Let’s Get Random – Entropy Explained

Author	josh
Date	2022-01-21

Overview

What is random? Is it singing a song out of tune at a socially inopportune time, or wearing a hat inside out? In the context we care about (cryptography), certainly not. Human beings are actually terrible sources of randomness as it turns out. But we need entropy for many critical cryptographic tasks, so let’s explore what entropy truly is and how we can generate it using computers.

Entropy

Why do we need entropy?

Entropy is of critical importance for modern cryptographic operations. Every time we encrypt a file, log into a webpage, or digitally sign a message, random sources are necessary.

The most common use of entropy in this context is for key generation – for either symmetric or asymmetric encryption, we must have a sufficiently random key. Otherwise, an attacker may be able to discover our secret key and decrypt the ciphertext – not ideal for the secrecy we are trying to accomplish.

As well, cryptographic “nonces” – throwaway random bits are useful for certain operations such as signing messages using elliptic curve algorithms. A prime example of this is the generation of Bitcoin transaction signatures – wallets that accidentally reuse nonces can leak the private key of the user.

True Random vs. CS-PRNGs

Now that we’ve established a use for entropy in cryptography, let’s further define what entropy is in this context. There’s three types of random sources to consider for our purposes: true random number generators, cryptographically secure pseudorandom number generators (CS-PRNGs), and PRNGs not suitable for cryptographic use.

The first time of RNG, true random number generators, are rather rare. There are a few sources of “true” randomness in the universe, one of which is radioactive decay. Although this can be measured, it is quite impractical for computing purposes. It would be annoying for us all to source and carry around a source of radioactivity for every day key generation, not to mention potentially error prone. What happens if our measurement device breaks, or our atom fully decays, and we start getting very predictable streams of bits?

This leads us to our second class of random sources: *pseudo*random number generators. These sources are not “truly” random in a physics sense, but rather provide sufficient unpredictability sufficient for computing purposes. There's two subtypes of PRNGs: those unsuitable for cryptographic use, and those that are considered *cryptographically secure* PRNGS. We don't particularly care about the first type. Non crypto-secure PRNGs such as the Merseinne Twister are useful for everyday programming tasks where security doesn't matter, such as computer games.

Cryptographically secure PRNGs are much more interesting. These sources provide random bits that, for all practical purposes, cannot be predicted – if you were to take a sampling of millions or billions of bits and plot them on a graph, they would be uniformly distributed; completely unpredictable using any statistical means. These CSPRNGs are therefore suitable for critical cryptographic use, such as key generation and nonces; where it is of the highest importance that an attacker cannot predict the bits in any way.

Seed -> Feed

The internal workings of these CSPRNGs use a mechanism I like to call "seed and feed". There's two components of this system: unpredictable data collected from the environment, and cryptographic functions that mix this data in statistically unpredictable ways.

The seed data can be collected any number of ways. A common mechanism in production environments is to sample user inputs: key press timings, mouse movement, etc. A secure encryption software like Veracrypt will ask the user to move the mouse around randomly for a period of time to help feed the entropy pool. There's also some interesting ways paranoid users can collect entropy from the environment that are less practical, but nevertheless effective. Physical sensors such as accelerometers, or the thermal noise from cameras can help seed the entropy pool. There's even manual user input such as dice rolls or card shuffling!

I have created two educational demos of this concept. One, called Entropal, collects dice roll input from the user and uses the random bits to form diceware passwords. It even uses 4 or 8 sided die to show the entropy collection as whole number bits. The other, called Entropy Collector, uses a hardware accelerometer to feed an entropy pool based on physical movement and generate passwords. These demos, although only for educational purposes, demonstrate how environmental data can seed CSPRNGs.

The "feed" part of CSPRNGs use several pseudorandom cryptographic primitives to mix the incoming "seed" data in a way that is statistically unpredictable. A production CSPRNG that can be studied is *Fortuna*. The underlying mechanisms are some pseudorandom primitive: such as the AES or Twofish encryption algorithms in CTR (counter) mode, or a cryptographic hash function like SHA-2/3.

Entropy

By constantly collecting environment data and mixing it via these cryptographic functions, we can get a reliable stream of random bits usable for keys and nonces. These CS-PRNGS come built into modern operating systems, and even microcontrollers such as those found in cryptocurrency hardware wallets. There's even ways to manually collect random data and generate keys/passwords/etc. from the data using this sort of methodology (for truly paranoid, educational, or enterprise purposes). The science of randomness is a fascinating building block for modern strong cryptography.