Being Eve HW

============Diffie-Hellman===============

Alice and Bob agree on g = 7 and p = 97.
Alice sent Bob the number 53.
Bob sent Alice the number 82.

Decryption method

1) Let X and Y be Alice and Bob's secret number respectively. Then, from the Diffie-Hellman algorithm,

$53 = 7^X \bmod 97$
$82 = 7^Y \bmod 97$

2) Compute X

```
def main():
    for i in range(0,97):
        if 53 == (7**i) % 97:
            print("X = " + str(i))
            print("Shared secret = " + str((82**i)%97))
main()
```

X = 22

3) Compute shared secret

$B^X \bmod p = 82^{22} \bmod 97 = 65$

Shared secret = 65

If the integers were much larger, then I wouldn't have figured out the correct X, since my for loop only goes up to 97

==============RSA==============

Plaintext message: Dear Bob, check this out. https://www.surveillancewatch.io/ See ya, Alice.

Decryption method

1) Find out Bob's p and q.

- Generate all prime numbers up to n_Bob//2. Integer division by 2 because the largest factor of n_Bob can't be greater than half of n_Bob

```
primes = prime_generator(n_Bob//2)
```
(see the **helper functions** section for function details)

- Both p and q must be divisors of n_Bob and primes. As such, iterate through the list of prime numbers that are less than the square root of n_Bob, and check whether (1) it is a divisor of n_Bob and (2) whether n_Bob divided by it gives a prime number. Only iterate though the square root of n_Bob to prevent finding the same pair of numbers twice.

```
#Find all pairs of p and q where pq = n_Bob and both p and q are prime
   potential_pairs = []
   for i in primes:
       if n_Bob % i == 0 and n_Bob/i in primes and i <= math.ceil(math.sqrt(n_Bob)):
           potential_pairs.append((i,int(n_Bob/i)))
```
- p and q are 389 and 419 (order doesn't matter)

2) Find d_Bob
- Compute lambda(n), which is lcm(389-1,419-1) = 81092
- Find d_Bob using brute force—iterate from 1 to a large number (arbitrarily chosen), and perform the following boolean check. Since it doesn't matter which d_Bob we chose, so long as it satisfies the condition specified in the boolean check below, we can exit the for loop as soon as we find one.

```
   for pair in potential_pairs:
       lamb_n = math.lcm(pair[0]-1,pair[1]-1)
       d_Bob = -1
       for i in range(1,1000000):
           if (e_Bob*i)%lamb_n == 1:
               d_Bob = i
               break
```
- d_Bob is 43665

3) Use Bob's private key to decrypt ciphertext to encoded message
- Use Bob's private key (n_Bob and d_Bob) and the formula below to decrypt the ciphertext into the encoded message

```
encoded = (cipherchar**d_Bob)%n_Bob
```

4) Along the way, decode the message
- Convert each decrypted number into hexadecimal
- Look one byte at a time, grab the hexadecimal value and use the ASCII chart to convert to text.

```
       plaintext = ""
       for cipherchar in ciphertext:
           encoded = (cipherchar**d_Bob)%n_Bob
```

```
        #String manipulation so it has the right format for the int and chr
functions
        encoded1 = str(hex(encoded))[:4]
        encoded2 = "0x" + str(hex(encoded))[4:]

        plaintext += chr(int(encoded1,16)) + chr(int(encoded2,16))

    print(plaintext)
```

Helper functions

```
def isPrime(num):
   for i in range(2, math.ceil(math.sqrt(num))):
       if num % i == 0:
           return False
   return True


def prime_generator(max):
   primes = []
   for i in range(2,max):
       if isPrime(i):
           primes.append(i)
   return primes
```

Show precisely where in your process you would have failed if the integers involved were much larger.

- If Bob's n is very large, that translates to having a very large lambda(n), which means that d_Bob might be larger than my for loop range (1000000) in step 2:

```
   for pair in potential_pairs:
       lamb_n = math.lcm(pair[0]-1,pair[1]-1)
       d_Bob = -1
       for i in range(1,1000000):
           if (e_Bob*i)%lamb_n == 1:
               d_Bob = i
               break
```

Why Alice's character encoding is insecure?

- Converting the decimal numbers into hexadecimal values, which stores two bytes at a time, plausibly suggests that each byte is an ascii representation of the plaintext character.
- The ascii character encoding is commonly used; the encoding and decoding scheme is available to the public. Anyone can look it up.