Introduction to stream ciphers

- **What is a stream cipher?**
- **Brief history of streams and ciphers**

  Let's dive in!

What is a stream cipher?

A stream cipher is a symmetric key cipher where:

- **each** plaintext digit is encrypted **one at a time** with the corresponding digit of the **keystream**, to give a digit of the ciphertext stream. - Wikipedia
- **Keystream**: a sequence of bits used to encrypt plaintext

1882 - One-time-pad

- invented by **Frank Miller** to encrypt **telegraph messages**
- the only **unbreakable** cipher
- highly **impractical** due to key management

1882 - Modern one-time-pad

- **Key**: a random sequence of bits
- the stream is **XORed** with the key
- the key is **as long as** the plaintext

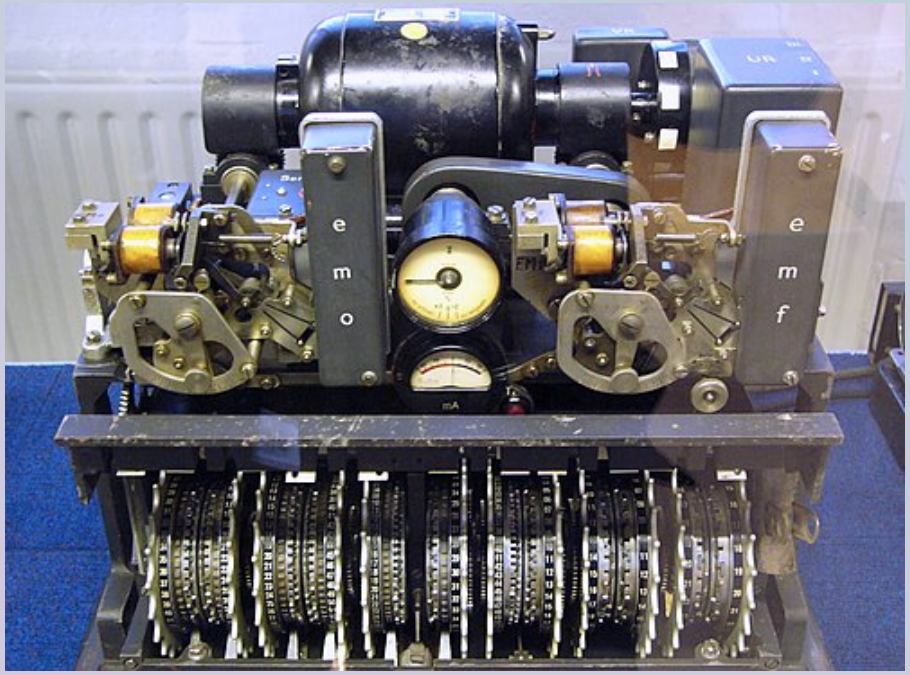
```
Message:     HELLO      → [01001000 01000101 01001100 01001100 01001111]
Key:         AGQZM      → [01100001 01000111 01010001 01011010 01001101]
Ciphertext:             → [00101001 00000010 00011101 00010110 00000010]
                        → [')'      \x02     \x1D     \x16     \x02   ] (bytes)
```

Modern stream ciphers

- **Key**: a random sequence of bits
- The stream is **XORed** with the **keystream**
- The keystream is **generated** from the key
- The keystream is now **pseudo-random**, thus it's not **perfectly secure**
- How to generate the most **secure keystream**?

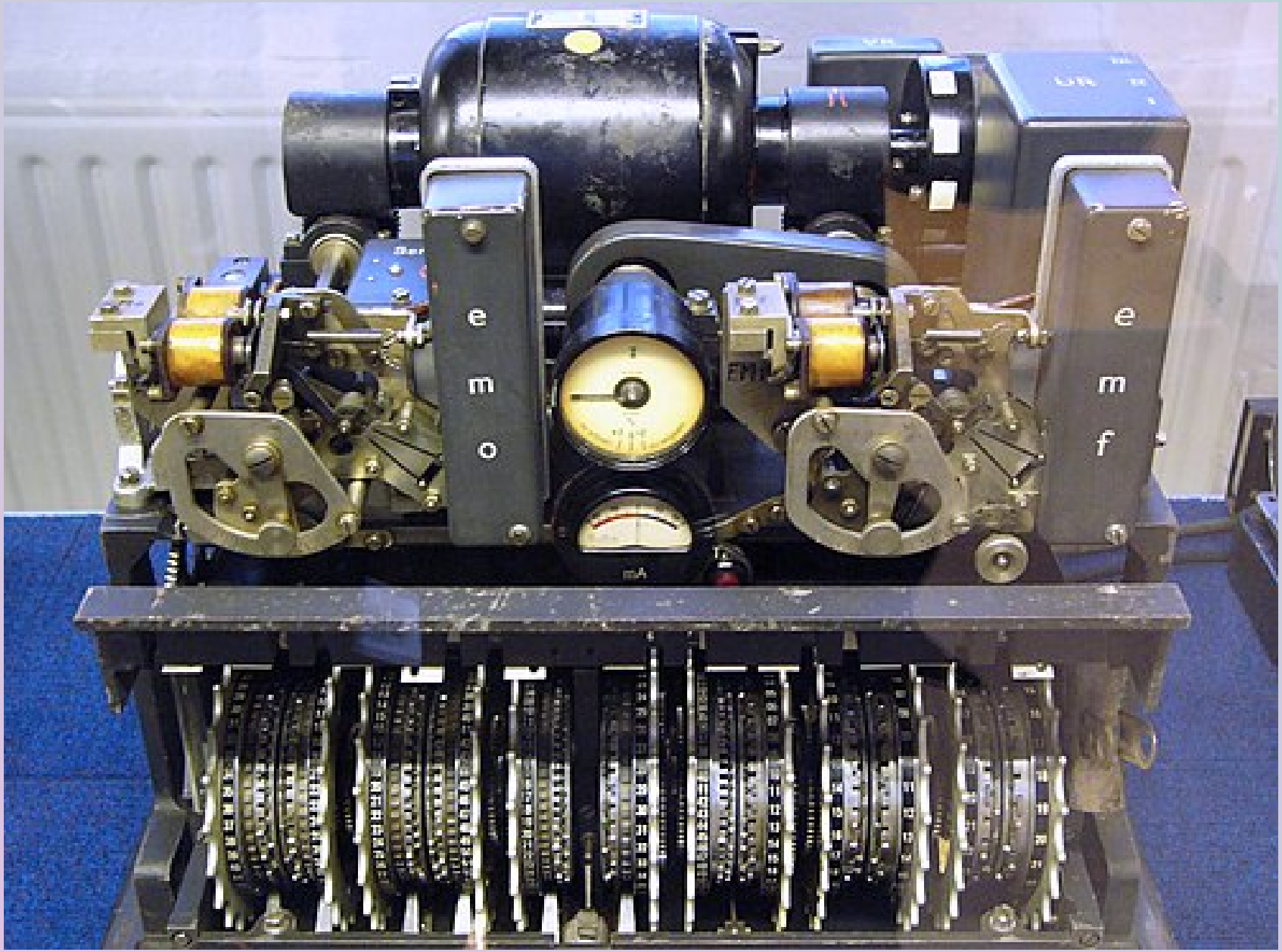
WWII - Lorenz-Schlüsselmaschine Schlüssel-Zusatz 40

- **Aka**: Lorenz SZ40 / Tunny
- used by the **German army** to encrypt RTTY messages
- **Synchronous** stream cipher



WWII - Lorenz SZ40 building blocks

- **2 rotors**, each with **5 wheels**
- χ rotors: 41, 31, 29, 26, 23
- ψ rotors: 43, 47, 51, 53, 59
- **2 motor wheels**: μ_{37}, μ_{61}
- **501+** "bit" key
- χ rotors shift every letter, ψ rotors shifts depending on the μ wheels
- **Ciphertext**: $M_i \oplus \chi_i \oplus \psi_i = C_i, i \in \{0 \ldots 4\}$


e
m
o
e
m
f
mA

WWII - Breaking the Lorenz SZ40

- **4000** characters of ciphertext were transmitted from **Athens** to **Vienna**
- the recipient asked (without encryption) for a **retransmission**,
- the sender **retransmitted** the slightly shorter message, **without changing the key**
- the **British** intercepted both messages

WWII - Breaking the Lorenz SZ40

- $P_1 \oplus K = C_1$
- $P_2 \oplus K = C_2$
- $C_1 \oplus C_2 = P_1 \oplus P_2$
- $C_1 \oplus C_2 \oplus P_1 = P_2$
- $C_1 \oplus P_1 = K$

WWII - Breaking the Lorenz SZ40

- **Bill Tutte** noticed a 41-bit pattern in November 1941
- By **January 1942**, the complete logical structure of the machine was known
- The first programmable computer, **Colossus**, was built to decode the Lorenz cipher
- The first Lorenz machine was captured in **1945**

Types of stream ciphers

Synchronous:

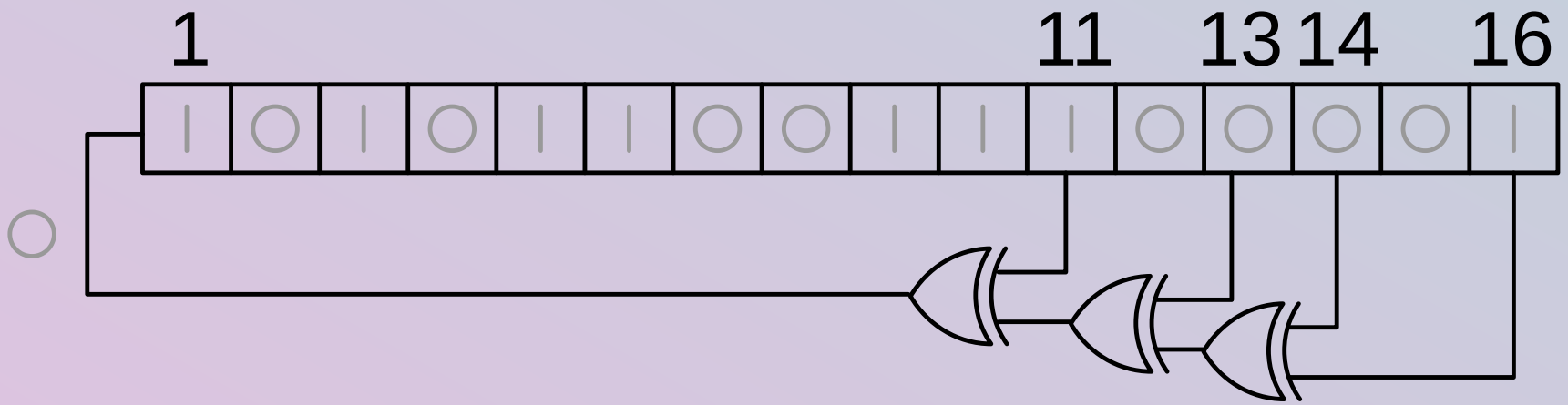
- the keystream is generated independently of the plaintext
- useless if 1 bit of keystream is lost

Self-synchronous:

- the keystream is generated from the previous ciphertext digits
- resistant to bit loss

Modern stream ciphers - building blocks

- **Linear Feedback Shift Register (LFSR)**: a shift register with feedback
- **Fibbonacci LFSR**: a type of LFSR where the feedback is taken from multiple bits

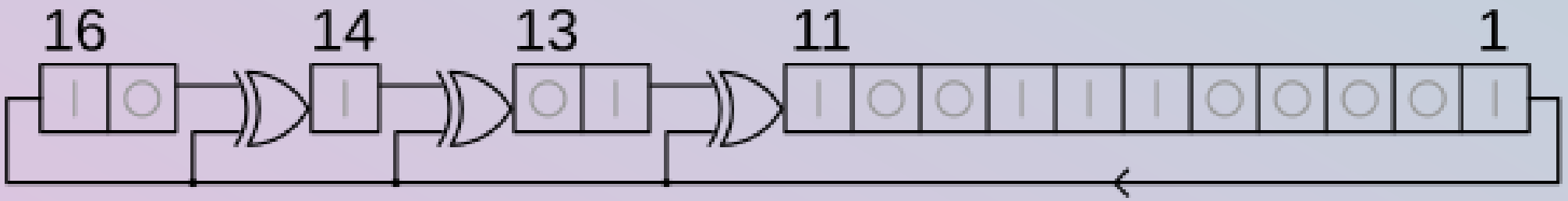


Modern stream ciphers - Fibonacci LFSR

- expressed as a polynomial mod 2
- $x^{16} + x^{14} + x^{13} + x^{11} + x^0 = x^0$
- The longest period is $2^{16} - 1 = 65535$
- This period can archieved by using a **primitive polynomial** over the Galois field GF(2^N)
  - The number of taps must be **even**
  - The taps must be **co-prime**

Modern stream ciphers - Galois LFSR

- Fibonacci LFSR cannot be **parallelized**
- The output is **the same** as with Fibonacci LFSR
- One round takes **1 clock cycle**

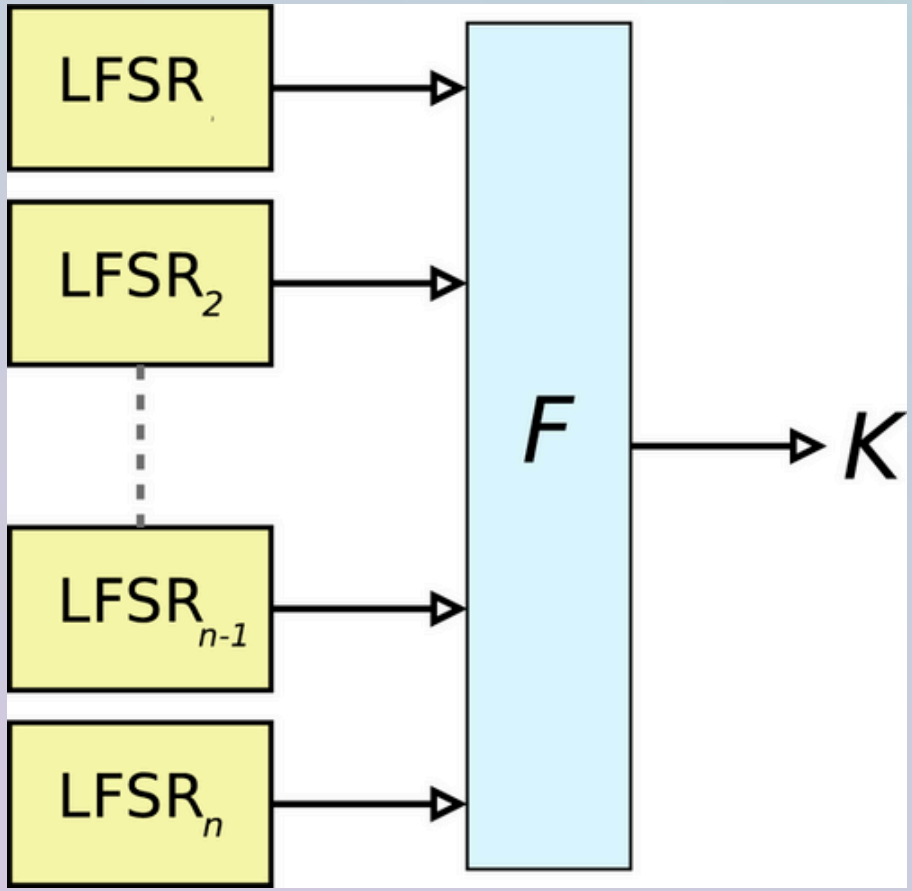


Non-linear feedback shift register (NLFSR)

- Safer than LFSR
- Uses a non-linear function to change the state
- We don't know how to build them...

Non-linear combining functions (NLCF)

- Fake a NLFSR
- Use a non-linear function to combine the bits of the LFSR



Clock-controlled generators

- Now we shift, now we don't

RC4

- Invented by **Ron Rivest** (**R** in **RSA**) in 1987
- Leaked in 1994
- Starts with a **key-scheduling algorithm (KSA)** to generate the initial state
- Then a **pseudo-random generation algorithm (PRGA)** is used to generate the keystream

RC4 - Key-scheduling algorithm (KSA)

```
for i from 0 to 255
    S[i] := i

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

RC4 - Pseudo-random generation algorithm (PRGA)

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    t := (S[i] + S[j]) mod 256
    K := S[t]
    output K
```

Use of RC4

- **WEP/WPA** (1997 - 2004)
- **SSL/TLS** (1999 - 2015)
- OpenBSD /dev/random (? - 2015)

Problems with RC4

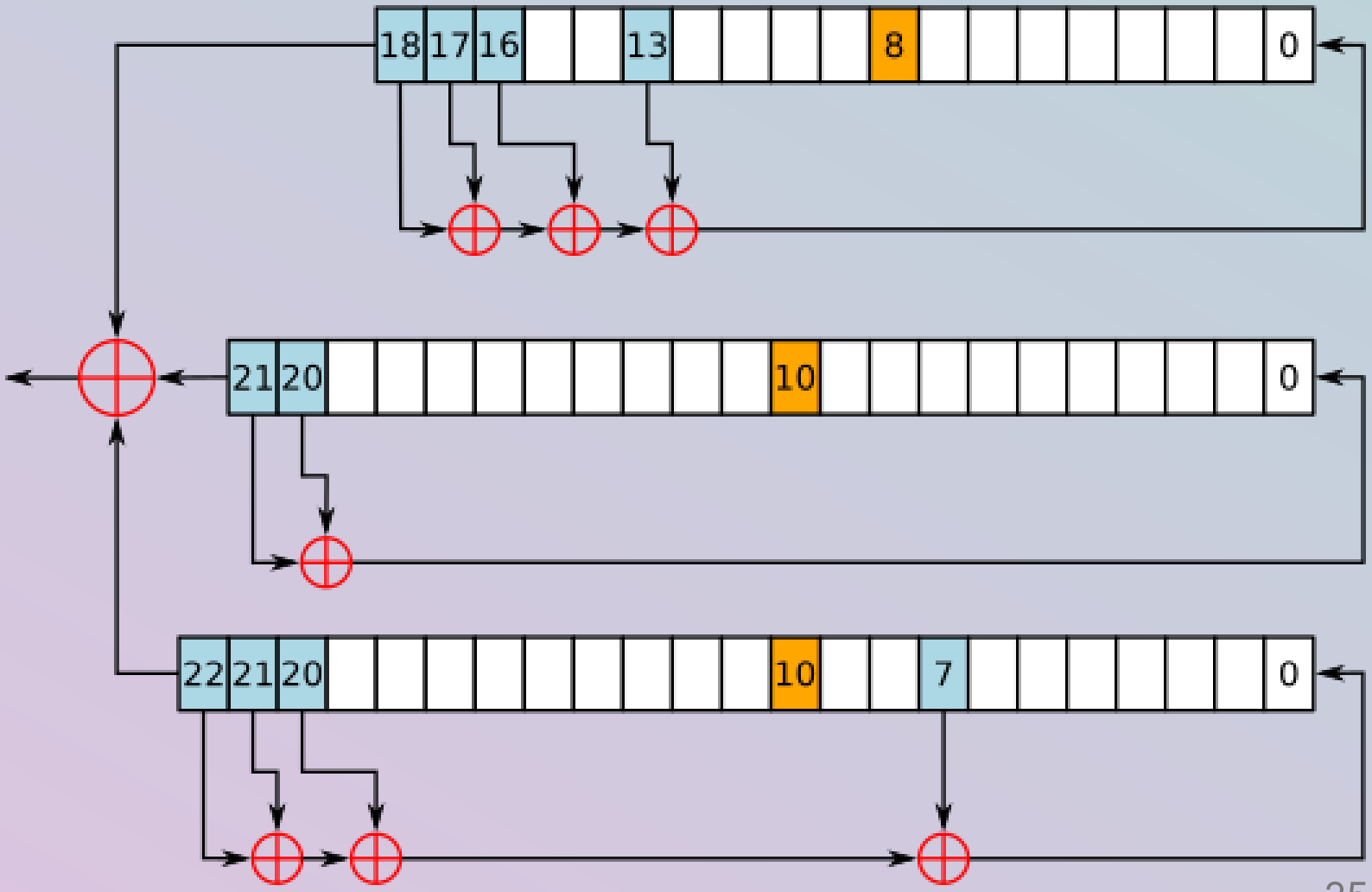
- **Biases** in the first bytes of the keystream
- No **initialization vector (IV)**, thus the same key produces the same keystream
- **Bit-flipping** attacks
- **7** diffrent attacks

A5/1

- Used in **GSM** (2G) mobile phones
- developed in **1987**
- **64-bit** key
- **224-bit** frame number

A5/1 - Building blocks

- **Key** is xored in (64 cycles)
- **Frame number** is xored in (22 cycles)
- 100 cycles discarded
- ready to generate **2*114** bits of keystream



A5/X flaws

- **Weak keys**: 64-bit keys that can be broken by brute force
- **Short period**: about 8Mb of randomess
- First broken in **1994** by **Ross Anderson** and **Ruediger Weis**
- Can be broken in **seconds** with a **FPGAs** and **Rainbow tables**
- Regularly broken by **NSA**

NESSIE (New European Schemes for Signatures, Integrity and Encryption) 2000 - 2003

- **F-FCSR**
- **SFINKS**
- **DECIM**
- **LEX**
- **Helix**
- **Phelix**
- And many more...

:(

- **F-FCSR** and **SFINKS** suffered from known plaintext attacks.
- **DECIM** was broken shortly after submission (differential attacks).
- **LEX** showed statistical biases in its output.
- **Helix** and **Phelix** had potential forgeries in authenticated encryption mode.
- **None were accepted** as a standard.

eSTREAM 2004 - 2008

- **eSTREAM** was a project to identify new stream ciphers after the **NESSIE** failure.
- It was divided into two profiles:
  - **Profile 1**: Software-oriented ciphers
  - **Profile 2**: Hardware-oriented ciphers

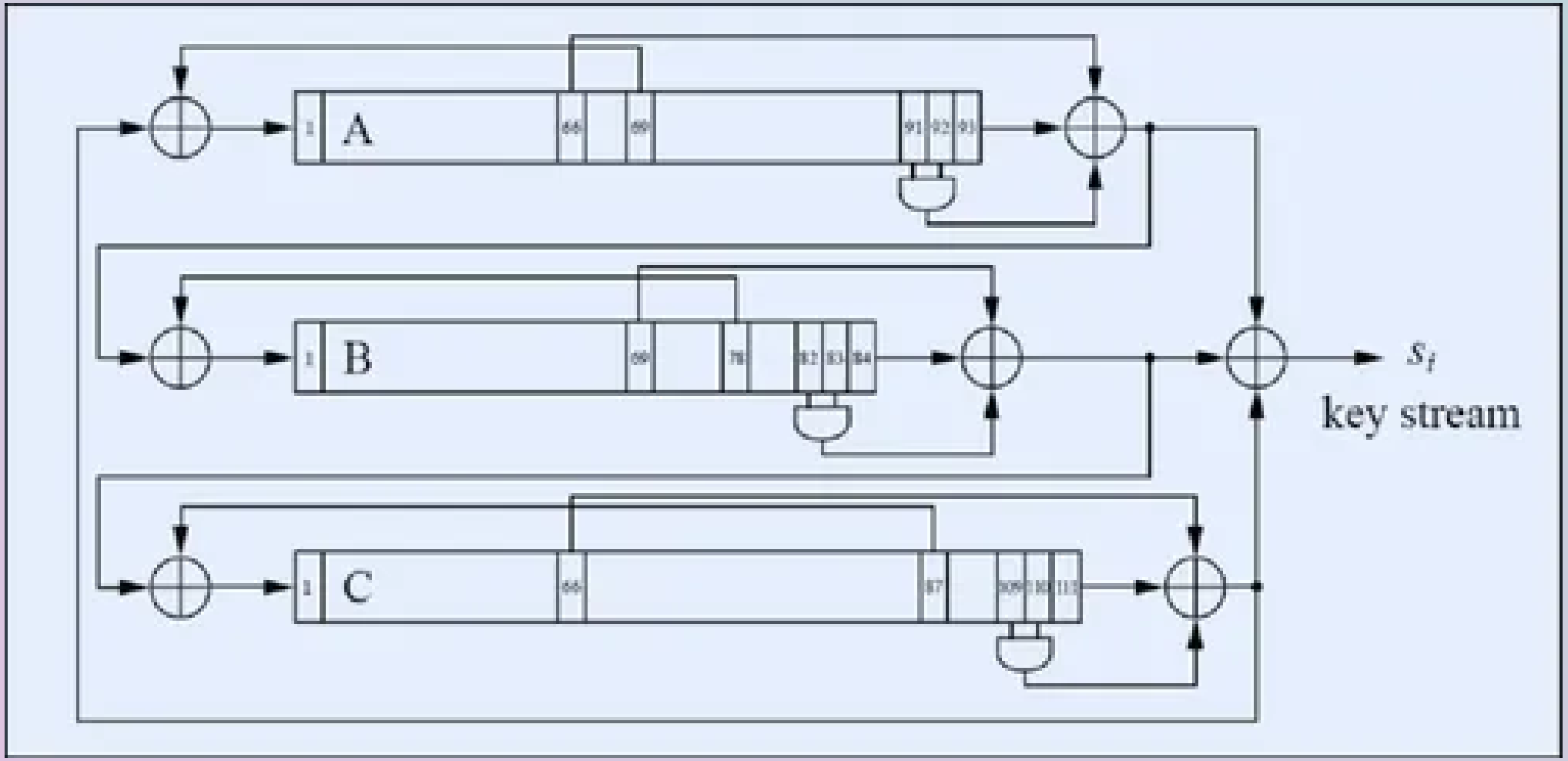
eSTREAM - Profile 1

- **HC-256**: High-speed software-oriented cipher
- **Rabbit**: Fast and efficient cipher
- **Salsa20**: A family of ciphers designed by **Daniel J. Bernstein**
- **Sosemanuk**: A cipher based on the Salsa20 core

eSTREAM - Profile 2

- **Trivium**: Simplest secure stream cipher
- **Grain**: designed for low-resource environments
- **MICKEY**: designed for low-resource environments

Trivium



Trivium

- **Key**: 80 bits
- **IV**: 80 bits
- **State**: 288 bits
- **Period**: 2^{64}

ChaCha20

- 128-bit **constant**: "expand 32-byte k"
- **Key**: 256 bits
- **Counter**: 32 bits
- **Nonce**: 96 bits

	Column 0	Column 1	Column 2	Column 3
Row 0	"expa"	"nd 3"	"2-by"	"te k"
Row 1	key[0]	key[1]	key[2]	key[3]
Row 2	key[4]	key[5]	key[6]	key[7]
Row 3	counter	nonce[0]	nonce[1]	nonce[2]

ChaCha20 - Operations

```
def QR(a, b, c, d){
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<=  8;
c += d; b ^= c; b <<<=  7;
}
```

ChaCha20 - Operations

```
// Odd round
QR(0, 4,  8, 12) // column 1
QR(1, 5,  9, 13) // column 2
QR(2, 6, 10, 14) // column 3
QR(3, 7, 11, 15) // column 4
// Even round
QR(0, 5, 10, 15) // diagonal 1 (main diagonal)
QR(1, 6, 11, 12) // diagonal 2
QR(2, 7,  8, 13) // diagonal 3
QR(3, 4,  9, 14) // diagonal 4
```

ChaCha20 - Use

- TLS
- OpenSSH
- QUIC
- Signal Protocol
- WhatsApp

Sources

- Wikipedia - Stream cipher
- Wikipedia - Vigenère cipher
- Wikipedia - One-time pad
- Wikipedia - Lorenz cipher
- Wikipedia - A5/1
- Wikipedia - RC4
- Wikipedia - eSTREAM
- Wikipedia - Trivium
- Wikipedia - ChaCha20