

# Applied Cryptography

## Week 4: Stream Ciphers

Bernardo Portela

M:ERSI, M:SI, M:CC - 25

# Context

Historically associated with fragility

- Insecure mobile phone communications
- Weak wi-Fi protocols
- Broken smart card implementations

## Context

Historically associated with fragility

- Insecure mobile phone communications
- Weak wi-Fi protocols
- Broken smart card implementations

This is **no longer the case**

- Used in Bluetooth; 4G; TLS; (...)

# Context

Historically associated with fragility

- Insecure mobile phone communications
- Weak wi-Fi protocols
- Broken smart card implementations

This is **no longer the case**

- Used in Bluetooth; 4G; TLS; (...)

## Overview

- We will take a look at the reason for said fame
  - A5/1; GSM; RC4
- As well as the reason for their comeback
  - Grain-128A (HW); Salsa20 (SW)

## Definition

Stream ciphers are Deterministic Random Number Generators (DRNGs). They take a known seed and generate randomness

## Definition

Stream ciphers are Deterministic Random Number Generators (DRNGs). They take a known seed and generate randomness

**Q: Why not PRNG – Probabilistic?**

## Definition

Stream ciphers are Deterministic Random Number Generators (DRNGs). They take a known seed and generate randomness

**Q: Why not PRNG – Probabilistic?** ... because then we would not be able to decrypt!

## Definition

Stream ciphers are Deterministic Random Number Generators (DRNGs). They take a known seed and generate randomness

**Q: Why not PRNG – Probabilistic?** ... because then we would not be able to decrypt!

$(k, n) \Rightarrow KS$

- k - Key; secret; 128/256 bits
- n -Nonce; public; 64/128 bits
- KS - Keystream; XOR'd with the plaintext
- $c \leftarrow KS \oplus m$

## Definition

Stream ciphers are Deterministic Random Number Generators (DRNGs). They take a known seed and generate randomness

**Q: Why not PRNG – Probabilistic?** ... because then we would not be able to decrypt!

$(k, n) \Rightarrow KS$

- k - Key; secret; 128/256 bits
- n -Nonce; public; 64/128 bits
- KS - Keystream; XOR'd with the plaintext
- $c \leftarrow KS \oplus m$

**Q: How to decrypt?**

## Definition

Stream ciphers are Deterministic Random Number Generators (DRNGs). They take a known seed and generate randomness

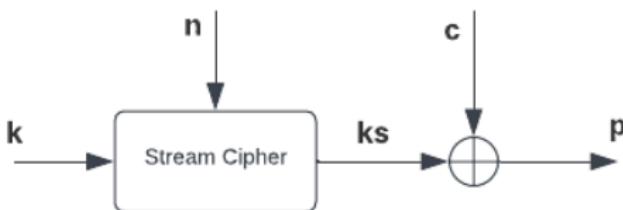
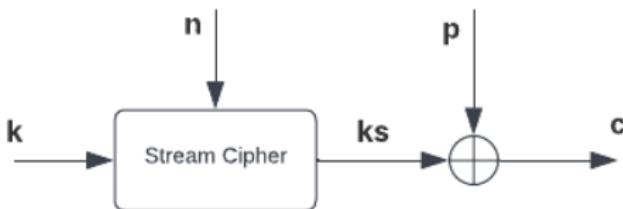
**Q: Why not PRNG – Probabilistic?** ... because then we would not be able to decrypt!

$(k, n) \Rightarrow KS$

- k - Key; secret; 128/256 bits
- n -Nonce; public; 64/128 bits
- KS - Keystream; XOR'd with the plaintext
- $c \leftarrow KS \oplus m$

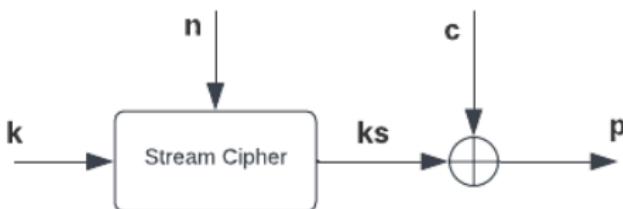
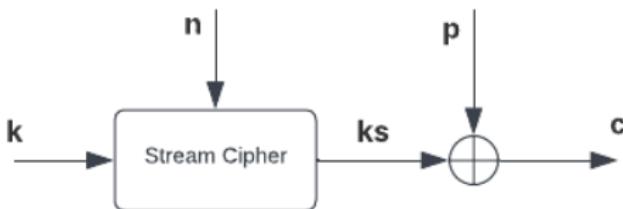
**Q: How to decrypt?** ... the same as encryption

## Stream cipher behavior



- $n$ : nonce (**N**umber used only **ONCE**)
- Sometimes called IV (Initialization Vector)

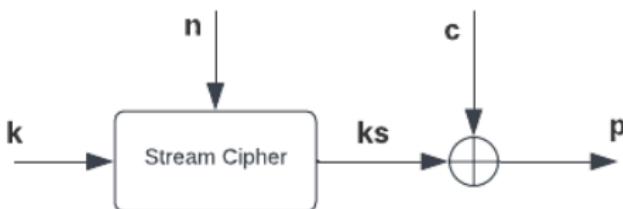
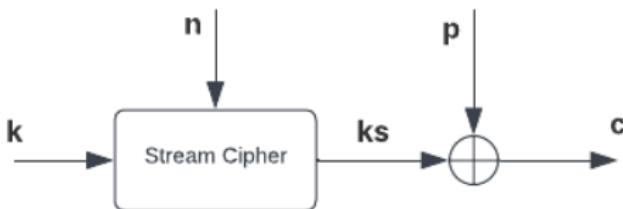
## Stream cipher behavior



- $n$ : nonce (**N**umber used only **O**NCE)
- Sometimes called IV (Initialization Vector)

**Q: Why?**

## Stream cipher behavior

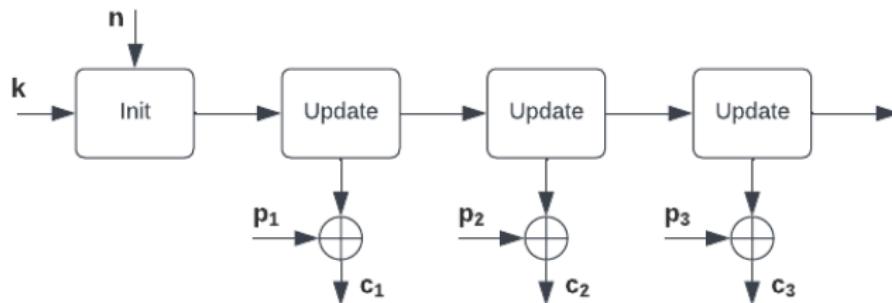


- $n$ : nonce (**Number used only ONCE**)
- Sometimes called IV (Initialization Vector)

**Q: Why?** Reveals duplicates

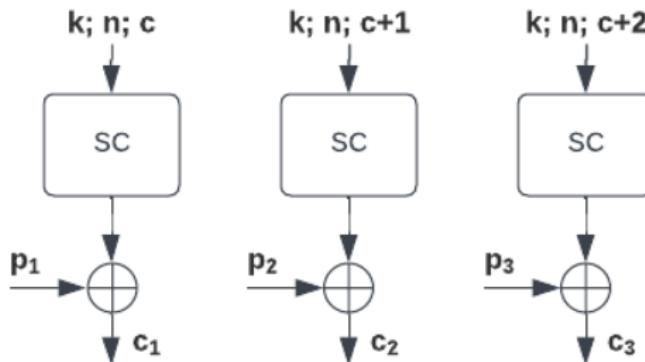
# Types of Stream Ciphers - 1

## Stateful Stream Ciphers (RC4)



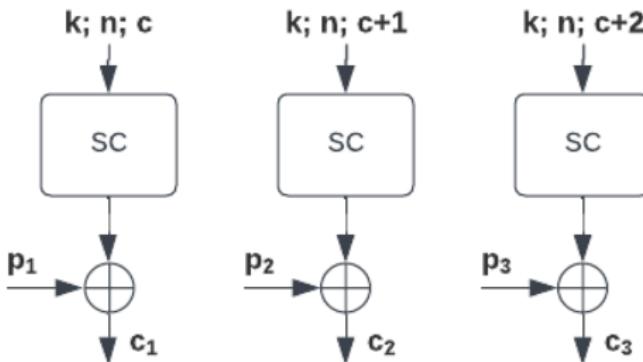
## Types of Stream Ciphers - 2

### Counter-based Stream Ciphers (Salsa20)



## Types of Stream Ciphers - 2

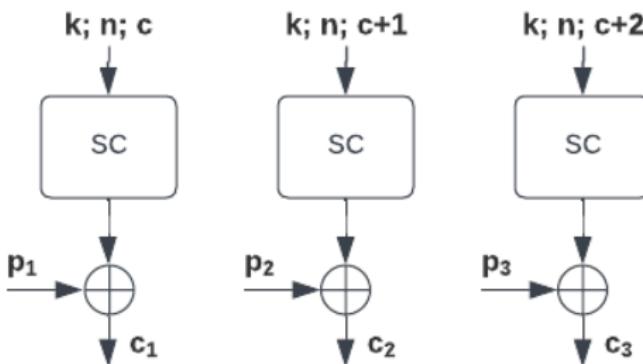
### Counter-based Stream Ciphers (Salsa20)



- Q: What does this remind you of?

## Types of Stream Ciphers - 2

### Counter-based Stream Ciphers (Salsa20)



- **Q: What does this remind you of?**
- AES-CTR follows a similar structure

# Stream Ciphers and Hardware

## Hardware-based ciphers

- Cipher is an electronic circuit implementing the algorithm; can only be used for that
  - Application-specific integrated circuits (ASICs)
  - Programmable logic devices (PLDs)
  - Field-programmable gate arrays (FPGAs)

# Stream Ciphers and Hardware

## Hardware-based ciphers

- Cipher is an electronic circuit implementing the algorithm; can only be used for that
  - Application-specific integrated circuits (ASICs)
  - Programmable logic devices (PLDs)
  - Field-programmable gate arrays (FPGAs)

## Software-based ciphers

- Algorithms tell the system what to do

# Stream Ciphers and Hardware

## Hardware-based ciphers

- Cipher is an electronic circuit implementing the algorithm; can only be used for that
  - Application-specific integrated circuits (ASICs)
  - Programmable logic devices (PLDs)
  - Field-programmable gate arrays (FPGAs)

## Software-based ciphers

- Algorithms tell the system what to do
- HW deals with bits; SW deals with words (32/64 bits)
- Traditionally, SCs were cheaper – used smaller circuits
- Today, block ciphers are not that different
- Distinction is not that obvious nowadays

# Feedback Shift Registers (FSRs)

- FSRs are popular because they are *simple* and *well-understood*
- Array of bits + feedback function  $f$

# Feedback Shift Registers (FSRs)

- FSRs are popular because they are *simple* and *well-understood*
- Array of bits + feedback function  $f$

$R_0$  is the initial state  $R_1$  is the next state, defined as:

- $R_0$  left-shifted by 1
- Leftmost bit of  $R_0$  is the output
- Rightmost bit of  $R_1$  filled by  $f(R_0)$

Stream Ciphers  
oooooo

Feedback Shift Registers  
○●oooooooooooo

Hardware-Oriented SCs  
oooo

Software-Oriented SCs  
oooooo

## FSR Example

4-bit FSR;  $f$  XORs all bits; initial state  $R_0 = 1100$

## FSR Example

4-bit FSR;  $f$  XORs all bits; initial state  $R_0 = 1100$

$$f(R_0) = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Output = 1

$$R_1 = 1000$$

## FSR Example

4-bit FSR;  $f$  XORs all bits; initial state  $R_0 = 1100$

$$f(R_0) = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Output = 1

$R_1 = 1000$

---

$$f(R_1) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

Output = 1

$R_2 = 0001$

## FSR Example

4-bit FSR;  $f$  XORs all bits; initial state  $R_0 = 1100$

$$f(R_0) = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Output = 1

$R_1 = 1000$

---

$$f(R_1) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

Output = 1

$R_2 = 0001$

---

$$f(R_2) =$$

Output =

$R_2 =$

## FSR Example

4-bit FSR;  $f$  XORs all bits; initial state  $R_0 = 1100$

$$f(R_0) = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Output = 1

$R_1 = 1000$

---

$$f(R_1) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

Output = 1

$R_2 = 0001$

---

$$f(R_2) = 1$$

Output = 0

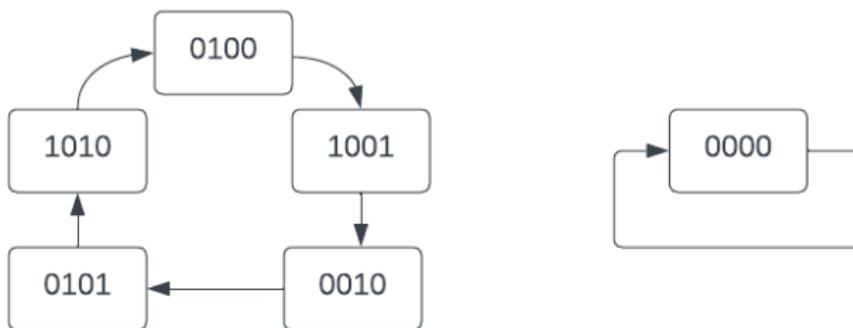
$R_2 = 0011$

## Period of an FSR

**1100**    0001    0110  
1000    0011    **1100**

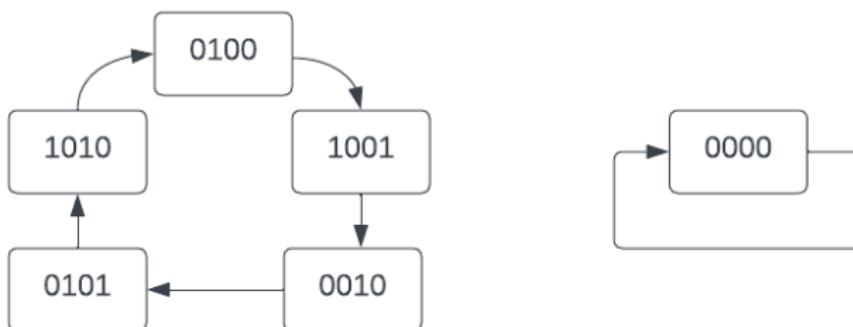
# Period of an FSR

<b>1100</b>	0001	0110
1000	0011	<b>1100</b>



## Period of an FSR

**1100**    0001    0110  
1000    0011    **1100**



- All FSRs have a period
- Larger period  $\Rightarrow$  more secure

## Linear Feedback Shift Registers (LFSRs)

- FSRs with a linear shift function
- Only some of the bits are XORed

# Linear Feedback Shift Registers (LFSRs)

- FSRs with a linear shift function
- Only some of the bits are XORed
- Selecting which bits to XOR is crucial to maximize period
- Function represented as a polynomial expression

$$1 + X + X^2 + \cdots + X^n$$

- $X^i$  only included if the  $i$ -th bit is XORed by the function
- Period is maximal if the polynomial is **primitive**

# Linear Feedback Shift Registers (LFSRs)

- FSRs with a linear shift function
- Only some of the bits are XORed
- Selecting which bits to XOR is crucial to maximize period
- Function represented as a polynomial expression

$$1 + X + X^2 + \cdots + X^n$$

- $X^i$  only included if the  $i$ -th bit is XORed by the function
- Period is maximal if the polynomial is **primitive**
- **Q: What is the maximum period of any  $n$ -bit LFSR?**

# Linear Feedback Shift Registers (LFSRs)

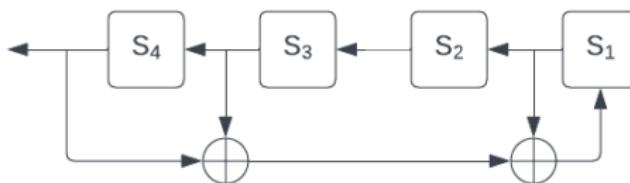
- FSRs with a linear shift function
- Only some of the bits are XORed
- Selecting which bits to XOR is crucial to maximize period
- Function represented as a polynomial expression

$$1 + X + X^2 + \cdots + X^n$$

- $X^i$  only included if the  $i$ -th bit is XORed by the function
- Period is maximal if the polynomial is **primitive**
- **Q: What is the maximum period of any  $n$ -bit LFSR?**  
 $2^n - 1!$  The state 000... will always shift to 000...

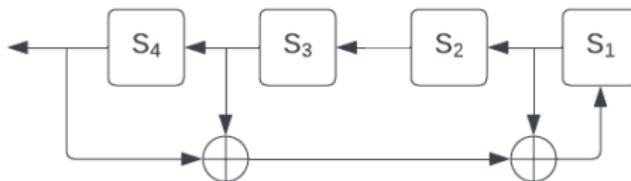
## Example #1 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X + X^3 + X^4$



## Example #1 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X + X^3 + X^4$

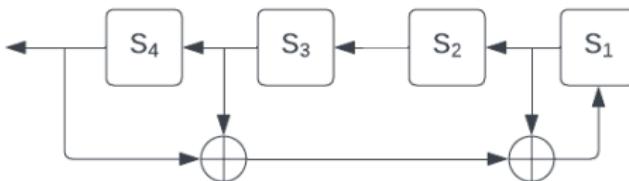


$$f(R_0) = S_4 + S_3 + S_1 = 0 + 0 + 1 = 1$$

$$R_1 = 0011$$

## Example #1 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X + X^3 + X^4$



$$f(R_0) = S_4 + S_3 + S_1 = 0 + 0 + 1 = 1$$

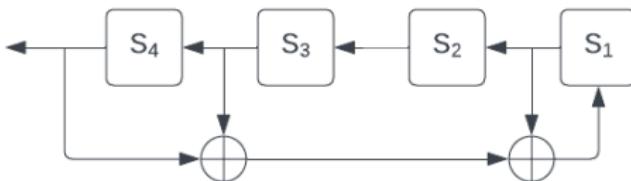
$$R_1 = 0011$$

$$f(R_1) =$$

$$R_2 =$$

## Example #1 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X + X^3 + X^4$



$$f(R_0) = S_4 + S_3 + S_1 = 0 + 0 + 1 = 1$$

$$R_1 = 0011$$

$$f(R_1) = 1$$

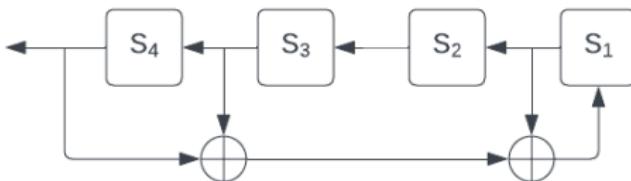
$$R_2 = 0111$$

$$f(R_2) =$$

$$R_3 =$$

## Example #1 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X + X^3 + X^4$



$$f(R_0) = S_4 + S_3 + S_1 = 0 + 0 + 1 = 1$$

$$R_1 = 0011$$

$$f(R_1) = 1$$

$$R_2 = 0111$$

$$f(R_2) = 0$$

$$R_3 = 1110$$

## Example #1 - Continued

$$R_0 = \mathbf{0001}$$

$$R_1 = 0011$$

$$R_2 = 0111$$

$$R_3 = 1110$$

$$R_4 = 1100$$

$$R_5 = 1000$$

$$R_6 = \mathbf{0001}$$

## Example #1 - Continued

$$R_0 = \mathbf{0001}$$

$$R_1 = 0011$$

$$R_2 = 0111$$

$$R_3 = 1110$$

$$R_4 = 1100$$

$$R_5 = 1000$$

$$R_6 = \mathbf{0001}$$

**Q: What is the maximum period?**

## Example #1 - Continued

$$R_0 = \mathbf{0001}$$

$$R_1 = 0011$$

$$R_2 = 0111$$

$$R_3 = 1110$$

$$R_4 = 1100$$

$$R_5 = 1000$$

$$R_6 = \mathbf{0001}$$

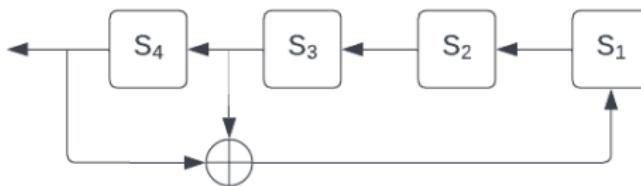
**Q: What is the maximum period?**  $2^4 - 1 = 15$

Indeed, the polynomial is not primitive! It is reducible

$$(1 + X^3)(1 + X) = 1 + X + X^3 + X^4$$

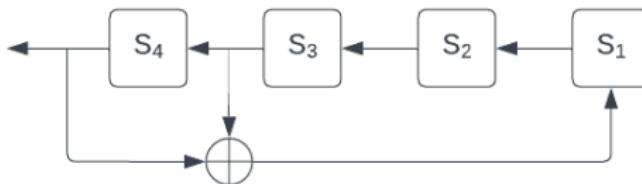
## Example #2 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X^3 + X^4$



## Example #2 - LFSR

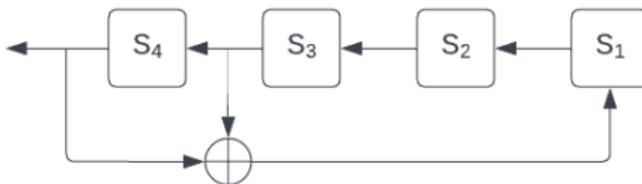
Initial state  $R_0 = 0001$ ; Polynomial  $1 + X^3 + X^4$



$$f(R_0) = S_4 + S_3 = 0 + 0 \\ R_1 = 0010$$

## Example #2 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X^3 + X^4$



$$f(R_0) = S_4 + S_3 = 0 + 0$$

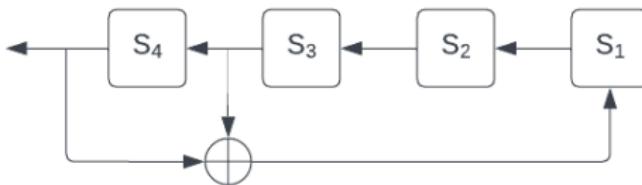
$$R_1 = 0010$$

$$f(R_1) =$$

$$R_2 =$$

## Example #2 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X^3 + X^4$



$$f(R_0) = S_4 + S_3 = 0 + 0$$

$$R_1 = 0010$$

$$f(R_1) = 0$$

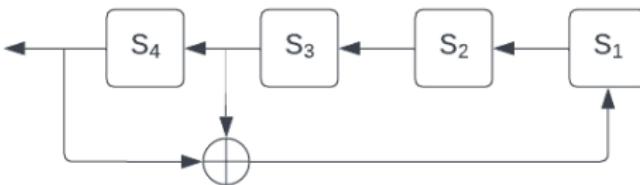
$$R_2 = 0100$$

$$f(R_2) =$$

$$R_3 =$$

## Example #2 - LFSR

Initial state  $R_0 = 0001$ ; Polynomial  $1 + X^3 + X^4$



$$f(R_0) = S_4 + S_3 = 0 + 0$$

$$R_1 = 0010$$

$$f(R_1) = 0$$

$$R_2 = 0100$$

$$f(R_2) = 1$$

$$R_3 = 1001$$

## Example #2 - Continued

<b>0001</b>	0011	0101	1110
0010	0110	1011	1100
0100	1101	0111	1000
1001	1010	1111	<b>0001</b>

## Example #2 - Continued

<b>0001</b>	0011	0101	1110
0010	0110	1011	1100
0100	1101	0111	1000
1001	1010	1111	<b>0001</b>

The polynomial  $1 + X^3 + X^4$  is primitive

LFSR has the maximum period: 15

Stream Ciphers  
oooooo

Feedback Shift Registers  
oooooooo●oooo

Hardware-Oriented SCs  
oooo

Software-Oriented SCs  
oooooo

## Insecurity of LFSRs

Directly using LFSR as a stream cipher is **insecure**

# Insecurity of LFSRs

Directly using LFSR as a stream cipher is **insecure**

- For size  $n$ , attacker needs  $n$  output bits to recover the initial state and feedback polynomial
- ... and thus guess all future keystreams
- Berlekamp-Massey algorithm
- You don't even need to know the size  $n$
- ... just have to repeat for all possible  $n$  until success

## Insecurity of LFSRs

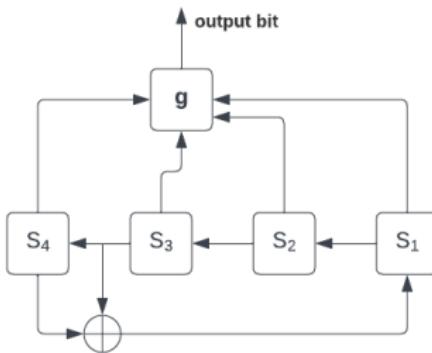
Directly using LFSR as a stream cipher is **insecure**

- For size  $n$ , attacker needs  $n$  output bits to recover the initial state and feedback polynomial
- ... and thus guess all future keystreams
- Berlekamp-Massey algorithm
- You don't even need to know the size  $n$
- ... just have to repeat for all possible  $n$  until success

The issue is that LFSRs are **linear** – short equations, solved by easy math

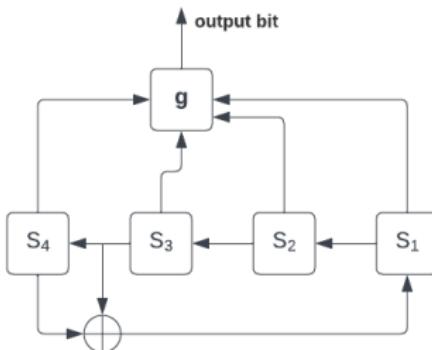
## Filtered LFSRs

One way to hide linearity is to filter outputs via a **nonlinear** function. These are called filtered LFSRs



## Filtered LFSRs

One way to hide linearity is to filter outputs via a **nonlinear** function. These are called filtered LFSRs



Straightforward attacks thwarted. But...

- Still vulnerable to
  - Algebraic attacks
  - Cube attacks (compute derivatives to reduce degree)
  - Fast correlation attacks (some behave like linear functions)

## NFSRs

- FSRs with a **non**linear shift function
- Some of the bits are XORed
- ... but we also use ANDs and ORs

## NFSRs

- FSRs with a **non**linear shift function
- Some of the bits are XORed
- ... but we also use ANDs and ORs

Consider feedback function  $X_1 + X_2 + X_1X_2 + X_3X_4$

We now replace  $X_1$  with the new bit:

$$\begin{aligned}(X_1 + X_2 + X_1X_2 + X_3X_4) + X_1 + (X_1 + X_2 + X_1X_2 + X_3X_4)X_1 + X_2X_3 \\ = X_1X_3X_4 + X_1X_2 + X_2X_3 + X_3X_4 + X_1 + X_2\end{aligned}$$

# NFSRs

- FSRs with a **non**linear shift function
- Some of the bits are XORed
- ... but we also use ANDs and ORs

Consider feedback function  $X_1 + X_2 + X_1X_2 + X_3X_4$

We now replace  $X_1$  with the new bit:

$$\begin{aligned} & (X_1 + X_2 + X_1X_2 + X_3X_4) + X_1 + (X_1 + X_2 + X_1X_2 + X_3X_4)X_1 + X_2X_3 \\ & = X_1X_3X_4 + X_1X_2 + X_2X_3 + X_3X_4 + X_1 + X_2 \end{aligned}$$

Equation has algebraic degree 3 and 6 terms. This quickly scales to incredibly complex equations.

## NFSR Characteristics

### The good

- NFSRs cryptographically stronger than LFSRs
- Output depends on state in a complex fashion
  - We will see in an example how

# NFSR Characteristics

## The good

- NFSRs cryptographically stronger than LFSRs
- Output depends on state in a complex fashion
  - We will see in an example how

## The bad

- Not a clear-cut way to ensure maximum period
- There is even no way to efficiently calculate the period
- For size  $n$ ,  $2^n$  trials to ensure maximum period

# NFSR Characteristics

## The good

- NFSRs cryptographically stronger than LFSRs
- Output depends on state in a complex fashion
  - We will see in an example how

## The bad

- Not a clear-cut way to ensure maximum period
- There is even no way to efficiently calculate the period
- For size  $n$ ,  $2^n$  trials to ensure maximum period

Modern stream ciphers combine LFSRs with NFSRs to get the best of both worlds, which incidentally is what Grain-128a does.

## -Key Takeaways-

- Stream ciphers generate keystream, to  $\oplus$  over the message

## Key Takeaways

- Stream ciphers generate keystream, to  $\oplus$  over the message
- Historically, stream ciphers were fragile and a better fit for HW-based solutions. Nowadays, neither is necessarily true

 Key Takeaways

- Stream ciphers generate keystream, to  $\oplus$  over the message
- Historically, stream ciphers were fragile and a better fit for HW-based solutions. Nowadays, neither is necessarily true
- FSR consist in an internal state, and repeated shifts over it

 Key Takeaways

- Stream ciphers generate keystream, to  $\oplus$  over the message
- Historically, stream ciphers were fragile and a better fit for HW-based solutions. Nowadays, neither is necessarily true
- FSR consist in an internal state, and repeated shifts over it
- Linear Feedback Shift Registers...
  - Have a linear feedback function
  - If they follow a primitive polynomial have maximum period
  - $2^n - 1$  for size  $n$
  - Cryptographically weak

 Key Takeaways

- Stream ciphers generate keystream, to  $\oplus$  over the message
- Historically, stream ciphers were fragile and a better fit for HW-based solutions. Nowadays, neither is necessarily true
- FSR consist in an internal state, and repeated shifts over it
- Linear Feedback Shift Registers...
  - Have a linear feedback function
  - If they follow a primitive polynomial have maximum period
  - $2^n - 1$  for size  $n$
  - Cryptographically weak
- Nonlinear Feedback Shift Registers...
  - Have a nonlinear feedback function
  - Impossible to know if they have the maximum period
  - Cryptographically challenging to solve

 Key Takeaways

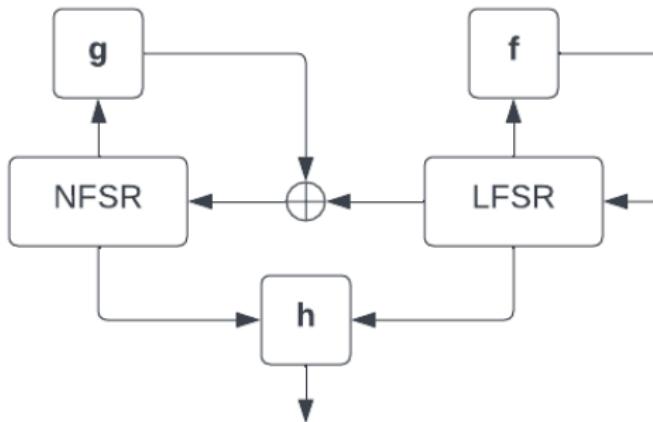
- Stream ciphers generate keystream, to  $\oplus$  over the message
- Historically, stream ciphers were fragile and a better fit for HW-based solutions. Nowadays, neither is necessarily true
- FSR consist in an internal state, and repeated shifts over it
- Linear Feedback Shift Registers...
  - Have a linear feedback function
  - If they follow a primitive polynomial have maximum period
  - $2^n - 1$  for size  $n$
  - Cryptographically weak
- Nonlinear Feedback Shift Registers...
  - Have a nonlinear feedback function
  - Impossible to know if they have the maximum period
  - Cryptographically challenging to solve
- Modern approaches combine LFSRs with NFSRs

## Grain-128a

- Proposed as part of a public competition
- Combines LFSR and NFSR (as promised)
- Works bit-by-bit; Good for hardware optimizations

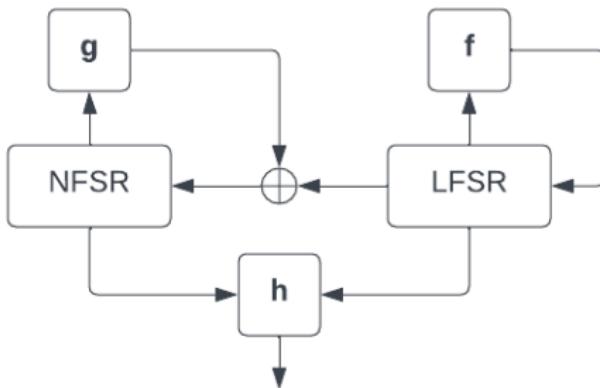
## Grain-128a

- Proposed as part of a public competition
- Combines LFSR and NFSR (as promised)
- Works bit-by-bit; Good for hardware optimizations



- No known devastating attacks on Grain-128a
  - Statistical attacks on chosen IV
  - Some related-key attacks
- Used in some smart cards. Compact and fast stream cipher

## Grain-128a - Continued



## Configuration

- LFSR has maximal period of  $2^{128} - 1$
- NFSR and nonlinear function  $h$  add cryptographic strength
- 128 bit key; 96 bit nonce
  - Key goes to NFSR internal state
  - Nonce goes to the LFSR, alongside 36 zero bits

## A5/1

- Used to encrypt voice communications over 2G
- Standard - 1987; Published - 1990; Attacks - 2000s
- Eventually broken in a practical sense
  - Some attacks are theoretical. They do not imply one can decrypt in real-time, or even in (pragmatically) short time

## A5/1

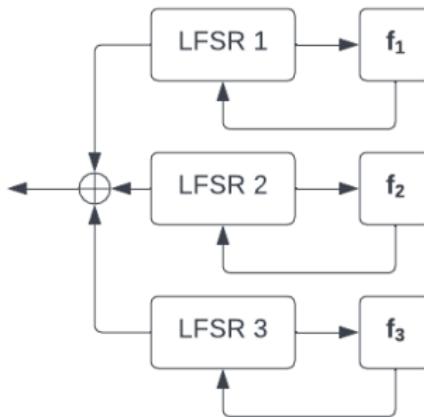
- Used to encrypt voice communications over 2G
- Standard - 1987; Published - 1990; Attacks - 2000s
- Eventually broken in a practical sense
  - Some attacks are theoretical. They do not imply one can decrypt in real-time, or even in (pragmatically) short time

## Configuration

- 64-bit key; 22-bit nonce
- Three LFSRs with 19, 22 and 23 bits
- Smart update mechanism with clocking rule
- Used to circumvent lack of NFSR

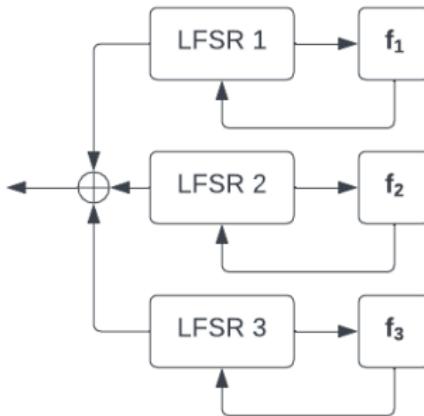
## A5/1 - Continued

- LFSRs have *clocking* bits
- Clocks registers whose bits are the majority value
- 2/3 clocked each update



## A5/1 - Continued

- LFSRs have *clocking* bits
- Clocks registers whose bits are the majority value
- 2/3 clocked each update



### Broken

- Subtle attacks
  - Internal linearity of A5/1
  - Simple irregular clocking system
- Brutal attacks: short key of A5/1

## RC4

- Takes 32/64-bit words instead of bits
- Proposed in 1987; reverse engineered in 1994
- Most widely used stream cipher
- Still used in numerous systems
  - ... despite being completely broken

# RC4

- Takes 32/64-bit words instead of bits
- Proposed in 1987; reverse engineered in 1994
- Most widely used stream cipher
- Still used in numerous systems
  - ... despite being completely broken

## Configuration

- One of the simplest ciphers ever created
- 128-bit key; No nonce  $\leftarrow ???$
- No crypto-like operations
- No XORs
- No multiplications
- We just swap bytes around

# RC4 - Continued

## Key Scheduling

---

```
j = 0
# set S to the array S[0] = 0, S[1] = 1, . . . , S[255] = 255
S = range(256)
# iterate over i from 0 to 255
for i in range(256):
    # compute the sum of v
    j = (j + S[i] + K[i % n]) % 256
    # swap S[i] and S[j]
    S[i], S[j] = S[j], S[i]
```

---

From *Serious Cryptography: A Practical Introduction to Modern Encryption*

- Initial state set to  $0, 1, \dots, 255$
- Afterwards, we run the keystream generation
- ... Which also consists in swaps

## RC4 - Continued

### Key Scheduling

---

```
j = 0
# set S to the array S[0] = 0, S[1] = 1, . . . , S[255] = 255
S = range(256)
# iterate over i from 0 to 255
for i in range(256):
    # compute the sum of v
    j = (j + S[i] + K[i % n]) % 256
    # swap S[i] and S[j]
    S[i], S[j] = S[j], S[i]
```

---

From *Serious Cryptography: A Practical Introduction to Modern Encryption*

- Initial state set to  $0, 1, \dots, 255$
- Afterwards, we run the keystream generation
- ... Which also consists in swaps

Looks absolutely trivial, but it took 20 years of cryptanalysis to find exploitable flaws.

## Attacks on RC4

Two main issues of RC4 have been exploited. No nonce in its original specification; and statistical biases on keystream generation

## Attacks on RC4

Two main issues of RC4 have been exploited. No nonce in its original specification; and statistical biases on keystream generation

### RC4 in WEP

- First generation Wi-Fi security protocol
- 24-bit nonce included in the wireless frame header
- **Problem #1!**  $2^{24/2} = 2^{12}$  (some megabytes of traffic) is usually enough to find two packets with the same nonce
- **Problem #2!** First keystream byte strongly dependent on first secret key byte – exploitable bias

### RC4 in TLS

## Attacks on RC4

Two main issues of RC4 have been exploited. No nonce in its original specification; and statistical biases on keystream generation

### RC4 in WEP

### RC4 in TLS

- Most important protocol on the internet
- 128-bit session key as nonce. Not as easily broken
- **Problem!** Statistical bias on the keystream. E.g. second keystream byte is zero with probability  $1/128 \neq 1/256$  (ideal)
- Statistically crucial given many ciphertexts of the same byte

## Salsa20

- Optimized for modern CPUs
- eSTREAM competition in 2005
- Counter-based stream cipher
- Used in many popular crypto libraries
  - Google Tink
  - LibSodium
  - OpenSSH

# Salsa20

- Optimized for modern CPUs
- eSTREAM competition in 2005
- Counter-based stream cipher
- Used in many popular crypto libraries
  - Google Tink
  - LibSodium
  - OpenSSH

## Configuration

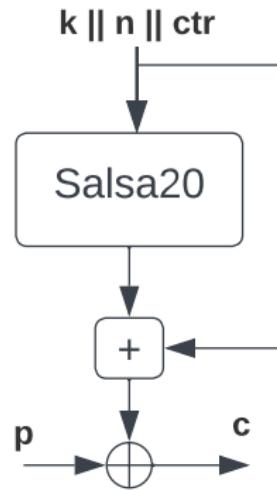
- Encrypts 512-bit plaintext blocks
- 256-bit key
- 64-bit nonce
- 64-bit counter
- 128-bit fixed constant words

# Salsa20

- Core transforms 512-bit blocks
- Adds the result to original value
- Otherwise insecure – invertible

## Quarter-Round Function

- Core transforms 32-bit words
- Structurally similar to AES
  - Column transformation
  - Row transformation



 Key Takeaways 

- Hardware-Oriented; bit-by-bit
- Good for HW optimization
  - Grain 128a
  - Combines LFSR and NFSR
  - Secure; Used in IoT
  - A5/1
  - Combines multiple LFSR
  - Very insecure; cracked after two decades

 Key Takeaways

- Hardware-Oriented; bit-by-bit
  - Good for HW optimization
    - Grain 128a
    - Combines LFSR and NFSR
    - Secure; Used in IoT
    - A5/1
    - Combines multiple LFSR
    - Very insecure; cracked after two decades
  - Software-Oriented; word-by-word
  - Better for servers/browsers
    - RC4
    - Bare specification; does not include nonce
    - WEP and TLS. Broken (for different reasons)
    - Salsa20
    - Modern cipher permuting 32-bit words
    - Very fast; accepted in TLS 1.3

# Applied Cryptography

## Week 4: Stream Ciphers

Bernardo Portela

M:ERSI, M:SI, M:CC - 25