PRNG REQUIREMENTS FOR CRYPTOGRAPHIC APPLICATIONS

Marian CREȚU[1], Marin DRĂGULINESCU[1]
[1]Faculty of Electronics, Telecommunications and Information Technology,
University POLITEHNICA of Bucharest, Romania
[1]marian.cretu77@yahoo.com

Keywords: PRNG, random, feedback register, significance interval, periodicity.

Abstract: *Cryptographic applications are based on keys for encryption and decryption processes, in order to protect the transferred information between trusted entities. Keys must be strong enough to resist in case of cryptanalytic attacks. Their properties are given by the key generator who is, most of the time, a PRNG (Pseudo-Random Number Generator). Depending on how difficult it is to predict numbers obtained from a generator output, encryption keys can get strong or weak. PRNG periodicity and speed are fundamental factors in the choice of the proper key generator.*

1. INTRODUCTION

A fundamental cryptographic goal is to generate pseudo-random numbers, used mainly in the construction of cryptographic keys, unpredictable for cryptanalyst. A common method to build a cryptosystem is to operate XOR (Exclusive OR function) random bit string with the message to be encrypted.

A PRNG (Pseudo-Random Number Generator) is an algorithm that generates strings of numbers relatively independent from them and approximates some random numbers properties. A number itself cannot be random, rather than the way it has been generated. To generate a random number, means that all numbers before it had an equal probability of occurrence.

True random numbers can be generated using hardware generators. Because any PRNG running on a computer is a deterministic algorithm, the generated string will have different properties that a truly random string of numbers.

The periodicity property is guaranteed by the fact that the generator uses a fixed amount of memory, leading that after a sufficient number of iterations, algorithm will return to an initial internal state and will repeat its behavior in an infinite cycle. In practice, some PRNGs characteristics may prevent them from passing statistic tests:

- lack of uniformity;
- successive values may not be independent;
- poor dimensional distribution;
- some bits are "more random" than others;
- shorter than expected for certain initial conditions.

A PRNG successfully passing this test is called CSPRNG (Cryptographically Secure PRNG), meaning that it have good statistical properties and cryptanalytic resistance.

2. CRYPTOGRAPHICALLY SECURE PRNG (CSPRNG)

A CSPRNG must succeed the "next-bit" test, where giving the first k bits of a random number, there is no polynomial algorithm to predict the next bit with a probability greater than ½. It was proven that a generator passing

this test will successfully pass all other statistical tests.

A CSPRNG must resist if its condition is known, meaning that it must be impossible to rebuild the string previously generated and must also be extremely difficult to deduce the next state generator.

BBS (Blum Blum Shub) generator is a CSPRNG algorithm with a high security, but slow. A secure block cipher can be converted to a CSPRNG by running in counter mode. Choose an arbitrary key and encrypt a block of bits with all values 0, then 1, 2, ..., etc. Counter's initial value can be any arbitrary nonzero value. The CSPRNG period constructed this way will be 2^n for the case of a block of n bits.

Mathematically, generating a pseudo-random series of numbers is equivalent to a simulation of a discrete random variable uniformly distributed.

A discrete random variable X with values in the set {1, 2, ... , n} is uniformly distributed if $P(X = i) = 1 / n$ for $1 \leq i \leq n$. Simulation of such variables is based on a function g: $M^k \rightarrow M$ where M is a subset of natural numbers that can be represented by the computer. To generate a series of numbers the initial values $x_1, x_2, ..., x_k$ are chosen to create the generator *seed*, using the recurrence relation:

$$x_n = g(x_{n-1}\, x_{n-2} \ldots x_{n-k}),\ n > k\ ; \qquad (1)$$

Since M is a finite string, the result is a periodically recurring x_n. There are two conditions for this generator to be accepted:

1. the period must be large relative to the number of generated values;
2. the generated values must not be sequentially related.

These conditions can be ensured by the proper choice of g function. The most commonly used methods are based on functions characterized by a recurrence like:

$$x_n = f(x_{n-1}\, x_{n-2} \ldots x_{n-k}) \bmod m\ ; \qquad (2)$$

$f: M^k \rightarrow M$ is a function and k and m are values that define the generator. The most common used methods are those for which f is a linear function:

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \ldots + a_k x_{n-k} + c, \bmod m; \qquad (3)$$

$a_1, a_2, \ldots, a_k$, c and m are integers characterizing the generator. $c \in \{0, 1, ..., m - 1\}$ and $x_1, x_2, ..., x_k$ are initial values; the generated values are set in the interval $\{0, 1, ..., m - 1\}$. The maximum period is m and therefore m value is chosen as the largest positive integer that can be represented by computer.

Parameters a_i and c are difficult to choose and they are usually determined by efficiency tests. The most common generator in use is the order 1 generator:

$$x_{n+1} = ax_n + c, \bmod m; \qquad (4)$$

For $c = 0$, the generator is called multiplicative congruent, and for $c \neq 0$ the generator is called mixed congruent.

2.1 Linear congruential generators (LCG)

LCG generate pseudo-random strings:

$$x_n = (ax_{n-1} + b) \bmod m\ ; \qquad (5)$$

where x_n is the nth element of the sequence and x_{n-1} is the previous item. The variables a, b and m are constant. The seed is the x_0 value.

This generator has a period $\leq m$. For a, b and m are well chosen (b prime to m), the generator will be a generator of maximum period (or length), with period equal to m [1]. Constants selection that ensure maximum period is treated in [2].

These constants define the LCG successfully passing the test for dimensions 2, 3, 4, 5 and 6 [3]. The advantage of LCG is speed because it requires fewer operations.

Their disadvantage is that it cannot be used for cryptography, because they are predictable. In some cases the use of a linear congruential generator (LCRNG, which is the type usually used in programming language libraries) can interact with the cryptosystem it is used with, for example using an LCRNG or truncated LCRNG with DSA (digital signature algorithm) makes it possible to recover the signer's secret key after seeing only three signatures [2, 3, 19].

All the LCG (linear, quadratic and cubic) were broken [5, 6].

ISSN – 1453 – 1119

2.2 Linear feedback shift registers (LFSR)

A LFSR is composed of two main parts: a shift register and a feedback function. Shift register is a string of bits. Register length is expressed in bits (for *n*-bit length is called *n*-bit). For one generated bit, all bits in the register are shifted to the right. The output register is the bit on the most right position that leaves the register. Register is complemented with a new bit in the most left position, which is calculated as a function value of other bits (Fig.1). Ciphers using shift registers are easy to implement [7].

A LFSR generator of *n*-bit can take one of the $2^n - 1$ possible states (excluding state in where all register bits are 0) and can generate a number of $2^n - 1$ pseudo-random bits with no repetition.

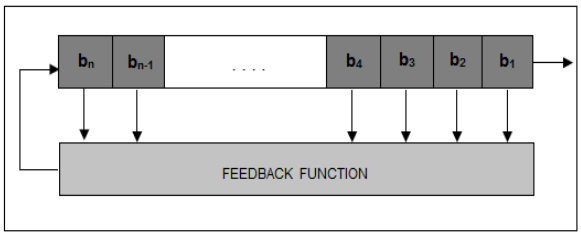

Fig.1. Linear feedback shift register

A detailed implementation and analyze for linear feedback shift register generators can be found in [8].

2.3 Generalized feedback shift register (GFSR)

GFSR is an attempt to improve the results of statistical tests and is based on the theory of primitive trinomials $x_p + x_q + 1$ [9]. GFSR can be expressed by linear recurrence equation:

$$x_i = x_{i-p} \oplus x_{i-q} ; \qquad (6)$$

where each x_i is a *w* size vector with components 0 or 1. $2^p - 1$ is the maximum period for this generator and is obtained when trinomial primitive $x^p + x^q + 1$ divides $x^n - 1$ and $n = 2^p - 1$ is the smallest value of *n*. The maximum period can be obtained if *n* is a Mersenne prime number. A number like $M_n = 2^n - 1$ where *n* is an integer is called Mersenne number and for M_n prime number, it is called Mersenne prime number [10].

2.4 Feedback with carry shift registers (FCSR)

A FCSR is similar to a LFSR generator. They share the shift register and feedback function but the difference is that FCSR uses a transport registry. Rather than operate XOR bits of the "tap", sequence of bit sums and gather with the contents of transport registry. The result is the new bit modulo 2 and it can be found in the left position of the shift register. This result is divided by 2 and becomes the new contents of the transport registry.

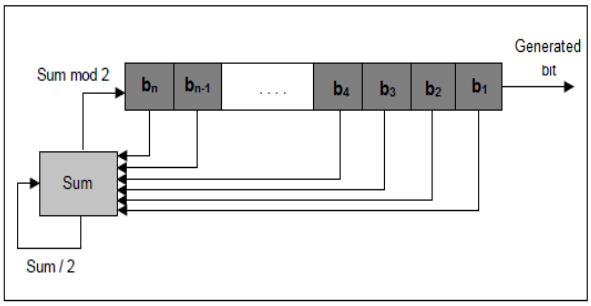

Fig.2. Feedback with carry shift register

As we can see in figure 2 the transport register is not a singular bit, but a number. Length in bits of the transport register must be at least *log2t*, where *t* is the number of bits in the "tap" sequence. The maximum period for FCSR is *q -1*, where *q* is the integer connection, which defines the "tap" sequence and is a prime:

$$q = 2q_1 + 2^2q_2 + 2^4q_4 + ... + 2^nq_{n-1} ; \qquad (7)$$

where *2* must be a primitive root.

If the FCSR leads to a *n*-bit string of 0 or 1, the initial state must be rejected. Because the initial state of a FCSR sequence corresponds to a string cipher key, it means that a generator based on FCSR will provide a lot of weak keys.

2.5 Nonlinear feedback shift registers (NFSR)

If in a LFSR or FCSR is used a more complicated feedback shift register, a nonlinear feedback shift register is obtained. The main problem is the lack of a mathematical theory to support their analysis and in practice may happen that:

- the output string to contain significantly more bits of 1 than 0;
- the generator period depend on the initial value;
- the output string is initially random, but go to a single cyclic value;
- the maximum period to be smaller than that expected.

The advantage is that the difficulty of theoretical analysis of generator's non-linearity is kept in cryptanalysis plan, with fewer ways to attack ciphers based on this type of number generators.

3. RNG REQUIREMENTS TO QUALIFY AS CSPRNG

There are some improvements that can be made to the RNG to increase its security against reverse engineering and solidify its classification as a CSPRNG:

- ***Using a prime number as the cache size.*** This is more useful when the RNG uses a small cache (<1000000 bits) but for all cache sizes this means that the cache cannot be filled with a set number of repeating strings.
- ***An optional mechanism to delete cache data behind the engine process.*** If the RNG is operated in a situation where it may be compromised and its state becomes known, deleting a small amount of cache data behind the engine process assures that the engine can never be successfully run in reverse. Fortunately the chaotic nature of the RNG makes it very tolerant of poor seeding and/or entropy loss from the cache. Suggested deletion method is to alternate forcing bits to 0 and 1, so their data is cleared but the average balance of 0 and 1 bits in the cache remains.
- ***Partial re-seeding of the cache.*** If the RNG is partially re-seeded during operation with a separate entropy source the Hybrid RNG result provides numerous benefits. Problems of potential global repeat can be defeated with an input either sequential or fractal, of a period larger than the expected use of the RNG. In the case that the period of use of the RNG is unknown or that it would operate continually, in risk that its state may be reverse engineered, a simple natural entropy source of any type can be used to perform a partial re-seeding of the cache data.

4. STATISTICAL TESTS

4.1 Random appearance

PRNG are used in cryptographic applications, particularly in key generation and should meet certain randomness conditions. Their output must be unpredictable in the absence of any information concerning input. Randomness degree of the generated string may be revealed by statistical tests, determining whether a generator is qualified to be used in cryptography.

Statistical tests cannot replace cryptanalysis, and therefore do not provide an absolute guarantee that a particular generator is suitable for cryptographic applications [11].

In [12] Ueli Maurer has proposed the famous test based on a statistic asymptotically related to the source entropy. This test consists in counting the distances between patterns in the output data stream.

In [13], J-S Coron and D. Naccache have proposed to modify this test in order to more fit precisely the source entropy (see [14] for the latest version of this test which is now part of [15]).

The first approach developed in [16] and [17] was to qualify RNG using statistical tests.

Canonical statistical tests include for instance the frequency test aimed to check uniformity of the outputs of the RNG and the long run test that verifies whether the RNG is not stuck at a given value during a defined period.

Classical statistical tests are combined with theoretical analysis of the cryptographic properties of the underlying algorithm [18].

On the other hand, TRNGs (true random generators) are trickier to evaluate. The AIS 31 standard [15] has defined a way to qualify the expected quality of TRNG.

NIST (National Institute of Standards and Technology) has developed a set of statistical

tests in order to determine the deviation of a binary string from randomness.

Pseudo-random numbers are required in most cryptographic applications, generally for encryption/decryption keys.

Many cryptographic protocols require random inputs such as digital signatures or authentication process.

4.2 Unpredictability

The PRNG generated numbers must be unpredictable. For the initial values not known, the next generated number must not be predictable, regardless of numbers previously known.

The reverse condition must be respected as well as for any generated value, the previous or initial values must be impossible to predict.

This is called forward and backward unpredictability and there should be no obvious correlation between initial values and any of the values generated from these.

All the elements of the output generated sequence should appear as an independent random event with probability ½.

4.3 String test

Properties of a random number can be described in terms of probabilities. There are an infinite number of possible statistical tests, each looking for the presence or absence of a "pattern" which, if detected, would indicate that the string is not random.

A statistical test is testing a specific null hypothesis (H_0), proving that the tested string is random. H_a is the alternative hypothesis and it assumes that the string is not random. For each test, the null hypothesis acceptance or rejection will lead to the conclusion that the generator produces or not random values.

Statistical hypothesis testing is a procedure for generating decision with two possible outcomes: H_0 or H_a. There are two types of errors: type I - the data are random but the hypothesis H_0 is rejected and type II - data are not random, but the hypothesis H_0 is accepted. Type I error probability is called level of significance and is denoted by α and it can be fixed before the test. α is the probability that the data is random while the test indicates that they are not random. In cryptography the common value for α is 0.01.

The probability of a type II error is denoted by β. β is the probability that the data are not random while the test indicates that they are random. β is not a chosen value, but may take different values corresponding to a number of different ways that may not be random.

Each test is based on a calculated value of statistic tests. If this value is denoted S and the critical value t, then:

α = type I error probability = $P(S>t||H_0$ is true)= P(reject $H0|H0$ is true)

β = type II error probability = $P(S\leq t||H_0$ is false)= P(accept $H0|H0$ is false)

If $p = 1$, the string appears to be perfectly random. $p = 0$ indicates a complete nonrandom string. For testing, α value is chosen for the level of significance. For $p \geq \alpha$ the null hypothesis is accepted, so the string appears to be random. For $p < \alpha$ the null hypothesis is rejected and the string is not random. α parameter indicates the a type I error. Usually α is chosen in the interval [0.001, 0.01].

A value $\alpha = 0.001$ indicates that one of 1000 strings is rejected by statistical test. For $p \geq 0.001$ a string will be considered random with a confidence threshold of 99. 9%.

5. CONCLUSIONS

For a PRNG to be considered cryptographically secure, following assumptions are made on randomly generated binary strings:

1. Uniformity: at any time of the bit string generation, the likelihood of a zero or one is the same, with the value ½. Expected number of 0 or 1 bits is $n / 2$, where n is the length of the string in bits.
2. Consistency: the behavior of a generator must be consistent relative to seeds. The generator must be tested for different input values.
3. Scalability: any test applicable to a string also applies to any substring randomly extracted. If a string is random, any of its substrings is also random, concluding that any such substring should successfully pass any test.

In this paper we realized a theoretical analyze of some well-known PRNG types and

the necessary requirements to be considered as CSPRNG.

In [19], a complex implementation and the verification techniques of these requirements are described by Peter Gutmann.

In addition, a number of studies can provide general advice on using and choosing random number sources [20, 21, 22].

6. REFERENCES

[1]. Sibley E.H., *"Random Number Generators: Good Ones Are Hard to Find"*, Communications of the ACM, v.31, n.10, Oct. 1988;

[2]. Press W.H., Flannery B.P., Teukolsky S.A., Vetterling W.T., *"Numerical Recipes in C: The Art of Scientific Computing"*, Cambridge University Press, 1988;

[3]. Coveyou C., MacPherson R.D., *"Fourier Analysis of Uniform Random Number Generators"*, Journal of the ACM, v.14, n.1, 1967;

[4]. Bellare M., Goldwasser S., Micciancio D., *"'Pseudo-random' Number Generation Within Cryptographic Algorithms: The DDS Case"*, Proceedings of Crypto'97, Springer-Verlag Lecture Notes in Computer Science No.1294, August 1997;

[5]. Reeds J.A., *"Cracking Random Number Generator"*, Cryptologia, n.1, 1977;

[6]. Krawczyk H., *"How to Predict Congruential Generators"*, Journal of Algorithms, v.13, n.4, Dec 1992;

[7]. Golomb S.W., *"Shift Register Sequences"*, San Francisco: Holden-Day, 1982;

[8]. Lewis T.G., Payne W.H., *"Generalized Feedback Shift Register Pseudorandom Number Algorithm"*, Journal of the Association for Computing Machinery, 1973;

[9]. Dascalescu A.C., Nidelea M., *"A Novel Method for Generating Encryption Keys"*, Scientific Bulletin of the Petru Maior University of Tirgu Mures, Vol. 6 (XXIII), 2009;

[10]. Caldwell C., *"Mersenne Primes: History, Theorems and Lists"*, University of Tennessee at Martin, 1994;

[11]. Meyer C.H., Tuchman W.L., *"Design Considerations for Cryptography"*, Proceedings of the NCC, v.42, AFIPS Press, Nov.1979;

[12]. Maurer, U. (1992). A universal statistical test for random bit generators. *J. Cryptology*, 5, no. 2, 89-105.

[13]. Coron, J.S., Naccache, D. *"An Accurate Evaluation of Maurer's Universal Test"*, in Proceedings of Selected Areas in Cryptography 98. LNCS 1556 P57-71. Springer-Verlag, 1998.

[14]. Coron, J.S. (1999). *"On the security of Random sources, in Public Key Cryptography: Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99"*, Kamakura, Japan, March 1999. Proceedings Editors: H. Imai, Y. Zheng (Eds.): vol. 1560. Lecture Notes in Computer Science, 1999;

[15]. *"Functionality Classes and Evaluation Methodology for True (physical) Random Number Generators"*, Version 3.1 September 2001. Online at: http://www.bsi.bund.de /zertifiz/zert/interpr/trngk31e.pdf

[16]. National Institute of Standards and Technology, *"Security Requirements for Cryptographic Modules, FIPS 140-1"*, Jan. 1994. Online at : http://www.nist.gov/itl /div897/pubs/fip140-1.htm.

[17]. National Institute of Standards and Technology, *"Security Requirements for Cryptographic Modules, FIPS 140-2"*, May. 2001. Online at: http://csrc.nist.gov /publications/fips/fips140-2/fips1402.pdf

[18]. *"Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators"*, Version 2.0 Dec. 1999. www.bsi.de/zertifiz/zert/interpr/ ais20e.pdf

[19]. Gutmann P., *"Cryptographic security architecture: design and verification"*, Springer-Verlag 2004.

[20]. Matthews T,. *"Suggestions for random number generation in software"*, RSA Data Security Engineering Report, 15 Dec. 1995 (reprinted in RSA Laboratories' Bulletin No.1, 22 January 1996).

[21]. Schneier B., *"Applied Cryptography (Second Edition)"*, John Wiley and Sons, 1996.

[22]. *"Cryptographic Random Numbers"*, IEEE P1363 Working Draft, Appendix G, Feb.1997.