Random Number Generation

"Random" numbers are used in:

1. Simulation,
2. Stream ciphers, and
3. Choosing a secret key.

Some desirable properties of a sequence of "random" numbers:

1. The sequence looks random—it passes statistical tests of randomness.

2. The sequence is unpredictable: knowing the algorithm and previous bits, one cannot guess the next bit(s), but perhaps the sequence can be reproduced.

3. The sequence cannot be reliably reproduced: If you run the RNG twice with the same input (as closely as possible), you get two different random sequences. The sequence cannot be compressed.

Examples of RNG's:

1. Linear congruential generators: Fix a (multiplier), b (increment), m (modulus), and x_0 (seed). Define x_i for $i \geq 1$ by

$$x_n = (ax_{n-1} + b) \bmod m.$$

The period is always $\leq m$. One example with maximum period uses $a = 9301$, $b = 49297$, and $m = 233280$. LCG's pass some statistical tests, are okay for simulation, and are efficient. However, they are worthless for cryptography because their linearity makes them easy to break.

2. An LFSR with an n-bit register always has period $\leq 2^n - 1$. It needs a primitive tap polynomial to get maximum period. Note that if $0 < b < a$, then $x^a + x^b + 1$ is primitive if and only if $x^a + x^{a-b} + 1$ is primitive. Example: $a = 31$, $b = 3$.

Some variations use several LFSR's connected by some non-linear muddle: Geffe, Beth-Piper. These are not good sources of cryptographically secure random numbers.

Additive generators such as $X_i = (X_{i-55} + X_{i-24}) \bmod 2^n$, which uses n-bit words and has period $2^{55} - 1$, are insecure.

FISH uses a pair of additive generators and is no good either.

Secure random number generators.

Assume your adversary has a copy of your key-generating program, any master key in it, and knows the time-of-day, process number, machine name, network address, etc., of your program and its machine.

Use as many of the following sources of randomness as possible:

Use the computer's clocks: UNIX seconds since January 1, 1970 (whole number + fractional part to the microsecond). Also set an alarm and increment a counter rapidly until interrupted. Then use the low-order bits of the counter.

Use any available special hardware to produce random bits: Geiger counter with a speck of plutonium, a capacitor to charge, an unstable oscillator, thermal noise, radio static, /dev/audio with no mike attached, disk position or time to read one block.

Use random system values such as CPU load and arrival time of network packets.

If there is a user present, have the user provide randomness by typing on the keyboard, moving the mouse or speaking into the mike.

Hash together (SHA) anything with at least some randomness. (Don't XOR it as was done in Kerberos 4.)

How to use biased "random" bits:

1. XOR several such bits together. Say the bit is 0 with probability $0.5 + e$ and 1 with probability $0.5 - e$, for some $0 \leq e < 0.5$. Then the XOR of two such bits is 0 with probability $(0.5 + e)^2 + (0.5 - e)^2 = 0.5 + 2e^2$. Then the XOR of four such bits is 0 with probability $0.5 + 8e^4$. In the limit when many such bits are XOR'ed, the probability that the XOR will be 0 will converge to 0.5.

2. Use the biased "random" bits in pairs: If the two bits in a pair are the same, skip; else output the first bit.

Neither 1. nor 2. works if adjacent bits are correlated.

Use at least two independent sources of random bits.