The Art and Math of Cryptography

A Practical Guide for Cybersecurity Professionals

Michele Bousquet

**** * Writing Sample * ****

Chapter 8
Random number generators

Random numbers are important to cryptography. For example, a symmetric key is often just a very long number generated by a computer-based random number generator (RNG).

Since random numbers are so important to cryptography, many mathematicians and programmers have devoted considerable time and energy to figuring out the best way to generate them for use in cryptography.

At present, there is currently no surefire algorithm or system that can churn out a truly random series of numbers. For this reason, several workarounds have been developed to make the output from an RNG get closer to the goal of "truly random."

When random isn't random

You can experience a quick practical example of how "random" isn't necessarily random by asking a few people to pick a random number between 1 and 10. Chances are that most people will choose 7, and very few will choose 1 or 10.

Historical RNGs

RNGs have been around for centuries, but not in computer form. Here are some real-world RNGs:

- ***Dice*** - Produces random numbers from 1 to 6 (or however many sides the dice has).
- ***Coin flip*** - Produces one of two outcomes, Heads or Tails, which could be expressed as 0 and 1.
- ***Mechanical RNGs*** - A physical device such as a tumbler full of numbered balls, cranked by hand until a ball pops out. Such devices are commonly used to choose lottery or Bingo numbers.

These RNGs, when used repeatedly and as intended, produce a truly random sequence of numbers. However, you have probably heard of instances where such systems have been tampered with to change the probabilities, such as weighted dice and magnetized balls. Such RNGs are called ***weighted RNGs***.

When weighted RNGs are used in cybersecurity, whether intentionally or by accident, hackers can find patterns that they can abuse to compromise security.

Hot Lotto RNG scam

Most stories of RNG compromise involve the gambling industry. In 2017, brothers Tommy and Eddie Tipton were convicted of collecting illegal lottery winnings after Eddie, a former security director of the Multi-State Lottery Association (MUSL), had tampered with the computer-based RNG that generated winning numbers for Hot Lotto lotteries in several states. The RNG was designed to produce random numbers most of the time, but on specific dates it would produce certain numbers. Eddie then traveled to various states on the specified dates and purchased Hot Lotto tickets with the winning numbers. Eddie was sentenced to 10 years in prison for his part in the scheme.

Uses of RNGs

Here are some common uses of random numbers in cybersecurity:

- Cryptographic key generation for data encryption, user authentication, and establishment of secure communication channels.
- Generation of random prime numbers for calculations of public/private key pairs.
- Generation of random passwords such as those used by password managers, or the ones suggested by your browser when you sign up for a website.

- Session key generation for those cryptographic protocols that generate a new key at the start of a session, and periodically refresh the key during the session.
- Generation of salts for increased security when storing passwords.

Requirements for RNGs

The quality of an RNG is measured by how it performs over time, after many numbers have been generated. To be considered a decent RNG, the software or system must:

- Produce numbers that are uniformly distributed across the scope of possible values. For example, when choosing a random number from 1 to 100, an equal number of results should be below 50 and above 50.
- Produce a series of numbers that follow no detectable pattern from one number to the next.
- Have a long "cycle length." Any RNG is ultimately going to start cycling, where numbers that have been picked before are picked again. The length of this cycle must be so long as to be useless in pattern detection.

In the context of RNGs, the term ***entropy*** refers to the measure of randomness or unpredictability of the generated numbers. The higher an RNG's entropy, the more random it is; thus, high entropy is desirable for RNGs. In practical terms, high entropy means the numbers generated by an RNG meet the requirements listed above.

Types of RNGs

RNGs come in two basic flavors: ***pseudorandom*** and ***true random***.

- ***Pseudorandom Number Generator (PRNG)*** - Uses a seed to start off the algorithm. A PRNG is deterministic in that the output is repeatable based on the seed. This type of generator is also called a Deterministic Random Bit Generator (DRBGs).
- ***True Random Number Generator (TRNG)*** - Does not use a seed, but instead uses sources of entropy to generate the random number. A TRNG is non-deterministic in that the output is not repeatable—it produces a different output every time. A Hardware Random Number Generator (HRNG) is a type of TRNG that uses physical phenomena such as thermal or atmospheric noise as a source of entropy.

PRNGs and TRNGs have different pros, cons, and uses. PRNGs tend to generate their output faster than TRNGs, but are more prone to low-entropy behavior such as predictable and patterned output, which makes PRNGs more prone to cryptographic attacks. PRNGs are often used for operations where speed is of the essence, such as key generation for TLS/SSL handshakes. Because such handshake keys expire quickly, the level of security that PRNGs provide is generally sufficient for these operations.

TRNGs are usually slower to produce output than PRNGs, but also tend to produce more random output, provided a sufficiently random source of entropy is used. The lower predictability makes the output of TRNGs more cryptographically secure, as attackers can't find patterns to exploit. TRNGs are often used for situations where extremely high entropy is required. Casino games and gambling applications, for example, favor TRNGs because their RNGs are required by law to pass a battery of tests for unpredictability, non-repeatability, and randomness. Military applications also prefer TRNGs because of their superior cryptographic security.

Linear congruential generators

The linear congruential generator (LCG) is a popular PRNG that uses just a few calculations with three preset numbers, including a very large prime number.

1. Multiply the previous number in the sequence by the first fixed number.
2. Add the second fixed number.
3. Perform a modulo operation with the prime number to get the output.

The simple nature of its calculations makes LCG easy to program and test. While LCGs can produce random sequences, a poor choice of preset numbers can lead to noticeable patterns in the output. However, it is relatively easy and fast to test a set of inputs for acceptably random output.

LCG remains popular today, particularly for non-critical applications, because of its ease of use.

How RNGs work

The heart of an RNG is its algorithm, the mathematical computations it makes to come up with random numbers. However, these algorithms need to be fed certain inputs to produce their output.

Seeds and sources of entropy

A seed is an initial value (a number) given to the RNG algorithm to start its processes. The seed might be a randomly generated number, programmer input, or user input.

Conversely, a source of entropy is a random number that gets added to the mix along the way. A value for the source of entropy might be the current time of day (down to the millisecond), fluctuations in electrical signals, atmospheric noise, or any handy source that a computer chip can access.

The main differences between seeds and sources of entropy are:

- A seed is introduced when the algorithm is started, while sources of entropy are brought in later.
- A seed can be set by a programmer or user at the time they start the RNG. Compare with a source of entropy, where the command to go and grab a value is part of the algorithm and cannot be controlled by the programmer or user.

If a source of entropy is random, why not just use that as the random number?

This is a good question. The answer lies in the fact that a source of entropy often has a limited set of possible values. For example, consider the use of "time of day" as a seed or source of entropy. Such a number would fall within a certain range of values, especially if a hacker knew roughly the time when the number was generated. Such sources of entropy are best used as a random factor for mathematical computations rather than as the random number itself.

Testing for entropy

There are numerous tests for RNG entropy, most of them involving some pretty advanced math. However, you can perform a simple entropy test for the first requirement, uniform distribution, with a simple column graph.

Let's use the imperfect human RNG mentioned at the start of this chapter: "Ask 100 people to pick a number between 1 and 10." Your test results might look something like this:

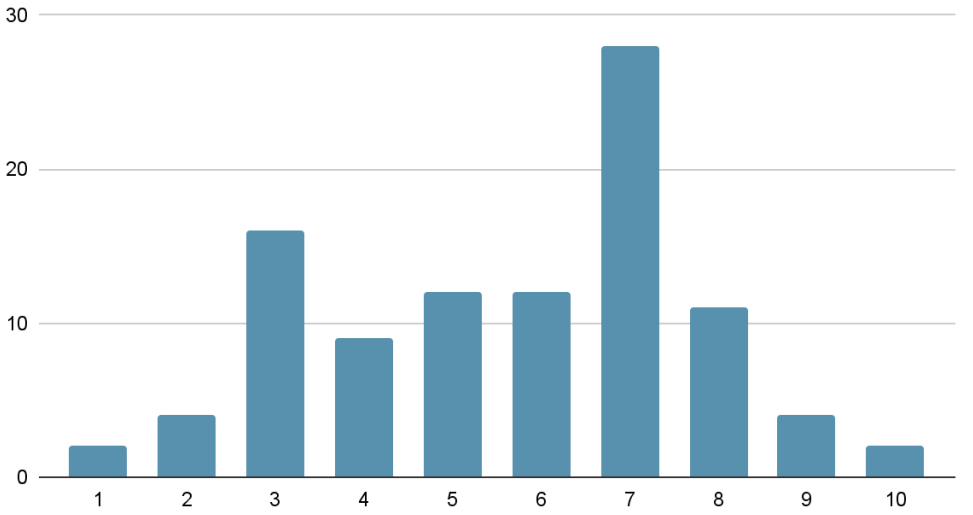

Figure 24: Distribution of number of people choosing numbers 1-10

There are great differences in the heights of the columns, because most people chose 7 or 3. The variations in column heights indicate that this method of random number generation has lousy entropy with regard to uniform distribution. High entropy would have shown even, or nearly-even, heights for the columns.

The simple column graph works for checking the frequency of a small set of numbers, but what about checking for the uniqueness of extremely large numbers? One method is to convert each number to binary, and count how many times the number 1 appears in each position.

Let's try this with 8-bit numbers and two small samples, just for illustrative purposes. Here are samples of random numbers from two different RNGs, each represented in binary.

	RNG 1	**RNG 2**
	1011 1101 0001 1110 1101 1000 0110 0011	0011 0101 0111 1101 0001 1001 0101 1001
Number of times 1 appears	2223 3222	0224 3204

By adding up the number of times 1 appears, we can get a picture of each RNG's entropy. At a glance, we can see that the totals for RNG 1 fluctuate between 2 and 3, and those for RNG 2 vary from 0 to 4. With RNG 1, the totals are "evened out" to some degree, meaning that RNG 1 has greater entropy than RNG 2.

In reality, the numbers would be much longer and the entropy calculations much more complex, but these simple examples should give you a sense of how RNG entropy is measured.