Quiz. 4

Problem 1

LFSR is simply an arrangement of n stages in a row with the last stage, plus any other stages, modulo-two added together and returned to the first stage. An algebraic expression can symbolize this arrangement of stages and tap points called the characteristic polynomial. One kind of characteristic polynomial called primitive polynomials over GF (2), the field with two elements 0, 1, can be used for pseudorandom bit generation to let linear- feedback shift register (LFSR) with maximum cycle length.

a) Is $x^8 + x^4 + x^3 + x^2 + 1$ a primitive polynomial?

Answer) Yes. The polynomial is irreducible over GF (2), and it generates a maximal length LFSR sequence of length 255.

b) What is the maximum cycle length generated by $x^8 + x^4 + x^3 + x^2 + 1$?

Answer) The maximum cycle length generated by $x^8 + x^4 + x^3 + x^2 + 1$ is $2^8 - 1 = 255$.

c) Are all irreducible polynomials primitive polynomials?

Answer) Not all irreducible polynomials are primitive polynomials. For example, $x^4+x^3+x^2+x+1$ is irreducible over GF (2), but it is not primitive because it does not generate a maximal length LFSR sequence of length 15.

Problem 2

Given the plaintext:

ATNYCUWEARESTRIVINGTOBEAGREATUNIVERSITYTHATTRAN

SCENDSDISCIPLINARYDIVIDESTOSOLVETHEINCREASINGLYCO

MPLEXPROBLEMSTHATTHEWORLDFACESWEWILLCONTINUET

OBEGUIDEDBYTHEIDEATHATWECANACHIEVESOMETHINGMU

CHGREATERTOGETHERTHANWECANINDIVIDUALLYAFTERALLT

HATWASTHEIDEATHATLEDTOTHECREATIONOFOURUNIVERSI

TYINTHEFIRSTPLACE

a) Please use $x^8 + x^4 + x^3 + x^2 + 1$ as a characteristic polynomial to write a Python. program to encrypt the following plaintext message with the initial key 00000001, then decrypt it to see if your encryption is correct.

Cipher text:

01000000010101100100101001010001010100110111010100010111110001
01010111000110100000110001101110111001100111010101010110100111100
00000001011101011001101010000011101111101100110111101011111100010
00110010000101000101000011010011010100110001100101001011101000
10011100101101100010000111001100111101101100000111101100110101 10
00010010001010010101111000010010001110111100100000000110100010 1
11000000000101110010010100101101001110010001110110110000011001 1
11100110001111011110011001110011001111011110000000100001100111111 11
10011000001101111101000101000100101001101000001101011000110111 11
01111001100011000010111011010010000000010100100110101000111101 10
10010010110101101000101001101011110101001010001000010011110100 0
01111101010110110101000100100011110010000010111111011110011001010
10110110010100111111110000111111000011010010010111001000010110110
00111010011110101011001110001001001011000001111101001001011111 00
01111110101010100101101110110001110011010111110100010100101011 1011
1110111111100110000101011011000011011010010011000000001100011 0001
00111010010010111100110001000010000111110110010001010010100111 11
10100000110110100000111010000000011111010000010110100110100000
01000111101111000110001101001101101101000101111110000000010111111
10011000111011101101001111100001110111010000010010000111101100 01
00111101011101001000011101011001101001000001111010010011111100 01
10000101111011010011010011111111101000111110101000001011010010 10
11001010110010010110111011100001111110111100111100000101100100 00
01111011000011000100010110110010100111000100000010111101011000 0
00001101011001011100011000000011000010000111110010100100001 00
10110100101010001111101001000101011010010010101111001100001110 11
11100010001010011100001000110001111110000110000101111001011110 11
01001101001010111010010011011110000001111101101000010011100011 00
10100000101001101100010000111001101010101101111001110111011100 01
01110011010001110111011101110101011111110011100110010100101100 0010
00010011001000001110111111001001110011011111110101000100110111 100
10110101101001100010001000001001110111101001000100111100000101 0
10101010010100100000101001011010000100110010100000000111010100
01010100011101010011101010100111100010111100100001000110011101 00
00011001110101100110010000001100111100010010011110111001100000 0
01101101101011010010011110100001001001100011110000010010110000 1
10110101000111010100011101110010000111010100000110110101010111 01
1000101111

Decrypted text:

ATNYCUWEARESTRIVINGTOBEAGREATUNIVERSITYTHATTRANSCEN
DSDISCIPLINARYDIVIDESTOSOLVETHEINCREASINGLYCOMPLEXPR
OBLEMSTHATTHEWORLDFACESWEWILLCONTINUETOBEGUIDEDBY
THEIDEATHATWECANACHIEVESOMETHINGMUCHGREATERTOGET
HERTHANWECANINDIVIDUALLYAFTERALLTHATWASTHEIDEATHAT
LEDTOTHECREATIONOFOURUNIVERSITYINTHEFIRSTPLACE

b) Due to the property of ASCII coding the ASCII A to Z, the MSB of each byte will be zero (left most bit); therefore, every 8 bits will reveal 1 bit of random number (i.e., keystream); if it is possible to find out the characteristic polynomial of a system by solving of linear equations?

Yes. Given a 8-stage LFSR, we know:

$$
\begin{cases}
a_n = (a_{n+1}C_7 + a_{n+2}C_6 + a_{n+3}C_5 + \ldots + a_{n+6}C_2 + a_{n+7}C_1 + a_{n+8}C_0) \mod 2 \\
a_{n+1} = (a_{n+2}C_7 + a_{n+3}C_6 + a_{n+4}C_5 + \ldots + a_{n+7}C_2 + a_{n+8}C_1 + a_{n+9}C_0) \mod 2 \\
a_{n+2} = (a_{n+3}C_7 + a_{n+4}C_6 + a_{n+5}C_5 + \ldots + a_{n+8}C_2 + a_{n+9}C_1 + a_{n+10}C_0) \mod 2 \\
a_{n+3} = (a_{n+4}C_7 + a_{n+5}C_6 + a_{n+6}C_5 + \ldots + a_{n+9}C_2 + a_{n+10}C_1 + a_{n+11}C_0) \mod 2 \\
a_{n+4} = (a_{n+5}C_7 + a_{n+6}C_6 + a_{n+7}C_5 + \ldots + a_{n+10}C_2 + a_{n+11}C_1 + a_{n+12}C_0) \mod 2 \\
a_{n+5} = (a_{n+6}C_7 + a_{n+7}C_6 + a_{n+8}C_5 + \ldots + a_{n+11}C_2 + a_{n+12}C_1 + a_{n+13}C_0) \mod 2 \\
a_{n+6} = (a_{n+7}C_7 + a_{n+8}C_6 + a_{n+9}C_5 + \ldots + a_{n+12}C_2 + a_{n+13}C_1 + a_{n+14}C_0) \mod 2 \\
a_{n+7} = (a_{n+8}C_7 + a_{n+9}C_6 + a_{n+10}C_5 + \ldots + a_{n+13}C_2 + a_{n+14}C_1 + a_{n+15}C_0) \mod 2
\end{cases}
$$

Knowing $a_0,\ a_1,\ \ldots,\ a_{15}$, can compute $C_0,\ C_1,\ \ldots,\ C_7$, thus can solve a 8-stage LFSR.

c) Extra credit: Write a linear equations program solving program to find the characteristic polynomial for this encryption with initial 00000001.

Answer) C_0=1, C_1=0, C_2=0, C_3=0, C_4=1, C_5=1, C_6=1, C_7=0

```
f16keyoutput = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]
solution = [0, 0, 0, 0, 0, 0, 0, 0]

for i in range(255):
    flg = 1
    for j in range(8):
        c = 0
        for k in range(8):
            c += solution[k] * f16keyoutput[j+1+k]
        if(f16keyoutput[j] != c%2):
            flg = 0
            break
    if flg:
        print(solution)
        break

    cnt = 0
    while(solution[cnt]):
        solution[cnt] = 0
        cnt += 1
    solution[cnt] = 1
```

Output:

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

Problem 3

RC4's vulnerability mainly arises from its inadequate randomization of inputs, particularly the initialization vector (IV) and key integration, due to its reliance on the initial setup by its Key Scheduling Algorithm (KSA). The cipher operates through two phases: KSA, which shuffles a 256-byte state vector based on the key to ensure dependency and randomization, and the Pseudo-Random Generation Algorithm (PRGA), where it further manipulates this state to produce a seemingly random output stream.

To help you understand the importance of randomization algorithms, here we provide the pseudocode for two slightly different shuffle algorithms.

Naïve algorithm:

```
For i from 0 to length(cards)-1
    Generate a random number n between 0 and length(cards)-1
    Swap the elements at indices i and n
EndFor
```

Fisher–Yates shuffle (Knuth shuffle):

```
For i from length(cards)-1 down to 1
    Generate a random number n between 0 and i
    Swap the elements at indices i and n
EndFor
```

a) Please write a Python program to simulate two algorithms with a set of 4 cards, shuffling each **a million times**. Collect the count of all combinations and output, for example:

```
$ python problem3.py
Naive algorithm:
[1 2 3 4]: 41633
[1 2 4 3]: 41234
... and so on Fisher-
Yates shuffle:
[1 2 3 4]: 41234
[1 2 4 3]: 41555
... and so on
```

```
Basic Shuffle Method(Naive algorithm):
(1, 2, 3, 4): 38876
(1, 2, 4, 3): 39046
(1, 3, 2, 4): 39125
(1, 3, 4, 2): 54849
(1, 4, 2, 3): 42638
(1, 4, 3, 2): 35332
(2, 1, 3, 4): 39126
(2, 1, 4, 3): 58176
(2, 3, 1, 4): 54776
```

```
(2, 3, 4, 1): 54987
(2, 4, 1, 3): 42885
(2, 4, 3, 1): 42951
(3, 1, 2, 4): 42909
(3, 1, 4, 2): 43117
(3, 2, 1, 4): 35159
(3, 2, 4, 1): 43165
(3, 4, 1, 2): 43331
(3, 4, 2, 1): 38958
(4, 1, 2, 3): 31095
(4, 1, 3, 2): 35153
(4, 2, 1, 3): 35179
(4, 2, 3, 1): 31032
(4, 3, 1, 2): 39241
(4, 3, 2, 1): 38894
Mean: 41666.666666666664, Standard Deviation: 7213.089835307905

Efficient Shuffle (Fisher-Yates):
(1, 2, 3, 4): 41638
(1, 2, 4, 3): 41956
(1, 3, 2, 4): 41492
(1, 3, 4, 2): 41787
(1, 4, 2, 3): 42064
(1, 4, 3, 2): 41871
(2, 1, 3, 4): 41617
(2, 1, 4, 3): 41978
(2, 3, 1, 4): 41396
(2, 3, 4, 1): 41390
(2, 4, 1, 3): 41866
(2, 4, 3, 1): 41172
(3, 1, 2, 4): 41726
(3, 1, 4, 2): 41769
(3, 2, 1, 4): 41762
(3, 2, 4, 1): 41688
(3, 4, 1, 2): 41469
(3, 4, 2, 1): 41570
(4, 1, 2, 3): 41231
(4, 1, 3, 2): 41580
(4, 2, 1, 3): 41916
(4, 2, 3, 1): 41976
(4, 3, 1, 2): 41473
(4, 3, 2, 1): 41613
Mean: 41666.666666666664, Standard Deviation: 235.43305252708726
```

b) Based on your analysis, which one is better, why?

Answer) Fisher-Yates shuffle is better because it demonstrates a more uniform distribution, as indicated by a lower standard deviation compared to the Naïve algorithm.

c) What are the drawbacks of the other one?

Answer) The Naïve method is biased as it swaps elements from the whole range of indices, resulting in unequal probabilities for each element at each position.
In contrast, the Fisher-Yates shuffle assures that each position can only swap with positions preceding it (or itself), resulting in a more uniform and unbiased distribution.

PS: I use NumPy as package in the program