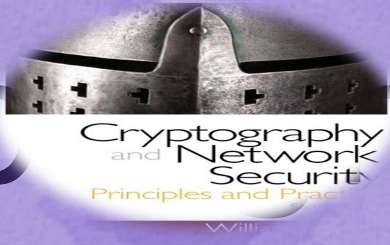
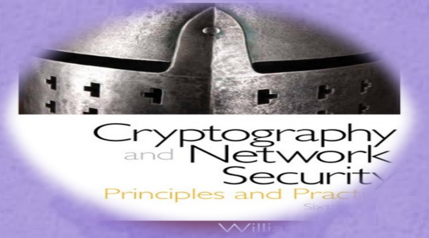


Cryptography and Network Security

Eighth Edition

by William Stallings

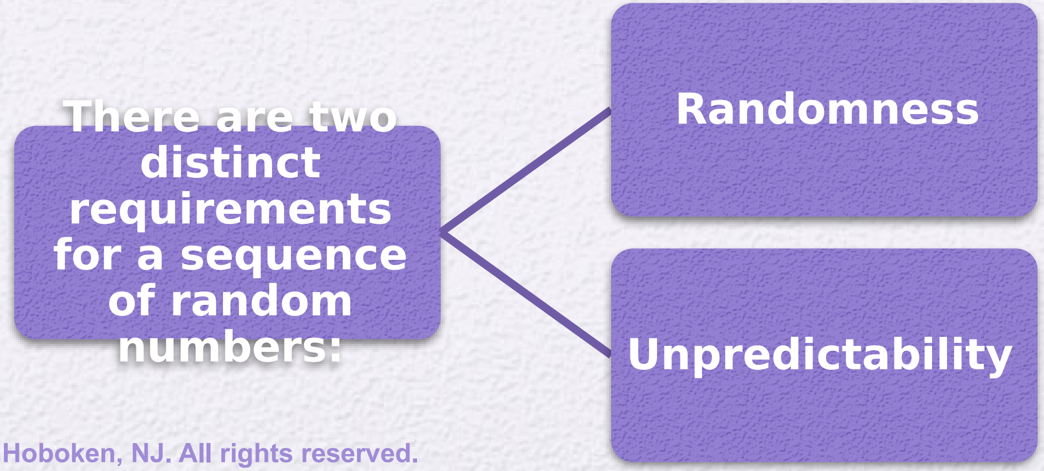


Chapter 8

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



There are two distinct requirements for a sequence of random numbers:

Randomness

Unpredictability

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**

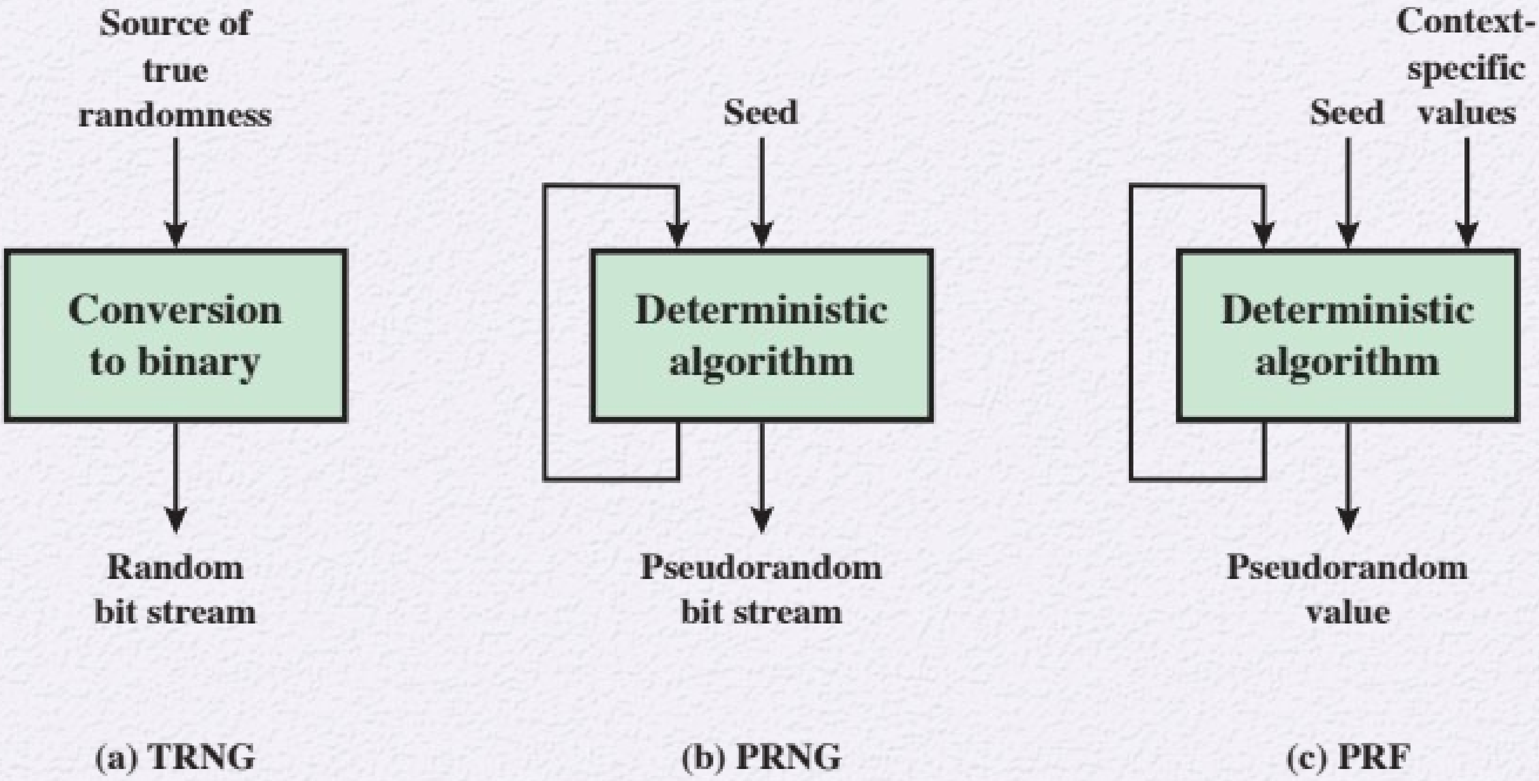
Unpredictability

- 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**

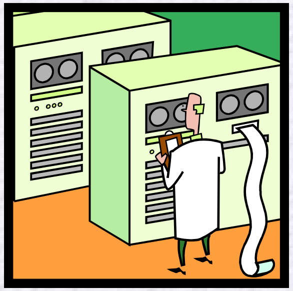
PRNG Requirements

- 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

Randomness

- The generated bit stream needs to appear random even though it is deterministic
- There is no single test that can determine if a PRNG generates numbers that have the characteristic of randomness
  - If the PRNG exhibits randomness on the basis of multiple tests, then it can be assumed to satisfy the randomness requirement
- NIST SP 800-22 specifies that the tests should seek to establish three characteristics:
  - Uniformity
  - Scalability
  - Consistency



Randomness Tests

- SP 800-22 lists 15 separate tests of randomness

Frequency test

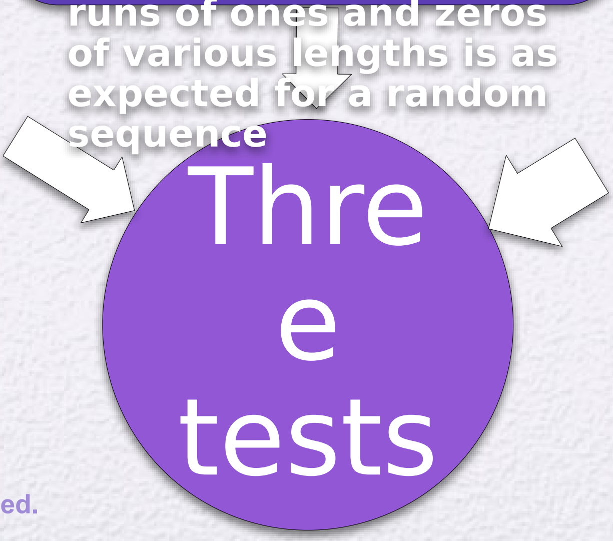
- The most basic test and must be included in any test suite
- Purpose is to determine whether the number of ones and zeros in a sequence is approximately the same as would be expected for a truly random sequence

Runs test

- Focus of this test is the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits bounded before and after with a bit of the opposite value
- Purpose is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence

Maurer’s universal statistical test

- Focus is the number of bits between matching patterns
- Purpose is to detect whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to



Three tests

Unpredictability

- A stream of pseudorandom numbers should exhibit two forms of unpredictability:
  - Forward unpredictability
    - If the seed is unknown, the next output bit in the sequence should be unpredictable in spite of any knowledge of previous bits in the sequence
  - Backward unpredictability
    - It should not be feasible to determine the seed from knowledge of any generated values
    - No correlation between a seed and any value generated from that seed should be evident
    - Each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2
- The same set of tests for randomness also provides a test of unpredictability
  - A random sequence will have no correlation with a fixed value (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

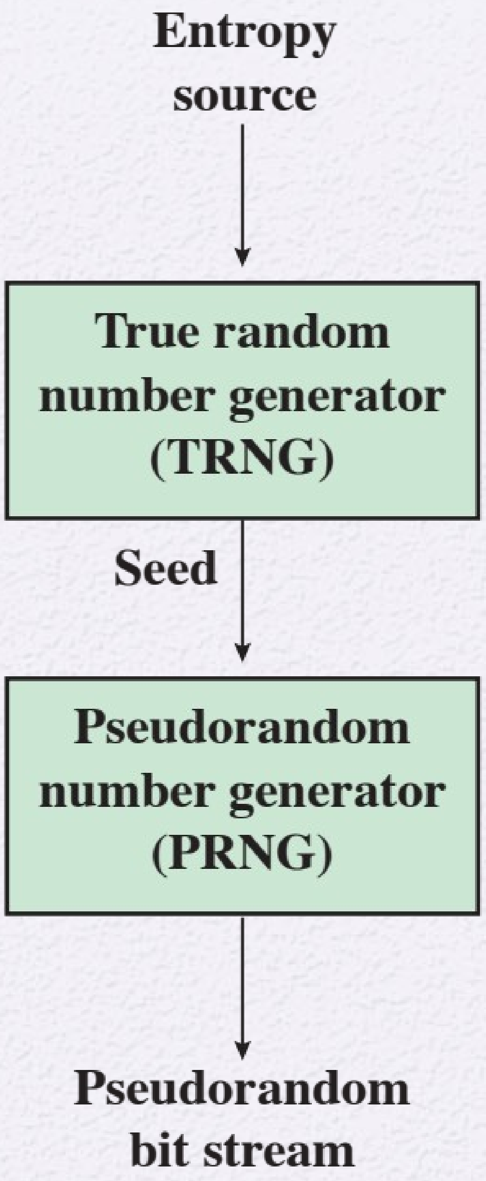

Entropy source
True random number generator (TRNG)
Seed
Pseudorandom number generator (PRNG)
Pseudorandom bit stream

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*

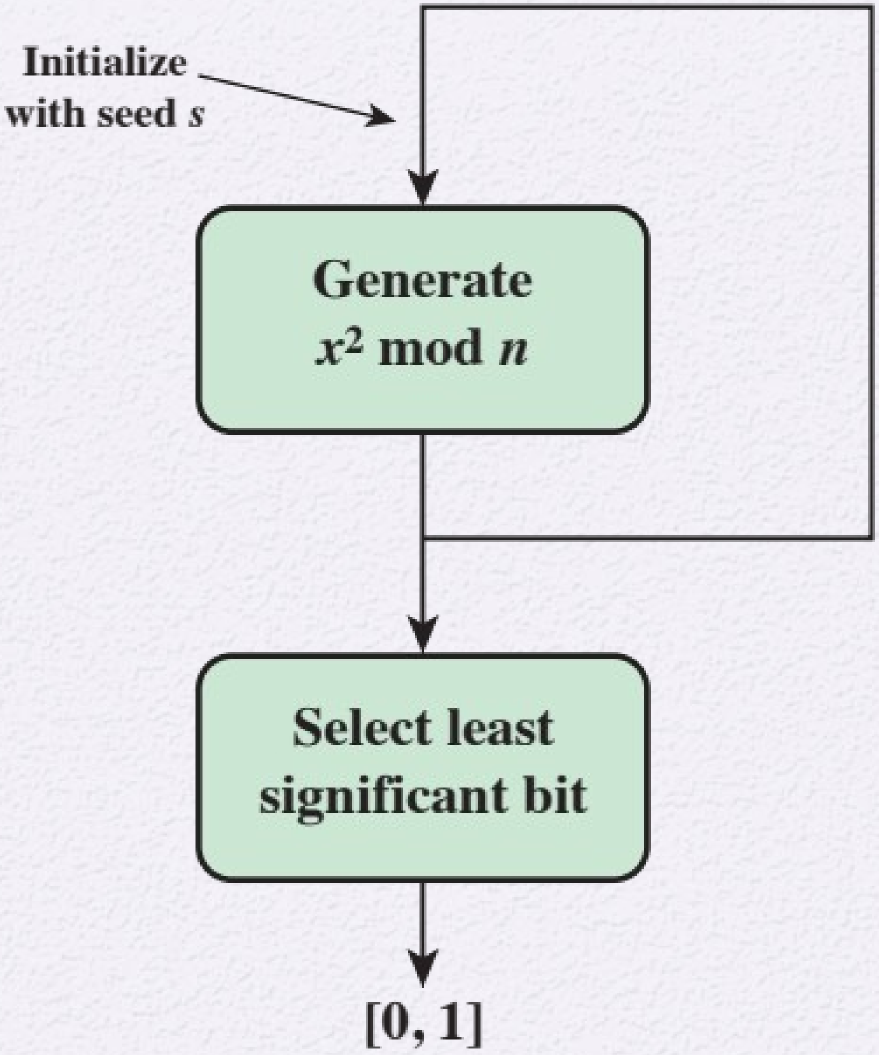
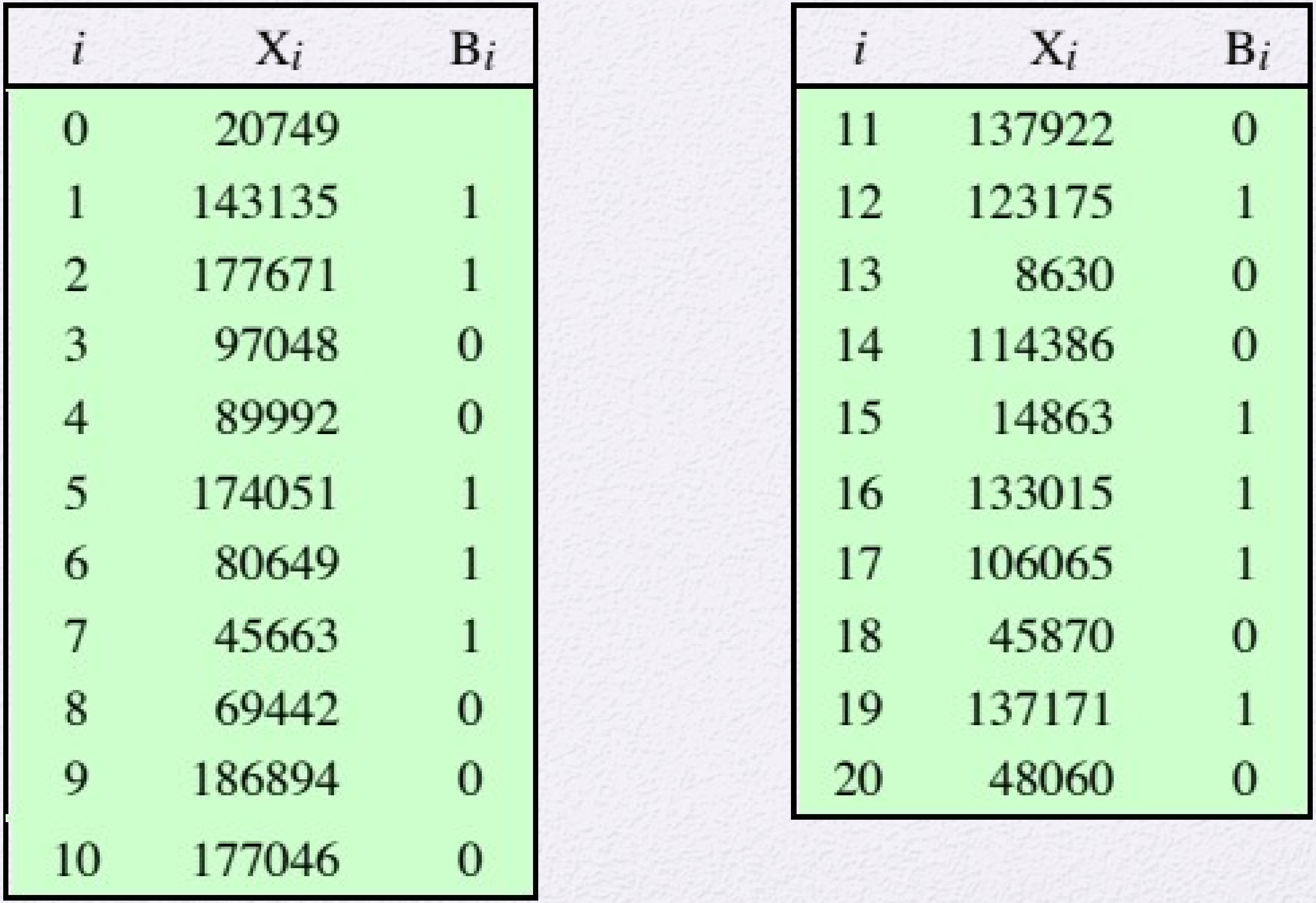


Figure 8.3 Blum Blum Shub Block Diagram

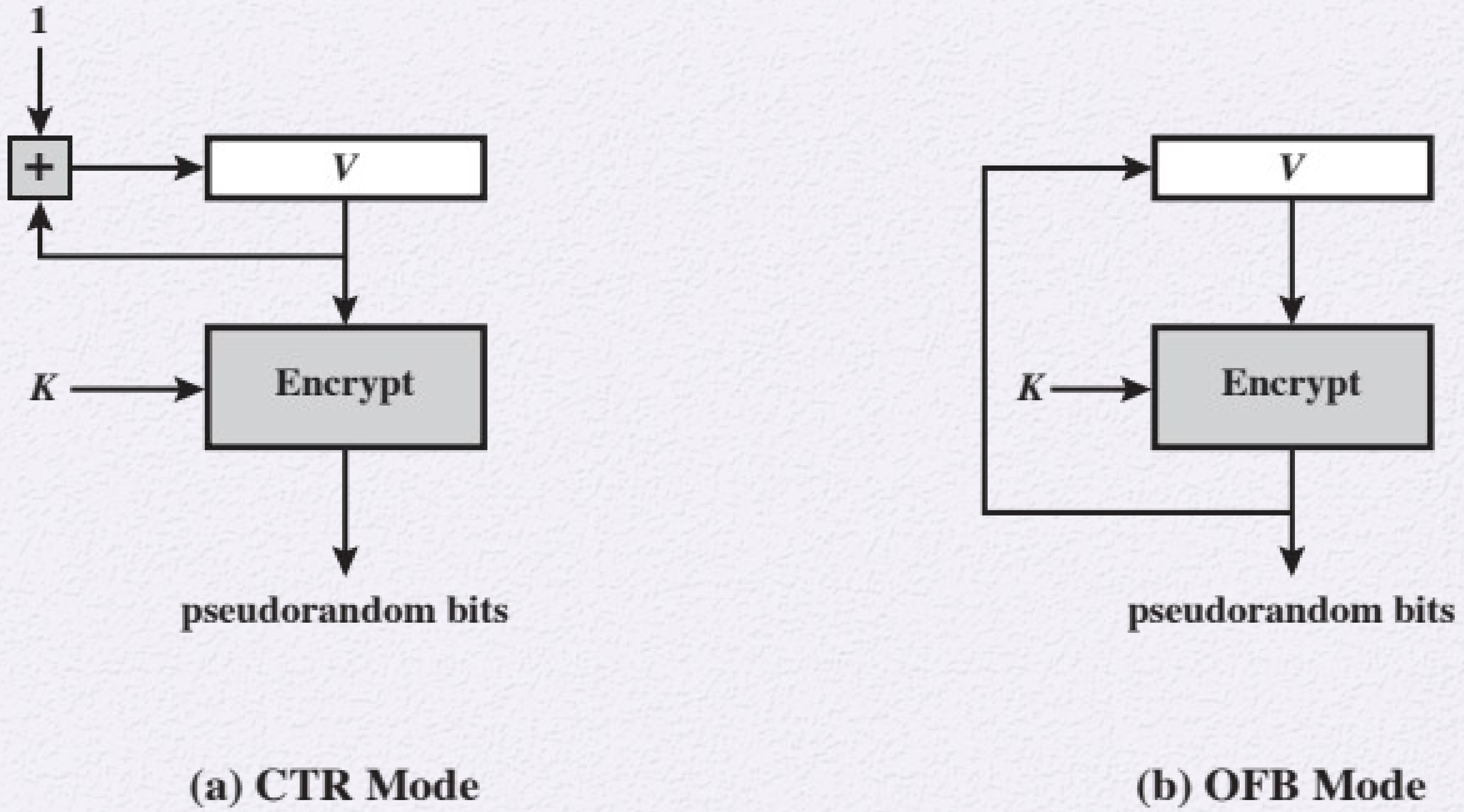
Table 8.1 Example Operation of BBS Generator



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

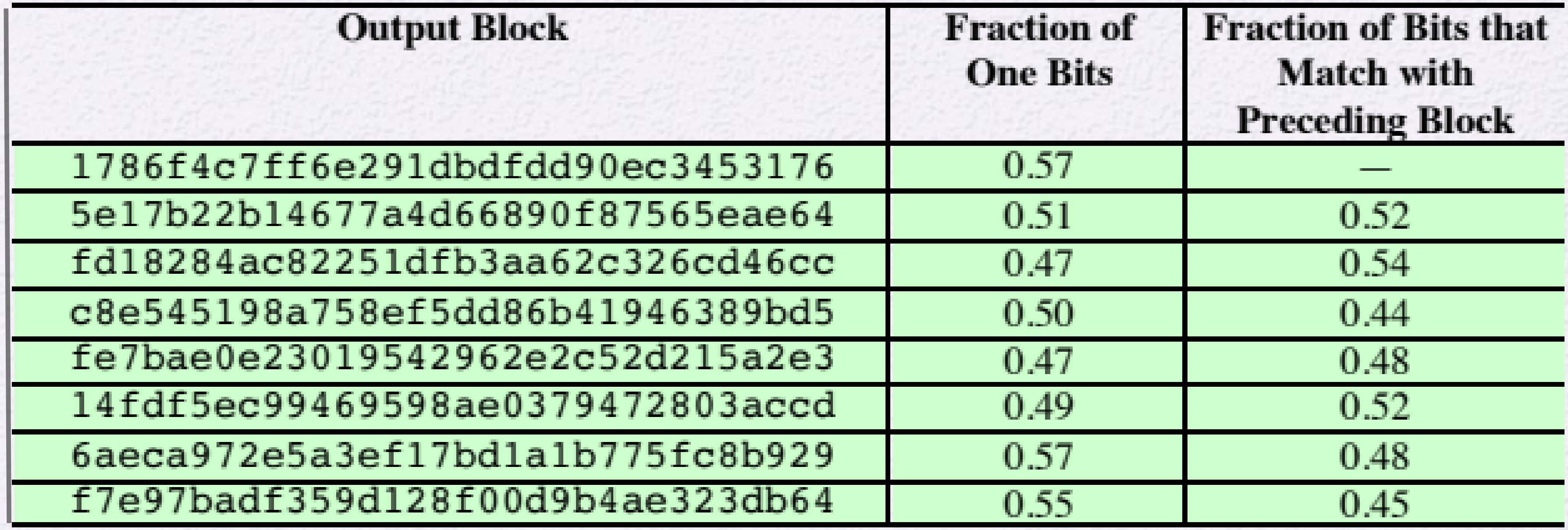


(a) CTR Mode

(b) OFB Mode

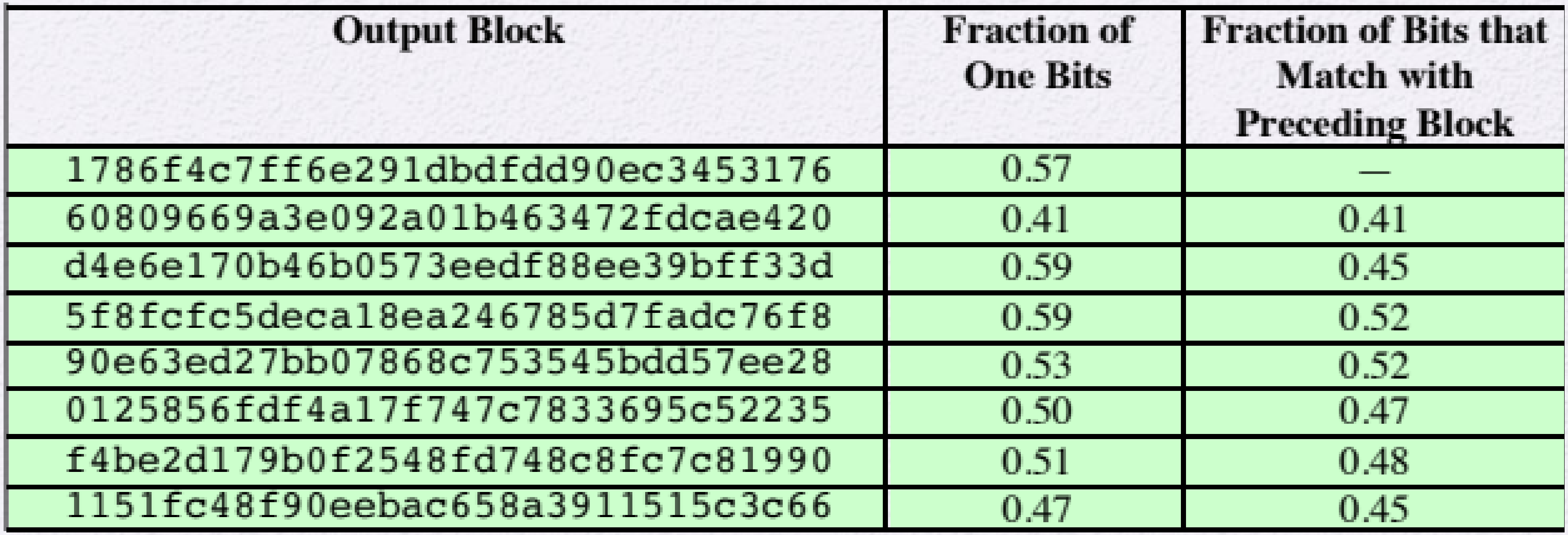
Figure 8.4 PRNG Mechanisms Based on Block Ciphers

Table 8.2 Example Results for PRNG Using OFB



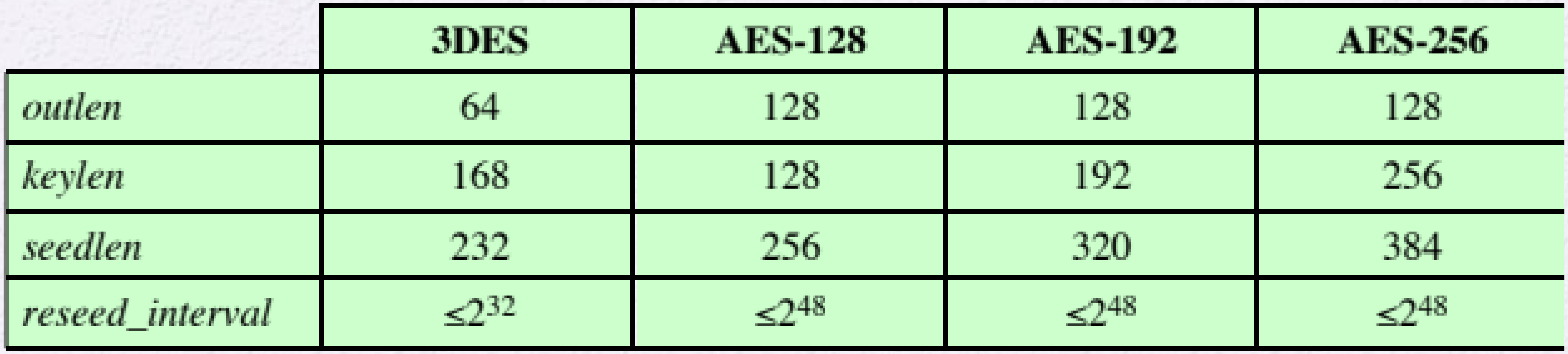
Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	—
5e17b22b14677a4d66890f87565eae64	0.51	0.52
fd18284ac82251dfb3aa62c326cd46cc	0.47	0.54
c8e545198a758ef5dd86b41946389bd5	0.50	0.44
fe7bae0e23019542962e2c52d215a2e3	0.47	0.48
14fdf5ec99469598ae0379472803accd	0.49	0.52
6aeca972e5a3ef17bd1a1b775fc8b929	0.57	0.48
f7e97badf359d128f00d9b4ae323db64	0.55	0.45

Table 8.3 Example Results for PRNG Using CTR



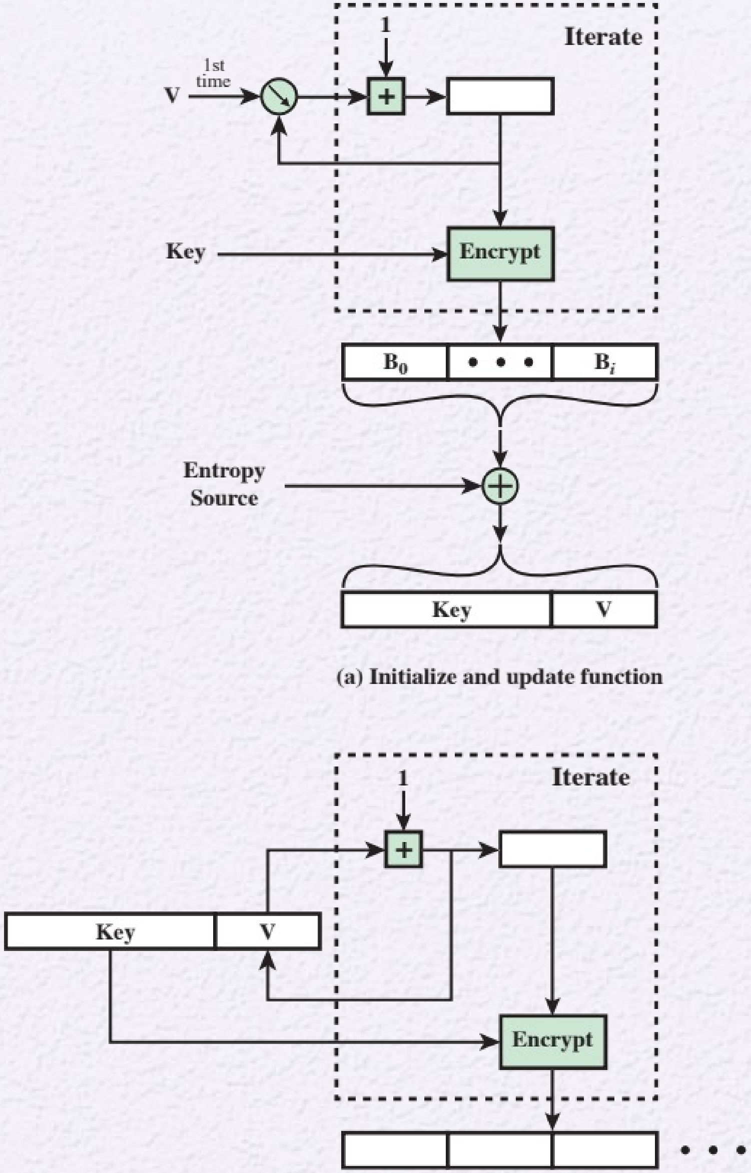
Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	—
60809669a3e092a01b463472fdcae420	0.41	0.41
d4e6e170b46b0573eedf88ee39bff33d	0.59	0.45
5f8fcfc5deca18ea246785d7fadc76f8	0.59	0.52
90e63ed27bb07868c753545bdd57ee28	0.53	0.52
0125856fdf4a17f747c7833695c52235	0.50	0.47
f4be2d179b0f2548fd748c8fc7c81990	0.51	0.48
1151fc48f90eebac658a3911515c3c66	0.47	0.45

Table 8.4



	3DES	AES-128	AES-192	AES-256
outlen	64	128	128	128
keylen	168	128	192	256
seedlen	232	256	320	384
reseed_interval	$\leq 2^{32}$	$\leq 2^{48}$	$\leq 2^{48}$	$\leq 2^{48}$

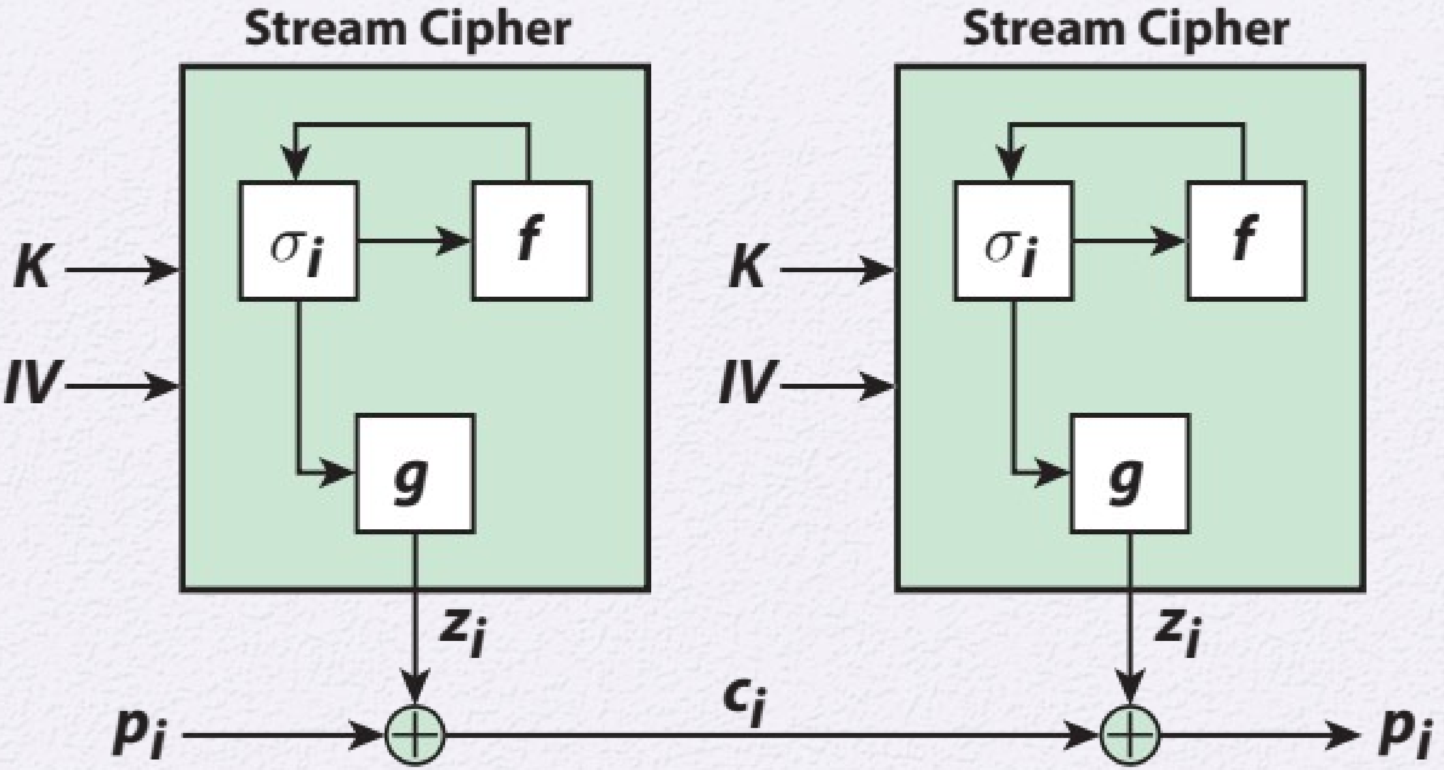
CTR_DRBG Parameters



(a) Initialize and update function

(b) Generate function

Figure 8.5 CTR_DRBG Functions



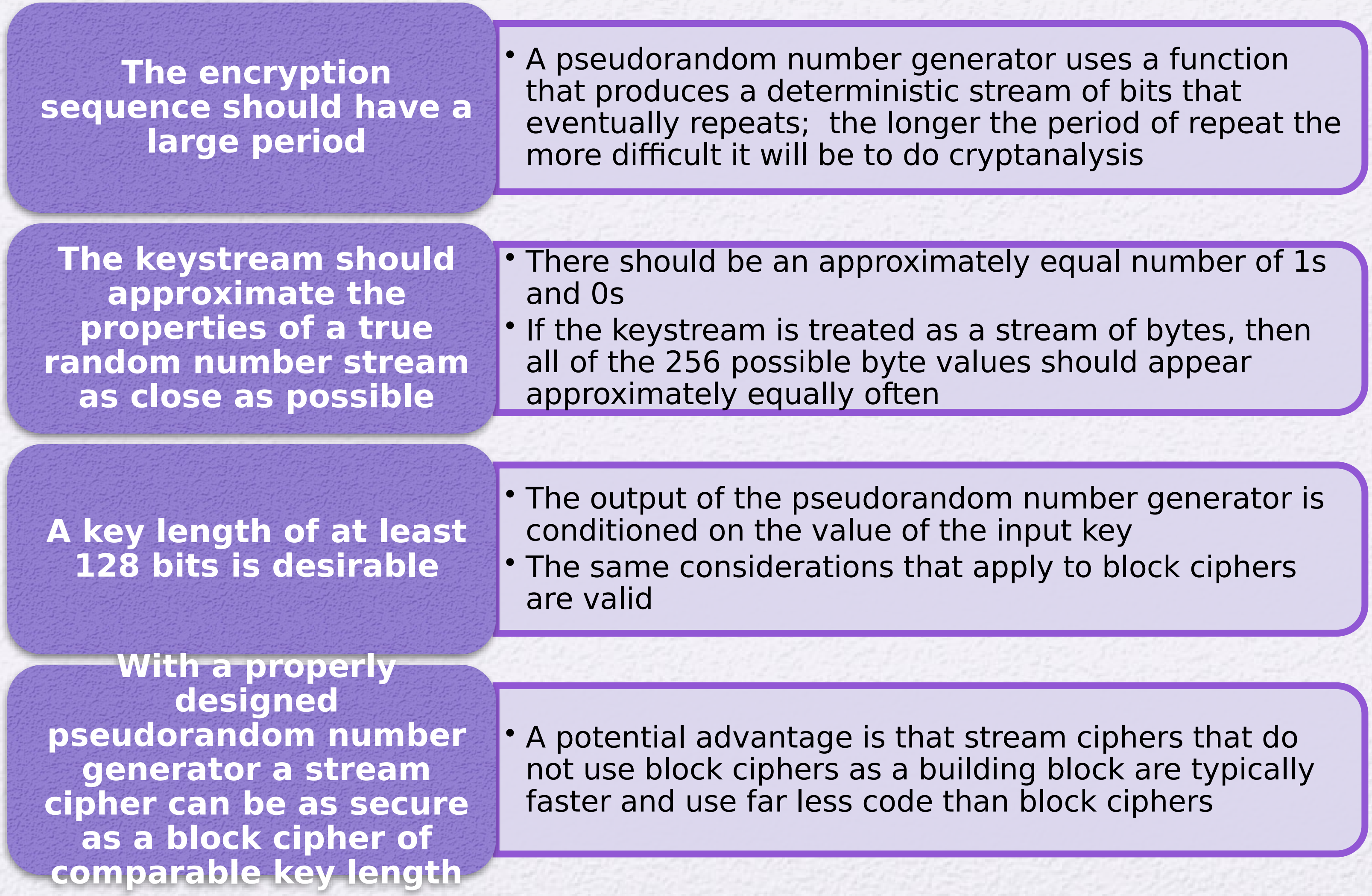
plaintext p_i
ciphertext c_i
keystream z_i

key K
Initialization Value IV

state σ_i
next-state function f
keystream function g

Figure 8.6 Generic Structure of a Typical Stream Cipher

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

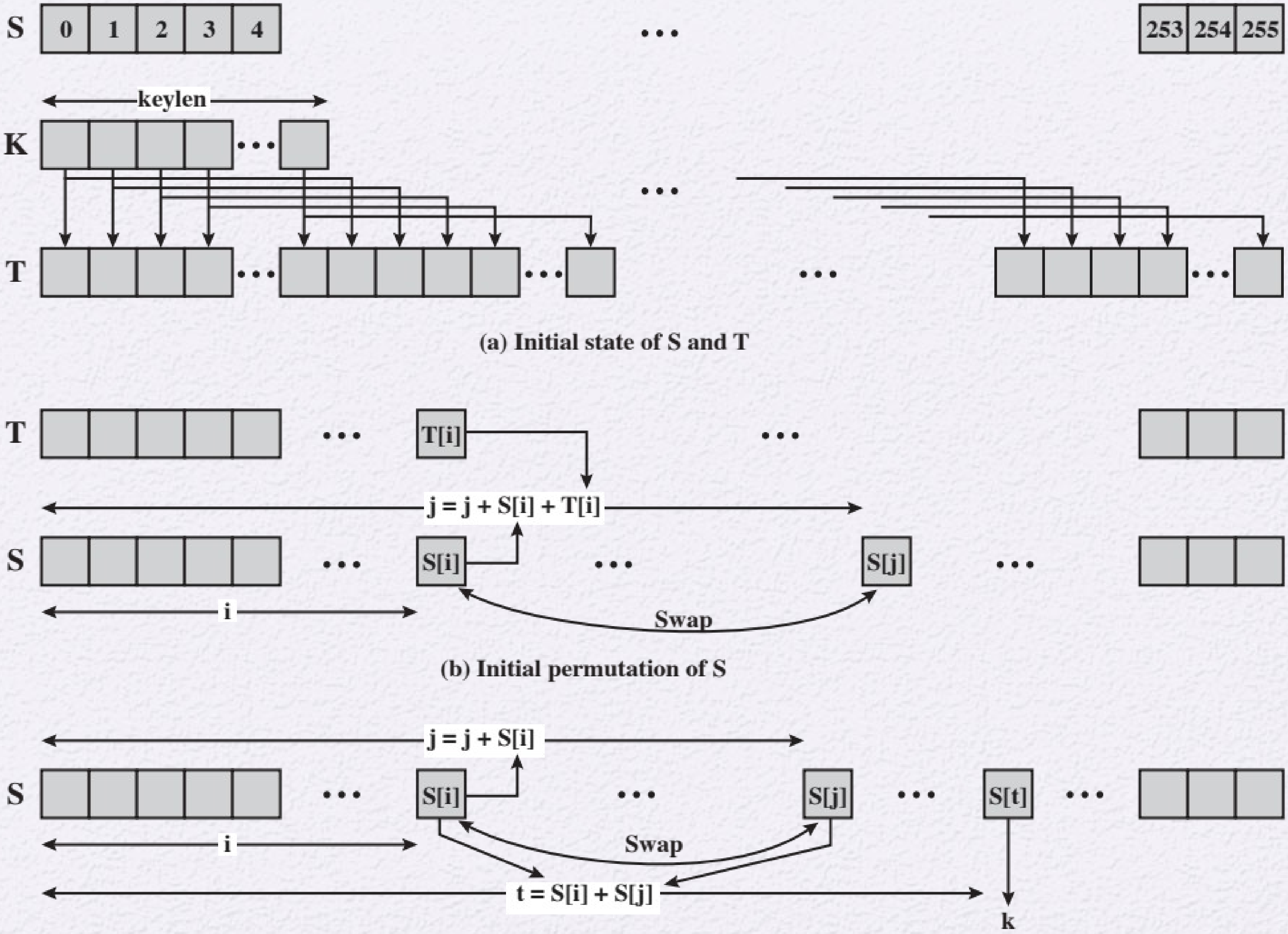
- The output of the pseudorandom number generator is conditioned on the value of the input key
- The same considerations that apply to block ciphers are valid

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



(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

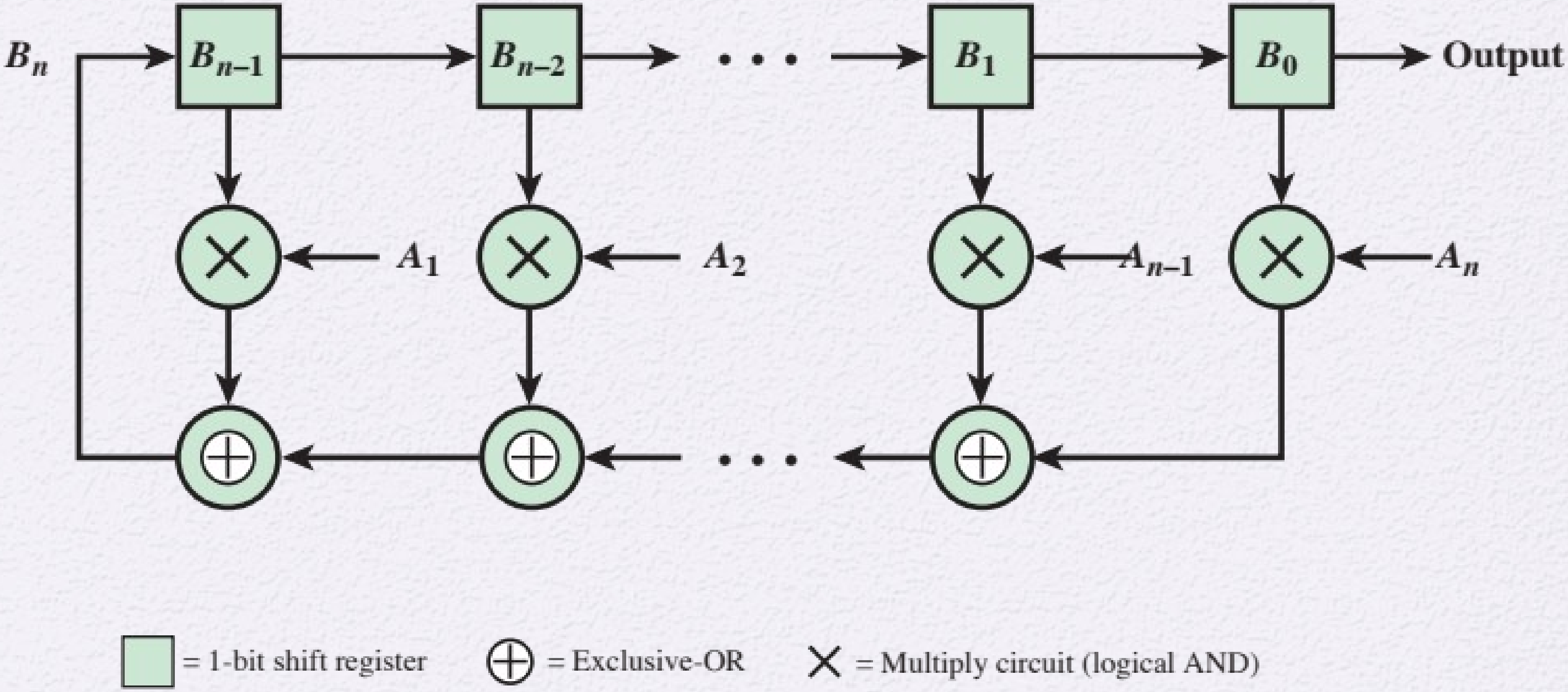
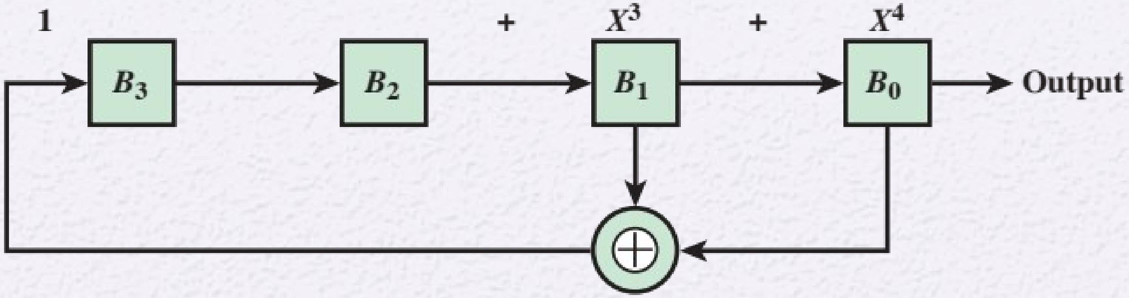


Figure 8.8 Binary Linear Feedback Shift Register Sequence Generator



(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

$$
\begin{array}{rllllllllllll}
 & 1+ & X+ & X^2+ & & X^4+ & & & X^7+ & X^8+ & \cdots & & \\ \cline{2-13}
1+X+X^3 \;\big) & 1 & & & & & & & & & & & \\
 & 1+ & X+ & & X^3 & & & & & & & & \\ \cline{2-5}
 & & X & & X^3 & & & & & & & & \\
 & & X+ & X^2+ & & X^4 & & & & & & & \\ \cline{3-6}
 & & & X^2+ & X^3+ & X^4 & & & & & & & \\
 & & & X^2+ & X^3+ & & X^5 & & & & & & \\ \cline{4-7}
 & & & & & X^4+ & X^5 & & & & & & \\
 & & & & & X^4+ & X^5+ & & X^7 & & & & \\ \cline{6-9}
 & & & & & & & & X^7 & & & & \\
 & & & & & & & & X^7+ & X^8+ & & X^{10} & \\ \cline{9-12}
 & & & & & & & & & X^8+ & & X^{10} & \\
 & & & & & & & & & X^8+ & X^9+ & & X^{11} \\ \cline{10-13}
\end{array}
$$

Figure 8.10 $1/(1 + X + X^3)$

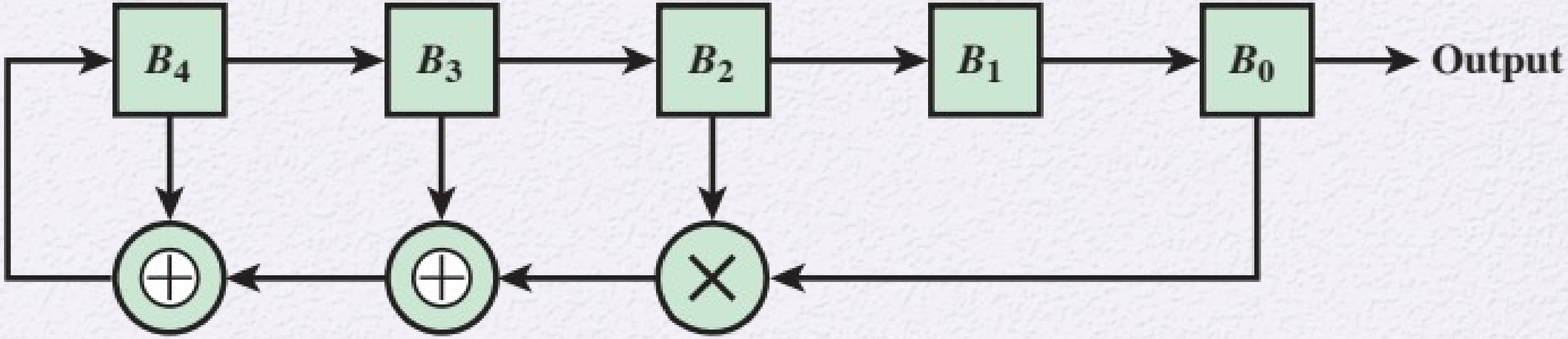
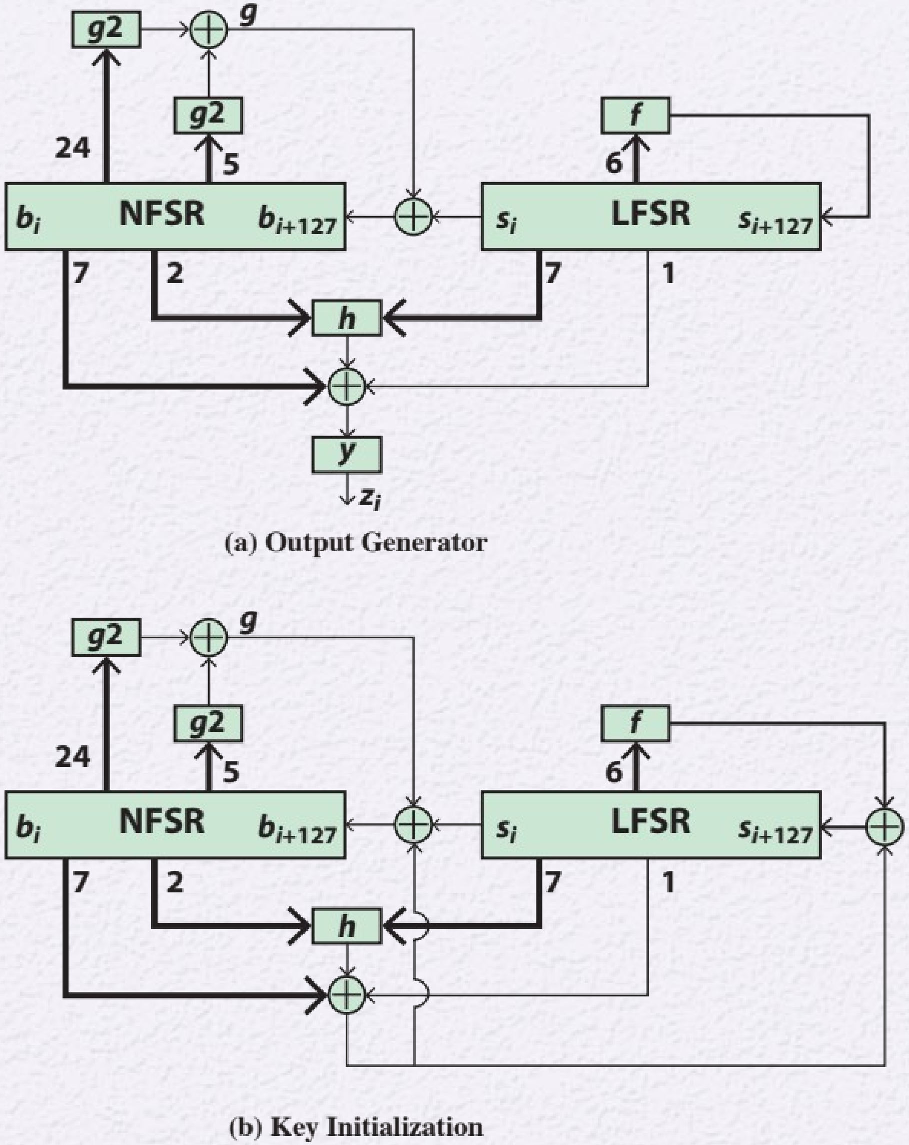


Figure 8.11 A Nonlinear Feedback Shift Register

Grain-128a

- Grain is a family of hardware-efficient stream ciphers
- Grain was accepted as part of the eSTREAM effort to approve a number of new stream ciphers
- The eSTREAM specification, called Grain v1, defines two stream ciphers, one with an 80-bit key and a 64-bit initialization vector (IV), and one with a 128-bit key and 80-bit IV
- Grain has since been revised and expanded to include authentication, referred to as Grain-128a
- The eSTREAM final report states that Grain has pushed the state of the art in terms of compact implementation
- Grain-128a consists of two shift registers, one with linear feedback and the second with nonlinear feedback, and a filter function
- The registers are couple by very lightweight, but judiciously chosen Boolean functions
- The LFSR guarantees a minimum period for the keystream, and it also provides balancedness in the output.
- The NFSR, together with a nonlinear filter, introduces nonlinearity to the cipher



(a) Output Generator

(b) Key Initialization

Figure 8.12 Grain-128a Stream Cipher

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

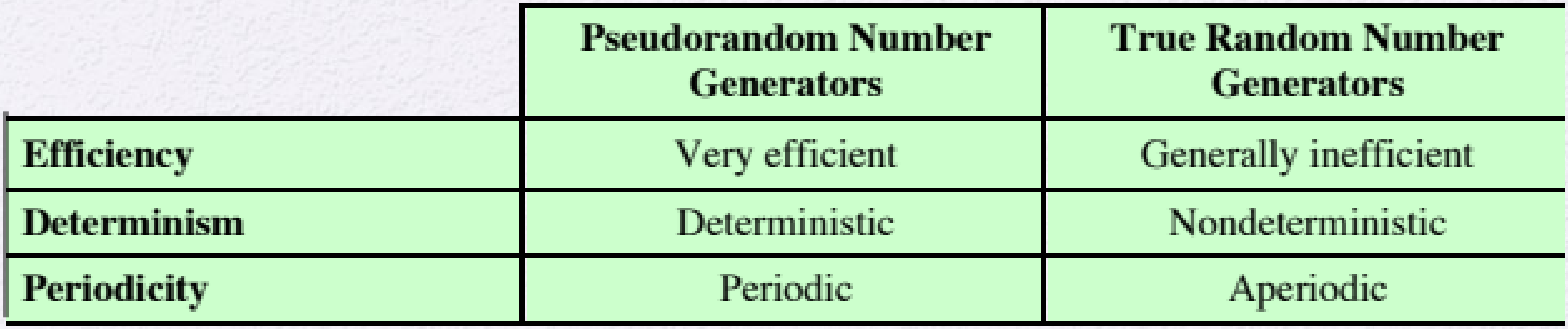
- The input from a sound digitizer with no source plugged in or from a camera with the lens cap on is essentially thermal noise
- If the system has enough gain to detect anything, such input can provide reasonable high quality random bits

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

Conditioning

- A TRNG may produce an output that is biased in some way (such as having more ones than zeros or vice versa)
- Biased
  - NIST SP 800-90B defines a random process as *biased* with respect to an assumed discrete set of potential outcomes if some of those outcomes have a greater probability of occurring than do others
- Entropy rate
  - NIST 800-90B defines entropy rate as the rate at which a digitized noise source provides entropy
  - Is a measure of the randomness or unpredictability of a bit string
  - Will be a value between 0 (no entropy) and 1 (full entropy)
- Conditioning algorithms/deskewing algorithms
  - Methods of modifying a bit stream to further randomize the bits
- Typically conditioning is done by using a cryptographic algorithm to scramble the random bits so as to eliminate bias and increase entropy
  - The two most common approaches are the use of a hash function or a symmetric block cipher

Hash Function

- A hash function produces an *n*-bit output from an input of arbitrary length

- A simple way to use a hash function for conditioning is as follows:
  - Blocks of *m* input bits, with *m* ≥ *n,* are passed through the hash function and the *n* output bits are used as random bits
  - To generate a stream of random bits, successive input blocks pass through the hash function to produce successive hashed output blocks

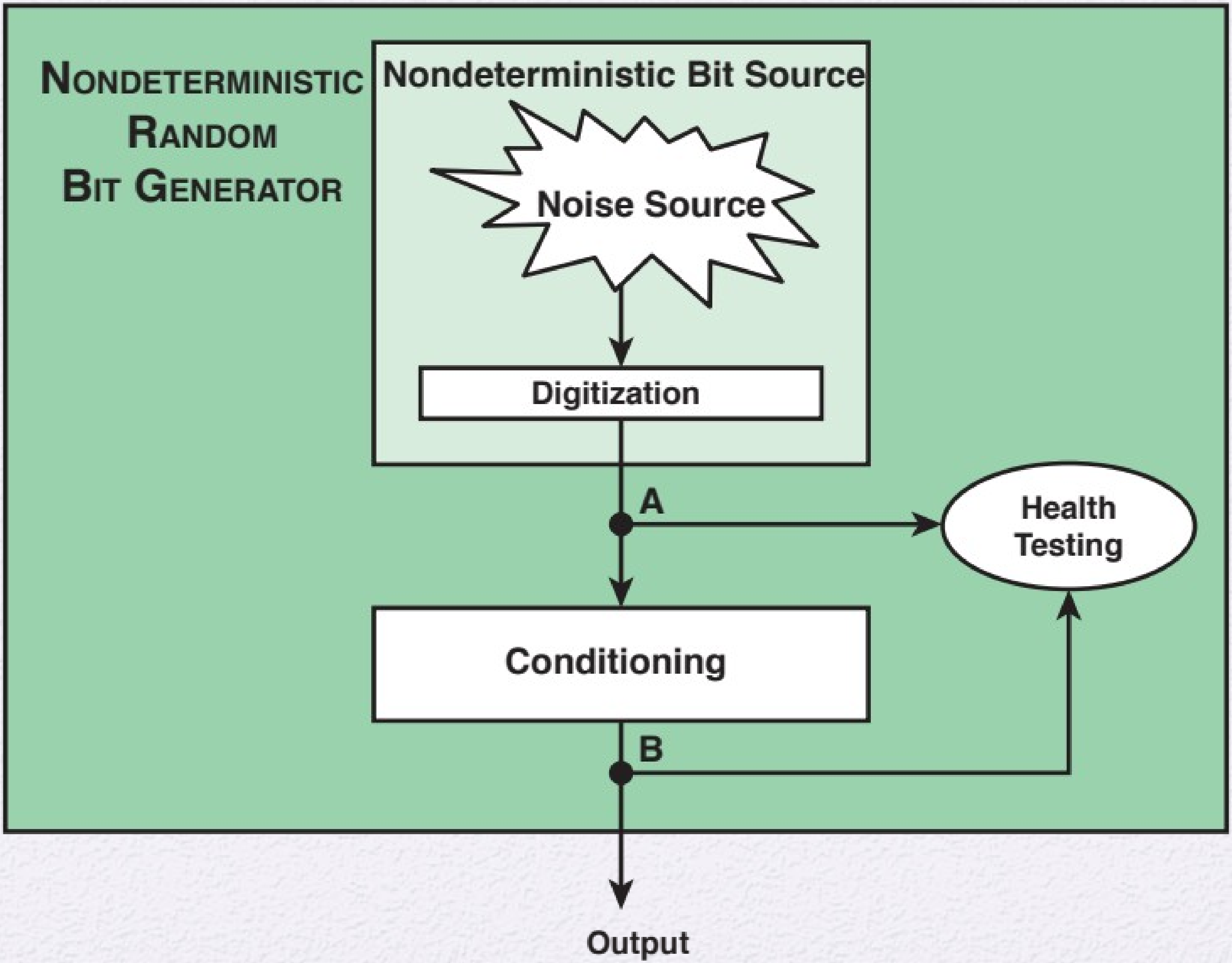

Nondeterministic Random Bit Generator
Nondeterministic Bit Source
Noise Source
Digitization
A
Health Testing
Conditioning
B
Output

Figure 8.13 NRBG Model

Health Tests on the Noise Source

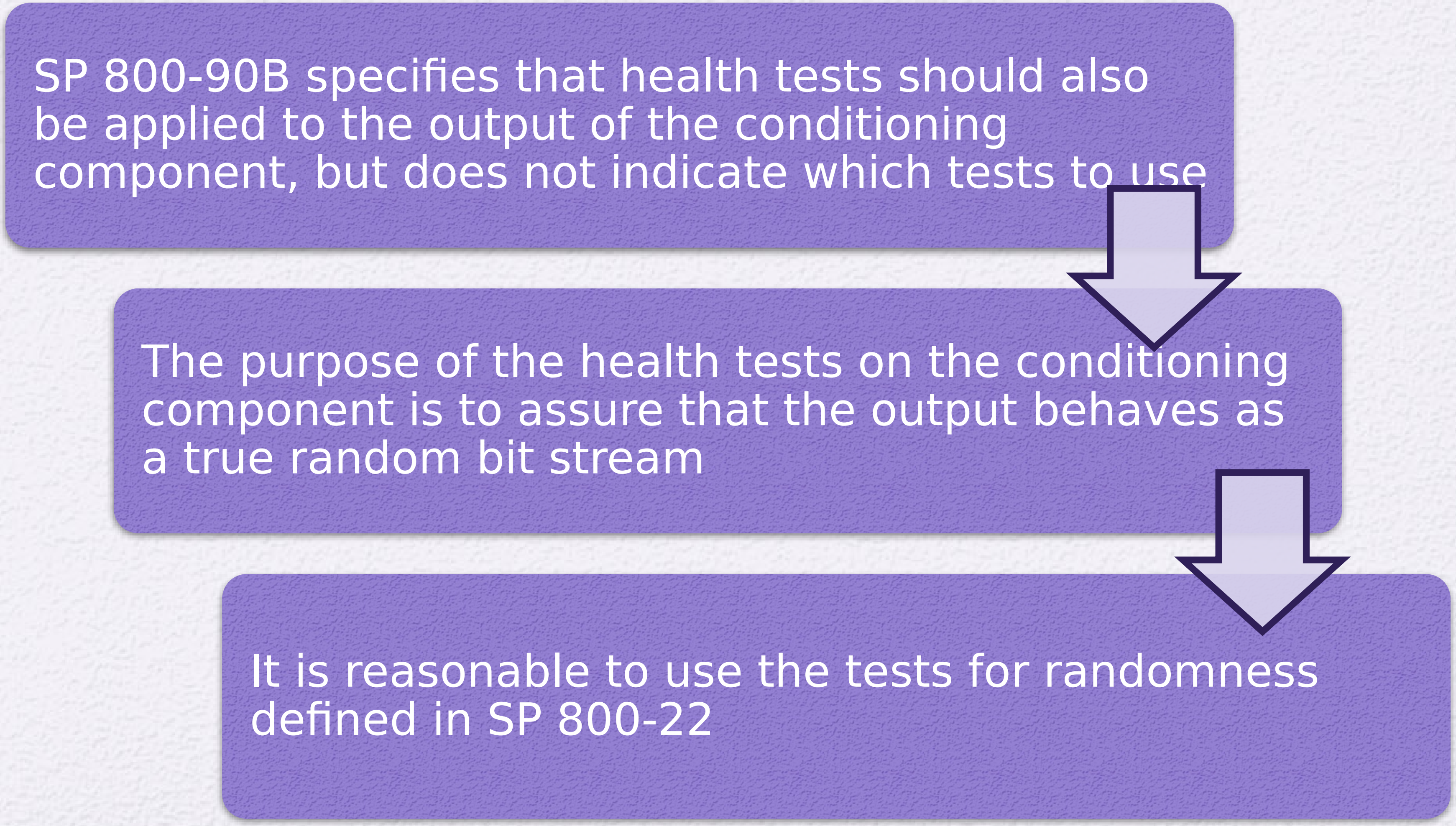
- The nature of the health testing of the noise source depends strongly on the technology used to produce noise

- In general, the assumption can be made that the digitized output of the noise source will exhibit some bias
  - Thus, traditional statistical tests are not useful for monitoring the noise source, because the noise source is likely to always fail
  - The tests on the noise source need to be tailored to the expected statistical behavior of the correctly operating noise source
  - The goal is not to determine if the source is unbiased, but if it is operating as expected

Health Tests on the Noise Source

- SP 800-90B specifies that continuous tests be done on digitized samples obtained from the noise source
  - The purpose is to test for variability and to determine if the noise source is producing at the expected entropy rate
- SP 800-90B mandates the use of two tests
  - Repetition Count Test
    - Designed to quickly detect a catastrophic failure that causes the noise source to become “stuck” on a single output value for a long time
    - Involves looking for consecutive identical samples
  - Adaptive Proportion Test
    - Designed to detect a large loss of entropy, such as might occur as a result of some physical failure or environmental change affecting the noise source
    - The test continuously measures the local frequency of occurrence of some sample value in a sequence of noise source samples to determine if the sample occurs too frequently

Health Tests on the Conditioning Function



SP 800-90B specifies that health tests should also be applied to the output of the conditioning component, but does not indicate which tests to use

The purpose of the health tests on the conditioning component is to assure that the output behaves as a true random bit stream

It is reasonable to use the tests for randomness defined in SP 800-22

Intel Digital Random Number Generator

- TRNGs have traditionally been used only for key generation and other applications where only a small number of random bits were required
  - This is because TRNGs have generally been inefficient with a low bit rate of random bit production
- The first commercially available TRNG that achieves bit production rates comparable with that of PRNGs is the Intel digital random number generator offered on new multicore chips since May 2012
  - It is implemented entirely in hardware
  - The entire DRNG is on the same multicore chip as the processors

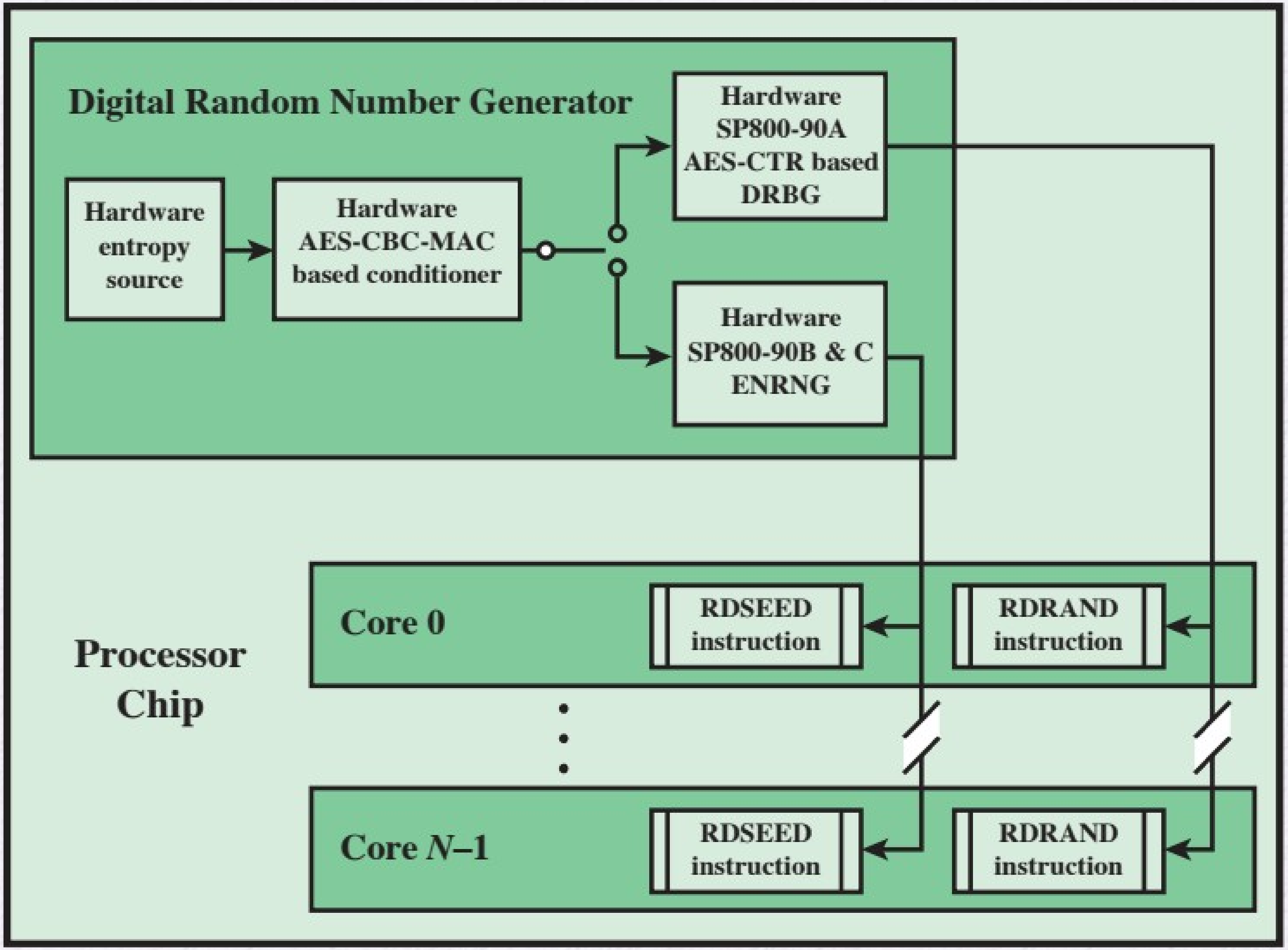

Digital Random Number Generator
Hardware entropy source
Hardware AES-CBC-MAC based conditioner
Hardware SP800-90A AES-CTR based DRBG
Hardware SP800-90B & C ENRNG
Processor Chip
Core 0
RDSEED instruction
RDRAND instruction
Core N–1
RDSEED instruction
RDRAND instruction

Figure 8.14 Intel Processor Chip with Random Number Generator

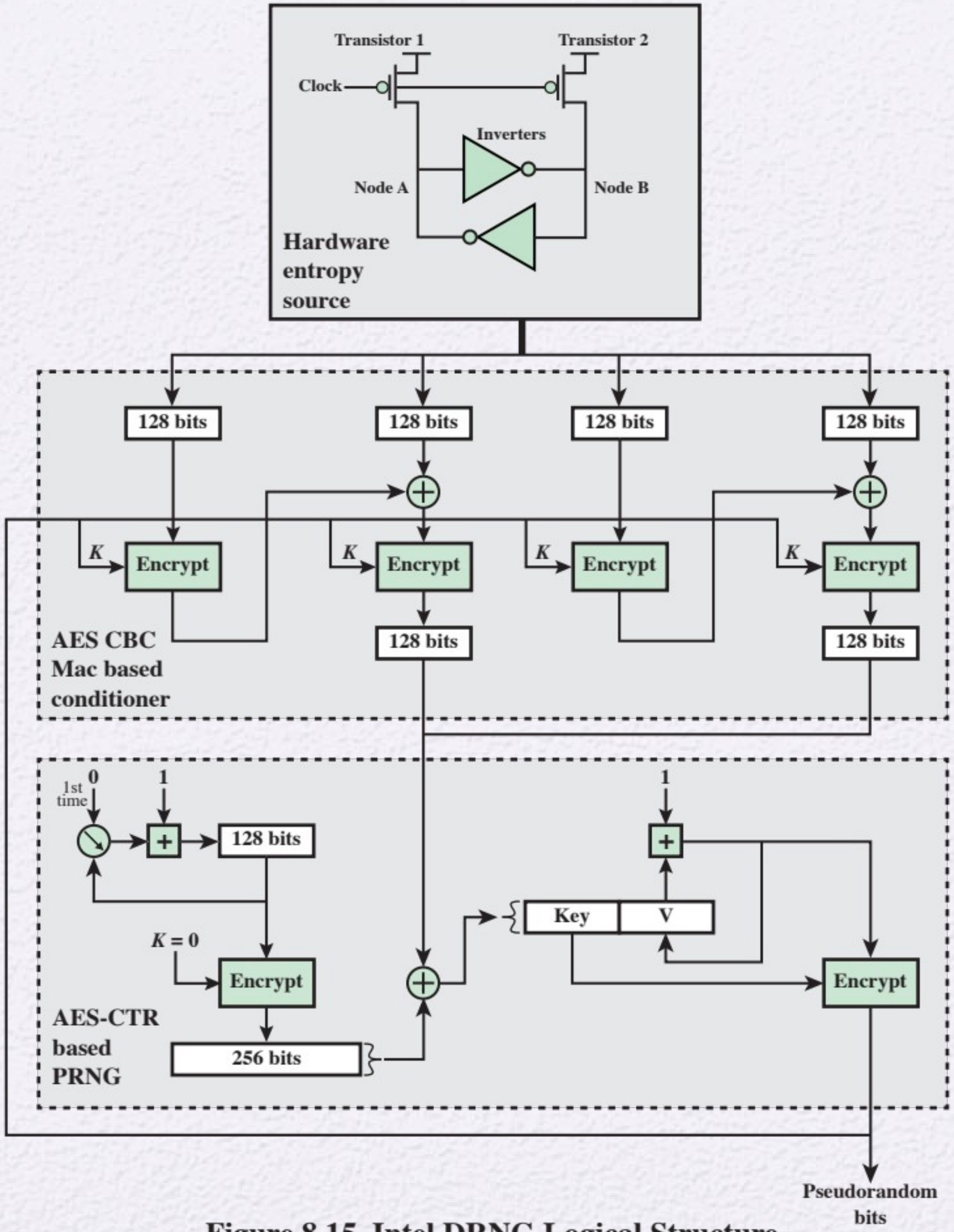
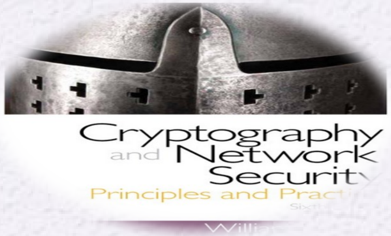


Figure 8.15 Intel DRNG Logical Structure

Summary

- Explain the concepts of randomness and unpredictability with respect to random numbers
- Present an overview of requirements for pseudorandom number generators
- Explain the significance of skew
- Present an overview of stream ciphers and RC4



- Understand the differences among true random number generators, pseudorandom number generators, and pseudorandom functions
- Explain how a block cipher can be used to construct a pseudorandom