National Chung Cheng University
Department of Information Management

Pseudorandom Number Generation and Stream Ciphers

Pei-Ju (Julian) Lee

National Chung Cheng University
Information Security
pjlee@mis.ccu.edu.tw

Fall, 2016

Overview

- An important cryptographic function is cryptographically strong pseudorandom number generation.
  - Pseudorandom number generators (PRNGs)
  - True random number generators (TRNGs)
- In RSA public-key encryption scheme, it has to generate prime number
  - To determine if a given number N is prime is difficult
    - A brute-force approach: divide N by every odd integer less than $\sqrt{N}$ (if N's order is 10^{150}, then brute-force is unrealistic)
    - Some algorithms test the primality of a number by using a sequence of random integers as input to relatively simple computation
      - If the sequence is less than $\sqrt{10^{150}}$, it's primality can be determined --- Randomization

The Use of Random Numbers

- Network security algorithms and protocols based on cryptography make use of random binary numbers:
  - Key distribution and reciprocal authentication schemes
    - Nonces are used to prevent replay attack
  - Session key generation
    - Valid for a short period of time
  - Generation of keys for the RSA public-key encryption algorithm
  - Generation of a bit stream for symmetric stream encryption

Cont'd

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**

- 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.

Unpredictability

Requirement

The sequence of numbers be statistically random

The successive members of the sequence are unpredictable

Ex. reciprocal authentication, session key generation, and stream ciphers

- “True” random sequences: each number is statistically independent of other numbers in the sequence ->Inefficiency
- Alternative: an opponent not be able to predict future elements of the sequence on the basis of earlier elements

Test of Randomness

- There is no such test to “prove” 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

Pseudorandom Numbers

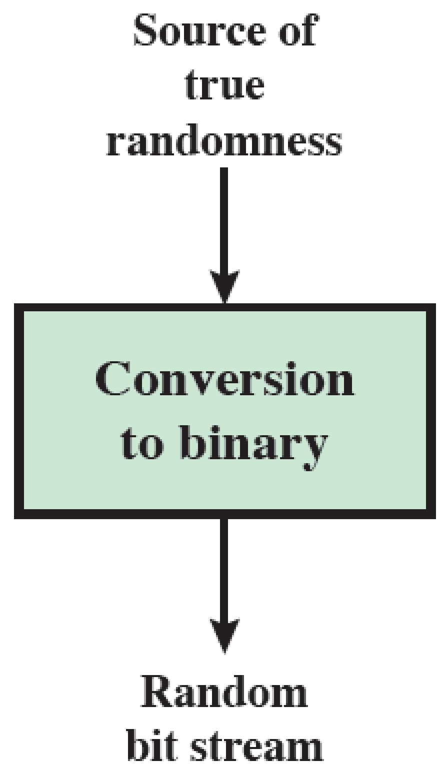
- Cryptographic applications typically make use of algorithmic techniques for random number generation
  - These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
    - These numbers can be computed by calculations
- If the algorithm is good, the resulting sequences will pass many tests of randomness and 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
  - E.g. keystroke timing patterns, disk electrical activity, mouse movements, and instantaneous values of the system clock
- Output: random binary bit stream

TRNG

- 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

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/values
  - so an adversary who knows the algorithm and the seed can reproduce the entire bit stream

Cont'd

Two different forms of PRNG (based on application)

Pseudorandom number generator (PRNG)

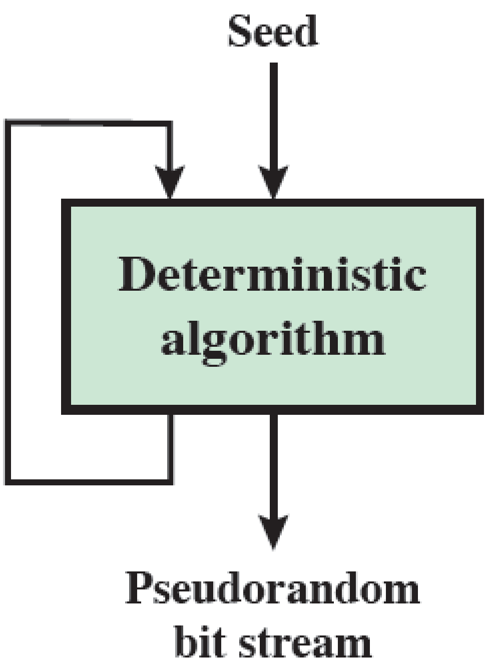
- **An algorithm that is used to produce an *open-ended sequence of bits***
- **Application: Input to a symmetric stream cipher**

Pseudorandom function (PRF)

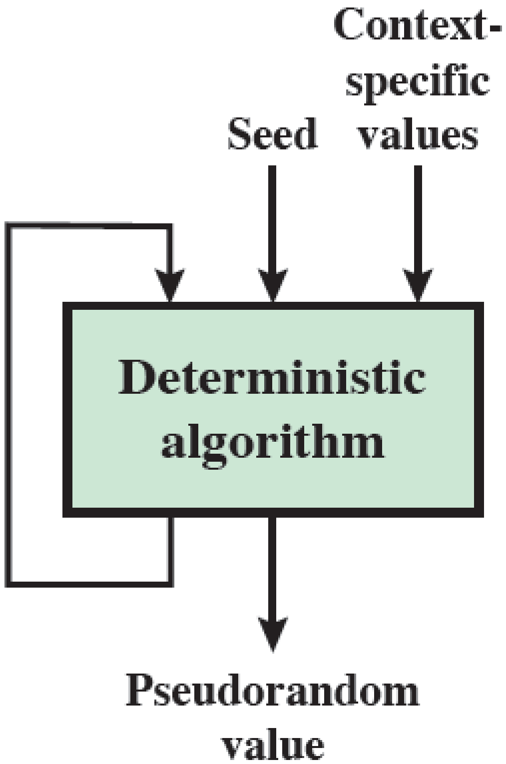
- **Used to produce a pseudorandom string of *bits of some fixed length***
- **Application: symmetric encryption keys and nonces**

- Other than the number of bits produced there is no difference between a PRNG and a PRF

PRNG and PRF



(b) PRNG



(c) PRF

- Input: seed plus some context specific value
- e.g. User ID, application ID

- 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. Further, a PRNG may also employ context-specific input

PRNG Requirements

- The basic requirement when a PRNG or PRF is used for a cryptographic application:
  - An adversary who does not know the seed is unable to determine the pseudorandom string
  - Especially to protect the output value of a PRF
- The requirement for secrecy of the output:
  - 1. Randomness
  - 2. Unpredictability
  - 3. Characteristics of the seed

1. Randomness

- Requirement: The generated bit stream *appear random* even though it is *deterministic*
  - No single test that can determine randomness;
  - However, if the PRNG exhibits randomness on the basis of multiple tests, then it can be assumed to satisfy the randomness requirement

Test of Randomness

- NIST SP 800-22 specifies that the tests should seek to establish 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
  - 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
  - Consistency
    - The behavior of a generator must be consistent across starting values (seeds)

Randomness Tests

- SP 800-22 lists 15 separate tests of randomness
  - Three tests in example:

Frequency test

- 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 total number of runs** of ones and zeros of various lengths is as expected for a random sequence
  - A run is an uninterrupted sequence of identical bits bounded before and after with a bit of the opposite value
  - e.g. 11100100001

Maurer's universal statistical test

- **The number of bits** between matching patterns (a measure that is related to the length of a compressed sequence)
- 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

Three tests

2. 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
    - Each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2

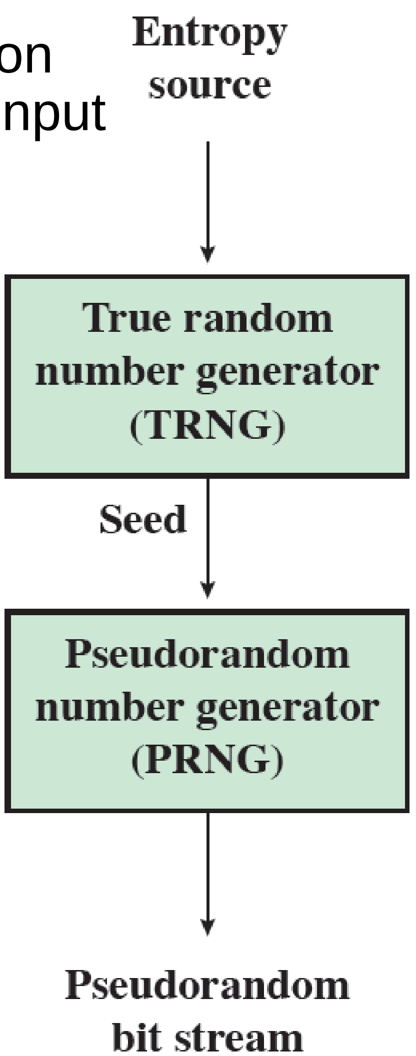
Cont'd

- 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.
  - 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).

3. Seed Requirements

- The seed that serves as input to the PRNG must be secure and unpredictable
  - Because the PRNG is a deterministic algorithm, anyone deduces the seed, then the output can be determined
  - The seed itself must be a random or pseudorandom number
  - Typically the seed is generated by TRNG (SP800-90)
  - Not feasible in stream cipher; however, can be used to generate stream cipher key only (54 or 128 bits)

- 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 for stream cipher e.g. RC4
  - Algorithms based on existing cryptographic algorithms
    - Have the effect of randomizing input data (to prevent a symmetric bloc cipher produce ciphertext that has certain regular patterns in it

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

- Symmetric block ciphers (following slides)
- Asymmetric ciphers (following classes)
- Hash functions and message authentication codes (following classes)

Pseudorandom Number Generators

- Two types of algorithms for PRNG
  - Linear Congruential Generators
  - Blum Blum Shub Generator

Linear Congruential Generator

- Parameters of the Algorithm [LEHM51]

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
- Large $m \approx 2^{31}$ is favorable (larger m, more random number can be generated)
  - E.g. a=7, c=0, m=32, X_0=1: {7,17,23,1,7,etc}, the sequence has a period of 4
  - The period becomes to 8 if we change a to 5

Random Number Generator Test

- Three tests in evaluating a random number generator [PARK88a]:
  - T_1: The function should be a full-period generating function. That is, the function should generate all the numbers from 0 through m-1 before repeating
  - T_2: The generated sequence should appear random
  - T_3: The function should implement efficiently with 32-bit arithmetic

Cont'd

- With respect to T_1, it can be shown that if *m* is prime and *c=0*, then for certain values of *a* the period of the generating function is *m-1*
- For 32-bit arithmetic, a convenient prime value of m is $2^{31}-1$

$$X_{n+1} = (aX_n) \bmod (2^{31}-1)$$

  - Only a handful of multipliers pass all three tests
  - One such value is $a = 7^5 = 16807$

Cont'd

- Strength
  - if the multiplier and modulus are properly chosen, the resulting sequence of numbers will be statistically indistinguishable from a sequence drawn at random
- Weakness
  - once an opponent knows that the linear congruential algorithm is being used and the parameters are known (e.g. a, c, m), then all numbers are known
- Opponents can determine values of X_1, X_2, X_3, then can determine the values of a, c, and m
  - $X_1 = (aX_0 + c) \bmod m$
  - $X_2 = (aX_1 + c) \bmod m$
  - $X_3 = (aX_2 + c) \bmod m$

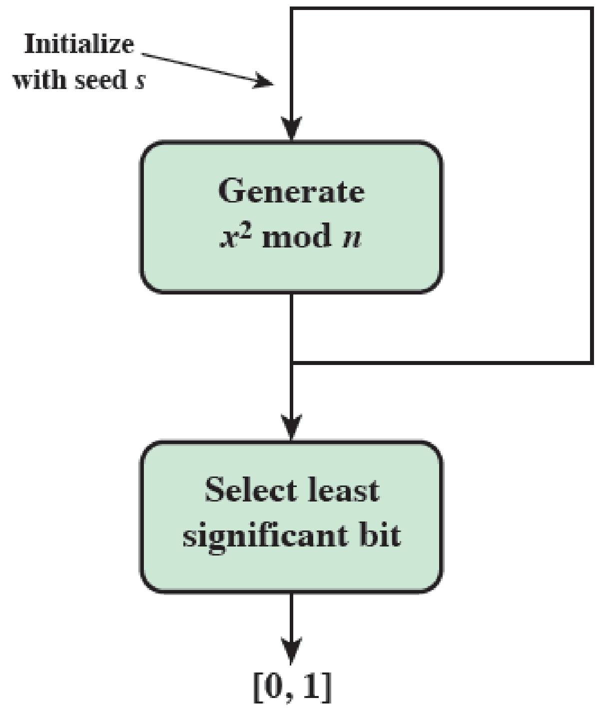
Blum Blum Shub (BBS) Generator

- 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)
  - CSPRBG: 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
    - *Given the first k bits of the sequence, there is not a algorithm can state the next bit will be 1 or 0*
- The security of BBS is based on the difficulty of factoring *n* *(i.e. determine its two prime factors p and q)*

BBS Generator

- Choose two large primes *p*, *q* where
  $p \equiv q \equiv r \bmod m$
- Let $n = p*q$, choose a random number *s*, such that *s* is relatively prime to *n*; this is equivalent to that neither *p* nor *q* is a factor of *s*

$$
\begin{aligned}
X_0 &= s^2 \bmod n \\
\textbf{for } i &= 1 \textbf{ to } \infty \\
X_i &= (X_{i-1})^2 \bmod n \\
B_i &= X_i \bmod 2
\end{aligned}
$$



- The least significant bit (B_i) is taken at each iteration

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

- n = 192649 = 383*503, and the seed s = 101355

PRNG using a Block Cipher

- For any block of plaintext, a symmetric block cipher produces an output that is apparently random
- If an established, standardized block cipher is used, such as DES or AES, then the security characteristics of the PRNG can be established

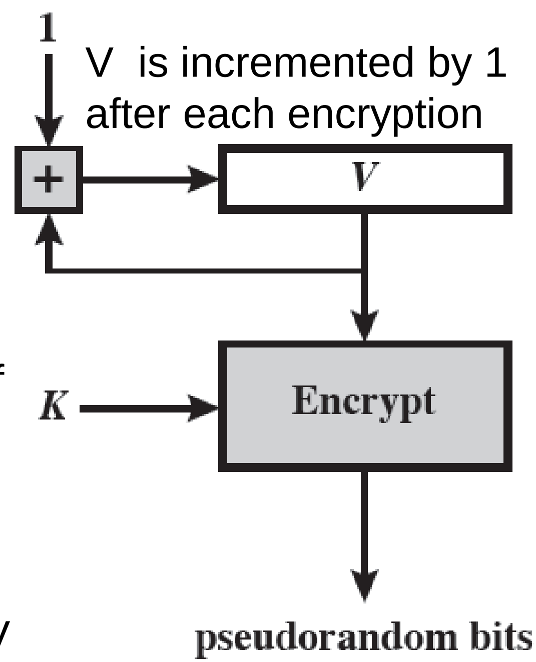
PRNG Using Block Cipher Modes of Operation

- Two approaches that use a block cipher to build a PRNG have gained widespread acceptance:
  - CTR mode
  - OFB mode

- The seed consists of two parts:
  - The encryption key value and a value V
  - These two values will be updated after each block of pseudorandom numbers is generated
    - For AES-128, the seed consists a 128-bit key and a 128-bit V

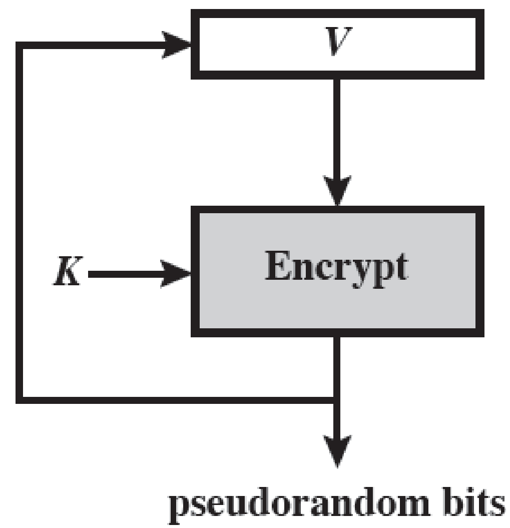
- In both cases, pseudorandom bits are produced one block at a time (e.g., for AES, PRNG bits are generated 128 bits at a time)

V is incremented by 1 after each encryption



(a) CTR Mode

V is updated to equal the value of the preceding PRNG block



(b) OFB Mode

Performance Comparison

Key:	cfb0ef3108d49cc4562d5810b0a9af60
V:	4c89af496176b728ed1e2ea8ba27f5a4

The 256 bits seed formed by: A random bit sequence of 256 bits was obtained from random.org, which use three radios tuned between stations to pick up atmospheric noise.
The total number of one bits in the seed is 124 (around 0.48)

Table 7.2 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 7.3 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

- If the fraction of bits that match between adjacent blocks differs substantially from 0.5, that suggests a correlation between blocks, which could be a security weakness.
- The results suggest no correlation.

ANSI X9.17 PRNG

- One of the cryptographically strongest PRNGs is specified in ANSI X9.17
  - A number of applications employ this technique including financial security applications and PGP

Input (DT_i, V_i)

- Two pseudorandom inputs:
  - A 64-bit representation of the **current date and time**
  - A 64-bit **seed** value; this is initialized to some arbitrary value and is **updated** during the generation process.

The algorithm makes use of triple DES for encryption. Ingredients are:

Keys (K_1, K_2)

- Use **three 3DES** encryption modules.
- All three make use of the **same pair of 56-bit keys**
  - The keys must be kept secret and are used only for pseudorandom number generation.

Output (R_i, V_{i+1})

- A 64-bit **pseudorandom number** and a 64-bit **seed** value.

Cont’d

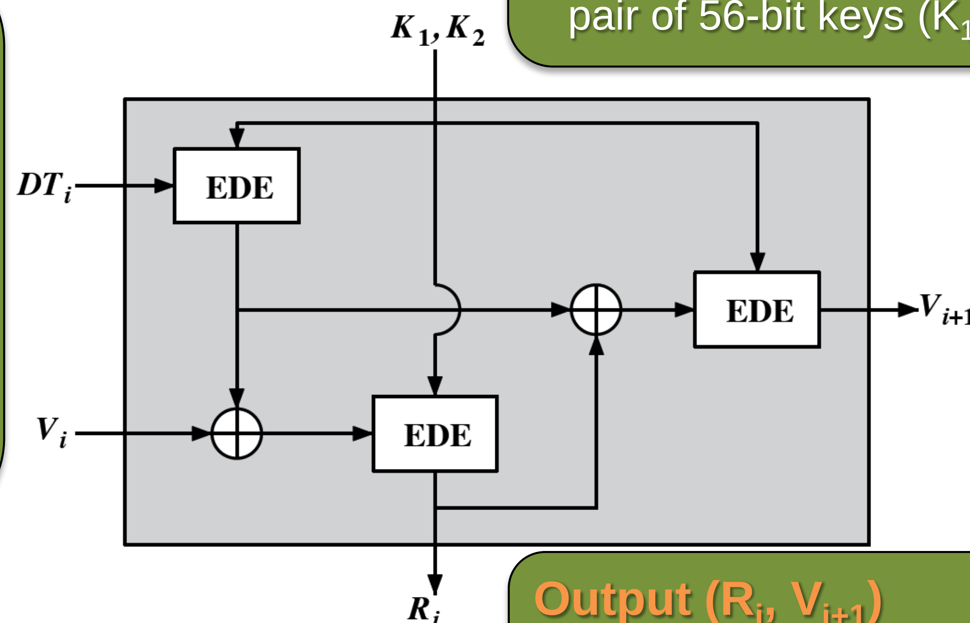
- EDE([K_1, K_2], X) refers to the sequence encrypt-decrypt-encrypt using two-key triple DES to encrypt X

Keys (K_1, K_2)

- Use of three 3DES encryption.
- All three make use of the same pair of 56-bit keys (K_1, K_2)

Input (DT_i, V_i)

- Two pseudorandom inputs drive the generator
- A 64-bit representation of the current date and time (DT_i)
- A 64-bit seed value(V_i); this is initialized to some arbitrary value and is updated during the generation process.



$$R_i = \mathrm{EDE}([K_1, K_2], [V_i \oplus \mathrm{EDE}([K_1, K_2], DT_i)])$$
$$V_{i+1} = \mathrm{EDE}([K_1, K_2], [R_i \oplus \mathrm{EDE}([K_1, K_2], DT_i)])$$

Output (R_i, V_{i+1})

- A 64-bit pseudorandom number
- A 64-bit seed value.

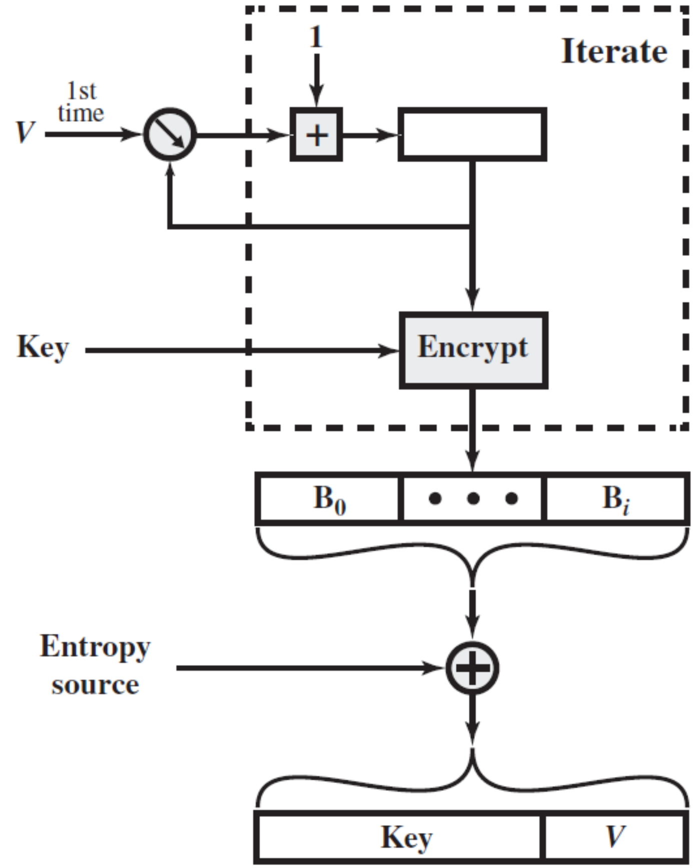
Cont’d

- Cryptographic strength:
  - 112 bits key
  - Three EDE encryptions (total of 9 DES encryptions)
  - Two inputs (date and time, a seed from other generator)

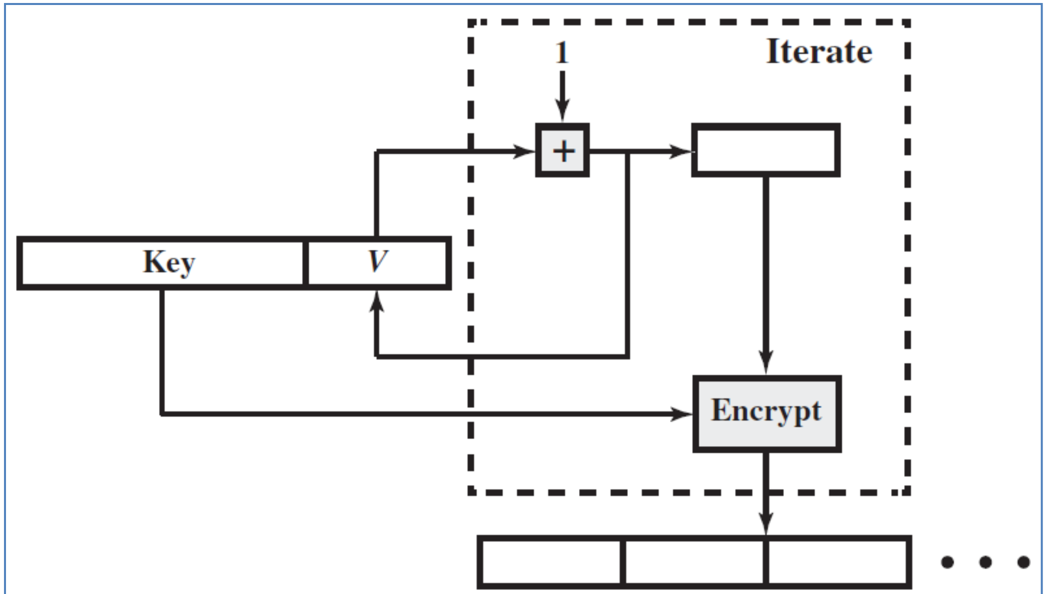
NIST CTR_DRBG

- The CTR algorithm for PRNG, called CTR_DRBG (counter mode-deterministic random bit generator)
  - Is the PRNG defined in NIST SP 800-90 based on the CTR mode of operation
  - Is part of the hardware random number generator implemented on all recent *Intel processor chips*
  - Entropy source: to provide random bits
    - E.g. from TRNG
    - Entropy is an information theoretic concept that measures unpredictability or randomness
  - The encryption algorithm:
    - E.g. 3DES with three keys or AES with a key size of 128, 192, or 256 bits

CTR_DRBG Functions



(a) Initialize and update function



(b) Generate function

Cont'd

- Initial function
  - Seed: the combination of *K* and *V*. The initial values are chosen arbitrarily
  - Produce at least *seedlen* bits
  - *V*: incremeted by 1 after each encryption
- Generate function
  - Each iteration uses the same encryption key
  - The counter value V is incremented by 1 for each iteration
- Update function
  - To enhance security, the number of bits generated by any PRNG should be limited
  - When reseed counter (incremented with each run) reaches Reseed-Interval, invoke update function
  - Both key and V values be updated for the generate function

CTR_DRBG Parameters

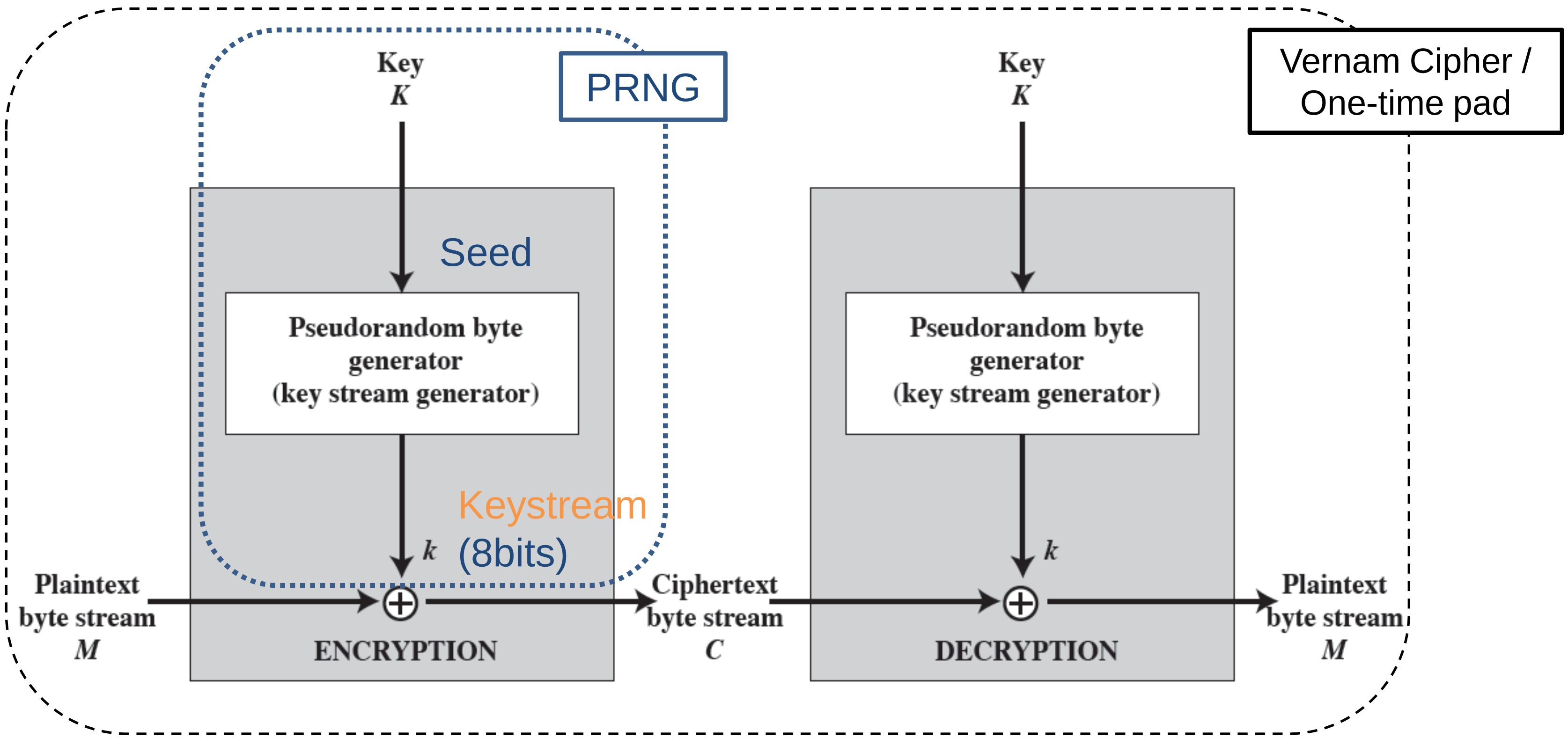
- The E algorithm used in the DRBG may be 3DES with 3 keys or AES-128/192/256
  - Output block length (*outlen*)
  - Key length (*keylen*)
  - Seed length (*seedlen*)
  - Reseed interval (*reseed_interval*)

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

Stream Ciphers

Stream Ciphers

- Typical stream cipher encryption: one byte at a time
  Alternatives: one bit at a time or on units larger than a byte at a time



Encryption

```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

Decryption

```
  10100000  ciphertext
⊕ 01101100  key stream
  11001100  plaintext
```

Stream Cipher Design Considerations [KUMA97]

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

Cont'd

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

Cont'd

- One advantage of a block cipher is that you can reuse keys
  - In stream cipher, the cryptanalysis will be quite simple
- A stream cipher can be constructed with any cryptographically strong PRNG

RC4

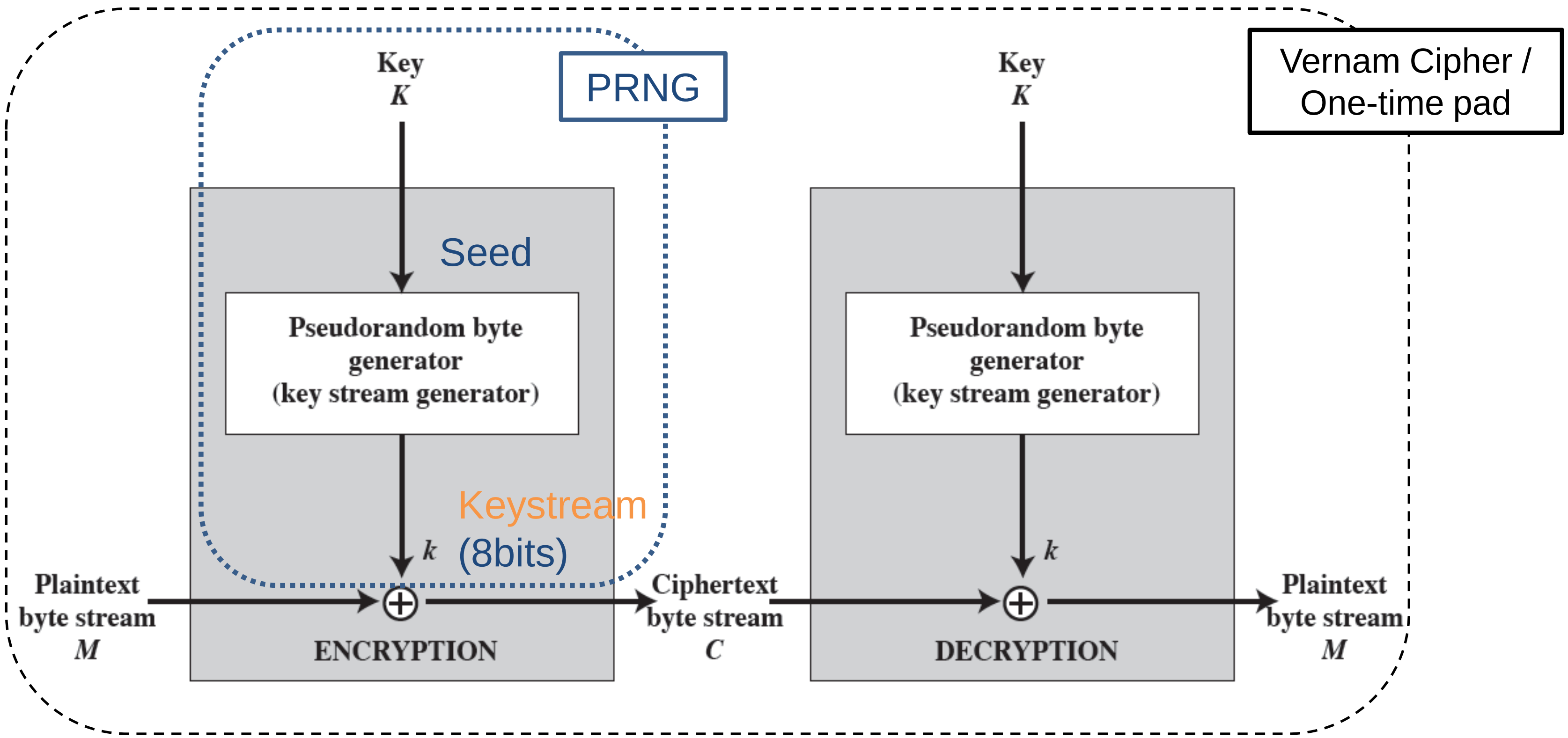
- Designed in 1987 by Ron Rivest for RSA Security
- Variable key size stream cipher with byte-oriented operations
  - A variable-length key of from 1 to 256 bytes is used to initialize a 256-byte state vector *S*, with elements S[0], S[1], …, S[255]. *S* contains a permutation of all 8-bit numbers from 0 through 255.
  - The period of the cipher is greater than 10^{100}
  - 8-16 machine operations are required per output byte
- Used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards (for communication between Web browsers and servers)
- Used in the Wired Equivalent Privacy (WEP) protocol and the newer WiFi Protected Access (WPA) protocol (for the IEEE 802.11 wireless LAN standard)

Cont’d

- A variable-length key (from 1 to 256 bytes, i.e. 8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0], S[1], …, 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 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.

Recall: Stream Ciphers

- Typical stream cipher encryption: one byte at a time
  Alternatives: one bit at a time or on units larger than a byte at a time



Encryption

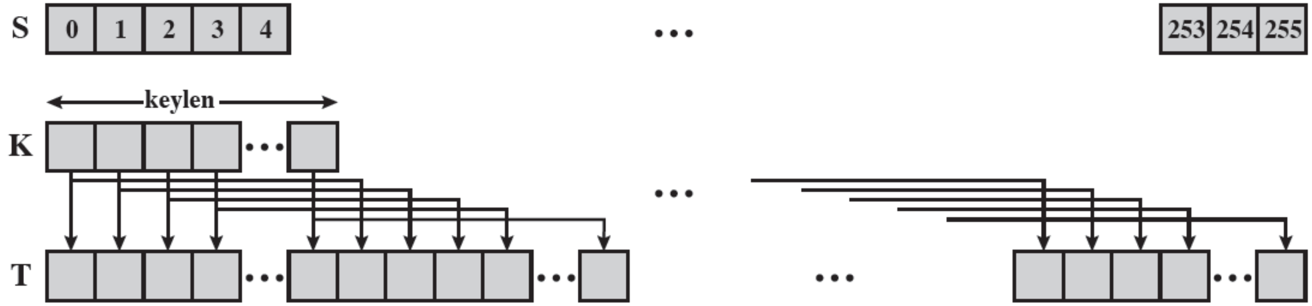
```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

Decryption

```
  10100000  ciphertext
⊕ 01101100  key stream
  11001100  plaintext
```

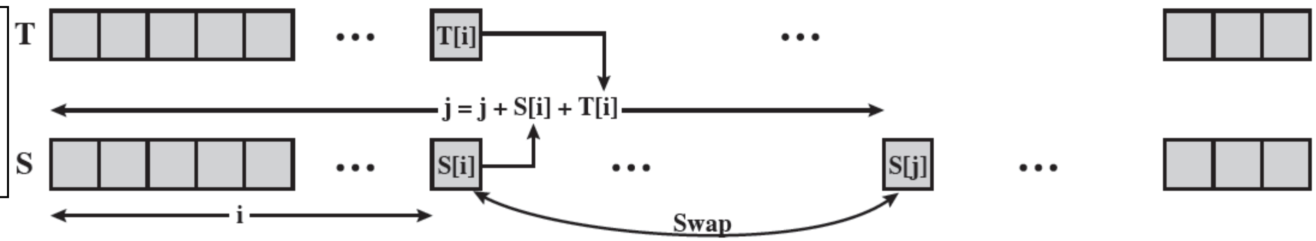
Cont'd

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



(a) Initial state of S and T

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

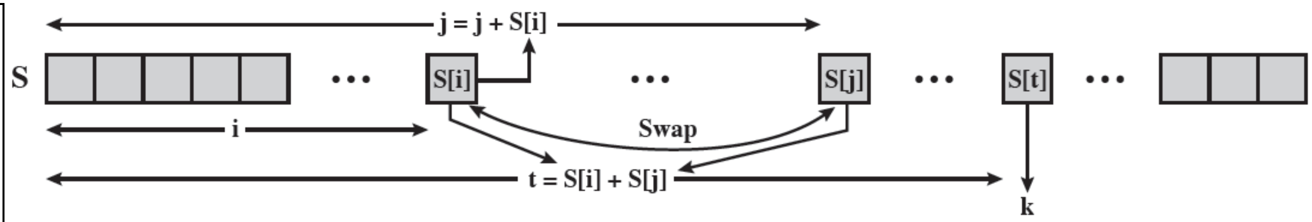


(b) Initial permutation of S

- Initialization of S
  - The entire S are set equal to the values from 0~255 in ascending order; i.e. S[0]=0, …S[255]=255.
  - The *keylen* byte elements are copied from K to T (temporary vectors)
- Initial Permutation of S
  - For each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]

Cont'd

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



To encrypt, XOR the value *k* with the next byte of plaintext.

To decrypt, XOR the value *k* with the next byte of ciphertext.

- Stream Generation
  - Once the S vector is initialized, the input key is no longer used
  - Cycling through all the elements of S[i], for each S[i], swapping S[i] with another byte in S according to a scheme dictated by the current configuration of S

- Strength
  - None of current approaches is practical against RC4 with a reasonable key length, such as 128 bits
  - But one application using RC4, WEP protocal for 802.11 wireless LAN network, is vulnerable to a particular attack approach

True Random Number Generator (TRNG)

- Entropy Sources
  - TRNG uses a nondeterministic source to produce randomness
    - Most operate by measuring unpredictable natural process, e,g, pulse dectors of ionizing radiation events, etc.
      - 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

Cont’d

- Operating systems typically provide a built-in mechanism for generating random numbers
  - Linux uses 4 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

Comparison of PRNGs and TRNGs

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

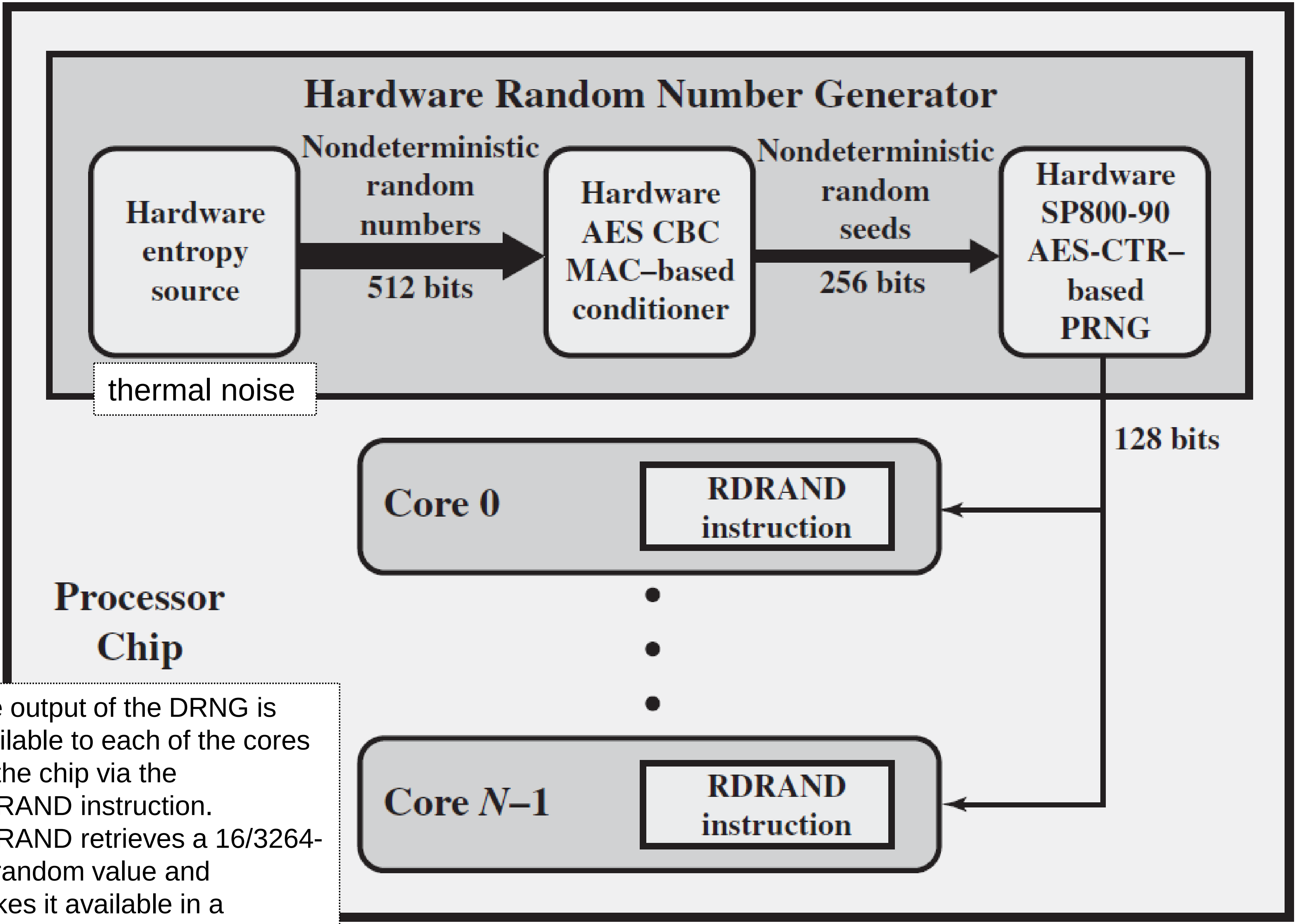
- Efficiency
  - produce many numbers in a short time
- Determinism
  - a given sequence of numbers can be reproduced at a later date if the starting point in the sequence is known
- Periodicity
  - the sequence will eventually repeat itself

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
    - To pass the bit stream through a hash function such as MD5 or SHA-1 (following classes)
      - 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

Intel Digital Random Number Generator (DRNG)

- TRNGs traditionally been used for
  - key generation
  - small number of random bits were required
    - Because TRNGs have been **inefficient** with a low bit rate of random bit production
- The first commercially available TRNG
  - Intel digital random number generator (DRNG) (2012)
  - Achieves bit production rates comparable with that of PRNGs
  - It is implemented entirely in hardware (higher computation speed compare with software)
  - The entire DRNG is on the same multicore chip as the processors (eliminates the I/O delay)



The output of the DRNG is available to each of the cores on the chip via the RDRAND instruction. RDRAND retrieves a 16/3264-bit random value and makes it available in a software-accessible register

- 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

- CBC-MAC or CMAC: encrypts its input using the cipher block chaining (CBC) mode The output of this stage is generated 256 bits at a time and is intended to exhibit true randomness with no skew or bias.

- 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.
- The algorithm used for this stage is CTR_DRBG