Pseudorandom Number Generation and Stream Ciphers

Raj Jain
Washington University in Saint Louis
Saint Louis, MO 63130
Jain@cse.wustl.edu

Audio/Video recordings of this lecture are available at:
http://www.cse.wustl.edu/~jain/cse571-11/
Overview

1. Principles of Pseudorandom Number Generation
2. Pseudorandom number generators
3. Pseudorandom number generation using a block cipher
4. Stream Cipher
5. RC4

Pseudo Random Numbers

- Many uses of random numbers in cryptography
  - nonces in authentication protocols to prevent replay
  - keystream for a one-time pad
- These values should be
  - statistically random, uniform distribution, independent
  - unpredictability of future values from previous values
- True random numbers provide this
- Psuedo ⇒ Deterministic, reproducible, generated by a formula
A Sample Generator

\[ x_n = f(x_{n-1}, x_{n-2}, \ldots) \]

- For example,
  \[ x_n = 5x_{n-1} + 1 \mod 16 \]
- Starting with \( x_0 = 5 \):
  \[ x_1 = 5(5) + 1 \mod 16 = 26 \mod 16 = 10 \]
- The first 32 numbers obtained by the above procedure: 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.
- By dividing x's by 16:
  0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125, 0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125.
Terminology

- **Seed** = $x_0$
- **Pseudo-Random**: Deterministic yet would pass randomness tests
- Fully Random: Not repeatable
- **Cycle length**, **Tail**, **Period**

![Diagram showing Seed, Tail, Cycle length, and Period]
Linear-Congruential Generators

- Discovered by D. H. Lehmer in 1951
- The residues of successive powers of a number have good randomness properties.

\[ x_n = a^n \mod m \]

Equivalently,

\[ x_n = ax_{n-1} \mod m \]

\( a = \text{multiplier} \)
\( m = \text{modulus} \)
Lehmer's choices: \( a = 23 \) and \( m = 10^8 + 1 \)

Good for ENIAC, an 8-digit decimal machine.

Generalization:
\[
x_n = a x_{n-1} + b \mod m
\]

Can be analyzed easily using the theory of congruences

\[\Rightarrow\] Mixed Linear-Congruential Generators or Linear-Congruential Generators (LCG)

Mixed = both multiplication by \( a \) and addition of \( b \)
Blum Blum Shub Generator

- Use least significant bit from iterative equation:
  - \( x_i = x_{i-1}^2 \mod n \)
  - where \( n = p \cdot q \), and primes \( p, q = 3 \mod 4 \)
- Unpredictable, passes next-bit test
- Security rests on difficulty of factoring \( N \)
- Is unpredictable given any run of bits
- Slow, since very large numbers must be used
- Too slow for cipher use, good for key generation
Random & Pseudorandom Number Generators

(a) TRNG

(b) PRNG

(c) PRF
Using Block Ciphers as PRNGs

- Can use a block cipher to generate random numbers for cryptographic applications,
- For creating session keys from master key
- CTR
  \[ X_i = E_K[V_i] \]
- OFB
  \[ X_i = E_K[X_{i-1}] \]

(a) CTR Mode
(b) OFB Mode
ANSI X9.17 PRG

Date/Time $D_{T_i}$

Seed $V_i$

Random Stream $R_i$

Keys $K_1, K_2$

Next Seed $V_{i+1}$
Natural Random Noise

- Best source is natural randomness in real world
- Find a regular but random event and monitor
- Do generally need special h/w to do this
  - E.g., radiation counters, radio noise, audio noise, thermal noise in diodes, leaky capacitors, mercury discharge tubes etc
- Starting to see such h/w in new CPU's
- Problems of bias or uneven distribution in signal
  - Have to compensate for this when sample, often by passing bits through a hash function
  - Best to only use a few noisiest bits from each sample
  - RFC4086 recommends using multiple sources + hash
Stream Ciphers

- Process message bit by bit (as a stream)
- A pseudo random **keystream** XOR’ed with plaintext bit by bit
  \[ C_i = M_i \oplus \text{StreamKey}_i \]
- But must never reuse stream key otherwise messages can be recovered
RC4

- A proprietary cipher owned by RSA DSI
- Another Ron Rivest design, simple but effective
- Variable key size, byte-oriented stream cipher
- Widely used (web SSL/TLS, wireless WEP/WPA)
- Key forms random permutation of all 8-bit values
- Uses that permutation to scramble input info processed a byte at a time
RC4 Key Schedule

- Start with an array $S$ of numbers: 0..255
- Use key to well and truly shuffle
- $S$ forms **internal state** of the cipher
  
  for $i = 0$ to 255 do
    
    $S[i] = i$
    
    $T[i] = K[i \mod \text{keylen}]$
  
  $j = 0$
  
  for $i = 0$ to 255 do
    
    $j = (j + S[i] + T[i]) \mod 256$
    
    swap ($S[i]$, $S[j]$)
RC4 Encryption

- Encryption continues shuffling array values
- Sum of shuffled pair selects "stream key" value from permutation
- XOR $S[t]$ with next byte of message to en/decrypt

$$i = j = 0$$

for each message byte $M_i$

$$i = (i + 1) \pmod{256}$$
$$j = (j + S[i]) \pmod{256}$$

swap($S[i], S[j]$)

$$t = (S[i] + S[j]) \pmod{256}$$

$$C_i = M_i \text{ XOR } S[t]$$
RC4 Overview

(a) Initial state of S and T

(b) Initial permutation of S

(c) Stream Generation
1. Pseudorandom number generators use a seed and a formula to generate the next number.
2. Stream ciphers xor a random stream with the plain text.
3. RC4 is a stream cipher.
Homework 7

a. Find the period of the following generator using seed $x_0=1$:

$$x_n = 5x_{n-1} \mod 2^5$$

b. Now repeat part a with seed $x_0=2$

c. What RC4 key value will leave S unchanged during initialization? That is, after the initial permutation of S, the entries of S will be equal to the values from 0 through 255 in ascending order.