Randomness

Aggelos Kiayias

Importance

- Most security related tools and protocols require parties to use “random coins”.
- The security analysis typically assumes that the underlying probability distributions of such sources operate ideally.
  - is this a reasonable assumption to make?

Examples

- In Diffie Hellman key exchange, parties choose random exponents.
- The key to any symmetric cipher such as AES must be selected randomly.
- It is important to include “nonces” into protocols to ensure liveness.
- etc.

Web Attacks

- In most web-services a user can request a "password-reset" operation.
- The new password is mailed to an address the user has.
- This creates the following simple attack : request a password reset and then brute-force login.
  - (if rapid-login attempts are allowed and entropy of new password is low this can work).

see Argyros-Kiayias http://crypto.di.uoa.gr/CRYPTO.SEC/Randomness_Attacks.html

Where to find randomness?

- “Anybody who contemplates arithmetic methods for the generation of random numbers is in a state of sin.” John von Neumann.

Biased coins.

- Suppose you have a coin that is biased.
- How do you simulate a perfectly unbiased coin with it?

62.5%

37.5%

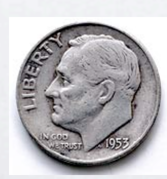
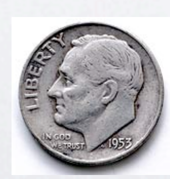
von Neumann's Algorithm

- Flip the coin twice:

case 1:   output 

case 2:   output 

case 3:   repeat

case 4:   repeat

Analysis

- First experiment:
  - Heads = 23.4375%, Tails = 23.4375%
  - no result yet: 53.125%
- With two experiments:
  - Heads ~= 35.988% Tails ~= 35.988%
  - no result yet ~= 28%
- Asymptotically: no result = 0, Heads=Tails=50%

Obtaining Randomness

- A *crystal oscillator* is a circuit that produces signal with a precise frequency. Uses quartz crystals.
  - piezolectric property : electricity causes vibration - vibration causes electricity.
- They are used to make clocks either for measuring time or for synchronizing complex digital circuits.
- **A PC is typically equipped with two such crystals: the CPU clock and the physical clock.**

Truerand

- Part of cryptolib cryptographic library.
- Extracts randomness from the number of times a counter is incremented (cpu clock) in a fixed physical time interval (e.g., 1/60th of a second) (physical clock) this puts the two PC clocks against each other.
- The two clocks have slight discrepancies and thus the counter will produce a different number every time.

Air Turbulence

- Air turbulence inside hard drives is a chaotic phenomenon.
- Reading/writing into the hard disk requires placing the head into the right track of the disk *[Seek Time]*. Then the disk must rotate to the right sector *[Rotational Latency]*.
- A disk-read will require a different amount of time (as measured from the CPU clock) from which one can extract bits of randomness.

User Input

- Events that are generated by user actions:
  - typing.
  - mouse movements.

Specialized Hardware

"True" RNG

Thermal Noise.
e.g. in Intel RNG

Quantum:
transmit
photons through a
semi-reflective mirror :
reflect/pass = 0/1

Cheap Randomness?

- The two basic procedures for randomness (discrepancies between the two clocks or between hard disk reads and a clock) will result in a number of CPU clock-ticks.
- Question: *given the output of such a counter where are the random bits?*
  - *where would you expect the most random area to be?*

/dev/random

- Collect "randomness" from various sources : key strokes, timings, network, mouse movement.
- Estimate "entropy".
- Block call if more bits are requested than available.
  - unblocking alternative : /dev/urandom

Entropy Extraction

- Entropy: how much "uncertainty" exists in a random variable.
- Formula: $-\sum_{x \in \Omega} \Pr[x] \log_2 \Pr[x]$
- Maximizes when all values are equiprobable.
- Given some random variable X that is "somewhat" uncertain define a random variable Y = g(X) that is "more uncertain."

min entropy

- An alternative measure : -log max Pr[X=v].

$$H_\infty(X)$$

it holds : $H_\infty(X) \leq H(X)$

Common Practices

- Form one or more random variables that have some entropy.
- Concatenate them and hash (e.g. `SHA-1,MD5`) to produce the random string.
- This is a very popular but not theoretically backed up practice (and potentially unsound).

Randomness Extraction

- A randomness extractor

$$E : \Omega \rightarrow \{0,1\}^k$$

if $X \leftarrow \mathcal{D}_\Omega$ and $\mathbf{entropy}(X) > c \cdot k$

then $E(X) \approx U$ where $U \leftarrow \{0,1\}^k$

Some caution : there (provably) do not exist randomness extractors that work for arbitrary probability distributions (assuming merely min-entropy)

A simple heuristic

- Consider a random variable X resulting from a counter (e.g., truerand, hard disk air turbulence etc.). Sample twice and

one bit each time

$$X_1 = 110101100101$$
$$X_2 = 110001010010$$

use this as the 1st von Neumann experiment

use this as the 2nd von Neumann experiment

...

note : truerand produces 32 bit output (MSBs very low entropy)

Not sure?

- Use different methods to come up with a number of random strings.
- Combine strings together using a XOR gate.

if any of $X_1, X_2, \ldots, X_N$ is random then $X_1 \oplus X_2 \oplus \ldots \oplus X_N$ is random

Beware : independence

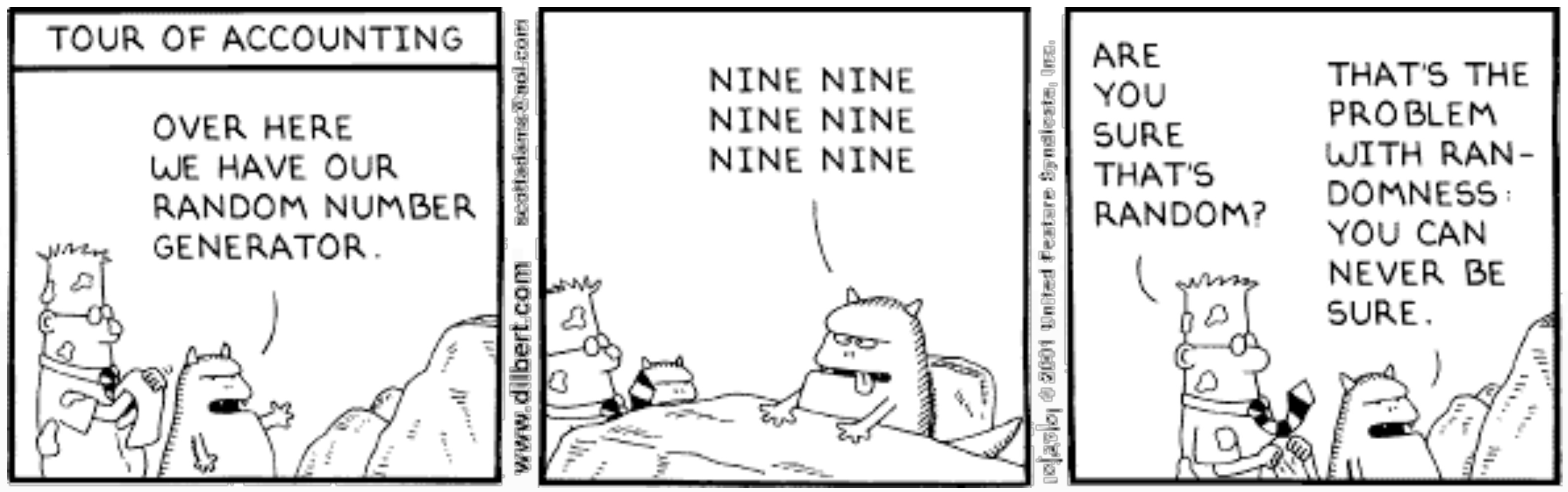
Independence

Observe:

$$0110101110110 1\oplus$$
$$1001010001001 0 =$$
$$1111111111111$$

It is important to XOR independent random sources.

What is randomness?



Is the string 000000000000 “random”?

In the uniform distribution over $\{0,1\}^{12}$

it is equiprobable as any other string

But how often a uniform distribution is observed in nature?

The ***Kolmogorov Complexity*** *of this string is very low.*

Efficiency

- Extracting "randomness" is expensive:
  - probing system events.
  - repeating many times to allow for unbiasing.
  - using specialized hardware.
  - X-ORing independent sources together to obtain good results.

Pseudorandom Number Generation

- Rationale : given that randomness extraction can be expensive follow the following strategy:
  - Use expensive means to extract a very good quality random seed.
  - Use a PRNG on this seed to stretch it to an arbitrarily long random looking sequence.

PRNGs : LCG

- Linear Congruential Generators.
- Recursively defined :

e.g., rand() of glibc used to be such a function

$$x_n = (a \cdot x_{n-1} + b) \mod m$$

What is the maximum period of an LCD ?

$$x_n = x_{n+m-1} \iff (a^{m-1} - 1)x_n + a^{m-2}b + \ldots ab + b = 0$$

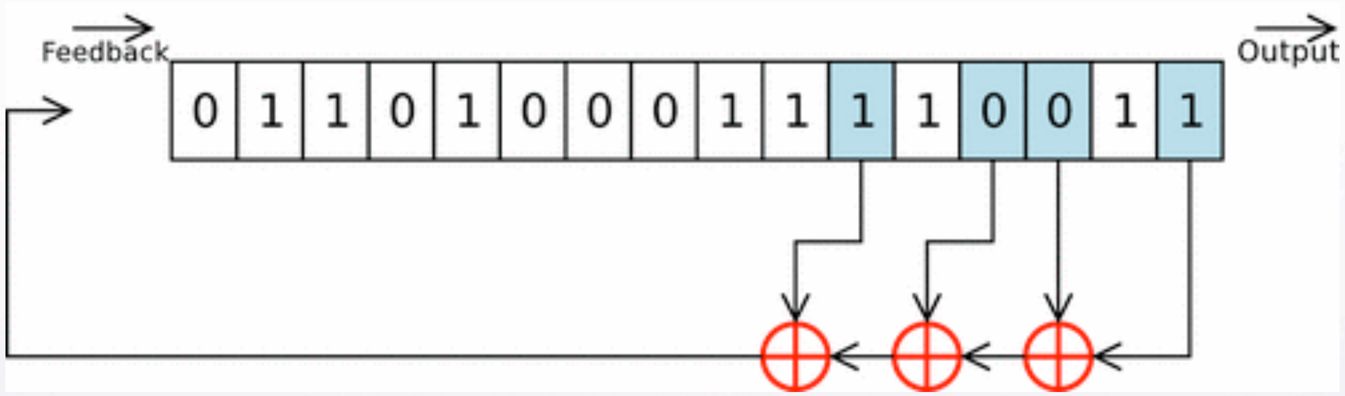
$$\iff (a^{m-1} - 1)(x_n + (a-1)^{-1}b) = 0$$

LCG maximal period

- Can be achieved if m is a prime number.
- LCG not secure for security applications.

PRNGs : LFSRs

- Linear feedback shift registers. Also solvable by a linear system.



- Composition of two linear feedback shift registers (the “shrinking generator”). Alleged of cryptographic quality (unproven).

Provable PRNGs

- A PRNG with a cryptographic proof of pseudorandomness:
- Example the BBS generator:

$$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \ldots$$

$$x_0 = \mathbf{seed}$$

$$x_i = (x_{i-1})^2 \bmod n$$

output

$$\mathbf{LSB}(x_1)\mathbf{LSB}(x_2)\mathbf{LSB}(x_3)\ldots$$

Theorem. Distinguishing output from random results in the recovery of the factors of n