Robust and Cryptographically Secure Pseudo-Random Bit Generation

Mpho Tjabane (110060)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 14, 2003

Keywords: Randomness, Bit generation, Concurrency, Threads.

1 Introduction

The security of cryptographic systems often depends on the generation of sequences of unpredictable numbers. A distinct requirement that is necessary but not sufficient for such a sequence is *randomness*. Two criteria that are used to validate that a sequence is random are a *uniform distribution* and *statistical independence*. The first criteria means that the frequency of occurence of each of the numbers in the sequence should be approximately the same. The second criteria means that values in the sequence should not be correlated. A number of statistical tests can be carried out on numbers of a sequence to determine if they match a particular distribution; however, there are no such tests for determining independence. Statistical independence in a sequence implies the unpredictability of its elements.
In software we often want to generate random sequences using deterministic algorithms. Deterministic here means that a machine will give the same output *every time* it is fed the same input. As deterministic machines, computers *cannot* generate random data without somehow computing it. This situation is very different from randomness generated by natural phenomena. Because a computer cannot generate "true" randomness, the data which it generates is called pseudo-random.
The focus of this research is designing a determistic algorithm that can produce pseudo-random output according to a number of statistical tests designed for this purpose. Statistics of such tests reveal the structure or a lack thereof within data items.

2 Research Objectives

A number of different pseudo-random number generators have been designed over the years. Most, if not all, of these algorithms were designed for implementation on serial (uniprocessor) computers. Today there exist multiprocessor computers and computers connected by LANs (local area networks) . The main aim of this research is to show how algorithm design has to evolve in order to take advantage of available technologies. In

particular, the research will:
♠ Reveal the design of random bit generator algorithms that are computationally infeasible to predict and have high bit throughput.
♠ Investigate efficient ways (of implementing such algorithms) that fully utilize present day hardware specifications, e.g. fully exploiting entire CPU registers.
♠ Perform comparisons of the cryptographic strength of the algorithms against that of their implementations.
♠ Investigate how parallelism in the algorithms can facilitate concurrent implementations on uniprocessor computers.
♠ Reveal limitations in implementations of the POSIX thread standard that complicate the task of scheduling multiple threads in cases where a parallel algorithm reduces to a parallel producer-consumer problem.

3 The TM3w Bit Generator

For purposes of reference, the algorithm constructed is named TM3w.[2] This bit generator is based on Fibonacci recurrences of the form

$$X_n = X_{n-r_i} + X_{n-s_i} \ mod \ 2^w \tag{1}$$

where $\{X_0, \ldots, X_{r_i}\}$ are initializing values, and w is a computer word length. The values r_i and s_i are chosen such that $x^{r_i} + x^{s_i} + 1 \ mod \ 2$ is a primitive polynomial, and $\forall \ i \neq j \neq k$, $gcd(r_i, r_j) = gcd(r_j, r_k) = 1$, and $r_i > r_j > r_k$. This ordering is necessary for the initialization scheme. Generators defined by equation (1) have a period $p = 2^{(w-1)}(2^{r_i} - 1)$. The random word generator constructed is driven by three additive generators as follows

$$A_n = (A_{n-47} + A_{n-5}) \ mod \ 2^{32}, \quad n \geq 47 \tag{2}$$

where

$$x^{47} + x^5 + 1 \ (mod \ 2) \tag{3}$$

is a defining primitive polynomial. The second generator is

$$B_n = (B_{n-41} + B_{n-3}) \ mod \ 2^{32}, \quad n \geq 41 \tag{4}$$

with a primitive polynomial

$$x^{41} + x^3 + 1 \ (mod \ 2) \tag{5}$$

The third and last generator is

$$C_n = (C_{n-35} + C_{n-2}) \ mod \ 2^{32}, \quad n \geq 35 \tag{6}$$

with a defining primitive polynomial

$$x^{35} + x^2 + 1 \ (mod \ 2) \tag{7}$$

[2] The Author's initials. 3w alludes to the fact that on a computer with a wordlength of w, the algorithm output is $3 \times w$ bits.

The initial states of the three generators are arrays containing 32-bit words. These arrays are derived from the initial key provided by the user and differ from each other by a bit rotation and omission of array elements. This key is as long as the initial state of the generator with the highest power of its primitive polynomial. For instance in this case, powers of primitive polynomials in equations (4), (6), and (8) are $\{47, 5\}, \{41, 3\}$ and $\{35, 2\}$ respectively, and the binary operation is addition modulo 2^{32}. The initial states are then derived as follows :

$$key = \{k_0, k_1, \ldots, k_{46}\} \tag{8}$$

where $\forall i = 0 : 46,\ k_i$ is a 32-bit array element. The initial state for the first generator is created as follows, first

$$\{k'_0, k'_1, \ldots, k'_{46}\} = \{k_0, k_1, \ldots, k_{46}\} <<< 20 \tag{9}$$

the value 20 is obtained from $(47 + 5) - 32$. Here, the symbol ¡¡¡ denotes denotes circular bit rotation. Equation (8) shows bit rotation of an entire array. After this rotation the initial state of generator A is obtained by the assignment

$$\{A_0, A_1, \ldots, A_{46}\} = \{k'_0, k'_0 \uplus k'_1, \ldots, \uplus_{i=0}^{46} k'_i\} \tag{10}$$

where $\uplus$ denotes *integer addition* modulo 2^{32}. The initial state of generator B is obtained from that of the first generator by the following process, first

$$\{A'_0, A'_1, \ldots, A'_{46}\} = \{A_0, A_1, \ldots, A_{46}\} <<< 12 \tag{11}$$

The value 12 is $(41 + 3) - 32$. The array in equation (9) is then reduced by discarding elements A'_{41} to A'_{46} and the following assignment made

$$\{B_0, B_1, \ldots, B_{40}\} = \{A'_0, A'_0 \uplus A'_1, \ldots, \uplus_{j=0}^{40} A'_j\} \tag{12}$$

In equations (8), (10) and below, bit rotation simply results in a different bit pattern. For generator C, the process is

$$\{B'_0, B'_1, \ldots, B'_{40}\} = \{B_0, B_1, \ldots, B_{40}\} <<< 5 \tag{13}$$

where the value 5 is $(35 + 2) - 32$. Array elements B'_{35} to B'_{40} are then discarded to obtain the initial state by,

$$\{C_0, C_1, \ldots, C_{34}\} = \{B'_0, B'_0 \uplus B'_1, \ldots, \uplus_{k=0}^{34} B'_k\} \tag{14}$$

The use of *incremental* modular addition, bit rotation, and bit omission adds to the difficulty of predicting this generator.

3.1 Random word generation

The process of word generation consists of iteration using equations (1), (3), and (5) and the initial states (9), (11), and (13) above. In addition, in order to attain confusion, the outputs from the equations above are further passed through equations (14) to (16) below.Let $< A_k >$ be the sequence of words by generator A, $< B_l >$ be the sequence of words by generator B, and $< C_m >$ be the sequence of words by generator C, for all k,

l, and m. Then *three* 32-bit output words $W_{j,1}$, $W_{j,2}$, and $W_{j,3}$ are generated using the equations,

$$W_{j,1} = ((A_k \odot B_l) \uplus A_{k+1}) \oplus C_m \tag{15}$$

$$W_{j,2} = ((B_l \odot C_m) \uplus B_{l+1}) \oplus A_k \tag{16}$$

$$W_{j,3} = ((C_m \odot A_k) \uplus C_{m+1}) \oplus B_l \tag{17}$$

where $\odot$, $\uplus$, and $\oplus$ denote integer multiplication modulo 2^{32}, integer addition modulo 2^{32}, and *exclusive*-or respectively. Lower bounds on the indexing variables j, k, l, and m are $j \geq 0$, $k \geq 46$, $l \geq 40$, and $m \geq 34$ respectively. The following ordered pairs are required to initialize the above combination generators; $\{A_{46}, A_{47}\}$, $\{B_{40}, B_{41}\}$, and $\{C_{34}, C_{35}\}$. Exhaustive search estimates show that the work factor for predicting both the intermediate and the final generators is prohibitively huge.

4 Concurrency

In the implementation of TM3w, equations (2), (4), and (6) together with equations (14), (15), and (16) define a producer-consumer problem with three producers and three consumers. Each producer gets input that is the output of each of the three producers. When implementing this configuration on a *uniprocessor*, all the players involved must share two computation resources: CPU time and memory. This gives rise to a synchronization problem in order to ensure that all get a chance to run. Each producer or consumer is implemented as a thread[3] which then access resources according to some controlling mechanisms. The controlling mechanisms provided by the POSIX standard are mutexes (mutual exclusions) and condition variables. These mechanisms allow manipulation of shared data by threads without interference from other threads.

5 Conclusions

From the work carried out, what emerges is that the proposed random bit generator is efficient in terms of high throughput, and it is cryptographically secure in terms of the work factor required to predict its output.
What also emerged is that the pursuit for concurrency on a uniprocessor machine results in an implementation that is *not* portable. This is because application programming interfaces that provide the means to concurrency are tied to particular operating systems. Although there exist four thread APIs, only two (Solaris and POSIX) have significant similarities. Even so, porting from one to the other requires a good knowledge of both.
The upsence of a 64-bit data type in the C programming language prevents single precision computations on 64-bit processors. The type **unsigned long** which is 32 bits long results in a 96-bit output. If a 64-bit data type was defined, the output would be 192 bits. A big difference.

[3] A thread is defined as a single flow of control within a process.