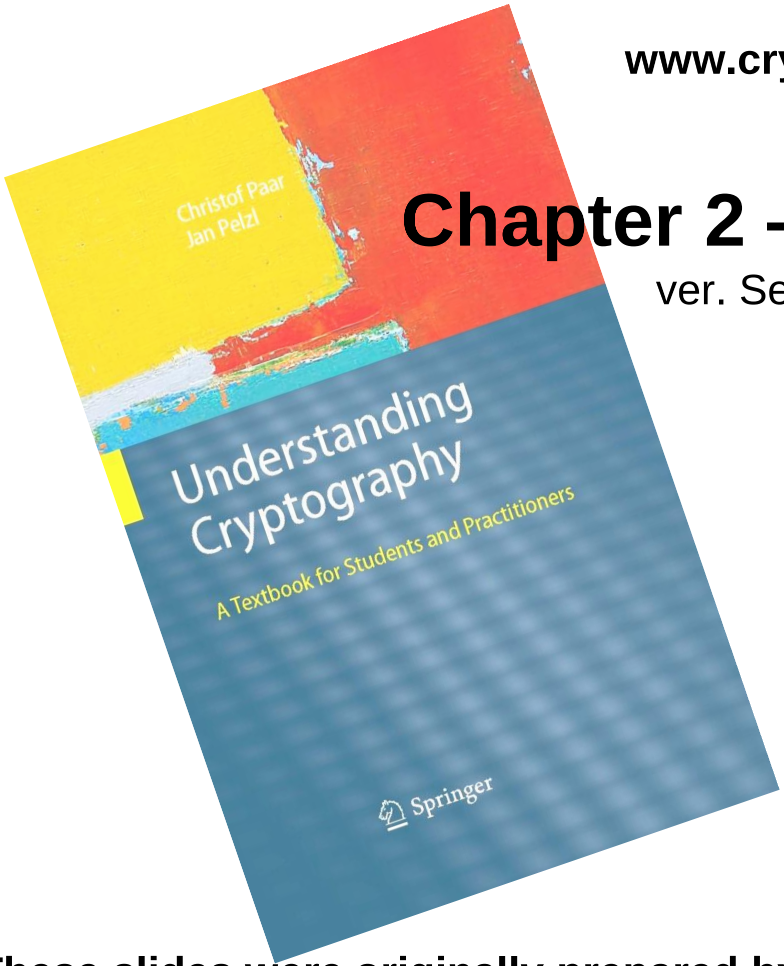
Understanding Cryptography – A Textbook for Students and Practitioners

by Christof Paar and Jan Pelzl

www.crypto-textbook.com


Christof Paar
Jan Pelzl
Understanding Cryptography
A Textbook for Students and Practitioners
Springer

Chapter 2 – Stream Ciphers

ver. September 17, 2024

These slides were originally prepared by Thomas Eisenbarth, Christof Paar and Jan Pelzl. Later, they were modified by Tomas Fabsic for purposes of teaching I-ZKRY at FEI STU.

Homework

- Read Chapter 2.
- Solve problems from the exercises set no. 2 and submit them to AIS by **30.9.2024 23:59**.

Some legal stuff (sorry): Terms of Use

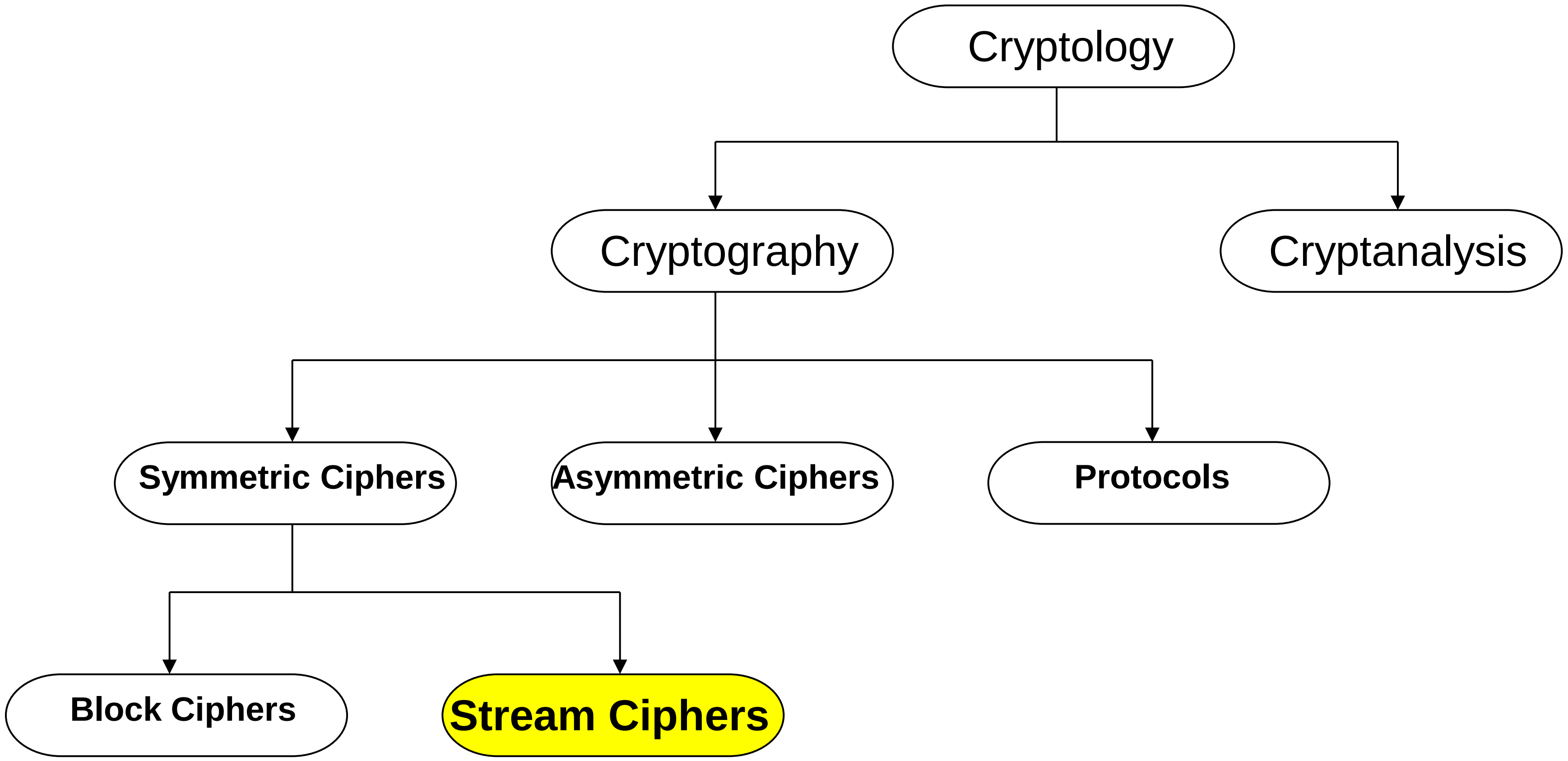
Content of this Chapter

- Intro to stream ciphers
- Random number generators (RNGs)
- Linear feedback shift registers (LFSRs)
- Modern stream ciphers

Content of this Chapter

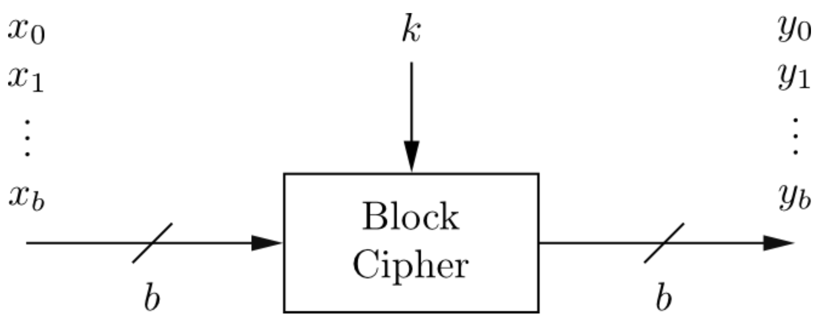
- **Intro to stream ciphers**
- Random number generators (RNGs)
- Linear feedback shift registers (LFSRs)
- Modern stream ciphers

Stream Ciphers in the Field of Cryptology



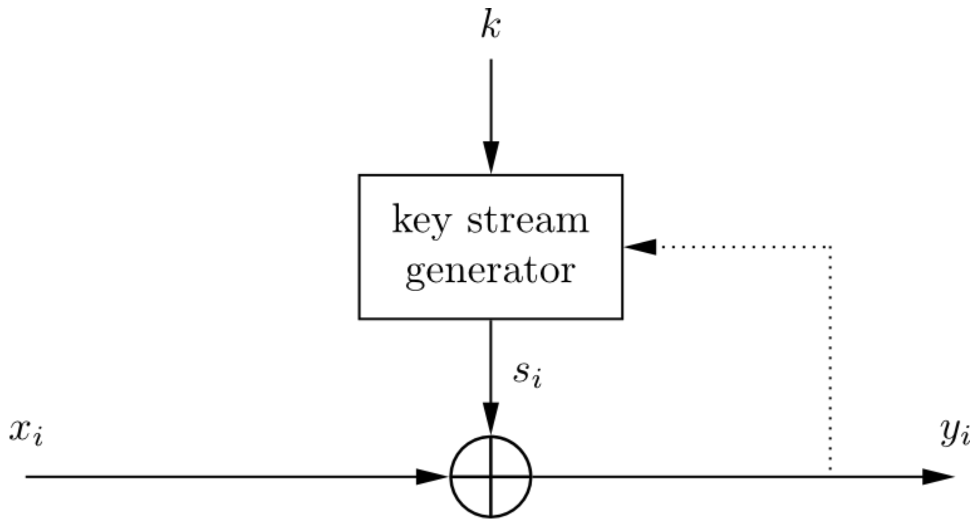
Stream Ciphers were invented in 1917 by Gilbert Vernam

Stream Cipher vs. Block Cipher



- **Block Ciphers:**
  - Always encrypt a full block (several bits)
  - The encryption of any plaintext bit in a given block depends on every other plaintext bit in the same block.
  - Block size is usually 128 bits.
  - Are more common for Internet applications

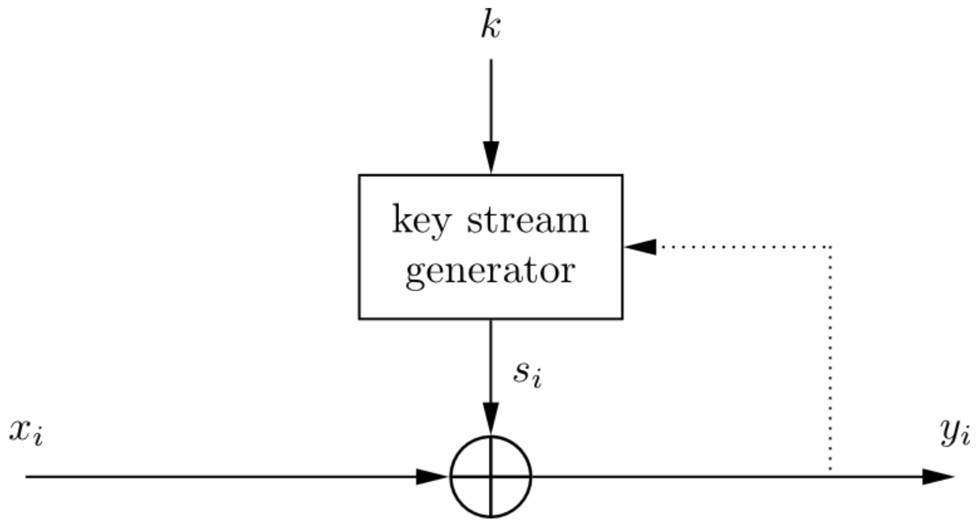
Stream Cipher vs. Block Cipher



(key stream generator = generátor prúdu bitov)

- **Stream Ciphers**
  - Encrypt bits individually
  - In the past, they were particularly relevant for applications with low computational resources, e.g., for cell phones or other small embedded devices.
  - Today, however, there exist also block ciphers with very low computational requirements (e.g. PRESENT).
  - At the same time, there are modern stream ciphers that are very well suited for high-speed software implementations (e.g. ChaCha).

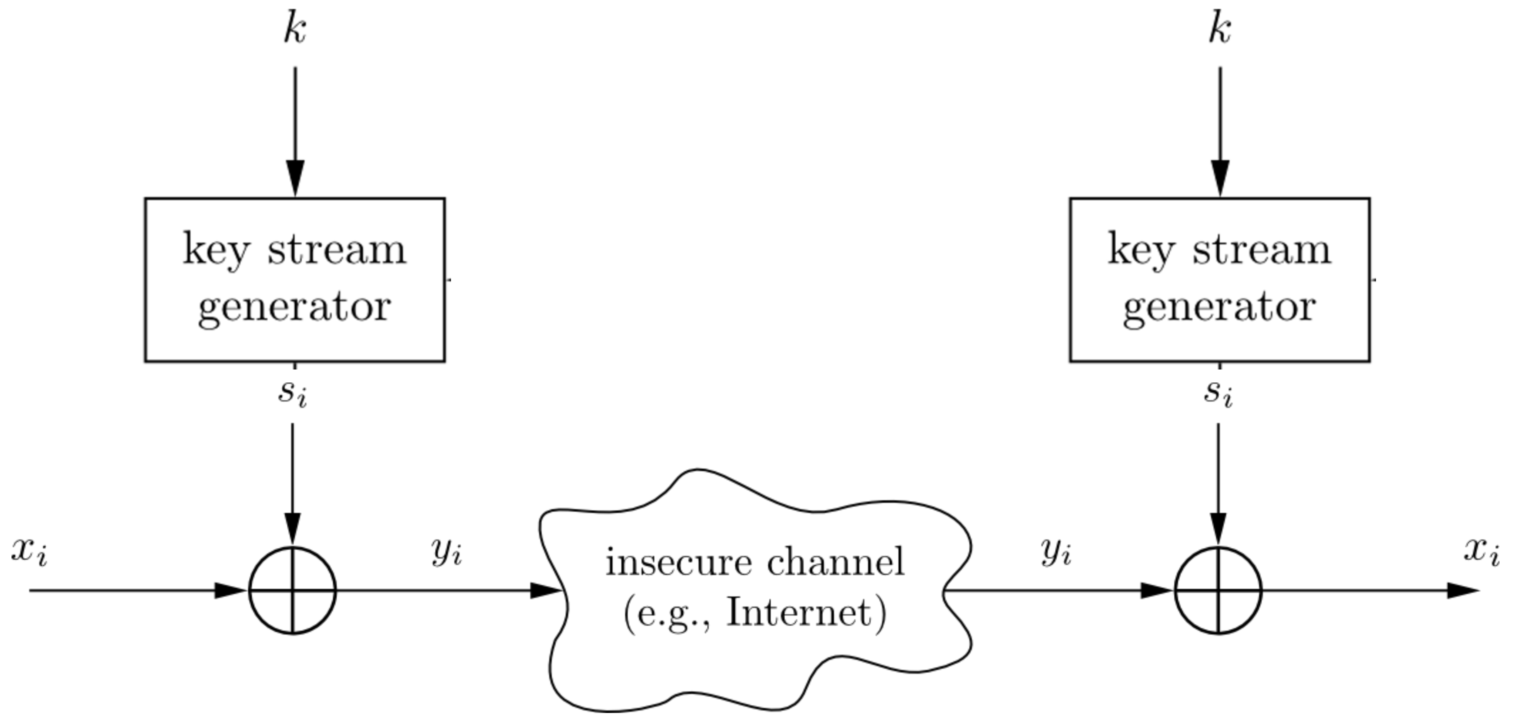
Synchronous vs. Asynchronous Stream Cipher



- **Synchronous Stream Cipher**
  - Key stream depend only on the key (and possibly an initialization vector IV)
  - Most modern stream ciphers are synchronous.
- **Asynchronous Stream Ciphers**
  - Key stream depends also on the ciphertext (dotted feedback enabled)

Encryption and Decryption with Stream Ciphers

Plaintext x_i, ciphertext y_i and key stream 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:** $x_i = e_{si}(y_i) = y_i + s_i \bmod 2$

Why is Modulo 2 Addition a Good Encryption Function?

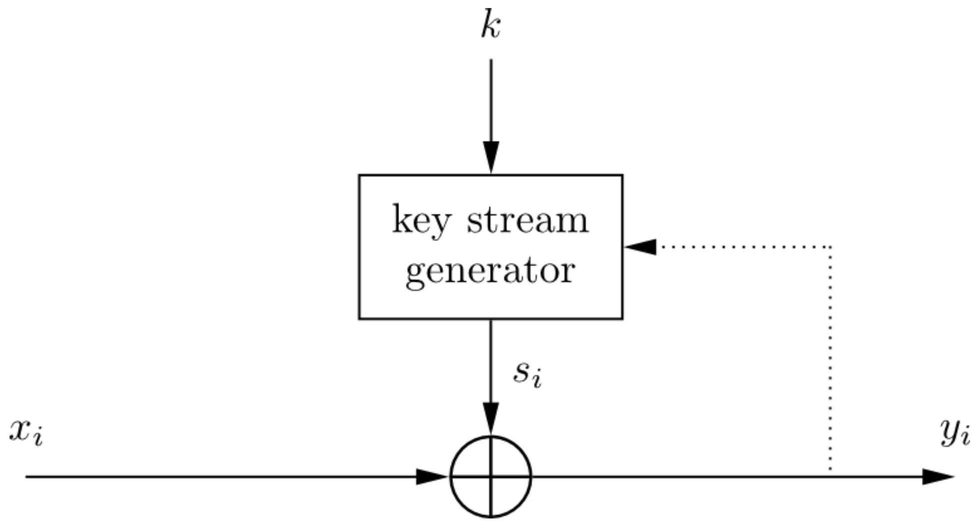
- Modulo 2 addition is equivalent to XOR operation
- For perfectly random key stream s_i , each ciphertext output bit has a 50% chance to be 0 or 1

  → Good statistic property for ciphertext
- Inverting XOR is simple, since it is the same XOR operation

x_i	s_i	y_i
0	0	0
0	1	1
1	0	1
1	1	0

- Imagine if stream ciphers used modulo 2 multiplication instead of modulo 2 addition. Why wouldn't it work?

How can a stream cipher be secure?



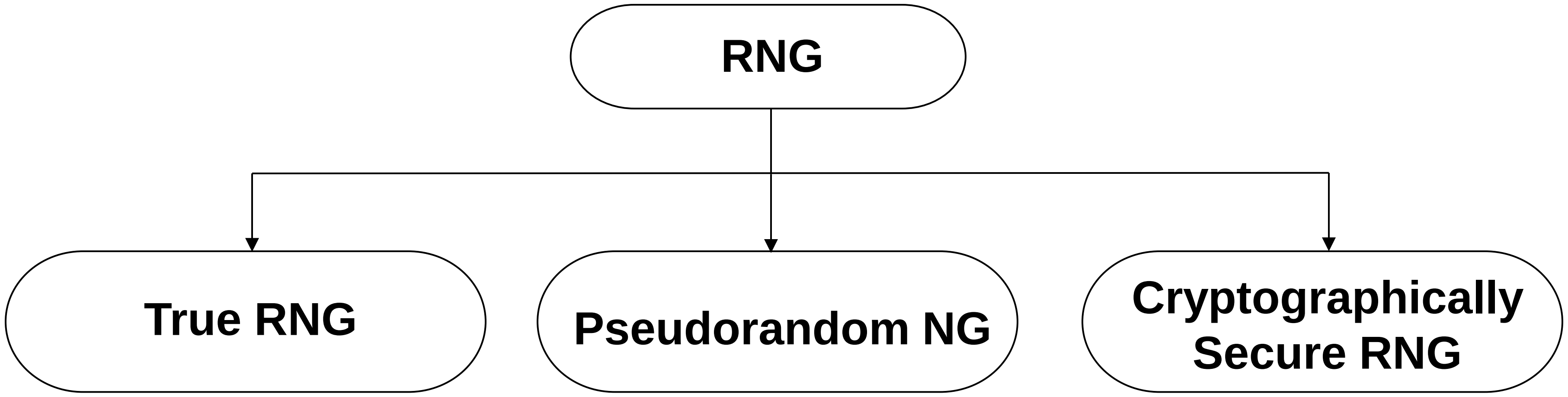
- ***Security of stream cipher depends entirely on the key stream s_i*** :
    - Should be ***random*** , i.e., Pr(s_i = 0) = Pr(s_i = 1) = 0.5
    - Must be ***reproducible*** by sender and receiver

- Important question: ***How do we build the key stream?***

Content of this Chapter

- Intro to stream ciphers
- **Random number generators (RNGs)**
- Linear feedback shift registers (LFSRs)
- Modern stream ciphers

- **Random number generators (RNGs)**



True Random Number Generators (TRNGs)

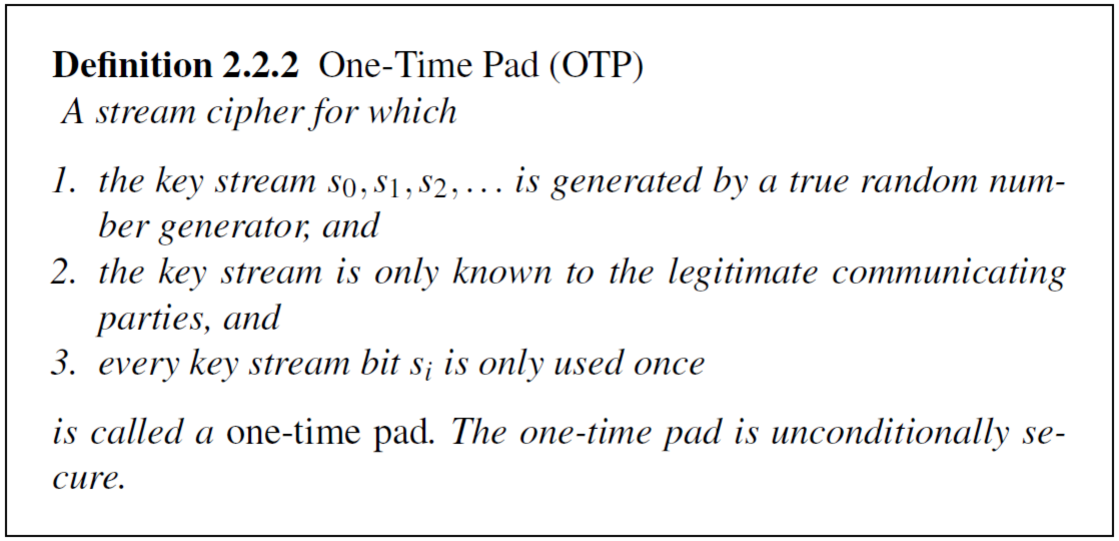
- Based on physical random processes: coin flipping, dice rolling, semiconductor noise, radioactive decay, mouse movement, clock jitter of digital circuits
- Output stream s_i should have good statistical properties:
  $Pr(s_i = 0) = Pr(s_i = 1) = 50\%$ (often achieved by post-processing)
- Output can neither be predicted nor be reproduced
- Typically used for generation of keys, nonces (number used only-once) and for many other purposes
- Disadvantage: they are slow

- ***Can we use TRNGs to build the key stream?***
  - Problem is that the output from TRNG is not reproducible.
  - In order to decrypt, the recipient has to learn the whole key stream through a secure channel.
  - But the key stream is as large as the message -> impractical.

One-Time Pad (OTP)

A stream cipher which uses a TRNG to produce the key stream is called a

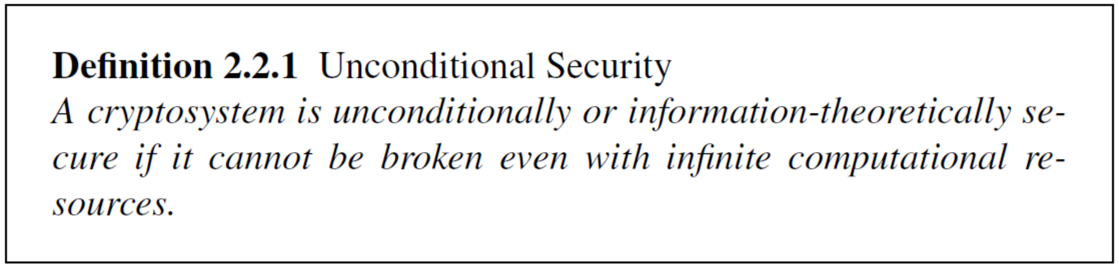
one-time pad (OTP) (Vernamova šifra)



Definition 2.2.2 One-Time Pad (OTP)
A stream cipher for which

1. *the key stream $s_0, s_1, s_2, \ldots$ is generated by a true random number generator, and*
2. *the key stream is only known to the legitimate communicating parties, and*
3. *every key stream bit s_i is only used once*

is called a one-time pad. *The one-time pad is unconditionally secure.*



Definition 2.2.1 Unconditional Security
A cryptosystem is unconditionally or information-theoretically secure if it cannot be broken even with infinite computational resources.

OTP is unconditionally secure if and only if the key k_i. is used once!

One-Time Pad (OTP)

Unconditionally secure cryptosystem:

$$y_0 = x_0 \oplus k_0$$
$$y_1 = x_1 \oplus k_1$$
$$\vdots$$

Every equation is a linear equation with two unknowns

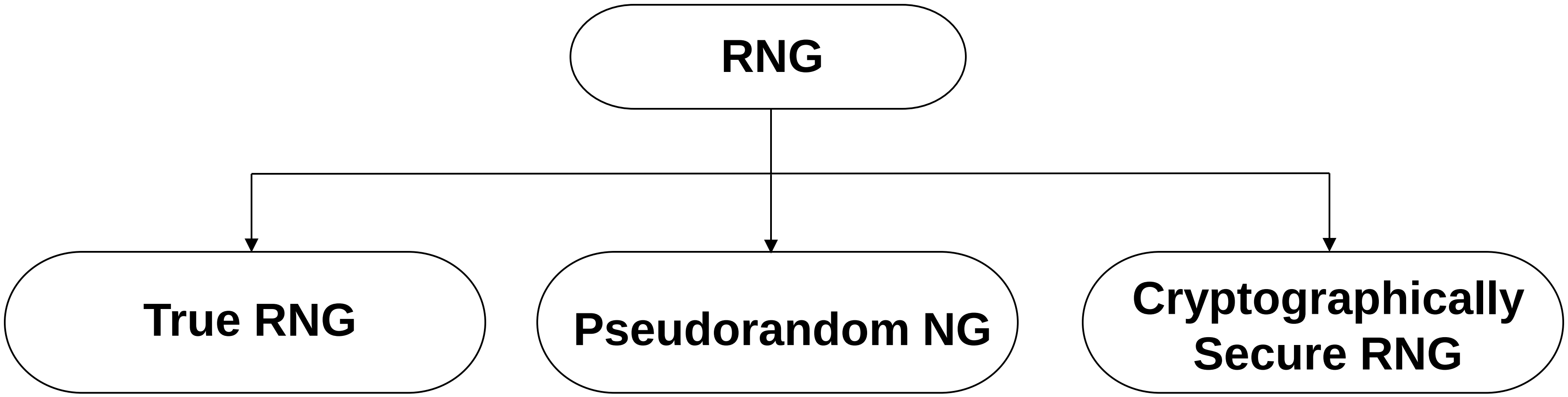
$\Rightarrow$ for every y_i are $x_i = 0$ and $x_i = 1$ equiprobable!

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

$\Rightarrow$ 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 email attachment.)

- **Random number generators (RNGs)**



Pseudorandom Number Generator (PRNG)

- Generate sequences from initial seed value
- Typically, output stream has good statistical properties
- Output can be reproduced and can be predicted

Often computed in a recursive way:

$$s_0 = seed$$

$$s_{i+1} = f(s_i, s_{i-1}, \ldots, s_{i-t})$$

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

$$s_0 = 12345$$

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

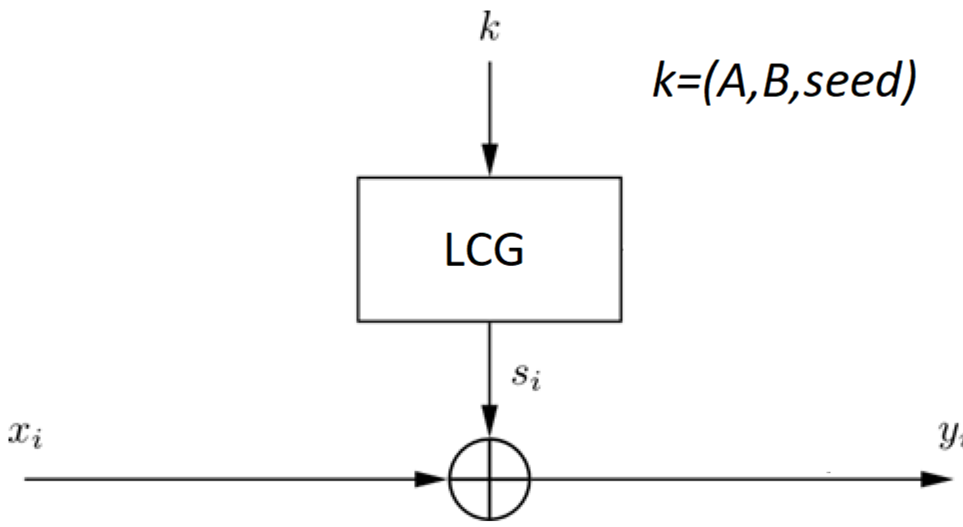
Most PRNGs have bad cryptographic properties!

Cryptanalyzing a Stream Cipher Based on a Simple PRNG

Simple PRNG: **Linear Congruential Generator (LCG)**

$$S_0 = seed$$

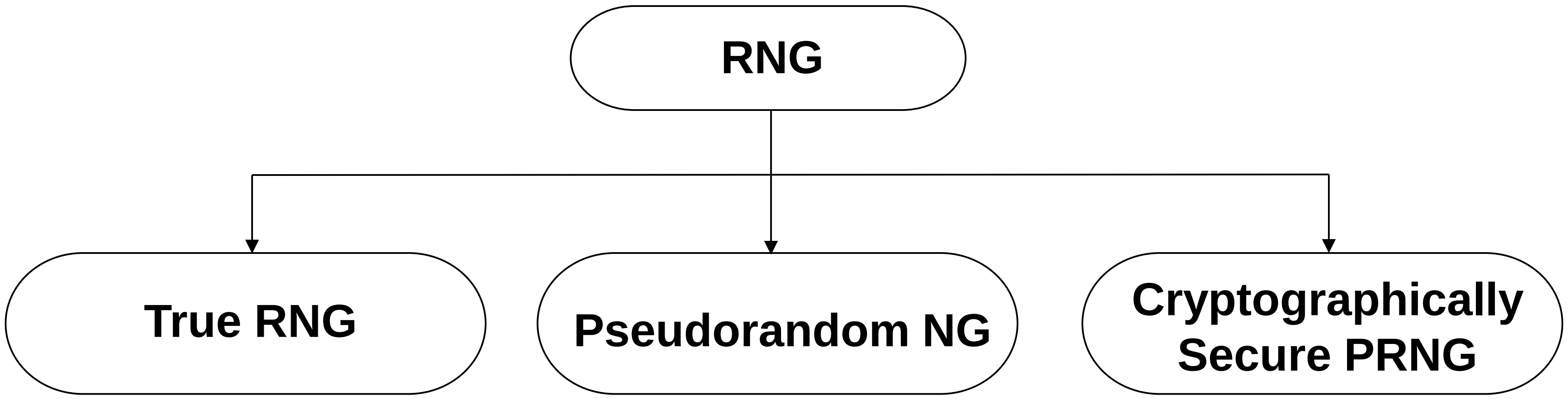
$$S_{i+1} = AS_i + B \bmod m$$



Assume

- unknown *A*, *B and* S_0 as key
- Size of *A*, *B* and S_i to be 100 bit
- Ciphertext is known
- First 300 bits of plaintext are known (e.g. header of the file)

- **Random number generators (RNGs)**

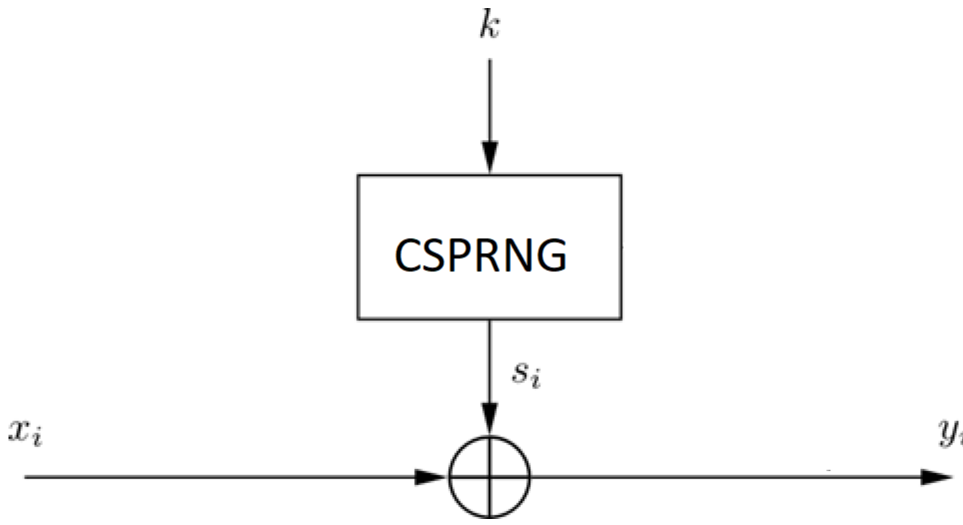


Cryptographically Secure Pseudorandom Number Generator (CSPRNG)

- 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).

- **Remark:** There are almost no other applications that need unpredictability, whereas many, many (technical) systems need PRNGs.
- It is CSPRNGs that are used in stream ciphers to generate key streams!



Uses of CSPRNGs

- CSPRNGs are needed in cryptography for:
  - stream ciphers
  - fast generation of keys for symmetric ciphers (also for block ciphers)
  - fast generation of “random” bits (e.g. nonces)
- Often, CSPRNG is used together with TRNG:
  - Firstly, TRNG generates a seed.
  - Then, CSPRNG expands the seed into more bits.

Constructions of CSPRNGs

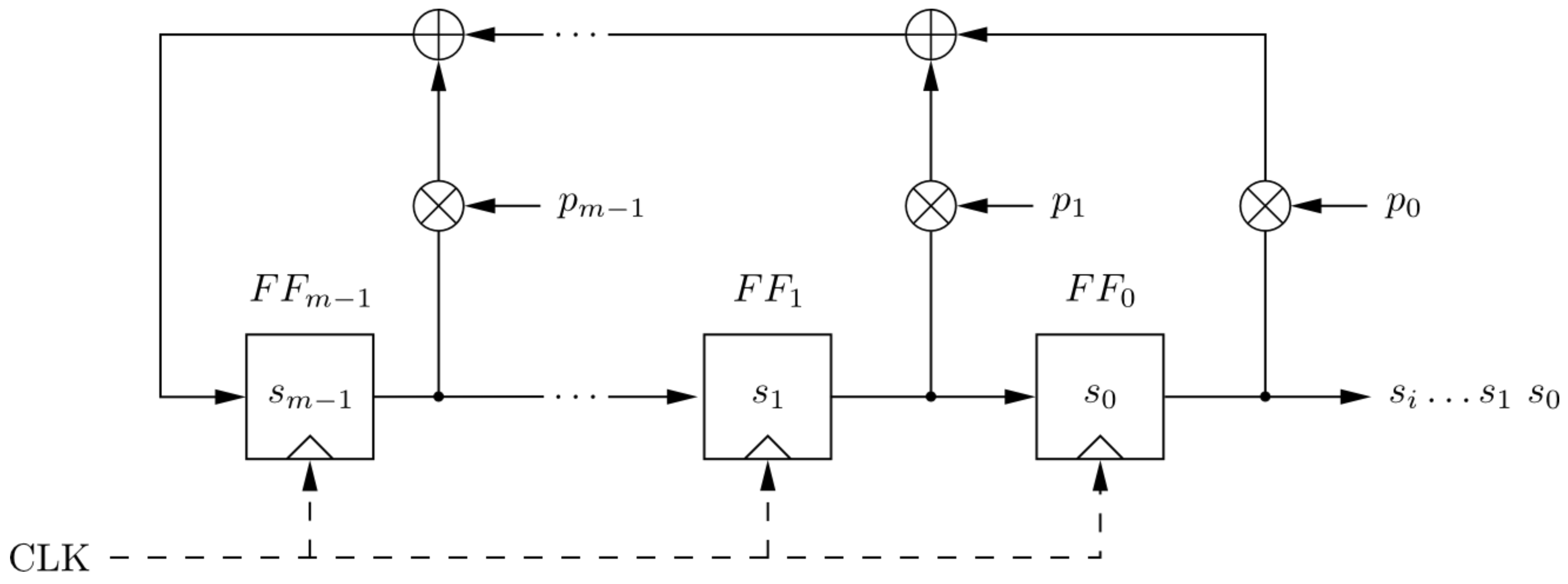
Popular constructions of CSPRNGs:

- Based on linear feedback shift registers (LFSRs)
  - Will be discussed next.
- Based on add-rotate-XOR (ARX) approach.
  - We will see this when we discuss stream ciphers Salsa20 and ChaCha.
- Based on block ciphers.

Content of this Chapter

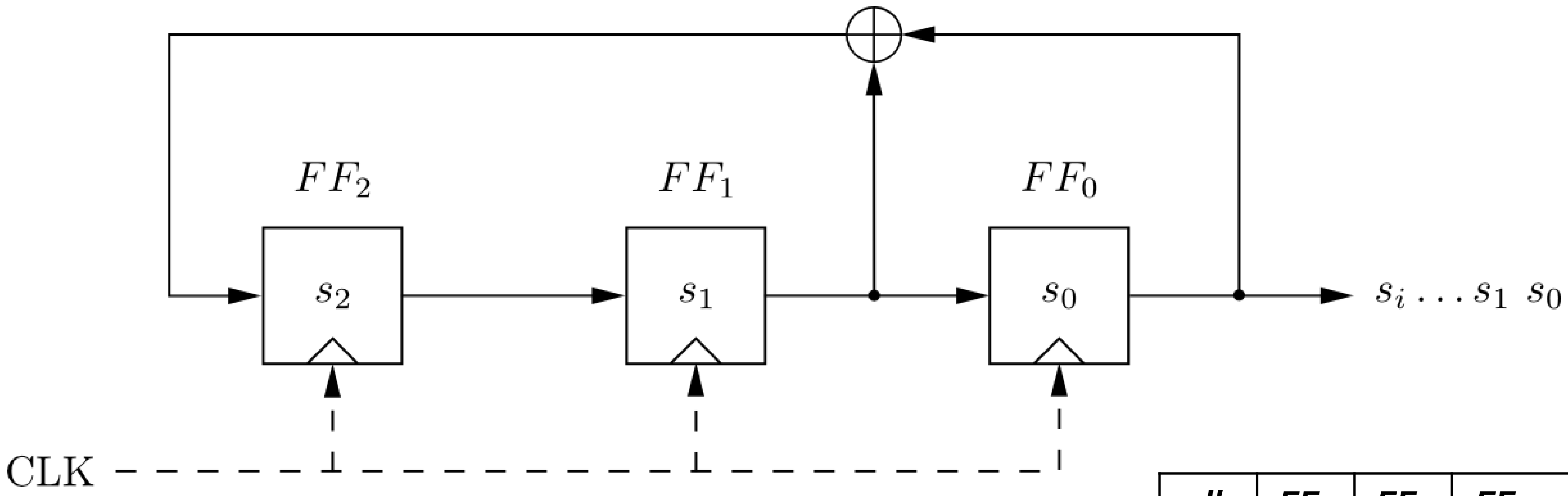
- Intro to stream ciphers
- Random number generators (RNGs)
- **Linear feedback shift registers (LFSRs)**
- Modern stream ciphers

Linear Feedback Shift Registers (LFSRs)



- Concatenated *flip-flops (FF*), i.e., a shift register together with a feedback path
- Feedback computes fresh input by XOR of certain state bits
- *Degree* *m* given by number of storage elements
- If p_i = 1, the feedback connection is present ("closed switch), otherwise there is not feedback from this flip-flop ("open switch")
- Output sequence repeats periodically
- Maximum output length**:** 2^m-1

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



clk	*FF₂*	*FF₁*	*FF₀=sᵢ*
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

- LFSR output described by recursive equation:

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

- Maximum output length (of 2^3-1=7) achieved only for certain feedback configurations, .e.g., the one shown here.

Security of LFSRs

LFSRs typically described by polynomials:

$$P(x) = x^m + p_{l-1}x^{m-1} + \ldots + p_1 x + p_0$$

- Single LFSRs generate highly predictable output
- If 2*m* output bits of an LFSR of degree *m* are known, the feedback coefficients p_i of the LFSR can be found by solving a system of linear equations*
- Because of this many stream ciphers use **combinations** of LFSRs

*See Chapter 2 of *Understanding Cryptography* for further details.

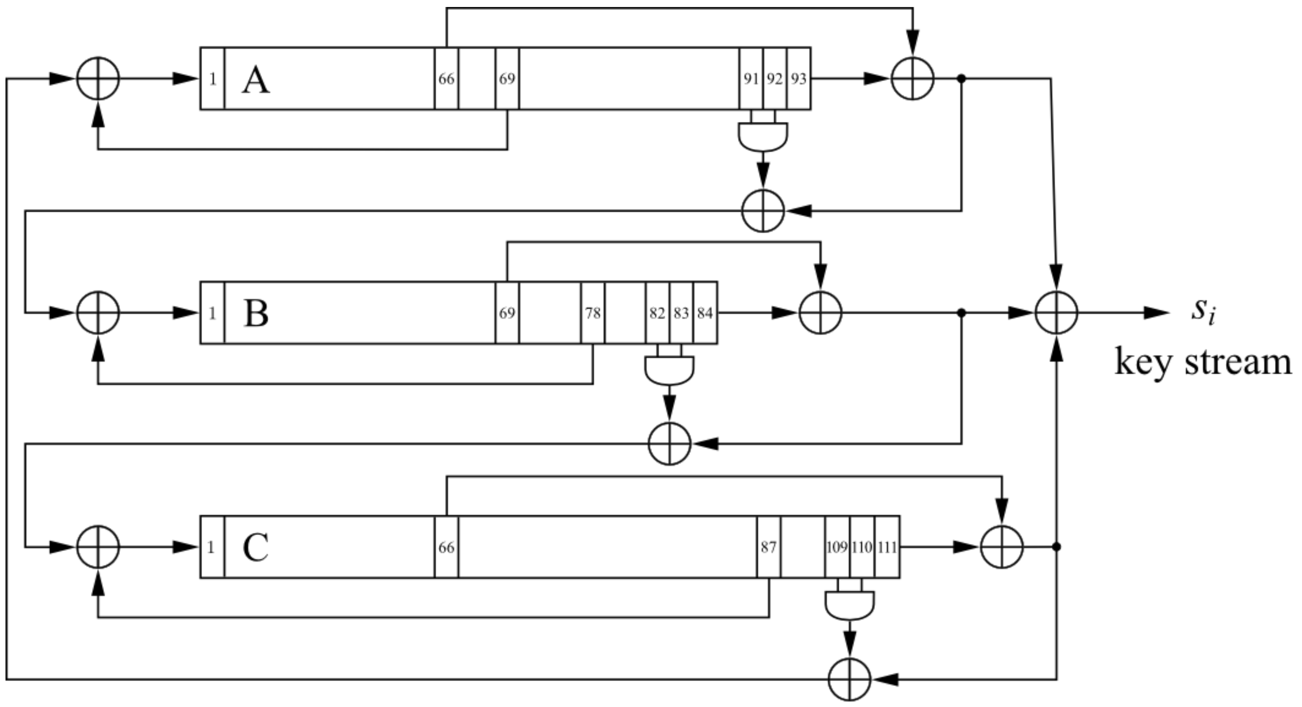
Content of this Chapter

- Intro to stream ciphers
- Random number generators (RNGs)
- Linear feedback shift registers (LFSRs)
- **Modern stream ciphers**

Initialization Vectors (IVs) (Inicializačné vektory)

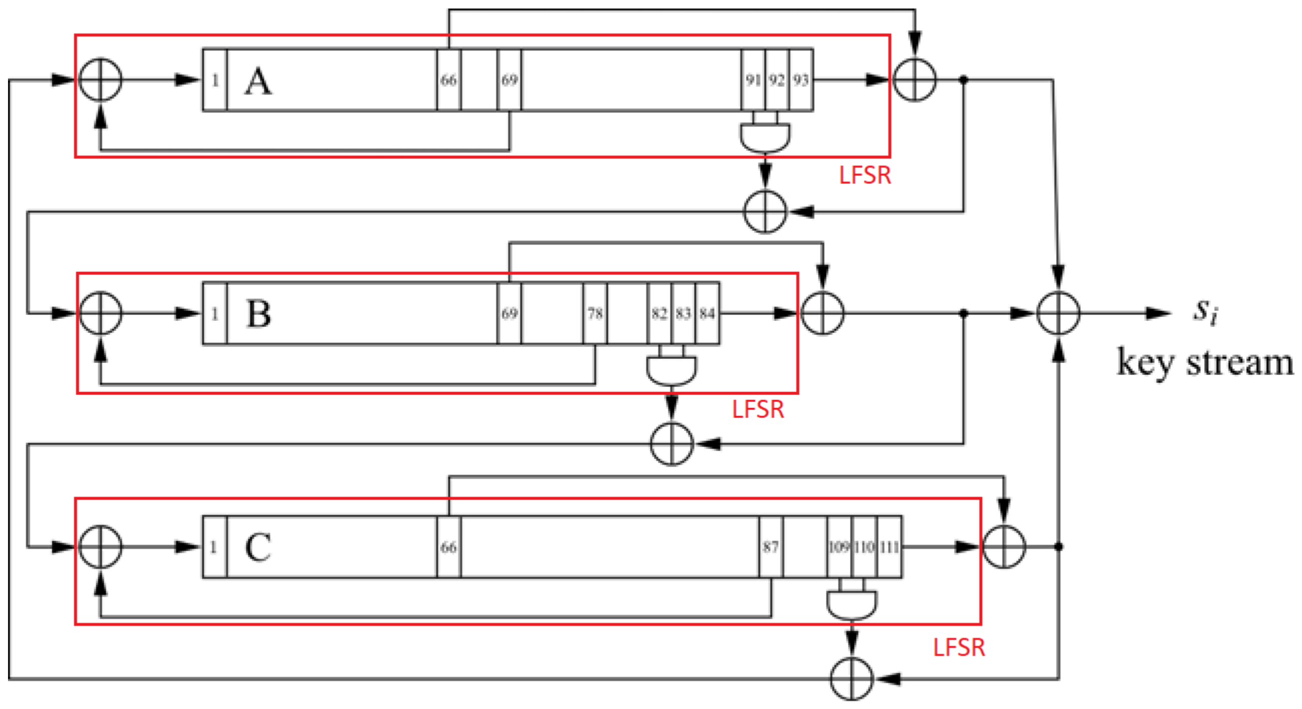
- Used in many stream cipher constructions.
- IV serves as a randomizer and should take a new value for every encryption session.
- Their main purpose is that two key streams produced by the cipher should be different, even though the key has not changed.
- IVs do not have to be kept secret.
- Methods for generating IVs are discussed in Section 5.1.2. of the book.

Trivium



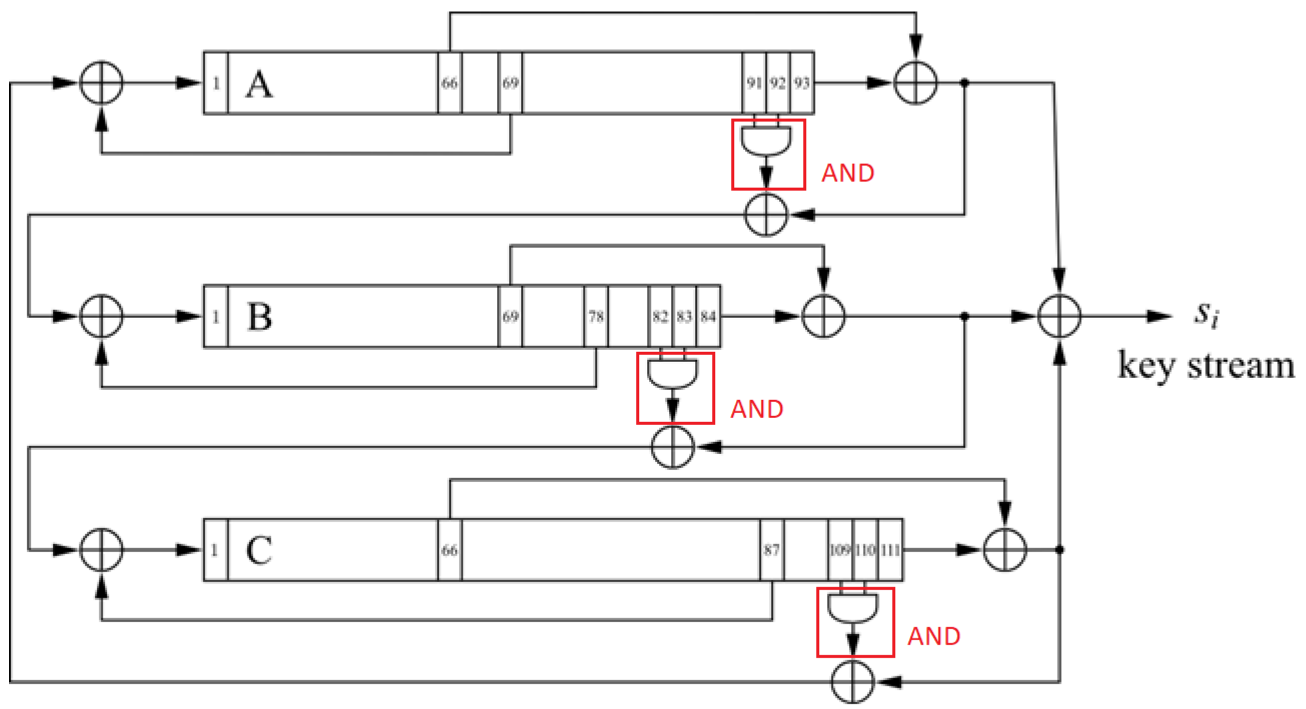
- Three LFSRs of length 93, 84, 111
- XOR-Sum of all three LFSR outputs generates key stream s_i
- Small in Hardware:
  - Total register count: 288
  - Non-linearity: 3 AND-Gates
  - 7 XOR-Gates (4 with three inputs)

Trivium



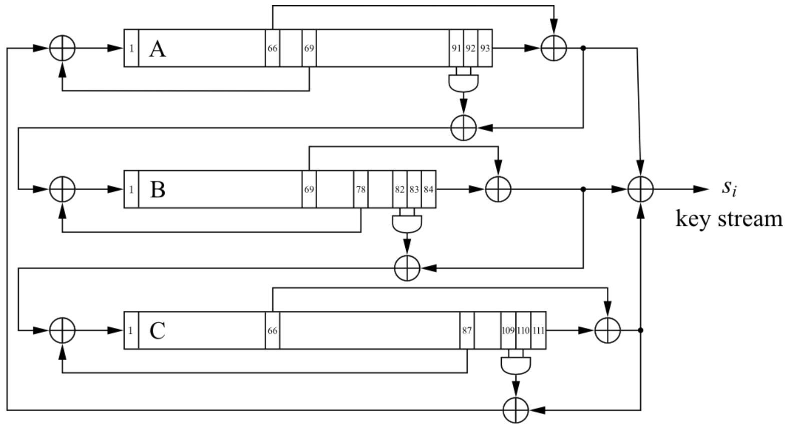
- Three LFSRs of length 93, 84, 111
- XOR-Sum of all three LFSRs outputs generates key stream s_i
- Small in Hardware:
  - Total register count: 288
  - Non-linearity: 3 AND-Gates
  - 7 XOR-Gates (4 with three inputs)

Trivium



- Three LFSRs of length 93, 84, 111
- XOR-Sum of all three LFSR outputs generates key stream s_i
- Small in Hardware:
  - Total register count: 288
  - Non-linearity: 3 AND-Gates
  - 7 XOR-Gates (4 with three inputs)

Operation of Trivium



Initialization:

- Load 80-bit IV into A
- Load 80-bit key into B
- Set c_{109} , c_{110} , c_{111} =1, all other bits 0

Warm-Up:

- Clock cipher 4 x 288 = 1152 times without generating output

Encryption:

- XOR-Sum of all three LFSR outputs generates key stream s_i

Properties:

- Can produce up to 2^{64} bits of output from an 80-bit key and an 80-bit IV.
- Was developed to be a very small and efficient cipher and is **not** intended for high-security applications!

Salsa20

- Salsa20 is a family of software-efficient stream ciphers.
- Based on add-rotate-XOR (ARX) approach.
- The original cipher has 20 rounds and is denoted by Salsa20/20.
- Salsa20/20 is already faster than AES (the most popular block cipher) on most CPUs.
- Salsa20 variants with a reduced round count, named Salsa20/12 and Salsa20/8, are even faster.
- In the following we will describe Salsa20 with 20 rounds.

Salsa20

- Supports key lengths of 256 and 128 bits. (256 is recommended by the designer)
- The core of Salsa20 is a function with a 512-bit input and a 512- bit output.
- For both encryption and decryption, Salsa20 processes:
    - the key (we will assume 256-bit key),
    - a 64-bit nonce (has the role of IV)
    - and a 64-bit block number,

  and generates a 512-bit block of key stream. (one can encrypt (or decrypt) 512 plaintext or ciphertext bits at once.
- Thus, for one choice of nonce Salsa20 can generate a key stream of length:

$$2^{64} * 512 = 2^{73} \text{ bits}$$

- Since each block depends only on the key, the nonce and the block number, the key stream blocks can be computed independently of each other and blocks can be computed in parallel. (advantageous for high-speed implementations).

Salsa20 – internal state

- The internal state of Salsa20 has 512 bits.
- It consists of sixteen 32-bit words u_i and can be arranged as a 4-by-4 matrix:

u_0	u_1	u_2	u_3
u_4	u_5	u_6	u_7
u_8	u_9	u_{10}	u_{11}
u_{12}	u_{13}	u_{14}	u_{15}

Salsa20 – initial state

- Eight 32-bit words are formed by the key k=[$k_0 k_1$ … k_7].
- Two words indicate the stream position p = [$p_0 p_1$].

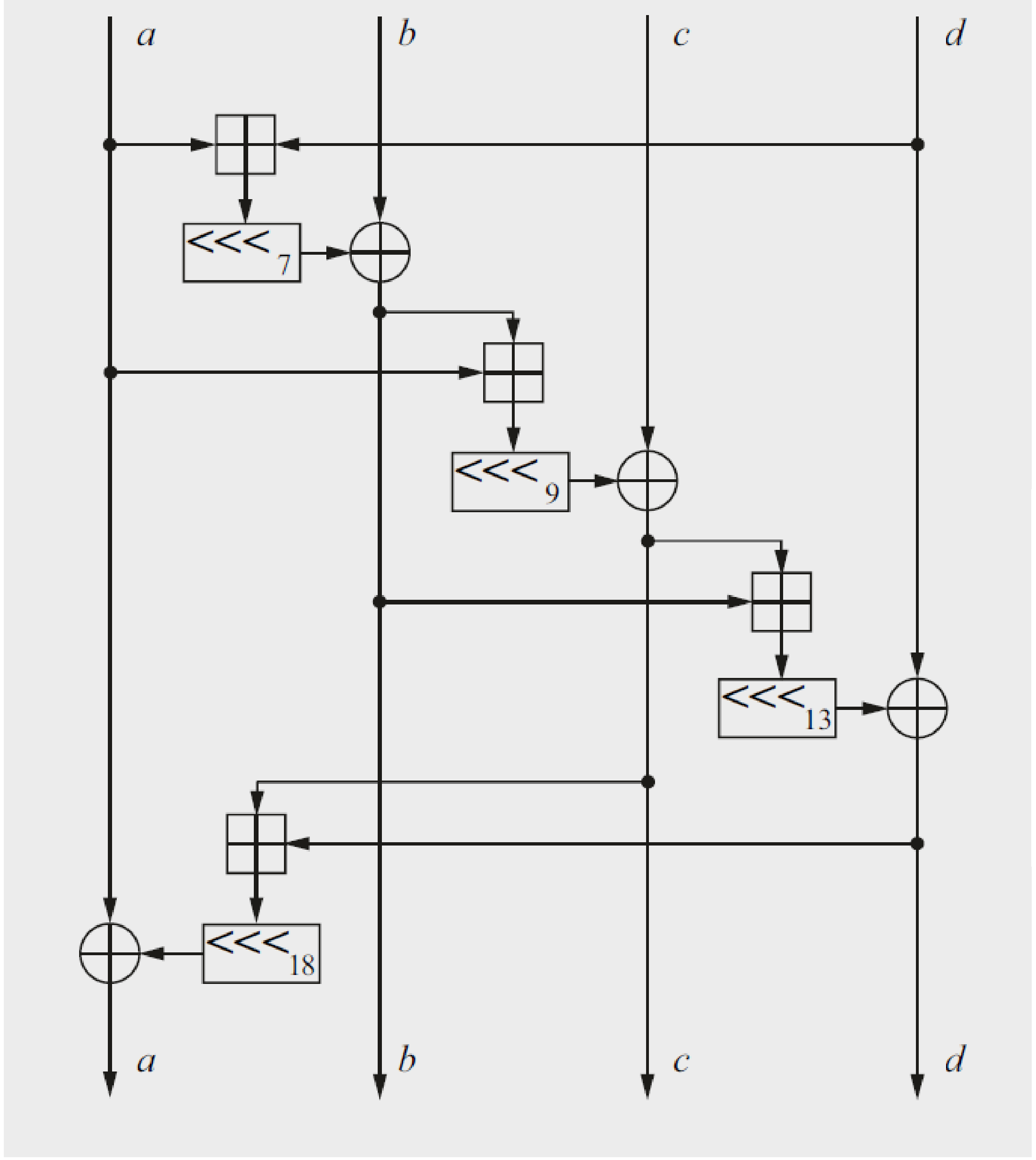
(p can be seen as a counter indicating the position of the current 512-bit block within the range of all 2^{64} 512-bit blocks of the key stream)

- Two words come from the nonce n = [$n_0 n_1$].
- Four words are a constant c = [$c_0 c_1 c_2 c_3$] (the constant c is given by the ASCII encoded string "expand 32-byte k").

c_0	k_0	k_1	k_2
k_3	c_1	n_0	n_1
p_0	p_1	c_2	k_4
k_5	k_6	k_7	c_3

Salsa20 – QR function

- The core operation in Salsa20 is the quarter-round function QR(a,b,c,d).
- It repeatedly applies three simple operations on 32-bit words:
  - 32-bit addition modulo 2^{32},
  - 32-bit XOR
  - and a constant 32-bit rotation by c positions to the left $ROTL^c$.
- The four-word output is computed from a four-word input by the quarter-round function QR as shown in Figure 2.8.

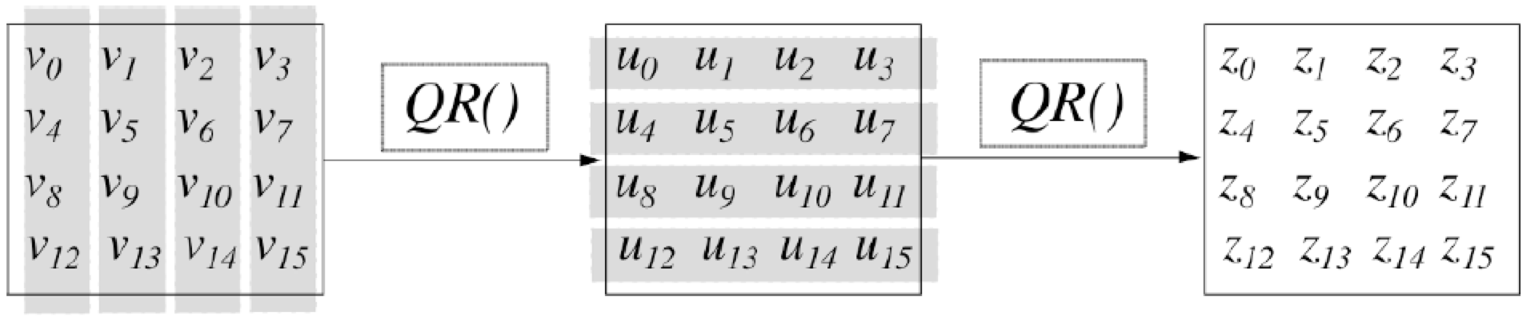


$$b = b \oplus ROTL^{7}(a+d)$$
$$c = c \oplus ROTL^{9}(b+a)$$
$$d = d \oplus ROTL^{13}(c+b)$$
$$a = a \oplus ROTL^{18}(d+c)$$

Fig. 2.8 Quarter-round function $QR(a,b,c,d)$ of Salsa20

Salsa20 - rounds

- Four quarter rounds form (not surprisingly) a round.
- Two consecutive rounds are called a double-round.
- In odd-numbered rounds, QR is applied to each of the four columns in the 4-by-4 matrix.
- In even-numbered rounds, QR is applied to each of the four rows.



v_0 v_1 v_2 v_3
v_4 v_5 v_6 v_7
v_8 v_9 v_{10} v_{11}
v_{12} v_{13} v_{14} v_{15}
$QR()$
u_0 u_1 u_2 u_3
u_4 u_5 u_6 u_7
u_8 u_9 u_{10} u_{11}
u_{12} u_{13} u_{14} u_{15}
$QR()$
z_0 z_1 z_2 z_3
z_4 z_5 z_6 z_7
z_8 z_9 z_{10} z_{11}
z_{12} z_{13} z_{14} z_{15}

Fig. 2.9 Double-round function of Salsa20

- For encryption or decryption, 20 rounds or 10 double-rounds are applied.

ChaCha

- ChaCha is another fast, software-oriented stream cipher.
- It follows the same basic design principles as Salsa20.
- The cipher can be configured with eight (ChaCha8), twelve (ChaCha12), or twenty rounds (ChaCha20).

Lessons Learned

- Stream ciphers are an important part of modern cryptography but are somewhat less widely used than block ciphers.
- The one-time pad is a provably secure symmetric cipher. However, it is highly impractical for most applications because the key length has to equal the message length.
- Stream ciphers sometimes require fewer resources, e.g., code size or chip area, for implementation than block ciphers, and they can be very fast.
- Secure and fast stream ciphers such as ChaCha20 can be built from functions that consist of the add-rotate-XOR operations.
- The requirements for a cryptographically secure pseudorandom number generator are far more demanding than the requirements for pseudorandom number generators used in other fields of engineering such as testing or simulation.
- Single LFSRs make poor stream ciphers despite their good statistical properties. However, careful combination of several LFSRs can yield strong ciphers.