Mini-Project Option #3: Breaking the code

RSA encryption is an important part of today's world. It allows people to exchange information securely over the Internet without worrying about third parties eavesdropping on their communications. The way this method works is by making use of a **public key** and a **private key**.

Let's say you have a friend Angela, who lives in Russia. She wants to send your mom a birthday gift, but she obviously doesn't want your mom to find out what the gift is. Since your mom has advanced knowledge in cryptography and hacking, you need to find a way to safely encrypt Angela's messages so the surprise isn't spoiled.

This is the message that Angela wants to send you:

> It's a matryoshka doll. It's a doll inside a doll inside a doll. It's a nested doll.

So you send your friend your **public key**. It is an integer, typically large. It is fine if your mom learns your public key - it is only useful to encrypt messages, not to decrypt them. Your public key is therefore an *encryption key*.

Now Angela takes your public key and uses it to convert the message into a set of numbers that looks like this:

> 75741183467675366186427574456707508878297291982135053250643238387653018959583309185196867135259642 21...

Your mom can't decipher this with your public key. The only person that can decrypt this is you, because only you have the **private key**, which is the *decryption* key. With your private key you can reverse engineer the cypher and get the original message back:

> It's a matryoshka doll. It's a doll inside a doll inside a doll. It's a nested doll.

Your public and private keys are "entangled", so that they are always related to each other right from the moment they are created. However, it is extremely difficult (basically impossible with today's computers) to infer a private key from a public key if both numbers are large enough.

Your task is to build an RSA encryption-decryption machine. For this you will need to know how to

- Create an RSA key pair
- Encrypt using a public key
- Decrypt using a private key

Creating an RSA key pair

You start with a pair of prime numbers *p* and *q*. Let's say that

$p = 13$
$q = 17$

In general, *p* should always be 13 or greater, and *q* should always be 17 or greater. Due to the fact that this project is a simplified version of real-world RSA, it is possible that some *p*, *q* pairs won't work. If you find yourself in that situation, just experiment with several pairs and choose one that does work.

The number *n* is defined as $n = p \times q$. In this case, $n = 13 \times 17 = 221$. The function $\phi(n)$ is defined as $\phi(n) = (p - 1)(q - 1)$, which in our example is equal to $\phi(n) = 12 \times 16 = 192$.

Now we need another integer *e* that is greater than 1 and less than $\phi(n)$. It is important that $\phi(n)$is not divisible by *e*. A good value for e in this example could be 5, because it is greater than one, is less than $\phi(n)$, and doesn't perfectly divide 192 (if you divide 192/5 the result is 38.4, which is not an integer). To check if one number is a factor of another number, you can use the `gcd()` function in Python's built-in `math` module. Like this: `math.gcd(192,5)`. If the result is equal to 1, it means that the numbers aren't factors of each other (gcd stands for "greatest common divisor"). A good idea is to choose an *e* that is small.

Your **public key** is two values: *n* and *e*. Here, your public key is the value 221 together with the value 5.

To calculate the private key *d*, we use this formula:

$d = (i \times \phi(n) + 1) / e$

where *i* is any integer. For the sake of simplicity, you can use $i = 2$ in your machine. So, our *d* would be:

$d = (2 \times 192 + 1)/5 = (384 + 1)/5 = 385/5 = 77$

In the end:

- Your public key is the numbers $n = 221$ and $e = 5$
- And your private key is $d = 77$

When you write your Python code, it is important that you make sure that all of your numbers are of type `int`. If you use decimal numbers, you're likely to run out of memory (the RSA algorithm can be a bit heavy on computation sometimes).

Encrypting using a public key

The cipher *c* (the encrypted version of the text) is calculated like this:

1. You assign each letter of the message a number. For example, you can use each letter's position in the alphabet.
2. You apply the encoding formula (described later) to the resulting sequence of numbers.

You can use this replacement dictionary for your letters[1]:

A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12
M	13
N	14
O	15
P	16
Q	17
R	18

[1] In your code, you may want to stick to only lowercase or only uppercase to reduce the complexity of your program.

S	19
T	20
U	21
V	22
W	23
X	24
Y	25
Z	26
space	27

So, the string: “apple” will be translated to [1, 16, 16, 12, 5], because A=1, P=16, L=12, and E=5. Similarly, “orange” would be [15, 18, 1, 14, 7, 5].

The formula to calculate *c* is (for each character):

$$c = translation^{e} \bmod n$$

Recall that mod is a function that finds the remainder of two numbers after you divide one by the other. For instance, $11 \bmod 2$ is 1, because when you divide 11 by two, you get 5, and the remainder is 1 (5 times 2 is 10, and you need to add 1 to get the original 11). In Python, the mod operator is % (so, you would write 11 % 2).

Our cipher for “apple” would be [1, 152, 152, 207, 31], because

$$1^{5} \bmod 221 = 1$$
$$16^{5} \bmod 221 = 152$$
$$16^{5} \bmod 221 = 152$$
$$12^{5} \bmod 221 = 207$$
$$15^{5} \bmod 221 = 31$$

Decrypting using a private key

To decrypt a cipher *c*, you simply need to use this formula:

$$decrypted = c^{d} \bmod n$$

So, for “apple”, this would be the decrypted message:

$$1^{77} mod\ 221 = 1$$
$$152^{77} mod\ 221 = 16$$
$$152^{77} mod\ 221 = 16$$
$$207^{77} mod\ 221 = 12$$
$$31^{77} mod\ 221 = 5$$

That you can easily backtranslate to “apple”.

Note: This is a very simplified version of the RSA algorithm. There are more robust ways to encrypt data using RSA in Python, but the method we present here will definitely give your mom a hard time at least.

Expected Inputs and Outputs

You will have to create two Python scripts - one for encrypting, and another one for decrypting. It is up to you what prime numbers to use as *p* and *q*, but the larger, the better.

The encryption script should take a string as user input (you can use Python’s `input()` function for that), and print the cipher.

The description script should be capable of taking the cipher as user input and printing the original message. Since the cipher is a list of integers, the description script must have some way of reading the cipher one element at a time (e.g. by separating the numbers with commas (33,66,77,88)).