Nick Alexander

April 20, 2015

Random Number Generation

The purpose of this assignment was to research, understand, and implement a pseudo-random number generator. Because a computer is incapable of true randomness, it is the responsibility of cryptographers and other experts to create methods of mathematically calculating an unpredictable number. The problem, however, is that a number can still be predicted if the method is known. In my generator, I use a method known as the multiply with carry method for generating pseudo-random numbers. While the method is not perfect, it has a period – or number of generations before a repeat is expected – of around 2^{64}. The method passes, consistently, 8 of the 15 statistical tests provided by the National Institute of Standards and Technology.

While the multiply with carry method may seem simple, the fact that both the initial seed and multiplier are incredibly large numbers makes it much more complex. On top of the large numbers, the bit shift and bitwise AND operation work on the bit level which can be difficult to calculate by hand. Lastly, the algorithm is further strengthened by the maximum size of an integer data type. Because Java has a wrap-around when the integer value is too large (i.e. too large of an integer wraps back to a negative value and continues to count back towards positive), the calculated value is mangled. The algorithm is very safe to anyone who cannot read the source code. If the code is open-source, the algorithm is much less secure since the math is trivial for a computer.

In terms of statistical testing, my algorithm was used to generate a 512 bit string as well as a 100 kilobit string which were ran through the NIST statistical test suite. In both instances, the Frequency, Block Frequency, Cumulative Sums, Runs, Longest Run, FFT, Approximate Entropy, and Serial tests all passed. The only difference between the two was that the Linear Complexity test passed in the small sample but failed in the large sample. Overall, my algorithm consistently passed 8 out of the 15 tests.

Another key element to the algorithm is that the "cheating" version and the legitimate version are the same. My argument is that anyone who knows the algorithm is able to calculate the value by hand and will be able to easily predict the next value. Of course, this becomes infinitely more difficult if you do not know the algorithm. However, since the method relies both on the system time in nanoseconds and the system time in milliseconds, trying to calculate the next value because more difficult because both time values change incredibly quickly. One would have to be able to perfectly extract the time values the instant that the generator does to properly calculate the next value with certainty.