

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**--------------------------------------------------------------**



**BÀI TẬP LỚN**

**MÔN: AN TOÀN VÀ BẢO MẬT THÔNG TIN**

**ĐỀ TÀI**

**XÂY DỰNG CHƯƠNG TRÌNH MÃ HOÁ VÀ GIẢI MÃ RSA**

|  |  |
| --- | --- |
| **GVHD:** | **Lê Thị Anh** |
| **Nhóm:** | **Nhóm 08** |
| **Thành viên nhóm:** | 1. **Trương Hoàng Sơn – 2021606050** 2. **Nguyễn Việt Cường – 2021601483** 3. **Vương Quốc Minh – 2021605642** 4. **Đỗ Quang Minh – 2021603233** |

*Hà Nội - Năm 2023*

**LỜI CẢM ƠN**

Đầu tiên, nhóm em xin gửi lời cảm ơn chân thành đến Trường Đại học Công nghiệp Hà Nội đã đưa môn học An toàn và bảo mật thông tin vào chương trình giảng dạy. Đặc biệt, nhóm em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn – Lê Thị Anh đã dạy dỗ, truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Trong thời gian tham gia lớp học An toàn và bảo mật thông tin của cô, nhóm em đã trang bị cho mình nhiều kĩ năng, kiến thức bổ ích, tinh thần học tập hiệu quả, nghiêm túc. Đây chắc chắn sẽ là những kiến thức quý báu, là hành trang để em có thể vững bước sau này.

Bộ môn An toàn và bảo mật thông tin là môn học thú vị, vô cùng bổ ích và có tính thực tế cao. Đảm bảo cung cấp đủ kiến thức, gắn liền với nhu cầu thực tiễn của sinh viên. Tuy nhiên, do vốn kiến thức còn nhiều hạn chế và khả năng tiếp thu thực tế còn nhiều bỡ ngỡ. Mặc dù nhóm em đã cố gắng hết sức nhưng chắc chắn bài tiểu luận khó có thể tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác, kính mong cô xem xét và góp ý để bài tiểu luận của nhóm em được hoàn thiện hơn.

Em xin chân thành cảm ơn!

**MỤC LỤC**

[LỜI MỞ ĐẦU 4](#_Toc151235064)

[CHƯƠNG I. TỔNG QUAN VỀ HỆ MÃ RSA 5](#_Toc151235065)

[**1.1. Giới thiệu về mật mã học** 5](#_Toc151235066)

[**1.2. Tổng quan về đề tài** 6](#_Toc151235067)

[**1.3. Ý tưởng chọn đề tài** 6](#_Toc151235068)

[**1.4. Nội dung nghiên cứu và kiến thức cần thiết** 7](#_Toc151235069)

[**1.5. Lĩnh vực nghiên cứu** 7](#_Toc151235070)

[CHƯƠNG 2. KẾT QUẢ NGHIÊN CỨU 8](#_Toc151235071)

[**2.1. Giới thiệu** 8](#_Toc151235072)

[**2.2. Nội dung thuật toán** 8](#_Toc151235073)

[2.2.1. Thuật toán Euclid 8](#_Toc151235074)

[2.2.2. Định lý Fermat 15](#_Toc151235075)

[2.2.3. Hàm số Euler 16](#_Toc151235076)

[2.2.4. Thuật toán Miller - Rabin: 18](#_Toc151235077)

[2.2.5. Định lý phần dư Trung Hoa 29](#_Toc151235078)

[2.2.6. Hệ mã hóa công khai RSA. 32](#_Toc151235079)

[**2.3. Thiết kế, cài đặt chương trình demo thuật toán** 35](#_Toc151235080)

[2.3.1. Mã hóa và giải mã RSA bằng Python 35](#_Toc151235081)

[**2.4. Thực hiện bài toán** 43](#_Toc151235082)

[2.4.1. Phân công công việc 43](#_Toc151235083)

[2.4.2. Trương Hoàng Sơn 44](#_Toc151235084)

[2.4.3. Đỗ Quang Minh 44](#_Toc151235085)

[2.4.4. Nguyễn Việt Cường 44](#_Toc151235086)

[2.4.5. Nguyễn Việt Cường 44](#_Toc151235087)

[2.4.6. Vương Quốc Minh 44](#_Toc151235088)

[CHƯƠNG 3: KIẾN THỨC LĨNH HỘI VÀ BÀI HỌC KINH NGHIỆM 45](#_Toc151235089)

[**3.1. Nội dung đã thực hiện** 45](#_Toc151235090)

[**3.2. Hướng phát triển** 46](#_Toc151235091)

[**3.3. Ứng dụng của RSA** 47](#_Toc151235092)

# LỜI MỞ ĐẦU

Ngày nay internet cùng với các dịch vụ phong phú của nó có khả năng cung cấp cho con người các phương tiện hết sức thuận tiện để trao đổi, tổ chức, tìm kiếm và cung cấp thông tin. Tuy nhiên, cũng như các phương thức truyền thống việc trao đổi, cung cấp thông tin điện tử trong nhiều lĩnh vực đòi hỏi tính bí mật, tính toàn vẹn, tính xác thực cũng như trách nhiệm về các thông tin được trao đổi. Bên cạnh đó, tốc độ xử lý của máy tính ngày càng được nâng cao, do đó cùng với sự trợ giúp của các máy tính tốc độ cao, khả năng tấn công các hệ thống thông tin có độ bảo mật kém rất dễ xảy ra. Chính vì vậy, với sự phát triển mạnh mẽ của Công nghệ thông tin, người ta không ngừng nghiên cứu các vấn đề bảo mật và an toàn thông tin để đảm bảo cho các hệ thống thông tin hoạt động an toàn.

Trong thực tế các hacker, các dạng virus luôn tấn công và là mối đe dọa của các nguồn tài nguyên thông tin. Những vấn đề đảm bảo an toàn thông tin trong các hệ thống máy tính là rất quan trọng.

Hiểu được điều đó, nhóm chúng em đã thực hiện xây dựng chương trình mã hóa và giải mã RSA. Từ những kiến thức đã được học cùng với việc nghiên cứu các tài liệu trên Internet, và nhất là nhờ sự hướng dẫn, chỉ bảo tận tình của cô Lê Thị Anh , nhóm chúng em đã hoàn thành đề tài này. Trong quá trình hoàn thành đề tài, chúng em còn gặp nhiều khó khăn, và sự hiểu biết của chúng em còn hạn hẹp nên kết quả vẫn còn nhiều thiếu sót.

Kính mong cô nhận xét và đóng góp ý kiến để nhóm có thể hoàn thiện được bài tập này.

# CHƯƠNG I. TỔNG QUAN VỀ HỆ MÃ RSA

## **1.1. Giới thiệu về mật mã học**

Mật mã học (Cryptography) là ngành khoa học nghiên cứu về việc đảm bảo an toàn thông tin. Mật mã học gắn liền với quá trình mã hóa nghĩa là chuyển đổi thông tin từ dạng “có thể hiểu được” thành dạng “không thể hiểu được” và ngược lại quá trình giải mã. Mật mã học giúp đảm bảo những tinh chất sau cho thông tin:

* Tính bí mật (confidentiatily): thông tin chỉ được tiết lộ cho những ai được phép.
* Tính toàn vẹn (integrity): thông tin không thể bị thay đổi mà không bị phát hiện
* Tính xác thực (authentication): người gửi (hoặc người nhận) có thể chứng minh đúng họ.
* Tính chống đối bỏ (non-repudiation): người gửi hoặc nhận sau này không thể chối bỏ việc đã gửi hoặc nhận thông tin.

Để thực hiện điều này chúng ta áp dụng các biện pháp xác thực và mã hóa. Và mật mã học là nghiên cứu về vấn đề mã hóa. Mã hóa dữ liệu là chuyển dữ liệu từ dạng này sang dạng khác hoặc sang dạng code mà chỉ có người có quyền truy cập vào khóa giải mã hoặc có mật khẩu mới có thể đọc được nó. Dữ liệu được mã hóa thường gọi là ciphertext, dữ liệu thông thường, không được mã hóa thì gọi là plaintext. Dữ liệu hoặc plaintext được mã hóa với một thuật toán mã hóa và một key mã hóa, tạo ra một ciphertext. Dữ liệu sau khi mã hóa chỉ có thể xem được dưới dạng ban đầu nếu giải mã với các key chính xác. Có hai loại mã hóa dữ liệu chính tồn tại: mã hóa bất đối xứng còn được gọi là mã hóa công khai, và mã hóa đối xứng. Thuật toán mã hóa bất đối xứng còn được gọi là mã hóa khóa công khai, sử dụng hai khóa khác nhau: khóa công khai và khóa riêng tư. Mục đích của việc mã hóa dữ liệu là bảo vệ dữ liệu số khi nó được lưu trữ trên các hệ thống máy tính và truyền qua Internet hay các mạng máy tính khác, là ngăn ngừa việc tấn công đánh cắp dữ liệu trái phép hoặc phòng ngừa việc mất mát dữ liệu khi bị tấn công vật lý như trộm đĩa cứng, máy tính xách tay hay thậm chí đột nhập vào hệ thống vẫn không thể xem được dữ liệu riêng tư, bí mật đã được bảo vệ bằng các thuật toán mã hóa mạnh mẽ.

## **1.2. Tổng quan về đề tài**

Mã hóa RSA chính là một thuật toán hay còn gọi là hệ mã hóa bất đối xứng có pháp vi ứng dụng rộng rãi và phổ biến, đặc biệt người ta sử dụng RSA rất nhiều ở công tác mã hóa, thiết lập chữ ký điện tử, với vai trò là một mã hóa khóa công khai. Bất cứ ai cũng có thể dùng khóa công khai để có thể mã hóa được nguồn dữ liệu muốn gửi đi thế nhưng để giải mã được dữ liệu gửi đi đó thì buộc phải có sự hỗ trợ của khóa bí mật.

Thuật toán RSA có hai [khóa](https://vi.wikipedia.org/wiki/Kh%C3%B3a_(m%E1%BA%ADt_m%C3%A3)): [khóa công khai](https://vi.wikipedia.org/wiki/M%E1%BA%ADt_m%C3%A3_h%C3%B3a_kh%C3%B3a_c%C3%B4ng_khai) (hay khóa công cộng) và [khóa bí mật](https://vi.wikipedia.org/w/index.php?title=Kh%C3%B3a_b%C3%AD_m%E1%BA%ADt&action=edit&redlink=1) (hay khóa cá nhân). Mỗi khóa là những số cố định sử dụng trong quá trình mã hóa và giải mã. Khóa công khai được công bố rộng rãi cho mọi người và được dùng để [mã hóa](https://vi.wikipedia.org/wiki/M%C3%A3_h%C3%B3a). Những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa bí mật tương ứng. Nói cách khác, mọi người đều có thể mã hóa nhưng chỉ có người biết khóa cá nhân (bí mật) mới có thể giải mã được.

Không giống với loại mã hóa có khóa đối xứng, loại khóa bí mật của RSA tuyệt đối không truyền được thông tin ra bên ngoài ngay cả khi có thiết bị nghe trộm thì nếu đối tượng xấu không có khóa bí mật cũng sẽ không thể giải mã được thông tin đó.

Như vậy rõ ràng với 2 tính năng mã hóa và giải mã tối ưu đến tuyệt đối ở một phương trình bất đối xứng như vậy cho nên giá trị của RSA vô cùng lớn, nó sẽ được sử dụng ở hầu hết mọi trường hợp cần bảo mật thông tin. Giao thức SSL hay TSL, HTTPS cùng với chứng chỉ điện tử đều sử dụng RSA.

## **1.3. Ý tưởng chọn đề tài**

Trong quá trình học tập bộ môn An toàn bảo mật thông tin, chúng em đã được học những thuật toán khác nhau, các cách để mã hóa và giải mã thông tin. Đề tài này chúng em được cô giáo phân công thực hiện, chúng em dựa trên cơ sở kiến thức đã học, cùng với đó là các nguồn khác nhau trên internet để hoàn thiện đề tài.

## **1.4. Nội dung nghiên cứu và kiến thức cần thiết**

Để hoàn thành đề tài này, chúng em đã vận dụng kiến thức của các thuật toán đã học và tìm hiểu về khóa công khai, khóa riêng tư, sinh khóa, mã hóa và giải mã RSA

Những kiến thức cần thiết để hoàn thành đề tài :

**+** Thuật toán Euclid

+ Định lý Fermat

+ Hàm số Euler

+ Thuật toán Miller-Rabin

+ Định lý phần dư trung hoa

**+** Kiến thức về phương pháp mã hóa RSA

+ Nắm vững các kiến thức về ngôn ngữ lập trình cơ bản python

## **1.5. Lĩnh vực nghiên cứu**

Chủ đề: RSA và có sản phẩm demo ứng dụng cụ thể sử dụng mã hóa RSA trong truyền tin (ví dụ chat mã hóa đầu cuối,…) bằng python

# CHƯƠNG 2. KẾT QUẢ NGHIÊN CỨU

## **2.1. Giới thiệu**

* Nhiệm vụ, công việc chính cần thực hiện
* Đề tài nghiên cứu: Xây dựng chương trình mã hóa và giải mã RSA
* Các bước thực hiện đề tài bao gồm:
* Nghiên cứu nội dung các thuật toán
* Thuật toán Euclid mở rộng
* Định lý Fermat
* Hàm số Euler
* Thuật toán Miller-Rabin
* Định lý phần dư Trung Hoa
* Tìm hiểu về hệ mã hóa công khai RSA
* Thiết kế và cài đặt chương trình demo thuật toán
* Hình thức sản phẩm: Sản phẩm ứng dụng
* Kết quả đạt được
* Nghiên cứu thuật toán ứng dụng
* Cài đặt demo thuật toán

## **2.2. Nội dung thuật toán**

### 2.2.1. Thuật toán Euclid

Giải thuật Euclid mở rộng được sử dụng để giải một phương trình vô định nguyên (còn được gọi laf phương trình Đi-ô-phăng) có dạng: ax + by = c

Trong đó a, b, c là các hệ số nguyên, x, y là các ẩn nhận giá trị nguyên. Điều kiện cần và đủ để phương trình này có nghiệm (nguyên) là  UCLN(a,b) là ước của c . Khẳng định này dựa trên một mệnh đề sau:

*Nếu* d = UCLN(a,b)*thì tồn tại các số nguyên x,y  sao cho ax + by = d*

Cơ sở lý thuyết của giải thuật

Giải thuật Euclid mở rộng kết hợp quá trình tìm ƯCLN *(a, b)* trong thuật toán Euclid với việc tìm một cặp số x, y thoả mãn phương trình Đi-ô-phăng. Giả sử cho hai số tự nhiên a, b, ngoài ra a>b>0. Đặt r0 = a , r1 = b, chia cho r0  được số dư r2 và thương số nguyên q1 . Nếu r2 = 0 thì dừng, nếu r2 khác không, chia r1  cho r2  được số dư r3,...Vì dãy các ri là giảm thực sự nên sau hữu hạn bước ta được số dư rm+2 = 0 .

r0 = q1 \* r1 + r2 , 0<r2<r\_{1}};

r1 = q2 \* r2 + r3, 0<r3<r2, 0< r3 < r2;

....;

rm-1 = qm\* rm + rm+1, 0 < rm+1 <rm

rm = qm+1 \* rm+1

trong đó số dư cuối cùng khác 0 là rm+1 = d. Bài toán đặt ra là tìm x, y sao cho

a \* x + b\* y = rm+1 (=d)

Để làm điều này, ta tìm x, y theo công thức truy hồi, nghĩa là sẽ tìm xi và yi sao cho:

a \* xi + b\* yi = rm+1 (=d) với i = 0,1,…

Ta có

a\*1+b\*0=a=r0 và a\*0+b\*1=b=r1 nghĩa là x0=1, x1=0 và y0=0, y1=1. (1)

Tổng quát, giả sử có

a\*xi+b\*y\_{i}=ri với i=0,1...

a\*xi+1+ b\*yi+1 = ri+1 với i=0,1...

Khi đó từ

ri = qi+1 \* ri+1 + ri+2

suy ra

ri - qi+1 \* ri+1 = ri+2

(a \* xi + b\* yi) - qi+1 \* (a\*xi+1+ b\*yi+1) = ri+2

a \* (xi - qi+1 \*xi+1) + b \* (yi - qi+1 \* yi+1) = ri+2

từ đó, có thể chọn

xi+2  = xi - qi+1 \*xi+1 (2)

yi+2 = yi - qi+1 \* yi+1 (3)

Khi i = m -1 ta có được xm+1 và ym+1 . Các công thức (1), (2), (3) là công thức truy hồi để tính x, y.

**Giải thuật**

{Thuật toán Euclide: a, b không đồng thời bằng 0, trả về gcd(a, b)}

function gcd(a, b);

begin

**while** b ≠ 0 **do**

begin

r:= a mod b; a:= b; b:= r;

end;

Result:= a;

end;

{Thuật toán Euclide mở rộng: a, b không đồng thời bằng 0, trả về cặp (x, y) sao cho a \* x + b \* y = gcd(a, b)

Về tư tưởng là ghép quá trình tính cặp số (x, y) vào trong vòng lặp chính của thuật toán Euclide.}

function Extended\_gcd(a, b);

begin

(xa, ya):= (1, 0);

(xb, yb):= (0, 1);

**while** b ≠ 0 **do**

begin

q:= a div b;

r:= a mod b; a:= b; b:= r; *//Đoạn này giống thuật toán Euclide.*

(xr, yr):= (xa, ya) - q \* (xb, yb); *//Hiểu là: (xr, yr):= (xa, ya) "mod" (xb, yb);*

(xa, ya):= (xb, yb);

(xb, yb):= (xr, yr);

end;

Result:= (xa, ya);

end;

Giải thuật sau chỉ thực hiện với các số nguyên a>b>0, biểu diễn bằng giải mã:

**Sub** Euclid\_Extended(a,b)

**Dim** x0, x, y,y1 **As** **Single**

x0=1: x1=0: y0=0: y1=1

**While** b>0

r= a **mod** b

**if** r=0 **then** **Exit** **While**

q= a / b

x= x0-x1\*q

y= y0-y1\*q

a=b

b=r

x0=x1

x1=x

y0=y1

y1=y

**Wend**

**Me**.Print d:=b, x, y

code

**End** **Sub**

**Ví dụ:** Với a=29, b=8, giải thuật trải qua các bước như sau:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước |  |  |  |  |  |  |  |  |  |  |
| 0 | 29 | 8 | 5 | 3 | 1 | 0 | 1 | 0 | 1 | -3 |
| 1 | 8 | 5 | 3 | 1 | 0 | 1 | -1 | 1 | -3 | 4 |
| 2 | 5 | 3 | 2 | 1 | 1 | -1 | 2 | -3 | 4 | -7 |
| 3 | 3 | 2 | 1 | 1 | -1 | 2 | -3 | 4 | -7 | 11 |
| 4 | 2 | 1 | 0 | 2 |  |  |  |  |  |  |

Kết quả thuật toán cho đồng thời d = UCLN(29,8) = 1 và x = -3, y=11  .  
Dễ dàng kiểm tra hệ thức  29\* (-3) + 8\*11 = 1

Áp dụng giải thuật Euclid mở rộng tìm số nghịch đảo trong vành  Zm

**Số nghịch đảo trong vành** Zm 

Trong lý thuyết số, vành Zm được định nghĩa là vành thương của Z với quan hệ đồng dư theo modulo m (là quan hệ tương đương) mà các phần tử của nó là các lớp đồng dư theo modulo m (m là số nguyên dương lớn hơn 1). Ta cũng có thể xét Zm  chỉ với các đại diện của nó. Khi đó

Zm = {0,1,..,m-1}

Phép cộng và nhân trong Zm  là phép toán thông thường được rút gọn theo modulo m:

a + b = (a + b) mod m

a \* b = (a \* b) mod m

Phần tử a của Zm  được gọi là khả nghịch trong Zm  hay khả nghịch theo modulo m nếu tồn tại phần tử a' trong Zm sao cho a\*a'=1 trong Zm  hay a\*a'=1 (mod m) . Khi đó a' được gọi là nghịch đảo modulo m của a. Trong lý thuyết số đã chứng minh rằng, số a là khả nghịch theo modulo m khi và chỉ khi ƯCLN của a và m bằng 1.  
Khi đó tồn tại các số nguyên x, y sao cho:

m\*x + a\*y = 1

Đẳng thức này lại chỉ ra y là nghịch đảo của a theo modulo m. Do đó có thể tìm được phần tử nghịch đảo của a theo modulo m nhờ thuật toán Euclid mở rộng khi chia m cho a.

**Giải thuật**

*//a, m > 0. Trả về a^-1 mod m, gcd(a, m) phải bằng 1, chú ý là ta không cần quan tâm y khi giải pt diophante a \* x + m \* y = 1*

**function** ModuloInverse(a, m);

**begin**

xa:= 1; xm:= 0;

**while** m ≠ 0 **do**

**begin**

q:= a **div** m;

xr:= xa - q \* xm;

xa:= xm;

xm:= xr;

r:= a **mod** m;

a:= m;

m:= r;

**end**;

Result:= xa;

**end**;

Giải thuật sau chỉ thực hiện với các số nguyên m>a>0, biểu diễn bằng giã mã:

**Procedure Euclid\_Extended (a,m)**

*int*, y0:=0,y1:=1;

**While** a>0 **do** {

r:= m mod a

if r=0 then Break

q:= m div a

y:= y0-y1\*q

y0:=y1

y1:=y

m:=a

a:=r

}

**If** a>1 **Then Return** "A không khả nghịch theo mođun m"

**else Return** " Nghịch đảo modulo m của a là y"

**Ví dụ**: Tìm số nghịch đảo (nếu có) của 30 theo môđun 101

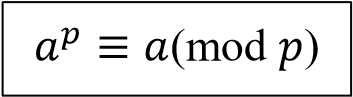
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bước i | **m** | **A** | **r** | **q** | **y0** | **y1** | **y** |
| 0 | **101** | **30** | 11 | 3 | 0 | 1 | -3 |
| 1 | 30 | 11 | 8 | 2 | 1 | -3 | 7 |
| 2 | 11 | 8 | 3 | 1 | -3 | 7 | -10 |
| 3 | 8 | 3 | 2 | 2 | 7 | -10 | 27 |
| 4 | 3 | 2 | 1 | 1 | -10 | 27 | **-37** |
| 5 | 2 | 1 | 0 | . | . | . | . |

Kết quả tính toán trong bảng cho ta -37 . Lấy số đối của 37 theo mođun 101 được 64. Vậy 30-1 mod 101 = 64.

**Ứng dụng**

Số nghịch đảo theo môđun được ứng dụng nhiều trong việc giải phương trình đồng dư, trong lý thuyết mật mã.

### 2.2.2. Định lý Fermat

“Nếu là một số nguyên dương và là một số nguyên tố thì “

Chứng minh 1. Sử dụng phương pháp quy nạp theo

Với = 1 thì mệnh đề luôn đúng.

Giả sử mệnh đề đúng đến a tức là p | ap - a .

Ta sẽ chứng minh mệnh đề đúng đến a + 1. Thật vậy:

(a+ 1)p − (a+ 1) = (ap - a ) + kp ak

Sử dụng p| Ckp với 1 ≤ k ≤p− 1 và giả thiết quy nạp ta suy ra

p |(a+ 1)p − ( a+ 1). Khi đó (a+ 1)p≡ (a+ 1) (mod p) .

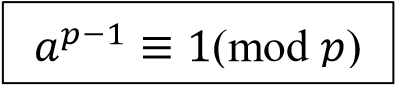
Vậy ta hoàn tất chứng minh.

Chứng minh 2. Giả sử rằng gcd(a, p ) =1 và cần chứng minh rằng ap-1-1≡ 1(mod p).

Xét các số nguyên a, 2a , … , ( p− 1)a mà các số dư khi chia cho p phân biệt ( nếu không thì, với ia≡ ja(mod p) thì p|( i−j )a hay là p|i − j, dấu “=” xảy ra chỉ nếu i=j ).

Do đó a. (2a) … (p− 1)a ≡ 12 … ( p− 1)(mod p) .

Vì gcd(q, (p− 1)!) = 1 nên ta suy ra điều phải chứng minh.

**Lưu ý.** Định lý này có thể biết gọn dưới dạng: 

### 2.2.3. Hàm số Euler

Cho số nguyên dương n, số lượng các số nguyên dương bé hơn n và nguyên tố cùng nhau với n được ký hiệu f(n) và gọi là hàm Euler.

Nhận xét: Nếu p là số nguyên tố, thì f(p) = p -1

Ví dụ: Tập các số nguyên không âm nhỏ hơn 7 là Z7 = {0, 1, 2, 3, 4, 5, 6}.

Do 7 là số nguyên tố, nên tập các số nguyên dương nhỏ hơn 7 và nguyên tố cùng nhau với 7 là Z7\* = {1, 2, 3, 4, 5, 6}.

Khi đó |Z| = f(p) = p - 1 = 7 - 1 = 6

\* Định lý về hàm Euler: Nếu n là tích của hai số nguyên tố p, q thì

f(n) = f(p) .f(q) = (p-1).(q-1)

\* Quan hệ Đồng dư

Khái niệm: Cho hai số nguyên a, b, m (m > 0). Ta nói rằng a và b “đồng dư” với nhau theo modulo m, nếu chia a và b cho m, ta nhận được cùng một số dư.

Ký hiệu: a º b (mod m).

Ví dụ: 17 º 5 (mod 3) vì chia 17 và 5 cho 3, được cùng số dư là 2.

Tính chất của quan hệ đồng dư:

- Với mọi số nguyên dương m ta có:

a ≡ a (mod m) với mọi a thuộc Z; (tính chất phản xạ)

a ≡ b (mod m) thì b ≡ a (mod m); (tính chất đối xứng)

a≡ b (mod m) và b≡c (mod m) thì a≡c (mod m); (tính chất bắc cầu)

- Tổng hay hiệu các đồng dư:

(a+b) (mod n) ≡ [(a mod n) + (b mod n)] (mod n)

(a-b) (mod n) ≡ [(a mod n) - (b mod n)] (mod n)

- Tích các đồng dư:

(a\*b) (mod n) ≡ [(a mod n) \* (b mod n)] (mod n)

\* Tập thặng dư thu gọn theo modulo

Khái niệm: Kí hiệu Z n = {0, 1, 2, ..., n-1} là tập các số nguyên không âm < n.

Với Z n và phép cộng (+) lập thành nhóm Cyclic có phần tử sinh là 1, phần tử trung lập e = 0, (Z n , +) gọi là nhóm cộng, đó là nhóm hữu hạn có cấp n.

Với phép (+) là phép cộng thông thường của các số nguyên.

Kí hiệu Z\*n = {x Î Z n , x là nguyên tố cùng nhau với n}. Tức là x phải ¹ 0.

Z\*n được gọi là Tập thặng dư thu gọn theo mod n, có số phần tử là f(n). Z\*n với phép nhân mod n lập thành một nhóm (nhóm nhân), phần tử trung lập e = 1. Tổng quát (Z\*n, phép nhân mod n) không phải là nhóm Cyclic. Nhóm nhân Z\*n là Cyclic chỉ khi n có dạng: 2, 4, p k hay 2p k với p là nguyên tố lẻ.

Ví dụ:Cho n = 21, Z\*n = {1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20}.

\* Độ phức tạp của thuật toán

Bài toán được diễn đạt bằng hai phần:

Đầu vào: Các dữ liệu vào của bài toán.

Đầu ra: Các dữ liệu ra của bài toán (kết quả).

Khái niệm Thuật toán: “Thuật toán” được hiểu đơn giản là cách thức để giải một bài toán. Cũng có thể được hiểu bằng hai quan niệm: trực giác hay hình thức như sau:

Một cách trực giác, Thuật toán được hiểu là một dãy hữu hạn các quy tắc (chỉ thị, mệnh lệnh) mô tả một quá trình tính toán, để từ dữ liệu đã cho đầu vào ta nhận được kết quả đầu ra của bài toán.

Một cách hình thức, người ta quan niệm thuật toán là một máy Turing. Thuật toán được chia thành hai loại: Đơn định và không đơn định. Chi phí thời gian của một quá trình tính toán là thời gian cần thiết để thực hiện một quá trình tính toán. Với thuật toán tựa Algol: Chi phí thời gian là số các phép tính cơ bản thực hiện trong quá trình tính toán.

Chi phí bộ nhớ của một quá trình tính toán là số ô nhớ cần thiết để thực hiện một quá trình tính toán. Gọi A là thuật toán, e là dữ liệu vào của bài toán đã được mã hóa bằng cách nào đó. Thuật toán A tính trên dữ liệu vào e phải trả một giá nhất định. Ta ký hiệu: tA(e) là giá thời gian và IA(e) là giá bộ nhớ.

### 2.2.4. Thuật toán Miller - Rabin:

#### \* Kiểm tra Miller-Rabin

Kiểm tra Miller-Rabin là một thuật toán xác suất để [kiểm tra tính nguyên tố](https://wiki.edu.vn/wiki/index.php?title=Ki%E1%BB%83m_tra_t%C3%ADnh_nguy%C3%AAn_t%E1%BB%91) cũng như các thuật toán kiểm tra tính nguyên tố: [Kiểm tra Fermat](https://wiki.edu.vn/wiki/index.php?title=Ki%E1%BB%83m_tra_Fermat) và [Kiểm tra Solovay-Strassen](https://wiki.edu.vn/wiki/index.php?title=Ki%E1%BB%83m_tra_Solovay-Strassen). Nó được đề xuất đầu tiên bởi [Gary L. Miller](https://wiki.edu.vn/wiki/index.php?title=Gary_L._Miller&action=edit&redlink=1) như một thuật toán tất định, dựa trên [giả thiết Riemann tổng quát](https://wiki.edu.vn/wiki/index.php?title=Gi%E1%BA%A3_thi%E1%BA%BFt_Riemann_t%E1%BB%95ng_qu%C3%A1t&action=edit&redlink=1); [Michael O. Rabin](https://wiki.edu.vn/wiki/index.php?title=Michael_O._Rabin&action=edit&redlink=1) đã sửa chữa nó thành một [thuật toán xác suất](https://wiki.edu.vn/wiki/index.php?title=Thu%E1%BA%ADt_to%C3%A1n_x%C3%A1c_su%E1%BA%A5t&action=edit&redlink=1).

Khi sử dụng kiểm tra Miller-Rabin chúng ta căn cứ vào một mệnh đề Q(p, a) đúng với các số nguyên tố p và mọi số tự nhiên <math>a \in A \subset \mathbb N</math> và kiểm tra xem chúng có đúng với số n muốn kiểm tra và một số <math>a \in A</math> được chọn ngẫu nhiên hay không? Nếu mệnh đề Q(n, a) không đúng, tất yếu n không phải là số nguyên tố, còn nếu Q(n,a) đúng, số n có thể là số nguyên tố với một xác suất nào đó. Khi tăng số lần thử, xác suất để n là số nguyên tố tăng lên.

\* Tiêu chuẩn kiểm tra Q(n, a):

\* Căn bậc hai của 1 trong <math>\mathbb{Z}\_p</math>

Trước hết là một bổ đề về căn bậc hai của đơn vị trong [trường hữu hạn](https://wiki.edu.vn/wiki/index.php?title=Tr%C6%B0%E1%BB%9Dng_h%E1%BB%AFu_h%E1%BA%A1n&action=edit&redlink=1)<math>\mathbb{Z}\_p</math>, trong đó *p* là số nguyên tố. Chắc chắn rằng 1 và -1 luôn là các căn bậc hai của 1 theo mođun *p*. Chúng là hai căn bậc hai duy nhất của 1. Thật vậy, giả sử rằng *x* là một căn bậc hai của 1 theo mođun *p*. Khi đó:

<math>x^2 \equiv 1\pmod{p}</math>

<math>x^2-1 \equiv 0\pmod{p}</math>

<math>(x - 1)(x + 1) \equiv 0\pmod{p}</math>

Từ đó, <math>x-1</math> hoặc <math>x+1</math> là chia hết cho *p*.

##### \* Tiêu chuẩn Miler-Rabin:

Bây giờ giả sử *p* là một số nguyên tố lẻ, khi đó *p* - 1 là số chẵn và ta có thể viết *p* − 1 dưới dạng <math>2^s \cdot m</math>, trong đó *s* là một số tự nhiên >=1 và *m* là số lẻ - Điều này nghĩa là ta rút hết các thừa số 2 khỏi *p* − 1. Lấy số *a* bất kỳ trong tập {*1,2,..,p-1*}. Xét dãy số <math>x\_k=a^{2^k.m}</math> với k=0,1,2,...,s. Khi đó <math>x\_k = (x\_{k-1})^2</math>, với k=1,2,...,s và <math>x\_s = a^{p-1}</math>

Từ định lý Fermat nhỏ:

<math>a^{p-1} \equiv 1\pmod{p}</math>

hay

<math>x\_s \equiv 1\pmod{p}</math>

hay

<math>{x\_{s-1}}^2 \equiv 1\pmod{p}</math>.

Do đó, hoặc <math>x\_{s-1}\equiv 1\pmod{p}</math> hoặc <math>x\_{s-1}\equiv -1\pmod{p}</math>.

Nếu <math>x\_{s-1}\equiv -1 \pmod p</math> ta dừng lại, còn nếu ngược lại ta tiếp tục với <math>x\_{s-2}</math>.

Sau một số hữu hạn bước

· hoặc ta có một chỉ số k, <math> 0 \le k \le s-1</math> sao cho <math>x\_{k} \equiv -1 \pmod{p}</math>,

· hoặc tới k=0 ta vẫn có <math>x\_{k} \equiv 1 \pmod{p}</math>.

Ta có mệnh đề Q(p,a) như sau:

*Nếu p là số nguyên tố lẻ và p* - 1 = <math>2^s \cdot m</math> thì với mọi a: 0<a<p-1:

* *hoặc <math>x\_k = a^{2^k \cdot m} \equiv 1 \pmod p</math>, với mọi k=0,1,2,...,s*
* *hoặc tồn tai k: <math>0 \le k \le s</math> sao cho <math>x\_k=a^{2^k \cdot m} \equiv -1 \pmod p</math>*.

\* Số giả nguyên tố

· Theo định lý Fermat nhỏ, với số nguyên tố *p* ta có với mọi a <math>\in</math> {1,2,...,p-1}:

<math> a^{p-1} \equiv 1 \pmod p</math>

· **Định nghĩa**. Hợp số *n* thoả mãn <math> a^{n-1} \equiv 1 \pmod n</math> với a nào đó được gọi là số giả nguyên tố Fermat cơ sở a.

· Số Carmichael: Hợp số *n* là số giả nguyên tố Fermat với mọi cơ sở a <math>\in</math> {1…n}, ƯCLN(a,n)=1 được gọi là số Carmichael.

· **Định nghĩa**: Hợp số *n* được gọi là số giả nguyên tố mạnh Fermat cơ sở *a* nếu nó thoả mãn mệnh đề Q(n,a).

\* Giải thuật kiểm tra Miller-Rabin

Miller-Rabin test

INPUT Số tự nhiên lẻ *n*.

OUPTUT NguyenTo: TRUE/FALSE

1. Phân tích <math>n-1 = 2^s \cdot m</math> trong đó s <math>\ge</math> 1 và m là số tự nhiên lẻ

2. Chọn ngẫu nhiên số tự nhiên a <math>\in</math> {2,...,n-1}.

3. Đặt <math>b=a^m \pmod n</math>

4. Nếu<math> b \equiv -1 \pmod n</math> thì trả về TRUE. Kết thúc.

5. Cho k chạy từ 0 đến s:

1. Nếu <math>b \equiv -1 \pmod n</math> thì trả về TRUE. Kết thúc.

2. Thay b:=<math> b^2 \pmod n</math>.

6. Trả lời FALSE. Kết thúc.

Xác suất trả lời sai

· *Định lý*: nếu n là hợp số dương lẻ thì trong các số a <math>\in</math> {2...n-1} tồn tại không quá <math>\frac {n-1} {4}</math> cơ sở a để n là số giả nguyên tố mạnh Fermat.

· Gọi A là biến cố "*Số n là hợp số*". B là biến cố "*Kiểm tra Miller-Rabin trả lời n là số nguyên tố*". Khi đó xác suất sai của kiểm tra này là xác suất để số n là hợp số trong khi thuật toán cho câu trả lời TRUE, nghĩa là xác suất điều kiện P(A|B).

Theo định lý trên nếu n là hợp số thì khả năng kiểm tra này trả lời TRUE xảy ra với xác suất không vượt quá <math>\frac 1 4</math>, nghĩa là P(B|A)<math>\le \frac 1 4</math>. Tuy nhiên để tính xác suất sai của kiểm tra Miller-Rbin cần tính xác suất diều kiện P(A|B). Dựa trên định lý về ước lượng số các số nguyên tố ta đưa ra ước lượng

<math>P(A)\approx 1 - \frac 2 {\ln n} \approx \frac {\ln n-2} {\ln n}</math>

Theo định lý Bayes trong lý thuyết xác suất ta có công thức để tính xác suất sai của kiểm tra Miller-Rabin là:

P(A|B) = <math>\frac {P(B|A)\*P(A)}{P(B)}</math>

=<math>\frac {P(B|A)\*P(A)}{P(B|A)P(A)+P(B|\overline A)\*P(\overline A)}</math>

Trong công thức này P(A) đã biết ở trên, P(B|A) <math>\le \frac 1 4</math>, còn <math>P(B|\overline A)</math>= 1 vì khi n là số nguyên tố thì chắc chắn mệnh đề Q(n,a) là đúng và <math>P(\overline A)= 1- P(A)= \frac 2 {\ln n}</math>. Từ đó

P(A|B)=<math>\frac {P(B|A)\*P(A)}{P(B|A)P(A)+P(\overline A)}</math>

<math>P(A|B)\approx \frac {P(B|A)\*(\ln n-2)}{P(B|A)\*(\ln n-2)+2}</math>

(Tham khảo: Douglas R. Stisnon. Cryptography Theory and Practice.)

Kiểm tra Miller-Rabin lặp

Theo công thức tính xác suất sai trên đây, với n lớn (cỡ 130 chữ số thập phân), nếu thực hiện phép thử Miller-Rabin chỉ một lần, xác suất sai là khá lớn, tới trên 90%. Để giảm xác suất sai, ta lặp lại phép thử k lần với k số ngẫu nhiên a khác nhau, nếu n vượt qua 50 lần thử thì P(B|A)<math> \le \frac 1 {4^k}</math>, khi thay vào công thức với 50 lần thử nếu cả 50 lần, phép thử đều *"dương tính"* thì xác suất sai giảm xưống chỉ còn là một số rất nhỏ không vượt quá <math>9 \cdot 10^{-29}</math>.

Sử dụng Miller trong thực tếmột cách tổng quát thì thuật toán này chưa được chứng minh là đúng. Tính đúng đắn của nó phụ thuộc vào giả thuyết Reimann.

Tuy nhiên, với nn nhỏ (n<3,317,044,064,679,887,385,961,981n<3,317,044,064,679,887,385,961,981) thì thuật toán đã được kiểm tra và chứng minh là đúng. Ngoài ra, ta cũng không cần kiểm tra hết tất các các số nguyên aa như ở trên, mà chỉ cần dùng một vài số là đủ.

Bảng sau được sử dụng để kiểm tra số nguyên tố trong thực tế:

|  |  |
| --- | --- |
| **Giới hạn của** nn | **Các số** aa **cần dùng** |
| n<2,047n<2,047 | 2 |
| n<1,373,653n<1,373,653 | 2, 3 |
| n<9,080,191n<9,080,191 | 31, 73 |
| n<4,759,123,141n<4,759,123,141 | 2, 7, 61 |
| n<1,122,004,669,633n<1,122,004,669,633 | 2, 13, 23, 1662803 |
| n<2,152,302,898,747n<2,152,302,898,747 | 2, 3, 5, 7, 11 |
| n<3,474,749,660,383n<3,474,749,660,383 | 2, 3, 5, 7, 11, 13 |
| n<341,550,071,728,321n<341,550,071,728,321 | 2, 3, 5, 7, 11, 13, 17 |
| n<3,825,123,056,546,413,051n<3,825,123,056,546,413,051 | 2, 3, 5, 7, 11, 13, 17, 19, 23 |
| n<318,665,857,834,031,151,167,461n<318,665,857,834,031,151,167,461 | 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 |
| n<3,317,044,064,679,887,385,961,981n<3,317,044,064,679,887,385,961,981 | 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 |

Xem bảng trên, ta thấy chỉ cần dùng 3 số thử: 2, 7, 61 là đủ để kiểm tra tính nguyên tố của mọi số nguyên 32-bit.

Để hiểu rõ hơn cách dùng, bạn đọc xem thêm phần dưới.

Cài đặt thuật toán

Trong phần này mình sẽ dùng Miller để giải 2 bài [PNUMBER](http://vn.spoj.com/problems/PNUMBER/) và [PRIME1](http://www.spoj.com/problems/PRIME1/).

Thuật toán gồm các giai đoạn chính:

* Phân tích n−1n−1 thành dạng 2s×d2s×d.
* Thử lần lượt các số aa trong bảng ở trên.
* Khi thử, ta cần dùng hàm lũy thừa nhanh để tính admodnadmodn.

Ta sẽ lần lượt cài đặt các hàm cơ bản trên. Trước hết, ta khai báo các thư viện và kiểu dữ liệu cần sử dụng:

#include <tuple>

#include <iostream>

using namespace std;

typedef unsigned long long ll;

Hàm phân tích thành dạng 2s×d2s×d:

pair<ll, ll> factor(ll n) {

ll s = 0;

while ((n & 1) == 0) {

s++;

n >>= 1;

}

return {s, n};

}

Hàm tính admodnadmodn:

ll pow(ll a, ll d, ll n) {

ll result = 1;

a = a % n;

while (d > 0) {

if (d & 1) result = result \* a % n;

d >>= 1;

a = a \* a % n;

}

return result;

}

Hàm kiểm tra xem “nếu ad≢1(modn)ad≢1(modn) và (ad)2r≢−1(modn)(ad)2r≢−1(modn) với mọi rr từ 00 đến s−1s−1 thì nn **không phải là số nguyên tố**”:

// Trả về false nếu n không phải là số nguyên tố.

bool test\_a(ll s, ll d, ll n, ll a) {

if (n == a) return true;

ll p = pow(a, d, n);

if (p == 1) return true;

for (; s > 0; s--) {

if (p == n-1) return true;

p = p \* p % n;

}

return false;

}

Hàm cuối cùng: Thử kiểm tra nn với các aa khác nhau. Vì ta đang giải bài [PNUMBER](http://vn.spoj.com/problems/PNUMBER/), giới hạn số trong bài này là n<200000n<200000, tra bảng trên ta thấy chỉ cần kiểm tra với a=2a=2 và a=3a=3 là đủ:

// Trả về true nếu n là số nguyên tố,

// false nếu n là hợp số.

bool miller(ll n) {

if (n < 2) return false;

if ((n & 1) == 0) return n == 2;

ll s, d;

tie(s, d) = factor(n-1);

return test\_a(s, d, n, 2) && test\_a(s, d, n, 3);

}

Giới hạn của bài PRIME1 là n≤109n≤109, nên ta cần dùng a=2a=2, a=7a=7 và a=61a=61.

Xử lý tràn số

Bảng ở trên có thể kiểm tra đến các số lớn hơn 64-bit, tuy nhiên khi cài đặt, nếu không sử dụng kiểu số nguyên lớn thì ta chỉ có thể sử dụng thuật toán với các số 32-bit.

Lý do là thuật toán có dùng phép nhân. Khi nhân 2 số 32-bit kết quả phải được chứa trong số 64-bit, nhân 2 số 64-bit thì kết quả phải được chứa trong số 128-bit. Vì vậy, để không bị tràn số thì thuật toán chỉ có thể dùng với số 32-bit.

Để khắc phục điều này, ta có thể tự viết hàm để vừa thực hiện nhân vừa thực hiện mod.

Đây là code tính a×b mod n a×b mod n của Dana Jacobsen trong một [câu trả lời](https://www.quora.com/How-can-I-execute-A-*-B-mod-C-without-overflow-if-A-and-B-are-lesser-than-C) trên Quora:

#include <stdint.h>

uint64\_t mulmod(uint64\_t a, uint64\_t b, uint64\_t n) {

// if (a >= n) a %= n; /\* Careful attention from the caller \*/

// if (b >= n) b %= n; /\* should make these unnecessary. \*/

uint64\_t r = 0;

if ((a|b) < (1ULL << 32)) return (a\*b) % n;

if (a < b) { uint64\_t t = a; a = b; b = t; }

if (n <= (1ULL << 63)) {

while (b > 0) {

if (b & 1) { r += a; if (r >= n) r -= n; }

b >>= 1;

if (b) { a += a; if (a >= n) a -= n; }

}

} else {

while (b > 0) {

if (b & 1) r = ((n-r) > a) ? r+a : r+a-n; /\* r = (r + a) % n \*/

b >>= 1;

if (b) a = ((n-a) > a) ? a+a : a+a-n; /\* a = (a + a) % n \*/

}

}

return r;

}

Đoạn code trên hơi phức tạp, bạn đọc có thể đọc đoạn code sau để hiểu tư tưởng thuật toán:

uint64\_t mulmod(uint64\_t a, uint64\_t b, uint64\_t n) {

uint64\_t r = 0;

while (b > 0) {

if (b % 2 != 0) r = (r + a) % n;

a = a \* 2 % n;

b = b / 2;

}

return r;

}

Đoạn code ở dưới đơn giản hơn, tuy nhiên nó chạy chậm hơn và chỉ chạy đúng vớn n có tối đa 63 bit.

Thay thế các phép nhân trong code Miller ở phần trên bằng hàm mulmod, ta có thể sử dụng được Miller cho các số 64-bit.

Độ phức tạp

Độ phức tạp của hàm mulmod là O(logn)O(log⁡n).

Mỗi lần kiểm tra tốn O(s+logd)=O(logn)O(s+log⁡d)=O(log⁡n) phép nhân, kết hợp với phép ĐPT của phép nhân nữa là O(log2n)O(log2⁡n).

Vậy độ phức tạp của Miller là O(klog2n)O(klog2⁡n) với kk là số các số aa cần kiểm tra.

### 2.2.5. Định lý phần dư Trung Hoa

Trong nhiều trường hợp ta muốn tìm cách để tăng tốc độ tính toán Modulo. Các phép toán trên modulo các số nhỏ tính nhanh nhiều so với các số lớn. Chính vì vậy nếu số lớn phân tích được thành tích của các số nhỏ, từng cặp nguyên tố cùng nhau, thì ta sẽ có cách tính hiệu quả nhờ vào định lý Phần dư Trung hoa. Tính toán trên modulo của một tích các số mod M với M= m1m2.mk , trong đó GCD(m, m) = 1, với mọii khác j. Định lý phần dư Trung Hoa cho phép làm việc trên từng modulo m riêng biệt. Vì thời gian tính toán các phép toán trên modulo tỷ lệ với kích thước của số lấy modulo nên điều đó sẽ nhanh hơn tính toán trên toàn bộ M.

\*Có thể triển khai Định lý Trung Hoa theo một số cách như sau:

Tính toán theo modulo số lớn. Để tính A mod M, với M khá lớn và A là biểu thức số học nào đó. Trước hết ta cần tính tất cả a = A mod mị. Sau đó sử dụng công thức

A-(40) mod M

trong đó

M= M/m

4 = M;x(M' mod m) for Isisk

Ví dụ: Tính 17 mod 77. Áp dụng định lý phần dư Trung họa, ta coi A = 178, m = 7, m = 11. Khi đó M = 11, M2 = 7 và 11- mod 7 = 4 mod 7= 2, suy ra c = 11\*2 = 22 7' mod 11 = 8, suy ra C2 = 7\*8 = 56

ay = 178 mod 7 = (17 mod 7) mod 7 = 38 mod 7 = (32)4 mod 7 = 2 az= 178 mod 11 = (17 mod 11) mod 11 = 68 mod 11 =

= (6)\* mod 11 = 34 mod 11 = 4 Vây A = 178 mod 77 = (2\*22 + 4\*56) mod 77 = 268 mod 77 = 37 mod 37

\*Giải hệ phương trình modulo. Cho a = x mod mị, với GCDm, m) = 1, với mọi 1

khác j. Khi đó ta cũng áp dụng Định lý phần dư Trung Hoa để tìm x. Ví dụ. Cho x = 5 mod 7 và x = 6 mod 11. Tìm x.

Áp dụng định lý phần dư Trung hoa, ta tính: 7'mod 11 = 8 và 11 mod 7 = 2. Như vậy x = (5\*2\*11+6\*8\*7) mod (7\*11) = 61 mod 77.

Một trong những kết quả hữu ích nhất của lý thuyết số là định lý còn lại của Trung Hoa (CRT). Về bản chất (CRT) nói rằng: Có thể tìm lại các số nguyên trong một phạm vi nhất định từ dư lượng của chúng theo mod có một bộ các module tương đối quan trọng theo cặp.

Mười số nguyên trong Z10, đó là số nguyên từ 0 đến 9. Có thể được khôi phục lại từ 2 dư lượng mod 2 và mod 5 là các phần tử tương đối nguyên tố của 10. Nói các dư lượng tồn tại của một chữ số thập phân x là r2 = 0 và r5 = 3.

Đó là: x mod 2 = 0 và x mod 5 = 3. Vì vậy x là số nguyên dương trong Z10, chia cho 5 dư 3, suy ra x = 8

CRT có thể được thể hiện bằng nhiều cách. Dưới đây là một công thức hữu ích khác: M = 𝛱𝑘 𝑚𝑖 trong đó: 𝑚𝑖 là cặp tương đối nguyên tố gcd(𝑚𝑖, 𝑚𝑗 )=1 với 1 ≤ i, j ≤ k và i ≠ j. Ta có thể biểu diễn bất kì số nguyên A trong ZM bởi 1 bộ k có các phần tử trong 𝑍𝑚𝑖 sử dụng tương ứng sau:

CRT có thể được thể hiện bằng nhiều cách. Dưới đây là một công thức hữu ích khác: M = 𝛱𝑘 trong đó: 𝑚𝑖 là cặp tương đối nguyên tố gcd(𝑚𝑖, 𝑚𝑗 )=1 với 1 ≤ i, j ≤ k và i ≠ j. Ta có thể biểu diễn bất kì số nguyên A trong ZM bởi 1 bộ k có các phần tử trong 𝑍𝑚𝑖 sử dụng tương ứng sau:

A ↔ (a1, a2, ..., ak)

Trong đó: A ϵ ZM ; 𝑎𝑖 ϵ 𝑍𝑚𝑖 và 𝑎𝑖 = A mod 𝑚𝑖 với 1 ≤ i ≤ k

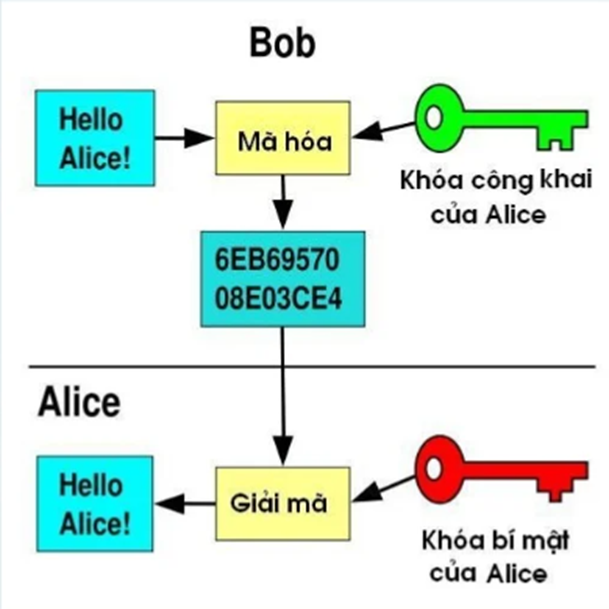
- Hai khẳng định của CRT như sau:

1. Lập bản đồ của phương trình (2.11) là một sự tương thích một – một giữa ZM và sản phẩm Cartesia 𝑍𝑚1 × 𝑍𝑚2 × × 𝑍𝑚𝑘. Nghĩa là đối với mỗi số nguyên A sao cho 0 ≤ A ≤ M, có 1 bộ k (a1, a2, ..., ak) với 0 ≤ 𝑎𝑖≤ 𝑚𝑖 đại diên cho nó và đối với mỗi bộ (a1, a2, , ak) có 1 số nguyên độc nhất A trong ZM

2. Các hoạt động thực hiện trên các thành phần của ZM có thể được thực hiện tương đương trên bộ k tương ứng bằng cách thực hiện các hoạt động độc lập trong mỗi vị trí ngang nhau trong hệ thống thích hợp.

### 2.2.6. Hệ mã hóa công khai RSA.

* Tổng quan về hệ mã công khai



*Sơ đồ của hệ mã công khai*

Hệ mã công khai sử dụng hai khóa có quan hệ toán học với nhau, tức là một khóa này được hình thành từ khóa kia: Người muốn nhận bản mã (Alice) tạo ra một khóa mật (private key) và từ khóa mật tính ra khóa công khai (public key) với một thủ tục không phức tạp, còn việc tìm khóa mật khi biết khóa công khai là bài toán khó giải được. Khóa công khai sẽ đưa đến cho người gửi bản tin (Bob) qua kênh công cộng. Và bản tin được Bob mã hóa bằng khóa công cộng. Bản mã truyền đến Alice, và nó được giải mã bằng khóa mật.

* Giới thiệu chung về hệ mã khóa công khai RSA

- Thuật toán được Ron Rivest, Adi Shamir và Len Adleman (R.S.A) mô tả lần đầu tiên vào năm 1977.

- Trước đó, vào năm 1973, Clifford Cocks – một nhà toán học người Anh đã mô tả một thuật toán tương tự.

- Nhưng tại thời điểm đó thì thuật toán này không khả thi và chưa bao giờ được thực nghiệm.

Mã hóa RSA chính là một thuật toán hay còn gọi là hệ mã hóa bất đối xứng có phạm vi ứng dụng rộng rãi và phổ biến trong công tác mã hóa và công nghệ chữ ký điện tử. Trong hệ mã hóa này, public key có thể chia sẻ công khai cho tất cả mọi người. Hoạt động gửi và nhận sẽ được can thiệp bởi RSA vì bản thân nó chứa hai khóa là khóa công khai và khóa bí mật để làm 2 nhiệm vụ bất đối xứng là mã hóa và giải mã. Không giống với loại mã hóa có khóa đối xứng, loại khóa bí mật của RSA tuyệt đối không được truyền thông tin ra bên ngoài ngay cả khi có thiết bị nghe trộm thì nếu đối tượng xấu không có khóa bí mật cũng sẽ không thể giải mã được thông tin đó. Như vậy rõ ràng với 2 tính năng mã hóa và giải mã tối ưu đến tuyệt đối ở một phương trình bất đối xứng như vậy cho nên giái trị của RSA vô cùng lớn, nó sẽ được sử dụng ở hầu hết mọi trường hợp cần bảo mật thông tin.

RSA được sử dụng rộng rãi Thuật toán mã hóa RSA thỏa mãn 5 yêu cầu của một hệ mã hiện đại:

+ Hệ bảo mật cao

+ Thao tác nhanh

+ Dùng chung được

+ Có ứng dụng rộng rãi

+ Có thể dùng xác định chủ nhân (dùng làm chữ ký điện tử)

Vì vậy, RSA được sử dụng rộng rãi trong công tác mã hóa và công nghệ chữ ký điện tử: truyền dẫn quỹ điện tử chuyển đổi thư điện tử, giao dịch tiền điện tử, thương mại điện tử tài chính- ngân hàng...

* Mô tả hoạt động

- Thuật toán RSA có hai khóa:

+ Khóa công khai (Public key): được công bố rộng rãi cho mọi người và được dùng để mã hóa.

+ Khóa bí mật (Private key): những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa bí mật tương ứng.

- Tạo khóa: Thực thi ở đầu người nhận

* Bước 1: Chọn ngẫu nhiên 2 số nguyên tố p, q
* Bước 2: Tính số làm modul: n=p\*q, ∅ₙ = (p-1)\*(q-1)
* Bước 3: Chọn ngẫu nhiên khóa mã hóa b (1<b<∅ₙ) sao cho GCD(b,∅ₙ)=1
* Bước 4: Giải phương trình sau để tìm khóa giải mã a:

a = b⁻¹ mod ∅ₙ (theo thuật toán Euclide mở rộng)

b\*a=1 mod ∅ₙ với 0 ≤ a ≤ ∅ₙ

* Bước 5: Kpub = {b ,n} , Kpr = {a, p ,q}

- Mã hóa:

y = eKpub(x) = xb mod n

x Zn = {0,1…..n-1}

- Giải mã:

x = dKpr(y) = ya mod n

* Độ an toàn mã hóa RSA

- Độ an toàn của hệ thống RSA dựa trên hai vấn đề: bài toán phân tích ra thừa số nguyên tố các số nguyên lớn và bài toán RSA.

- Vì vậy, muốn xây dựng hệ RSA an toàn thì n=p\*q phải là một số đủ lớn, để không có khả năng phân tích nó về mặt tính toán. Để đảm bảo an toàn nên chọn các số nguyên tố p và q từ 100 chữ số trở lên.

|  |  |
| --- | --- |
| Số các chữ số trong số được phân tích | Thời gian phân tích |
| 50 | 4 giờ |
| 75 | 104 giờ |
| 100 | 74 năm |
| 200 | 4000 năm |
| 300 | 500000 năm |
| 500 | 4 x 10 ^ 25 năm |

Bảng thời gian phân tích mã RSA

- Cách thức phân phối khóa công khai là một trong những yếu tố quyết định đối với độ an toàn của RSA.

- Vấn đề này nảy sinh ra một lỗ hổng gọi là Man-in-the-middle attack (tấn công vào giữa).

+ Khi A và B trao đổi thông tin thì C có thể gửi cho A một khóa bất kì để A tin rằng đó là khóa công khai của B gửi.

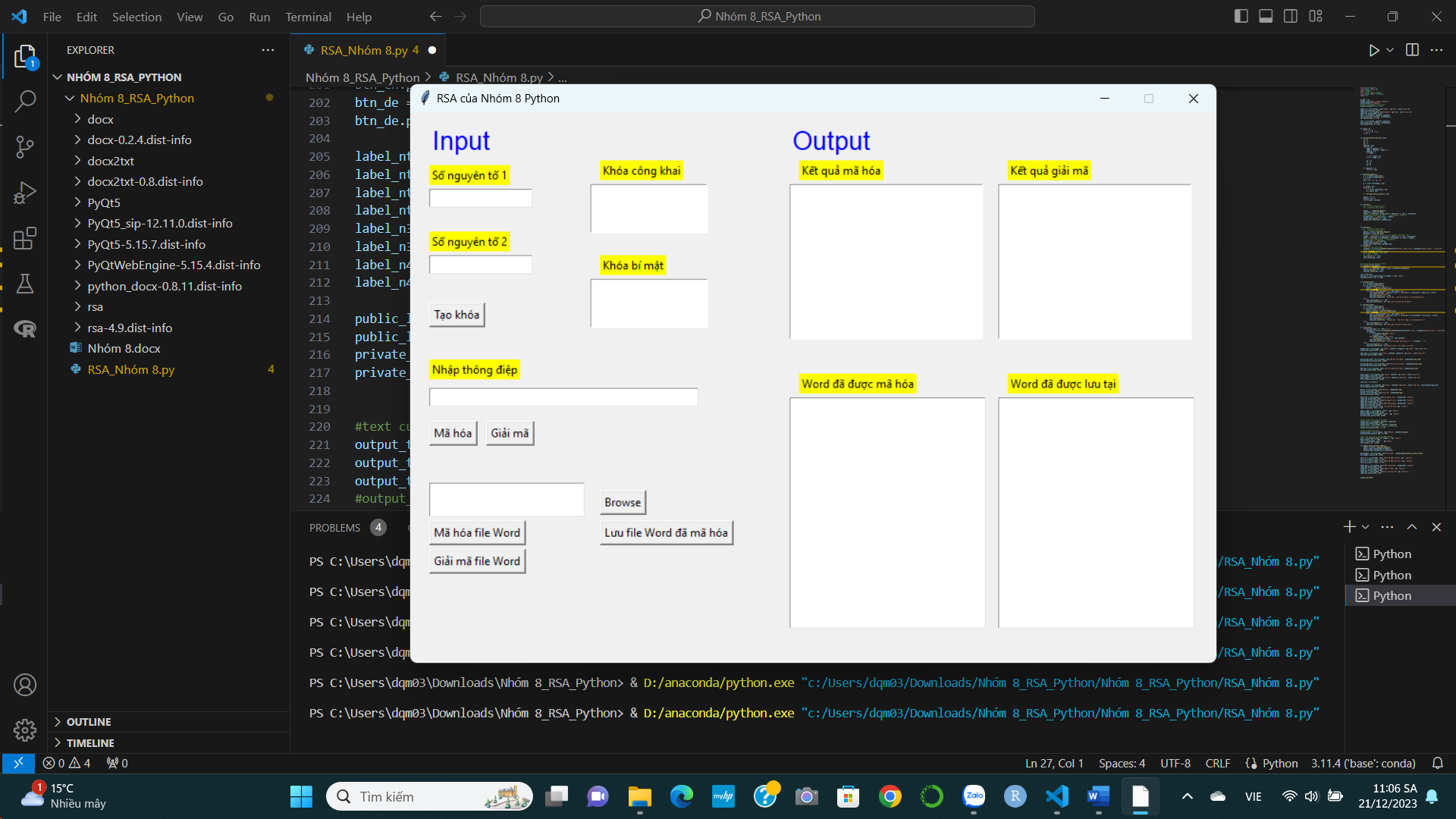
+ Sau đó C sẽ giải mã và đánh cắp được thông tin. Đồng thời mã hóa lại thông tin theo khóa công khai của B và gửi lại cho B.

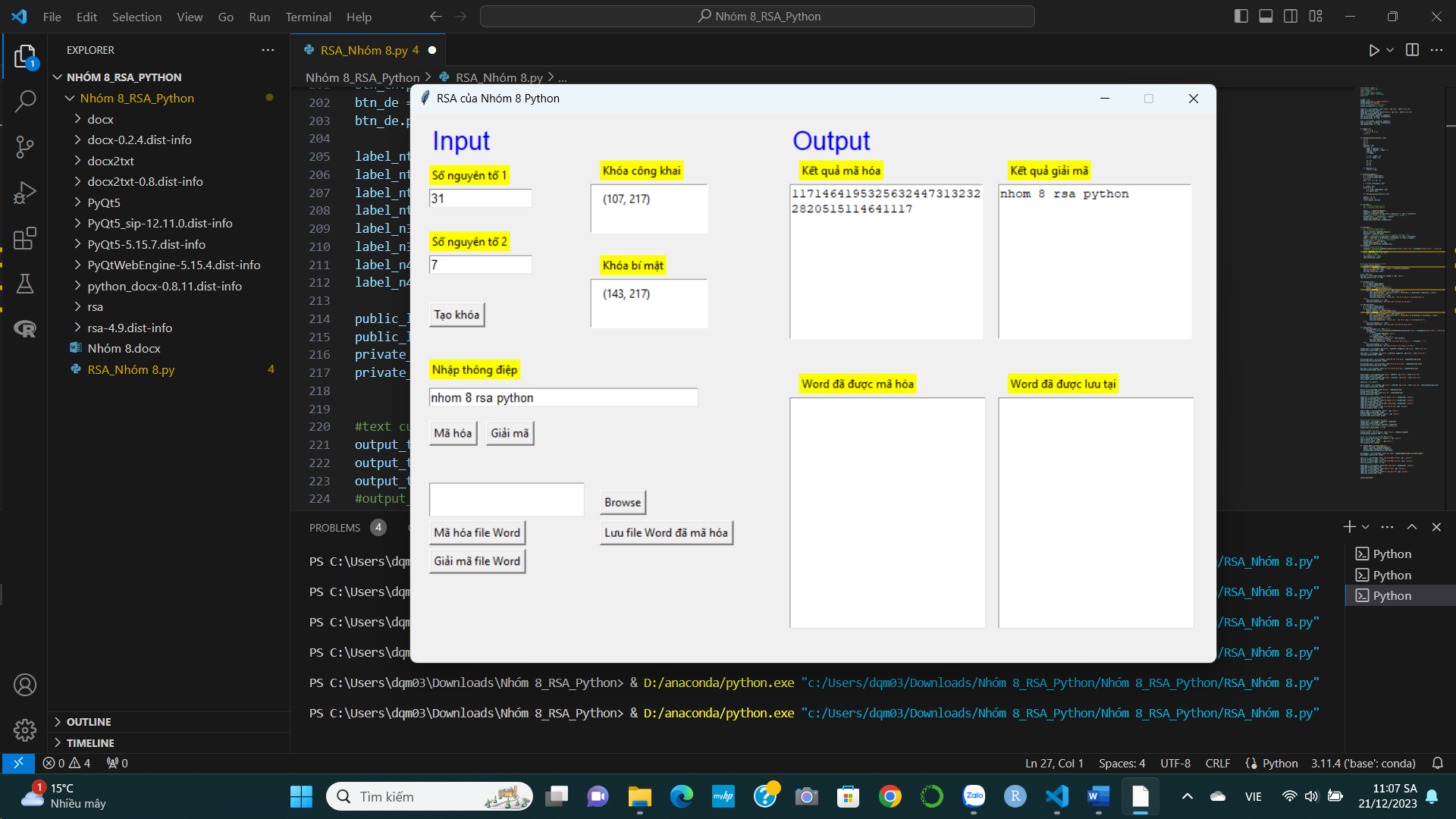
+ Về nguyên tắc, cả A và B đều không phát hiện được sự can thiệp của C.

## **2.3. Thiết kế, cài đặt chương trình demo thuật toán**

### 2.3.1. Mã hóa và giải mã RSA bằng Python

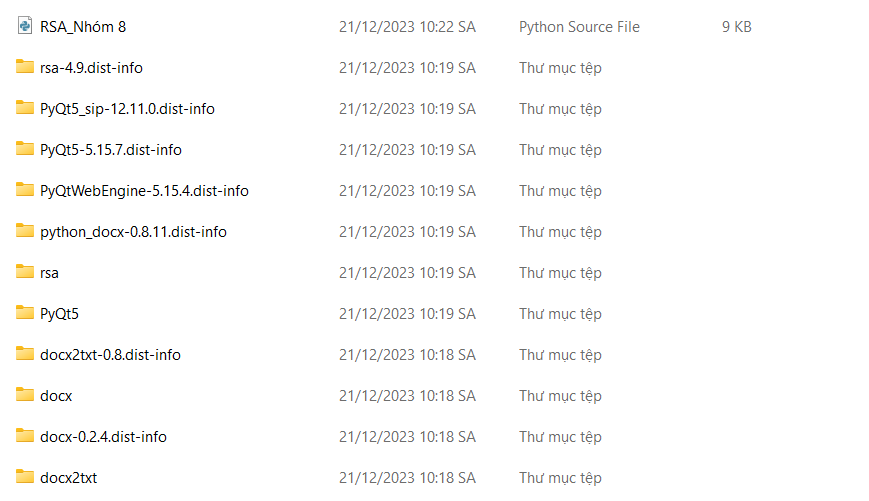
2.3.1.1. Giao diện chương trình demo



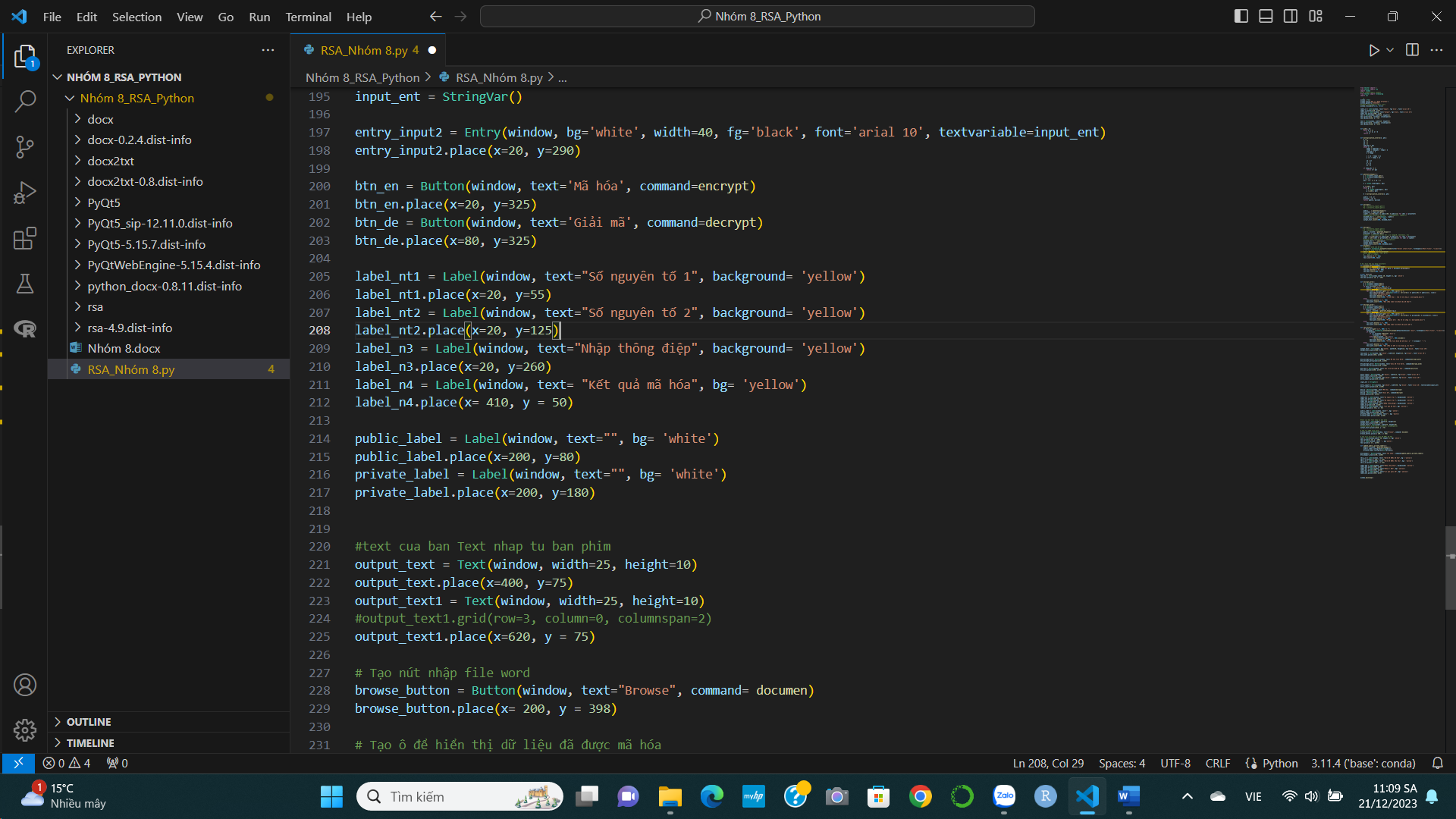


2.3.1.2. Cài đặt và triển khai

*-* Bước 1:Giải nén và mở Visual Studio Code



* Bước 2: Chọn  để chạy chương trình:



2.3.1.3. Code chương trình

#Khai báo thư viện

from tkinter import \*

from tkinter import ttk

import random

from tkinter import filedialog

import sys

import pickle

#import docx

#import docx2txt

#import rsa

#from tkinter import PhotoImage

#from tkinter import messagebox

#import docx

#from docx import Document

#hiển thị cửa sổ

window = Tk()

window.title('RSA của nhóm 8 Python')

window.geometry('850x530')

#window.iconbitmap("C:\Py\L.ico")

window.resizable(False, False)

label\_in = Label(window, *text*="Input", *fg*='blue', *font*='arial 20')

label\_in.place(*x*=20, *y*=10)

label\_out = Label(window, *text*="Output", *fg*='blue', *font*='arial 20')

label\_out.place(*x*=400, *y*=10)

#Thuật toán

def gcd(*a*, *b*):

    while *b* != 0:

*a*, *b* = *b*, *a* % *b*

    return *a*

def multiplicative\_inverse(*e*, *phi*):

    d = 0

    x1 = 0

    x2 = 1

    y1 = 1

    temp\_phi = *phi*

    while *e* > 0:

        temp1 = temp\_phi // *e*

        temp2 = temp\_phi - temp1 \* *e*

        temp\_phi = *e*

*e* = temp2

        x = x2 - temp1 \* x1

        y = d - temp1 \* y1

        x2 = x1

        x1 = x

        d = y1

        y1 = y

    if temp\_phi == 1:

        return d + *phi*

def generate\_keypair():

    p = int(entry\_input.get())

    q = int(entry\_input1.get())

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = random.randrange(1, phi)

    g = gcd(e, phi)

    while g != 1:

        e = random.randrange(1, phi)

        g = gcd(e, phi)

    d = multiplicative\_inverse(e, phi)

    public = (e, n)

    private = (d, n)

    return public, private

# mã hóa

def encrypt():

    #p = int(entry\_input1.get())

    #q = int(entry\_input2.get())

    public, \_ = generate\_keypair()

    plaintext = input\_ent.get()

    cipher = [(ord(char) \*\* public[0]) % public[1] for char in plaintext]

    #messagebox.showinfo('Mã hóa', ''.join(map(str, cipher)))

    encoded\_text = ''.join(map(str, cipher))

    output\_text.delete('1.0', END)

    output\_text.insert(END, encoded\_text)

#Giải mã

def decrypt():

   # p = int(entry\_input1.get())

   # q = int(entry\_input2.get())

    public, private = generate\_keypair()

    plaintext = input\_ent.get()

    #cipher = [int(text[i:i+3]) for i in range(0, len(text), 3)]

    cipher = [(ord(char) \*\* public[0]) % public[1] for char in plaintext]

    plain = [chr((char \*\* private[0]) % private[1]) for char in cipher]

    #output\_label.config(text= ''.join(plain))

    decoded\_text = ''.join(plain)

    output\_text1.delete('1.0', END)

    output\_text1.insert(END, decoded\_text)

#Lưu khóa

def save\_keypair():

    with open("keypair.pickle", "rb") as f:

        keypair = pickle.load(f)

    with open("public\_key.txt", "w") as f:

        f.write(str(keypair[0]))

    with open("private\_key.txt", "w") as f:

        f.write(str(keypair[1]))

    text\_boxt.delete('1.0', END)

    text\_boxt.insert(END, "Đã lưu khóa công khai vào public\_key.txt và khóa bí mật vào private\_key.txt")

#tạo khóa

def update\_public\_private\_labels():

    public, private = generate\_keypair()

    public\_label.config(*text*=str(public))

    private\_label.config(*text*=str(private))

#Tạo nút và hiển thị kết quả Lưu khóa

text\_boxt = Text(window, *bg*='white', *width*=20, *height*=3, *fg*='black', *font*='arial 10')

text\_boxt.place(*x*=180, *y*=234)

button\_save = Button(window, *text*="Lưu khóa", *command*=save\_keypair)

button\_save.place(*x*=20, *y*=230)

def documen():

    # Mở file word cần mã hóa

    filepath = filedialog.askopenfilename(*title*="Select a Word file", *filetypes*=[("Word files", "\*.docx")])

    # Đọc nội dung file word

    text = docx2txt.process(filepath)

    #with open(filepath, 'rb') as f:

    #    data = f.read()

    text.delete('0,1', END)

    text.insert(END, text)

# Hàm hiển thị nội dung file Word

def display\_text(*filename*):

    document = Document(*filename*)

    text = "\n".join([para.text for para in document.paragraphs])

    text\_box.delete("1.0", END)

    text\_box.insert(END, text)

global text\_box

text\_box = Text(window,*width*= 20, *height*= 2, *bg*= 'white')

text\_box.place(*x*= 20, *y* = 390)

#Mã hóa file

def encrypt\_word():

    p = int(entry\_input1.get())

    q = int(entry\_input2.get())

    if text\_word.get('1.0', END) != "":

        public, \_ = generate\_keypair(p,q)

        text = Document()

        for line in text\_word.get('1.0', END).split('\n'):

            text.add\_paragraph(''.join(map(lambda *x*: chr((ord(*x*) \*\* public[0]) % public[1]), line)))

            text.save('encrypted.docx')

            text\_word.delete('1.0', END)

            text\_word.insert(END, "Đã mã hóa và lưu thành công vào encrypted.docx!")

    else:

        text\_word.delete('1.0', END)

        text\_word.insert(END, "Bạn chưa chọn file Word để mã hóa!")

#Giả mã file

def decrypt\_word():

    p = int(entry\_input1.get())

    q = int(entry\_input2.get())

    if text\_word.get('1.0', END) != "":

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

        text = Document()

        for line in text\_word.get('1.0', END).split('\n'):

            text.add\_paragraph(''.join(map(lambda *x*: chr((ord(*x*) \*\* private[0]) % private[1]), line)))

            text.save('decrypted.docx')

            text\_word.delete('1.0', END)

            text\_word.insert(END, "Đã giải mã và lưu thành công vào decrypted.docx!")

    else:

        text\_word.delete('1.0', END)

        text\_word.insert(END, "Bạn chưa chọn file Word để giải mã!")

#Lưu file

def save\_file():

    if text\_word.get('1.0', END) != "":

        filename = filedialog.asksaveasfilename(*defaultextension*=".docx", *filetypes*=[("Word files", "\*.docx")])

        if filename:

            if not filename.endswith('.docx'):

                filename += '.docx'

            with open(filename, 'wb') as f:

                f.write(text\_word.get('1.0', END).encode())

            text\_word.delete('1.0', END)

            text\_word.insert(END, "Đã lưu file Word đã mã hóa vào " + filename + " !")

    else:

        text\_word.delete('1.0', END)

        text\_word.insert(END, "Bạn chưa có bất cứ nội dung gì để lưu!")

#Tạo output

output\_text = Text(window, *bg*='white', *width*=29, *height*=12, *fg*='black', *font*='arial 10')

output\_text.place(*x*=400, *y*=300)

text\_word = Text(window, *bg*='white', *width*=29, *height*=12, *fg*='black', *font*='arial 10')

text\_word.place(*x*=620, *y*=300)

text = Text(window, *width*=15, *height*=3)

text.grid(*row*=3, *column*=0, *columnspan*=2)

text.place(*x*=190, *y* = 75)

text = Text(window, *width*=15, *height*=3)

text.grid(*row*=3, *column*=0, *columnspan*=2)

text.place(*x*=190, *y* = 155)

#Tạo nút

btn\_encrypt\_word = Button(window, *text*='Mã hóa file Word', *command*=encrypt\_word)

btn\_encrypt\_word.place(*x*=20, *y*=430)

btn\_decrypt\_word = Button(window, *text*='Giải mã file Word', *command*=decrypt\_word)

btn\_decrypt\_word.place(*x*=20, *y*=460)

btn\_save = Button(window, *text*='Lưu file Word đã mã hóa', *command*=save\_file)

btn\_save.place(*x*=200, *y*=430)

btn\_en = Button(window, *text*='Mã hóa', *command*=encrypt)

btn\_en.place(*x*=20, *y*=325)

btn\_de = Button(window, *text*='Giải mã', *command*=decrypt)

btn\_de.place(*x*=80, *y*=325)

btn\_keypair = Button(window, *text*='Tạo khóa', *command*=update\_public\_private\_labels)

btn\_keypair.place(*x*=20, *y*=200)

# Tạo nút nhập file word

browse\_button = Button(window, *text*="Browse", *command*= documen)

browse\_button.place(*x*= 200, *y* = 398)

#Tạo thanh nhập thông tin từ bàn phím

entry\_input = Entry(window, *bg*='white', *width*=15, *fg*='black', *font*='arial 10')

entry\_input.place(*x*=20, *y*=80)

entry\_input1 = Entry(window, *bg*='white', *width*=15, *fg*='black', *font*='arial 10')

entry\_input1.place(*x*=20, *y*=150)

input\_ent = StringVar()

entry\_input2 = Entry(window, *bg*='white', *width*=40, *fg*='black', *font*='arial 10', *textvariable*=input\_ent)

entry\_input2.place(*x*=20, *y*=290)

#Hiển thị thông điệp

label\_nt1 = Label(window, *text*="Số nguyên tố 1", *background*= 'yellow')

label\_nt1.place(*x*=20, *y*=55)

label\_nt2 = Label(window, *text*="Số nguyên tố 2", *background*= 'yellow')

label\_nt2.place(*x*=20, *y*=125)

label\_n3 = Label(window, *text*="Nhập thông điệp", *background*= 'yellow')

label\_n3.place(*x*=20, *y*=260)

label\_n4 = Label(window, *text*= "Kết quả mã hóa", *bg*= 'yellow')

label\_n4.place(*x*= 410, *y* = 50)

public\_label = Label(window, *text*="", *bg*= 'white')

public\_label.place(*x*=200, *y*=80)

private\_label = Label(window, *text*="", *bg*= 'white')

private\_label.place(*x*=200, *y*=180)

fil\_lbl = Label(window, *text*= "Chọn file")

fil\_lbl.place(*x*= 24, *y* = 370)

#text cua ban Text nhap tu ban phim

output\_text = Text(window, *width*=25, *height*=10)

output\_text.place(*x*=400, *y*=75)

output\_text1 = Text(window, *width*=25, *height*=10)

#output\_text1.grid(row=3, column=0, columnspan=2)

output\_text1.place(*x*=620, *y* = 75)

# Tạo ô để hiển thị dữ liệu đã được mã hóa

text = Text(window,*width*= 20, *height*= 2, *bg*= 'white')

text.place(*x*= 20, *y* = 390)

text = Label(window, *text*= '', *bg*='white')

text.place(*x*= 23 , *y*=395)

#Hiển thị thông điệp

lbl\_w\_in = Label(window, *text*= "Word đã được mã hóa", *bg* = 'yellow')

lbl\_w\_in.place(*x* = 410, *y* = 275)

lbl\_w\_ou = Label(window, *text*= "Word đã được lưu tại", *bg* = 'yellow')

lbl\_w\_ou.place(*x* = 630, *y* = 275)

lab\_sa\_fi = Label(window, *text*= 'Khóa được lưu')

lab\_sa\_fi.place(*x*=180, *y* = 211)

label\_pub = Label(window, *text*="Khóa công khai", *background*= 'yellow')

label\_pub.place(*x*=200, *y*=50)

label\_pri = Label(window, *text*="Khóa bí mật", *bg*= 'yellow')

label\_pri.place(*x*=200, *y*=131)

label\_out = Label(window, *text*="Kết quả giải mã", *bg*= 'yellow')

label\_out.place(*x*=630, *y*=50)

window.mainloop()

## **2.4. Thực hiện bài toán**

### 2.4.1. Phân công công việc

|  |  |  |
| --- | --- | --- |
| STT | Nội dung | Người thực hiện |
| 1 | Tổng quan về hệ mã RSA | Trương Hoàng Sơn |
| 5 | Thuật toán Euclid | Vương Quốc Minh |
| 6 | Định lý Fermat | Đỗ Quang Minh |
| 7 | Hàm số Euler | Đỗ Quang Minh |
| 8 | Thuật toán Miller-Rabin | Nguyễn Việt Cường |
| 9 | Định lý phần dư trung hoa | Nguyễn Việt Cường |
| 10 | Hệ mã hóa công khai RSA | Trương Hoàng Sơn |
| 11 | Kiến thức lĩnh hội và bài học kinh nghiệm | Vương Quốc Minh |

### 2.4.2. Trương Hoàng Sơn

* Tổng quan về hệ mã RSA
* Hệ mã hóa công khai RSA
* Viết chương trình demo với ngôn ngữ python

### 2.4.3. Đỗ Quang Minh

* Hàm số Euler
* Định lý Fermat
* Viết chương trình demo với ngôn ngữ python

### 2.4.5. Nguyễn Việt Cường

* Định lý phần dư trung hoa
* Thuật toán Miller-Rabin
* Viết chương trình demo với ngôn ngữ Python

### 2.4.6. Vương Quốc Minh

* Thuật toán Euclid
* Kiến thức lĩnh hội và bài học kinh nghiệm
* Viết chương trình demo với ngôn ngữ python

# CHƯƠNG 3: KIẾN THỨC LĨNH HỘI VÀ BÀI HỌC KINH NGHIỆM

## **3.1. Nội dung đã thực hiện**

Qua thời gian thực hiện, dưới sự hướng dẫn tận tình của cô Lê Thị Anh, nhóm em đã cố gắng hoàn thành đề tài theo đúng yêu cầu và thời gian quy định. Trong suốt quá trình nghiên cứu và thực hiện đề tài, nhóm chúng em đã lĩnh hội được những kiến thức, nội dung và kỹ năng sau đây:

* *Nội dung đã làm trong bài tập lớn*
* Thuật toán Euclid
* Định lý Fermat
* Hàm số Euler
* Thuật toán Miller-Rabin
* Định lý phần dư trung hoa
* Thành thạo ngôn ngữ lập trình python để thiết kế và cài đặt demo mã hóa và giải mã RSA.
* Hệ mã hóa RSA
* *Bài học kinh nghiệm*
* Các kỹ năng cần phải có như :
* Kỹ năng làm việc nhóm
* Kỹ năng tóm tắt, phân tích và giải quyết vấn đề
* Kỹ năng nghiên cứu, tìm tòi và học hỏi
* Kiến thức bắt buộc:
* Tiếng anh cơ bản
* Các môn toán học và lập trình
* Kiến thức chuyên sâu :
* Có kiến thức nền tảng về máy tính (phần cứng, phần mềm) và hệ thống mạng
* Hiểu và nắm bắt về luật  an toàn thông tin
* Vận dụng tốt ngôn ngữ lập trình Python
* Hiểu và vận hành quy trình phát triển phần mềm
* Phân tích lỗ hổng, virus, mã độc, phân tích đánh giá hệ thống
* Có chuyên môn về mã hóa thông tin, an toàn cơ sở dữ liệu
* Đối với mỗi cá nhân:
* Có ý thức nâng cao hiểu biết, nhận thức bản thân về an toàn thông tin. Tự trau dồi kinh nghiệm ứng phó sự cố bảo mật cũng như vận hành các quy trình bảo mật mới.
* Thường xuyên cập nhật phần mềm, hệ điều hành máy tính cá nhân lên phiên bản mới nhất. Không sử dụng phần mềm crack.
* Đề cao cảnh giác khi duyệt email, kiểm tra kỹ tên người gửi để phòng tránh lừa đảo. Tuyệt đối không tải các file đính kèm hoặc nhấp vào đường link không rõ nguồn gốc.

## **3.2. Hướng phát triển**

Đứng trước xu hướng phát triển mạnh mẽ của Công nghệ thông tin việc nghiên cứu các vấn đề bảo mật và an toàn thông tin để đảm bảo cho các hệ thống thông tin hoạt động an toàn là điều vô cùng quan trọng và bức thiết. Với đề tài xây dựng chương trình mã hóa và giải mã RSA, tuy chương trình cài đặt chưa được hoàn hảo, song nếu có thời gian phát triển và hoàn thiện hơn thì chương trình sẽ rất có ích và có tính ứng dụng cao.

- Chứng thực (authentication) là quá trình một server xác định xem bạn là ai. Phương pháp chứng thực đơn giản nhất mà bạn sử dụng hằng ngày được gọi là “đăng nhập” (log in). Hằng ngày, bạn nhập một mật khẩu tĩnh (không thay đổi trong vài tháng, hoặc cả đời), và server sẽ dò xem mật khẩu của bạn có khớp với mật khẩu được lưu hay không. Tuy nhiên, một số người đặt mật khẩu rất dễ đoán, và mật khẩu có khả năng bị đánh cắp rất cao bằng những thủ thuật đơn giản. Do đó, trong một số các ứng dụng, một “mật khẩu điện tử” sẽ giúp việc chứng thực trở nên an toàn hơn. Phương thức chứng thực này thường được sử dụng trong SSH để đăng nhập vào máy khác (thường là server) mà không cần phải nhập mật khẩu.

- Ghi chú: Server không lưu trực tiếp mật khẩu của bạn, mà lưu bản hash của nó, để nếu khi hash bị lộ thì mật khẩu của bạn vẫn tương đối an toàn.

- Thay vì một mật khẩu bình thường, người đăng nhập sẽ sinh ra một ra một bộ khóa công khai và bí mật. Sau đó, người đăng nhập sẽ đăng ký khóa công khai lên server (khóa bí mật không bao giờ bị tiết lộ ra ngoài). Mỗi khi cần đăng nhập vào server, các bước sau đây sẽ xảy ra:

+ Server sinh ra một số ngẫu nhiên a và gửi cho người đăng nhập.

+ Người đăng nhập sử dụng khóa bí mật để tạo mật văn b gửi lại cho server.

+ Server sử dụng khóa công khai (được đăng ký trước đó) để giải mã. Nếu kết quả sau khi giải mã lại bằng a, đó là bằng chứng cho việc người đăng nhập sở hữu đúng khóa bí mật.

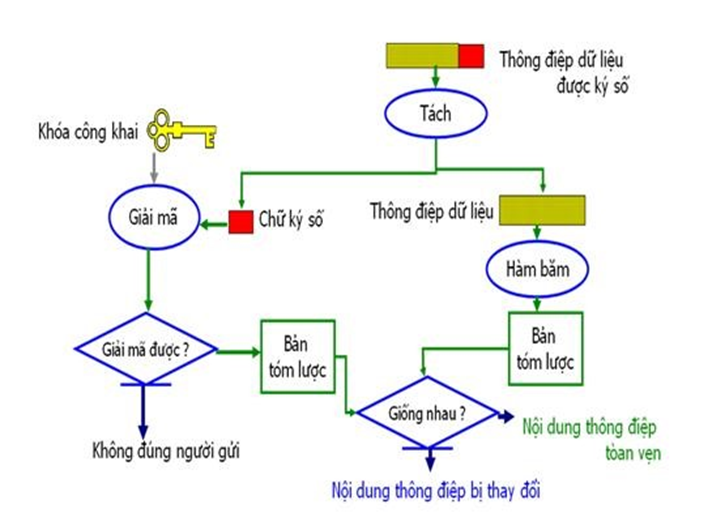
- Nếu một kẻ mạo nhận muốn đăng nhập vào server, sau khi được nhận a từ server, hắn không có đúng khóa bí mật và gửi lại mật văn giả b’, thì khi server giải mã ra lại sẽ có a’ khác a và do đó hắn không thể đăng nhập. Mặt khác, hắn không thể đoán hay tái sử dụng mật văn b, bởi vì a là ngẫu nhiên và sẽ thay đổi cho mỗi lần đăng nhập khác nhau.

- Một điều bất tiện khi sử dụng “mật khẩu điện tử” này là khóa bí mật chỉ hợp lệ trên một máy duy nhất (bạn có thể copy sang máy khác, nhưng điều đó làm khóa bí mật của bạn gặp nhiều rủi ro bị tiết lộ hơn). Tuy nhiên, bạn hoàn toàn có thể tạo một bộ khóa riêng cho mỗi máy và đăng ký nhiều khóa công khai cho một tài khoản, như vậy bạn có thể đăng nhập ở bất kỳ máy nào.

- Tuy nhiên, nếu kẻ tấn công nghe lén những gói tin của bạn, mặc dù hắn không biết được mật khẩu, hắn vẫn có thể biết được những thông tin bạn truyền sau khi bạn đăng nhập, ví dụ lịch sử giao dịch ngân hàng của bạn. Do đó, bạn cần phải có một cách để bảo vệ đường truyền của bạn.

## **3.3. Ứng dụng của RSA**

**-**  Chữ ký điện tử

****

**-** Ứng dụng chữ ký điện tử trong kê khai thuế

