PRNGs - RC4 & Modes of Operations

Outline

Stream Cipher & RC4

Modes Of Operation

Pseudorandom Number Generators (PRNGs)

Random Numbers

many uses of **random numbers** in cryptography

1. Authentication protocols to prevent replay
2. Temporary session keys
3. Public key generation
4. Stream key generation

It is critical that these values are

1. Statistically random: uniform distribution, independent
2. Unpredictable: can't predict future values from previous values

"true" random sequences provide these requirements but care needed with generated random numbers

Pseudorandom Number Generators (PRNGs)

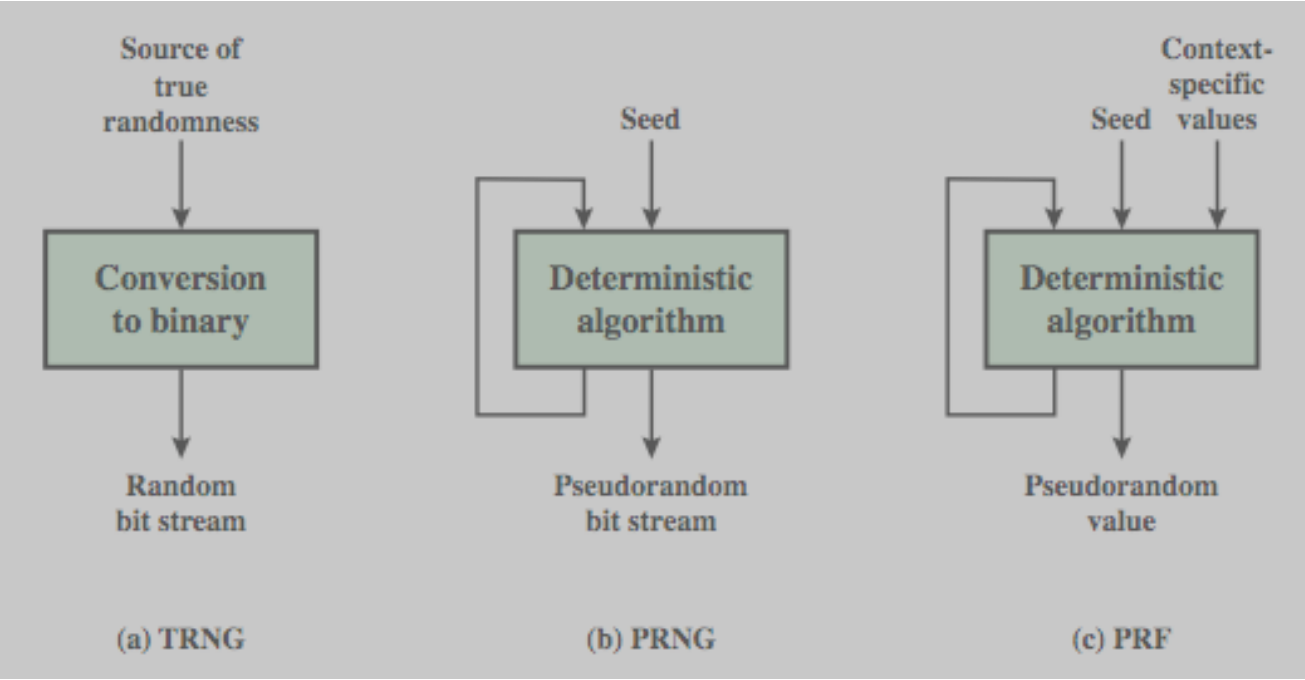
often use deterministic algorithmic techniques to create "random numbers"

- although are not truly random
- can pass many tests of "randomness"

known as "pseudorandom numbers"

created by "Pseudorandom Number Generators (PRNGs)"

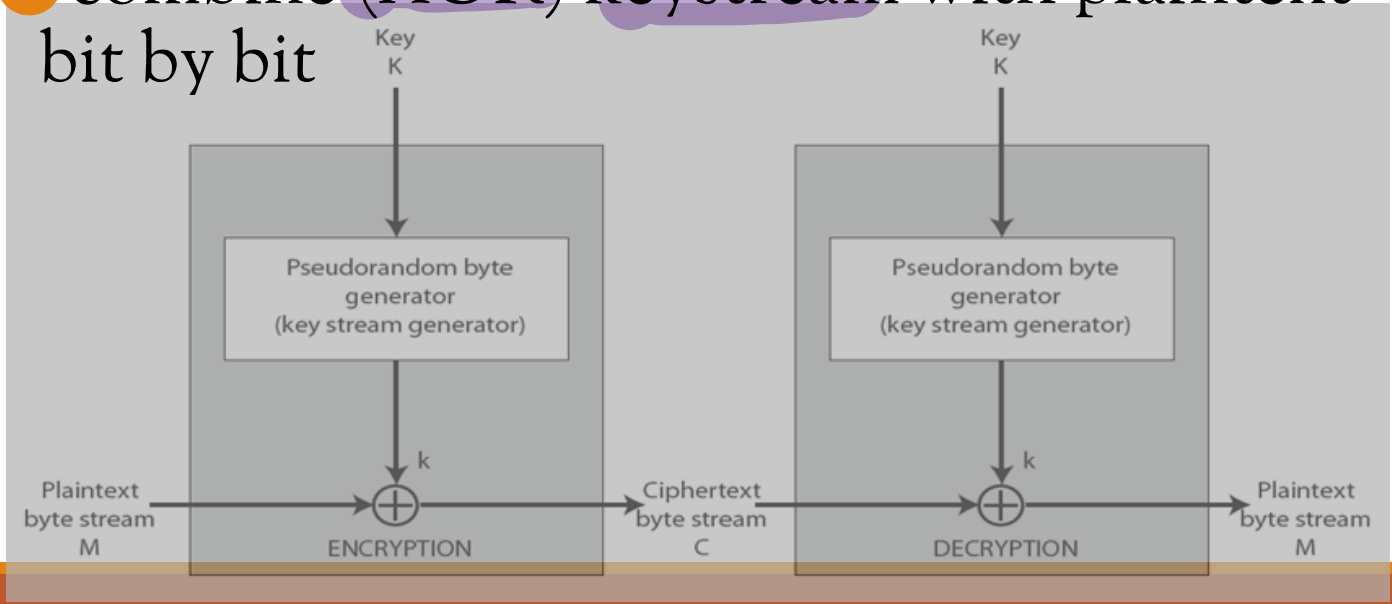
Random & Pseudorandom Number Generators



Stream Ciphers and rc4

Stream Cipher Structure

- process plaintext bit by bit (or as a byte)
- generate a pseudo random **keystream**
- combine (XOR) keystream with plaintext bit by bit



Stream Cipher Properties

properly designed, can be as secure as a block cipher with same key length

but usually simpler & faster

however must never reuse same key

- otherwise cryptanalysis is quite simple

Example: RC4

3. 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/WPA)
- 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. S forms **internal state** of the cipher
- Each of the 256 entries in S are then swapped with the j-th entry in S

```
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]);
```

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 (k=S[t])

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 Example

- instead of the full 256 bytes, we will use 8 x 3-bits.
- the state vector S is 8 x 3-bits. S can take the values 0 to 7
- K = [1 2 3 6]
- P = [1 2 2 2]
- 1- Generate Stream:
  - Initialize the state vector S and temporary vector T. S is initialized so the S[i] = i, and T is initialized so it is the key K (repeated as necessary)

    S = [0 1 2 3 4 5 6 7]

    T = [1 2 3 6 1 2 3 6]
  - Perform Initial Permutation

```
j = 0;
for i = 0 to 7 do
            j = (j + S[i] + T[i]) mod 8
            Swap(S[i],S[j]);
end
```

RC4 Example

For i = 0:
j = (0 + 0 + 1) mod 8
= 1
Swap(S[0],S[1]);

S = [1 0 2 3 4 5 6 7]

For i = 1:
j = 3
Swap(S[1],S[3])
S = [1 3 2 0 4 5 6 7];

For i = 2:
j = 0
Swap(S[2],S[0]);
S = [2 3 1 0 4 5 6 7];

For i = 3:
j = 6;
Swap(S[3],S[6])
S = [2 3 1 6 4 5 0 7];

For i = 4:
j = 3
Swap(S[4],S[3])
S = [2 3 1 4 6 5 0 7];

For i = 5:
j = 2
Swap(S[5],S[2]);
S = [2 3 5 4 6 1 0 7];

For i = 6:
j = 5;
Swap(S[6],S[4])
S = [2 3 5 4 0 1 6 7];

For i = 7:
j = 2;
Swap(S[7],S[2])
S = [2 3 7 4 0 1 6 5];

RC4 Example

- The first iteration:

  S = [2 3 7 4 0 1 6 5]
  i = (0 + 1) mod 8 = 1
  j = (0 + S[1]) mod 8 = 3
  Swap(S[1],S[3])
  S = [2 4 7 3 0 1 6 5]
  t = (S[1] + S[3]) mod 8 = 7
  k = S[7] = 5

```
i, j = 0;
while (true) {
        i = (i + 1) mod 8;
        j = (j + S[i]) mod 8;
        Swap (S[i], S[j]);
        t = (S[i] + S[j]) mod 8;
        k = S[t]; }
```

Remember, **P** = [1 2 2 2]

So our first 3-bits of ciphertext is obtained by: k XOR P
5 XOR 1 = 101 XOR 001 = 100 = 4

RC4 Example

The second iteration:
S = [2 4 7 3 0 1 6 5]
i = (1 + 1) mod 8 = 2
j = (2 + S[2]) mod 8 = 1
Swap(S[2],S[1])
S = [2 7 4 3 0 1 6 5]
t = (S[2] + S[1]) mod 8 = 3
k = S[3] = 3

Second 3-bits of ciphertext are:
3 XOR 2 = 011 XOR 010 = 001 = 1

The third iteration:
S = [2 7 4 3 0 1 6 5]
i = (2 + 1) mod 8 = 3
j = (1 + S[3]) mod 8 = 4
Swap(S[3],S[4])
S = [2 7 4 0 3 1 6 5]
t = (S[3] + S[4]) mod 8 = 3
k = S[3] = 0

Third 3-bits of ciphertext are:
0 XOR 2 = 000 XOR 010 = 010 = 2

RC4 Example

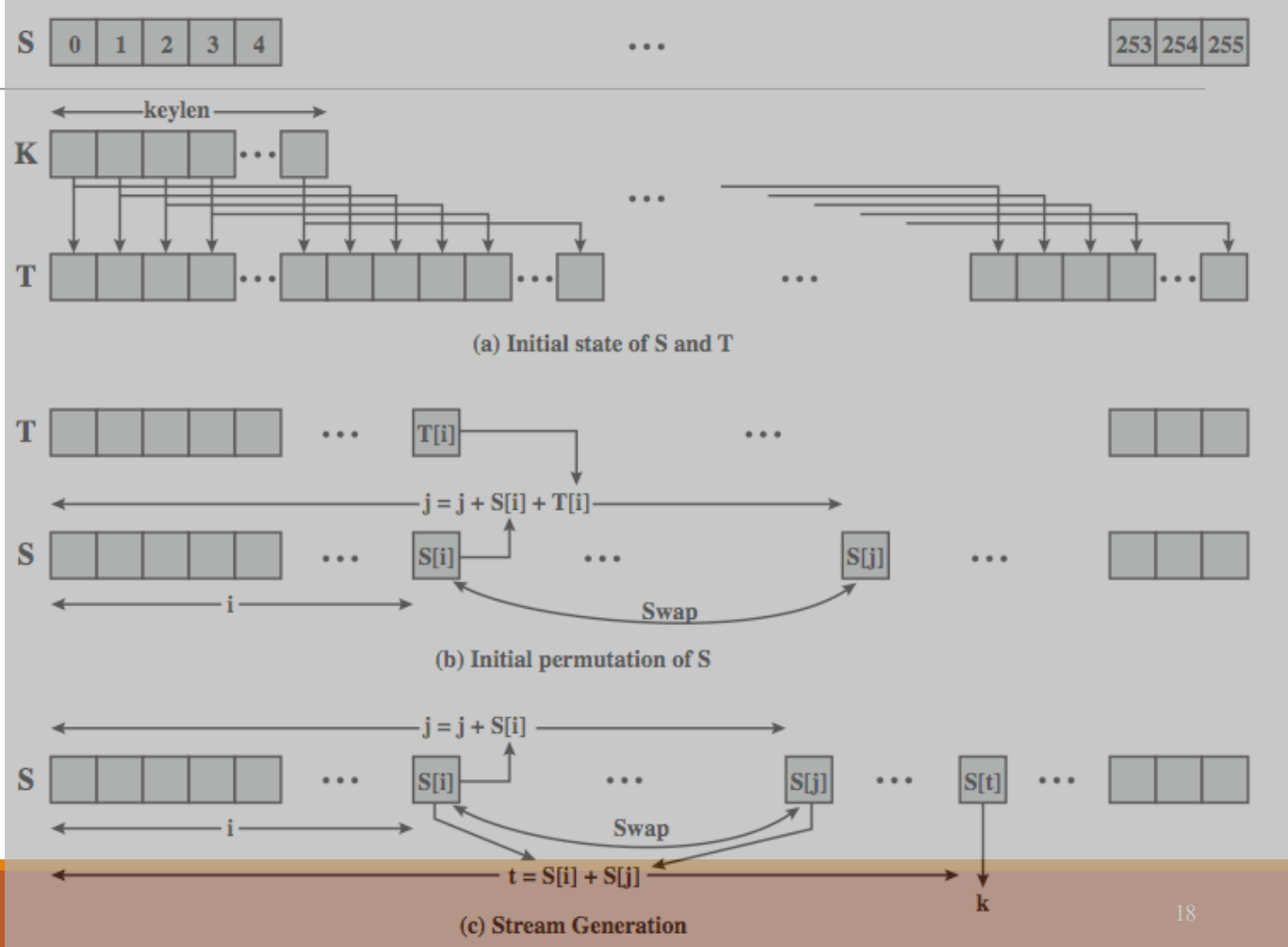
The final iteration:
S = [2 7 4 0 3 1 6 5]
i = (1 + 3) mod 8 = 4
j = (4 + S[4]) mod 8 = 7
Swap(S[4],S[7])
S = [2 7 4 0 5 1 6 3]
t = (S[4] + S[7]) mod 8 = 0
k = S[0] = 2

Last 3-bits of ciphertext are:
2 XOR 2 = 010 XOR 010 = 000 = 0

So to encrypt the plaintext stream P = [1 2 2 2] with key K = [1 2 3 6] using our simplified RC4 stream cipher we get C = [4 1 2 0]. or in binary:

P = 001010010010, K = 001010011110 and C = 100001010000

RC4 Overview



RC4 Security

claimed secure against known attacks

- have some analysis, but none practical

since RC4 is a stream cipher, must **never reuse a key**

have a concern with WEP, but due to the way keys are generated for use as input to RC4 rather than RC4 algorithm itself

Modes of Operation

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

- e.g. a message of 1000 bytes

NIST SP 800-38A defines 5 modes have **block** and **stream** modes to cover a wide variety of applications can be used with any block cipher

Message Padding

at end of message must handle a possible last short block

- which is not as large as blocksize of cipher
- pad either with known non-data value (eg nulls)
- or pad last block along with count of pad size
  - eg. [b1 b2 b3 0 0 0 0 5]
  - means have 3 data bytes, then 5 bytes pad+count
- this may require an extra entire block over those in message

there are other modes, which avoid the need for an extra block

Electronic Codebook Book (ECB)

message is broken into independent blocks which are encrypted

Each block encrypted using the same key

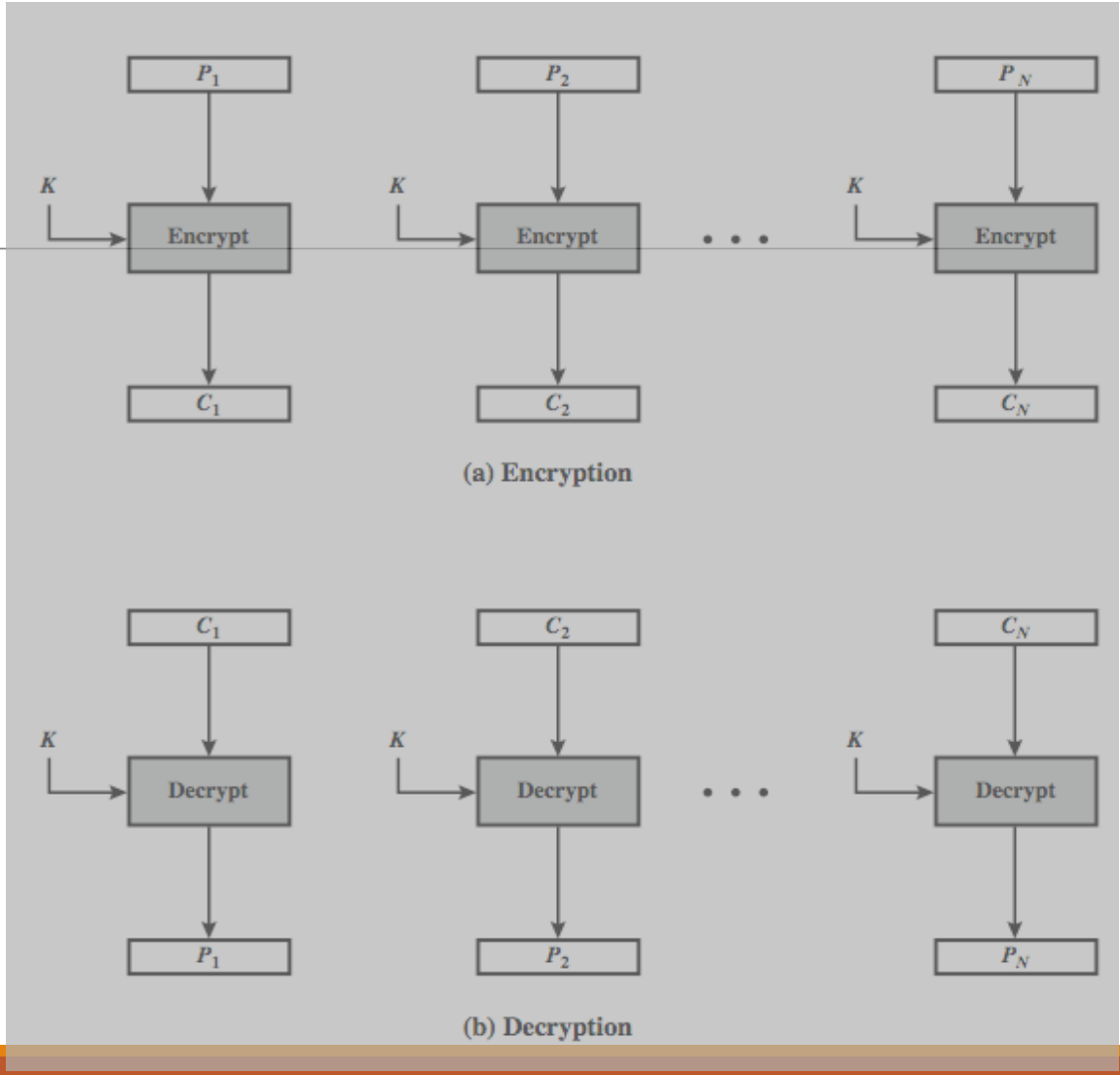
each block is a value which is substituted, like a codebook, hence name

each block is encoded independently of the other blocks

$$C_i = E_K(P_i)$$

uses: secure transmission of single values

Electronic Codebook Book (ECB)



Advantages and 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.

e.g. Session encryption keys

Cipher Block Chaining (CBC)

message is broken into blocks

linked together in encryption operation

each previous cipher blocks is chained with current plaintext block, hence name

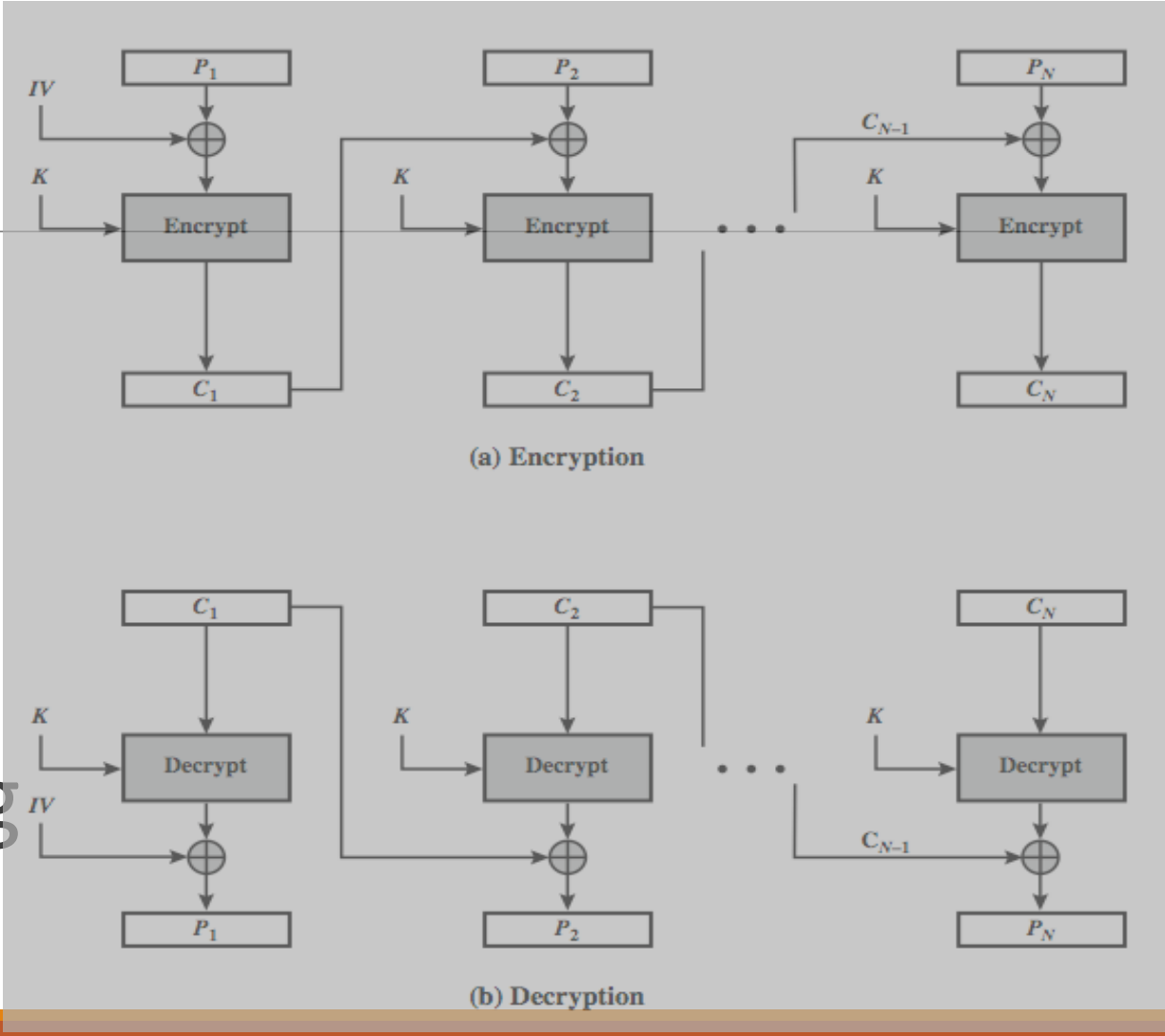
use Initial Vector (IV) to start process

$C_i = E_K(P_i \text{ XOR } C_{i-1})$

$C_{-1} = IV$

uses: bulk data encryption, authentication

Cipher Block Chaining (CBC)



Advantages and 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 sent in clear, attacker can change bits of first block, and change IV to compensate
- hence IV must either be a fixed value
- (as in EFTPOS Electronic Funds Transfer at Point Of Sale)
- or must be sent encrypted in ECB mode before rest of message

Stream Modes of Operation

block modes encrypt entire block

may need to operate on smaller units

- real time data

convert block cipher into stream cipher

- cipher feedback (CFB) mode
- output feedback (OFB) mode
- counter (CTR) mode

use block cipher as some form of **pseudo-random number** generator

Using Block Ciphers as PRNGs

for cryptographic applications, can use a block cipher to generate random numbers

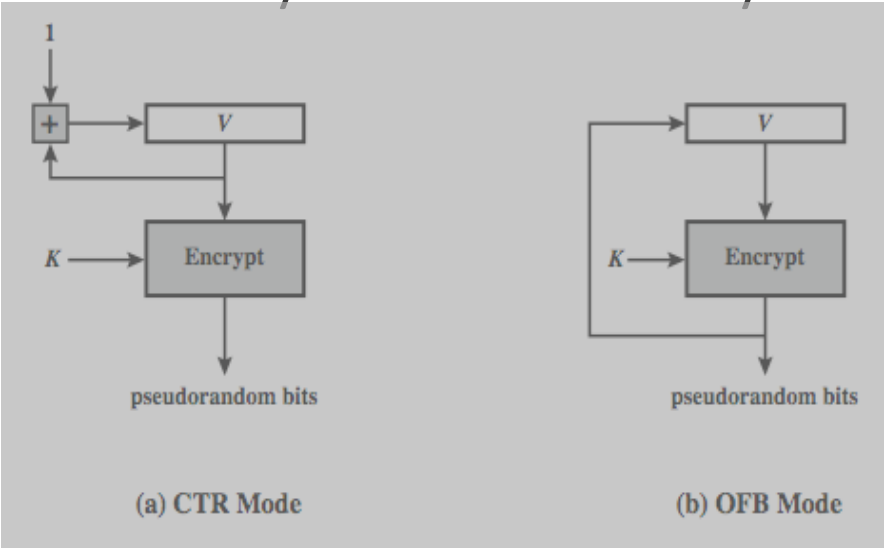
often for creating session keys from master key

CTR

$$X_i = E_K[V_i]$$

OFB

$$X_i = E_K[X_{i-1}]$$



Cipher FeedBack (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

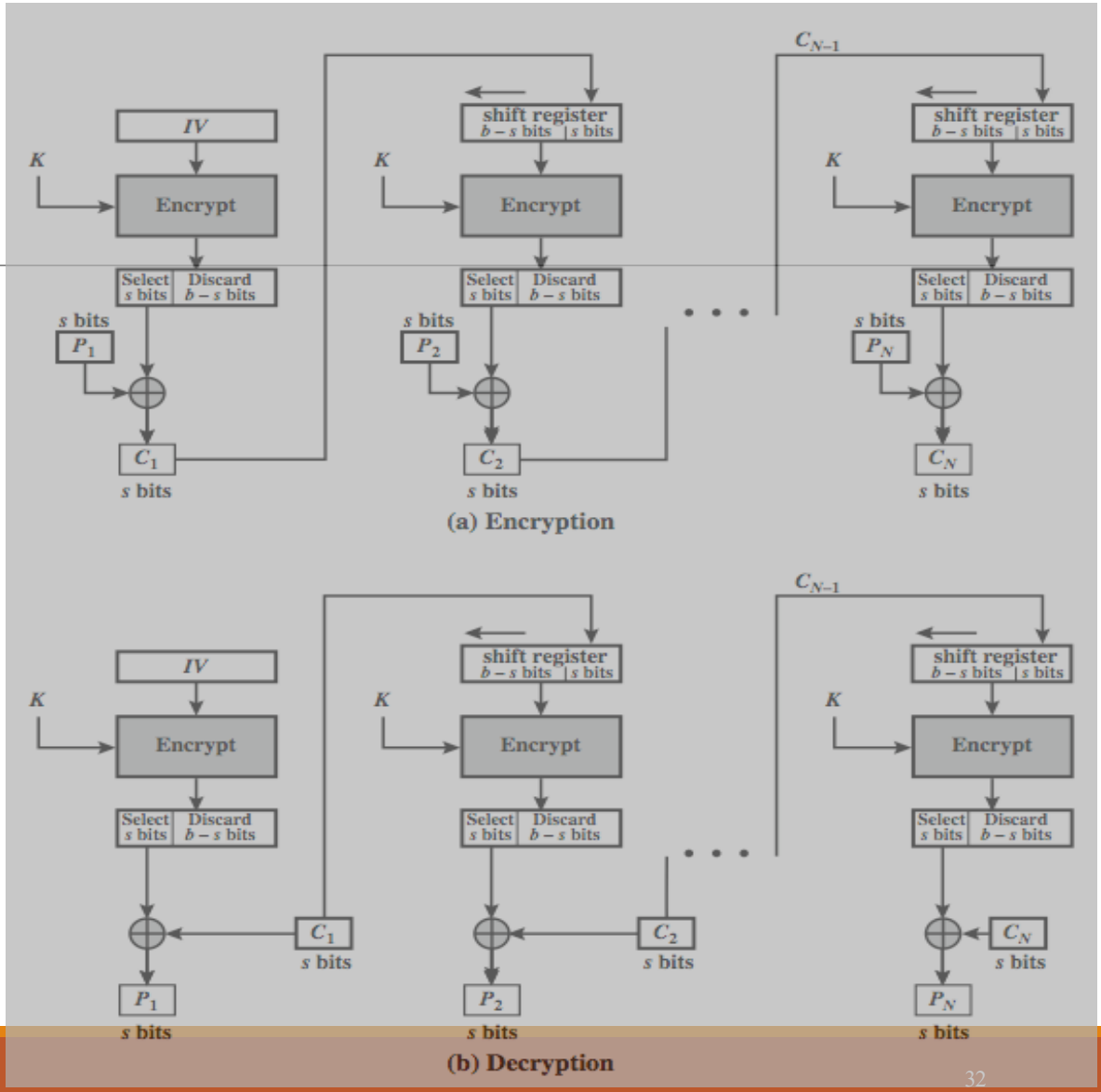
most efficient to use all bits in block (64 or 128)

$C_i = P_i \text{ XOR } E_K(C_{i-1})$

$C_{-1} = IV$

uses: stream data encryption, authentication

s-bit Cipher Feed Back (CFB-s)



Advantages and Limitations of CFB

appropriate when data arrives in bits/bytes

most common stream mode

errors propagate for several blocks after the error

Output FeedBack (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

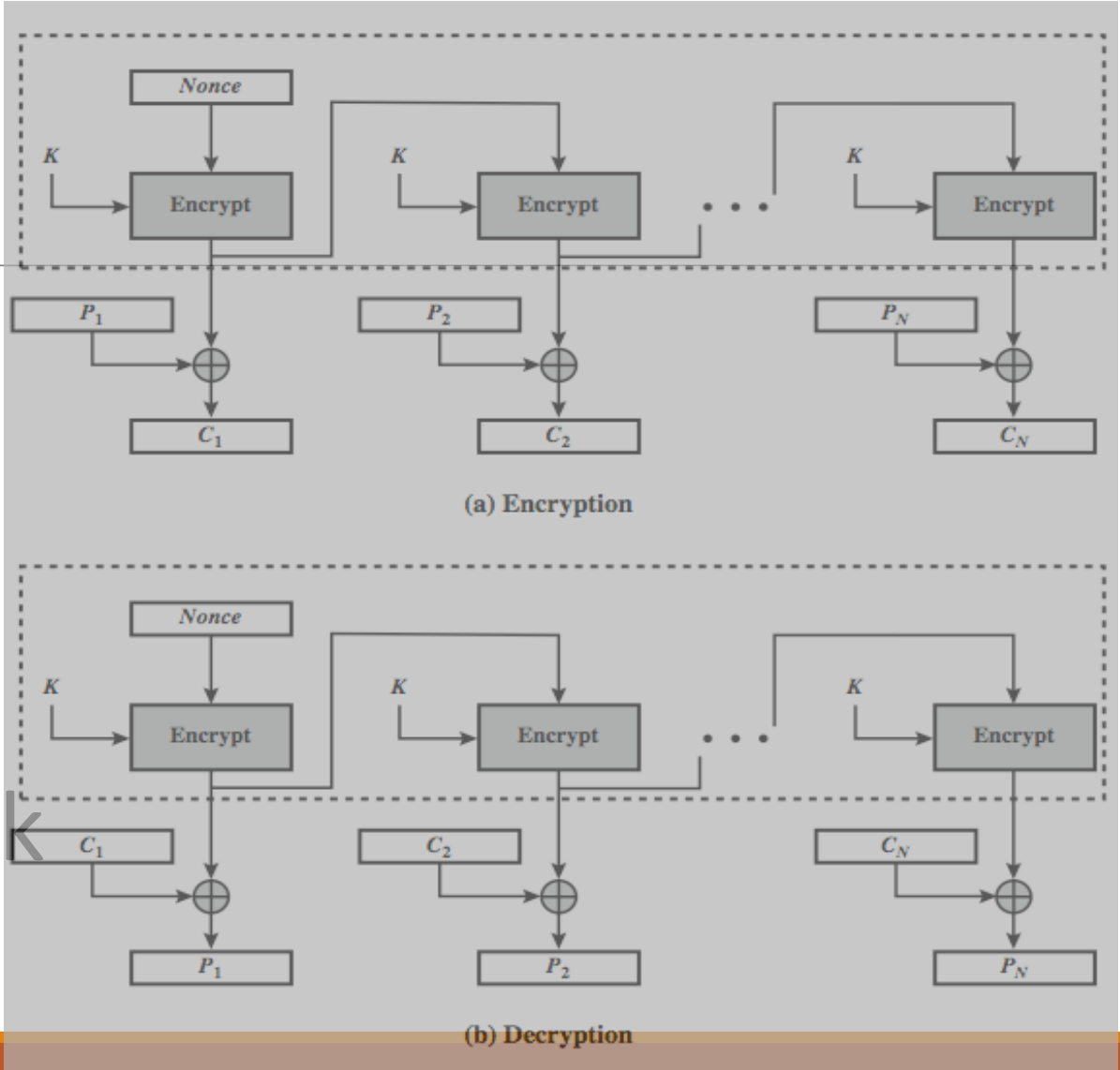
$$O_i = E_K(O_{i-1})$$

$$C_i = P_i \text{ XOR } O_i$$

$$O_{-1} = IV$$

uses: stream encryption on noisy channels

Output FeedBack (OFB)



Advantages and Limitations of OFB

needs an IV which is unique for each use

- if ever reuse attacker can recover outputs

Advantage: bit errors do not propagate

Disadvantage: more vulnerable to message stream modification

sender & receiver must remain in sync

only use with full block feedback

- subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

Counter (CTR)

a “new” mode, though proposed early on

similar to OFB but encrypts counter value rather than any feedback value

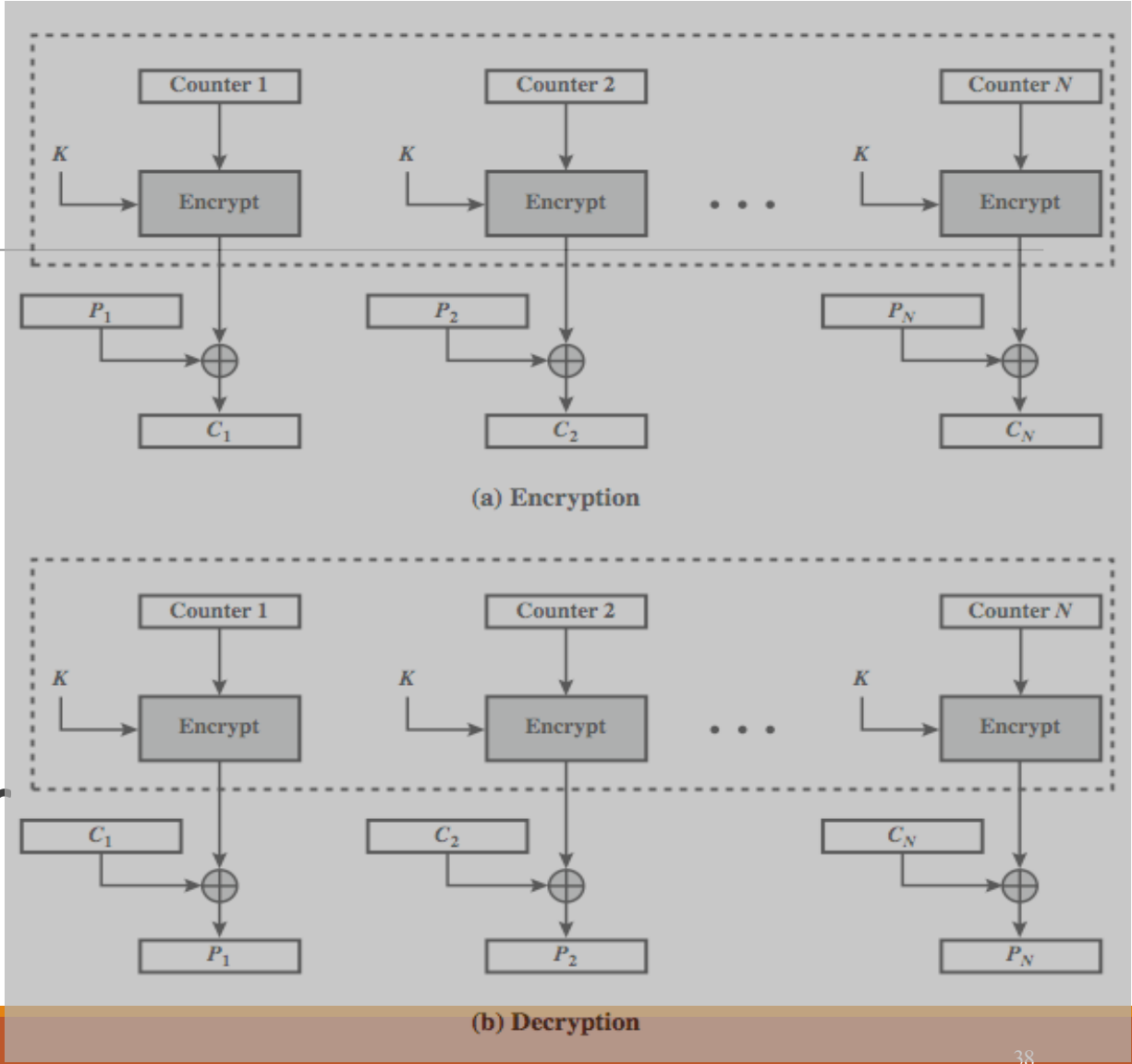
must have a different key & counter value for every plaintext block (never reused)

$O_i = E_K(i)$

$C_i = P_i \text{ XOR } O_i$

uses: high-speed network encryptions

Counter (CTR)


Counter 1
Counter 2
Counter N
K
Encrypt
P_1
P_2
P_N
C_1
C_2
C_N
(a) Encryption
(b) Decryption

Advantages and Limitations of CTR

efficiency

- can do parallel encryptions in h/w or s/w
- can preprocess in advance of need
- can process blocks of plaintext or ciphertext in a random-access fashion

provable security (good as other modes)

but must ensure never reuse key/counter values, otherwise could break (cf OFB)

References

Network Security Essentials: Applications and Standards, chapter 2 .

Network Security : Private Communication in a Public World, chapter 3.