

# Random Number Generator

Petrus E. Manurung 13518110  
 Program Studi Teknik Informatika  
 Sekolah Teknik Elektro dan Informatika  
 Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia  
 elison.petrus@itb.ac.id

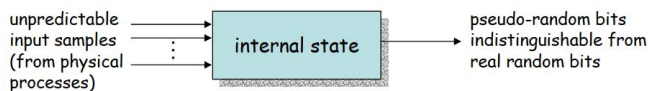
**Abstract**—One use of discrete mathematics (number theory) is in making a computer program that can produce random numbers. However, because computers are deterministic systems, no computer can produce truly random numbers. Instead, what is used to generate ‘randomness’ is a modulo operation. This is called pseudo random (‘false’ randomness).

**Keywords**—generate, mod, pseudo, random.

## I. INTRODUCTION

A random number is a number that cannot be predicted by an observer before it is generated. If the number is generated within the range  $[0, N-1]$ , then its value cannot be predicted with any better probability than  $1/N$  (this is true even if the observer is given all previously generated numbers).

A cryptographic pseudo-random number generator (PRNG) is a mechanism that processes somewhat unpredictable inputs and generates pseudo-random outputs. If designed, implemented, and used properly, then even an adversary with enormous computational power should not be able to distinguish the PRNG output from a real random sequence.



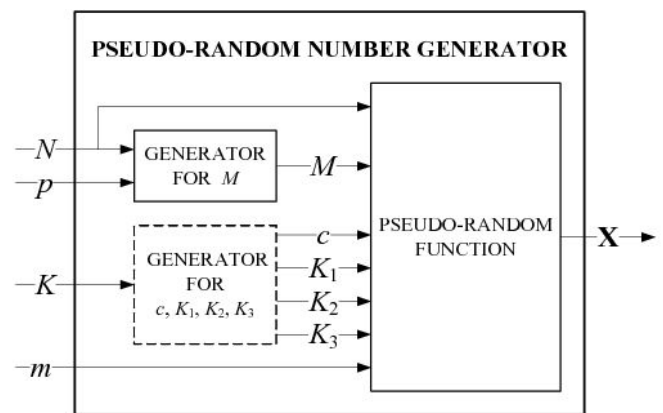
Picture 1. From Levente Buttyan. Illustration of pseudo random generator.

A random number generator is an algorithm that, based on an initial seed or by means of continuous input, produces a sequence of numbers or respectively bits. We demand that this sequence appears “random” to any observer.

This topic leads us to the question: What is random? Most people will claim that they know what randomness means, but if they are asked to give an exact definition they will have a problem doing so. In most cases terms like unpredictable or uniformly distributed will be used in the attempt to describe the necessary properties of random numbers. However, when can a particular number or output string be called unpredictable or uniformly distributed?

In the context of random numbers and RNGs the notions of “real” random numbers and true random number generators (TRNGs) appear quite frequently. By real random numbers we

mean the independent realizations of a uniformly distributed random variable, by TRNGs we denote generators that output the result of a physical experiment which is considered to be random, like radioactive decay or the noise of a semiconductor diode. In certain circumstances, RNGs employ TRNGs in connection with an additional algorithm to produce a sequence that behaves almost like real random numbers.



Picture 2. From semanticscholar.org. Illustration of pseudo random generator.

There are many ways of generating pseudo random number. One of them is Linear Congruential Generator(LCG).

## II. MODULO

A modulo is an operation that gives the remainder of a division as the result. A modulo can be expressed in

$$a \bmod n = p;$$

where  $a$  and  $n$  are integer, an  $p$  is an integer  $\geq 0$ .

Example:

1.  $27 \bmod (5) = 2$
2.  $48 \bmod (7) = 6$

Here are the properties of modulo:

1. Identity:  
 $(a \bmod n) \bmod n = a \bmod n$ .  
 $nx \bmod n = 0$  for all positive integer values of  $x$ .  
 If  $p$  is a prime number which is not a divisor of  $b$ , then  
 $abp-1 \bmod p = a \bmod p$ , due to Fermat's little

theorem.

2. Inverse:  

$$[(-a \bmod n) + (a \bmod n)] \bmod n = 0.$$
 $b^{-1} \bmod n$  denotes the modular multiplicative inverse, which is defined if and only if  $b$  and  $n$  are relatively prime, which is the case when the left hand side is defined:  $[(b^{-1} \bmod n)(b \bmod n)] \bmod n = 1.$
3. Distributive:  
 $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n.$   
 $ab \bmod n = [(a \bmod n)(b \bmod n)] \bmod n.$
4. Division (definition):  $a/b \bmod n = [(a \bmod n)(b^{-1} \bmod n)] \bmod n$ , when the right hand side is defined (that is when  $b$  and  $n$  are coprime). Undefined otherwise.

Inverse multiplication:  $[(ab \bmod n)(b^{-1} \bmod n)] \bmod n = a \bmod n.$

### III. LINEAR CONGRUENTIAL GENERATOR(LCG)

As a first important class of elementary “classical” pseudo-random number generators we consider one-step recursive formulas that use linear congruences. They are very fast, have long periods, and their quality is easily analyzed due to their plain structure.

This simple formula generates a sequence of pseudo-random numbers:

$$(1) X_n = AX_{n-1} + B.$$

The recursive sequence  $(X_n)_{n \in \mathbb{N}}$  depends on four integer parameters:

- the module  $m$  where  $m \geq 2$ ,
- the multiplier  $A \in [0 \dots M - 1]$ ,
- the increment  $B \in [0 \dots M - 1]$ ,
- the initial value  $X_0 \in [0 \dots M - 1]$ .

We call this recursive formula a linear congruential generator, in the case  $B = 0$  also a multiplicative generator, in the case  $B \neq 0$ , a mixed congruential generator. Furthermore we call

$$S : \mathbb{Z}/m\mathbb{Z} \longrightarrow \mathbb{Z}/m\mathbb{Z}, S(X) = AX + B \bmod M.$$

the generating function of the generator. Formula (1) then becomes

$$X_n = S(X_{n-1}).$$

Programming a congruential random generator is extremely easy, even in assembler languages. The algorithm works very fast. Moreover the pseudorandom numbers are good if the parameters  $m, a, b$  are suitably chosen. In contrast the choice of the initial value is unrestricted. This freedom allows a reasonable variation of the generated pseudo-random numbers.

Use of the pseudo-random sequence as a bitstream for XOR encryption requires that we consider the initial value  $X_0$ , or the complete parameter set  $(M, A, B, X_0)$ , as effective key, and keep it secret.

Remarks and examples of LCG(from K. Pommerening):

1. Since  $X_n$  may assume only  $m$  different values the sequence is periodic with a period length  $\leq M$ ; including a possible preperiod.

2. Choosing  $A = 0$  obviously doesn't make sense. Also for  $A = 1$  we get a useless sequence, namely  $X_0, X_0 + B, X_0 + 2B, X_0 + 3B, \dots$ , that also mod  $m$  contains several regular subsequences.

3. For  $M = 13, A = 6, B = 0, X_0 = 1$  we get the sequence 6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1 of period length 12 that looks like a fairly random permutation of the integers 1 to 12, despite the small module.

4. Choosing the multiplier  $A = 7$  instead of 6 we get a much less sympathetic sequence: 7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1.

5. If  $A$  and  $M$  are coprime, then the sequence is purely periodic (no preperiod). For  $A \bmod M$  is invertible, hence  $AC \equiv 1 \pmod{M}$  for some  $C$ . Thus always  $X_{n-1} = CX_n - CB \pmod{M}$ . If  $X_{\mu+\lambda} = X_{\mu}$  with  $\mu \geq 1$ , then also  $X_{\mu+\lambda-1} = X_{\mu-1}$  etc., finally  $X_{\lambda} = X_0$ .

6. By induction we immediately get (2)  $X_k = A^k X_0 + (1 + A + \dots + A^{k-1}) \cdot B \bmod M$  for all  $k$ —a definite warning about the poor randomness of the sequence: Formula (2) allows direct access to any element of the sequence. Note that the coefficient of  $b$  is  $(A^k - 1)/(A - 1)$  where the division is mod  $m$ .

7. Let  $m = 2^e$  and  $A$  be even. Then  $X_k = (1 + A + \dots + A^{e-1}) \cdot B \bmod M$  for all  $k \geq e$ , hence, after a certain preperiod, the period has length 1. More generally common divisors of  $a$  and  $m$  reduce the period. We want to avoid this effect.

8. Let  $d$  be a divisor of  $m$ . Then the sequence  $Y_n = X_n \bmod D$  is the analogous congruential sequence for the module  $d$ , generated by the formula  $Y_n = AY_{n-1} + B \bmod D$ . Hence the sequence  $(X_n)$ , if considered mod  $D$ , has a period  $\leq D$  that might be very short.

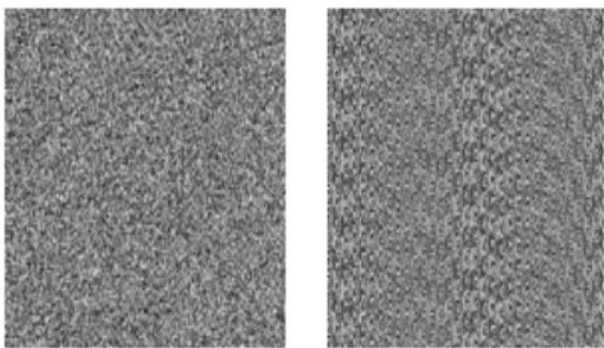
9. This effect is especially inconvenient in the case of a power  $m = 2^e$ : Then the least significant bit of  $X_n$  has a period of length at most 2, hence alternates between 0 and 1, or is constant. And the  $k$  least significant bits together have a period of at most  $2^k$ .

10. Thus a module  $m$  with many divisors, in particular a power of 2, has a serious handicap for random generation compared with a prime module. However the quality of the pseudo-random sequence is often sufficient for applications where the generated numbers are divided by  $m$ , that means, are used as pseudo-random reals in the interval  $[0, 1]$ , and are rounded by several places at the right end. But in a cryptographic setting such numbers are virtually useless.

LCG has a disadvantages of being ‘easily’ spotted periodic

'loop', when used in small numbers. To counter this it is recommended to choose M:

- $M = 2^{32}$  that exhausts the 32 bit range and moreover is computationally efficient,
- $M = 2^{31} - 1$  that is the maximum 32 bit integer, and computationally almost as efficient as a power of 2. Another advantage: This number is prime (claimed by Mersenne in 1644, proved by Euler in 1772), and this enhances the quality of the pseudo-random sequence. More generally these arguments apply to Fermat primes  $2^k + 1$  and Mersenne primes  $2^k - 1$ . The next prime of this kind is  $2^{61} - 1$ .



Picture 3. From K.M. Uma Maheswari et al. Truly random vs Pseudo random.

LCG example 1:

$$X_n = f(X_{n-1}, X_{n-2}, \dots)$$

$$X_n = 5X_{n-1} + 1 \pmod{16}$$

Starting with  $X_0=5$ :

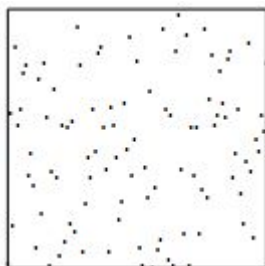
$$X_1 = 5(5) + 1 \pmod{16} = 26 \pmod{16} = 10$$

The first 32 numbers obtained by the above procedure 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.

In this example there is periodic looping after 16 numbers.

LCG example 2:

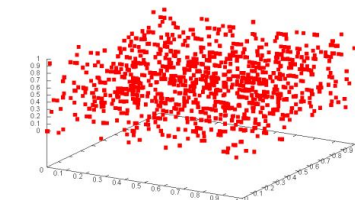
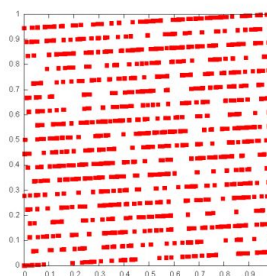
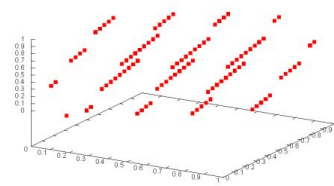
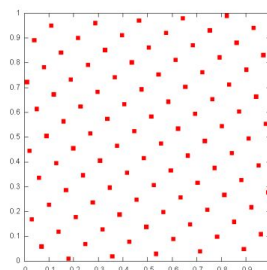
Table below shows the first 100 members of a sequence that is generated with the module  $M = 2^{31} - 1 = 2147483647$ , the multiplier  $A = 397204094$ , the increment  $B = 0$ , and the initial value  $X_0 = 58854338$ . The picture below illustrates the table. Here we cannot see any visible patterns because of the big numbers, not like previous example.



Picture 4. From K. Pommerening. X axis is 0 to 100. Y axis is 0 to  $2^{31} - 1$ .

1292048469	319941267	173739233	1992841820
345565651	2011011872	31344917	592918912
1827933824	1691830787	857231706	1416540893
1184833417	145217588	589958351	1776690121
1330128247	558009026	1479515830	1197548384
1627901332	929586843	19840670	1268974074
1682548197	760357405	666131673	1642023821
787305132	1314353697	167412640	1377012759
963849348	971229179	247170576	1250747100
703109068	1791051358	1978610456	1746992541
177131972	1844679385	1328403386	1811091691
1586500120	1175539757	74957396	753264023
468643347	821920620	1269873360	963348259
1698955999	139484430	30476960	1327705603
1266305157	1337811914	1808105128	640050202
37935526	1185470453	2111728842	380228478
808553600	934194915	824017077	881361640
1492263703	414709486	298916786	1883338449
771128019	558671080	1935988732	798347213
120356246	1378842534	37149011	272238278
1190345324	1006355270	1161592162	1079789655
220609946	1918105148	791775291	979447727
1160648370	779600833	1170336930	1271974642
375813045	1089009771	280197098	1144249742
1236647368	1729816359	650188387	1714906064

Table for example 2.



Picture 5. From Prof. Dr. Mesut Güneş Ch. 6 Random-Number Generation. Above is bad generator, below is the good one. Look at how random it is compared to the bad one.

#### IV. TRUE RANDOM NUMBER GENERATOR

Unlike the pseudo random one, true random generates its random number from something that is really hard to model with mathematics. For example, [www.random.org](http://www.random.org) generates random number from observing the noise from the atmosphere. There is many thing that can generate true random beside noise from the atmosphere, such as thermal noise, photoelectric effect, and other phenomenon involving quantum properties. So because we observe, we cannot create any algorithm that can generate true random generator. But we create a hardware that observe these phenomenons, the, gather data from them thus creating true random number.

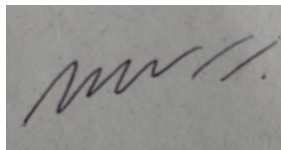
#### REFERENCES

- [1] Prof. Dr. Mesut Güneş , Ch. 6 Random-Number Generation.
- [2] Gentle, J. E. (1998, 2003). *Random Number Generation and Monte Carlo Methods*.Springer-Verlag: NY.
- [3] Pommerening, K., Bitstream Ciphers, Ch 1.3 Linear Congruential Generator.
- [4] Dutang, Cristophe. et al. "A note on random number generation," 2009.
- [5] Uma Maheswari, K. M.et al. "PSEUDO RANDOM NUMBER GENERATORS ALGORITHMS AND APPLICATION," *International Journal of Pure and Applied Mathematics*, Volume 118 No. 22 2018, 331-336.
- [6] <https://www.random.org/randomness>.

#### PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Desember 2019



Petrus E. Manurung 13518110