Université Libre de Bruxelles 2023-2024
Faculté des Sciences

INFO-F-405: Introduction to cryptography

2. Symmetric-key techniques

Keystream generators and stream ciphers

Exercise 1

What happens if the diversifier (or nonce) is not unique, i.e., if the stream cipher is incorrectly used and that two distinct plaintexts are encrypted with the same key and the same diversifier? What consequences does it have in the case of a known plaintext attack?

Exercise 2

Let us consider two linear feedback shift registers (LFSRs) of 4 bits. For each, simulate its execution and determine the cycles. What is the period of the longest possible cycle? Which LFSR has it?

1. $1 + D^2 + D^4$: an iteration is $(b_1, b_2, b_3, b_4) \leftarrow (b_2 + b_4, b_1, b_2, b_3)$.
2. $1 + D^3 + D^4$: an iteration is $(b_1, b_2, b_3, b_4) \leftarrow (b_3 + b_4, b_1, b_2, b_3)$.

Exercise 3

The Mantin-Shamir attack on RC4 shows that the second byte of keystream has a probability of about $\frac{2}{257}$ of taking value 0 and a probability of about $\frac{1}{257}$ of taking each of the other 255 possible values. What is the probability of winning the IND-CPA game when exploiting this property? Give an upper bound on the security strength s of RC4.

Block ciphers

Exercise 4

Complementating a bit string means changing the value of each bit or, equivalently, adding 1 to all bits (modulo 2 of course). Mathematically, if x is a bit string of length n, the complementation of x can be written as :

$$\overline{x} = x \oplus 1^n.$$

Let x and y be two bit strings of length n. Can you simplify the following expressions?

- $\overline{x} + \overline{y}$
- $\overline{x + y}$
- $\overline{(x \| y)}$
- $f(\overline{x})$,

where f is a bit transposition from n bits to n bits. Note that in this case we have $f(1^n) = 1^n$.

Exercise 5

Complementating a bit string means changing the value of each bit, or equivalently, adding 1 to all bits (modulo 2 of course). DES has a non-ideal property: If the input block and the key are complemented, then the output block is complemented as well. In other words, if $\mathrm{DES}_k(x) = y$, then $\mathrm{DES}_{\overline{k}}(\overline{x}) = \overline{y}$, with the overline to indicate complementation.

The goal of this exercise is to describe what happens inside the DES when input block and the key are complemented. (And by "what happens", it is meant "how do the processed values change when computing $\mathrm{DES}_{\overline{k}}(\overline{x})$ instead of $\mathrm{DES}_k(x) = y$".)

1. If the key is complemented, what happens to the subkeys?
2. If the input block is complemented, what happens to the left and right parts at the beginning of the Feistel network?
3. Consider the first round. The right part R enters the f function and goes through the expansion E. What happens to the output of E?

4. Still inside the *f* function, what happens when $E(R)$ is XORed with the subkey K?
5. What happens at to the left part L after being XORed with $f(R)$?
6. What happens in the remaining rounds and to the output?
7. Does this complementation property involve a particular property of the S-boxes?

Exercise 6

During each round of AES, the state (consisting of a 4×4 matrix of elements of GF(256), represented as bytes) undergoes the function MixColumns, which is meant to produce diffusion among the bytes of the state. For each column of the state, this function consists in a simple multiplication between the column and a fixed matrix **M**. If $(s_0, s_1, s_2, s_3)^\top$ is the input of the function (i.e., a column of the state), we get the output $(b_0, b_1, b_2, b_3)^\top$:

$$\begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

What is the output of MixColumns for the column input $s = (\texttt{37}, \texttt{21}, \texttt{A5}, \texttt{C0})^\top$?

Exercise 7

Let **M** be the matrix of MixColumns. So a column of 4 bytes $(s_0, s_1, s_2, s_3)^\top$ is mapped to $(b_0, b_1, b_2, b_3)^\top = \mathbf{M}(s_0, s_1, s_2, s_3)^\top$ after MixColumns.

Because it is a linear operation, we have that $(b_0, b_1, b_2, b_3)^\top = \mathbf{M}(s_0, 0, 0, 0)^\top + \mathbf{M}(0, s_1, 0, 0)^\top + \mathbf{M}(0, 0, s_2, 0)^\top + \mathbf{M}(0, 0, 0, s_3)^\top$.

1. Using the property above, propose a way to implement MixColumns using only look-up's in precomputed tables and XORs. How many tables do you need? What is the size of each table? What do they contain?
2. Can you extend the reasoning so as to implement MixColumns $\circ$ SubBytes in a similar way? *Hint:* SubBytes processes each byte individually.
3. Can you find a variant with just one table? *Hint:* The matrix **M** has a special form.

Exercise 8

The Electronic Codebook (ECB) mode is a flawed encryption mode on top of a block cipher. What fundamental property makes it an inherently insecure mode?

How could an adversary easily win the IND-CPA game with just a few queries, even if he is not allowed to make queries with identical plaintext values? Detail the queries made by the adversary and the choice of (m_0, m_1).

Exercise 9

AES with CTR mode Suppose that a 160 byte message m was encrypted with AES-128 in a counter mode in order to obtain the ciphertext c. Let's suppose that the 10th byte of c was corrupted during a transmission.

a. Would the receiver be able to detect which byte was corrupted?

b. If we decrypt c, how many bytes of m would be affected by the error?

c. Does this number depend on the position where the error occurred?

Exercise 10

AES with CBC mode Suppose that a 160 bytes message m was encrypted with AES-128 in CBC mode in order to obtain the ciphertext c. The block size is 16 bytes. Let's suppose that the 10th byte of c was corrupted during a transmission.

a. Would the receiver be able to detect which byte was corrupted?

b. If we decrypt c, how many bytes of m would be affected by the error?

c. Does this number depend on the position where the error occurred?

Exercise 11

Consider the following encryption mode called DXCTR for "Diversifier-Xored CounTeR". It is a block cipher-based mode very similar to the original CTR, but the diversifier is XORed into the output of the block cipher instead of being part of its input. More precisely, DXCTR works as in Algorithm 1 and is illustrated in Figure 1. However, DXCTR is flawed.

1. What is/are intuitively the security problem(s) of DXCTR?
2. Describe how an adversary can win the IND-CPA game against DXCTR with only practical data and time complexities.
3. Why the original CTR mode does not have this problem?

Algorithm 1 The DXCTR encryption mode on top of block cipher E with block size n bits and key size m bits.

```
**Input:** secret key $K \in \mathbb{Z}_2^m$, plaintext $p \in \mathbb{Z}_2^*$ and diversifier $d \in \mathbb{N}$
**Output:** ciphertext $c \in \mathbb{Z}_2^{|p|}$

Cut $p$ into blocks of $n$ bits $(p_0, p_1, p_2, \ldots, p_l)$, except for the last one that can be shorter
Generate the keystream, for $i = 0$ to $l$
    $k_i = E_K(\text{block}(i)) \oplus \text{block}(d)$
(here block($x$) encodes the integer $x$ into a string in $\mathbb{Z}_2^n$)
Truncate the last keystream block $k_l$ to the size of $p_l$
Compute the ciphertext $c_i = k_i \oplus p_i$, for $i = 0$ to $l$
```

Exercise 12

Consider SHAKE128, a standard sponge function with permutation f, capacity $c = 256$ bits and rate $r = 1344$ bits. Let us assume that it is used in an application that uses it for the authentication of small messages.

In more details, the application computes a 128-bit MAC on a message M under the secret key K as SHAKE128($K\|M$). The secret key is 128-bit long and M is sufficiently short so that $K\|M$ fits in a single bock of r bits, even after padding[1].

1. Write symbolically, i.e., as a mathematical expression, the MAC as a function of the key and of the message for the restricted use case of this application.
2. Knowing that f and its inverse f^{-1} can both be evaluated efficiently, can the secret key be recovered from the value of the MAC? If not, please justify briefly. If so, please describe how.

[1]For simplicity, we assume that the input $K\|M$ is simply padded as $K\|M\|10^*$, i.e., with a single bit 1 followed by the minimum number of bits 0 such that the padded message is r-bit long.

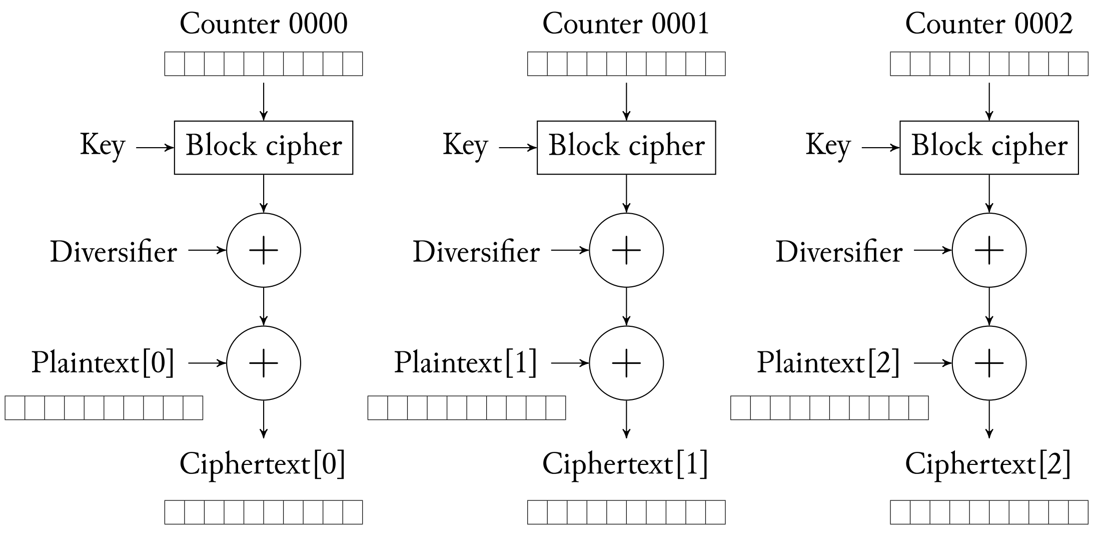

Figure 1: The DXCTR encryption mode.

3. Same question if we extend the MAC length to 1600 bits.
4. Same question if we extend the MAC length to 1600 bits and set $c = 0$, $r = 1600$.

Design of symmetric primitives

Exercise 13

Design of symmetric crypto primitives Below are the specifications of a cryptographic permutation under development, called ABCD. We first give the specifications of the permutation, then ask you some questions about it.

ABCD is a 256-bit permutation. The authors lacked inspirations and gave it its name because the 256-bit input block and running state are represented as 4 words (or rows) of 64 bits each, denoted a, b, c and d. We denote the individual bits in a rows using subscripts, e.g., row a is composed of the bits $a_0, a_1, \ldots, a_{63}$. The four bits (a_i, b_i, c_i, d_i) at the same position in each row is called a column. In other words, the state can be viewed as a grid of 4 rows by 64 columns.

The evaluation of the permutation consists in 13 rounds. The evaluation of permutation goes as follows:

```
for each round $r = 0$ to 12 do
    ShakeColumns
    PreShiftRows
    StirColumns
    PostShiftRows
    AddRoundConstant($r$)
```

We now describe the five step mappings AddRoundConstant, ShakeColumns, PreShiftRows, StirColumns and PostShiftRows, all specified at the bit level, or more formally, in the Galois field $\mathrm{GF}(2) = \{0, 1\}$. In this field, $x + y$ (resp. xy) denotes the modulo-2 addition (resp. multiplication) of $x, y \in \mathrm{GF}(2)$. We use the operator $\oplus$ to express the component-wise addition of vectors of elements in GF(2), such as rows, columns or states.

Definition of AddRoundConstant(r)

```
for each column $i = 0$ to 63 do
    if $i \leq r$ then
        $a_i \leftarrow a_i + 1$
```

Definition of ShakeColumns

```
for each column $i = 0$ to 63 do
    $p \leftarrow a_i + b_i + c_i + d_i$
    $a_i \leftarrow a_i + p$
    $b_i \leftarrow b_i + p$
    $c_i \leftarrow c_i + p$
    $d_i \leftarrow d_i + p$
```

Definition of PreShiftRows

```
$a \leftarrow a$
$b \leftarrow \mathrm{rot}^1(b)$
$c \leftarrow \mathrm{rot}^3(c)$
$d \leftarrow \mathrm{rot}^9(d)$
```

For a row x, $\mathrm{rot}(x)$ denotes the operation that consists in cyclically shifting all the bits by one position, i.e.,

$$(x_0, x_1, x_2, \ldots, x_{63}) \to (x_1, x_2, \ldots, x_{63}, x_0).$$

Definition of StirColumns

```
for each column $i = 0$ to 63 do
    $a_i \leftarrow a_i + b_i c_i$
```

$b_i \leftarrow b_i + c_i d_i$
$c_i \leftarrow c_i + d_i a_i$
$d_i \leftarrow d_i + a_i b_i$

Definition of PostShiftRows

$a \leftarrow a$
$b \leftarrow \text{rot}^1(b)$
$c \leftarrow \text{rot}^5(c)$
$d \leftarrow \text{rot}^{25}(d)$

A: Classify each step mapping as linear, affine or non-linear. For affine or linear mappings, please provide a short justification (one line). For non-linear mappings, please provide an example of input pair that violates the definition of affinity.

B: Point out the step mapping(s) that provide diffusion, with a short justification.

C: For the step mapping ShakeColumns, please explain what happens at the output when:

- one flips 1 bit at the input;
- one flips 2 bits of the same column at the input;
- one flips 2 bits of different columns at the input.

D: Write the specifications of the inverse of ABCD.

E: For each the five step mappings AddRoundConstant, ShakeColumns, PreShiftRows, StirColumns and PostShiftRows, would you characterize it as *bend*, *mix*, *notch* or *shuffle*?

F: Without AddRoundConstant, ABCD would satisfy an (undesired) symmetry property. Which one is it? By symmetry property, we are looking for an invertible operation S on 256-bit strings that commutes with ABCD, i.e., S is such that $\forall x \in \{0,1\}^{256}$, we have $\text{ABCD}(S(x)) = S(\text{ABCD}(x))$.