Quiz. 4

(Deadline March 28, 2024)

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?

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

c) Are all irreducible polynomials primitive polynomials?

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.

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?

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

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
```

Hint: you can use ***random*** *library.*

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

c) What are the drawbacks of the other one, and what causes these drawbacks?

What to turn in:

a) The file you need to upload is structured as follows:

- <student_id>.zip
  - problem2.py
  - problem3.py
  - <student_id>.pdf

b) The <student_id>.pdf file should contain instructions on how to run your code, the screenshots of the result of program, and the solution to the provided problems.