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

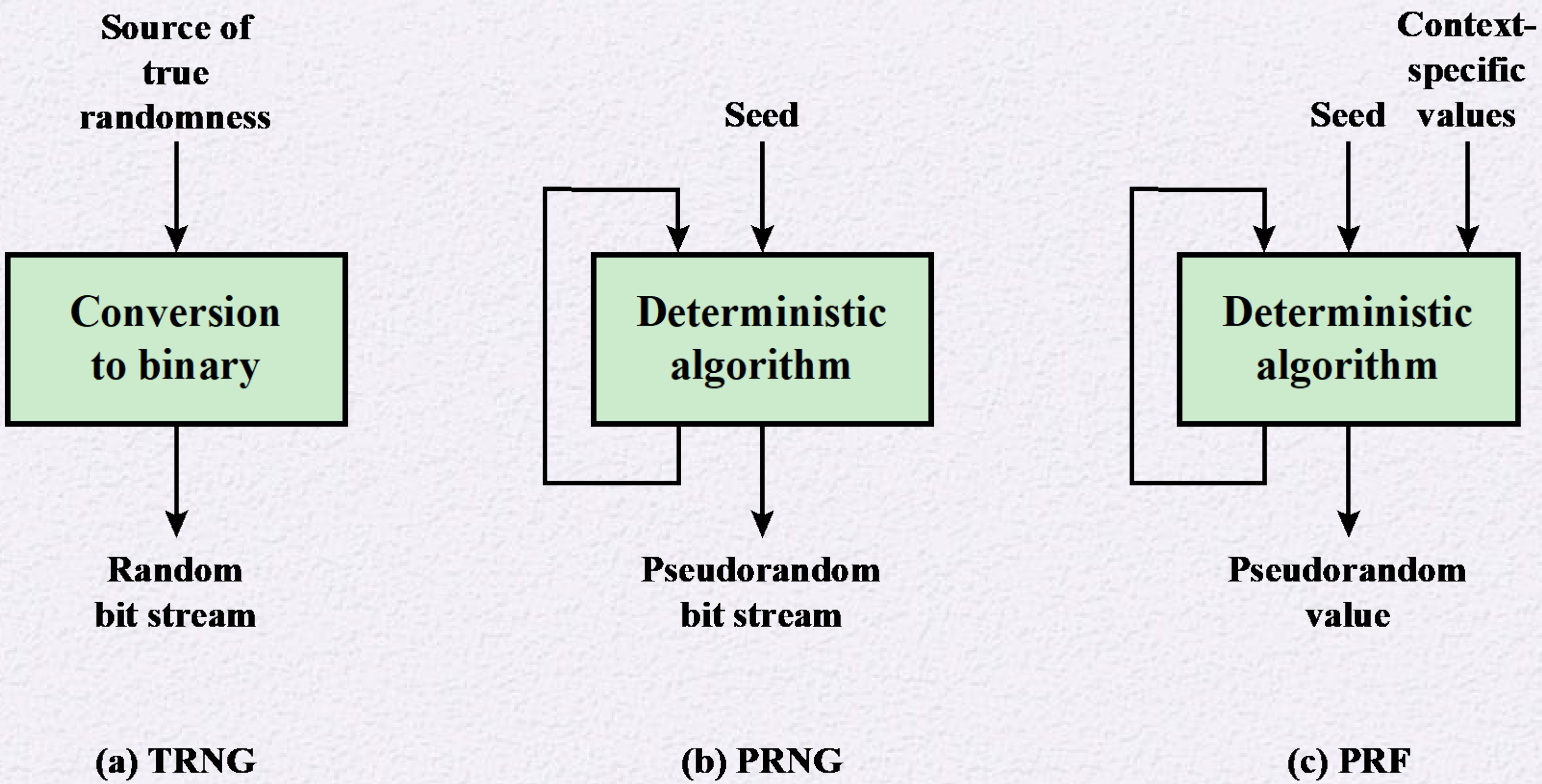
- No one subsequence in the sequence can be inferred from the others

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 exhibit random properties
- 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)

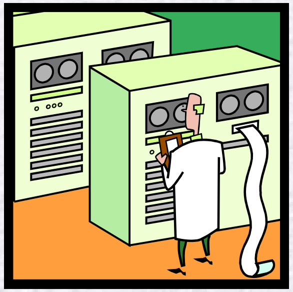
- Used to produce a pseudorandom string of bits of some fixed length
- Examples are symmetric encryption keys and nonces

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 (on different seeds)



Randomness Tests

- SP 800-22 lists 15 separate tests of randomness

Three tests

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 be non-random

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

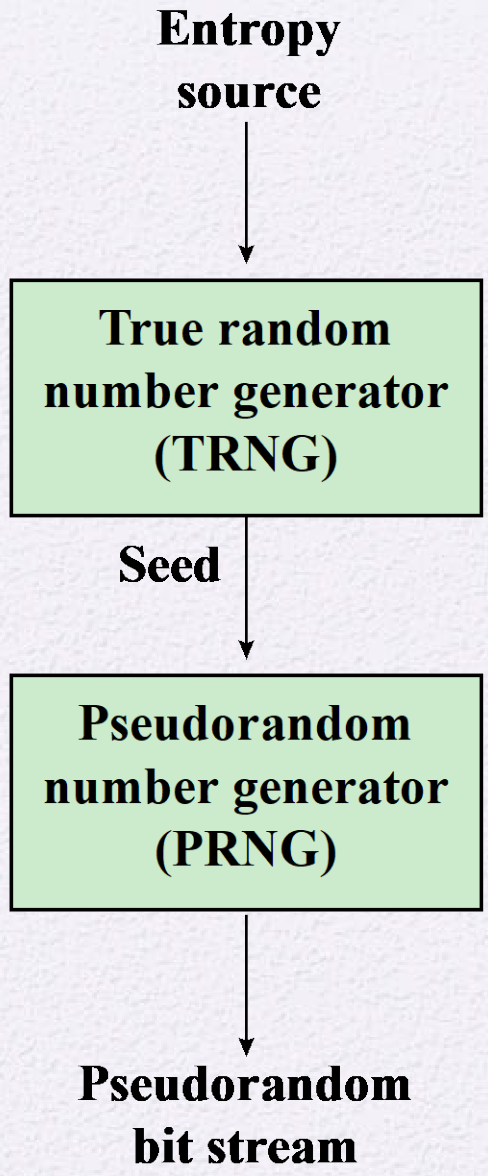


Figure 8.2 Generation of Seed Input to PRNG

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 (BBS) Generator

- Seed generation algorithm
  - Choose ***p*** and ***q***, such that $p \equiv q \equiv 3 \pmod{4}$
  - Find $n = p * q$
  - Choose random **s**, such that **s** is relatively prime to **n** i.e. neither ***p*** or ***q*** is a factor of ***s***
  - Use **s** as the seed of the algorithm
- We can choose several seed values for each value of ***n***
- ***p*** and ***q*** are usually chosen to be large prime numbers
- Breaking the cipher requires factoring ***n*** into ***p*** and ***q*** (Integer Factorization Problem)

Blum Blum Shub (BBS) Generator

- Random number generation algorithm
  - $x_0 = s^2 \bmod n$
  - For $i = 1$ to ∞
    - $x_i = (x_{i-1})^2 \bmod n$
    - $B_i = x_i \bmod 2$ (B_i is a bit value)

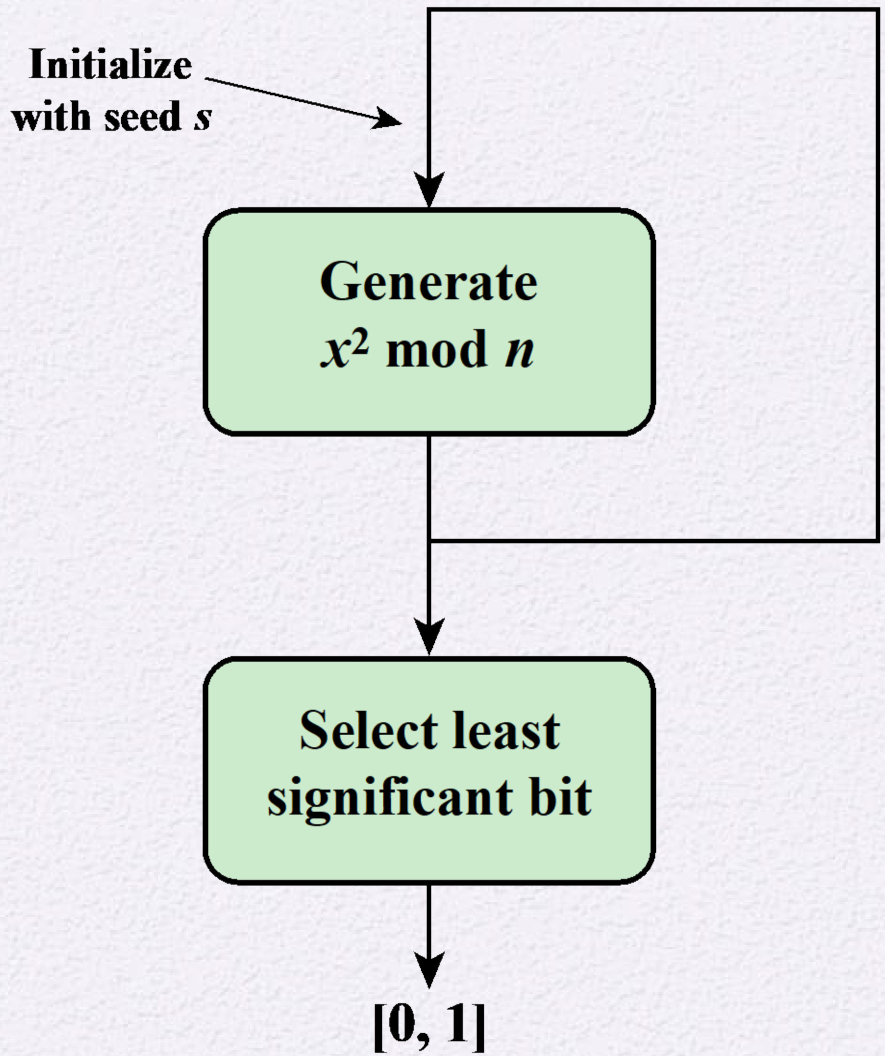


Figure 8.3 Blum Blum Shub Block Diagram

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

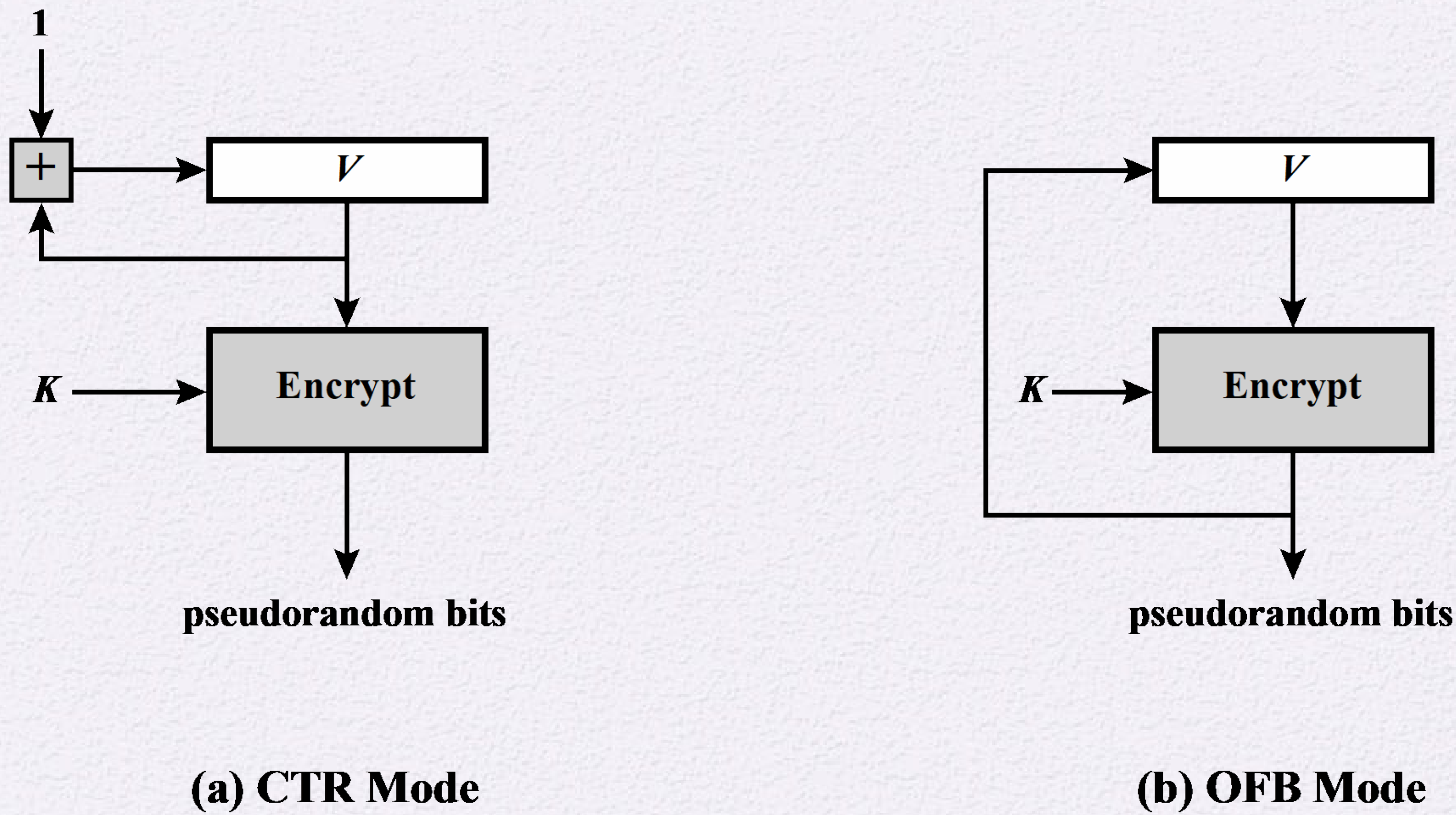
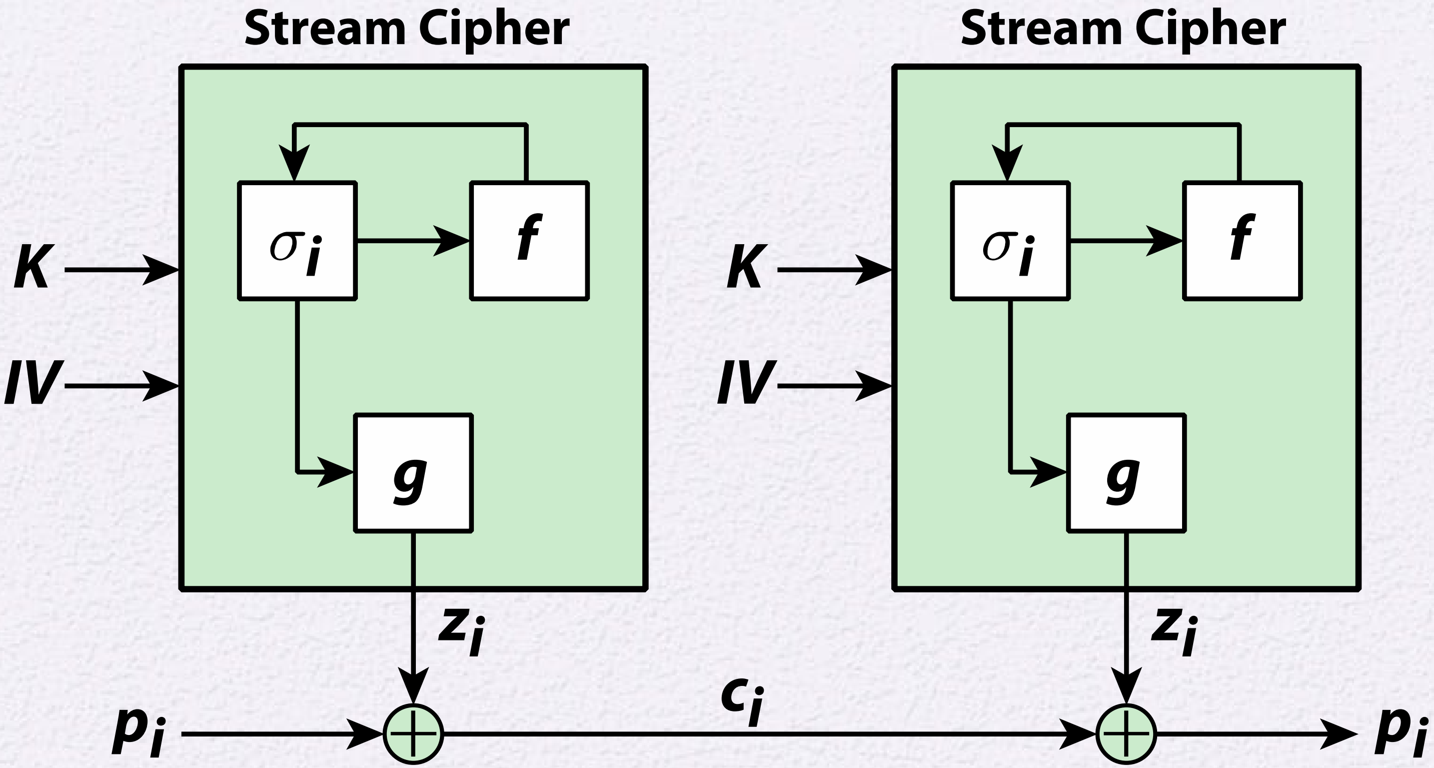


Figure 8.4 PRNG Mechanisms Based on Block Ciphers



plaintext p_i
ciphertext c_i
keystream z_i

key K
Initialization Value IV

state σ_i
next-state function f
keystream function g

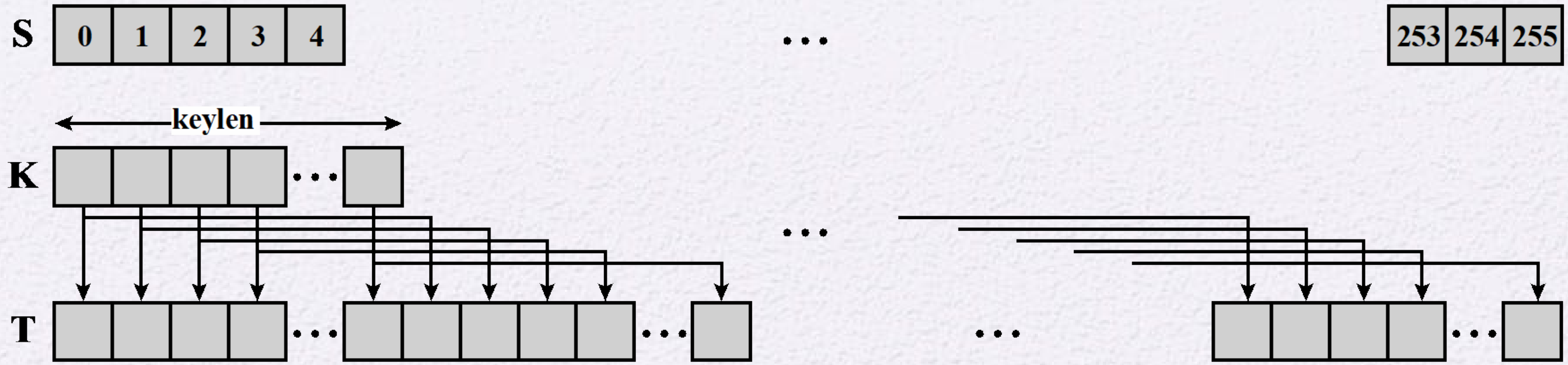
Figure 8.6 Generic Structure of a Typical Stream Cipher

Stream Cipher Design Considerations

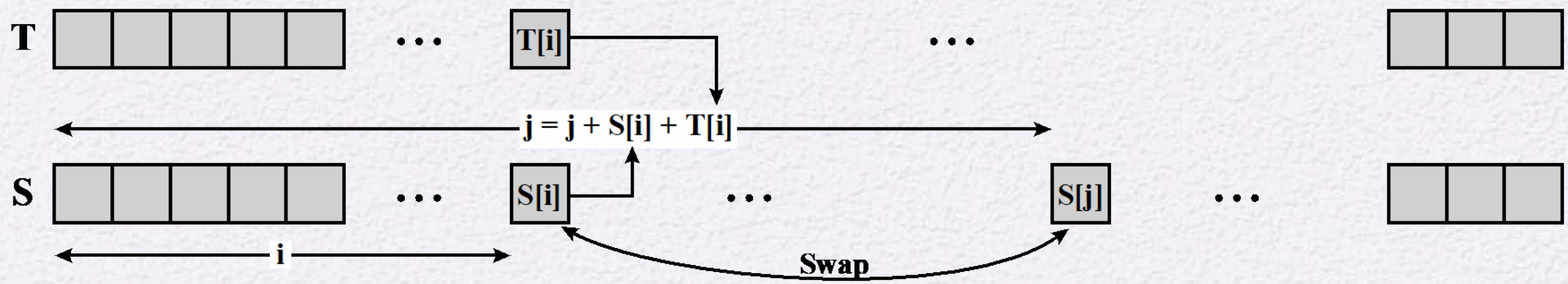
The encryption sequence should have a large period	• A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats; the longer the period of repeat the more difficult it will be to do cryptanalysis
The keystream should approximate the properties 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

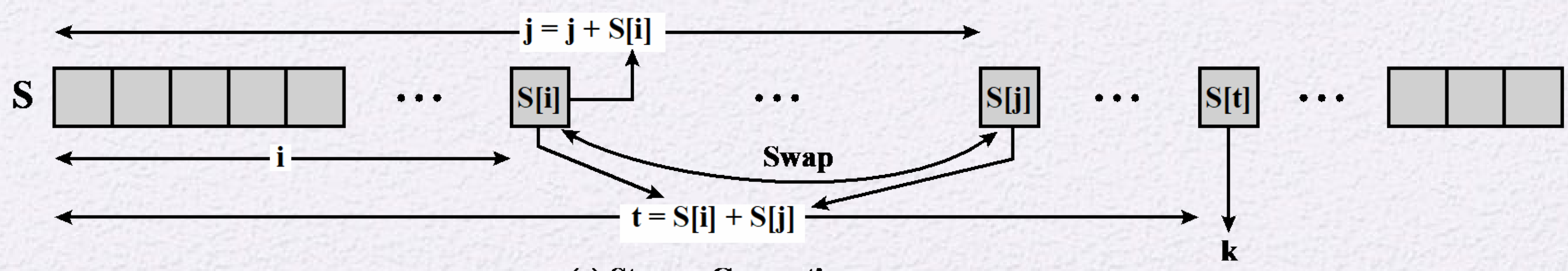
- Designed in 1987 by Ron Rivest for RSA Security
- Variable key size stream cipher with byte-oriented operations
- Based on the use of a random permutation
- Eight to sixteen machine operations are required per output byte and the cipher can be expected to run very quickly in software
- RC4 is used 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

RC4- Pseudocode

- Initial Permutation of array ***S***
  - ***j*** = **0**
  - For ***i*** = **0** to **255** *(Complete the task for all values of array **S**)*
    - ***j*** = ***j*** + ***S[i]*** + ***T[i]*** mod **256** *(Generate new index value **j**)*
    - Swap(***S[i]***,***S[j]***) *(Swap current value of array **S** i.e. **S[i]** with value in **S[j]**)*

RC4- Pseudocode

- Keystream Generation
  - ***j*** = **0**
  - For ***i*** = **0** to **255** *(Complete the task for all values of array* ***S,*** *start over when you get to the end)*
    - ***j*** = ***j*** + ***S[i]*** mod **256** *(Generate new index value* ***j****)*
    - Swap(***S[i]***,***S[j]***) *(Swap current value of array* ***S*** *i.e.* ***S[i]*** *with value in* ***S[j].*** *This generates a new permutation of array* ***S****)*
    - **t** = ***S[i]*** + ***S[j]*** mod **256** *(Generate new index value* **t** *using swapped values)*
    - Output byte in ***S[t]*** as ***k*** (***k*** *is next byte stream value that is used to XOR the next byte of plaintext)*

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

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

Summary

- Explain the concepts of randomness and unpredictability with respect to random numbers
- Present an overview of requirements for pseudorandom number generators
- 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 number generator