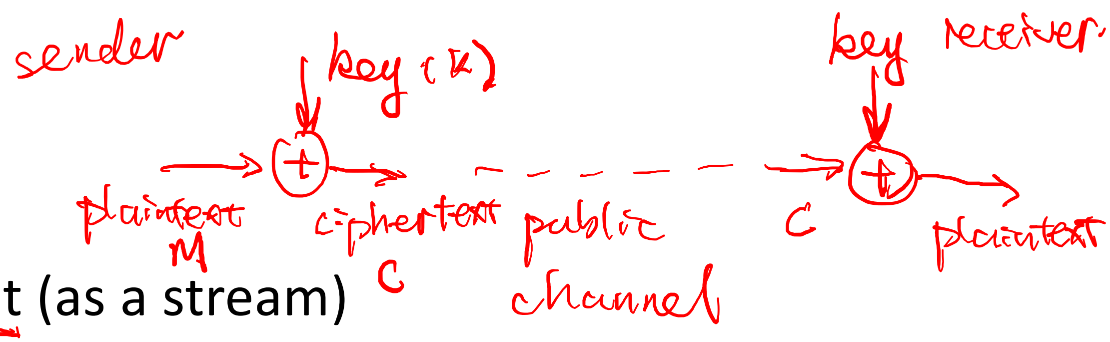
Stream Ciphers



- process the message bit by bit (as a stream)
- typically have a (pseudo) random **stream key**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys any statistically properties in the message
  - $C_i = M_i \text{ XOR } \boxed{\text{StreamKey}_i}$
- what could be simpler!!!!
- but must never reuse stream key
  - otherwise, can remove effect and recover messages, M⊕K⊕K = M

stream cipher

$M=1$, $1 \oplus 0 = 1$

$M=0$, $0 \oplus 0 = 0$

$C_i = M_i \oplus \boxed{K_i}$ ← as random as possible

security

ideal case: truely random

// 0

Associativity $M \oplus K \oplus K = M \oplus (K \oplus K)$

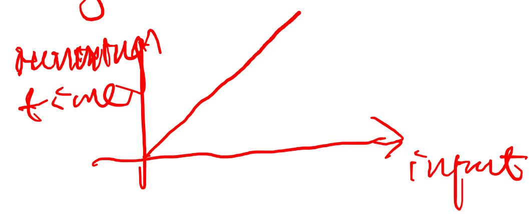
Decrypt $= M \oplus 0 = M$

Stream Cipher Properties

- some design considerations are:
  - statistically random → key
  - depends on large enough key → reuse and independent
  - large linear complexity → increase input linear, running time O(n), → even faster
  - correlation immunity
  - confusion
  - diffusion

independent

plaintext



RC4 → wifi WEP

swap( )

XOR

swap

key

random

long

How to generate Stream Key?

- How to generate Stream Key?

PRNG

Stream Ciphers

- Idea: replace "rand" by "pseudo rand"

computational identify

- Use Pseudo Random Number Generator
  - A secure PRNG produces output that looks indistinguishable from random
  - An attacker who can't see the internal PRNG state can't learn any output

small key

- PRNG: $\{0,1\}^s \rightarrow \{0,1\}^n$ $n >> s$ state,
  - expand a short (e.g., 128-bit) random seed into a long (typically unbounded) string that "looks random"

s

PRNG { ① seed(key) ② Generate( ) }

- Secret key is the seed
  - Basic encryption method: $E_{key}[M] = M \oplus PRNG(key)$

$= M \oplus Key$

Stream Ciphers

Key distribution chapter 4, PKI,

- Protocol: Alice and Bob both seed a secure PRNG with their symmetric secret key, and then use the output as the key for stream key

& static

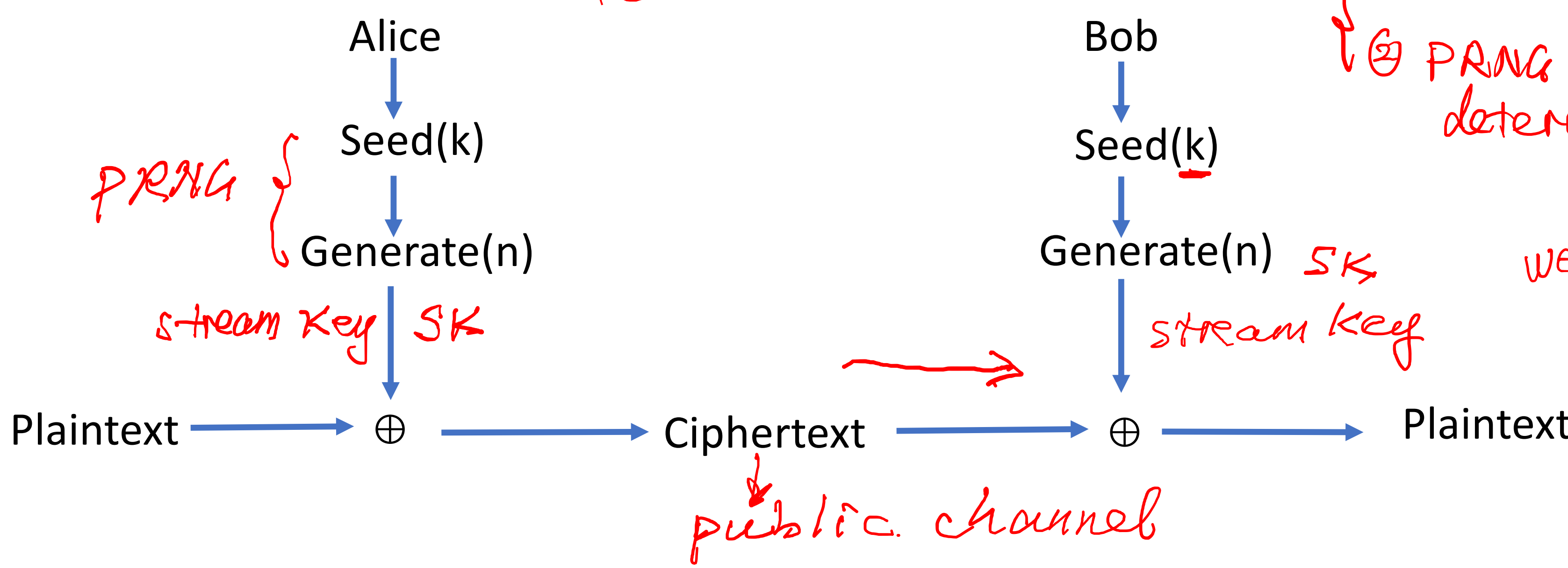
K is shared / preshared

→ small

① K is same

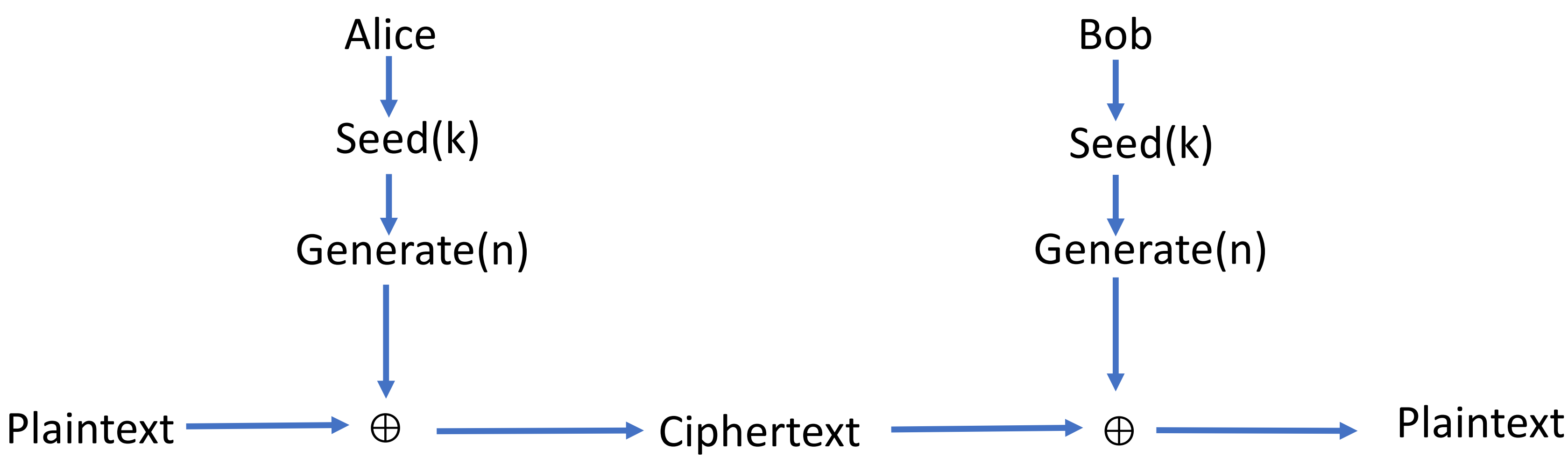
② PRNG is deterministic

weakness?



Stream Ciphers: Encrypting Multiple Messages

- How do we encrypt multiple messages without key reuses?



Stream Ciphers: Encrypting Multiple Messages

→ synchronizat

IV.

10 1 0 | 0 1

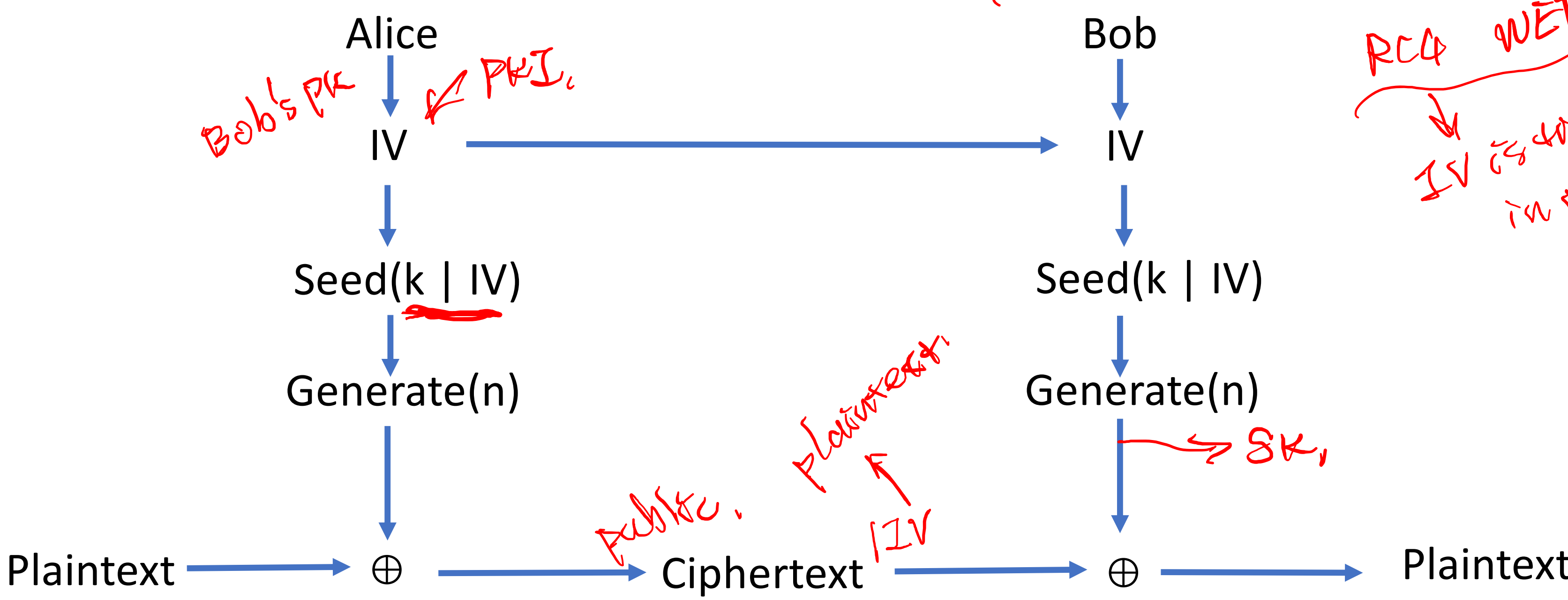
cyphertext

- Solution: For each message, seed the PRNG with the key and a random IV, concatenated("|"). Send the IV with the ciphertext

seed

RC4 WEP V.

IV is transmitted in plaintext

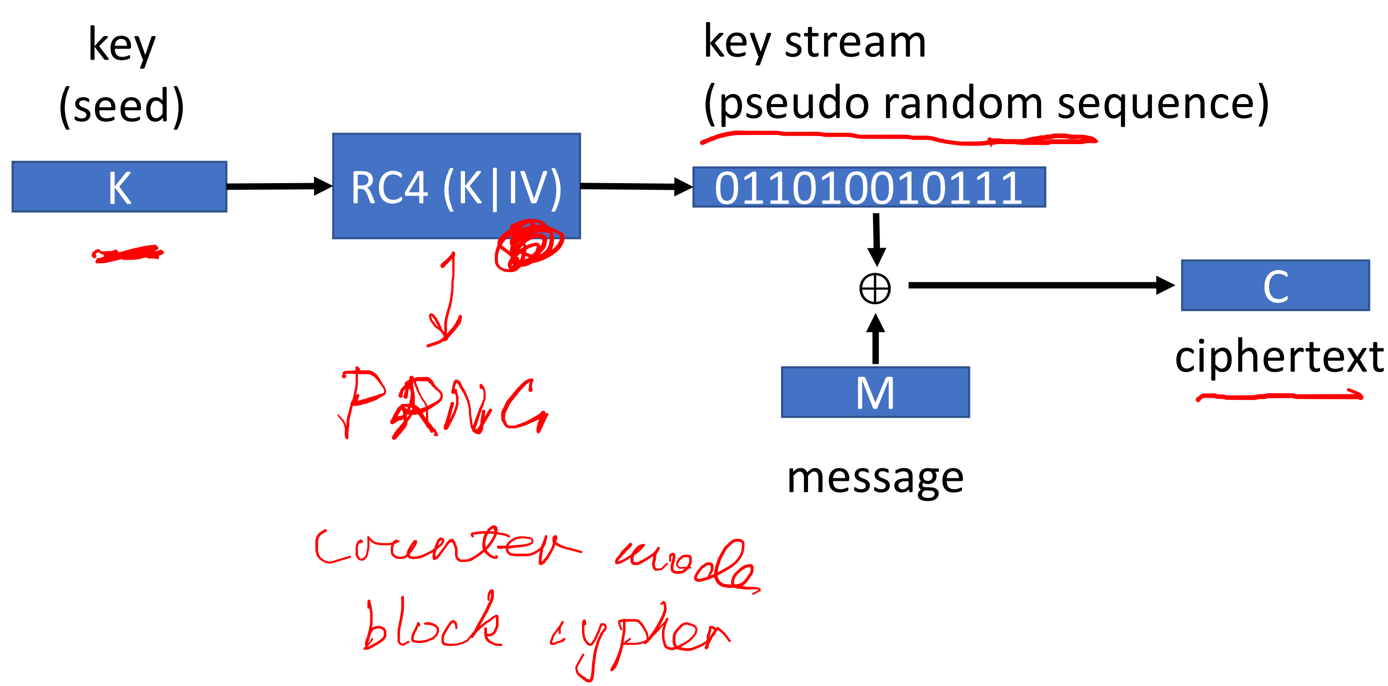


Real-world example: RC4

- a proprietary cipher designed in 1987
- Extremely simple but effective! → fast.
- Very fast - especially in software
- Easily adapts to any key length, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP, WAP) → Transport
- A trade secret by RSA Security 1994 leaked to public.
- uses that permutation to scramble input info processed a byte at a time

swaps

RC4 Stream Cipher



① RC4 Key Schedule ② Encryption

- 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

```
/* Initialization */
for i = 0 to 255 do          256
S[i] = i;
T[i] = K[i mod keylen];      internal
                             K + IV

/* 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]);  -> diffusion
```

Throw away ~~T & K~~, retain S

$T[i] = K[i \bmod keylen]$ $K = [1, 2, 3, 4]$ $keylen = 4$

0

If $keylen > i$; $T[i] = [1, 2, 3, 4]$ $0 \bmod 4 = 0 \Rightarrow K[0]$

0 1 2 3 $1 \bmod 4 = 1 \Rightarrow K[1]$

$T[i] = K[i]$

If $keylen \le i$ $T[4] = K[4 \bmod 4] = K[0] = 1$

$T[5] = K[5 \bmod 4] = K[1] = 2$

$T = [1, 2, 3, 4, 1, 2] 3\ 4, 1\ 2\ 3\ 4$

repeat pattern, $\Rightarrow$ reuse key

∴ add IV to be large enough