Exercise:
Encrypting and Decrypting with RSA

In this exercise, you will encrypt and decrypt numbers using a simple version of the RSA algorithm. Each team should have two members. ***Each*** team-member should complete the exercise as Bob, then pass along some information (but not the whole sheet!) to the other team-member, who will complete the exercise as Alice. In this way, both team-members will play both roles in the exercise. You should conceal your actual numbers from your team-members. You can consider your code for this exercise “prototype” code – you can throw it away and start design when you start the lab.

Instructions for Bob:

We will be doing B=16-bit RSA.

1. Select the encryption exponent e=17. (In practice, e=65537 would often be used for larger p and q.)
2. Calculate p like this: (Write results in the table below)
   a. Select a (B/2=) 8-bit random number. You can use `random.randint(i,j)` to select an integer x satisfying `i`<=x<=`j`. (You need to `import random` to use `random`.)
   b. Set the two highest bits and the lowest bit (to 1). This forms our tentative *p.*
   You can look at the binary form of, e.g. p, using `"{0:b}".format(p)`. You can set the highest bit in p using p = p | 0b10000000, and set all three bits similarly, by putting ones in the positions of the bits you wish to set in the 0bNNNNNNNN number used above.
   c. Check if p is prime. If p is not prime, add 2 to p and try again. (You can use an inefficient program to check if the number is prime; e.g., check if all numbers smaller than p do not divide p).
   d. Check if the number (p-*1*) is co-prime with e=17, i.e., gcd(p-1,e)=1. If not, add 2 to p and try again. (This step is necessary to ensure that we can find a d such that ed = 1 (mod z))
   Note: since e=17 is prime, you can simply check that (p-1) mod e≠0.

Initial random number (decimal)	Initial random number (binary)	*p (decimal)*	*p (binary)*	Is *p* prime?	Is (p-1)%e≠0 ?
	Final:				

(Instructions for Bob, continued.)

3. Repeat step 2 to select *q*. (Note: *q* must be different from *p*. Start over if *q* will equal *p*.)

Initial random number (decimal)	Initial random number (binary)	*q (decimal)*	*q (binary)*	Is *q* prime?	Is (q-1)%e≠0 ?
	Final:				

4. Calculate the modulus $n = pq$

   $n =$

5. Calculate the totient z = (*p*-1)*(*q*-1)

   z =

6. Select the decryption exponent *d* such that (*de*) mod z = 1. You can simply "guess and check" all values of 1 < d < z. Only one value of d will work if e is selected as in step 5. (This would usually be done using the Extended Euclid's Algorithm.) This is your private key. Do not reveal d, p, q, n, or z to Alice or Trudy!

   d =

7. Provide your public key [e;n] (that is, simply the numbers e and n) to Alice and Trudy. (This simulates posting your public key on your personal website…)
8. Wait for Alice to send you a secret message.
9. Once you receive the secret message from Alice, you can decrypt it using your private key. Suppose *c* is the ciphertext. Compute the original message *m* as $m = c^d \bmod n$. For smaller numbers you can simply compute this as `(c**d)%n`.

   c = m =

   Don't reveal the secret message to Trudy!

Instructions for Alice:

1. You will receive the public key [e;n] (That is, simply the numbers e and n) from Bob. Write it here:

   e = n =

2. Select any number $0 <= m < n$ for your plaintext secret message. If you like, you can encrypt a sequence of ASCII characters as separate messages *m* (that is, using block encryption.)

   m =

3. Compute the ciphertext c as $c = m^e \bmod n$. For smaller numbers you can simply compute this as `(m**e)%n`.

   c =

4. Give the ciphertext message *c* to Bob and Trudy. This simulates Trudy eavesdropping on the wire.

Instructions for Trudy: (optional)

(If you have extra time, you may want to play this role – simply get [e;n] and c from another team!)

1. Wait to receive the public key [e;n] (This is simply the numbers e and n) from Bob.

   e =

   n =

2. Factor n to find p and q (Use brute-force Python loop. This is the hard step that makes RSA secure for large numbers.)

   p =

   q =

3. Compute z = (p-1)*(q-1)

   z =

4. Now compute *d* the same way as Bob did: Select the decryption exponent *d* such that (*de*) mod z = 1. You can simply "guess and check" all values of d < z. Only one value of d will work if e is selected as in step 5. (This would usually be done using the Extended Euclid's Algorithm.)

5. Wait to receive (eavesdrop) on the ciphertext message *c* from Alice to Bob.

   c =

6. Decrypt the *c*, ciphertext message: Compute the original message *m* as $m = c^d \bmod n$. For smaller numbers you can simply compute this as (c**e)%n.

   m =

Acknowledgement: The simple form of the RSA encryption/decryption used in this exercise is based on Avi Kak's lecture notes on cryptography, available at
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf

and from the text, Kurose & Ross, Computer Networking: A Top-Down Approach, 6th Edition, Section 8.2.2, pp. 684-688