ECE 520 Exercise 3

**Date:** 4/11/20, Saturday, week 11

**Due Date:** 4/23/20 Thursday, week 13

**Total Points:**  45 points

Type the solutions in word document and submit to es-egr-03=>EGR520001=>Submittal folder => Exercise 3. Type your C / C++ / C# / Java / Verilog programming solutions in the corresponding file format and paste your output at the end of your programs..

## Analysis Questions

1. (10%) RSA: Private key computation for small e and n.
   1. For the public key (e, n) = (13, 2881), compute the decipher key d in the RSA algorithm.

d = 853 (see source code)

* 1. Does the decipher key d exist if (e, n) = (7, 2881)? Why?

d does not exist because the gcd(e,phi(n)) must equal 1 to be valid, in this case, phi(n) = phi(2881) = 2772, and the gcd(e,phi(n)) = gcd(7,2772) = 7 which is not 1, so they are not relatively prime to each other so the decryption key cannot be generated.

1. (10%) Repeat RSA question for bigger n

For the public key (e, n) = (13, 35209), compute the decipher key d in the RSA algorithm or explain why it does not exist.

d = 16,069 (see source code)

### (10%) Compute Euler’s totient function (n) for n = 1997, 1999, 2001, 2002, 2003, 2004, and 2005.

(See Source Code)

phi(1997) = 1996

phi(1999) = 1998

phi(2001) = 1232

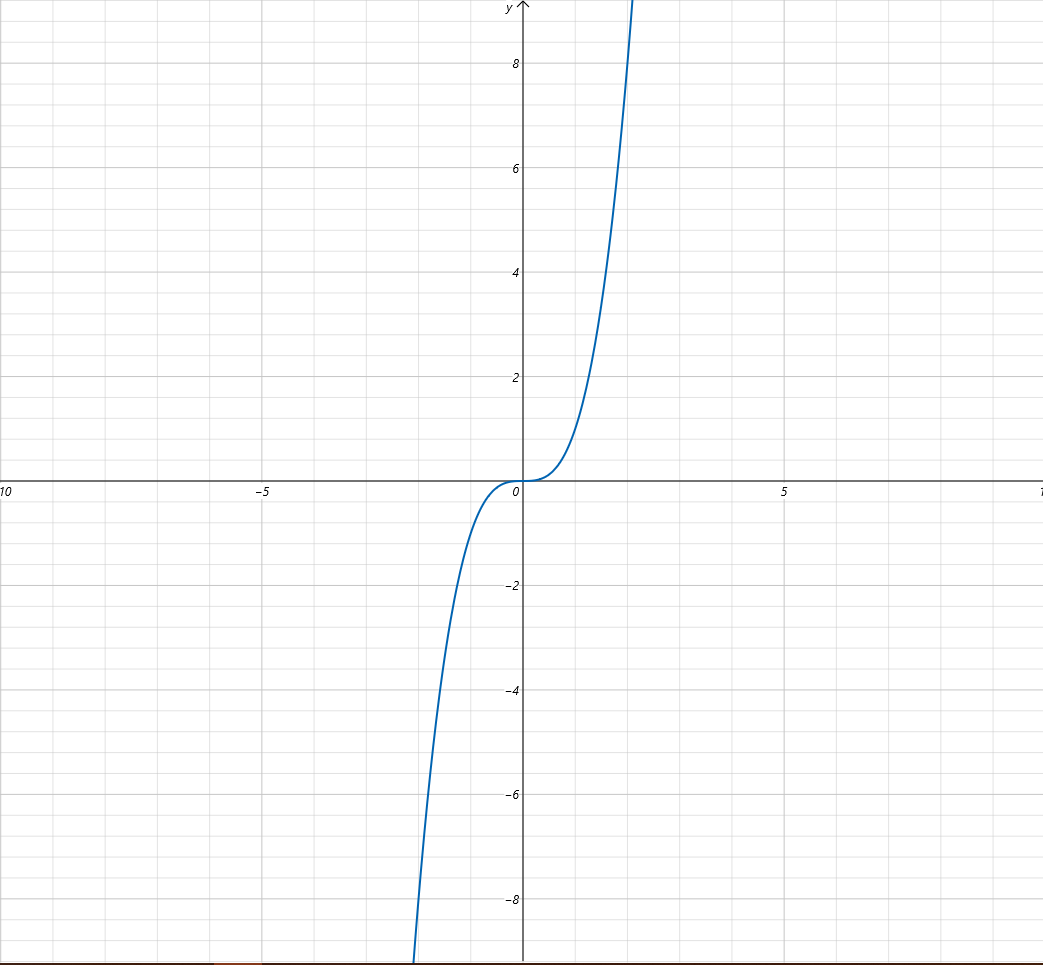
phi(2002) = 720

phi(2003) = 2002

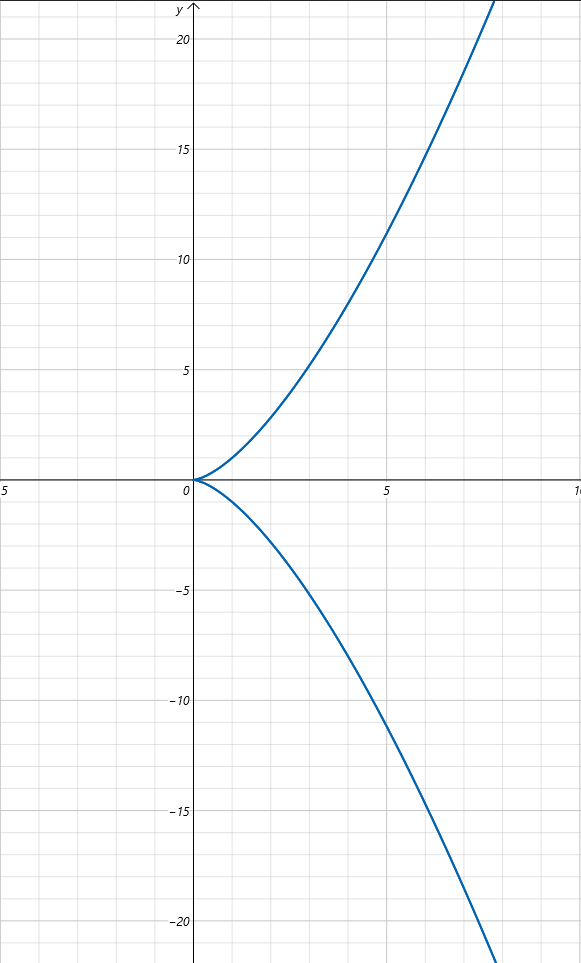
phi(2004) = 664

phi(2005) = 1600

1. (15%) Elliptic curves over Zp.
   1. Sketch the curve y = x3 (for the sketch questions, do it electronically if you can. Otherwise just skip that). Note that y = x3 is a function used very often that you can possibly get from the internet.



* 1. Sketch the curve y2 = x3. (this is the simplest elliptic curve with a = b = 0 from equation (10.1) on page 310).



* 1. Find all the nonzero integer solutions modulo 11 of y2 = x3 (we do *not* consider (y, x) = (0, 0) as a solution, but (y, x) = (1, 1) is one. Is there any other? Can you find a quick way?).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x or y | y\*y | y\*y mod 11 | x\*x\*x | x\*x\*x mod 11 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 4 | 4 | 8 | 8 |
| 3 | 9 | 9 | 27 | 5 |
| 4 | 16 | 5 | 64 | 9 |
| 5 | 25 | 3 | 125 | 4 |
| 6 | 36 | 3 | 216 | 7 |
| 7 | 49 | 5 | 343 | 2 |
| 8 | 64 | 9 | 512 | 6 |
| 9 | 81 | 4 | 729 | 3 |
| 10 | 100 | 1 | 1000 | 10 |

Points at which y^2 = x^3 mod 11:

|  |  |
| --- | --- |
| 1: (1,1) (1,10) | 6: None |
| 2: None | 7: None |
| 3: (9,5) (9,6) | 8: None |
| 4: (5,9) (5,2) | 9: (4,3) (4,8) |
| 5: (3,7) (3,4) | 10: None |

* 1. Find all the nonzero integer solutions modulo 17 of y2 = x3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| y or x | y\*y | y\*y mod 17 | x\*x\*x | x\*x\*x mod 17 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 4 | 4 | 8 | 8 |
| 3 | 9 | 9 | 27 | 10 |
| 4 | 16 | 16 | 64 | 13 |
| 5 | 25 | 8 | 125 | 6 |
| 6 | 36 | 2 | 216 | 12 |
| 7 | 49 | 15 | 343 | 3 |
| 8 | 64 | 13 | 512 | 2 |
| 9 | 81 | 13 | 729 | 15 |
| 10 | 100 | 15 | 1000 | 14 |
| 11 | 121 | 2 | 1331 | 5 |
| 12 | 144 | 8 | 1728 | 11 |
| 13 | 169 | 16 | 2197 | 4 |
| 14 | 196 | 9 | 2744 | 7 |
| 15 | 225 | 4 | 3375 | 9 |
| 16 | 256 | 1 | 4096 | 16 |

Points at which y^2 = x^3 mod 17:

|  |  |
| --- | --- |
| 1: (1,1) (1,16) | 9: (15,14) (15,3) |
| 2: (8,11) (8,6) | 10: None |
| 3: None | 11: None |
| 4: (13,2) (13,15) | 12: None |
| 5: None | 13: (4,8) (4,9) |
| 6: None | 14: None |
| 7: None | 15: (9,7) (9,10) |
| 8: (2,5) (2,12) | 16: (16,13) (16,4) |

Extra Credit

E5. (15%) For the (e, n) = (13, 2881) in #11 here, use RSA algorithm to encode the first two lines of any of the 5 Shakespeare plays you had done in exercise 1. You may use any integer mapping for your characters such as A = 00, B = 01, etc. or use A = 65, B = 66, a = 97, b = 98, .. etc. The later way may be preferable since it encodes lower case as well as upper case. You may group 2 characters in a block, 3 characters in a block, or let every single character be in a block alone.

Original Text: To be, or not to be- that is the question: Whether 'tis nobler in the mind to suffer The slings and arrows of outrageous fortune Or to take arms against a sea of troubles, And by opposing end them. To die- to sleep- No more; and by a sleep to say we end The heartache, and the thousand natural shocks That flesh is heir to. 'Tis a consummation Devoutly to be wish'd. To die- to sleep. To sleep- perchance to dream: ay, there's the rub!

Encoded Text: [84, 111, 32, 98, 101, 44, 32, 111, 114, 32, 110, 111, 116, 32, 116, 111, 32, 98, 101, 45, 32, 116, 104, 97, 116, 32, 105, 115, 32, 116, 104, 101, 32, 113, 117, 101, 115, 116, 105, 111, 110, 58, 32, 87, 104, 101, 116, 104, 101, 114, 32, 39, 116, 105, 115, 32, 110, 111, 98, 108, 101, 114, 32, 105, 110, 32, 116, 104, 101, 32, 109, 105, 110, 100, 32, 116, 111, 32, 115, 117, 102, 102, 101, 114, 32, 84, 104, 101, 32, 115, 108, 105, 110, 103, 115, 32, 97, 110, 100, 32, 97, 114, 114, 111, 119, 115, 32, 111, 102, 32, 111, 117, 116, 114, 97, 103, 101, 111, 117, 115, 32, 102, 111, 114, 116, 117, 110, 101, 32, 79, 114, 32, 116, 111, 32, 116, 97, 107, 101, 32, 97, 114, 109, 115, 32, 97, 103, 97, 105, 110, 115, 116, 32, 97, 32, 115, 101, 97, 32, 111, 102, 32, 116, 114, 111, 117, 98, 108, 101, 115, 44, 32, 65, 110, 100, 32, 98, 121, 32, 111, 112, 112, 111, 115, 105, 110, 103, 32, 101, 110, 100, 32, 116, 104, 101, 109, 46, 32, 84, 111, 32, 100, 105, 101, 45, 32, 116, 111, 32, 115, 108, 101, 101, 112, 45, 32, 78, 111, 32, 109, 111, 114, 101, 59, 32, 97, 110, 100, 32, 98, 121, 32, 97, 32, 115, 108, 101, 101, 112, 32, 116, 111, 32, 115, 97, 121, 32, 119, 101, 32, 101, 110, 100, 32, 84, 104, 101, 32, 104, 101, 97, 114, 116, 97, 99, 104, 101, 44, 32, 97, 110, 100, 32, 116, 104, 101, 32, 116, 104, 111, 117, 115, 97, 110, 100, 32, 110, 97, 116, 117, 114, 97, 108, 32, 115, 104, 111, 99, 107, 115, 32, 84, 104, 97, 116, 32, 102, 108, 101, 115, 104, 32, 105, 115, 32, 104, 101, 105, 114, 32, 116, 111, 46, 32, 39, 84, 105, 115, 32, 97, 32, 99, 111, 110, 115, 117, 109, 109, 97, 116, 105, 111, 110, 32, 68, 101, 118, 111, 117, 116, 108, 121, 32, 116, 111, 32, 98, 101, 32, 119, 105, 115, 104, 39, 100, 46, 32, 84, 111, 32, 100, 105, 101, 45, 32, 116, 111, 32, 115, 108, 101, 101, 112, 46, 32, 84, 111, 32, 115, 108, 101, 101, 112, 45, 32, 112, 101, 114, 99, 104, 97, 110, 99, 101, 32, 116, 111, 32, 100, 114, 101, 97, 109, 58, 32, 97, 121, 44, 32, 116, 104, 101, 114, 101, 39, 115, 32, 116, 104, 101, 32, 114, 117, 98, 33]

Encrypted Encoded Text: [408, 1476, 168, 1293, 912, 2280, 168, 1476, 1668, 168, 496, 1476, 284, 168, 284, 1476, 168, 1293, 912, 2129, 168, 284, 975, 1638, 284, 168, 708, 921, 168, 284, 975, 912, 168, 567, 1803, 912, 921, 284, 708, 1476, 496, 2331, 168, 2151, 975, 912, 284, 975, 912, 1668, 168, 2569, 284, 708, 921, 168, 496, 1476, 1293, 2324, 912, 1668, 168, 708, 496, 168, 284, 975, 912, 168, 1787, 708, 496, 2304, 168, 284, 1476, 168, 921, 1803, 1239, 1239, 912, 1668, 168, 408, 975, 912, 168, 921, 2324, 708, 496, 2593, 921, 168, 1638, 496, 2304, 168, 1638, 1668, 1668, 1476, 177, 921, 168, 1476, 1239, 168, 1476, 1803, 284, 1668, 1638, 2593, 912, 1476, 1803, 921, 168, 1239, 1476, 1668, 284, 1803, 496, 912, 168, 2444, 1668, 168, 284, 1476, 168, 284, 1638, 729, 912, 168, 1638, 1668, 1787, 921, 168, 1638, 2593, 1638, 708, 496, 921, 284, 168, 1638, 168, 921, 912, 1638, 168, 1476, 1239, 168, 284, 1668, 1476, 1803, 1293, 2324, 912, 921, 2280, 168, 518, 496, 2304, 168, 1293, 747, 168, 1476, 1191, 1191, 1476, 921, 708, 496, 2593, 168, 912, 496, 2304, 168, 284, 975, 912, 1787, 98, 168, 408, 1476, 168, 2304, 708, 912, 2129, 168, 284, 1476, 168, 921, 2324, 912, 912, 1191, 2129, 168, 2424, 1476, 168, 1787, 1476, 1668, 912, 2744, 168, 1638, 496, 2304, 168, 1293, 747, 168, 1638, 168, 921, 2324, 912, 912, 1191, 168, 284, 1476, 168, 921, 1638, 747, 168, 177, 912, 168, 912, 496, 2304, 168, 408, 975, 912, 168, 975, 912, 1638, 1668, 284, 1638, 1307, 975, 912, 2280, 168, 1638, 496, 2304, 168, 284, 975, 912, 168, 284, 975, 1476, 1803, 921, 1638, 496, 2304, 168, 496, 1638, 284, 1803, 1668, 1638, 2324, 168, 921, 975, 1476, 1307, 729, 921, 168, 408, 975, 1638, 284, 168, 1239, 2324, 912, 921, 975, 168, 708, 921, 168, 975, 912, 708, 1668, 168, 284, 1476, 98, 168, 2569, 408, 708, 921, 168, 1638, 168, 1307, 1476, 496, 921, 1803, 1787, 1787, 1638, 284, 708, 1476, 496, 168, 2078, 912, 1286, 1476, 1803, 284, 2324, 747, 168, 284, 1476, 168, 1293, 912, 168, 177, 708, 921, 975, 2569, 2304, 98, 168, 408, 1476, 168, 2304, 708, 912, 2129, 168, 284, 1476, 168, 921, 2324, 912, 912, 1191, 98, 168, 408, 1476, 168, 921, 2324, 912, 912, 1191, 2129, 168, 1191, 912, 1668, 1307, 975, 1638, 496, 1307, 912, 168, 284, 1476, 168, 2304, 1668, 912, 1638, 1787, 2331, 168, 1638, 747, 2280, 168, 284, 975, 912, 1668, 912, 2569, 921, 168, 284, 975, 912, 168, 1668, 1803, 1293, 1768]

Decrypted Encoded Text: [84, 111, 32, 98, 101, 44, 32, 111, 114, 32, 110, 111, 116, 32, 116, 111, 32, 98, 101, 45, 32, 116, 104, 97, 116, 32, 105, 115, 32, 116, 104, 101, 32, 113, 117, 101, 115, 116, 105, 111, 110, 58, 32, 87, 104, 101, 116, 104, 101, 114, 32, 39, 116, 105, 115, 32, 110, 111, 98, 108, 101, 114, 32, 105, 110, 32, 116, 104, 101, 32, 109, 105, 110, 100, 32, 116, 111, 32, 115, 117, 102, 102, 101, 114, 32, 84, 104, 101, 32, 115, 108, 105, 110, 103, 115, 32, 97, 110, 100, 32, 97, 114, 114, 111, 119, 115, 32, 111, 102, 32, 111, 117, 116, 114, 97, 103, 101, 111, 117, 115, 32, 102, 111, 114, 116, 117, 110, 101, 32, 79, 114, 32, 116, 111, 32, 116, 97, 107, 101, 32, 97, 114, 109, 115, 32, 97, 103, 97, 105, 110, 115, 116, 32, 97, 32, 115, 101, 97, 32, 111, 102, 32, 116, 114, 111, 117, 98, 108, 101, 115, 44, 32, 65, 110, 100, 32, 98, 121, 32, 111, 112, 112, 111, 115, 105, 110, 103, 32, 101, 110, 100, 32, 116, 104, 101, 109, 46, 32, 84, 111, 32, 100, 105, 101, 45, 32, 116, 111, 32, 115, 108, 101, 101, 112, 45, 32, 78, 111, 32, 109, 111, 114, 101, 59, 32, 97, 110, 100, 32, 98, 121, 32, 97, 32, 115, 108, 101, 101, 112, 32, 116, 111, 32, 115, 97, 121, 32, 119, 101, 32, 101, 110, 100, 32, 84, 104, 101, 32, 104, 101, 97, 114, 116, 97, 99, 104, 101, 44, 32, 97, 110, 100, 32, 116, 104, 101, 32, 116, 104, 111, 117, 115, 97, 110, 100, 32, 110, 97, 116, 117, 114, 97, 108, 32, 115, 104, 111, 99, 107, 115, 32, 84, 104, 97, 116, 32, 102, 108, 101, 115, 104, 32, 105, 115, 32, 104, 101, 105, 114, 32, 116, 111, 46, 32, 39, 84, 105, 115, 32, 97, 32, 99, 111, 110, 115, 117, 109, 109, 97, 116, 105, 111, 110, 32, 68, 101, 118, 111, 117, 116, 108, 121, 32, 116, 111, 32, 98, 101, 32, 119, 105, 115, 104, 39, 100, 46, 32, 84, 111, 32, 100, 105, 101, 45, 32, 116, 111, 32, 115, 108, 101, 101, 112, 46, 32, 84, 111, 32, 115, 108, 101, 101, 112, 45, 32, 112, 101, 114, 99, 104, 97, 110, 99, 101, 32, 116, 111, 32, 100, 114, 101, 97, 109, 58, 32, 97, 121, 44, 32, 116, 104, 101, 114, 101, 39, 115, 32, 116, 104, 101, 32, 114, 117, 98, 33]

Decrypted Decoded Text: To be, or not to be- that is the question: Whether 'tis nobler in the mind to suffer The slings and arrows of outrageous fortune Or to take arms against a sea of troubles, And by opposing end them. To die- to sleep- No more; and by a sleep to say we end The heartache, and the thousand natural shocks That flesh is heir to. 'Tis a consummation Devoutly to be wish'd. To die- to sleep. To sleep- perchance to dream: ay, there's the rub!

Appendix (Code)

File:



def euclidAlgo(larger, smaller):

if(smaller > larger):

temp = larger

larger = smaller

smaller = temp

rem = larger % smaller

if(rem == 0):

return smaller

return euclidAlgo(smaller, rem)

def extEuclidAlgo(a, b):

a, b = abs(a), abs(b)

x0, x1, y0, y1 = 1, 0, 0, 1

while b != 0:

q, a, b = a // b, b, a % b

x0, x1 = x1, x0 - q \* x1

y0, y1 = y1, y0 - q \* y1

return a, x0, y0

def generatePublic(p,q):

n = p\*q

phi = (p-1)\*(q-1)

e = 2

while(euclidAlgo(e,phi) != 1):

e = e+1

if(e >= phi):

return( "not possible since e < phi(n), phi(n)={0} and e = {1}" .format(phi,e))

return e

def generatePrivate(e,n):

phi = eulerPhi(n)

gcd = euclidAlgo(e,phi)

if(gcd == 1):

d = extEuclidAlgo(e,phi)[1]

else:

return ("not possible since the gcd of {0} and {1} is {2} and not 1" .format(e,phi,gcd))

return d

def primeFactors(number):

factors = []

for i in range(2, number + 1):

if(number % i == 0):

prime = True

for j in range(2, (i//2 + 1)):

if(i % j == 0):

prime = False

break

if (prime):

factors.append(i)

return factors

def eulerPhi(n):

factors = primeFactors(n)

phi = n

for i in range (0, len(factors)):

temp = 1 - (1/factors[i])

phi \*= temp

return int(phi)

def encryptRSA(x,e,n):

return pow(x,e,n)

def decryptRSA(y,d,n):

return pow(y,d,n)

def paarExample():

print("Paar Example, p=3, q=11")

p = 3

q = 11

e = generatePublic(p,q)

d = generatePrivate(e,p\*q)

print("phi: {0}" .format(eulerPhi(p\*q)))

print("encryption key: {0}" .format(e))

print("decryption key: {0}" .format(d))

def e3p11a():

print("\nExercise 3, Problem 11a")

e = 13

n = 2881

d = generatePrivate(e,n)

print("decryption key: {0}" .format(d))

def e3p11b():

print("\nExercise 3, Problem 11b")

e = 7

n = 2881

d = generatePrivate(e,n)

print("decryption key: {0}" .format(d))

def e3p12():

print("\nExercise 3, Problem 12")

e = 13

n = 35209

d = generatePrivate(e,n)

print("decryption key: {0}" .format(d))

def e3p14():

print("\nExercise 3, Problem 14")

n = [1997,1999,2001,2002,2003,2004,2005]

phi = [0,0,0,0,0,0,0]

for i in range (0,len(n)):

phi[i] = eulerPhi(n[i])

print("phi({0}) = {1}" .format(n[i],phi[i]))

def e3pEC():

print("\nExercise 3, Extra Credit")

e = 13

n = 2881

d = generatePrivate(e,n)

text = "To be, or not to be- that is the question: Whether 'tis nobler in the mind to suffer The slings and arrows of outrageous fortune Or to take arms against a sea of troubles, And by opposing end them. To die- to sleep- No more; and by a sleep to say we end The heartache, and the thousand natural shocks That flesh is heir to. 'Tis a consummation Devoutly to be wish'd. To die- to sleep. To sleep- perchance to dream: ay, there's the rub!"

textEncoded = list(text.encode())

textEncrypted = []

for i in range(0,len(textEncoded)):

textEncrypted.append(encryptRSA(textEncoded[i],e,n))

textDecrypted = []

for i in range(0,len(textEncrypted)):

textDecrypted.append(decryptRSA(textEncrypted[i],d,n))

textDecryptedPT = bytes(textDecrypted).decode()

print("Original Text: {0}" .format(text))

print("Encoded Text: {0}" .format(textEncoded))

print("Encrypted Encoded Text: {0}" .format(textEncrypted))

print("Decrypted Encoded Text: {0}" .format(textDecrypted))

print("Decrypted Decoded Text: {0}" .format(textDecryptedPT))

part = "start"

while(part != "exit"):

part = input("Which question would you like to display? [11a, 11b, 12, 14, or EC]: ")

if(part == "11a"):

e3p11a()

elif(part == "11b"):

e3p11b()

elif(part == "12"):

e3p12()

elif(part == "14"):

e3p14()

elif(part == "EC"):

e3pEC()

elif(part == "all"):

e3p11a()

e3p11b()

e3p12()

e3p14()

e3pEC()