Pseudo-Random Generators

Topics

- Why do we need random numbers?
- Truly random and Pseudo-random numbers.
- Definition of pseudo-random-generator
- What do we expect from pseudo-randomness?
- Testing for pseudo-randomness.
- Example for PRNG algorithm.
- Linux PRNG

Why do we need random numbers?

- Simulation
- Sampling
- Numerical analysis
- Computer programming (e.g. randomized algorithm)
- Elementary and critical element in many cryptographic protocols Usually:
  - "… Alice picks **key K** at random …"
  - Cryptosystems only secure if keys random.
  - Session keys for symmetric ciphers.
  - Nonce in different protocols (to avoid replay)

Cryptography relies on randomness

- To encrypt e-mail, digitally sign documents, or spend a few dollars of electronic cash over the internet, we need random numbers.
- If random numbers in any of these applications are insecure, then the entire application is insecure.

Truly Random Numbers

- Random bits are generated by a hardware that's based on physical phenomena.
- Those numbers cannot be reliably reproduced or predicted.
- Generation of (truly) random bits is an inefficient procedure in most practical systems: slow & expensive.
- Storage and transmission of a large number of random bits may be impractical.

Pseudo-Random Numbers

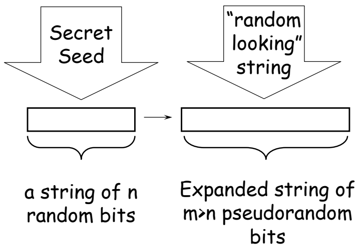
Pseudorandom - Having the appearance of randomness, but nevertheless exhibiting a specific, repeatable pattern.

Random numbers are very difficult to generate, especially on computers which are designed to be deterministic devices.

The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's state.

Pseudo-Random Numbers

- An Efficient (polynomial time) deterministic algorithm G.



Random looking

Random looking means that:

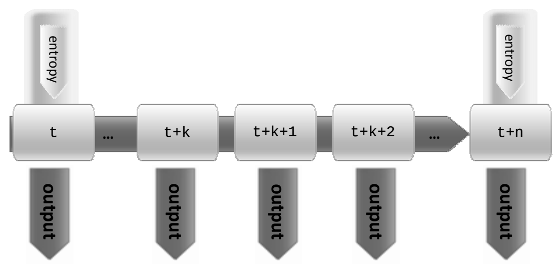
- If the number is in the range: 0…n.
- And there are m numbers to be generated.
- An observer given m-1 out of m numbers, cannot predict the m^{th} number with better probability than 1/n.

The Seed

Can't create randomness out of nothing.

- True physical sources of randomness that cannot be predicted:
  - Noise from a semiconductor device (Hardware).
  - Resource utilization statistics and system load (Software).
  - User's mouse movements.
  - Device latencies.
- Use as a minimum security requirement the length n of the seed to a PRNG should be large enough to make brute-force search over all seeds infeasible for an attacker.

Normal RNG Operation



The difference between Truly Random and Pseudo-Random

If one knows: The algorithm & seed used to create the numbers.

He can predict all the numbers returned by every call to the algorithm.

With genuinely random numbers, knowledge of one number or a long sequence of numbers is of no use in predicting the next number to be generated.

What do we expect from pseudo-randomness?

- Long period : The generator should be of long period (the period of a random number generator is the number of times we can call it before the random sequence begins to repeat).
- Fast computation: The generator should be reasonably fast and low cost.

What do we expect from pseudo-randomness?

- Unbiased: The output of the generator has good statistical characteristics.
- Unpredictable: Given a few first bits, it should not be easy to predict, or compute, the rest of the bits.
- Uncorrelated sequences - The sequences of random numbers should be serially uncorrelated.

Some basic ideas for tests

- Randomness is a probabilistic property: The properties of a random sequence can be characterized in terms of probability.
- The following tests may be applied:
  - Monobit Test: Are there equally many 1's like 0's?
  - Serial Test (Two-Bit Test): Are there equally many 00, 01, 10, 11 pairs?

Linear Congruential Method

Example for PRNG algorithm

Properties of Random Numbers

- Two important statistical properties:
  - Uniformity
  - Independence.
- Random Number, R_i, must be indepen m a uniform distribution with pdf:

$$f(x) = \begin{cases} 1, & 0 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

$$E(R) = \int_0^1 x\,dx = \left.\frac{x^2}{2}\right|_0^1 = \frac{1}{2}$$

Figure: pdf for random numbers

Linear Congruential Method

Techniques]

- To produce a sequence of integers, X_1, X_2, ... between 0 and $m-1$ by following a recursive relationship:

$$X_{i+1} = (aX_i + c) \bmod m, \quad i = 0,1,2,...$$

The multiplier | The increment | The modulus

- The selection of the values for a, c, m, and X_0 drastically affects the statistical properties and the cycle length.
- The random integers are being generated [$0,m-1$], and to convert the integers to random numbers:

$$R_i = \frac{X_i}{m}, \quad i = 1,2,...$$

Examples

[LCM]

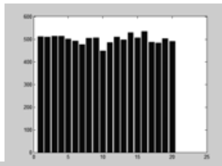
- Use $X_0 = 27$, $a = 17$, $c = 43$, and $m = 100$.
- The X_i and R_i values are:

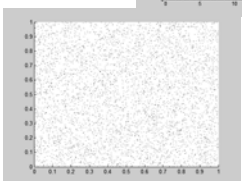
$X_1 = (17*27+43) \bmod 100 = 502 \bmod 100 = 2,$
$R_1 = 0.02;$
$X_2 = (17*2+43) \bmod 100 = 77,$
$R_2 = 0.77;$
$X_3 = (17*77+43) \bmod 100 = 52,$
$R_3 = 0.52;$
$X_4 = (17* 52 +43) \bmod 100 = 27,$
$R_4 = 0.27$

A Good LCG Example

```
X=2456356; %seed value
for i=1:10000,
   X=mod(1664525*X+1013904223,2^32);
   U(i)=X/2^32;
end
edges=0:0.05:1;
M=histc(U,edges);
bar(M);
hold;

figure;
hold;
for i=1:5000,
   plot(U(2*i-1),U(2*i));
end
```





Linear Congruence Generators In Cryptography

- However, even high quality classical generators are mostly not usable in cryptography. Why?
- Because given several successive numbers that were generated by LCG, it is possible to compute the modulus and the multiplier with reasonable efficiency.
- Meaning: there is always the risk of “reverse engineering” of the generators.

RNG Security Requirements

- Pseudo-randomness
  Output is indistinguishable from random
- Backward security
  RNG outputs cannot be compromised by a break-in in the past
- Forward security
  RNG outputs cannot be compromised by a break-in in the future

Pseudo-Random Generators In Cryptography

- If generators are needed in cryptographic applications, they are usually created using the cryptographic primitives, such as:
  – block ciphers
  – hash functions
- There is a natural tendency to assume that the security of these underlying primitives will translate to security for the PRNG.

Linux PNRG

Linux PNRG

- Implemented in the kernel.
  – Entropy based PRNG
- Used by many applications
  – TCP, PGP, SSL, S/MIME, …
- Two interfaces
  – Kernel interface – *get_random_bytes (non-blocking)*
  – User interfaces –
    /dev/random *(blocking)*
    /dev/urandom *(non-blocking)*

Entropy estimation

- A counter estimates the physical entropy in the LRNG
- Increased on entropy addition (from OS events)
- Decreased on data extraction.
- *blocking* and *non-blocking* interfaces
  - *Blocking* interface does not provide output when entropy estimation reaches zero
  - *Non-blocking* interface always provides output
  - Blocking interface is “considered more secure”

Entropy Collection

- Events are represented by two *32*-bit words
  - Event type.
    - E.g., mouse press, keyboard value
  - Event time in milliseconds.
- Bad news:
  - Actual entropy in every event is very limited
- Good news:
  - There are many of these events…

LRNG structure