CSCI474 - Course Project Proposal

Thomas Applegate
Colorado School of Mines
Golden, CO, USA
tapplegate@mines.edu

Kaelyn Boutin
Colorado School of Mines
Golden, CO, USA
kvboutin@mines.edu

Gabrielle Hadi
Colorado School of Mines
Golden, CO, USA
ghadi@mines.edu

Addison Hart
Colorado School of Mines
Golden, CO, USA
addisonhart@mines.edu

Et Griffin
Colorado School of Mines
Golden, CO, USA
egriffin@mines.edu

Isabelle Neckel
Colorado School of Mines
Golden, CO, USA
ineckel@mines.edu

Abstract—The National Institute of Standards and Technology, NIST, defines fifteen separate tests for the randomness and unpredictability of random numbers nistbook. True Random, Deskewed True Random and Pseudorandom are the three classes of random numbers that are going to be compared using the tests outlined by NIST. We plan to use a Python implementation that will provide us the P-values to compare and contrast to determine the random numbers that are closest to being truly random. The generation of the random numbers will also be considered when discussing the choice in relation to cryptography.

Index Terms—random numbers, pseudorandomness, NIST, python

I. Introduction

Random number and bit generation is used heavily in cryptography and security applications, but it is difficult to get truly random numbers. Effective randomness is neccesary to ensure security in cryptographic applications, as cryptoanalytic attacks exist which can leverage patterns in encrypted data to decrypt without the neccesary credentials. For example, a team led by Nadia Heninger found that poorly seeded pseudo random number generation, or PRNG, could be used at scale to derive RSA private keys heninger2012mining.

The solution is not as simple as exclusive usage of true random numbers. Because the generation of true random numbers, or TRNG, is computationally expensive and requires a source of entropy, which is limited, TRNGs cannot be used for all the places in cryptographic algorithms where randomness is neccesary, like the usage of nonces in OFB or initialization vectors in CFB, especially if large amounts of data is encrypted or the software is running on constrained devices. TRNGs may also be vulnerable to adversarial modification of the environment by the attacker, like changing the temperature of a device if the source of entropy is a temperature reading barak2003true. Additionally, an attacker may be able to leverage the loss in randomness as soon as the entropy pool empties, which can happen extremely fast if it is the only source of randomness used. Because of this the standard today is to utilize TRNGs only for the generation of seeds for PRNGs 8276259.

The SP 800 90 was created to provide guidelines for the generation of pseudo random numbers for cryptographic use. There are 3 parts to the SP 800 90 series: 90A talks about mechanisms for the generation of random bits using deterministic methods, 90B discusses how valid certain ideas of randomness and entropy are, and finally 90C specifically talks about construction and implementation of random bit generators. Random Bit Generation must be checked for randomness, so a total of fifteen statistical tests were developed and evaluated in order to determine the validity of a generator's statistical randomness. The fifteen tests are as follows

1) Frequency (monobits) tests: This test determines whether the number of ones and zeros generated in a sequence are approximately the same as should be generated for a true random sequence.
2) Test for Frequency Within a Block: This test tries to determine if the frequency of ones in a M-bit block is approximately half the size of the block.
3) Runs Test: This test determines whether switching between substrings is too fast or slow.

4) Test For Longest Run of Ones in a Block: This test determines if the longest runs of ones in a block is similar to the longest run in a truly random generation.
5) Random Binary Matrix Rank Test: This test checks for linear dependence in a fixed length substring.
6) Discrete Fourier Transform (Spectral) Test: This test detects periodic features that could indicate the lack of randomness.
7) Non-Overlapping (Aperiodic) Template Matching Test: This test is used to reject sequence that show too many occurrences of a given non-periodic pattern (similar to the zero's run but allows for statistical independence amongst tests).
8) Overlapping (Periodic) Template Matching Test: This test is used to reject sequence that show changes from the expected number of runs of ones of a given length.
9) Maurer's Universal Statistical Test: This test detects whether or not a sequence can be compressed without loss of information, a overly compressible sequence is considered non-random.
10) Linear Complexity Test: This test determines if a sequence is complex enough to be considered random.
11) Serial Test: The test determines the number of occurrences of 2m m-bit overlapping patterns is what should be expected for a random sequence.
12) Approximate Entropy Test: This test compares the frequency of overlapping blocks of conservative lengths against what is expected for a random sequence.
13) Cumulative Sum (Cusum) Test: This test determines if the cumulative sum of partial sequence in the test sequences are too large or too small relative to random sequences.
14) Random Excursions Test: This test is used to determine if a number of visits to a state within a random walk except what is expected for a random sequence.
15) Random Excursions Variant Test: This test detects deviation from the expected number of occurrences of various states in a random walk.

There are multiple random generators that pass these tests, but how much do these random generators pass by. Why is that important or interesting for randomness. These are all interesting questions that could be researched by trying these fifteen tests are a few generators and reading into the results.

The goal of this research is to compare and contrast different random generators using these fifteen tests. Our research will compare True Random, Deskewed True Random, and Pseudorandom, to see how these classes stack up against one another and compare the time cost of each.

II. Methods

To test the randomness of various algorithms, we will use the NIST randomness tests as implemented in Python rtestsuite. The tests require a binary string of indeterminate length as the input. Each of the fifteen tests will output the P-value, as well as the result of whether the P-value means that the data can be considered truly random or not. For each of the algorithms that we test, we will use many samples of equal length to compare their randomness to each other.

III. Expected Results

We wish to use this work to prove what algorithms in each class, whether True Random, Deskewed True Random or Pseudorandom, consistently perform the best in the NIST randomness tests and are therefore relevant in designing cryptographically secure algorithms, as well as what the strengths and weaknesses of the different classes are. We also intend to prove why specific classes are better suited for certain cryptographic tasks than others. We also intend to show why some algorithms necessitate complexity, like hardware input or SHA hashing, and which algorithms are unnecessarily computationally intensive or complicated for the quality of their output.

IV. Conclusion

NIST provides 15 tests to compare and contrast random number generators. These tests promote different qualities of ideal random numbers. The team plans to use python libraries for the NIST tests, and various pseudorandom and deskewed true random number generating algorithms to compare them to true random numbers. We will summarize the results of these comparisons, and then advise on the generating algorithms suitable for cryptographic use.