Homework #2

Due date: 12/11/2022
Notes:

- If you used Python codes for your 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**", "**hw2_helper.py**", and "**client.py**" that you can use for the homework questions.
- 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 IT website for VPN connections.
- **The server's IP address** is provided as "http://10.92.55.4:6000". This IP is provided by default in **client.py.** Check the code.

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 function *checkQ1a.*

b. (**8 pts**) Find a generator in Z_n^*. Send your answer to the server using 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 =
12971142097853774608886730934213242678590198968987459448589637155501
99865737054261727888057261785094677480406791687340958844335970176040

12172054368990172572715857537355524013819947862920969421702067385445
12224267306495899196866613854438036552045602995241496202871180617578
49281318261278858206440919513443183 87

q =
17406667240508597265780888177897852058280976323514735837433240996632
29872907454164052204143230047829067573625791571179144949271984426455
81197584273451379119673753279114693557694861941678350357667191083878
10082892019850377453927128926336366647364198130180304138099281532660
26076063619436733737013253098735 1081

Knowing that **d = e^{-1} mod ϕ(n)**. Compute **m = c^{d} mod n**. Decode m into 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 following ciphertexts that are encrypted using SALSA20 and want to obtain the plaintexts. Luckily, owner of the messages is lazy and uses same key and nonce for all the messages. You also know that the owner uses a 64-bit number the key.

Key: `14656892184006070584`

ciphertext 1:
```
b"Vbq\x8a\xe3\xb7Rgl-
\x14\x8bNS\xeb\x01\xbd\xdf\x1f\x14\x84{\xdanX,\xa5\x98RM\x98\r\xd7\x1e\
x9dO\x14\xa7\x8cX\xcb\xad\xf2\xc9\x1f\xc1]\xef\x908I\xe0\xcf\x10%.ulh\x
e7\xd6\x9d<\xb9a\xda\xb0\xa2d\xe9\x18\xef9\x99ttP\x9blw\x0e\xe7\xd6\xbb
1\xf4?\x16kf\x87\x19\xbe\x94O\xe8\x1d\x08\xe4\xff)\x99']\xda\x191=|H"
```
ciphertext 2:
```
b'\eda\x01q+]\x8c\x06[\xa2/\xb8\xcaX\x1f\x8f:\xc97\x0f)\xa5\x84Y\t\xdc\
x07\xd2L\xb3V\x14\xad\x8bU\x99\xa3\xf2\x9dK\xc8V\xab\xdd\nS\xe9\xcf\x05
$r,\t<\x9e\xd0\x9b<\xbcx\x99\xaf\xed7\xf9\x13\xff9\x88r
\\\x9b}>\x1d\xeb'
```
ciphertext 3:
```
b'ea,\x14\x88NW\xbfh\xb9\xcdX\x0f\x83}\xc0cX5\xa5\x9e\x1e^\xd0\x03\xc5\
x1e\xa3U@\xa1\x85H\xc0'
```

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+ to get your hands on this stream cipher).
(Hint: You can assume new Salsa instance is created for each encryption operation)

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

a. n = 15930899774896282134199789358470375202928146251919022163719 75
a = 10854844595480699462641909943250659815474794903573851741986 06

b = 95318974643982165609408435625572584452874934183471678444579 4

b. n = 160438127964801337012133761194967786483003991766832070490691 2
a = 363513302982222769246854729203529628172715297372073676369299
b = 130689943291728127833514099336130167804931752775925797856824 1

c. n = 591375382219300240363628802132113226233154663323164696317092
a = 114360136501326441636144142972711086736662009148382893288986 2
b = 368444135753187037947211618249879699701466381631559610698826

d. n = 722232417010638129500185345578613705150903797901014019064 96
a = 798442746309714903219853299207137826650460450190001016593820
b = 263077027284763417836483401088884721142505761791336585685868

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

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

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 sequences are predictable? (**Hint:** You can use the functions in lfsr.py)

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

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

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

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

Also, encrypted in the ciphertext you also know that there is a message to you from the instructor; and therefore, the message ends with the instructor's name. 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)