Chapter 7
Random Numbers

February 15, 2010

7

In the following random numbers and random sequences are treated as two manifestations of the same thing. A series of random numbers strung together is considered to be a random sequence. A random sequence chopped up into (fixed length?) blocks is considered to give rise to a series of random numbers.

Random numbers are needed in cryptology as:

1. Keys for symmetric algorithms
2. Seeds for finding primes (example: RSA)
3. As sequences to help factorising (Pollard's Monte Carlo Method)
4. In challenge/response mechanisms
5. Sequences for Vernam cyphers etc

We need to generate pseudo-random numbers. We need to be able to test a sequence to see if it is "random" (example: cypher-text).

A random sequence of, say, $N = 2n$ bits should have equal numbers of 0's and 1's. That is 2^{n-1} 0's and 2^{n-1} 1's. The standard deviation around this mean is $\sigma = \frac{1}{4N^{1/2}}$. i.e. Numbers of 0's is equal to the number of 1's $= 2^{n-1} \pm 0.25 * 2^{\frac{n}{2}}$.

• A random sequence should also have no periodicity (zero autocorrelation)

• A random sequence should also have a geometrical distribution of any given pattern, example: k 0's or k 1's.

More specifically, if we define a run length of k 0's as k 0's followed by 1, given a start 10 (so that a run length of 3 0's is ...10001) then the expected value of k, with $Prob(runlength = k) = 2^{-k}$, is $E(k) = 2$.

If there are $2^n (= N)$ bits then (provided $k \leq n - 2$). Number of runs of length k 0's is 2^{n-2-k}, number of runs of length k 1's is 2^{n-2-k} and number of runs of length n 0's or 1's is 1. With, in all, 2^{n-2} runs of 0's and 2^{n-2} runs of 1's.

Example $n = 6,\ N = 64 = 2^6$
We should have

$32{=}2^5$	$0's\ and\ 1's$
$16{=}2^4$	$runs\ of\ 0's\ of\ average\ length\ 1$
	$runs\ of\ 1's\ of\ average\ length\ 1$
8	$runs\ of\ single\ 0's\ or\ 1's$
4	$runs\ of\ two\ 0's\ or\ 1's$
2	$runs\ of\ three\ 0's\ or\ 1's$
1	$run\ of\ four\ 0's\ or\ 1's$
1	$run\ of\ six\ 0's\ or\ 1's$

But these are expectation values. In reality, with really random, sequences these figures will only be approximated.

Really random numbers and sequences cannot be rationally constructed. We construct pseudo-random sequences.

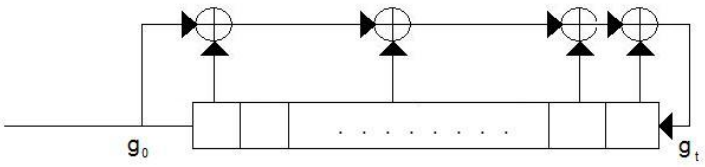
For really random sequences we need a random seed and some random perturbations of some regular pattern.

Random seeds are relatively easily provided - example: by measuring data and intervals between key strokes input at a keyboard or perhaps from a physical source such as a quantum generator. Random perturbations are harder. Perhaps a computer programme may read an independent fine-resolution timer to get some "random" input - provided the programme itself contains a certain fluidity in its progress.

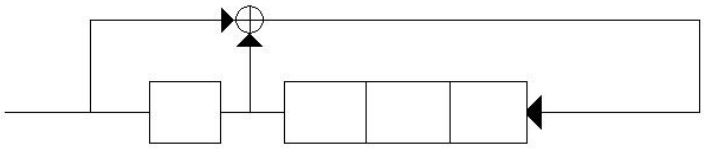
(A pseudo-random sequence which adheres too closely to the supposed ideal of the geometrical distribution of run length of 0's and 1's would be rejected by a chi-square test!)

Some methods for generating pseudo-random sequences:

1. **Maximum length sequence**
   One takes an (irreducible) primitive polynomial $g(x)$ of degree t over $GF(2)$. One forms the linear Feed Back Shift Register (LFBSR)



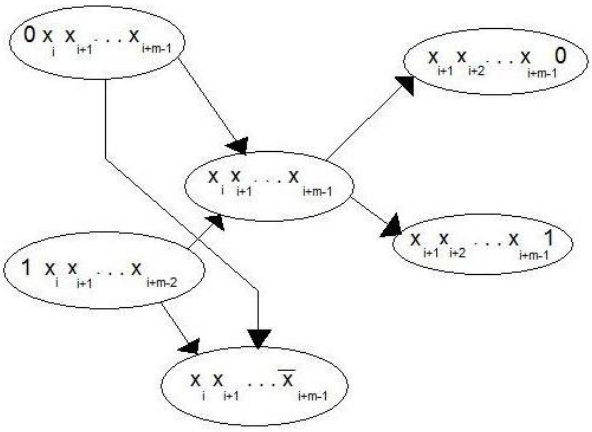
where the connections correspond to the non-zero coefficients in $g(x) = g_t x^t + g_{t-1} x^{t-1} + \ldots + g_1 x + g_0$. The output stream is a Maximum Length Sequence (MLS).



Example $g(x) = x^4 + x + 1$
Note that there are not four 0's (0000) because the scheme is linear and we would always have all zero's. Otherwise we have a geometric distribution. The register contains successively all possible non-zero patterns.

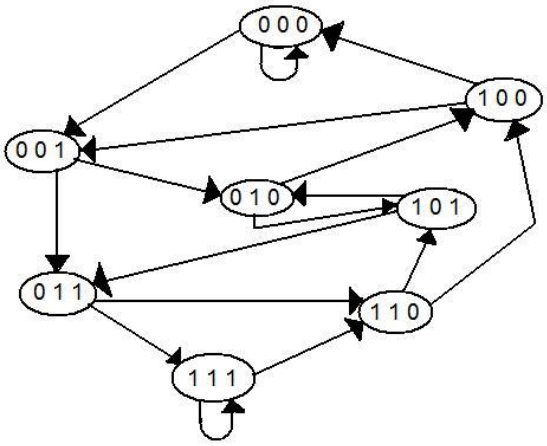
Sequence shifted out $\rightarrow$	$\downarrow$
Content of register is as shifted	0001
	0010
	0100
	1001
	0011
	0110
	1101
	1010
	0101
	1011
	0111
	1111
	1110
	1100
	1000
	.....
	0001
	= 000100110101111 | 00...

2. **De Bruijn Sequences (Non-linear, allow "all-zeros")**
   These are Hamiltonian paths (a path visiting every node once) in a state diagram. We move from state to state designated by $(x_i\ x_{i+1} \ldots x_{i+m-1})$.
   The state has predecessor $(\frac{0}{1}\ x_i\ x_{i+1} \ldots x_{i+m-2})$ and successor $(x_{i+1}\ x_{i+2} \ldots x_{i+m-1}\ \frac{0}{1})$ (and an associate $(x_i\ x_{i+1} \ldots \bar{x}_{i+m-1})$).



The sequence emitted is the (overlapping) sequence of state names $x_{i+m} = f(x_i,\ x_{i+1},\ \ldots\ x_{i+m-1})$ to generate the Hamiltonian path. It can be shown that $f() \equiv x_i + g(x_{i+1}.\ \ldots\ x_{i+m-1})$ if the associate is to be obtainable (and x_i's predecessor unambiguous).

Example



There are two Hamiltonian paths:

000
001
011
111
110
101
010
100

giving 00011101 | 00011101 | 000

and 000
001
010
101
011
111
110
100

giving 00010111 | 00010111 |

It can be shown that in an n-graph (binary) with n bits per state there exist $2^{2^{n-1}-n}$ Hamiltonian paths. (It can also be shown that H_n, a Hamiltonian path in an n-graph, is equivalent to E_{n-1}, an Eulerian path in an $(n-1)$-graph. An Eulerian path visits every side (line) once.)

In our example $n = 3$ so $2^{2^{n-1}-n} = 2$.

3. **Linear Congruential Generator (LCG)**
LCGs generate sequences of pseudo-random numbers *mod* m. The general formula is $x_{n+1} = (ax_n + c)\ mod\ m$ equivalently $x_n = a^n x_0 + \frac{c(a^n-1)}{a-1}\ mod\ m.\ (a \neq 1,\ but\ = x_0 + nc\ if\ a = 1)$.

The question is: What should the parameters $a,\ c$ and m be? We want the sequence of integers to go through the full range *mod* m, i.e. $0,\ 1,\ \ldots\ m-1$ in some permuted order.

Since 0 is included, we may take $x_0 = 0$ and have $x_n = a^n x_0 + \frac{c(a^n-1)}{a-1}\ mod\ m$. Since 1 is included we have $1 = cr - km$ for some $r = (\frac{a^n-1}{a-1}),\ k$. Therefore c and m must be co-prime.

We can choose m prime, but often the form of m is dictated by the application. If m is not prime with a factor d, then we have

$$\begin{aligned} \mathrm{x}'_n &= \mathrm{c}'r' \ mod\ d \\ \mathrm{x}'_n &= \mathrm{x}_n \ mod\ d \\ \mathrm{c}' &= \mathrm{c} \bmod \mathrm{d} \end{aligned}$$

Thus there is an "inner" loop, which if $m = 2^t$ would show itself as repetitive cycles of the least significant bits of x_n! Not very random! So avoid this.

It can be shown that the periodicity of $x_n = \lambda$ with $\lambda = LCM(\lambda_i)$ where $m = \Pi p_i^{e_i}$ and $\lambda_i = periodicity\ mod\ p_i^{e_i}$. To get maximum periodicity we require $\lambda_i = p_i^{e_i}$. It can be shown that this is achieved if $(a-1)$ is divisible by p_i for all i:
Conclusion:

(a) c is relatively prime to m

(b) $(a-1)$ is divisible by all prime factors p_i of m (special case when $p_i = 2$)

Example $m = 45 = 3^2 * 5$
$a = 16$ so $a - 1 = 15 = 3 * 5$
$x_0 = 0,\ x_1 = 7,\ x_2 = 29,\ x_3 = 21,\ etc$
0, 7, 29, 21, 28, 5, 42, 4, 26, 18, 25, 2, 39, 1, 23, 15, 22, 44, 36, 43, 20, 12, 19, 41, 33, 40, 17, 9, 16, 38, 30, 32, 14, 6, 13, 35, 27, 34, 11, 3, 10, 32, 24, 31, 8, (0).

4. **Shuffling** The pseudo-random sequence produced by for example an LCG can be "shuffled" to randomise it. For example one can incorporate the date and time=d, which hopefully flicks over rapidly. One uses a buffer $b(i)$ $i = 1,\ k$ holding LCG output. A possible scheme is (with $x_i \leq m$ the LCG output)

Initialise for	i=1, k b(i)=x_i	
	n=k+1	
Loop : *Set*	t=d mod k	$\leftarrow$
	j=t $\oplus k * \frac{x_n}{m}$	$\uparrow$
Output	b(j)	$\uparrow$
Set	b(j)=x_n	$\uparrow$
	n=n+1	$\rightarrow$

The output is a function of the next x_n read and the time.

There are many other schemes and programs available for generating random numbers. It is important that they generate numbers of at least 2048 bits if they are to be useful cryptographically, and that their 'seed' and any other external input cannot be guessed or generated by an attacker. For example,

with RSA an attacker might find it easier to generate the factors p, q of the modulus n (by running a version of the victim's key-generation program) rather than trying to factorise n.