Activity: RSA Cryptography Lab

Let's implement the ideas behind RSA cyprtography to create a public encryption key that you can use to have your friends send you secure messages.

We will use SAGE to compete this activity. The easiest way to get started with this is to use the online SAGE cell at https://sagecell.sagemath.org/. Of course if you have SAGE installed, you can use that.

1. First you will need to select two very large prime numbers. Let's shoot for 20-25 digits. Luckily, SAGE has the command `next_prime()` which will give you the next prime larger than your input. So pick a random input and get two primes. You will want to save these as a constant. So `p = next_prime(20)` for example.

2. Now you can compute $n = pq$ and $m = (p-1)(q-1)$. Notice that you *could* ask SAGE to find $\varphi(n)$ using `euler_phi(n)`, but you should think about why this is a really really really bad idea.

3. Now we need E and D. Recall that we want $\gcd(E, m) = 1$ and $DE \equiv 1 \pmod{m}$. How are you going to find these?

SAGE has the command `gcd(E, m)` that will compute the gcd of the two inputs. There is also the command `inverse_mod(E, m)` which will run the Euclidean algorithm forwards and backwards on E and m.

4. You can now publish the encryption pair (n, E) so your friends can send you messages. If they had a message x to send you, they would simply need to compute $x^E \pmod{()}n$. SAGE can do this with `mod(x^E,n)`, but if x and E are large, this would take a really long time.

Luckily, there is a better way: `power_mod(x,E,n)` computes the same thing, but uses the method of repeated squaring to make this much more efficient. You will need to use this to decrypt the message y when you take $y^D \pmod{n}$. So once you get a message, do that.

5. The last piece of the puzzle is what to do with the number you get out of the decryption. You will find some x, but need that to be translated into text you can read.

This depends on the agreed upon method for translating a string of symbols into numbers. Suppose we have a string called `message`. We can create a list of ASCII code values (0 through 127) using `digits = [ord(letter) for letter in message]`. That is, `ord()` converts a single letter into its ASCII code.

We then must convert a collection of letters into a single number base 128. SAGE can do this using the `ZZ()` function. So `ZZ(digits,128)`. This will only work if $128^k < n$, where k is the number of digits. So in practice, we would break the longer message into chunks and encrypt each chunk separately.

To undo this coding, you can break down the received message `received` using `digits = received.digits(base=128)`, which makes a list of digits. Then you can create a list of letters using `letters = [chr(ascii) for ascii in digits]`. Finally, put these letters into a string using `''.join(letters)`.