An Overview of Cryptographically Secure Pseudorandom Number generators and BBS

Divyanjali
M.Tech Researcher
Apaji Institute,
Banasthali Vidyapith,
Rajasthan, India

Ankur
M.Tech Researcher
Apaji Institute,
Banasthali Vidyapith,
Rajasthan, India

Vikas Pareek
Associate Professor
Apaji Institute
Banasthali Vidyapith,
Rajasthan, India

ABSTRACT
In this manuscript we have presented a literature survey of cryptographically securepseudo random number generators, their requirements regarding statistical properties and next bit test. The paper also provides a brief overview of Blum Blum Shub (BBS) Generator specifically, which is considered to be the best cryptographically secure pseudorandom number generator. We have performed the rigorous testing of BBS generator on National Institute of Science and Technology (NIST) statistical test suite*2.1.1*. Scatter plot and *P-value* distribution graphs are also included in the manuscript to support the conclusion.

General Terms
Random number generation

Keywords
Blum Blum Shub generator,Cryptographically SecurePseudoRandom Bit Generator,RSA generator

1. INTRODUCTION
Random numbers are generated with the help of a computer program and are deterministic in nature, they are developed to beused in simulation to simulate natural phenomena and where a vast amount of random digits are needed. In the process of generating random digits a random seed of length *l*is obtained from a true random source, is then supplied to the algorithm to produce a longer bit stream of random bits of length $k>>l$.

Many algorithms are designed to generate these long sequence of random bits but to predict the next generated bit in the sequence is difficult for a adversary but not impossible, so to protect the communication over internet, as SSL handshake process keys, Encryption keys are generated with these random number generators it is necessary that the generated random bits should be obtained from a cryptographically secure pseudo random number generator. Some Cryptographically secure pseudorandom number generators (CSPRNG) uses the source of entropy that are truly unbiased, fully random and unbreakable by any intruder, like Lavarand [1], Simon Cooper and Landon Curt Noll introduced the new version of Lavarand by replacing the lava lamps with another source of entropy a webcam with its lens cap on. The thermal “noise” produced by the webcam is captured it is digitized and then a Hash algorithm is applied over it, that do the mixing of numbers strip off unwanted sections of predictability and generates cryptographically secure random sequences. The new generator turn out to be more popular than the old lava lamp because it is license-free, patent-free, open source and easily available to user on the website. But in present manuscript we emphasize on cryptographically secure pseudo random numbers generators based on way function problems.

One of the most important tasks of random number generators is key generation but its uses are not limited to cryptography. Depending upon nature of generators, random number generators are classified; if the random number generator is based on any one-way function, it is easy to compute $y=f(x)$ but very difficult to compute $x=f^{-1}(y)$, for example; Discrete log problem (Blum Micali Generator) [2], Quadratic residue problem (Blum Blum Shub) [3], Hash (FIPS -186) [4],Elliptic curve cryptography [5], Subset sum [6], Integer factorization (RSA generator) [7] it is said to be cryptographically secure pseudo random number generator (CSPRNG). A property of these RNGs is that there is no algorithm exist, which can find out next bit to be generated in the sequence given previous bits without knowledge of seed in polynomial time. If the RNG is to be used in simulation, it need not to be cryptographically secured but should have long cycle length and uniform distribution over the range of number domain.

A cryptographically secure pseudorandom bit is the bit that must be non-deterministic in nature and can be used for cryptographic purpose. To confirm this, next bit test is to be performed. Next-bit test is finding an polynomial time algorithm that can predict $(k+1)^{th}$ bit with probability greater than 1/2, when given first k bits as input for bit generators and generates random bits that can be used for cryptographic application.

2. LITERATURE SURVEY
Computer’s first requirement of random numbers was for simulations and numerical computations such as Monte Carlo calculations. It is significant to note that the requirement of randomness is different in cryptographic applications than that of simulation. Prediction is often a prerequisite when we talk about cryptographically secure pseudo random number generator.

First cryptographically secure pseudo random number generator [7], in terms of unpredictability, was introduced by Adi Shamir in 1981 who is one of the inventors of RSA [8]. Shamir uses the core concept of RSA i.e. the problem of Integer factorization to ensure the security of his new CSPRNG, with the assumption of the strength of this CSPRNG is equivalent to the security of the RSA cryptosystem because of intractability of the problem of Integer factorization. On the other hand, since it uses modular exponentiation of huge numbers, makes it slow as well as not suitable for practical applications.

Micali-Schnorr introduced a slight improved and modified version of RSA generator is more efficient than the RSA since $\left\lfloor N\left(1-\frac{2}{e}\right)\right\rfloor$ bits are generated per exponential by e, where e is encryption key for RSA. However each exponentiation requires one modular squaring per bit.

Subsequently another Cryptographically secure PRBG is introduced Blum Blum Shub generator [3] also known as BBS generator which works on the concept of quadratic residue one way function. Its security also depends upon the intractability of the Integer factorization of modulus *N*. BBS generator requires only one modular squaring per bit, instead of one modular exponentiation. In contrast to RSA generator, BBS generator is also slow but suitable for practical applications like session key generation, Public key cryptosystem, nonce etc. Many other generators were also proposed as cryptographically secure PRBG but RSA and BBS generators are the most famous among all of them.

3. REQUIREMENTS OF CRYPTOGRAPHICALLY SECURE PRBG

A good pseudo random generator is needed to qualify on some standards to prove to be truly random. To verify this many statistical test are performed that are used to find whether there exist any correlation or not and the generated bit sequence have a good period length and uniform distribution over $U(0,\ 1)$. But unfortunately, there is no statistical test that can accomplish that if a PRNG passes all the tests of statistical test suite like National Institute of Science and Technology (NIST 2.1.1) [9] is flawless and cryptographically secure.

The requirement of normal PRBGs are satisfied by a cryptographically secure PRBG but contrarily is not true. They have to pass the statistical test suites as well as also have to prove the immunity over any adversary's attack, so that one should be unable to crack the PRBG.

Andrew Yao, the Knuth prize winner scientist mentioned in his paper in 1982 [10] that if a generator is passing the next bit test can pass all other polynomial time statistical test for randomness and should satisfy the property of unpredictability. It is necessary for a good PRBG to hold the property of unpredictability. There exist two types of unpredictability [11]:

- **Forward unpredictability**

Forward unpredictability is inability to predict next number inthe sequence, with probability more than 1/2 i.e. 0.5 without the knowledge of seed, given that previous numbers are known.

- **Backward unpredictability**

In backward unpredictability seed should not be derived fromthe output of pseudo random number generator [12] this done because if the seed is derived then pseudorandom number generator is no more secure and its randomness is compromised.

James Reeds [13] specify two notable standards of randomness in his paper "Cracking a random number generator" in 1977.

- The usual statistical standard states that a sequence of numbers, which cannot be distinguished from a sequence of true random numbers chosen from the unit interval, isconsidered random. Sequence should be uniformly distributed over the given space of numbers. PRNGs that are acceptable by these standards are suitable for simulations, sampling, games, computer algorithms, medicine and similar applications.
- If a PRBG is to be used for a cryptographic application it should satisfy some key properties i.e. unpredictability is most important rather than uniform distribution and long cycle length. It is less important whether the sequence is uniformly distributed, but it is essential that the generated numbers should not contribute to adversary to find previous number or the next number going to generate.

4. BBS: THE CRYPTOGRAPHICALLY SECURE PRBG

Blum Blum Shub is a provably secure pseudo random number generator, proposed by Lenore Blum, Manuel Blum and Michael Shub in 1986 [3]. The algorithm is considered to beas secure as quadratic residue problem, an NP-complete problem. In other words, to break the BBS is equivalent to solve the quadratic residue problem, which in turn, would solve the NP-complete problem the basis of cryptography. Due to this reason thealgorithm is most preferable algorithm for cryptographic purpose like key generation. This section explains the algorithm along with the quadratic residue problem.

4.1 Quadratic Residue Problem

For natural numbers a, n i.e. $a, n \in N$ and $a \in Z_n$, a is said to be quadratic residue n if and only if a number x with $x \in Z_n$exists and$a$ is congruent to x squared modulo n i.e.

$$a \equiv x^2 mod\ n \qquad (1)$$

$QR_n$is used to represent the set of all quadratic residues modulo n and QNR_nstands for the set of all quadratic non-residues modulo n[14]. If n is an odd prime then its QR_nwould contain elements less than $n/2$.

The quadratic residue problem is to check whether a given number a is quadratic residue modulo n or not. To know this one has to find a number x for which condition given in equation (1) is satisfied, that can be done only by using brute-force technique to exhaust Z_n. However, to compute its reverse i.e. finding a given numbers x, n, is quite easy to calculate, hence it is considered to be an NP-complete problem. The subsequent sectionillustrates how BBS uses the same problem as a one-way function.

4.2 BBS: The Algorithm

To use the concept of quadratic residue, Blum, Blum and Shub proposed to choose the two odd primes p and q, and compute $n = p \cdot q$. Then square modulo n of seed is computed and theresulting number is considered to be the first number generated. The seed is replaced with generated number insubsequent iterations and a square modulo n is computedagain, generating a number per iteration. To geta bit sequence, least significant bit of generated number isextracted per iteration and is added to the generated binarysequence. Hence BBS needs only one square (ormultiplication) per bit generated. Let the seed is s, $s \in Z_n$, thenfirst number is

$X_1 \equiv s^2 mod\ n$ and the bit $b_1 \equiv X_1 mod\ 2$

For i^{th} iteration

$\mathrm{X_i \equiv X_{i-1}^2 mod\ n}$ and $\mathrm{b_i \equiv X_i mod\ 2}$

This way the algorithm needs to compute only one square operation to generate a bit, which is much less than any of the other cryptographically secure algorithm.

Following is the pseudo code of the algorithm:

BLUM_BLUM_SHUB (*SEED*):

1. $X_0 = SEED$
2. Choose two odd prime p and q
3. $n \leftarrow p \cdot q$
4. $l \leftarrow$ length of sequence
5. **for**$i \leftarrow 1$ to l
6. **do**$X_i \equiv X_{i-1}^2 mod\ n$
7. **do**$b_i \equiv X_i mod\ 2$
8. **return** B = $<b_1 b_2 b_3 \ldots b_l>$

Example

Let $p = 101$, $q = 97$ and seed $X_0 = 128$

$n = 101 \cdot 97 = 9797$

$X_1 \equiv 128^2 mod\ 9797$	= 6587	b_1=1
$X_2 \equiv 6587^2 mod\ 9797$	= 7453	b_2=1
$X_3 \equiv 7453^2 mod\ 9797$	= 8016	b_3=0
$X_4 \equiv 8016^2 mod\ 9797$	= 7530	b_4=0
$X_5 \equiv 7530^2 mod\ 9797$	= 5661	b_5=1
$X_6 \equiv 5661^2 mod\ 9797$	= 934	b_6=0
$X_7 \equiv 934^2 mod\ 9797$	= 423	b_7=1
$X_8 \equiv 423^2 mod\ 9797$	= 2583	b_8=1

The generate bit sequence of above example is 11001011.

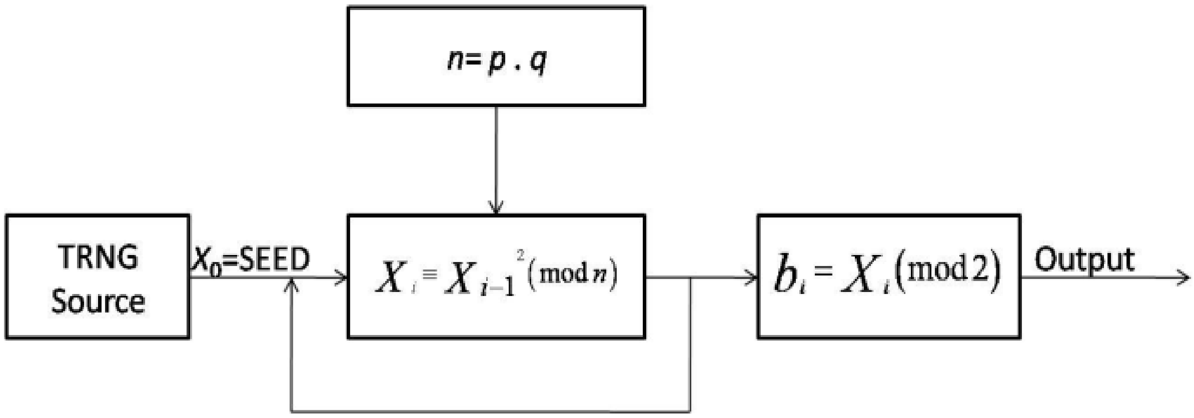

Fig 1: Schematic diagram of BBS

In this example of BBS generator, here we extract the least significant bits of the generated sequence X_i as it is mentioned in the original draft of the paper proposed by the Blum Blum and Shub in 1978. But if we extract the k least significant bits from the sequence rather than extracting one bit per squaring that is costlier and makes the functioning of generator slow, k bits per squaring increase the speed and also do not affect the security of algorithm.

5. TESTING OF BBS

The randomness of BBS is tested rigorously using statistical test suites and scatter plots. The PRBGs used forcryptographic 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 availablesuch as NIST statistical test suite [9], DIEHARD test [15], Donald Knuth's statistical test suite [16], and the Crypt-XS statistical test suite [17]. They all perform a number of tests to find different type of non-randomness. None of these is perfect in itself. Juan Soto [18] has shown that not all the testsare needed to be performed and the NIST statistical test suiteis the best in one of these. Hence we have used NIST statistical test suite*sts-2.1.1*, regarded as most precise tests of randomness to analyze the statistical properties of BBS generator.

5.1 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 [9]:

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 [9]. 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. The α = 0.01 indicatesthat one would expect 1 sequence in 100 sequences to be rejected. The *P-value* of *P-values* ($P\text{-}value_T$), describesGoodness-of-fit Distribution test on the *P-values* obtained for an arbitrary statistical test (i.e., a *P-value* of the *P-values*).

5.2 Statistical Test Results of BBS

For testing of suggested algorithm, we have generated 1000 sequences, each of 10^6 bits. Each of the sequence is generatedfrom different randomly chosen seed. The seeds are provided from a true random source - Random.org [19].The generation of numbers has been done using C language library- GCC (GNU Compiler Collection) [20], and GMP(GNU Multiple Precision) arithmetic library [21] to handle large numbers. NIST 2.1.1 test battery is applied over each of thesesequence 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 2 respectively:

Table 1. NIST parameter List

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. NIST2.1.1 testing results

S. No.	Name of test	No. of sequences with P-value≥0.01 (Success)	P-value of P-values	Proportion ofsequences passing the test
1	Frequency test	983	0.057146	0.983
2	Block Frequency test	988	0.837781	0.988
3	Cumulative Sums test			
	1) Forward sums test	983	0.307077	0.994
	2) Reverse sums test	983	0.268917	0.983
4	Runs test	989	0.452173	0.989
5	Longest Run test	983	0.801865	0.983
6	Rank test	987	0.970302	0.987
7	FFT test	986	0.070299	0.991
8	Non-Overlapping Template matching test (Template Length = 9)			
	1. Template = 000000011	981	0.90569	0.981
	2. Template = 110000000	990	0.310049	0.99
	3. Template = 111001010	989	0.884671	0.989
	4. Template = 111001100	985	0.548314	0.985
	5. Template = 111100000	982	0.263572	0.982
	6. Template = 111101110	994	0.00087	0.994
	7. Template = 111110100	988	0.496351	0.988
	8. Template = 111011100	994	0.161703	0.994
9	Overlapping Template test	983	0.167184	0.983
10	Universal test	990	0.417219	0.99
11	Approximate Entropy test	990	0.630872	0.993
12	Random Excursions test			
	1. x = -4	615	0.550479	0.992

	2. x = -3	613	0.446255	0.989
	3. x = -2	613	0.540661	0.989
	4. x = -1	617	0.258434	0.995
	5. x = 1	614	0.886546	0.99
	6. x = 2	613	0.623687	0.987
	7. x = 3	615	0.913523	0.992
	8. x = 4	614	0.808301	0.99
13	Random Excursion Variant test			
	1. x = -9	600	0.103035	0.985
	2. x = -8	602	0.3824	0.988
	3. x = -7	601	0.631914	0.986
	4. x = -6	600	0.618038	0.985
	5. x = -5	600	0.821041	0.985
	6. x = -4	600	0.414525	0.985
	7. x = -3	603	0.6562	0.99
	8. x = -2	604	0.379555	0.991
	9. x = -1	604	0.855534	0.991
	10. x= 1	605	0.652733	0.993
	11. x = 2	597	0.505865	0.98
	12. x = 3	592	0.126536	0.972
	13. x = 4	597	0.272297	0.98
	14. x = 5	598	0.312791	0.981
	15. x = 6	598	0.429618	0.981
	16. x = 7	599	0.021627	0.983
	17. x = 8	602	0.011722	0.988
	18. x = 9	604	0.855534	0.991
14	Serial test	991	0.853049	0.991
15	Linear Complexity test	992	0.298282	0.992

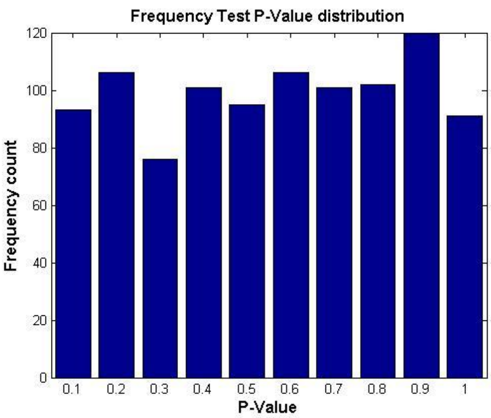

Fig 2: Frequency Test

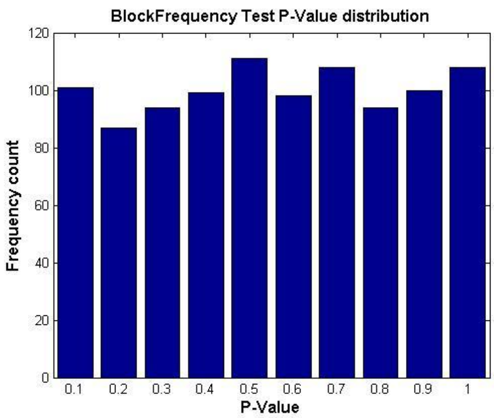

Fig 3: Block Frequency Test

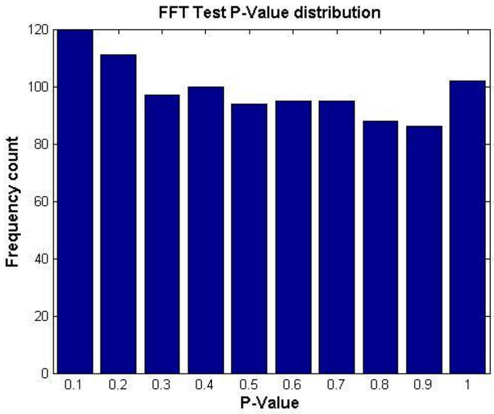

Fig 4: FFT Test

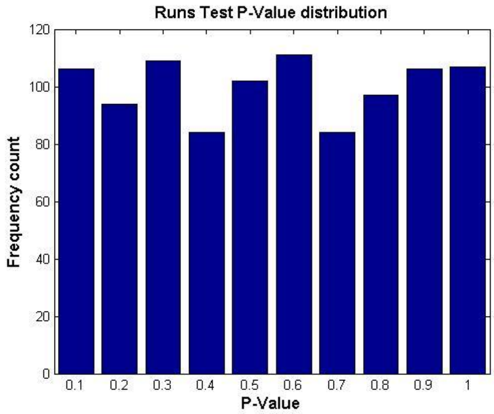

Fig 5: Runs Test

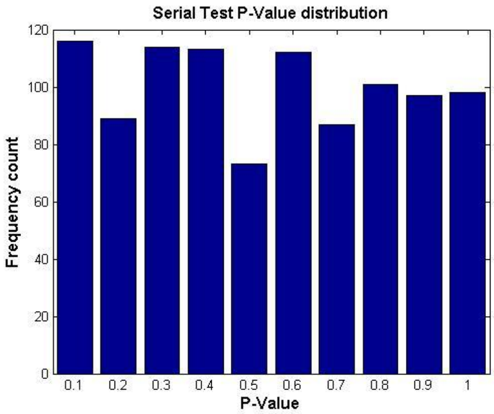

Fig 6: Serial Test

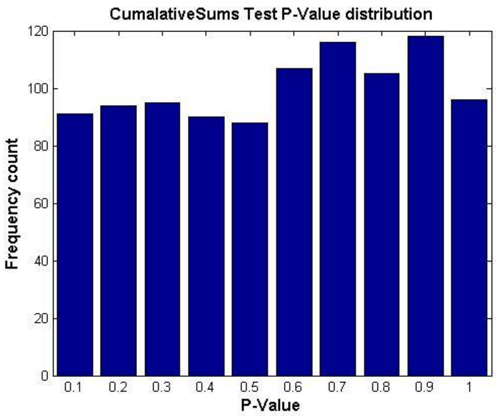

Fig 7: Cumulative Sums Test

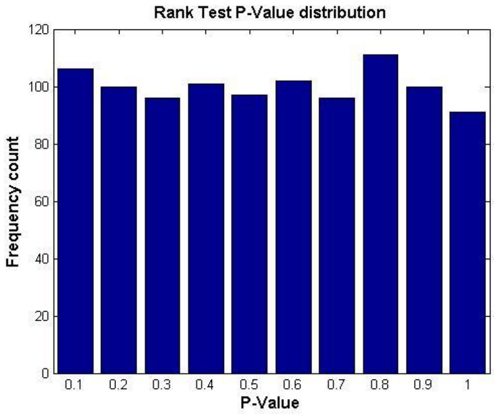

Fig 8: Rank Test

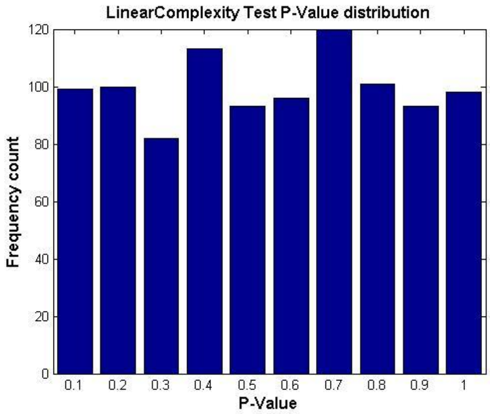

Fig 9: Linear Complexity Test

Uniform-distribution of *P-values* for 1000 number of binary sequences has also been represented by histograms by dividing the complete interval of *P-values* [0, 1] is 10 equal sub-intervals and the *P-values* that lie in each subinterval are plotted. For four of these tests, this distribution has been displayed in Figure 2 to 5 for four of these tests.

It is clear from the Table 2 and the Figure 2 to 5 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 test suite (*sts-2.1.1*), which is provided in the appendix of this manuscript. We have performed the test on some other samples also and these samples pass the tests as well.

5.3 Scatter Plot

Scatter Plots are used toshow uniformity oruniformdistribution of the numbers. The scatter plot of numbers generated for BBS have also been plotted in MATLAB. Themotivation behind this is to shows the distribution graphicallyrather than statistically in probabilistic terms. The figure 6 contains the scatter plot of first 1000 numbers generated by BBS with value of *p*= 98207 and *q*= 101111 and seed is given at random.

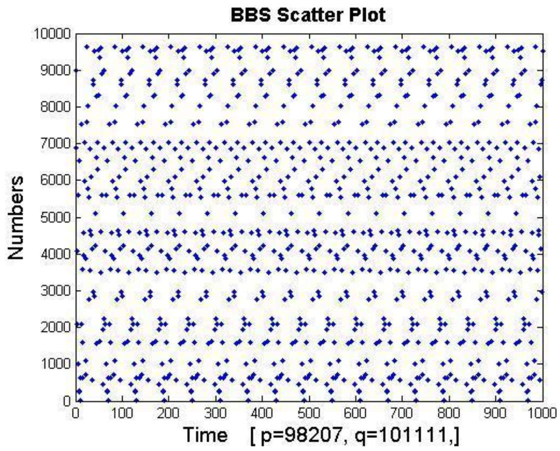

Fig 6: Scatter Plot of BBS

As the scatter plot shows, there exists a correlation between the numbers generated in subsequent iterations, but it is not undesirable as James Reed [13] proved in his paper that cryptographically secure pseudorandom number generators need not to have uniform distribution; they must be unpredictable and provably secure.

6. CONCLUSION

Since it is widely believed that no polynomial time algorithmexist that can guess the next bit, more than the probability of0.5 generated by BBS generators and by our study we conclude that BBS,which is based on Composite quadratic residue is provably secure, is random enough to be used in cryptographic applications when computed with very large numbers. The sequence is generated in BBS by extracting least significant bits b_i from x_i, but if we extract the k least significant bits from each x_i, the security of PRBG still remain intractable as well as fast generation can also be achieved.

7. ACKNOWLEDGMENTS

We would like to thank Banasthali Vidyapith for letting us use their resources, providing support and encouragement. It waspossible to carry out the above stated research only with the help and assistance of the institute.

8. REFERENCES

[1] Landon Curt Noll ,http://www.lavarnd.org/

[2] Blum M. and Micali S.1984. How to generate cryptographically strong sequences of pseudo-random bits. SIAM Journal on Computing 13, pp. 850–864.

[3] Blum L., Blum M.and Shub M. 1986.A simple unpredictablepseudorandom numbergenerator.SIAM JournalonComputing15,pp. 364–383.

[4] FIPS 186. 1994.Digital signaturestandard. Federal Information Processing Standards Publication 186,U.S. Departmentof Commerce/N.I.S.T.,NationalTechnical Information Service, Springfield,Virginia.

[5] Lap-Piu Lee and Kwok-Wo WongJanuary–February 2004. A Random Number Generator Based on Elliptic Curve Operations. Elsevier,Computers & Mathematics with Applications Volume47,Issues 2–3,pp.217–226.

[6] Merkle R. C.Hellman M. 1978.Hiding informationand Signature in TrapdoorKnapsack,IEEETransaction on InformationTheory,vol.24,pp.525-530.

[7] Shamir A. 1983. On the generationofCryptographically StrongPseudorandom Sequences. ACMTransactionson Computer Systems, 1,pp.38–44.

[8] RivestR. Shamir A.Adleman L.1978.AMethod forObtainingDigital Signaturesand Public-Key Cryptosystems.Communications of the ACM21(2), 120–126.

[9] Rukhin A.Soto J. Nechvatal J.Smid M. Barker E. Leigh S. Levenson M. Vangel M. Banks D. Heckert A. Dray J.SanVo 2001 Statisticaltestsuiteforrandom and pseudorandom numbergeneratorsforcryptographic applications.NISTspecial publication800-22.

[10] Andrew Chi-Chih Yao 1982. Theory and Applications of Trapdoor Functions. InProceedingsof the23rdIEEE Symposiumon FoundationsofComputer Science.

[11] Stinson D. R. 2006. Cryptography Theory and Practice.Taylor&Francis Group, 3rd.edition, pp.324-325.

[12] Park S. K. Miller K. W. 1988. RandomNumber Generators: Good onesare hard tofind. Communications ofthe ACM 31,1192-1201.

[13] Reeds J. A. 1977.Cracking'arandomnumber generator. Cryptologia,1(1).

[14] Nowak D. 2009. On Formal Verification of Arithmetic Based Cryptographic Primitives, Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Proceedings, volume 5461 of Lecture Notes in Computer Science, pp. 368-382.

[15] Marsaglia G. 1995. DIEHARD statistical tests. *http://www.stat.fsu.edu/pub/diehard/*,lastaccessed onJuly26,2013.

[16] Knuth D. E. 1998.TheartofComputer Programming:SeminumericalAlgorithms.Addison Wesley,Reading, USA.

[17] Gustafson H. Dawson E. Nielsen L. Caelli W. 1994. A Computer package for measuringthe Strengthof EncryptionAlgorithms,J.Computer Security,vol.13, pp.687-697.

[18] Soto J. 1999. Statisticaltestingofrandom number generators, Proc.of22ndNationalInformation SystemSecurity Conference, retrieved from http://csrc.nist.gov/groups/ST/toolkit/rng/documents/nissc-paper.pdf

[19] Haahr M.Haahr S. Random.org.http://random.org,last accessed onJuly26,2013.

[20] GCC:the GNU Compiler Collection.http://gcc.gnu.org/, lastaccessed onJuly26,2013.

[21] The GNU Multiple PrecisionArithmetic Library. http://gmplib.org/,lastaccessed onJuly26,2013.

APPENDIX

NIST *sts-2.1.1* output file *finalAnalysisReport.txt* for Blum Blum Shub generator:

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <./data/bbs_out>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 93 106  76 101  95 106 101 102 129  91  0.057146    983/1000    Frequency
101  87  94  99 111  98 108  94 100 108  0.837781    988/1000    BlockFrequency
 91  94  95  90  88 107 116 105 118  96  0.307077    983/1000    CumulativeSums
 95  92  95  89  84 119 105 113 108 100  0.268917    983/1000    CumulativeSums
106  94 109  84 102 111  84  97 106 107  0.452173    989/1000    Runs
108 101 105  98 110 102 105  88  88  95  0.801865    983/1000    LongestRun
106 100  96 101  97 102  96 111 100  91  0.970302    987/1000    Rank
132 111  97 100  94  95  95  88  86 102  0.070299    986/1000    FFT
 98 107  95 106 106 100  86 105 103  94  0.900569    989/1000    NonOverlappingTemplate
102  83 101  88  99 117 111 101 109  89  0.310049    990/1000    NonOverlappingTemplate
109  88  99  97 106  97 111 100  94  99  0.884671    989/1000    NonOverlappingTemplate
115 100  93 101  99 100  92  94 117  89  0.548314    985/1000    NonOverlappingTemplate
 97 113 120  83 110  95  92  95  94 101  0.263572    982/1000    NonOverlappingTemplate
107 111  92  98 106  94  90  91  99 112  0.703417    991/1000    NonOverlappingTemplate
 71  98  89  97  86 129  89 121 115 105  0.000870    994/1000    NonOverlappingTemplate
117 116  91 100  94  91  88 106 100  97  0.408275    983/1000    NonOverlappingTemplate
100  86  94 117  91 111 101  93  99 108  0.496351    988/1000    NonOverlappingTemplate
104  98  96  87 104 106 102  92  96 115  0.773405    986/1000    NonOverlappingTemplate
107 106  97  92  98 105 101  86 112  96  0.794391    992/1000    NonOverlappingTemplate
 91 100 123  93 100  95 105  94  88 111  0.342451    993/1000    NonOverlappingTemplate
 98  99  98  96 111  85 110  87  98 118  0.377007    991/1000    NonOverlappingTemplate
 80 123  97  96 105  99 114  98  99  89  0.161703    994/1000    NonOverlappingTemplate
103 105  95  86  93 102 113  87  93 123  0.200115    991/1000    NonOverlappingTemplate
 98 106 100  98 104 107  92  89  94 112  0.856359    985/1000    NonOverlappingTemplate
116 108 104 102  84  86  85 102 118  95  0.133404    989/1000    NonOverlappingTemplate
110  98 105 100  98  90  80  94 116 109  0.345650    988/1000    NonOverlappingTemplate
105 111 105  88  94 103  91 101  84 118  0.332970    990/1000    NonOverlappingTemplate
106 101  97 110  93  90  95 102 104 102  0.944274    992/1000    NonOverlappingTemplate
104  94  98  95 112  87  98 112 100 100  0.796268    993/1000    NonOverlappingTemplate
109 106 112 101  94  85  85 101 105 102  0.556460    988/1000    NonOverlappingTemplate
 98 100 113 100 106  92  95  92 111  93  0.805569    990/1000    NonOverlappingTemplate
104  90  82 105 109 117 100 109  87  97  0.279844    991/1000    NonOverlappingTemplate
 88 123 103  98  91 109  98  91  92 107  0.314544    994/1000    NonOverlappingTemplate
103  99  90 111 106 101  91  94 101 104  0.910091    989/1000    NonOverlappingTemplate
 97  99  99  84 106 104  94 104 104 109  0.861264    990/1000    NonOverlappingTemplate
121 107  91  98  95  98 114  95  84  97  0.282626    991/1000    NonOverlappingTemplate
115 101  96  93  94 106 105  97  82 111  0.492436    987/1000    NonOverlappingTemplate
 97 101 103  96  88  94  96  89 109 127  0.235589    988/1000    NonOverlappingTemplate
 93 101  88  94  99 103 106  90 101 125  0.348869    992/1000    NonOverlappingTemplate
103 122 102 105 101 104  88 100  90  85  0.344048    986/1000    NonOverlappingTemplate
 89 103  89  98 102  96 100 123 100 100  0.530120    988/1000    NonOverlappingTemplate
102  91  95 103  99 115 100 100 102  93  0.912724    995/1000    NonOverlappingTemplate
 92 117 112 108  98  95 102  85  94  97  0.471146    991/1000    NonOverlappingTemplate
 84  99 113  99  94 111 103  99  95 103  0.711601    991/1000    NonOverlappingTemplate
 97  75 105  97  97 115 105 107  99 103  0.361938    989/1000    NonOverlappingTemplate
 94 111 118  97  93  85  95 101  94 112  0.375313    989/1000    NonOverlappingTemplate
 98  98  92 118 109 105  86  98 104  92  0.552383    990/1000    NonOverlappingTemplate
101 104  99  99 109  86  94  96 117  95  0.676615    987/1000    NonOverlappingTemplate
 87 110  79  93  84 110 114 107 100 116  0.063615    991/1000    NonOverlappingTemplate
 92  91 107  96 105  94 108 103 109  95  0.875539    994/1000    NonOverlappingTemplate
 91 102 104  97  95  96 105 107  97 106  0.975012    996/1000    NonOverlappingTemplate
111  90 105  89 105  98 102  94 115  91  0.593478    989/1000    NonOverlappingTemplate
 92 120 107  88 107  94  90  93  99 110  0.357000    991/1000    NonOverlappingTemplate
101 101  97  81 110 102  94 107  96 111  0.639202    990/1000    NonOverlappingTemplate
129 106 100  83  92 102 104  90 100  94  0.127393    991/1000    NonOverlappingTemplate
119  85  90  97 104  86 101 109 109 100  0.296834    987/1000    NonOverlappingTemplate
101 111 113  99 100 104  94 100 101  77  0.461612    991/1000    NonOverlappingTemplate
108  97 100  99 107  82 111  93 118  85  0.233162    990/1000    NonOverlappingTemplate
106 107  93  96 116 109  82  99  90 102  0.422638    988/1000    NonOverlappingTemplate
 87  88 111 105 107  94  94  99 123  92  0.228367    990/1000    NonOverlappingTemplate
 84 123 107  94  90  99 122  94  97  90  0.066882    988/1000    NonOverlappingTemplate
110 114  90  86  95 106 106  87  89 117  0.177628    989/1000    NonOverlappingTemplate
103 112 117 101  86  92 101 110  88  90  0.313041    991/1000    NonOverlappingTemplate
106  94 101  90 115  95 100 104  96  99  0.870856    991/1000    NonOverlappingTemplate
 77 115 117  89 105  91 101 103 119  83  0.022760    993/1000    NonOverlappingTemplate
```

116	99	74	99	128	96	90	101	88	109	0.014550	988/1000	NonOverlappingTemplate
110	103	117	96	103	98	86	107	90	90	0.463512	995/1000	NonOverlappingTemplate
86	117	98	113	87	104	96	113	94	92	0.257004	994/1000	NonOverlappingTemplate
104	96	92	78	108	96	92	106	118	110	0.222480	994/1000	NonOverlappingTemplate
111	98	87	106	110	93	95	95	109	96	0.713641	995/1000	NonOverlappingTemplate
100	108	110	115	96	108	95	90	95	83	0.429923	986/1000	NonOverlappingTemplate
88	109	91	98	102	107	92	102	106	105	0.841226	990/1000	NonOverlappingTemplate
99	115	90	93	117	104	100	102	94	86	0.422638	989/1000	NonOverlappingTemplate
82	101	96	99	111	96	105	98	115	97	0.593478	990/1000	NonOverlappingTemplate
89	86	97	94	93	120	89	101	108	123	0.084037	989/1000	NonOverlappingTemplate
103	103	94	122	102	100	90	93	101	92	0.579021	989/1000	NonOverlappingTemplate
119	86	84	103	91	101	97	109	106	104	0.314544	983/1000	NonOverlappingTemplate
108	106	108	89	91	89	105	103	105	96	0.777265	987/1000	NonOverlappingTemplate
102	115	74	106	102	104	91	92	109	105	0.206629	987/1000	NonOverlappingTemplate
106	91	110	77	123	93	103	103	103	91	0.111389	994/1000	NonOverlappingTemplate
90	90	93	100	108	106	110	90	104	109	0.693142	984/1000	NonOverlappingTemplate
93	104	107	91	86	104	96	105	99	115	0.664168	992/1000	NonOverlappingTemplate
98	107	95	106	107	98	87	105	103	94	0.907419	989/1000	NonOverlappingTemplate
114	97	84	90	110	105	83	101	120	96	0.125200	988/1000	NonOverlappingTemplate
101	92	85	94	110	111	109	95	118	85	0.212184	993/1000	NonOverlappingTemplate
107	83	94	107	102	108	104	110	88	97	0.574903	988/1000	NonOverlappingTemplate
107	118	102	99	88	101	88	81	117	99	0.154629	984/1000	NonOverlappingTemplate
113	94	113	104	103	95	105	97	88	88	0.589341	987/1000	NonOverlappingTemplate
103	87	91	96	92	109	98	116	99	109	0.572847	993/1000	NonOverlappingTemplate
117	101	109	77	109	98	115	78	97	99	0.048093	982/1000	NonOverlappingTemplate
108	126	86	94	91	90	98	102	98	107	0.205531	993/1000	NonOverlappingTemplate
92	102	104	98	92	107	103	96	102	104	0.981940	991/1000	NonOverlappingTemplate
91	112	108	97	111	104	106	83	100	88	0.433590	985/1000	NonOverlappingTemplate
85	102	98	104	104	112	83	111	104	97	0.490483	995/1000	NonOverlappingTemplate
95	102	99	96	103	92	103	112	100	98	0.973055	992/1000	NonOverlappingTemplate
123	92	107	120	91	86	91	97	93	100	0.103138	984/1000	NonOverlappingTemplate
92	95	98	110	96	103	106	108	102	90	0.896345	986/1000	NonOverlappingTemplate
105	115	104	105	80	120	88	107	94	82	0.058243	992/1000	NonOverlappingTemplate
90	98	98	97	109	87	96	100	113	112	0.641284	992/1000	NonOverlappingTemplate
110	109	91	92	104	88	105	120	92	89	0.278461	985/1000	NonOverlappingTemplate
105	108	94	97	114	100	87	86	103	106	0.595549	989/1000	NonOverlappingTemplate
98	105	95	92	106	99	104	106	87	108	0.883171	983/1000	NonOverlappingTemplate
100	114	97	105	97	97	112	100	88	90	0.703417	989/1000	NonOverlappingTemplate
93	97	101	119	98	103	98	79	109	103	0.377007	987/1000	NonOverlappingTemplate
100	103	92	109	98	114	98	94	91	101	0.854708	993/1000	NonOverlappingTemplate
111	103	108	98	107	103	93	98	94	85	0.769527	982/1000	NonOverlappingTemplate
109	105	104	105	98	94	94	110	93	88	0.820143	995/1000	NonOverlappingTemplate
92	104	103	95	96	105	97	99	112	97	0.956729	990/1000	NonOverlappingTemplate
96	98	94	90	89	127	114	89	104	99	0.145326	989/1000	NonOverlappingTemplate
97	101	109	107	94	110	101	73	103	105	0.319084	985/1000	NonOverlappingTemplate
101	86	97	116	106	105	97	97	98	97	0.784927	987/1000	NonOverlappingTemplate
93	82	94	114	91	101	94	95	114	122	0.112708	993/1000	NonOverlappingTemplate
101	115	106	80	93	111	104	97	107	86	0.274341	990/1000	NonOverlappingTemplate
108	101	96	115	89	90	99	104	92	106	0.695200	990/1000	NonOverlappingTemplate
104	93	94	93	100	90	106	112	101	107	0.851383	985/1000	NonOverlappingTemplate
133	108	105	102	97	100	91	98	85	81	0.028625	988/1000	NonOverlappingTemplate
94	103	101	108	90	100	91	104	115	94	0.771469	990/1000	NonOverlappingTemplate
103	112	101	109	112	103	100	80	82	98	0.264901	994/1000	NonOverlappingTemplate
100	90	101	119	113	110	96	90	82	99	0.229559	989/1000	NonOverlappingTemplate
107	112	106	92	87	109	88	85	111	103	0.317565	987/1000	NonOverlappingTemplate
111	95	102	101	98	87	111	88	108	99	0.684890	987/1000	NonOverlappingTemplate
110	103	89	95	94	115	98	97	102	97	0.796268	991/1000	NonOverlappingTemplate
102	122	98	92	100	99	87	100	89	111	0.377007	990/1000	NonOverlappingTemplate
111	87	100	90	107	111	107	91	93	103	0.587274	997/1000	NonOverlappingTemplate
105	103	114	103	94	102	100	95	98	86	0.830808	988/1000	NonOverlappingTemplate
105	104	100	108	82	92	89	101	110	109	0.538182	988/1000	NonOverlappingTemplate
107	97	87	97	115	98	94	85	104	116	0.352107	991/1000	NonOverlappingTemplate
124	119	89	95	88	92	100	100	107	86	0.081510	991/1000	NonOverlappingTemplate
104	108	116	106	90	95	113	90	84	94	0.305599	990/1000	NonOverlappingTemplate
102	91	83	98	105	99	107	113	101	101	0.715679	986/1000	NonOverlappingTemplate
111	103	99	114	100	81	119	96	76	101	0.058612	992/1000	NonOverlappingTemplate
90	111	94	91	102	102	101	100	108	101	0.903338	990/1000	NonOverlappingTemplate
97	123	106	88	102	93	96	81	111	103	0.172816	987/1000	NonOverlappingTemplate
105	106	105	93	107	79	103	96	104	102	0.668321	992/1000	NonOverlappingTemplate
122	106	80	94	86	91	98	100	118	105	0.069863	986/1000	NonOverlappingTemplate
109	107	110	98	91	106	86	99	94	100	0.755819	991/1000	NonOverlappingTemplate
111	105	115	83	109	104	99	113	76	85	0.044508	992/1000	NonOverlappingTemplate
98	93	91	103	102	89	115	98	105	106	0.781106	993/1000	NonOverlappingTemplate
86	110	109	106	108	91	86	90	91	123	0.101311	993/1000	NonOverlappingTemplate
115	103	82	84	104	102	104	105	95	106	0.404728	982/1000	NonOverlappingTemplate
106	115	81	116	97	100	87	90	105	103	0.219006	992/1000	NonOverlappingTemplate
123	108	96	89	107	109	88	95	85	100	0.184549	979/1000 *	NonOverlappingTemplate

93	122	92	113	87	95	109	90	97	102	0.240501	990/1000	NonOverlappingTemplate
97	101	105	103	100	106	105	88	96	99	0.976266	990/1000	NonOverlappingTemplate
112	87	72	121	115	107	103	84	102	97	0.013102	991/1000	NonOverlappingTemplate
108	123	99	96	95	76	106	90	103	104	0.132640	990/1000	NonOverlappingTemplate
100	120	87	116	90	106	103	87	95	96	0.224821	981/1000	NonOverlappingTemplate
86	107	93	99	106	97	91	95	109	117	0.518106	994/1000	NonOverlappingTemplate
100	95	96	106	105	97	92	94	116	99	0.861264	992/1000	NonOverlappingTemplate
90	95	94	106	108	95	94	107	101	110	0.858002	993/1000	NonOverlappingTemplate
95	86	98	109	79	98	104	100	124	107	0.125200	988/1000	NonOverlappingTemplate
102	99	95	112	111	101	99	99	86	96	0.825505	986/1000	NonOverlappingTemplate
89	86	108	118	106	92	94	128	84	95	0.024688	994/1000	NonOverlappingTemplate
97	103	90	91	106	88	110	103	99	113	0.680755	996/1000	NonOverlappingTemplate
99	108	106	89	86	105	118	93	96	100	0.502247	992/1000	NonOverlappingTemplate
92	103	110	90	86	104	96	105	99	115	0.583145	992/1000	NonOverlappingTemplate
122	83	102	114	98	84	102	100	102	93	0.167184	983/1000	OverlappingTemplate
124	101	93	91	106	97	91	106	93	98	0.417219	990/1000	Universal
99	95	103	104	103	93	93	102	88	120	0.630872	990/1000	ApproximateEntropy
66	69	57	55	69	68	60	57	70	49	0.550479	615/620	RandomExcursions
65	60	57	76	63	73	59	59	59	49	0.446255	613/620	RandomExcursions
59	71	56	55	69	69	71	58	51	61	0.540661	613/620	RandomExcursions
66	57	61	75	71	52	74	55	55	54	0.258434	617/620	RandomExcursions
65	65	67	67	58	66	65	51	56	60	0.886546	614/620	RandomExcursions
67	50	69	57	63	54	59	67	72	62	0.623687	613/620	RandomExcursions
63	67	52	55	65	64	66	67	63	58	0.913523	615/620	RandomExcursions
58	67	53	63	55	61	59	65	73	66	0.808301	614/620	RandomExcursions
56	70	62	64	63	54	63	47	73	68	0.446255	612/620	RandomExcursionsVariant
70	50	58	68	56	60	66	57	69	66	0.684024	613/620	RandomExcursionsVariant
58	65	53	70	56	62	72	55	61	68	0.707249	615/620	RandomExcursionsVariant
56	69	58	69	50	63	67	55	63	70	0.637119	611/620	RandomExcursionsVariant
63	67	67	64	54	57	56	61	72	59	0.861473	610/620	RandomExcursionsVariant
62	67	68	50	61	64	61	60	64	63	0.938551	610/620	RandomExcursionsVariant
66	67	49	55	66	74	57	69	71	46	0.145858	613/620	RandomExcursionsVariant
64	60	63	65	53	62	76	60	62	55	0.777948	614/620	RandomExcursionsVariant
49	65	66	58	78	66	56	62	54	66	0.379967	612/620	RandomExcursionsVariant
57	63	52	68	55	64	66	75	66	54	0.560348	615/620	RandomExcursionsVariant
55	56	61	57	77	63	59	76	54	62	0.369073	612/620	RandomExcursionsVariant
48	71	68	58	65	67	44	59	68	72	0.144531	614/620	RandomExcursionsVariant
71	53	70	60	65	59	57	59	61	65	0.858847	617/620	RandomExcursionsVariant
64	62	62	52	67	67	60	62	59	65	0.970348	617/620	RandomExcursionsVariant
61	61	64	58	66	52	72	59	68	59	0.858847	616/620	RandomExcursionsVariant
52	71	52	61	67	78	55	61	68	55	0.258434	617/620	RandomExcursionsVariant
53	64	56	64	64	59	63	66	61	70	0.938551	618/620	RandomExcursionsVariant
55	60	64	58	60	63	61	61	72	66	0.957555	617/620	RandomExcursionsVariant
116	89	114	113	73	112	87	101	97	98	0.035406	986/1000	Serial
107	108	108	97	78	89	123	86	107	97	0.072514	988/1000	Serial
99	100	82	113	93	96	125	101	93	98	0.192724	991/1000	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 980 for a
sample size = 1000 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 606 for a sample size = 620 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```