Cryptography Project

Implementation and Analysis of RSA Algorithm

Contents

[Figure Table 1](#_Toc60638060)

[Introduction 2](#_Toc60638061)

[Implementation 2](#_Toc60638062)

[Encryption procedure 2](#_Toc60638063)

[Generating public key 2](#_Toc60638064)

[Generating private key 2](#_Toc60638065)

[Encrypting a message 3](#_Toc60638066)

[Decrypting a message 3](#_Toc60638067)

[Output of the implementation 4](#_Toc60638068)

[Encryption 4](#_Toc60638069)

[Decryption 5](#_Toc60638070)

[Performance analysis 6](#_Toc60638071)

[Conclusion 7](#_Toc60638072)

# Figure Table

[Figure 1 Output of Encryption 4](#_Toc60638073)

[Figure 2 Output of Encryption 5](#_Toc60638074)

[Figure 3 Performance of Encryption 6](#_Toc60638075)

[Figure 4 Performance of Decryption 6](#_Toc60638076)

# Introduction

This project includes an implementation of the RSA encryption algorithm. The RSA algorithm is an asymmetric algorithm, meaning it requires two keys in order to function instead of one. The keys in this algorithm are called the public key and the private key, and they are acquired through calculations that include prime numbers. The public key is, as its name suggests, given to the public, whilst the private one should only be known by its owner alone. This being an asymmetric algorithm means that even if third parties will have access to the public key, they would not be able to decrypt the message to the missing private key, making this type of encryption methods highly effective and efficient. Furthermore, the calculations needed to produce the public and private keys, and then encrypt the message are very fast and easy to compute when in possession of both keys, on the other hand the size of said numbers makes splitting the keys and finding the base numbers used to make them extremely difficult.

# Implementation

In order to implement and analyze this algorithm, the use of python object-oriented programming language was necessary to run a code and extract the necessary readings to analyze the speed of this algorithm. Excel was used to plot and visualize the extracted data and making sense of it.

# Encryption procedure

In order to start the encryption, process we need to generate both a public and a private key.

## Generating public key

To generate the public key, we need to choose two prime numbers **P** and **Q**, the product of those two primes **n** will be the first part of the public key.

**n = P\*Q**

Now we need to find a small exponent **e**, the conditions that govern what the exponent is are as follows:

1. An integer.
2. Not be a factor of n.
3. 1 **< e <** [Φ(n)](https://www.geeksforgeeks.org/eulers-totient-function/) (Φ(n) = (**P** -1)\*(**Q** -1)).

Our public key would be made of **n** and **e**.

## Generating private key

Here we will use the same Φ(n) used in the generation of the public key.

Now the private key **d** would be as follow:

**d = (k\*** Φ(n) + 1) / **e**, for some integer **k**

## Encrypting a message

To encrypt a message, we turn the letters of that message into numbers, based on their position in the alphabet (e.g. A = 1…)

Then the encrypted message would simply be:

**c** = (data as integers) **e** mod **n**

For example, if we were to encrypt the word Ali, the data as integers would be:

A = 1, L = 12, I = 9

**c** = 1129**e** mod **n**

## Decrypting a message

To decrypt data, the steps are as simple and straightforward as encrypting it, given you are aware of the parameters.

Decrypted data = **cd** mod **n**

And you convert the resulting number into alphabets using the same method before.

# Output of the implementation

## Encryption

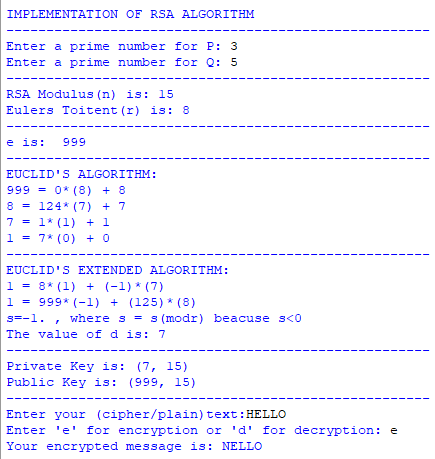


Figure Output of Encryption

## Decryption

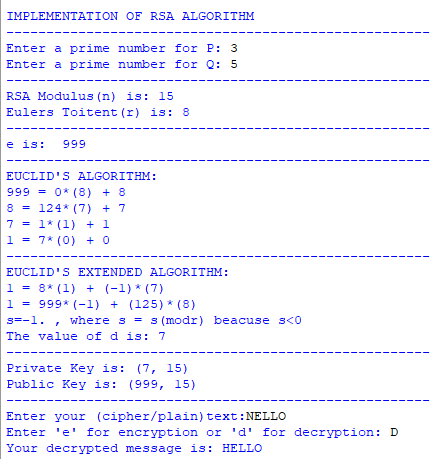


Figure Output of Encryption

# Performance analysis

Figure Performance of Encryption

Figure Performance of Decryption

# Conclusion

We can see how efficient and fast the algorithm is by observing how it performs with little variation to the speed even when the input size increases a lot. This shows how the RSA encryption algorithm is a great encryption algorithm with its fast encryption and decryption time but very long time to crack considering how taxing the process of factorizing big numbers is. Further use of the very fast and easy to generate cipher and plain text using this encryption method, one can easily generate new keys to use and make any brute force attack (most popular attacks on these kind of algorithms) redundant by disregarding all the options tested.