EE793: Home Assignment

Topics in Cryptology

April 23, 2023

DEPARTMENT OF ELECTRICAL ENGINEERING
IIT BOMBAY

Analysis of Linear Complexity Distribution of Functions

Instructor: Prof Virendra Sule

GROUP MEMBERS
Bhavya Kohli, 20D070021
Amruta Mahendra Parulekar, 20D070009
Sanjhi Priya, 20D070070
Priyanshi Gupta, 200070061
Pulkit Paliwal, 20D100021
Bhavya Singh, 200040036

Contents

1 Theory

We are given a map F mapping $\mathbb{F}_n^2$ to itself and a point y in the range of the map giving an equation $y = F(x)$. Since y is in the range of the map F this equation has a solution x. The assignment problem is aimed to study the sequence called the recurrence sequence and denoted and defined as

$$S(\zeta, x) = \{y, F(y), F^{(2)}(y), ..., F^{(M-1)}(y)\}$$

Because of the finiteness of the field the sequence repeats previous values after a certain number of terms. The term $F^{(k+1)}(y) = F(F^{(k)}(y))$ and $F^{(2)}(y) = F(F(y))$ is the double composition evaluated at y. Since the sequence $S(F,y)$ repeats, there exist two numbers r ≥ 0 and N called as the pre-period and period respectively such that

$$F^{(j+r+N)}(y) = F^{j+r}(y), \text{ for j = 0, 1, 2,...} \tag{1}$$

The sequence is called periodic of period N if r = 0 and N is the smallest such number in the above equation. The periodicity equation $F^{(N)}$(y) = y is associated with a polynomial X^N - 1. The recurrence relation [1] shows that

$$(X^N - 1)(F^{(j)}(y)) = 0 \text{ for j = 0, 1, 2,...}$$

Hence for a periodic sequence there possibly exists a smallest degree polynomial

$$m(X) = X^m - \sum_{i=0}^{m-1} \alpha_i X^i$$

such that the sequence satisfies the *linear recurrence relation* $m(X)(F^{(j)}(\text{y})) = 0$ which is the same as

$$F^{(m+j)}(y) = \sum_{i=0}^{m-1} \alpha_i F^{(j+i)}(y) \tag{2}$$

Such a minimal polynomial is unique and its degree is called the *Linear Complexity* (LC) of the sequence $S(F,y)$. In this assignment, the objective is to create the data of densities of linear complexities of maps F.

The Berlekamp Massey Algorithm:
The problem to be solved is to find the LC and the minimal polynomial of a vector sequence

$$\hat{y} = \{y_0, y_1, y_2, ..., y_{(M-1)}\}$$

using the BM algorithm, where M is any one of 2n, $2n^2$, $2n^3$. Here, moreover each y_i is a vector

$$y_i = \{y_{i1}, y_{i2}, y_{i3}, ..., y_{in}\}^T$$

The BM function takes only the scalar sequence of field numbers

$$\hat{s} = \{s_0, s_1, s_2, ..., s_{(M-1)}\}$$

as input and computes the minimal polynomial $m_{\hat{s}}(X)$ of the sequence. The degree m of the minimal polynomial is the LC of the sequence.

2 Problem 1: LC distribution of the exponential map in prime fields

2.1 Problem Statement

Given that p is a prime number and ζ is a primitive element in $\mathbb{F}_p$, the exponential map $F(\mathrm{x}) : \mathbb{F}_p \to \mathbb{F}_p$ is defined as:

$$y = F(\mathrm{x}) = \zeta^x \bmod \mathrm{p}$$

The aim of this experiment is to gather the data on distribution of LC of recursively generated sequences $S(F,y,p)$ by the exponential map in the group of units of prime fields F_p.

2.2 Methodology

1. First, we chose a prime p of bit length n between 16 to 18 bits.
2. Then, we fixed a primitive element ζ of $\mathbb{F}_p$, which was generated randomly.
3. For a sample of 1000 random x in $\mathbb{F}_p$, we computed $y = F(x)$, after which we generated the sample of recursive sequences

   $$S(\zeta, x) = \{y, F(y), F^{(2)}(y), ..., F^{(M-1)}(y)\}$$

   in $\mathbb{F}_p$ for $M = O(n^2)$.
4. These sequences were zero-padded to make each of them 18 bit long.
5. Next, we computed the Linear Complexity of $S(\zeta,x)$ using the Berlekamp Massey algorithm over $\mathbb{F}_p$.
6. For this, the minimum polynomial was computed for the first bit of each sequence and then subsequently for the remaining bits.
7. At every iteration, the new minimum polynomial was taken as the LCM of the previous and current minimum polynomial.
8. The linear complexity is the degree of the final minimum polynomial.
9. Then, we drew the histogram of LC values.
10. Finally, repeated this procedure for 5 different primes p.

2.3 Code

```
import numpy as np
from sage.matrix.berlekamp_massey import berlekamp_massey as bm
import matplotlib.pyplot as plt
from tqdm import tqdm
#Importing the required libraries

def allprimroots(p, lim=None):
    if lim is None: lim = p-1
    k = []
    for i in (range(2, lim)):
        if (GF(p)(i).multiplicative_order() == p-1):
            k.append(i)
    return k
# This function finds all elements in  prime fields which have order p-1

def zero_pad(x,target,pre='pre'):
    if pre=='pre':
        return [0] * (target-len(x)) + x
    elif pre=='post':
        return x + [0] * (target-len(x))
# This function zero pads the sequences generated to be of length 18

def generate_sequence(zeta,x,M):
    # c == y => first term
    c = zeta ** x
    Sval = [c]
    c_bits = ZZ(c).bits()
    c_bits = zero_pad(c_bits,1,'post')
    Sbits = [c_bits]
    found,f = False,None
    for i in range(M):
        c = zeta ** c
        if c == Sval[0]:
            found = True
            f = Sval[-2]
        Sval.append(c)
        c_ = ZZ(c).bits()
        c_ = zero_pad(c_,1,'post')
        Sbits.append(c_)
    return Sval, Sbits, [found,f]
# This function generates recursive sequence y, F(y), F(F(y))..and so on

randints = np.random.randint(2**16,2**18,size=5)
p_ = [next_prime(i) for i in randints]
# Creating a list of 5 random elements between 16 to 18 bits
LCs_main = []
founds = []
for p in p_:
    ff = GF(p)
    # Converting each element to the prime field
    prim_elements = allprimroots(p,200)
    zeta = ff(np.random.choice(prim_elements))
    # Getting a random primitive element of the prime field ( zeta )
```

```
    map_ = lambda zeta,x: zeta**x
    # Defining the exponential map
    k = 4; l = ceil(log(p,2)); M = k*l*l
    # M is of order n ^2
    xrange = np.random.randint(p,size=1000)
    # Generating 1000 random values the range 0 to p -1
    LCs = []
    num_found = 0
    for x in tqdm(xrange,ncols=75):
        x_ = ff(x)
        # Converting each value into the prime field
        Sval, Sbits, [found,inv] = generate_sequence(zeta,x_,M-1)
        # Sbits is the bit representation of values in Sval
        S = np.array(Sbits).transpose()
        S_0 = [GF(2)(i) for i in S[0]]
        # Converting the first bit of each value in Sval to F2
        minpoly = bm(S_0)
        # Calculating the min polynomial for S_o using bm algo
        for i in range(1,len(S)):
            S_i = [GF(2)(i) for i in S[i]]
            # Converting the remaining bits of each value in Sval to F2
            minpoly_i = bm(S_i)
            # Calculating the min polynomial for all S_i using bm algo
            minpoly = lcm(minpoly, minpoly_i)
            # Taking running lcm of the previous and current minpolys
        LC = minpoly.degree()
        # LC is the degree of the minimum polynomial
        LCs.append(LC)
        num_found += int(found)
    LCs_main.append(LCs)
    # Creating the final LC list for plotting
    founds.append(num_found)

i = 0
print(zeta)
for n,lc_ in enumerate(LCs_main):
    lc = [i for i in lc_ if i > 2000]
    plt.figure(figsize = (10,5))
    plt.title(r"Histogram for for LC > 2000 for $p = {}$ ".format({p_[n]})
    , fontsize = 15)
    plt.xlabel(r"Linear Complexity of $F_{}$".format({p_[n]}),fontsize=15)
    plt.ylabel("Frequency",  fontsize = 15)
    plt.hist(lc,bins=50,edgecolor='black',color='skyblue')
    plt.show()
    lc = [i for i in lc_ if i < 2000]
    plt.figure(figsize = (10,5))
    plt.title(r"Histogram for for LC < 2000 for $p = {}$ ".format({p_[n]})
    , fontsize = 15)
    plt.xlabel(r"Linear Complexity of $F_{}$".format({p_[n]}),fontsize=15)
    plt.ylabel("Frequency", fontsize = 15)
    plt.hist(lc,bins=50,edgecolor='black',color='skyblue')
    plt.show()
    i = i + 1
#Plotting code
```

2.4 Plots

The following plots illustrate the distribution of Linear Complexities for different values of p. Due to many high and low LC values but no intermediate values being present, the histograms are depicted in two separate splits.

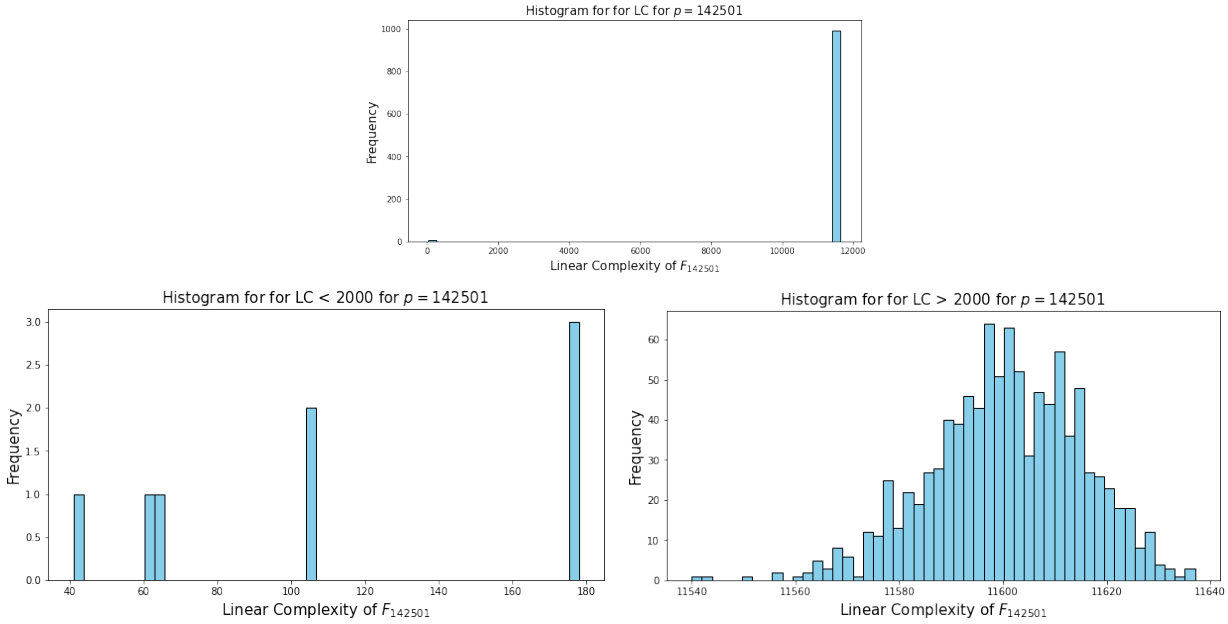

Figure 1: Histogram for p=142501

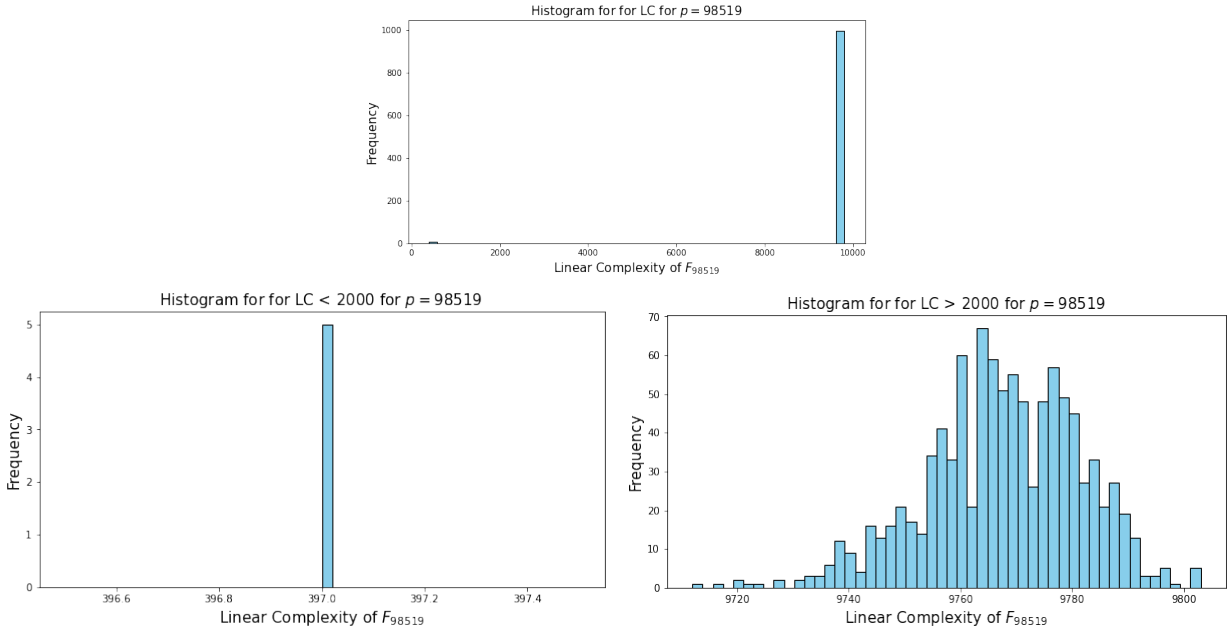

Figure 2: Histogram for p=98519

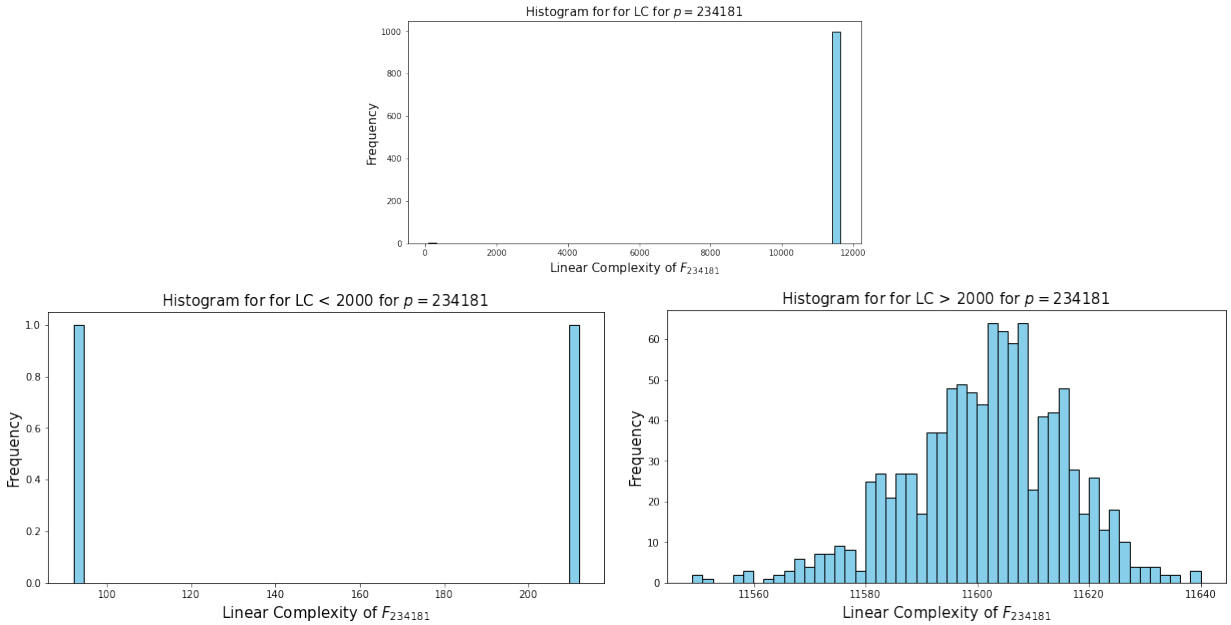

Figure 3: Histogram for p=234181

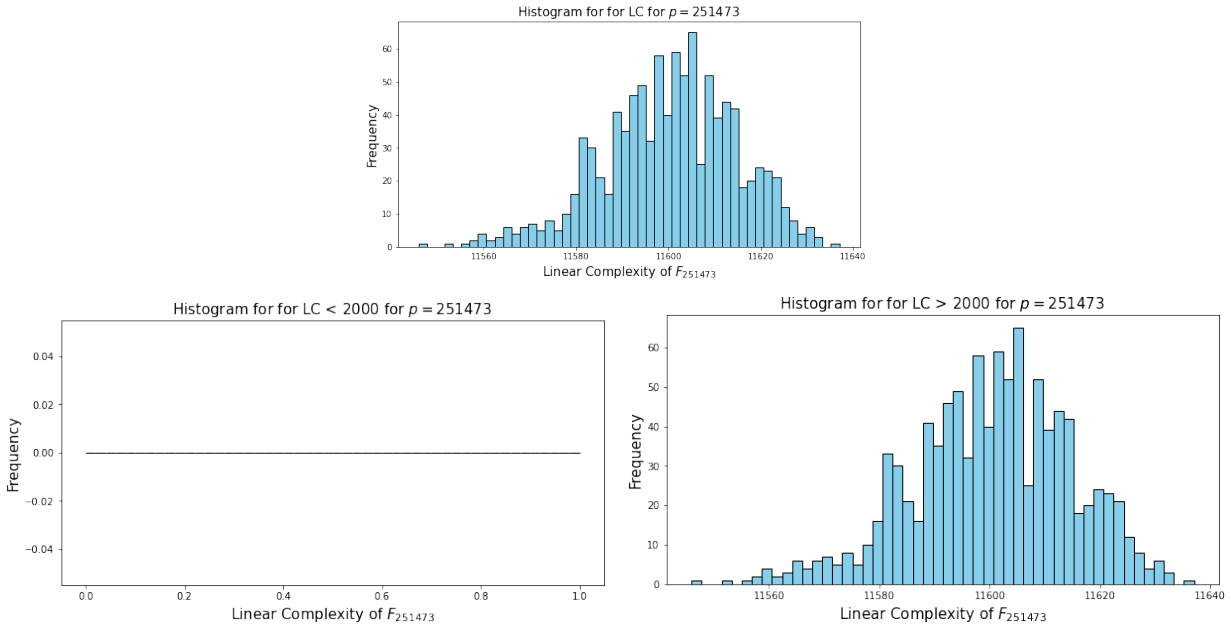

Figure 4: Histogram for p=251473

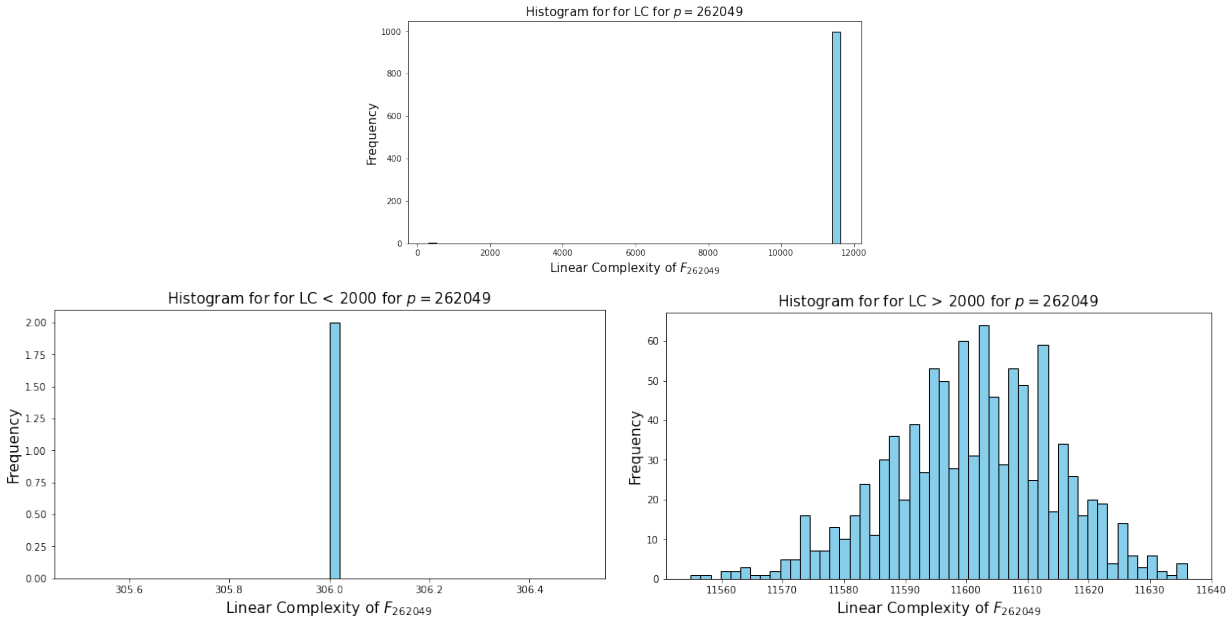

Figure 5: Histogram for p=262049

2.5 Observations and Conclusions

- The Linear Complexities are mostly concentrated at values > 9000 and < 600 for the primes $\{142501, 98519, 234181, 251473, 262049\}$
- The frequencies of LCs for $LC < 600$ may be higher or lower than the frequencies of LCs for $LC > 9000$ for the primes $\${142501, 98519, 234181, 251473, 262049\}$. However, the frequencies in the former range are sparsely distributed while those in the latter r
- There do not exist any LCs in the range $[600 : 9000]$
- Over different values of primes p, the LC of the sequence has never gone above 12000. Till the value of p = 142501, we notice that the LC's are concentrated near approximately $p/10$

Table 1: Inversion accuracies for the different p values

p value	**Inversion accuracy (%)**
142501	1
98519	1.6
234181	1.9
251473	0
262049	0.8

3 Problem 2: Distribution of LC of quadratic residue function

3.1 Problem Statement

Given distinct odd primes p,q and $n = pq$, the quadratic residue function $F(\mathrm{x})$ is defined as:

$$y = F(\mathrm{x}) = x^2 \bmod \mathrm{N}$$

where x is said to be a square root y modulo N.
The aim of this experiment is to gather the data on distribution of LC of recursively generated sequences $S(F,y_0)$ by the quadratic residue function. It is known that the problem of computing square root mod N is equivalent to factoring N. Hence inversion of the map locally is computationally equivalent to factoring.

3.2 Methodology

1. First, we chose a pair of distinct primes p and q of length 18 bits such that N has length 36 bits.

2. For a sample of 1000 random x_0 in [0,N - 1] we computed $y_0 = F(x_0)$. For these y_0, we computed the sequences

   $$S(F, y_0) = \{y, F(y_0), F^{(2)}(y_0), ..., F^{(M-1)}(y_0)\}$$

   for $M = O(n^2)$.

3. Next, we represented each $S(F,y_0)$ as a sequence of n-bit vectors in $\mathbb{F}_n^2$.

4. These sequences were zero-padded to make each of them the same length (equal to the bit length of N).

5. Next, we computed the Linear Complexity of $S(\zeta,x)$ using the Berlekamp Massey algorithm.

6. For this, the minimum polynomial was computed for the first bit of each sequence and then subsequently for the remaining bits.

7. At every iteration, the new minimum polynomial was taken as the LCM of the previous and current minimum polynomial.

8. The linear complexity is the degree of the final minimum polynomial.

9. Then, we drew the histogram of LC values.

10. Finally, we repeated the above procedure and collected histogram data for five different pairs of primes p,q such that bit length of N was between 16 to 18 bits.

3.3 Code

```
def zero_pad(x,target,pre='pre'):
    if pre=='pre':
        return [0] * (target-len(x)) + x
    elif pre=='post':
        return x + [0] * (target-len(x))
# This function zero pads the sequences to be of the bit length of N
def generate_sequence(x,M):
    c = mod(x**2,N)
    Sval = [c]
    c_bits = ZZ(c).bits() #
    c_bits = zero_pad(c_bits,l,'post') #
    Sbits = [c_bits]
    found, f = False, None
    for i in range(M):
        c = mod(c**2, N)    #f(x)
        if c == Sval[0]:
            found = True
            f = Sval[-2]
        Sval.append(c)
        c_ = ZZ(c).bits()
        c_ = zero_pad(c_,l,'post')
        Sbits.append(c_)
    return Sval, Sbits, [found, f]
# This function generates recursive sequence y, F(y), F(F(y)).. so on

k = 5
l = ceil(log(N,2))
M = k*l*l
xrange = np.random.randint(N,size=1000)
# Generating 1000 random values the range 0 to p -1
LCs = []
num_found = 0
# here, we took only first 50 samples, as execution time was very high
for x_ in tqdm(xrange[:50],ncols=75):
    Sval, Sbits, [found, inv] = generate_sequence(x_,M)
    # Sbits is the bit representation of values in Sval
    S = np.array(Sbits).transpose()
    S_0 = [GF(2)(i) for i in S[0]]
    # Converting the first bit of each value in Sval to F2
    minpoly = bm(S_0)
    # Calculating the min polynomial for S_o using bm algo
    for i in range(1,len(S)):
        S_i = [GF(2)(i) for i in S[i]]
        # Converting the remaining bits of each value in Sval to F2
        minpoly_i = bm(S_i)
        # Calculating the min polynomial for all S_i using bm algo
        minpoly = lcm(minpoly, minpoly_i)
        # Taking running lcm of the previous and current minpolys
    LC = minpoly.degree()
    LCs.append(LC)
    num_found += int(found)
# at this stage, the list ‘LCs‘ contains 50 entries, one for each value of
    x, and ‘num_found‘ contains the number of correct inverses found
```

3.4 Plots

Following plots illustrate the distribution of Linear Complexities for different p and q values.

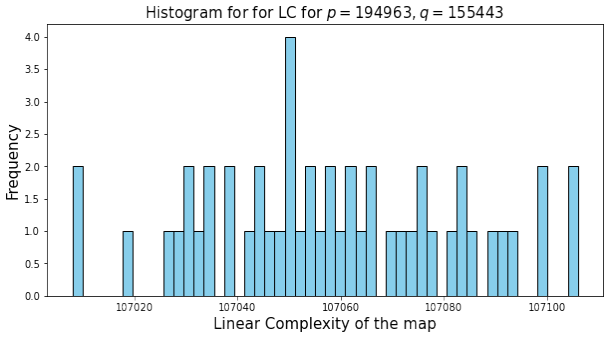

Figure 6: Histogram for p=194963,q=155443

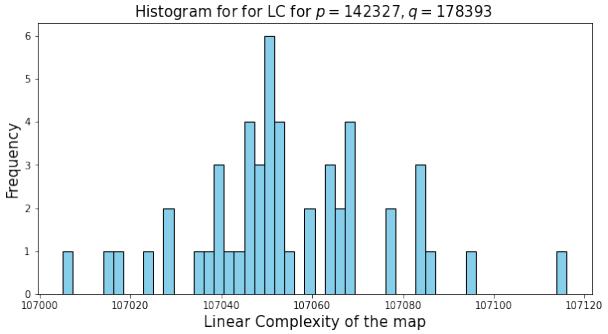

Figure 7: Histogram for p=142327,q=178393

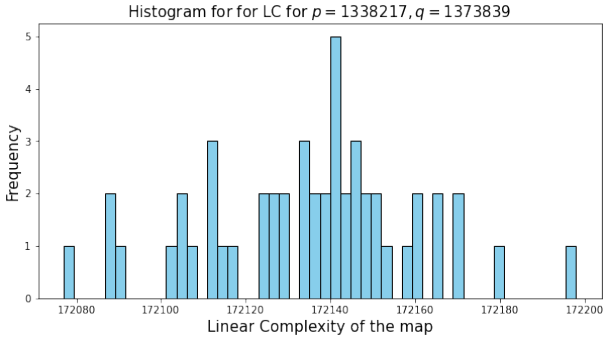

Figure 8: Histogram for p=1338217,q=1373839

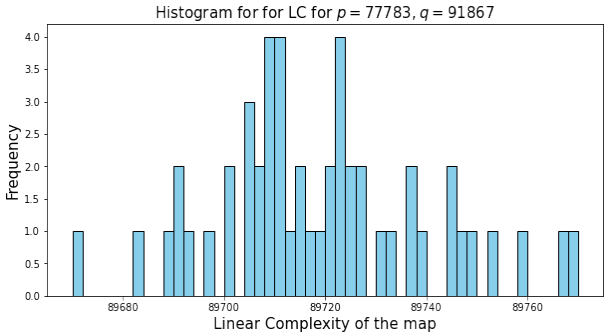

Figure 9: Histogram for p=77783,q=91876

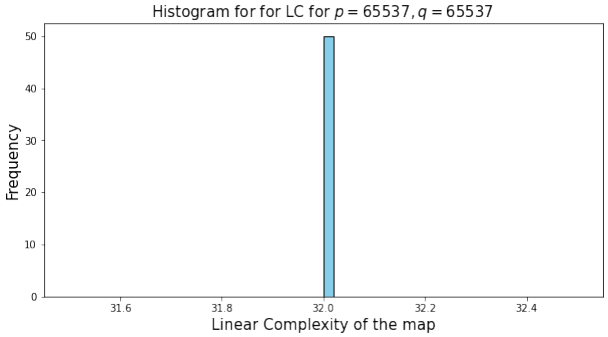

Figure 10: Histogram for p=65537,q=65537

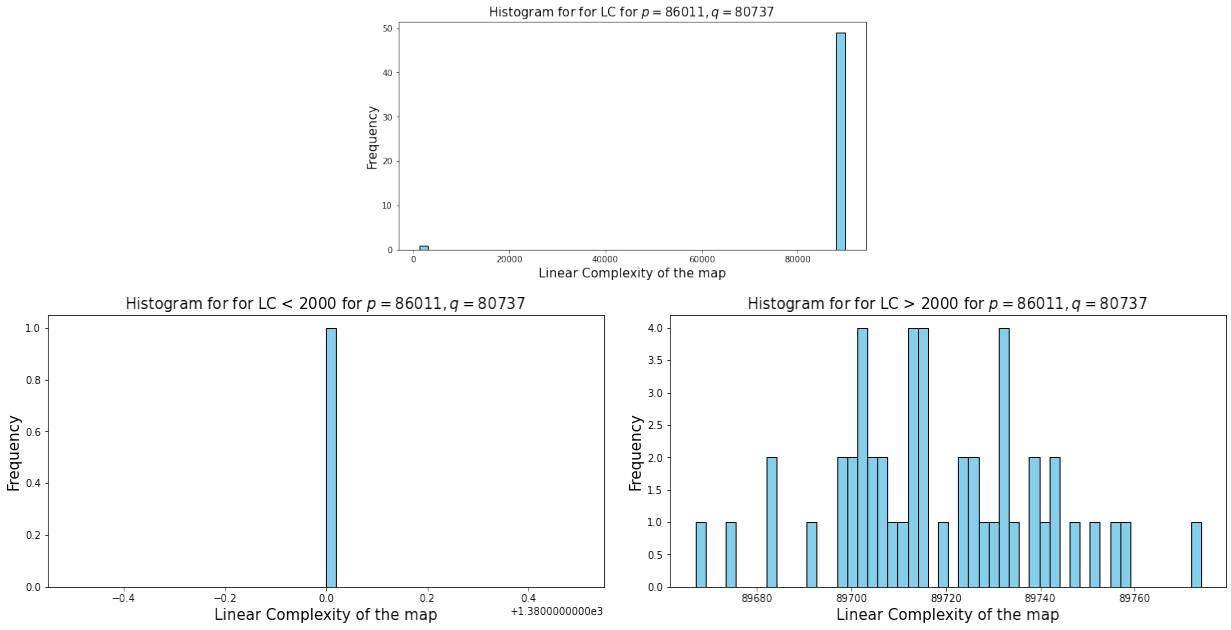

Figure 11: Histogram for p=86011,q=80737

3.5 Observations and Conclusions

- The inversion accuracy for this problem was much worse when compared to the previous one, and it was rare to get even one correct inversion for all the different values of x.
- The average LC for different values of p and q is around 100-120k. No clear relation between p,q,N and LC was observed.
- When p and q are close together, the LC values are almost all equal, with little to no spread. This can be seen in the case above, with p = q = 65537.