A Novel Unpredictable Temporal based Pseudo Random Number Generator

Aqeel Khalique
Jamia Hamdard
Hamdard University
New Delhi

Auqib Hamid Lone
Jamia Hamdard
Hamdard University
New Delhi

Syed Shahabuddin Ashraf
Jamia Hamdard
Hamdard University
New Delhi

ABSTRACT
Random numbers are sequences of integers which show absolutely no relation to each other anywhere in the sequence. Pseudo random numbers (PRNs) are random numbers which can be generated deterministically. Pseudo random numbers are widely used in cryptographic protocols to provide secrecy. A wide variety of pseudo random number generators (PRNGs) were proposed earlier which exhibit characteristics of PRNG and generate pseudo random numbers. In this paper, we propose a temporal based algorithm to generate pseudo random numbers. Our algorithm utilizes dynamic system clock and product of two large prime numbers generated randomly. Using dynamic system clock, we ensure that the seed is obtained from entropy source of some physical phenomenon and product of large prime numbers generated randomly ensure randomness of generated numbers. Our algorithm is easy to implement on any computing device and can be used for generating sequence of pseudo random numbers for desired purpose.

General Terms
Pseudo Random Generators, Pseudo Random Number Generation Algorithm.

Keywords
Random Number Generation, Pseudo Random Numbers, Cryptography.

1. INTRODUCTION
Group of numbers or sequences of integers which show absolutely no relation to each other anywhere in the sequence are considered to be random numbers. Use of random numbers drives the attention of many researchers. Many network security algorithms and protocols based on cryptography make use of random binary numbers. In gambling, random number generation plays a vital role towards hitting the odd. Designing a good random number generator (RNG) is need of an hour, because it is an important cryptographic primitive widely used for one time pad [1], key generation [2] and authentication protocol [3]. The security of random number generators relies on the assumption that future values in the random number sequence can't be predicted from the observed sequence. The generation of high quality randomness is vital and cornerstone of many cryptographic random data generators and the importance of careful design of cryptographic random data generators cannot be underestimated.

Broadly, RNGs are classified into two categories namely true random number generators (TRNG) and PRNG based on their application to cryptography. TRNG extracts the randomness from some physical phenomenon having some entropy source. PRNG expands a key (seed) into a long sequence of random bits based on a deterministic algorithm. It is convenient for a PRNG to be seeded again, i.e., one can bring additional source of entropy after pseudo random bits have been generated. Instead throwing away the current state of PRNG, re- seeding combines the current state of the PRNG with the new seeding material and thus makes PRNG a deterministic generator. The question comes in the mind why should we bother about random number generation as it is merely a call away because RNG are built into most of compilers. Unfortunately, these random number generators are not secure enough for cryptographic applications and probably not even very random. Good PRNGs must be cryptographically secure, a property that is often satisfied, if it passes the standard battery of NIST tests [4].

The PRNG proposed in this paper is a novel temporal based pseudo random number generator where randomness is extracted using dynamic system clock/time and large prime numbers chosen randomly. The equation to obtain PRNG bits are calculated under modulo operation of group under N where N=P*Q and P, Q are large prime numbers of 160 bits generated randomly. As the seed in the proposed algorithm is taken from dynamic system clock/time, which is more preferred in PRNGs when the seed is taken from entropy source of physical phenomenon. The proposed algorithm does not produce repeated sequences for a definite period based on frequency of execution. Moreover the proposed algorithm is deterministic in nature because it depends on Dynamic System clock.

2. BACKGROUND AND CLASSIFICATION
Random Numbers are generally classified into two broad categories namely true random numbers (TRNs) and pseudo random numbers (PRNs).

2.1 True Random Numbers
Irrespective of the importance of random number generation, surprisingly few TRNGs have been reported. Mostly there are three commonly used techniques in the literature, namely oscillator sampling, direct amplification and discrete time chaos. In 1984, Fairfield, Mortenson and Coulhart [5] developed the first RNG based on Oscillator phase noise. In 2001, Stojanovski et. al. [6] implemented an analog chaos based RNG in a 0.8 µm CMOS process utilizing switched current techniques. Petrie et. al., combined oscillator sampling, direct amplification and discrete time chaos to produce an analog VLSI chip which was robust to power supply noise and substrate signal coupling [7].

2.1.1 Features of True Random Numbers:

- Generation by physical phenomena having some entropy source
- Generation is limited by the speed with which entropy harvests
- Randomness – The sequence should be random as defined statistically.
  - Uniformity - The distribution of bits in the sequence should be uniform i.e., the frequency of occurrence of ones and zeros should be approximately equal.
  - Independence - No one subsequence in the sequence can be inferred from the others.
- Unpredictability - True random numbers are statistically independent of other numbers in the sequence and therefore unpredictable.
- Uses - gambling, modeling & simulation, security algorithms & protocols
- Examples - radioactive decay, keystroke timing, patterns, disk electrical activity, mouse movements, and instantaneous values of the system clock.

Typically used for technical purposes - atmospheric noise, thermal noise, electromagnetic or quantum phenomena

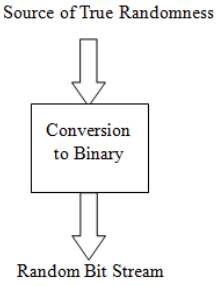

Figure 1: True Random Number Generator

2.2 Pseudo Random Numbers

Generation of PRNG's starts way back when output of Linear Feedback Shift Register was used for producing random binary sequences [8]. In 1986, Wolfran [9] proposed a method to generate Random numbers by connected cellular automata's. In 2002 P. Martin [10], evaluated different PRNG's implementation on FPGA's. In 2001, Robert K Watkins [11] introduced another FPGA implementation of PRNG, their design used genetic algorithm to generate set of PRNG's.

2.2.1 Features of Pseudo Random Numbers

- Generated by mathematical/computational algorithms
- Randomness – The sequence should appear random even though it is deterministic.
  - Uniformity - The frequency of occurrence of ones and zeros should be approximately equal.
  - Scalability - If a sequence is random, then any extracted subsequence should also be random.
  - Consistency - The generator should behave consistently based on starting values.
- Unpredictability – Pseudo random numbers should exhibit unpredictability.
  - Forward unpredictability - If the seed is unknown the next sequence should be unpredictable even though previous sequences are known.
  - Backward unpredictability – It should not be feasible to determine the seed from any generated sequences.
- Seed must be secure or it can be generated from entropy source of any physical phenomenon.
- Uses - modeling & simulation, cryptography, security algorithms & protocols.

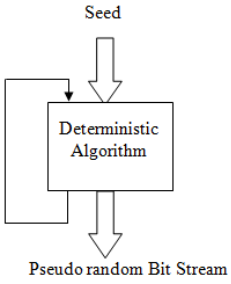

Figure 2: Pseudo Random Number Generator

2.3 Importance of Random Numbers

Random numbers are widely used in various industries including gambling, online betting, security applications, session management, financial transaction security, cryptographic applications, algorithmic research, modeling & simulation, network protocols.

3. GENERATION METHODS

3.1 Hardware based generators

Hardware based RNGs use entropy source as seed from some physical phenomenon to generate random numbers which are called true random numbers. They are primarily used in modeling, simulation and gambling. The entropy source for hardware based generators can be time difference between emissions during radioactive decay, thermal noise from electronic component, frequency instability of running oscillator, atmospheric noise, any video input etc.

Generators based on the radioactive decay and thermal noise have to be built externally to the device using the random bits and thus can be manipulated by some adversaries. Generators based on oscillators and capacitors can be built on VLSI devices and can be enclosed in tamper resistant hardware and hence secured from active adversaries.

3.2 Software based generators

Designing a random bit generator in software is even more difficult than doing so in hardware. Processes upon which software random bit generators may be based include system clock, time difference between key presses, input/output buffers etc.

The behavior of such processes can vary considerably depending on various factors such as the computer hardware or platform on which they are implemented. It may also be difficult to prevent an adversary from observing or manipulating these processes. For example, if an adversary has an idea about a particular time at which random sequence is being generated, then he can guess the content of the system clock at that time with a high degree of accuracy. A

well designed software random bit generator should utilize good sources of randomness which are available.

4. POPULAR PRNGs REVIEW

In this section we review the theoretical principles, algorithms and properties of PRNGs that are mostly based on algebraic concepts and are used as building blocks for the PRNGs implementations in various programming languages. We will use this review as reference when stating the theoretical PRNG of each implementation in applicable analysis sections.

4.1 Blum Blum Shub (BBS)

Blum Blum Shub (BBS) is a PRNG proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub [12] that is derived from Michael O. Rabin's oblivious transfer mapping. BBS generates the sequence of random numbers using the deterministic expression:

$$x_{n+1} = x_n^2 \bmod M$$

where, M = p*q, product of two large primes p and q.

x_0 should be an integer such that gcd (x_0, M) =1

$x_0 \neq 0$ and $x_0 \neq 1$

$p \equiv 3 \pmod 4$

$q \equiv 3 \pmod 4$

BBS generator is very slow. However, there is a proof reducing its security to the computational difficulty of computing modular square roots. When the primes are chosen appropriately distinguishing the output bits from random should be at least as difficult as factoring M.

4.2 Linear Congruential Generator (LCG)

A linear congruential generator (LCG) is an algorithm that yields a sequence of pseudo-randomized numbers calculated with a discontinuous piecewise linear equation. LCG was developed by D. H. Lehmer in 1949 [13]. The method represents one of the oldest and best-known pseudorandom number generator algorithms. LCG is defined by the recurrence relation:

$$x_{n+1} = (ax_n + c) \bmod m$$

where, x_0 *is the initial value (seed) for* $n \geq 0$ *and* $0 \leq x_0 < m$

a, c and m are constants

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

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

The requirement constraints for initializing these values are such that *c* and *m* are relatively prime.

LCG generates sequence of maximum periodic length *m* and it strongly depends on the initialization values including seed.

4.3 Linear Feedback Shift Register (LFSR)

Linear Feedback Shift Register (LFSR) [14] is a shift register whose input bit is a linear function of the previous state. LFSR are prominent building blocks in many cryptographic fields such as stream ciphers. They are mostly used as they are easy to implement in hardware, produce sequences of larger period, have good statistics properties and can be analyzed using algebraic techniques.

LFSR has three functional components: a shift register, a linear feedback function and a clock which times (seeds) when the shift occurs. The shift register is a sequence of bits generated at that particular time. Each time an output bit is needed, the generator is stepped by shifting all the bits 1 position to the right. The new left-most bit (input bit) is computed as a function of the other bits in the register. The output bit is the bit in stage 0 (least significant bits). The feedback function is XOR operation of bits in the registers. Figure 3 shows an example configuration of LFSR [15].

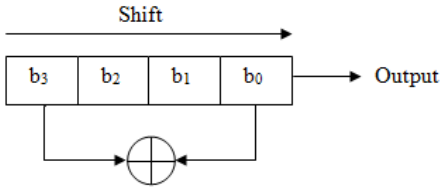

Figure 3: Linear Feedback Shift Register

5. A NOVEL UNPREDICTABLE TEMPORAL BASED PSEUDO RANDOM NUMBER GENERATOR

We proposed a novel temporal based pseudo random number generator which utilizes dynamic system clock and product of two large prime numbers generated randomly. By using dynamic system clock, we ensured that the seed is obtained from entropy source of some physical phenomenon and product of large prime numbers generated randomly ensured randomness of generated numbers.

5.1 Algorithm

We generated random binary sequences by using the algorithm given below:

Function Prime (integer Limit)

INPUT: *Limit → A positive Integer.*

OUTPUT: *Prime → A List of Prime Numbers from 1 to Limit -1.*

Variable isPrime → Boolean

Variable Prime → List of Integers

Loop i from 1 to Limit - 1

isPrime = True

Loop j from 2 to i - 1

If (i mod j) == 0

isPrime = False

End if

End Loop

If isPrime = True

add i to List Prime

End if

End Loop

Function ModFun(List Prime)

INPUT*: List Prime → List of Prime Numbers*

OUTPUT: *N → Number modulo Function*

Variable rnd → new Random( )

Variable nextval → rnd.Next (n1, n2)

// n1 and n2 are relatively prime

*Variable Limit1 → nextval * (Prime.count/nextval)*

Variable Limit2 → Limit1 - ((Prime.count/nextval)

Variable P → Prime.item (Limit1 – 1)

Variable Q → Prime.item (Limit2 - 1)

*Variable N → P * Q*

Function RndToken(integer N)

INPUT: *N → Number for Modulo Operation*

OUTPUT: *RndToken → Random Token or Random Number*

*Variable RndToken → ((System clock Ticks + 1) * (System clock hour + 1) * a) Mod N*

// a and N are relatively prime

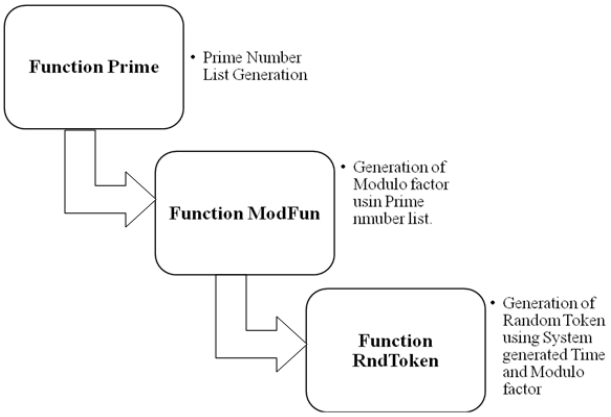

Figure 4: Flow Diagram of Algorithm

6. TEST RESULTS AND ANALYSIS

The algorithm was implemented and executed on Intel i3 2.4 Ghz processor with 4 GB RAM. It generates sequence of binary random numbers which satisfies above mentioned properties of pseudo random numbers. NIST has mentioned a test suite for testing 15 tests mentioned in the document [4]. For many of the tests in this test suite, the assumption has been made that the size of the sequence is large up to the order of 10^3 to 10^7. We had tested our binary random numbers on several test suites which are discussed in the following section.

Frequency Test: Frequency Test determines number of zeroes and ones in a sequence. The test assesses the ratio of number of zeroes and number of ones and requirement for passing the test is that the ratio should be approximately unity.

Frequency Test within a Block: Frequency Test within a Block determines the frequency of ones in an M-bit block. The test assesses that for M-bit block the frequency of ones should be approximately M/2.

Runs Test: Runs Test determines the total number of runs (uninterrupted sequence of identical bits) in the sequence. The test assesses the oscillation between zeroes and ones in the sequence.

Test for the Longest Run of Ones in a Block: Test for the Longest Run of Ones in a Block determines longest run of ones in M-bit block. The test assesses whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones in the sequence.

Non-Overlapping Template Matching Test: Non-Overlapping Template Matching Test determines the number of occurrences of pre-specified target strings. The test assesses and detects generators that produce too many occurrences of a given non-periodic (aperiodic) pattern. It uses a M-bit window of string sequence to search for specific pattern. If the pattern is not found, the window slides one bit position. If the pattern is found, the window is reset to the bit after the found pattern and the search resumes.

Overlapping Template Matching Test: Overlapping Template Matching test determines the number of occurrences of pre-specified target strings. It uses a M-bit window of string sequence to search for specific pattern. If the pattern is not found, the window slides one bit position. If the pattern is found, the window slides only one bit and the search resumes.

Maurer's "Universal Statistical" Test: Maurer's "Universal Statistical" Test determines number of bits between matching patterns. The test assesses whether or not the sequence can be significantly compressed without loss of information. If the sequence can be compressed significantly it is considered to be non-random.

Cumulative Sums Test: Cumulative Sums Test determines the cumulative sum of bits in the sequence. The adjusted weights of -1, +1 are given to zeroes and ones respectively. The test assesses whether the cumulative sum for random sequence is too large or too small.

We have tabulated the test result for a sample of random sequences generated by our algorithm. Table 1 shows the binary sequences and their result for respective test suites. We have restricted our test compliance to six test suites for conciseness of this work. Any sequence which satisfies the requirement of passing the corresponding test as specified by NIST test suite [4] has been marked as "✓" in the respective cells. However, if it fails to satisfy, a "✗" has been marked in the cells corresponding to the tests. Figure 5 shows numerical analysis of the test results pertaining to random sequences generated. The graph in Figure 5 has random sequences on x-axis and discrete values of corresponding tests on y-axis calculated as specified by NIST test suite [4] with approximation. Each sequence has been presented for six tests using histograms. Therefore, for every sequence six histograms represent discrete values on y-axis. For instance, first sequence "10101010010101010010" holds a value of 0.91 for Frequency Test, 0.8 for Frequency Test within a Block, 0.6 for Runs Test, 0.8 for Longest Run of Ones in a Block, 0.9 for Non-Overlapping Template Matching Test, 0.6 for Overlapping Template Matching Test, 0.9 for Maurer's "Universal Statistical" Test, 0.045 for Cumulative Sums Test.

Table 1: Test Suite of the Proposed Algorithm

Binary Random Sequence	Frequency Test	Frequency Test within a Block	Runs Test	Test for the Longest Run of Ones in a Block	Non-Overlapping Template Matching Test	Overlapping Template Matching Test	Maurer's "Universal Statistical" Test	Cumulative Sums Test
10101010010101010010	✓	✓	✓	✓	✓	✓	✓	✓
00010010100100101001	✓	✓	✓	✓	✓	✓	✓	✓
10101010000010010010	✓	✓	✓	✓	✓	✓	✓	×
10100010000100101010	✓	✓	✓	✓	✓	✓	✓	×
01001010101001010001	✓	✓	✓	✓	✓	✓	✓	✓
10100101010101010101	✓	✓	✓	✓	✓	✓	✓	✓
01010101001000010101	✓	✓	✓	✓	×	×	✓	✓
10101001001010010001	✓	✓	✓	✓	×	×	✓	✓
10100101010101010100	✓	✓	✓	✓	✓	✓	✓	✓
01010101010010101010	✓	✓	✓	✓	✓	✓	✓	✓

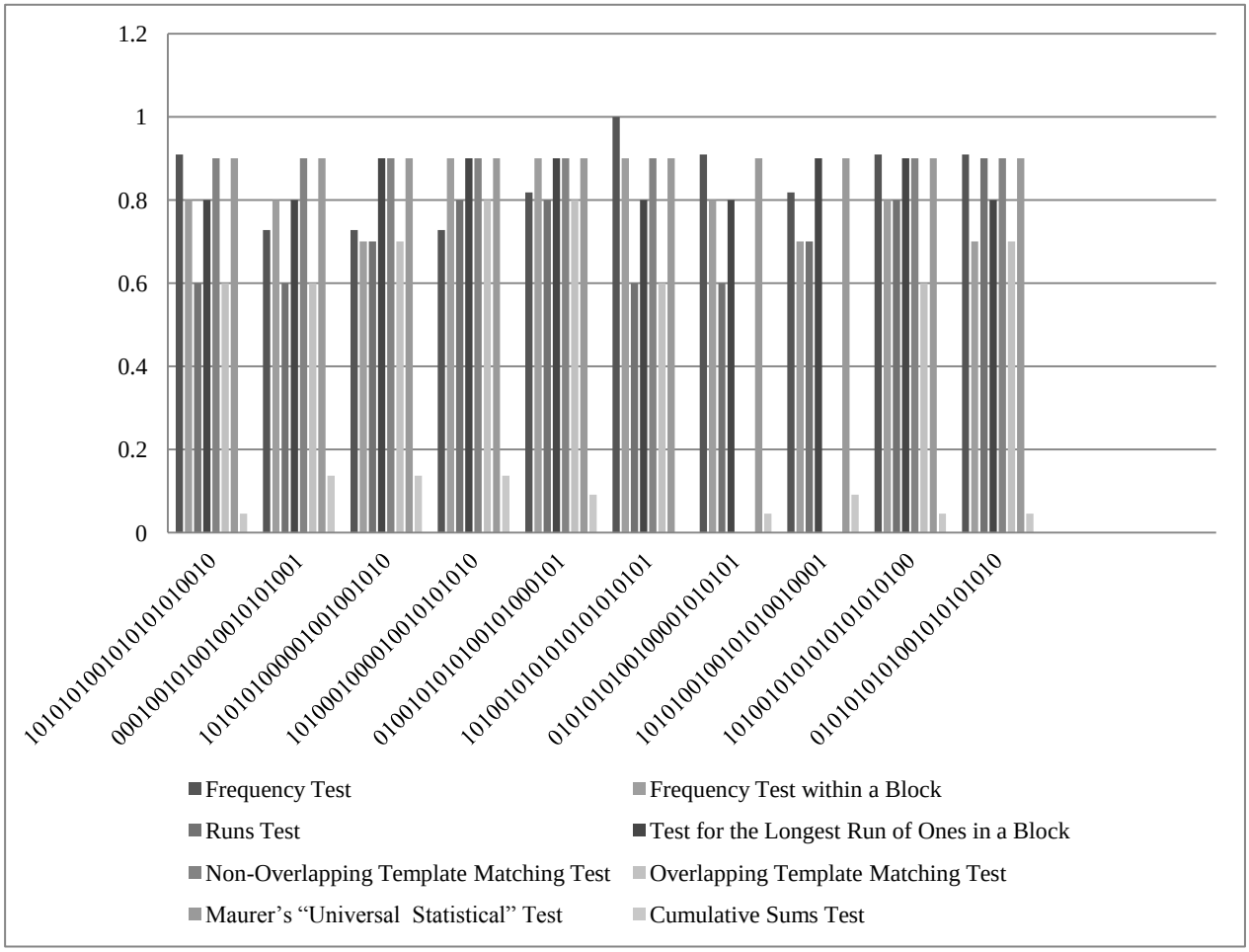

Figure 5: Numerical Analysis of Test Results

7. CONCLUSION

PRNGs are widely used in various security, cryptographic and networking protocols. In this paper, we proposed a novel algorithm for generating pseudo random numbers using dynamic system clock as entropy source from physical phenomenon and product of large prime numbers which determine our seed (initial value). We also formulated an expression through which binary sequences of random numbers are generated. This algorithm can be easily implemented on any computing device and can be utilized to serve the requirement of a pseudo random number generator. Also, this algorithm is far simpler and at the same time generates sequence of pseudo random numbers in binary form which fulfill randomness and unpredictability. The scope of temporal based pseudo random numbers can be utilized in several different application domains for random number generations. The unpredictability aspect of true randomness can be achieved by utilizing any entropy source from physical phenomenon. In future, other physical phenomenon can be utilized in several existing random number generating algorithms.

8. ACKNOWLEDGMENTS

We thank our friends for motivation and support in presenting this paper.

9. REFERENCES

[1] Schneier, B. 1996. Applied Cryptography. John Wiley & Sons. 2nd Edition.

[2] Ramaswamy, R. 1989. Application of key generation and distribution algorithm for secure communication in open system interconnection architecture. In Proceedings of International Carnahan Conference on Security Technology.

[3] Rinne, T., Ylonen, T., Kivinen, T. and Sami, M.S. 2002. SSH Authentication Protocol. Network Working Group, Internet Draft. IETF.

[4] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J. and Vo S. 2010. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22.

[5] Fair, R.C., Mortenson, R.L. and Coulthart, K. B. 1984. An LSI Random Number Generator (RNG). Advances in Cryptography: Proceedings of Crypto 84. Vol. 196.

[6] Stojanovski, T., Pil, J. and Kocarev L. 2001. Chaos-based random number generators. Part II: practical realization. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions. Vol. 48.

[7] Petrie, C. S. and Connelly J. 2000. A noise-based IC random number generator for applications in cryptography. IEEE Transactions on Circuits and Systems. Vol.47.

[8] Knuth, D. 1981. The Art of Computer Programming. Addison-Wesley.

[9] Wolfram, S. 1986. Random sequence generation by cellular automata. Advances in Applied Mathematics. Vol. 7.

[10] Martin, P. 2002. An analysis of random number generators for a hardware implementation of genetic programming using FPGAs and Handel-C. Proceedings of the Genetic and Evolutionary Computation Conference.

[11] Watkins, R. K., Isaacs, J. C. and Foo S. Y. 2001. Evolvable random number generators: A schemata-based approach. Genetic and Evolutionary Computation Conference Late Breaking Papers.

[12] Blum, L., Blum, M. and Shub M. 1986. A simple unpredictable pseudo-random number generator. SIAM Journal on computing. Vol. 15.

[13] Lehmer, D. H. 1949. Mathematical methods in large-scale computing units. 2nd Symposium on Large-Scale Digital Calculating Machinery. pp. 141-146.

[14] Klein, A. 2013. Stream Ciphers. Chapter 2: Linear Feedback Shift Registers.

[15] Aviv Sinai, 2011. Pseudo Random Number Generators in Programming Languages: http://portal.idc.ac.il/en/schools/cs/research/documents/sinai_2011.pdf