RSA Secret-Sharing Activity

In this activity, groups of students will set up the RSA keys and practice encoding and decoding messages between groups.

** Be sure your Kernel for this activity is set to **SageMath 9.4**.

Step 1: Setup

1. Pick two prime numbers, p and q.

- Keep these secret: $p =$ _____ $q =$ _____.
- Try `random_prime(2^50,lbound=100)` if you need help choosing a prime.

2. Find the number $b = \text{lcm}((p-1)\cdot(q-1))$.

- $b =$ _____. Keep b secret, too.

3. Find the number $n = p \cdot q$.

- $n =$ _____. You will want to publicize n.

4. Now, pick a public-key exponent e, such that e and n are relatively prime.

- $e =$ _____. You will want to publicize e.

5. Finally, find your secret private-key exponent d. To do this, find the number d such that $e \cdot d \pmod{b} = 1$.

- Here we need to do the extended Euclidean algorithm to find integers u, v so that $1 = e \cdot u + b \cdot v$.
- We know how to do this by hand, but SageMath can do it using the `xgcd` command.
- The syntax is `xgcd(e,b)` and this will return a triple `(gcd(e,b),u,v)` where

$$\gcd(e, b) = e \cdot u + b \cdot v.$$

- Remember that if u is negative, just add b until you get a positive value. This will be your private key d.
- $d =$ _____. Do NOT tell anyone d.

Step 2: Secret sharing

1. Find another team with whom you want to exchange messages, and tell them your modulus n and your public-key exponent e. They should also tell you their modulus (call it n_2) and their public-key exponent (call it e_2).

- Their key: $n_2 =$ _____, $e_2 =$ _____.

2. To send a message, convert it to numbers using the `StringToInt` command along with the `map` and `list` commands.

```
message='hello'
plaintext=list(map(StringToInt,message))
```

3. Your message: _____

- Its numerical equivalent: _____

4. For each number m in your `plaintext` variable, calculate $m^{e_2} (\text{mod } n_2)$. Write a function[1] to do this encoding. Then, you can map it to the list. Call the result `ciphertext`. This is your encoded message. Note: *You are encoding the message using the other team's public key.*

```
def encode(m,e2,n2):
  your code here...
```

5. Your encrypted message: _____

6. Then, give your encrypted message to the team from whom you got n_2, e_2. They should give you a sequence of numbers as well.

- Their encrypted message: _____

7. Now, to decrypt their message, use your secret exponent d. For each number c in their message, compute $c^d (\text{mod } n)$. Write a function to do this decoding. Then, you can map it to the list. *Note that you are decoding the message using your private key.* This will reveal the numbers in their original message!

```
def decode(c,d,n):
  your code here...
```

8. Use `IntToString` to convert these back to letters to get their message to you.
9. Their message: _____

[1] You can use the command `pow(m,e,n) for large integers.` to efficiently compute $m^e (\text{mod } n)$.