HW 1 Nicholas O'Brien 02.21

Process.

We had written a python program that implemented the ElGamal cryptosystem (via $\mathbb{Z}_p$). The first objective was to decrypt the provided ciphertext. We were initially given a private key so the decryption part was pretty straight forward then we just converted the numbers back to letters. The reader should also take note that we used the fact that in $\mathbb{Z}_p$ and element's (say γ) inverse is simply just γ^{p-2}, this is much more efficient than just using the Extended Euclidian algorithm. The next step was to implement Shanks' Algorithm. Again, as we had the private key we had something to verify our implementation . This is because the private key would be the solution to equation $\log_\alpha(\beta)$. Once we had that, we found the k values for each encrypted block. They were verified by then encrypting the plaintext with the found k value and comparing with given ciphertext. The k values were discovered by using Shanks' Algorithm to solve $log_\alpha(y_1)$.

Source Code.

```
import sys
import math
from random import SystemRandom
from data import ct_pairs


class ElGamal():
    a = None # Private Key set to null initially

    def __init__(self, p, alpha, beta):
        self.p = p
        self.alpha = alpha
        self.beta = beta
        self.a = None

    def setPrivateKey(self, a):
        self.a = a

    def decrypt(self, y_1, y_2):
        if self.a is None:
            print("You must add the private key to decrypt.")
            return None
        new_y_1 = self.raiseByExponent(y_1, self.a)
        inverse_y_1 = self.modInverse(new_y_1)
        return self.multiply(y_2, inverse_y_1)

    def encrypt(self, x, k=None):
        if k is None:
            rand = SystemRandom()
            k = rand.randrange(3,  self.p-1) # range used to be safe
        y_1 = self.raiseByExponent(self.alpha, k)
        beta_exp = self.raiseByExponent(self.beta, k)
        y_2 = self.multiply(beta_exp, x)
        return (y_1, y_2)
```

```
# Will find log_alpha(b) with Shanks' Algorithm
def discreteLog(self, b):
    m = math.ceil(math.sqrt(self.p))
    L_1 = []
    for j in range(m):
        L_1.append((j, self.raiseByExponent(self.alpha, m*j)))
    L_2 = []
    alpha_inv = self.modInverse(self.alpha)
    for i in range(m):
        alpha_neg_i = self.raiseByExponent(alpha_inv, i)
        L_2.append((i, self.multiply(alpha_neg_i, b)))
    L_1.sort(key=lambda x: x[1])
    L_2.sort(key=lambda x: x[1])
    for first in L_1:
        for second in L_2:
            if(first[1] == second[1]):
                j = first[0]
                i = second[0]
                return (m*j + i) % self.p
    return None

def modInverse(self, element):
    return self.raiseByExponent(element, self.p-2)

def raiseByExponent(self, element, exp):
    if exp is 1:
        return element
    gArray = []
    gArray.append(element)
    r = self.multiply(element, element)
    gArray.append(r)

    for i in range(2, exp.bit_length()):
        r = self.multiply(r, r)
        gArray.append(r)

    lowestBit = 0
    for i in range(exp.bit_length()):
        if ((exp & (1 << i)) != 0):
            lowestBit = i
            break
    r = gArray[lowestBit]
    lowestBit += 1

    for i in range(lowestBit, exp.bit_length()):
        if ((exp & (1 << i)) != 0):
            r = self.multiply(r, gArray[i])
    return r

def multiply(self, e_0, e_1):
    return (e_0 * e_1) % self.p
```

```
def numbersToText(numbers):
    L_1 = numbers % 26
    numbers -= L_1
    numbers //= 26
    L_2 = numbers % 26
    numbers -= L_2
    numbers //= 26
    L_3 = numbers % 26
    char_1 = chr(L_1 + 97)
    char_2 = chr(L_2 + 97)
    char_3 = chr(L_3 + 97)
    return char_3 + char_2 + char_1


if __name__ == '__main__':
    if (len(sys.argv) is not 5 and len(sys.argv) is not 4):
        print("python ElGamal.py <p> <alpha> <beta> [a]")
        sys.exit(1)

    p = int(sys.argv[1])
    alpha = int(sys.argv[2])
    beta = int(sys.argv[3])
    cryptosystem = ElGamal(p, alpha, beta)


    if(len(sys.argv) is 5):
        a = int(sys.argv[4])
    else:
        print("Private Key not given, trying discrete logarithm...")
        a = cryptosystem.discreteLog(beta)

    cryptosystem.setPrivateKey(a)


    for ct in ct_pairs:
        y1 = ct[0]
        y2 = ct[1]
        pt = cryptosystem.decrypt(y1, y2)
        k = cryptosystem.discreteLog(y1)
        print((y1, y2), "D=> ", end="")
        print("(" +numbersToText(pt)+ "," +str(k)+ ")", end="")
        t1, t2 = cryptosystem.encrypt(pt, k)
        print(" C=>", (t1, t2))
```

Plaintext.

she stands up in the garden where she has been working and looks into the distance she has sensed a change in the weather there is another gust of wind a buckle of noise in the air and the tall cypresses sway she turns and moves uphill towards the house climbing over a low wall feeling the first drops of rain on her bare arms she crosses the loggia and quickly enters the house

K Values.

All k values are linked with their plaintext. Below is the collections of these tuples of the form (pt_i, k_i).

(she,29705)	(sta,28841)	(eis,19834)	(ano,28127)	(cli,23470)	(mbi,14625)
(nds,18076)	(upi,21011)	(the,23493)	(rgu,23276)	(ngo,15068)	(ver,12242)
(nth,478)	(ega,1576)	(sto,19873)	(fwi,24538)	(alo,26877)	(wwa,31153)
(rde,20710)	(nwh,29302)	(nda,20081)	(buc,3376)	(llf,12058)	(eel,27970)
(ere,29115)	(she,29705)	(kle,163)	(ofn,17942)	(ing,20418)	(the,23493)
(has,2500)	(bee,7195)	(ois,31668)	(ein,15635)	(fir,23027)	(std,338)
(nwo,11446)	(rki,23119)	(the,23493)	(air,25742)	(rop,21252)	(sof,6083)
(nga,17240)	(ndl,25197)	(and,30716)	(the,23493)	(rai,25580)	(non,10847)
(ook,31536)	(sin,22194)	(tal,3644)	(lcy,8353)	(her,13296)	(bar,6197)
(tot,18381)	(hed,21976)	(pre,3442)	(sse,5312)	(ear,23944)	(mss,4567)
(ist,3815)	(anc,23155)	(ssw,9456)	(ays,7017)	(hec,12372)	(ros,11920)
(esh,22519)	(eha,11024)	(het,30925)	(urn,19382)	(ses,27020)	(the,23493)
(sse,5312)	(nse,17622)	(san,10312)	(dmo,23389)	(log,11002)	(gia,16408)
(dac,16220)	(han,15385)	(ves,17771)	(uph,15738)	(and,30716)	(qui,3977)
(gei,25967)	(nth,478)	(ill,10919)	(tow,8290)	(ckl,25042)	(yen,3430)
(ewe,17891)	(ath,7036)	(ard,17272)	(sth,13868)	(ter,7305)	(sth,13868)
(ert,15019)	(her,13296)	(eho,4766)	(use,15963)	(eho,4766)	(use,15963)

□