Cryptography and Network Security

Eighth Edition

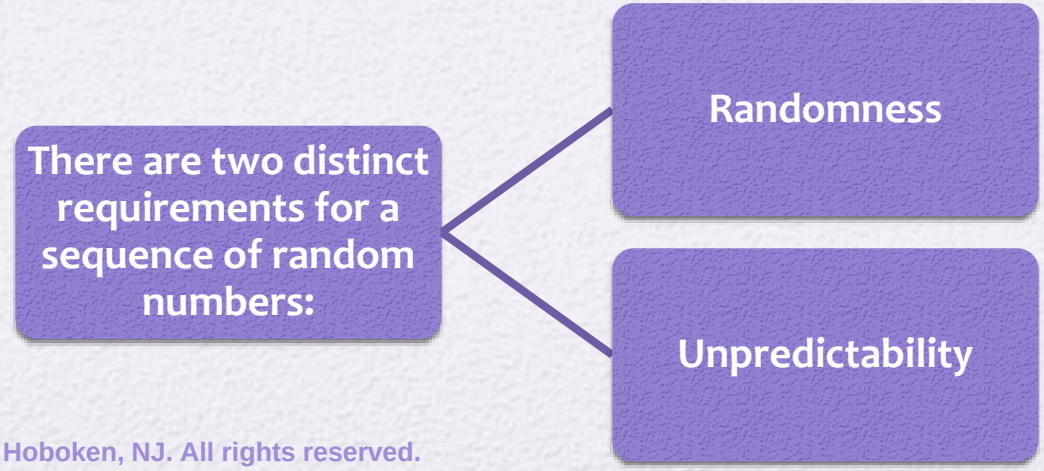
by William Stallings

Chapter 8

Random Bit Generation and Stream Ciphers

Random Numbers

- A number of network security algorithms and protocols based on cryptography make use of random binary numbers:
  - Key distribution and reciprocal authentication schemes
  - Session key generation
  - Generation of keys for the RSA public-key encryption algorithm
  - Generation of a bit stream for symmetric stream encryption



Randomness

- The generation of a sequence of allegedly random numbers being random in some well-defined statistical sense has been a concern

Two criteria are used to validate that a sequence of numbers is random:

Uniform distribution

- The frequency of occurrence of ones and zeros should be approximately equal

Independence

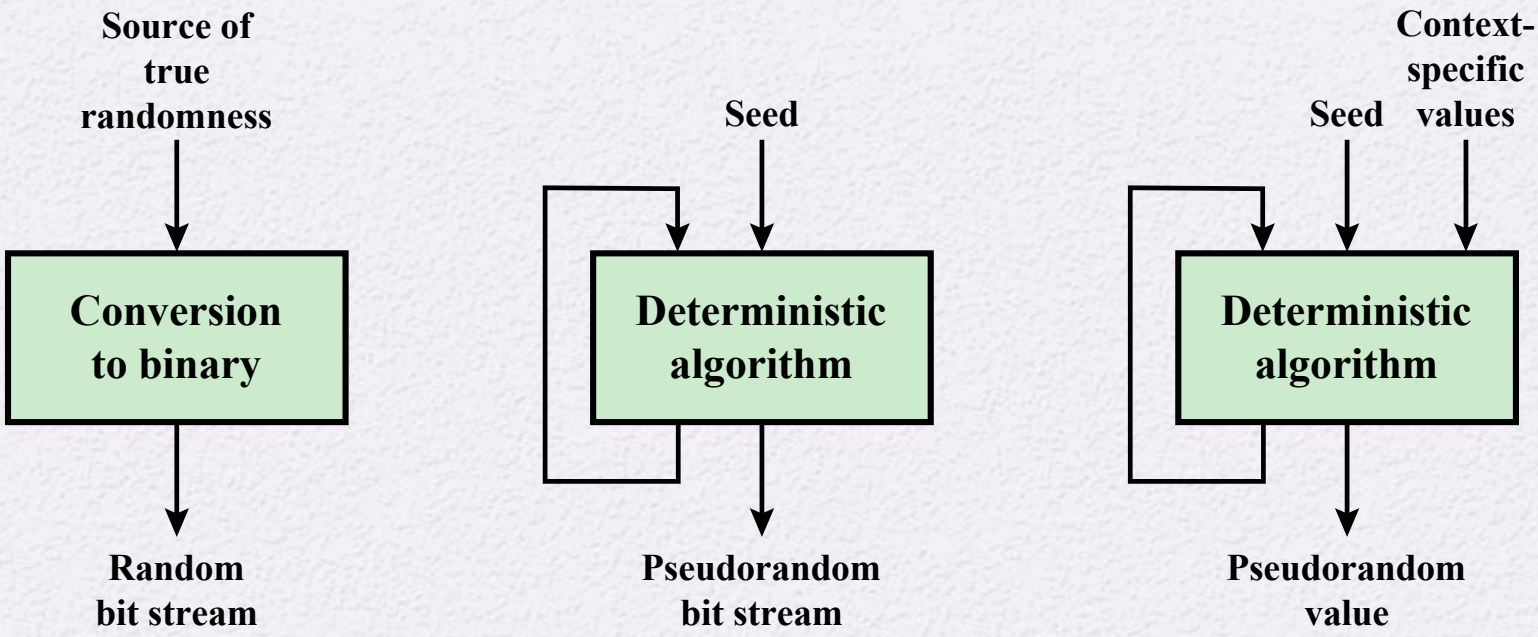
- No one subsequence in the sequence can be inferred from the others

Unpredictability

- The requirement is not just that the sequence of numbers be statistically random, but that the successive members of the sequence are unpredictable
- With "true" random sequences each number is statistically independent of other numbers in the sequence and therefore unpredictable
  - True random numbers have their limitations, such as inefficiency, so it is more common to implement algorithms that generate sequences of numbers that appear to be random
  - Care must be taken that an opponent not be able to predict future elements of the sequence on the basis of earlier elements

Pseudorandom Numbers

- Cryptographic applications typically make use of algorithmic techniques for random number generation
- These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
- If the algorithm is good, the resulting sequences will pass many tests of randomness and are referred to as *pseudorandom numbers*



TRNG = true random number generator
PRNG = pseudorandom number generator
PRF = pseudorandom function

Figure 8.1 Random and Pseudorandom Number Generators

True Random Number Generator (TRNG)

- Takes as input a source that is effectively random
- The source is referred to as an *entropy source* and is drawn from the physical environment of the computer
  - Includes things such as keystroke timing patterns, disk electrical activity, mouse movements, and instantaneous values of the system clock
  - The source, or combination of sources, serve as input to an algorithm that produces random binary output
- The TRNG may simply involve conversion of an analog source to a binary output
- The TRNG may involve additional processing to overcome any bias in the source

Pseudorandom Number Generator (PRNG)

- Takes as input a fixed value, called the *seed*, and produces a sequence of output bits using a deterministic algorithm
  - Quite often the seed is generated by a TRNG
- The output bit stream is determined solely by the input value or values, so an adversary who knows the algorithm and the seed can reproduce the entire bit stream
- Other than the number of bits produced there is no difference between a PRNG and a PRF

Two different forms of PRNG

Pseudorandom number generator

- An algorithm that is used to produce an open-ended sequence of bits
- Input to a symmetric stream cipher is a common application for an open-ended sequence of bits

Pseudorandom function (PRF)

- Used to produce a pseudorandom string of bits of some fixed length
- Examples are symmetric encryption keys and nonces

PRNG Requirements

- The basic requirement when a PRNG or PRF is used for a cryptographic application is that an adversary who does not know the seed is unable to determine the pseudorandom string
- The requirement for secrecy of the output of a PRNG or PRF leads to specific requirements in the areas of:
  - Randomness
  - Unpredictability
  - Characteristics of the seed

Seed Requirements

- The seed that serves as input to the PRNG must be secure and unpredictable
- The seed itself must be a random or pseudorandom number
- Typically the seed is generated by TRNG

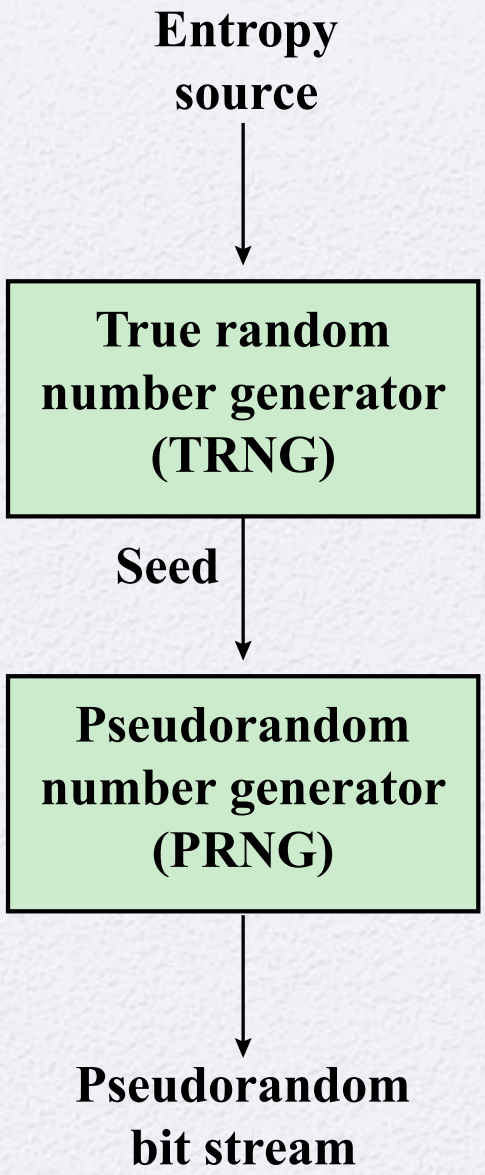


Figure 8.2 Generation of Seed Input to PRNG

Algorithm Design

- Algorithms fall into two categories:
  - Purpose-built algorithms
    - Algorithms designed specifically and solely for the purpose of generating pseudorandom bit streams
  - Algorithms based on existing cryptographic algorithms
    - Have the effect of randomizing input data

Three broad categories of cryptographic algorithms are commonly used to create PRNGs:

- Symmetric block ciphers
- Asymmetric ciphers
- Hash functions and message authentication codes

Linear Congruential Generator

- An algorithm first proposed by Lehmer that is parameterized with four numbers:

m	the modulus	$m > 0$
a	the multiplier	$0 < a < m$
c	the increment	$0 \leq c < m$
X_0	the starting value, or seed	$0 \leq X_0 < m$

- The sequence of random numbers $\{X_n\}$ is obtained via the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m$$

- If m, a, c, and X_0 are integers, then this technique will produce a sequence of integers with each integer in the range $0 \leq X_n < m$

- The selection of values for a, c, and m is critical in developing a good random number generator

Blum Blum Shub (BBS) Generator

- Has perhaps the strongest public proof of its cryptographic strength of any purpose-built algorithm
- Referred to as a *cryptographically secure pseudorandom bit generator* (CSPRBG)
  - A CSPRBG is defined as one that passes the *next-bit-test* if there is not a polynomial-time algorithm that, on input of the first k bits of an output sequence, can predict the $(k + 1)$st bit with probability significantly greater than 1/2
- The security of BBS is based on the difficulty of factoring n

Blum Blum Shub Generator

- based on public key algorithms
- First, choose two large prime numbers, *p* and *q*, that both have a remainder of ***3*** when divided by ***4***.

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

means that (p mod 4) =(q mod 4) = 3

- Let $n = p \times q$.
- Next, choose a random number ***S***, such that ***S*** is relatively prime to ***n***; this is equivalent to saying that neither p nor q is a factor of s.
- Then the BBS generator produces a sequence of bits B_i according to the following algorithm:

$$X_0 = S^2 \bmod n$$
$$for\ i = 1\ to\ \infty$$
$$X_i = (X_{i-1})^2 \bmod n$$
$$B_i = X_i \bmod 2$$

- Thus, the least significant bit is taken at each iteration

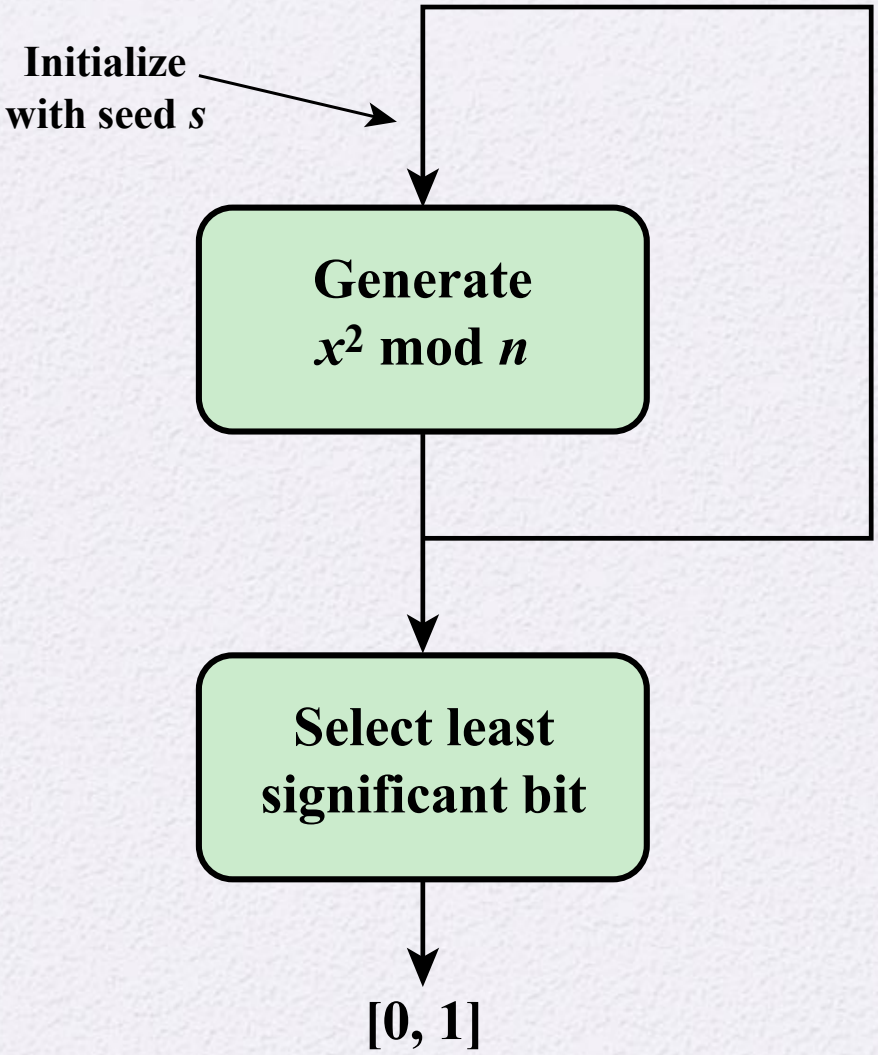


Figure 8.3 Blum Blum Shub Block Diagram

Blum Blum Shub Generator

Example, For Blum Blum Shub with the values p=11, and q=7, and the $\boldsymbol{X_0}$ = 35, find the values of B_1, B_2, B_3.

Solution:

n=11*7=77

$X_i=(X_{i-1})^2 \bmod n$

$B_i= X_i \bmod 2$

$X_1=(X_0)^2 \bmod 77=(35)^2 \bmod 77=70$

$B_1= X_1 \bmod 2=70 \bmod 2= 0$

$X_2=(X_1)^2 \bmod 77=(70)^2 \bmod 77=49$

$B_2= X_2 \bmod 2=49 \bmod 2= 1$

$X_3=(X_2)^2 \bmod 77=(49)^2 \bmod 77=14$

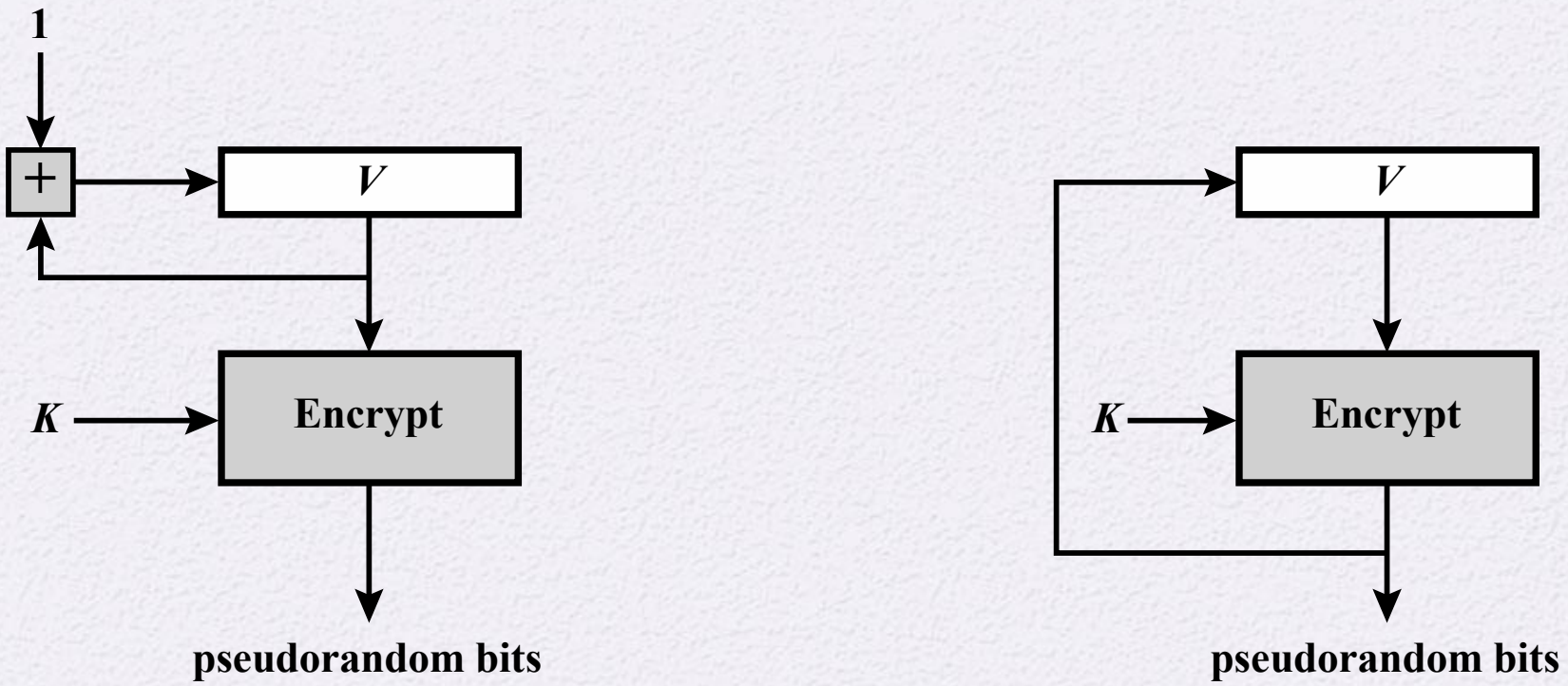
$B_3= X_3 \bmod 2=14 \bmod 2= 0$

Table 8.1 Example Operation of BBS Generator

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
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

PRNG Using Block Cipher Modes of Operation

- Two approaches that use a block cipher to build a PNRG have gained widespread acceptance:
  - CTR mode
    - Recommended in NIST SP 800-90, ANSI standard X.82, and RFC 4086
  - OFB mode
    - Recommended in X9.82 and RFC 4086



(a) CTR Mode

(b) OFB Mode

Figure 8.4 PRNG Mechanisms Based on Block Ciphers

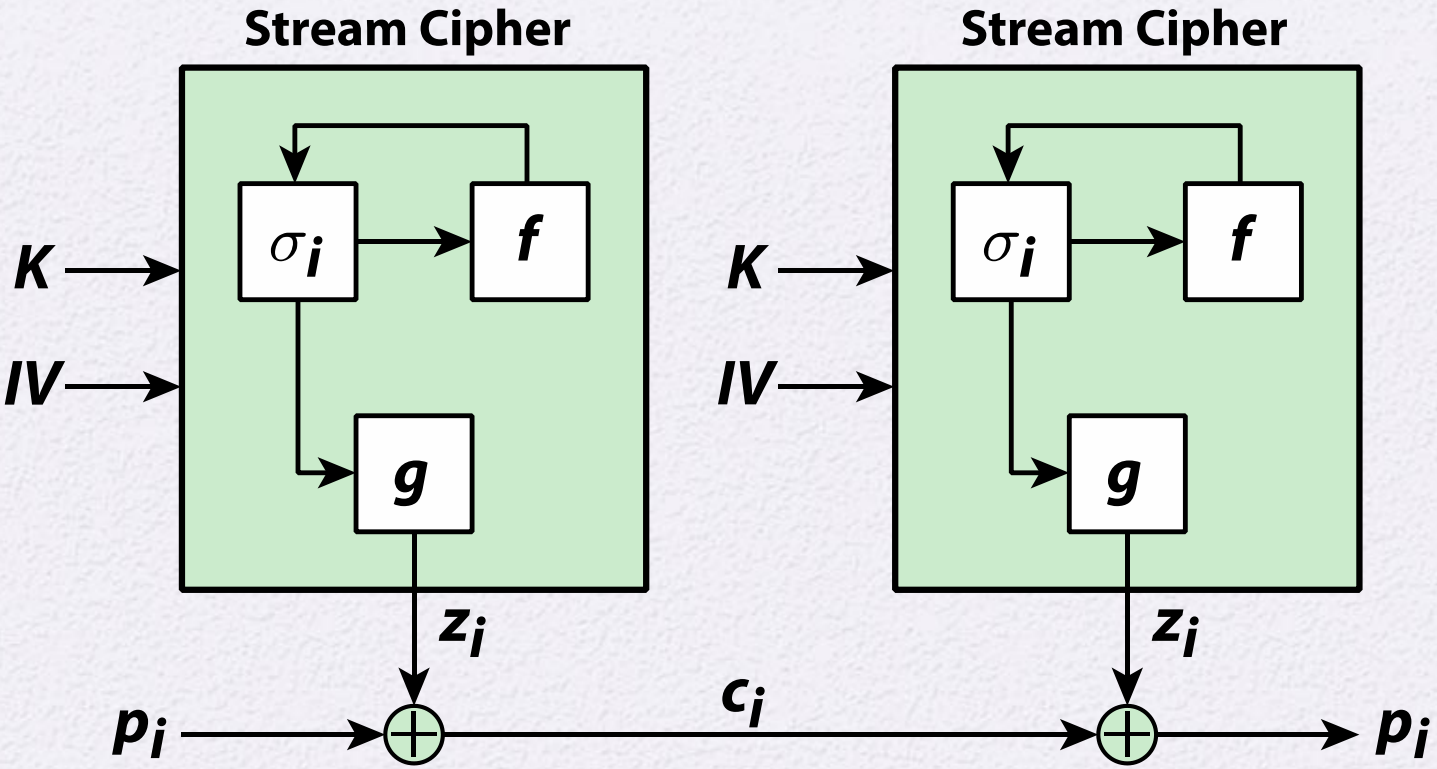
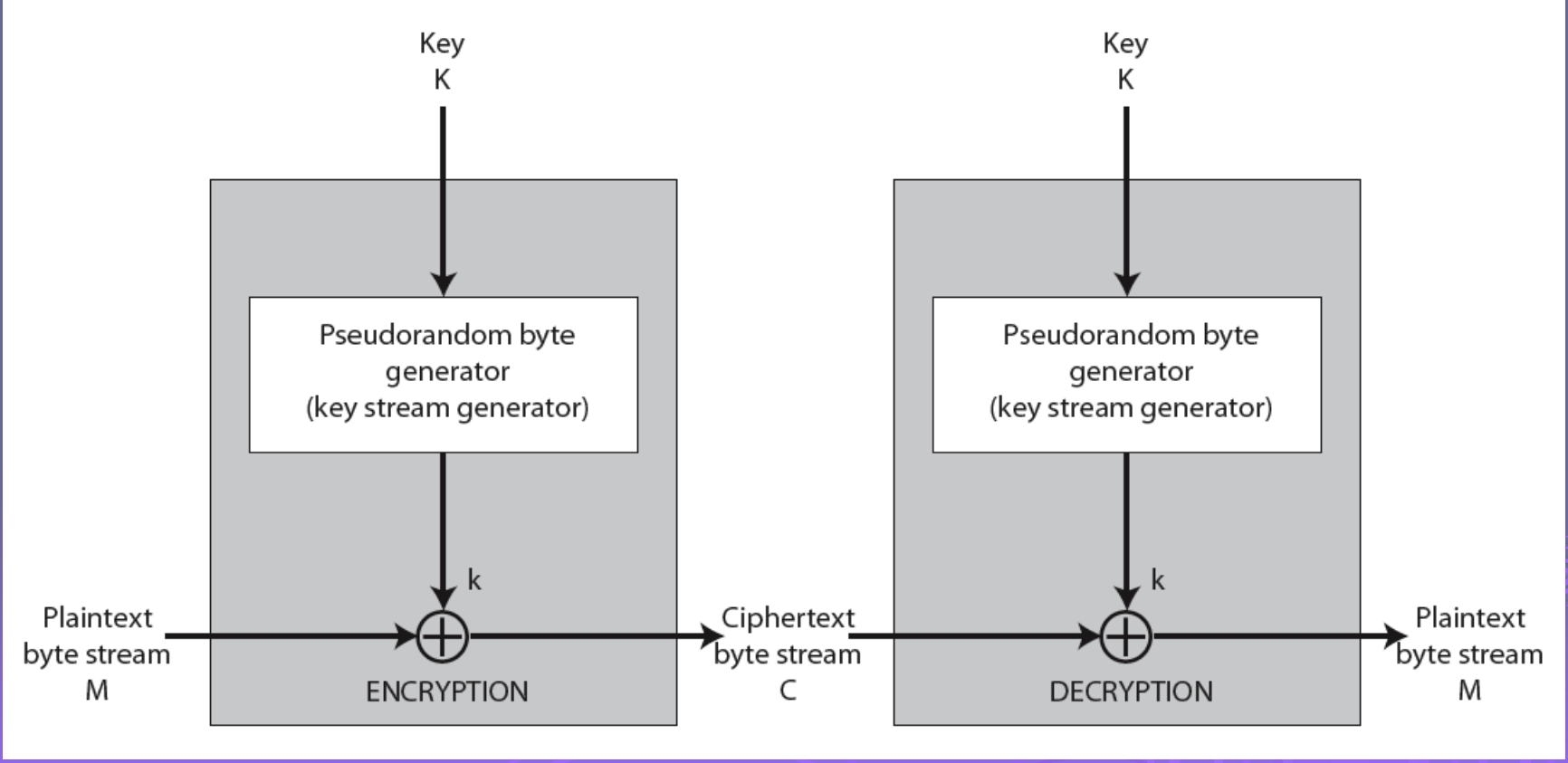


Figure 8.6 Generic Structure of a Typical Stream Cipher

Stream Ciphers

- process message bit-by-bit or byte-by-byte (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message
  - $C_i = M_i \text{ XOR } StreamKey_i$
- but must never reuse stream key
  - otherwise can recover messages (cf book cipher)

Stream Cipher Structure



Stream Cipher Design Considerations

The encryption sequence should have a large period	• A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats; the longer the period of repeat the more difficult it will be to do cryptanalysis
The keystream should approximate the properties of a true random number stream as close as possible	• There should be an approximately equal number of 1s and 0s • If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often
A key length of at least 128 bits is desirable	• The output of the pseudorandom number generator is conditioned on the value of the input key • The same considerations that apply to block ciphers are valid
With a properly designed pseudorandom number generator a stream cipher can be as secure as a block cipher of comparable key length	• A potential advantage is that stream ciphers that do not use block ciphers as a building block are typically faster and use far less code than block ciphers

RC4

- Designed in 1987 by Ron Rivest for RSA Security
- Variable key size stream cipher with byte-oriented operations
- Based on the use of a random permutation
- Eight to sixteen machine operations are required per output byte and the cipher can be expected to run very quickly in software
- RC4 is used in the WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard
- It is optional for use in Secure Shell (SSH) and Kerberos
- RC4 was kept as a trade secret by RSA Security until September 1994 when the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list

RC4 Key Schedule

Initialization of S

- starts with an array S of numbers: 0..255, in ascending order
- use key to well and truly shuffle
- A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K, and then K is repeated as many times as necessary to fill out T.
- S forms **internal state** of the cipher

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

RC4 Key Schedule

Initialization of S

- Next we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255], and for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]:

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

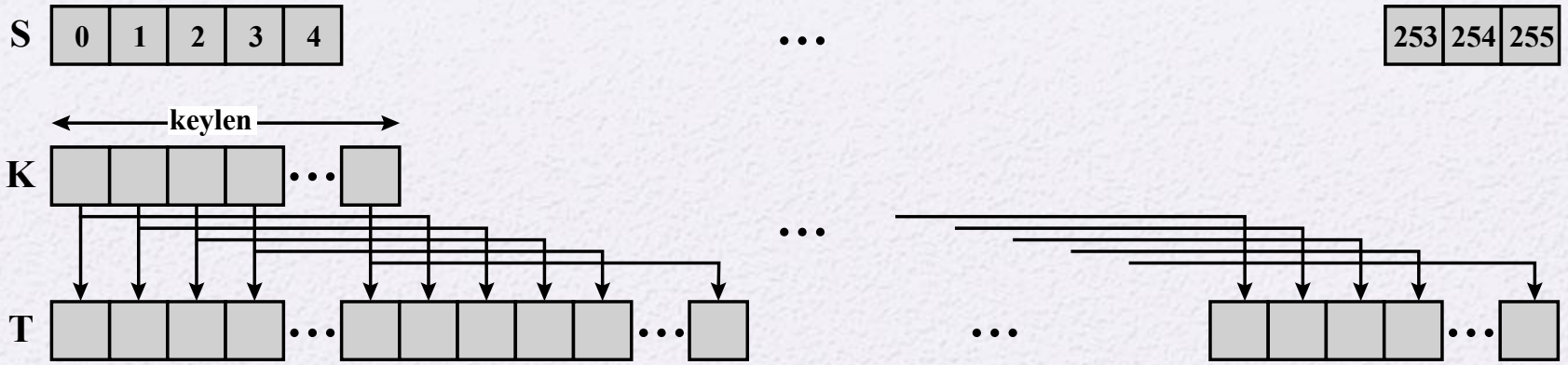
Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255.

RC4 Encryption

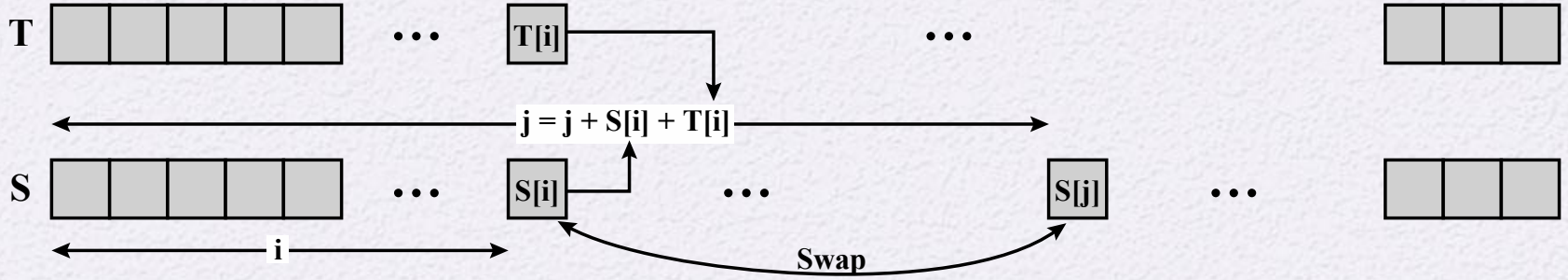
Stream Generation

- **Once the S vector is initialized, the input key is no longer used.**
- Stream generation 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

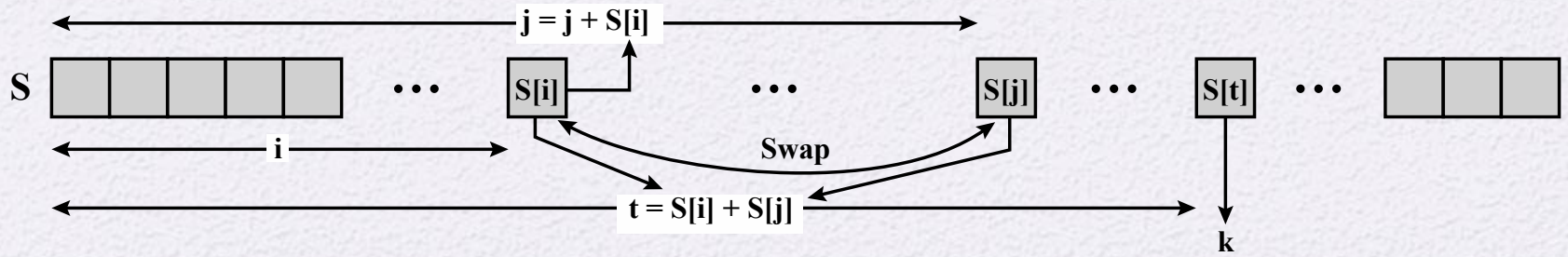
```
 /* Stream Generation */
i, j = 0;
for each message byte Mi
while (true)
  i = (i + 1) mod 256;
  j = (j + S[i]) mod 256;
Swap (S[i], S[j]);
t = (S[i] + S[j]) mod 256;
k = S[t];
Ci = Mi XOR k
```



(a) Initial state of S and T



(b) Initial permutation of S



(c) Stream Generation

Figure 8.7 RC4

Strength of RC4

- A fundamental vulnerability was revealed in the RC4 key scheduling algorithm that reduces the amount of effort to discover the key
- Recent cryptanalysis results exploit biases in the RC4 keystream to recover repeatedly encrypted plaintexts
- As a result of the discovered weaknesses the IETF issued RFC 7465 prohibiting the use of RC4 in TLS
- In its latest TLS guidelines, NIST also prohibited the use of RC4 for government use

Stream Ciphers Using Feedback Shift Registers

With the increasing use of highly constrained devices there has been increasing interest in developing new stream ciphers that take up minimal memory, are highly efficient, and have minimal power consumption requirements

Most of the recently developed stream ciphers are based on the use of feedback shift registers (FSRs)

- FSRs exhibit the desired performance behavior, are well-suited to compact hardware implementation, and there are well-developed theoretical results on the statistical properties of the bit sequences they produce
  - An FSR consists of a sequence of 1-bit memory cells
  - Each cell has an output line, which indicates the value currently stored, and an input line
  - At discrete time instants, known as clock times, the value in each storage device is replaced by the value indicated by its input line
  - The effect is as follows: The rightmost (least significant) bit is shifted out as the output bit for this clock cycle; the other bits are shifted one bit position to the right; the new leftmost (most significant) bit is calculated as a function of the other bits in the FSR

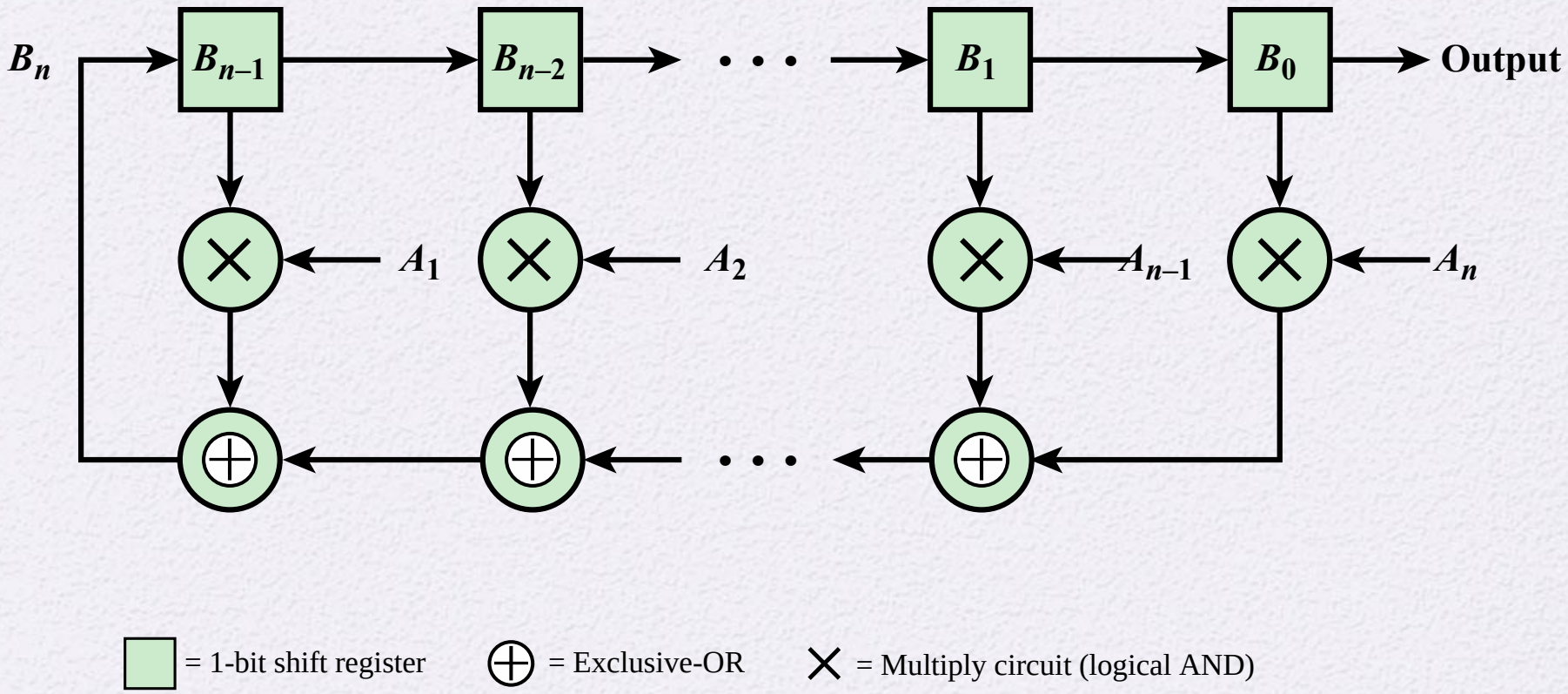
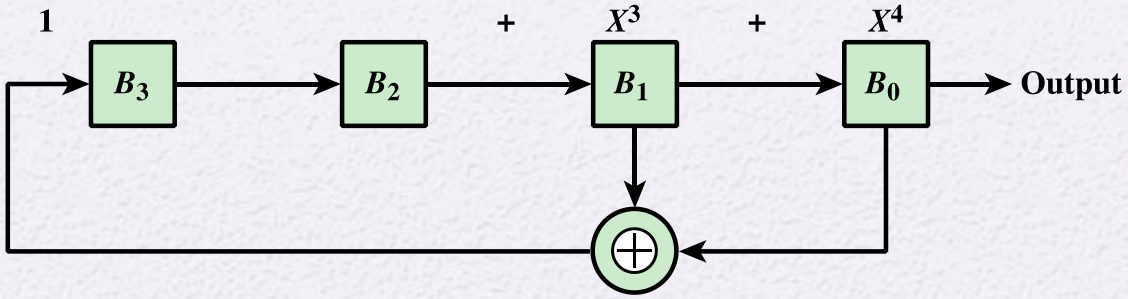


Figure 8.8 Binary Linear Feedback Shift Register Sequence Generator



(a) Shift-register implementation

State	B_3	B_2	B_1	B_0	$B_0 \oplus B_1$	output
Initial = 0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	1	0
3	1	0	0	1	1	1
4	1	1	0	0	0	0
5	0	1	1	0	1	0
6	1	0	1	1	0	1
7	0	1	0	1	1	1
8	1	0	1	0	1	0
9	1	1	0	1	1	1
10	1	1	1	0	1	0
11	1	1	1	1	0	1
12	0	1	1	1	0	1
13	0	0	1	1	0	1
14	0	0	0	1	1	1
15 = 0	1	0	0	0	0	0

(b) Example with initial state of 1000

Figure 8.9 4-Bit Linear Feedback Shift Register

Entropy Sources

- A true random number generator (TRNG) uses a nondeterministic source to produce randomness
- Most operate by measuring unpredictable natural processes such as pulse detectors of ionizing radiation events, gas discharge tubes, and leaky capacitors
- Intel has developed a commercially available chip that samples thermal noise by amplifying the voltage measured across undriven resistors
- LavaRnd is an open source project for creating truly random numbers using inexpensive cameras, open source code, and inexpensive hardware
  - The system uses a saturated CCD in a light-tight can as a chaotic source to produce the seed; software processes the result into truly random numbers in a variety of formats

Possible Sources of Randomness

RFC 4086 lists the following possible sources of randomness that can be used on a computer to generate true random sequences:

Sound/video input

- The input from a sound digitizer with no source plugged in or from a camera with the lens cap on is essentially thermal noise
- If the system has enough gain to detect anything, such input can provide reasonable high quality random bits

Disk drives

- Have small random fluctuations in their rotational speed due to chaotic air turbulence
- The addition of low-level disk seek-time instrumentation produces a series of measurements that contain this randomness

There is also an online service (random.org) which can deliver random sequences securely over the Internet

Table 8.5

	Pseudorandom Number Generators	True Random Number Generators
Efficiency	Very efficient	Generally inefficient
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic

Comparison of PRNGs and TRNGs