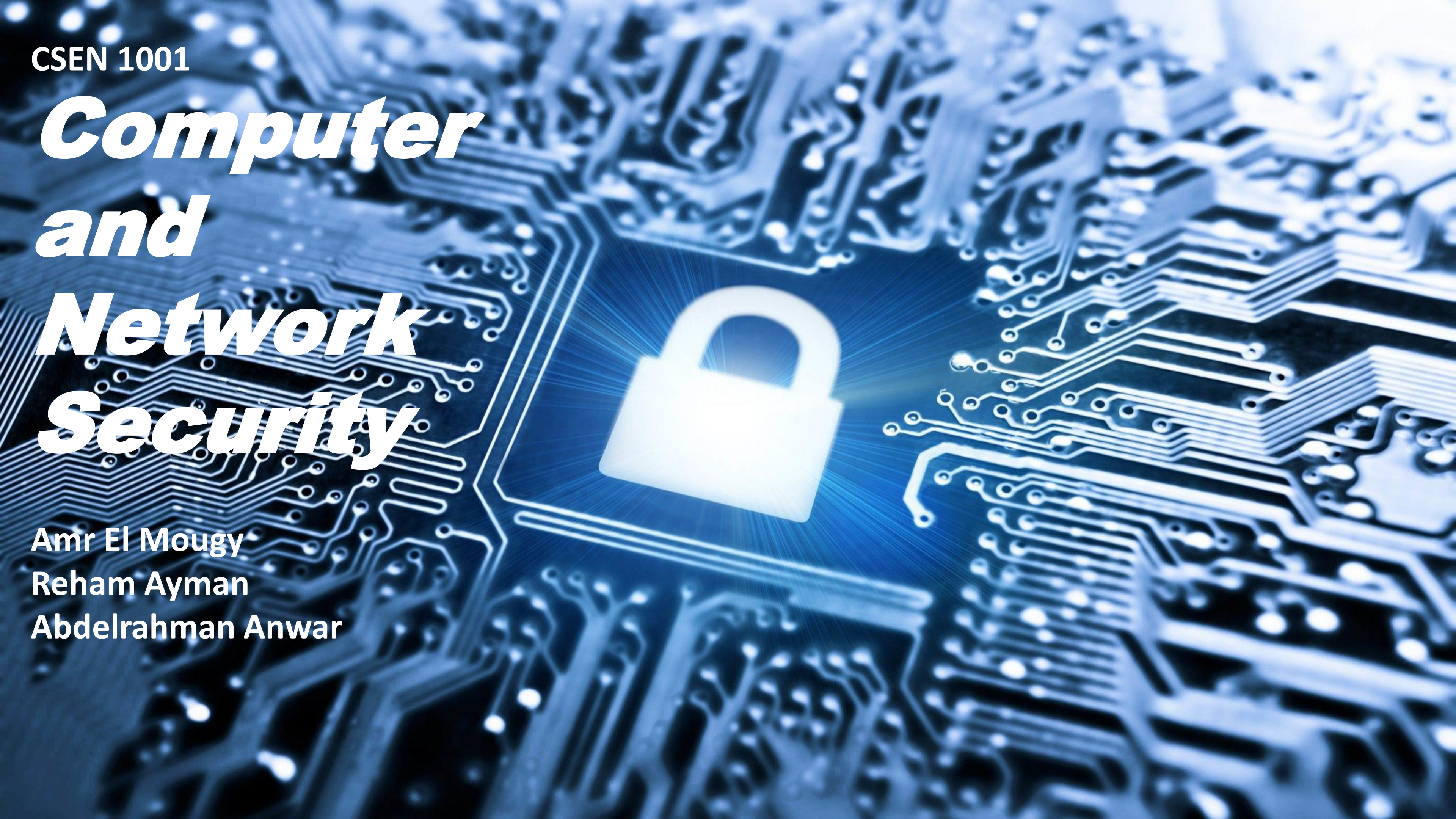


CSEN 1001

Computer and Network Security

Amr El Mougy
Reham Ayman
Abdelrahman Anwar

Lecture (4)

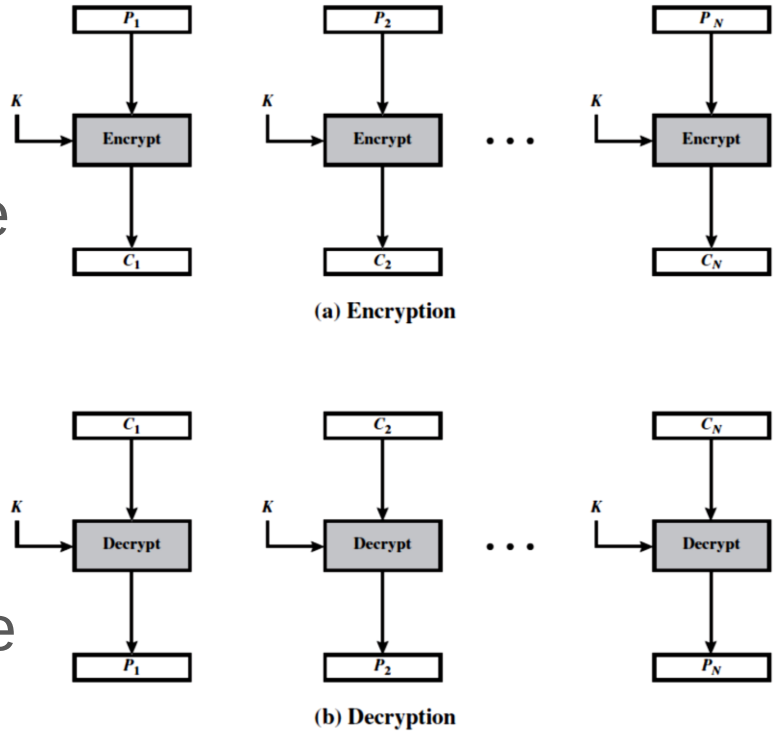
Symmetric Ciphers – cont'd

Modes of Operation

- Block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks with 56-bit key
- Need some way to en/decrypt arbitrary amounts of data in practise
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
- Subsequently 5 defined for AES & DES
- Have **block** and **stream** modes

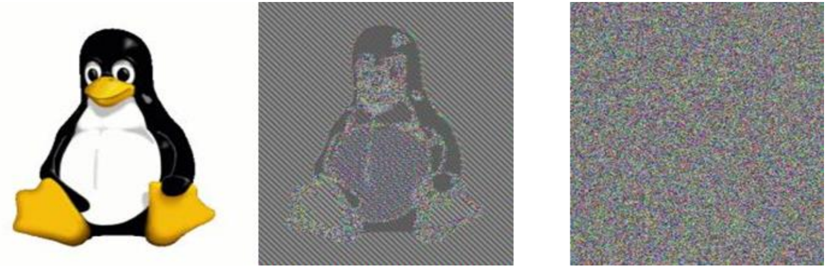
Electronic Codebook Mode (ECB)

- Message is broken into independent blocks which are encrypted
- Each block is a value which is substituted, like a codebook, hence name
- Each block is encoded independently of the other blocks
  - $C_i = E_K(P_i)$
- Uses: secure transmission of single values



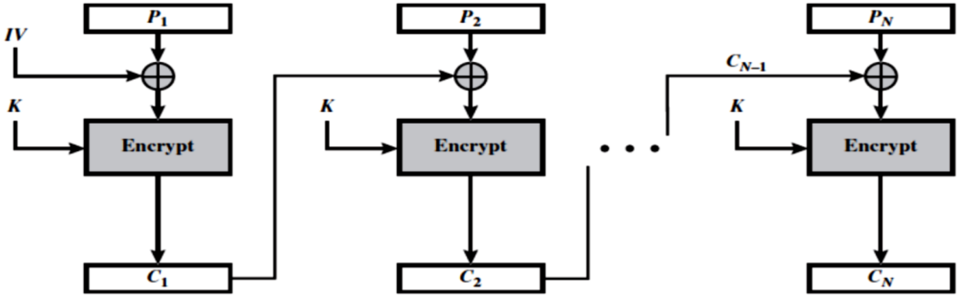
Limitations of ECB

- Message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- Weakness is due to the encrypted message blocks being independent
- Main use is sending a few blocks of data

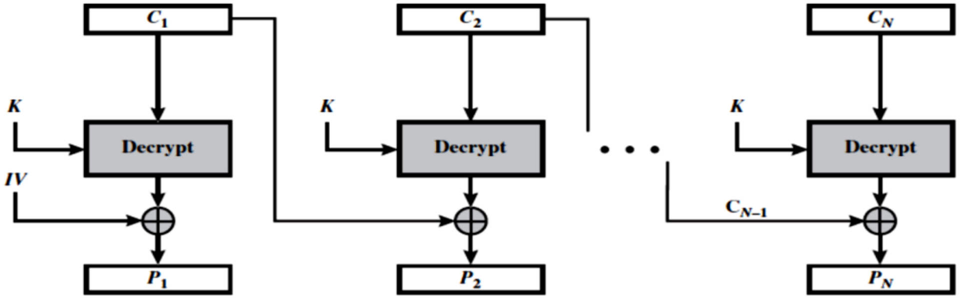


Cipher Block Chaining Mode (CBC)

- Message is broken into blocks
- Linked together in encryption operation
- Each previous cipher blocks is chained with current plaintext block, hence name
- Use Initial Vector (IV) to start process
  - $C_i = E_K(P_i \text{ XOR } C_{i-1})$
  - $C_0 = IV$
- Uses: bulk data encryption



(a) Encryption



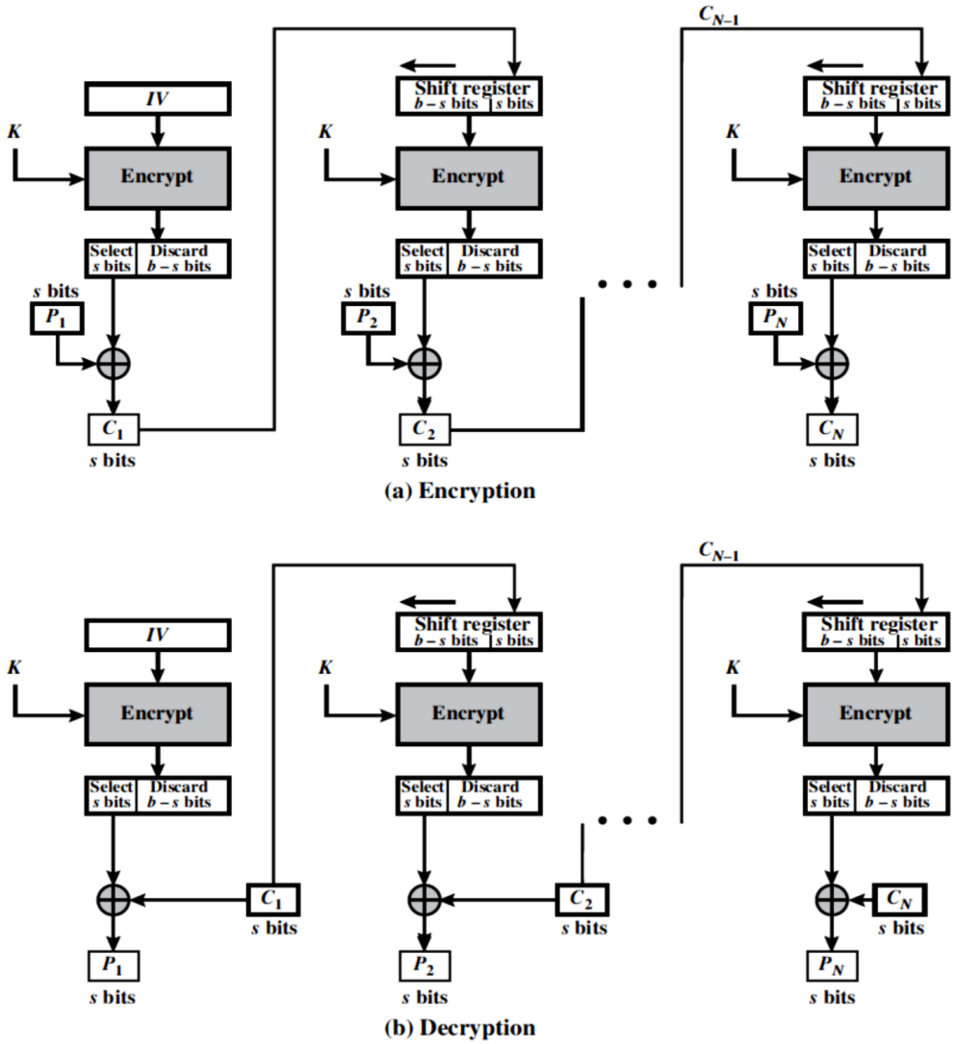
(b) Decryption

Limitations of CBC

- A ciphertext block depends on **all** blocks before it
- Any change to a block affects all following ciphertext blocks
- Need **Initialization Vector** (IV)
  - which must be known to sender & receiver
  - if predictable, attacker can change bits of first block, and change IV to compensate
    - $C1 = E(K, [IV \oplus P1])$
    - $P1 = IV \oplus D(K,C1)$
    - $P1[i] = IV[i] \oplus D(K, C1)[i]$
    - $P1[i]' = IV[i]' \oplus D(K,C1)[i]$
  - hence IV must an unpredictable value
  - can be sent encrypted in ECB mode before rest of message

Cipher Feedback Mode (CFB)

- Message is treated as a stream of bits
- Added to the output of the block cipher
- Result is feed back for next stage (hence name)
- Standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
  - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- Most efficient to use all bits in block (64 or 128)
  - $C_i = P_i \text{ XOR } E_K(C_{i-1})$
  - $C_0 = IV$
- Uses: stream data encryption

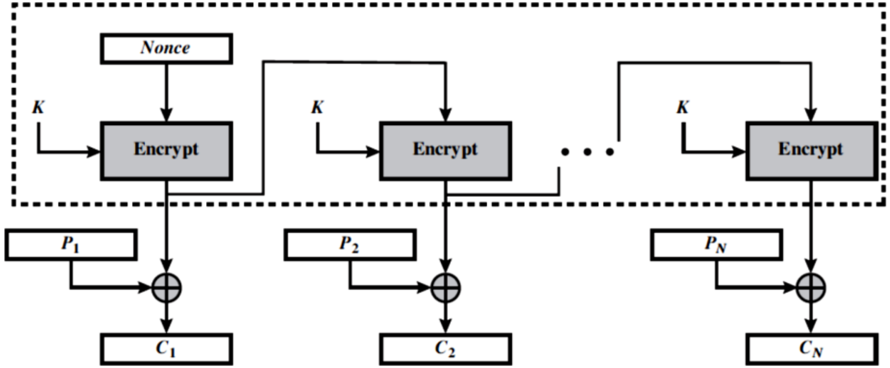


Limitations of CFB

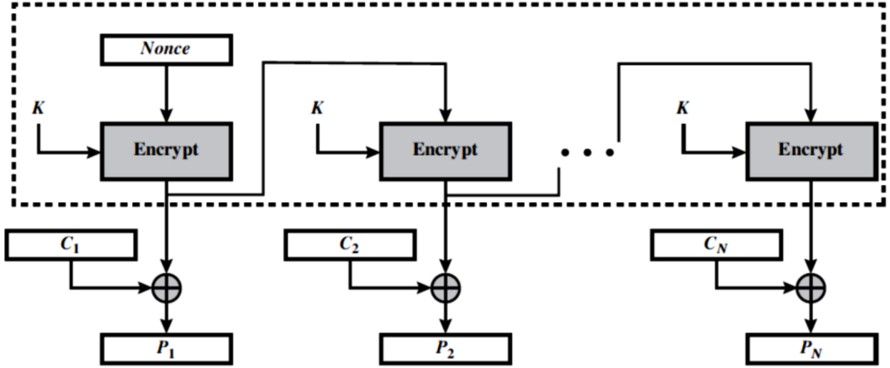
- Appropriate when data arrives in bits/bytes
- Most common stream mode
- Limitation is need to stall while doing block encryption after every n-bits
- Note that the block cipher is used in **encryption** mode at **both** ends
- Errors propagate for several blocks after the error

Output Feedback Mode (OFB)

- Message is treated as a stream of bits
- Output of cipher is added to message
- Output is then feed back (hence name)
- Feedback is independent of message
- Can be computed in advance
  - $C_i = P_i \text{ XOR } O_i$
  - $O_i = E_K(O_{i-1}) \quad i>1$
  - $O_1 = E(\text{Nonce})$
- Uses: stream encryption on noisy channels



(a) Encryption



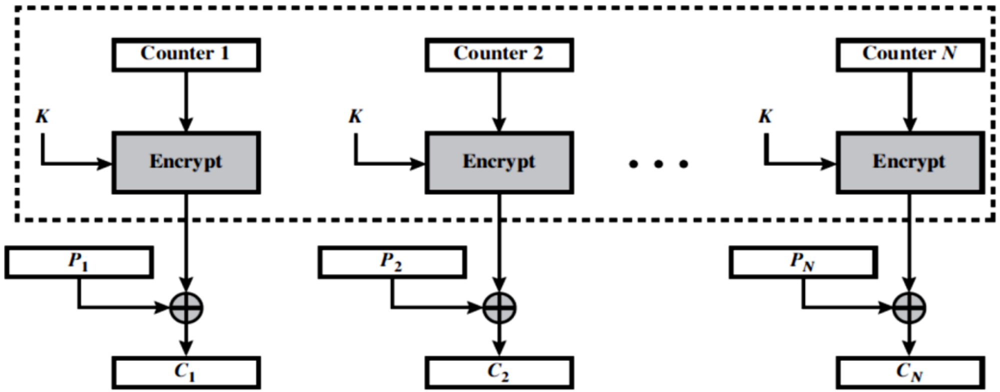
(b) Decryption

Limitations of OFB

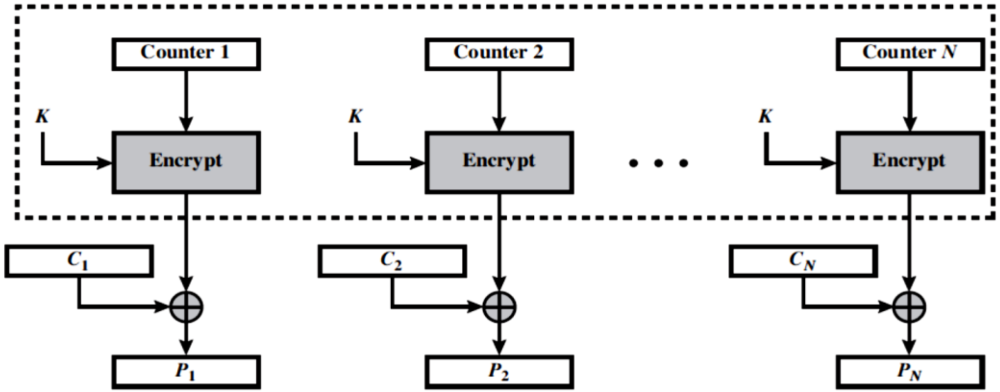
- Bit errors do not propagate
- More vulnerable to message stream modification
- A variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- Sender & receiver must remain in sync
- Originally specified with m-bit feedback
- Subsequent research has shown that only **full block feedback** (i.e. CFB-64 or CFB-128) should ever be used

Counter Mode (CTR)

- Relatively "new" mode, though proposed early on
- Similar to OFB but encrypts counter value rather than any feedback value
- Must have a different key & counter value for every plaintext block (never reused)
  - $C_i = P_i \text{ XOR } O_i$
  - $O_i = E_K(i)$
- Uses: high-speed network encryptions



(a) Encryption



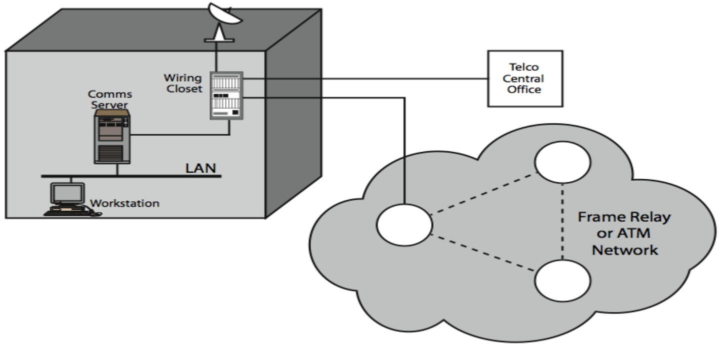
(b) Decryption

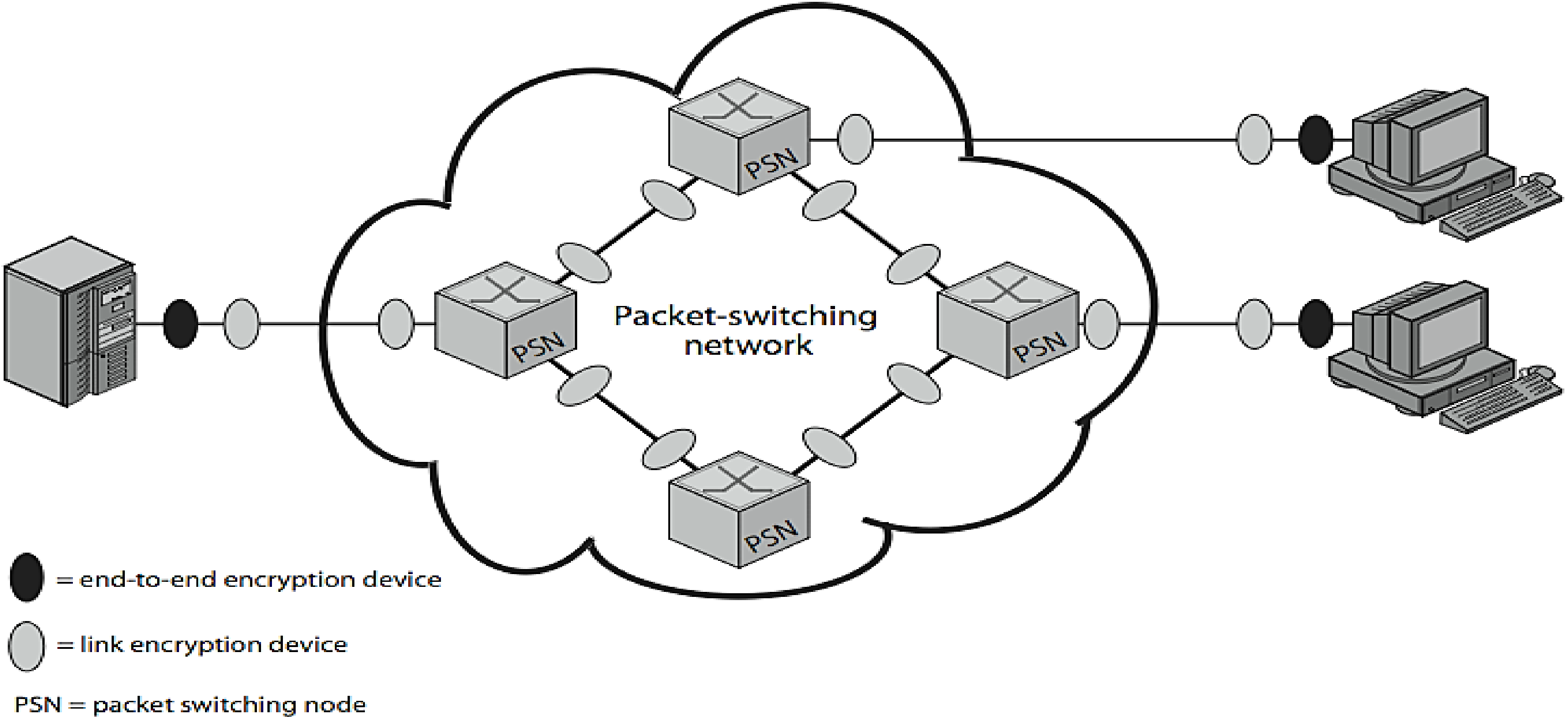
Advantages of CTR

- Efficiency
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need
  - good for bursty high speed links
- Random access to encrypted data blocks
- Provable security (good as other modes)
- But must ensure never reuse key/counter values, otherwise could break (cf OFB)

Confidentiality using Symmetric Encryption

- Have two major placement alternatives
- **Link encryption**
  - encryption occurs independently on every link
  - implies must decrypt traffic between links
  - requires many devices, but paired keys
- **End-to-end encryption**
  - encryption occurs between original source and final destination
  - need devices at each end with shared keys



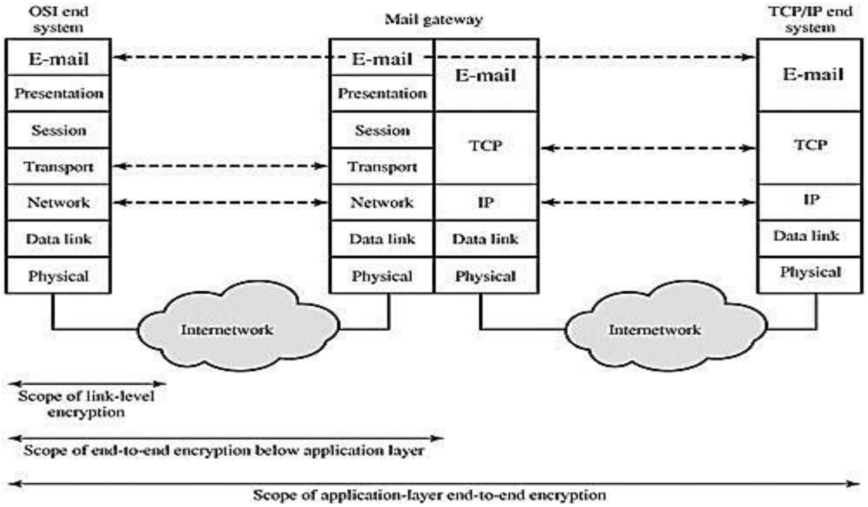

Packet-switching network
PSN
PSN
PSN
PSN
PSN
= end-to-end encryption device
= link encryption device
PSN = packet switching node

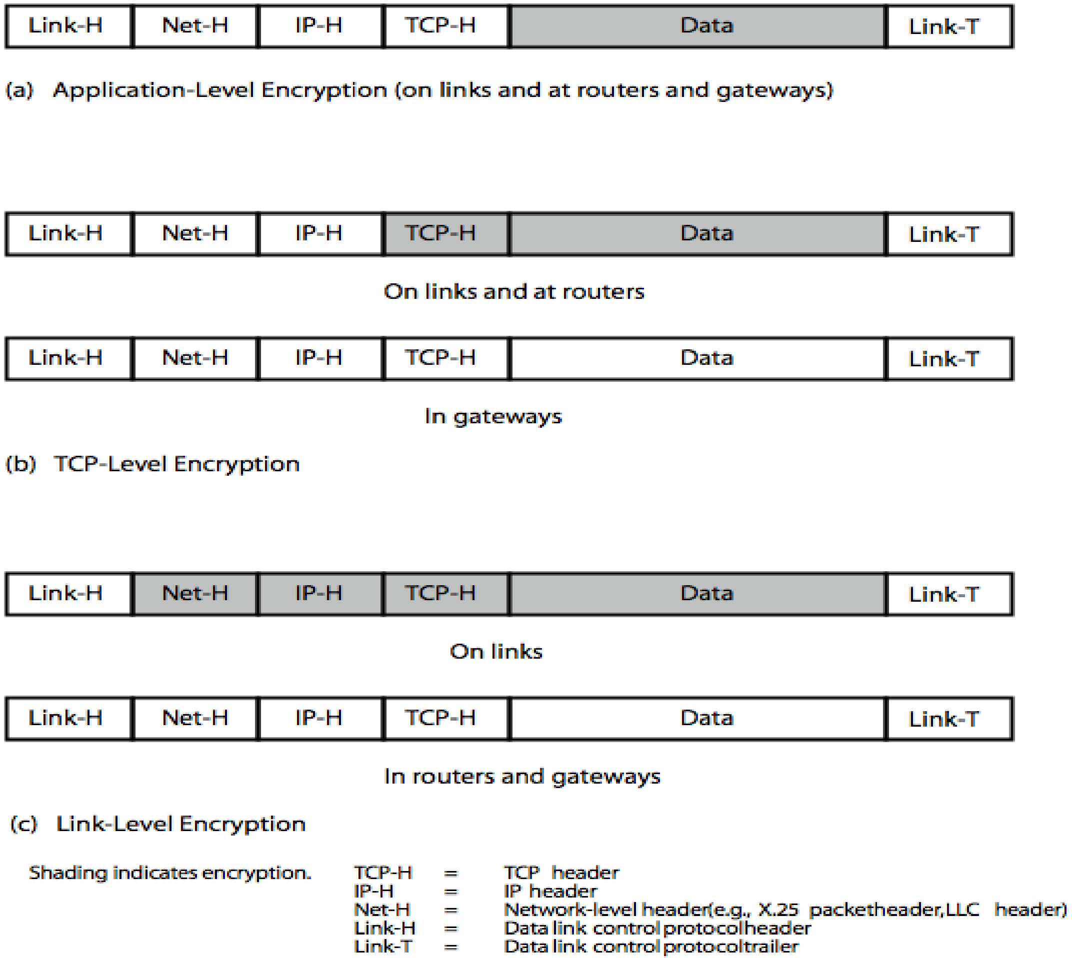
Placement of Encryption

- When using end-to-end encryption must leave headers in clear
  - so network can correctly route information
- Hence although contents protected, traffic pattern flows are not
- Ideally want both at once
  - end-to-end protects data contents over entire path and provides authentication
  - link protects traffic flows from monitoring

Placement of Encryption

- Can place encryption function at various layers in OSI Reference Model
  - link encryption occurs at layers 1 or 2
  - end-to-end can occur at layers 3, 4, 6, 7
  - as move higher less information is encrypted but it is more secure though more complex with more entities and keys





Link-H
Net-H
IP-H
TCP-H
Data
Link-T
(a) Application-Level Encryption (on links and at routers and gateways)
Link-H
Net-H
IP-H
TCP-H
Data
Link-T
On links and at routers
Link-H
Net-H
IP-H
TCP-H
Data
Link-T
In gateways
(b) TCP-Level Encryption
Link-H
Net-H
IP-H
TCP-H
Data
Link-T
On links
Link-H
Net-H
IP-H
TCP-H
Data
Link-T
In routers and gateways
(c) Link-Level Encryption
Shading indicates encryption.
TCP-H = TCP header
IP-H = IP header
Net-H = Network-level header(e.g., X.25 packetheader,LLC header)
Link-H = Data link control protocolheader
Link-T = Data link control protocoltrailer

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

Requirements of Sequences of Random Numbers

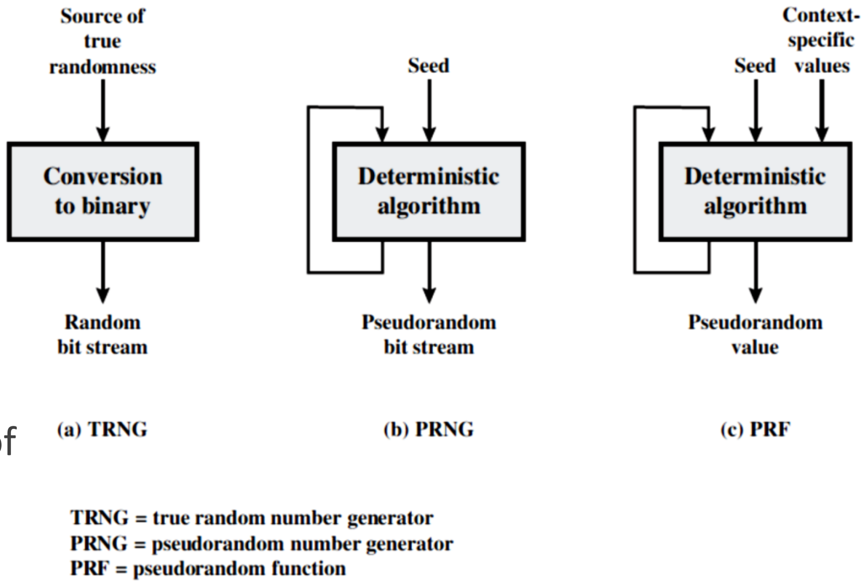
- **Randomness** The following two criteria are used to validate that a sequence of numbers is random:
  - **Uniform distribution:** the frequency of occurrence of ones and zeros should be approximately equal.
  - **Independence:** No one subsequence in the sequence can be inferred from the others.

- **Unpredictability** each number is statistically independent of other numbers in the sequence and therefore unpredictable.

- Random numbers are seldom used; rather, sequences of numbers that appear to be random are generated by some algorithm.

Types of Number Generators

- A TRNG takes as input a source that is effectively random; the source is often referred to as an **entropy source**
- PRNG takes as input a fixed value, called the **seed**, and produces a sequence of output bits using a deterministic algorithm
- Has two types:
  - **Pseudorandom number generator**: An algorithm that is used to produce an open-ended sequence of bits is referred to as a PRNG
  - **Pseudorandom function (PRF):** A PRF is used to produced a pseudorandom string of bits of some fixed length.



PRNG Requirements

- Basic requirement: an adversary who does not know the seed is unable to determine the pseudorandom string
- ***RANDOMNESS*** a pseudo random bit stream appear random even though it is deterministic.
- There is no single test for randomness. Soln: apply many tests to see if the PRNG exhibits the following features
  - **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. 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

PRNG Requirements

- *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 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.

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 (common to choose *c* = 0 and *m* prime)
  - generated sequence should appear random
  - efficient implementation
- note that an attacker can reconstruct sequence given a small number of values
- have possibilities for making this harder

Using the Linear congruential Generator

- If *a, c,* and *m* are known, observing one number in the sequence will lead to the discovery of all numbers
- Even if *a, c*, and *m* are secret, observing three numbers will lead to their calculation (3 equations in 3 unknowns)
- Ways to make it more secure:
  - Modify the number stream using a known system (ex: internal clock)
  - Ex: use the clock to restart the stream after *N* numbers, using the clock value as the new seed.

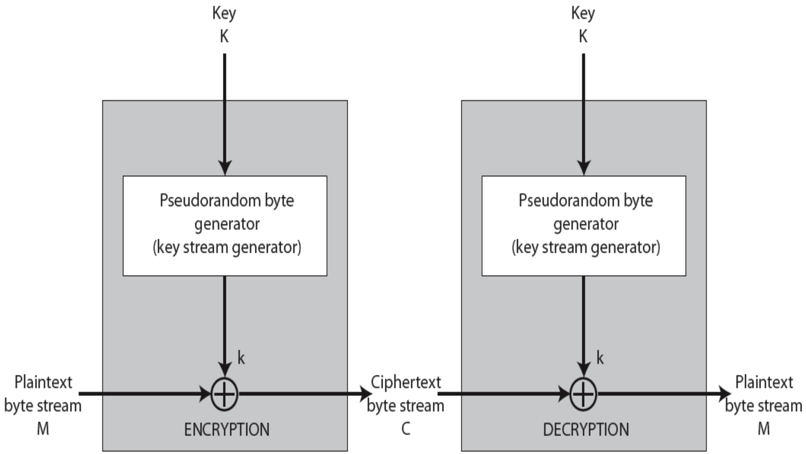
Blum Blum Shub

- 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 $p \pmod 4,\ q \pmod 4 = 3$
- 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

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

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
  - $C_i = M_i \text{ XOR } StreamKey_i$
- But must never reuse stream key
  - otherwise can recover messages



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

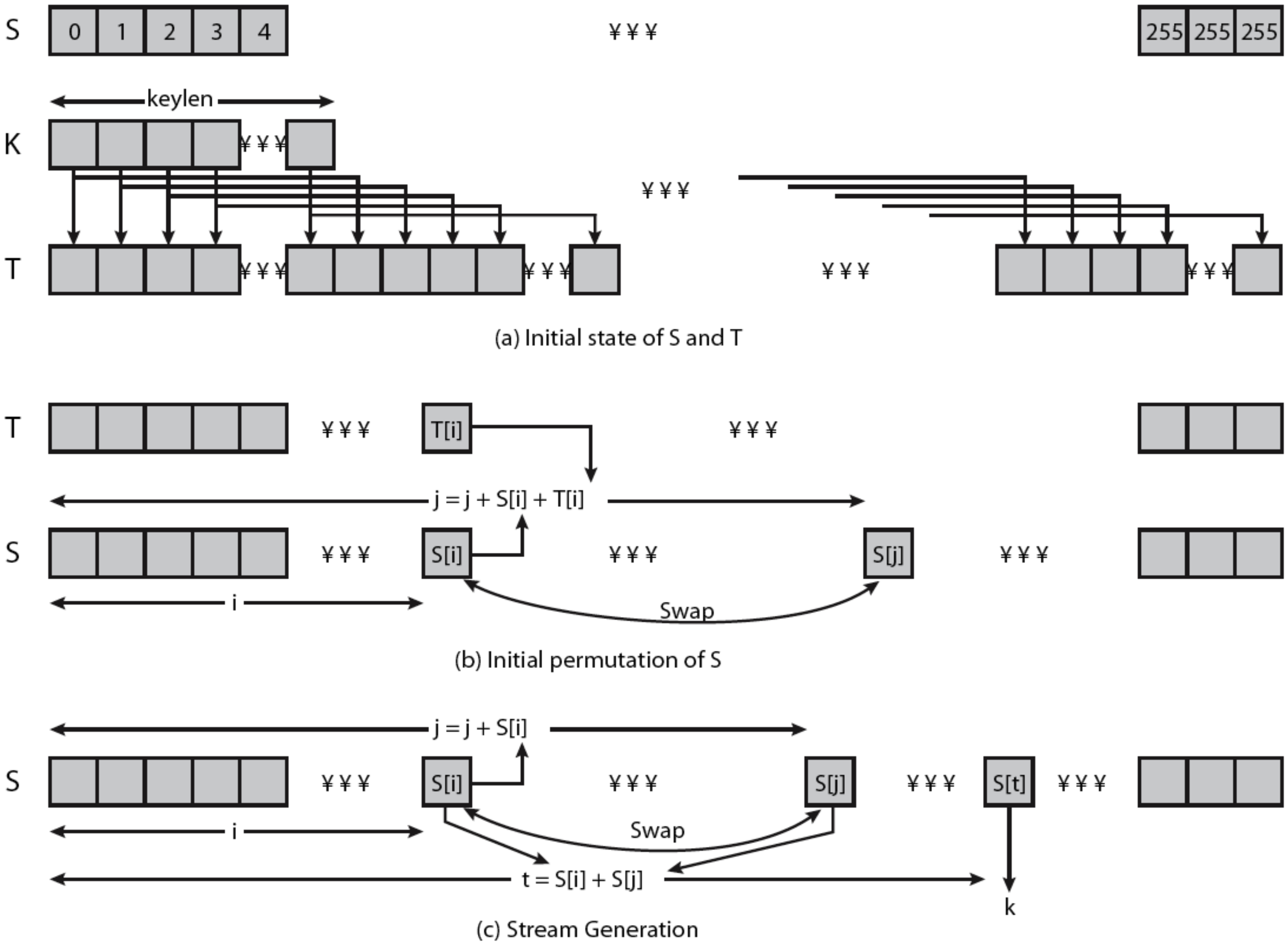
```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen])
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```

RC4 Encryption

- Encryption continues shuffling array values
- Sum of shuffled pair selects "stream key" value from permutation
- XOR S[t] 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



S
0
1
2
3
4
¥ ¥ ¥
255
255
255
keylen
K
¥ ¥ ¥
¥ ¥ ¥
T
¥ ¥ ¥
¥ ¥ ¥
¥ ¥ ¥
¥ ¥ ¥
(a) Initial state of S and T
T
¥ ¥ ¥
T[i]
¥ ¥ ¥
j = j + S[i] + T[i]
S
¥ ¥ ¥
S[i]
¥ ¥ ¥
S[j]
¥ ¥ ¥
i
Swap
(b) Initial permutation of S
j = j + S[i]
S
¥ ¥ ¥
S[i]
¥ ¥ ¥
S[j]
¥ ¥ ¥
S[t]
¥ ¥ ¥
i
Swap
t = S[i] + S[j]
k
(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

- Content in these slides are attributed to William Stallings