Cryptography and Network Security Chapter 7

Fifth Edition
by William Stallings
Lecture slides by Lawrie Brown
(with edits by RHB)

Chapter 7 – Stream Ciphers and Random Number Generation

The comparatively late rise of the theory of probability shows how hard it is to grasp, and the many paradoxes show clearly that we, as humans, lack a well grounded intuition in this matter.

In probability theory there is a great deal of art in setting up the model, in solving the problem, and in applying the results back to the real world actions that will follow.

— The Art of Probability, Richard Hamming

Outline

- pseudorandom number generation
- stream ciphers
- RC4
- true random numbers

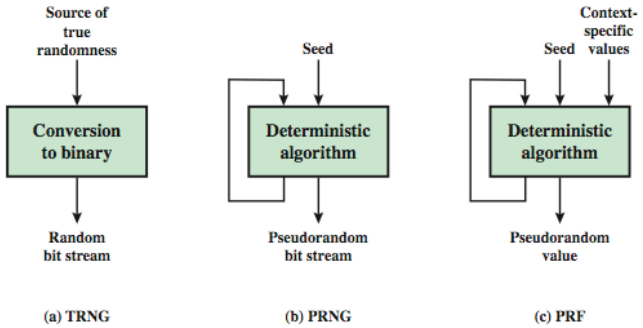
Random Numbers

- many uses of **random numbers** in cryptography
  - nonces in authentication protocols to prevent replay
  - session keys
  - public key generation
  - keystream for a one-time pad
- in all cases its critical that these values are
  - statistically random, uniform distribution, independent
  - have unpredictability of future from previous values
- true random numbers provide this
- care needed with generated random numbers

Pseudorandom Number Generators (PRNGs)

- often use deterministic algorithmic techniques to create "random numbers"
  - although are not truly random
  - can pass many tests of "randomness"
- known as "pseudorandom numbers"
- created by "Pseudorandom Number Generators (PRNGs)"

Random & Pseudorandom Number Generators



PRNG Requirements

- randomness
  - uniformity, scalability, consistency
- unpredictability
  - forward & backward unpredictability
  - same tests used to check both
- characteristics of the seed
  - secure
  - if known adversary can determine output
  - so must be random or pseudorandom number

Linear Congruential Generator

- common iterative technique using:
  $X_{n+1} = (aX_n + c) \bmod m$
- given suitable values of parameters can produce a long random-like sequence
- suitable criteria to have are:
  - function generates a full-period (period always exists)
  - generated sequence should appear random
  - efficient implementation with 32-bit arithmetic
- note that an attacker can reconstruct sequence given a small number of values (knowing $a\ c\ m$)
- have possibilities for making this harder

Blum Blum Shub Generator

- based on public key algorithms
- use least significant bit from iterative equation:

  $x_0 = s^2 \bmod n$

  LOOP $x_i = x_{i-1}^2 \bmod n$

  $b_i = x_i \bmod 2$

  where $n = p.q$, and primes $p, q = 3 \bmod 4$
- 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

Example Operation of BBS

i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

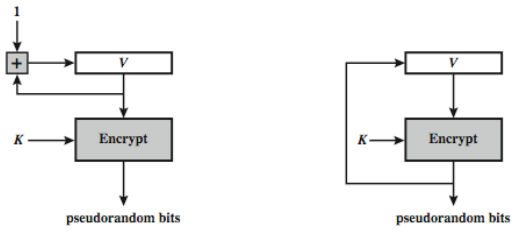
i	X_i	B_i
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0

Using Block Ciphers as PRNGs

- for cryptographic applications, can use a block cipher to generate random numbers
- often for creating session keys from master key
- CTR

  $X_i = E_K[V_i]$
- OFB

  $X_i = E_K[X_{i-1}]$

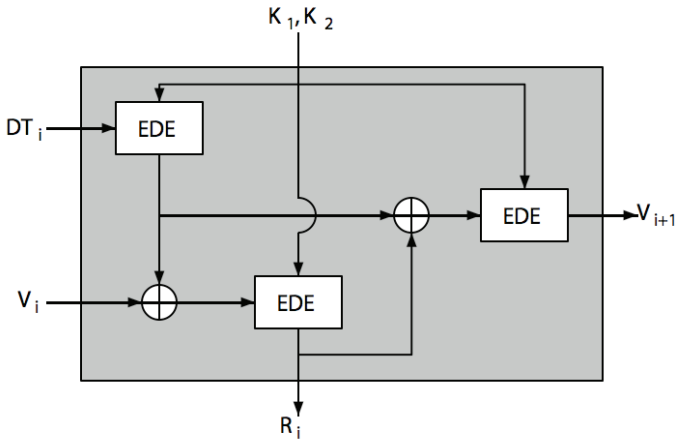


(a) CTR Mode (b) OFB Mode

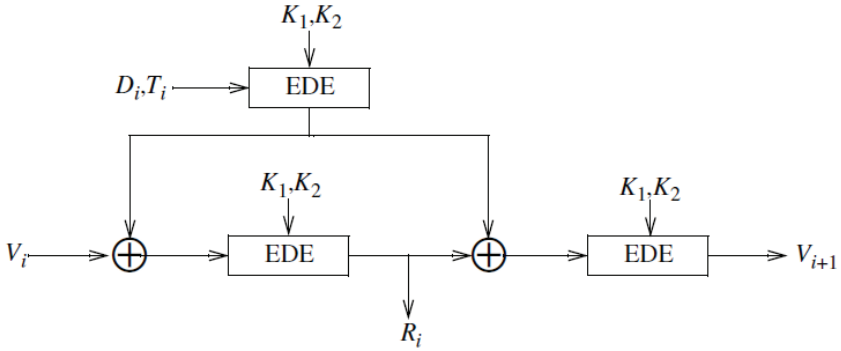
ANSI X9.17 PRG

- a relatively complicated construction, including timestamps, and multiple encryptions
- date and time (DT_i)
- uses 2-key (K_1,K_2) triple DES (3 times per random bit R_i)
- feeds back between rounds (V_i)

ANSI X9.17 PRG



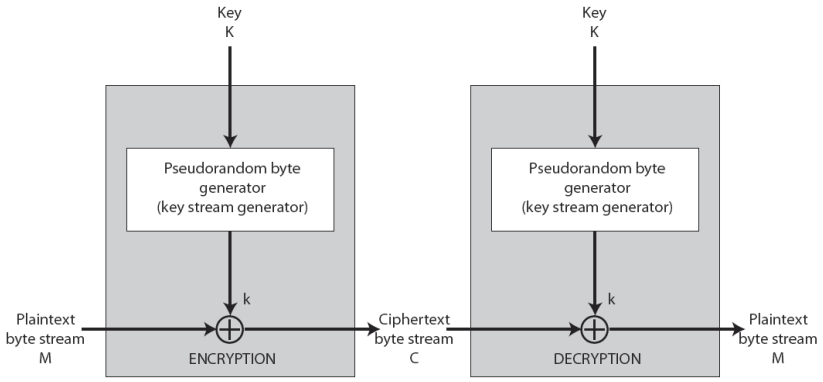
ANSI X9.17 PRNG



Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- Which is combined (XOR) with plaintext bit by bit (cf. one-time pad)
- randomness of **stream key** completely destroys statistically properties in message
  - $C_i = M_i$ XOR $StreamKey_i$
- but must never reuse stream key
  - otherwise can recover messages (via XOR of msgs)

Stream Cipher Structure



Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions of keystream
  - statistically random
  - depends on large enough key
  - large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

RC4

- a proprietary cipher owned by RSA DSI
- a Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- was 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
- these days, known to have vulnerabilities

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle S
- S forms **internal state** of the cipher

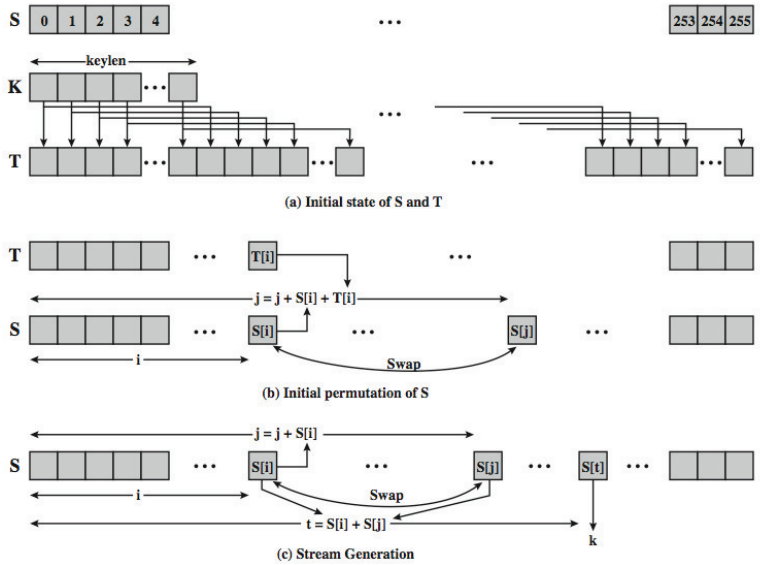
```
for i = 0 to 255 do
   S[i] = i
   T[i] = K[i mod keylen]
j = 0
for i = 0 to 255 do
   j = (j + S[i] + T[i]) (mod 256)
   swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value from permutation
- XOR S[t] with next byte of msg. to en/de-crypt

```
i = j = 0
for each message byte Mi
   i = (i + 1) (mod 256)
   j = (j + S[i]) (mod 256)
   swap (S[i], S[j])
   t = (S[i] + S[j]) (mod 256)
   Ci = Mi XOR S[t]
```

RC4 Overview



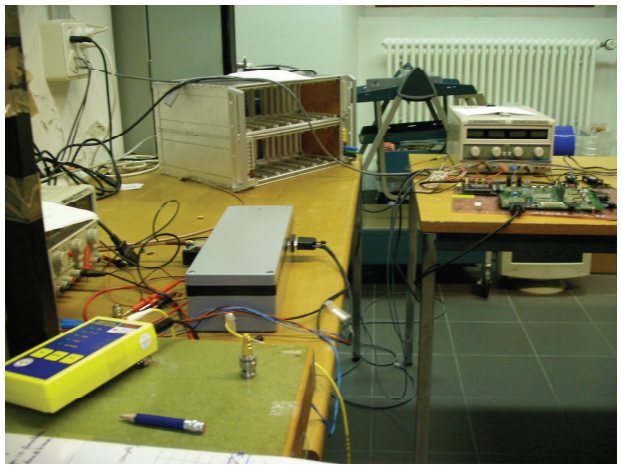
RC4 Security

- claimed secure against known attacks when managed properly
  - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**
- these days, RC4 is known to be biased (unequal numbers of 0s and 1s in the keystream)
- NSA has been breaking RC4 for many years!

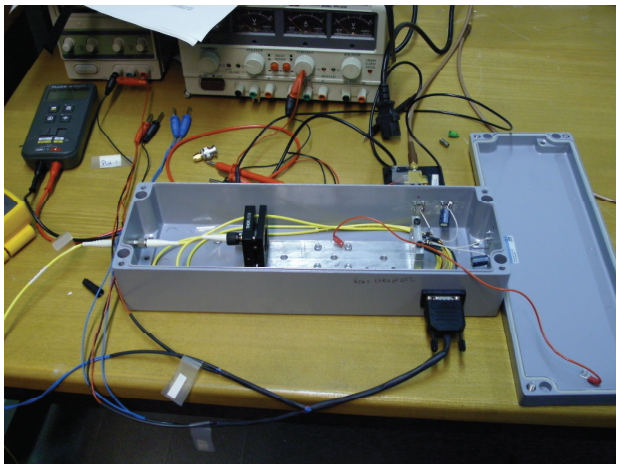
Natural Random Noise

- 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
  - eg. radiation counters, radio noise, audio noise, thermal noise in diodes, leaky capacitors, mercury discharge tubes etc … photon detectors …
- see such h/w in better contemporary 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

Photon Detector TRNG

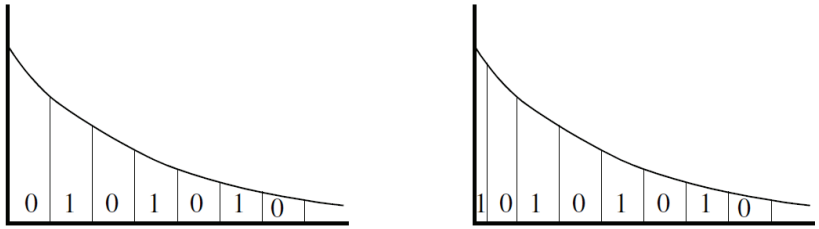


Photon Detector TRNG

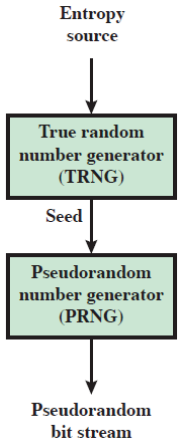


Compensating for physical bias

Exponential decay leads to unequal bins



Using a TRNG to give seed



Published Sources

- a few published collections of random numbers
- Rand Co., in 1955, published 1 million numbers
  - generated using an electronic roulette wheel
  - has been used in some cipher designs cf Khafre
- earlier Tippett in 1927 published a collection
- issues are that:
  - these are limited
  - too well-known for most uses