The Procedure

1. Bob chooses prime numbers p, q so that $pq = n > c$, where c is the size of the largest message they expect to encrypt. The n is called the key length.

2. Bob also chooses e, d by first computing $\varphi(n) = (p-1)(q-1)$, then choosing e so that $\gcd(e, \varphi(n)) = 1$. Bob can then release the keypair (e, n), and now Alice, Bob, and Eve respectively have their own step to perform.

   Bob: Will compute the multiplicative inverse of $e \pmod{\varphi(n)}$, using Wolfram-Alpha, in order to find his private key d.

   Alice: Will choose a message which is converted to a large natural number M. We have many ways of doing this, however for the purpose of this exercise we will use $a = 1, b = 2, \ldots, z = 26$, and a space is 00. For messages longer than a single character, simply combine the letters. For example, a message "abc" will be encoded with $M = 010203$. Alice encrypts her message text with Wolfram-Alpha using $M^e \pmod{n}$, and reveals it to both Bob and Eve.

   Eve: Will use Wolfram-Alpha to attempt to factor n, and if successful, uses the p, q she finds in order to find d (see Bob's instructions above).

3. Now Bob and Eve (if successful) should reveal the decrypted messages; and confirm whether they recieved it.

Exercise 1

For the first exercise, delegate who will be Bob, Alice, and Eve, and perform the procedure with $p = 2$ and $q = 13$. This first exercise is exactly enough to encrypt a message of one character. As a group, fill out the following:

1. What was the message sent by Alice?

2. What was Bob able to decrypt?

3. What was Eve able to encrypt?

4. How private was the message?

5. For what purposes would you use this level of encryption?

Exercise 2

For the second exercise, use `bigprimes.org` or some other random number generating site to choose some primes p, q with more digits. Try to ensure that the primes are small enough that Eve has a chance at breaking the message. All participants will need Wolfram-Alpha.

1. What was the message sent by Alice?

2. What was Bob able to decrypt?

3. What was Eve able to encrypt?

4. How private was the message?

5. For what purposes would you use this level of encryption?

Exercise 3

For the third and final exercise, use `bigprimes.org` or some other random number generating site to choose some primes p, q with enough digits that Bob and Alice can agree that Eve will not be able to factor n using Wolfram-Alpha (How can they check this!).

1. What was the message sent by Alice?

2. What was Bob able to decrypt?

3. What was Eve able to encrypt?

4. How private was the message?

5. For what purposes would you use this level of encryption?

Bonus Exercise

As an addition to the manual exercise above, your group will now be guided through how the above steps can be easily automated through a simple python script. The general generation of keys, encoding and decoding of messages, and encrypting and decrypting of messages is provided through an `rsa.py` file. It is your task to create character files for Bob, Alice, and Eve that will perform the steps you and your group members had to do manually.

alice.py: Alice is the simplest of our three characters as all that is required for this portion is to encode, encrypt, and return a message.

1. Import the `rsa.py` file.
2. Gather the inputs from Bob, aka the public key and the key length.
3. Use the functions from the `rsa.py` file to encode the message.
4. Once you have your encoded message, encrypt the message.
5. Print the encrypted message as a string.

   Paste your code below:

bob.py: Bob's task is slightly more complex as prime inputs will be required and the numpy and sympy libraries will be utilised.

1. Import the `rsa.py` file, and the numpy and sympy libraries.
2. Create a boolean variable to keep track of whether the user will input their primes, or if they should be provided by the user.

3. If the user needs to input primes, then prompt the user to give two primes. If not, proceed as below.
4. Now prompt the user to enter a lower bound for the amount of characters in the message.
5. Using `pow()`, generate a lower bound for the message length, setting 10 as the base, and the length from above as the exponent.
6. To generate our primes, we will use `sympy.randprime()`, where our input will be lower, and lower*2. Do this twice such that values of p and q are present.
7. Check that the primes are unique.
8. Finally, we print our values for p and q as a string, separated by commas.
9. Now we must generate and return these keys to Bob so he may share them with Alice and Eve, to do this we use our `rsa.py` file, and use `generate_keys` to encrypt our p and q values.
10. Print all three values (the public key, private key, and key length) as strings.
11. Lastly, once Alice has created and encrypted a message, Bob can decode it. Promt the user for the message obtained from Alice.
12. Then we use `decrypt_message` from the `rsa.py` file, feeding it the above message from Alice, the private key, and the key length.
13. Finally, we use `decode_message` from the `rsa.py` file on the above value obtained from `decrypt_message` to obtain the decrypted message from Alice.

    Paste your code below:

eve.py: It is Eve's task to collect all inputs from Alice and Bob in order to decrypt and decode the messages.

1. Import the `rsa.py` file, and the numpy and sympy libraries.
2. Gather the inputs from Bob and Alice, aka the public key, key length, and the message from Alice.
3. Next, using `factorint()` from the sympy library, factor the key length to find p and q. (Make sure to assign this to a variable, for example, `factors`).
4. Now you will want to check if the length of this is equal to 2. If it is you will want to raise a `ValueError`, as n should never have exactly two prime factors. If it is not, proceed by returning the factors to Eve (you can obtain these from the dictionary returned by `factorint` using `factors.keys()`).
5. Print the primes, p and q, you have found as strings separated by commas.
6. Now you must set your variables (p, q, and n). p and q will be easy enough as they are your primes. n will simply be the two primes multiplied by one another.
7. To get the modular inverse, we need $\varphi(n)$, which we can obtain by multiplying $p-1$ and $q-1$.
8. Now to get the modular inverse, we will use `mod_inverse` from the sympy library, with the inputs being the public key and $\varphi(n)$.
9. Finally, to decrypt and decode the message we will use the `decrypt_message` function from the `rsa.py` file, our input being the message from Alice, the modular inverse, and our n from earlier.
10. The above leaves us with the decrypted message from Alice, which can be printed as a string.

    Paste your code below:

The above code can be cross referenced with the provided solutions so as to ensure correct documentation.