**Answers**

**1)**

a) To find |Z\*|, I used the value that is sent (386). Then, I found every value that has gcd(x, 386) = 1. Therefore, the result is that value, which is 192.

b) Generator should generate all values of the group Z\*. Therefore, in a part I also generated a set of values of that group. In the b part, I tried possible values of group if they are generating that set. Then, I found the value 5 which is a generator for that group.

c) To find a subgroup generator that has length 64, I used the same method as in part b. However, I found a number that exactly generates 64 elements of the group, which is value 11.

**2)**

In question 2, we had equation d = e-1 mod(phi(n)). Firstly, to find phi(n), I checked if p and q is primes, and they were. Then phi(n) = (p-1)\*(q-1) by theorem. Since now we know phi(n), I found modular inverse of e in mod(phi(n)). Therefore, I found value d. Now, I know c and d, I can find m. However, directly finding cd is hard and time consuming, therefore I used pow function with a modulus value n (p\*q). Then I changed m value to ASCII and found the result:

In another parallel universe, your univesity ID would be: 451 . In case you do not like it, ask Loki to change it for you.

**3)** Question 3 is tricky one. However, since we know nonce of the first text, and key, we can generate new salsa20 for first ciphertext. When we decrypt it, it gives us:

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

In ciphertext2, we know that some of nonce parts are corrupted. Therefore, I tried to add a value to ciphertext in the beginning and do not use its nonce at all. I added Vbq to beginning and it gave me:

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

In ciphertext3, I decided to clear nonce by hand. Iteratively, I deleted parts from beginning. Than, cipher text become this:

b'x14\x88NW\xbfh\xb9\xcdX\x0f\x83}\xc0cX5\xa5\x9e\x1e^\xd0\x03\xc5\x1e\xa3U@\xa1\x85H\xc0'

With correct result:

what is right not what is easy

**4)**

We know the theorem says:

If gcd(a,n) = 1 there is exactly one solution for ax=bmod(n)

And the solution is b \* a-1 mod(n)

If gcd(a,n) = d != 1

If d does not divide b

No solution

Else d many solutions

By using this theorem, I found out that

A has a unique solution which is:

778214478105812676636756719791275483076257993476674633986349

B has no solution

C has 2 solutions (gcd is 2):

[214452804613963486637685100244716357445929679678527437334799, 510140495723613606819499501310772970562507011340109785493345]

D has 4 solutions (gcd is 4)

[896049914695400631894506737412603426006253201283161992061, 18951860339961353869399140376877946054778848148808512468685, 37007670765227307106903774016343288683551443096333862945309, 55063481190493260344408407655808631312324038043859213421933]

**5)**

In fifth question, we look for keystreams’ periods. We generate random seed for LFSRs and use it to generate a keystream with enough length. To check if generates maximum period sequence we look for if equals to 2n-1

Results:

A can generate maximum period since it generated a sequence with 63, which is 26-1

B cannot generate maximum period since it could not generate a sequence with 26-1, instead it generated with period of 14.

C can generate maximum period since it generated a sequence with 31, which is 25-1

**6)**

If current linear complexity of a sequence is bigger than expected linear complexity, we say that it is non-predictable.

With respect to that, I found these results:

x1: Expected: 44.72222222222222 Linear complexity: 45, since Expected < Linear complexity, not predictable

x2: Expected: 43.72222222222222 Linear complexity :29, since expected > linear complexity, predictable

x3: Expected: 51.22222222222222 Linear complexity: 37, since expected > linear complexity, predictable

**7)** About question 7, I learned that reciprocal functions can generate reverse keys from the internet. (Initial knowledge)

We know that the text ends with Atil Utku Ay. With that knowledge we can generate part of keystream with length 84 by xoring it with ciphertext. With that initial key, we can find a connection polynomial using the function BM, which has 27 length.

Then, using that connection polynomial, I generated a new connection polynomial which is reciprocal function of original one to find keystream from reverse.

Then, as a seed I used first 27 bits of the initial keystream and added it to solution key stream. Using that seed and reciprocal function, I generated a keystream.

However, since we are using LFSR and we already knew that we added the last part of the keystream to beginning (some part of Atil Utku Ay), I reversed keystream.

Using reversed keystream, I XORed it with ciphertext. Using XORed result value, I generated the plaintext with bin2ASCII function:

Dear Student,

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

You have just earned 20 points.

Congrats!

Best, Atil Utku Ay