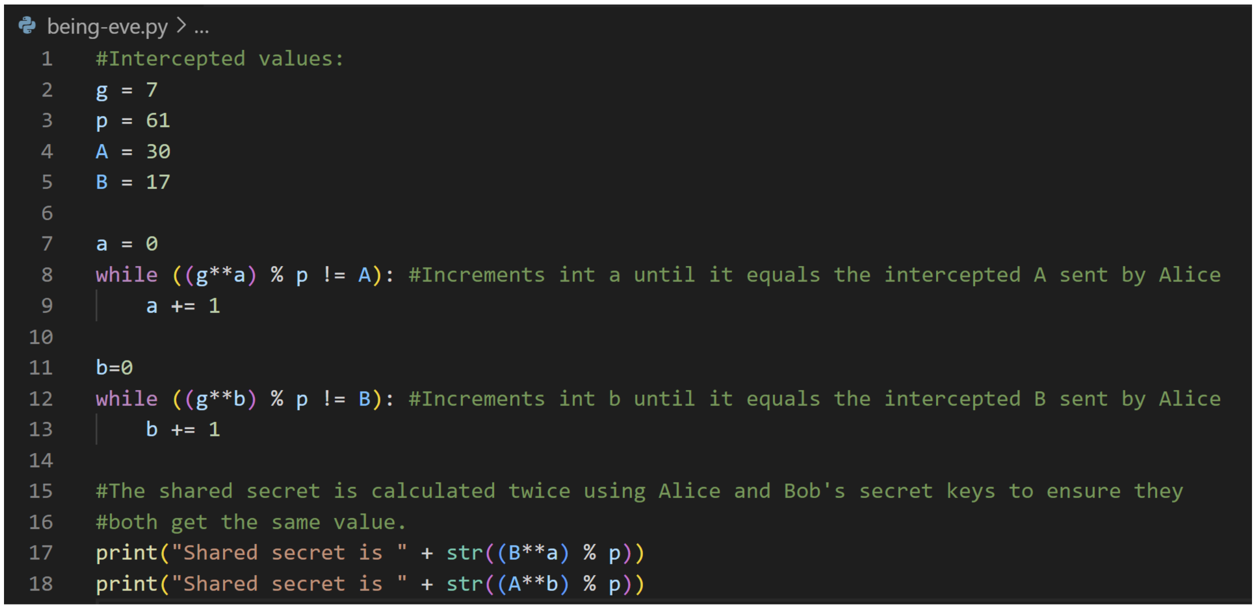
Author: Kyle Machalec

DIFFIE HELLMAN

The shared secret is 6.

The comments in the code below describe the method I used to find the shared secret.



being-eve.py > ...

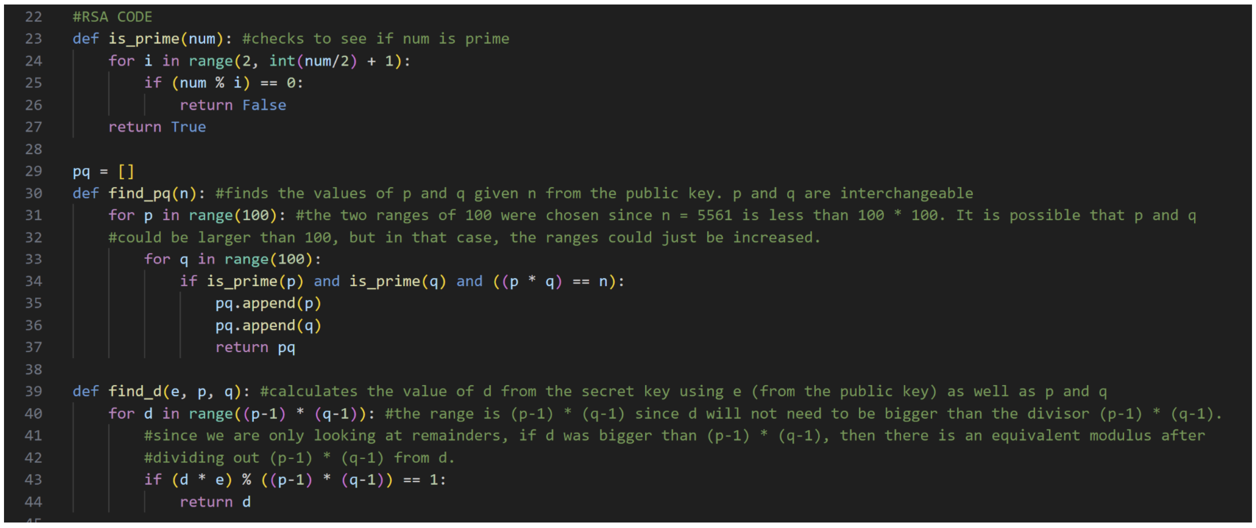
```python
1   #Intercepted values:
2   g = 7
3   p = 61
4   A = 30
5   B = 17
6
7   a = 0
8   while ((g**a) % p != A): #Increments int a until it equals the intercepted A sent by Alice
9       a += 1
10
11  b=0
12  while ((g**b) % p != B): #Increments int b until it equals the intercepted B sent by Alice
13      b += 1
14
15  #The shared secret is calculated twice using Alice and Bob's secret keys to ensure they
16  #both get the same value.
17  print("Shared secret is " + str((B**a) % p))
18  print("Shared secret is " + str((A**b) % p))
```

If Alice and Bob chose very large values for g, p, a, and b, the while loops on lines 8-9 and lines 12-13 would not be able to be completed. Computers are particularly bad at calculating a modulo, so the repeated calculations on line 8 and line 12 would take an unreasonable amount of time to complete.

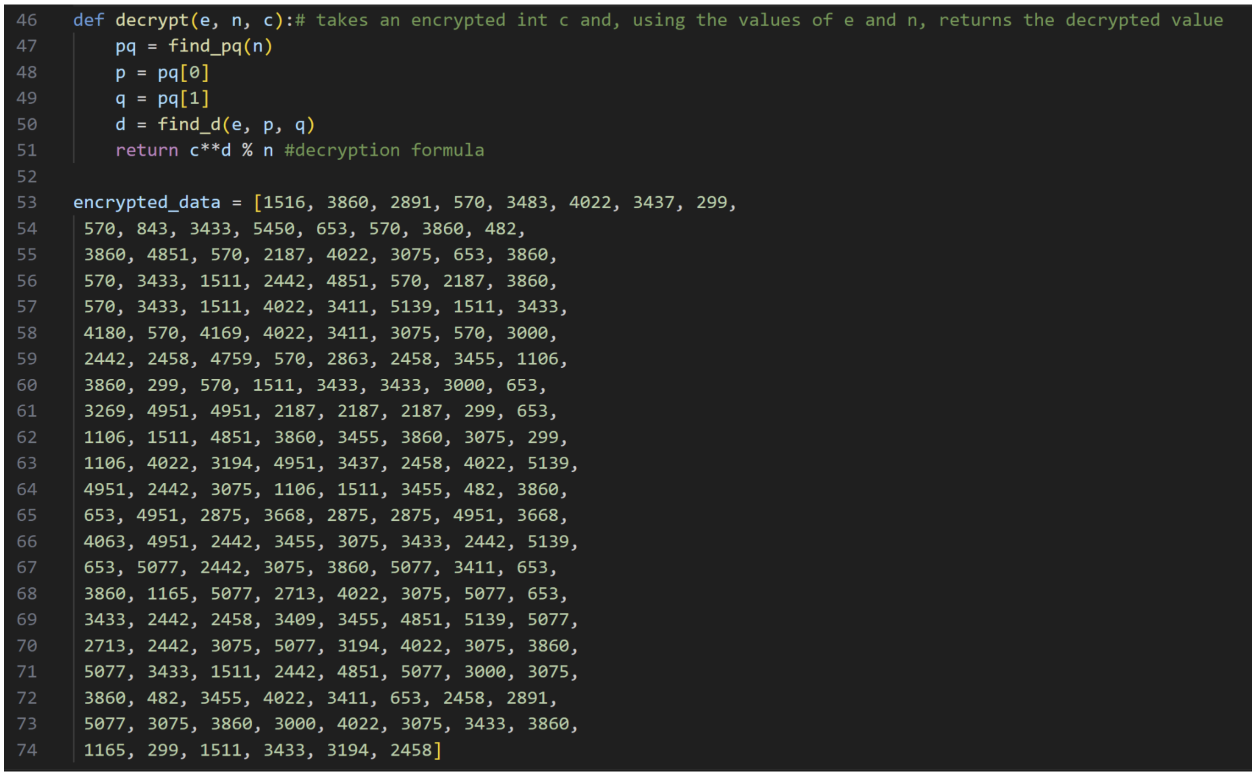
RSA

Shared secret: Hey Bob. It's even worse than we thought! Your pal, Alice. https://www.schneier.com/blog/archives/2022/04/airtags-are-used-for-stalking-far-more-than-previously-reported.html

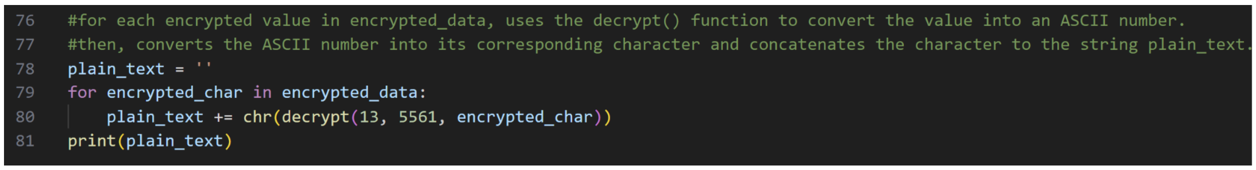
The comments in the code below describe the method I used to find the shared secret.



```python
#RSA CODE
def is_prime(num): #checks to see if num is prime
    for i in range(2, int(num/2) + 1):
        if (num % i) == 0:
            return False
    return True

pq = []
def find_pq(n): #finds the values of p and q given n from the public key. p and q are interchangeable
    for p in range(100): #the two ranges of 100 were chosen since n = 5561 is less than 100 * 100. It is possible that p and q
    #could be larger than 100, but in that case, the ranges could just be increased.
        for q in range(100):
            if is_prime(p) and is_prime(q) and ((p * q) == n):
                pq.append(p)
                pq.append(q)
                return pq

def find_d(e, p, q): #calculates the value of d from the secret key using e (from the public key) as well as p and q
    for d in range((p-1) * (q-1)): #the range is (p-1) * (q-1) since d will not need to be bigger than the divisor (p-1) * (q-1).
        #since we are only looking at remainders, if d was bigger than (p-1) * (q-1), then there is an equivalent modulus after
        #dividing out (p-1) * (q-1) from d.
        if (d * e) % ((p-1) * (q-1)) == 1:
            return d
```



```python
def decrypt(e, n, c):# takes an encrypted int c and, using the values of e and n, returns the decrypted value
    pq = find_pq(n)
    p = pq[0]
    q = pq[1]
    d = find_d(e, p, q)
    return c**d % n #decryption formula

encrypted_data = [1516, 3860, 2891, 570, 3483, 4022, 3437, 299,
 570, 843, 3433, 5450, 653, 570, 3860, 482,
 3860, 4851, 570, 2187, 4022, 3075, 653, 3860,
 570, 3433, 1511, 2442, 4851, 570, 2187, 3860,
 570, 3433, 1511, 4022, 3411, 5139, 1511, 3433,
 4180, 570, 4169, 4022, 3411, 3075, 570, 3000,
 2442, 2458, 4759, 570, 2863, 2458, 3455, 1106,
 3860, 299, 570, 1511, 3433, 3433, 3000, 653,
 3269, 4951, 4951, 2187, 2187, 2187, 299, 653,
 1106, 1511, 4851, 3860, 3455, 3860, 3075, 299,
 1106, 4022, 3194, 4951, 3437, 2458, 4022, 5139,
 4951, 2442, 3075, 1106, 1511, 3455, 482, 3860,
 653, 4951, 2875, 3668, 2875, 2875, 4951, 3668,
 4063, 4951, 2442, 3455, 3075, 3433, 2442, 5139,
 653, 5077, 2442, 3075, 3860, 5077, 3411, 653,
 3860, 1165, 5077, 2713, 4022, 3075, 5077, 653,
 3433, 2442, 2458, 3409, 3455, 4851, 5139, 5077,
 2713, 2442, 3075, 5077, 3194, 4022, 3075, 3860,
 5077, 3433, 1511, 2442, 4851, 5077, 3000, 3075,
 3860, 482, 3455, 4022, 3411, 653, 2458, 2891,
 5077, 3075, 3860, 3000, 4022, 3075, 3433, 3860,
 1165, 299, 1511, 3433, 3194, 2458]
```



```python
#for each encrypted value in encrypted_data, uses the decrypt() function to convert the value into an ASCII number.
#then, converts the ASCII number into its corresponding character and concatenates the character to the string plain_text.
plain_text = ''
for encrypted_char in encrypted_data:
    plain_text += chr(decrypt(13, 5561, encrypted_char))
print(plain_text)
```

As stated in the comment above, each value in the list encrypted_data represents one character that has been encoded into an ASCII number and then encrypted using RSA.

The function find_pq() on lines 30-37 has a runtime of $O(n^2)$ due to the two loops. With big enough values of p and q, resulting in a massive value of n, an unattainable amount of computing power would be needed to run find_pq() in a reasonable amount of time.

Alice's message would still be insecure even if Bob's keys involved larger integers because each value in the encrypted data corresponds to a particular ASCII value. As a result, there would still be repeated values (like 3860 representing "e") that a frequency analysis could use to crack the encryption without the need for a brute-force calculation of p and q.