School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

# COMP3911 Secure Computing

## 6: Issues With Randomness

Nick Efford

https://comp3911.info

Slides are customised for XJCO3911 by Turgay Celik (Turgay.Celik@swjtu.edu.cn)

# Objectives

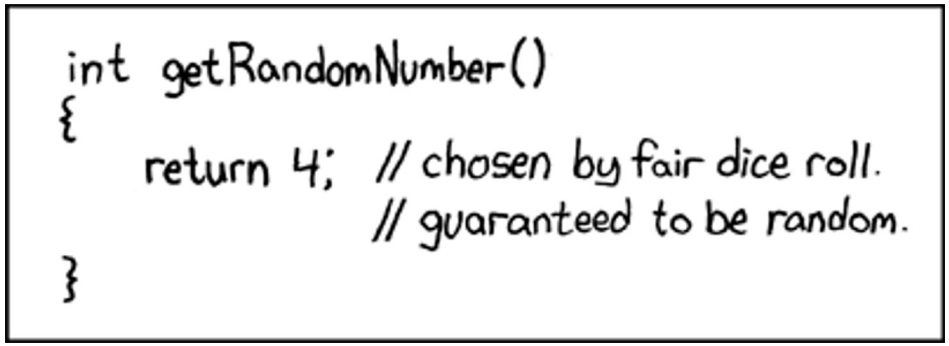
UNIVERSITY OF LEEDS

- To consider whether computers are good at generating sequences of random numbers
- To examine the security implications of bad random number generators (RNGs)
- To explore secure RNG implementations

# ‘Random’ Number Generators

UNIVERSITY OF LEEDS

https://xkcd.com/221/



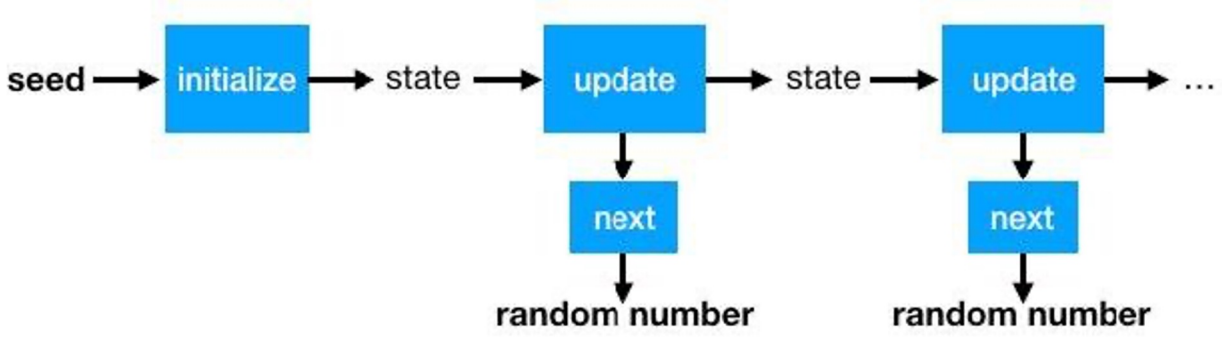

“Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin.”

John von Neumann (1951)

# ‘Random’ Number Generators

UNIVERSITY OF LEEDS

- Actually *pseudo*-random number generators (PRNGs), producing a deterministic sequence of values that merely appears to be unpredictable
- Starting point in the sequence is determined by the **seed**, typically a 32-bit or 64-bit integer value
- A given seed always generates the same sequence!



# Attacks Against PRNGs

UNIVERSITY OF LEEDS

- Cryptanalytic attack
- Brute-forcing
- Discovery of internal state
  - Observe enough output values and we can figure out how the generator was seeded
  - Knowledge of seed = predictable behaviour
  - Easier than you might think!

# Linear Congruential Generator

UNIVERSITY OF LEEDS

$$n_{i+1} = (an_i + c) \bmod m$$

- Starting point is the seed, $n_0$
- Period is $\leq m$ (depends on parameter values)
- glibc: $m = 2^{31}$, $a = 1103515425$, $c = 12345$
- Fast and requires minimal memory

# Mersenne Twister

UNIVERSITY OF LEEDS

- Widely used; default PRNG in Python, Ruby, etc
- Slower and requires more storage than LCG
- Period is enormous: $2^{19937} - 1$ (a Mersenne prime)
- … so it is better suited to producing large quantities of pseudorandom data
- … BUT internal state can be deduced from only 624 consecutive observations!

# What Is Wrong With This?

UNIVERSITY OF LEEDS

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
  srand(time(NULL));

  for (int i = 0; i < 5; ++i) {
    printf("%d\n", rand());
  }

  return 0;
}
```

`srand` initialises PRNG with given 32-bit seed

`rand` produces next number in the sequence

# Cigital's Internet Poker Exploit

UNIVERSITY OF LEEDS


Our cards

We can't see cards of the other players

… but we can compute what they will be!

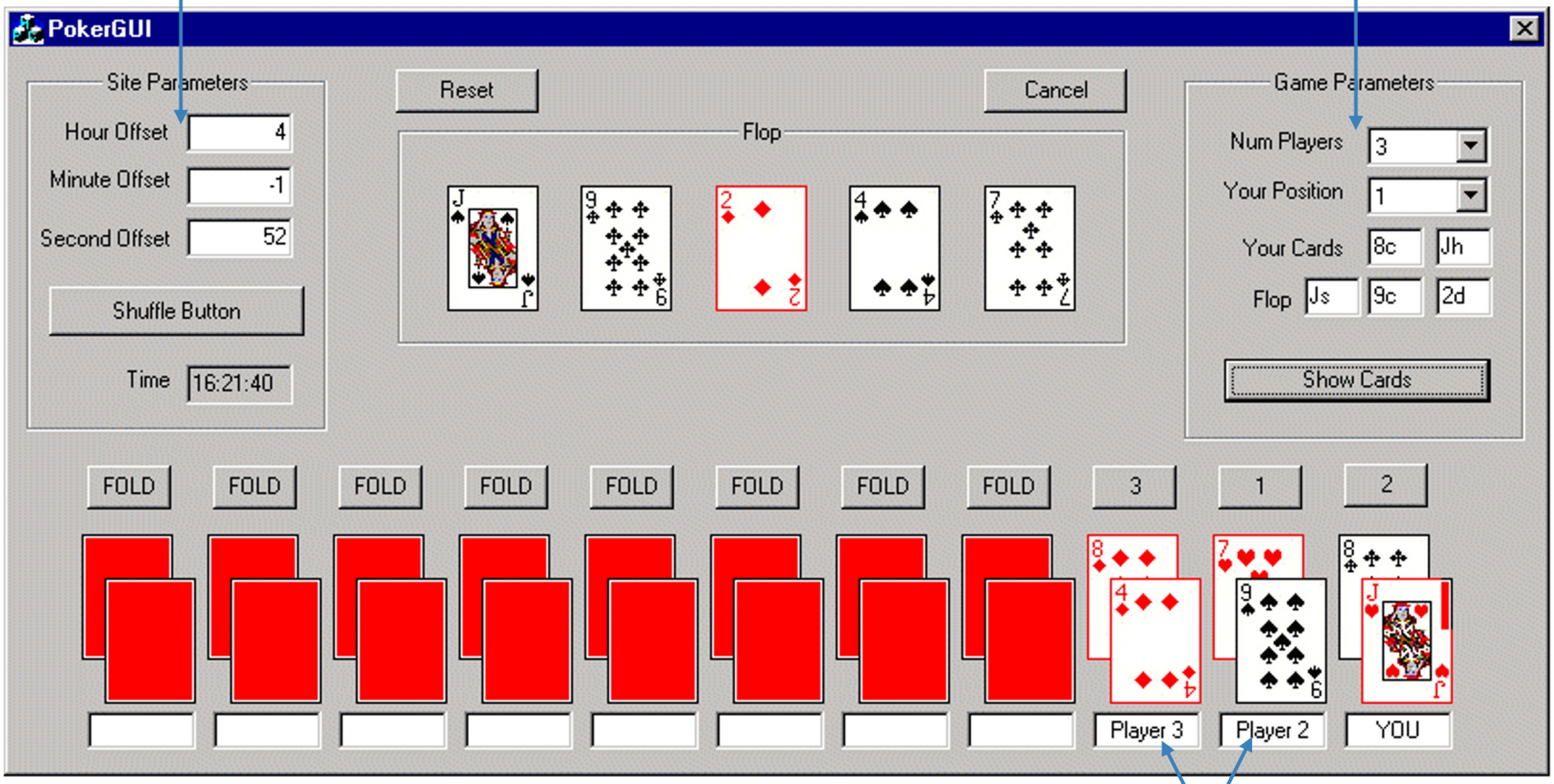
# Cigital's Internet Poker Exploit

UNIVERSITY OF LEEDS

- Flawed PRNG used for deck shuffling
  - Non-cryptographic algorithm
  - 32-bit seed, so 52! (about $2^{226}$) possible shuffles reduces to around 4 billion
- PRNG seed chosen poorly
  - Milliseconds since midnight on system clock;
    4 billion shuffles reduces to 86,400,000
  - If we can sync closely to server's clock, we can reduce this figure significantly…

Synchronise clock & hit Shuffle button

Specify your cards and 3 from Flop




Program computes shuffle, then predicts other players' hands!

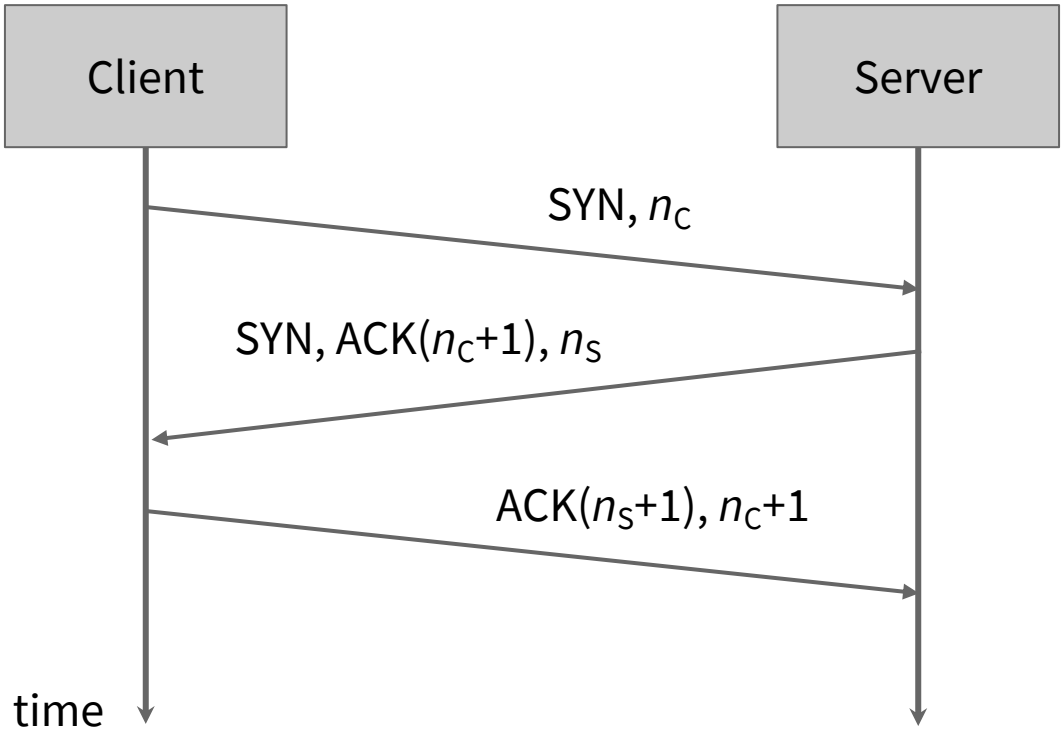
# Does This Really Matter?

UNIVERSITY OF LEEDS

- TLS uses a pseudorandom **session key** to perform symmetric encryption/decryption of data
- PKC is used to exchange session key securely
- No need to break this encryption if we can successfully predict what the session key is!
- … which is precisely what happened in 1996 to the very first implementation (SSL), in Netscape 1.1
  - PRNG seed could be determined from time of day and process IDs on local system

# Another Example: TCP ISNs

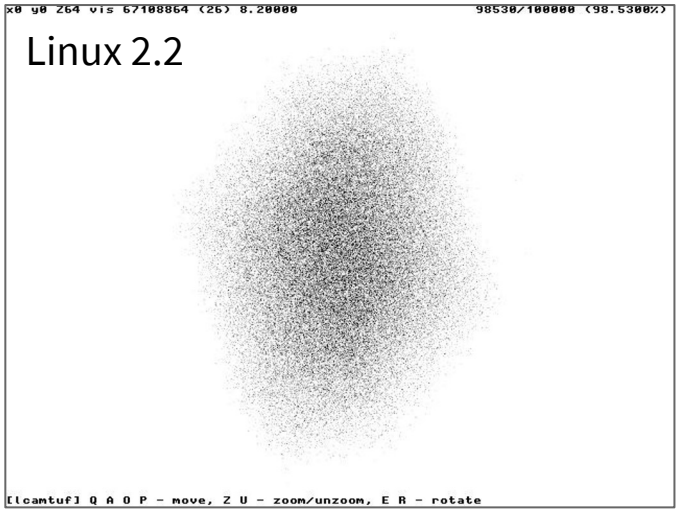
UNIVERSITY OF LEEDS

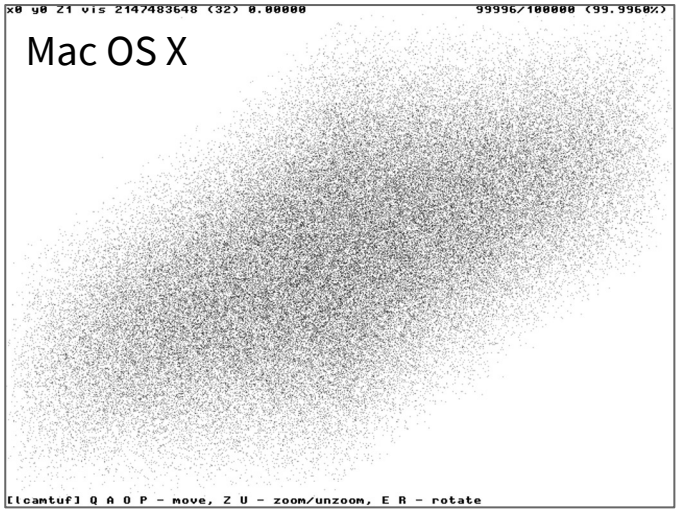


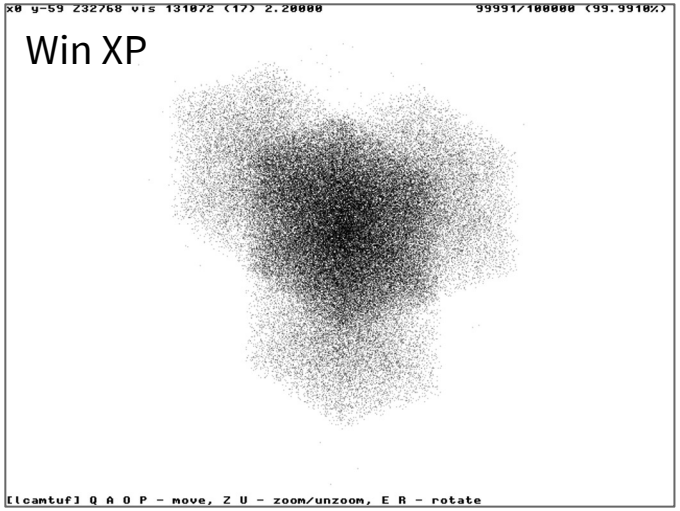

**Initial sequence numbers** (ISNs) $n_C$ and $n_S$ have to be random to reduce risk of TCP session hijacking, etc

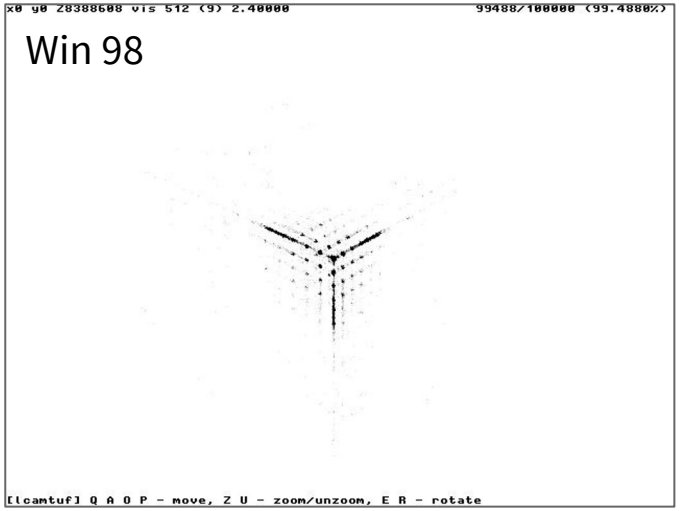
# ISN Predictability (circa 2002)

UNIVERSITY OF LEEDS



# We're Still Getting It Wrong…

UNIVERSITY OF LEEDS

- Sony PS3 bootloader checked code to make sure it had been signed using Sony's private key
- Elliptic Curve Digital Signature Algorithm (ECDSA) requires a random nonce to be used for each signature…
- …but Sony used a fixed value…
- …allowing their private key to be recovered by the fail0verflow hacking group in December 2010

# Android bug batters Bitcoin wallets

## Old flaw, new problem

By Richard Chirgwin 12 Aug 2013 at 00:43 9 SHARE ▼

Users of Android Bitcoin apps have woken to the unpleasant news that an old pseudo random number generation bug has been exploited to steal balances from users' wallets.

The Bitcoin Foundation's announcement, here, merely states that an unspecified component of Android "responsible for generating secure random numbers contains critical weaknesses, that render all Android wallets generated to date vulnerable to theft."

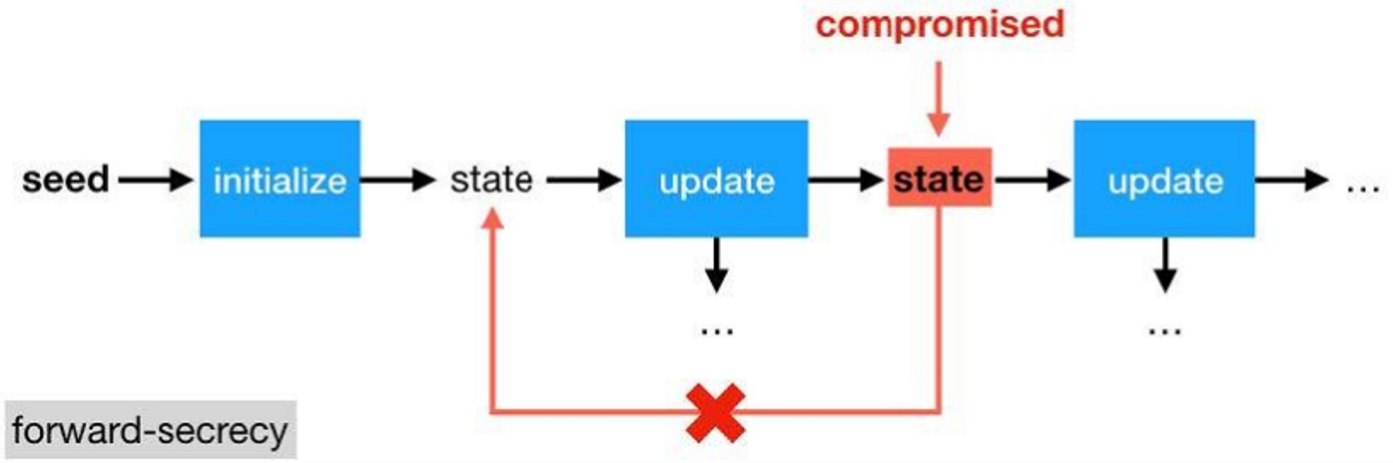
# Better PRNGs

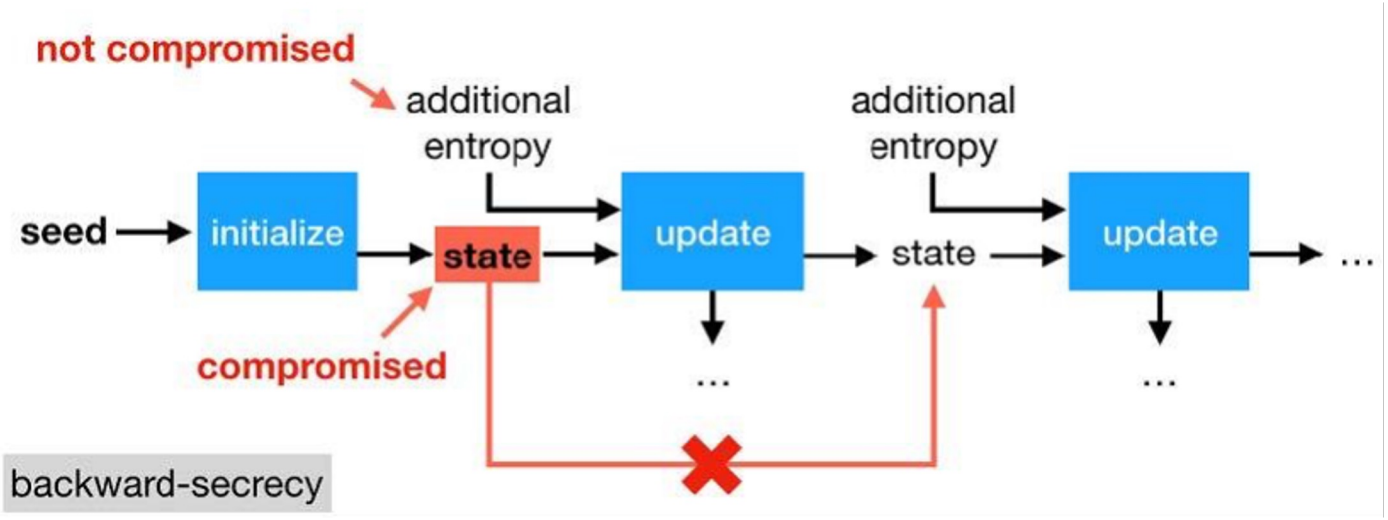
UNIVERSITY OF LEEDS

- **Cryptographically Secure PRNGs** (CSPRNGs) produce numbers that are hard to predict, even when attacker has full knowledge of the algorithm used
  - Symmetric cipher in CTR mode (seed used as key)
  - Hashing of {seed || counter}
  - (+ steps for forward / backward secrecy)
- Critical dependence on seed quality!
- Seed value must be **as random as possible**

# Forward & Backward Secrecy

UNIVERSITY OF LEEDS



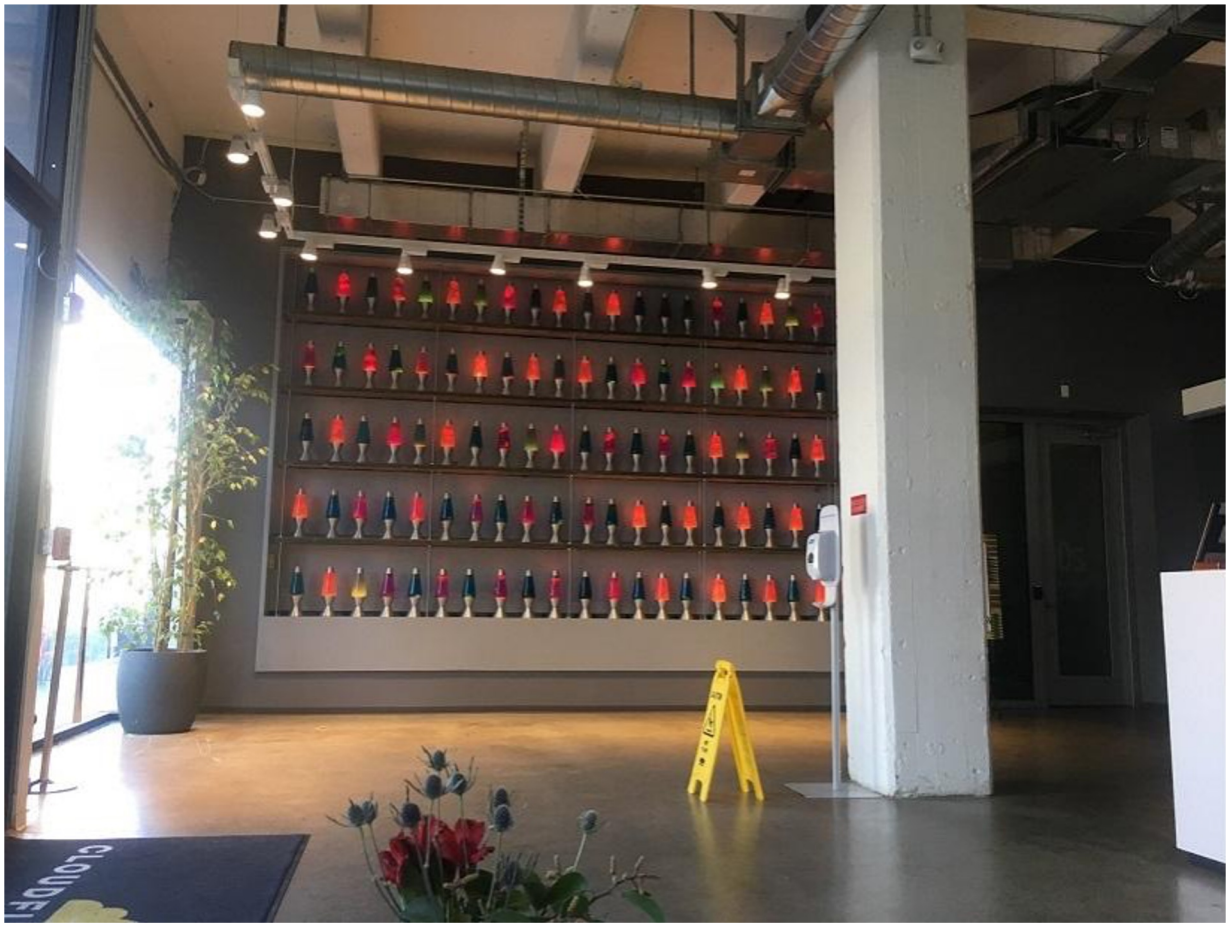
# Entropy Collection

UNIVERSITY OF LEEDS

- **Entropy** of a seed measures its randomness; the more entropy we have, the better
- Different sources, of varying practicality
  - Radioactive decay (via special hardware)
    - https://www.fourmilab.ch/hotbits/
  - Images of chaotic processes
  - Noise (thermal, radio, etc)
  - Keyboard and mouse movement
  - Events internal to OS (disk access, thread timing…)

# LavaRand

UNIVERSITY OF LEEDS

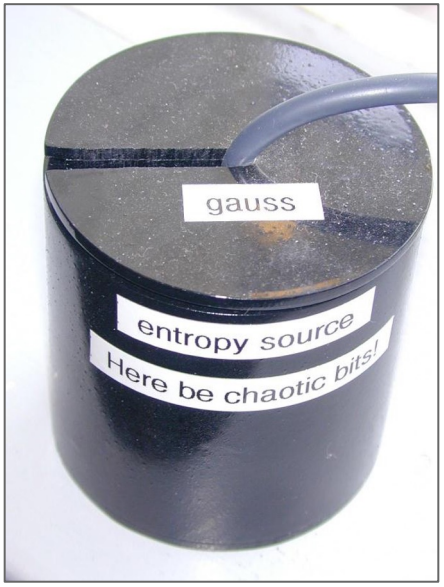


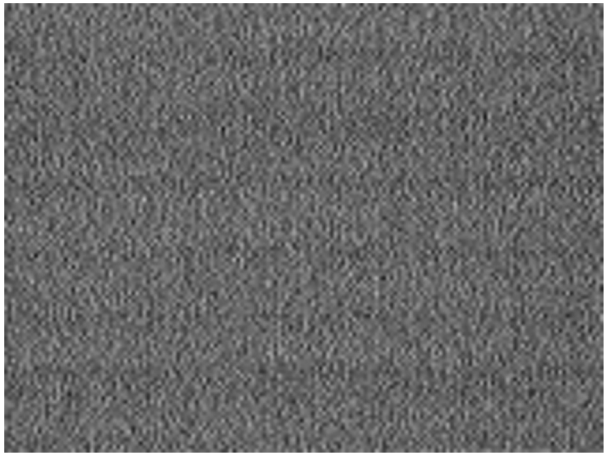

See the Cloudflare blog for more details

# LavaRand Reimagined

UNIVERSITY OF LEEDS

- Uses images from a webcam CCD in complete darkness
- Gain set to high, to reveal thermal noise
- Single 19,200-pixel image yields 340 to 1,420 random bytes, after processing

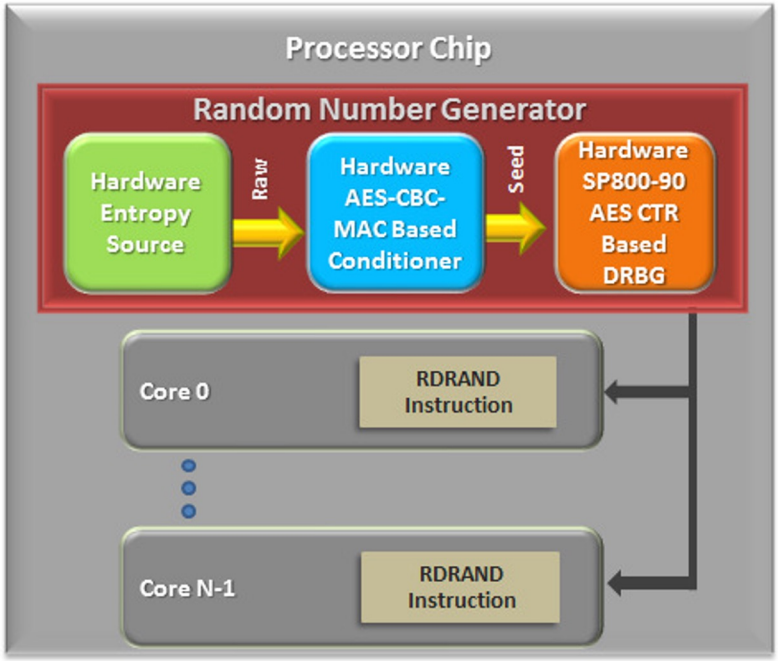


# On-Chip Randomness

UNIVERSITY OF LEEDS

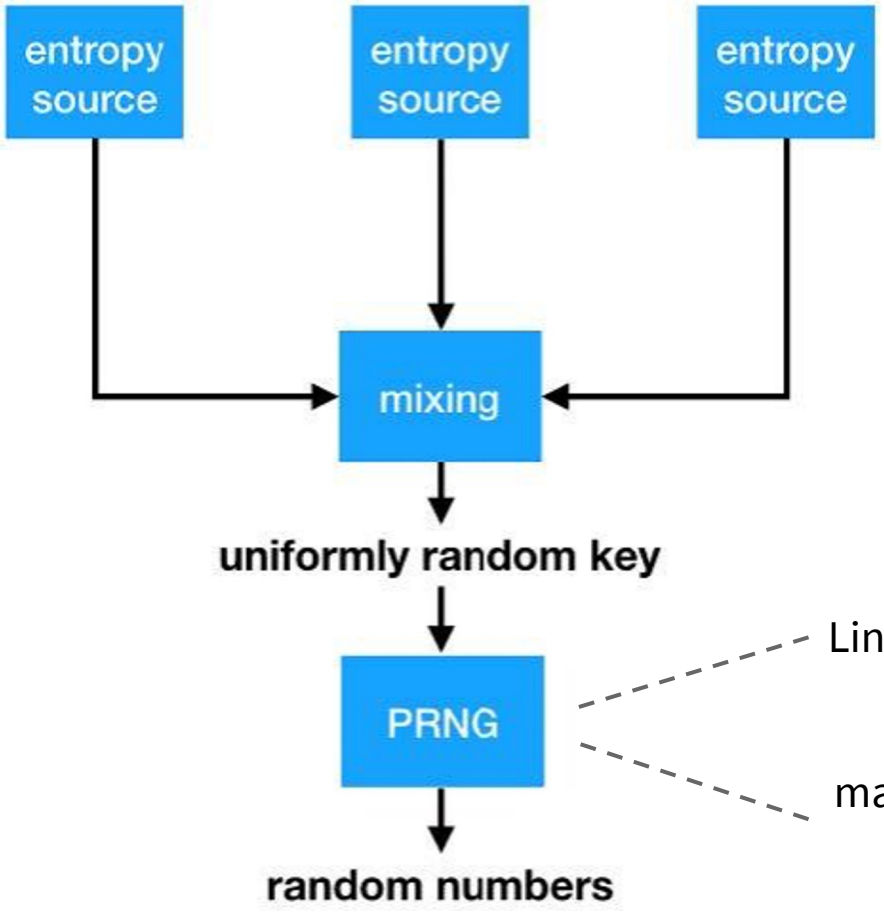
- Available in recent Intel & AMD CPUs
- Uses thermal noise as an entropy source
- RDRAND instruction produces high-quality 128-bit random numbers, issuing at most 511 before reseeding itself
- RDSEED instruction produces random seeds for use with software CSPRNG




(see CrossTalk attack, CVE-2020-0543)

# Randomness From Your OS

UNIVERSITY OF LEEDS




Linux: ChaCha20 stream cipher

macOS (since 2019): Fortuna

# Low-Level Approach

UNIVERSITY OF LEEDS

- Linux and macOS provide good-quality randomness via standard file-like device `/dev/urandom`
- You can open this like any file and read bytes from it
- Example: `urandom.py`

there is also `/dev/random`, which blocks – but you don't really need this

# Higher Level Approaches

UNIVERSITY OF LEEDS

In Python, use the `secrets` module:

```
import secrets
key = secrets.token_bytes(16)    # 16 bytes

rng = secrets.SystemRandom()
print(rng.random())              # float, [0.0,1.0)
```

In Java, use the `SecureRandom` class:

```
SecureRandom rng = new SecureRandom();
byte[] data = new byte[16];
rng.nextBytes(data);
```

# Summary

UNIVERSITY OF LEEDS

We have

- Seen that output from standard PRNGs is insufficiently random for security purposes
- Highlighted real cases where insufficient randomness has been exploited in various ways
- Note that we should use cryptographically-strong PRNGs (good stream cipher or hashed counter)
- … and that these should be seeded using a good source of entropy (OS or external device)

# 24 Deadly Sins…

UNIVERSITY OF LEEDS

SIN 20

WEAK RANDOM NUMBERS

Howard, LeBlanc & Viega, *24 Deadly Sins of Software Security*

# Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- Code examples
- Exercise 10
- Untwister: a seed recovery tool for PRNGs
- Strange attractors & TCP sequence number analysis
- RFC 6528: Defending Against Sequence Number Attacks
- HotBits: random numbers via radioactive decay
- Cloudflare blog post on LavaRand
- Myths about /dev/urandom