Lecture Two

Randomness

3. Random and Pseudorandom Bit Generation

"**Random**" numbers have a great many uses e.g., numerical simulations, sampling, numerical analysis, testing computer chips for defects, decision making, coding and cryptography and state machines, and etc. They are valuable resources in some cases, they can speed up computation, they can improve the rate of communication of partial information between two users, and they can also be used to solve problems in synchronous distributed computations that impossible to solve by deterministic means.

A sequence of number is random if each number in the sequence is independent of the proceeding numbers; there are no patterns to help us to predict any number of the sequence. Of course, truly random numbers are hard to come by, or even impossible to get. Thus, the so-called random numbers are actually pseudorandom numbers. Since the invention of the first electronic computer, researchers have been trying to find efficient ways to generate random numbers on a computer.

For our purposes, a sequence generator is pseudo-random if it has this property: **It looks random**. This means that it passes all the statistical tests of randomness that we can find.

Cryptographic applications demand much more of a pseudo-random-sequence generator than do most other applications. Cryptographic randomness does not mean just statistical randomness, although that is part of it. For a sequence to be crypto-graphically secure pseudo-random, it must also have this property: **It is unpredictable**. It must be computationally infeasible to predict what the next random bit

will be, given complete knowledge of the algorithm or hardware generating the sequence and all of the previous bits in the stream.

A (true) random bit generator requires a naturally occurring source of randomness. Designing a hardware device or software program to exploit this randomness and produce a bit sequence that is free of biases and correlations is a difficult task. Additionally, for cryptographic applications, the generator must not be subject to observation or manipulation by an adversary.

A one-way function *f* can be utilized to generate pseudorandom bit sequences by first selecting a random seed s, and then applying the function to the sequence of values s, s+1, s+2, ...; the output sequence is *f*(s), *f*(s+1), *f*(s+2) Depending on the properties of the one-way function used, it may be necessary to only keep a few bits of the output values *f*(s+i) in order to remove possible correlations between successive values.

We shall briefly introduce an example of the methods for generating random numbers based on linear congruence. The following algorithm which is called **Linear Congruential Generation** (LCG) generates a sequence S of random numbers $S=\{x_1,x_2,\ldots,x_k\}$.

INPUT : x_0,a,b,m,k
PROCESS : For j := 1 to k
$x_j := a.x_{j-1} + b \pmod m$
EndFor {j}
OUPUT : the sequence S
END.

Example (3.1):

Let $x_0=5$, a=11, b=73, m=1399 and k=5, then

$x_1=11(5)+73 \pmod{1399} = 128$

x_2=11(128)+73 (mod 1399) = 82

x_3=11(82)+73 (mod 1399) = 975

x_4=11(975)+73 (mod 1399) = 1005

x_5=11(1005)+73 (mod 1399) = 1335

S = {128,82,975,1005,1335}.

3.1 Hardware-Based Generators

Hardware-based random bit generators exploit the randomness which occurs in some physical phenomena. Such physical processes may produce bits that are biased or correlated. Examples of such physical phenomena include:

1. Elapsed time between emission of particles during radioactive decay.
2. Thermal noise from a semiconductor diode or resistor.
3. The frequency instability of a free running oscillator.
4. The amount a metal insulator semiconductor capacitor is charged during a fixed period of time.
5. Air turbulence within a sealed disk drive which causes random fluctuations in disk drive sector read latency times.
6. Sound from a microphone or video input from a camera.

Generators based on the first two phenomena would, in general, have to be built externally to the device using the random bits, and hence may be subject to observation or manipulation by an adversary.

3.2 Software-Based Generators

Designing a random bit generator in software is even more difficult than doing so in hardware. Processes upon which software random bit generators may be based include:

1. the system clock.

2. elapsed time between keystrokes or mouse movement.
3. content of input/output buffers.
4. user input.
5. operating system values such as system load and network statistics.

The behavior of such processes can vary considerably depending on various factors, such as the computer platform. It may also be difficult to prevent an adversary from observing or manipulating these processes. For instance, if the adversary has a rough idea of when a random sequence was generated, he (she) can guess the content of the system clock at that time with a high degree of accuracy. A well-designed software random bit generator should utilize as many good sources of randomness as are available. Using many sources guards against the possibility of a few of the sources failing, or being observed or manipulated by an adversary. Each source should be sampled, and the sampled sequences should be combined using a complex mixing function. The purpose of the mixing function is to distill the (true) random bits from the sampled sequences.

3.3 Requirements for Random Number Generators

Ideally a pseudo-random number generator would produce a stream of numbers that:

1. Are uniformly distributed.
2. Are uncorrelated.
3. Never repeats itself.
4. Satisfy any statistical test for randomness.
5. Are reproduceable (for debugging purposes).
6. Are portable (the same on any computer).
7. Can be changed by adjusting an initial “seed” value.
8. Can easily be split into many independent subsequences.

9. Can be generated rapidly using limited computer memory.

In practice it is impossible to satisfy all these requirements exactly. Since a computer uses finite precision arithmetic to store the state of the generator, after a certain period the state must match that of a previous iteration, after which the generator will repeat itself. Also, since the numbers must be reproducible, they are not truly random, but generated by a deterministic iterative process, and therefore cannot be completely uncorrelated.