

ANALYSIS OF A PSEUDORANDOM NUMBER GENERATOR WITH RANDOM CYCLES

Ciprian RĂCUCIU, PhD, Professor,

Dan GRECU, PhD, Lecturer,

Radu BORIGA, PhD candidate, Lecturer,

Ana Cristina DĂSCĂLESCU, PhD candidate, Lecturer

Titu Maiorescu University, Faculty of Computer Science, Bucharest

ABSTRACT:

The development of the informational society, which has led to an impressive growth of the information volume circulating in the computer networks, has accelerated the evolution and especially using the instruments of the modern cryptography. Should not be neglected but the fragility of the encryptions it is offered by the commercial products. The vulnerability of encryption systems can be much accelerated if the generating process of the keys is predictable. The proposed objective of this article is to identify a safe way to build a symmetric cryptographic keystream. In this respect, the article presents an ample statistical analysis of a pseudorandom number generator with random period, based on Fibonacci's recurrence, in order to validate it from a cryptographic point of view. The periods of the generated pseudorandom sequences are analyzed in detail and their random quality, too.

1. INTRODUCTION AND MOTIVATION

The symmetric algorithms string type can be used in cryptography for secret information exchange on the basis of a key that is shared in common both by the sender and the receiver, as well. Even though the type block algorithms are considered safer, the string type algorithms of the modern cryptography are four or five times quicker. Nevertheless, very few efficient string type algorithms have been published. In one of the amplest European projects that evaluate the cryptographic algorithms, *NESSIE*¹, the results of the chapter "string type algorithms" have been surprising: none of the proposed algorithms has fulfilled the security and performance conditions requested.

A symmetric algorithm of string type generates the keys string independent of the clear text and the cipher text. Thus, the running characters $x_1x_2x_3\dots$ from the clear text are combined with a string of keys $k_1k_2k_3\dots$. In this case, the encryption may be described by the equations $S_{i+1} = f(S_i, k)$, $z_i = g(S_i, k)$, $y_i = h(z_i, x_i)$, where S_0 is the initial status and can be determined by the key k , f is the status function, g is the function that produces the keys string z and h is the exit function which combines the string of keys with the clear text x_i in order to obtain the ciphered text y_i .

The main problem is to generate such a encryption key theoretically unlimited. This can be realized either randomly, either on the basis of an algorithm that starts from a small sequence of encryption keys. In this sense, the article presents a statistic analysis of a pseudorandom number generator with stochastic periods based on the Fibonacci recurrence modified through the disturbance parameters.

2. ABOUT PRNG

It is hard to imagine that a cryptographic application that is well defined does not use random numbers. The session keys, the initialization vectors, the single parameters from the digital signature operations, all of them have are based on random numbers.

Unfortunately, a lot of cryptographic applications are not based on a safe source of random bits such as thermal noise in electrical circuits or precise timing of Geiger counter clicks. Instead, in order to generate such random values, the cryptographic applications use a cryptographic mechanism named PRNG (Pseudo-Random Number Generator).

A pseudorandom number generator (PRNG) is an algorithm used for generating a sequence of numbers, relatively independent, that approximates certain properties of random numbers. Examples of this

¹ *New European Schemes for Signatures, Integrity and Encryption (NESSIE) is a project within the Information Society Technologies (IST) Programme of the European Commission.*

include the ANSI X9.17 key generation mechanism and the RSAREF 2.0 PRNG. Any *PRNG* has a secret state denoted by S . Upon request, it must generate outputs that are indistinguishable from random numbers to an attacker who doesn't know and cannot guess S . In this, it is very similar to a stream cipher. Additionally, however, a *PRNG* must be able to alter its secret state by processing input values that may be unpredictable to an attacker. A *PRNG* often starts in a state that is guessable to an attacker (usually unintentionally) and must process many inputs to reach a secure state. Sometimes, the input samples are processed each time an output is generated (e.g., ANSI X9.17). Other times, the input samples are processed as they become available (e.g. RSAREF 2.0).

In order to generate string of numbers the initial values x_1, x_2, \dots, x_k are established and form the seed of the generator using the recurrence relation:

$$x_n = g(x_{n-1}, x_{n-2}, \dots, x_{n-k}), \quad n > k, \quad g : M \rightarrow M \quad (1)$$

where M is a finite set, because the recurrent string x_n is periodic.

For the generator thus defined to be acceptable it must fulfill at least two conditions:

1. the period must be large relative to the generated number of values;
2. the generated values shouldn't sequential correlated: considering substrings of p values that are successively generated representing coordinates of points within the space with p dimensions; these points must fill out the p dimensional space. In the presence of a sequential correlation, these points group in a short number of hyperplanes. The fulfillment of these conditions can be ensured by a good choice of function g .

The pseudorandom number generators used in cryptographic applications, especially at the generation of keys, must fulfill certain quality conditions of the random generated bits. Especially, their output must be unpredictable in the absence of any information regarding the input data. The random degree of a generated string may be emphasized through statistic tests. These tests aim to determine whether a generator is qualified to be used in cryptographic purposes. In this respect several batteries of statistic tests, such as *NIST*, *Diehard*, etc. have been developed. They can confirm the random properties of a string. The interpretation of these deviations should take into consideration as possible causes the fact that the generator presents designing defects as well as the fact that the tested binary string presents abnormalities which is explainable by the appearance of the randomly generated data.

It makes the following assumptions on random binary strings to be tested:

1. *The uniformity*: in any moment of the bit string generation, the appearance probability of a zero or one is the same value, namely $1/2$. The expected number of zero bytes (respectively one) is $n/2$, where n is the length of the string in bytes.
2. *The scalability*: any test applicable to a string applies also to any string randomly extracted. So, any such substring should successfully pass any randomly test.
3. *The consistency* – the behavior of a generator must be consistent relative to initial values (seeds). The generator must be tested for different initial values.

3. THE PSEUDORANDOM NUMBER GENERATORS WITH RANDOM PERIOD RANROT

In the last years the pseudorandom number generators, based on Fibonacci recurrence, have become more and more popular both in serial and parallel applications. This is because it is easy to implement it and responds quite well to random tests. These generators, named in the specialized literature "*Lagged Fibonacci Generators (LFG)*", have been extremely studied. George Marsaglia has made an extensive study on these generators in order to determine their maximum period and how to select the parameters for the initial state. The general form of a LFG is $LF[r, s, m, \circ; x_i \{0, \dots, r-1\}]$, where $r > s > 0$ are gap parameters, \circ is a binary operation, m is the base and $x_i \{0, \dots, r-1\}$ is a sequence of r initial values (seed). For $n > r$ the recurrence may be defined as follows:

$$x_n = x_{n-r} \circ x_{n-s} \quad (2)$$

The regular operations are *addition*, *subtraction*, *multiplication* – mode m and *bitwise exclusive - or* if m is a power of two. Usually, $m = 2^N$ where N is the length of the memory word.

The maximum repetition period can be achieved only if certain conditions are satisfied by the initial values and by the gap parameters. Brenet has recently showed that if $m = 2^N$ and the polynomial $x^r + x^s + 1$ is irreducible primitive over $GF(2)$, the maximum repetition period is $p = 2^{N-1}(2^r - 1)$ ([2, 3]).

In practice, the LFG generators still shows a number of lacks. The output is sensitive to initialization conditions. Another problem is related to its random quality. Its is known the fact that LFGs don't pass certain random statistic tests, such as the *Birthday Spacing* test from the test battery Diehard proposed by Marsaglia. Moreover, by the way of implementation of the binary operations there is a leak of information from the least significant bits to the significant ones, information which does not transfer in contrary. To eliminate this shortcoming various options have been tried, such as transport to the most significant bit, operation which improves the length of the period but not the random character ([5]).

In 1997, the Danish *Agner Fog* proposes a new class of pseudorandom number generators based on the Fibonacci recurrence. Known as the *RANROT generators*, they realize a disturbance of the insignificant bits with the bit rotation operations ([4]).

Several types of generators *RANROT* are shown below. In the case of the *type A* generator the bits are rotated after addition and in the case of *type B* the bits are rotated turned before addition. There is a case when there are more than two terms, as the *type B3* below, and there is a case in which parts of the bits strings can be rotated, separately, as the generator of *type W*.

$$\text{Type A: } X_n = ((X_{n-j} + X_{n-k}) \bmod 2^b) \text{ rotr } r \quad (3)$$

$$\text{Type B: } X_n = ((X_{n-j} \text{ rotr } r_1) + (X_{n-k} \text{ rotr } r_2)) \bmod 2^b \quad (4)$$

$$\text{Type B3: } X_n = ((X_{n-i} \text{ rotr } r_1) + (X_{n-j} \text{ rotr } r_2) + (X_{n-k} \text{ rotr } r_3)) \bmod 2^b \quad (5)$$

$$\begin{aligned} \text{Type W: } Z_n &= ((Y_{n-j} \text{ rotr } r_3) + (Y_{n-k} \text{ rotr } r_1)) \bmod 2^{b/2} \\ Y_n &= ((Z_{n-j} \text{ rotr } r_4) + (Z_{n-k} \text{ rotr } r_2)) \bmod 2^{b/2} \\ X_n &= Y_n + Z_n \cdot 2^{b/2} \end{aligned} \quad (6)$$

Each X_n is an unsigned integer represented on b bits, Y_n and Z_n are unsigned integers represented on $b/2$ bits and i, j and k are unsigned integers with the property $0 < i < j < k$. The operation *rotr* represents a circular rotation of the bits to the right ([4]).

Initially proposed for applications like Monte Carlo, the RANROT generators can be successfully used in cryptography if the gap and rotation parameters are secret. Unlike conventional generators, the RANROT generators have random lengths of the cycle, i.e. inclusive their period is random. Also, the disturbance created by the bit rotation operations greatly increases the length from the LFG.

4. THE ANALYSIS OF THE PERIODS

Very few mathematical results have been derived about the randomness properties of RANROT generators, making it necessary to rely on statistical tests rather than theoretical performance. In this respect, it was done an exhaustive search of the periods of the generator RANROT of type W. For the RANROT generator type W (6), the disturbance is realized through circular rotations of the slopes of a string ([7]).

The values are calculated in a vector with k elements, named S_n . The initial status is $S_1 = (x_1, x_2, \dots, x_k)$ and the transition from one state to another is accomplished by a shift to the left of the form $(x_{n-k}, x_{n-k+1}, \dots, x_{n-1}) \rightarrow (x_{n-k+1}, x_{n-k+2}, \dots, x_n)$, where x_n is calculated according to the recurrence (4).

The seed of the generator W is given by the movement parameters j and k , circular rotation parameters

r_1, r_2, r_3, r_4 , the length of the buffer k and the initial state $S_1 = (x_1, x_2, \dots, x_k)$.

Even though the choice of parameters for RANROT generators is not critical, to obtain the best performance, a few rules must be complied ([4]):

1. an important aspect is that all bits in the buffer state to be independent. For this reason the movement parameters j and k must be prime numbers between them and their odd difference;
2. to eliminate possible symmetries in the generated string, the circular rotation parameters r_1, r_2, r_3, r_4 must be distinct and nonzero integers, prime with the length of the buffer k and the length of the memory word b ;

- the results are poor if the rotation parameters r_1, r_2, r_3, r_4 are too small or too close to the value $b/2$. Thus, the circular rotation parameters are generated as random numbers from interval $[0, b/2)$ that meet the restrictions from 2.

The seeding of the W generators was done randomly using random function. Therefore, are initially generated the movement and circular rotation parameters k, j, r_1, r_2, r_3, r_4 as well as the first k values of the initial state S_1 . The following values are calculated using the recurrence given by the formula (6).

The self-test proposed by Agner Fog is implemented by saving a copy of the initial contents of the state buffer. After each step of the algorithm, the current state of S_n is compared to the initial state S_1 . In this case, the maximum period that can be obtained for a generated pseudorandom string is 2^{k-b} . For this reason, the generated string may contain many permutations of the k values from the initial state. To eliminate this shortcoming, it is proposed a different the test: at every step of the algorithm the last generated number x_n is compared with the initial state S_1 . If the number x_n is found among the k values of the initial buffer, the algorithm stops.

To achieve an extensive statistical analysis of the period of the generator W, a large number of tests were required. Their management was assured by the automated system described in [1].

It have been generated 5000 pseudorandom strings and the obtained results are graphically represented, as follows:

- in figure 1 were represented the periods of the generated pseudorandom strings, sorted ascending.

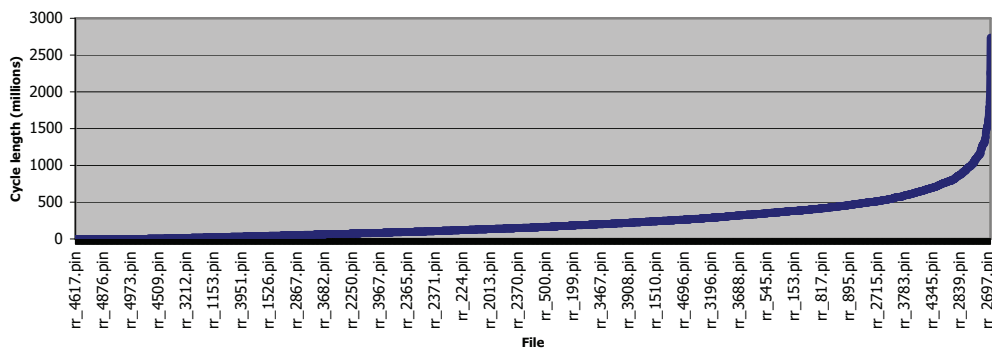


Fig. 1 The cycle lengths of the generated pseudorandom strings

- after we had established certain intervals for the periods, in figure 2 we have graphically represented their frequency along each interval.

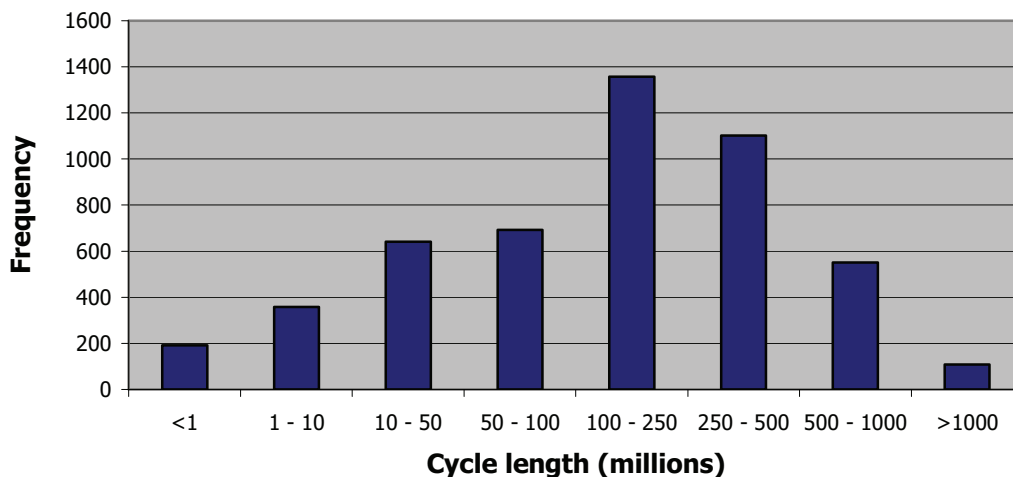


Fig. 2 Distribution of the cycle lengths

5. THE RANDOMNESS ANALYSIS

The generated pseudorandom string of W recurrence was put at the Diehard battery of statistical tests. Proposed by Marsaglia in 1997, to determine certain weaknesses of the PNRG, the battery includes 17 statistical tests. Most tests generate a p-value, which is distributed uniformly within interval (0, 1).

These p-values are obtained through $p=F(X)$, where F is the distribution function attributed to sample the random variable X. If a p-value is equal with 1, then the string is randomly perfect. A p-value of zero indicates a completely not random string. For tests is chosen a level of significance α . If the p-value is greater than α , then the string appears to be random. Typically α is chosen in the interval [0.001, 0.01]. A value $\alpha = 0.001$ indicates the possibility that 1000 of strings to be rejected by the random test. For a value $p \geq 0.001$ will be considered a random string with a confidence level of 99.9%. The Diehard battery of tests provides a total of 215 p-values. [6]

Using a sample of 2000 generated files, in figure 3 we have graphically represented, for each file, the number of valid p-values provided by Diehard battery of tests.

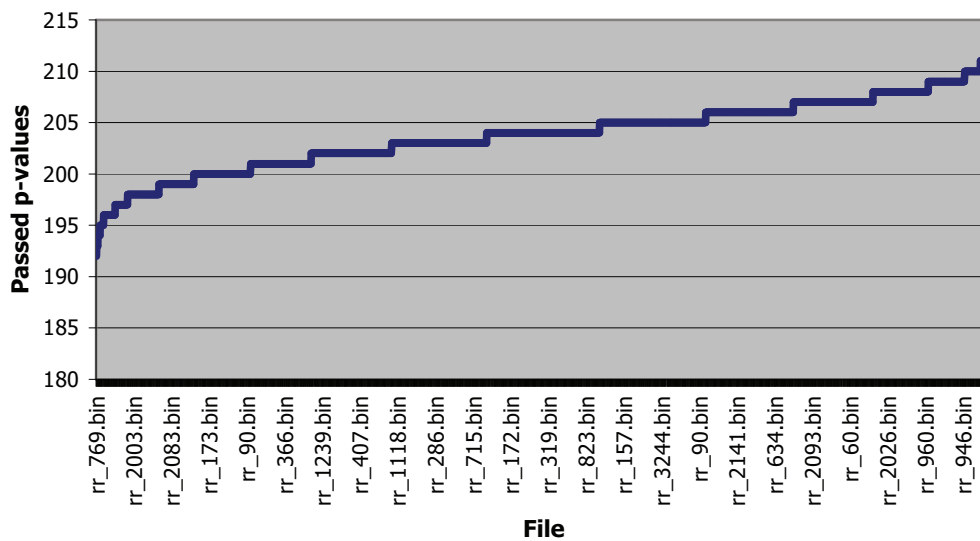


Fig 3. Passed p-values of the generated

In figure 4 we have graphically represented the distribution of the total number of valid p-values provided by Diehard for each generated string. It's easy to see that the worst generated strings have passed 192 p-values (in other words, almost 90% of the p-values generated by Diehard) and the best generated tests have passed 213 p-values (almost 99% of the p-values generated by Diehard).

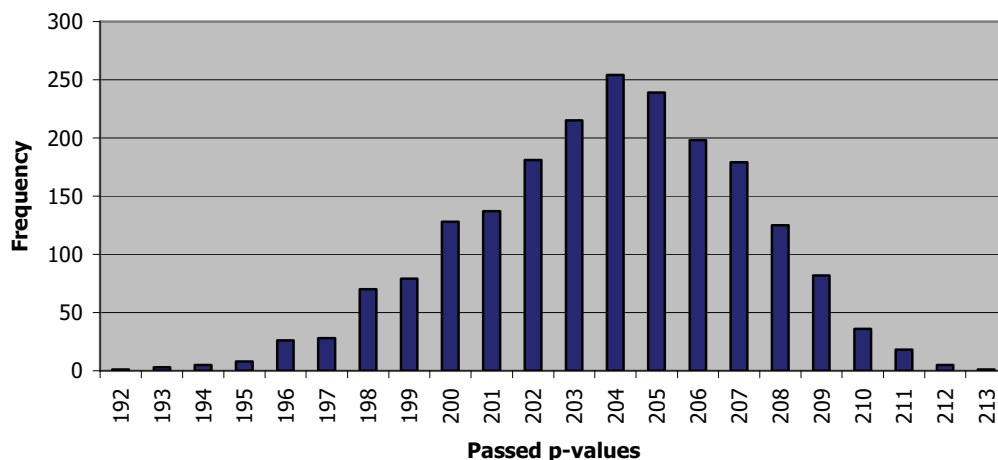


Fig 4. Distribution of passed n-values

The results obtained from the statistical analysis of the proposed generator, based on the RANROT generator of type W, validates its use for building a secure symmetric key.

6. CONCLUSIONS

In the current environment of information security it's mandatory to make a rigorous analysis of the encryption systems used to transmit secret information along the telecommunications networks. The proposed objective was to build a symmetric keystream, secure from a cryptographic point of view. In this respect, it was made an extensive statistical analysis of the proposed pseudorandom number generator, based on the Ranrot generator of type W. The period of the proposed generator was analyzed in detail, as well as its randomness. The obtained results, discussed above, lead us to the conclusion that the proposed generator can be used in cryptography, if its initial parameters are secret.

7. REFERENCES

- [1] Boriga, R. and Dăscălescu, A.C. - *An Automated System for Testing the Period and Randomness of a PRNG*, *Proceedings of the 4th edition of the International Conference "Education and Creativity for a Knowledge Society"*, Bucharest, 2010
- [2] Brent, R.P. - *Fast and reliable random number generators for scientific computing*, *Proceedings of PARA'04 Workshop on the State-of-the-Art in Scientific Computing*, pp. 1–10, 2004
- [3] Brent, R.P. - *On the periods of generalized Fibonacci recurrences*, *Math Compute* 63, pp. 398-412, 1994
- [4] Fog, A. - *Chaotic Random Number Generators with Random Cycle Lengths*, www.agner.org, 2001
- [5] Marsaglia, G. - *Diehard battery of tests of randomness*, *The Marsaglia random number CDROM*, Department of Statistics, Florida State University, 1995
- [6] Marsaglia G. and Tsang W.W. - *Some difficult-to-pass tests of randomness*, *Journal of Statistical Software*, Vol. 7, Issue 03, 2002
- [7] Răcuciu, C. - *Contributions related to the improvement of the Image Encryption Methods*, PhD Thesis, Bucharest, 2003