Journal of Modern Mathematics and Statistics 5 (4): 84-88, 2011
ISSN: 1994-5388

Introduction to Secure PRNGs

Majid Babaei
Department of Computer Engineering,
Shahrood University of Technology, Shahrood, Iran

Abstract: Pseudo-Random Number Generators (PRNGs) are required for generating secret key in cryptographic algorithm, generating sequence of packet in the network simulation (workload generators) and the other applications in the various fields. In this study, there will be discussed a list of some requirements for generating reliable random sequence and then will be presented some PRNGs method which are based on combinational chaotic logistic map. In the final study after a brief introduction of two statistical test packets, TestU01 and NIST suite tests, the PRNG methods which are presented in this study will be appraised under these test packets and the results will be reported.

Key words: Cryptography, chaotic random bit generator, NIST test suite, TestU01, PRNGs, Iran

INTRODUCTION

Pseudo-random Number Generators (PRNGs) are useful in many applications such as the crypto-system in network communication also one of the important methods in simulation is Mont Carlo which needs to generate very high quality pseudo-random sequence and calculation of numerical integration which is not solved by regular methods (Jun and Kocher, 1999; Wang *et al*., 2006; Ergun and Ozoguz, 2007).

These days with the mention of values security threats (Jun and Kocher, 1999; Wang *et al*., 2006; Ergun and Ozoguz, 2007; Hu *et al*., 2009; Patidar *et al*., 2009; Fechner and Osterloh, 2010; Shparlinski, 2004; Wang, 2000; Kumar *et al*., 2010; Rahimov *et al*., 2010; Babaei, 2011; Menezes *et al*., 1997; Fechner and Osterloh, 2010), the statistical quality of PRNGs is becoming more important than in the past. For example, the super computer might be generate 10^9 random numbers per sec and the cryptography algorithm needs 10^{16} random numbers to create a secure channel in the very important communication, thus small correlation or other weaknesses in the generated sequence could easily lead to the critical leak in the several network layers.

The distributions should be prepared based on their commercial applications such as normal, exponential, poisson and so on. But in this study, researchers consider only the generation of uniformly distributed numbers. In more detailed will be focused on the real number sequence which is uniformly distributed on the interval (0, 1).

The basic point in generating pseudo-random number is that these generators are deterministic because the digital computers are not able to generate truly random numbers. So, the statistical test needs to be presented and the PRNGs should pass the number of important statistical tests before being release for security usage in the communication networks.

Overview: Generation random numbers by combinational chaotic maps is one of the best methods to improve statistical properties. For example, Wang *et al*. (2006) proposed a pseudorandom number generator based on a z-logistic map. Ergun and Ozoguz (2007) proposed that the novel random bit sequence of a non-autonomous chaotic electronic circuit. Then, Hu *et al*. (2009) proposed a true random number generator by computer mouse movement. Patidar *et al*. (2009) designed a random bit generator based on two chaotic logistic maps which is generated by comparing the outputs of the both chaotic logistic maps. Recently, Fechner and Osterloh (2010) presented a meta-level true random number generator.

Generating high quality PRNG: In many researches, there have been discussed the requirements for a good PRNG (Jun and Kocher, 1999; Wang *et al*., 2006; Ergun and Ozoguz, 2007; Hu *et al*., 2009; Patidar *et al*., 2009; Fechner and Osterloh, 2010; Shparlinski, 2004; Wang, 2000; Kumar *et al*., 2010; Rahimov *et al*., 2010; Babaei, 2011; Menezes *et al*., 1997; Fechner and Osterloh, 2010).

Uniformity of distribution: Uniformity of distribution is the main concern in the statistical tests of random sequences. It means that at any point in the generation of a sequence of random or pseudo-random bits, the occurrence of a 0 or 1 is equally likely (Shparlinski, 2004).

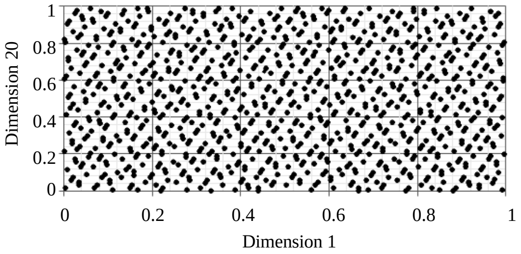

Fig. 1: Halton sequences in dimentions 1×20

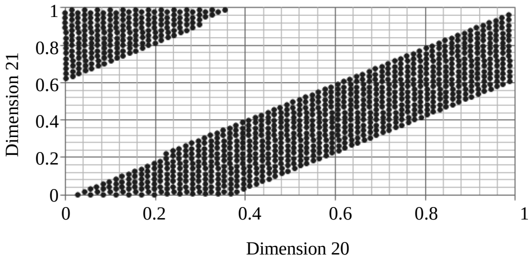

Fig. 2: Halton sequences in dimension 20×21

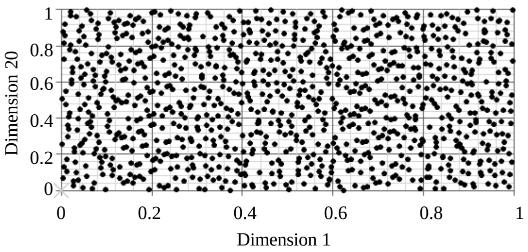

Fig. 3: Sobol sequences in dimension 1×20

Independence: Each part of a random sequence should be independent from other parts. In the sample sequence (m_0, m_1, ...), random numbers generate in the d-dimensional space so, the sample part of this sequence:

$$d-\text{tuples}=m_{dn}, m_{dn+1},\ldots, m_{dn+d-1}$$

Which is uniformity distribution should be independence in the d-dimensional cube $[0, 1]^d$ for example in Halton sequence (Wang, 2000) which is the low discrepancy method to generate random numbers decreased independence in the high dimensional (Fig. 1 and 2). Also, this multi-dimensional clustering is clear in the high dimensional, the Sobol sequence (Kumar *et al*., 2010) (Fig. 3 and 4).

Efficient length of period: Some classic algorithms for PRNGs such as Middle square method and Middle product method, although have the unique characteristic to generate pseudo-random numbers but they not enough length of periods. This problem solved by Rahimov *et al*. (2010).

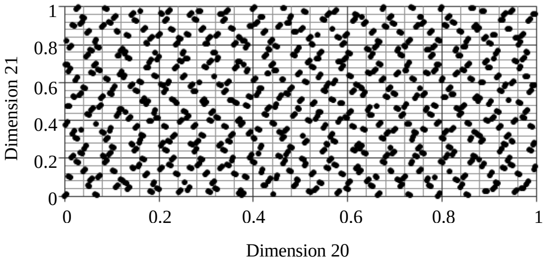

Fig. 4: Sobol sequences in dimension 20×21

Unpredictability: Unpredictability is one of the important points in cryptography because the random sequence with these advantages (i.e., efficient length of period, good independency and uniformity of distribution) could be predictable thus existing a lot of threats in the secret communication. So, we need to generate the sequence unpredictability.

In the sample sequence (m_0, m_1, ..., m_{n-1}, m_n), in the best conditions if a group of hackers have the largest part of this sequence (i.e., m_0, m_1, ..., m_{n-1}). In the unpredictable PRNG, they are not able to guess m_n with probability >50%. The chaotic maps in combination with predictable PRNGs method such as LFSR (which is implemented by the small number of register) are able to improve advantage of unpredictability by using excusive or operation between LFSR's system and chaotic logistic map, this theorem proved by Babaei (2011).

MATERIALS AND METHODS

Improving PRNGs: In the main part of this study, researchers discuss about how the PRNGs weaknesses could be improved. Now-a-days, a lot of scientists research on this subject and prepared reliable methods to improve these defects in generators, especially the applications of PRNGs in cryptography need to be more efficient than other applications.

Decimation method: In this method, two PRNGs generate random numbers in the two different sequences (the type of PRNGs may be same or different). Combination based on this method is able to generate more efficient random sequence. Decimation algorithm is described as:

```
float* Decimation()
{
 float *finalSeq;
 boolean continueGenerating=True;
 int countPRNG=0;
 char ans;
 finalSeq=new float[100];
 while(continueGenerating)
 {
  int loopLength=PRNG1();
  for(int i=1; i<=loopLength; i++)
  flooat randomNumber=PRNG2();
  finalSeq[countPRNG]=randomNumber;
}
cout<<”Do you want more?(y/n)”<<endl;
cin>>ans;
if(ans == ‘n’ || ans ==’N’)
break;
}
return finalSeq;
}
```

On the other words, this algorithm in Line 10 generates sequence (p_1, p_2, p_3, ..., p_n) and in Line 12 generates sequence (q_1, q_2, q_3, ..., q_n), so Decimation method generated final sequence as result:

$$q_{p_1}, q_{p_1+p_2}, q_{p_1+p_2+p_3}, \ldots, q_{p_1+p_2+\ldots+p_n}$$

Many researches proved generated random sequence by this method is more efficient than discrete sequences [], it means that distribution(SeqDecimation)> distribution(PRNG1) and distribution(SeqDecimation)> distribution(PRNG2).

XOR operation and combination PRNGs: One of the popular models to improve PRNG's defects is combining k numbers of generator by XOR operation. For example if each of the generators is defined by a primitive trinomial such as:

$$PT_k(x) = x^{rk} + x^{sk} + 1$$

This is the main structure of Fibonacci LFSR generators which rk is distinct primitive degrees then proved that combination of these k generator has period at least:

$$2^{\omega-1}\prod_{k=1}^{k}(2^{rk}-1)$$

In this case, present various PRNGs with different efficiency which can be classified into two main groups. This classification done by Babaei (2011) and divided combination generators based on constructors components:

Class 1: Classic generator XOR classic generator
Class 2: Classic generator XOR modern generator
Class 3: Modern generator XOR modern generator

Based on this classification, classic generator (e.g., Low discrepancy methods (Wang, 2000), High discrepancy methods, LFSR methods (Babaei, 2011) and modern generators contain any type of discrete chaotic maps (e.g., Henon, Logistic and Gauss map).

Shuffling method: In this method, two PRNGs generate two different random sequences like Decimation method, one of the sequence stores in the buffer area and the other chooses from buffer side. Shuffling algorithm is described:

```
float* Shuffling()
{
float *finalSeq;
float *buffer;
int bufferLength = 100;
char ans;
boolean continueGenerating=True;
int countPRNG=0;
finalSeq=new float[100];
buffer=new float[100];
while(continueGenerating)
{
 for(int i=0;i<bufferLength;i++)
 buffer[i]=PRNG1;
 int selection = PRNG2;
 finalSeq[countPRNG]=buffer[selection];
 cout<<”Do you want more?(y/n)”<<endl;
 cin>>ans;
 if(ans == ‘n’ || ans ==’N’)
 break;
}
return finalSeq;
}
```

RESULTS AND DISCUSSION

Statistical tests: Reliable and secure PRNGs are implemented based on strong mathematical analysis of their properties. After that the sample sequences generate and submitted to empirical statistical tests. These statistical tests disclose varied weaknesses in the sample sequences, so to achieve this goal in the source code of these tests the sub-function is responsible for mapping the sequence of random numbers into interval (0, 1) as a real variable number X because in this interval have a better approximation than the other intervals. For random variable, X passing approximation distribution tests is so important to generate secure PRNGs but it's not enough. In order to confidence a PRNGs, especially as parts of cryptography algorithms, it should be tested as an input parameter in the softwares system test. In the next sub-sections briefly introduced the best systems. Table 1-3 showed the results of well-known or widely used PRNGs beside proposed PRNGs by Rahimov *et al*. (2010) and Babaei (2011).

TestU01: This test is designed in the four classes of modules; implementing a per-programed of PRNG; implementing statistical tests; implementing per-defined batteries of tests and implementing tools for applying tests to entire families of PRNGs. These modules are implemented in the ANSI C language and offer the best

Table 1: Results of TestU01 for various PRNGs

	Butteries tests				
Generators	Log$_2^{\rho}$	t-32	Small crush	Crush	Big crush
LCG[a]	24	3.9	14	-	-
LCG[b]	57	4.2	1	10	17
LFib[c]	85	3.8	2	9	14
MSM	101	3.0	5	45	-
Choatic MSM[d]	27	3.2	9	10	16
MPM	105	3.2	7	47	-
Choatic MPM[d]	29	3.4	10	11	13
Fibonacci LFSR	30	4.1	17	-	-
Glaois LFSR	31	4.0	15	-	-
Choatic LFSR[d]	32	4.2	9	12	14

Table 2: Results of NIST for various PRNGs (Part 1)

Generators	Frequency	Block frequency	CuSums forward	CuSums backward
LCG[a]	0.804645	0.764534	0.193567	0.002323
LCG[b]	0.985634	0.893467	0.229087	0.012678
LFib[c]	0.875379	0.026789	0.679834	0.126789
MSM	0.908733	0.128908	0.873456	0.009367
Choatic MSM[d]	0.804645	0.322901	0.265567	0.090388
MPM	0.837330	0.127835	0.783606	0.091678
Choatic MPM[d]	0.963720	0.762609	0.126709	0.201289
Fibonacci LFSR	0.535558	0.256881	0.125567	0.558502
Glaois LFSR	0.269087	0.269087	0.390767	0.389001
Choatic LFSR[d]	0.606499	0.483676	0.553505	0.769260

Table 3: Results of NIST for various PRNGs (Part 2)

Generators	Rans	Long run	Rank	FFT
LCG[a]	0.876522	0.0036340	0.347851	0.000147
LCG[b]	0.753678	0.1256200	0.892736	0.000951
LFib[c]	0.595634	0.0913567	0.012673	0.000566
MSM	0.463678	0.0012370	0.347851	0.000159
Choatic MSM[d]	0.569766	0.0666730	0.248649	0.000159
MPM	0.673640	0.0873670	0.001267	0.000159
Choatic MPM[d]	0.883830	0.2837090	0.337328	0.000159
Fibonacci LFSR	0.578382	0.0123430	0.859903	0.000159
Glaois LFSR	0.369001	0.1556720	0.790510	0.000159
Choatic LFSR[d]	0.425020	0.1742490	0.967341	0.000159

a: (224, 16598013, 12820163); b: (259, 1313, 0); c: (231, 55, 24) and d: Logistic map

collection of utilities for the empirical statistical testing (McCullough, 2006). The results of the test suites is classified into three classes, small crush (consist of 10 tests), crush (consist of 60 tests) and big crush (consist of 45 tests). In Table 1, the column log$_2^{\rho}$ in the mathematical equation shows the number of period length ρ in the logarithm in base 2. The column t-32 shows the CPU time which is required to generate a sample sequence with length 10^8 of random numbers on a 32 bit computer. This computer has intel pentium processor of clock speed 2.8 GHz which the Ubuntu 8.10 as OS is running on it. Also, the small dash (-) indicates that the test was not applied to this particular PRNG, usually because the PRNG was already failed into smaller battery. The results of TestU01 is shown in Table 1.

NIST: One of the most powerful statistical tests is NIST tests suite, it is contain 15 tests which they are based on null hypothesis testing. This package focuses on large types of general non-randomness on the target sequences (NIST, 2000). All of the tests are standard normal and the amount of the Chi-square as reference distribution. So if the current sequence which is under test is non-random, the software calculates an unacceptable value for sequence distribution. The results of eight NIST tests are shown in Table 2 and 3.

CONCLUSION

In this study, researchers discussed about some important factors to generate pseudo-random numbers such as uniformity of distribution, independence, efficient length of period and unpredictability.

After that some good PRNG methods presented which are proved their reliability by the researcher in his previous publications. Finally, the statistical tests (i.e., TestU01 and NIST suite tests) are described and the results are reported.

REFERENCES

Babaei, M., 2011. Improved performance of LFSR's system with discrete chaotic iterations. World Applied Sci. J., (In Press).

Ergun, S. and S. Ozoguz, 2007. Truly random number generators based on a non-autonomous chaotic oscillator. AEU-Int. J. Electron. Commun., 62: 235-242.

Fechner, B. and A. Osterloh, 2010. A meta-level true random number generator. Int. J. Crit. Comput. Based Syst., 1: 267-279.

Hu, Y., X. Liao, K.W. Wong and Q. Zhou, 2009. A true random number generator based on mouse movement and chaotic cryptography. Chaos Solitons Fractals, 40: 2286-2293.

Jun, B. and P. Kocher, 1999. The intel random number generator. White Paper Prepared for Intel Corporation.

Kumar, P.S., R. Subramanian and D.T. Selvam, 2010. Ensuring data storage security in cloud computing using sobol sequence. Proceeding of the 1st International Conference on Parallel, Distributed and Grid Computing, Oct. 28-30, Solan, pp: 217-222.

McCullough, B.D., 2006. A review of TESTU01. J. Applied Econ., 21: 677-682.

Menezes, A.J., P.C. van Oorschot and S.A. Vanstone, 1997. Handbook of Applied Cryptography. CRC Press, USA., ISBN: 0-8493-8523-7.

Patidar, V., K.K. Sud and N.K. Pareek, 2009. A pseudo random bit generator based on chaotic logistic map and its statistical testing. J. Informatical, 33: 441-452.

Rahimov, H., M. Babaei and H. Hassanabadi, 2010. Improving middle square method RNG using chaotic map. J. Applied Math., 2: 482-486.

Shparlinski, I.E., 2004. On the uniformity of distribution of the decryption exponent in fixed encryption exponent RSA. Inform. Proc. Lett., 92: 143-147.

Wang, X., 2000. Randomized halton sequences. J. Math. Comput. Modell., 32: 887-899.

Wang, L., F.P. Wang and Z.J. Wang, 2006. Novel chaos-based pseudo-random number generator. Acta Phys. Sin., 55: 3964-3968.