Fifth Homework Cybersecurity

Andrea Morelli 1845525

1 First Ideas

To solve the homework i started documenting myself on the Shamir secret sharing technique.To split a secret in multiple shards we have to create a polynomial of degree k-1, where k is the minimum number of shards to reconstruct the secret, and insert in the coefficient of degree 0 the secret that we want to split.All the other coefficients of greater degree can be selected at random.Once we create a polynomial with those coefficients we have to calculate the value of the polynomial in n point x to get the y and we have to store the tuples (x,y) that will become our shards.In order to reconstruct the secret we base ourselves on the theorem that states that a polynomial of degree k-1 is uniquely identified by k point and if we have k points we can reconstruct it for example using the Lagrange interpolation.

2 Create shards

The first part of the algorithm so to create the shards was pretty straightforward.I generated k coefficients and the first is overwritten with s.After that i create x points at random and all differents and i evaluate the function in all of those points,storing x and y in the list of shards that will be returned

```
def createShards(k, n, s):
    g=0
    coefficients =[r.randint(0, maxRandomCoefficients) for i in range(k)]
    coefficients[0] = s
    print("those are the coefficients from 0 to k-1")
    print(coefficients)
    shards = []
    for i in range(n):
        y = 0
        l=[i[0] for i in shards]
        x=0
        while(True):
            x = r.randint(0, maxRandomCoefficients)
            if x not in l:
                break
        for j in range(k):
            y += (x**j)*coefficients[j]
        shards.append((x, y))
    print("shards are generated")
    return shards
```

3 Reconstruct the secret

The part of reconstructing the secret was instead more difficult to solve.The first idea that i had was to create a linear system of equations with coefficients as incognitas where xi are the and yi are the shards.

$$\begin{cases} x_1^0 k_0 + x_1^1 k_1 + \ldots + x_1^n k_n = y_1 \\ \vdots \\ x_n^0 k_0 + x_n^1 k_1 + \cdots + x_n^n k_m = ym \end{cases} \tag{1}$$

The problem with this approach was that with computers i had to use numpy to solve the system and having big numbers numpy had to convert everything to float and the results where not stable for big numbers.Internally numpy solves it inverting the matrix A and multiplying it by b. The second idea was instead inspired by the lagrange interpolation algorithm:

$$P(x) = \sum_{i=0}^{n} f(a_i) \prod_{j \neq i, j=0}^{n} \frac{x - a_j}{a_i - a_j} \tag{2}$$

The problem with this approach was instead that to fully implement it would have been difficult taking into account all the multiplications of literals in the formula. Focusing only on the elements with degree 0 is instead easy to implement. To implement it i just focused on the literals with degree 0 and i treated them only without taking into account the others using the formula:

$$f(0) = \sum_{j=0}^{k-1} y_j \prod_{\substack{m=0 \\ m \neq j}}^{k-1} \frac{x_m}{x_m - x_j} \tag{3}$$

to get only the the coefficient with degree 0 i used this formula that has not the problems described above.To implement it the first thing that i did was documenting myself on arbitrary precision types in python because i needed a good precision in the divisions in the formula otherwise the secret resulting from the operations could be wrong. I used the object Decimal to have a desired precision with the method getcontext().prec allowing us to declare a precision for the decimal numbers.I used those objects to deal with the precision in the division of literals. The script i wrote to implement the formula and decrypt the secret from k shards is:

```
def findSecret(shards, k):
    if len(shards) < k:
        print("you need more shards")
        return -1
    secret = Decimal(0)
    for i in range(k):
        li = Decimal(1)
        xi = shards[i][0]
        yi = shards[i][1]
        for j in range(k):
            xj = shards[j][0]
            if i == j:
                continue
            li *= -Decimal(Decimal(xj)/(xi-xj))
        secret += Decimal(yi*li)
    return round(Decimal(secret))
```

I cicle 2 times the first k shards ,if are available otherwise i return -1, to firstly compute the division between xj and xi-xj if xi!=xj and i multiply this newly found value to the corresponding yi to get the secret that is the sum of all this values as in the formula above.I used the decimals everytime with a getcontex().prec=10**4 to have a good approximation of the secret and to be sure that the one i return is the correct one.In the end i round the secret to get an integer and i return it.

4 Drawing the polynomial

To draw the polynomial i used the library matplotlib.

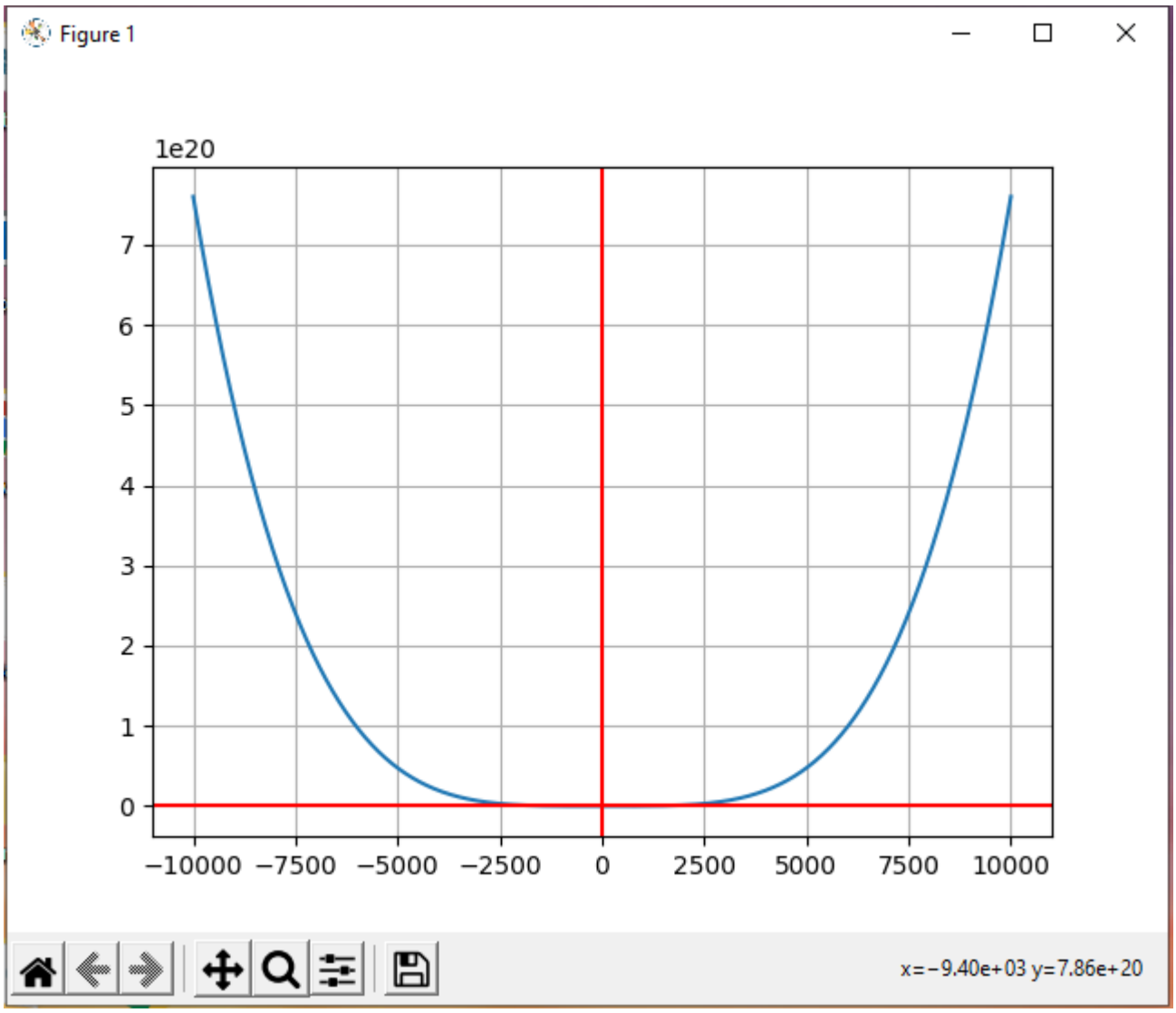
```
def drawFromCoefficients(coefficients):
    size = 10000
    x = np.linspace(-size, size,10000)
    y=[]
    for i in range(len(x)):
        sum=0
        for j in range(len(coefficients)):
            sum+=(x[i]**j)*coefficients[j]
        y.append(sum)
    plt.plot(x, y)
    plt.grid()
    plt.axhline(color='r')
    plt.axvline( color='r')
    plt.show()
```

I used this to evaluate the points of the polynomial in the x in the range -size,+size to obtain the y values and i plotted them with a grid and highlighting the axes x=0 and y=0. The code uses the coefficients to draw a polynomial so i used this function in the first phase when i generated the polynomial. To draw the polynomial from the generated shards i instead used the function fit to construct a polynomial from all the points that were given (shards) and then once constructed the polynomial with his coefficient i used the function above to draw it.The function to fit a polynomial from the shard and then draw it is below:

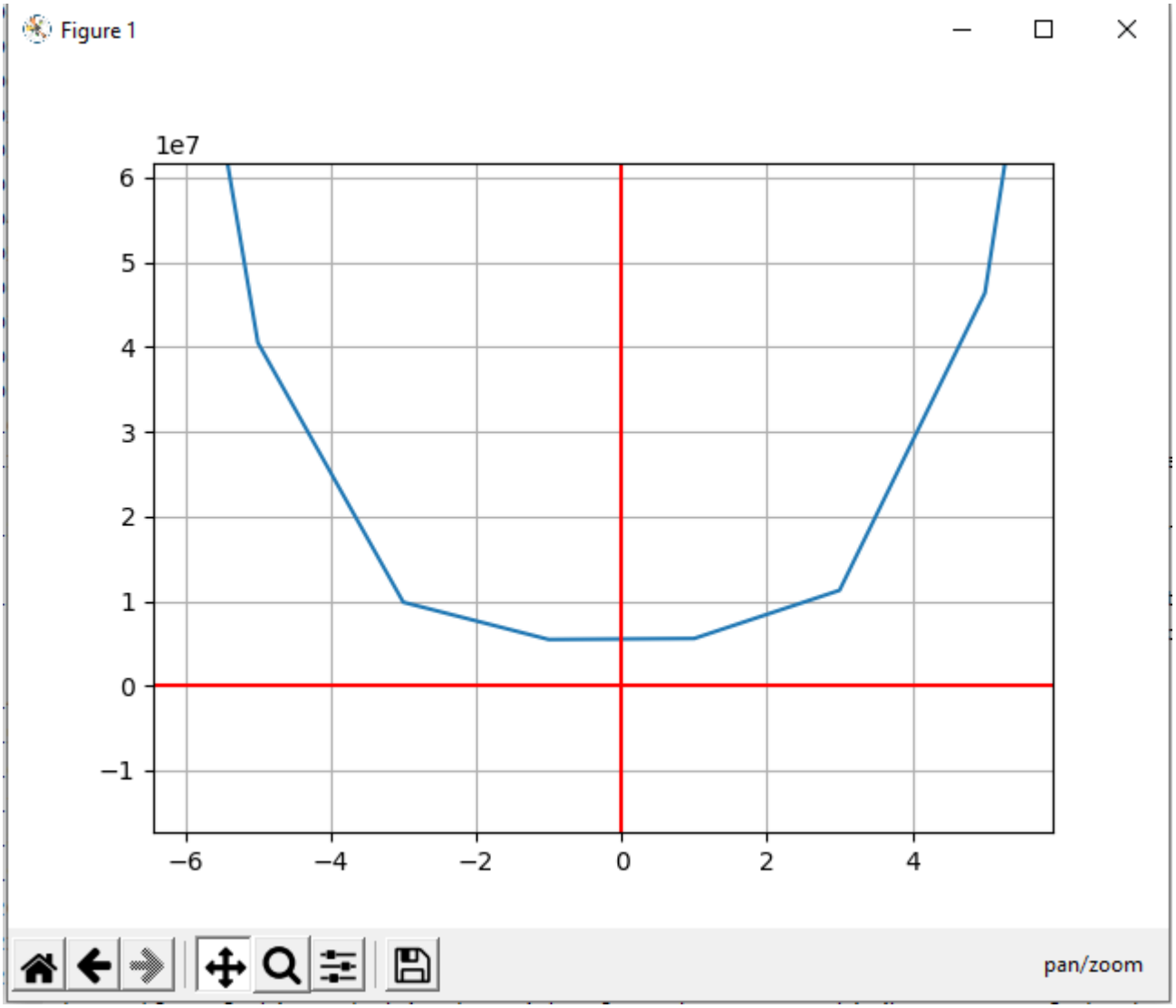
```
def drawFromShards(shards,k):
    x=np.array([i[0]for i in shards ],dtype="float64")
    y=np.array([i[1]for i in shards],dtype="float64")

    coefficients=np.polynomial.Polynomial.fit(x, y, k)
    coefficients=[i.round(0) for i in coefficients]
    drawFromCoefficients(coefficients)
```

The problem of this method is that with values that are too big(k,n,range of the domain of coefficients) the function resulting may be different from the original one and that is because numpy has to cast all the values to float64 to fit a polynomial to the points so the only reliable function that can draw the polynomial is the first one used once the coefficients are generated. Another idea i had was to sort all the shards and then print the shards directly but in this way the secret that is stored on x=0 may not be seen if there is no shard with x=0 so i didn't implement this idea. To draw the complete polynomial from the shards i should have implemented the complete lagrange interpolation,taking into account also literals of degree ¿0.

5 Final drawing results



That is a random polynomial drawn



If we zoom closely we can see the secret stored

6 Final script source code

```
import random as r
import numpy as np
from decimal import *
from matplotlib import pyplot as plt
from numpy.polynomial.polynomial import Polynomial
getcontext().prec = 10**4
maxRandomCoefficients = 10**5


def drawFromCoefficients(coefficients):
    size = 10000
    x = np.linspace(-size, size,10000)
    y=[]
    for i in range(len(x)):
        sum=0
        for j in range(len(coefficients)):
            sum+=(x[i]**j)*coefficients[j]
```

```
        y.append(sum)
    plt.plot(x, y)
    plt.grid()
    plt.axhline(color='r')
    plt.axvline( color='r')
    plt.show()


def drawFromShards(shards,k):
    x=np.array([i[0]for i in shards ],dtype="float64")
    y=np.array([i[1]for i in shards],dtype="float64")

    coefficients=np.polynomial.Polynomial.fit(x, y, k)
    coefficients=[i.round(0) for i in coefficients]
    drawFromCoefficients(coefficients)


def createShards(k, n, s, draw):
    g=0
    coefficients =[r.randint(0, maxRandomCoefficients) for i in range(k)]
    coefficients[0] = s
    print("those are the coefficients from 0 to k-1")
    print(coefficients)
    shards = []
    for i in range(n):
        y = 0
        l=[i[0] for i in shards]
        x=0
        while(True):
            x = r.randint(0, maxRandomCoefficients)
            if x not in l:
                break
        for j in range(k):
            y += (x**j)*coefficients[j]
        shards.append((x, y))
    print("shards are generated")
    if draw == True:
        drawFromCoefficients(coefficients)
    return shards


def findSecret(shards, k):
    if len(shards) < k:
        print("you need more shards")
        return -1
    secret = Decimal(0)
    for i in range(k):
        li = Decimal(1)
        xi = shards[i][0]
        yi = shards[i][1]
        for j in range(k):
            xj = shards[j][0]
            if i == j:
                continue
            li *= -Decimal(Decimal(xj)/(xi-xj))
        secret += Decimal(yi*li)
```

```
    return round(Decimal(secret))


if __name__ == "__main__":
    k = 5
    n = 10
    s = 5423789
    draw = True
    shards = createShards(k, n, s, draw)
    print(findSecret(shards, k))
    drawFromShards(shards,k)
```