2 Quotes

When generating random numbers or bits on a computer, we will be restricted to finite bit strings. What is a finite random sequence? In the words of Kolmogoroff,

> *"A finite sequence is random if there is no short sequence that describes it fully, in some unambigous mathematical notation."*

In the very same spirit, Calude[1] defines

> *"A string is random if it cannot be algorithmically compressed."*

Hence, in the context of Kolmogorov complexity,

$$\text{Randomness} \;=\; \text{Incompressibility}.$$

For practitioners it will be interesting (and amusing) to know what theoreticians think of their theoretical findings:

> *"In the previous section, we recommended new definitions of pseudorandomness in terms of strong unpredictability and Kolmogoroff complexity. In practice, we may not need such stronger definition.*
>
> *. . . it is practically sufficient to consider the sequences which withstand the following five basic tests: frequency test, serial test, polar test, runs test, autocorrelation test."*

Randomness of finite sequences is in the eye of the beholder. As Kendall says,

> *"Finite sequences can only be random with respect to the tests of randomness used."*

Therefore, we will have to study tests for randomness a little bit closer.

3 Types of Random Number Generators

We may distinguish random number generators (RNGs) by their construction principle:

- hardware-based RNGs,
- software-based (algorithmic) RNGs,

and by the intended application:

- RNGs for stochastic simulation (keyword: Monte Carlo method),
- RNGs for cryptographic applications.

Software-based RNGs, i.e., *algorithms* to produce random numbers, may be divided into:

- linear algorithms, and
- nonlinear algorithms.

Deterministic algorithms to produce random numbers are often called *pseudo*-random number generators. We will not use this notion here.

Every RNG has its advantages and its deficiencies. No RNG is perfect. RNGs are tools and the appropriate range of applications for a given RNG has to be chosen with care.

Hardware RNGs rely upon physical phenomena like electronic noise or radioactive decay. Their properties are the following:

- slow,
- not portable,
- unpredictable output,
- theoretical analysis is impossible.

A typical example of a hardware RNGs is SG100, which is based on electronic noise. Software RNGs employ deterministic algorithms to stretch a short seed

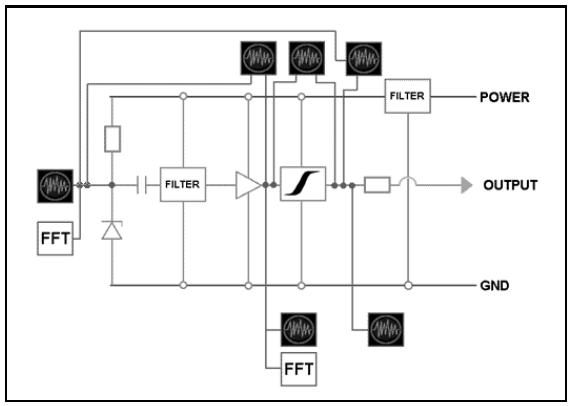

Figure 1: SG100 (Protego SA, Sweden)

into a long sequence. They have the following properties:

- (usually) very fast,
- portable,
- generate reproducible output,
- theoretical analysis is possible.

These algorithms may be divided into two classes, linear and nonlinear RNGs. Linear RNGs are based on a linear arithmetical operation, like a linear congruence, a linear shift register, etc. These RNGs are well-known in simulation, but insecure for cryptography.

Nonlinear RNGs employ nonlinear arithmetical operations like modular inversion in a finite field, etc. Some of these generators are quite efficient, but most are much slower than their linear counterparts. Nonlinearity does not necessarily mean cryptographic security (see Menezes et al.[11] for references). Well-known examples of nonlinear RNGs are inversive-congruential generators (ICGs; see Hellekalek[4] for a survey), AES, the advanced encryption standard (see Daemen and Rijmen[3] for the algorithm and Hellekalek and Wegenkittl[5] for statistical results), and the stream ciphers of modern applied cryptography.

4 Criteria

What is a good RNG?

The answer to this question will depend on the application. Three aspects will have to be studied in the assessment of an RNG:

- Is there some kind of theoretical analysis of the algorithm available?
- What statistical test results are available?
- What are the practical aspects of our RNG?

4.1 Theoretical Analysis

Theoretical analysis of a software-based RNG refers to the mathematical study of properties like the period length, the inner structure, and correlations within output streams.

If we cannot say anything about the period length of the output streams, an important information will be missing . For most applications, such RNGs should be avoided. If we have no idea about the existence or absence of simple (geometric) structures or simple bit patterns in the output stream then, again, we should have second thoughts about using this RNG. If such intrinsic structures have not been studied at all, then not only the security of our RNG is at risk, but also stochastic simulations may go wrong (see Compagner[2]):

> *"Monte Carlo results are misleading when correlations hidden in the random numbers and in the simulated system interfere constructively."*

There is an important difference between cryptographic RNGs and simulation RNGs. The algorithms for simulation RNGs are much simpler than their counterparts in cryptography. As a consequence, it is much easier to derive theoretical results for a simulation RNG than for a cryptographic RNG. In cryptography

linear algorithms are avoided for reasons of security. One uses highly nonlinear algorithms. Hence, in most cases, there is no chance to carry out the type of number-theoretic correlation analysis that is so common among simulation RNGs (for the latter, see Niederreiter[13]).

4.2 Statistical Testing

Statistical testing of RNGs means that the RNG is treated as a black box. No use is made of the knowledge of the algorithm.

Given observations $x_1, \ldots, x_n$ of the random variables $X_1, \ldots, X_n$, we want to estimate some parameter or figure of merit θ or the distribution of the X_i's.

We employ a function $\hat{\theta}$, the so-called test statistic, which assigns to the sample $x_1, \ldots, x_n$ the value $\hat{\theta}(x_1, \ldots, x_n)$. The function $\hat{\theta}$ should be chosen such that $\hat{\theta}(x_1, \ldots, x_n)$ is as close as possible to the target θ. Statistical test designs differ by the choice of the target distribution and, if the same distribution or parameter θ is considered, by the choice of the test statistic $\hat{\theta}$. Some test statistics are numerically more efficient than others, and some have better theoretical properties or better convergence rates (if the sample size n tends to infinity).

In order to illustrate such a situation, we will compare entropy estimators like the approximate entropy of Pincus and Singer[14, 15], Maurer's[10] Universal Test, and the well-known overlapping serial test (see Wegenkittl[17] for the relationship between these statistics).

For random number generation, there are two well-known batteries of statistical tests, DIEHARD of Marsaglia (see `http://www.cs.hku.hk/~diehard` for a recent version), and the NIST test suite [12]. The extensive new test bench TestU01 of L'Ecuyer and Simard[9] will provide much more extensive possibilities for statistical testing (see `http://www.iro.umontreal.ca/~simardr/`).

5 Examples and Links

In the final part of my talk, I will point out an interesting concept called "HAVEGE" by Seznec and Sendrier[16] to gather entropy that appears much more effective (and secure) than, for example, `dev/random`.

As starting points to the topics of RNGs for simulation, I would like to recommend the excellent survey of L'Ecuyer[8]. David Wagner has collected numerous links to randomness for crypto, see `http://www.cs.berkeley.edu/~daw/rnd/`.

References

[1] C. Calude. *Information and Randomness.* Springer Verlag, 1994.

[2] A. Compagner. Operational conditions for random-number generation. *Phys. Review E*, **52**:5634–5645, 1995.

[3] J. Daemen and V. Rijmen. *The Design of Rijndael.* Springer Verlag, New York, 2002.

[4] P. Hellekalek. Inversive pseudorandom number generators: concepts, results, and links. In C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, editors, *Proceedings of the 1995 Winter Simulation Conference*, pages 255–262, 1995.

[5] P. Hellekalek and S. Wegenkittl. Empirical evidence concerning AES. *ACM TOMACS*, **13**:322–333, 2003.

[6] D. Kahn. *The Code Breakers.* Macmillian, New York, 1967.

[7] D.E. Knuth. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, Reading, Mass., third edition, 1998.

[8] P. L'Ecuyer. Random number generation. In J.E. Gentle, W. Haerdle, and Y. Mori, editors, *Handbook of Computational Statistics*, pages 35–70. Springer, New York, 2004.

[9] P. L'Ecuyer and R. Simard. *TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators*, 2002. Software user's guide.

[10] U. Maurer. A universal statistical test for random bit generators. *J. Cryptology*, **5**:89–105, 1992.

[11] A. J. Menezes, P. C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997.

[12] National Institute of Standards and Technology (NIST) Special Publication 800-22. *A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications*, 2001. Available from `http://csrc.nist.gov/rng`.

[13] Harald Niederreiter and Igor E. Shparlinski. Recent advances in the theory of nonlinear pseudorandom number generators. In *Fang, Kai-Tai (ed.) et al., Monte Carlo and quasi-Monte Carlo methods 2000. Proceedings of a conference, held at Hong Kong Baptist Univ., Hong Kong SAR, China, November 27 - December 1, 2000*, pages 86–102. Springer, Berlin, 2002.

[14] S. Pincus and B. H. Singer. Randomness and degrees of irregularity. *Proc. Natl. Acad. Sci. USA*, **93**:2083–2088, 1998.

[15] S. Pincus and B. H. Singer. A recipe for randomness. *Proc. Natl. Acad. Sci. USA*, **95**:10367–10372, 1998.

[16] A. Seznec and N. Sendrier. HAVEGE: A user-level software heuristic for generating empirically strong random numbers. *ACM TOMACS*, **13**:334–346, 2003.

[17] S. Wegenkittl. Monkeys, gambling, and return times: Assessing pseudorandomness. In P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 625–631, Piscataway, N.J., 1999. IEEE Press.