Homework #2

Name & ID: Ege Kaan Özalp / 28989
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^*.

Check out the file Question1_EgeKaanOzalp_CS411.py for codes and more details!

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

Answer: 460.

Explanation: To create the group Z^*_n, I need to find every num ∈ {1...n-1} such that gcd(num, n) = 1. Adding such numbers to the group, I found that the answer should be 460. I received the "Congrats!" message from the server.

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

Answer: 3.

Explanation: To find a generator, I need to find an element e ∈ Z^*_n such that Z^*_n can be created by using the powers of e. Applying this logic to the code, I found the generator should be 3. I received the "Congrats!" message from the server.

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.*

Answer: 23.

Explanation: To find a subgroup of Z^*_n whose order is t, I need to find a num such that (num^i) mod(n) ≠ 1 for i < t and (num^t) mod(n) = 1. Implementing this logic to the code, I found that the num should be 23. I received the "Congrats!" message from the server.

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 =
16381263243811640233465195523887788805147169859580069932297961503570
31053534985989000177544790827453903051834803263861939287620230066973
25502630355995540302095536983747674239699082775937971908945314983176
63963471952308266465512528622033998128204311757643510859226574447467
28263344544203258472332091180537454 79

q =
16799131140628182989327790751738092674329777043723781769808884372983
74136804071210359937249424243280491002269030669194189635767391307543
75674323262394889417412537943169688299724092631996519692955388293697
04833154003066950459141910043866095248690360658156983609093060836948
68713568250286545693860866740538461 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***.

Check out the file Question2_EgeKaanOzalp_CS411.py for codes and more details!

Answer: "I think I have 171 unread e-mails. Is that a lot?"

Explanation: Received the e and c values from the server. Found n = p*q. Since p and q are both prime numbers: phi(p) = p-1 and phi(q) = q-1. Also, since they are both prime numbers, they are also coprime numbers. Therefore, phi(n) = phi(p*q) = phi(p) * phi(q) = (p-1)*(q-1).

phi(n) =
27519098948194396865394509473949139518515084937354564552013704479888
95921051348068510408520518434558652508119576806814719898466611960060
30031847287765127828191443960624572416311044354035572714510882500042
52439796127285449430044600366723508594256191700059310816873405910691
33796949922651964182669686823433768395711909791318664256371585555924
20147831772235808010116285384862982351999285655130475485558137636025
65830549276713150555126342230399118647104044079105095965060431441835
99874636739571211854664820909842322766800975619596564374199374941045
96652412005281822114449180508874723323496323821097980291262504562526
10216

After finding phi(n), I found the d value which is:

d = e^{-1} mod(phi(n)) =
25838750868222929355985480633600267658651819741631339573401939788655
75126621247110730379295192933379154220139947314525289767101251604863
39328899948984493267357602582304630937317236877120254513834937617852
15344325547581930928804838351526867703182383910621867659525820452735
32538696731687888559159053690372113807970727231620957690974699833749
64084827045753208510762006984275969916851819776978866169737072188346
95201628316116811525427125752125642697926648500402859243064684108451
18370610191009501457459799842628622176625734055628131789770523748894
23363716005798807796244374215035002185810427721985902070357337580122
40047

After that, I found the m value using c, d and n values:

m =
28813416906475393180373254504996578266905911395353487049329095340247
44299638226268210068060099329454047537255613953087

Finally, I decoded the m value and found the message:

Message = I think I have 171 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)

Check out the file Question3_EgeKaanOzalp_CS411.py for codes and more details:

Answers:

Decoded text for ciphertext3: An expert is a person who has made all the mistakes that can be made in a very narrow field.

Decoded text for ciphertext2: Somewhere, something incredible is waiting to be known.

Decoded text for ciphertext1: The first principle is that you must not fool yourself and you are the easiest person to fool.

Explanation:

I started with trying to decode ciphertexts. At the beginning, I could only decode ciphertext3, which means it is the only message that have not got any corrupted bytes. After decoding ciphertext3, I had the key and the nonce value. Then, to decode ciphertext1 and ciphertext2, I tried to figure out the missing parts of the nonce by iterating up to 8 bytes since nonce is an 8-byte value. To do so, I have created a loop to iterate over all 8 possibilities for each of the ciphertexts. Then I found the decoded texts.

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.

Check out the file Question4_EgeKaanOzalp_CS411.py for codes and more details:

a. n = 216354984213419843216841324876541321321684631320165468132166 6
a = 790561357610948121359486508174511392048190453149805781203471
b = 78921354653131684678979564651384798798632132148979875645312 2

Answer:
gcd(a, n) = d = 1, therefore, there is exactly one solution:
111563634314800439832213513866100835794512614711477009341482 6

b. n = 32136585498651351689796513216584798461321134784632135168546 66
a = 78965131546987965132156498463521365498415321321658498465313 8
b = 79879651321354984612165498465213416879651321685498432135498 7

Answer:
gcd(a, n) = d = 2 and 2 does not divide b. Therefore, there is no solution.

c. n = 54651321658846846521341894985132112315846513218496548974982 22
a = 65465213216549846523132165494651321685498465213216584965131 2
b = 98796513213549874965213168498465321658798651514987961351684 4

Answer:
gcd(a, n) = d = 2 and it divides the b value. Therefore, there are 2 solutions:

- **184045108563697882707983051431202214996694119114301061438590 0**
- **457301716857932115314692526356862776575926685206783806313501 1**

d. n = 628586750910622229500189454278765738384656297901015675064224 4
a = 798442746309714903987853299207137826650460450190001016593820
b = 263077027284763417836483408268884721142505761791336585685868

Answer:
gcd(a, n) = 4 and it divides the b value. Therefore, there are 4 solutions:

- **120574576795431477647425259344685590574672051332591719355582**
- **169204145407198705139789889504159993653631279608513090701614 3**
- **326350833134854262514837253073851428249795354083767009467670 4**

- **48349752086250981988988461664354286284595942855902092823372 65**

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**)

Check out the file Question5_EgeKaanOzalp_CS411.py for codes and more details:

P_1:
Initial state: [1, 1, 1, 0, 0, 1, 1]
First period: 127
Since max period is 2^7-1 = 127 and the period of p_1 is 127, it generates the maximum period sequence.

P_2:
Initial state: [0, 0, 0, 1, 0, 0]
First period: 21
Since max period is 2^6-1 = 63 and the period of p_2 is 21, it does not generate the maximum period sequence.

P_3:
Initial state: [0, 0, 0, 0, 1]
First period: 31
Since max period is 2^5-1 = 31 and the period of p_3 is 31, it generates the maximum period sequence.

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]

Answer:
Expected linear complexity of the sequence of x1: 37.72222222222222
The complexity of x1 based on BMA is: 36
Since the complexity of x1 less than the expected complexity, x1 is predictable.

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, 1, 1, 0, 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]

Answer:
Expected linear complexity of the sequence of x2: 40.22222222222222
The complexity of x2 based on BMA is: 43
Since the complexity of x2 bigger than the expected complexity, x2 is not predictable.

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, 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]

Answer:
Expected linear complexity of the sequence of x3: 45.22222222222222
The complexity of x3 based on BMA is: 31
Since the complexity of x3 less than the expected complexity, x3 is 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, 0, 1, 1, 0, 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, 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, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 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, 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, 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)