International Journal of Hybrid Information Technology
Vol. 9, No.7 (2016), pp. 299-312
http://dx.doi.org/10.14257/ijhit.2016.9.7.27

The Hybrid Pseudo Random Number Generator

Tanvir Ahmed and Md. Mahbubur Rahman

Department of Computer Science & Engineering
Bangladesh University of Business & Technology (BUBT)
Dhaka, Bangladesh
tanvircse10@gmail.com, mahabub.cse.buet@gmail.com

Abstract

Random number generation is an art and science of deterministically generating a Sequence of numbers that is difficult to distinguish from a true random sequence. It is become important tool in cryptography to generate secrete key. There are so many approaches to generate a pseudo random number. But they have some limitations too. In this paper we represent a new random number generation algorithm with the help of existing conventional methods & some new tricks to enhance the programs randomness. We already know some methods like mid-square or congruent. But, in our random numbers, we just use a congruential method which is pretty much different with conventional methods. That's why we can consider it as a hybrid method of pseudo random number generator.

***Keywords**: Cryptography, PRNG, NIST, DIEHARD, TRNG, Monte Carlo Method, Kolmogorov Test, Chi-square Test, Poker Test*

1. Introduction

Random numbers are essential elements in a great number of solutions in computer science. Randomized algorithms require a random source to ensure computational complexity bounds and sampling methods often require randomness to accurately represent the terms they are surveying. However, one of the most important uses of random numbers originated from cryptographic security protocols and algorithms. Cryptographic applications use random numbers to generate public/private keys, create initial parameter values, and to generate random numbers into protocols and padding schemes. Basically, the term random number originated from the Pseudorandom Number Generator (PRNG) which is a deterministic software algorithm which *create* randomness. A PRNG takes an input string of bits, or a bit-vector, and generates a longer output bit-vector which "appears" random. Although there are so many definitions of what it means for a bit-vector to "appear" random, one basic rule needs that a pseudorandom number is indistinguishable from a "true" random number. That is, given a true random number and a pseudorandom number, it is totally computationally uncontrolled to execute an algorithm that could recognize the pseudorandom numbers. It is strongly related to a basic criteria which requires that every "next" output bit from the generated random number must be unpredictable or not correlated to all prior bits.

Today there are so many pseudorandom number generators algorithm found, but not all of them satisfy the basic requirement and are therefore not cryptographically sound. For example, the built in C rand() function would not create "good" cryptographic keys. It can be shown, that any candidate from *one-way function* can be used directly as pseudorandom number generator that is cryptographically sound. Therefore, we commonly used as a PRNG. Unfortunately, PRNGs suffer from two major security disadvantages. Firstly, PRNGs demand some inputs which deterministically governs the output. To securely use a PRNG, this input also called "seed" must be kept secret.

Secondly, All PRNGs can only generate a fixed number bits before they cycle and repeat themselves. On the other hand Hardware random number generators (HRNG) do not suffer from these two issues since they generate aperiodic “random” output without input any seed value. To avoid this, they generate output bits by exploiting inherent unpredictability in complex physical systems and procedures. That’s why, the security of any HRNG is directly tied to the probability of modeling and imitating the underlying physical system. Although hardware generators are quite difficult to design, and even more difficult to validate, a number of candidate HRNG designs have been developed [3, 6, 8, 12].

1.1. Formal Definition

The random variable $X:\Omega \to E$ is a moderate function from the set of possible outcomes Ω to some set E. The technical axiomatic definition refers Ω to be a probability space and E to be a measurable space (see Measure-theoretic definition) [15].

Although X is basically a real-valued function ($E = \mathbb{R}$), it does not return a probability. The probabilities of different outcomes or a set of outcomes (events) are already given by the probability measure P with which Ω is equipped. Rather, X denotes some numerical property that outcomes in Ω may have. *e.g.* the number of heads in a random collection of coin flips, the height of a random person *etc*. The probability that X takes value ≤ 3 is the measure of the set of outcomes $\{\omega \in \Omega : X(\omega) \leq 3\}$, denoted $P(X \leq 3)$ [14].

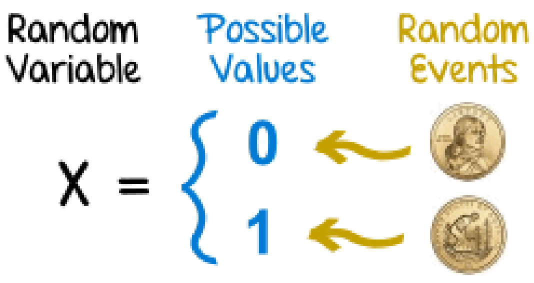


Figure 1. Random Variable

A new random variable Y can be defined by applying a real Borel measurable function $g:\mathbb{R} \to \mathbb{R}$ to the outcomes of a real-valued random variable X. The cumulative distribution function of Y is $F_Y(y) = \mathrm{P}(g(X) \leq y).$

If function g is invertible, *i.e.* g^{-1} exists, and is either increasing or decreasing, then the previous relation can be extended to obtain

$$F_Y(y) = \mathrm{P}(g(X) \leq y) = \begin{cases} \mathrm{P}(X \leq g^{-1}(y)) = F_X(g^{-1}(y)), & \text{if } g^{-1} \text{ increasing,} \\ \\ \mathrm{P}(X \geq g^{-1}(y)) = 1 - F_X(g^{-1}(y)), & \text{if } g^{-1} \text{ decreasing.} \end{cases}$$

and, again with the same hypotheses of inevitability of g, assuming also differentiability, we can find the relation between the probability density functions by differentiating both sides with respect to y, in order to obtain

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{dg^{-1}(y)}{dy} \right|.$$

If there is no invertibility of g but each y admits at most a countable number of roots (*i.e.* a finite, or countably infinite, number of x_i such that $y = g(x_i)$) then the previous relation between the probability density functions can be generalized with

$$f_Y(y) = \sum_i f_X(g_i^{-1}(y)) \left| \frac{dg_i^{-1}(y)}{dy} \right|$$

Where, $x_i = g_i^{-1}(y)$. The formulas for densities do not demand g to be increasing.

So, a random variable's possible values might represent the possible outcomes of a yet-to-be-performed experiment, or the possible outcomes of a past experiment whose already-existing value is uncertain (for example, due to imprecise measurements or quantum uncertainty). They may also conceptually represent either the results of an "objectively" random process (such as rolling a die) or the "subjective" randomness that results from incomplete knowledge of a quantity. The meaning of the probabilities assigned to the potential values of a random variable is not part of probability theory itself but is instead related to philosophical arguments over the interpretation of probability. The mathematics works the same regardless of the particular interpretation in use [13].

1.2. Properties of a Pseudo Random Number

A sequence of random numbers has two important statistical properties.

1. Uniformity and,
2. Independence.

Each random number is an independent sample drawn from a continuous uniform distribution between an interval 0 and 1. Probability density function of this distribution is given by,

$$f(x) = \begin{cases} 1, & \text{for } 0 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

Reader can see that equation (4.1) is same as equation (2.8) with *a* = 0 and *b* = 1. The expected value of each random number *Ri* whose distribution is given by (4.1) is given by,

$$E(R) = \int_0^1 x\,dx = \left[\frac{x^2}{2}\right]_0^1 = \frac{1}{2}$$

and variance is given by,

$$\mathrm{Var}(R) = \int_0^1 x^2 dx - [E(R)]^2 = \left[\frac{x^3}{3}\right]_0^1 - \left[\frac{1}{2}\right]^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$$

Test for the Uniformity of Random Numbers: If the interval between 0 and 1 is divided into *n* equal intervals and total of *m* (where *m* > *n*) random numbers are generated between 0 and 1 then the test for uniformity is that in each of *n* intervals, approximately (*m/n*) random numbers will fall.

1.3. Area of Applications

Random numbers are used in various sections, some of are given below:

1. Cryptography
2. Random sampling
3. Monte Carlo Method
4. Statistics
5. Simulation & Modeling
6. Lottery
7. Random distribution

1.4. Different Types of Generator

Here we concerned with the generation of random numbers (uniform) using digital computers. In chapter three, we have given one of the method of generating random numbers, which is dice rolling. With this method we can generate random numbers between two and twelve. Some other conventional methods are coin flipping, card shuffling and roulette wheel. But these are very slow ways of generating random numbers. We can generate thousands of random numbers using computers in no time. Random number hereafter will also be called random variant.

Congruential or Residual Generators

One of the common methods, used for generating the pseudo uniform random numbers is the congruence relationship given by,

$$X_{i+1} \equiv (aX_i + c)(\mathrm{mod}\ m), \quad i = 1,2,\ldots,n$$

Where, multiplier *a*, the increment *c* and modulus *m* are non-negative integers. It means, if (*aXi* + *c*) is divided by *m*, then the remainder is *i* 1 *X* + . In this equation *m* is a large number such that $m \leq 2w - 1$, where *w* is the word length of the computer in use for generating the (*m*– 1) numbers and (*i* = 0) is seed value. By seed value, we mean any initial value used for generating a set of random numbers. Seed value should be different for different set of random numbers.

In order, the numbers falling between 0 and 1, we must divide all *Xis*' by (*m*– 1). To illustrate the equation, let us take, *a* = 3, *X*0 = 5, *c* = 3 and *m* = 7. Then

(*i*) $X1 \equiv (3 \times 5 + 3)\ (\mathrm{mod}\ 7) = 4$
(*ii*) $X2 \equiv (3 \times 4 + 3)\ (\mathrm{mod}\ 7) = 1$
(*iii*) $X3 \equiv (3 \times 1 + 3)\ (\mathrm{mod}\ 7) = 6$
(*iv*) $X4 \equiv (3 \times 6 + 3)\ (\mathrm{mod}\ 7) = 0$
(*v*) $X5 \equiv (3 \times 0 + 3)\ (\mathrm{mod}\ 7) = 3$
(*vi*) $X6 \equiv (3 \times 3 + 3)\ (\mathrm{mod}\ 7) = 5$
(*vii*) $X7 \equiv (3 \times 5 + 3)\ (\mathrm{mod}\ 7) = 4$

Thus we see that numbers generated are, 4, 1, 6, 0, 3, 5, 4. Thus there are only six non repeating numbers for *m* = 7. Larger is *m*, more are the non-repeated numbers. Thus period of these set of numbers is *m*. There is a possibility that these numbers may repeat before the period *m* is achieved. Let in the above example *m* = 9, then we see that number generated are 0, 3, 3, 3, 3,… This means after second number it starts repeating. It has been shown [53] that in order to have non-repeated period *m*, following conditions are to be satisfied,

(*i*) *c* is relatively prime to *m*, *i.e.*, *c* and *m* have no common divisor.
(*ii*) $a \equiv 1\ (\mathrm{mod}\ g)$ for every prime factor *g* of *m*.
(*iii*) $a \equiv 1\ (\mathrm{mod}\ 4)$ if *m* is a multiple of 4.

Condition (*i*) is obvious whereas condition (*ii*) means $a = g\{a/g\} + 1$, where number inside the bracket { } is integer value of *a*/*g*. Let *g* be the prime factor of *m*; then if $\{a/g\} = k$, then we can write

$a = 1 + gk$

Condition (*iii*) means that

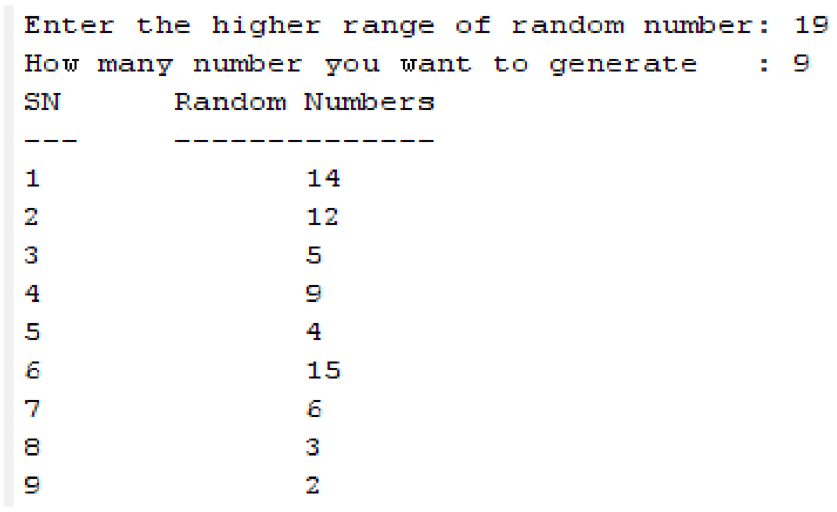
$a = 1 + 4\{a/4\}$

if *m*/4 is an integer. Based on these conditions we observe that in the above example *m* = 9 had a common factor with *a*, thus it did not give full period of numbers. Name pseudo random numbers, is given to these random numbers. Literal meaning of pseudo is false. They are called pseudo because to generate them, some known arithmetic operation is used, which can generate non-recurring numbers but they may not be truly uniformly random.

The MATLAB code is given below:

```
m= input('Enter the higher range of random number: ');
lmt= input('How many number you want to generate   : ');
a=13;b=1;r=1;

disp('SN      Random Numbers ')
disp('---     -------------- ')

for i=1:lmt

    n=(a*r)+b;
    x=mod(n,m);
    fprintf('%d              %d\n',i,x);
    r=x;

end
```

Output of Congruence random number:



```
Enter the higher range of random number: 19
How many number you want to generate    : 9
SN       Random Numbers
---      --------------
1                14
2                12
3                5
4                9
5                4
6                15
7                6
8                3
9                2
```

Mid Square Random Number Generator

This is one of the earliest method for generating the random numbers. This was used in 1950s, when the principle use of simulation was in designing thermonuclear weapons. Method is as follows:

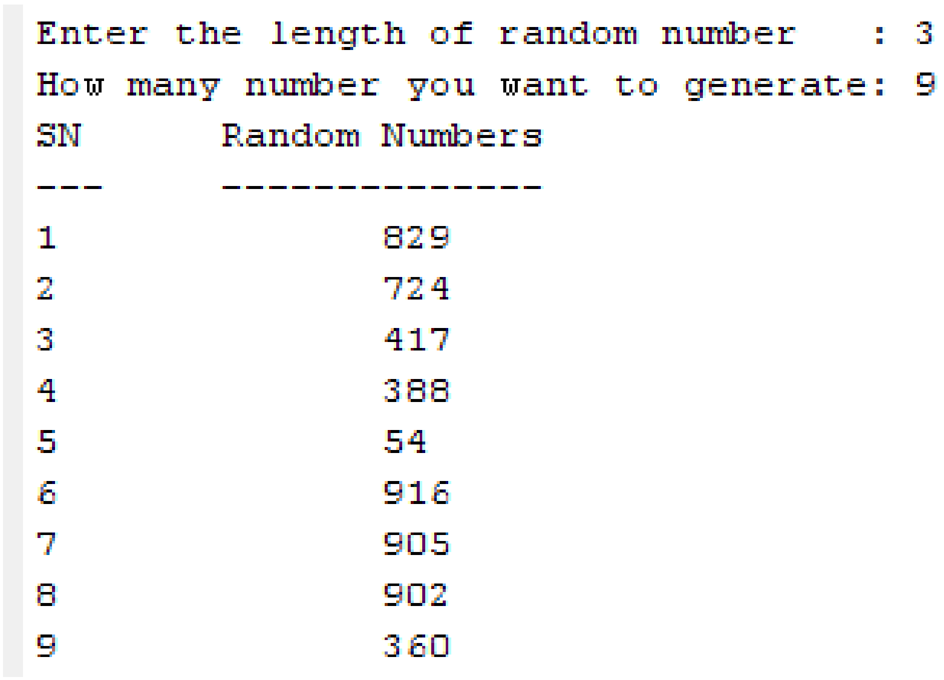
1. Take some *n* digit number.
2. Square the number and select *n* digit number from the middle of the square number.
3. Square again this number and repeat the process.

For example Let us assume a three digit seed value as 123.
Step 1: Square of 123 is 15129. We select mid three numbers which is 512.
Step 2: Square of 512 is 262144. We select mid two numbers which is 21.
Step 3: Repeat the process. Thus random numbers are 512, 21, …

The MATLAB code is given below:

```
n= input('Enter the length random number      : ');
lmt= input('How many number you want to generate: ');
seed=5673;
disp('SN      Random Numbers ')
disp('---     -------------- ')

for i=1:lmt

temp=num2str(seed^2);
    s=size(temp);
    st=(s(1,2)/2);
    x=str2num(temp(st:(st+n-1)));
    fprintf('%d              %d\n',i,x);
    seed=x;

end
```

Output of Mid-square random number:



```
Enter the length of random number    : 3
How many number you want to generate: 9
SN      Random Numbers
---     --------------
1               829
2               724
3               417
4               388
5               54
6               916
7               905
8               902
9               360
```

2. Motivation

There are so many random number generation models that are used in real-world. But each of them have some limitations as:

1. Mid-Square random number generator is useless if any output consist lots of zeros
2. Congruence random number generator have redundancy problem

According to these reason we are motivated to construct a new generation algorithm that show the maximum optimized result.

3. Proposed Model

3.1. Idea

For every random number generation algorithm we see that a seed value is given at the initialization of the program. If the seed value is same each time and we use an arithmetic operations to generate a number, then it show the same sequence of random numbers each trial. To optimize it we use system time stamps millisecond part as a seed value. The millisecond is rapidly change within a very short time. So, using this trick; same sequence have not returned anymore. On the other hand we use a multiplicative congruence method for arithmetic operation. After each trail we update the input value for the next iteration. For the better concept of our method we can consider the following image:

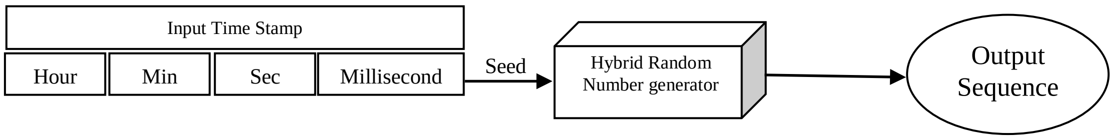


Figure 2. Mechanism of our Model

Well, this is our proposed model. It is very much similar to conventional model, but the difference is in input seed & the arithmetic formula of generator.

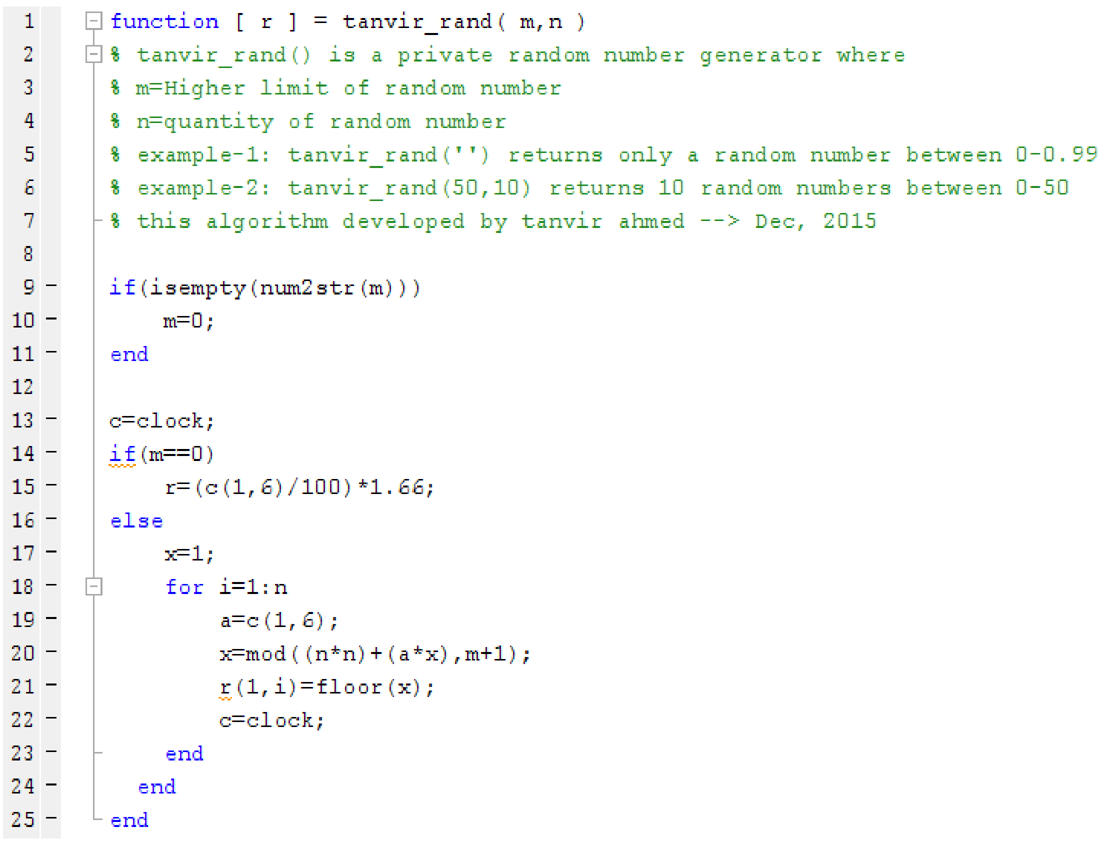
3.2. Algorithm

For implementing this model we need to follow the following algorithm:

```
1. Input m & n from user where m=maximum limit and n=quantity of random number
2. Take timestamp & clip microsecond part's 4 digits into c which considered as seed value
3. Set x=1
4. for i=1 to n
5. x=(n^2 + ax) mod (m+1)
6. r(1,i)=floor(x)
7. Update c
8. Loop end
9. Return r
```

3.3. Code implementation

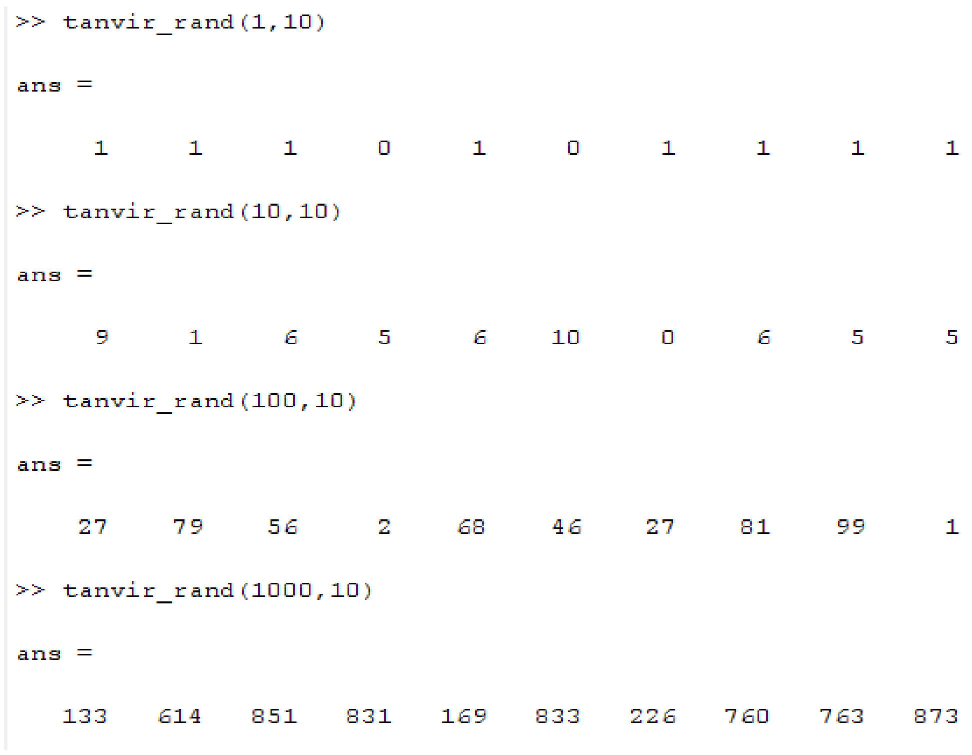
The MATLAB code for the random number generation function is given below:



```
function [ r ] = tanvir_rand( m,n )
% tanvir_rand() is a private random number generator where
% m=Higher limit of random number
% n=quantity of random number
% example-1: tanvir_rand('') returns only a random number between 0-0.99
% example-2: tanvir_rand(50,10) returns 10 random numbers between 0-50
% this algorithm developed by tanvir ahmed --> Dec, 2015

if(isempty(num2str(m)))
    m=0;
end

c=clock;
if(m==0)
    r=(c(1,6)/100)*1.66;
else
    x=1;
    for i=1:n
        a=c(1,6);
        x=mod((n*n)+(a*x),m+1);
        r(1,i)=floor(x);
        c=clock;
    end
  end
end
```

3.4. Output

Here some output pattern for different input parameters are given below:

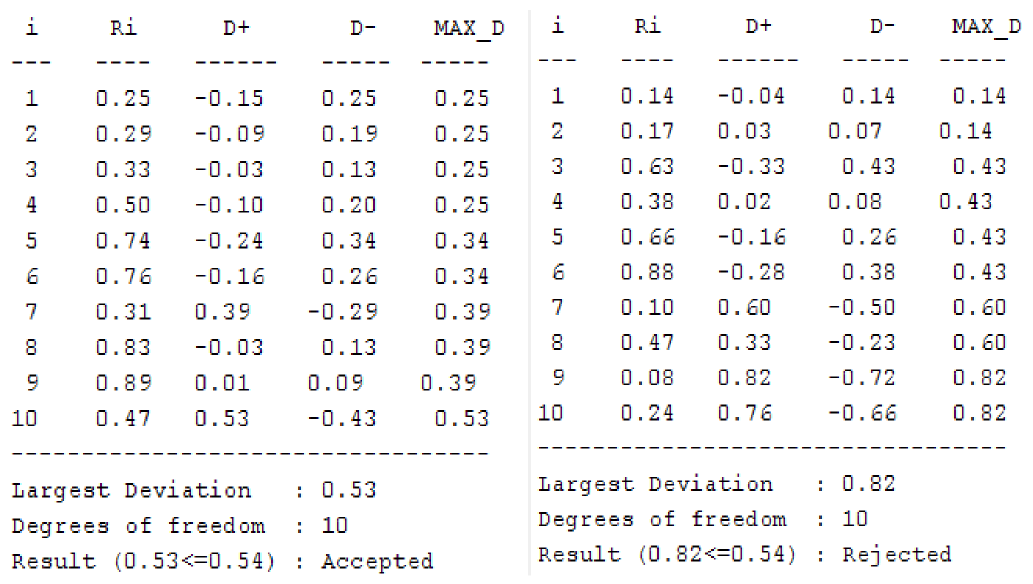


```
>> tanvir_rand(1,10)

ans =

     1     1     1     0     1     0     1     1     1     1

>> tanvir_rand(10,10)

ans =

     9     1     6     5     6    10     0     6     5     5

>> tanvir_rand(100,10)

ans =

    27    79    56     2    68    46    27    81    99     1

>> tanvir_rand(1000,10)

ans =

   133   614   851   831   169   833   226   760   763   873
```

4. Simulation & Randomness Analysis

For checking randomness of a sequence of random numbers, there we use various methods of technique to determine the acceptance of random number. Some are given below for our model:

Kolmogorov Test:



```
 i     Ri      D+       D-     MAX_D
---   ----   ------   -----   -----

 1    0.25   -0.15    0.25    0.25
 2    0.29   -0.09    0.19    0.25
 3    0.33   -0.03    0.13    0.25
 4    0.50   -0.10    0.20    0.25
 5    0.74   -0.24    0.34    0.34
 6    0.76   -0.16    0.26    0.34
 7    0.31   0.39    -0.29    0.39
 8    0.83   -0.03    0.13    0.39
 9    0.89   0.01    0.09    0.39
10    0.47   0.53    -0.43    0.53
----------------------------------

Largest Deviation   : 0.53
Degrees of freedom  : 10
Result (0.53<=0.54) : Accepted
```

```
 i     Ri      D+       D-     MAX_D
---   ----   ------   -----   -----

 1    0.14   -0.04    0.14    0.14
 2    0.17   0.03    0.07    0.14
 3    0.63   -0.33    0.43    0.43
 4    0.38   0.02    0.08    0.43
 5    0.66   -0.16    0.26    0.43
 6    0.88   -0.28    0.38    0.43
 7    0.10   0.60    -0.50    0.60
 8    0.47   0.33    -0.23    0.60
 9    0.08   0.82    -0.72    0.82
10    0.24   0.76    -0.66    0.82
----------------------------------

Largest Deviation   : 0.82
Degrees of freedom  : 10
Result (0.82<=0.54) : Rejected
```

Figure 3. Kolmogorov Test at Confidence Level 0.1

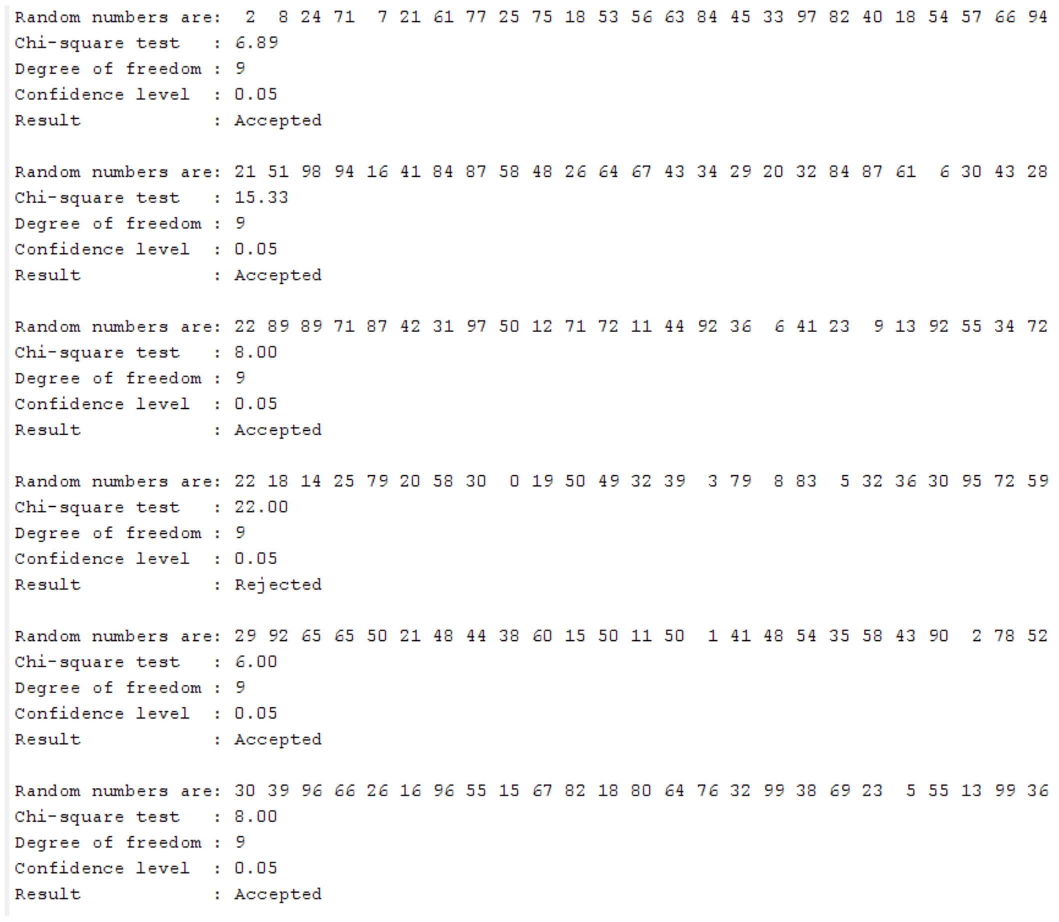
Comparative Kolmogorov Test:

Here we apply Kolmogorov test for several times with different generator, that are currently presented in the world. Then make average of them and compare it with our model. The demographics of comparative Kolmogorov test is given below:

Table 2. Comparative Kolmogorov Test

SN	Generator	Degree of Freedom	Accuracy Rate	Average Kolmogorov test value	Result
01	C	10	95%	0.91	Rejected
02	C++	10	95%	0.73	Rejected
03	java	10	95%	0.53	Accepted
04	php	10	95%	0.67	Rejected
05	MATLAB	10	95%	0.71	Rejected
06	Our Model	10	95%	0.51	Accepted

Chi-square Test:



```
Random numbers are:  2  8 24 71  7 21 61 77 25 75 18 53 56 63 84 45 33 97 82 40 18 54 57 66 94
Chi-square test   : 6.89
Degree of freedom : 9
Confidence level  : 0.05
Result            : Accepted

Random numbers are: 21 51 98 94 16 41 84 87 58 48 26 64 67 43 34 29 20 32 84 87 61  6 30 43 28
Chi-square test   : 15.33
Degree of freedom : 9
Confidence level  : 0.05
Result            : Accepted

Random numbers are: 22 89 89 71 87 42 31 97 50 12 71 72 11 44 92 36  6 41 23  9 13 92 55 34 72
Chi-square test   : 8.00
Degree of freedom : 9
Confidence level  : 0.05
Result            : Accepted

Random numbers are: 22 18 14 25 79 20 58 30  0 19 50 49 32 39  3 79  8 83  5 32 36 30 95 72 59
Chi-square test   : 22.00
Degree of freedom : 9
Confidence level  : 0.05
Result            : Rejected

Random numbers are: 29 92 65 65 50 21 48 44 38 60 15 50 11 50  1 41 48 54 35 58 43 90  2 78 52
Chi-square test   : 6.00
Degree of freedom : 9
Confidence level  : 0.05
Result            : Accepted

Random numbers are: 30 39 96 66 26 16 96 55 15 67 82 18 80 64 76 32 99 38 69 23  5 55 13 99 36
Chi-square test   : 8.00
Degree of freedom : 9
Confidence level  : 0.05
Result            : Accepted
```

Figure 4. Chi-Square Test at Confidence Level 0.1

Comparative Chi-square Test:

Here we apply Chi-square test for several times with different generator, that are currently presented in the world. Then make average of them and compare it with our model. The demographics of comparative Chi-square test is given below:

Table 2. Comparative Chi-square Test

SN	Generator	Degree of Freedom	Accuracy Rate	Average Chi-square test value	Result
01	C	10	95%	18.98	Rejected
02	C++	10	95%	16.39	Rejected
03	java	10	95%	14.25	Accepted
04	php	10	95%	15.80	Accepted
05	MATLAB	10	95%	13.87	Accepted
06	Our Model	10	95%	11.20	Accepted

5. Outputs & Results

We apply various types of randomness test on our model using MATLAB. After complete them, we investigate the actual view of our model.

5.1. Demographics of Randomness Test

Here we generate 30 number between 0-100 using different random number generators and apply following randomness test strategy. The Demographics of those samples are depicted by the following table:

Table 3. Demographics of All Randomness Test

SN	Generator	Degree of Freedom	Accuracy Rate	Acceptance Test Average Value			Result
				Kolmogorov	Chi-Square	Poker	
01	C	10	95%	0.91	18.98	4.75	Reject
02	C++	10	95%	0.73	16.39	1.29	Accept
03	java	10	95%	0.53	14.25	7.48	Reject
04	php	10	95%	0.67	15.80	8.34	Reject
05	MATLAB	10	95%	0.71	13.87	4.68	Accept
06	Our Model	10	95%	0.51	11.20	0.75	Accept

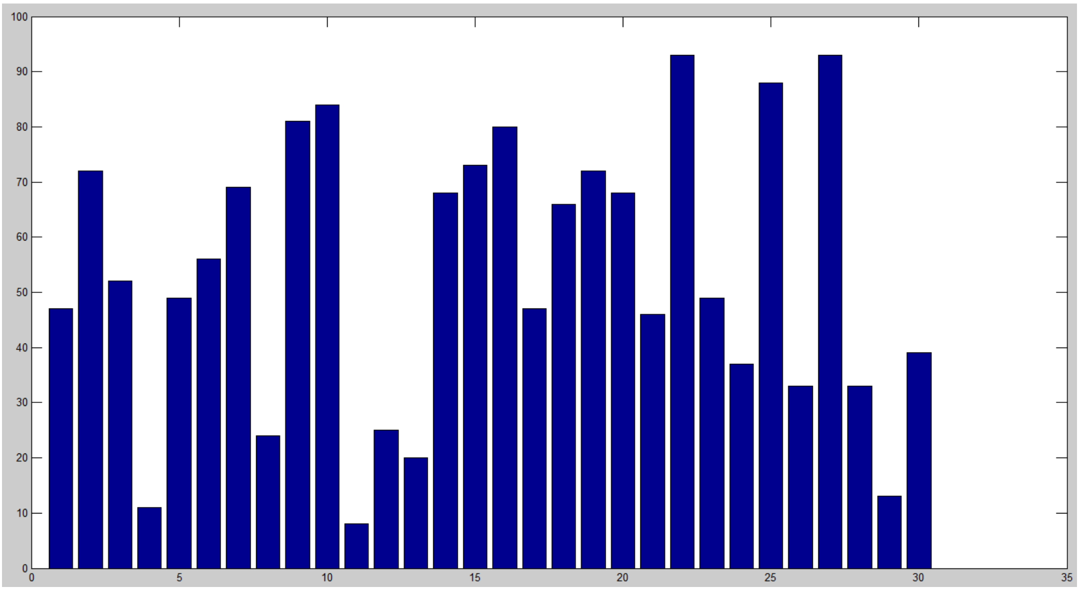


Figure 5. Bar-Chart for Generated Random Numbers of Our Model

6. Advantages & Disadvantage

Every system or model has some good or bad sides. Here we consider both portions. After observing everything of our proposed model following advantages are found:

1. No need to fix any seed value, here seed is collected from current microsecond which is rapidly changed after each iteration.
2. No need to scaling it, because a higher limit is predetermined.
3. It has high randomness & don't make any cycle.
4. It is used to any kind of applications.

On the other hand this model has some limitations. The flipping points are:

1. Its randomness depends on current time **c** & previous value of **x**

2. If the value of higher limit **m** is less than **10** then increasing the repetition of same number.
3. If **m** is greater than **2^{64}-1** then it cannot calculate the random number.

7. Conclusions

Random numbers are prevalent in many computer science applications. These applications include network protocols design (*e.g.*, TCP sequence number), algorithmic research (*e.g.*, random algorithms), various unique identifiers (*e.g.*, UUID) and security protocols (*e.g.*, TLS). Random numbers are considered a basic building block in almost every cryptographic scheme.

Having a right source of random numbers is a complex assumption of many protocol systems. There have been several high profile failures in random numbers generators that led to severe practical problems.

For this wide application of random numbers we just trying to find a robust & efficient algorithm to make it more effective in the relevant domain. In this connection, we also enjoy to work with such kind of challenging research. We think it is truly successful.

References

[1] R. W. Baldwin, “Preliminary analysis of the BSAFE 3.x pseudorandom number generators”, Technical Report 8, RSA Laboratories, **(1998)**.

[2] R. Davies, “Hardware random number generators”, In 15th Australian Statistics Conference, **(2000)**.

[3] R. B. P. Dept., “The Evaluation of Randomness of RPG100 by Using NIST and DIEHARD Tests”, Technical report, FDK Corporation, **(2003)**.

[4] D. Eastlake and S. Crocker, “RFC 1750: Randomness recommendations for security”, **(1994)**.

[5] D. Eastlake and P. Jones, RFC 3174: US secure hashing algorithm 1 (SHA1), **(2001)**.

[6] B. Jun and P. Kocher, “The Intel Random Number Generator”, Cryptography Research Inc. white paper, **(1999)**.

[7] S. Kim, K. Umeno and A. Hasegawa, “On the NIST Statistical Test Suite for Randomness”, In IEICE Technical Report, vol. 103, no. 449, **(2003)**, pp. 21-27.

[8] P. Kohlbrenner and K. Gaj, “An embedded true random number generator for fpgas”, In FPGA ’04: Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, pages 71–78. ACM Press, **(2004)**.

[9] J.-W. Lee, D. Lim, B. Gassend, E. G. Suh, M. van Dijk and S. Devadas, “A Technique to Build a Secret Key in Integrated Circuits with Identification and Authentication Applications”, In Proceedings of the IEEE VLSI Circuits Symposium, **(2004)**.

[10] D. Lim, “Extracting Secret Keys from Integrated Circuits”, Master’s thesis, Massachusetts Institute of Technology, **(2004)**.

[11] NIST Special Publication 800-22. A statistical test suite for random and pseudorandom number generators for cryptographic applications, Information Technology Laboratory of the National Institute of Standards and Technology, **(2000)**.

[12] C. Petrie and J. Connelly, “A Noise-based IC Random Number Generator for Applications in Cryptography”, IEEE TCAS II, vol. 46, no. 1, **(2000)**, pp. 56–62.

[13] W. Schindler and W. Killmann, “Evaluation criteria for true (physical) random number generators used in cryptographic applications”, In CHES ’02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems,. Springer-Verlag, **(2003)**, pp. 431–449.

[14] Securitytechnet random number generator references, http://www.securitytechnet.com/crypto/algorithm/random.html.

[15] G. E. Suh, C. W. O’Donnell, I. Sachdev and S. Devadas, “Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions”, Technical report, MIT CSAIL CSG Technical Memo 483, **(2004)**.

Authors

Tanvir Ahmed, he is a popular web application developer in Bangladesh. He takes B.Sc. Engg. in CSE from Bangladesh University of Business & Technology (BUBT), one of the topmost private university of Bangladesh. He is a talented computer scientist. Now he worked with a software company as a senior web developer and IT Officer. His research interests are Internet Security, Data mining, Web database, Artificial Intelligence & Image processing.

Md. Mahbubur Rahman, he has been lecturing in CSE since mid of 2011, he received his B.Sc.Engg. in CSE from Patuakhali Science and Technology University in 2011 and M.Sc. Engg. in CSE at Bangladesh University of Engineering and Technology(BUET), Bangladesh. He is now serving one of the top most private Universities in Bangladesh named Bangladesh University of Business and Technology (BUBT). His research interests are Digital Forensics, Secure and Trustworthy Computing, Data mining, Graph theory, Neural Network.