# **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: in Q1.py, my n is 271. Question asks that how many prime numbers of 271. You can see my code named is Q1.py answer is 270, since 271 is prime number. Also, I check my answer with checkQ1a

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

Answer: if a number is generator of $Z_{271}^*$ , which means power of number between 1-272 modules of 271 give list of prime numbers of 271. They can generate each other. You can see my code Q2.py. there are 72 generator and 72 congrats when call checkQ2b function. Also from formula:

270*(1-1/2) *(1-1/3) *(1-1/5) =72

(**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.*

In Q1.py, My t is 27. By Langrange Theorem(The order of a subgroup divides the order of the group )

To find a subgroup of 27, first I create order_list by using primitive elements rule which is that if g be a primitive elements for the prime p(271). Then if n is an integer, then g**n = a (mod n) if and only if n = 0(mod p-1) so added by if condition `if (p-1) % i == 0:`
I crete order_list. then function of findpowerGroup create generator list for every order. Since my order is 27. Just I found it. Also I check it.

**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 =
6217989640456499244361770989424105452062435555865828842269617883927461183313666224143016269407623140154558444912827898840497058001598514054245108704979 4069

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

Answer: in Q2.py,I just follow the mathematical steps actualy by using code, I find m by following steps first find n then find phi and find d by modinv function. In decode part my code is not worked I run my code another computer without doing any changed and I found ptext

```
"Answer to the ultimate question of life, the universe, and everything is not 42. it is 271"
```

And calling checkQ2 function I check my answer.

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**: `3141592653589793 23`

**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: in Q3.py, I used your code replace random secret with key in the question and replace ctext with ciphertext. End the of the decode I get error.

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 = 1004336277661868922213726307852668192601482105278882874657 31
   a = 3368199759702842838193628067704324441882963076675570620839 73
   b = 2524509698132374681666360812029019001157061272296546508131 7

   **b.** n = 3013008832985606766641178923558004577804446315836648623971 93
   a = 1070400563622371146605725585064882995936005838597136294785034
   b = 1267565499436628521023818343520287296453722217373643204657115

   **c.** n = 3013008832985606766641178923558004577804446315836648623971 93
   a = 6082401824657968716397797138692147137214384438631106783271 34

b = 721959177061605729962797351110052890685661147676448969745292

Answer: in Q4.py, for part a gcd(a,n) is 1 so it has 1 solution. b gcd(a,n) is not equal 1 so I check divisibility of b. they cannot divisible b so they does not have solution. Part c, it is divisible by gcd(a,n). that means there are exactly 3 solution.Using formula of lecture slayt I found solution set for part b. and I found x for part a and c. also I check their inversibility for x since their a does not have inverse. To solve this equation, I solved x ≡ a*b mod n. so just a3 have inverse so x3 calculatable.

**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)

Answer: in Q5.py, we learned from lecture, maximum period sequence formula: 2**L-1. Since both L is equal to 5. 2**5-1 = 31 and p1 reach maximum period sequence; however, L is p2 cannot reach maximum period sequence. P2 has same L so maximum period should be 31 but pr cannot reach you can see its first period value.

**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 Q6.py, all of them predictable. I used BM for L(s**n) value and I used it expected linear complexity of a random sequence. I calculate cal = n/2+2/9 value n is  len(x) and compare with L since L = 31 is smaller than Cal they are predictable. It is repeat theirself inside of x.

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