

Bloková šifra v režimu CBC:

1. Zadání:

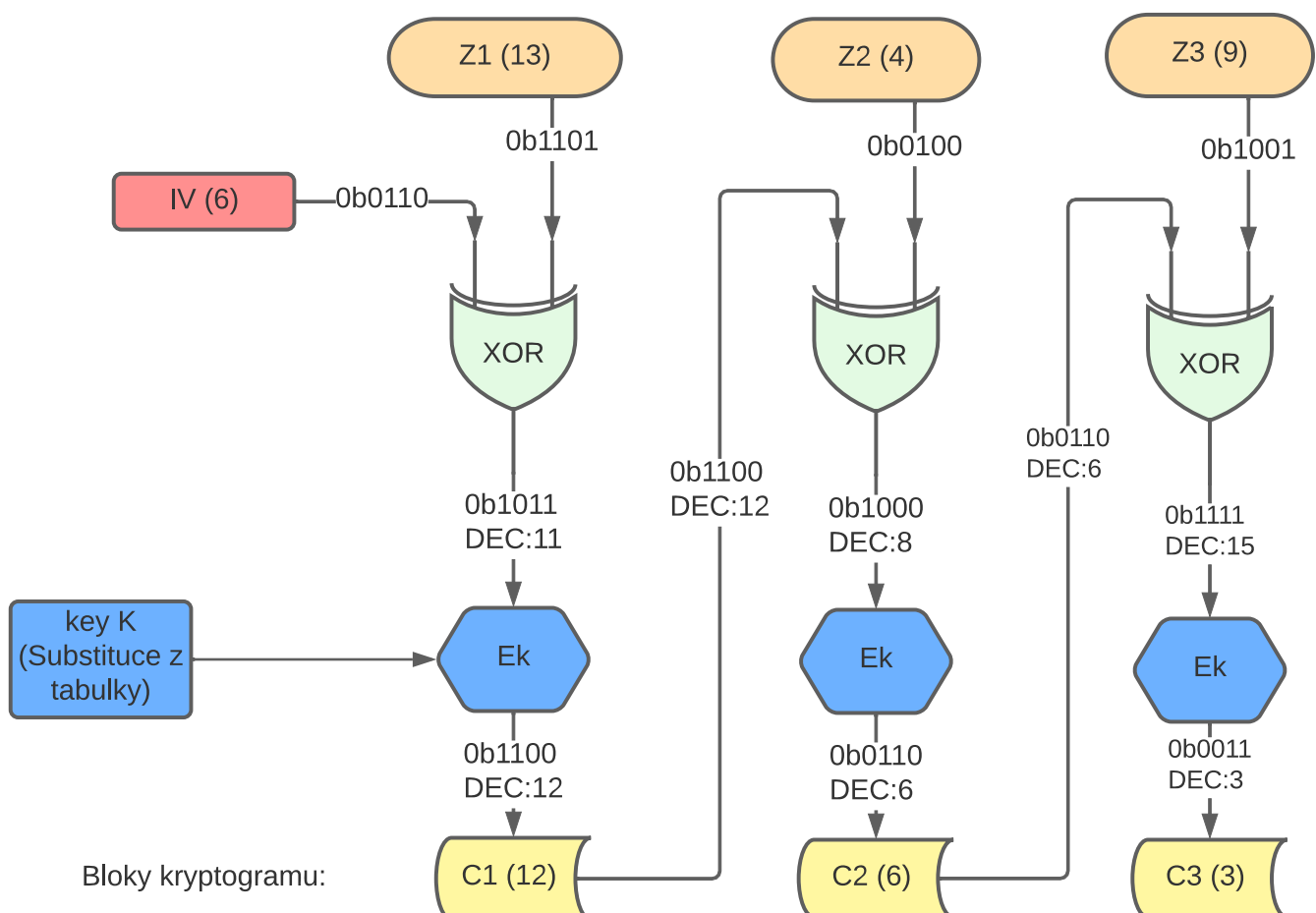
Mějme zprávu $Z = (13, 4, 9)$, kde jednotlivá čísla jsou bloky zprávy. Tuto zprávu zašifrujte v režimu CBC pro inicializační vektor $IV = 6$. Vypočítaný kryptogram pro kontrolu dešifrujte. Šifrování E a dešifrování D je dáno substitucemi podle tabulky 1. K provedení operací XOR si dekadická čísla převedte na čtyřbitová čísla. Pro daný provozní režim nakreslete diagramy podle první přednáškové prezentace (snímek č. 21), přičemž v datových blocích schématu uveďte dekadicky i binárně hodnotu příslušného vstupu, či výstupu.

Table 1: Šifrovací substituce $y = E(x, K)$

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Y	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3

2. Vypracování:

Na následujícím obrázku jsou vyznačeny stavy v decimální i binární podobě pro každý "stupeň" blokové šifry. Tyto hodnoty byly vypočítány pomocí python scriptu přiloženého na konci tohoto pdf souboru. Nicméně pro lepší zobrazení scriptu můžete využít link na můj github repozitář → [PYTHON CBC.py](#)



Výpočet pečeti HMAC

1. Zadání:

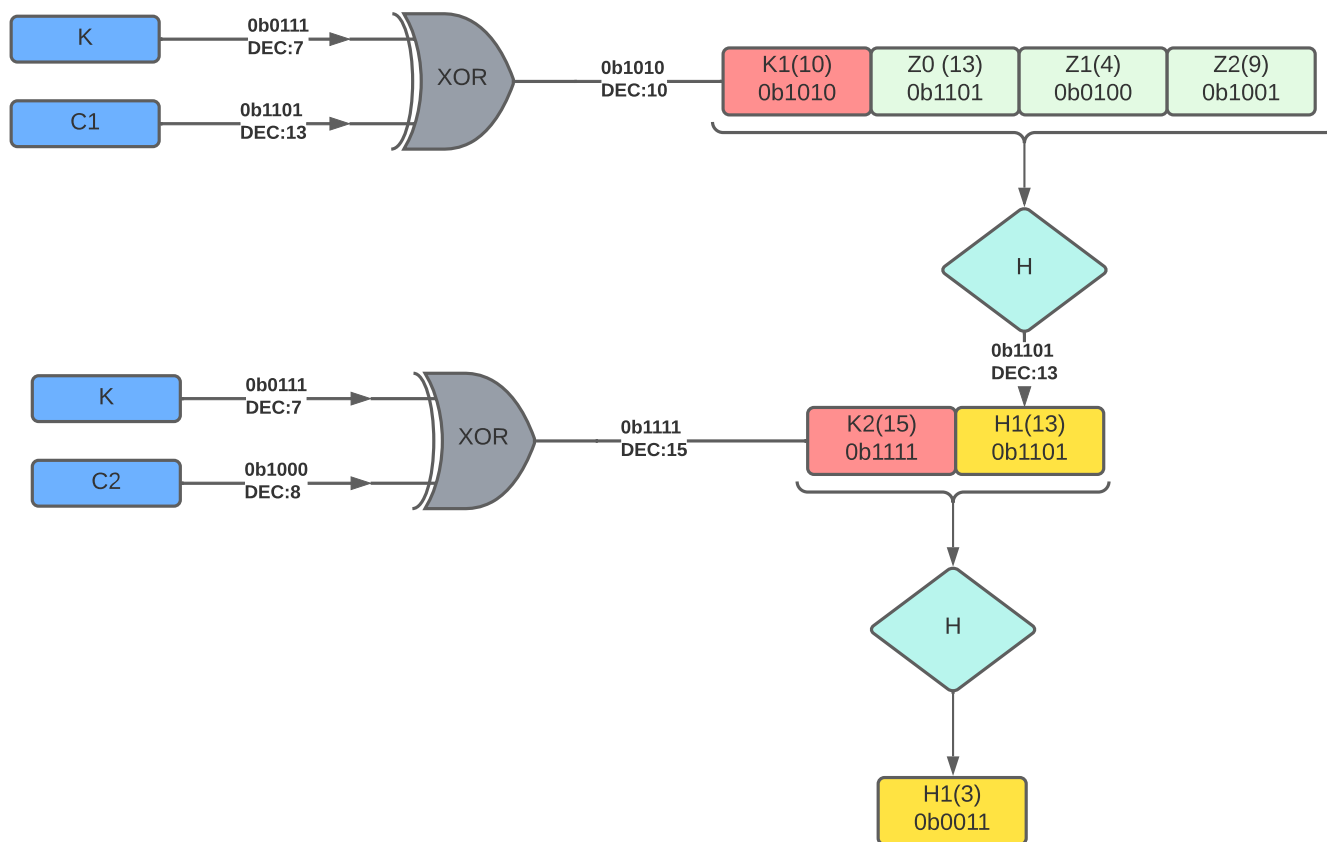
Mějme zprávu $Z = (13, 4, 9)$, kde jednotlivá čísla jsou bloky zprávy. Pro tuto zprávu vypočítejte technikou HMAC pečeť P . Pečetící klíč $K = 7$, konstanta $C_1 = 13$ a $C_2 = 8$. K provedení operací XOR si dekadická čísla převedte na čtyřbitová čísla. Hešovací funkce H je definována následovně:

$$h = \left(\sum_{i=1}^t a^i \cdot v_i \right) \bmod 17$$

kde hešovací konstanta $a = 11$, v_i je i -tý blok hešovaného vstupu a t počet bloků na vstupu. V prvním hešování tedy bude $t = 4$, protože první blok je výsledek xorování klíče a konstanty a další bloky jsou bloky zprávy. Ve druhém hešování bude $t = 2$. Pečeť P je výstup z druhého hešování, tj. $P = h_2$.

2. Vypracování:

Podobně jako v předcházejícím úkolu byla úloha řešena pomocí python [scriptu](#). Na následujícím obrázku jsou znázorněny jednotlivé "stavy" heshovací funkce HMAC v binární i decimální podobě.



RSA podpis

1. Zadání:

Byla Vám doručena zpráva $Z = (13, 4, 9)$, jejíž RSA podpis $DS = 5$. Ověřte, zda je tato zpráva autentická. Znáte veřejný ověřovací klíč udávaného autora $VK = 3$, jeho modulus $n = 33$ a víte, že byla použita hešovací funkce H ze 2. příkladu.

2. Vypracování:

Příložené soubory

1. CBC.py

```
1 def formatInputs(num)->str:
2     """this function formats input number (in this example 4) into the following string:
3     (DEC:4, BIN:0100)"""
4     return f"(DEC:{num}, BIN:{bin(num)[2:].zfill(4)})"
5
6 message_blocks = [13,4,9] #input message blocks
7 table_output = [4,10,9,2,13,8,0,14,6,11,1,12,7,15,5,3] #Encryption substitution table
8 initial_vector = 6 #IV
9
10 ##### ENCRYPTION#####
11 output_cryptograms_list = [] #help variable to store cryptograms
12 input_encryption_list = [] #help variable to store inputs for encryption function (output of
    XOR)
13 i = 0
14 for blocks in message_blocks: #for each message block
15     if i == 0: #first XOR has one input in form of initial vector, therefore this condition..
16         print(f"xor_{i+1}: init_vect{formatInputs(initial_vector)} XOR Z_{i}{formatInputs(
            blocks)}")
17         input_encryption_list.append(blocks^initial_vector)#output of XOR
18
19     else:
20         print(f"xor_{i+1}: init_vect{formatInputs(output_cryptograms_list[-1])} XOR Z_{i}{
            formatInputs(blocks)}")
21         input_encryption_list.append(blocks^output_cryptograms_list[-1])#output of XOR
22
23     print(f"Encription_input_{i+1}: {formatInputs(input_encryption_list[-1])}")
24     #XOR output serves as pointer to the item of encryption table_output list
25     output_cryptograms_list.append(table_output[input_encryption_list[-1]])
26     print(f"Cryptogram_{i+1}: {formatInputs(output_cryptograms_list[-1])}")
27     i+=1
28
29 ##### DECRYPTION #####
30 i = 0
31 for cryptograms in output_cryptograms_list:
32     print(f"Output of Dk_{i+1}: {formatInputs(table_output.index(cryptograms))}")
33     if i == 0:
34         print(f"Decrypted message block_{i+1}: {formatInputs(table_output.index(cryptograms)^
            initial_vector)}")
35     else:
36         print(f"Decrypted message block_{i+1}: {formatInputs(table_output.index(cryptograms)^
            output_cryptograms_list[i-1])}")
37     i+=1
```

Output:

```
xor_1: init_vect(DEC:6, BIN:0110) XOR Z_0(DEC:13, BIN:1101)
Encription_input_1: (DEC:11, BIN:1011)
Cryptogram_1: (DEC:12, BIN:1100)
xor_2: init_vect(DEC:12, BIN:1100) XOR Z_1(DEC:4, BIN:0100)
Encription_input_2: (DEC:8, BIN:1000)
Cryptogram_2: (DEC:6, BIN:0110)
xor_3: init_vect(DEC:6, BIN:0110) XOR Z_2(DEC:9, BIN:1001)
Encription_input_3: (DEC:15, BIN:1111)
Cryptogram_3: (DEC:3, BIN:0011)
```

2. HMAC.py

```
1 message_blocks = [13,4,9] #input message blocks
2 seal_key = 7
3 Constatnt_C1 = 13
4 Constatnt_C2 = 8
5 block_size = 4 #constants are 4 bit long.. 13 = 0b1101
6
7 def hashFunction(input):
8     a = 11#constant
9     t = int(len(input)/4) #count of blocks with length 4bits
10    sum_result = 0 #variable to sum
11    for i in range(t): #for each block
12        for x in range(4):#for each bit in block of length 4
```

```

13         sum_result += pow(a,i)*input[x+4*i]
14     result = sum_result%17 #modulo 17
15     return result
16
17 K_1 = Constatnt_C1^seal_key #XOR key with constant to get new keys
18 K_2 = Constatnt_C2^seal_key #XOR key with constant to get new keys
19
20 #K_1_list in for of [MSB, , ,LSB]
21 K_1_list_of_bits = [int(i) for i in bin(K_1)[2:].zfill(4)]
22 #EXTEND message blocks to the K1 key
23 for blocks in message_blocks:
24     message_list_of_bits = [int(i) for i in bin(blocks)[2:].zfill(4)]
25     K_1_list_of_bits.extend(message_list_of_bits)
26
27 h_1 = hashFunction(K_1_list_of_bits) #HASH K1|Z(0)|Z(1)|Z(2)
28
29 #create blocks in form of K2|h1
30 h_1_list = [int(i) for i in bin(h_1)[2:].zfill(4)]
31 K_2_list_of_bits = [int(i) for i in bin(K_2)[2:].zfill(4)]
32 K_2_list_of_bits.extend(h_1_list)
33
34
35 h2 = hashFunction(K_2_list_of_bits) #HASH K2|h1
36
37 print(f"K1: (DEC:{K_1}, BIN: {bin(K_1)[2:].zfill(4)})")
38 print(f"K2: (DEC:{K_2}, BIN: {bin(K_2)[2:].zfill(4)})")
39 print(f"h1: (DEC:{h_1}, BIN: {bin(h_1)[2:].zfill(4)})")
40 print(f"h2: (DEC:{h2}, BIN: {bin(h2)[2:].zfill(4)})")

```

Output:

```

K1: (DEC:10, BIN: 1010)
K2: (DEC:15, BIN: 1111)
h1: (DEC:13, BIN: 1101)
h2: (DEC:3, BIN: 0011)

```