Levi Seibert

00086809

lseiber1@my.athens.edu

ITE 523 – Dr. Yahia Fadlalla

Random Numbers and Cryptography

Cryptography is a foundational factor in today's technological world. A key component of cryptography is randomness. Random numbers make ciphertext unguessable, or least more unguessable (how often are ciphertexts 100% secure?). Without randomity, cryptography would be elementary. In order to protect messages, adversaries need to view ciphertexts as completely random gibberish [1]. To fulfill this requirement, random number sequences are applied to various components of the cryptographic process, including the key, the initialization vector, nonces, salt, padding strings, and blinding factors [1]. In reality, random numbers are the foundation of cryptography. But what is a random number? To the everyday person, a random number is just some value that is picked from a range of numbers by chance. This is true, to an extent, but it leaves several questions that need to be answered. How are random numbers selected? What determines if a number is truly random? How does a computer, which is built on predictability actually produce unpredictability?

Before any questions on random numbers can be answered, we need to answer the foundational question, what does random even mean? According to Thilina Rathnasooriya, randomness is the "outcome of a probabilistic process that produces independent, uniformly distributed and unpredictable values that can not be readily reliably reproduced" [1]. This mean a random number has to not be dependent on any other values, it needs to be uniformly distributed (have equal probabilities of being a 0 or 1; remember that computers work in binary), and it needs to not be guessable. In addition, it should not be able to recreated. Mengdi, Xiaojuan, Yayun, and Siwei from the China Electric Power Research Institute similarly state that the random sequences found in cryptography need to be uniformly distributed (no bit-bias), need to have all elements be independent of each other, and need to be unpredictable based on the previous sequence elements [2]. In summary, a random number is one is completely unguessable. No logic or mathematics can be used to deduce what value will be chosen.

When it comes to selecting random numbers, there are two methods that can be used by computers. To the common person, their results differ only slightly, but for technological (especially cryptographically) purposes, the difference is extremely significant. Computer-generated random numbers are either true random numbers, or, more often, are pseudo-random, meaning they appear to be random, but are not in actuality. In of itself, a computer cannot produce a truly random number via software alone. Computers are mathematical machines, and, as such, will produce a predictable result based on the input supplied. They can make a pretty good attempt at randomity, and for non-technical purposes, they are useful and can produce values that suffice.

A truly random number is produced by a Random Number Generator, or RNG (also called TRNGs for True Random Number Generators and NDRNGs for Non-Deterministic Random Number Generators). The term “number generator” is sometimes also replaced by “bit generator” (resulting in acronyms like RBG and NDRBG) [3, 4]. True random numbers rely on a good source entropy, which is the “measurement of uncertainty or disorder in a system” [3]. Rich entropy is critical to producing values that cannot be guessed and are mathematically unpredictable. The best way to harness entropy is by looking outside a computer. RNGs make use of hardware in order to gather hard-to-predict values (like wind speeds, computer mouse movements, etc.). These non-deterministic inputs give a random “seed” value that can then be inputted into a cryptographically-strong pseudo-random number generator in order to produce more random numbers [1]. So, while true random numbers are computer generated to an extent, they rely on external data in order to produce true randomness.

Pseudo-random number generators take an initial value (called the seed) and perform a series of deterministic mathematical functions on the seed to produce a nearly-random value [1]. The underlying issue with pseudo-random number generators (PRNGs or DRNGs/DRBGs for Deterministic Random Number/Bit Generators) is that they can be determined solely by the seed value. If the initial value is known, and the attacker knows what PRNG algorithm is in play, then the “random” numbers can be predicted, which fails their earlier definition of randomity [5]. That being said, as long as the seed is kept secure, the numbers produced from a PRNG are unpredictable (but with associated risks) [4]. Relying solely on a pseudo-random number sequence is insecure and is likely not suitable for encryption. Since they are completely internal-based and only rely on software, not hardware, pseudo-random number generators are faster than their true random number counterparts. PRNGS are also periodic, meaning that after a given amount of time, their results will begin to repeat. This is a major flaw, but it does have usefulness in some applications, like code testing. The longer the period (the amount of time before the results repeat), the more secure the pseudo-random number generator is [5].

The general methodology for creating pseudo-random numbers is to follow the four-step process described by Arobelidze:

1. Accept a seed value input number,
2. Perform a series of mathematical operations to produce a random number,
3. Use the calculated number as the seed for the next iteration, and
4. Repeat the process for as many iterations as random numbers needed [5].

One of the most common PRNGs is the Linear Congruential Generator (or LCG). This algorithm takes as input a seed value as well as a multiplier (a), an increment (b) and a modulus (m). As long as a few initial conditions are met, the arithmetic calculation of the random number is the result of multiplying the seed by a, adding b, and performing the modulus operation with m [5]. For example, if the seed is 20, the multiplier is 5, the increment is 7, and the modulus is 9, then the resulting “random” number is equal to ((20 * 5) + 7) mod 9, or 107 mod 9, which is equal to 8. This 8 would then be used as the “seed” value for the next random number calculation: ((8 * 5) + 7) mod 9 = 2. If the original seed had been changed to 21, then the result of the first iteration would then be 4 (112 mod 9), and the result of the second would be 0 (27 mod 9). These results are seemingly random (shifting the original seed by one value dramatically modifies the resulting number sequences), however, knowing the initial values of the seed and variables allows an attacker to compromise the number generation and guess the pending calculations, thus absolving the cryptographic strengthen that may have already been in place.

In order to determine whether or not a number sequence is in fact truly random, a series of tests have been developed. The problem with tests for randomity is that it is actually unprovable, one can only really prove if something is not random [4]. This means that these tests have to not fail in order to have a level of confidence that the number sequence is in fact random. The three leading standardization organizations with standards on random numbers are NIST, with their SP 800-22, SCA with their GB/T 32915-2016, and BSI, with AIS 20/AIS 31. When these three standards are combined, there are 24 resulting tests for randomization. These tests include the frequency test, which evaluates the proportion of 0s to 1s in the sequence; the run test, which checks whether runs of 0s and 1s are consistent; and many other more complex evaluations [2].

Randomness is a foundation to cryptography and cybersecurity. As such, the algorithms that produce such numbers need to tested and proven before accepted as valid. An encryption cipher could be 100% effective (like a One-Time Pad), but if the key values are not randomized, then an attacker may be able to identify the key and thus disrupt the secrecy that was expected. Since true random numbers rely on external hardware components, they are often slow and as such, inconvenient. However, as is often the case in the realm of computing, convenience is a small price to pay for safety. Computer users, especially those that send packets across the public Internet, should be aware of the protocols at work behind-the-scenes so that they may make decisions on how to best protect their data and keep it out of the hands of the wrong individuals. It may sound dumb to investigate the deep details of an algorithm, like how are random numbers selected, but one wrong choice could lead to a world of regret.

References:

[1] T. Rathnasooriya, "Importance of True Randomness in Cryptography," *LinkedIn*. https://www.linkedin.com/pulse/importance-true-randomness-cryptography-thilina-rathnasooriya/.

[2] Z. Mengdi, Z. Xiaojuan, Z. Yayun, and M. Siwei, "Overview of Randomness Test on Cryptographic Algorithms," *Journal of Physics: Conference Series*, vol. 1861, no. 1, p. 012009, Mar. 2021, doi: https://doi.org/10.1088/1742-6596/1861/1/012009.

[3] H. Sidhpurwala, "Understanding random number generators, and their limitations, in Linux," *Redhat.com*, 2019. https://www.redhat.com/en/blog/understanding-random-number-generators-and-their-limitations-linux.

[4] "The Importance of True Randomness in Cryptography," *Design And Reuse*. https://www.design-reuse.com/articles/27050/true-randomness-in-cryptography.html.

[5] A. Arobelidze, “Random Number Generator: How Do Computers Generate Random Numbers?,” *freeCodeCamp.org*, Oct. 26, 2020. https://www.freecodecamp.org/news/random-number-generator/.