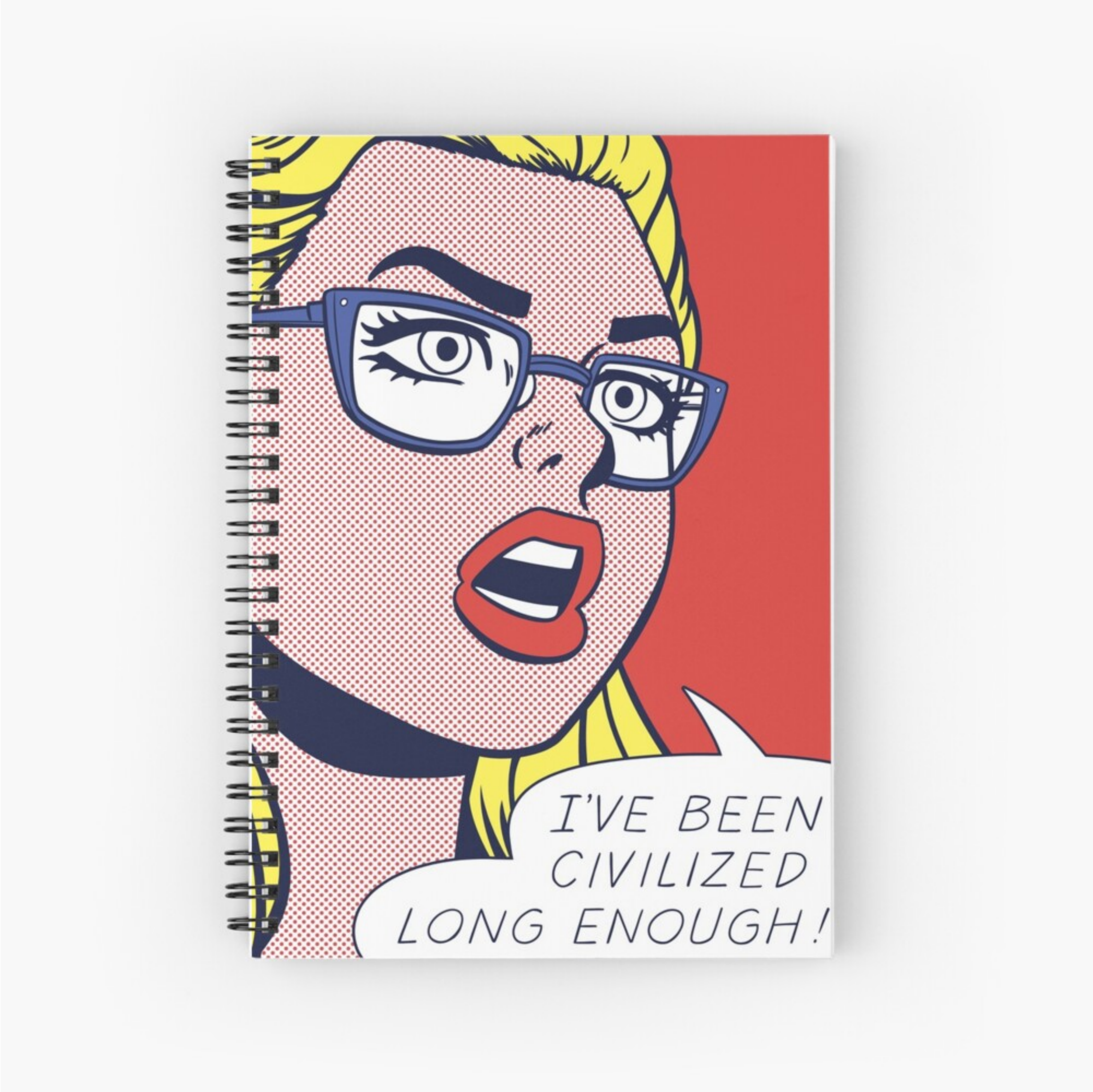
Random Numbers

Applied Cryptography

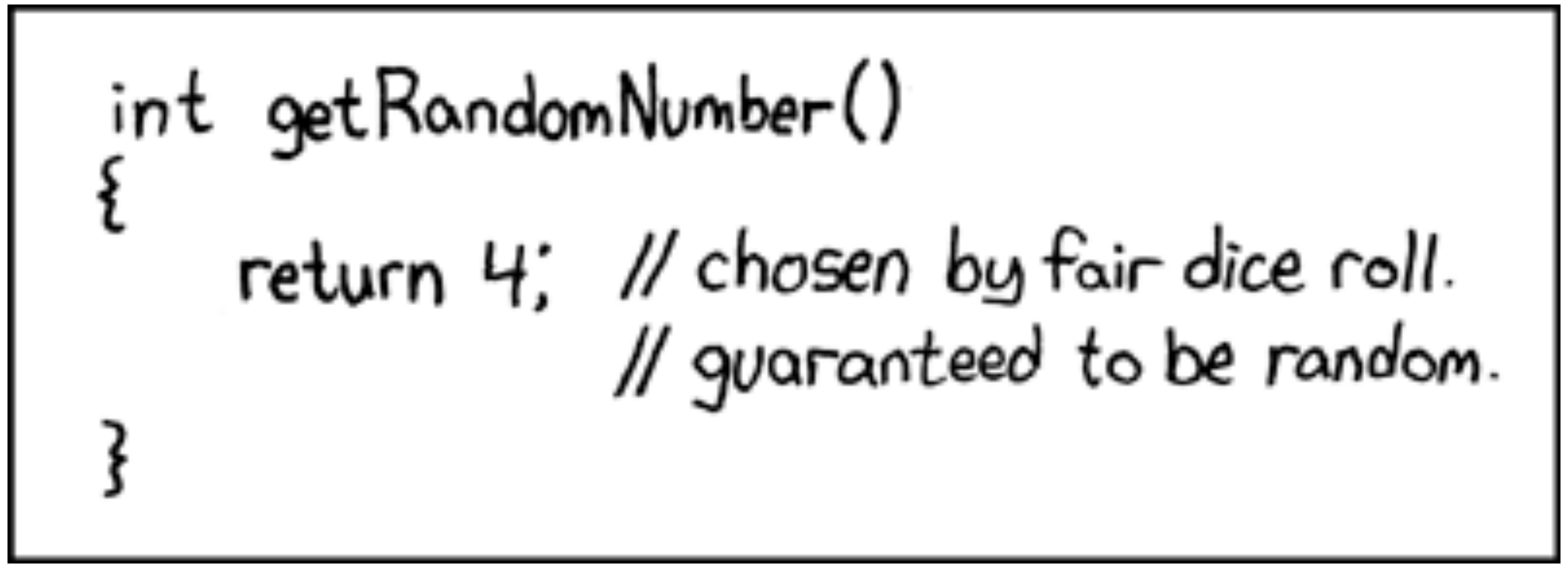
Crypto Primitives



The Four Primitives

- There are four primitives which are considered the building blocks of digital cryptography
  - **Random Number Generation**
  - Symmetric Encryption
  - Asymmetric Encryption
  - Hash Functions
- These primitives get combined to add the CIA (confidentiality, integrity, authentication) properties to data

Random Number Generation



https://xkcd.com/221/

Simple Cipher, Simple Break

- Simple ciphers are not cryptographically secure because they are so quick and easy to break
- Breaking a caesar cipher can be done with trial and error by trying different shifts and looking at the results
- It is easier if you have a useful amount of caesar ciphertext to just analyze the frequency of characters and match it to the language of the plaintext that was encrypted to find the value for the shift (the key)
- Simple ciphers always produce the same ciphertext for any given plaintext/key making frequency analysis fast and reliable
- More sophisticated substitutions require more ciphertext and analysis of additional characteristics of the ciphertext but can still be done fairly quickly with modern computing power since most will produce the same ciphertext every time for a specific plaintext/key

Cryptographically Secure

- A method of protecting data is considered cryptographically secure if the effort to crack it makes cracking it infeasible in a useful timeframe

Random Numbers

- Predictable characteristics of ciphertext, or flaws in algorithms which cause predictable ciphertext patterns that may be exploited to discover keys, are not good for maintaining confidentiality
- Producing multiple ciphertexts from one plaintext is an important tool in making cryptanalysis difficult by eliminating or greatly reducing patterns in ciphertext
  - e.g. rainbow tables can be unusably larger if the result of encrypting a specific plaintext can be many different ciphertexts
- Random numbers are included in secure algorithms to scramble ciphertext in unpredictable ways

Random Number

- A number which cannot be predicted

RNG

- Random Number Generator, a program whose goal is to produce random numbers

Random Number Generation

- Most random number sources in modern operating systems and programming languages do not generate random numbers, they generate pseudo-random numbers
- Random numbers used in cryptography do not have to be truly random, they only need to be unpredictable by an attacker
- Underestimating what is required to be unpredictable is a significant contributor to success in defeating encryption algorithms
- There are many algorithms for producing pseudo-random numbers, all of them balance predictability against speed

PRNG or PSRNG

- Pseudo-random number generator, a program or algorithm designed to produce numbers which are difficult to predict

PRNG Algorithms

- PRNG algorithms generally need a seed value to start with and it is supposed to be random, which is then repeatedly hashed and/or encrypted with pieces of each result carved out and fed back in as the seed for the next iteration
- Some algorithms use a key as a seed
- Event measurements and very high resolution timestamps are often used for seed numbers
- Some CPUs provide instructions to leverage the encryption hardware built into them to create random numbers, because encrypted data is unpredictable, just like a random number is supposed to be
- RNG algorithms are at the heart of crypto and as such they are constantly being probed for ways to break them (i.e. figure out how to predict their output)
- When an algorithm is shown to be predictable, it is no longer considered cryptographically secure

CSPRNG

- Cryptographically secure pseudo random number generator

Seed

- An initial value used as the starting data for a PRNG algorithm, a random number is desired for this

Hash

- The result of applying a formula to data such that output is significantly affected by any changes in the original data

RNG Implementations

- CPUs can provide RNG capabilities, such as the RDRAND instructions in newer Intel CPUs but these are generally untrusted as sole sources because of suspicions of NSA backdoors in the hardware
- Operating systems include facilities for PRNG typically drawing on multiple sources of pseudo-random numbers and combining them
  - UNIX-like systems have /dev/random and /dev/urandom which provide random numbers with different characteristics
  - Windows has SystemPRNG (CNG-based) and ProcessPRNG (BCryptGenRandom-based) exposed via multiple libraries with different characteristics in newer Windows versions, and CryptGenRandom in older Windows versions
  - Consumer embedded systems like routers typically have weak PRNG because they tend to use the same sources with the same starting values (seeds) every time they boot although this is improving
- Programming languages may include their own PRNG and may use operating system facilities to seed them
- There are add-on software packages to do PRNG, and hardware devices which can be added to systems that can provide RNG

Tool Samples

https://www.thisiswhyimbroke.com/the-ultimate-swiss-army-knife/

PRNGs in Linux
Simple examples

```
echo $RANDOM
rand
dd if=/dev/urandom bs=1M count=1 of=/dev/null
openssl rand 1048576|dd of=/dev/null
hpenc -r -b 10M -c 100 |dd bs=10M of=/dev/null
```

- **/dev/random** and **/dev/urandom** can provide a stream of random bits
- **/dev/urandom** is high performance
- **/dev/random** is higher quality
- Variables like **$RANDOM** in shells
- **rand**(1) command
- **random**(3) system call for programs, various programming languages have their own library random number generation functionality
- **openssl**, **hpenc**, etc. can produce random numbers

ent

A CLI tool for measuring entropy in data

```
man ent

man ent | ent

man ent | ent -c

man ent | caesar 3

man ent |
 caesar 3 |
 ent -c |
 awk '/^[0-9]/{print $2,$3}' |
 sort -nk 2 |
 tail 5

dd if=/dev/urandom bs=1M count=1 | ent
```

- Available as a standard package
- Provides several measurements
- Can display frequency tables
- Can provide indicators as to randomness of data
- Can provide indicators of information density
- Useful for cryptanalysis and forensics

rngtest

Part of the rng-tools package

```
dd if=/dev/urandom bs=1M count=1 | rngtest
rngtest: entropy source drained
rngtest: bits received from input: 8388608
rngtest: FIPS 140-2 successes: 418
rngtest: FIPS 140-2 failures: 1
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 1
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=2857142857.143; avg=16793587174.349; max=0.000)bits/s
rngtest: FIPS tests speed: (min=37.326; avg=72.537; max=79.473)Mibits/s
rngtest: Program run time: 119384 microseconds
```

- Another program to evaluate the quality of a random number source
- Runs tests from the FIPS140-2 standard
- Small numbers of what it calls failures are acceptable (e.g. less than 10 for 10MB of random data)
- The standard is from 2001, but still applicable to government agency use approvals for software
- Installing the package also enables rngd, a daemon whose purpose is to increase kernel entropy availability

dieharder

An extensive RNG test suite

*We need all those brilliant Belgian cryptographers to go "alright we know that these encryption algorithms we are using today work, **typically it is the random number generators that are attacked** as opposed to the encryption algorithms themselves. How can we make them [secure], how can we test them?"*
- Edward Snowden in 2014

"... with Intel's design ...
On the positive side, the presence of a PRNG means that the underlying RNG circuit can get pretty borked (e.g., biased) without the results being detectable by your application. On the negative side, *the underlying RNG circuit can get pretty borked without the results being detectable in your application."*
- Matthew Green, same blog article

https://blog.cryptographyengineering.com/2014/03/19/how-do-you-know-if-rng-is-working/

- Produced by Robert G. Brown at Duke University and actively maintained
- GPL license
- Can run a full set of tests or specific tests
- Full tests can take a long time
- Complete overkill for most users of crypto tech
- Provides some of the high level testing needed to uncover bad data sources, bad algorithms, co-opted hardware

openssl

Popular crypto swiss army knife

```
openssl speed rand

openssl rand 1048576 | dd of=/dev/null

openssl rand 1048576 | ent

openssl rand 1048576 | rngtest

openssl rand 1048576 | dieharder -a
```

- openssl can perform many crypto tasks, including PRNG
- Subcommand "rand" followed by how much random data you want
- Not super fast but quite usable, decent quality
- Seriously cross-platform
- Very large user community

RNG Creation

https://twitter.com/MarcSRousseau

DON'T

Your inability to imagine how to break your own creation is in no way a reflection of the abilities of others