**Başak Amasya – 26628**

**CS411 – HW2 – SOLUTIONS**

**1)** Please refer to the Q1_student.py for the solution.

a) p=823. First, I should construct $Z_{823}$*. I checked whether 823 is prime or not with the given function and saw that it is a prime number. That's why, every positive integer smaller than 823 will be in $Z_{823}$*. To find the generators of this group, I tried every number in the group. Calculated all of their exponential values with respect to modulus 823 and checked whether they constructed $Z_{823}$*, meaning that number should generate all the numbers from 1 to 822. The generators of $Z_{823}$* are [3, 6, 7, 10, 14, 17, 20, 22, 23, 24, 38, 41, 43, 44, 45, 47, 48, 54, 56, 58, 65, 68, 73, 75, 76, 78, 80, 92, 99, 101, 105, 108, 111, 112, 116, 122, 126, 127, 134, 136, 143, 149, 150, 151, 153, 155, 156, 157, 160, 164, 165, 171, 172, 176, 179, 180, 182, 184, 185, 186, 188, 192, 193, 197, 202, 207, 213, 221, 229, 231, 237, 243, 244, 245, 247, 249, 250, 252, 254, 259, 260, 261, 263, 265, 268, 277, 283, 285, 292, 294, 295, 298, 299, 302, 304, 306, 307, 309, 314, 318, 321, 328, 330, 331, 339, 341, 344, 346, 347, 350, 351, 352, 354, 357, 360, 363, 364, 367, 369, 370, 372, 376, 377, 382, 384, 385, 386, 387, 396, 399, 401, 405, 407, 411, 414, 417, 420, 421, 423, 425, 426, 431, 432, 433, 434, 435, 442, 443, 444, 445, 448, 457, 458, 461, 464, 467, 474, 483, 485, 489, 491, 497, 498, 500, 501, 507, 510, 520, 527, 533, 534, 535, 539, 541, 542, 543, 544, 545, 547, 549, 550, 553, 557, 559, 565, 567, 570, 572, 575, 577, 581, 582, 583, 584, 585, 587, 588, 589, 597, 598, 600, 603, 608, 609, 611, 617, 618, 619, 620, 622, 624, 627, 633, 634, 636, 640, 654, 655, 657, 661, 665, 669, 681, 684, 685, 686, 690, 692, 695, 699, 703, 706, 708, 709, 710, 713, 714, 716, 719, 721, 723, 726, 734, 736, 738, 739, 742, 746, 749, 751, 753, 757, 764, 766, 770, 771, 773, 774, 786, 787, 788, 790, 791, 792, 793, 802, 807, 808, 810, 814, 819, 821]. Any of these numbers can be picked as a generator for the answer of the question.

b) t=137. 137 is a prime number as well. I tried every number in $Z_{823}$* to see if that number generates a subgroup whose order is 137. I know that subgroups are cyclic, so if that number is a generator, when the orderth ($137^{th}$) exponentiation of that number is equal to 1 in modulus 823, that number should be a generator of a subgroup whose order is 137. I tried numbers in $Z_{823}$* (except 1) to find the generators. These generators are found as [8, 18, 25, 26, 42, 51, 55, 60, 62, 64, 69, 71, 79, 83, 95, 98, 103, 106, 118, 119, 121, 123, 129, 132, 135, 140, 141, 144, 145, 148, 161, 163, 167, 170, 178, 181, 194, 195, 200, 208, 209, 211, 218, 219, 227, 228, 230, 233, 257, 262, 269, 287, 293, 297, 301, 305, 308, 315, 319, 324, 329, 333, 335, 336, 337, 348, 361, 374, 408, 410, 419, 428, 429, 430, 440, 450, 452, 455, 465, 468, 470, 478, 480, 481, 496, 503, 506, 511, 512, 513, 523, 532, 537, 548, 551, 552, 556, 568, 591, 599, 601, 606, 607, 613, 625, 632, 646, 650, 664, 671, 676, 677, 693, 698, 727, 729, 730, 732, 733, 735, 737, 741, 756, 760, 762, 777, 783, 784, 789, 794, 795, 796, 804, 811, 812, 818]. Any of these numbers can be picked as a generator for the answer of the question. (I also validated my answers by trying these generators and check whether they generate 137 unique numbers in modulus 823 and all of them did.)

**2)** Please refer to the Q2_student.py for the solution.

Since n is a very large number, I used a more practical way to calculate its phi function, than using the phi function given in helper code. n's prime factorization is p*q, so phi of n should be (p-1)*(q-1) from the shortcut given in the lecture slides. Then with the help of given functions, I calculated d, which is inverse of e with respect to phi of n. After that, I calculated $c^d$ in modulus n. It should be noted that I used built-in pow function of Python for this, since it uses a much more efficient algorithm (binary left to

right algorithm) than **. To decode m into Unicode string, I calculated minimum number of bytes necessary to represent m in bytes form and used to_bytes built-in function of Python to convert m into a bytes form. Then I used decode built-in function to convert it to a string form in UTF-8 encoding. m is "Your secret number is 779".

**3)** Please refer to hw2_q3.py for the solution.

I used a function to solve the equation $ax \equiv b \bmod n$, for any a, b and n. This function first checks the gcd of a and n. If these numbers are relatively prime, meaning their gcd is 1, then there is one unique solution for this equation. If that's not the case, whether d divides b or not is checked, by looking at d modulus b. If d divides b, meaning their modulus is 0, then there are exactly d number of solutions for this equation. If not, then there is no solution for this equation. It should be noted that, n is the same number for all 3 equations in this question.

a) This equation does not have a solution. a and n are not relatively prime, their gcd is 5 so it cannot have a unique solution, a-inverse does not exist. Then we need to check if d divides b and we see that there is a remainder which is 3, so we cannot proceed with trying to solve the equation.

b) This equation has exactly 5 solutions. a and n are not relatively prime, their gcd is 5, like in option a. But unlike option a, d divides b in this equation, there is no remainder. That's why, we can calculate these 5 different solutions. We first calculate our new a, b and n by dividing the numbers by d, so that we'll obtain an equation that we can solve. Then we calculate this new a's inverse with respect to new n and by multiplying with new b and taking modulus again, we obtain the unique solution of this new equation which is also one of the solutions of the original equation. Then we add the new modulus, d-1 times to calculate the other solutions. These 5 solutions are:
113687137494180043727898218252158652 18, 308265219325034668011990590718568438 93, 502843301155889292296082963184978225 68, 697421382986743916580175335651388012 43, 891999464817598540864267708117797799 18 .

c) There is a unique solution. a and n are relatively prime, their gcd is 1. So, we directly calculate a inverse in modulus n. Then, multiply with b and take its modulus again to reach the unique solution of this equation which is 759407906151265598556069587953484916 11.

**4)** Please refer to hw2_q4.py for the solution.

L=7, since degree of these polynomials is 7. So, maximum period should be 2^L – 1 = 2^7 – 1 = 127. We need to check if these connection polynomials' periods are 127. The initial state does not matter in this question, so I just generated it randomly. Then with the help of the given functions in the helper code, I found the keystreams generated by these polynomials and calculated the periods of these keystreams. P2 ($x^7 + x + 1$) always gives a period of 127, so we can say that it's a primitive polynomial, it generates the maximum period sequences. However, P1 ($x^7 + x^3 + x^2 + 1$) never generates period of 127, its period might differ in each run giving numbers such as 2, 31 or 62 but never 127. We can say that P1 does not generate maximum period sequences, it's not a primitive polynomial.

**5)** Please refer to hw2_q5.py for the solution.

I used BMA function (given in helper code) to calculate these sequences' linear complexities. Each of the sequences' length is 150. The expected linear complexities should be around half of their length, which

is 75, if they are unpredictable. But each of the sequences' linear complexity was calculated as 37 (37 is also the length of the shortest LFSR's that can be generated by these sequences), which is considerably less than 75, so these sequences have low linear complexity. x1, x2 and x3 are predictable sequences.

It can be also observed that, number of 0's and number of 1's are not equal in these sequences, which supports that they are predictable sequences as well.

**6)** Please refer to hw2_q6.py for the solution.

We know the beginning of the plaintext and ciphertext. I first converted the beginning of the plaintext to binary form and XOR'ed it with the beginning of ciphertext in binary form, in order to obtain the beginning of the key (inverse of the XOR is again XOR operation). With the help of BM algorithm, I obtained the connection polynomial of this key, which is 1+x+x^2+x^5+x^27. After that I needed to find the initial state to obtain the rest of the key as well. I tried to do an exhaustive search on every possible state, but the computations just couldn't finish since there were 2^27 different possibilities so that approach didn't work. Then I realized that I already know the beginning of the key and the length of the shortest LFSR that generates this key which is 27. So, the first part of the key should correspond to the initial state since stream ciphers work in a shifting manner. But we should get it in a reverse order because we append the number to the back because we are using a shift register. By reversing the first 27 bits of the key, I obtained the initial state. Using the initial state and connection polynomial, I obtained the full key. After that, I just XOR'ed the key with the cipher text and obtained the binary form of the plaintext and used the given function to convert it into the ASCII form. Plain text is:

Dear Student,
You have worked hard, I know taht; but it paid off:)
You have just earned 20 bonus points. Congrats!
Best,
Erkay Savas