Let’s Get Random – Entropy and Cryptocurrencies

A Little About Me...

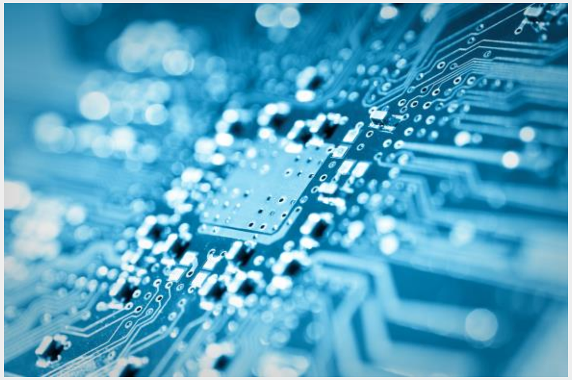
- Software Engineer @ Microsoft in Pittsburgh
- Tech Educator @ chaintuts
- Love to build free and open source technical education!
  - Interested in cryptocurrencies, computer science, security topics

Why Talk About Entropy?

- Entropy (randomness) is critical to applied cryptography/cryptocurrencies
- Broken randomness breaks cryptographic systems
- It’s interesting!

Entropy (In CS)

- Different (but similar) to entropy in physics – the degree of disorder in a system
- Randomness collected by a computer system for use in things like key generation, nonces, etc.
- There's different "levels" of randomness for our purposes

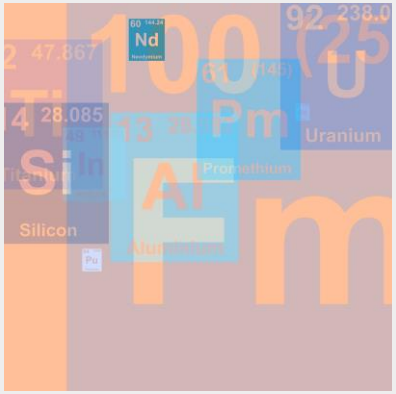


Types of Entropy

- True random sources
- Non-cryptographic pseudorandom numbers
- Cryptographically-secure pseudorandom numbers

True Random Sources

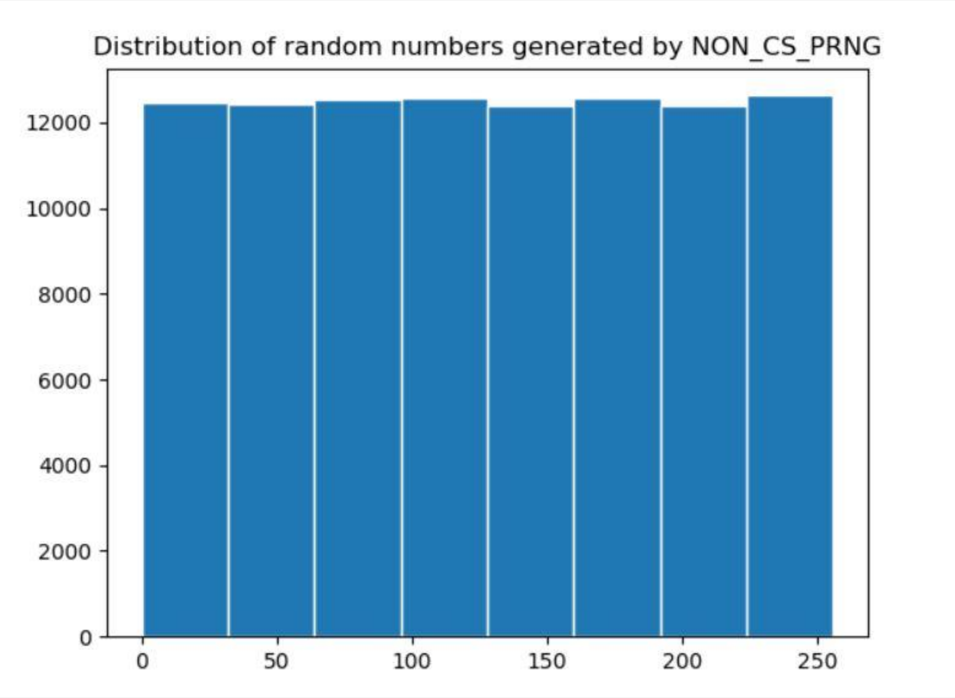
- There are a few sources of true randomness in the universe
- One of them is radioactive decay
- This entropy is possible to record, but impractical
- We can’t easily carry around radioactive isotopes, nor expect the systems not to break

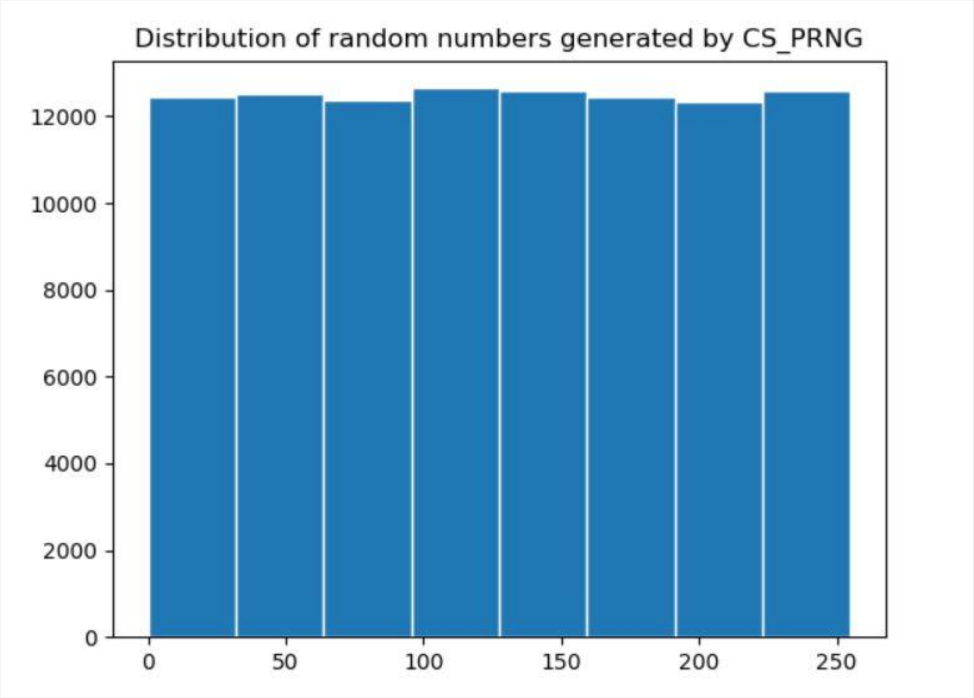


Pseudorandomness

- Instead of relying on physical TRNGs, let’s use computer algorithms instead
- Problem: computers are very predictable
- There’s different algorithms for getting random bits
- Two types: PRNG, CS-PRNG

PRNG vs. CS-PRNG





Non-CS PRNG – Mersenne Twister

vs.

CS-PRNG – OS source via Python os.urandom()

Histogram showing the distribution of random numbers generated… what's the actual difference?

PRNG vs. CS-PRNG

- It's not just about *uniform distribution.*
- It's about *predictability* of new random bits
- Given 624 numbers, you can predict next numbers from Mersenne Twister
- Cannot predict output of CS-PRNG function given previous bits

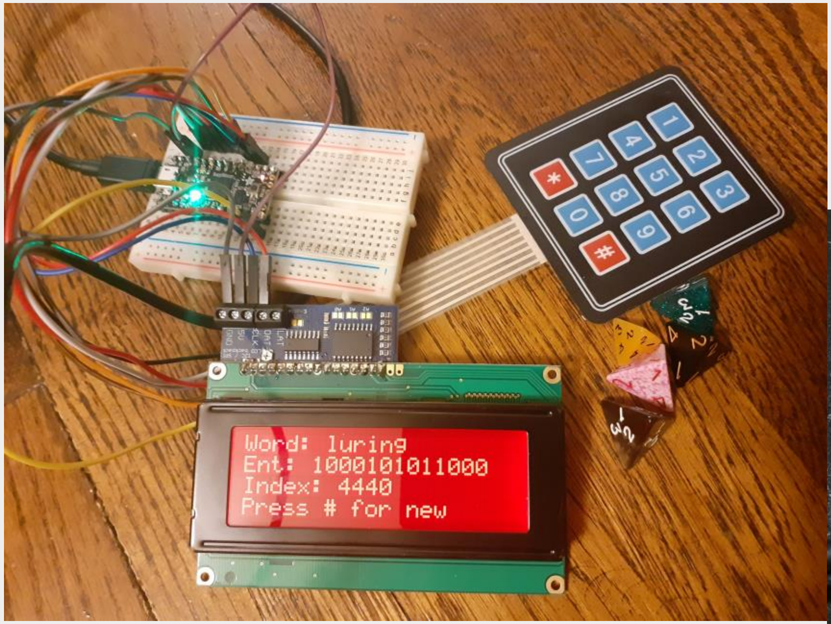
PRNG vs. CS-PRNG

- Why do we care?
- It’s critically important for many cryptographic uses of RNGs that the numbers be unpredictable
- Ex: key generation for a crypto exchange
- Imagine an attacker gets hold of 650 keys in a leak, and then can predict new keys!

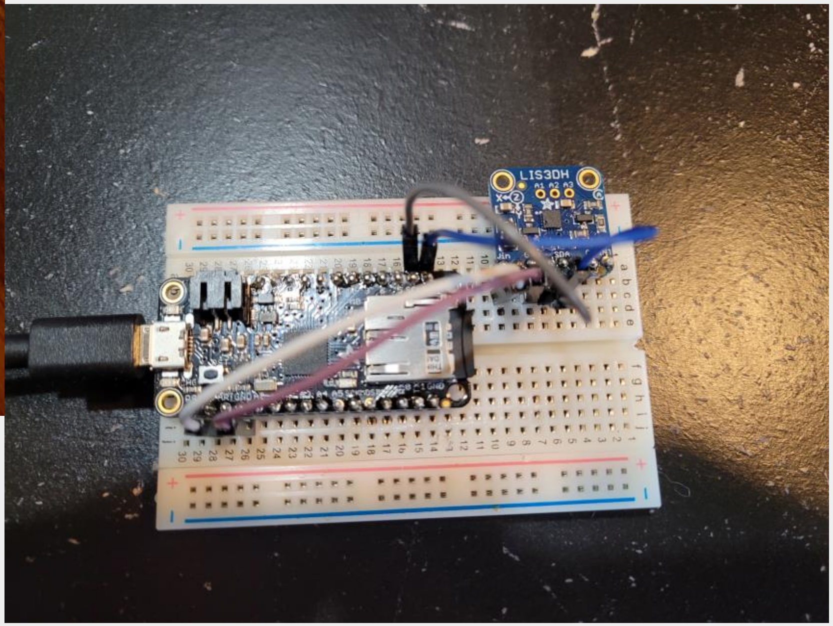
CS-PRNGs

- **Seed -> Feed**
- Use relatively unpredictable environmental sources
  - Mouse movement, key press timings, accelerometer, temperature, etc.
  - Can even be manually generated, like dice rolls
- Feed those sources to a cryptographic function
  - SHA hashing algorithm, AES block cipher in CTR mode
  - Production algorithm: Fortuna
- Get unpredictable random bits out the other end

Demos:



https://github.com/chaintuts



Entropal: Diceware password generator showing bits of entropy

EntropyCollector: Generates passwords from accelerometer entropy

Uses of Cryptographic Randomness in Cryptocurrencies

- Key generation
- Signatures
- Proof-of-Work
- Proof-of-Stake
- and what happens when these things break…

Key Generation

- Private keys are the core of cryptocurrency ownership – your keys, your coins. Not your keys, not your coins!
- Key *must* be random – keys themselves are 256 bits, but often start from 128 bit seed which is key stretched
- 256 bit keys give so many possibilities, collisions are near impossible *if the keys are random*
    - *2^256 is around 10^64, on the order of the number of atoms in the observable universe*

Key Generation - Breakage

- What happens if keys aren’t random enough?
- Pretty much worst-case scenario for crypto – the coins are stolen and moved to an attacker wallet.
- Practical example: *Brainwallets*
  - SHA-256 hash of a user-generated passphrase like “"Interior Crocodile Alligator” -> used as 256 bit private key
  - Ryan Castellucci did a DEFCON topic and they released a tool called brainflayer
  - They note that brainwallets are often drained in minutes to hours, even with seemingly strong passphrases
  - *Humans are bad at entropy*

Signatures/Nonces

- Signature algorithms such as secp256k1 (elliptic curve used in Ethereum and Bitcoin) require a nonce when signing messages (like transactions)
- Nonce *must* be one time use
- Nonce reuse can easily reveal *private key*
- Ex: https://github.com/chaintuts - NotOKReuse
- Sloppy wallet programming can lead to theft

Some Less Obvious Uses…

- Proof-of-Work and Proof-of-Stake!

- We have to make it difficult for an attacker to change the state of the blockchain, and to *sustain* an attack

- In other words, we need the block “winner” to be *chosen at random*

- **But,** we also want to reward those that invest more in mining/staking with a higher *probability* of being selected

Proof-of-Work

- PoW uses the pseudo-random property of cryptographic hash functions like SHA-256
- Require a block with a certain number of leading zero bits (ex: 4, real difficulty much higher)
  - block + 0 -> 1aef0...
  - block + 1 -> ed032...
  - block + 2 -> 123ac8... . .....
  - block + 4000 -> 0000a2
- Difficulty: the more 0 bits, more guesses on average it takes to find a block
- The mechanism is simple: more hashpower, higher chance of being the first to guess correctly

Proof-of-Stake

- PoS again uses the pseudo-random property of hashes like SHA with a *commit-reveal* scheme for RNGs on the blockchain
- Validator generates a random number from its CS source, hashes, *commits* to blockchain
- Later, the number is *revealed*. The number can be hashed by other nodes to verify it's the same hash that was committed
- Numbers XORed for a random validator selection
- It's impractical for nodes to game the system in their favor, since they can't easily change number after the commit (by finding a hash collision)

Let’s Get Random

- There’s different types of RNGs – and the ones we need are *cryptographically secure sources*
    - Use proven systems – like your OS’s random source
    - Ex: Python os.urandom() NOT the random module
- Random numbers are critical for obvious things like key generation/nonces
- But they’re also critical for things like PoW and PoS algorithms that secure these networks
- Getting random *wrong* has serious consequences, like coin theft

Questions?