Pseudo Random Number Generators

Random numbers

- Important:
  - Key generation for PKS
    - Primality testing
  - Key generation for symmetric ciphers
  - Nonces (one-time values)
  - Randomness makes guessing impossible

Requirements on a sequence of RN

- Randomness (statistical)
  1. Uniform distribution: relative frequency curve flat
  2. Independence: no single value can be inferred from others in the sequence
- Unpredictability (practical)
  - Future elements not predictable from earlier
  - Even though the sequence is generated by a deterministic algorithm!

Sources of randomness

- True randomness
  - Physical noise generators
    - Radiation event detectors, etc
    - Impractical, slow, low precision
  - Tables of statistically random numbers
    - Limited in size
    - Predictable
  - Algorithms
    - Deterministic: not statistically random
    - Pseudo-randomness suffices (if good enough)

Requirements on random number generation function

- Should generate full period $[0, m]$ before repeating the sequence
- Should pass reasonable tests on statistical randomness
- Should be efficiently implemented

Linear congruences

- Lehmer, 1951:
  \[ x_{n+1} = (ax_n + c) \mod m, \text{ given } x_0, a, c \text{ and } m \]
- Examples:
  \[ a=c=1 \text{ gives } x_{n+1} = (x_n + 1) \mod m \]
  \[ a=7, c=0, m=32, x_0=1 \text{ gives } \{ 7, 17, 23, 1 \} \]
- If m prime, c=0, som a pass all three tests
  Ex: \( m = 2^{31} - 1, a = 7 \) widely used for statistics
Linear congruences, cont

- Linear congruences are fast, simple, pass requirements
- Linear congruences are predictable
  - Given the parameters $a$, $c$, $m$, a single $x$ makes the rest predictable
  - Given a part of the sequence, parameters can be found
- Ex: given $x_n, x_{n+1}, x_{n+2}, x_{n+3}$
  $x_{n+1} = (ax_n + c) \mod m$
  $x_{n+2} = (ax_{n+1} + c) \mod m$
  $x_{n+3} = (ax_{n+2} + c) \mod m$

Linear Feedback Shift Registers

- $n$-bit shift register that pseudo-randomly scrolls between $2^n-1$ values
- Fast – minimal combinational logic involved
- Shift register $R=(r_r, \ldots, r_0)$ of bits
- Tap sequence $T=(t_r, \ldots, t_0)$ of bits
- Output: $r_t$
- Feedback: $r'_t = c_r$ for $i \in [1, n-1]$
- So, $R'=HR \mod 2$, where $H$ is an $n \times n$ matrix whose first row is $T$ and the rest has $1$ on the subdiagonal, $0$ otherwise

LFSR, cont

- An $n$-bit LFSR generates a pseudo-random bit sequence of length $2^n-1$ if $T$ causes $R$ to cycle through all non-zero values before repeating
- This happens if the polynomial $T(x) = t_n x^n + t_{n-1} x^{n-1} + \ldots + t_1 x^1 + 1$ is primitive
- A primitive polynomial of degree $n$ is an irreducible polynomial that divides $x^n+1$ but not $x^d+1$ for any $d$ that divides $2^n-1$

LFSR for encryption

- LFSR can be used in Vernam ciphers $c_i = m_i \oplus k_i$
- Easily broken: $2n$ pairs of $(c,m)$ sufficient:
  - $m_i \oplus c_i = m_i \oplus (m_i \oplus k_i) = k_i$ for $i \in [1, 2n]$
  - Let $X = (k_1, k_2, \ldots, k_{n-1}, k_n), (k_{n+1}, \ldots, k_{2n-1}, k_{2n})$
  - Let $Y = (k_{n+1}, k_{n+2}, \ldots, k_{2n-1}, k_{2n}), (k_1, \ldots, k_{n-1}, k_n)$
  - $Y = HX \mod 2$, and since $X$ is always nonsingular $r$, $H = YX^{-1} \mod 2$, and $T$ is the first row of $H$.
  - Inverting $X$ is $O(n^3)$: $1$ day for $n=1000, 1$ MIPS

LFSR example

- $T = (1,0,0,1)$
- $H = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
- $T(x) = x^4 + x + 1$ is primitive: given non-zero $R$ it generates all 15 non-zero values of $Z_{16}$
  - 0001, 1000, 1100, 1110, 1111, 0111, 1011, 1010, 1101, 0110, 0111, 1001, 0100, 0101
- Output stream (rightmost bits):
  - 10011110101100

LFSR (cont)

- Combinations of LFSR:
  - Geffe: $z = (a \otimes b) \oplus (-b \otimes c)$
  - where $a = \text{LFSR}(7)$, $b = \text{LFSR}(5)$, $c = \text{LFSR}(8)$
  - gives a period $(2^7-1)(2^5-1)(2^8-1) > 10^6$
  - Still weak: $p(z=a) = \frac{1}{2}$, $p(z=c) = \frac{1}{4}$
  - GSM uses "A5" with LFSRs of length 19, 22, 23
- LFSRs are fast!
Cryptographic RNGs

- In cryptography, we want to reduce redundancy and give minimal information about \( m \) given \( c \)
- Use this for random number generation!
- Examples:
  - Cyclic encryption: \( x_i = E_k(n_i \mod m) \) where \( n_{i+1} = n_i + 1 \)
  - \( n_i \neq n_{i+1}, x_i \neq x_{i+2} \) and decryption without \( k \) is hard, so the sequence is (computationally) unpredictable!
  - E.g., use DES in OFB mode, use PRNG instead of counter

Ansible X9.17 PRNG

- Uses three triple DES encryptions (112-bit key)
- Two "random" sources: date/time and seed
- Feedback of seed value
- Random value \( R_i \) does not reveal seed \( V_{i+1} \)

Blum Blum Shub

- \( p,q \) large primes s.t. \( p \equiv 3 \pmod 4 \)
  - \( n=pq \)
  - \( s \) random s.t. \( \gcd(n,s)=1 \)
- Output: bit sequence \( B_i \)
- \( x_0 = s^2 \mod n \)
  - \( x_i = (x_{i-1})^2 \mod n; \)
  - \( B_i = x_i \mod 2; \)

Blum Blum Shub is a CSPRBG

- The BBS is a cryptographically secure pseudo-random bit generator (CSPRBG): it passes the next-bit test:
  - Given the first \( k \) bits, there is no polynomial algorithm to predict the next bit with probability \( > \frac{1}{2} \)
- Security based on factorization of \( n \)