

**Figure 1 :** *For each generated pseudorandom number by Yarrow's algorithm a bitmap is created to visualize its randomness*

# THE RANDOM OF RANDOMNESS

APR 2018

*A detained enquiry about the different methodology used by true random number generators (TRNGs) and pseudo-random number generators (PRNGs) as well as the mathematical model behind them.*

# TABLE OF CONTENTS

## Contents

Introduction	1
Pseudo Random Number Generator	4
Different Types of PRNGs	6
Linear Congruential Generator (LCG)	6
Yarrow algorithm (CSPRNG)	10
True Random Number Generators	15
TRNG: which physical phenomenon is best suited to be the input to produce TRN: a quantum phenomenon or a phenomenon with a chaotic behavior?	17
Sources	19
Appendix A- table list the parameter of LCGs that are commonly used, including the built-in rand() function in many runtime libraries of various compilers	20
Appendix B- LCG Implementation in C	22
Appendix C: Yarrow's PRNG entropy collection from mouse click in C	23
Contact Information	24

# Introduction

Randomness is defined as the lack of pattern or predictability in events. Random number can be defined to be a number generated for, or part of, a set that exhibits statistical randomness. Random number can be discussed related to a sequence of numbers. Referring in terms of sequence of random numbers, each number drawn must be statistically independent of the others and the pool of numbers from which the sequence is chosen from must be uniformly distributed. Uniform distribution guarantees that each number is equally likely to be picked and there is no way to predict the random number. Statistically independent means that selection of one number from the number pool does not affect the chances of selection of another number.

Random numbers are extremely important or used in many critical purposes such as, simulating and modeling complex phenomena, generating data encryption keys and for selecting random samples from larger data sets. Therefore, generating strong random numbers are of paramount importance. The meaning of strong in respective of random number means that can pass the stringent test of randomness from certain randomness test suites, e.g. TestU01 or NIST Random Number Test Suite. TestU01 is a library, implements in C, that offers a collection of utilities for the empirical randomness testing of random number generators<sup>1</sup>.

---

<sup>1</sup> Pierre L'Ecuyer & Richard Simard (2007), "TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators", ACM Transactions on Mathematical Software, 33: 22.

The process of generating random number can be extremely difficult as from common knowledge, we know that computers follow a strict algorithm and one cannot make a computer do anything by chance. If it seems that a computer is doing something by chance then that is a design flaw or fault algorithm in use and, after debugging the issue the results become completely predictable. After years of research two main categories of random number generator are developed: Pseudo-Random Number Generators (PRNGs) and True Random Number Generators (TRNGs).

Randoms numbers play a huge part in cryptography. Random numbers are used in public key generation, session keys, initialization vectors, and many other places. If the random number is not strong, then the entire application and the protocol is insecure.

Characteristic	Pseudo-Random Number Generators	True Random Number Generators
Efficiency	Excellent	Poor
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic
Application		Most Suitable Generator
Lotteries and Draws		TRNG
Games and Gambling		TRNG
Random Sampling (e.g., drug screening)		TRNG
Simulation and Modelling		PRNG
Security (e.g., generation of data encryption keys)		TRNG

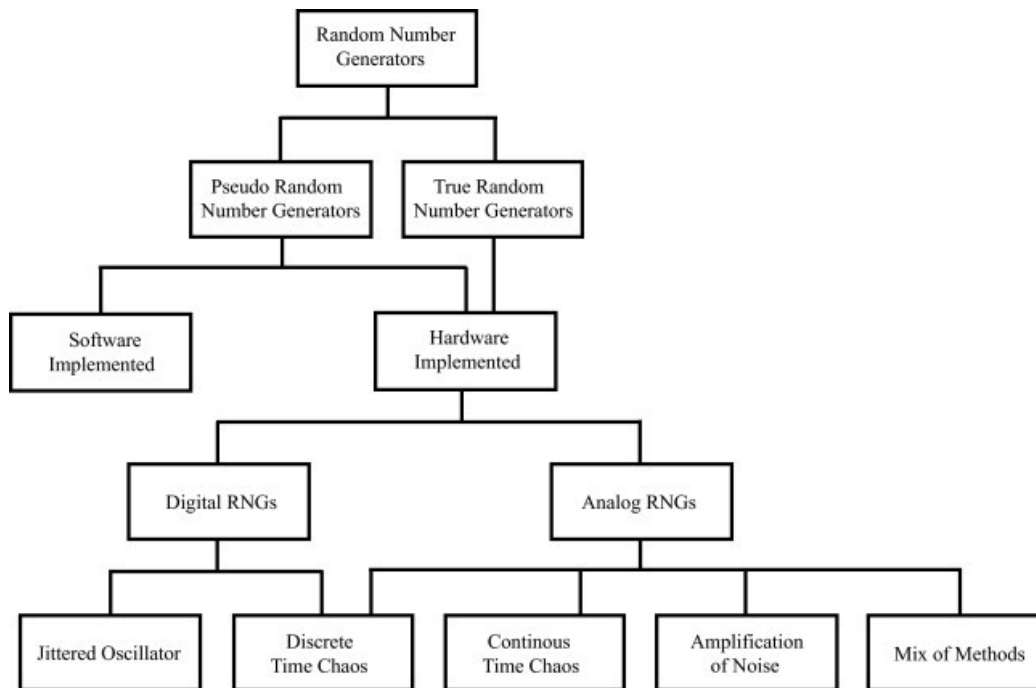
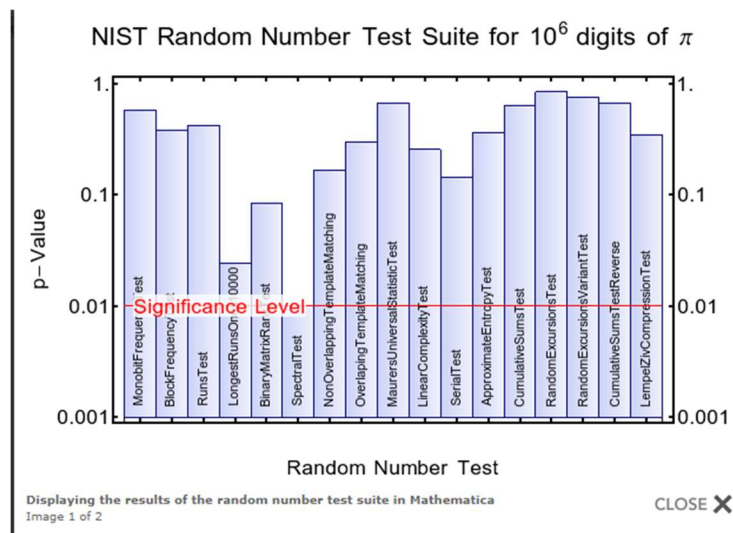


Figure 2: Classification of random number generators.



## Pseudo Random Number Generator

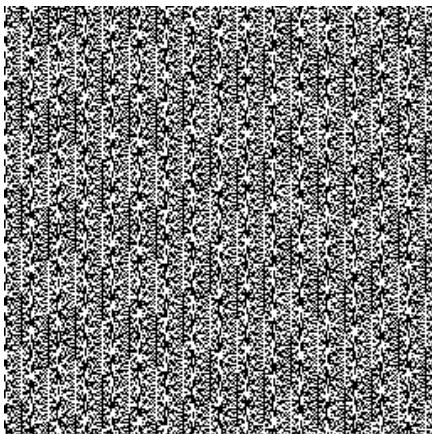
Pseudo random number, as the name prefix 'pseudo' suggest, are not truly random, but rather PRNG (Pseudo random number generators) are algorithms that use mathematical formulae or pre-calculated table to produce a sequence of numbers that appear to be random. The difference between PRNGs and TRNGs is that when a person is using a PRNG he is getting a number from an existing list which was compiled beforehand by using a mathematical formula, whereas person is using TRNG is actually getting the computer to compute the number at runtime by taking a certain physical phenomenon as an input.

PRNGs have certain characteristics: they are efficient, periodic and deterministic. Efficient means that a PRNG can produce many outputs in a relatively short burst of time, periodic means that the number sequence will ultimately repeat itself and deterministic means, the output can be recreated in the future if the input is known. Efficiency is a need characteristic if the target applications needs a lot of data but want to spend shorter amount of time in generating them or the user does not care about randomness of every output of data. Determinism is helpful when the user wants to get back the output sequence in a much later date for rechecking and retesting purposes. Common applications showing these needs are simulations and modelling applications. PRNGs are absolutely not suitable for applications where it is important to have numbers which are completely unpredictable or truly random, e.g. data encryption and gambling. Certain good PRNGs exist like the linear congruential method, permuted congruential generator, inverse congruential generators and Yarrow

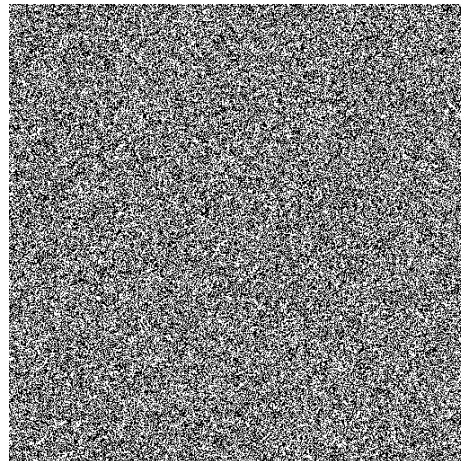
algorithm which can pass the most stringent randomness test. Yarrow's algorithm is unique as it belongs to the family of CSPRNG or Cryptographically Secure Pseudo-Random Number Generator and is almost as random as a TRNG.

PRNG behavior changes from one OS to another. One of the most famous example, is the PRNG in the computer language, PHP. The PRNG for GNU/Linux can produce certain strong random numbers but for Microsoft Windows it does an extremely bad job. This can be easily visualized by looking at the bitmap generated by PHP for Windows and Linux side by side, e.g. Figure 3. As evident from the figure, the RN generated by rand() in PHP is extremely predictable.

Two good PRNG that I will do a further case study on is linear congruential method<sup>2</sup> and Yarrow algorithm.



**Figure 3** *Bitmap generated with PHP rand() on Microsoft Windows*



**Figure 4** *Bitmap generated with PHP rand() on Linux*

---

<sup>2</sup> "Linear Congruential Generators" by Joe Bolte, Wolfram Demonstrations Project.

## Different Types of PRNGs

### LINEAR CONGRUENTIAL GENERATOR (LCG)

Linear Congruential generator (LCG) is one of the oldest and best-known pseudo number generator algorithm. The random number are calculated using a piecewise linear equation.

The recurrence relation to define the generator is:

$$X_{n+1} = (aX_n + c) \bmod m$$

$X$  is the sequence of pseudo random number

$m$ ,  $0 < m$  is the modulus

$a$ ,  $0 < a < m$  is the multiplier

$c$ ,  $0 \leq c < m$  is the increment

$X_0$ ,  $0 \leq X_0 < m$  is the seed or the start value.

The mathematical advantage for LCG is that with the correct selection of parameters ( $m$ ,  $a$ ,  $c$ ), the period can be long and unknown and vice-versa, with incorrect selection of parameter, the period would be too short. The most sensitive parameters are modulus,  $m$  and multiplier,  $a$ . Classic example of a widely used PRNG where incorrect selection of parameter



resulted in generation of RN which seemed strong but later when further analysis was done it failed the spectral test miserably which is discussed in the footnote 3, is IBM's RANDU<sup>3</sup>.

The parameter section broadly falls in three categories:

- $c = 0$  and  $m$  is prime. This is a special recurrence equation known as Lehmer<sup>4</sup> RNG formulation. The  $a$  is chosen to be a primitive root of the integer modulo  $m$  and the range of values or the period it can take ranges from 1 to  $m-1$ . One restriction is that  $X_0$  has to be chosen between 1 and  $m-1$ , because we are working in integer modulo  $m$  ring. The two disadvantages are that it needs explicit reduction step and it is hard to convert  $1 \leq X < m$  to uniform random bits.
- $c = 0$  and  $m, a$  are power of 2. This method produces efficient LCH with long periods, but it has unequal period distribution for the all bits. The lower bits have shorter period distribution, than the higher bits. Only the significant bit of  $X$  reaches full period. For example, if  $a \equiv 5 \pmod{8}$ , then Bit 3 repeats with a period of 4, bit 4 has a period of 8, and so on.
- $c \neq 0$  and  $m, c$  are relatively prime ( $\text{g.c.d}(m, c) = 1$ ),  $a - 1$  are divisible by all prime factors of  $m$  and  $a - 1$  is divisible by 4 if  $m$  is divisible by 4<sup>5</sup>. These

---

<sup>3</sup> If a plot is made of 100,000 values generated by RANDU, it can be clearly seen that the values falls in between 15 two-dimensional planes. RANDU's parameters were  $a = 65539$ ,  $c = 0$ , and  $m = 2^{31}$ .

<sup>4</sup> W.H. Payne; J.R. Rabung; T.P. Bogyo (1969). "Coding the Lehmer pseudo-random number generator" (PDF). Communications of the ACM. 12 (2): 85–86. doi:10.1145/362848.362860.

<sup>5</sup> Knuth, Donald (1997). Seminumerical Algorithms. The Art of Computer Programming. 2 (3rd ed.). Reading, MA: Addison-Wesley Professional.

specific parameter conditions are referred to as Hull-Dobell Theorem<sup>6,7</sup>. The period is equal to  $m$  and the generator is not sensitive to  $c$  as long as it is relatively prime to  $m$ .  $C = 1$  is commonly chosen as often  $m$  is a power of 2.

Appendix A shows a table list the parameter of LCGs that are commonly used, including the built-in `rand()` function in many runtime libraries of various compilers.

The advantages of LCGs are that they are extremely fast and require relative less memory compared to another algorithm. These make then a good option when multiple independent streams are needed.

The disadvantage of LCGs are:

1. LCG have a small state because of the small modulus  $m$  chose. But, it is proved that it passed stringent BigCrush randomness suite if the modulus is larger than 96-bits<sup>8</sup>.
2. LCG's random number outputted lie on at most,  $(n! * m)^{1/n}$  hyperplanes<sup>9</sup>. A hyperplane is a subspace whose dimension is one less than that of its ambient space. In this context, a space is thought to be 3-dimensional then its


---

<sup>6</sup> Hull, T. E.; Dobell, A. R. (1962-01-01). "Random Number Generators" (PDF). *SIAM Review*. 4 (3): 230–254. doi:10.1137/1004061. Retrieved 2016-06-26.

<sup>7</sup> Severance, Frank (2001). *System Modeling and Simulation*. John Wiley & Sons, Ltd. p. 86. ISBN 0-471-49694-4.

<sup>8</sup> O'Neill, Melissa E. (5 September 2014). *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation* (PDF) (Technical report). Harvey Mudd College. pp. 6–7. HMC-CS-2014-0905.

<sup>9</sup> Marsaglia, George (September 1968). "Random Numbers Fall Mainly in the Planes" (PDF). *PNAS*. 61 (1): 25–28. Bibcode:1968PNAS...61...25M. doi:10.1073/pnas.61.1.25.



hyperplanes are then 2-dimensional planes, This can related back to serial correlation between successive values of the sequence of  $X_n$ . This is one of the reasons why randomness suite have a spectral test component to it, which measures the point spacing in the 2D planes.

3. If  $m$  is a power of 2, then the lower order bits have shorter period. This characteristic makes the user think critically for the parameter of LCGs when a high-quality randomness is needed. Monte Carlo Simulations require a high order modulus, that's why the modulus has to be greater than the cube root of the number of random samples are required. For, example 32-bit LCG can produce thousand random values and 64-bit two million random values.

LCGs are not recommend for cryptographic applications, whereas Yarrow's algorithm is a proven to be one of the best CSPRNG. LCGs' has most applications in embedded systems and gaming console, where there is a restriction in resources.

Appendix B provides an implementation of LCG in C.

## YARROW ALGORITHM (CSPRNG)

Yarrow is created by John Kelsey, Bruce Schneier and Niels Ferguson. Yarrow algorithm is the ancestor of Fortuna<sup>10</sup>, which is a relatively new SCPRNG. Yarrow is unpatented, copy-right free and free software available to the end user.. The yarrow can be used as a TRNG when it is accepting random input from analog sources. The code that is used to gain the analog random input from mouse click is shown in Appendix C. The TRNG is discussed in detail in later sections.

There are three main design principles that the creator of yarrow based the implementation upon were:


- Easy to use by programmer who has no cryptography background
- Resistance to attack
- Reusability of existing building blocks
- Keeping the information secure even when the key is compromised
- Enough entropy that can be used to keep the PRNG random and generate random sequence of numbers

The three components of Yarrow are:

1. Entropy accumulator – Entropy means the lack of orderliness. The entropy accumulator collects the disorderliness or the random events that happened in the

---

<sup>10</sup> An analysis of Fortuna—and a proposed improvement—can be found in Y. Dodis, A. Shamir, N. Stephens-Davidowitz, D. Wichs, How to Eat Your Entropy and Have it Too—Optimal Recovery Strategies for Compromised RNGs, Cryptology ePrint Archive, Report 2014/167, 2014.



computer, usually from the human interacting with the computer. Yarrow accumulates entropy in two pools, the fast pool and the slow pool. The fast pool provides frequent reseeds of the keys to keep the period of the key in circulation as small as possible. The slow pool makes sure that the reseed values are secure when the entropy is maximum.

2. Reseed mechanism and generating mechanism- Reseeding mechanism is the bridge between the entropy accumulator and the generating mechanism. The reseeding mechanism updates the key constantly, so that if the entropy pool is compromised to the attacker then the duration of window for him to guess the final output of the reseeding mechanism is minimal. The current key and the hash of all inputs since start up is used to reseed the fast pool to generate the new key. The new key from the slow pool is generated in the similar fashion except the hash of all input to the slow pool is used. Both reseeding resets the entropy estimation of the fast pool to zero, but the last reseeding also sets the entropy estimation of the slow pool to zero, so that the new keys must come from a freshly generated entropy. It is a precaution taken to remove all the possible old key values so handle cryptanalytic attacks.
3. Reseed Control Component – This control is used to prevent iterative guessing attacks and protection against divulging too much information to the attacker who has got the hashed key due to infrequent seeding. This is done by reseed the fast pool whenever it passes a threshold value and it uses the slow pool to reseed itself.

Yarrow-160 is the latest Yarrow version succeeded by Fortuna. It uses two special algorithms to keep the pool of entropy and the reseeding secure: one-way hash function

and block cipher. Yarrow-160 uses SHA1 Hash function and Three-key triple DES respectively.

The functions to generate Yarrow:

$$C \leftarrow (C + 1) \bmod 2^n$$


$$R \leftarrow E_k(C)$$

$$K \leftarrow \text{Next } k \text{ bits of PRNG output}$$

Yarrow-160 uses three-key triple-DES in counter mode to generate outputs.  $C$  is an  $n$ -bit counter value;  $K$  is the key. Yarrow makes sure to keep track of the output, so that if the key is compromised then the leak of old output could be stopped immediately. Once some system security parameter  $P_g$  is reached, the algorithm will generate  $k$  bits of PRNG output and use them as the new key. The security parameter is set to 10, means  $P_g = 10$ , it is kept small intentionally so that there is minimal number of outputs can be backtracked. The reseeding key is hashed using the SHA1 and reseeded using the triple-DES (3DES) symmetric key block cipher.

Advantages of Yarrow:

1. Yarrow is efficient CSPRNG
2. Yarrow can be used by programmer with no cryptographic background
3. Entropy estimation of Yarrow is conservative, stopping exhaustive search attacks
4. To block cryptanalytic attack, Yarrow is designed on block cipher which are itself secure against these attacks

- 
5. The attacker is stopped from manipulating the input samples, as it uses cryptographic hash functions to process input samples and then uses a secure update function to combine the samples with the existing key.  
  
Iterative guessing attack is prevented as the only time the key is reseeded by the pool when it is content that the entropy pool is completely unpredictable.

#### Disadvantages of Yarrow

1. The output is as secure as the generating mechanism, if the generating mechanism is compromised then so is the output.
2. The procedure to determine when the pool has enough entropy or randomness that it can be used for reseed purposes<sup>11</sup> is an extremely difficult process to implement which and it is called entropy estimation.
3. The strength of yarrow is directly proportional to the size of the key. Yarrow-160 has the key size of 160 bits, if the security is required of 512 bits, yarrow-160 cannot do it.
4. SHA1 has been a topic of controversy as SHA1 has been broken and is not considered as secure as it used to be<sup>12</sup>.

---

<sup>11</sup> "Fortuna – A Cryptographically Secure Pseudo Random Number Generator – CodeProject". Retrieved 18 October 2016.

<sup>12</sup> Stevens, Marc; Bursztein, Elie; Karpman, Pierre; Albertini, Ange; Markov, Yarik (2017-02-23). "SHattered". SHattered. Retrieved 2017-04-27.

These pitfalls led to the creation of Fortuna, which is considered to be one of the best cryptographically secure true random number generators<sup>13</sup>.

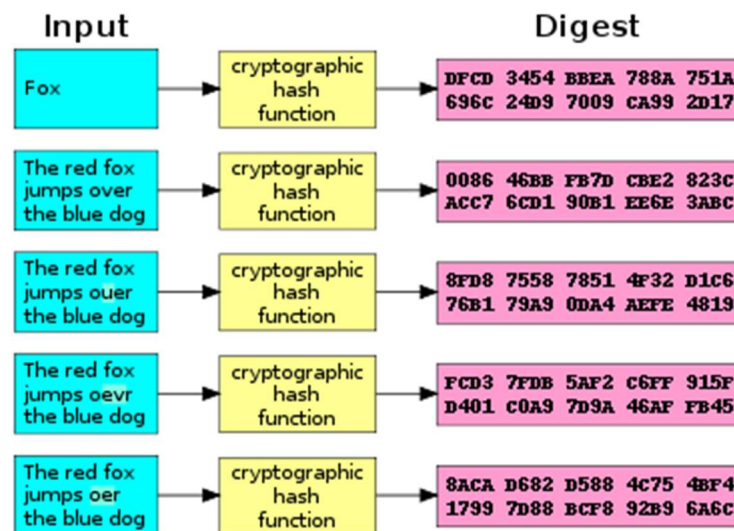


Figure 5:SHA-1 at work

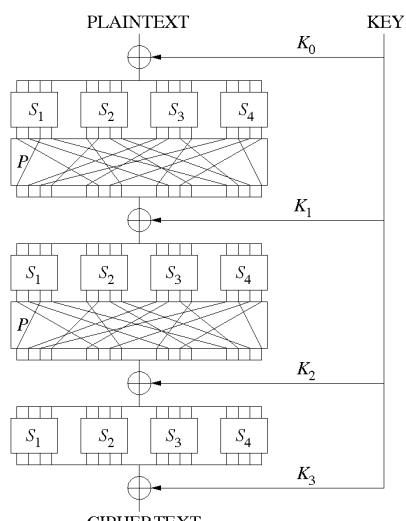


Figure 6A sketch of a substitution-permutation network with 3 rounds, encrypting a plaintext block of 16 bits into a ciphertext block of 16 bits

<sup>13</sup> "Questions & Answers about Yarrow". Schneier on Security. Retrieved 2016-02-15



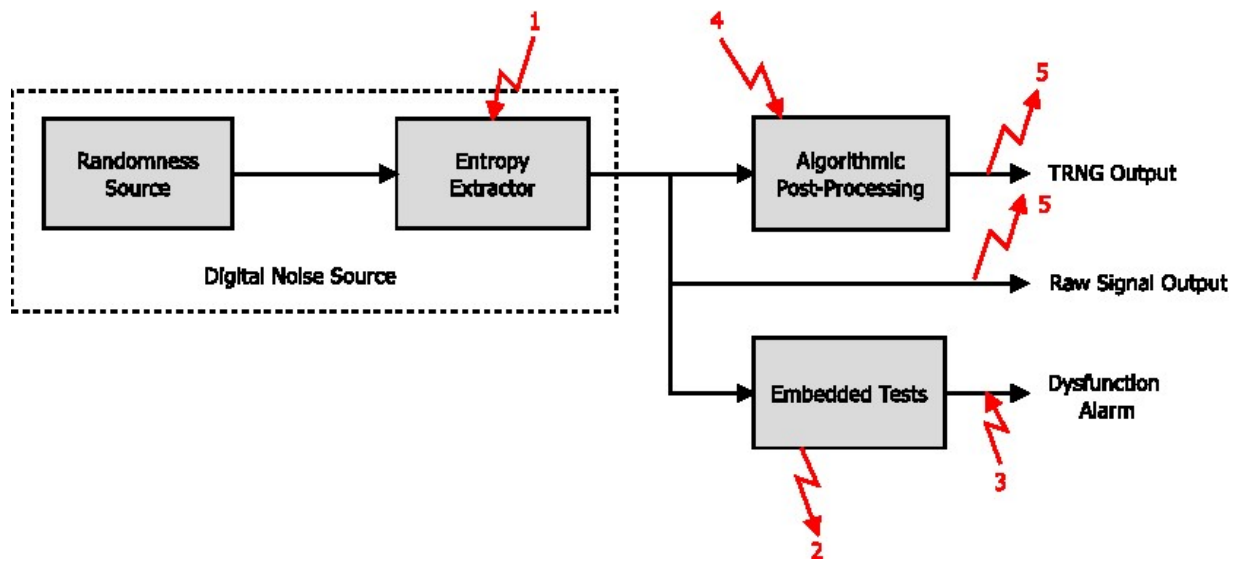
## True Random Number Generators

True Random Number generators instead of using a single mathematical formula to precompute a list, it tries to extract the randomness from a physical environment as the source of input for PRNG. For example, the seed to the TRNG can be the mouse movements, the amount of time between keystrokes from a person, or use a mic to pick up the atmospheric noise around the computer. But, certain precautions should be taken, for example, if a fan is rotating near the mic, it will pick the fan's noise, and since the fan is a rotating device, the probabilities of it producing random noise is not as likely it will just picking up periodic noise. The use of time interval between keystrokes as a random event can be an erroneous idea as many a times, the keystrokes are buffered by the OS, and the TRNG will see that the keystrokes are passes simultaneous, and there may not be any randomness at all.

The two most well developed way to generate true random number is collecting little random variations in data from the received input from radioactive decay source and/or atmospheric noise. Radioactive decay source has the advantage over keystroke recording algorithm is that the point of time at which a radioactive material will decay is completely random, and it can be easily detected and fed into the computer and most importantly it will be avoiding any buffering mechanism. One of the most prominent research facility that use this technique is HotBits service at FourmiLab in Switzerland. The technique of using atmospheric noise as random input is done by a website called random.org. The procedure to generate true

random number is similar in the sense that the input banks on the little, unpredictable changes in the input data that is usually fed into a PSRG at runtime to yield the random number.

The characteristics of TRNGs are vastly different than PRNGs. TRNG are non-deterministic, aperiodic, and inefficient. Non-deterministic means that the output sequence cannot be reproduced in the future, inefficient means that it takes longer duration of time to generate the number and aperiodic means that the sequence of number do not repeat themselves.



**Figure 7:** (1,3,4) determines the random number generators and ( 2,5) are the output of the TRNG

## TRNG: which physical phenomenon is best suited to be the input to produce TRN: a quantum phenomenon or a phenomenon with a chaotic behavior?

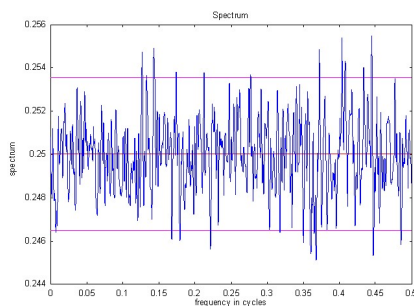
The builders of a TRNG usually give a philosophical thought to the question about which physical phenomenon is best suited to be the input to produce TRN: a quantum phenomenon or a phenomenon with a chaotic behavior? This is an important discussing factor regarding the accurate use of the physical phenomenon of TRNG. The event is evaluated regarding the extent of determinism found in it. Determinism is a view that proposes that everything that happens is predetermined since the big bang. The TRNG that uses atomic decay to generate RN is using a quantum phenomenon, whereas TRNG that uses the atmospheric noise to generate a RN is using phenomenon with a chaotic behavior.

Quantum mechanics is a branch of theoretical physics that describes the universe mathematically at the atomic and sub-atomic level. The TRNG uses the fact that subatomic particles behave randomly in specific conditions and they are inherently non-deterministic. Whereas, chaotic system relies on the fact that minute changes to the input data will cause drastic changes to the overall behavior of the overall result. The butterfly effect is a hypothetical experiment using this rationale, that a butterfly flapping its wings here will cause tornado thousands of miles apart.

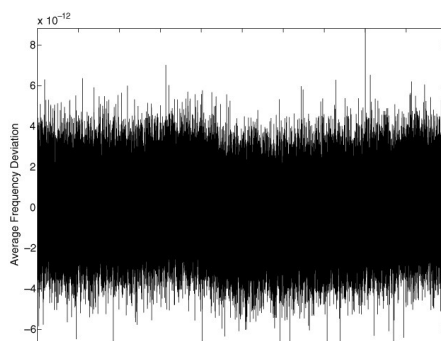
Determinist believe that the event resulting from a quantum event is better suited for TRNG as quantum physics is inherently non-deterministic in sub-atomic level. Whereas,

quantum phenomenon with chaotic behavior e.g. the TRNG using an atmospheric noise as a physical phenomenon is a deterministic event as hypothetically, one with the knowledge of position and velocity of all the molecules in the atmosphere can accurately predict the noise. But, realistically to predict the event one needs to have enormous computing power and gathering the required information is a herculean or impossible task by itself. Hypothetically, quantum generators are not non-deterministic either as one might argue that soon we will discover equations which would accurately model the behavior of the sub-atomic particles, thus, removing their randomness characteristic. Hence, a hard determinist views quantum event and event with chaotic behavior similar in regard to their ability to produce random number.

There are two likely succeeding scenarios that we have discovered equation that predict all quantum events or that we have developed a means to accurately extrapolate all the noise made in the atmosphere. Ultimately, then the choice of event boils down to which hypothetical situations mentioned above is likely to become a practical situation in the duration of time when the specific TRNG will be in use.



**Figure 8:** Atmospheric noise gathered using a radio



**Figure 9** Atomic noise created by a carbon-14 isotope

## Sources

Hull, T. E.; Dobell, A. R. (1962-01-01). "Random Number Generators" (PDF). *SIAM Review*. 4 (3): 230–254. doi:10.1137/1004061. Retrieved 2016-06-26.

Knuth, Donald (1997). *Seminumerical Algorithms. The Art of Computer Programming*. 2 (3rd ed.). Reading, MA: Addison-Wesley Professional.

Marsaglia, George (September 1968). "Random Numbers Fall Mainly in the Planes" (PDF). *PNAS*. 61 (1): 25–28. Bibcode:1968PNAS...61...25M. doi:10.1073/pnas.61.1.25.

O'Neill, Melissa E. (5 September 2014). *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation* (PDF) (Technical report). Harvey Mudd College. pp. 6–7. HMC-CS-2014-0905.

Pierre L'Ecuyer & Richard Simard (2007), "TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators", *ACM Transactions on Mathematical Software*, 33: 22.

Severance, Frank (2001). *System Modeling and Simulation*. John Wiley & Sons, Ltd. p. 86. ISBN 0-471-49694-4.

W.H. Payne; J.R. Rabung; T.P. Bogoyo (1969). "Coding the Lehmer pseudo-random number generator" (PDF). *Communications of the ACM*. 12 (2): 85–86. doi:10.1145/362848.362860.

Y. Dodis, et.al (Feb 2014).” An analysis of Fortuna—and a proposed improvement—can be found in *How to Eat Your Entropy and Have it Too—Optimal Recovery Strategies for Compromised RNGs*”.ePrint Archive.

Appendix A- table list the parameter of LCGs that are commonly used, including the built-in `rand()` function in many runtime libraries of various compilers

Source	modulus <i>m</i>	multiplier <i>a</i>	increment <i>c</i>	output bits of seed in <i>rand()</i> or <i>Random</i> ( <i>L</i> )
Borland C/C++	$2^{32}$	22695477	1	bits 30..16 in <i>rand()</i> , 30..0 in <i>lrand()</i>
glibc (used by GCC)	$2^{31}$	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ C99, C11: Suggestion in the ISO/IEC 9899	$2^{31}$	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	$2^{32}$	134775813	1	bits 63..32 of ( <i>seed</i> * <i>L</i> )
Turbo Pascal	$2^{32}$	134775813 (0x8088405 <sub>16</sub> )	1	
Microsoft Visual/Quick C/C++	$2^{32}$	214013 (343FD <sub>16</sub> )	2531011 (269EC3 <sub>16</sub> )	bits 30..16

RtlUniform from Native API	$2^{31} - 1$	2147483629 (7FFFFFFED <sub>16</sub> )	2147483587 (7FFFFFFC3 <sub>16</sub> )	
C++11's <code>minstd_rand</code>	$2^{31} - 1$	48271	0	
MMIX by Donald Knuth	$2^{64}$	6364136223846793005	1442695040888963407	
Java's java.util.Random, POSIX [ln]rand48, glibc[ln]rand48[_r]	$2^{48}$	25214903917 (5DEECE66D <sub>16</sub> )	11	bits 47...16
<code>random0</code>  If $X_n$ is even then $X_{n+1}$ will be odd, and vice versa—the lowest bit oscillates at each step.	$134456 = 2^{37.5}$	8121	28411	
POSIX [jm]rand48, glibc [mj]rand48[_r]	$2^{48}$	25214903917 (5DEECE66D <sub>16</sub> )	11	bits 47...15
POSIX [de]rand48, glibc[de]rand48[_r]	$2^{48}$	25214903917 (5DEECE66D <sub>16</sub> )	11	bits 47...0
<i>Formerly common:</i> RANDU	$2^{31}$	65539	0	

## Appendix B- LCG Implementation in C

```
1. #include <stdio.h>
2.
3. /* always assuming int is at least 32 bits */
4. int rand();
5. int rseed = 0;
6.
7. inline void srand(int x) {
8.     rseed = x;
9. }
10.
11. #ifndef MS_RAND
12. #define RAND_MAX ((1U << 31) - 1)
13.
14. inline int rand() {
15.     return rseed = (rseed * 1103515245 + 12345) & RAND_MAX;
16. }
17.
18. #else /* MS rand */
19.
20. #define RAND_MAX_32 ((1U << 31) - 1)
21. #define RAND_MAX ((1U << 15) - 1)
22.
23. inline int rand()
24. {
25.     return (rseed = (rseed * 214013 + 2531011) & RAND_MAX_32) >> 16;
26. }
27.
28. #endif /* MS_RAND */
29.
30. int main() {
31.     int i;
32.     printf("rand max is %d\n", RAND_MAX);
33.
34.     for (i = 0; i < 10; i++)
35.         printf("%d\n", rand());
36.
37.     return 0;
38. }
```



## Appendix C: Yarrow's PRNG entropy collection from mouse click in C

```
LRESULT CALLBACK MouseHook(int nCode, WORD wParam, LONG lParam)
{
    static DWORD then = 0;
    static DWORD now = 0;
    WORD diff;
    LARGE_INTEGER time;
    DWORD waitret;
    static LONG running = FALSE;

    /*Do not process message if nCode <0 or ==HC_NOREMOVE */
    if ( (nCode < 0) || (nCode == HC_NOREMOVE) )
        return CallNextHookEx(ghhookMS, nCode,wParam, lParam);

    /* Make sure that this thread is not already running this function */
    if(InterlockedExchange(&running,TRUE) == TRUE)
        return CallNextHookEx(ghhookKB, nCode,wParam, lParam); /* It was already locked */

    switch(wParam) /*Message type*/
    {
        case WM_LBUTTONDOWN:          /*Get timing from button clicks...*/
        case WM_RBUTTONDOWN:
        case WM_MBUTTONDOWN:
        case WM_NCLBUTTONDOWN:
        case WM_NCRBUTTONDOWN:
        case WM_NCMBUTTONDOWN:
            /*Start mutex*/
            waitret = WaitForSingleObject(writeMutex,MOUSETIMEWAIT);
            if(waitret != WAIT_OBJECT_0)
            {
                /* Could not capture the mutex for some reason, so abandon hook procedure */
                InterlockedExchange(&running,FALSE);
            }
    }
}
```

## Contact Information

KUNAL MUKHERJEE  
COMPUTER ENGINEER



Tel 812-550-3890  
Km411@evansville.edu

**DILBERT** By SCOTT ADAMS

