Implementation of the RSA Algorithm in Python

Fernando Murillo Bravo
Sevilla, España
fermurbra@alum.us.es

I. Introduction

I have developed the RSA algorithm in python to be able to encrypt and decrypt messages. The RSA algorithm helps us to keep a conversation private (authenticity and confidentiality) thanks to a pair of keys (public key and private key).

How to use it

Each participant generates a **private key**, which must remain secret, and a **public key**, which can be shared openly.

To send a secure message to user A, you must encrypt the message using A's public key. Only A, who possesses the corresponding private key, will be able to decrypt it.

If someone sends you a message, it means they used your public key for encryption, and only you can decrypt it with your private key.

This secure communication is made possible by the mathematical difficulty of deriving the private key from the public key, a process based on operations involving two large prime numbers.

Relevant Files

utils, where we find all the mathematical tools we will need.
setup, where we calculate the private key and the public key.
encrypt, where we encrypt the message with the public key.
decrypt, where we decrypt the message with the private key.
main, where we execute all the parts.

II. Mathematics

In this section, I will explain in full detail how the RSA algorithm works at a math level.

Public key
(e, N)
Private key
(d, N)

PUBLIC KEY

First, we choose two large prime numbers **p** and **q**, and compute their product: **n = p × q**. Then, we calculate Euler's totient function of n:

$$\phi(n) = (p-1)(q-1) \tag{1}$$

Next, we select a number **e** such that:

$$1 < e < \phi(n) \tag{2}$$

$$\gcd(e, \phi(n)) = 1 \tag{3}$$

This ensures that e is coprime with $\phi(n)$, which is required for the modular inverse to exist.

PRIVATE KEY

Having calculated the public key, we now compute the private key component **d**, which is the modular inverse of e modulo $\phi(n)$:

$$d \equiv e^{-1} \pmod{\phi(n)} \tag{4}$$

This can be done using the Extended Euclidean Algorithm, which provides coefficients that satisfy Bézout's identity.

ENCRYPTION

To encrypt a message, we first encode each character, for instance using ASCII. Once we have a numeric representation **M**, the ciphertext **C** is computed using the public key (e, n):

$$C = M^e \mod n \tag{5}$$

DECRYPTION

To decrypt the ciphertext **C**, we use the private key (d, n) to recover the original message **M**:

$$M = C^d \mod n \tag{6}$$

The result will still be in its encoded form (e.g. ASCII), which can be converted back to text.

III. Code

In this section, I will explain how I develop the code for the RSA algorithm. In addition, I will explain the problems that I had while developing the code. If you want to find more details you can find it in the comments of the code.

UTILS

First, I started to develop the mathematical tools that I would need in the future. We find two main functions, the Extended Euclidean Algorithm and the generation of prime numbers.

Extended Euclidean Algorithm and Modular Inverse: The goal of this function is to find **x**, which is the modular inverse of e modulo $\phi(n)$, and also the private key. To do this, we change the modular equation (4) into a Diophantine equation. Then, we can solve it using the Extended Euclidean Algorithm.

This part is difficult because we need to keep all the steps of the divisions $a = b \cdot q + r$, so we can go back step by step and find the value of x. In addition, I had problems when the value of b is bigger than a because the x and y didn't match.

$$ex + \phi(n)y = 1 \tag{7}$$

Prime Test and Prime Generator: The goal of this function is to generate a prime number with k digits. To do this, we randomly generate a number with k digits and apply a primality test. In this case, we use the Miller-Rabin test repeatedly until a prime number is found.

SETUP

Setup RSA: The goal of this function is to return all the data that one person needs if they want to use the RSA algorithm. We will provide the public and private key. In conclusion, I provided p, q, N, e, d, ϕ. Furthermore, we will use all the mathematical functions that I have developed and the theory of the RSA that I explain at the beginning.

ENCRYPT AND DECRYPT

Encrpyt: On the one hand, the goal of the encrypt function is to use the public key to transform the message into ciphertext. However, before using the public key, it is essential to first encode the message using ASCII, so that each character can be represented as a numerical value..

Decrypt: On the other hand, the goal of the decrypt function is to use the private key to transform the ciphertext into a message. However, after using the private key, we must apply the encode to get the initial message.

To sum up, I have been using the function pow(base, exp, mod) that it is implemented by python. It is so useful because I can calculate the base raised to the power of exp modulo mod.

MAIN

Next, we are going to look the heart of the code where I joined all the features to make the RSA algorithm to work. Moreover, I developed a simple and interactive command line interface to prove that the code works. So, at the beginning you should choose to create a user, send a message or

read a message. If you create an user, the program will ask you how many digits the prime numbers should have. Then you will receive both keys. If you want to send a message, you need to write the emitter and the receiver.

IV. Conclusion

Finally, we were able to develop a function that generates a prime number using the Miller-Rabin algorithm. Moreover, we applied the Extended Euclidean Algorithm and the theory behind the RSA algorithm, allowing us to encrypt and decrypt messages using the public and private keys.

One possible improvement for the future is to ensure that the person who sends the message is really who they claim to be. Currently, the program simply asks for your username, but we could enhance it by using the sender's private key in addition to the recipient's public key. This way, we can trust the identity of the emitter.

In conclusion, we have developed an algorithm similar to RSA. It is useful for encrypting messages and preventing unauthorized people from reading them. The most important thing is to keep the private key secret, because if someone is able to guess it, they could send messages pretending to be you.