PSEUDO-RANDOM NUMBER GENERATOR

M19W0132 - Ryan Limanjaya

The Kyoto College of Graduate Studies for Informatics

2020

Pseudo-random generator (will refer as PRNG) is a deterministic method that used to generate truly random likewise sequence of number by give it a truly random number that come from the measurement of a noise (called as seed) and use it as the initial input to start the calculation. Nevertheless, there is limitation in Pseudo-random generator algorithm before it repeats its own sequence and it is called "the period". With middle-square method the period size determined by the length of the initial seed and it is roughly around 10^n with *n* was the initial seed length[9] [10]. The typical structure of PRNG will be a finite set S of states, function $f : S \rightarrow S$, output space of U, and output function $g : S \rightarrow U$. With the output space will always to be (0,1) and the generator given initial value of S_0 (the seed), then it will $S_n = f(S_{n-1}), N = 1,2,3, \dots, U_n = g(S_n)$ [11].

PRNG been applied in many ways such as Monte Carlo simulations, test pattern generation, cryptography, and telecommunication system. When implementing PRNG, a good implementation needs to meet certain criteria such as

- long period number of sequences: with only one seed it should capable to give relatively long period of sequence.
- pass empirical statistical test: after generating long sequence of random number, it should be pass various statistical test that the number are uniformly distributed, and independent.
- mathematical basis.
- fast without wasting memory: some case requires vast amount of random number therefore the generator should be fast and not greedy in resource.
- multiple streams / high throughput rate: some cases work efficiently when utilize parallel computation therefore generator must capable to work independent each other.
- Unpredictability: when applied for security matter good generator should have unpredictability therefore it will take longer for attacker to find the pattern.
- ease of installation and seeding: generator should be flexible and easy to deploy also it should have flexible seeder therefore the generator does not have constrain with one system only.
- reproducibility.

Despite of that in practical application PRNG usually have only one trait between resource efficient with unpredictability due to linear structure[11] [4].

Many algorithms used to apply PRNG in real life, here some example of them:

- Linear Congruential Generators: it works by repeatedly applying a given recurrence relation to the seed to create a sequence of number. The graph produced from this method usually look random except when the modulus is a multiple of the multiplier. In mathematical form the product can be interpreted as $X_1, X_2, X_3, \ldots$ between zero and *m-1* with recursive relationship of $X_{i+1} = (aX_i + c) mod\ m, i = 0,1,2, \ldots$ with X_0as the seed, a is constant multiplier, c is increment, and m is the modulus. When $c \neq 0$ the form is called *mixed congruential method* and when $c = 0$ it is called *multiplicative congruential method* [12] [13].
- Linear-feedback shift register (LFSR): this is a shift register where the input bit is the output of a linear function of two or more of its previous states[14]. It requires to have great feedback function and seed to generate sequence of bit that appear random. There is wide selection of how LFSR implemented and closely related method such as Mersenne Twister that implemented as default generator in python since version 2.3[7].
- Mersenne Twister first described by Makoto Matsumoto and Takuji Nishimura in 1998, the sequence was not crypto secure but it has excellent statistical properties and because it has long period suitable as a basic component for stream cipher. It engages on a seed with value 19337 bits long. This value stored in an array with 624 element that contain 32-bit values[15].
- "counter-based" random number generators (CBRNGs) one kind of PRNG that only use integer counter as internal state[8]. Said to be the fastest generator in the present time by the implementation of *Parallel Random Numbers: As Easy as 1, 2, 3*[5] and claimed to be improved the speed twice by *Squares: A Fast Counter-Based RNG* [6].

There are many implementations of PRNG in computer science field. It could be used for cryptographic key generator as it described and explained in this paper [1]. This document also state that PRNG could be implemented in processor for creating a stream on modern processor and how it improved by implementing the latest CBRNG in their product [16] [17]. By implementing PRNG into Monte Carlo simulation PRNG could be used for visualize risk of forecasting model. This article explain how a casino use Monte Carlo simulation to ensure they profit [2]. This paper described how PRNG used to help a warehouse pick the best order for they[3].

References:

[1] Elaine Barker. 2020. *Recommendation for Key Management: Part 1 – General*. National Institute of Standards and Technology. DOI:https://doi.org/10.6028/NIST.SP.800-57pt1r5

[2] Rohan Joseph. 2018. The house always wins : Monte Carlo Simulation. *Medium*. Retrieved August 14, 2020 from https://towardsdatascience.com/the-house-always-wins-monte-carlo-simulation-eb82787da2a3

[3] Mariusz Kostrzewski. 2019. Comparison of the order picking processes duration based on data obtained from the use of pseudorandom number generator. *Transportation Research Procedia* 40, (January 2019), 317–324. DOI:https://doi.org/10.1016/j.trpro.2019.07.047

[4] Chung-Yi Li, Yuan-Ho Chen, Tsin-Yuan Chang, Lih-Yuan Deng, and Kiwing To. 2012. Period Extension and Randomness Enhancement Using High-Throughput Reseeding-Mixing PRNG. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20, 2 (February 2012), 385–389. DOI:https://doi.org/10.1109/TVLSI.2010.2103332

[5] John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. 2011. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (SC '11), Association for Computing Machinery, New York, NY, USA, 1–12. DOI:https://doi.org/10.1145/2063384.2063405

[6] Bernard Widynski. 2020. Squares: A Fast Counter-Based RNG. *arXiv:2004.06278 [cs]* (May 2020). Retrieved August 15, 2020 from http://arxiv.org/abs/2004.06278

[7] 2020. List of random number generators. *Wikipedia*. Retrieved August 15, 2020 from https://en.wikipedia.org/w/index.php?title=List_of_random_number_generators&oldid=965642712

[8] 2020. Counter-based random number generator (CBRNG). *Wikipedia*. Retrieved August 15, 2020 from https://en.wikipedia.org/w/index.php?title=Counter-based_random_number_generator_(CBRNG)&oldid=972221547

[9] Pseudorandom number generators (video). *Khan Academy*. Retrieved August 14, 2020 from https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/random-vs-pseudorandom-number-generators

[10] Cryptography - Pseudo-Random Number Generators. Retrieved August 14, 2020 from https://crypto.stanford.edu/pbc/notes/crypto/prng.html

[11] book_chap3.pdf. Retrieved August 14, 2020 from https://www.math.arizona.edu/~tgk/mc/book_chap3.pdf

[12] Linear Congruential Method. Retrieved August 15, 2020 from https://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node40.html

[13] Wolfram Demonstrations Project brings ideas to life with over 11k interactive #WolframNotebooks for education, research, recreation & more. #WolframDemo #WolfLang. Retrieved August 15, 2020 from /LinearCongruentialGenerators/

[14] Linear Feedback Shift Register - an overview | ScienceDirect Topics. Retrieved August 15, 2020 from https://www.sciencedirect.com/topics/mathematics/linear-feedback-shift-register

[15] The Mersenne Twister. Retrieved August 15, 2020 from http://www.quadibloc.com/crypto/co4814.htm

[16] Device API Overview. Retrieved August 15, 2020 from http://docs.nvidia.com/cuda/curand/index.html

[17] New counter-based Random Number Generators in Intel® Math Kernel... *Intel*. Retrieved August 15, 2020 from https://www.intel.com/content/www/us/en/develop/articles/new-counter-based-random-number-generators-in-intel-math-kernel-library.html