Pseudo Random Bit Generation Using Arithematic Progression

Divyanjali
Assistant Professor
CEA Department, GLA University
Mathura, India
dvynjli@gmail.com

Ankur
Researcher Associate
AIM&ACT, Banasthali Vidyapith
Rajasthan, India
ankur.rathi9@gmail.com

Trishansh Bhardwaj
M.Tech Reseracher
CSE Department, NIT Warangal,
Telangana, India
itrishansh@gmail.com

***Abstract*—In our day-today life, we performs a lot of tasks that involves direct or indirect use of random numbers e.g. games, lotteries, simulations and most important cryptography and data communication security. Although, the field of pseudo random number generation is important, it is very difficult to find a good generator for several of applications. In present manuscript, we explore the possibility of a new Pseudorandom Random Number Generator and gives its testing results on NIST test battery.**

Keywords— New Algorithm, NIST statistical test report, Random Bit Generator

I. INTRODUCTION

A pseudo-random bit sequence is an output of any deterministic algorithm, such that every set of bits has an equal chance of being chosen from the universe of numbers [1]. These pseudo random bits are generated using pseudo random number generators (PRBGs). PRBG takes seed of length n is as input and produces the output of length $l(n)$, with $l(n) >> n$ is called pseudo random sequence.

Pseudo-random numbers are widely used for simulation, numerical analysis, testing of programs and hardware using random data, decision making in lotteries and games and cryptography [2]. A good PRNG must possess several of qualities such as unpredictability, large period length, uniform distribution, efficiency, portability, repeatability, and a good structure. The present manuscript explores the possibility of a new congruential function, which can generate pseudo-random numbers to be used in simulation, numerical analysis, decision making, and computer programming.

II. THE PROPOSED ALGORITHM

The algorithm proposed in this text exhibits good statistical properties while tested on NIST (National Institute of Standards and Technology) statistical test suit specified in NIST Special Publication 800-22 [3], and hence fit itself to provide source of randomness in almost every non-cryptographic application such as simulation, testing, gaming, randomized algorithm, etc. The algorithm, testing methodologies and NIST statistical test suite reports are included in subsequent sections.

A. Concept

The key idea of algorithm is to add some sequential numbers from range [1, M-1] i.e. Z_M and take modulo M of the sum. The selection of numbers to be added in the first iteration of sequence depends upon the seeds provided – first seed, say X_0, defining the start of sequence and second seed, say Y_0, defining the count of numbers to be added. So in the first iteration, number generated would be the sum of Y_0 numbers starting from X_0 in finite field Z_M and the least significant bit of this number would be the first output bit of the sequence. We will store this number in X_1 and use it instead of X_0 for next iteration and so on. At i^{th} iteration the generated bit b_i can be given by the equations

$$X_i = \sum_{i=X_{i-1}}^{X_{i-1}+Y_0-1} i \ mod \ M \quad (1)$$

And,

$$b_i = X_i mod\ 2 \quad (2)$$

B. Proposed Algorithm

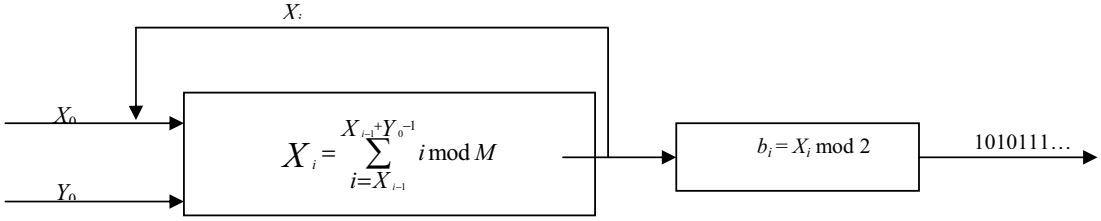

Fig. 1. Schematic diagram of Proposed Algorithm

Example

Let M=10007, Seeds X_0 =5324 and Y_0 = 10.Then,

$$X_1 = \sum_{i=X_0}^{X_0+Y_0-1} i \bmod M \equiv \sum_{i=5324}^{5333} i \bmod 10007 \equiv 3250 \qquad b_1=0$$

$$X_2 = \sum_{i=X_1}^{X_1+Y_0-1} i \bmod M \equiv \sum_{i=3250}^{3259} i \bmod 10007 \equiv 2524 \qquad b_2=0$$

The output sequence B of length using above example 8-bit is 00101010. The sequence can be reproduced, given that both the seeds are same as the sequence to be reproduces.

The problem can also be reduced to sum of numbers from an Arithmetic Progression with common difference d = 1,

2327-0659/15 $31.00 © 2015 IEEE
DOI 10.1109/ACCT.2015.90

initial term = X_0, number of terms to be added = Y_0. Sum of numbers in AP is given by

$$S = \frac{n}{2}(2a + (n-1)d) \quad (3)$$

In this case the equation (1) can be written as

$$X_i = Y_0(2X_{i-1} + Y_0 - 1)/2 \ mod\ M \quad (4)$$

Above equation is a more effective way to compute sum of the numbers rather than looping around. For this reason we have used equation (4) to generate random numbers throughout this research.

C. Pseudo code of the Algorithm

Following is the pseudo-code of the algorithm:

RANDOM ($SEED_1$, $SEED_2$, M)

1 $X_0 \leftarrow SEED_1$
2 $Y_0 \leftarrow SEED_2$
3 $l(n) \leftarrow$ length of output sequence
4 **for** $i \leftarrow 1$ **to** $l(n)$
5 **do** $X_i = Y_0(2X_{i-1} + Y_0 - 1)/2 \ mod\ M$
6 **do** $b_i \leftarrow X_i \bmod 2$
7 **return** $B \leftarrow (b_1 b_2 b_3 \ldots b_{l(n)})$

The output of above algorithm is a $l(n)$ bit number B.

III. Results and Discussion

A. Statistical Testing

A PRBG to be used for simulation and other non-cryptographic purpose need to more randomness, while PRBG used for cryptographic purpose needs to be cryptographically secure and unpredictable. To be confident about randomness of number generated from a PRBG, it is important to test its output sequences. There are several of tests batteries available such as NIST statistical test suite [3], DIEHARD test [4], Donald Knuth's statistical test suite [5], and the Crypt-XS statistical test suite [6]. They all perform a number of tests to find different type of non-randomness. None of these is perfect in itself. Juan Soto [7] has shown that not all the tests are needed to be performed and the NIST statistical test suite is the best one of these. Hence we have used NIST test suite, regarded as most rigorous tests of randomness to analyze our proposed PRBG.

The NIST suite

The NIST Test Suite, consisting of 15 tests, is a statistical package that tests the randomness of (arbitrarily long) binary sequences. These sequences can be produced either by hardware or by software based random number generators. The tests are [3]:

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Tests for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Linear Complexity Test,
11. The Serial Test,
12. The Approximate Entropy Test,
13. The Cumulative Sums (Cusums) Test,
14. The Random Excursions Test, and
15. The Random Excursions Variant Test.

Further information about these tests is not the subject of present manuscript. The test suite calculates *P-value*, the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested [3]. A significance level (α) can be chosen for the tests. If *P-value* $\geq \alpha$, then the null hypothesis is accepted, otherwise null hypothesis is rejected for the sequence under consideration. After calculating *P-value* for each sequence, proportion of sequences being passed is calculate. If the proportion falls inside the confidence interval, calculated as $\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$, where $\hat{p}$= 1-α and m is the sample size, the generated is accepted on this basis. The α = 0.01 indicates that one would expect 1 sequence in 100 sequences to be rejected. The P-value of P-values ($P\text{-}values_T$), describes Goodness-of-fit Distributional test on the *P-values* obtained for an arbitrary statistical test.

Statistical Testing Results of Proposed PRBG

For testing of suggested algorithm, we have generated 1000 sequences, each of 10^6 bits. Each of the sequence is generated from different randomly chosen seed X_0 and Y_0. The seeds are provided from a true random resource- *random.org* [8].The generation of numbers has been done using C language library-*gcc* (GNU Compiler Collection) [9], and *GMP* (GNU Multiple Precision) arithmetic library [10] to handle large numbers. NIST test battery is applied over each of these sequence and *P-values* for all 15 tests are computed. The significance level α is set to 0.01. So, minimum 980 sequences must pass the test when sample size m = 1000 i.e. for all of the tests except random excursion and random excursion variant which have sample size of m = 609 and hence needs 595 sequences to pass the test for sequences to be considered random. The parameters used for testing and results of NIST suite are summarized in Table 1 and Table 2 respectively.

Table 1. Parameters used for testing

No of sequences tested	1,000
Length of each binary sequences	1,000,000 bits
Significance level	0.01
Block size	16
Template size	9
Maximum number of templates	40

Table 2. Summary of NIST testing results

S. No.	Name of test	No. of sequences with P-value≥0.01 (Success)	P-value of P-values	Proportion of sequences passing test
1	Frequency test	995	0.365253	0.99500000

2	Block Frequency test	988	0.292519	0.98800000
3	Cumulative Sums test			
	1. Forward sums test	994	0.463512	0.99400000
	2. Reverse sums test	994	0.442831	0.99400000
4	Runs test	986	0.394195	0.98600000
5	Longest Run test	989	0.536163	0.98900000
6	Rank test	987	0.836048	0.98700000
7	FFT test	991	0.011959	0.99100000
8	Non-Overlapping Template matching test (Template Length = 9)			
	1. Template=000000011	991	0.995162	0.991000000
	2. Template=110000000	996	0.971006	0.996000000
	3. Template=111001010	991	0.975644	0.991000000
	4. Template=111001100	986	0.907419	0.986000000
	5. Template=111100000	991	0.994488	0.991000000
	6. Template=111101110	993	0.984881	0.993000000
	7. Template=111110100	997	0.875539	0.997000000
	8. Template=111011100	991	0.846338	0.991000000
9	Overlapping Template	992	0.906069	0.992000000
10	Universal test	987	0.618385	0.987000000
11	Approximate Entropy test	993	0.207730	0.993000000
12	Random Excursions test			
	1. x = -4	602	0.894201	0.988505747
	2. x = -3	597	0.761392	0.980295567
	3. x = -2	605	0.950407	0.993431856
	4. x = -1	602	0.274568	0.988505747
	5. x = 1	599	0.212371	0.983579639
	6. x = 2	606	0.408571	0.995073892
	7. x = 3	599	0.977580	0.983579639
	8. x = 4	599	0.322901	0.983579639
13	Random Excursion Variant test			
	1. x = -9	600	0.103035	0.985221675
	2. x = -8	602	0.382400	0.988505747
	3. x = -7	601	0.631914	0.986863711
	4. x = -6	600	0.618038	0.985221675
	5. x = -5	600	0.821041	0.985221675
	6. x = -4	600	0.414525	0.985221675
	7. x = -3	603	0.656200	0.990147783
	9. x = -2	604	0.379555	0.991789819
	10. x = -1	604	0.855534	0.991789819
	11. x = 1	605	0.652733	0.993431856
	12. x = 2	597	0.505865	0.980295567
	13. x = 3	592	0.126536	0.972085386
	14. x = 4	597	0.272297	0.980295567
	15. x = 5	598	0.312791	0.981937603
	16. x = 6	598	0.429618	0.981937603
	17. x = 7	599	0.021627	0.983579639
	18. x = 8	602	0.011722	0.988505747
	19. x = 9	604	0.855534	0.991789819
14	Serial test	991	0.853049	0.991000000
15	Linear Complexity test	992	0.298282	0.992000000

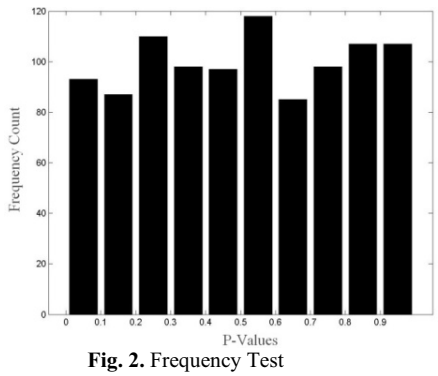

Fig. 2. Frequency Test

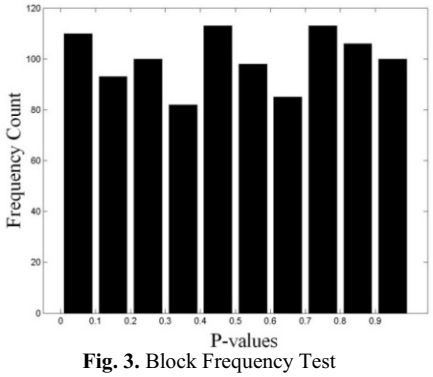

Fig. 3. Block Frequency Test

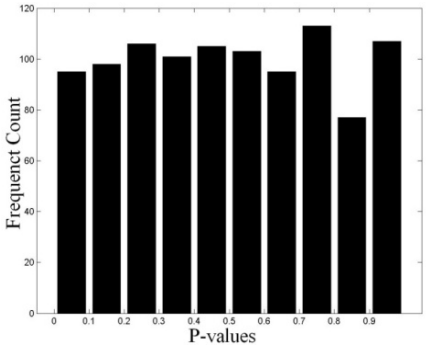

Fig. 4. Cumulative Sums Test

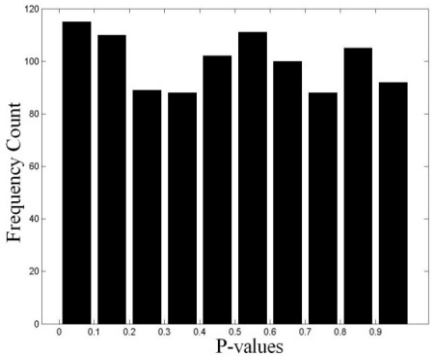

Fig. 5. Run Test

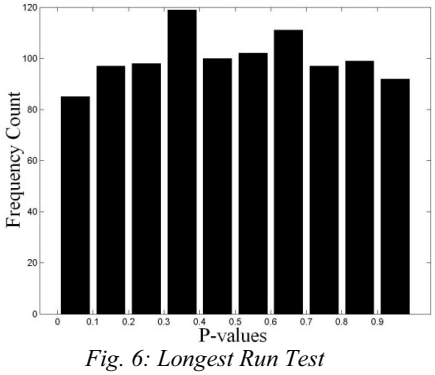

Fig. 6: Longest Run Test

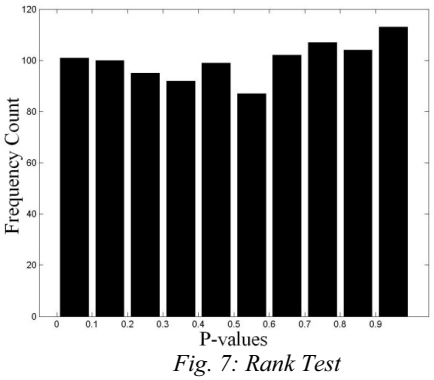

Fig. 7: Rank Test

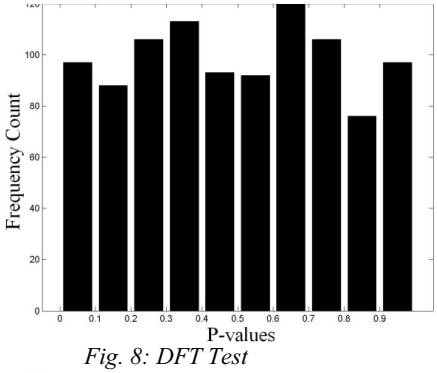

Fig. 8: DFT Test

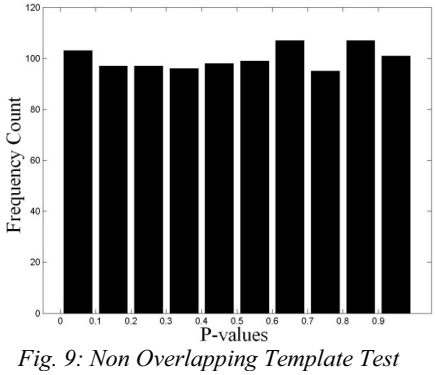

Fig. 9: Non Overlapping Template Test

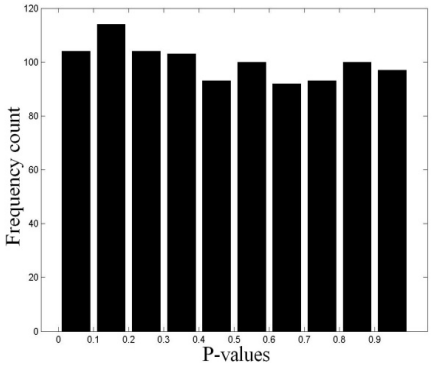

Fig. 10: Overlapping Test

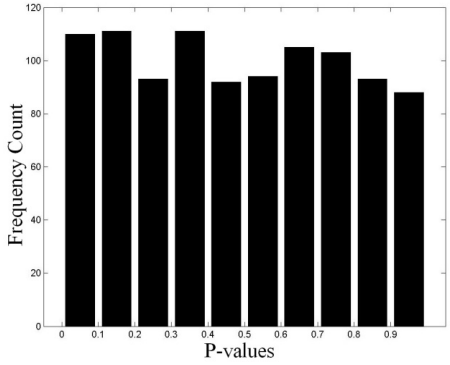

Fig. 11: Universal Test

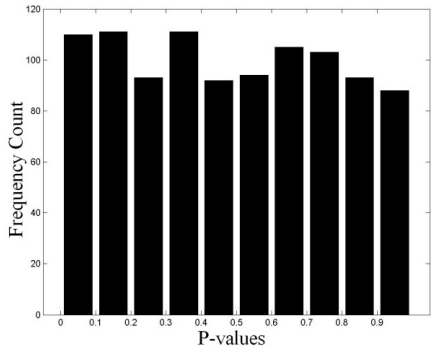

Fig. 12: Approximate Entropy

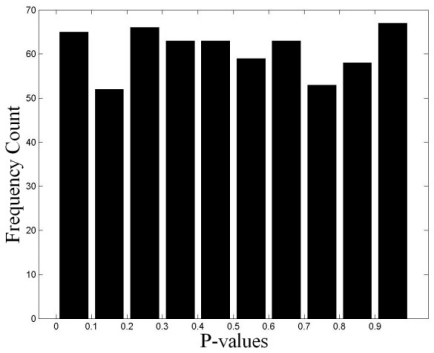

Fig. 13: Random Excursion

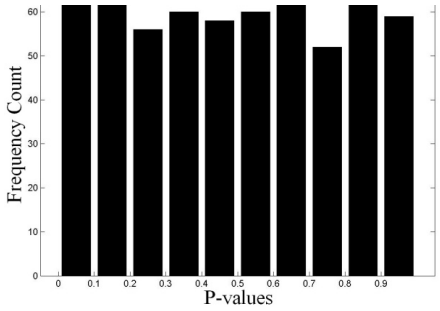

Fig. 14: Random Excursion Variant

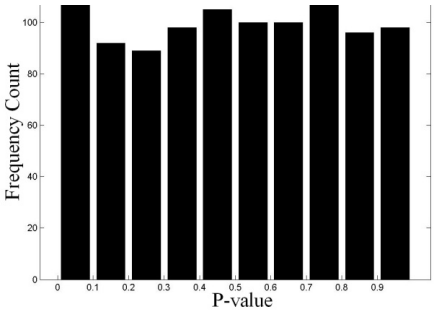

Fig. 15: Serial Test

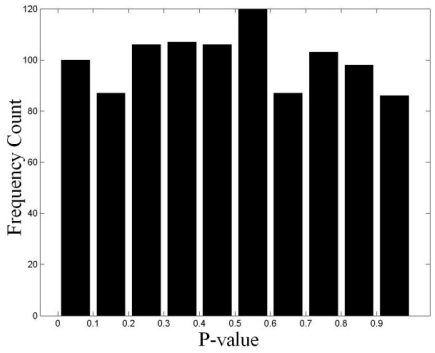

Fig. 16: Linear Complexity Test

Uniform-distribution of *P-values* for 1000 number of binary sequences has also been represented by histograms of *P-Values* for each test. The complete interval of *P-values* [0, 1] is divided into 10 equal sub-intervals and the *P-values* that lie in each subinterval has been counted and displayed in Figure 2 to 16. These graphs have been plotted using MATLAB [11].

It is clear from the Table 2 and the Figure 2 to 16 that the $P\text{-}value_T$ for each of these tests lies in confidence interval i.e. the tested binary sequence passes all these tests. These all figures and tables are designed using *finalAnalysisReport.txt* file generated by NIST. We have performed the test on some other samples also and these samples pass the tests as well.

B. Furthur Analysis

Scatter Plot: To show uniformity or uniform distribution of the numbers, we have also plotted the scatter diagram and correlation of generated numbers in MATLAB. The motivation behind this is to show the distribution graphically rather than statistically in probabilistic terms. The figure 17 contains the scatter plot of first 1000 numbers generated by the proposed algorithm with value of M=9576890767 and seeds X_0, Y_0 given at random and figure 18 shows correlation between values generated at time t and t+1 with same parameters.

Both the figures 17 and 18 exhibits that proposed algorithm has no correlation and its values are properly distributed to be called random.

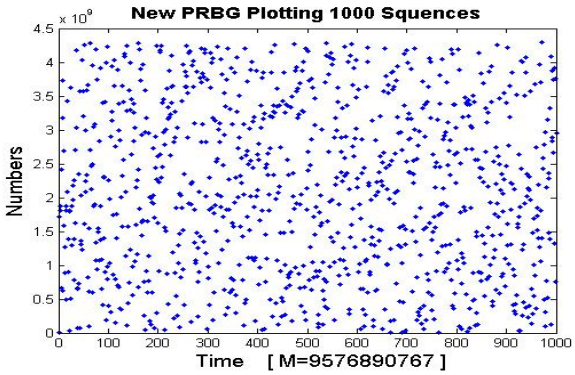

Fig. 6. Scatter Plot of Proposed Algorithm

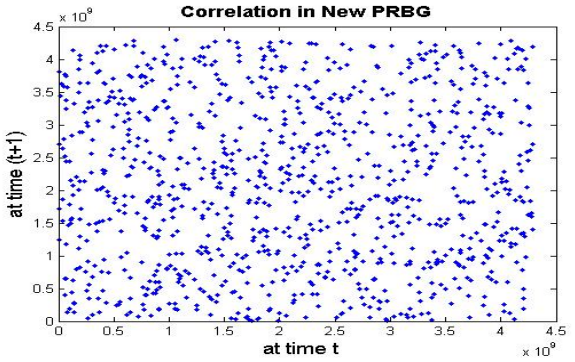

Fig. 7. Correlation Graph

Timing Analysis: Fast number generation is a desirable property of pseudo-random number generators; especially the ones to be used for simulation must generate numbers fast enough. We have computed the total running time in nanoseconds to generate one million numbers with following testing parameters:

- Linux environment
- gcc compiler
- Intel core i3 processor, 3192 Mhz
- 4 GB RAM

With no complier optimization and including time consumed in initialization i.e. reading seed from *stdin* buffer, the proposed algorithm generated 10^9 numbers in 28,904,178,124 ns. On average it generates a number in 28.904 ns. The parameters used for testing are $M = 4294967087$.

IV. Conclusion

The field of pseudorandom number generation has been explored a lot, but there is still so much of possibility to have new and better generators. We have developed and tested a one of these possibilities and created a novel pseudo random number generation algorithm. The results on NIST statistical test suite show that the performance of proposed PRBG is up to the mark and better than a lot of PRBGs in use. It encourages us to further explore the suggested PRBG.

Acknowledgement

We acknowledge our regards and thanks to Banasthali Vidyapith for providing us an opportunity to perform such a research in their premises and letting us use their resources.

References

[1] J. E. Gentle, Random Number Generation and Monte Carlo Methods, Springer, 1998, pp. 217.

[2] D. E. Kunth, The Art of Computer Programming, vol 2: Semi-Numerical Algorithms, 2nd ed., Addison- Wesley, 1981.

[3] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, San Vo, *Statistical test suite for random and pseudo random number generators for cryptographic applications*, NIST special publication 800-22, 2001.

[4] G. Marsaglia, DIEHARD statistical tests, *http://www.stat.fsu.edu/pub/diehard/*, 1995, last accessed on July 26, 2013.

[5] D. E. Knuth, The art of ComputerProgramming: Semi Emperical algorithms, Addison Wesley, 1998, Reading, USA.

[6] H. Gustafson, E. Dawson,L. Nielsen, W. Caelli, *"A computer package for measuring the strength of encryption algorithms"*, J. Computer Security, vol. 13 (1994), pp. 687-697.

[7] J. Soto, *Statistical testing of random number generators*, Proc. of 22nd National Information System Security Conference http://csrc.nist.gov/groups/ST/toolkit/rng/documents/nissc-paper.pdf

[8] M. Haahr, S. Haahr, *Random.org*, http://random.org , last accessed on July 26, 2013.

[9] *GCC, the GNU Compiler Collection*, http://gcc.gnu.org/ , last accessed on July 26, 2013.

[10] *The GNU Multiple Precision Arithmetic Library,* http://gmplib.org/ , last accessed on July 26, 2013.

[11] *Matlab: The Language of Technical Computing*, http://www.mathworks.in/products/matlab/, last accessed on July 26, 2013.