Cryptography
CS 555

Topic 15: Stream Ciphers

An Existential Crisis?

- We have used primitives like PRGs, PRFs to build secure MACs, CCA-Secure Encryption etc…

- Do such primitives exist? In practice?

- How do we build them?

Recap

- Hash Functions/PRGs/PRFs, CCA-Secure Encryption, MACs

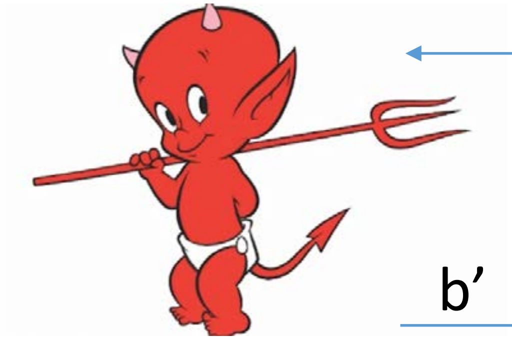
Goals for This Week:

- Practical Constructions of Symmetric Key Primitives

Today’s Goals: Stream Ciphers

- Linear Feedback Shift Registers (and attacks)
- RC4 (and attacks)
- Trivium

PRG Security as a Game



R

b’

Random bit b

If b=1

$r \leftarrow \{0,1\}^n$

R = G(r)

Else

$0,1\}^{\ell(n)}$

ppt attacker

negligible function

$\forall$ $\Pr\Big[$ $Guesses\ b' = b\Big] \le \frac{1}{2} + \mu(n)$

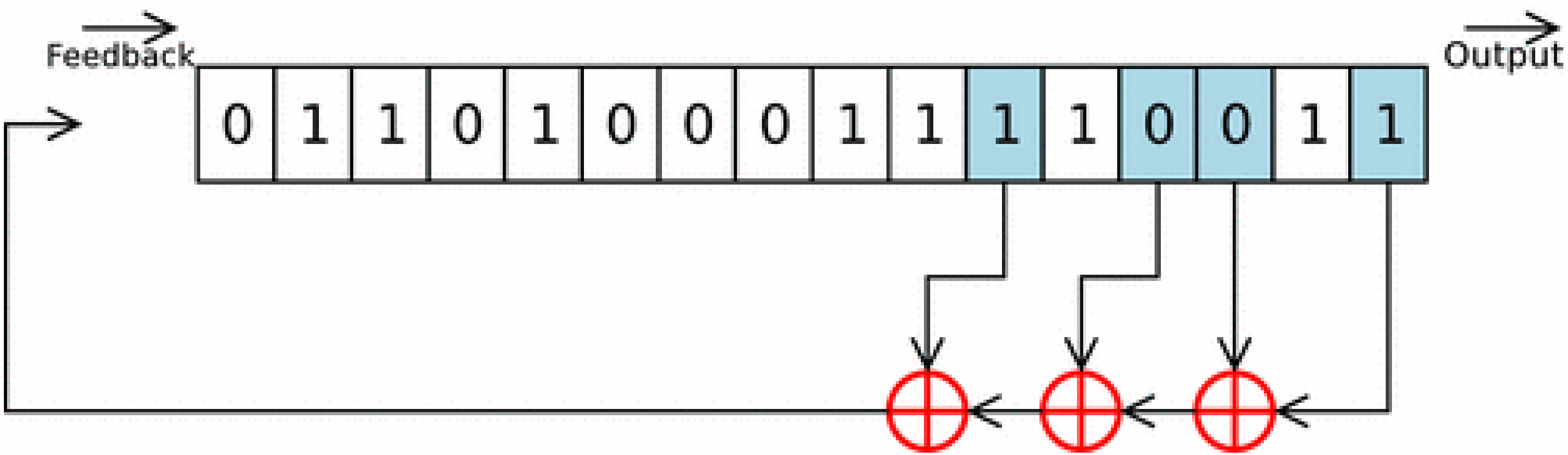
Stream Cipher vs PRG

- PRG pseudorandom bits output all at once

- Stream Cipher
  - Pseudorandom bits can be output as a stream
  - RC4, RC5 (Ron’s Code)

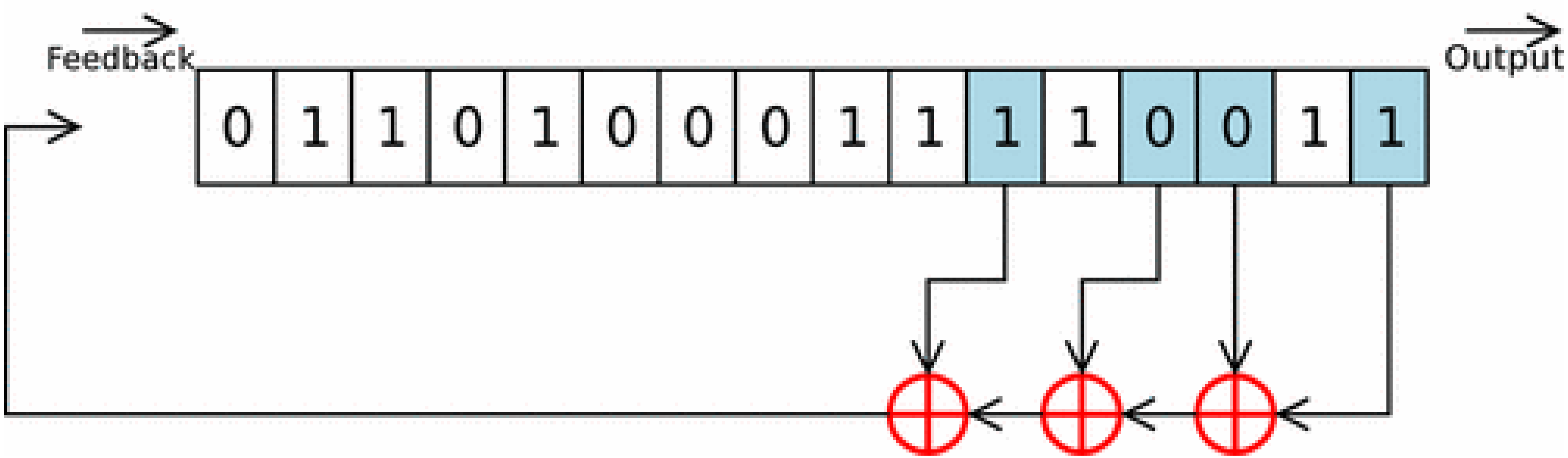
```
$st_0$ := Init(s)
**For** i=1 to $\ell$:
    $(y_i, st_i)$:=GetBits($st_{i-1}$)
**Output**: $y_1,\ldots,y_\ell$
```

Linear Feedback Shift Register



Linear Feedback Shift Register

- State at time t: $s_{n-1}^t, \ldots, s_1^t, s_0^t$ (n registers)
- Feedback Coefficients: $\mathbf{S} \subseteq \{0, \ldots, n\}$

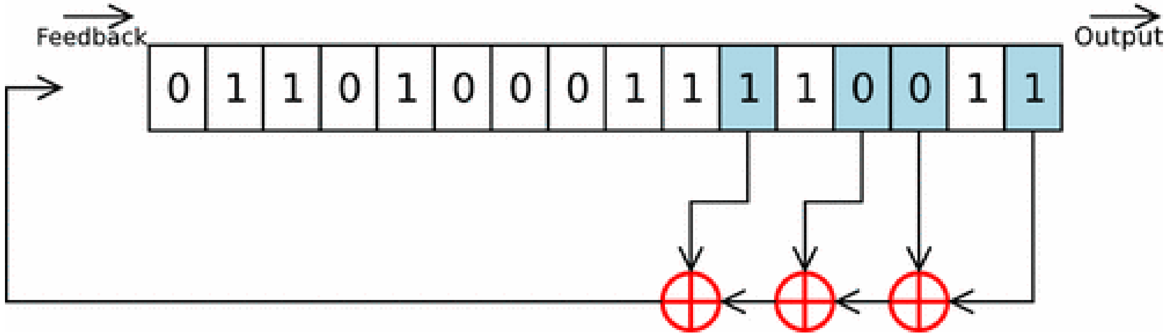


Linear Feedback Shift Register

- State at time t: $s_{n-1}^t, \ldots, s_1^t, s_0^t$ (n registers)
- Feedback Coefficients: $\mathbf{S} \subseteq \{0, \ldots, n-1\}$
- **State at time t+1:** $\bigoplus_{i \in S} s_i^t, s_{n-1}^t, \ldots, s_1^t,$

$$s_{n-1}^{t+1} = \bigoplus_{i \in S} s_i^t, \qquad \text{and} \qquad s_i^{t+1} = s_{i+1}^t \text{ for i} < \text{n} - 1$$

Output at time t+1: $y_{t+1} = s_0^t$



Linear Feedback Shift Register

- **Observation 1**: First n bits of output reveal initial state

$$y_1, \dots, y_n = s_0^0, s_1^0, \dots, s_{n-1}^0$$

- **Observation 2**: Next n bits allow us to solve for n unknowns

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & otherwise \end{cases}$$

$$y_{n+1} = y_n x_{n-1} + \cdots + y_1 x_0$$

Linear Feedback Shift Register

- **Observation 1**: First n bits of output reveal initial state

$$y_1, \dots, y_n = s_0^0, s_1^0, \dots, s_{n-1}^0$$

- **Observation 2**: Next n bits allow us to solve for n unknowns

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & otherwise \end{cases}$$

$$y_{n+1} = y_n x_{n-1} + \cdots + y_1 x_0 \text{ mod } 2$$

Linear Feedback Shift Register

- **Observation 2**: Next n bits allow us to solve for n unknowns

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & otherwise \end{cases}$$

$$\left.\begin{aligned} y_{n+1} &= y_n x_{n-1} + \cdots + y_1 x_0 \text{ mod } 2 \\ &\vdots \\ y_{2n} &= y_{2n-1} x_{n-1} + \cdots + y_n x_0 \text{ mod } 2 \end{aligned}\right\} \text{N unknowns \& N linear independent constraints}$$

Removing Linearity

- Attacks exploited linear relationship between state and output bits

- **Nonlinear Feedback:**

$$\cancel{s_{n-1}^{t+1} = \bigoplus_{i\in S} s_i^t ,}$$

Non linear function

$$s_{n-1}^{t+1} = g(s_0^t, s_1^t, \ldots, s_{n-1}^t)$$

Removing Linearity

- Attacks exploited linear relationship between state and output bits

- **Nonlinear Combination:**

Non linear function

~~$\boldsymbol{y_{t+1} = s_0^t}$~~

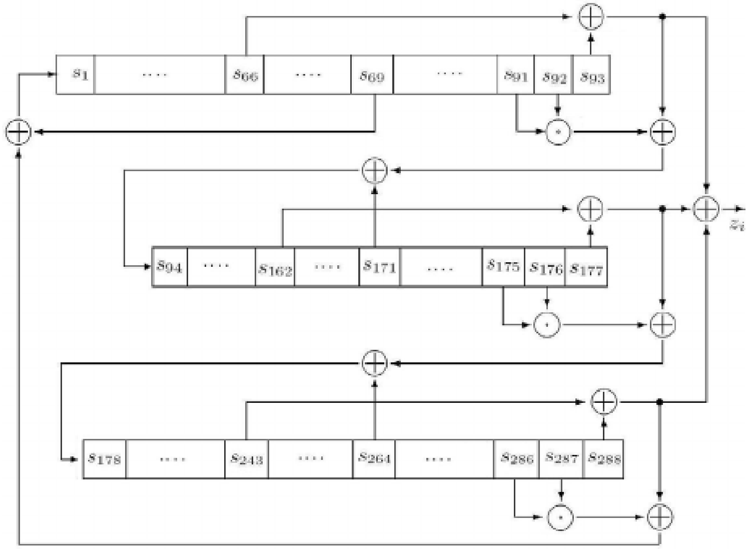
$$y_{t+1} = f(s_0^t, s_1^t, \dots, s_{n-1}^t)$$

- **Important**: f must be balanced!

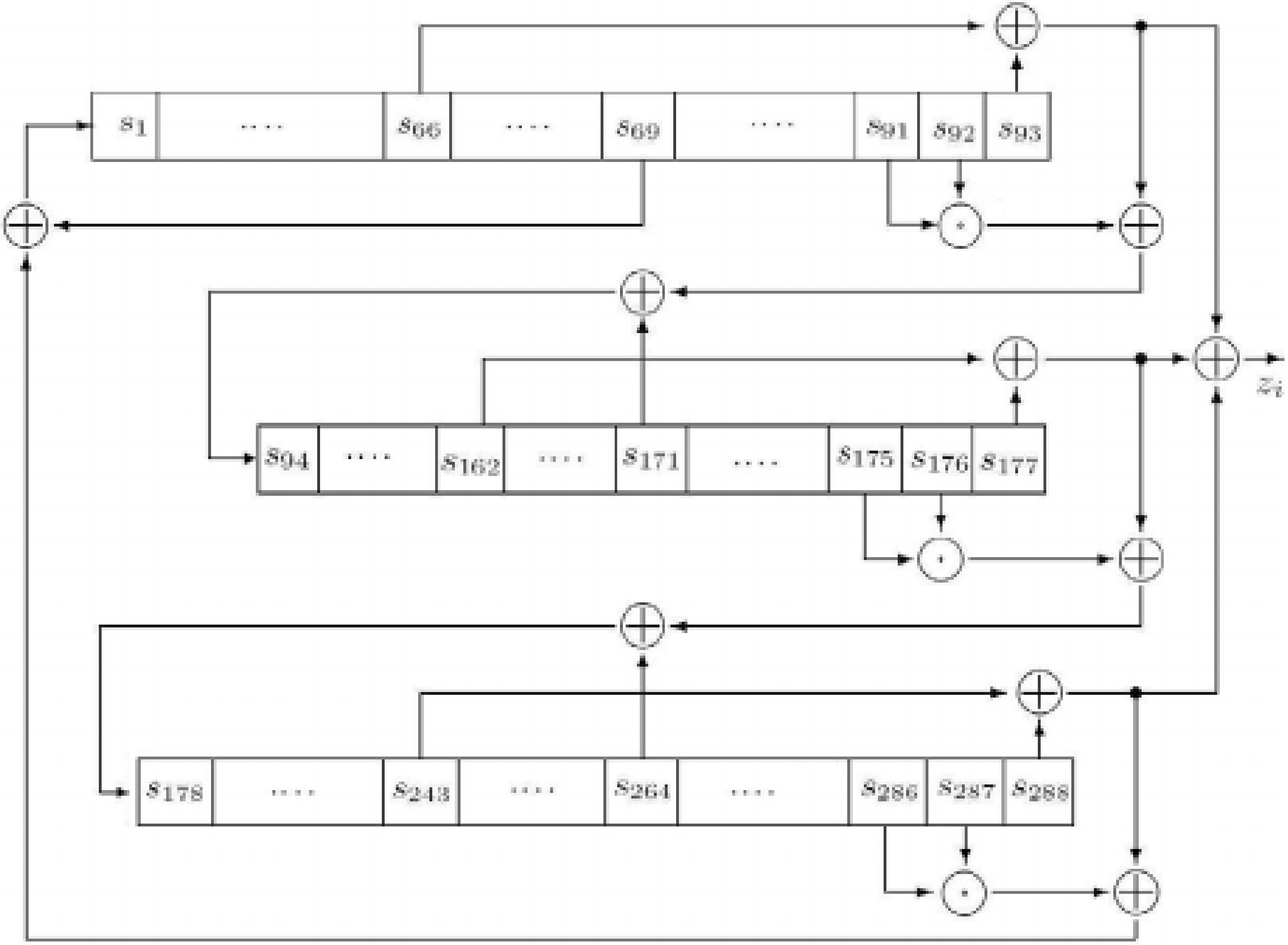
$$\Pr[f(x) = 1] \approx \frac{1}{2}$$

Trivium (2008)

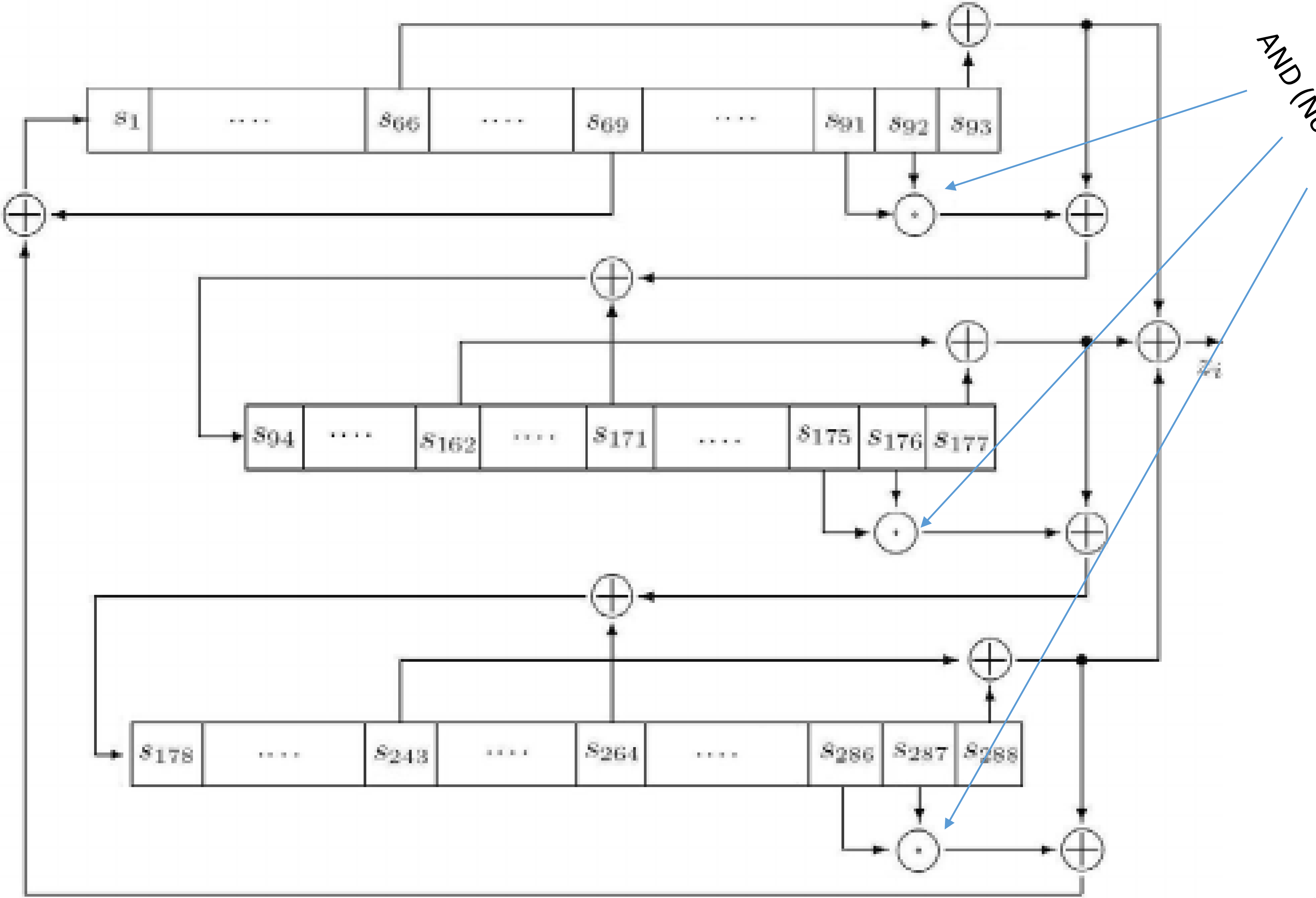
- Won the eSTREAM competition
- Currently, no known attacks are better than brute force
- Couples Output from three nonlinear Feedback Shift Registers
- First 4*288 “output bits” are discared



Trivium (2008)

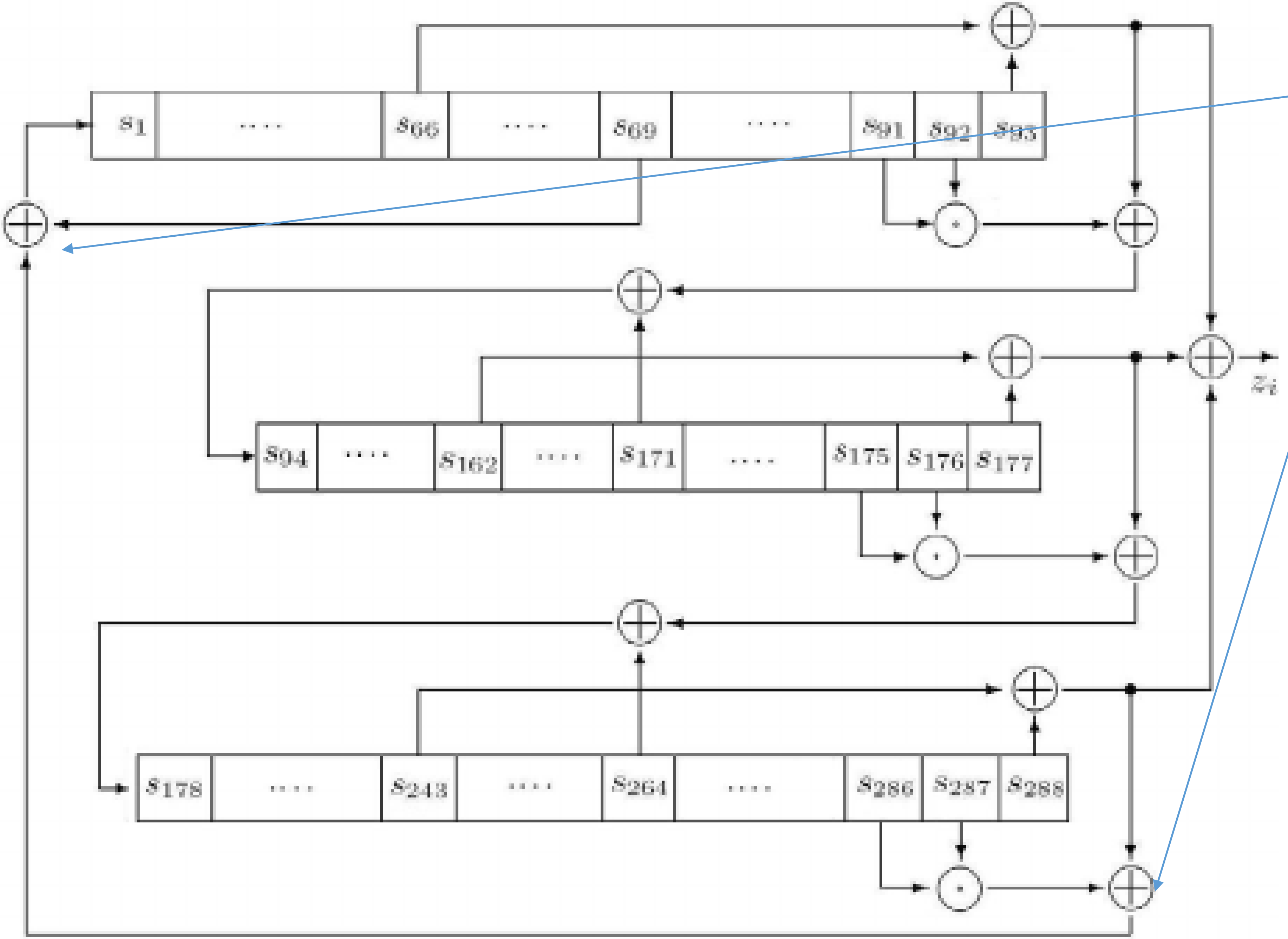


Trivium (2008)



AND (Non-linear)

Trivium (2008)



(Non-linear) Feedback

Combination Generator

- Attacks exploited linear relationship between state and output bits

- **Nonlinear Combination:**

Non linear function

$$\cancel{\boldsymbol{y_{t+1} = s_0^t}}$$

$$y_{t+1} = f(s_0^t, s_1^t, \dots, s_{n-1}^t)$$

- **Important**: f must be balanced!

$$\Pr[f(x) = 1] \approx \frac{1}{2}$$

Feedback Shift Registers

- Good performance in hardware

- Performance is less ideal for software

The RC4 Stream Cipher

- A proprietary cipher owned by RSA, designed by Ron Rivest in 1987.
- Became public in 1994.
- Simple and effective design.
- Variable key size (typical 40 to 256 bits),
- Output unbounded number of bytes.
- Widely used (web SSL/TLS, wireless WEP).
- Extensively studied, not a completely secure PRNG, when used correctly, ~~no known attacks exist~~
- **Newer Versions**: RC5 and RC6
- **Rijndael** selected by NIST as AES in 2000

The RC4 Cipher

- The cipher internal state consists of
  - a 256-byte array S, which contains a permutation of 0 to 255
    - total number of possible states is 256! ≈ 2^{1700}
  - two indexes: i, j

```
i = j = 0
Loop
   i = (i + 1) (mod 256)
   j = (j + S[i]) (mod 256)
   swap(S[i], S[j])
   output S[S[i] + S[j] (mod 256)]
End Loop
```

Distinguishing Attack

- Let S_0 denote initial state
- Suppose that S_0[2]=0 and S_0[1]$= \mathrm{X} \neq 0$

	1	2	3	…	X	…	255
S_0	$S_0[1]\neq 0$	0	S_0[3]		S_0[X]		S_0[255]

```
i = j = 0
Loop
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    output S[S[i] + S[j] (mod 256)]
End Loop
```

Distinguishing Attack

- Let S_0 denote initial state
- Suppose that S_0[2]=0 and S_0[1]= $X \neq 0$

	1	2	3	…	X	…	255
S_0	$\boldsymbol{X \neq 0}$	0	S_0[3]		**S_0[X]**		S_0[255]

i=1, j =X

```
i = j = 0
Loop
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    output S[S[i] + S[j] (mod 256)]
End Loop
```

Distinguishing Attack

	1	2	3	…	X	…	255
S_0	$\boldsymbol{X \neq 0}$	0	$S_0[3]$		$\boldsymbol{S_0}$**[X]**		$S_0[255]$
S_1	$\boldsymbol{S_0}$**[X]**	0	$S_0[3]$		$\boldsymbol{X \neq 0}$		$S_0[255]$

i=1, j =X

Output $\mathbf{y_1}$= S_1[S[i]+S[j]]

i=2, j =X

```
i = j = 0
Loop
   i = (i + 1) (mod 256)
   j = (j + S[i]) (mod 256)
   swap(S[i], S[j])
   output S[S[i] + S[j] (mod 256)]
End Loop
```

Distinguishing Attack

	1	2	3	…	X	…	255
S_0	$\boldsymbol{X \neq 0}$	0	$S_0[3]$		$\boldsymbol{S_0}$**[X]**		$S_0[255]$
S_1	S_0[X]	**0**	$S_0[3]$		$\boldsymbol{X \neq 0}$		$S_0[255]$
S_2	S_0[X]	$\boldsymbol{X \neq 0}$	$S_0[3]$		**0**		

i=2, j =X

```
i = j = 0
Loop
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    output S[S[i] + S[j] (mod 256)]
End Loop
```

Output:

$$y_2 = S_2[S_2[2]+S_2[X]]$$
$$= S_2[0+X]$$
$$=0$$

Distinguishing Attack

Let $\mathrm{p} = \Pr[S_0[2]{=}0 \text{ and } S_0[1] \neq 2]$

$$p = \frac{1}{256}\left(1 - \frac{1}{255}\right)$$

- Probability second output byte is 0

$$\Pr[y_2 = 0 \mid S_0[2]{=}0 \text{ and } S_0[1] \neq 2]p + \Pr[y_2 = 0 \mid S_0[2] \neq 0 \text{ or } S_0[1] \neq 2](1-p)$$

$$= p + (1-p)\frac{1}{256}$$

$$= \frac{1}{256}\left(1 - \frac{1}{255}\right) + \left(1 - \frac{1}{256} + \frac{1}{256}\frac{1}{255}\right)\frac{1}{256}$$

$$\approx \frac{2}{256}$$

Other Attacks

- Wired Equivalent Privacy (WEP) encryption used RC4 with an initialization vector

- Description of RC4 doesn't involve initialization vector…
    - But WEP imposes an initialization vector
    - K=IV || K'
    - Since IV is transmitted attacker may have first few bytes of K!

    - Giving the attacker partial knowledge of K often allows recovery of the entire key K' over time!

Next Class

- Read Katz and Lindell 6.2-6.2.2
- Block Ciphers