Basics of Cryptology
Symmetric cryptography

Block vs. Stream ciphers

- Stream ciphers process a bit or byte at a time during encryption/decryption
- Block ciphers encrypt block at a time
  - Message is divided into blocks and encrypted

Stream ciphers

- Many ciphers before 1940
- Enigma
- A5 (GSM towards base station)
- WEP (802.11)
- RC-4 (Ron’s Code)

Block ciphers

- DES
- 3-DES
- AES
- RC-2
- RC-5
- IDEA
- Blowfish
- Cast
- Gost

Outline

- Stream ciphers
  - RC4
- Block ciphers
  - Electronic codebooks(ECB) , Cipher-block chaining (CBC) , Cipher feedback (CFB), Output feedback (OFB), Counter (CTR)
  - Substitution-permutation networks
  - Feistel ciphers
  - DES (overview)
  - 3DES (overview)
  - AES (overview)

Stream ciphers

RC4

- It is made of two main steps
  1. Keystream creation

     A pseudo-random stream of bits is created which depends on the key that typically is between 40 and 256 bits.
  2. Encoding/decoding

     The pseudo-random stream is used to encode/decode by using bitwise XOR operation

- It is similar to the Vernam cipher but it uses pseudo-random bits, rather than an ad-hoc sequence. The pseudo-random bits depend on the key.
- To generate the keystream, the cipher makes use of two internal variables:
  - S that is a permutation of all 256 possible bytes (S contains 256 bytes).
  - Two 8-bit index-pointers (denoted "`i`" and "`j`").

RC4

- Initialization (Key-scheduling algorithm - KSA)

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + Key[i mod |Key|]) mod 256
    swap values of S[i] and S[j]
endfor
```

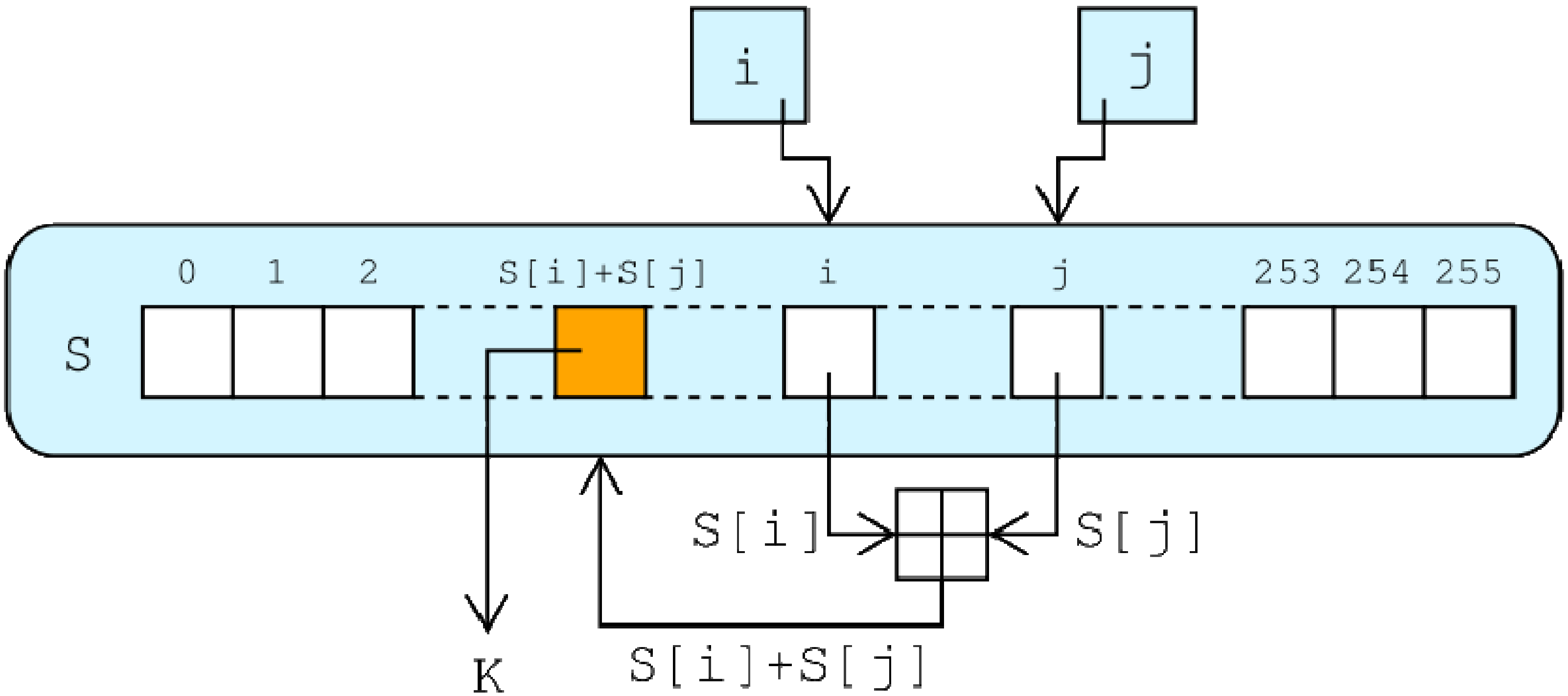
- In this way we obtain a permutation S[0]S[1]…S[255] which is a function of the key

RC4

- Keystream generation (Pseudo-random generation algorithm - PRGA)

```
i := 0
j := 0
while length of the plaintext:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

RC4



- K is used in XOR with the next byte of the message to produce the next byte of either ciphertext or plaintext

RC4

- Decoding is identical to encoding since the inverse of XOR is XOR itself:
  1. Keystream creation
  2. Encoding/decoding

RC4 - Example

- We use 3 bits instead of 8 (8 instead of 256 symbols)
- `Key = [2 3 1 6]    m = [2 1 3 1]`
- Initialization (Key-scheduling algorithm - KSA)

```
for i from 0 to 7
    S[i] := i
endfor
j := 0
for i from 0 to 7
    j := (j + S[i] + Key[i mod |Key|]) mod 8
    swap values of S[i] and S[j]
Endfor
```

LET US SEE THE EXECUTION STEP BY STEP:

- ```
  for i from 0 to 7
      S[i] := i
  endfor
  ```
- `S = [0 1 2 3 4 5 6 7]`
  ```

RC4 - Example

- We use 3 bits instead of 8 (8 instead of 256 symbols)
- `Key = [2 3 1 6]     m = [2 1 3 1]`
- `Inizialization (KSA)`

```
j := 0
for i from 0 to 7
    j := (j + S[i] + Key[i mod |Key|]) mod 8
    swap values of S[i] and S[j]
endfor
```

- `S = [0 1 2 3 4 5 6 7]`
- `i=0`
  `j = 0 + 0 + 2 mod 8= 2`
  `S = [2 1 0 3 4 5 6 7]`
- `i=1`
  `j = 2 + 1 + 3 mod 8 = 6`
  `S = [2 6 0 3 4 5 1 7]`
- `i=2`
  `j = 6 + 0 + 1 mod 8 = 7`
  `S = [2 6 7 3 4 5 1 0]`
- `i=3`
  `j = 7 + 3 + 6 mod 8 = 0`
  `S = [3 6 7 2 4 5 1 0]`
- `i=4`
  `j = 0 + 4 + 2 mod 8 = 6`
  `S = [3 6 7 2 1 5 4 0]`
- `i=5`
  `j = 6 + 5 + 3 mod 8 = 6`
  `S = [3 6 7 2 1 4 5 0]`
- `i=6`
  `j = 6 + 5 + 1 mod 8 = 4`
  `S = [3 6 7 2 5 4 1 0]`
- `i=7`
  `j = 4 + 0 + 6 mod 8 = 2`
  `S = [3 6 0 2 5 4 1 7]`

RC4 - Example

- S = [3 6 0 2 5 4 1 7]   m = [2 1 3 1]
- Keystream generation (PRGA)

```
i := 0 j := 0
while length of the plaintext:
    i := (i + 1) mod 8
    j := (j + S[i]) mod 8
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 8]
    output K
endwhile
```

- First iteration
  - i = 0 + 1 mod 8 = 1; j = 0 + 6 mod 8 = 6
  - S = [3 1 0 2 5 4 6 7]
  - s[1] + S[6] mod 8 = 1 + 6 mod 8 = 7
  - K = S[7] = 7
- Second iteration
  - i = 1 + 1 mod 8= 2; j = 6 + 0 mod 8 = 6
  - S = [3 1 6 2 5 4 0 7]
  - S[2] + S[6] mod 8 = 6 + 0 mod 8 = 6
  - K = S[6] = 0
- Third iteration
  - i = 2 + 1 mod 8= 3; j = 6 + 2 mod 8 = 0
  - S = [2 1 6 3 5 4 0 7]
  - S[3] + S[0] mod 8 = 3 + 2 mod 8 = 5
  - K = S[5] = 4
- Fourth iteration
  - i = 3 + 1 mod 8 = 4; j = 0 + 5 mod 8 = 5
  - S = [2 1 6 3 4 5 0 7]
  - S[4] + S[5] mod 8 = 4 + 5 mod 8 = 1
  - K = S[1] = 1

K = 7 0 4 1

RC4 - Example

- K = 7 0 4 1
- m = [2 1 3 1]
- K is used to encode/decode m by using bitwise XOR operation
- C = 7 XOR 2 = 111 XOR 010 = 101 = 5
- C = 0 XOR 1 = 000 XOR 001 = 001 = 1
- C = 4 XOR 3 = 100 XOR 011 = 111 = 7
- C = 1 XOR 1 = 001 XOR 001 = 000 = 0