RSA

Theory:

Public-key cryptography uses two separate keys, one for encryption (the public key) and one for decryption (the private key). Anyone with the public key can compute an encrypted message that only the owner of the private key can read.

RSA was the first algorithm that demonstrated this concept. Its security assumptions are based on complexity theory: computing the product of two prime numbers is easy (polynomial time), but there is no efficient algorithm for factoring them back (so far, all factorization methods are in the non-polynomial class).

The keys for the RSA algorithm are generated the following way:

1. Choose two different large (128 bit) random prime numbers ***p*** and ***q.*** Figure out how to generate large prime numbers in python
2. Calculate ***n = pq***
   - ***n*** is the modulus for the public key and the private keys
3. Calculate the totient: ***phi (n)=(p-1)(q-1)***.
4. Choose an integer ***e*** such that 1 < ***e*** < ***phi(n)***, and ***e*** is coprime to ***phi(n)***
   - ***e*** is released as the public key exponent. You may take e = 11 here, if the number 11 is coprime to ***phi(n)***
5. Compute ***d*** to satisfy the congruence relation ***d*e*** ≡ ***1 (mod(phi(n)))***
   - ***d*** is kept as the private key exponent. There is an iterative and recursive way to calculate it. Recursion should be a faster solution

The public key is made of the modulus ***n*** and the public (or encryption) exponent ***e***.

The private key is made of the modulus ***n*** and the private (or decryption) exponent ***d*** which must be kept secret.

Encrypting a message: ***c = m^e (mod n)***

Decrypting a message: ***m = c^d (mod n)***

Using pow(a,b,c) would be a faster solution

Procedure:

Colab Notebook Link for this lab:
https://colab.research.google.com/drive/19K5o0j67kZtuGnQyG3B1HahGV0IU18og?usp=sharin
g

Submission form: https://forms.gle/4dhgtcJaCKhANpv4A