Homework #2

Due date: 05/11/2021
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", "client.py" and "hw2_helper.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**. Then, you can run your code as usual. See IT website for VPN connections.

  An alternative server is also provided at "cryptlygos.pythonanywhere.com" in case you have problems with VPN. (Establishing a VPN connection along with **Google Colab** is not straightforward so you can use this server. But it may be a bit slower.)
  Both urls are stated in the **client.py.** Check the code.

1. (**20 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.*

Answer: I wrote a function that takes number n as argument and returns an array, which contains all numbers that are relatively prime to n. In our case, server sent me n = 179. I call phiList function with n, and it returns the group as array. The answer is the length of that array, which is 178. I sent my answer to the server and it said Congrats! (I submitted my Colab file as well, you can see my code in there. Thanks)

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

Answer: For this question, phiList function, which is explained 1.a, and findGenerators function was used. In findGenerators function, firstly, phiList function was used (with n = 179) for founding all elements in the Z_{179}^*. We iterate over each number in the group. In each iteration there is another for loop, which iterates over each number in the group. In inner for loop we calculate generated number as num^{power}mod(n). *num* comes from outer for loop and *power* comes from inner for loop. Then we append generated number to the generated number array. If this array is equal to array which represents the group Z_{179}^{*}, we append *num* to generators array. At the end of the function, generators array is returned.

All generators were obtained with the help of this function and they were printed. Also, all elements in the generators array were send to the server, and Congrats! Message were received for each number in generators array which was obtained by findGenerators function. (I submitted my Colab file as well, you can see my code in there. Thanks)

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 =
2373654091808847940781787603170106664430106488295887529616721481901443837401166167283021095553950725206699938406735615905683587778141947902331314913944 4707

q =
621798964045649924436177098942410545206243555586582884226961788392746118331366622414301626940762314015455844491282789884049705800159851405424510870497 94069

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

Answer:

Firstly we obtained e and c from the server. Then we calculate n as p*q.

ϕ(p*q)=(p−1)*(q−1), where n = p*q, therefore;
ϕ(n)=(p−1)*(q−1) => we can obtain ϕ(n) with this formula

We know $d = e^{-1} \bmod \phi(n)$. Therefore, we can use modinv function which was provided to us. We can obtain d as d = modinv(e, $\phi(\mathrm{n})$)

We can calculate $m = c^d \bmod(n)$ with python pow(c,d,n) function where pow(c,d,n) = c^d mod(n).

Finally, we can convert *m* to the byte array with using built-in to_byte function. Then we can decode this byte array into string with using utf-8.

Then i sent the message i found to the server, and it returned the message Congrats!

(I submitted my Colab file as well, you can see my code in there. Thanks)

3. (**20 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 number pi as the key.

Key: `314159265358979323`

ciphertext 1:
```
b'1v-
\xdda\x9d\x13\xf5y\xd4M\xcc\xc2\xd5\xc9\xe8\xca\xfcF\xe1\x7f\xdd\xabM,=
c\xa6\x9e\xd2M\x11;9Bpna\x91\xb8\xf5z>\x0cZ\x83\x11\xa7\x01\x1b\xc2\xc5
$>\x10\xa2>"#\xc0\x98\xa4\xc2\xbd\xa1\xce\x0f\x17]\x8c_\xee\xadT|'
```
ciphertext 2:
```
b'\x9d\x131v-
\xdda\xe9\xf3,\xca\x02\xd1\xc9\x9a\xda\xe1\xce\xfcM\xed1\xdb\xb9\r,\x1b
-
\xa6\x88\x84JTo7N>p}\x9b\xfb\xa6e?\x0bQ\xc6_\xa7\x1d\x1a\x87\x8c78\x1a\
xa9\x7f!!\xce\xdd\xe9\xd6\xbd\xf5\x9a\t\x17G\xc9K\xf2\xecDl\xb0\xca\x86
\xa6\xd7\xde\xe5zxf\xd0\xado\xea'
```
ciphertext 3:
```
b"\x00\x04\x00\x00\x00\x00\xfd7\xc1\x02\xcf\xc9\x82\xc4\xe1\xc7\xf1D\xe
f\x7f\xdd\xab\x10,\x00,\xea\x9d\xc1IC!qJ1ma\x9b\xba\xe7f>\x01N\x83\x0b\
xa7\x0c\t\xde\xc537\x1b\xfby='\xca\x89\xe8\xda\xee\xf3\xdf\x14\x06V\xc5
[\xf5\xadOj\xa9\xc1\x86\xb4\xdd\x8d\xff}|f\xd2\xado\xe6r\xf6\xcf\xe3\xf
1H\xa6\xdaA\xcb\x17"
```

However, during the transmission some bytes of two of the messages 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 new Salsa instance is created for each encryption operation)

Answer: The key length doesn't match with the SALSA20 key length, however we know that the owner uses the number pi as the key, so that we can adjust the key length accordingly. Therefore, we can found the key as 31415926535897932384626433832795028841971693993751058209749445923078164062862. Then we can convert the key to bytes with built-in to_byte function. Next, we know that the nonce is the first 8 byte of the ciphertext, and question provides that owner uses same nonce for the all messages. Also question says that two messages are corrupted, therefore we should generate 3 possible nonce and trying decryption with these possibilities. Then I tried to use the same approach that we saw in salsa20.py code which was provided in sucourse. However, results that I obtained seems I did something wrong. (I submitted my Colab file as well, you can see my code in there. Thanks)

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 = 100433627766186892221372630785266819260148210527888287465731
   a = 336819975970284283819362806770432444188296307667557062083973
   b = 25245096981323746816663608120290190011570612722965465081317

   b. n = 301300883298560676664117892355800457780444631583664862397193
   a = 1070400563622371146605725585064882995936005838597136294785034
   b = 1267565499436628521023818343520287296453722217373643204657115

   c. n = 301300883298560676664117892355800457780444631583664862397193
   a = 608240182465796871639779713869214713721438443863110678327134
   b = 721959177061605729962797351110052890685661147676448969745292

   Answer:
   1. if ax ≡ b mod n, then there is one solution if gcd(a,n) = 1.
   2. If gcd(a,n) not equal to 1, then if b is divisible by gcd(a,n), there exists gcd(a,n) solutions that are obtained by ax/d = b/d mod n/d where d is gcd(a,n).
   3. If gcd(a,n) not equal to 1, then if b is not divisible by gcd(a,n), there is no solution.

   In function findSolution above algorithm was implemented. Then we call that function for our three cases and the followings are obtained;

   a. There is one solution (case 1): 56884393062303769019751445983612369117060043083722821988604
   b. No solution (case 3)

c. There are three solutions (case 2):
    i. 960927937475610528842702189849989036171710514555173902 7963
    ii. 110042907140942997509799652683766709621865315673440026493694
    iii. 210476534907129889731172283469033528882013526201328313959425

(I submitted my Colab file as well, you can see my code in there. Thanks)

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

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

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

6. (**12 pts**) Consider a random number generator that generates the following sequences. Are they unpredictable? (**Hint:** You can use the functions in lfsr.py)

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

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

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

Answer: In this question I used provided BM() function, which takes a sequence as parameter and can return the complexity of that sequence as L. I gave each sequence to the BM function and obtain their linear complexities as 31 (for all of them). Sequence can be said unpredictable when its linear complexity is bigger than its length / 2. However, linear complexities for all sequences were smaller than their respective length/2, therefore, all of the sequences are predictable. (I submitted my Colab file as well, you can see my code in there. Thanks)

7. (**16 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, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1,
1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0]

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 ASCII 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)