111550113 謝詠晴

Cryptography Engineering Quiz. 4

Problem 1

a) Yes, it is a primitive polynomial.

```python
a = [[0] * 8 for _ in range(263)]
for i in range(0,263):
    ans = ""
    degree = [0]*8
    if 0 <= i <= 7 :
        degree[i] = 1
        a[i] = degree

    else:
        for j in range(0,8):
            a[i][j]=(a[i-4][j]+a[i-5][j]+a[i-6][j]+a[i-8][j])%2

    for j in range(0,8):
        if(a[i][j]==1):
            ans+=(f"x^{j} ")

    print(f"a^{i} = {ans}")
```

$x^8 + x^4 + x^3 + x^2 + 1$ is irreducible. Then check if it is primitive by verifying if it generates a maximal length sequence when used in LFSR. And it turned out that it will have a repetitive sequence after iterating 256 times.

```
a^0 = x^0
a^1 = x^1
a^2 = x^2
a^3 = x^3
a^4 = x^4
a^5 = x^5
a^6 = x^6
a^7 = x^7
a^8 = x^0 x^2 x^3 x^4
a^9 = x^1 x^3 x^4 x^5
a^10 = x^2 x^4 x^5 x^6
a^11 = x^3 x^5 x^6 x^7
a^12 = x^0 x^2 x^3 x^6 x^7
a^13 = x^0 x^1 x^2 x^7
a^14 = x^0 x^1 x^4
a^15 = x^1 x^2 x^5
a^16 = x^2 x^3 x^6
a^17 = x^3 x^4 x^7
a^18 = x^0 x^2 x^3 x^5
a^19 = x^1 x^3 x^4 x^6
a^20 = x^2 x^4 x^5 x^7
```

```
a^245 = x^0 x^3 x^5 x^6 x^7
a^246 = x^0 x^1 x^2 x^3 x^6 x^7
a^247 = x^0 x^1 x^7
a^248 = x^0 x^1 x^3 x^4
a^249 = x^1 x^2 x^4 x^5
a^250 = x^2 x^3 x^5 x^6
a^251 = x^3 x^4 x^6 x^7
a^252 = x^0 x^2 x^3 x^5 x^7
a^253 = x^0 x^1 x^2 x^6
a^254 = x^1 x^2 x^3 x^7
a^255 = x^0
a^256 = x^1
a^257 = x^2
a^258 = x^3
a^259 = x^4
a^260 = x^5
a^261 = x^6
a^262 = x^7
a^263 = x^0 x^2 x^3 x^4
a^264 = x^1 x^3 x^4 x^5
a^265 = x^2 x^4 x^5 x^6
```

```
a^500 = x^0 x^3 x^5 x^6 x^7
a^501 = x^0 x^1 x^2 x^3 x^6 x^7
a^502 = x^0 x^1 x^7
a^503 = x^0 x^1 x^3 x^4
a^504 = x^1 x^2 x^4 x^5
a^505 = x^2 x^3 x^5 x^6
a^506 = x^3 x^4 x^6 x^7
a^507 = x^0 x^2 x^3 x^5 x^7
a^508 = x^0 x^1 x^2 x^6
a^509 = x^1 x^2 x^3 x^7
a^510 = x^0
a^511 = x^1
a^512 = x^2
a^513 = x^3
a^514 = x^4
a^515 = x^5
a^516 = x^6
a^517 = x^7
a^518 = x^0 x^2 x^3 x^4
a^519 = x^1 x^3 x^4 x^5
a^520 = x^2 x^4 x^5 x^6
```

b) Since the maximum cycle length of an LFSR using a primitive polynomial of degree n is $2^n - 1$. Therefore, $x^8 + x^4 + x^3 + x^2 + 1$ has a highest degree 8 and then its maximum cycle length is $2^8 - 1 = 255$.

c) No, while the reverse is true, not all irreducible polynomials are primitive polynomials. Being irreducible means, it cannot be factored into polynomials of lower degree, however, it doesn't guarantee that the polynomial will generate a maximal length sequence when used in an LFSR.

For example, $x^8 + x^4 + x^3 + x + 1$ is irreducible but not primitive since it doesn't generate a maximal length sequence which is 255.

```
a^0 = x^0
a^1 = x^1
a^2 = x^2
a^3 = x^3
a^4 = x^4
a^5 = x^5
a^6 = x^6
a^7 = x^7
a^8 = x^0 x^1 x^3 x^4
a^9 = x^1 x^2 x^4 x^5
a^10 = x^2 x^3 x^5 x^6
a^11 = x^3 x^4 x^6 x^7
a^12 = x^0 x^1 x^3 x^5 x^7
a^13 = x^0 x^2 x^3 x^6
a^14 = x^1 x^3 x^4 x^7
a^15 = x^0 x^1 x^2 x^3 x^5
a^16 = x^1 x^2 x^3 x^4 x^6
a^17 = x^2 x^3 x^4 x^5 x^7
a^18 = x^0 x^1 x^5 x^6
```

```
a^51 = x^0
a^52 = x^1
a^53 = x^2
a^54 = x^3
a^55 = x^4
a^56 = x^5
a^57 = x^6
a^58 = x^7
a^59 = x^0 x^1 x^3 x^4
a^60 = x^1 x^2 x^4 x^5
a^61 = x^2 x^3 x^5 x^6
a^62 = x^3 x^4 x^6 x^7
a^63 = x^0 x^1 x^3 x^5 x^7
a^64 = x^0 x^2 x^3 x^6
a^65 = x^1 x^3 x^4 x^7
a^66 = x^0 x^1 x^2 x^3 x^5
a^67 = x^1 x^2 x^3 x^4 x^6
a^68 = x^2 x^3 x^4 x^5 x^7
a^69 = x^0 x^1 x^5 x^6
```

Problem 2

a) run problem2.py

[Encryption]

- First, using ‘ord()’ and ‘bin()’ function to convert each character of plaintext into its binary ASCII representation.
- Store the binary representation in list m[].
- Each bit of m[] is XORed with the corresponding bit of key[]., and the result is appended to the ‘ciphertext’ string.
- ‘ct’ string and ‘cipherword[]’ list are used for decryption process.
- Use LFSR to generate a pseudo-random key stream. If the ‘tmp’ bit stored from the first position of the key is 1, XOR key and polynomial.

```python
key = [0,0,0,0,0,0,0,1]
polynomial = [1,0,0,0,1,1,1,0,1]
plaintext = "ATNYCUWEARESTRIVINGTOBEAGREATUNIVERSITYTHATTRANSCENDSDISCIPLINARYDIVIDESTOSOLVETHEINCREASINGLYCO\
MPLEXPROBLEMSTHATTHEWORLDFACESWEWILLCONTINUETOBEGUIDEDBYTHEIDEATHATWECANACHIEVESOMETHINGMUCHGREATERTOGETHERTHA\
NWECANINDIVIDUALLYAFTERALLTHATWASTHEIDEATHATLEDTOTHECREATIONOFOURUNIVERSITYINTHEFIRSTPLACE"
ciphertext = ""
decrypted_text = ""
cipherchar = []
for i in range(len(plaintext)):
    m = [0]*8
    ct =""
    character = bin(ord(plaintext[i])) # convert to binary representation
    for j in range(2,9):
        m[j-1] = int(character[j])

    for j in range(8):
        ciphertext += str(key[j] ^ m[j])
        ct += str(key[j] ^ m[j])
    cipherchar.append(ct)

    #LFSR to generate key stream
    tmp = key[0]
    for j in range(7):
        key[j] = key[j+1]

    key[7] = 0
    if (tmp == 1):
        for i in range(8):
            key[i] = key [i] ^ polynomial[i+1]
```

[Decryption]

- Decryption works similarly to encryption. The difference is that plaintext is the XOR result of key and ciphertext.
- ‘cipherchar[]’ is a list that stores the binary representation of each character in the plaintext.

```
#decryption
key = [0,0,0,0,0,0,0,1]
for ch in cipherchar:
    pt = ""
    for j in range(8):
        pt += str(key[j] ^ int(ch[j]))
    tmp = key[0]
    for j in range(7):
        key[j] = key[j+1]
    #LFSR
    key[7] = 0
    if (tmp == 1):
        for i in range(8):
            key[i] = key [i] ^ polynomial[i+1]
    pt = chr(int(pt,2))
    decrypted_text += pt

print("decrypted text:")
print(decrypted_text)
```

Result:

```
ciphertext:
0100000001010110010010100101000101010011011101010001011111000101010111000110100000110001101110111001100
1110101010101101001110000000001011101011001101010000011101111101100110111101011111000100011001000010100
0101000011010011010100110001100101001011101000100111001011011000100001110011001111011011000001111011001
1010110000100100010100101011110000100100011101111001000000011010001011100000000010111001001010010110100
1110010001110110110000011001111100110001111011110011001110011001111011110000001000011001111111110011000
0011011111010001010001001010011010000011010110001101111101111001100011000010111011010010000000101001001
1010100011110110100100101101011010001010011010111101010010100010000100111101000011111010101101101010001
0010001111001000000101111110111100110010101011011001010011111111000011111100001101001001011100100000010110
1100011101001111010101100111000100100101100000111110100100101111100011111101010101001011011101100011100
1101011111010001010010101110111110111111100110000101011011000011011010010011000000011000110001001110100
1001011110011000100001000011111011001000101001010011111101000001101101000001110100000000111110100000010111
0100110100000010001111011110001100011010011011011010001011111100000000101111111001100011101110110100111
1100001110111010000001001000011110110001001111010111010010000111010110011011100100000110010000011110110 0
0011000100010110110010100110010000011110111001000001111011000011000100010110110010100111000100000010111
1100100000111101100001100010001011011001010011100010000000101111010110000000011010110010111000110000000 1
1000010000111110010100100001001011010010101000111110100100010101101001001010111100110000111011111000100
0101001110000100011000111111000011000010111100101111011010011010010101110100100110111100000011111011010
0001001110001100101000001010011011000100001110011010101010110111100111011101110001011100110100011101110 11
1011101010111111100111001100101001011000010000100110010000011101111110010011100110111111101010001001101
1110010110101101001100010001000001001110111101001000100111100000101010101010010100100000101001011010000
1001100101000000001110101000101010001110101001110101010011110001011110010000100011001110100000110011101 0
1100110010000000110011110001001001111011100110000000110110110101101001001111010000100100110001111000001 0
01011000011011010100011101010001110111001000011101010000011011010101011101100010111 1

decrypted text:
ATNYCUWEARESTRIVINGTOBEAGREATUNIVERSITYTHATTRANSCENDSDISCIPLINARYDIVIDESTOSOLVETHEINCREASINGLYCOMPLEXPR
OBLEMSTHATTHEWORLDFACESWEWILLCONTINUETOBEGUIDEDBYTHEIDEATHATWECANACHIEVESOMETHINGMUCHGREATERTOGETHERTHA
NWECANINDIVIDUALLYAFTERALLTHATWASTHEIDEATHATLEDTOTHECREATIONOFOURUNIVERSITYINTHEFIRSTPLACE
```

If I convert the ciphertext to ASCII code:

```
Ciphertext (ASCII):
@VJQSuÅ\h1»ÕZpÖjû7¯ÈQCMLe.ËbÏlÍa"á#¼ÑpÉKNGlóó9ð!óú(Ðk1Ú@Rj=¤µ¢õ(ô> ¨äï2 ÿÃIrÇOVq%é/ÕKv9¯¢ß0 Ò`1Ò^bû"ýÐtÐZhïÓmàóÚ|; ö'®ë4Ò~0%¦ú>
´¬¬îÞx,Øb-âëÖ\`0ÊBZT}"´æñáü0¾½¦Òoí       ÆPSbÕo;¸¹£»º¿ÊXBdù9¿¨ËZb éÁUJAKBeÔTu:§ÈFtÖd
ñ'¹ÛZOBLx%ÔuÈuÕv/
```

b) Yes, it is possible to find out. Given a 8-stage LFSR, we have:

$$\begin{cases} a_n = (a_{n+1}C_7 + a_{n+2}C_6 + a_{n+3}C_5 + \cdots + a_{n+7}C_1 + a_{n+8}C_0)\ mod2 \\ a_{n+1} = (a_{n+2}C_7 + a_{n+3}C_6 + a_{n+4}C_5 + \cdots + a_{n+8}C_1 + a_{n+9}C_0)\ mod2 \\ a_{n+2} = (a_{n+3}C_7 + a_{n+4}C_6 + a_{n+5}C_5 + \cdots + a_{n+9}C_1 + a_{n+10}C_0)\ mod2 \\ a_{n+3} = (a_{n+4}C_7 + a_{n+5}C_6 + a_{n+6}C_5 + \cdots + a_{n+10}C_1 + a_{n+11}C_0)\ mod2 \\ a_{n+4} = (a_{n+5}C_7 + a_{n+6}C_6 + a_{n+7}C_5 + \cdots + a_{n+11}C_1 + a_{n+12}C_0)\ mod2 \\ a_{n+5} = (a_{n+6}C_7 + a_{n+7}C_6 + a_{n+8}C_5 + \cdots + a_{n+12}C_1 + a_{n+13}C_0)\ mod2 \\ a_{n+6} = (a_{n+7}C_7 + a_{n+8}C_6 + a_{n+9}C_5 + \cdots + a_{n+13}C_1 + a_{n+14}C_0)\ mod2 \\ a_{n+7} = (a_{n+8}C_7 + a_{n+9}C_6 + a_{n+10}C_5 + \cdots + a_{n+14}C_1 + a_{n+15}C_0)\ mod2 \end{cases}$$

Knowing a_0 to a_7, we can compute coefficients C_0 to C_7. And in general, if we know 16 output bits, solving a 8-stage LFSR is possible.

c) $C_0 = 1,\ C_1 = 0, C_2 = 0, C_3 = 0, C_4 = 1, C_5 = 1, C_6 = 1, C_7 = 0$

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

- The code of this part is in problem2.py.
- Let n =0, a [ ] is a_0 to a_{15}, and c [ ] is the coefficients we want.
- Using brute force method, there are 255 combinations of c [].
- 'ac' is the sum of $a_{j+1+k}C_k$ and see if after mod2, it will equal a_j or not

```
# # bonus: 2-c
# using the MSB of a0 to a15
a = [0]*16
for i in range(16):
    a[i] = int(cipherchar[i][0])
c = [0, 0, 0, 0, 0, 0, 0, 0]


for i in range(255):
    flag = 1
    for j in range(8):
        ac = 0 # the sum of c*a
        for k in range(8):
            ac += c[k] * a[j+1+k]
        if(a[j] != ac%2):
            flag = 0
            break
    if flag == 1:
        print(c)
        break

    n = 0
    while(c[n]):
        c[n] = 0
        n += 1
    c[n] = 1
```

Problem 3

a) run problem3.py

Import 'random' and 'permutations' from 'itertools'

```
import random
from itertools import permutations
```

The implementation of the two shuffle algorithms:

```
# Naïve shuffle
def naive(cards):
    for i in range(len(cards)):
        n = random.randint(0, len(cards)-1)
        cards[i], cards[n] = cards[n], cards[i]
    return cards

# Fisher-Yates shuffle
def fisher_yates(cards):
    for i in range(len(cards)-1, 0, -1):
        n = random.randint(0, i)
        cards[i], cards[n] = cards[n], cards[i]
    return cards
```

Use 'shuffle_simulation()' function to simulate these two shuffle algorithms.

```
# simulate 10^6 times
def shuffle_simulation(shuffle_func, times=1000000):
    cards = [1,2,3,4]
    perm = permutations(cards)
    perm_count = {}
    for p in perm:
        perm_count[p] = 0

    for _ in range(times):
        shuffled_cards = shuffle_func(cards.copy())
        perm_count[tuple(shuffled_cards)] += 1

    return perm_count
```

Results:

```
Naive algorithm:
(1, 2, 3, 4): 39099
(1, 2, 4, 3): 39114
(1, 3, 2, 4): 38896
(1, 3, 4, 2): 54808
(1, 4, 2, 3): 42882
(1, 4, 3, 2): 35337
(2, 1, 3, 4): 39095
(2, 1, 4, 3): 58846
(2, 3, 1, 4): 54827
(2, 3, 4, 1): 54464
(2, 4, 1, 3): 42591
(2, 4, 3, 1): 43067
(3, 1, 2, 4): 42797
(3, 1, 4, 2): 42545
(3, 2, 1, 4): 35542
(3, 2, 4, 1): 42823
(3, 4, 1, 2): 42820
(3, 4, 2, 1): 39327
(4, 1, 2, 3): 31395
(4, 1, 3, 2): 35150
(4, 2, 1, 3): 35279
(4, 2, 3, 1): 31165
(4, 3, 1, 2): 38911
(4, 3, 2, 1): 39220
```

```
Fisher-Yates shuffle:
(1, 2, 3, 4): 41640
(1, 2, 4, 3): 41474
(1, 3, 2, 4): 41785
(1, 3, 4, 2): 41807
(1, 4, 2, 3): 41909
(1, 4, 3, 2): 41812
(2, 1, 3, 4): 41878
(2, 1, 4, 3): 41321
(2, 3, 1, 4): 41791
(2, 3, 4, 1): 41719
(2, 4, 1, 3): 41754
(2, 4, 3, 1): 41509
(3, 1, 2, 4): 41628
(3, 1, 4, 2): 41497
(3, 2, 1, 4): 41320
(3, 2, 4, 1): 41730
(3, 4, 1, 2): 41957
(3, 4, 2, 1): 41486
(4, 1, 2, 3): 41609
(4, 1, 3, 2): 41778
(4, 2, 1, 3): 41623
(4, 2, 3, 1): 41622
(4, 3, 1, 2): 41565
(4, 3, 2, 1): 41786
```

b) Fisher-Yates shuffle is a better choice since the distribution of each permutation's count is more uniform than Naïve algorithm.
c) The main drawback of the Naïve algorithm is that the randomized results are not uniform distribution. Because it only swaps based on the randomly generated one, it is prone to causing certain segments to be switched more.