# Problem 1

Quiz. 4

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 polyno- mial. 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.

1. 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.

1. 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.

1. Are all irreducible polynomials primitive polynomials?

Answer) Not all irreducible polynomials are primitive polynomials. For example, x4+x3+x2+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

1. 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: 0100000001010110010010100101000101010011011101010001011111000101010111000110100000110001101110111001100111010101010110100111000000000101110101100110101000001110111110110011011110101111100010001100100001010001010000110100110101001100011001010010111010001001110010110110001000011100110011110110110000011110110011010110000100100010100101011110000100100011101111001000000011010001011100000000010111001001010010110100111001000111011011000001100111110011000111101111001100111001100111101111000000100001100111111111001100000110111110100010100010010100110100000110101100011011111011110011000110000101110110100100000001010010011010100011110110100100101101011010001010011010111101010010100010000100111101000011111010101101101010001001000111100100000101111110111100110010101011011001010011111111000011111100001101001001011100100001011011000111010011110101011001110001001001011000001111101001001011111000111111010101010010110111011000111001101011111010001010010101110111110111111100110000101011011000011011010010011000000011000110001001110100100101111001100010000100001111101100100010100101001111110100000110110100000111010000000111110100000101101001101000000100011110111100011000110100110110110100010111111000000001011111110011000111011101101001111100001110111010000010010000111101100010011110101110100100001110101100110100100000111101001001111110001100001011110110100110100111111111101000111110101000001011010010101100101011001001011011101110000111111101111001111000001011001000001111011000011000100010110110010100111000100000010111101011000000001101011001011100011000000011000010000111110010100100001001011010010101000111110100100010101101001001010111100110000111011111000100010100111000010001100011111100001100001011110010111101101001101001010111010010011011110000001111101101000010011100011001010000010100110110001000011100110101010110111100111011101110001011100110100011101110111011101010111111100111001100101001011000010000100110010000011101111110010011100110111111101010001001101111001011010110100110001000100000100111011110100100010011110000010101010101001010010000010100101101000010011001010000000111010100010101000111010100111010101001111000101111001000010001100111010000011001110101100110010000001100111100010010011110111001100000001101101101011010010011110100001001001100011110000010010110000110110101000111010100011101110010000111010100000110110101010111011000101111

Decrypted text: ATNYCUWEARESTRIVINGTOBEAGREATUNIVERSITYTHATTRANSCENDSDISCIPLINARYDIVIDESTOSOLVETHEINCREASINGLYCOMPLEXPROBLEMSTHATTHEWORLDFACESWEWILLCONTINUETOBEGUIDEDBYTHEIDEATHATWECANACHIEVESOMETHINGMUCHGREATERTOGETHERTHANWECANINDIVIDUALLYAFTERALLTHATWASTHEIDEATHATLEDTOTHECREATIONOFOURUNIVERSITYINTHEFIRSTPLACE

1. 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?

A screenshot of a computer

Description automatically generated

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

Answer) C0=1, C1=0, C2=0, C3=o, C4=1, C5=1, C6=1, C7=0

A screenshot of a computer

Description automatically generated

Output:

A screenshot of a computer

Description automatically generated

# Problem 3

RC4’s vulnerability mainly arises from its inadequate randomization of inputs, particu- larly 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

1. 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

1. 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.

1. 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