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.*

For this question the n,t pair provided to me is

$n = 233$, $t = 8$

Answer for the first question is 232.

We need to find the count of the numbers smaller than 233 and coprime with 233.

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

We need to find a number such that it can generate all the numbers in the congruence class of 233.

The number satisfies this condition is 3.

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.*

We need to find a number that can generate t items (a subgroup) with it powers in modula 233.

This number is 12.

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 =
1638126324381164023346519552388778880514716985958006993229796150357031053534985989000177544790827453903051834803263861939287620230066973255026303559955403020955369837476742396990827759379719089453149831766396347195230826646551252862203399812820431175764351085922657444746728263344544203258472332091180537454 79

q =
1679913114062818298932779075173809267432977704372378176980888437298374136804071210359937249424243280491002269030669194189635767391307543756743232623948894174125379431696882997240926319965196929553882936970483315400306695045914191004386609524869036065815698360909306083694868713568250286545693860866740538461 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***.

P and Q are prime numbers. So, phi(n) is equal to n*(1-1/p)*(1-1/q) = (p-1)*(q-1)

Later, I used egcd function to find the inverse of d in modula phi(n). I wrote a right to left binary exponentiation algorithm to effectively find the c^d mod n. Than, I decoded the resulting message.

The result is:

I think I have 986 unread e-mails. Is that a lot?

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\x
8fE\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\x
8bK\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\
x90V\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\xd
2\x01\xf0\xe6\x9e7\xc8ww4e\x90\x01D\xee\xe1ULh9\xd9\x11jW\xdc\x95\x84\x
9aE.\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)

Ciphertext 3 is the true text. It has a true nonce and it can be decrypted easily.

Ciphertext 1 and 2 has some missing and corrupted bytes in the nonce. To decrypt, ciphertext 1, I skip the first 5 bytes. Similarly, to decrypt ciphertext 2, I skip the first 7 bytes. I saw this manually by eye because some part of the nonce are similar. Also, we know that nonce is 8 byte; so, the corrupted part can only be in the first 8 bytes.

decoded text1: The first principle is that you must not fool yourself and you are the easiest person to fool.
decoded text2: Somewhere, something incredible is waiting to be known.
decoded text3: An expert is a person who has made all the mistakes that can be made in a very narrow field.

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 = 2163549842134198432168413248765413213216846313201654681321666
   a = 7905613576109481213594865081745113920481904531498057812034 71
   b = 7892135465313168467897956465138479879863213214897987564531 22

b. n = 32136585498651351689796513216584798461321134784632135168546666
a = 78965131546987965132156498463521365498415321321658498465313 8
b = 79879651321354984612165498465213416879651321685498432135498 7

c. n = 5465132165884684652134189498513211231584651321849654897498222
a = 6546521321654984652313216549465132168549846521321658496513 12
b = 98796513213549874965213168498465321658798651514987961351684 4

d. n = 628586750910622229500189454278765738384656297901015675064 2244
a = 79844274630971490398785329920713782665046045019000101659382 0
b = 26307702728476341783648340826888472114250576179133658568586 8

The algorithm that I used to solve this question as the following.

```
# we first check gcd of n and a
# if it is 1,
   # then we will find the inverse of a mod n
   # we will multiply b with the inverse of a mod n
   # then we will take mod n of the result
   # we will print the result
# else
   # we will divide n,a,b with gcd of n and a
   # if b is not divisible by gcd of n and a
      # we will print "No solution"
   # else
      # let's say the new n is n1, new a is a1, new b is b1
      # we will find the inverse of a1 mod n1
      # we will multiply b1 with the inverse of a1 mod n1
      # then we will take mod n1 of the result
      # there would be gcd(n, a) solutions
```

a)

```
###################### Question 4A ######################
The solution is:
111563634314800439832213513866100835794512614711477009341482 6
```

b)

```
###################### Question 4B ######################
There are no Solutions!
```

c)
###################### Question 4C ######################
The solutions are:
184045108563697882707983051431202214996694119114301061438590 0
457301716857932115314692526356862776575926685206783806313501 1

d)
###################### Question 4D ######################
The solutions are:
120574576795431477647425259344685590574672051332591719355582
169204145407198705139789889504159993653631279608513090701614 3
326350833134854262514837253073851428249795354083767009467670 4
483497520862509819889884616643542862845959428559020928233726 5

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**)

Since the initial state of LFSR affects the maximum period, I used all possible initial states the check the maximum period. For instance for the p1(x) function there are 2^7 = 128 possible initial states. I checked all possible maximum periods using these initial states.

The answer for the question is following:
P1 period is
Max period: 127
Max period initial state: [1, 0, 1, 1, 1, 0, 0]
L and C(x): (7, [1, 1, 0, 1, 0, 1, 0, 1])

P2 period is
Max period: 21
Max period initial state: [1, 0, 1, 0, 1, 0]
L and C(x): (6, [1, 0, 1, 0, 0, 1, 1])

P3 period is
Max period: 31
Max period initial state: [0, 0, 1, 1, 1]
L and C(x): (5, [1, 1, 0, 1, 1, 1])

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, 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, 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, 1, 1, 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, 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, 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, 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]

Using the Berlekamp-Massey algorithm we can detect the linear complexity of a sequence. Berlekamp-Massey algorithm can work correctly when the given complexity is smaller than the half of the given sequence. If the complexity of a sequence bigger than the half of the length of the given sequence, it cannot ensure that the output polynomial is correct and the sequence is predictable.

I run the Berlekamp-Massey algorithm for the sequences:

Len x1 75 ,BM (36, [1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1])
Len x2 80 ,BM (43, [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1])
Len x3 90 ,BM (31, [1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1])

As it can be shown from the results, the complexity of x1 and x2 sequences are big enough to make them unpredictable.

75 / 2 = 37 which is to close to 36. So we cannot ensure that this sequence is predictable.
80 / 2 = 40 which is smaller than 43. This sequence is not predictable.
90 / 2 = 45 which is bigger than 31. This sequence is predictable.

X1 = Unpredictable
X2 = Unpredictable
X3 = Predictable

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, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 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, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 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, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 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, 0, 0, 1, 0, 0, 0, 1, 0,
0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 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, 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)

The end of the message is 'Erkay Savas' which generates a 77 length bitstream.

But this bitstream is placed at the end of the message. While the stream cipher starts working from beginning to end(LFSR). Therefore, I reversed the given ciphertext and known plaintext. Now, the known plaintext placed at the beginning. Using XOR, I found first 77 bit length keystream. I put this keystream in the BM algorithm to find the polynomial and the complexity of the polynomial.

L: 27
Cx: [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

After, I started my initial state using the complexity and my known 77-bit length keystream. Using LFSR, I generated all the keystream. With the keystream and the ciphertext, I decrypted the message. Finally, I take reverse of the decrypted message to find the original form.

Dear Student,
Outstanding job on tackling this challenging problem!
Congratulations!
Best, Erkay Savas