CRYPTOGRAPHY AND SECURITY

Practice

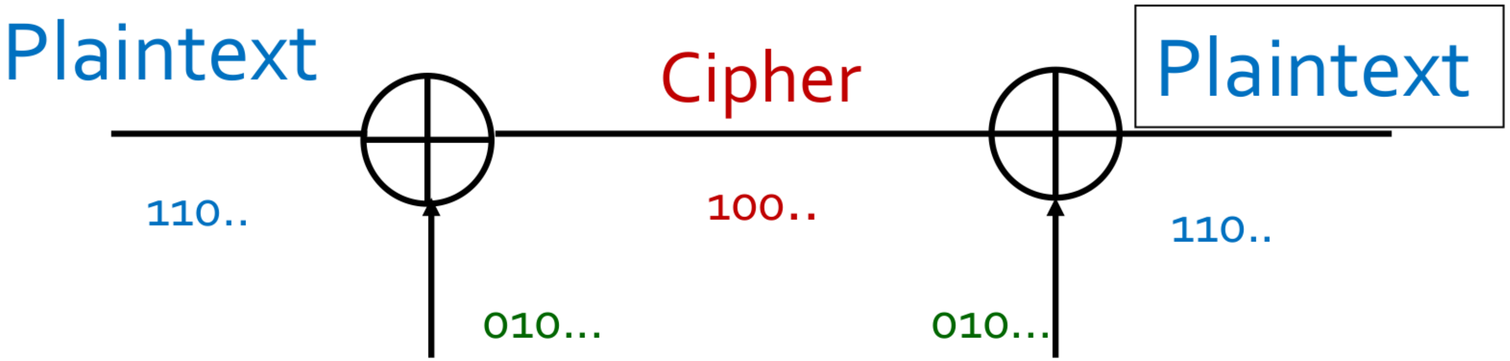
IP-18FKVKRBG

Lecture 6

2022-2023-2

Stream Cipher

OTP: One Time Pad



- Provable secure provided

  - Key K is random

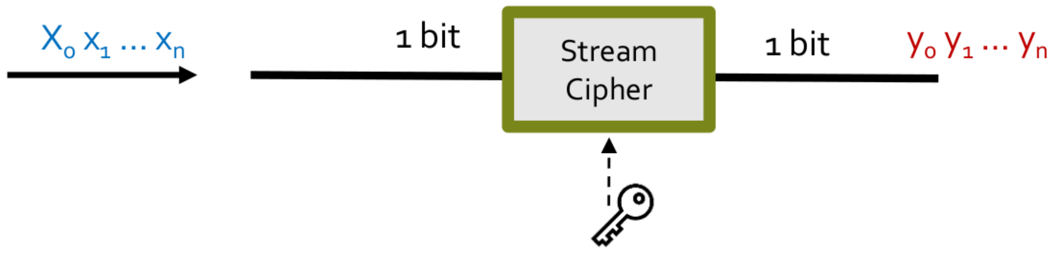
  - Key K is as long as the message

  - Key K is used only one time

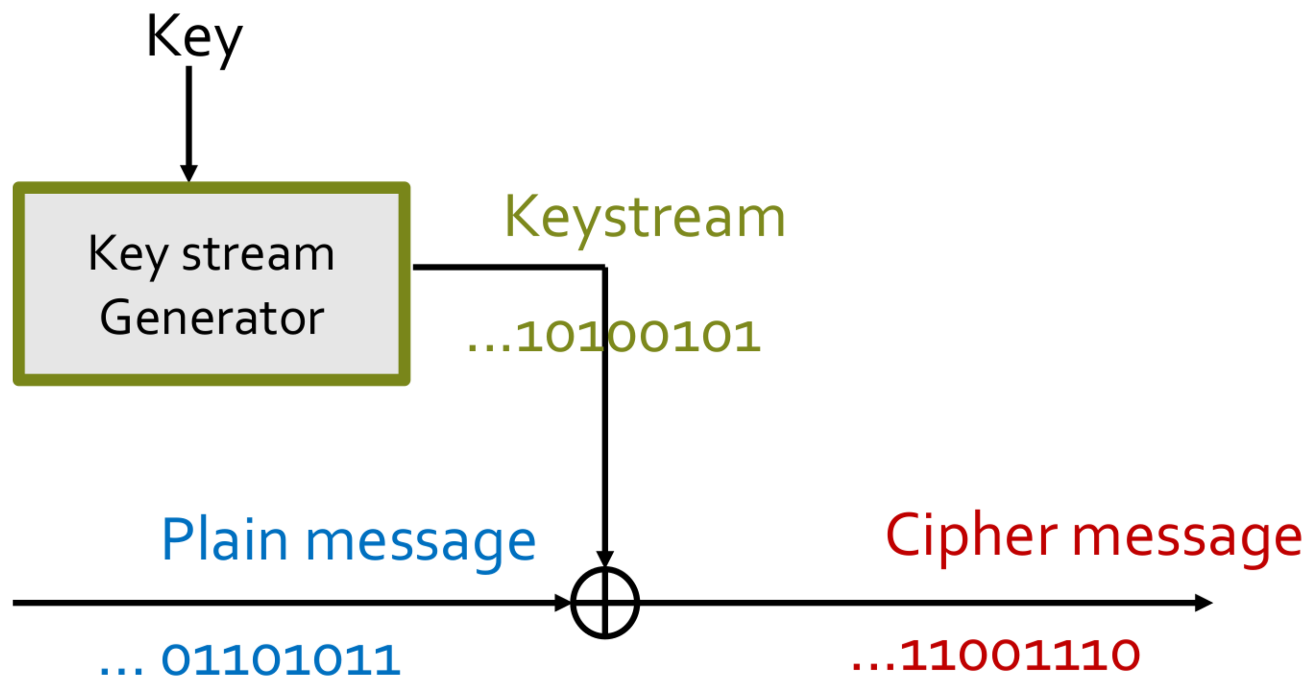
K = 010...

Stream ciphers

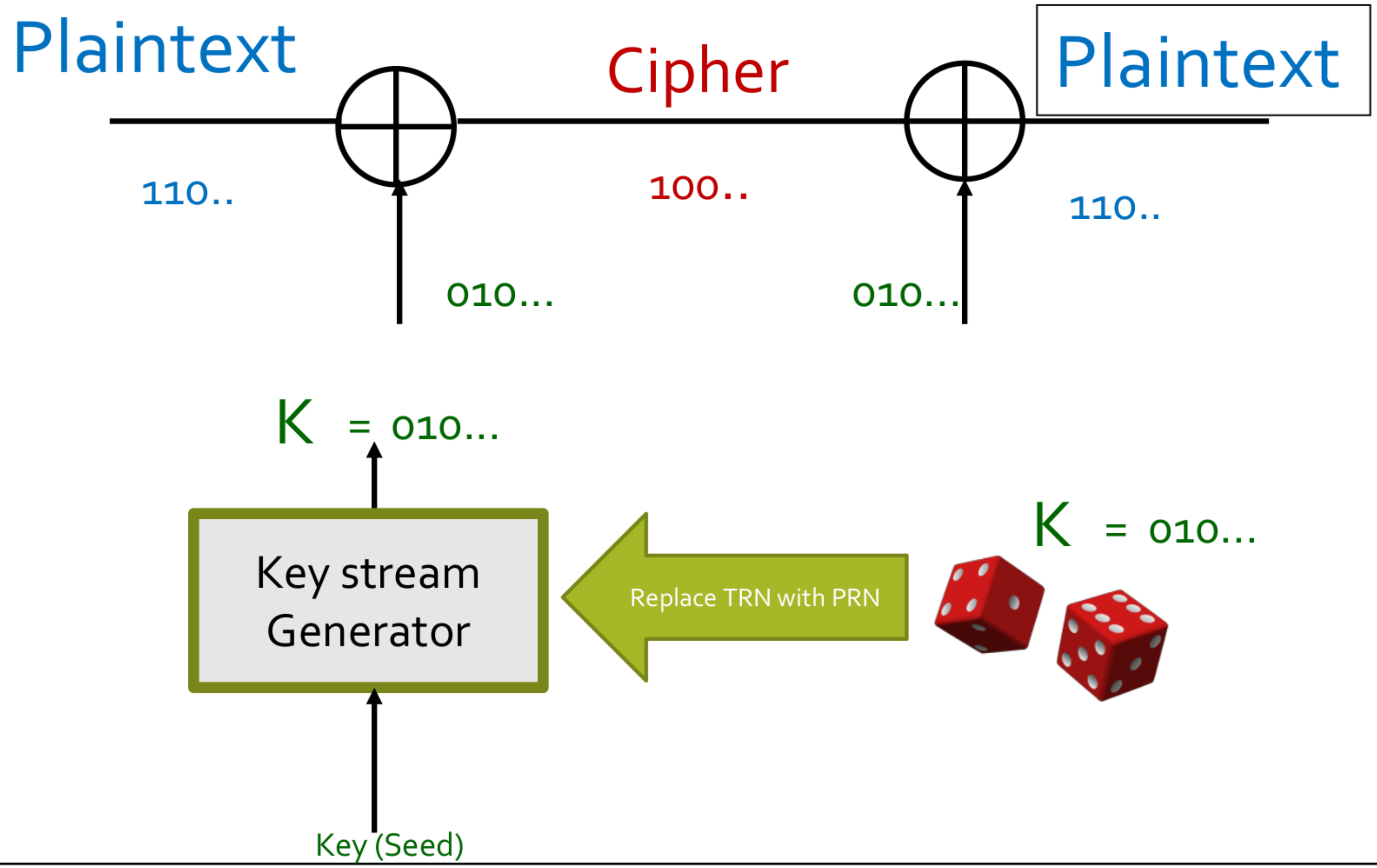
- In One-Time Pad, a key is a random string of length at least the same as the message
- Stream ciphers:
  - Idea: replace "rand" by "pseudo rand"
  - Use a "Pseudo" Random Number Generator
  - G: $\{0,1\}^k \rightarrow \{0,1\}^n$
    - expand a short (e.g., 128-bit) random seed 'k' into a long (e.g., 10^6 bit) string that "looks random"
  - Secret key is the seed



Key Generator



Stream Cipher



LFSR Introduction

LFSR (Linear Feedback Shift Register)

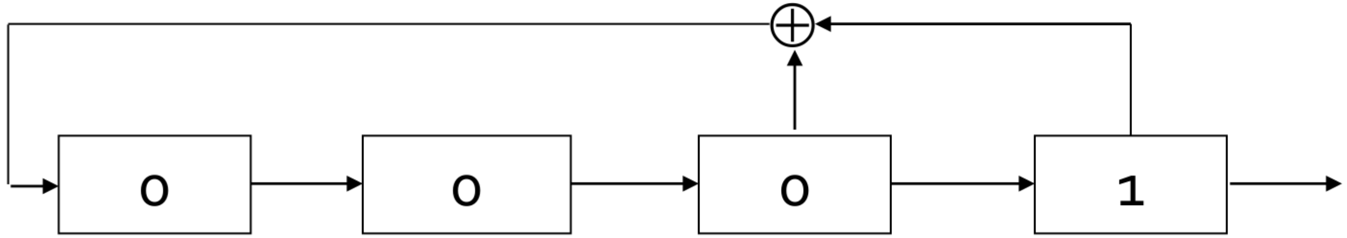
- A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.

- The most commonly used linear function of single bits is exclusive-or (XOR).

- An LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a **very long cycle**.

- The **Berlekamp–Massey** algorithm is an algorithm that will find the shortest linear feedback shift register (LFSR) for a given binary output sequence.

LFSR Example



- The seed is the key

- Starting with 1000, the output stream is:
- 1000 1001 1010 1111 000
- Repeat every $2^4 - 1$ bit

Warming up code

```
def shift_and_print(iv):
    while iv:
        ot = 1 & iv         # save the rightmost bit
        iv >>= 1            # shift
        print(ot)
```

```
def shift_and_yield(iv):
    while iv:
        ot = 1 & iv         # save the rightmost bit
        iv >>= 1            # shift
        yield ot
```

```
shift_and_print(int(hex(0x63), 16))
shift_and_print(int('1001110', 2))

print('Output in one line = ',

      ''.join(str(output) for output in  shift_and_yield(int('1101001', 2))))
```

LFSR

```python
def lfsr_generate(seed, mask, n):
    seed_int = int(seed, 2)
    mask_int = int(mask, 2)
    nbits = len(seed)

    state = seed_int
    while n > 0:
       output = 1 & state # get the most right bit

       _mask, _state, new_bit = mask_int, state, 0
       while _mask:
           new_bit ^= (1 & _mask) * (1 & _state)
           _mask >>= 1
           _state >>= 1

       state = state >> 1 | new_bit << (nbits - 1)

       yield output, state
       n -= 1
```

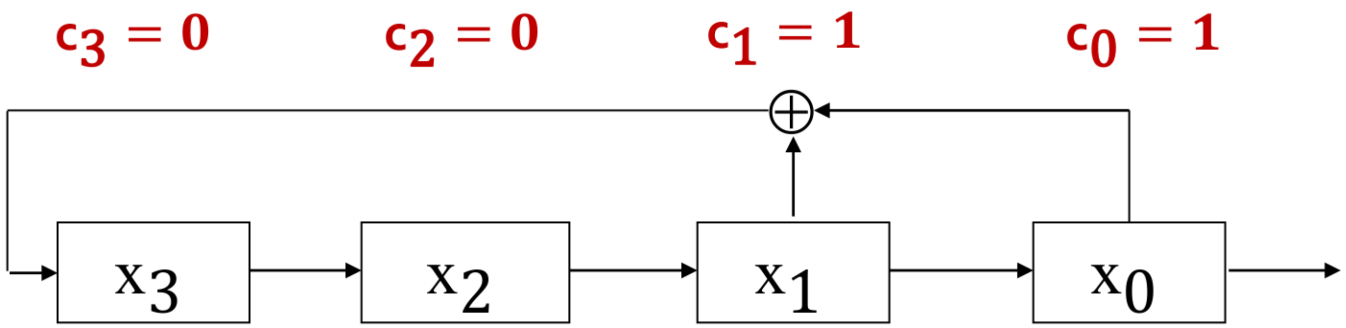
```python
seed = '0001'
mask = '0101'
samples = 20

key = lfsr_generate(seed, mask, samples)
key_str = ''.join(str(x) for x, _ in key)
key_hex = hex(int(key_str, 2))[2:]
```

LFSR Cracking

LFSR: Cryptanalysis I

- A LFSR can be described as:

$$x_{n+i} = \sum_{j=0}^{m-1} c_j x_{i+j} \bmod 2$$



$$x_4 = c_0 x_0 + c_1 x_1 + c_2 x_2 + c_3 x_3 \bmod 2$$
$$x_4 = x_0 + x_1 \bmod 2$$

LFSR: Cryptanalysis II

- Vulnerable to known-plaintext attack

- Knowing **2n output** bits, one can
- construct **n linear equations** with **n unknown variables** $c_0, \ldots, c_{n-1}$
- recover $c_0, \ldots, c_{n-1}$

LFSR: Cryptanalysis III

- Can determine the minimum polynomial $f(x)=x^n+c_{n-1}x^{n-1}+\ldots+c_0$ of a sequence (s_t) from $2n$ successive bits $\mathbf{s_0, s_1, \ldots, s_{2n-1}}$

$$\begin{bmatrix} s_0, & s_1, & \ldots, s_{n-1} \\ s_1, & s_2, & \ldots, s_n \\ \ldots & \ldots & \ldots \\ s_{n-1}, & s_n, & \ldots, s_{2n-2} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \ldots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} s_n \\ s_{n+1} \\ \ldots \\ s_{2n-1} \end{bmatrix}$$

- Matrix has rank **n** if minimum polynomial has rank n
- There exists a very efficient algorithm due to **Berlekamp-Massey** to calculate $c_0, c_1, \ldots, c_{n-1}$ in $O(n^2)$ operations

LFSR: Cryptanalysis IV

```
from berlekampmassey import bm


poly = list(bm('1000101000101000101'))[-1]
print("[*] Recovering secret key  : ",poly[::-1])
```