**Homework #2**

Due date: 13/11/2020

Notes:

* If you used Python codes for questions, compress them along with an answer sheet (a docx or pdf file).
* Name your winzip file as “CS41507\_hw02\_yourname.zip”
* Attached are “myntl.py”, “lfsr.py”, and “bonus\_helper.py” that you can use for the homework questions.

1. (**20 pts**) Use the Python code “**Q1\_student.py**” given in the assignment package to communicate with the server. The server will send you a prime number **p** and the number **t**, which is the order of a subgroup of . Please read the comments in the Python code.

Consider the group .

* 1. (**10 pts**) Find a generator in . Send your answer to the server using Q1\_student.py

I solved this problem using 2 nested for loops. The upper for loop, iterates through the all possible elements from 1 to p, and similarly the inner loop iterates through all elements from 1 to p. But inside the inner loop, we take the power of the elements that are came from the upper part from 1 to p. Based on the results, if that element generates the whole set of the Zp, then that element would be the generator of the group Zp. Since, we only need one generator for this question, I terminated the iteration process when I found 5 as the generator of Zp among several other options. Then based on my answer, I received ‘Congrats’ message from the server. The codes could be reached from Q1\_student.py file along with comments.

* 1. (**10 pts**) Consider a subgroup of , whose order is **t**. Find a generator of this subgroup and send the generator to the server.

Among the possible elements of the group Zp, I searched for the elements in which the order (total number of unique elements that they generate) is equal to t. Since the group formed by that particular element would also be a subgroup of Zp and the number of elements in that group is equal to t, that element would be the generator of the subgroup of Zp. I returned the first option “20” as a result among several other options and received ‘Congrats!’ message from the server. The codes could be reached from Q1\_student.py file.

(**20 pts**) Use the Python code “**Q2\_student.py**” given in the assignment package to communicate with the server. The server will send 4 numbers: p, q, e, and c where n=p×q

Compute m = cd mod n (where d = e-1 mod φ(n)). Decode m into Unicode string and send the text you found to the server using Q2\_student.py

I have calculated m, based on the following equalities:

n = p \* q, phi\_n = (p-1) \* (q-1), d =modinv(e,phi\_n), m = pow(c,d,n).

Based on the equations, I found the m value as: m = 561395015603525619642719231840467292642446911062474466276400 . Then, I decoded message and found the message as “Your secret number is 840”. Afterwards, I received the “Congrats!” message from the server. The codes could be reached from the Q2\_student.py file.

1. (**30 pts**) Solve the following equations of the form ax ≡ b mod n and find all solutions for x if a solution exists. In case there is no solution, your answer must be “NO SOLUTION”, and explain why there is no solution.
   1. n = 97289040915427312142046186233204893375

a = 61459853434867593821323745103091100940

b = 22119567361435062372463814709890918083

Since gcd(a,n) = 5, and 5 does not divide b, there is NO SOLUTION. The codes could be reached from the submitted hw.py file.

* 1. n = 97289040915427312142046186233204893375

a = 87467942514366097632147785951765210855

b = 3291682454206668645932879948693825640

Since gcd(a,n) = 5 and 5 divides b, there are exactly 5 solutions. The solutions are as follows. The codes could be reached from the submitted hw.py file.

Solutions = [11368713749418004372789821825215865218, 30826521932503466801199059071856843893, 50284330115588929229608296318497822568, 69742138298674391658017533565138801243, 89199946481759854086426770811779779918]

* 1. n = 97289040915427312142046186233204893375

a = 74945727802091171826938590498744274413

b = 54949907590247169540755431623509626593

Since gcd(a,n) = 1 and 1 divides b, there is exactly one solution. The solution is as follows. The codes could be reached from the submitted hw.py file.

Solution = [75940790615126559855606958795348491611].

1. (**15 pts**) Consider the following binary connections polynomials for LFSR:

P1(x) = x7 + x3 + x2 + 1

P2(x) = x7 + x + 1

Do they generate maximum period sequences?

(**Hint:** You can use the functions in **lfsr.py**)

After setting the coefficients of the binary polynomial, I have used the findPeriod(stream) function provided in lfsr.py file. Then, if the result is 2^7 -1, the polynomial generates the maximum period sequence. If not, it does not generate. Depending on the initial state, the period of the P1(x) is 62 or less. (sometimes 31 or even less). On the other hand, the period of the P2(x) is always 127 which is equal to 2^7 -1. Therefore, P2(x) generates the maximum period sequence while P1(x) does not. The codes could be reached from the hw.py file.

1. (**15 pts**) Consider a random number generator that generates the following sequences. Are they unpredictable? Explain your answer. (**Hint:** You can use the functions in lfsr.py)

x1 = [1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1]

x2 = [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]

x3 = [1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0]

I have used the Barlekamp Massey algorithm provided in the lfsr.py file. The first element that this method returns is the linear complexity of the stream input . The linear complexity should be at least N/2 in order a sequence to be unpredictable where N is the length of the bit streams. In all three cases, the linear complexity of the sequences are 37, which is less than 75 (150 / 2), where 150 is the length of the sequence. Therefore, the sequences x1, x2, and x3 are both predictable. The codes could be reached from the hw.py file.

**Bonus Question**

1. (**20 pts**) Consider the following ciphertext bit stream encrypted using a stream cipher. And you strongly suspect that an LFRS is used to generate the key stream:

ctext = [1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0]

Also, encrypted in the ciphertext you also know that there is a message to you from the instructor; and therefore the message starts with “Dear Student”. Find the plaintext. For this you need to find the connection polynomial of the LFSR first. Note that the ASCII encoding (seven bits for each ASCII character) is used.

**(Hint:** You can use the ASCII2bin(msg) and bin2ASCII(msg) functions (in **bonus\_helper.py**) to make conversion between ASCII and binary)

# Firstly, I converted the starting part of the plain text ‘Dear Student’ from ascii to bin and performed xor operation with the corresponding starting part of the cipher text. This would give us the initial part of the key stream. Then, this key stream was used as the parameter of the BM(keystream) method and we obtained both state length and the connection polynom. Afterwards, when we take first state length of the key stream and reversed the array, we obtained the initial state. Then based on the initial state and connection polynom parameters, I called the LFSR(S,C) method length of the cipher text many times. The output of the LFSR method at each iteration is xor’ed with the corresponding cipher text and thus the bin version of the plain text is found. Finally, I converted the bin version of the plain text to ascii, and found the decrypted message as follows.

Decrypted Message:

Dear Student,

You have worked hard, I know taht; but it paid off:)

You have just earned 20 bonus points. Congrats!

Best,

Erkay Savas

The codes could be reached from hw.py file along with comments.