Q1

{'n': 367, 't': 61}

A) In Z367 There are 366 elements -🡪 [1,2,3,……364,365,366]

B)

For an element to be a generator in Z367 it’s powers (1 to 366)should generate the Z367. To find such elements I created a list of elements 1 to 366 (initial) and iterated over i= 1 to 366 (all possible generators). At all iterations I created another loop (x= 1to 366) to append the value “ i^x mod n “ to check list.After the loop finishes there are 366 elements on checklist and if the checklist equals to initial list that is 1 to 366 this value is a generator. Please check Q1.py file to see the output.

OUTPUT OF Q1

Z367 has generators [6, 10, 11, 12, 17, 19, 20, 22, 34, 39, 42, 43, 48, 54, 58, 65, 69, 70, 71, 76, 77, 78, 79, 80, 88, 90, 93, 96, 97, 99, 103, 108, 111, 115, 116, 119, 123, 127, 130, 131, 133, 136, 139, 140, 143, 150, 152, 153, 154, 155, 157, 159, 160, 165, 171, 172, 176, 179, 180, 183, 185, 186, 193, 194, 197, 198, 201, 205, 206, 218, 219, 221, 223, 238, 239, 241, 246, 249, 250, 254, 255, 257, 263, 265, 267, 269, 272, 273, 275, 276, 278, 282, 285, 294, 300, 301, 305, 306, 307, 310, 312, 314, 316, 317, 326, 330, 331, 334, 335, 336, 337, 339, 341, 344, 349, 351, 353, 354, 363, 365]

C)

For an element in Z367 to be a generator of a subgroup with order 61 elements power to the 61 should be relatively prime to the 367. In other words (element= e ) e^order mod(n) = 1.

In the Q1\_C.py I iterated over all possible elements starting from 2 to 366 and checked if its power of order modulus n is equal to 1. If this is the case I appended the element to a generator list.

THE OUTPUT OF Q1\_C

Generator of the subgroup of order 61: [7, 8, 9, 15, 25, 46, 47, 49, 52, 56, 59, 63, 64, 72, 74, 81, 87, 101, 105, 106, 107, 114, 120, 122, 124, 132, 134, 135, 137, 145, 151, 164, 175, 178, 190, 199, 200, 204, 209, 211, 220, 225, 226, 229, 242, 258, 281, 292, 299, 322, 323, 327, 329, 332, 338, 340, 343, 346, 362, 364]

Q2

e=24167380932807504239964762959050608181052072818924462420440944589853986666535152290678483164713087547939617706544933073937438743674849799104159491240839348260324246460988250940447812376328941082868235473291224963034913948314545930105300659271488671935139477857113987097236226884629242409351548871694620398772197336903444812437569224804275890088559668152413817783852890241047035413304048206793334836912450607705141780806375259618885462560707010734585871217215939271094068309518364230203157486297286544871904254305379453493937519141682453326416659514227521354356779202232509796359880023499074722911112885933118651425829

c=21194027877221209252532944554115296298068550273026811097724631984955956461694198755220230294168005128470003106327250916246670574858779239543516027440141559406664276723402769767498014210688563103182668963229853712170892259971587732776722679519976771734885031418486068674400808900963734636871151823761474113661814991520069244818528455745426962838850855604369765397024497931688768425376697288667268860274316895709676926571337544100556805069889255058627417615809131539051280113984608934633401822372246989424850341142597153022251648989642057062549894044882277755239829946583573285048270025093875316034571444410295888222961

For given p,q,e,c values:

To compute m = c ^d mod n where d =e^-1 mod φ(n) .

We first need to find φ(n) and n = p x q.

p and q are primes, thus φ(n) is (p-1)\*(q-1).

gcd of e and φ(n) is 1, thus e^-1 exists in modulus φ(n).

I used modinv funtion to find “d”. Then pow(c,d,n) function (which calculates c^d mod n) is used to

find m. Then I decoded the bit sequence into Unicode string, the output is. (Please check Q2.py)

THE OUTPUT OF Q2.py:

The gcd of e,phi\_n 1

M is 2881341690647539318037325450499657826808932916300128772987740947324424292399968451717740717339921500559562128659280959

I think I have 556 unread e-mails. Is that a lot?

Q4

The steps are as follows:

1. Check gcd(a,n):

If gcd is 1 there is 1 solution.

If gcd is bigger than one, Then

1. Check if gcd divides b

If true: There are number of gcd solutions.

a’= a/gcd, b’= b/gcd, c’= c/gcd

Find modular inverse of (a’,n’) and

X= (b’\*inv)%n’

The other solutions are: [ X + (n’) , X + (2n’)….. , X+ (gcd-1)n’]

If false: There is no solution.

Please check Q4.py for the outputs.

a)

GCD1 is 1

B1 MOD GCD1 is 0

There is a solution which is 1115636343148004398322135138661008357945126147114770093414826

b)

GCD2 is 2

B2 MOD GCD2 is 1

gcd2 does not divide b2---> There is no solution

c)

GCD3 is 2

B3 MOD GCD3 is 0

BECAUSE GCD3 DIVIDES B3 THERE ARE 2 SOLUTIONS

1840451085636978827079830514312022149966941191143010614385900

4573017168579321153146925263568627765759266852067838063135011

d)

GCD4 is 4

B4 MOD GCD4 is 0

BECAUSE GCD4 DIVIDES B4 THERE ARE 4 SOLUTIONS

120574576795431477647425259344685590574672051332591719355582

1692041454071987051397898895041599936536312796085130907016143

3263508331348542625148372530738514282497953540837670094676704

4834975208625098198898846166435428628459594285590209282337265

Q5

In order the connection polynomials to be maximum period, when we run the lfsr algorithm they have to generate the inital state at (2^order -1) steps. Please check Q5.py for the outputs.

p1 (x) = x 7 + x 5 + x 3 + x + 1

Initial state = 0,0,0,0,0,1 ->[1, 0, 0, 0, 0, 0, 0]->[1, 1, 0, 0, 0, 0, 0]->[1, 1, 1, 0, 0, 0, 0]………… [0, 0, 0, 1,0,1, 0]->[0, 0, 0, 0, 1, 0, 1]->[0, 0, 0, 0, 0, 1, 0]->[0, 0, 0, 0, 0, 0, 1] (126 iterations.)

THE PERIOD IS 127. 2^7-1 = 128-1= 127 which is the maximum period that 7th degree polynomial offers.

p2 (x) = x 6 + x 5 + x 2 + 1

Initial state: [0, 0, 0, 0, 0, 1]->

[1, 0, 0, 0, 0, 0]->[0, 1, 0, 0, 0, 0]->[1, 0, 1, 0, 0, 0]->[0, 1, 0, 1, 0, 0]->[1, 0, 1, 0, 1, 0]->[1, 1, 0, 1, 0, 1]->

[0, 1, 1, 0, 1, 0]->[0, 0, 1, 1, 0, 1]->[1, 0, 0, 1, 1, 0]->[1, 1, 0, 0, 1, 1]->[1, 1, 1, 0, 0, 1]->[0, 1, 1, 1, 0, 0]->

[1, 0, 1, 1, 1, 0]->[1, 1, 0, 1, 1, 1]->[1, 1, 1, 0, 1, 1]->[1, 1, 1, 1, 0, 1]->[0, 1, 1, 1, 1, 0]->[0, 0, 1, 1, 1, 1]->

[0, 0, 0, 1, 1, 1]->[0, 0, 0, 0, 1, 1]->[0, 0, 0, 0, 0, 1]

THE PERIOD IS 21 which is not the maximum period = 2^6-1 = 63.

p3 (x) = x 5 + x 4 + x 3 + x + 1

Initial state: [0, 0, 0, 0, 1]->[1, 0, 0, 0, 0] -> [1, 1, 0, 0, 0] -> [1, 1, 1, 0, 0] -> [0, 1, 1, 1, 0] -> [0, 0, 1, 1, 1] -> [1, 0, 0, 1, 1] -> [1, 1, 0, 0, 1] -> [0, 1, 1, 0, 0] -> [1, 0, 1, 1, 0] -> [1, 1, 0, 1, 1] -> [1, 1, 1, 0, 1] -> [1, 1, 1, 1, 0] -> [1, 1, 1, 1, 1] -> [0, 1, 1, 1, 1] -> [1, 0, 1, 1, 1] -> [0, 1, 0, 1, 1] -> [0, 0, 1, 0, 1] -> [0, 0, 0, 1, 0] -> [1, 0, 0, 0, 1] -> [0, 1, 0, 0, 0] -> [0, 0, 1, 0, 0] -> [1, 0, 0, 1, 0] -> [0, 1, 0, 0, 1] -> [1, 0, 1, 0, 0] -> [0, 1, 0, 1, 0] -> [1, 0, 1, 0, 1] -> [1, 1, 0, 1, 0] -> [0, 1, 1, 0, 1] -> [0, 0, 1, 1, 0] -> [0, 0, 0, 1, 1] -> [0, 0, 0, 0, 1]

THE PERIOD IS 31 2^5-1 = 32-1= 31 which is the maximum period that 5th degree polynomial offers.

P1 and P3 generates maxiumum period of sequences, P2 does not.

Q6

When we run BMA algorithm on these sequences the output is:

Length of x1 75

L1 and C1(x): (36, [1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1])

Length of x2 80

L2 and C2(x): (43, [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1])

Length of x3 90

L3 and C3(x): (31, [1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1])

Their linear complexity is 36,43,31 at the same time their lengths are 75,80,90. Linear complexity is

not as long as sequences’ length. That shows that there is a pattern that makes this sequences

predictable. Thus, these sequences are predictable.