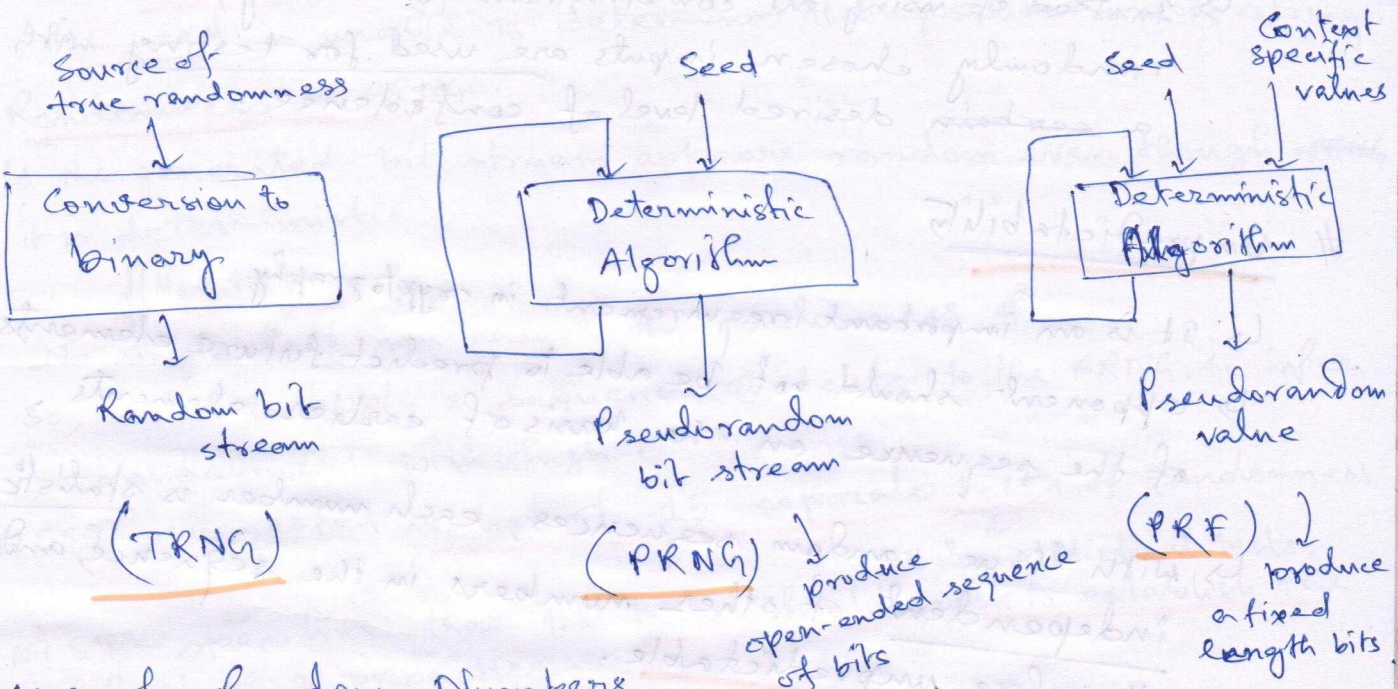
Random Bit Generation (RBG) → Chapter 8 in Stalling's Book



Use of Random Numbers

- ↳ in key distribution
- ↳ in mutual authentication } nonces are used to protect replay attack
- ↳ session key Generation
  - ↳ for a particular transaction. Valid for a short period of time.
- ↳ Generation of keys in RSA
- ↳ Generation of bit stream for symmetric stream cipher e.g. RC4

Randomness

↳ the concern has been that the sequence of number be random in some well-defined statistical sense.

two criteria

Uniform Distribution

↳ i.e. frequency of occurrence of ones and zeros should be approximately equal.

↳ we have well-defined tests to determine this.

Independence

↳ i.e. No sub-sequence in the sequence can be inferred from others.

↳ we don't have such direct tests

↳ we do the negation approach

↳ Perform a number of tests to show the not independence. If it fails for all, then it proves with high confidence that the sequence is independent.

Randomization

↳ Instead of using all combinations for testing, a subset of randomly chosen inputs are used for testing with a ~~certain~~ desired level of confidence.

Unpredictability

↳ It is an important requirement in cryptography.

↳ opponent should not be able to predict future elements of the sequence on the basis of earlier elements.

↳ with "true" random sequences, each number is statistically independent of other numbers in the sequence, and therefore, unpredictable.

Pseudo random number

↳ if any algorithmic technique is used to generate random numbers, it actually generate pseudo random numbers.

TRNG :: Physical source of true randomness

↳ Entropy source is drawn from the physical environment of the computer.

↳ example: keystroke timing, patterns, disk electrical activity, mouse movements, instantaneous values of the system clock, etc.

⇓

these are I/P to TRNG.

So, TRNG simply do the conversion to binary.

PRNG ::

↳ In PRNG, the O/P bit stream is determined solely by the I/P values.

↳ So, who knows the Algo & seed, can produce the entire bit stream again.

PRNG Requirements

↳ the basic requirement is that an adversary who does not know the seed is unable to determine the pseudorandom string.

(1) Randomness in PRNG

↳ the generated bit stream appears random even though it is deterministic.

How to test ?

No single test can ensure the randomness of

So, need to apply a sequence of tests to the PRNG, to infer randomness with high confidence.

↳ NIST suggested a list of 15 separate tests of randomness.
(all are statistical tests.)

↳ NIST specifies that the tests should seek to establish the following 3 properties

(A) Uniformity: At any point in the generation of a sequence, the occurrence of a zero or one is equally likely, that is, the probability of each is exactly 1/2.

(B) Scalability: Any test applicable to a sequence can also be applied to sub-sequences extracted at random.

(C) Consistency: the behavior of a generator must be consistent across starting values (seeds).

(2) Unpredictability in PRNG

- Forward unpredictability
  - ↳ unpredicable the future/next bit from the present bit if seed is not known.
- Backward unpredictability
  - ↳ not feasible to determine the seed from the knowledge of any generated values.

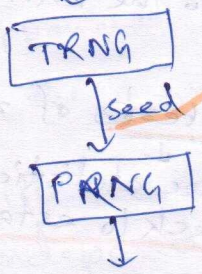
↳ The same set of tests for randomness also provide a test of unpredictability.

e.g. Frequency test — i.e. number of zero & one must be approximately equal.

③ Seed Requirements

↳ The seed must be unpredictable.

↳ In fact, the seed itself must be a random or pseudorandom ~~number~~ number.



Points to ponder

↳ if we use TRNG to generate seed, then what is the need of PRNG?

Ans: w.r.t. ~~cryptography (eg.~~ stream cipher)

the sender should transmit the key to the receiver. (key ← stream cipher)

So, if PRNG is used, only 54 or 128 bits key need to transmit to the receiver.

↑ It is same for PRF also.

this is seed.

Algorithm Design

↳ There are many algorithms.

Roughly 2 categories

(A) → Purpose-built algorithms

(B) → Algo based on existing cryptographic Algorithms.

(A) Purpose-built Algo

↳ Designed specifically and solely for generating pseudo-random bit stream.

↳ These are general purpose algo, provided by operating systems (OS) Either,

↳ or, designed specifically for use in a stream cipher.

↳ e.g. RC4.

(B) Algo based on cryptographic Algo

- Symmetric block cipher
- Assymetric block cipher
- Hash function and Message Authentication Code

PRNG under purpose-built algo.

e.g.
1. → Linear Congruential Generators
2. → Blum Blum Shub Generator

1. Linear Congruential Generators by Lehmer

$$x_{n+1} = (a x_n + c) \bmod m$$

m is the modulus, $m > 0$
a is the multiplier, $0 < a < m$
c is the increment, $0 \leq c < m$
x_0 is the starting value i.e. seed $0 \leq x_0 \leq m$

↳ This technique will produce a sequence of integers with each integer $0 \leq x_n \leq m$

↳ Selection of values for a, c, m is critical in developing good random number generator.

↳ A common criterion is that m to be very large.
  ↳ ≈ maximum representable non-negative integer for a given computer
  ≈ 2^{31}

#↳ Three tests are proposed to evaluate a random number generator

T_1: the function should generate all the number from 0 through $(m-1)$ before repeating.

T_2: generated sequence should appear random

T_3: function should implement efficiently with 32-bit arithmetic

↳ One good choice ⇒ $a = 7^5$, $c = 0$, $m = 2^{31} - 1$

{ Sign ⇒ 31st bit
Exponent ⇒ 23-30 bits
Fraction ⇒ 0-22 bits

Pros: If the multiplier and modulus are properly chosen, the resulting sequence of numbers will be statistically indistinguishable from the sequence drawn at random from the set $1, 2, \ldots (m-1)$.

Cons: If the opponent knows the used algorithm (i.e. linear congruential generator) and has knowledge of a small part of the sequence, it is sufficient to determine the parameters of the algorithm.

Solv:

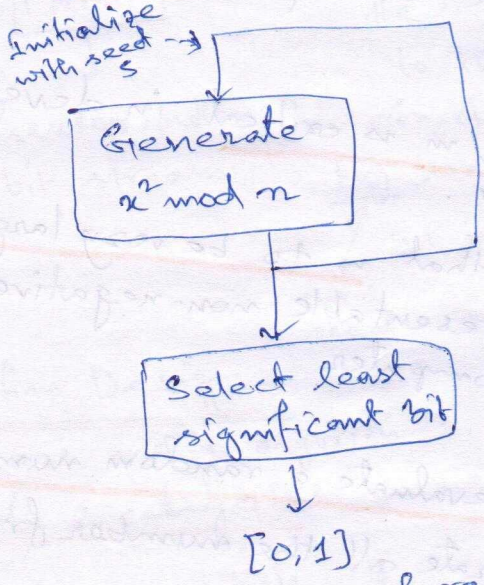
↳ It is desirable to make the actual sequence used ~~non-predi~~ non-reproducible.

↳ How?

↳ by using internal system clock to modify the random number stream.

e.g. restart the sequence after every N numbers using the current clock value (mod m) as the new seed.

(2) Blum Blum Shub Generator (BBS)



↳ what is x and n here?

Approach

(a) First, choose two large prime numbers p and q such that

$$p \equiv q \equiv 3 \pmod 4$$

i.e. $(p \bmod 4) = (q \bmod 4) = 3$ ← This is called congruent modulo (**)

(b) Choose a random number s such that s is relatively prime to n where $n = p \times q$. (++)

In other words, this is equivalent to saying that neither p nor q is a factor of s.

(c) Then BBS produces a sequence of bits B_i as follows:

$$x_0 = s^2 \bmod n$$

for $i = 1$ to ∞

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

$$B_i = x_i \bmod 2$$

(**) Congruent Modulo

Two integers a and b are said to be congruent modulo n, if $(a \bmod n) = (b \bmod n)$. This is written as

$$a \equiv b \pmod n.$$

see chapter 2 (sec 2.3) in Stallings book.

(++) Relatively Prime

Two integers are relatively prime iff their only common positive integer factor is 1.

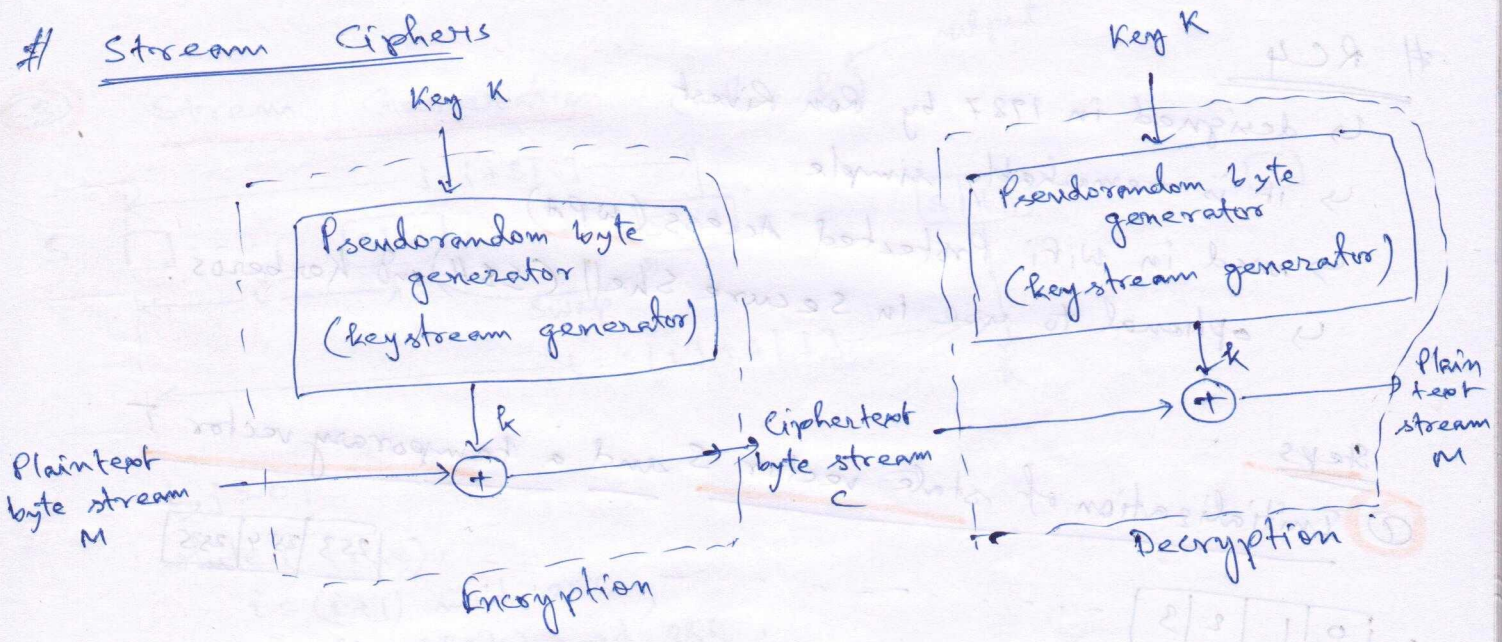
see chapter 2 (sec 2.2) in Stallings book

↳ The BBS is referred as cryptographically secure pseudo-random bit generator (CSPRBG).

↳ CSPRBG should pass the next-bit test.

Next-bit Test: A pseudorandom bit generator is said to pass the next-bit test if there is not a polynomial-time algo that, on input of the first k bits of an o/p sequence, can predict the (k+1)st bit with probability significantly greater than 1/2.

Stream Ciphers



↳ a typical stream cipher encrypts plaintext one byte at a time
↳ although, it may be designed for <1 byte or >1 byte.

↳ A stream cipher can be as secure as a block cipher of comparable key length.

Advantages of stream cipher
↳ typically faster } → modern block cipher has diminished this advantage.
↳ use far less code

Disadvantage
↳ Reuse of key make it more easy to do cryptanalysis.

Applications
↳ which require encryption/decryption of a stream of data eg. over a data communications channel, or a browser/web link.

Design Considerations for a stream cipher

① the encryption sequence should have a large period.
↳ The longer the period of repeat the more difficult it will be to do cryptanalysis.

② the keystream should approximate the properties of true random number stream as close as possible.
↳ there should be almost equal number of 1s and 0s.

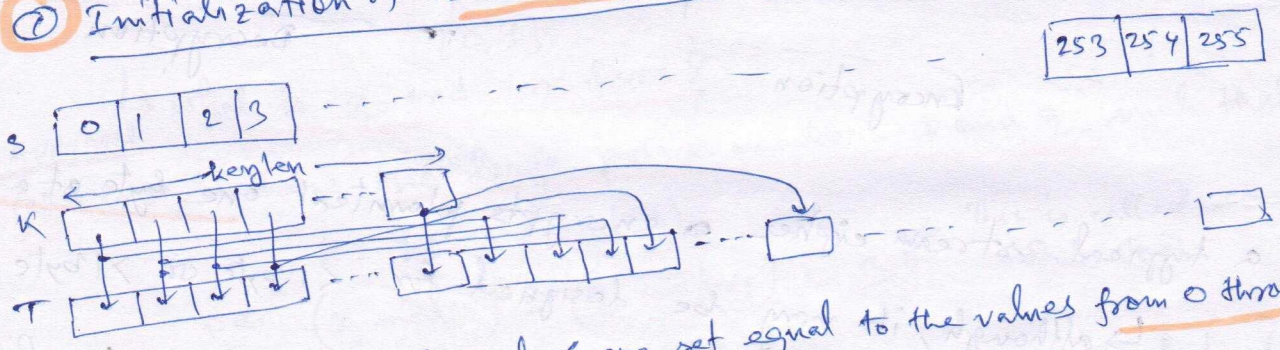
③ the key needs to be sufficiently long to guard against brute-force attack.

RC4

↳ designed in 1987 by Ron Rivest
↳ it is remarkably simple
↳ used in WiFi Protected Access (WPA)
↳ optional to use in Secure Shell (SSH) and Kerberos.

Steps

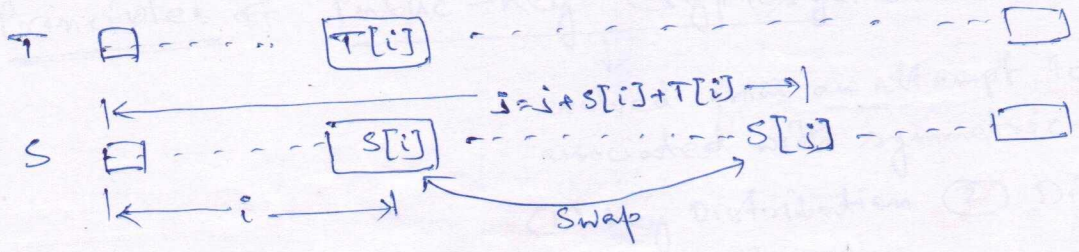
① Initialization of state vector S and a temporary vector T



↳ To begin, the entries of S are set equal to the values from 0 through 255 in ascending order.
↳ Let a key of length keylen bytes.
↳ The first keylen elements of T are copied from key K, and then K is repeated as many times as required to fill out T.

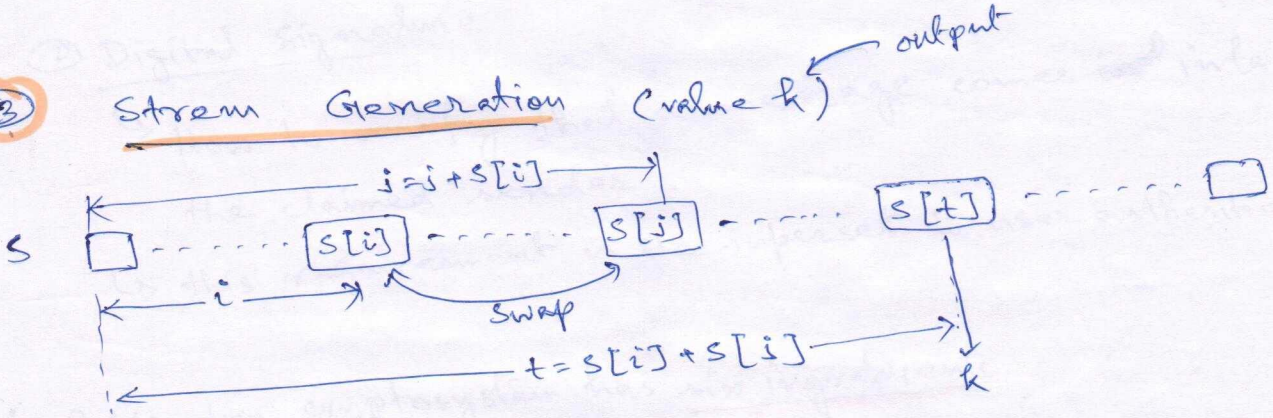
```
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod keylen];
```

(2) Initial Permutations of S (using T)




```
j=0;
for i=0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);
```

(3) Stream Generation (value k)




```
i,j=0;
while (true)
    i = (i+1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    k = S[t];
```

Note: Input key is no longer used in this step.

↳ stream generation (i.e. k) involves cycling through all the elements ~~startin~~ of S[i], (i.e. S[0] ... S[255]).