TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN môn cấu trúc rời rạc**

**Essay Discrete Structures**

*Người hướng dẫn*: **Nguyễn Quốc Bình**

*Người thực hiện*: **Nguyễn Hoàng Phúc**

Lớp **: 21H50301**

Khoá  **: 25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN môn cấu trúc rời rạc**

**Essay Discrete Structures**

*Người hướng dẫn*: **Nguyễn Quốc Bình**

*Người thực hiện*: **Nguyễn Hoàng Phúc**

Lớp **: 21H50301**

Khoá  **: 25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

LỜI CẢM ƠN

Em muốn cảm ơn thầy vì đã cho em cơ hội làm bt và trao dồi học hỏi kinh nghiệm từ những bài tập lớn bổ ích này, trong quá trình học tập và rèn luyện thầy đã giúp cho em thấu hiểu thêm nhiều hơn về kỹ năng lập trình cách lập trình lẫn sự tỉ mĩ trong code , em muốn cảm ơn thầy Bình

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của TS Nguyễn Văn A;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày tháng năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Hoàng Phúc*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

1. Finding an Inverse Modulo n

 Conduct research on Finding an Inverse Modulo n using the extended Euclidean

algorithm. Give your own examples.

 Implement a Python program to find an Inverse Modulo n using the extended

Euclidean algorithm. Related libraries are NOT allowed.

 Test the implemented program using sample data and verify the results. Capture

your screen results and explain them in your report document.

2. RSA cryptosystem

 Conduct research on RSA cryptosystem. Understand the mathematical concepts

behind the RSA cryptosystem, including prime number generation, modular

arithmetic, extended Euclidean algorithm, prime factorization, etc. Give your

own examples.

 Implement a Python program to encrypt and decrypt a message with the RSA

cryptosystem. Cryptography libraries are allowed.

 Test the implemented RSA cryptosystem using sample messages and verify the

results. Capture your screen results and explain them in your report document.

 Analyze the efficiency and security of the implemented RSA cryptosystem.

 Discuss the potential security threats and limitations of the RSA cryptosystem.

 Conclude with recommendations for improving the RSA cryptosystem

implementation.

MỤC LỤC

Contents

[Chapter 1 Finding an Inverse Modulo n 4](#_Toc132125495)

[1.1 Extended Euclidean Algorithm 4](#_Toc132125496)

[1.2 he Extended Euclidean Algorithm for finding the inverse of a number mod n. 5](#_Toc132125497)

[1.3 Implement a Python program to find an Inverse Modulo n using the extended Euclidean algorithm. 6](#_Toc132125498)

[Chapter 2 RSA cryptosystem 8](#_Toc132125499)

[*2.1* *mathematical concepts* behind the RSA cryptosystem 8](#_Toc132125500)

[2.1.1 prime number generation 8](#_Toc132125501)

[2.1.2 Extended Euclidean algorithm 9](#_Toc132125502)

[2.1.3 prime number generator an extended Euclidean algorithm in RSA cryptosystem 10](#_Toc132125503)

[2.2 Implement a Python program to encrypt and decrypt a message with the RSA cryptosystem. Cryptography libraries are allowed. 11](#_Toc132125504)

[2.2 Analyze the efficiency and security of the implemented RSA cryptosystem. 15](#_Toc132125505)

[2.3 potential security threats and limitations of the RSA cryptosystem. 16](#_Toc132125506)

[2.4 recommendations for improving the RSA cryptosystem implementation. 17](#_Toc132125507)

[TÀI LIỆU THAM KHẢO 19](#_Toc132125508)

[PHỤ LỤC 20](#_Toc132125509)

**DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT**

**CÁC KÝ HIỆU**

*f Tần số của dòng điện và điện áp (Hz)*

*p Mật độ điện tích khối (C/m3)*

**CÁC CHỮ VIẾT TẮT**

CSTD Công suất tác dụng

MF Máy phát điện

BER Tỷ lệ bít lỗi

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 2.1: Kiến trúc FTP 1](#_Toc387689394)

**DANH MỤC BẢNG**

[Bảng 3.1 Ví dụ cho chèn bảng 1](#_Toc387689363)

# Chapter 1 Finding an Inverse Modulo n

## 1.1 Extended Euclidean Algorithm

The Division Algorithm for Integers states that when we divide one integer by another (nonzero) integer, we obtain an integer quotient (the "answer") plus a remainder (generally a rational number). This is a fundamental concept taught in grade school. For example, when we divide 16 by 5, we get 3 as the quotient and 1/5 as the remainder. We can express this division as follows without reference to the division operation:

16 = 3\*5 + 1

Here, we have multiplied the divisor 5 with both the quotient 3 and the remainder 1/5 then sum both of them together to obtain the dividend 16. The Division Algorithm for Integers can be formally stated as follows: If a and b are positive integers, there exist unique non-negative integers q and r such that

a = qb + r, where 0 ≤ r < b.

The quotient is denoted by q and the remainder by r.

The greatest common divisor of two integers a and b, denoted by gcd(a,b), is the largest integer that divides (without remainder) both a and b. For example, gcd(15, 5) = 5, gcd(7, 9) = 1, gcd(12, 9) = 3, and gcd(81, 57) = 3.

The Euclidean Algorithm is a method for finding the gcd of two integers by repeated application of the division algorithm. We divide the divisor by the remainder until the remainder is 0. The gcd is the last non-zero remainder in this algorithm. To summarize, the Division Algorithm for Integers and the Euclidean Algorithm are important concepts in elementary number theory.

Here is a step by step example to find the gcd of 888 and 54 by the Euclidean Algorithm:

(1)888 = 16\*54 + 24

54 = 24\*2 + 6

24 = 6\*4 + 0

Let say that gcd(a,b) = r ,it can be proving that there exist integers x and y so that :

x(a) + y(b) =r

By reversing all the step in the Euclidean algorithm we can then find these integers x and y we can do this with the example (1)

We begin in the last line :

6 = 54 - 24\*2

We implement the 24 = 888 -16\*54 Into the above equation

6 = 54 – (888 -16\*54)\*2 = 54 –2\*888 +32\*54= -2\*888 + 33\*54

So in conclusion we will have the x and y being x =-2 and y = 33

## 1.2 he Extended Euclidean Algorithm for finding the inverse of a number mod n.

The extended Euclidean algorithm can also be used for finding the modular of an inverse number, this can only be archive when both number are coprime (i.e., their GCD is 1 )

Here how the extended Euclidean algorithms works to fund the modular inverse of ‘a’ modulo ‘n’

Let say that we have the equation a\*x(mod n) = 1 x is the number in the range 0 to n , this equation works because both x = 1/a i.e. the inverse of a mod n

To do this we first need to implement the Euclidean algorithm let give a and n Numbers being a= 27 and n =392

We must now find the integers that represent x and y in x(n) + y(a) =r

First step is finding the gdc :

392 = 14\*27 + 14

27 = 14\*1 + 13

14 = 13 + 1

So the gcd = 14

Second step is to reversing the gcd

14 = 392 – 14\*27

13 = 27 – 14\*1

1 = 14 – 13

From the above equation we know that we can insert 13 = 27 – 14\* 1 into the equation we will have : 1 = 14 – (27 – 14\*1) = 2\*14 -27

We can now insert 14 = 392 – 14\*27 into the equation

1 = 2\*(392 -14 \* 27) -27 = 2\*392 -29\*27 when we mod it with 392 we will have 27\*363 =1 is the same as a\*x(mod n) = 1 above so we have found our inverse mod being 363

## 1.3 Implement a Python program to find an Inverse Modulo n using the extended Euclidean algorithm.

def extended\_gcd(a,b):

    if(b==0):

        return a, 1, 0

    else :

        gcd , x1 ,y1 = extended\_gcd(b , a%b)

        x = y1

        y  = x1 -(a//b)\*y1

        return gcd  , x , y

def InverseModulo(invert\_num : int ,mod\_num : int):

    gcb, x, y = extended\_gcd(invert\_num , mod\_num)

    if(gcb != 1):

        return "error"

    else :

        return x%mod\_num

print("this is the inverse mod of number 27 mod 392 ",format(InverseModulo(27,392)))

print("this is the inverse mod of number 888 mod 54",format(InverseModulo(888,54)))

* The python programme above have two function extended\_gcb(a,b) and inverseModula(inverst\_num ,mod\_num)

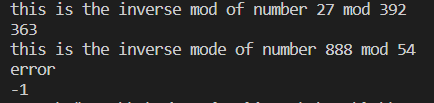
InverseModula :

* If you want to find the mod of an inverse number you much first call this function
* This function took the output of extended\_gcd into variable in order of gcd , x, y
* First check if the gcd of equal to 0 then return the error if true
* if the check is false the function will return the number x%mod\_num

extended\_gcb:

* the extended\_gbc algorithms will uses a recursive function to calculate the x and y in x(a) + y(b) =r
* the function will keep on making the recursive call until it reach a break point being if b ==0 and return a tuple (a,1,0)

this is the result of the program :



We can see that the output of the first input being correct as 363

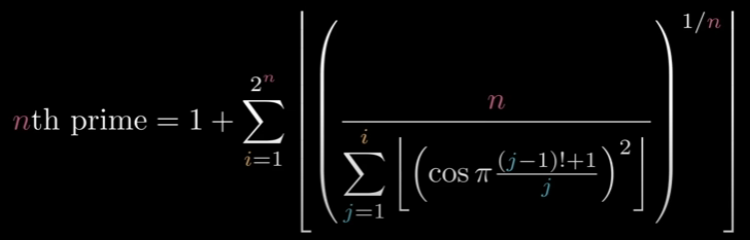
Contradicting the first output is the second with -1 being incorrect

We can concluded that the reason for this is because 888 inverse of mod 54 don’t have a gcd of 1

# Chapter 2 RSA cryptosystem

## *mathematical concepts* behind the RSA cryptosystem

### 2.1.1 prime number generation

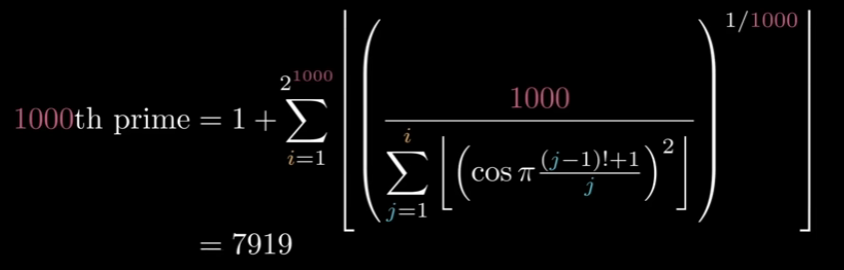


Calculating the nth prime number that was attributed to C.P. Willans. However, it should be noted that this formula is not widely known or used, and it is not considered to be an efficient or practical method for generating large prime numbers.

While this formula provides a way to calculate the nth prime number, it is not efficient for large values of n, as the term O(log log n) represents an error term that grows as n increases. Therefore, other methods such as sieving algorithms, prime generating polynomials, and probabilistic methods are used to generate large prime numbers in practice.

If we take a number that is 1 and insert it into the n position of the equation it will give us the first prime number

If we take a number that is 1000 and insert it into the n position of the equation it will give us the 1000 prime number



### 2.1.2 Extended Euclidean algorithm

Recall in chapter 1 that the Euclidean algorithm is a way to find the greatest common division,

This is done by repeatedly subtracting the two numbers , the subtraction will begin a loop So if we keep subtracting repeatedly the larger of two, we end up with GCD.

Let say that we will divide instead of subtracting we will get the smallest number , the algorithms will only stop when we hit a number with the value of 0

There exists integer coefficients x and y : ax + by =gcd(a,b)

Using the values calculated by the recursive call gcd(b%a, a), the extended Euclidean algorithm updates the results of gcd(a, b). Let x1 and y1 be the values of x and y calculated by the recursive call

Here is an example of how to find x and y with a= 888 and b= 54

6 = 54 - 24\*2

We implement the 24 = 888 -16\*54 Into the above equation

6 = 54 – (888 -16\*54)\*2 = 54 –2\*888 +32\*54= -2\*888 + 33\*54

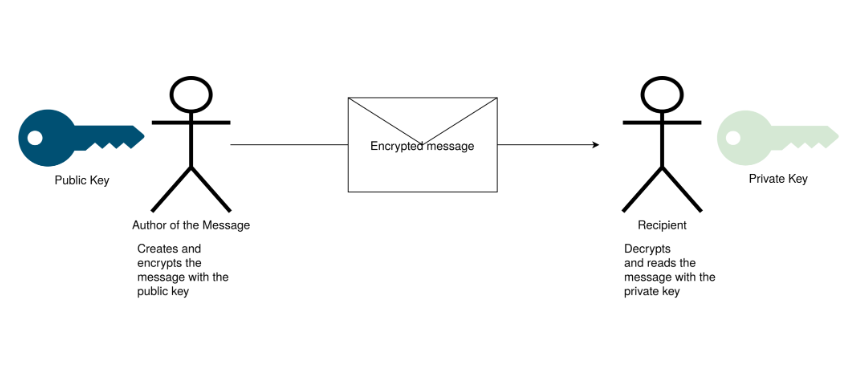
So in conclusion we will have the x and y being x =-2 and y = 33

Then if you can calculate the gcd by placing it into the algorithms -2\*888 + 33\*54 = 6

### prime number generator an extended Euclidean algorithm in RSA cryptosystem

there are two different types of method encryption symmetric encryption and asymmetric encryption

in the symmetric cases , there is only one type of key and the only measure of privacy and safety is that the key is share by only two people the person who send the message and the person who receive the message. Asymmetric encryption on the other hand have two different type one for encrypting and the other for decrypting or in other word one for public use and the other for private , the public key is uses for decrypting as the name imply it is public and can be look up in the key database the private key on the other hand is given out to the receiver for decryption and can only be used by the receiver .



This is how we can create private and public key:

Step 1: choose two random number prime numbers, let say p and q

Step 2: come up with two number that is the product of those numbers p\*q

Step 3: calculate the Phi function of both: Φ(N) = (p-1)\*(q-1)

Step 4: Choose a number e that both coprime to and smaller than Φ(N)

Step 5: Find d, find d by using this equation de(mod Φ(N)) =1

Example :

Let have p =2 and q =7 for our equation

The product of the two number will be N =14

Φ(N) = (p-1)\*(q-1) = 1 \*6

we will choose 5 for this example

Finding d , 5\*d (mod 6) = 1 there are a lot of option for d to satisfy this equation so in this example we will pick 11

So then we will have are public key being (5,14) and our private being (11,14)

Encrypting the messages:

Let say that we have a message call O it will be transform into an integer 9

We will be using the public and private key in the above example

Encryption (5,14)

Text : O -> 9

decrypting the messages:

Decryption(11,14)

So then we get our initial text being O

## 2.2 Implement a Python program to encrypt and decrypt a message with the RSA cryptosystem. Cryptography libraries are allowed.

import random

import math

from Crypto.Util import number

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def modinv(a, mod\_num):

    \_, x, \_ = extended\_gcd(a, mod\_num)

    return x%mod\_num

def extended\_gcd(a,b):

    if(b==0):

        return a, 1, 0

    else :

        gcd , x1 ,y1 = extended\_gcd(b , a%b)

        x = y1

        y  = x1 -(a//b)\*y1

        return gcd  , x , y

def is\_prime(num):

    if num == 2 or num == 3:

        return True

    if num < 2 or num % 2 == 0:

        return False

    for i in range(3, int(math.sqrt(num)) + 1, 2):

        if num % i == 0:

            return False

    return True

def generate\_keypair(p, q):

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = random.randint(2, phi)

    g = gcd(e, phi)

    while g != 1:

        e = random.randint(2, phi)

        g = gcd(e, phi)

    d = modinv(e, phi)

    return ((e, n), (d, n))

def encrypt(msg, public\_key):

    e, n = public\_key

    cipher = [pow(ord(char), e, n) for char in msg]

    return cipher

def decrypt(cipher, private\_key):

    d, n = private\_key

    msg = [chr(pow(char, d, n)) for char in cipher]

    return ''.join(msg)

if \_\_name\_\_ == '\_\_main\_\_':

    p = number.getPrime(32)

    q = number.getPrime(32)

    public\_key, private\_key = generate\_keypair(p,q)

    print(private\_key)

    print(public\_key)

    print("Public Key: ", public\_key)

    print("Private Key: ", private\_key)

    message = "POKEMON!"

    print("Original Message: ", message)

    encrypted\_message = encrypt(message, public\_key)

    print("Encrypted Message: ", encrypted\_message)

    decrypted\_message = decrypt(encrypted\_message, private\_key)

    print("Decrypted Message: ", decrypted\_message)

to start the first thing we need to do is to import 3 library that being :

import random for random number generation

import math for mathematic calculation

from Crypto.Util import number for crypto number manipulation

there are three main function to the program being generate key pair , decryption , encryption

generate key pair :

* the function will generate the three main key use for both encryption and decryption
* the variable d is calculated by the modinv function which call the extended extended euclidean algorithm function to calculate
* modinv will be giving the input being e as the number and phi as the mod

encryption :

* the encryption function will take in the message plus the public key then thought a loop will generate an array of number for each character in the word

decryption :

* the decryption function will take in the array of number given by the encryption algorithm and return a message

if \_\_name\_\_ == '\_\_main\_\_':

    p = number.getPrime(32)

    q = number.getPrime(32)

    public\_key, private\_key = generate\_keypair(p,q)

    print(private\_key)

    print(public\_key)

    print("Public Key: ", public\_key)

    print("Private Key: ", private\_key)

    message = "POKEMON!"

    print("Original Message: ", message)

    encrypted\_message = encrypt(message, public\_key)

    print("Encrypted Message: ", encrypted\_message)

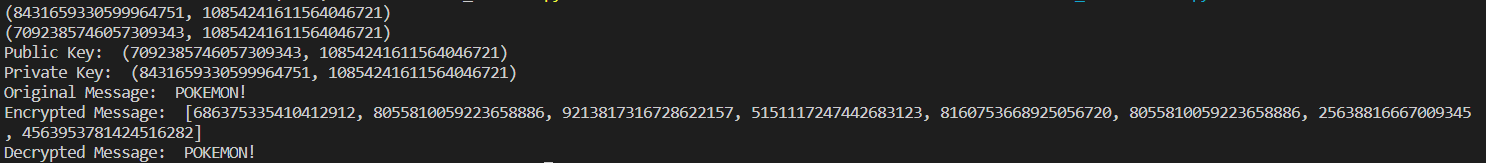
    decrypted\_message = decrypt(encrypted\_message, private\_key)

    print("Decrypted Message: ", decrypted\_message)

to run the program we first need to create to random prime number p and q

we will use the random prime function in the random library to do this

we can see that the message we need to decrypt is the word “POKEMON!”

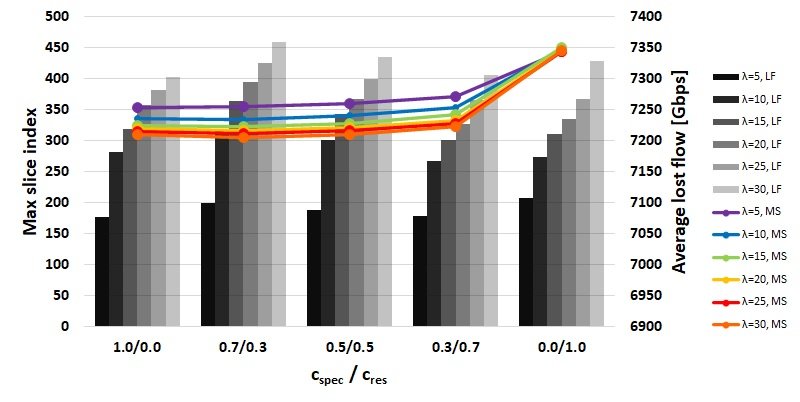


the encrypted message as we can see is an array of number for each character in the array and the output decrypted message will be POKEMON

## Analyze the efficiency and security of the implemented RSA cryptosystem.

The RSA cryptosystem is one of the most widely used public-key cryptosystems for securing data transmission over the internet. It relies on the difficulty of factoring large integers to provide security. In this answer, I will analyze the efficiency and security of the implemented RSA cryptosystem.

Efficiency: The efficiency of the RSA cryptosystem depends on the size of the keys used. A larger key size increases security but also increases the computational overhead. The key size also affects the size of the message that can be encrypted. Generally, a key size of 2048 bits or higher is recommended for secure encryption.



The RSA algorithm is computationally intensive, especially for large keys. The key generation process involves finding large prime numbers, which can be time-consuming. The encryption and decryption processes involve modular exponentiation, which can also be computationally expensive. However, with modern hardware and optimized algorithms, the RSA cryptosystem is still considered efficient for most practical purposes.

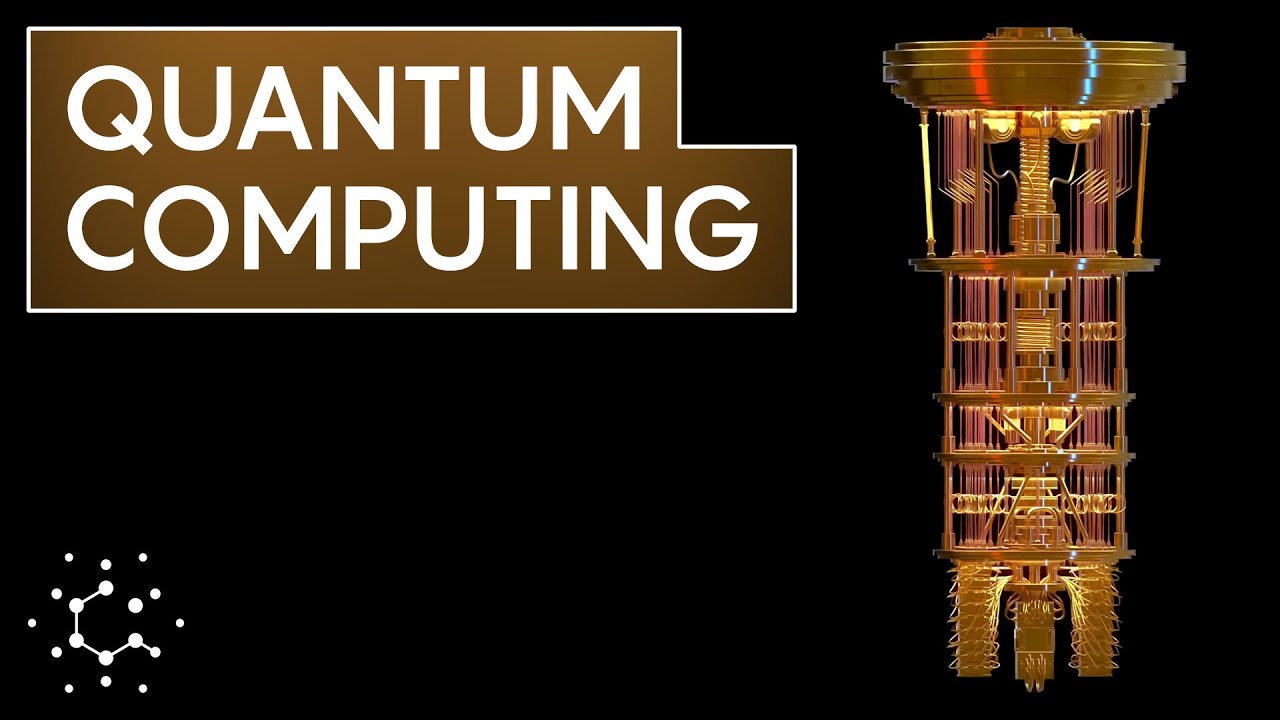
Security: The security of the RSA cryptosystem depends on the difficulty of factoring large integers. The security of RSA is based on the fact that it is computationally infeasible to factor the product of two large prime numbers.

To break the RSA cryptosystem, an attacker would need to factor the public key, which would allow them to calculate the private key and decrypt messages. However, with sufficiently large keys, factoring the public key is currently believed to be computationally infeasible for any practical purposes. Therefore, the RSA cryptosystem is considered secure against attacks from classical computers.



## potential security threats and limitations of the RSA cryptosystem.

with the advent of quantum computers, the security of the RSA cryptosystem is threatened. Quantum computers have the potential to break the RSA cryptosystem by factoring large numbers much faster than classical computers. Therefore, there is ongoing research into post-quantum cryptography, which aims to develop new cryptographic algorithms that are secure against attacks from both classical and quantum computers.



## 2.4 recommendations for improving the RSA cryptosystem implementation.

Use larger key sizes: Increasing the key size is a simple way to improve the security of RSA. A larger key size makes it more difficult to factor the public key and therefore makes it more secure against attacks. The recommended key size for RSA is 2048 bits or higher.

Use strong random number generators: The security of RSA relies on the randomness of the prime numbers used to generate the keys. Therefore, it is essential to use a strong random number generator to ensure the primes are truly random and not predictable.

Use secure key generation: The key generation process must be secure to prevent an attacker from predicting the private key. This can be achieved by using a secure random number generator and testing the generated primes for primality.

Use padding schemes: Padding schemes are used to add randomness to the message being encrypted, which improves the security of RSA. Padding schemes such as OAEP (Optimal Asymmetric Encryption Padding) and PKCS#1 v1.5 padding are commonly used in RSA implementations.

Use hardware acceleration: RSA computations can be computationally expensive, especially for large key sizes. Hardware acceleration, such as using dedicated hardware like ASICs (Application Specific Integrated Circuits), can significantly improve the efficiency of RSA.

Regularly update the implementation: RSA is a widely used cryptosystem, and new attacks and vulnerabilities are constantly being discovered. Therefore, it is crucial to regularly update the implementation with the latest security patches and improvements.

# TÀI LIỆU THAM KHẢO

**Tiếng Việt**

**Tiếng Anh**

1. <https://www.baeldung.com/cs/prime-numbers-cryptography>
2. <https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm>
3. <http://www-math.ucdenver.edu/~wcherowi/courses/m5410/exeucalg.html>

# PHỤ LỤC