Øving K16

Oppgave 1

a) Gitt følgende rekursive følge/LFSR ... Hva blir perioden med nøklene

1. K = 1000
2. K = 0011
3. K = 1111

In [233]:
```
N = 20

Z = [[1, 0, 0, 0],
     [0, 0, 1, 1],
     [1, 1, 1, 1]]

for z in Z:
    for i in range(N):
        z.append((z[i] + z[i+1] + z[i+2] + z[i+3]) % 2)

print(Z[0])
print(Z[1])
print(Z[2])
```

```
[1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1]
```

1. K = 1000, periode 5
2. K = 0011, periode 5
3. K = 1111, periode 5

b) Gjør det samme for følgende LFSR: ...

In [234]:
```
N = 20

Z = [[1, 0, 0, 0],
     [0, 0, 1, 1],
     [1, 1, 1, 1]]

for z in Z:
    for i in range(N):
        z.append((z[i] + z[i+3]) % 2)

print(Z[0])
print(Z[1])
print(Z[2])
```

```
[1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0]
[0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0]
[1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1]
```

1. K = 1000, periode 15
2. K = 0011, periode 15
3. K = 1111, periode 15

Oppgave 2

Bruk Autokey-chifret, med P = C = Z_{29}.

- La k = 17 og krypter teksten goddag

In [235]:
```
ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ'

def toNumbers(string):
    numbers = []

    for char in string:
        num = ALPHABET.index(char.upper())
        numbers.append(num)

    return numbers
```

In [236]:
```
def encrypt(p, K):
    p = toNumbers(p)

    c = [(p[0] + K) % 29]

    for i in range(1, len(p)):
        c.append((p[i] + p[i-1]) % 29)

    return c
```

In [237]:
```
K = 17
p = 'goddag'

encrypt(p, K)
```

Out[237]: [23, 20, 17, 6, 3, 6]

- Gitt k = 5, dekrypter 23 08 23 12 21 02 04 03 17 13 19.

In [238]:
```
ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ'

def toString(numbers):
    string = ''

    for num in numbers:
        string += ALPHABET[num]

    return string
```

In [239]:
```
def decrypt(c, K):
    p = [(c[0] - K) % 29]

    for i in range(1, len(c)):
        p.append((c[i] - p[i-1]) % 29)

    return toString(p)
```

In [240]:
```
K = 5
c = [23, 8, 23, 12, 21, 2, 4, 3, 17, 13, 19]

decrypt(c, K)
```

Out[240]: 'STEINSPRANG'

Oppgave 3

Vi definerer en HMAC som følger:

- K = 1001
- ipad = 0011
- opad = 0101
- h er midtkvadratmetoden, dvs. vi regner x^2 (mod 2^8) og henter de midterste fire sifrene (skriv ut tallet med 8 bits, med ledende nuller hvis det trengs

a) Finn HMAC til meldingen 0110

In [241]:
```
def h(x):
    return ((x**2 % (2**8)) & 0b00111100) >> 2
```

In [242]:
```
opad = 0b0101
ipad = 0b0011

def HMAC(x, K):
    return bin(h((K ^ opad) << 4 | h((K ^ ipad) << 4 | x )))[2:].zfill(4)
```

In [243]:
```
K = 0b1001
x = 0b0110

HMAC(x, K)
```

Out[243]: '0100'

b) Du mottar meldingen 0111 og HMAC 0100. Er det grunn til å tro at meldingen ikke er autentisk?

Har:

$x = 0100$ og $x' = 0111$

Disse gir samme HMAC, altså er:

$h(x) = h(x')$, men $x \neq x'$

Dette gir en kollisjon. Hashfunksjoner skal unngå kollisjoner, og dersom hashfunksjonen er implementert korrekt skal ikke dette forekomme. Det er defor god grunn til å tro at meldingen ikke er autentisk.

Oppgave 4

Bruk Cæsarchifferet y = x + 3 (mod 24) og finn CBC-MACen til de to meldingene

$x = 1101111110100001$
$x' = 0010110000011111$

In [244]:
```
def encrypt(x, K):
    return (x + K) % 2**4
```

In [245]:
```
def CBC_MAC(x, K):
    y = [0b0000]

    for i in range(1, len(x)):
        y.append(encrypt(y[i-1] ^ x[i], K))

    return bin(y[-1])[2:].zfill(4)
```

In [246]:
```
K = 3
messages = [[0b1101, 0b1111, 0b1010, 0b0001],
            [0b0010, 0b1100, 0b0001, 0b1111]]

for x in messages:
    print(CBC_MAC(x, K))
```

```
1101
0001
```

Oppgave 5

En (dårlig) variant av AES er nesten som AES: Den har samme blokkstørrelse på 128 bits, samme nøkkellengde på 128 bits, og utfører følgende tre trinn som i AES

- ADDROUNDKEY (original nøkkel)
- SUBBYTES
- SHIFTROWS

NB! Første rundenøkkel er samme som original nøkkel. Ingen XOR på slutten. Bruk nøkkelen

67 71 35 c4 ff da e5 ff 1c 54 e1 fd 7f 2e 88 b7

a) Krypter meldingen

24 59 66 0c 99 da 9b 00 d6 55 fd 20 e9 ff 46 95

In [247]:
```
sbox = [[0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7
, 0xAB, 0x76],
        [0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4
, 0x72, 0xC0],
        [0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8
, 0x31, 0x15],
        [0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27
, 0xB2, 0x75],
        [0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3
, 0x2F, 0x84],
        [0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C
, 0x58, 0xCF],
        [0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C
, 0x9F, 0xA8],
        [0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF
, 0xF3, 0xD2],
        [0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D
, 0x19, 0x73],
        [0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E
, 0x0B, 0xDB],
        [0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95
, 0xE4, 0x79],
        [0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A
, 0xAE, 0x08],
        [0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD
, 0x8B, 0x8A],
        [0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1
, 0x1D, 0x9E],
        [0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55
, 0x28, 0xDF],
        [0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54
, 0xBB, 0x16]]
```

In [248]:
```
def add_round_key(c, K):
    return c ^ K
```

In [249]:
```
def sub_bytes(c):
    for i in range(len(c)):
        for j in range(len(c)):
            c[i][j] = sbox[math.floor(c[i][j] / 16)][c[i][j] % 16]

    return c
```

In [250]:
```
def shift_rows(c):
    copy = c.copy()

    for i in range(len(c)):
        for j in range(len(c)):
            c[i][j] = copy[i][(j + i) % len(c)]

    return c
```

In [251]:
```
import math
import numpy as np

def encrypt(p, K, rounds):
    N = int(math.sqrt(len(p)))

    K = np.array(K).reshape(N, N).T
    p = np.array(p).reshape(N, N).T

    c = p.copy()

    for _ in range(rounds):
        c = add_round_key(c, K)
        c = sub_bytes(c)
        c = shift_rows(c)

    return c.T.reshape(1, 16)[0].tolist()
```

In [252]:
```
K = [0x67, 0x71, 0x35, 0xc4, 0xff, 0xda, 0xe5, 0xff, 0x1c, 0x54, 0xe1, 0xfd, 0x7f, 0x2e, 0x
88, 0xb7]
p = [0x24, 0x59, 0x66, 0x0c, 0x99, 0xda, 0x9b, 0x00, 0xd6, 0x55, 0xfd, 0x20, 0xe9, 0xff, 0x
46, 0x95]

c = encrypt(p, K, 10)
print(' '.join(hex(n) for n in c))
```

```
0xf8 0xbc 0x59 0xd6 0x46 0xf6 0x92 0x28 0x49 0x74 0x9a 0x8a 0xdd 0x0 0xd6 0x1e
```

b) Dekrypter meldingen

26 FA 83 E7 2D CD 5D B8 C4 DC EB 12 70 CF D6 1E

In [253]:
```
sbox_inv = [[0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81,
0xF3, 0xD7, 0xFB],
            [0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4,
0xDE, 0xE9, 0xCB],
            [0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42,
0xFA, 0xC3, 0x4E],
            [0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D,
0x8B, 0xD1, 0x25],
            [0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D,
0x65, 0xB6, 0x92],
            [0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7,
0x8D, 0x9D, 0x84],
            [0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8,
0xB3, 0x45, 0x06],
            [0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01,
0x13, 0x8A, 0x6B],
            [0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0,
0xB4, 0xE6, 0x73],
            [0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C,
0x75, 0xDF, 0x6E],
            [0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA,
0x18, 0xBE, 0x1B],
            [0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78,
0xCD, 0x5A, 0xF4],
            [0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27,
0x80, 0xEC, 0x5F],
            [0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93,
0xC9, 0x9C, 0xEF],
            [0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83,
0x53, 0x99, 0x61],
            [0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55,
0x21, 0x0C, 0x7D]]
```

In [254]:
```
def inv_add_round_key(p, K):
    return p ^ K
```

In [255]:
```
def inv_sub_bytes(p):
    for i in range(len(p)):
        for j in range(len(p)):
            p[i][j] = sbox_inv[math.floor(p[i][j] / 16)][p[i][j] % 16]

    return p
```

In [256]:
```
def inv_shift_rows(p):
    copy = p.copy()

    for i in range(len(p)):
        for j in range(len(p)):
            p[i][j] = copy[i][(j - i) % len(p)]

    return p
```

In [257]:
```
import math
import numpy as np

def decrypt(c, K, rounds):
    N = int(math.sqrt(len(c)))

    K = np.array(K).reshape(N, N).T
    c = np.array(c).reshape(N, N).T

    p = c.copy()

    for _ in range(rounds):
        p = inv_shift_rows(p)
        p = inv_sub_bytes(p)
        p = inv_add_round_key(p, K)

    return p.T.reshape(1, 16)[0].tolist()
```

In [281]:
```
K = [0x67, 0x71, 0x35, 0xc4, 0xff, 0xda, 0xe5, 0xff, 0x1c, 0x54, 0xe1, 0xfd, 0x7f, 0x2e, 0x
88, 0xb7]
c = [0x26, 0xFA, 0x83, 0xE7, 0x2D, 0xCD, 0x5D, 0xB8, 0xC4, 0xDC, 0xEB, 0x12, 0x70, 0xCF, 0x
D6, 0x1E]

p = decrypt(c, K, 10)
print(' '.join(hex(n) for n in p))
```

```
0x7 0x24 0xe4 0x81 0xd 0x21 0x52 0x0 0x68 0xc2 0x2d 0x60 0x1 0x51 0x46 0x7a
```

Oppgave 6

Gitt følgende 128-bits AES-nøkkel, i heksadesimal notasjon,

2B7E151628AED2A6ABF7158809CF4F3C

Kjør KEYEXPANSION, begrenset som følger:

- Bare de første 6 ordene, w[0], ..., w[5]
- Ikke bruk SUBWORD.

Algoritmen jobber med 32-bits ord, og hver heksadesimale tegn er 4 bits.

In [259]:
```
def toBinary(w):
    return w[0] << 24 | w[1] << 16 | w[2] << 8 | w[3]
```

In [260]:
```
def rotWord(w):
    return (w << 8) & (2**32 - 1) | w >> 24
```

In [276]:
```
K = [0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x
4F, 0x3C]
rcon = 0x01 << 24

w = [0] * 6
for i in range(4):
    w[i] = toBinary(K[4*i: 4*i+4])

w[4] = w[0] ^ rotWord(w[3]) ^ rcon
w[5] = w[4] ^ w[1]

for i in range(len(w)):
    print(f'w[{i}]: 0b{bin(w[i])[2:].zfill(32)} 0x{hex(w[i])[2:].zfill(8)}')
```

```
w[0]: 0b00101011011111110000101010010110 0x2b7e1516
w[1]: 0b00101000101011101101001010100110 0x28aed2a6
w[2]: 0b10101011111110111000101011000100 0xabf71588
w[3]: 0b00001001110011111010011110011100 0x09cf4f3c
w[4]: 0b11100101001100010010100100011111 0xe531291f
w[5]: 0b11001101100111111111101110111001 0xcd9ffbb9
```