**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 . Please read the comments in the Python code.

Consider the group .

* 1. (**4 pts**) How many elements are there in the group? Send your answer to the server using function *checkQ1a.*

*There are 240 elements in the group*

* 1. (**8 pts**) Find a generator in . Send your answer to the server using function *checkQ1b.*

A generator is 317

* 1. (**8 pts**) Consider a subgroup of , whose order is **t**. Find a generator of this subgroup and send the generator to the server using function *checkQ1c.*

The generator is 111

1. (**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 = 129711420978537746088867309342132426785901989689874594485896371555019986573705426172788805726178509467748040679168734095884433597017604012172054368990172572715857537355524013819947862920969421702067385445122242673064958991968666138544380365520456029952414962028711806175784928131826127885820644091951344318387

q = 174066672405085972657808881778978520582809763235147358374332409966322987290745416405220414323004782906757362579157117914494927198442645581197584273451379119673753279114693557694861941678350357667191083878100828920198503774539271289263633646647364198130180304138099281532660260760636194367337370132530987351081

Knowing that **d = e-1 mod φ(n)**. Compute **m = cd mod n**. Decode m into Unicode string and send the text you found to the server using the function ***checkQ2***.

P and q are prime numbers so we can say phi(n) is equal to (p-1)x(q-1)

I used the modinv function and pow function to calculate m and then decoded it into a Unicode string

1. (**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\xe7\xd6\x9d<\xb9a\xda\xb0\xa2d\xe9\x18\xef9\x99ttP\x9blw\x0e\xe7\xd6\xbb1\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)

Firstly I checked all three ctexts to find out which one didn’t have an error. And then using the key and nonce of the working one, ciphertext 1, I looked at ctext 2 and 3 accordingly

Ctext1 = In the Imaginary world of Witchcraft and Wizardry, You pick side either with Grindelwald or Dumbledore

Ctext2 = This might be a wrong decryption of the message. It is up to you to decide

Ctext3 = Do what is right not what is easy

1. **(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.
   1. n = 1593089977489628213419978935847037520292814625191902216371975

a = 1085484459548069946264190994325065981547479490357385174198606

b = 953189746439821656094084356255725844528749341834716784445794  
solution is 778214478105812676636756719791275483076257993476674633986349

* 1. n = 1604381279648013370121337611949677864830039917668320704906912

a = 363513302982222769246854729203529628172715297372073676369299

b = 1306899432917281278335140993361301678049317527759257978568241

solution doesn’t exist because gcd of a and n does not divide b

* 1. n = 591375382219300240363628802132113226233154663323164696317092

a = 1143601365013264416361441429727110867366620091483828932889862

b = 368444135753187037947211618249879699701466381631559610698826

there are 2 solutions for this because gcd of a and n is 2 and b is divisible by 2 =

214452804613963486637685100244716357445929679678527437334799

and

510140495723613606819499501310772970562507011340109785493345

* 1. n = 72223241701063812950018534557861370515090379790101401906496

a = 798442746309714903219853299207137826650460450190001016593820

b = 263077027284763417836483401088884721142505761791336585685868

there are 4 solutions for these values =

55063481190493260344408407655808631312324038043859213421933 896049914695400631894506737412603426006253201283161992061 18951860339961353869399140376877946054778848148808512468685 37007670765227307106903774016343288683551443096333862945309

1. (**10 pts**) Consider the following binary connections polynomials for LFSR:

p1(x) = x6 + x5 + x4 + x+ 1

p2(x) = x6 + x2 + 1

p3(x) = x5 + x3 + 1

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

for p1 first period is 63 and max period is also 63 so it generates

for p2 first period is 14 and max period is 62 so it doesn’t generate

for p3 first period is 31 and max period is also 31 so it generates

1. (**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, 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, 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]

Using the berlekamp massey algorithm I found out their linear complexity and then evaluated them with their expected linear complexity.

X1 complexity = 44.7 , expected complexity = 45 so not predictable

X2 complexity = 29 , expected complexity = 43.7 so predictable

X3 complexity = 37 , expected complexity = 51.2 so predictable

1. (**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, 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, 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, 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)

I used “Atıl Utku Ay” and encoded it into binary, then because I knew this had to be the ending of the message I reversed both the ctext and binary of the known part. And then I used the reverse known part to make a part of the key. And then found the pattern of the key and used it to find the encoded plain text and then reversed it at the end again finding the result =

Dear Student,

You have worked hard, I know that; but it paid off:)

You have just earned 20 points.

Congrats!

Best, Atil Utku Ay