What we have learned

- Basics of cryptography
  - Crypto terms
  - Kerckhoffs' principles
  - Shift crypto
  - Vegenere
  - Double transposition
  - One-time pad
  - Codebook cipher

Quiz Time

Quiz 1: What does Kerckhoffs' Principle assume for cryptographic design?

- A: Assume that both the algorithm and the key are open to the attacker
- B: Assume that both the algorithm and the key are secret to the attacker
- C: Assume that the algorithm is secret, and the key is open
- **D: Assume that the algorithm is open, and the key is secret**

Quiz 2: Caesar’s cipher (shift by 3)

- Given that the Caesar's cipher was used, find the plaintext that corresponds to the following ciphertext:

E L Q J K D P W R Q

b i n g h a m t o n

Quiz 3: Use one-time pad for encryption

- Suppose that the following table is used for one-time pad,

```
e=000  h=001  i=010  k=011  l=100  r=101  s=110  t=111
```

The ciphertext is “LIKE,” and the plaintext is “till.” What is the key?

- A: 1 1 1, 1 0 0
- B: 1 0 0, 1 1 1, 0 0 0, 0 1 1
- C: **0 1 1, 0 0 0, 1 1 1, 1 0 0**
- D: 0 0 0, 0 1 1, 1 0 0, 1 1 1

Quiz 4: Use codebook for encryption

- Suppose that the following is an excerpt from the decryption codebook for a classic codebook cipher.

  123 once

  199 or
  202 maybe

  221 twice

  233 time

  332 upon

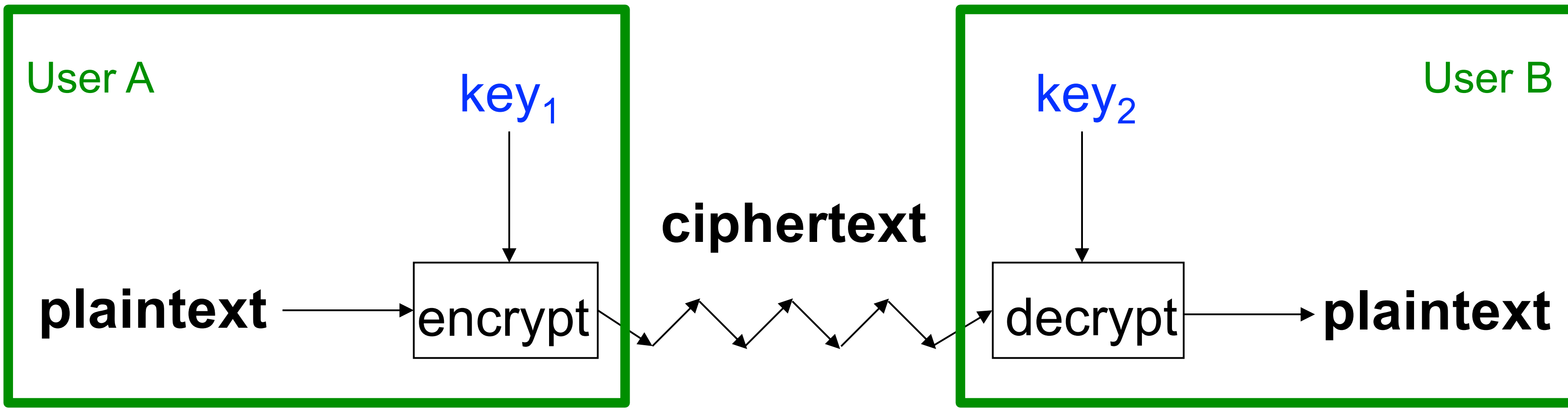
  451 a
- Decrypt the ciphertext: 242, 554, 650, 464, 532, 749, 567 assuming that the following additive sequence was used to encrypt the message: 119, 222, 199, 231, 333, 547, 346
- **once upon a time or maybe twice**

Stream Ciphers

Claude Shannon

- Founded field of information theory
- His 1949 paper: *Inform. Thy. of Secrecy Systems*
- Cipher Design Principles
  - **Confusion** — obscure relationship between plaintext and ciphertext
    - Substitution: One-time Pad
  - **Diffusion** — spread plaintext statistics through the ciphertext
    - Transposition: Double transposition
- Proved one-time pad is secure

Cryptosystem - Refresh



$Key_1 = Key_2$: Symmetric

$Key_1 \neq Key_2$: Asymmetric

Symmetric Key Crypto

- **Stream cipher** — based on one-time pad
  - Except that key is relatively short
  - Key is *stretched* into a long **keystream**
  - Keystream is used just like a one-time pad
- **Block cipher** — based on codebook concept
  - Block cipher key determines a codebook
  - Each key yields a different codebook
  - Employs both “confusion” and “diffusion”

One-Time Pad, Encryption

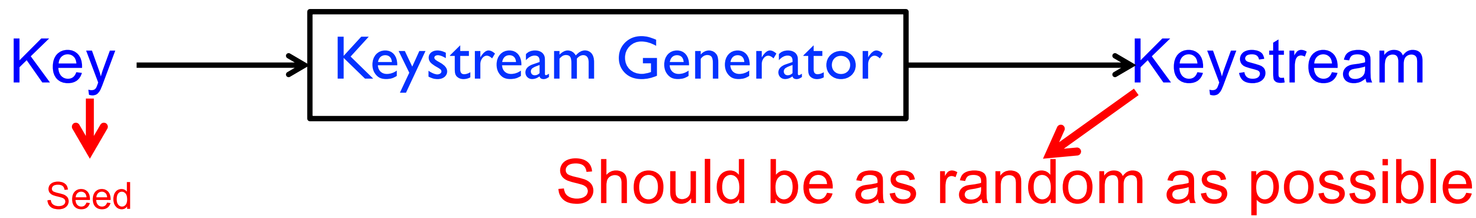
`e=000   h=001   i=010   k=011   l=100   r=101   s=110   t=111`

Encryption: Plaintext ⊕ Key = Ciphertext

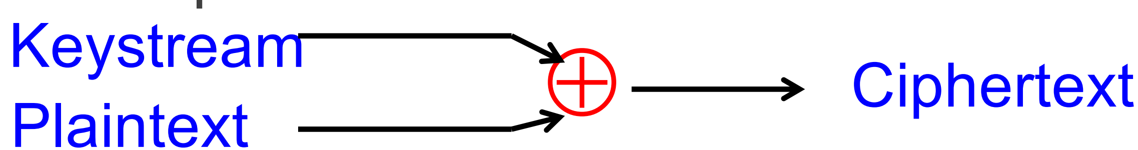
	h	e	i	l	h	i	t	l	e	r
Plaintext:	001	000	010	100	001	010	111	100	000	101
Key:	111	101	110	101	111	100	000	101	110	000
Ciphertext:	110	101	100	001	110	110	111	001	110	101
	s	r	l	h	s	s	t	h	s	r

Stream Cipher - Encryption

- A **keystream generator** takes a key K of n bits in length and stretches it into a long **keystream**

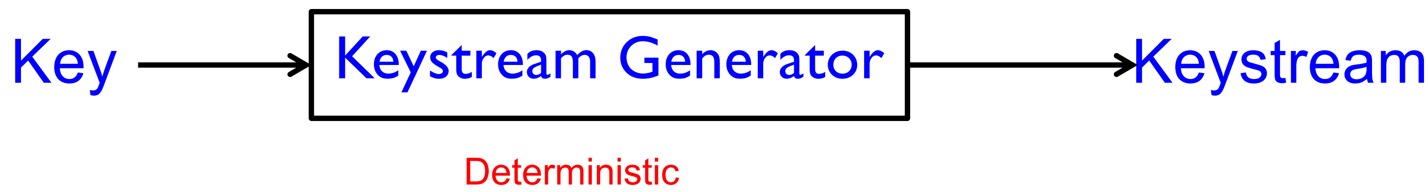


- **Encryption**: The keystream is XORed with the plaintext P to produce ciphertext C.

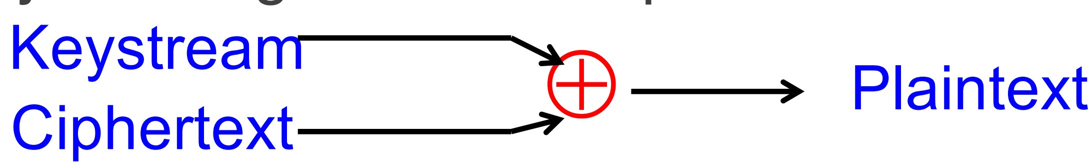


Stream Cipher - Decryption

- A **keystream generator** takes a key K of n bits in length and stretches it into a long **keystream**



- **Decryption**: The same keystream is used to recover the plaintext by XORing it with the ciphertext



Stream Ciphers

We'll discuss two stream ciphers

- **A5/1**
  - Based on shift registers
  - Used in GSM mobile phone system
- **RC4**
  - Based on a changing lookup table
  - Used at many places

A5/1

A5/1 and A5/2 - History

- There are seven A5 ciphering algorithms which have been defined for GSM use
  - A5/1 developed in 1987
  - A5/2 developed in 1989
  - Both kept secret initially – against the Kerckhoffs' principle
  - Reverse engineered in 1999

- A5/2 is weaker than A5/1, and was used instead of the stronger A5/1 for export, certain countries outside of Europe

A5/1 and GSM

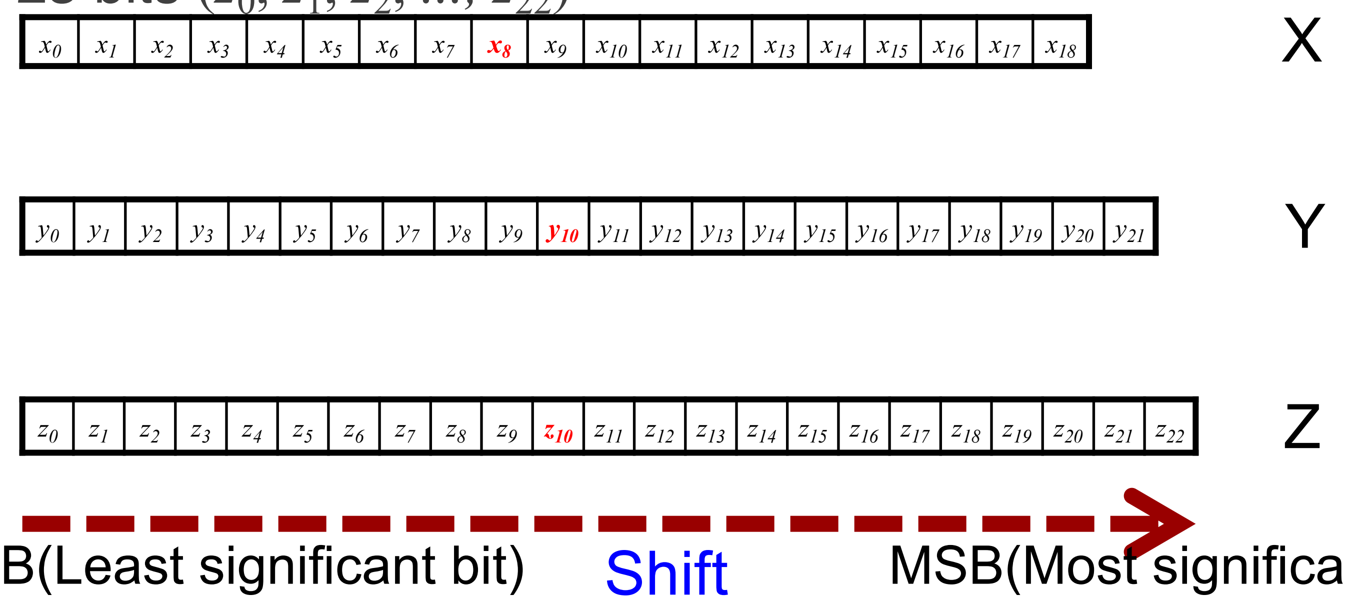
- **Goal**: provides over-the-air communication privacy in the GSM mobile phone standard
- Transmission in GSM: is organised as sequences of *bursts*
  - For one direction, one burst is sent every 4.615 milliseconds and contains 114 bits available for information
- A5/1 is used to produce for each burst a 114 bit sequence of keystream which is XORed with the 114 bits data

A5/1 keystream generator in detail

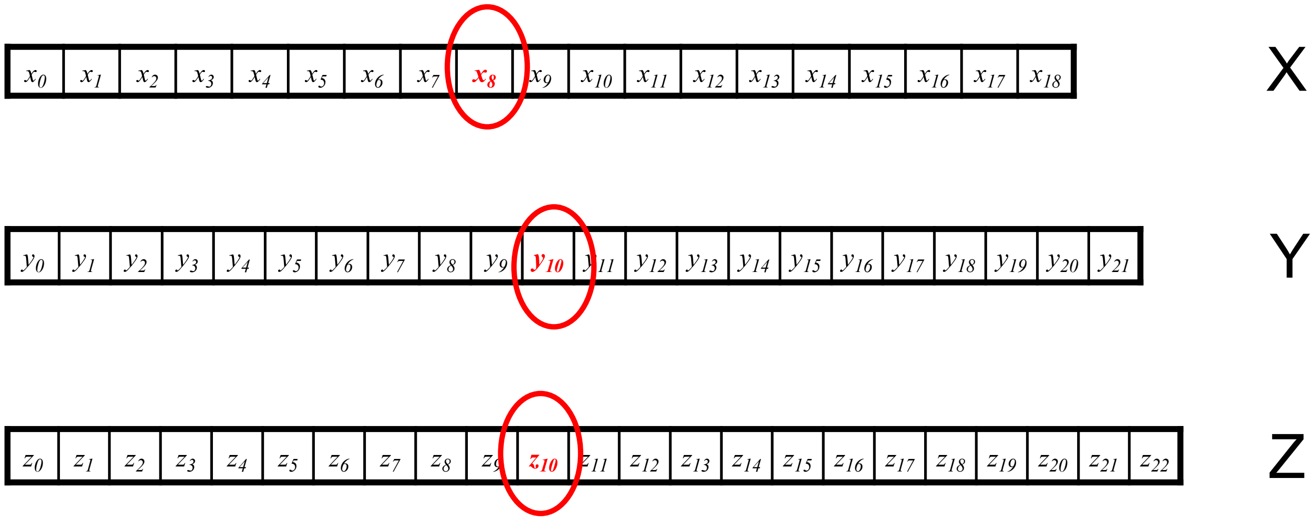
- **Input**: 64-bit secret key K_c, 22-bit *publicly known* Frame number
  - Each GSM transmission has a unique frame number
- **Output**: Two 114-bit blocks of keystream are generated to encrypt uplink and downlink data(2 directions)
- **Data**: 114 bit for each direction
- **Encryption**: Bitwise XOR

A5/1: Shift Registers

- A5/1 uses 3 ***linear feedback shift registers (LFSRs)***
  - X: 19 bits $(x_0, x_1, x_2, \ldots, x_{18})$
  - Y: 22 bits $(y_0, y_1, y_2, \ldots, y_{21})$
  - Z: 23 bits $(z_0, z_1, z_2, \ldots, z_{22})$



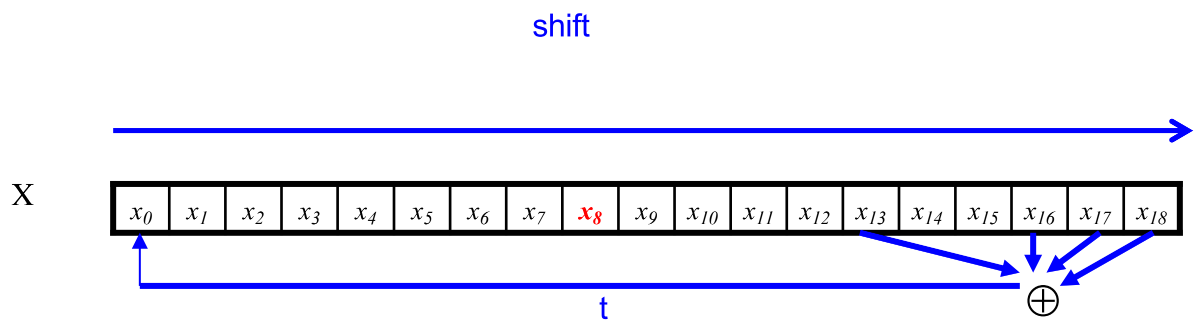
Majority of three clocking bits



- At each cycle: $m = \mathrm{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\mathrm{maj}(0,1,0) = 0$ and $\mathrm{maj}(1,1,0) = 1$

If $x_8 = m$

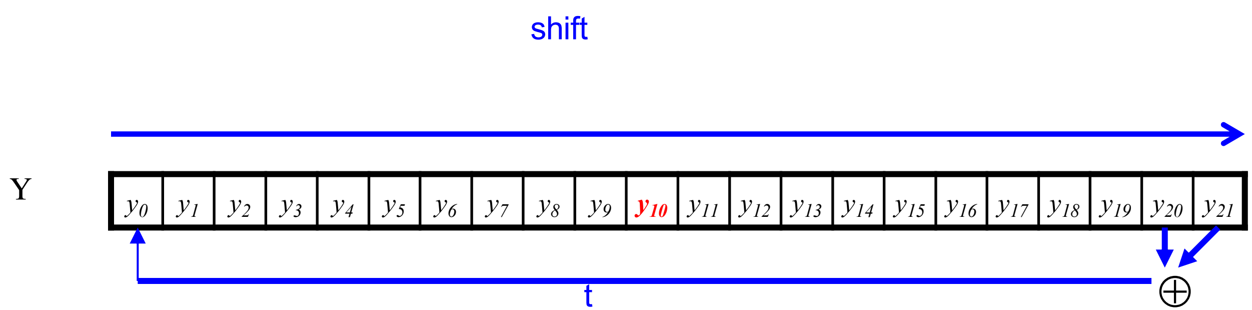
- If $x_8 = m$ then register X *steps* (or *clocks*) and the following series of operations occur
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$



Step(Clock): Advancing the state of the register

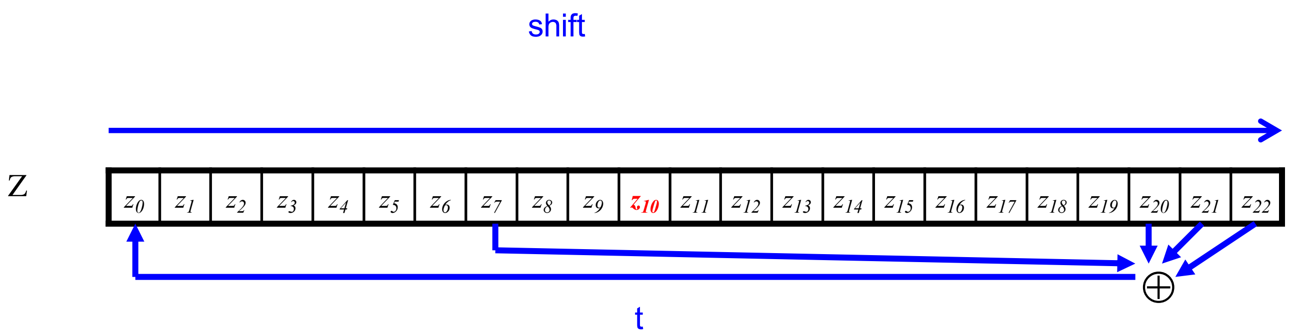
If $y_{10} = m$

- If $y_{10} = m$ then Y *steps (or clocks)*
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21, 20, \ldots, 1$ and $y_0 = t$

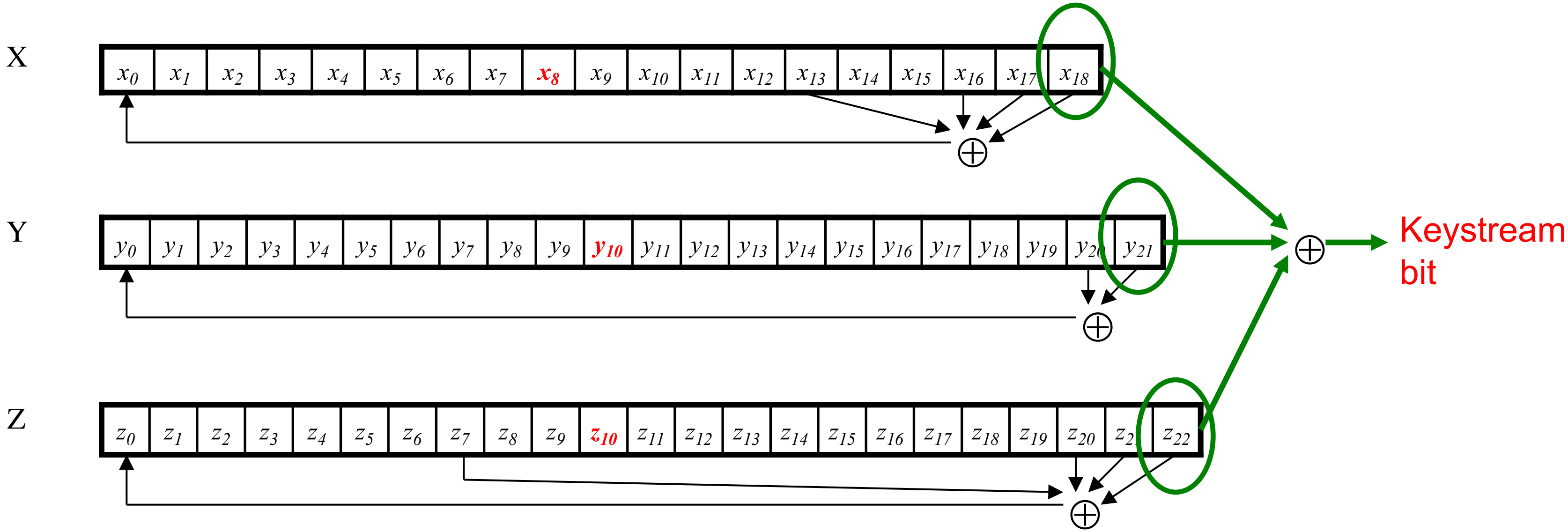


If $z_{10} = m$

- If $z_{10} = m$ then Z *steps (or clocks)*
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for $i = 22,21,\ldots,1$ and $z_0 = t$

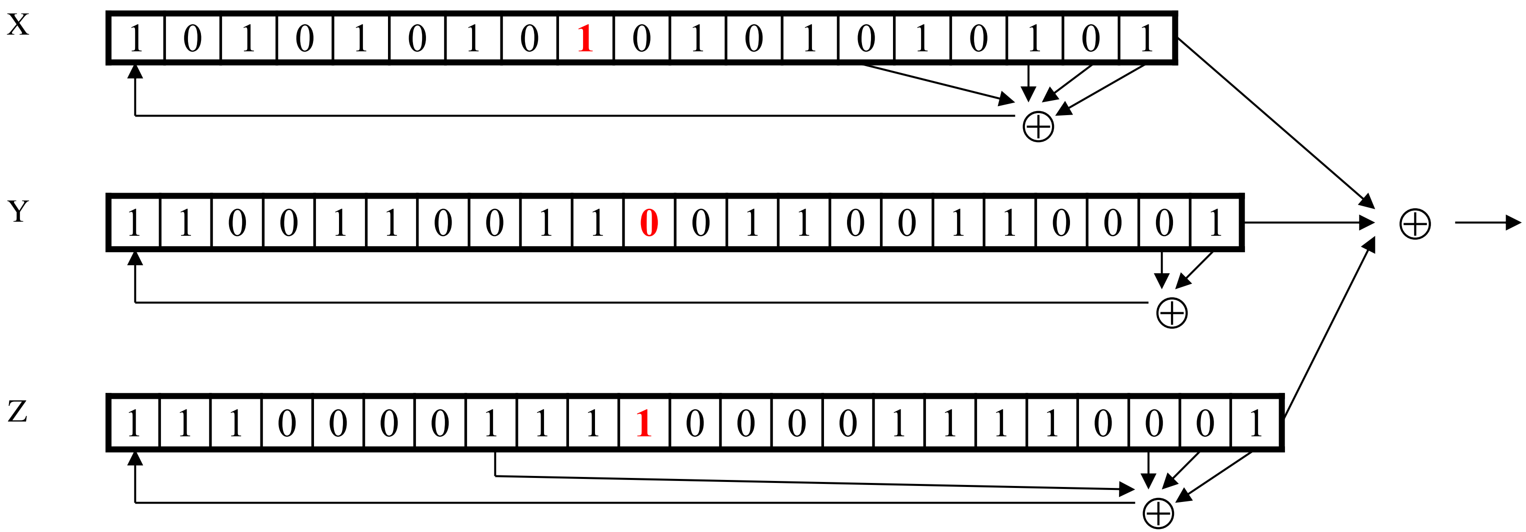


Keystream bit



- Keystream **bit** is $x_{18} \oplus y_{21} \oplus z_{22}$

A5/1 example



- In this example, $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(\mathbf{1},\mathbf{0},\mathbf{1}) = \mathbf{1}$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of most significant bits of registers
- Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$

BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK

A5/1 Initialization

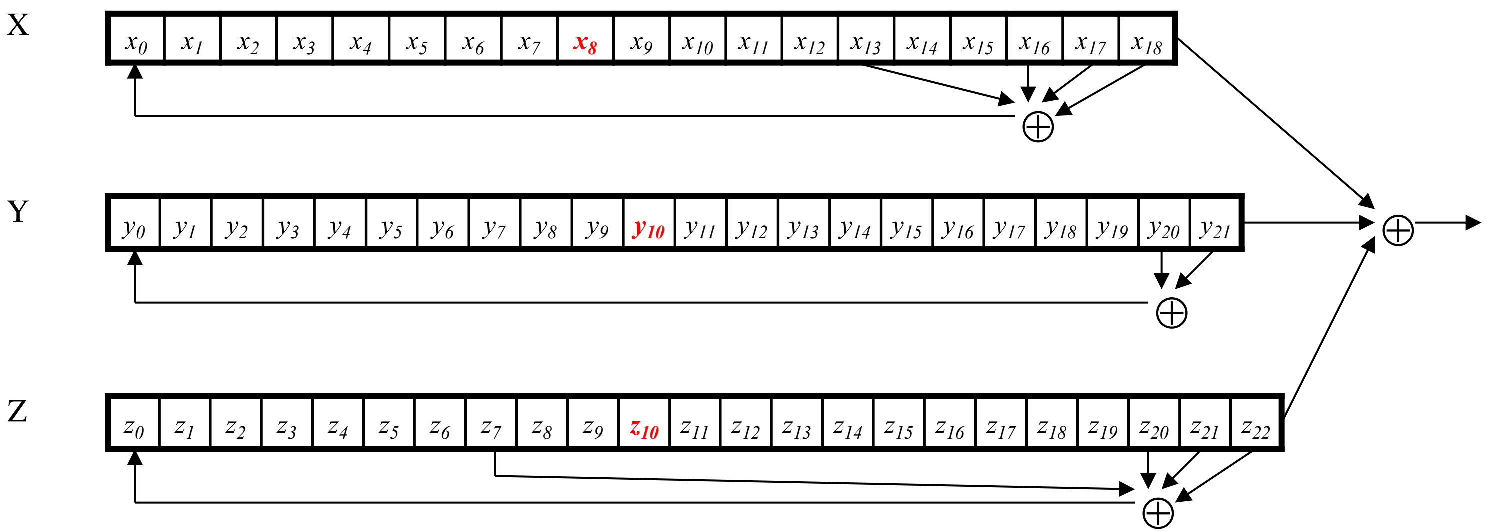
- All three registers are zero'ed
- Clock the three registers **regularly** for 64 cycles
  - During cycle `i`, XOR `ith` bit of key into the least significant bits of each 3 registers
- Clock **regularly** for another 22 cycles
  - During cycle `i`, XOR `ith` bit of frame number(22 bits) into least significant bits of each 3 registers
- 100 cycles are run using the majority clocking, the output is discard
- End result is the initial state

(regularly: forgets the majority rule, always clocks)

A5/1 cipher operation

- Clock **irregularly** for 228 cycles
  - output 2 114 bits keystream
  - First 114 bits for downlink, last 114 bits for uplink
- XOR the keystream with the data

A5/1 summary



- Each variable here is a single bit
- Key is used as **initial fill** of registers (64 bits)
- Each register clocks (or not) based on $\text{maj}(x_8, y_{10}, z_{10})$
- Keystream bit is XOR of the most significant bits of registers

Shift Register Crypto

- Shift register crypto efficient in hardware
- Often, slow if implement in software
- In the past, very popular
- Today, more is done in software due to fast processors
- Shift register crypto still used sometimes
  - Especially in resource-constrained devices

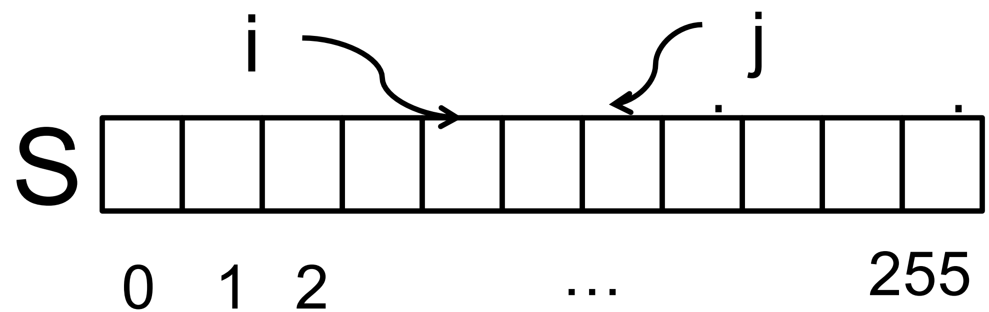
RC 4

RC4 - Background

- Was designed by Ron Rivest in 1987
  - Meaning: “Rivest Cipher 4”
  - There are other symmetric ciphers by Ron Rivest: RC2, RC5, and RC6 (block ciphers)
- Optimized for software implementation as every step produces a byte (rather than a bit by A5/1).
- Still used, although known to have issues

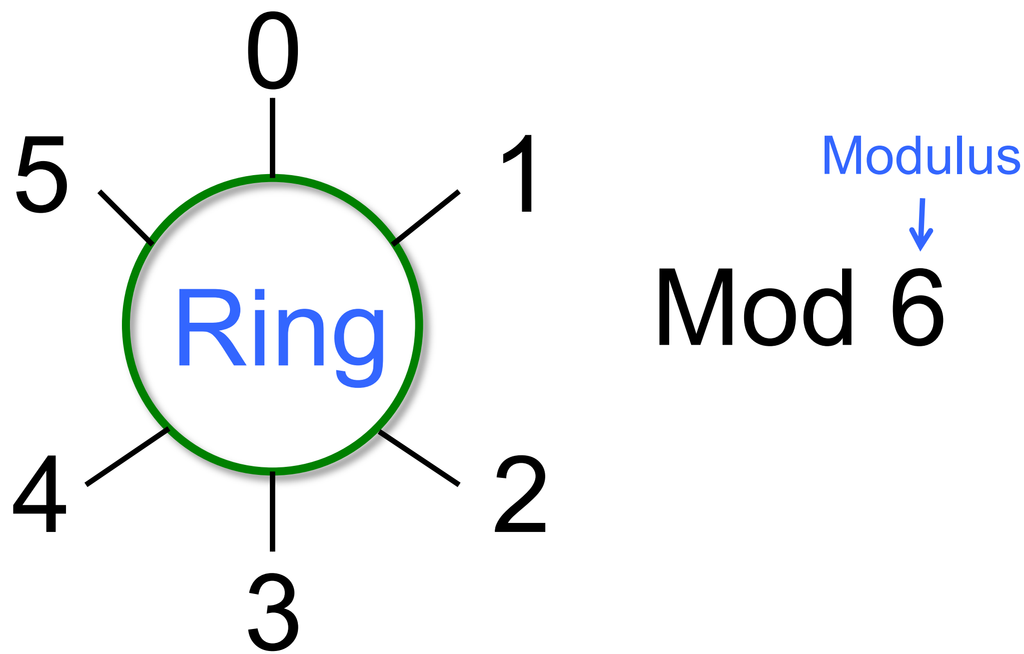
RC4

- Input: key[] contains N bytes of key
  - N typically from 5 to 16 bytes, leading to key length of 40 to 128 bits
- A *self-modifying* **lookup table** S, which always contains a permutation of the byte values 0,1,…,255, and is initialized with the key
  - Two 8-bit **index-pointers**
  - Pointer i, j



- At Each step, RC4 does the following
  - Swap elements in the lookup table S
  - Selects a keystream byte from table S
- Each Step produce a **byte**

Modular arithmetic - Refresh



- Modular addition:
  - ((a mod n) + ((b mod n)) mod n = (a + b) mod n

RC4 - Step 1: Initialization

- S[] is permutation of 0, 1,..., 255
- key[] contains N bytes of key

RC4 - Step 1: Initialization

- S[] is permutation of 0, 1,..., 255
- key[] contains N bytes of key

```
for i = 0 to 255
  S[i] = i
endfor
```

RC4 - Step 1: Initialization

- S[] is permutation of 0, 1,..., 255
- key[] contains N bytes of key

```
for i = 0 to 255
  S[i] = i
endfor
j = 0
for i = 0 to 255
  j = (j + S[i] + key[i mod N]) mod 256
  swap(S[i], S[j])
endfor
i = j = 0
```

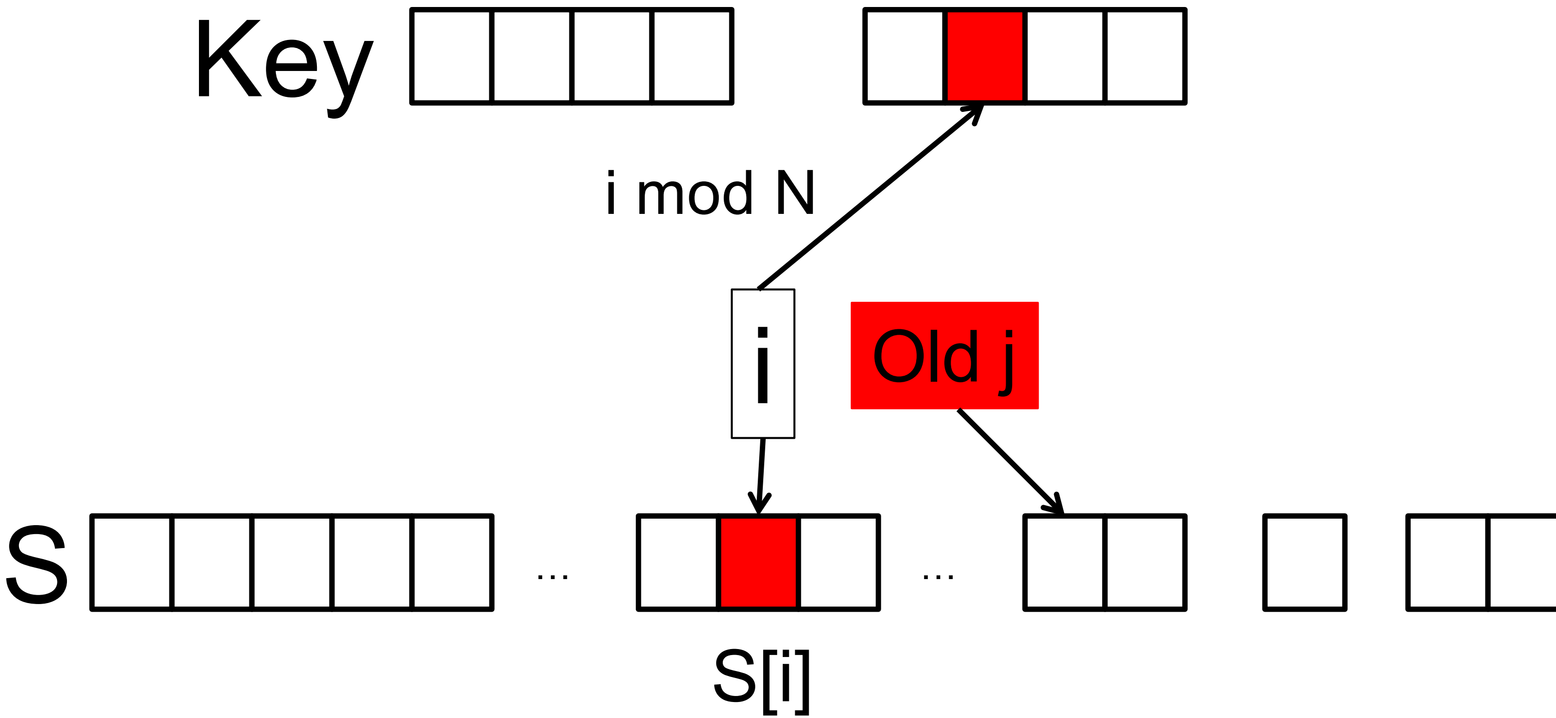
RC4 - Step 1: Initialization

- S[] is permutation of 0, 1,..., 255
- key[] contains N bytes of key

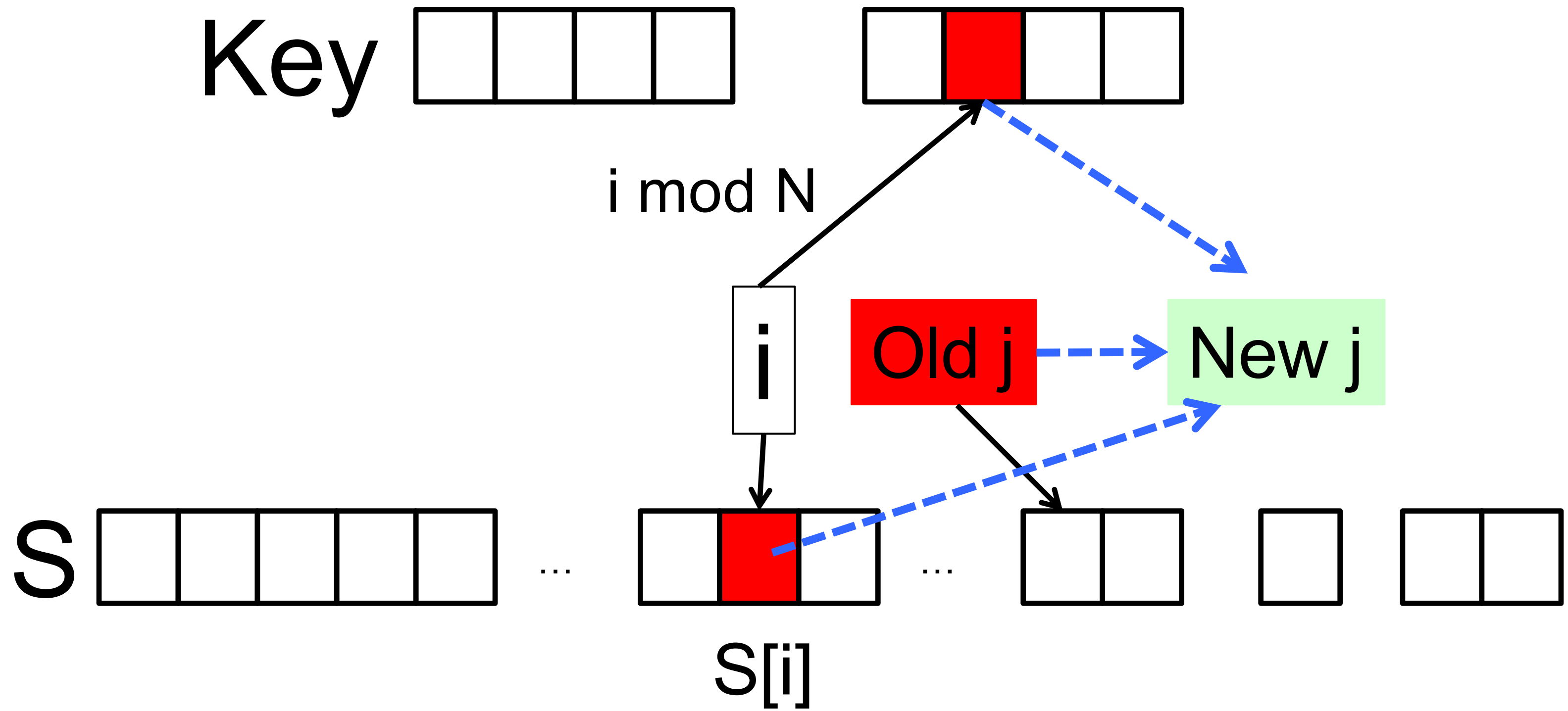
```
for i = 0 to 255
  S[i] = i
endfor
j = 0
for i = 0 to 255
  j = (j + S[i] + key[i mod N]) mod 256
  swap(S[i], S[j])
endfor
i = j = 0
```

Initialize the permutation in S

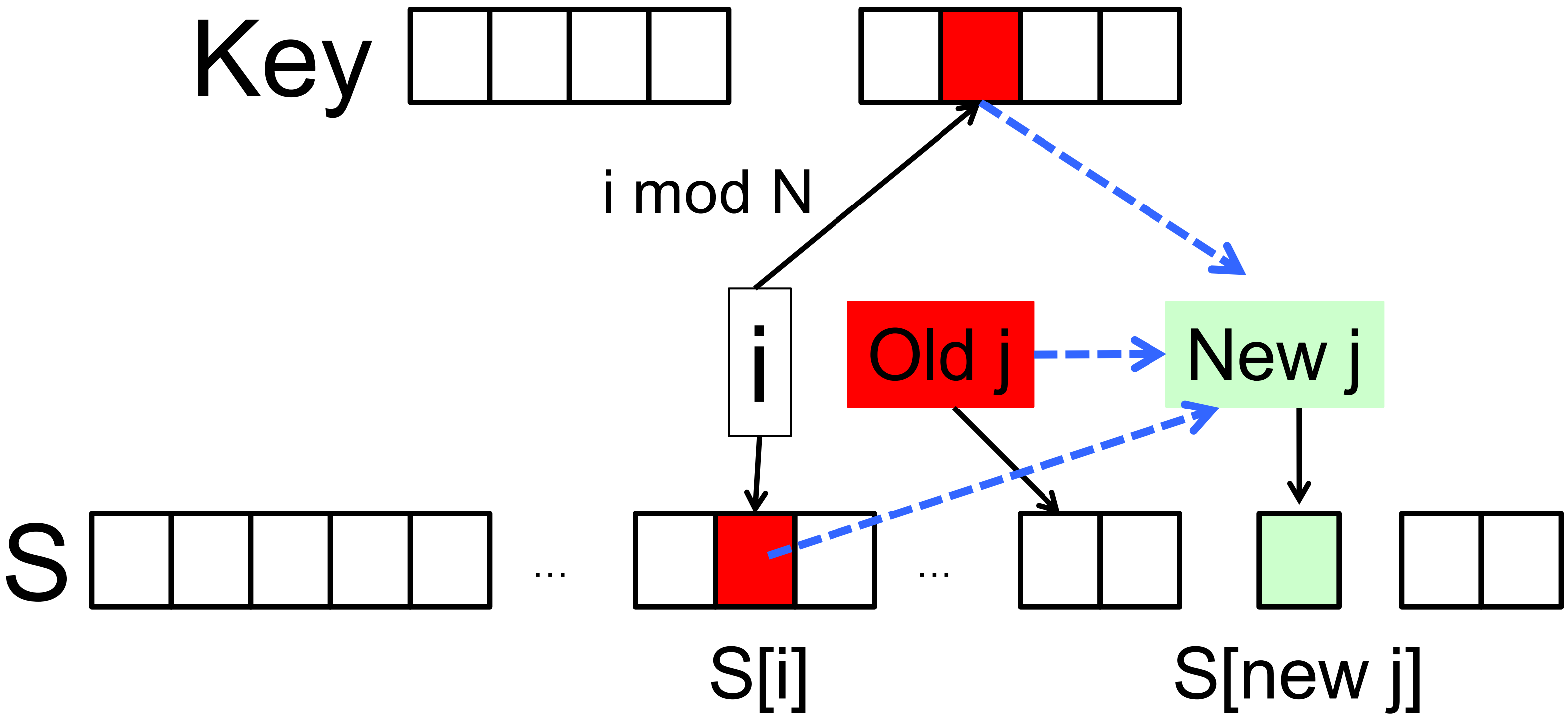
RC4 - After increasing i by 1 (modulo 256)



RC4 - After increasing i by 1 (modulo 256)

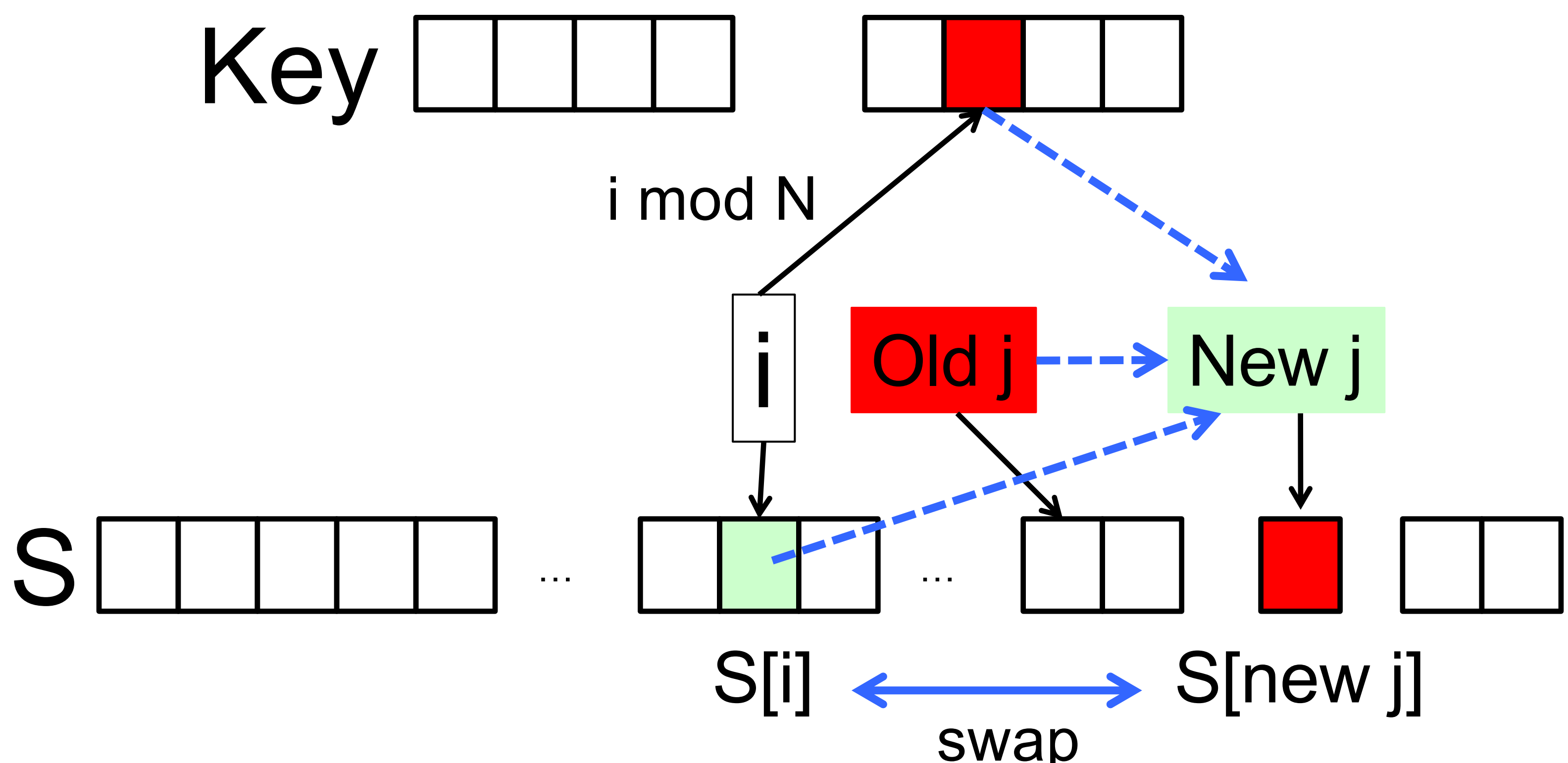


RC4 - Obtain S[new j]



BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

RC4 - Swap between S[i] and S[new j]



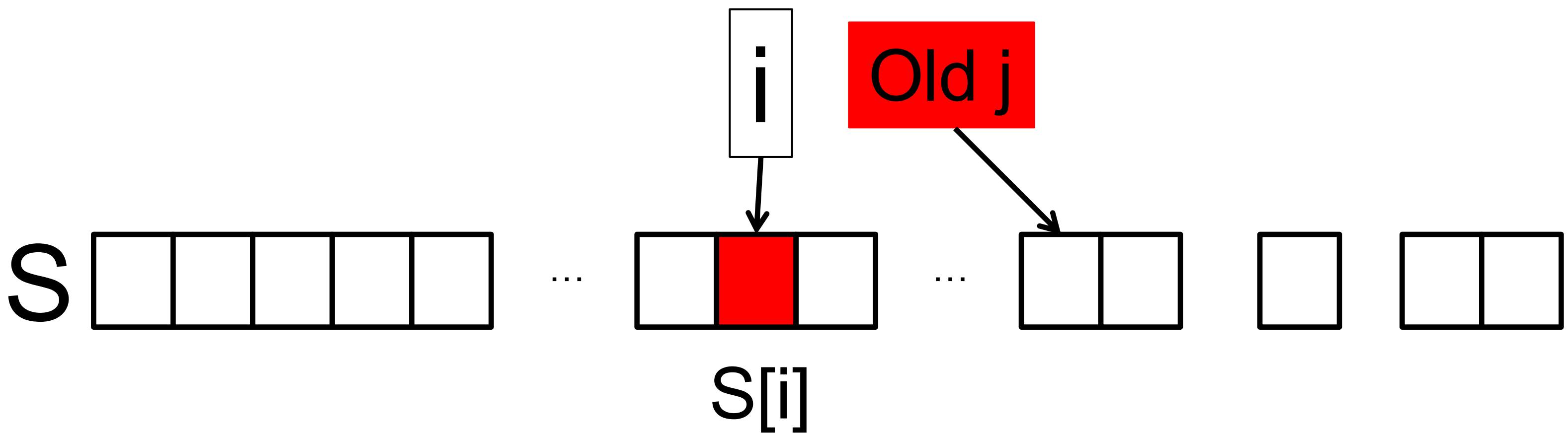
RC4 - Step 2: Pseudo-random generation algorithm

- For each keystream byte, swap elements in table and select byte

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap(S[i], S[j])
t = (S[i] + S[j]) mod 256
keystreamByte = S[t]
```

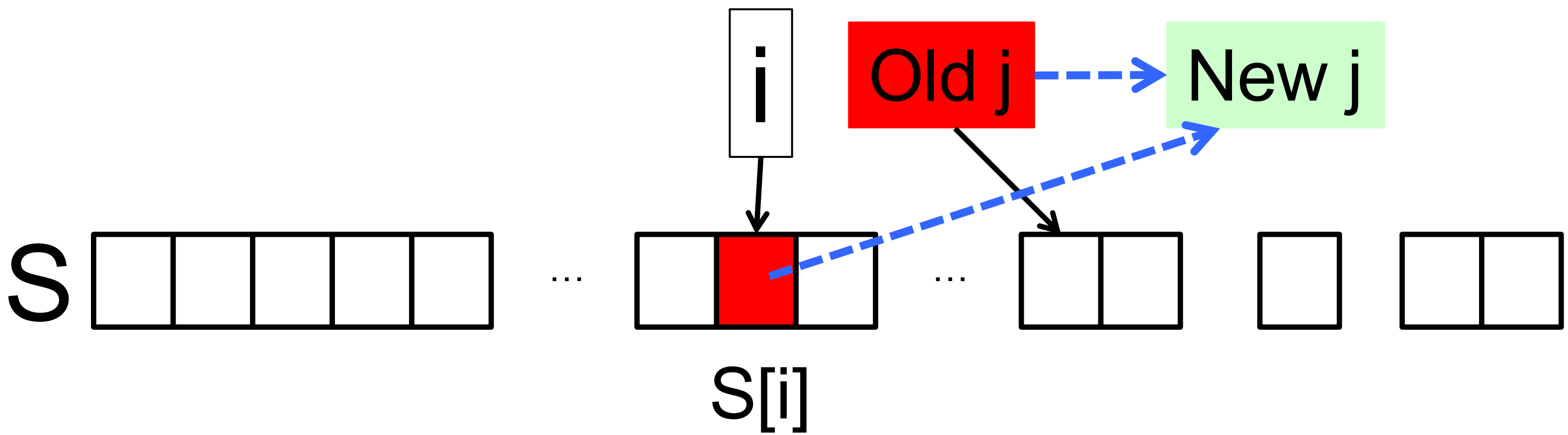
- **Note:** first 256 bytes should be discarded
  - Otherwise, related key attack exists

RC4 - Key[…] not used!



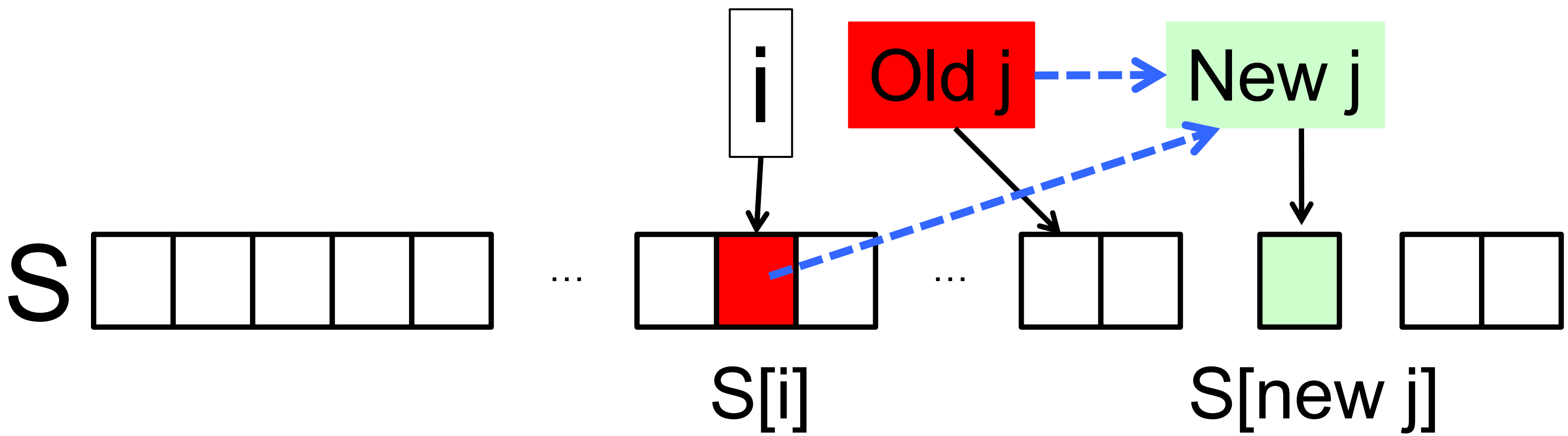
BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

RC4 - Key[…] not used!



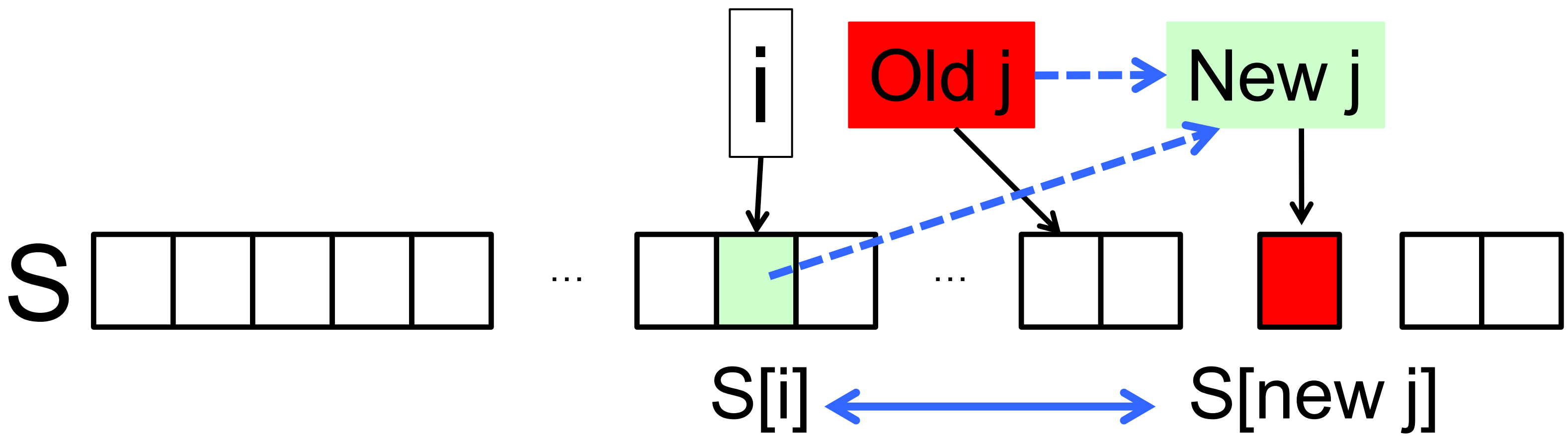
BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

RC4 - Obtain S[new j]

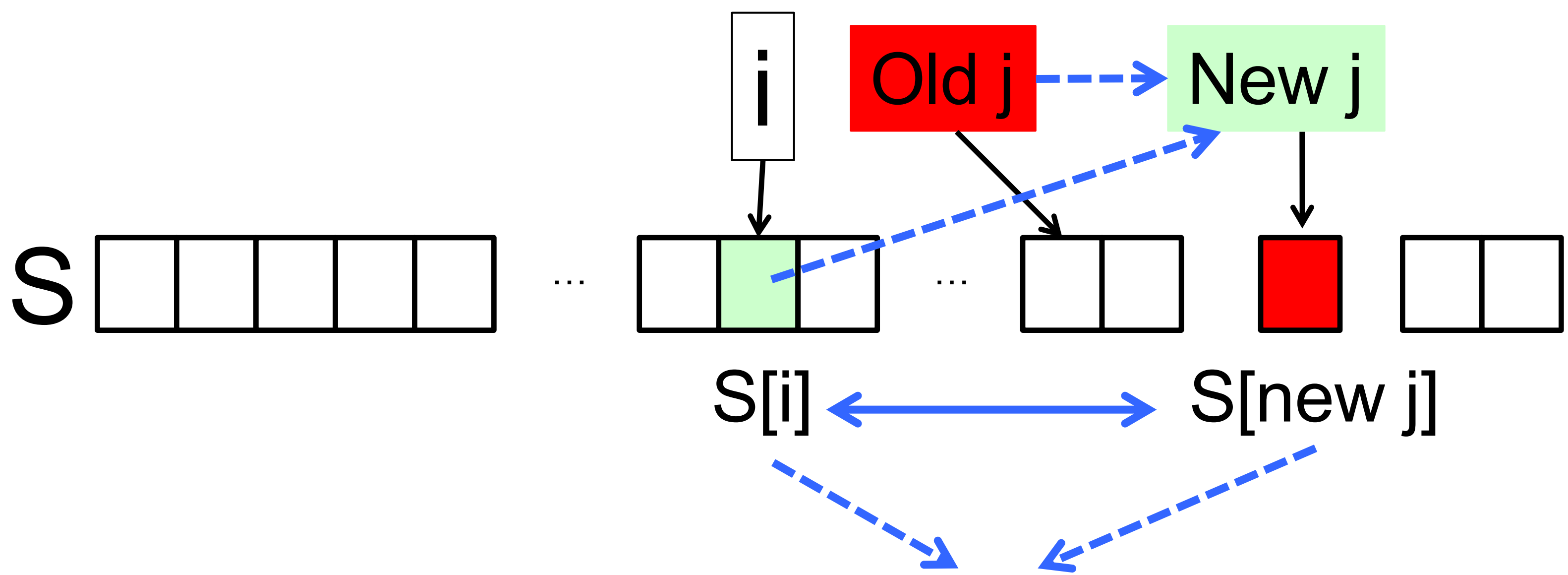


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

RC4 - Swap between S[i] and S[new j]

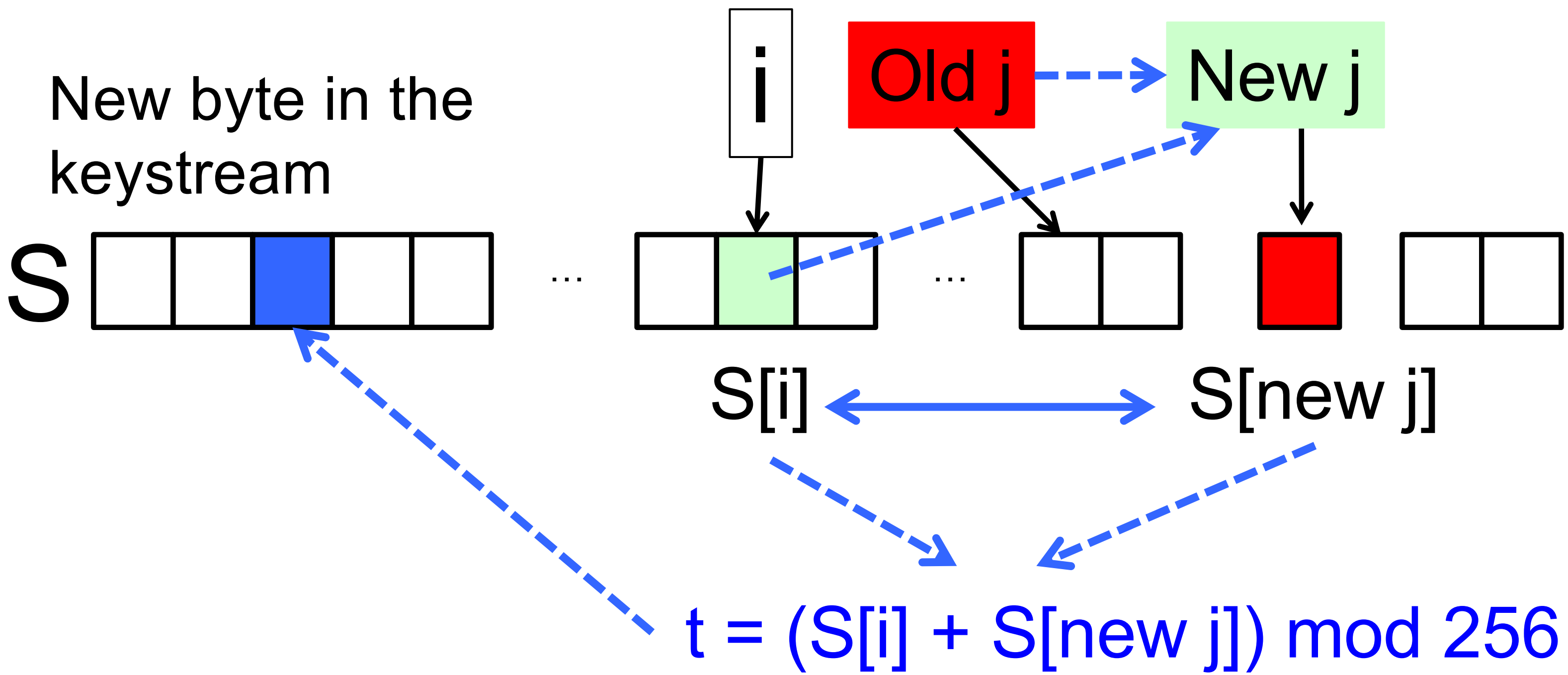


RC4 - Produce new byte



$t = (S[i] + S[new\ j]) \bmod 256$

RC4 - Produce new byte



RC4 - Summary

- Simple for **software implementation** as it is essentially just a lookup table S containing a permutation of all possible 256 byte values

- It was a “strong” cipher as each time a byte of keystream is produced, the lookup table is modified – a **moving target** for cryptanalyst – in such a way that it is always a permutation of all possible 256 byte values.

- Not deemed as secure any more.

RC4 fading out of modern browsers

Browser Makers To End RC4 Support In Early 2016

40

Posted by **Soulskill** on Wednesday September 02, 2015 @05:20AM from the out-with-the-old dept.

msm1267 writes:

> Google, Microsoft and Mozilla today announced they've settled on an early 2016 timeframe to permanently deprecate the shaky RC4 encryption algorithm in their respective browsers. Mozilla said Firefox's shut-off date will coincide with the release of Firefox 44 on Jan. 26. Google and Microsoft said that Chrome and Internet Explorer 11 (and Microsoft Edge) respectively will also do so in the January-February timeframe. Attacks against RC4 are growing increasingly practical, rendering the algorithm more untrustworthy by the day.

Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware(A5/1)
  - Speed was needed to keep up with audio, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
  - People declared “the death of stream ciphers”
  - Block ciphers are more popular now
  - But stream ciphers are still widely used