Information Security
CSD-410

Computer Science and Engineering Department
National Institute of Technology

Instructor: **Dr. Lokesh Chouhan**

Slide Sources:
Cryptography and Network Security
by
William Stallings
adapted and supplemented

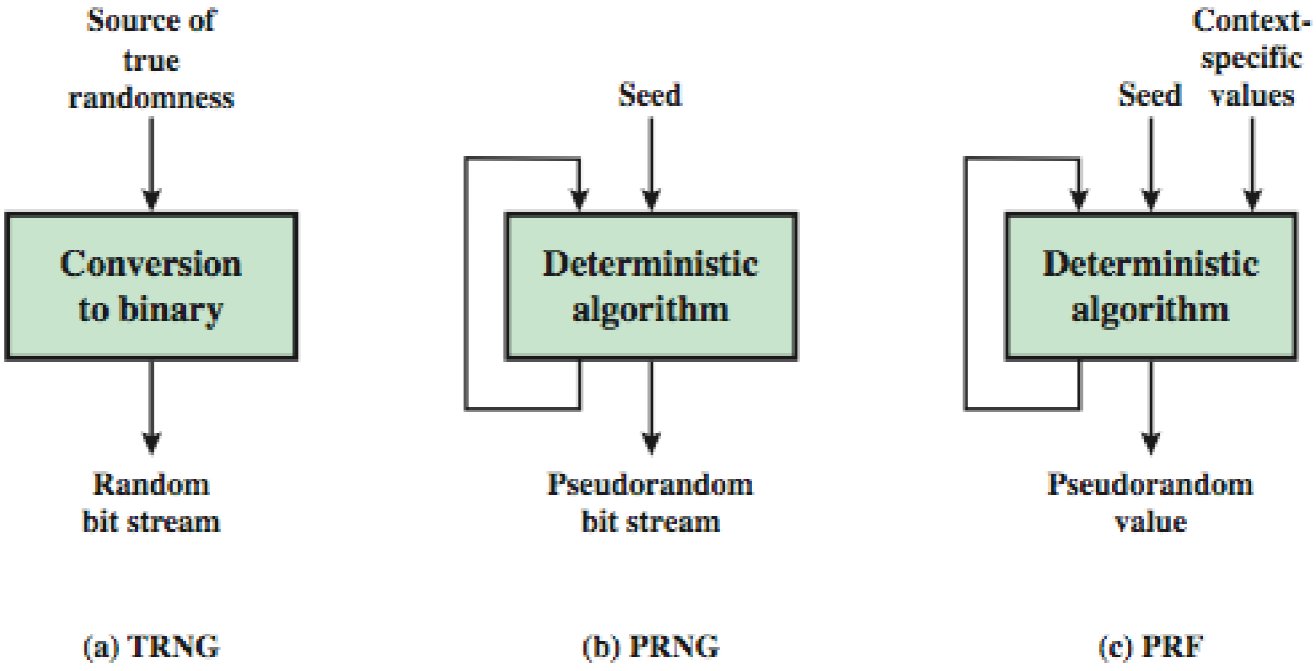
Random Numbers

- many uses of **random numbers** in cryptography
  - nonces in authentication protocols to prevent replay
  - session keys
  - public key generation
  - keystream for a one-time pad
- in all cases its critical that these values be
  - statistically random, uniform distribution, independent
  - unpredictability of future values from previous values
- true random numbers provide this
- care needed with generated random numbers

Pseudorandom Number Generators (PRNGs)

- often use deterministic algorithmic techniques to create “random numbers”
  - although are not truly random
  - can pass many tests of “randomness”
- known as “pseudorandom numbers”
- created by “Pseudorandom Number Generators (PRNGs)”

Random & Pseudorandom Number Generators



PRNG Requirements

- randomness
  - uniformity, scalability, consistency
- unpredictability
  - forward & backward unpredictability
  - use same tests to check
- characteristics of the seed
  - secure
  - if known adversary can determine output
  - so must be random or pseudorandom number

Linear Congruential Generator

- common iterative technique using:

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

- given suitable values of parameters can produce a long random-like sequence
- suitable criteria to have are:
  - function generates a full-period
  - generated sequence should appear random
  - efficient implementation with 32-bit arithmetic
- note that an attacker can reconstruct sequence given a small number of values
- have possibilities for making this harder

Blum Blum Shub Generator

- based on public key algorithms
- use least significant bit from iterative equation:
  - $x_i = x_{i-1}^2 \bmod n$
  - where $n=p.q$, and primes $p,q=3 \bmod 4$
- unpredictable, passes **next-bit** test
- security rests on difficulty of factoring N
- is unpredictable given any run of bits
- slow, since very large numbers must be used
- too slow for cipher use, good for key generation

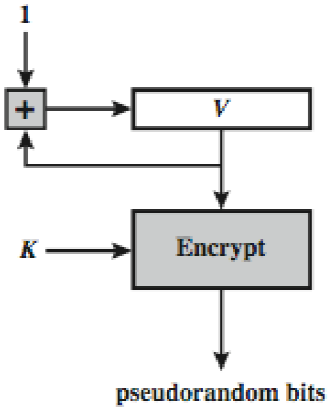
Using Block Ciphers as PRNGs

- for cryptographic applications, can use a block cipher to generate random numbers
- often for creating session keys from master key
- CTR

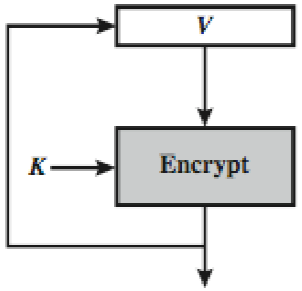
$$X_i = \mathrm{E}_K[\mathrm{V}_i]$$

- OFB

$$X_i = \mathrm{E}_K[X_{i-1}]$$

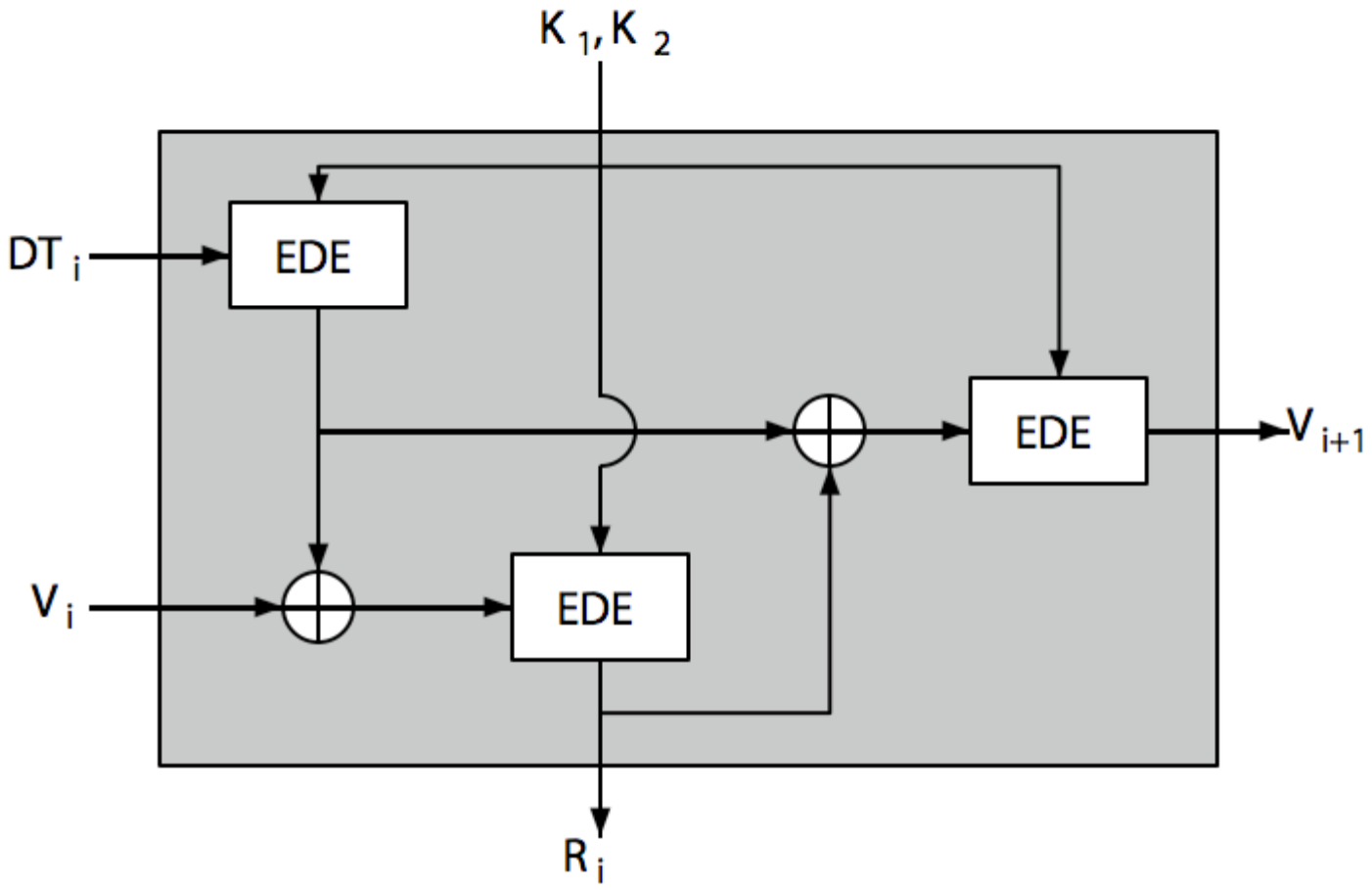


(a) CTR Mode



(b) OFB Mode

ANSI X9.17 PRG



Blowfish

- a symmetric block cipher designed by Bruce Schneier in 1993/94
- characteristics
  - fast implementation on 32-bit CPUs
  - compact in use of memory
  - simple structure eases analysis/implemention
  - variable security by varying key size
- has been implemented in various products

Blowfish Key Schedule

- uses a 32 to 448 bit key
- used to generate
  - 18 32-bit subkeys stored in K-array K_j
  - four 8x32 S-boxes stored in $S_{i,j}$
- key schedule consists of:
  - initialize P-array and then 4 S-boxes using pi
  - XOR P-array with key bits (reuse as needed)
  - loop repeatedly encrypting data using current P & S and replace successive pairs of P then S values
  - requires 521 encryptions, hence slow in rekeying

Blowfish Encryption

- uses two primitives: addition & XOR
- data is divided into two 32-bit halves L_0 & R_0

for i = 1 to 16 do

$R_i = L_{i-1}$ XOR P_i;

L_i = F[R_i] XOR R_{i-1};

$L_{17} = R_{16}$ XOR P_{18};

$R_{17} = L_{16}$ XOR i_{17};

- where

F[a,b,c,d] = (($S_{1,a}$ + $S_{2,b}$) XOR $S_{3,c}$) + $S_{4,a}$

Discussion

- key dependent S-boxes and subkeys, generated using cipher itself, makes analysis very difficult
- changing both halves in each round increases security
- provided key is large enough, brute-force key search is not practical, especially given the high key schedule cost

RC5

- a proprietary cipher owned by RSADSI
- designed by Ronald Rivest (of RSA fame)
- used in various RSADSI products
- can vary key size / data size / no rounds
- very clean and simple design
- easy implementation on various CPUs
- yet still regarded as secure

RC5 Ciphers

- RC5 is a family of ciphers RC5-w/r/b
  - w = word size in bits (16/32/64) nb data=2w
  - r = number of rounds (0..255)
  - b = number of bytes in key (0..255)
- nominal version is RC5-32/12/16
  - ie 32-bit words so encrypts 64-bit data blocks
  - using 12 rounds
  - with 16 bytes (128-bit) secret key

RC5 Key Expansion

- RC5 uses 2r+2 subkey words (w-bits)
- subkeys are stored in array `S[i]`, i=0..t-1
- then the key schedule consists of
  - initializing S to a fixed pseudorandom value, based on constants e and phi
  - the byte key is copied (little-endian) into a c-word array L
  - a mixing operation then combines L and S to form the final S array

RC5 Encryption

- split input into two halves A & B

  $L_0 = A + S[0];$

  $R_0 = B + S[1];$

  for i = 1 to r do

  $L_i = ((L_{i-1} \text{ XOR } R_{i-1}) <<< R_{i-1}) + S[2 \times i];$

  $R_i = ((R_{i-1} \text{ XOR } L_i) <<< L_i) + S[2 \times i + 1];$

- each round is like 2 DES rounds
- note rotation is main source of non-linearity
- need reasonable number of rounds (eg 12-16)

RC5 Modes

- RFC2040 defines 4 modes used by RC5
  - RC5 Block Cipher, is ECB mode
  - RC5-CBC, is CBC mode
  - RC5-CBC-PAD, is CBC with padding by bytes with value being the number of padding bytes
  - RC5-CTS, a variant of CBC which is the same size as the original message, uses ciphertext stealing to keep size same as original

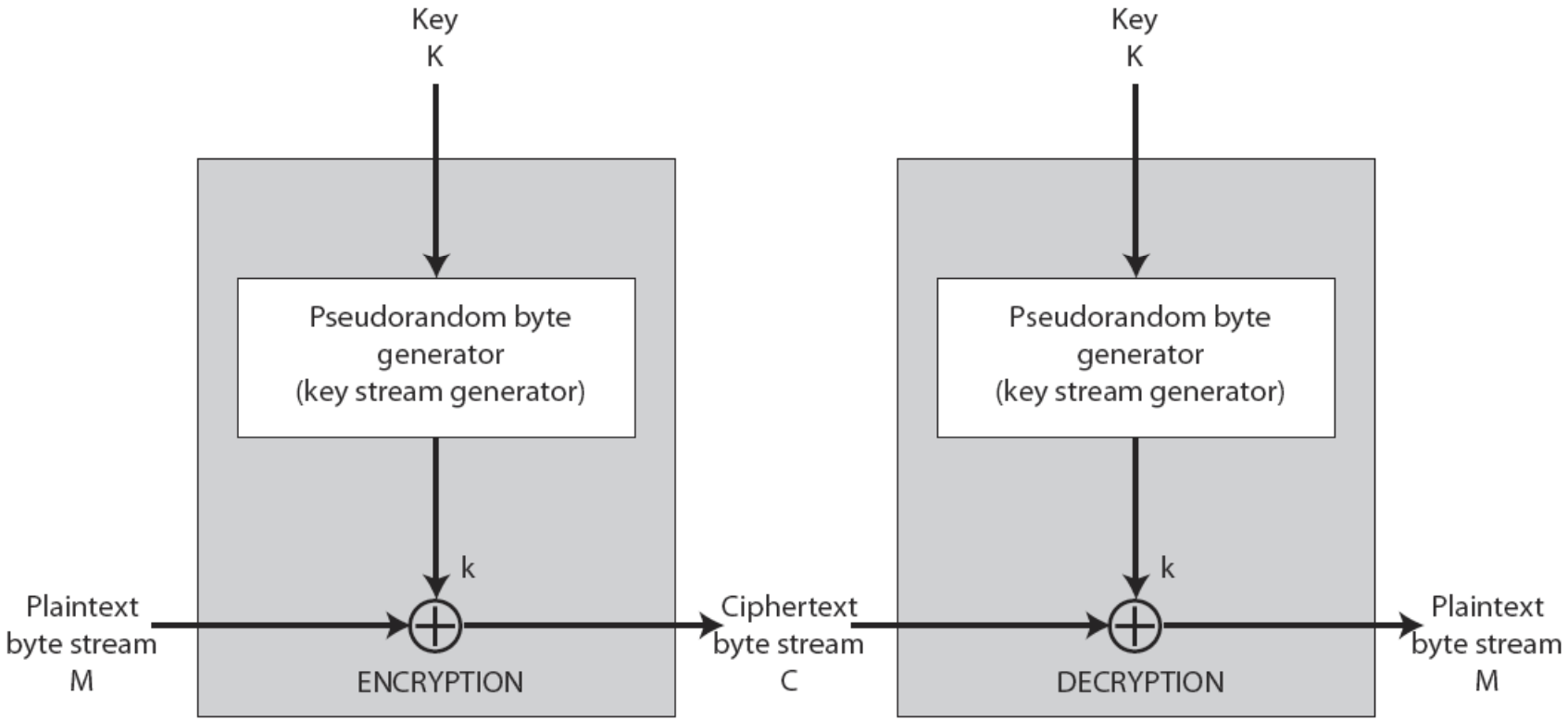
Block Cipher Characteristics

- features seen in modern block ciphers are:
  - variable key length / block size / no rounds
  - mixed operators, data/key dependent rotation
  - key dependent S-boxes
  - more complex key scheduling
  - operation of full data in each round
  - varying non-linear functions

Stream Ciphers

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

Stream Cipher Structure



Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher
- given a key k of length l bytes

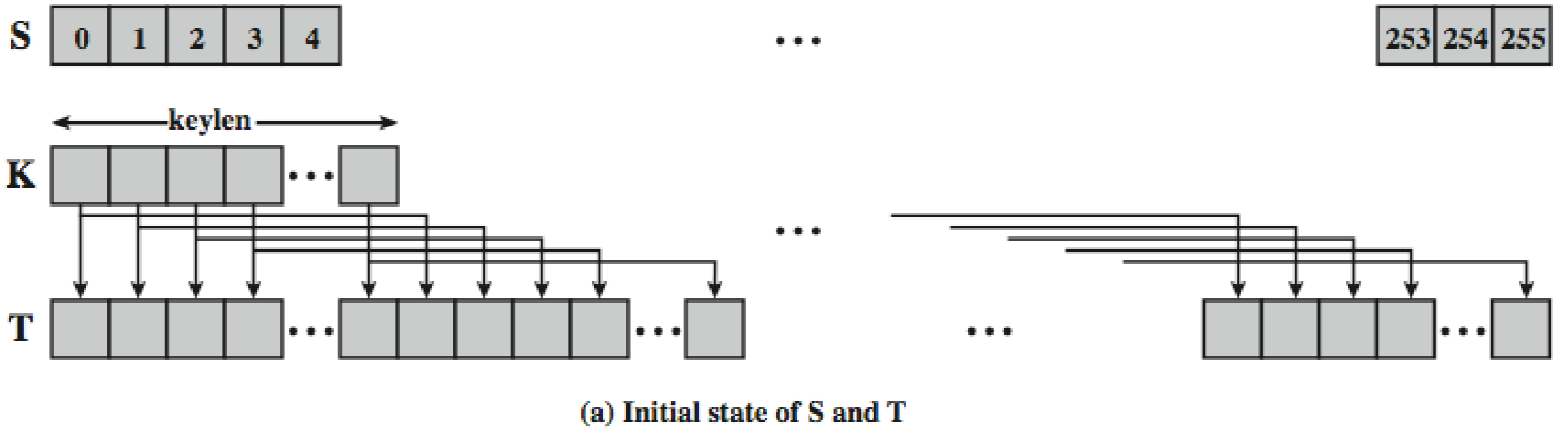
```
for i = 0 to 255 do
   S[i] = i
j = 0
for i = 0 to 255 do
   j = (j + S[i] + k[i mod l]) (mod 256)
   swap (S[i], S[j])
```

RC4 Encryption

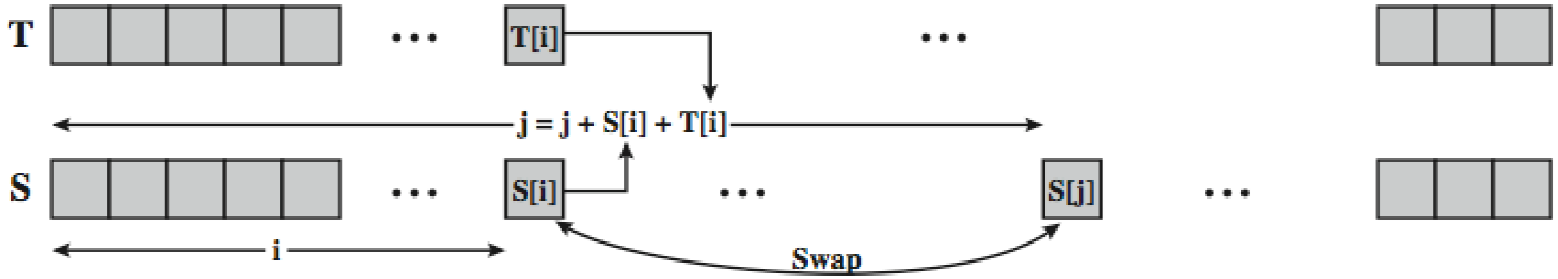
- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value
- tXOR with next byte of message to en/decrypt

```
i = j = 0
for each message byte M_i
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    C_i = M_i XOR S[t]
```

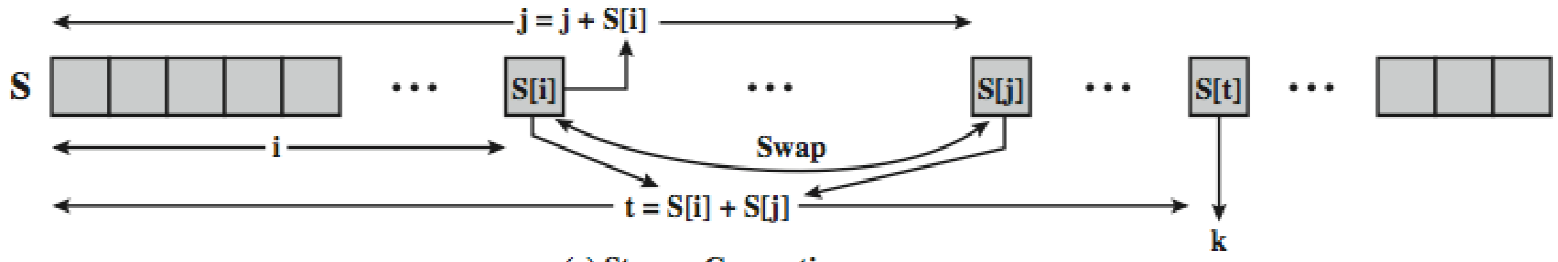
RC4 Overview



(a) Initial state of S and T



(b) Initial permutation of S



(c) Stream Generation

RC4 Security

- claimed secure against known attacks
  - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**
- have a concern with WEP, but due to key handling rather than RC4 itself

Summary

- have considered:
  - some other modern symmetric block ciphers
  - Triple-DES
  - Blowfish
  - RC5
  - briefly introduced stream ciphers
  - RC4