Lecture 10

Stream Ciphers

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
  - $C_i = M_i$ XOR $StreamKey_i$
- what could be simpler!!!!
- but must never reuse stream key
  - otherwise, can remove effect and recover messages, M⊕K⊕K = M

How to generate Stream Key?

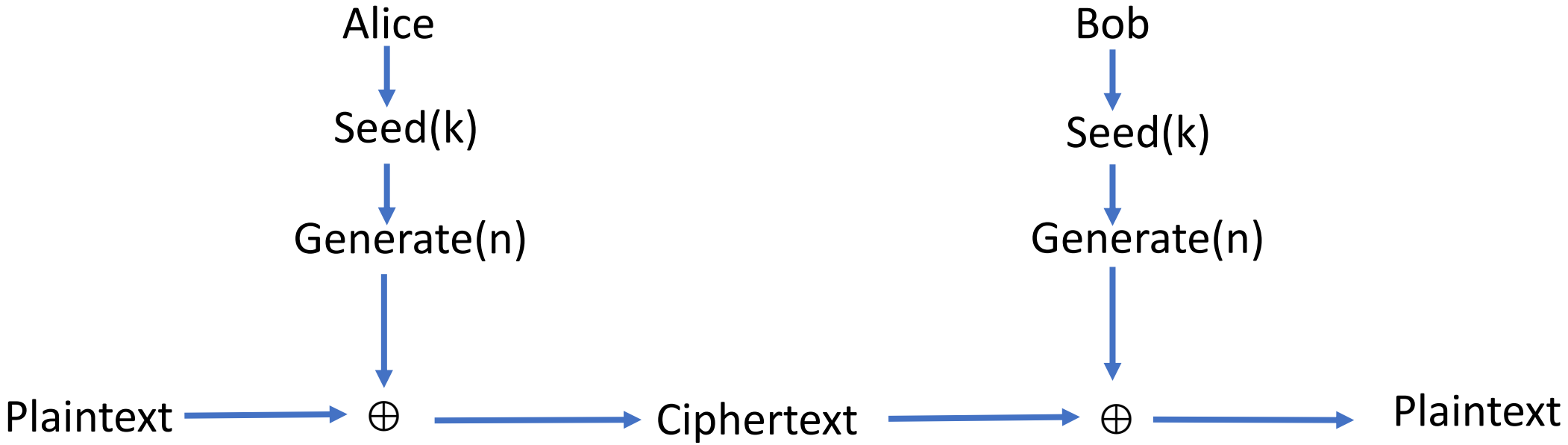
- How to generate Stream Key?

Stream Ciphers

- Idea: replace “rand” by “pseudo rand”
- 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
- PRNG: $\{0,1\}^s \rightarrow \{0,1\}^n$
  - expand a short (e.g., 128-bit) random seed into a long (typically unbounded) string that “looks random”
- Secret key is the seed
  - Basic encryption method: $E_{key}[M] = M \oplus PRNG(key)$

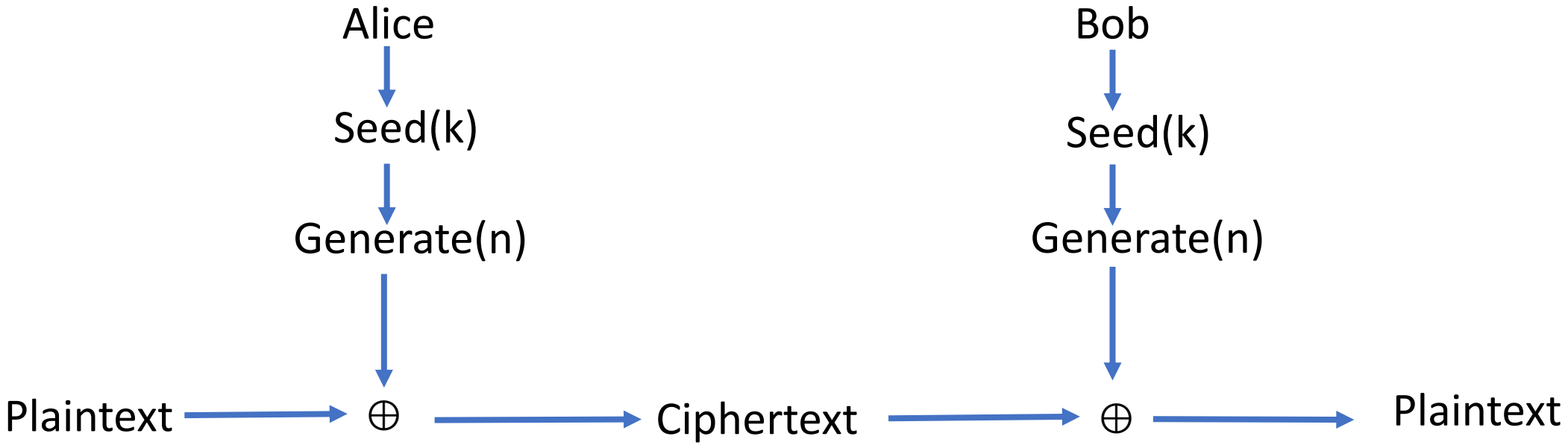
Stream Ciphers

- 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



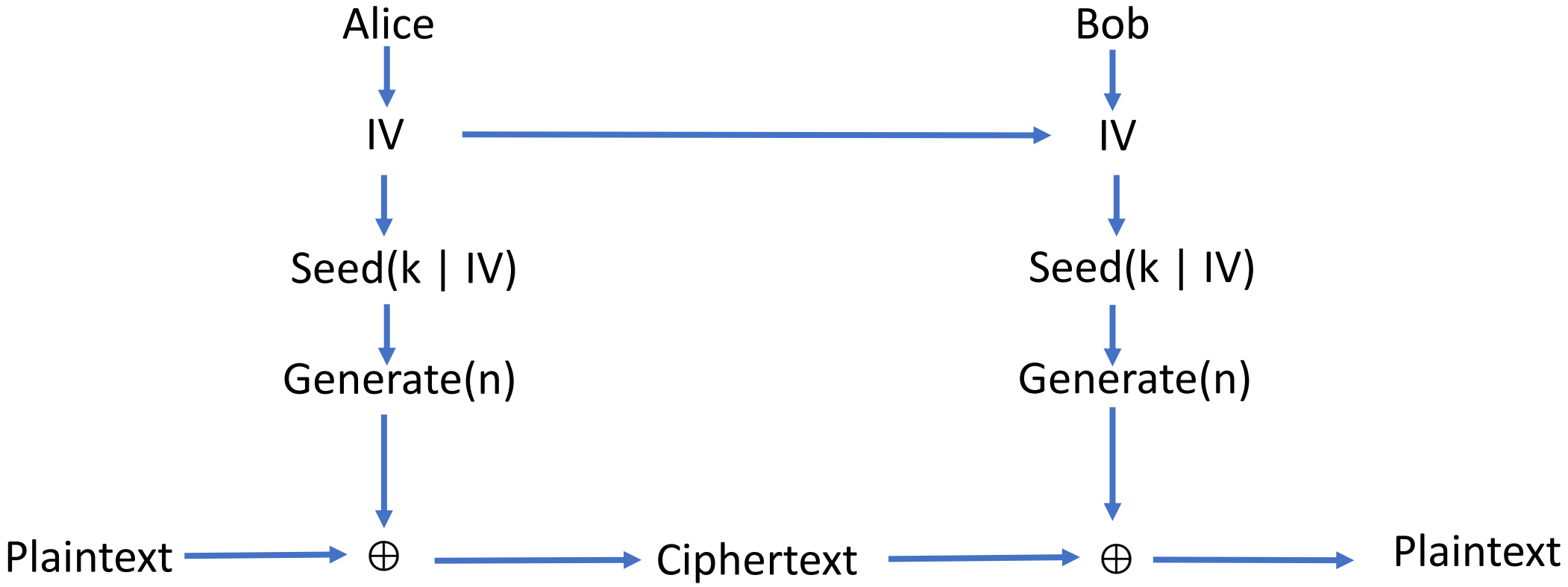
Stream Ciphers: Encrypting Multiple Messages

- How do we encrypt multiple messages without key reuses?



Stream Ciphers: Encrypting Multiple Messages

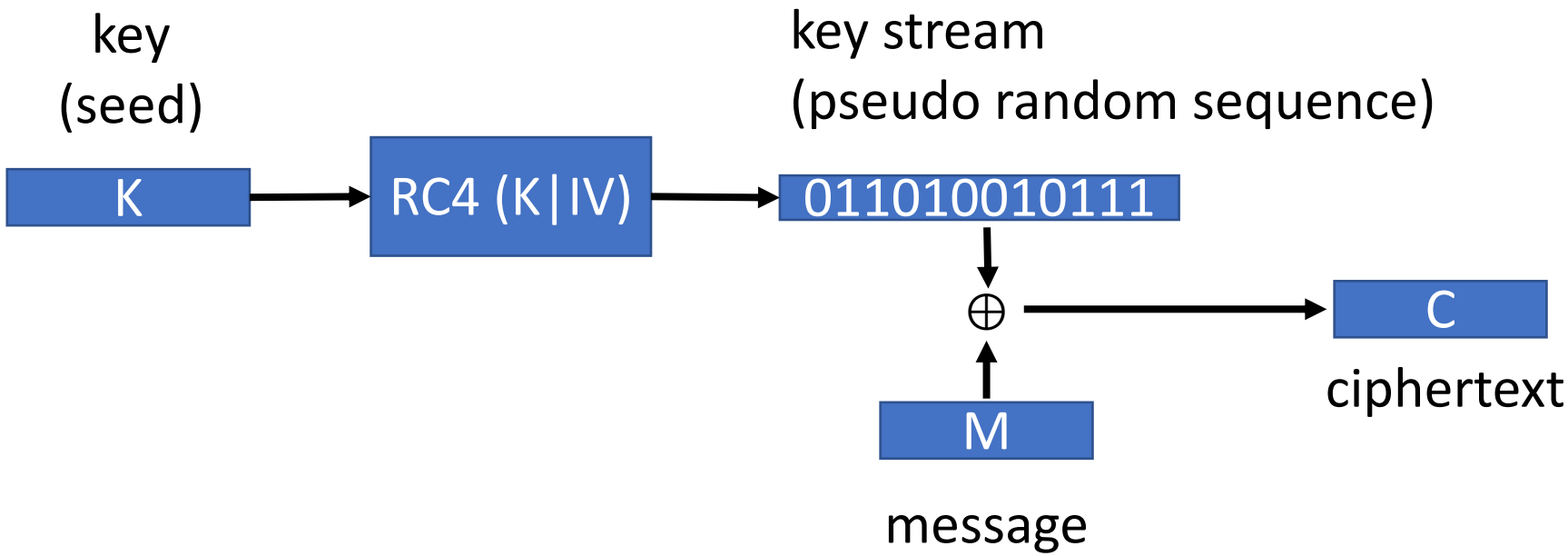
- Solution: For each message, seed the PRNG with the key and a random IV, concatenated("|"). Send the IV with the ciphertext



Real-world example: RC4

- A proprietary cipher designed in 1987
- Extremely simple but effective!
- Very fast - especially in software
- Easily adapts to any key length, byte-oriented stream cipher
- Uses that permutation to scramble input info processed a byte at a time
- Widely used (web SSL/TLS, wireless WEP, WPA)

RC4 Stream Cipher



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

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

```
/* 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]);
```

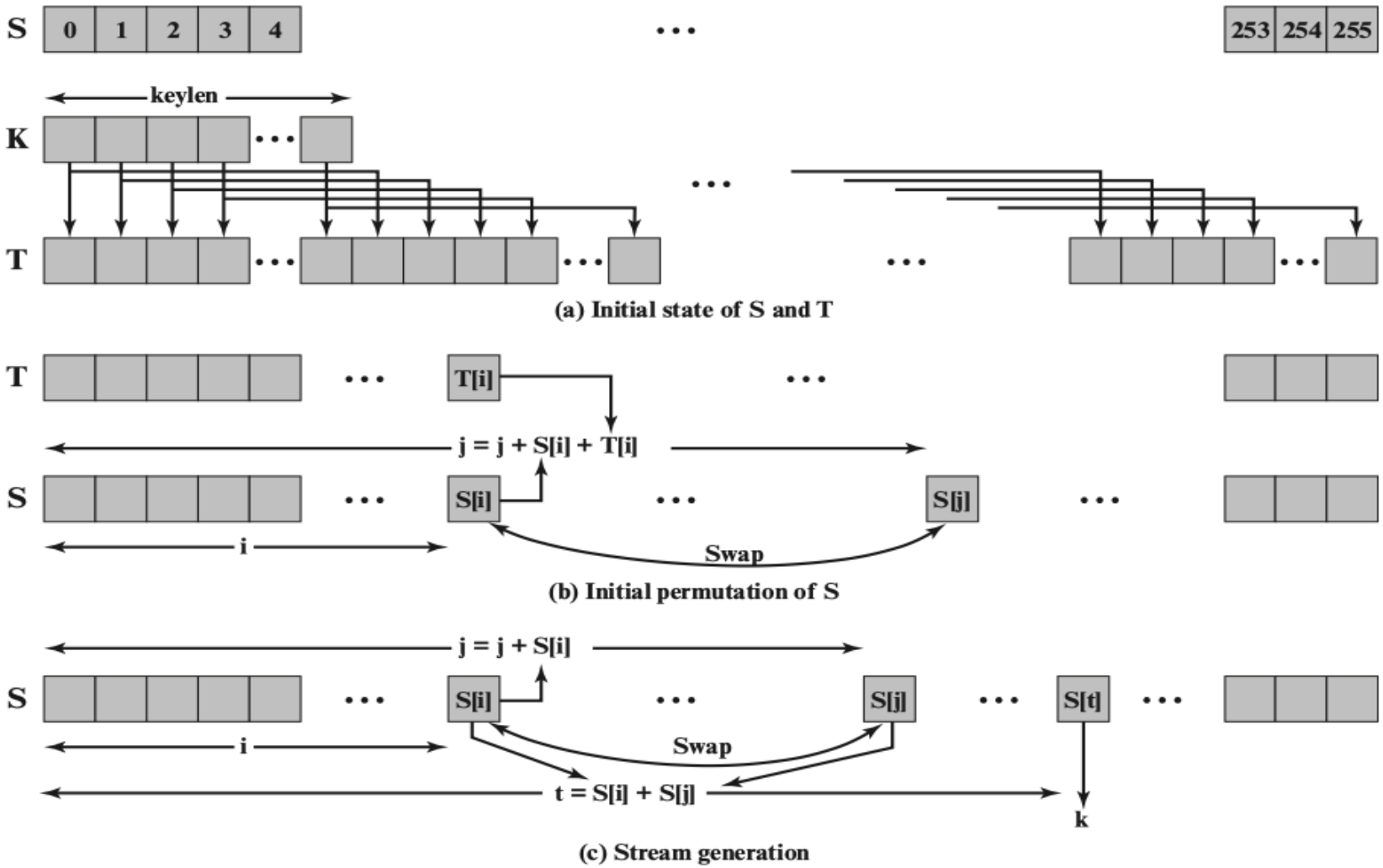
Throw away T & K, retain S

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value
- XOR 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
253
254
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