Homework #2

Due date: 10/11/2023
Notes:

- Compress the Python codes for your questions 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**", "**hw2_helper.py**", and "**client.py**" that you can use for the homework questions.
- Do not submit .ipynb files, **only .py** scripts will be considered. You can work on Colab but please, submit a Python file in the end.
- Use "client.py" to communicate with the server. The main server is located at the campus therefore you need to **connect to the campus network using VPN (**ONLY IF YOU ARE OUTSIDE THE CAMPUS**).** Then, you can run your code as usual. See the IT website for VPN connections.
- **The server's address** is provided as "http://harpoon1.sabanciuniv.edu:9999/". This address is provided by default in **client.py.** Check the code.
- Brute-forcing the solutions (trying to query all possible solutions) in a server-related question would **not** be considered a valid answer and will result in a score of 0 for the respective question.
- Once you get the values in Q1 & Q2 please keep them somewhere locally so you don't need to query the server again.

1. (**18 pts**) Use the Python function ***getQ1*** in "**client.py**" given in the assignment package to communicate with the server. The server will send you a number **n** and the number **t**, which is the order of a subgroup of Z_n^*. Please read the comments in the Python code.

Consider the group Z_n^*.

a. (**4 pts**) How many elements are there in the group? Send your answer to the server using the function *checkQ1a.*

b. (**8 pts**) Find a generator in Z_n^*. Send your answer to the server using the function *checkQ1b.*

c. (**8 pts**) Consider a subgroup of Z_n^*, whose order is **t**. Find a generator of this subgroup and send the generator to the server using function *checkQ1c.*

2. (**10 pts**) Use the Python code ***getQ2*** in "**client.py**" given in the assignment package to communicate with the server. The server will send 2 numbers: **e**, and **c**.

Also, **p** and **q** are given below where n=p×q

p =
16381263243811640233465195523887788805147169859580069932297961503570
31053534985989000177544790827453903051834803263861939287620230066973
25502630355995540302095536983747674239699082775937971908945314983176
63963471952308266465512528622033998128204311757643510859226574447467
28263344544203258472332091180537454 79

q =
16799131140628182989327790751738092674329777043723781769808884372983
74136804071210359937249424243280491002269030669194189635767391307543
75674323262394889417412537943169688299724092631996519692955388293697
04833154003066950459141910043866095248690360658156983609093060836948
68713568250286545693860866740538461 73

Compute m = c^d mod n (where d = e^{-1} mod φ(n)). Decode m into a Unicode string and send the text you found to the server using the function ***checkQ2***.

3. (**18 pts**) Consider the following attack scenario. You obtained the following ciphertexts that are encrypted using SALSA20 and want to obtain the plaintexts. Luckily, the owner of the messages is lazy and uses the same key and nonce for all the messages.

Key: `14192977154127950076`

ciphertext 1:

```
b'1U\xe0N\xb6\x8c\x19<H\xac\x1f]bm\x0f\xe8\xe9z\xfe*\xad\xaa\x8e@\x81\x8f
E\xcfBe\xd0\x96\xe0\x08t\xef\t\x9bg(\x86`/
8_\xcc\xdbF\xde\x13w\xb1e\xec\x92Au\xfd\xea\xbeU\xf1\xda
\xb1\xc5D\xb1\xf9\x9as\xd9?{z
\x90R[\xee\xe0XLv=\xd9\x10jN\xdc\x87\x82\xdfO%Z\xb7P'
```

ciphertext 2:

```
b'N\x0e\xb6^\xccU\xe0\x8b\x1e4\r\xbd\x1eJc|\x03\xb8\xe8|\xfd,\xb0\xb2\x8b
K\xc6\xc6_\x81U\x7f\xd4\x86\xa9\x13w\xff\t\x9fa{\x85!(;%\x11\xcd\x94]\xdd
\x13l\xbb0\xf5\x8fKn\xf5\xe4'
```

ciphertext 3:

```
b'\xccU\xe0N\x0e\xb6^1\x99\x1fy\r\xb2\x06Jcm\x0f\xf1\xe83\xf1i\xb4\xbf\x9
0V\xce\x88\x16\x98^b\x91\x8a\xa1\x02;\xf7H\x92w{\x93,-o8\x17\xcf\x94D\xdb
@z\xbf{\xfb\x92\x04m\xf3\xab\xab\x1b\xf6\x9b7\xfe\xd2\x01\xf0\xe6\x9e7\xc
8ww4e\x90\x01D\xee\xe1ULh9\xd9\x11jW\xdc\x95\x84\x9aE.\x1b'
```

However, during the transmission of the ciphertexts, some bytes of 2 out of the 3 messages were corrupted. One or more bytes of the nonce parts are missing.
Attack the ciphertexts and find the messages. (See the Python code **salsa.py** in Sucourse+)
(**Hint:** You can assume a new Salsa instance is created for each encryption operation)

4. **(12 pts)** Solve the following equations of the form $ax \equiv b \bmod n$ and find all solutions for x if a solution exists. Explain the steps and the results.

a. n = 216354984213419843216841324876541321321684631320165468132166 6
a = 790561357610948121359486508174511392048190453149805781203471
b = 789213546531316846789795646513847987986321321489798756453122

b. n = 321365854986513516897965132165847984613211347846321351685466 6
a = 789651315469879651321564984635213654984153213216584984653138
b = 798796513213549846121654984652134168796513216854984321354987

c. n = 546513216588468465213418949851321123158465132184965489749822 2
a = 654652132165498465231321654946513216854984652132165849651312
b = 987965132135498749652131684984653216587986515149879613516844

d. n = 628586750910622229500189454278765738384656297901015675064224 4
a = 798442746309714903987853299207137826650460450190001016593820
b = 263077027284763417836483408268884721142505761791336585685868

5. (**10 pts**) Consider the following binary connections polynomials for LFSR:

$p_1(x) = x^7 + x^5 + x^3 + x + 1$
$p_2(x) = x^6 + x^5 + x^2 + 1$
$p_3(x) = x^5 + x^4 + x^3 + x + 1$

Do they generate maximum period sequences? (**Hint:** You can use the functions in **lfsr.py**)

6. (**12 pts**) Consider the following sequences that are generated using different random number generators. Which of the sequences are predictable? (**Hint:** You can use the functions in **lfsr.py**)

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

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

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

7. (**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 =

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

Also, encrypted in the ciphertext you also know that there is a message to you from one of the instructors; and therefore, the message ends with that instructor's name (Turkish characters mapped to English letters. Such as ş -> s). Try to find the connection polynomial and plaintext. Is it possible to find them? Explain if it is not.

Note that the ASCII encoding (seven bits for each sASCII character) is used.
(Hint: You can use the `ASCII2bin(msg)` and `bin2ASCII(msg)` functions (in **hw2_helper.py**) to make conversion between ASCII and binary)