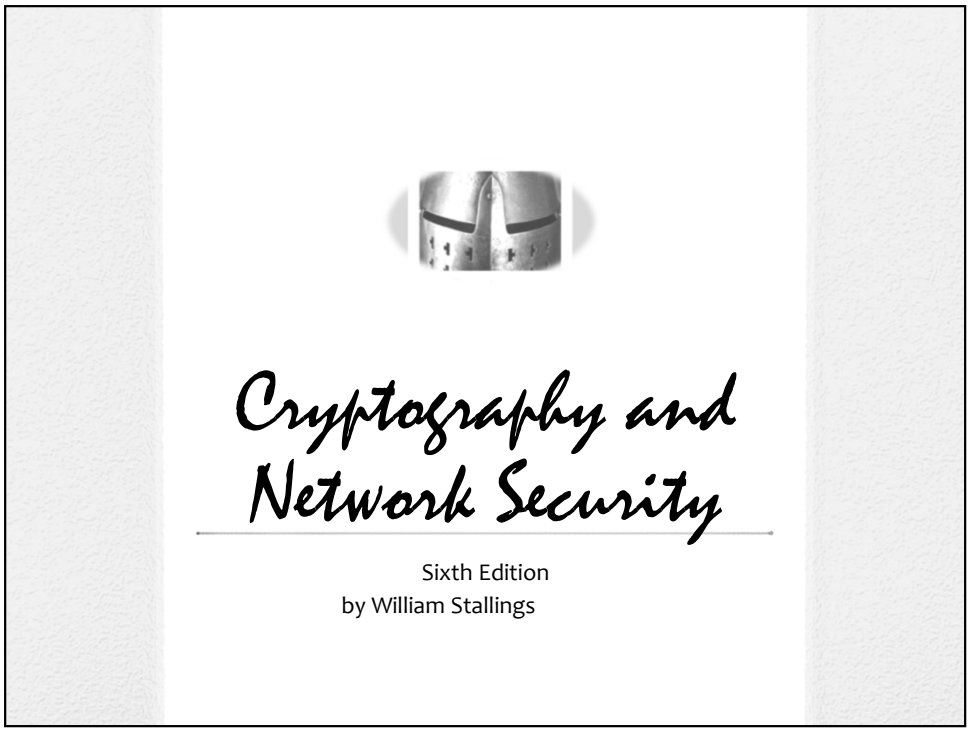
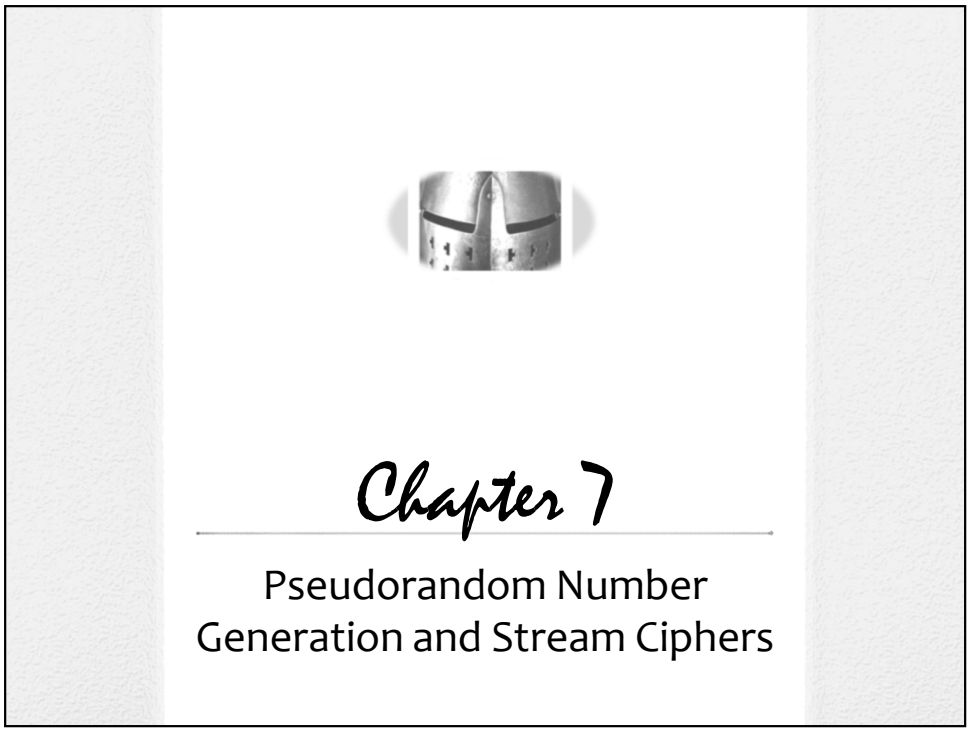


Lecture slides prepared for "Cryptography and Network Security", 6/e, by William Stallings, Chapter 7 – "Pseudorandom Number Generation and Stream Ciphers".



An important cryptographic function is cryptographically strong pseudorandom number

generation. Pseudorandom number generators (PRNGs) are used in a variety

of cryptographic and security applications. We begin the chapter with a look at the

basic principles of PRNGs and contrast these with true random number generators

(TRNGs). Next, we look at some common PRNGs, including PRNGs based on the

use of a symmetric block cipher.

The chapter then moves on to the topic of symmetric stream ciphers, which are

based on the use of a PRNG. The chapter next examines the most important stream

cipher, RC4. Finally, we examine TRNGs.

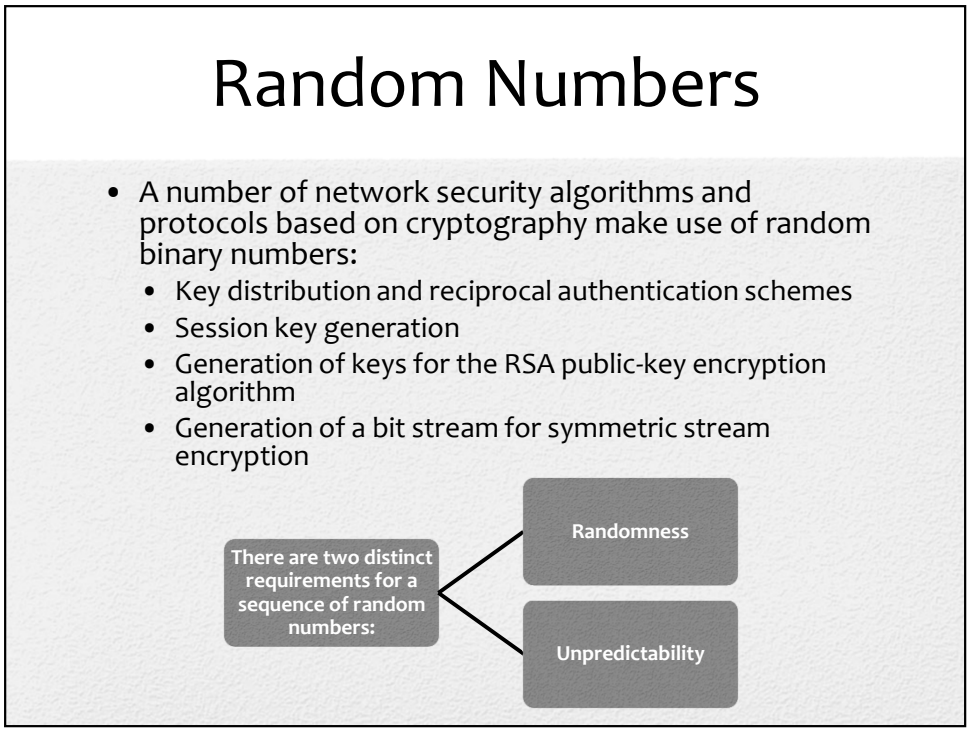
> *"The comparatively late rise of the theory of probability shows how hard it is to grasp, and the many paradoxes show clearly that we, as humans, lack a well grounded intuition in this matter."*
>
> *"In probability theory there is a great deal of art in setting up the model, in solving the problem, and in applying the results back to the real world actions that will follow."*
>
> ***— The Art of Probability,***
>
> **Richard Hamming**

Random numbers play an important role in the use of encryption for various network

security applications. In this section, we provide a brief overview of the use

of random numbers in cryptography and network security and then focus on the

principles of pseudorandom number generation.



A number of network security algorithms and protocols based on cryptography make use of random binary numbers. For example,

• Key distribution and reciprocal (mutual) authentication schemes, such as those discussed in Chapters 14 and 15. In such schemes, two communicating parties cooperate by exchanging messages to distribute keys and/or authenticate each other. In many cases, nonces are used for handshaking to prevent replay attacks. The use of random numbers for the nonces frustrates an opponent's efforts to determine or guess the nonce, in order to repeat an obsolete transaction.

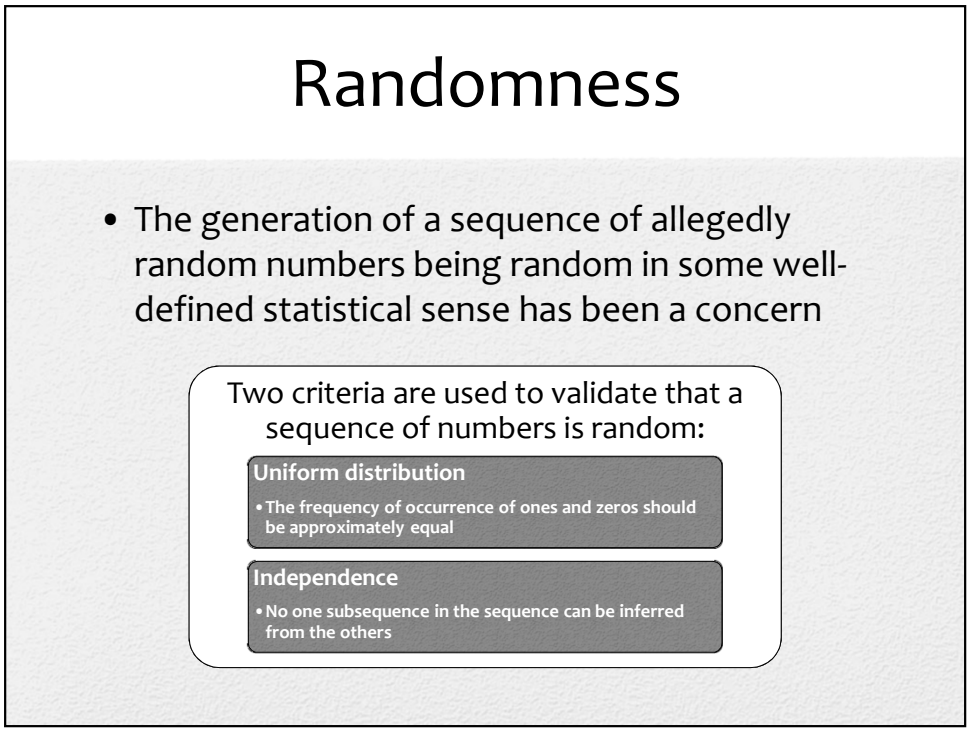
• Session key generation. We will see a number of protocols in this book where a secret key for symmetric encryption is generated for use for a particular transaction (or session) and is valid for a short period of time. This key is

generally called a session key.

- Generation of keys for the RSA public-key encryption algorithm (described in Chapter 9).

- Generation of a bit stream for symmetric stream encryption (described in this chapter).

These applications give rise to two distinct and not necessarily compatible requirements for a sequence of random numbers: randomness and unpredictability.



Traditionally, the concern in the generation of a sequence of allegedly random numbers has been that the sequence of numbers be random in some well-defined statistical sense. The following two criteria are used to validate that a sequence of numbers is random:

• Uniform distribution: The distribution of bits in the sequence should be uniform; that is, 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.

Although there are well-defined tests for determining that a sequence of bits matches a particular distribution, such as the uniform distribution, there is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate
if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently

strong. That is, if each of a number of tests fails to show that a sequence of bits is

not independent, then we can have a high level of confidence that the sequence is in

fact independent.

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

In applications such as reciprocal authentication, session key generation, and stream ciphers, 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. Although true random numbers are used in some applications, they have their limitations, such as inefficiency, as is discussed shortly. Thus, it is more common to implement algorithms that generate sequences of numbers that appear to be random. In this latter case, 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*

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. However, if the algorithm is

good, the resulting sequences will pass many tests of randomness. Such numbers are

referred to as pseudorandom numbers .

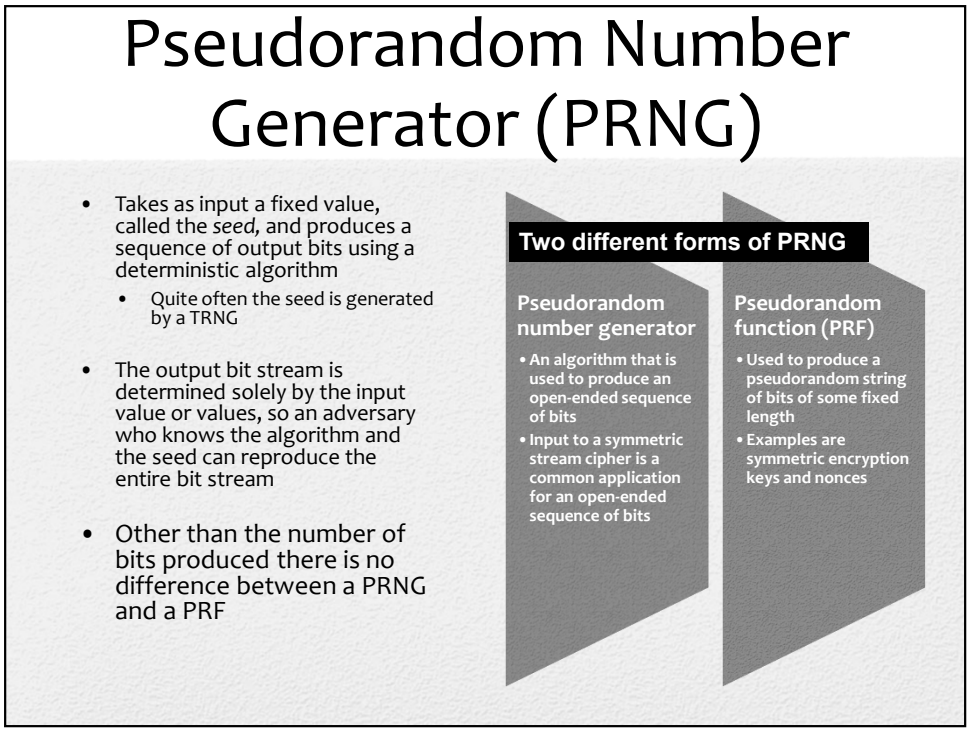
True Random Number Generator (TRNG)

- 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

A TRNG takes as input a source that is effectively random; the source is often referred to as an entropy source.

In essence, the entropy source is drawn from the physical environment of the computer and could include 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; this is discussed in Section 7.6.



In contrast, a 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. Typically, as shown, there is some feedback path by which some of the results of the algorithm are fed back as input as additional output bits are produced. The important thing to note is that the output bit stream is determined solely by the input value or values, so that an adversary who knows the algorithm and the seed can reproduce the entire bit stream.

Figure 7.1 shows two different forms of PRNGs, based on application.

• Pseudorandom number generator: An algorithm that is used to produce an open-ended sequence of bits is referred to as a PRNG. A common application

for an open-ended sequence of bits is as input to a symmetric stream cipher, as
discussed in Section 7.4. Also, see Figure 3.1a.

• Pseudorandom function (PRF): A PRF is used to produced a pseudorandom string of bits of some fixed length. Examples are symmetric encryption keys and nonces. Typically, the PRF takes as input a seed plus some context specific values, such as a user ID or an application ID. A number of examples of PRFs will be seen throughout this book, notably in Chapters 17 and 18.

Other than the number of bits produced, there is no difference between a PRNG and a PRF. The same algorithms can be used in both applications. Both require a seed and both must exhibit randomness and unpredictability. Further, a PRNG application may also employ context-specific input. In what follows, we make no distinction between these two applications.

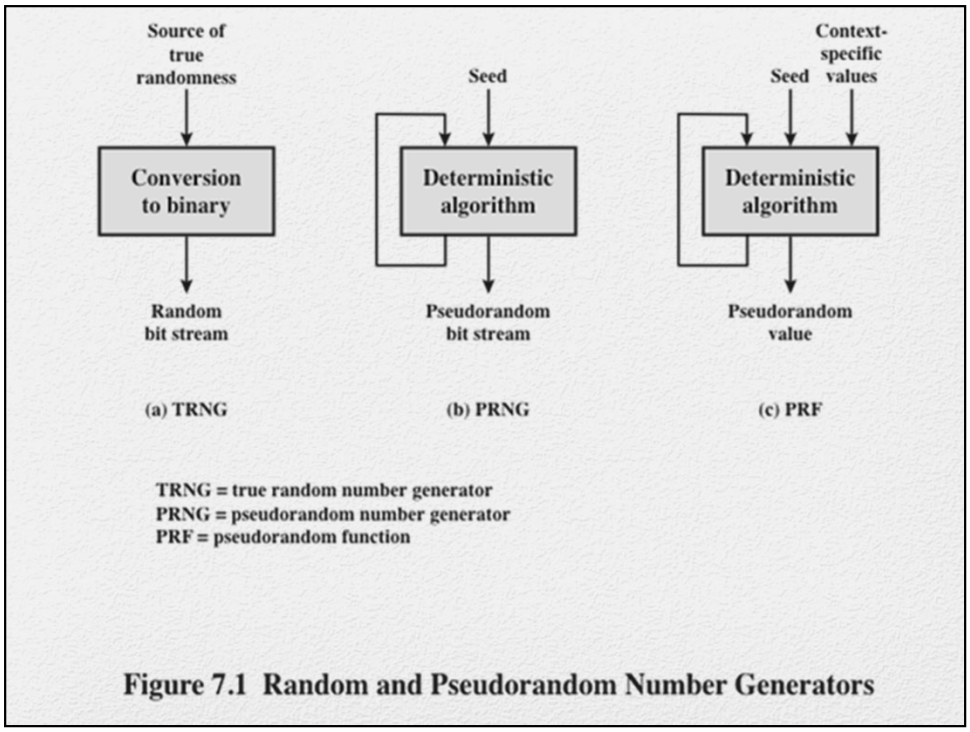


Figure 7.1 Random and Pseudorandom Number Generators

Figure 7.1 contrasts a true random number generator (TRNG) with two forms of pseudorandom number generators.

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

When a PRNG or PRF is used for a cryptographic application, then the basic requirement
is that an adversary who does not know the seed is unable to determine the pseudorandom string. For example, if the pseudorandom bit stream is used in a stream cipher, then knowledge of the pseudorandom bit stream would enable the adversary to recover the plaintext from the ciphertext. Similarly, we wish to protect the output value of a PRF. In this latter case, consider the following scenario. A 128-bit seed, together with some context-specific values, are used to generate a 128-bit secret key that is subsequently used for symmetric encryption. Under normal circumstances, a 128-bit key is safe from a brute-force attack. However, if the PRF does not generate effectively random 128-bit output values, it may be possible for an adversary to narrow the possibilities and successfully use a brute force attack.

This general requirement for secrecy of the output of a PRNG or PRF leads to specific requirements in the areas of randomness, unpredictability, and the characteristics
of the seed. We now look at these in turn.

Randomness

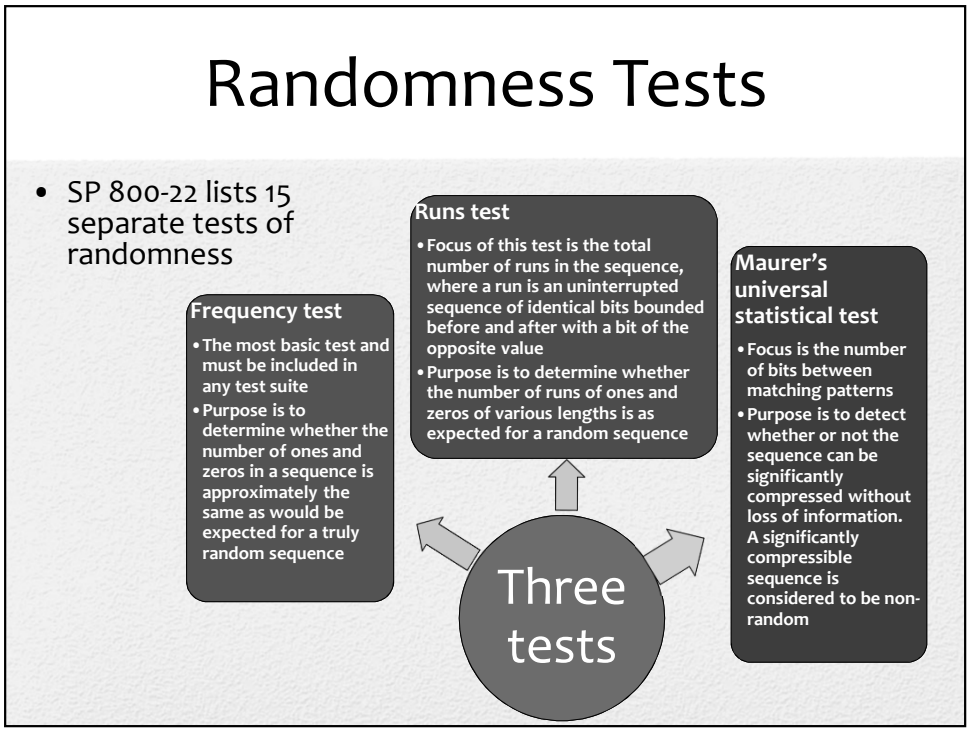
- The generated bit stream needs to appear random even though it is deterministic
- There is no single test that can determine if a PRNG generates numbers that have the characteristic of randomness
  - If the PRNG exhibits randomness on the basis of multiple tests, then it can be assumed to satisfy the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

In terms of randomness, the requirement for a PRNG is that the generated bit stream 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. The best that can be done is to apply a sequence of tests to the PRNG. 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 (A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications) specifies that the tests should seek to establish the following three characteristics.

• Uniformity: At any point in the generation of a sequence of random or pseudorandom bits, the occurrence of a zero or one is equally likely, i.e., the probability of each is exactly 1/2. The expected number of zeros (or ones) is n /2, where n = the sequence length.

• Scalability: Any test applicable to a sequence can also be applied to subsequences extracted at random. If a sequence is random, then any such extracted subsequence should also be random. Hence, any extracted subsequence should pass any test for randomness.

• Consistency: The behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed or an TRNG on the basis of an output produced from a single physical output.



SP 800-22 lists 15 separate tests of randomness. An understanding of these tests requires a basic knowledge of statistical analysis, so we don't attempt a technical description here. Instead, to give some flavor for the tests, we list three of the tests and the purpose of each test, as follows.

• Frequency test: This is the most basic test and must be included in any test suite. The purpose of this test 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: The 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. The purpose of the runs test 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: The focus of this test is the number of bits between matching patterns (a measure that is related to the length of a compressed

sequence). The purpose of the test 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)

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 also 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 provide a test of unpredictability. If the generated bit stream appears random, then it is not possible to predict some bit or bit sequence from knowledge of any previous bits. Similarly, if the bit sequence appears random, then there is no feasible way to deduce the seed based on the bit sequence. That is, 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

For cryptographic applications, the seed that serves as input to the PRNG must be secure. Because the PRNG is a deterministic algorithm, if the adversary can deduce the seed, then the output can also be determined. Therefore, the seed must be unpredictable. In fact, the seed itself must be a random or pseudorandom number.

Generation of Seed Input to PRNG

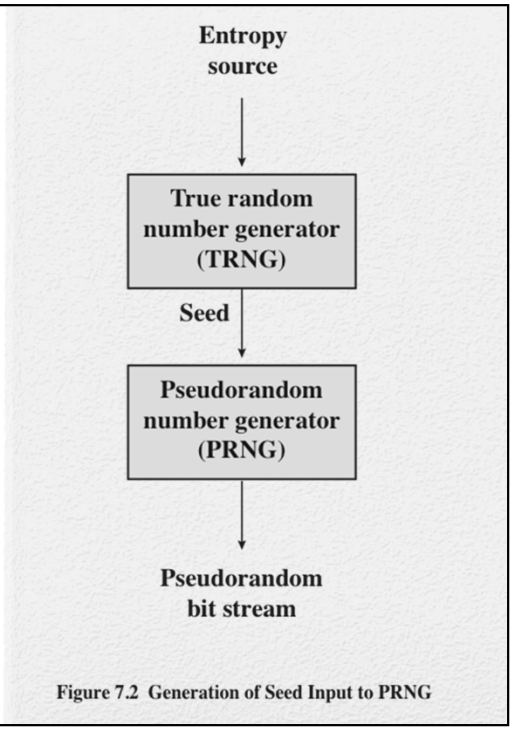


Figure 7.2 Generation of Seed Input to PRNG

Typically, the seed is generated by a TRNG, as shown in Figure 7.2. This is the scheme recommended by SP800-90. The reader may wonder, if a TRNG is available, why it is necessary to use a PRNG. If the application is a stream cipher, then a TRNG is not practical. The sender would need to generate a keystream of bits as long as the plaintext and then transmit the keystream and the ciphertext securely to the receiver. If a PRNG is used, the sender need only find a way to deliver the stream cipher key, which is typically 54 or 128 bits, to the receiver in a secure fashion.

Even in the case of a PRF application, in which only a limited number of bits is generated, it is generally desirable to use a TRNG to provide the seed to the PRF and use the PRF output rather than use the TRNG directly. As is explained in Section 7.6, a TRNG may produce a binary string with some bias. The PRF would have the effect of "randomizing" the output of the TRNG so as to eliminate that bias.

Finally, the mechanism used to generate true random numbers may not be able to generate bits at a rate sufficient to keep up with the application requiring the random bits.

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

Cryptographic PRNGs have been the subject of much research over the years, and a wide variety of algorithms have been developed. These fall roughly into two
categories.

• Purpose-built algorithms: These are algorithms designed specifically and solely for the purpose of generating pseudorandom bit streams. Some of these algorithms are used for a variety of PRNG applications; several of these are described in the next section. Others are designed specifically for use in a stream cipher. The most important example of the latter is RC4, described in Section 7.5.

• Algorithms based on existing cryptographic algorithms: Cryptographic algorithms
have the effect of randomizing input data. Indeed, this is a requirement of such algorithms. For example, if a symmetric block cipher produced ciphertext that had certain regular patterns in it, it would aid in the process of cryptanalysis. Thus, cryptographic algorithms can serve as the core of PRNGs.

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

—Symmetric block ciphers: This approach is discussed in Section 7.3.

—Asymmetric ciphers: The number theoretic concepts used for an asymmetric cipher can also be adapted for a PRNG; this approach is examined in Chapter 10.

—Hash functions and message authentication codes: This approach is examined in Chapter 12.

Any of these approaches can yield a cryptographically strong PRNG. A purpose-built algorithm may be provided by an operating system for general use. For applications that already use certain cryptographic algorithms for encryption or authentication, it makes sense to reuse the same code for the PRNG. Thus, all of these approaches are in common use.

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	$0 \le c < m$
X_0	the starting value, or seed	$0 \le X_0 < m$

- 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 $0 \le X_n < m$
- The selection of values for a, c, and m is critical in developing a good random number generator

A widely used technique for pseudorandom number generation is an algorithm first

proposed by Lehmer [LEHM51], which is known as the linear congruential method.

The algorithm is parameterized with four numbers, as follows:

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

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.

We would like m to be very large, so that there is the potential for producing

a long series of distinct random numbers. A common criterion is that m be nearly

equal to the maximum representable nonnegative integer for a given computer.

Thus, a value of m near to or equal to 231 is typically chosen.

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*

A popular approach to generating secure pseudorandom numbers is known as the *Blum, Blum, Shub* (BBS) generator (see Figure 7.3), named for its developers.

It has perhaps the strongest public proof of its cryptographic strength of any purpose-built algorithm.

The BBS is referred to as a cryptographically secure pseudorandom bit generator (CSPRBG). A CSPRBG is defined as one that passes the next-bit test , which, in turn, is defined as follows [MENE97]: A pseudorandom bit generator is said to pass 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. In other words, given the first k bits of the sequence,

there is not a practical algorithm that can even allow you to state that the next bit

will be 1 (or 0) with probability greater than 1/2. For all practical purposes, the

sequence is unpredictable. The security of BBS is based on the difficulty of factoring n .

That is, given n , we need to determine its two prime factors p and q .

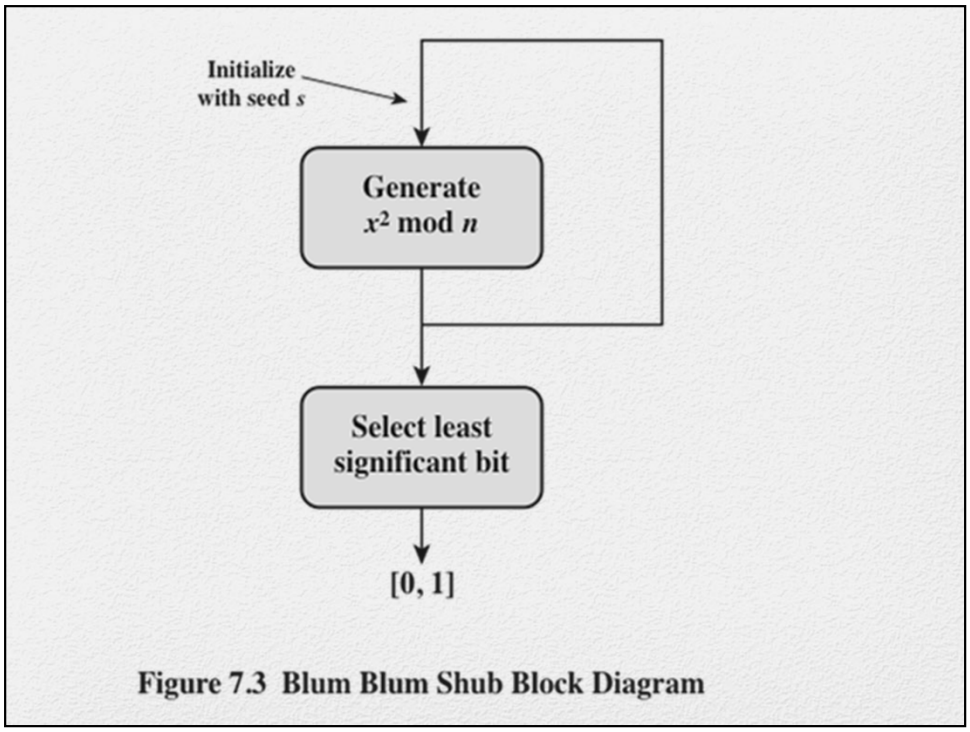


Figure 7.3 Blum Blum Shub Block Diagram

A popular approach to generating secure pseudorandom numbers is known as the Blum, Blum, Shub (BBS) generator (see Figure 7.3), named for its developers

[BLUM86].

Table 7.1

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

Table 7.1, shows an example of BBS operation.

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

Two approaches that use a block cipher to build a PNRG have gained widespread

acceptance: the CTR mode and the OFB mode. The CTR mode is recommended in

NIST SP 800-90, in the ANSI standard X9.82 (Random Number Generation), and in

RFC 4086. The OFB mode is recommended in X9.82 and RFC 4086.

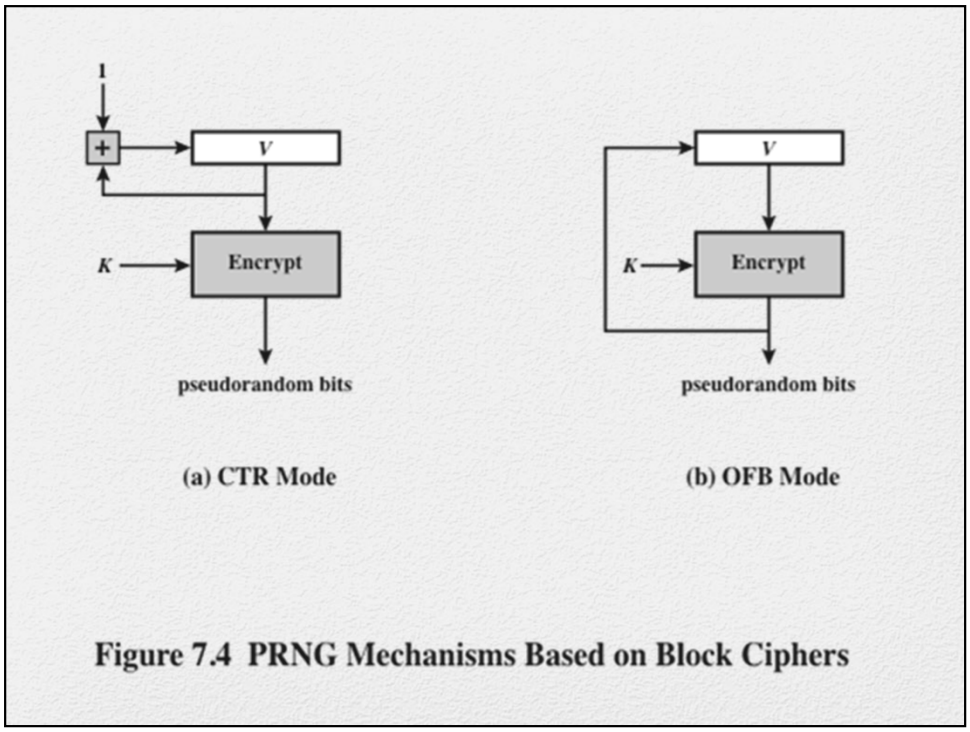


Figure 7.4 PRNG Mechanisms Based on Block Ciphers

Figure 7.4 illustrates the two methods. In each case, the seed consists of two parts: the encryption key value and a value V that will be updated after each block of pseudorandom numbers is generated. Thus, for AES-128, the seed consists of a 128-bit key and a 128-bit V value. In the CTR case, the value of V is incremented by 1 after each encryption. In the case of OFB, the value of V is updated to equal the value of the preceding PRNG block. In both cases, pseudorandom bits are produced one block at a time (e.g., for AES, PRNG bits are generated 128 bits at a time).

Table 7.2

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

Example Results for PRNG Using OFB

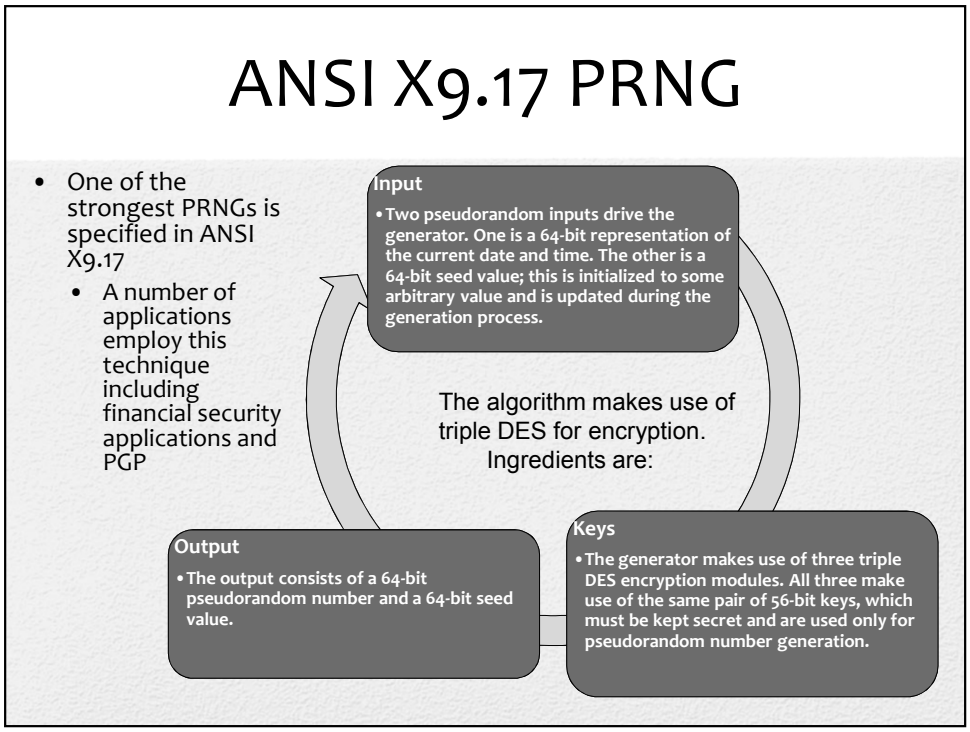
For the OFB PRNG, Table 7.2 shows the first eight output blocks (1024 bits) with two rough measures of security. The second column shows the fraction of one bits in each 128-bit block. This corresponds to one of the NIST tests. The results indicate that the output is split roughly equally between zero and one bits. The third column shows the fraction of bits that match between adjacent blocks. If this number differs substantially from 0.5, that suggests a correlation between blocks, which could be a security weakness. The results suggest no correlation.

Table 7.3

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

Example Results for PRNG Using CTR

Table 7.3 shows the results using the same key and V values for CTR mode. Again, the results are favorable.



One of the strongest (cryptographically speaking) PRNGs is specified in ANSI X9.17. A number of applications employ this technique, including financial security

applications and PGP (the latter described in Chapter 19).

Figure 7.5 illustrates the algorithm, which makes use of triple DES for encryption.

The ingredients are as follows.

• Input: Two pseudorandom inputs drive the generator. One is a 64-bit representation

of the current date and time, which is updated on each number

generation. The other is a 64-bit seed value; this is initialized to some arbitrary value and is updated during the generation process.

• Keys: The generator makes use of three triple DES encryption modules. All three make use of the same pair of 56-bit keys, which must be kept secret and are used only for pseudorandom number generation.

• Output: The output consists of a 64-bit pseudorandom number and a 64-bit seed value.

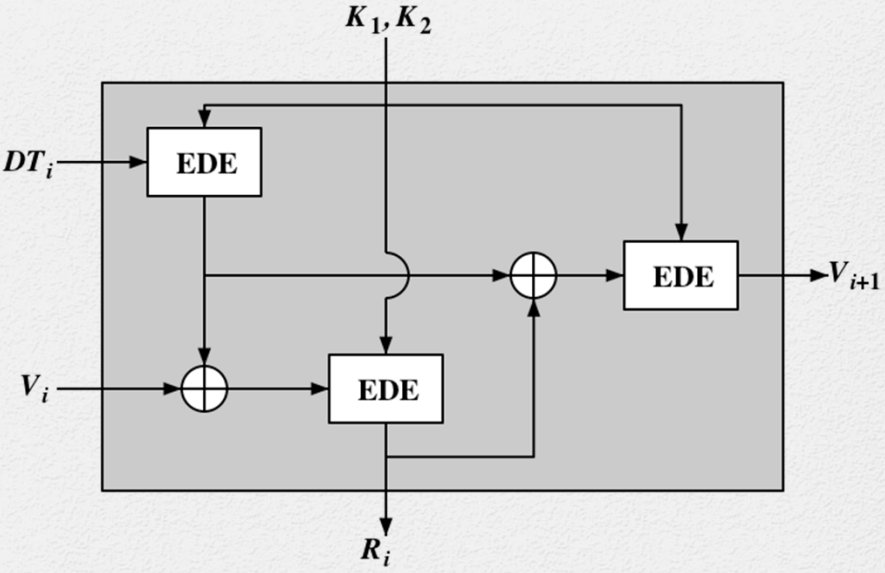


Figure 7.5 ANSI X9.17 Pseudorandom Number Generator

Figure 7.5 illustrates the algorithm, which makes use of triple DES for encryption.

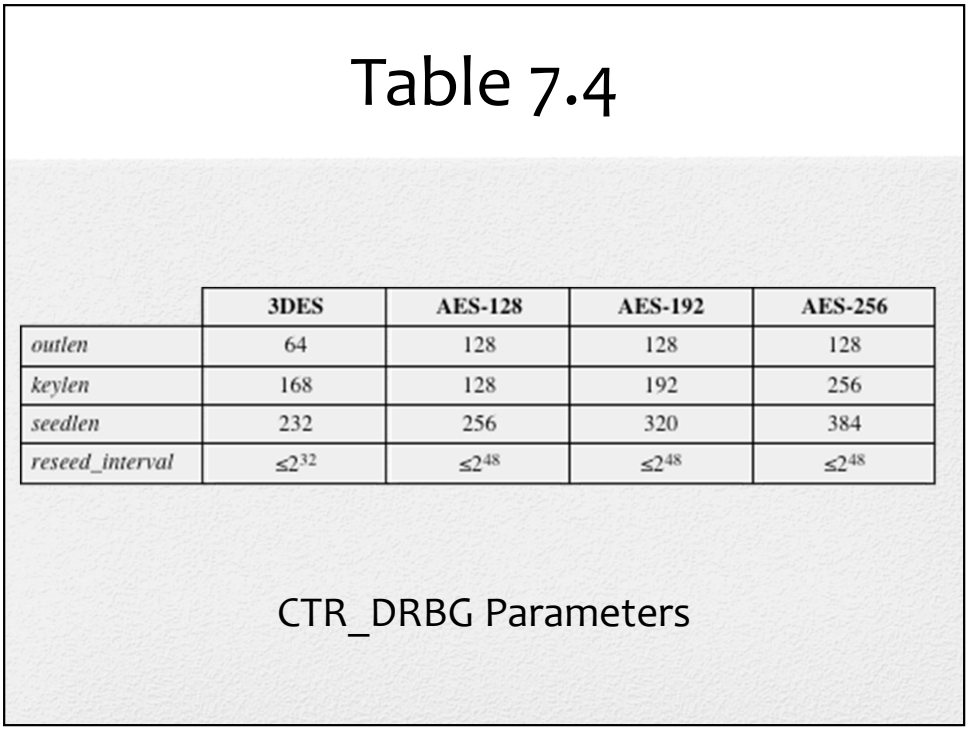
NIST CTR_DRBG

- Counter mode-deterministic random bit generator
- PRNG defined in NIST SP 800-90 based on the CTR mode of operation
- Is widely implemented and is part of the hardware random number generator implemented on all recent Intel processor chips
- DRBG assumes that an entropy source is available to provide random bits
  - Entropy is an information theoretic concept that measures unpredictability or randomness
- The encryption algorithm used in the DRBG may be 3DES with three keys or AES with a key size of 128, 192, or 256 bits

We now look more closely at the details of the PRNG defined in NIST SP 800-90
based on the CTR mode of operation. The PRNG is referred to as CTR_DRBG
(counter mode–deterministic random bit generator). CTR_DRBG is widely implemented
and is part of the hardware random number generator implemented on all
recent Intel processor chips (discussed in Section 7.6).

The DRBG assumes that an entropy source is available to provide random
bits. Typically, the entropy source will be a TRNG based on some physical source.
Other sources are possible if they meet the required entropy measure of the application.
Entropy is an information theoretic concept that measures unpredictability,
or randomness; see Appendix F for details. The encryption algorithm used in the
DRBG may be 3DES with three keys or AES with a key size of 128, 192, or 256 bits.

Table 7.4



	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

Four parameters are associated with the algorithm:

• Output block length (outlen): Length of the output block of the encryption algorithm.

• Key length (keylen): Length of the encryption key.

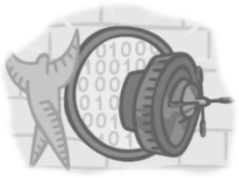
• Seed length (seedlen): The seed is a string of bits that is used as input to a DRBG mechanism. The seed will determine a portion of the internal state of the DRBG, and its entropy must be sufficient to support the security strength of the DRBG. seedlen = outlen + keylen .

• Reseed interval (reseed_interval): Length of the encryption key. It is the maximum
number of output blocks generated before updating the algorithm with a new seed.

Table 7.4 lists the values specified in SP 800-90 for these parameters.

CTR_DRBG

Functions



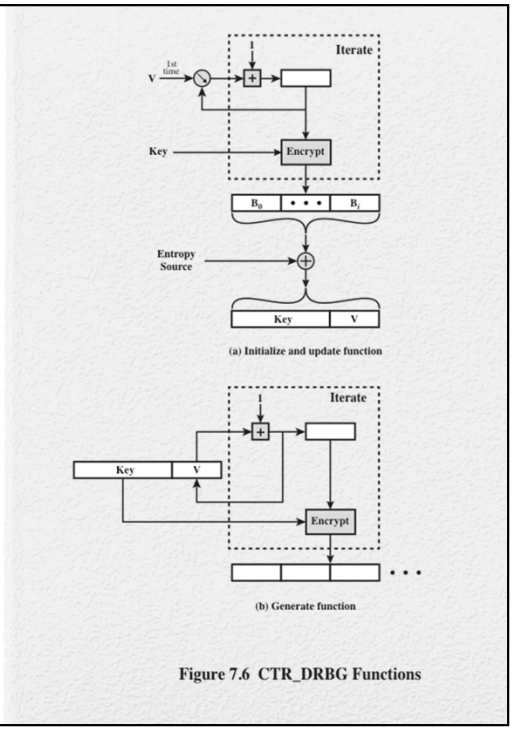


Figure 7.6 CTR_DRBG Functions

Figure 7.6 shows the two principal functions that comprise CTR_DRBG.
We first consider how CTR_DRBG is initialized, using the initialize and update function (Figure 7.6a). Recall that the CTR block cipher mode requires both an encryption key K and an initial counter value, referred to in SP 800-90 as the counter V . The combination of K and V is referred to as the seed . To start the DRGB operation, initial values for K and V are needed, and can be chosen arbitrarily. As an example, the Intel Digital Random Number Generator, discussed in Section 7.6, uses the values K = 0 and V = 0. These values are used as parameters for the CTR mode of operation to produce at least seedlen bits. In addition, exactly seedlen bits must be supplied from what is referred to as an entropy source . Typically, the entropy source would be some form of TRNG.

With these inputs, the CTR mode of encryption is iterated to produce a Sequence of output blocks, with V incremented by 1 after each encryption. The process
continues until at least seedlen bits have been generated. The leftmost seedlen bits of output are then XORed with the seedlen entropy bits to produce a new seed. In turn, the leftmost keylen bits of the seed form the new key and the rightmost outlen bits of the seed form the new counter value V .

Once values of Key and V are obtained, the DRBG enters the generate phase and is able to generate pseudorandom bits, one output block at a time (Figure 7.6b). The encryption function is iterated to generate the number of pseudorandom bits desired. Each iteration uses the same encryption key. The counter value V is incremented by 1 for each iteration.

To enhance security, the number of bits generated by any PRNG should be limited. CTR_DRGB uses the parameter reseed_interval to set that limit. During the generate phase, a reseed counter is initialized to 1 and then incremented with each iteration (each production of an output block). When the reseed counter reaches reseed_interval , the update function is invoked (Figure 7.6a). The update function is the same as the initialize function. In the update case the Key and V values last used by the generate function serve as the input parameters to the update function. The update function takes seedlen new bits from an entropy source and produces a new seed (Key, V). The generate function can then resume production of pseudorandom bits. Note that the result of the update function is to change both the Key and V values used by the generate function.

Stream Ciphers

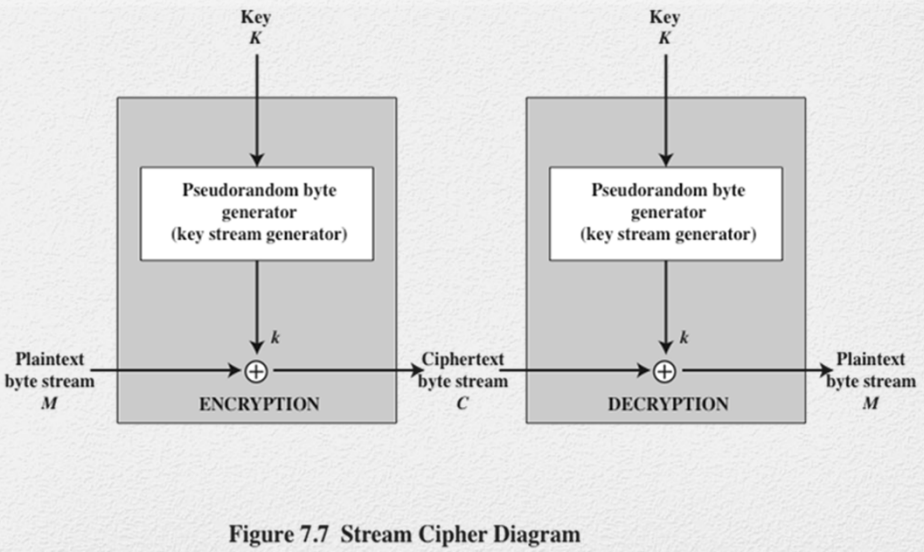
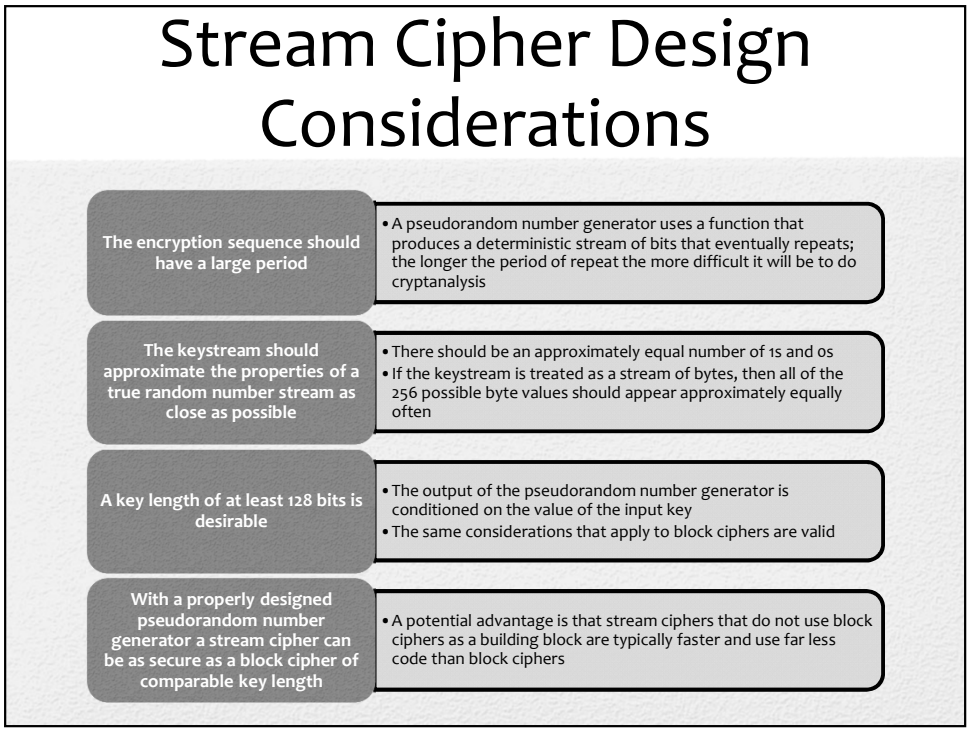


Figure 7.7 Stream Cipher Diagram

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 7.7 is a representative diagram of stream cipher structure. In this structure, a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. The output of the generator, called a keystream , is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.



The stream cipher is similar to the one-time pad discussed in Chapter 2. The difference is that a one-time pad uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream.

[KUMA97] lists the following important design considerations for a stream cipher.

1. 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. This is essentially the same consideration that was discussed with reference to the Vigenère cipher, namely that the longer the keyword the more difficult the cryptanalysis.

2. The keystream should approximate the properties of a true random number

stream as close as possible. For example, 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. The more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.

3. Note from Figure 7.7 that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations that apply to block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

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 of a stream cipher is that stream ciphers that do not use block ciphers as a building block are typically faster and use far less code than do block ciphers. The example in this chapter, RC4, can be implemented in just a few lines of code. In recent years, this advantage has diminished with the introduction of AES, which is quite efficient in software. Furthermore, hardware acceleration techniques are now available for AES. For example, the Intel AES Instruction Set has machine instructions for one round of encryption and decryption and key generation. Using the hardware instructions results in speedups of about an order of magnitude compared to pure software implementations [XU10].

One advantage of a block cipher is that you can reuse keys. In contrast, if two

plaintexts are encrypted with the same key using a stream cipher, then cryptanalysis
is often quite simple [DAWS96]. If the two ciphertext streams are XORed together,
the result is the XOR of the original plaintexts. If the plaintexts are text strings,
credit card numbers, or other byte streams with known properties, then cryptanalysis
may be successful.

For applications that require encryption/decryption of a stream of data, such as
over a data communications channel or a browser/Web link, a stream cipher might
be the better alternative. For applications that deal with blocks of data, such as file
transfer, e-mail, and database, block ciphers may be more appropriate. However,
either type of cipher can be used in virtually any application.

A stream cipher can be constructed with any cryptographically strong PRNG,
such as the ones discussed in Sections 7.2 and 7.3. In the next section, we look at a
stream cipher that uses a PRNG designed specifically for the stream cipher.

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
- Used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards that have been defined for communication between Web browsers and servers
- Is also used in the Wired Equivalent Privacy (WEP) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable

key size stream cipher with byte-oriented operations. The algorithm is based

on the use of a random permutation. Analysis shows that the period of the cipher

is overwhelmingly likely to be greater than 10^{100} [ROBS95a]. 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 Secure Sockets Layer/Transport

Layer Security (SSL/TLS) standards that have been defined for communication between

Web browsers and servers. It is also used in the Wired Equivalent Privacy

(WEP) protocol and the newer WiFi Protected Access (WPA) protocol that are

part of the IEEE 802.11 wireless LAN standard. RC4 was kept as a trade secret by

RSA Security. In September 1994, the RC4 algorithm was anonymously posted on

the Internet on the Cypherpunks anonymous remailers list.

The RC4 algorithm is remarkably simple and quite easy to explain. A variable-length

key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state

vector S, with elements S[0], S[1], c , S[255]. At all times, S contains a permutation

of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte k (see

Figure 7.7) is generated from S by selecting one of the 255 entries in a systematic

fashion. As each value of k is generated, the entries in S are once again permuted.

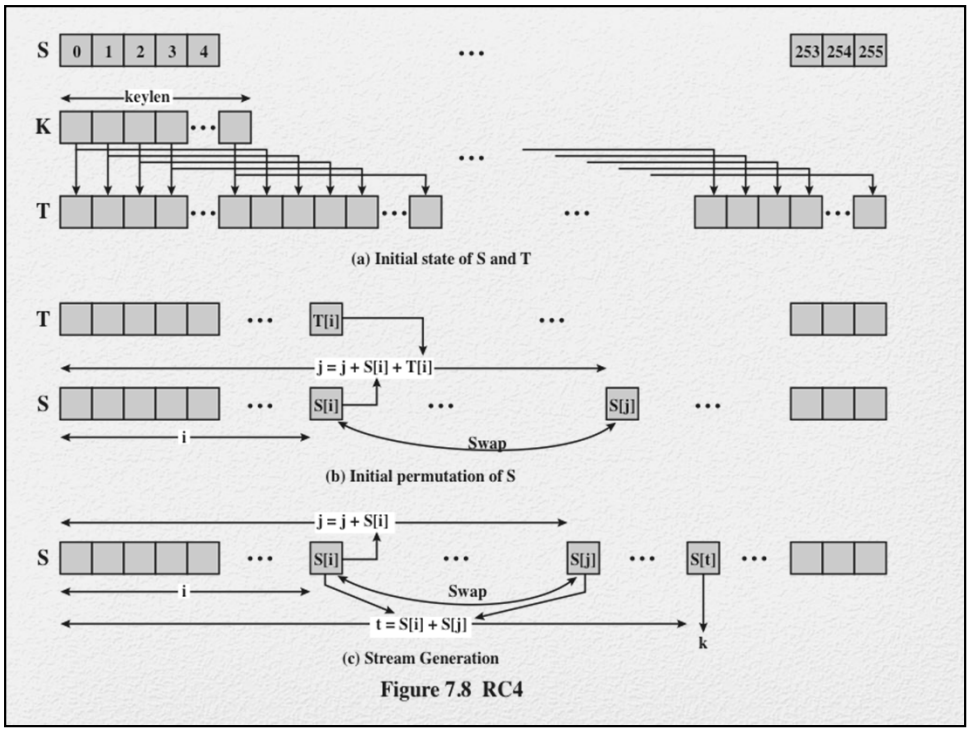
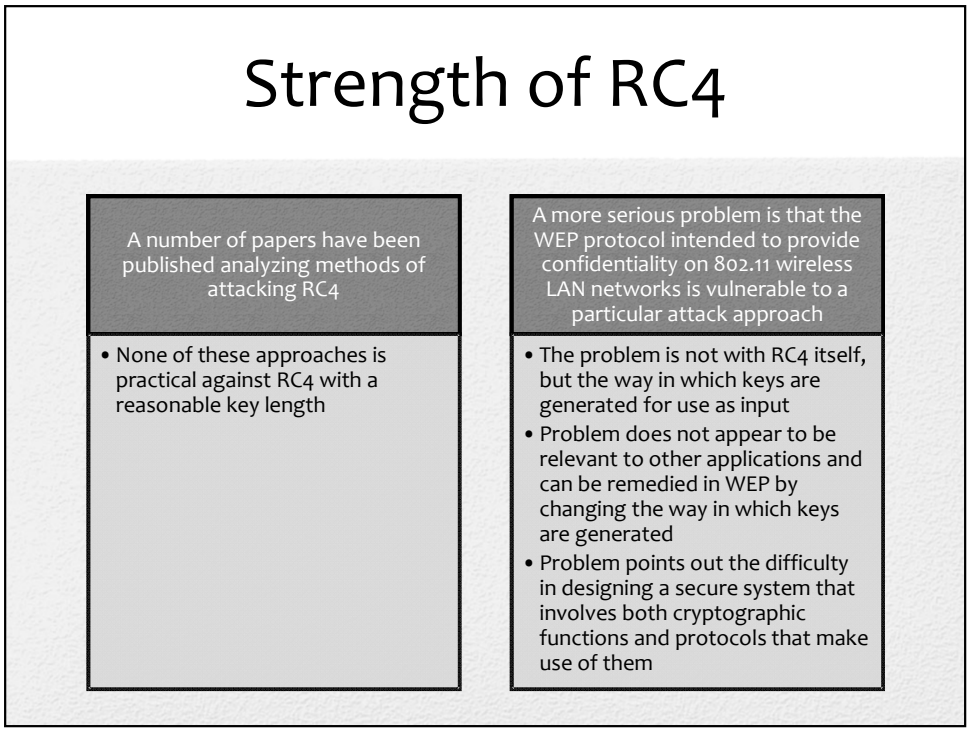


Figure 7.8 RC4

Figure 7.8 illustrates the RC4 logic.

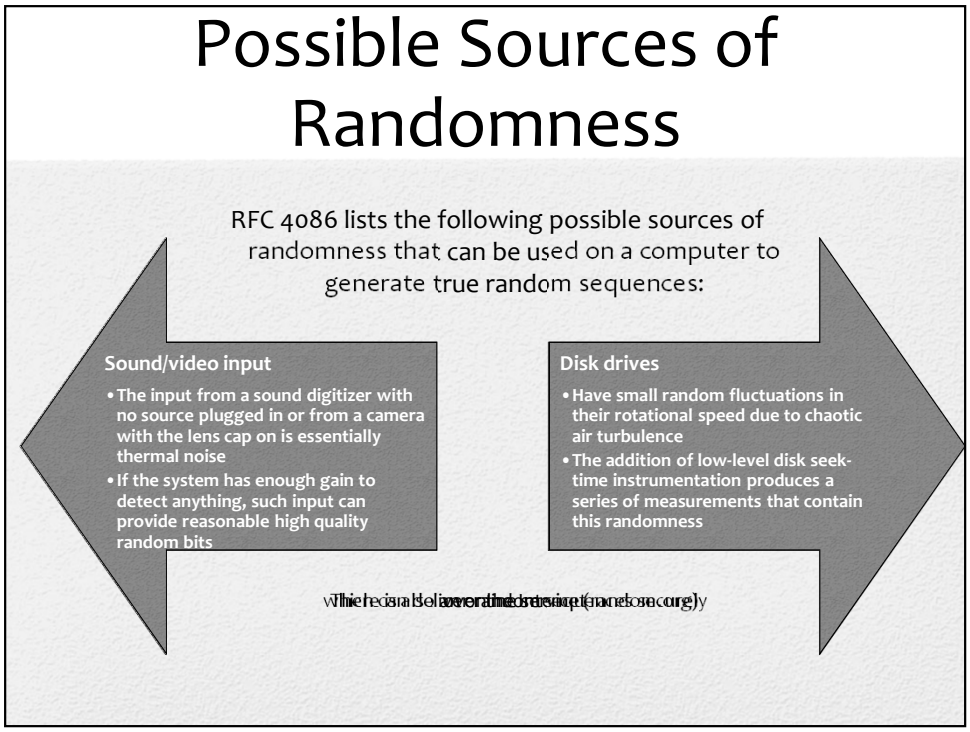


A number of papers have been published analyzing methods of attacking RC4 (e.g., [KNUD98], [FLUH00], [MANT01]). None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem is reported in [FLUH01]. The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4. This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic functions and protocols that make use of them.

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

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 [JUN99]. 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.



RFC 4086 lists the following possible sources of randomness that, with care, easily can be used on a computer to generate true random sequences.

• Sound/video input: Many computers are built with inputs that digitize some real-world analog source, such as sound from a microphone or video input from a camera. 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 reasonably high quality random bits.

• Disk drives: Disk drives have small random fluctuations in their rotational speed due to chaotic air turbulence [JAKO98]. The addition of low-level disk seek-time instrumentation produces a series of measurements that contain this randomness. Such data is usually highly correlated, so significant processing
is needed. Nevertheless, experimentation a decade ago showed that, with such processing, even slow disk drives on the slower computers of that day could easily produce 100 bits a minute or more of excellent random data.

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

Table 7.5

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

Comparison of PRNGs and TRNGs

Table 7.5 summarizes the principal differences between PRNGs and TRNGs. PRNGs are efficient, meaning they can produce many numbers in a short time, and deterministic, meaning that a given sequence of numbers can be reproduced at a later date if the starting point in the sequence is known. Efficiency is a nice characteristic
if your application needs many numbers, and determinism is handy if you need to replay the same sequence of numbers again at a later stage. PRNGs are typically also periodic, which means that the sequence will eventually repeat itself. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes.

TRNGs are generally rather inefficient compared to PRNGs, taking considerably longer time to produce numbers. This presents a difficulty in many applications. For example, cryptography system in banking or national security might need to generate millions of random bits per second. TRNGs are also nondeterministic, meaning that a given sequence of numbers cannot be reproduced, although the same sequence may of course occur several times by chance. TRNGs have no period.

Skew

- A TRNG may produce an output that is biased in some way, such as having more ones than zeros or vice versa
  - Deskewing algorithms
    - Methods of modifying a bit stream to reduce or eliminate the bias
    - One approach is to pass the bit stream through a hash function such as MD5 or SHA-1
    - RFC 4086 recommends collecting input from multiple hardware sources and then mixing these using a hash function to produce random output
- Operating systems typically provide a built-in mechanism for generating random numbers
  - Linux uses four entropy sources: mouse and keyboard activity, disk I/O operations, and specific interrupts
    - Bits are generated from these four sources and combined in a pooled buffer
    - When random bits are needed the appropriate number of bits are read from the buffer and passed through the SHA-1 hash function

A TRNG may produce an output that is biased in some way, such as having more ones than zeros or vice versa. Various methods of modifying a bit stream to reduce or eliminate the bias have been developed. These are referred to as deskewing algorithms . One approach to deskew is to pass the bit stream through a hash function,
such as MD5 or SHA-1 (described in Chapter 11). The hash function produces an n -bit output from an input of arbitrary length. For deskewing, blocks of m input bits, with $m \geq n$, can be passed through the hash function. RFC 4086 recommends collecting
input from multiple hardware sources and then mixing these using a hash function to produce random output.

Operating systems typically provide a built-in mechanism for generating random numbers. For example, Linux uses four entropy sources: mouse and keyboard activity, disk I/O operations, and specific interrupts. Bits are generated from these four sources and combined in a pooled buffer. When random bits are needed, the appropriate number of bits are read from the buffer and passed through the SHA-1 hash function [GUTT06].

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

As was mentioned, 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 (DRNG) [TAYL11], offered on new multicore chips since May 2012.

Two notable aspects of the DRNG:

1. It is implemented entirely in hardware. This provides greater security than a facility
that includes a software component. A hardware-only implementation should also be able to achieve greater computation speed than a software module.

2. The entire DRNG is on the same multicore chip as the processors. This eliminates

the I/O delays found in other hardware random number generators.

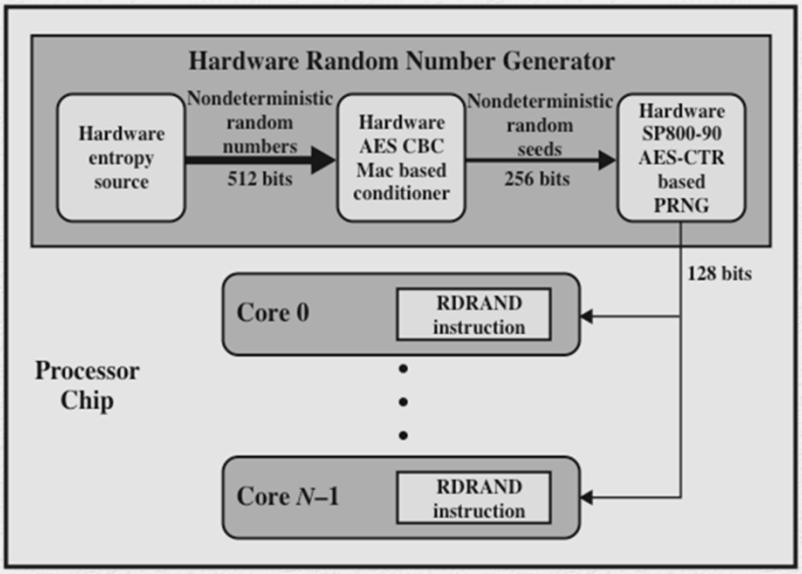


Figure 7.9 Intel Processor Chip with Random Number Generator

Figure 7.9 shows the overall structure of the DRNG. The first stage of the DRNG generates random numbers from thermal noise. The heart of the stage consists of two inverters (NOT gates), with the output of each inverter connected to the input of the other. Such an arrangement has two stable states, with one inverter having an output of logical 1 and the other having an output of logical 0. The circuit is then configured
so that both inverters are forced to have the same indeterminate state (both inputs and both outputs at logical 1) by clock pulses. Random thermal noise within the inverters soon jostles the two inverters into a mutually stable state. Additional circuitry is intended to compensate for any biases or correlations. This stage is capable, with current hardware, of generating random bits at a rate of 4 Gbps.

The output of the first stage is generated 512 bits at a time. To assure that the bit stream does not have skew or bias, a second stage of processing randomizes its input using a cryptographic function. In this case, the function is referred to as CBC-MAC or CMAC, as specified in NIST SP 800-38B. In essence, CMAC encrypts its input using the cipher block chaining (CBC) mode (Figure 6.4) and outputs the final block. We examine CMAC in detail in Chapters 12. The output of

this stage is generated 256 bits at a time and is intended to exhibit true randomness
with no skew or bias.

While the hardware's circuitry generates random numbers from thermal noise much more quickly than its predecessors, it's still not fast enough for some of today's computing requirements. To enable the DRNG to generate random numbers as quickly as software PRNG, and also maintain the high quality of the random numbers, a third stage is added. This stage uses the 256-bit random numbers to seed a cryptographically secure PRNG that creates 128-bit numbers. From one 256-bit seed, the PRNG can output many pseudorandom numbers, exceeding the 3-Gbps rate of the entropy source. An upper bound of 511 128-bit samples can be generated per seed. The algorithm used for this stage is CTR_DRBG, described in Section 7.3.

The output of the DRNG is available to each of the cores on the chip via the RDRAND instruction. RDRAND retrieves a 16-, 32-, or 64-bit random value and makes it available in a software-accessible register.

Preliminary data from a pre-production sample on a system with a third generation Intel® Core™ family processor produced the following performance [INTE12]: up to 70 million RDRAND invocations per second, and a random data production rate of over 4 Gbps.

Intel DRNG Logical Structure

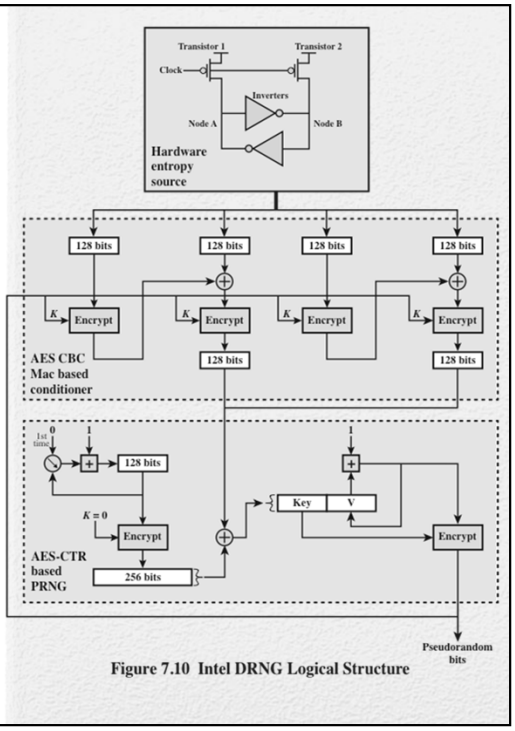


Figure 7.10 Intel DRNG Logical Structure

(Figure 7.10 is located on page 226 in textbook)

Figure 7. 10 provides a simplified view of the logical flow of the Intel DRNG. As was described,
the heart of the hardware entropy source is a pair of inverters that feed each other. Two transistors, driven by the same clock, force the inputs and outputs of both inverters to
the logical 1 state. Because this an unstable state, thermal noise will cause the configuration
to settle randomly into a stable state with either Node A at logical 1 and Node B at logical 0, or the reverse. Thus the module generates random bits at the clock rate.

The output of the entropy source is collected 512 bits at a time and used to feed to two CBC hardware implementations using AES encryption. Each implementation
takes two blocks of 128 bits of “plaintext” and encrypts using the CBC mode. The output of the second encryption is retained. For both CBC modules, an all-zeros key is used initially. Subsequently, the output of the PRNG stage is fed back to become the key for the conditioner stage.

The output of the conditioner stage consists of 256 bits. This block is provided as input to the update function of the PRNG stage. The update function is initialized with the all-zeros key and the counter value 0. The function is iterated twice to produce a 256-bit block, which is then XORed with the input from the conditioner stage. The results are used as the 128-bit key and the 128-bit seed for the generate

function. The generate function produces pseudorandom bits in 128-bit blocks.

Summary

- Principles of pseudorandom number generation
  - The use of random numbers
  - TRNGs, PRNGs, and PRFs
  - PRNG requirements
  - Algorithm design
- Pseudorandom number generators
  - Linear congruential generators
  - Blum Blum Shub generator
- Pseudorandom number generation using a block cipher
  - PRNG using block cipher modes of operation
  - ANSI X9.17 PRNG
  - NIST CTR_DRBG

- Stream ciphers
- RC4
  - Initialization of S
  - Stream generation
  - Strength of RC4
- True random number generators
  - Entropy sources
  - Comparison of PRNGs and TRNGs
  - Skew
  - Intel digital random number generator
  - DRNG hardware architecture
  - DRNG logical structure

Chapter 7 summary.