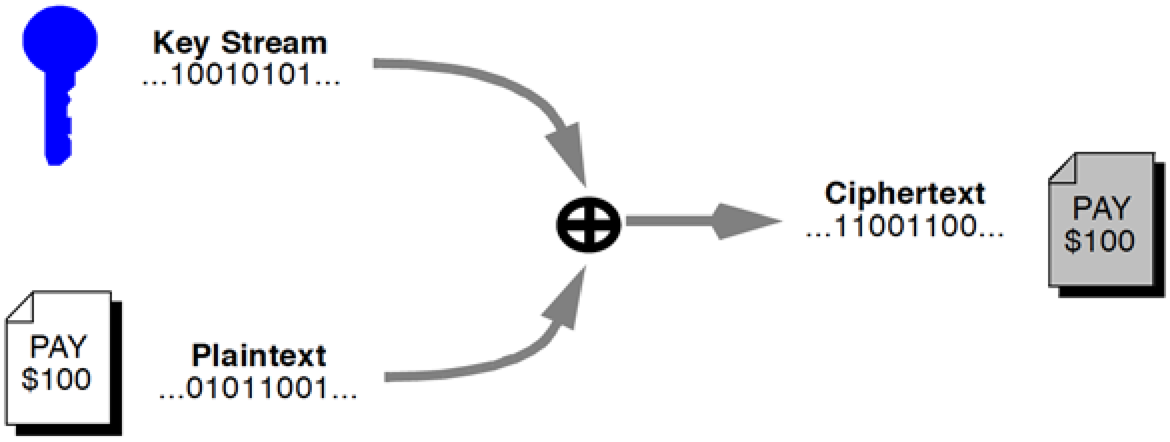
Stream Ciphers



Outline

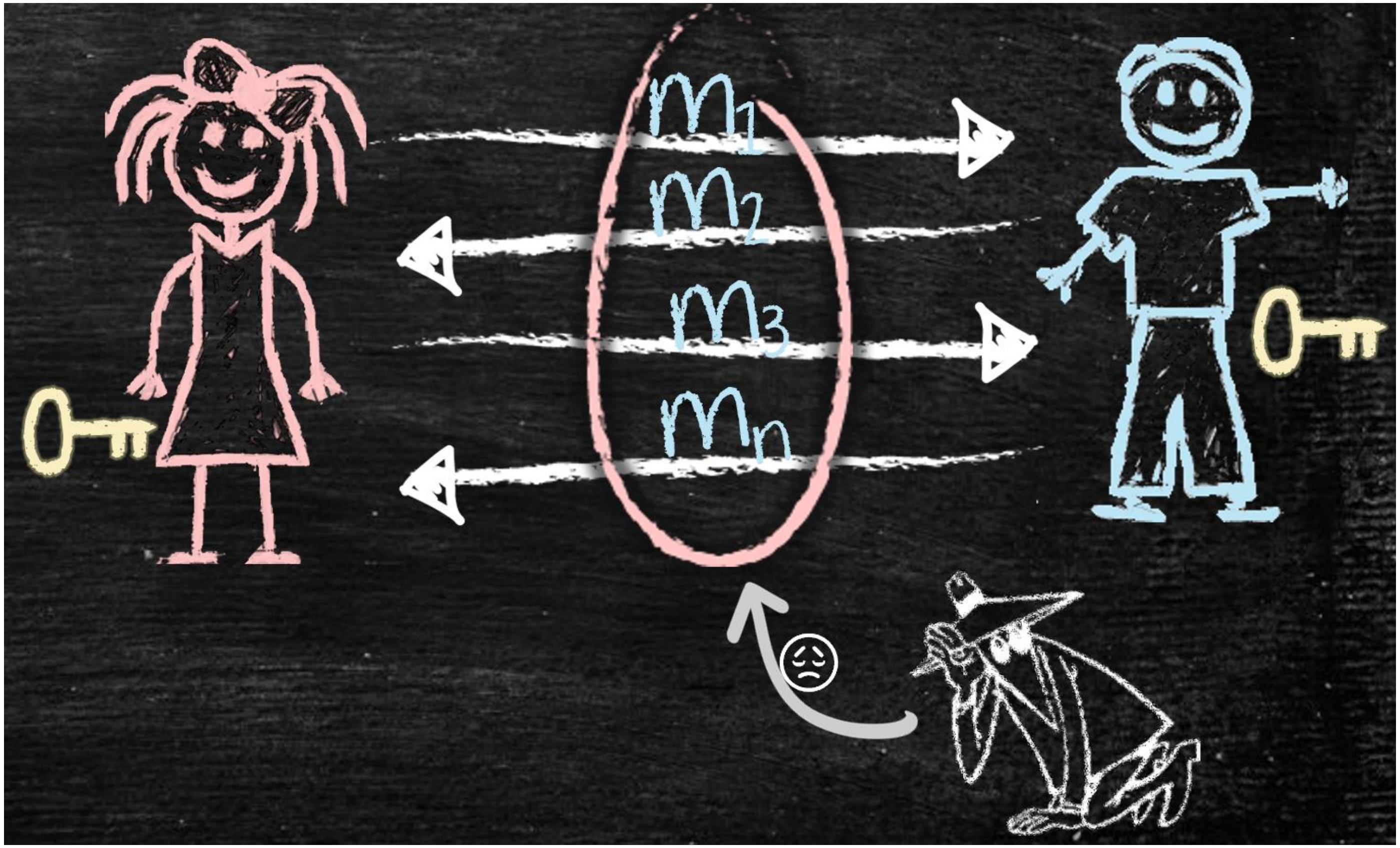
- Intro to stream ciphers
- Random Number Generators (RNGs)
- One-Time Pad (OTP)
- Linear Feedback Shift Registers (LFSRs)
- Stream Ciphers Examples
  - A5/1 Cipher
  - RC4 Cipher

Intro to stream ciphers

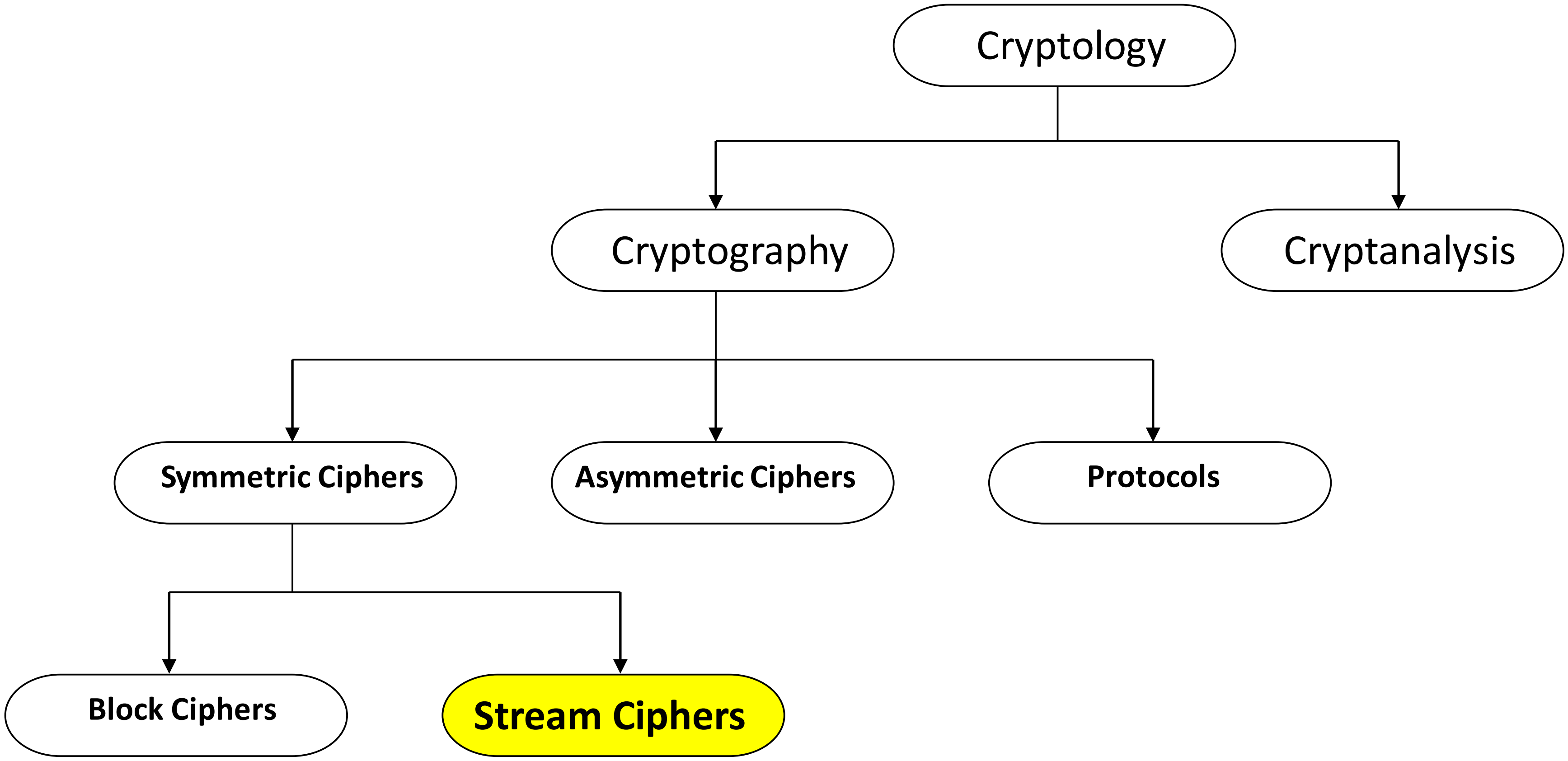
Back

Symmetric Key Cryptography

- A cryptographic technique where both parties in the communication share the same key

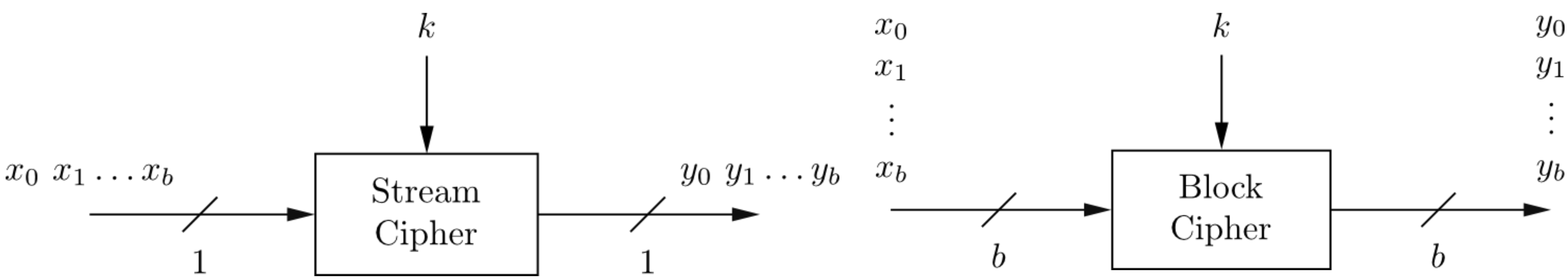


Stream Ciphers in the Field of Cryptology



Stream Ciphers were invented in 1917 by Gilbert Vernam

Stream Cipher vs. Block Cipher



- **Stream Ciphers**
  - Encrypt bits individually
  - Usually small and fast → common in embedded devices (e.g., A5/1 for GSM phones)
- **Block Ciphers:**
  - Always encrypt a full block (several bits)
  - Are common for Internet applications

Stream Ciphers

- Type of symmetric key crypto
- Use a fixed length key to produce a **pseudo-random stream of bits**
  - Same key gets you the same stream
- XOR those bits with your PT in order to encrypt
- XOR those same bits with your CT in order to decrypt
  - Inverting XOR is simple, since it is the same XOR operation
- Tries to approximate a one-time-pad

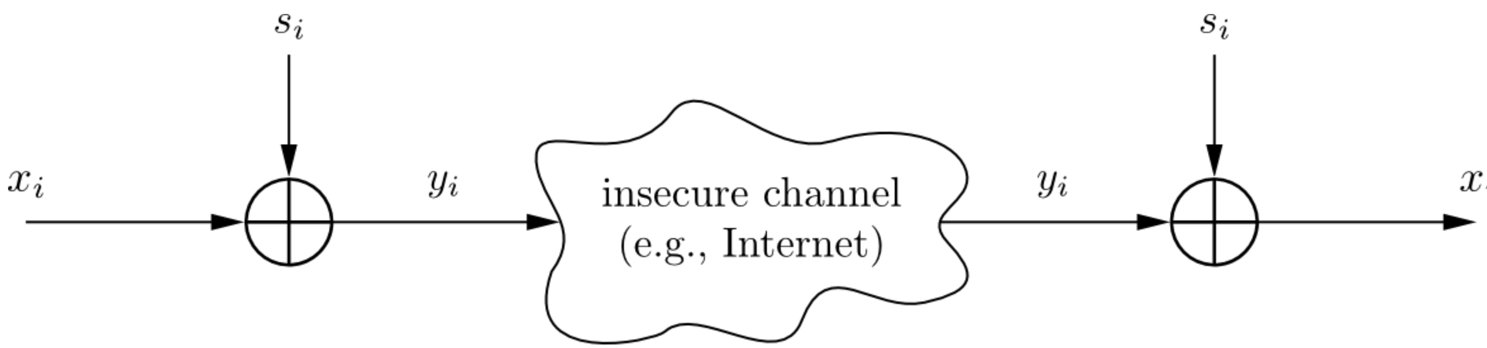
Real-Word Stream Ciphers

- RC4
  - Used in WEP for wireless network security
  - One option in TLS/HTTPS for encrypting web traffic
  - Not recommended for use anymore

- A5/1
  - Use for encrypting GSM phone data and conversations
  - NSA is known to be routinely breaking it

Encryption and Decryption with Stream Ciphers

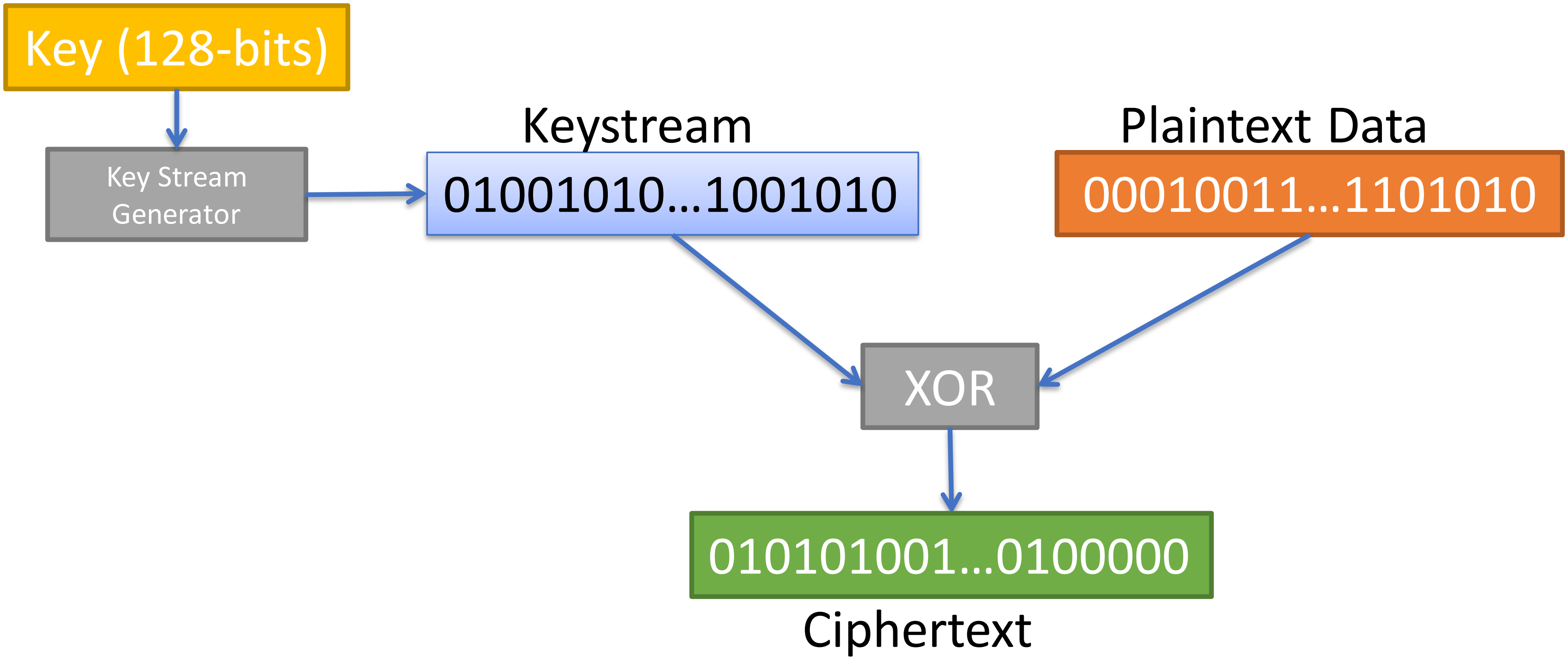
Plaintext $\boldsymbol{x_i}$, ciphertext $\boldsymbol{y_i}$ and key stream $\boldsymbol{s_i}$ consist of individual bits



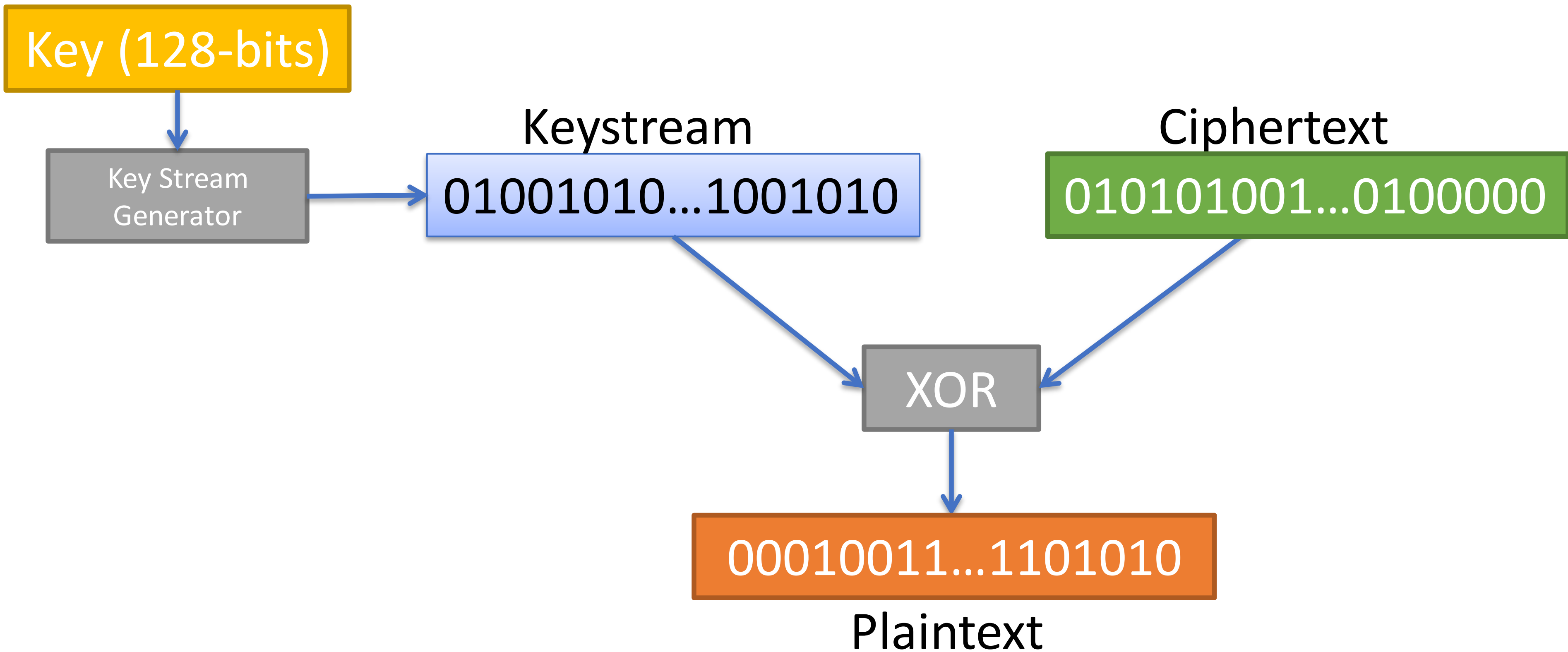
- Encryption and decryption are simple additions modulo 2 (aka XOR)
- Encryption and decryption are the same functions
- **Encryption:** $y_i = e_{si}(x_i) = x_i + s_i \bmod 2 \quad x_i, y_i, s_i \in \{0,1\}$
- **Decryption:** $x_i = e_{si}(y_i) = y_i + s_i \bmod 2$

$\mathbf{x_i}$ XOR	$\mathbf{s_i}$	$\mathbf{y_i}$
0	0	0
0	1	1
1	0	1
1	1	0

Stream Cipher Encryption Example



Stream Cipher Decryption Example



Why Does XOR Work Here?

- A few properties of XOR:

$$A \oplus A = 0$$
$$A \oplus 0 = A$$
$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

- Using XOR for encryption:

$$PT \oplus KEY = CT$$
$$CT \oplus KEY = PT$$
$$(PT \oplus KEY) \oplus KEY = PT$$
$$PT \oplus (KEY \oplus KEY) = PT$$
$$PT \oplus (0) = PT$$
$$PT = PT$$

XOR Example

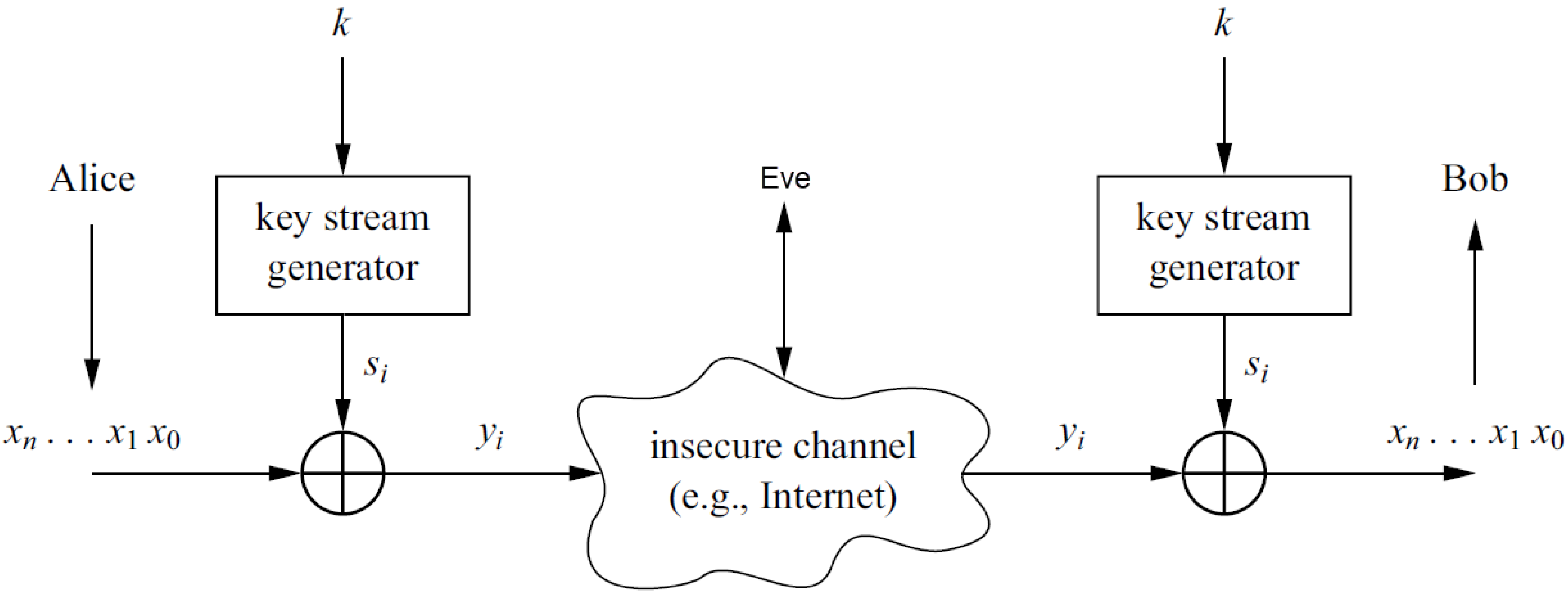
- Encrypt

Plaintext:	0110
Key:	1100
Ciphertext:	1010

- Decrypt

Ciphertext:	1010
Key:	1100
Plaintext:	0110

Key Stream Generator



- Security of stream cipher depends entirely on the key stream $\boldsymbol{s_i}$:
  - Should be **random** , i.e., $Pr(s_i = 0) = Pr(s_i = 1) = 0.5$
  - Must be **reproducible** by sender and receiver
- For perfectly random key stream s_i , each ciphertext output bit has a 50% chance to be 0 or 1
  → Good statistic property for the keystream

Stream Cipher: Throughput

Performance comparison of symmetric ciphers (Pentium4):

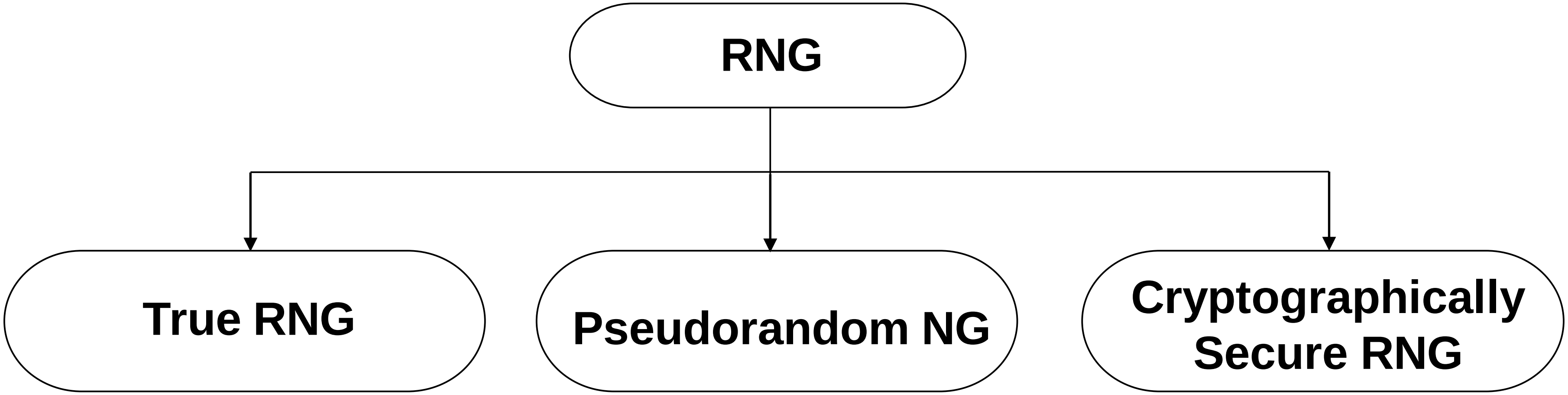
Cipher	Key length	Mbit/s
DES	56	36.95
3DES	112	13.32
AES	128	51.19
RC4 (stream cipher)	(choosable)	211.34

Source: Zhao et al., Anatomy and Performance of SSL Processing, ISPASS 2005

Random number generators (RNGs)

Back

Random number generators (RNGs)



- Randomness is found everywhere in cryptography: **generation of secret keys**, **encryption schemes**, **attacks on cryptosystems**
- Without randomness, cryptography would be impossible because all operations would become predictable, and therefore insecure

True Random Number Generators (TRNGs)

- Based on physical random processes: coin flipping, dice rolling, semiconductor noise, radioactive decay, mouse movement, clock jitter of digital circuits
- Output stream s_i should have good statistical properties: $Pr(s_i = 0) = Pr(s_i = 1) = 50\%$
- Output can neither be **predicted** nor be **reproduced**

Typically used for generation of keys, nonces (used only once values), one-time pads and for many other purposes

Generating Keys

- Secret keys are the crux of cryptographic security and should be randomly generated so that they are **unpredictable** and **secret**
- e.g., Use the OpenSSL toolkit to generate a random symmetric key by dumping pseudorandom bytes

```
openssl rand -hex 16
```

```
4d138b893c8b59c2363f5f3ddfc0ed55
```

- Generating asymmetric keys (e.g., web site's public key and its associated private key may be valid for years). It requires constructing a private key and its respective public key, ensuring that both satisfy all the necessary criteria

```
openssl genrsa 2048 -out example.key
```


```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA6OQ6RBPoQNdWw1n9xjZ6d0uMUz2lcNEIvdXKKpp+u+ScLeWa/iQ9gLX0NmHb5+6BWnkqb1exJ/ccZWWIISrDqtbwfMITtRlxyvSEm6hME1rgXVdYGsxbEX1wMUfDHMt3fWpAOHwtBBMZfT
6QH/3Ki5zw4GFMrrqrfc6/HHYl9nmXMsbdD2eTQxT9KX5qKgXF3pLRn79xlJ+vzoAITEyBJasltKEZnY/aKZS5PvXsMndtKsrNHo/RMDKVHj+YrZ8du3W+rIayMmUXgj5XwIr3Vcj626lPr53RVrUZuw2RvjtF
YcgvLp6/kR5wY/GVAtzJLUBQWGYanqV8U3sjFKZ7hwIDAQABAoIBAQCC7awEEd0EdwuED9en04DOCF7/fSzBXAM6FjLt+KU/JKrY8rhMIDujXKfK2wEyRtfZ8Tx2FCq4eSgxqP77BnuyU9wnh0RlD8T1gLn8eI
UsbG6uiygxCKo6Dvc//UQnsAuhKlaoascz/zjZPI7yLU/tZXHV812U7cTzmYO/kHyPL4GYQUV1t6ByQHefV5jOpaG0mL+/LM9BXC46PGt93kFWRNnIqEAc+nfOK4uk2phXQiZTfljoe+u8TGw1SDEsdj16h7VY
Olt+VihZ0PWwpZIUO2t9mgIR7UdsirRdA022BUrCjLllnmegZOH8fd2xen4e+a5ev5CONn0X4H99DaYhAoGBAP4/JyNEHvgh5ukL3klBD9prAPQDEpfRVIFDvFvuXF6Oyr6wtRARr9TB1YzaOTzA2t5R/YGWsx
/8suF77tSvgUbFGenzvnC/m/oPIECc+36JyCCGaFyDymjokQQi4AaclsmzC4GVmnDGu1l4Z49nCwbCUj/WbGyS6wel4Ah7sdkjAoGBAOp/X9FszuLSdtrkPRoAdNjl65s+Qk3BUwWykrfUWeuVaohUFxxho8ne
V+kMc43UUIF3k+2D/jyUsHmB4OHdjCLU3C3huSNmRHDpbDEWFgajnCIroywQSaEdcYlow/69kT674vofeuQoYQQtmpI+vWPbL8kFgWWwS+aOjNm+9+RNAoGAbpxLCqy4THtzWjAvpO8JVpz27THpBOOtQA+YAu
bQiNLWod9+5l7LgGRT49OM00GT9uT0xUwq4d+ucyrX3O/97iwR+hZw49x4n3G1NmcVgxeuWXggLx+nD70eWg5SKI43i8i5WMcdSbMsDuCzLVBHG8muvtcqlJHWbp1bulUWKsECgYAqNCB3naCifAR+IsPIcTqW
atL7jrn6KogZp2j2gtZoKu0QeinqprwGseiafe4yIXsXwgCyp6XTG2OjireBjxEQl8TE0TIz58z7kVcygmoWac3I7Jeib5AA2j6730ofB9kjNEeGgrHobEgBq399QKNvXFx+Tmpadd9DMHazHcaZpQKBgBAJsz
Z+LVDPV66p+fNW0m4NmaH4IRl97qyH+aINiI1ufqwcrAnHOe9y5mMbNCxGTZ8qUU4kR2XFtKwpN1yrpUJHAGrffu6befPbEnqJIKr+gDwvEBUlE4w96whZS/EVbhYX/w8gOP4j+ibTQewAnAuCiV64sdVbBPy0
AmJ5YJrP-----END RSA PRIVATE KEY-----
```


```
openssl rsa -in example.key -pubout
```

Pseudorandom Number Generator (PRNG)

- Many PRNGs exist such as **Linear Congruential Generator (LCG)**
  - Generate sequences from initial seed value
  - Output can be **reproduced** and can be **predicted**

The key stream is computed in a recursive way:

$$S_0 = seed$$

$$S_{i+1} = AS_i + B \bmod m$$

Unknowns *A*, *B and* S_o can be initialized using the key

Example: *rand() function* in ANSI C:

$$s_0 = 12345$$

$$s_{i+1} = 1103515245 s_i + 12345 \bmod 2^{31}$$

Most PRNGs have bad cryptographic properties but they are efficient and practical!

Cryptanalyzing LCG

Linear Congruential Generator (LCG):

$$S_0 = seed$$

$$S_{i+1} = AS_i + B \bmod m$$

- Can be **cracked** if 3 output are known, i.e. S_1, S_2 and S_3 by solving:

$$S_2 = AS_1 + B \bmod m$$

$$S_3 = AS_2 + B \bmod m$$

$$A \equiv (S_2 - S_3)/(S_1 - S_2) \bmod m$$

$$B \equiv S_2 - S_1(S_2 - S_3)/(S_1 - S_2) \bmod m$$

...directly reveals A and B. All S_i can be computed easily!

Bad cryptographic properties due to the linearity of most PRNGs

Cryptographically Secure PRNG (CSPRNG)

- Special PRNG with additional property:

Output must be **unpredictable**

More precisely: Given n consecutive bits of output s_i , the following output bits s_{n+1} cannot be predicted (in polynomial time)

- Needed in cryptography, in particular for stream ciphers
- Example algorithms:
  - The Blum Blum Shub algorithm
  - Other https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

One-Time Pad (OTP)

One-Time Pad (OTP)

One-Time Pad (OTP)

- A cryptosystem based on Vernam's stream cipher. Has these properties:

  Let the plaintext, ciphertext and key consist of individual bits

  $x_i, y_i, k_i \in \{0,1\}$

Encryption:	$e_{k_i}(x_i) = x_i \oplus k_i$
Decryption:	$d_{k_i}(y_i) = y_i \oplus k_i$

- OTP is **unconditionally** secure if and only if the (1) the **keystream k_i is truly random** and (2) k_i is **used only once!**
- **Unconditionally secure cryptosystem =** cannot be broken even with *infinite* computational resources
- But impractical because of difficulty of key distribution and management

One-Time Pad (OTP)

Unconditionally secure cryptosystem:

$$y_0 = x_0 \oplus k_0$$

$$y_1 = x_1 \oplus k_1$$

Every equation is a linear equation with two unknowns

$\Longrightarrow$ for every y_i are $y_i = 0$ and $y_i = 1$ equiprobable!

$\Longrightarrow$ This is true if k_0, k_1, ... are independent, i.e., all k_i have to be generated truly random

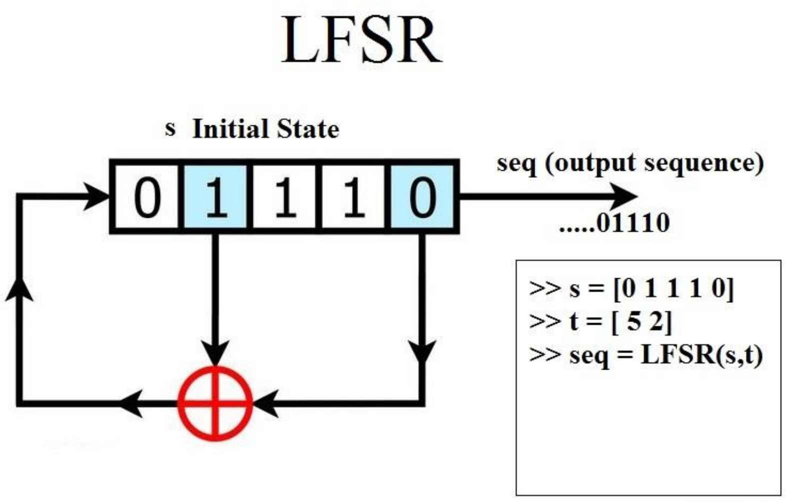
$\Longrightarrow$ It can be shown that this systems can *provably* not be solved

Disadvantage: For almost all applications the OTP is **impractical** since the key must be as long as the message! (Imagine you have to encrypt a 1GByte message)

Linear feedback shift registers (LFSRs)

Back

Linear Feedback Shift Registers (LFSRs)



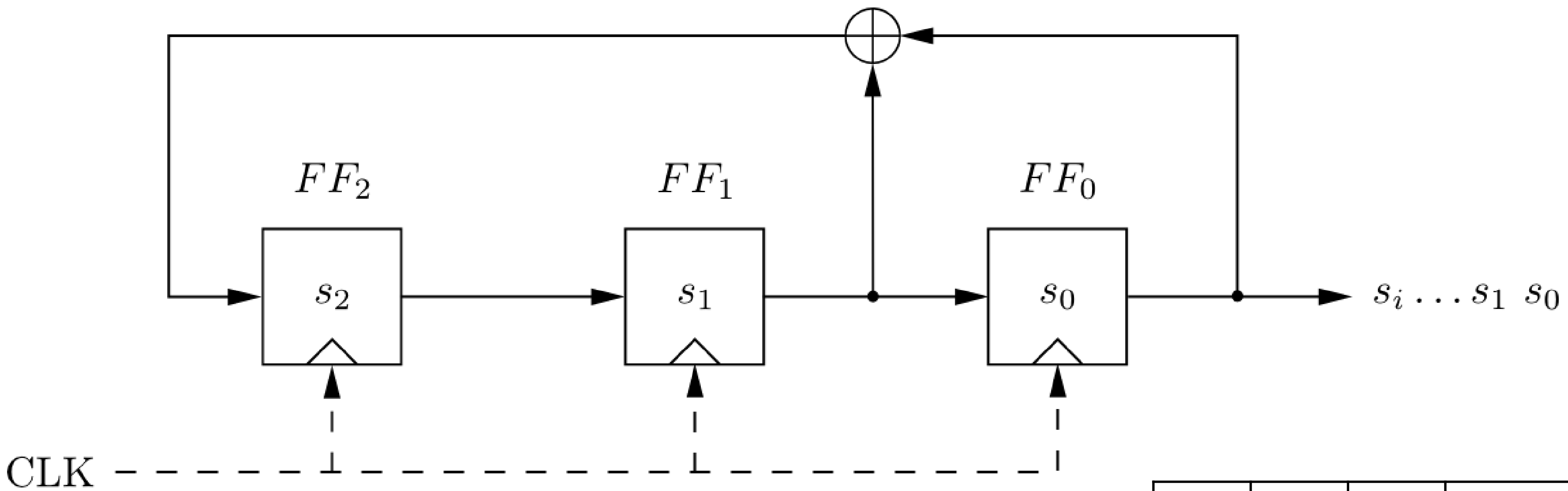
- Is a shift register whose input bit is a linear function (e.g., XOR) of its previous state:

  - shift all the bits one position to the right and

  - replace the vacated bit by *XOR* of certain bits of the new state

- Output sequence repeats periodically
- Feedback computes fresh input by XOR of certain state bits
- *Degree m* given by number of storage elements
- Maximum output length: 2^m-1

Linear Feedback Shift Registers (LFSRs): Example with m=3



- LFSR output described by recursive equation:

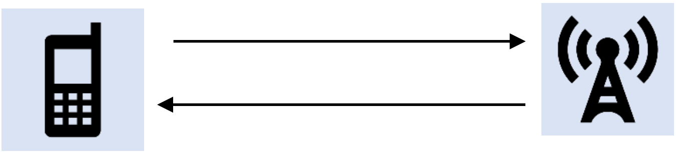
$$s_{i+3} = s_{i+1} + s_i \bmod 2$$

- Maximum output length (of 2^3-1=7)

- Vulnerable to attacks => many stream ciphers use **combinations** of LFSRs

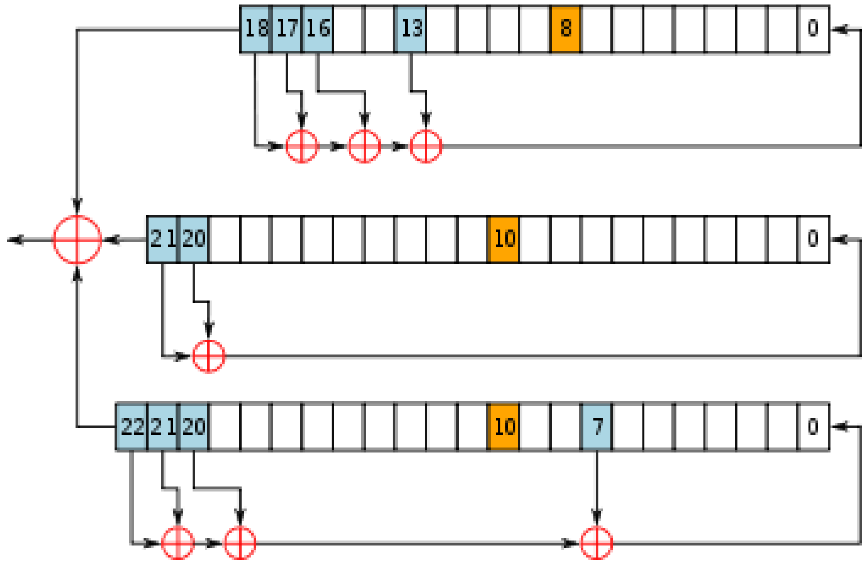
clk	*FF_2*	*FF_1*	*FF_0=s_i*
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0
8	0	1	0

A5/1 Cipher



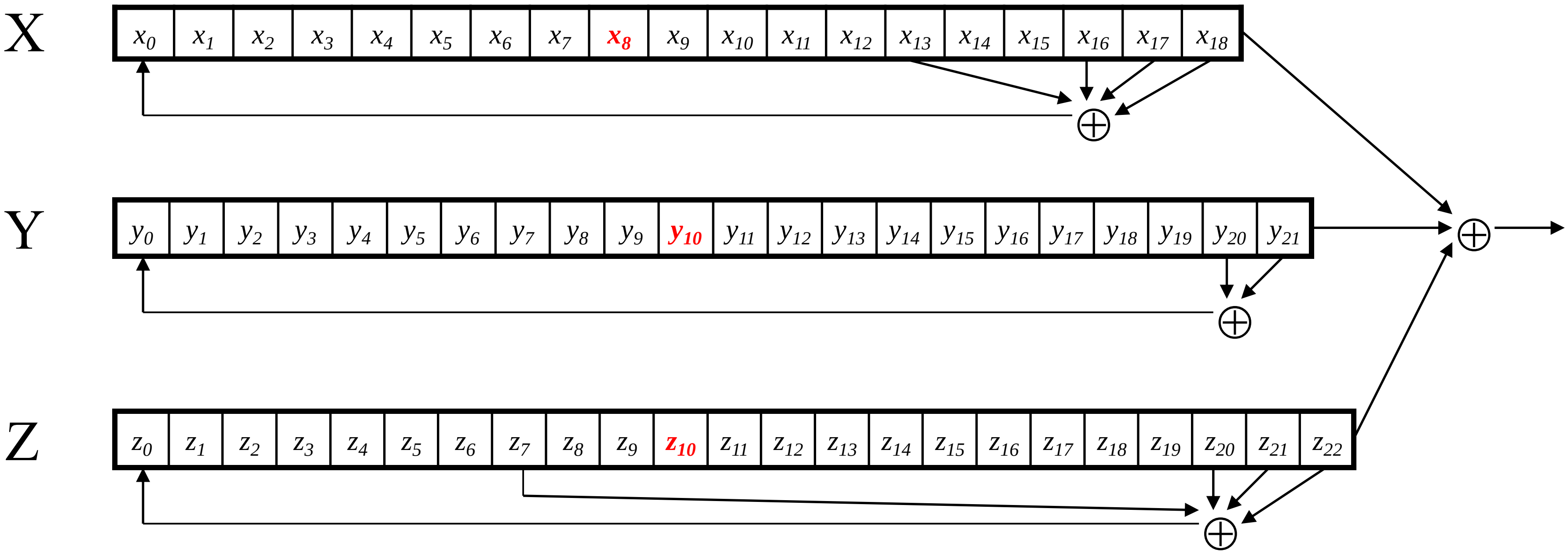
Back

A5/1 Stream Cipher



- A5/1 uses 3 LFSRs (X, Y, Z)
  - 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 $(z_0, z_1, z_2, \ldots, z_{22})$
- XOR-Sum of all three NLFSR outputs generates key stream s_i

A5/1 Keystream

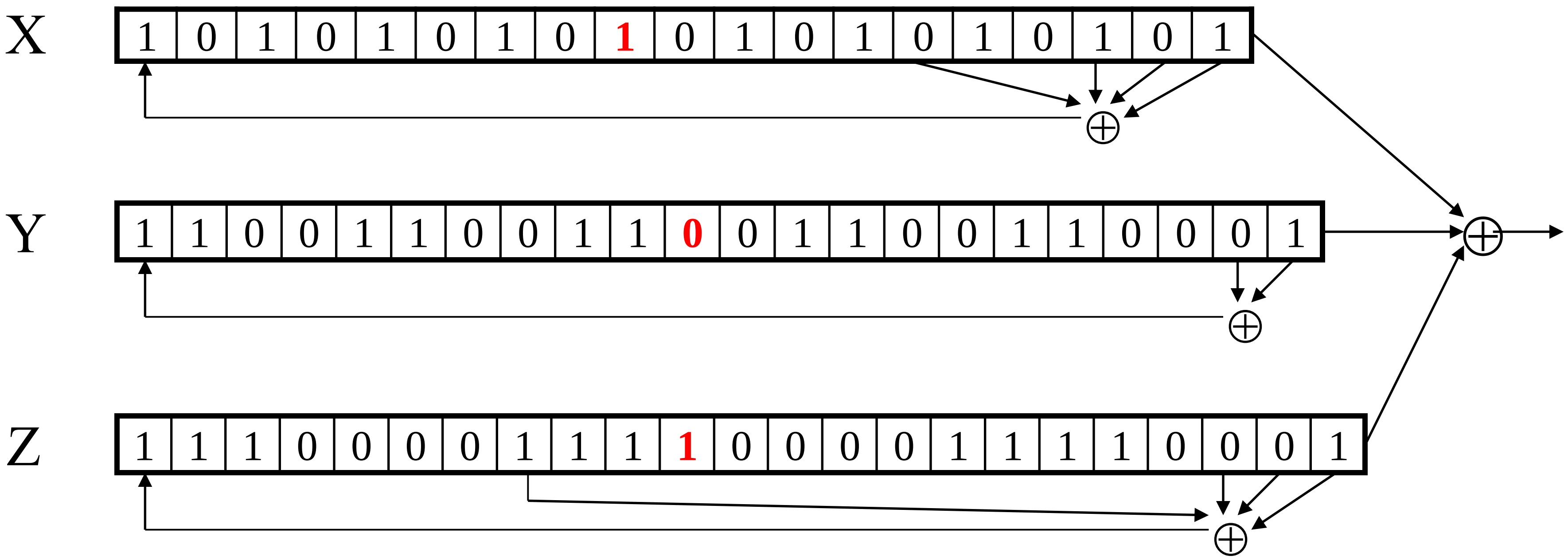


- Each variable here is a single bit
- Key is used as **initial fill** of registers
- Each register steps (or not) based on $\mathrm{maj}(x_8, y_{10}, z_{10})$
- Keystream bit is XOR of rightmost bits of registers

A5/1 Keystream

- At each step: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\text{maj}(0,1,0) = 0$ and $\text{maj}(1,1,0) = 1$
- If $x_8 = m$ then X *steps*
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y *steps*
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21,20,\ldots,1$ and $y_0 = t$
- If $z_{10} = m$ then Z *steps*
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for $i = 22,21,\ldots,1$ and $z_0 = t$
- Keystream **bit** is $(x_{18} \oplus y_{21} \oplus z_{22})$

A5/1



- 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 right bits of registers
- Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$

RC4 Cipher

Back

RC4

- Rivest Cipher 4 (RC4)
- A self-modifying lookup table
- Table always contains a permutation of the byte values $0,1,\ldots,255$
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream **byte** from table
- Each step of RC4 produces a **byte**
  - Efficient in software
- Each step of A5/1 produces only a bit
  - Efficient in hardware

RC4 Initialization

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

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

RC4 Keystream

- 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]
```

- Use keystream bytes like a one-time pad
- **Note:** first 256 bytes should be discarded
  - Otherwise, related key attack exists

Summary

- Stream ciphers produce a pseudo-random stream of bits that you XOR with PT to produce CT and vise-versa
- Stream ciphers require fewer resources and suitable for use in constrained environments such as cell phones (e.g., A5/1 cipher)
- The requirements for a *cryptographically secure PRNGs* are far more demanding than the those used in other applications such as testing or simulation
- The One-Time Pad is a provable secure symmetric cipher. However, it is highly impractical for most applications because the key length has to equal the message length and it must be used only once
- Single LFSRs make poor stream ciphers despite their good statistical properties. However, careful combinations of several LFSRs can yield strong ciphers