COS 490 — "Real" Random Sequence Generator

Samuel Barton

As a part of this independent study into cryptography I decided to try to make a random number generator. This ended up being reasonably challenging, but quite rewarding. In the end what I have created is a hybrid random number generator which creates what should be truly random numbers in practice, however in theory they should probably not be considered "real" random numbers. As I built the random number generator I considered several possible seeds of randomness:

- CPU clock slippage
- Temperature fluctuation
- ambient noise
- keyboard key latency
- radioactive decay

Now these are all reasonable sources of randomness, but some, namely the radioactive decay, are rather hard to setup, and others, like the keyboard key latency, are very slow for generaitng long random sequences. Given these limitations I only considered two of these options, CPU clock slippage, and temparature fluctuation. The ambient noise source, while perfectly reasonable, required having a microphone handy, and since I didn't I went for temparature as my analog source.

1 Digital Source — CPU Clock Slippage

The usleep(n) command exists on every POSIX system, and enables halting a program for n microseconds of CPU clock time; however, more than one process runns on a machine at a time, and there is overhead associated with every call the operating system must handle. Thus, when one calls usleep(10) our program will hault for at least, but never exactly 10 microseconds. Since there is no way to predetermine the exact number of microseconds the program will hault, as it is a funciton of the overhead associated with handling the usleep() system call and the current CPU load, and the CPU scheduler, we have ourselves a source of randomness.

To generatea a stream of random bits from the random source we have we can simply save the current time elapsed to sleep n microseconds, and then compare the last time elapsed with the current time. If they are the same, we return 0, and if they differ we return 1. This gives us a random distribution of zeros and ones which has a slight bias towards zero and a tendency to generate clumps of the same value. Given that the timeout used was 10 microseconds we are able to generate 100,000 random bits per second, or 100KB/s.

2 Analog Source — Temperature fluctuation

Temperature, being the amount of kinetic energy of air molecules in a given area, flucuates randomly given the random number of air molecules in a given area, and the random distribution of kinetic energies among the particles. Thus, if we can accurately measure temperature over time, we have a source of randomenss. The difficulty lies in measuring the temperature accurately enough to see the flucuations. I had access to an Arduino and the assoicaited temperature sensor it came with. I was able to build a sensor circuit which could meausre the temperature accurately enough to generate a random sequence, but it lacked the sensitivity to be very useful given that most of the time it read the temperature as staying constant as it only saw the large changes in temperature. Beyond this, the sequence took too long to generate given the low baudrate of the connection to the sensor. So sadly, after figuring out how to access the Arduino's serial console and retrieve output, this source had to be abandoned as well.

3 Final algorithm

In the end I wound up using the time slippage of the CPU as my real random source. As was mentioned before, the sequences generated were random, but had large groupings of 1's and 0's, which will be a problem going forward. To aliviate this issue, I called upon the pseudorandom number generator POSIX provides and took the XOR of that and the time slippage to get the sequence. So each bit in the sequence is the exclusive or of the CPU time slip and the value returned by calling the random() method and masking it to only give 0 or 1. This sequence is random, and has no large groups of zeros or ones. It has a slight bias towards 1, 0.14% to be exact.

4 Conclusion

This project was very fun and interesting to complete, and I now have the ability to generate a random sequence of bits for any other project which might need them. The beauty of this alrogithm is that if I want to generate integers I just have to take 32 values from the sequence and voila a random integer is born.