Simulating True Randomness

Techniques, Applications, and Limitations

Noah Johnson

INTRODUCTION

Computers are intentionally logical in that every operation they carry out is based on processing information stored as bit patterns in memory. However, in applications such as cryptography, simulation, sample collection, and gaming, we desire randomness which is usually expressed with the generation of random numbers. Humans cannot choose numbers randomly either since their conceptions of randomness rely on uniqueness and/or sentimental value of chosen numbers. Hence, algorithms which simulate the purest form of randomness attainable must be studied.

WHAT'S RANDOMNESS?

Randomness is best understood through two components:

1. The numbers chosen must be **uniformly distributed** such that each number has an **equal probability** of being chosen.
2. The numbers must be chosen **independently** of each other; all patterns must be **unpredictable.**

PSEUDORANDOMNESS

x$_0$:	1234	01522756
x$_1$:	5227	27321529
x$_2$:	3215	10336225
x$_3$:	3362	11303044
	...	

Figure 1: Middle-squares method

The first attempt at computational pseudorandomness came from von Neumann's **middle-squares method**. Essentially, taking some four-digit input seed, square it, add 0s from the left to ensure the result is 8 digits, and repeat the process with the 4 middle digits. When the seed gets sufficiently small, however, the seed converges to 0 as enough iterations are performed.

$$x_n = ax_{n-1} + c \bmod m$$

$$a = 13, c = 5, m = 22, x_0 = 7$$

$$x_1 = (13x_0 + 5) \bmod 22$$
$$x_1 = (13(7) + 5) \bmod 22$$
$$x_1 = 96 \bmod 22 = 8$$

$$x_2 = (13x_1 + 5) \bmod 22$$
$$x_2 = 109 \bmod 22 = 21$$

$$x_3 = (13x_2 + 5) \bmod 22$$
$$x_3 = 278 \bmod 22 = 14$$

...

Figure 2: Linear congruential generator (LCG) algorithm

As computational power improved, a more complex pseudo-RNG (PRNG) was created, and it was subsequently used in the implementation of Java, C++, PHP, and SQL. This **linear congruential generator** uses modular arithmetic on a linear transformation from the last value. It requires intelligent parameter choices so that given a large enough modular divisor m, the period is sufficiently long that independence is assured.

PSEUDORANDOMNESS / CONCERNS

- Let x_0 be the seed
- $x_{1,1} = x_0 \text{ XOR } (x_0 \ll 30)$
- $x_{1,2} = 6C078965_{16} x_{1,1} = 1812433253 x_{1,1}$
- $x_{1,3} = x_{1,2} + 1$ (1 represents current dim. index)
- $x_{1,4} = x_{1,3} \text{ XOR } (x_{1,3} \gg 11)$
- $x_{1,5} = x_{1,4} \text{ XOR } \left((x_{1,4} \ll 7) \text{ AND } 9D2C5680_{16}\right)$
- $x_{1,6} = x_{1,5} \text{ XOR } \left((x_{1,5} \ll 15) \text{ AND } EFC60000_{16}\right)$
- $x_1 = x_{1,6} \text{ XOR } (x_{1,6} \gg 18)$

Figure 3: Mersenne Twister algorithm

In languages such as Python, Lisp, and R, an even more complex PRNG called the **Mersenne Twister** is the default random number generator. Rather than using mathematical techniques as the LCG did, it performs multiple bitwise operations, such as XOR, AND, and left and right shifts, in sequence with 32-bit values. Like the LCG, it also requires intelligent parameters in the AND coefficients and the number of bits by which to shift.

It is very easy to choose imperfect parameters for PRNGs, regardless of how complex the underlying algorithm is. Further, if attackers determine the state of the generator, it becomes very easy to determine future sequences of numbers, which makes RSA key encryption extremely vulnerable. This requires **cryptographically secure PRNGs** which, even if the state and some output value are known, past and future sequence values cannot be reverse-engineered because the next bit of any given random bit string cannot be determined in such a generator.

PHYSICAL RNGs

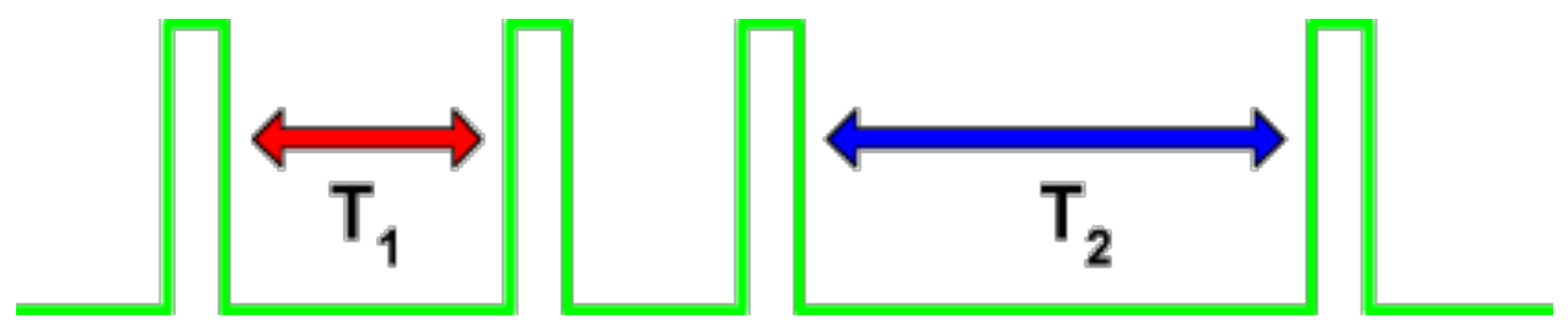

Figure 4: Bit determination in HotBits

Physical random number generators leverage the entropy which is apparent in many natural phenomena. The one with the most entropy that we can measure is the quantum mechanics dictating radioactive decay. A website called **HotBit**s hosts a computer with a Geiger counter measuring the amount of time a particular of radioactive isotope Cs-137 decays. Their algorithm involves choosing two pairs of particles and determining how much longer the second atom of the pair decays compared to the first atom. If this length of time is greater for the first pair, the bit is chosen to be 1; else, the bit is chosen to be 0. As humans, our conception of science makes it impossible to determine which pair will have the larger decay interval for any two arbitrary pairs.

The common random number generator Internet service **RANDOM.ORG** also uses physical phenomena, namely electrical energy from the atmosphere as a result of lightning strikes. From multiple 8-bit signals detected by specialized radio broadcast towers, all but one bit of the first signal is discarded, and the rest remain as skew correction is performed. If a transition occurs between a pair of bits (that is, the values are different), only the first bit is kept; and if the pair is made up of the same value, then both bits are discarded. Despite the clear inefficiencies of losing 75% of the input bit string which already has high entropy, the numbers generated by RANDOM.ORG satisfy the NIST suite of randomness validation tests and is suitable for use in a gaming context.

RANDOMNESS EXTRACTORS

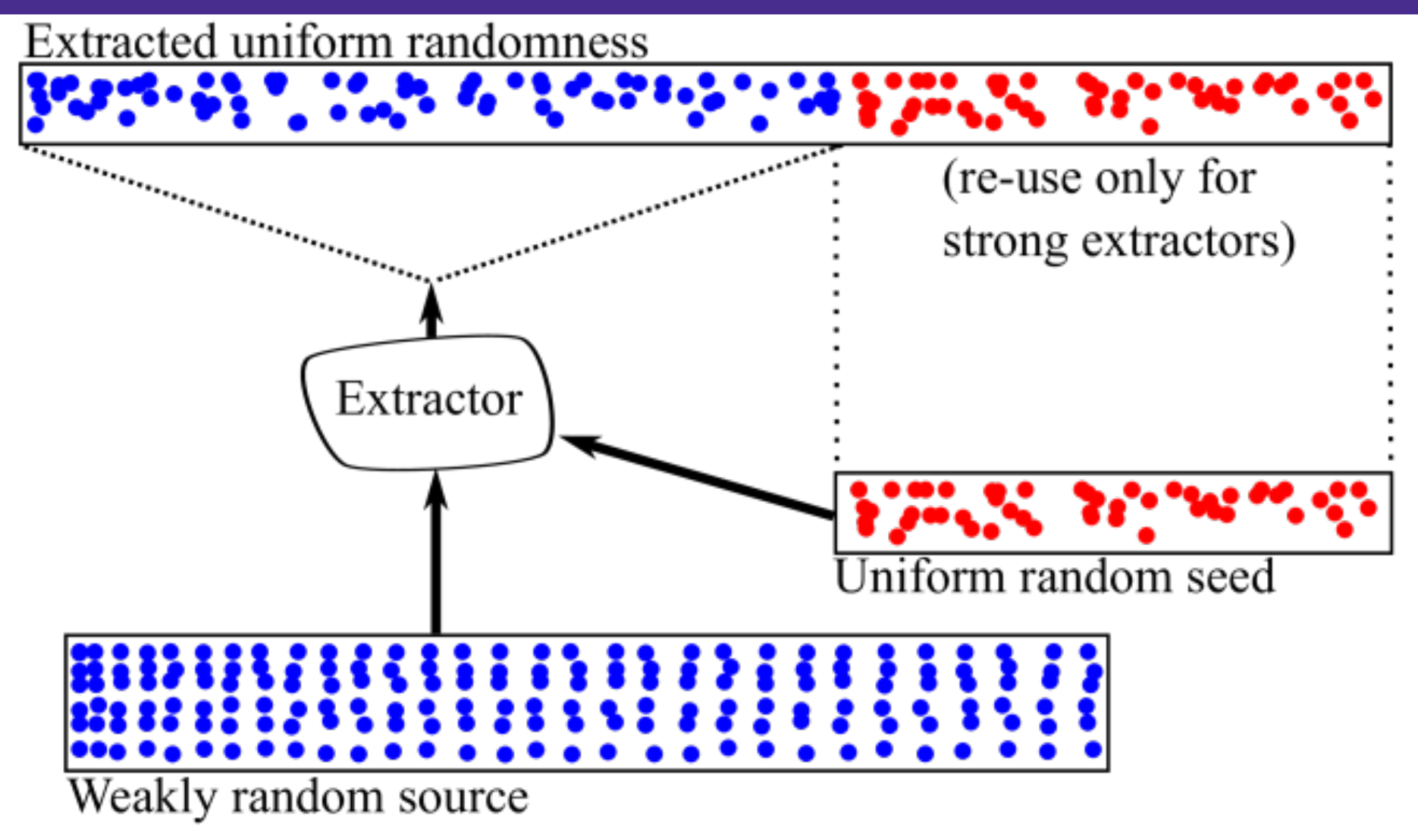
Extracted uniform randomness
(re-use only for strong extractors)
Extractor
Uniform random seed
Weakly random source

Figure 5: High-level interpretation of a randomness extractor

Considering the difficulty of finding completely unbiased PRNGs and efficiently-implemented cryptographically secure PRNGs, we would like to take two "weak" sources of randomness to remove the more obvious biases and create a stronger random bit sequence. Using an extractor function, it is possible for a bit sequence to be transformed into one which is some ε distance from the perfect uniform distribution U. Sources from which sufficient randomness can be extracted include a "weakly random" bit string and some pseudorandomly-generated seed or two "weakly random" bit strings. However, it is becoming clearer true randomness may not be possible, since we cannot verify that our ε distance is exactly 0 for all bit strings.

QUANTUM RANDOMNESS

The first feasible application of quantum computing technology has been documented as random number generation. The underlying architecture of a quantum bit, or qubit, is a probabilistic distribution of 0 and 1; that is, it has a certain percentage of 0 and the remaining percentage of 1. As a quantum logic gate called a Hadamard gate entangles multiple probabilistic qubits simultaneously and outputs a bit dictated by the combined distribution, the individual probabilities of 0 and 1 approach 50% each.

TRUE RANDOMNESS?

Each of these algorithms in their current form has some form of bias for which we cannot entirely control. Hence, the idea that true randomness is not currently possible is clear. However, since true randomness hinges on the idea of some sequence being uniform and unpredictable for infinite terms, I argue true randomness will never be possible, and we must use increasingly better estimates.

ACKNOWLEDGMENTS

I would like to thank Prof. Bryan Catron for overseeing this research, the Furman Computer Science Department for providing me the opportunity to perform this research, and the Furman Advantage.