A Survey of Random Numbers in Cryptography

EE595: Course Project - Progress

Prepared by Paul Adams

Introduction

For my course project I have proposed to survey random number generation in cryptography. While I have found the course material very interesting and applicable from a personal perspective, it is focused less on technical skills. For that reason, I have endeavored to emulate a lab component by biasing my project toward the algorithmic and software techniques associated with random number generators (RNGs). However, the math is quite challenging and the field is wide-ranging. I have sought for a unifying theme to this project, but have found that challenging. Below, I will outline my efforts to this point.

Algorithms/Software for Evaluating Quality of RNGs

My initial approach was to implement or utilize software that would evaluate the quality of a random number stream. I would then use these quality measures to evaluate various algorithms. However, I have learned that there is no all-encompassing formula or algorithm that exhaustively evaluates a random stream. In general, random number testing software will run a suite of tests that verify the stream does not have any known bad qualities. That is, the tests demonstrate necessary but insufficient qualities of RNGs. Fundamentally, the nature of a random stream is that it's elements defy correlation.

Entropy Estimation

I read the paper by Chakrabarti et al. on the subject of estimating entropy. I had made the assumption that there was a natural link to random streams and that by implementing the algorithm I would be able to quantify the entropy of a stream and thereby estimate it's quality. The paper references a c++ implementation of the algorithm. I downloaded, spent the time getting it to compile, but struggled to interpret the results. I then came to appreciate that the algorithm had application to network analyzers and not random number streams. In summary, I have abandoned this approach as not relevant to the paper

Software for Testing Random Number Generators

TestU01

TestU01 [http://simul.iro.umontreal.ca/testu01/tu01.html] is a suite of software implemented in c whose aim is to test RNGs for flaws. I downloaded and began to examine and found that the authors expect the user to customize their tests with the provided library of tools. I ran some exploratory tests on some binary streams. This set of tools is descended from the pioneering work of Donald Knuth in *The Art of Computer Programming.*

randomtests

randomtests is the only pure python library for testing RNGs that I was able to find. However, it's pedigree is unknown unlike TestU01 which has widespread support. Additionally, tests for RNGs tend to be computationally intensive, which begs for a c implementation. Ideally, I would have found a python API to TestU01, but there doesn't appear to be one.

Evaluation of Specific Random Sources

RANDU

To this point, I have only evaluated the text book example of a bad RNG. I have written code to implement and display some illustrative plots.

NIST Randomness Beacon

Initially, I had anticipated using the output from [https://beacon.nist.gov/home], concatenating multiple records to emulate a hardware RNG stream. The website generates a stream of 512 bits once every 60 seconds using quantum entanglement observation as the source of entropy. While this works, it misses the security aspects of random streams. The stream could never be used for cryptographical applications as it's state is fundamentally public. Rather, it can be used as a secure time-stamping mechanism since it is infeasible to know the bits corresponding to a given time-stamp until they have been released.

I have written code that retrieves the random records and stores them to a database and also attempted to evaluate its quality using TestU01.

Example of Bad Random Source

I would like to implement an example of how a random number source can lead to a vulnerability in a cryptosystem. One idea is to implement a textbook version of RSA, which I have already coded, and demonstrate public/private key collision.