CS411-HW2

Ahmet Furkan Ün

November 12, 2023

Question 1

(a)

After getting the number from the server, let's call it $n = 926$, I used Euler's totient function, $\phi(n)$, to figure out how many numbers are in the group Z^*_{926}. This function counts the numbers less than n that are coprime with 926, in other words $gcd(i, 926) = 1\ \forall i \in [2, 926]$.

When I applied this to $\phi(926)$, it told me there are 462 numbers in the group. I then sent this count to the server using the provided `checkQ1a` function, and the server confirmed that my count was correct.

(b)

A generator within a group is an element that, when raised to distinct powers, generates each of the other elements in the group. To determine a generator for the group Z^*_{926}, my script iteratively tested various elements to check if they are actually the generator. To check whether if a number is generator of Z^*_n, we need to check if it generates all the 462 numbers.

Luckily 3 generates 462 elements of $\phi(926)$ therefore 3 is a generator. So my iterative process did not take much. Then I gave 3 to the `checkQ1b` function to confirm the correctness and my answer is validated.

(c)

To find a generator of a subgroup of Z^*_{926} with an order t = 7, I started checking numbers from 2. For each number the first thing I checked was whether the number is the generator of the group Z^*_{926}. If the number is not an element of the group, it cannot be the generator of the subgroup. For the numbers that are elements of the the group, I checked even further $g^{t=7} \bmod n = 1$. If the equation holds, then the number is a generator of the subgroup with degree t=7. As a result of my approach, I found the generator as 497 and confirmed id with `checkQ1c` function.

Question 2

We know that $n = p * q$, therefore the $\phi(n) = (p - 1) * (q - 1)$. After calculating the $\phi(n)$, to find d, I used modinv function given as helper, and calculated the modular inverse of e with respect to mod $\phi(n)$. After that step the result m should be directly c^d mod n. However the numbers d and c are very big so it is impossible to calculate them with regular exponentiation operation. To overcome this, I used Binary Left-to-Right Algorithm to speed up the process. After that I successfully calculated the m and decoded it into the plaintext:
"I think I have 926 unread e-mails. Is that a lot?"

Question 3

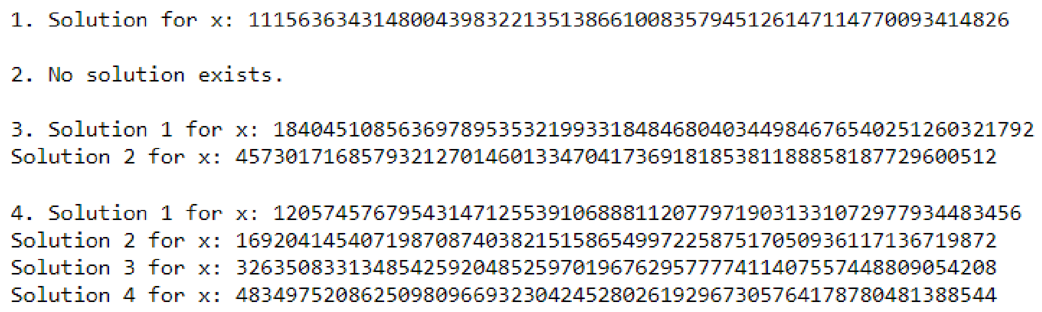
The first thing that I realize from the question that only the 2 of the 3 texts is corrupted. Therefore I can try to decrypt every text with try and except block to get the correct nonce. If the decryption process gives me an error, this means the ciphertext is corrupted. After trying every ciphertext, I found that the ctext3 is not corrected and I successfully decrypted it. Then, by using the same nonce I created a new Salsa instance for every other ciphertext. Since I did not know how many bits are corrupted, for each ctext, I tried to trim first n bits $\forall n \in [1, 9]$

By performing the proccess I described, I obtained
Decoded ctext1: ”The first principle is that you must not fool yourself and you are the easiest person to fool.”
Decoded ctext2: ”Somewhere, something incredible is waiting to be known.”
Decoded ctext3: ”An expert is a person who has made all the mistakes that can be made in a very narrow field.”

Question 4

To solve the equations of the form ax = b mod n, the first thing I do is to find $d = gcd(a, n)$. After finding it there are 3 options.

If d = 1, there is a unique solution which is $(a^{-1} * b)$ mod n
If $d \nmid b$, there is no solution
Else, there are d solutions can be calculated by the formula in the lecture slides.
By applying this procedure, I found:



```
1. Solution for x: 1115636343148004398322135138661008357945126147114770093414826

2. No solution exists.

3. Solution 1 for x: 1840451085636978953532199331848468040344984676540251260321792
Solution 2 for x: 4573017168579321270146013347041736918185381188858187729600512

4. Solution 1 for x: 120574576795431471255391068881120779719031331072977934483456
Solution 2 for x: 1692041454071987087403821515865499722587517050936117136719872
Solution 3 for x: 3263508331348542592048525970196762957777411407557448809054208
Solution 4 for x: 4834975208625098096693230424528026192967305764178780481388544
```

Question 5

To determine whether the functions generates the maximum period or not, I created a random initial state for each polynomial and generate a keystream using LFSR. After that I gave the keystream to the FindPeriod function and compare the results with $2^L - 1$ since it is theoretical maximum period. Here are the results:
Period for $x^7 + x^5 + x^3 + x + 1$: $127 = 127$. Generates maximum period sequences.
Period for $x^6 + x^5 + x^2 + 1$: $21 < 63$. Does not generate maximum period sequences.
Period for $x^5 + x^4 + x^3 + x + 1$: $31 = 31$. Generates maximum period sequences.

Question 6

I used Berlekamp-Massey algorithm to determine the linear complexity (L) of each given binary sequence. Theoretically, the expected complexity should be $E(L(s^n)) = n/2 + 2/9$ where n is the length of the binary sequence. Comparing the expected complexity with Berlekamp-Massey complexity gives me the predictability of the sequences.
x1 is predictable since linear complexity $<$ expected complexity ($36 < 37.72$)
x2 is unpredictable since linear complexity $>$ expected complexity ($43 > 40.22$)
x3 is predictable since linear complexity $<$ expected complexity ($31 < 45.22$)

Question 7

The first thing I did was to find the bit version of the known plaintext ("Erkay Savas"). After that XORed the plaintext with the corresponding ciphertext which is the last 77 bits. After the XOR operation I obtained the last 77 characters of the keystream. To find the connectivity polynomial, I used Berlekamp-Massey algorithm on the keystream obtained from known plaintext and find the Connectivity Polynomial used to in LFSR. The order (L) of the LFSR is 27. Therefore I get the last 27 elements of the keystream, which is the last state of the LFSR when de encrption is done. Starting from the 28th element from the end, I compute the keystream backwards.

After I found the keystream, I XORed it with the ciphertext to obtain the given output:
"Dear Student,
Outstanding job on tackling this challenging problem!
Congratulations!
Best, Erkay Savas"