BLOCK CIPHER AND NON-LINEAR SHIFT REGISTER BASED RANDOM NUMBER GENERATOR QUALITY ANALYSIS

Robertas Smaliukas

October 2015

Technical Report MII-DS-07T-14-3
Abstract

Random Number Generators (RNGs) are an important building block for algorithms and protocols in cryptography. They are paramount in the construction of encryption keys and other cryptographic algorithm parameters. In practice, statistical testing is employed to gather evidence that a generator indeed produces numbers that appear to be random. In this report, random number generator based on three parallel linear feedback shift registers is tested using A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications and possible improvements to increase quality of it’s randomness using non-linear shift registers, adding switchers, adding controllers, combining parallel linear with non-linear shift registers and self shrinking methods are analysed.

Key words: random number generator, hypothesis test, P-value, Linear Feedback Shift Register (LFSR), Non-linear Feedback Shift Register (NFSR).
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1 Linear Feedback Shift Register</td>
<td>4</td>
</tr>
<tr>
<td>2 Random number (binary) sequence tests and results</td>
<td>6</td>
</tr>
<tr>
<td>3 Linear Feedback Shift Register improvement analysis</td>
<td>8</td>
</tr>
<tr>
<td>4 References</td>
<td>13</td>
</tr>
</tbody>
</table>
Introduction

A random number generator is a computational or physical device (or software) designed to generate a sequence of numbers or symbols that lack any pattern. There are two principal methods used to generate random numbers. One measures some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. The other uses computational algorithms that produce long sequences of apparently random results, which are in fact completely determined by a shorter initial value, known as a seed or key. Coin flipping is an example of the first principal while Pseudo Random Number is the example of the second principal. A "random number generator" based solely on deterministic computation cannot be regarded as a "true" random number generator, since its output is inherently predictable. A true random system would have no restriction on the same item appearing two, three or more times in succession or in the sequence of numbers. Whereas Unique Random number generator generates the sequence of numbers in which no one can be duplicated. In the field of cryptography, Random number generators are very useful as it facilitate the ability to run the same sequence of random numbers again by starting from the same random seed. So long as the seed is secret. Sender and receiver can generate the same set of numbers automatically to use as keys. Random number generators have a vital applications in gambling, completely randomized design, statistical sampling, computer simulation, and other areas where producing an unpredictable result is desirable. Many such applications of randomness have led to the development of several different methods for generating random data. Many of these have existed since ancient times; including dice, coin flipping, and the shuffling of playing cards. But those are not sufficient enough to fulfill are requirements so now a day’s some other techniques are also used like: linear congruential generator, middle square method, probability density function, inversion method, acceptance-rejection method, hash function based random number generator, linear feedback shift register method etc.

In computer security, suitable metrics are needed to investigate the degree of randomness for binary sequences produced by cryptographic random number generators (RNGs). Today, researchers are developing new hardware and software based RNGs. However, few standards address statistical analysis techniques that should be employed in practice. This paper will: list advantages and disadvantages of RNG based on feedback shift register, display experimental RNG testing results for all 10 RNGs using “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”.

1 Linear Feedback Shift Register

A Linear Feedback Shift Register is a shift register whose input state is a linear function of its previous state. The only linear functions of single bits are XOR and inverse-XOR; thus it is a shift register whose input bit is driven by the exclusive-or (XOR) of some
bits of the overall shift register value. The L-bit initial value of LFSR is called seed where L is called its length, the stream values produced by the register is completely determined by previous state. It can produce various random sequences by varying the taps. The bit position that affects next state is called tap.

LFSR can also be used as a random number generator. By using the range (R) of random number it can be determined that how many bits (B) will be grouped together to represent a random number by the formula:

\[
B = \log_2(R + 1)
\]

If the range of random number is 0 to 63 then number of bits will be 6 to represent the random number. Generated sequences of random numbers are repeated to generate the required count of random numbers.

The fastest moving era of computer science demands the non repeating random numbers in some applications. At those situations the existing approach can not satisfy the demand, that's why LFSR Based Unique Random Number Generators came in the focus.

The pseudo noise PN generator is responsible for generating perfect random sequence.

**The proposed random number generator**

The proposed generator consists of three maximal length linear feedback shift registers (LFSRs), later referenced just as 3REG generator with thirty one, nineteen and fourteen taps. The feedback functions are chosen primitive to achieve a maximum period for each register [10]. The feedback functions of the LFSRs are:

\[
\text{Feedback functions of the LFSRs are:}
\]

Fig. 1 represents the RNG based on three linear feedback shift registers algorithm, which will be used for testing. The outputs of these LFSRs are connected through XOR gate. The period of the generated sequence is:
The key length of this pseudorandom number generator is \((14+19+31) = 64\) so 64 initial values (0s or 1s) are required. This algorithm can be easily programmed using any modern programming language. Another big advantage is that it works faster than most other RNGs.

2 Random number (binary) sequence tests and results

There are an infinite number of possible statistical tests, each assessing the presence or absence of a “pattern” which, if detected, would indicate that the sequence is nonrandom. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed “complete.” In addition, the results of statistical testing must be interpreted with some care and caution to avoid incorrect conclusions about a specific generator.

The NIST Test Suite is a statistical package consisting of 15 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. These tests focus on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. The 15 tests are:

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.

A statistical test is formulated to test a specific null hypothesis \((H_0)\). The null hypothesis under each test is that the sequence being tested is random. Associated with this null hypothesis is the alternative hypothesis \((H_a)\) states that the sequence is not random. For each applied test, a decision or conclusion is derived that accepts or rejects the null hypothesis, i.e., whether the generator is (or is not) producing random values, based on the sequence that was produced.

Each test is based on a calculated test statistic value, which is a function of the data. If the test statistic value is \(S\) and the critical value is \(t\), then the Type I error probability is \(P(S > t \mid H_0 \text{ is true}) = P(\text{reject } H_0 \mid H_0 \text{ is true})\), and the Type II error probability is \(P(S \leq t \mid H_0 \text{ is false}) = P(\text{accept } H_0 \mid H_0 \text{ is false})\). The test statistic is used to calculate a P-value that summarizes the strength of the evidence against the null hypothesis. For these tests, each P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. If a P-value for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A P-value of zero indicates that the sequence appears to be completely non-random. A significance level \((\alpha)\) can be chosen for the tests. If \(P\text{-value} \geq \alpha\), then the null hypothesis is accepted; i.e., the sequence appears to be random. If \(P\text{-value} < \alpha\), then the null hypothesis is rejected; i.e., the sequence appears to be non-random. The parameter \(\alpha\) denotes the probability of the Type I error. Typically, \(\alpha\) is chosen in the range \([0.001, 0.01]\).

Proposed Three Linear Feedback Shift Register Generator failed the The Linear Complexity Test. All ten 5,000,000 byte sequences produced a P-value equal to zero, meaning the null hypothesis (sequence being tested is random) under this test is rejected. However 3REG generator did pass all the other tests for all 10 experimental sequences.

The following table depicts test-by-test average of 10 P-values for the above reference generators that passed the NIST tests.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>LCG</th>
<th>G-SHA1</th>
<th>BBSG</th>
<th>MSG</th>
<th>3REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5540677</td>
<td>0.6254126</td>
<td>0.5947807</td>
<td>0.5292109</td>
<td>0.3447995</td>
</tr>
</tbody>
</table>
The constant 3REG failure during the linear complexity test suggests that this generator might be easily hacked based on the output bytes and recreated by an intruder who doesn't have the starting seed 64-bit combination. Even very long sequences generated by 3REG do not pass the linear complexity test and P-values for this particular test are equal 0, meaning that this type of generator does not qualify as secure cryptographic generator.

The other results put BBSG at the top of PRNGs with P-value average of: 0.484205887 closely followed by G-SHA1 and LCG, while MSG trails having significantly lower of average 15 test P-value. When it comes to good quality of randomness an experiment using very long sequence approach shows that BBSG, G-SHA1 and LCG are the best choice.

## 3 Linear Feedback Shift Register improvement analysis

Non-Linear Feedback Shift Registers (NLFSRs) have been proposed as an alternative to Linear Feedback Shift Registers (LFSRs) for generating pseudo-random sequences for stream ciphers. In an (n,k)-NLFSR, the feedback can be taken from any of the n bits, and the next state functions can be any Boolean function of up to k variables. Our motivation for considering this type NLFSRs is that their Galois configuration makes it possible to compute each next state function in parallel, thus increasing the speed of output sequence generation. Thus, for stream cipher application where the encryption speed is important, (n,k)-NLFSRs may be a better alternative than the traditional Fibonacci ones. After deriving a number of properties of (n,k)-NLFSRs it is possible to
demonstrate that they are capable of generating output sequences with good statistical properties which cannot be generated by the Fibonacci type of NLFSRs. Second, it is shown that the period of the output sequence of an \((n,k)\)-NLFSR is not necessarily equal to the length of the largest cycle of its states. We also present an algorithm for estimating the length of cycles of states of \((n,k)\)-NLFSRs which uses Binary Decision Diagrams for representing the set of states and the transition relation on this set.

For example We could improve proposed LSFR like this:

In cryptography, the shrinking generator is a form of pseudorandom number generator intended to be used in a stream cipher. It was published in Crypto 1993 by Don Coppersmith, Hugo Krawczyk, and Yishay Mansour.

The shrinking generator uses two linear feedback shift registers. One, called the A sequence, generates output bits, while the other, called the S sequence, controls their output. Both A and S are clocked; if the S bit is 1, then the A bit is output; if the S bit is 0, the A bit is discarded, nothing is output, and we clock the registers again. This has the disadvantage that the generator's output rate varies irregularly, and in a way that hints at the state of S; this problem can be overcome by buffering the output.

Despite this simplicity, there are currently no known attacks better than exhaustive search when the feedback polynomials are secret. If the feedback polynomials are known, however, the best known attack requires less than \(A \times S\) bits of output.
The output of the selection rule does not have to be binary, as shown below.

A self-shrinking generator is a pseudorandom generator, which is based on the shrinking generator concept. Variants of the self-shrinking generator based on a linear feedback shift register (LFSR) are studied for use in cryptography.
The ideas of self-shrinking and shrinking generators could also help especially when it comes to the test Linear Complexity test, that is failed every time so far.

During the analysis it was found that there are many other ways to experiment with LSFR algorithm, for example:

Windmill generator, where LSFR are not parallel:

Grain where NSFR and LSFR are used together:
Implementation of control clock:
4 References