**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

Logo

Description automatically generated**KHOA CÔNG NGHỆ THÔNG TIN**

**LAB FINAL PROJECT REPORT**

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MSSV:** 19120548

**Họ tên sinh viên:** Phùng Anh Khoa

**Lớp:** 21\_21

**Học phần:** Nhập môn mã hóa mật mã

**GVHD:** Đặng Trần Minh Hậu

**Thành phố Hồ Chí Minh, tháng 1 năm 2024**

Table of Contents

[**I.** **Bài 1: Kiểm tra số nguyên tố** 7](#_Toc157194893)

[**1.** **Phương pháp/thuật toán triển khai (lý thuyết)** 7](#_Toc157194894)

[**1.1.** **Định nghĩa số nguyên tố** 7](#_Toc157194895)

[**1.2.** **Giới thiệu một số cách kiểm tra số nguyên tố** 7](#_Toc157194902)

[**1.3.** **Thuật toán Miller-Rabin** 7](#_Toc157194909)

[**2.** **Điểm mấu chốt mà bạn nghĩ là quan trọng nhất/tâm đắt nhất/đóng góp nhất của bản thân cho bài giải** 11](#_Toc157194931)

[**3.** **Khó khăn gặp phải khi thực hiện** 12](#_Toc157194952)

[**II.** **Bài 2: Mã hóa RSA – Sinh khóa** 13](#_Toc157194957)

[**1.** **Phương pháp/thuật toán triển khai (lý thuyết)** 13](#_Toc157194958)

[**1.1.** **Cơ chế sinh khóa của hệ mã RSA** 13](#_Toc157194959)

[**1.2.** **Chứng minh** 13](#_Toc157194963)

[**2.** **Điểm mấu chốt mà bạn nghĩ là quan trọng nhất/tâm đắt nhất/đóng góp nhất của bản thân cho bài giải** 15](#_Toc157194987) [15](#_Toc157194990)

[**3.** **Khó khăn gặp phải khi thực hiện** 15](#_Toc157194991)

1. **Bài 1: Kiểm tra số nguyên tố**
   1. **Phương pháp/thuật toán triển khai (lý thuyết)**
   2. **Định nghĩa số nguyên tố**

Cho một số nguyên n ∈ Z lớn hơn 1. Khi đó, n được gọi là số nguyên tố khi và chỉ khi n thỏa tính chất n chỉ có hai ước dương là 1 và chính nó n.

∀a ∈ Z+, a | n ⇐⇒ a = 1 ∨ a = n

Trong đó:

• Z là tập các số nguyên.

• Z+ là tập các số nguyên dương.

• a | n có nghĩa a là ước của n, tức là ∃k ∈ Z, n = ka.

* 1. **Giới thiệu một số cách kiểm tra số nguyên tố**

**-** Phương pháp vét cạn: Thử lần lượt các số nguyên từ 1 đến n. Nếu chỉ có 1 và n là ước của n thì n là số nguyên tố.

Cải tiến: Thay vì thử các số nguyên từ 1 đến n, ta thử các số nguyên dương a ≤ √ n

- Thuật toán Miller–Rabin: Đây là thuật toán ngẫu nhiên dạng Monte Carlo, tức kết quả trả ra không phải lúc nào cũng chính xác, nhưng có thể cải thiện xác suất chính xác bằng cách thực hiện thuật toán với các dữ liệu đầu vào khác nhau.

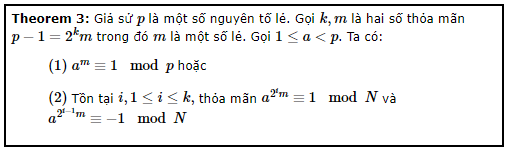
Tham khảo: <https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>.

* Thuật toán AKS: Không giống với thuật toán Miller-Rabin, đây là thuật toán tất định, tức kết quả trả ra luôn chính xác.

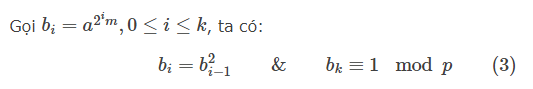
Tham khảo: <https://en.wikipedia.org/wiki/AKS_primality_test>.

* 1. **Thuật toán Miller-Rabin**

Thuật toán Miller-Rabin có độ chính xác khá cao ngay cả khi đầu vào là số Carmichael. Thuật toán được xây dựng dựa trên định lí sau:



**Chứng minh**: Gọi số nguyên x là căn bậc 2 đồng dư của a modulo p nếu a≡x2modp. Nếu p là một số nguyên tố, ta có thể dễ dàng chứng minh được pm1 là căn bậc 2 đồng dư duy nhất của 1 modulo p.



Gọi i0 là số nhỏ nhất sao cho b i0 ≡ 1 mod q. Số i0 tồn tại vì bk ≡ 1 mod p. Theo phương trình (3), ta có b2i0 ≡ 1 mod q và do đó, b i0 ≡ −1 mod q. Từ đó ta suy ra Theorem 3.

Gọi a là một bằng chứng (witness) cho biết n là hợp số nếu cả (1) và (2) trong Theorem 3 đều không thỏa mãn. Với bằng chứng a và số nguyên N, thủ tục Witness(a, N) sẽ trả về COMPOSITE nếu a là bằng chứng cho biết N là hợp số và trả về PRIME nếu ngược lại. Mã giả của thuật toán Miller-Rabin kiểm tra tính nguyên tố như sau:

MillerRabinTesting(N):  
    a ← a random number in {2,…,n−1}  
    **if** GCD(a, N) ≠ 1  
        return COMPOSITE  
    **else if** ModPower(a, N−1, N) ≠ 1         [[aN-1 ≢ 1 mod N]]  
        return COMPOSITE  
    **else**  
        return Witness(a, N)

Witness(a, N):  
    (k, M) ← Decompose(N − 1)         [[write N – 1 = 2km]]  
    B[0] ← ModPower(a, m, N)  
    **if** B[0] = 1  
        return PRIME  
    **else**  
        i ← 1  
        **while** i ≤ k  
            B[i] ← (B[i − 1] ∗ B[i − 1]) mod N  
            **if**  B[i] = 1         [[i is the smallest s.t bi ≡ 1 mod N]]  
                **if**  B[i − 1] = N − 1         [[B[i − 1] ≡ −1 mod N]]  
                    return PRIME  
                **else**  
                    return COMPOSITE  
    return COMPOSITE

Decompose(p):  
    i ← 0  
    **while** p is even  
        i ← i + 1  
        p ← p/2  
    return (i, p)

Trong mã giả trên, B[i] chỉ phụ thuộc vào B[i − 1]. Do đó, bạn có thể tiết kiệm bộ nhớ một chút bằng cách sử dụng hai biến X, Y thay vì cả mảng B[1, 2, …, k]. Thủ tục Decompose thực hiện O(k) thao tác bit để tách N – 1 = 2km. Thủ tục Witness sử dụng k phép nhân hai số logN bit và O(k) phép toán cơ bản.

Do k ≤ log(N − 1), ta suy ra:

**Theorem 4:**Thuật toán MillerRabinTesting sử dụng O(log3N) thao tác bit để kiểm tra tính nguyên tố của số nguyên N.

Ví dụ: N = 561 = 3 ∗ 11 ∗ 17 là số Carmichael, hãy xem thuật toán Miller-Rabin thực hiện như thế nào. Trước hết tách N – 1 = 560 = 24 ∗ 35, do đó k = 5,m = 35. Giả sử a = 8, ta có:

B[0] = am mod N = 461

B[1] = 4612 mod N = 463

B[2] = 4622 mod N = 67

B[3] = 672 mod N = 1

Do B[3] = 1, B[2] ≠ N − 1, thuật toán Miller-Rabin sẽ trả lại COMPOSITE  
Theo Theorem 3, thuật toán Miller-Rabin luôn trả lại PRIME nếu đầu vào N là số nguyên tố. Cũng như thuật toán Fermat, thuật toán Miller-Rabin cũng có lỗi nếu đầu vào là hợp số. Tuy nhiên, ngay cả khi đầu vào N là số Carmichael, thuật toán Miller-Rabin có lỗi khá nhỏ. Do đó thuật toán này thường được sử dụng trong thực tế để kiểm tra tính nguyên tố của một số nguyên.

**Theorem 5:**Xác suất thuật toán MillerRabinTesting trả về PRIME với đầu vào hợp số N là:

Pr[error] ≤ 1/2

Về mặt lý thuyết có thể chứng minh được xác suất lỗi của thuật toán Miller-Rabin trong Theorem 5 là Pr[error] ≤ 1/4. Tuy nhiên chứng minh sẽ phức tạp hơn.

* 1. **Điểm mấu chốt mà bạn nghĩ là quan trọng nhất/tâm đắt nhất/đóng góp nhất của bản thân cho bài giải**

**Chứng minh Theorem 5** Ta chỉ cần chứng minh Theorem 5 với trường hợp N là số Carmichael vì phép thử Fermat là một thủ tục con của thuật toán Miller-Rabin. Trước hết ta chứng minh:

**Claim 1**: Không có số Carmichael có dạng N = pt với t ≥ 2 và p là một số nguyên tố.

Thật vậy, giả sử Claim 1 là sai, i.e, tồn tại số Carmichael có dạng N = pt, t ≥ 2. Nhóm ZN có bậc | ZN | = *φ* (pt) = pt − 1(p − 1) trong đó *φ* là hàm Euler. Từ đó suy ra  với mọi a ∈ ZN.

Do N là số Carmichael, aN – 1 ≡ 1 mod N, từ đó suy ra: pt -1(p − 1)|N = pt − 1. Tuy nhiên, điều này là không thể vì pt − 1(p − 1) chia hết cho p còn pt − 1 thì không. Do đó Claim 1 là đúng.

Như vậy, N không phải là lũy thừa của một số nguyên tố. Gọi x là một *số mũ tồi* nếu tồn tại a sao cho ax ≡ −1 mod N. Định nghĩa Sx={a|1 ≤ a ≤ N − 1, ax ≡ ±1 mod N}. Ta sẽ chứng minh:

**Claim 2**: Nếu N là một hợp số lẻ và không phải là lũy thừa của một số nguyên tố thì Sx là một nhóm con đúng của ZN.

Ta sẽ tìm một số b ∈ ZN mà b ∉ Sx. Trước hết ta có thể kiểm tra các tiên đề của nhóm đối với Sx. Vì N là một hợp số lẻ và không phải là lũy thừa của một số nguyên tố, tồn tại N1, N2 lẻ, nguyên tố cùng nhau, sao cho N = N1⋅N2. Vì x là một số mũ tồi, tồn tại a sao cho ax ≡ 1 mod N. Áp dụng Chinese Remainder Theorem, ta tìm được

b ∈ ZN sao cho:

b ≡ a mod N1

b ≡ 1 mod N2

Từ đó ta suy ra bx ≡ ax ≡ −1 mod N1 và bx ≡ 1 mod N2.

Nếu b ∈ Sx, ta có bx ≡ ±1 mod N. Ta có hai trường hợp:

1. bx ≡ 1 mod N, ta sẽ có bx ≡ 1mod N1, trái với điều ta vừa chứng minh
2. bx ≡ −1 mod N, suy ra bx ≡ −1 mod N2, trái với định nghĩa của b

Do đó, b ∉ Sx.

Như vậy, Claim 2 là đúng. Gọi x = 2im với 0 ≤ i ≤ k là số mũ tồi lớn nhất. x tồn tại vì R lẻ và (−1)R ≡ −1 mod N. Giả sử a ∈ {2,3,…,n − 1} không phải là bằng chứng cho biết N là hợp số. Gọi 0 ≤ i ≤ k là số trong Theorem 3, có hai trường hợp:

1. Nếu am ≡ 1 mod N ⇒ ax ≡ 1 mod N, suy ra a ∈ Sx
2. Nếu a2im ≡ −1 mod N, suy ra 2im là một số mũi tồi. Do x là số lớn nhất, ta có i¯ ≥ i, và do đó ax¯ = (a2im)2i¯ − i ≡ ±1 mod N. Suy ra a ∈ Sx¯.

Theo Claim 2, Sx¯ là một nhóm con đúng của ZN, ta có xác suất số ngẫu nhiên

a ∈ Sx¯ là Pr[a ∈ Sx¯] ≤ ≤ 1/2. Từ đó suy ra Theorem 5.

* 1. **Khó khăn gặp phải khi thực hiện**

Thông thường trong các ứng dụng, chúng ta phải kiểm tra tính nguyên tố của các số rất lớn. Các số này có số lượng bít biểu diễn lớn, do đó, rất dễ bị hiện tượng tràn số nếu chúng ta thực thi không cẩn thận. Xét trường hợp N là số nguyên 31 bit, khi thực hiện tính ab mod N sử dụng thủ tục ModPower(a, b, N) ở trên, chúng ta sẽ phải thực hiên phép nhân (x ∗ x) mod N hoặc x ∗ x ∗ a mod N với x, a là các số nguyên 31 bit. Kết quả của các phép nhân này có thể vượt quá 32 bit, và do đó chúng ta sẽ bị tràn số nếu chúng ta sử dụng kiểu **int** (giả sử bạn sử dụng ngôn ngữ C++). Ta có thể giải quyết vấn đề này như sau.

Trước hết tính (x ∗ x ∗ a) mod N = ((x ∗ x) mod N) ∗ a mod N. Sau đó, ta sẽ khai báo một biến kiểu unsigned long long để lưu giữ kết quả của x ∗ x trước khi lấy  mod N. Tuy nhiên không phải lúc nào chúng ta cũng có thể sử dụng biến kiểu dữ liệu lớn hơn.

Giả sử kiểu dữ liệu lớn nhất chúng ta có là k bit (thông thường là 64 bit hoặc 128 bit), trong khi N có kích thước k − 1 bít. Để tính tích (x ∗ x) mod N, ta có thể sử dụng hai số k bit để lưu trữ tích của hai số k − 1 bit trước khi lấy modulo, tuy nhiên ta sẽ phải tự thực hiện phép modulo. May mắn là có một kĩ thuật giúp chúng ta tính (x ∗ x) mod N sử dụng các số k bit mà không phải thực hiện lai phép modulo. Tuy nhiên, chúng ta sẽ phải sử dụng O(log2x) thao tác bit cho phép nhân x ∗ x. Thuật toán như sau:

ModMultiplication(a, b, N):  
    **if** b = 1  
        return a  
    **else**  
        x ← ModMultiplication(a, ⌊b/2⌋, p)  
        **if**  b is even  
            return (x + x) mod N  
        **else**  
            return ((x + x) mod N + a) mod N

1. **Bài 2: Mã hóa RSA – Sinh khóa**
   1. **Phương pháp/thuật toán triển khai (lý thuyết)**
   2. **Cơ chế sinh khóa của hệ mã RSA**

Cho số nguyên dương N = pq với p và q là các số nguyên tố. Khi đó, với cặp số nguyên dương (e, d) thỏa mãn φ(N) = (p − 1)(q − 1) là ước của ed − 1, tức là ed = 1 (mod φ(N)), ta có cặp khóa của hệ mã RSA như sau:

• Khóa công khai: Cặp số nguyên (N, e).

• Khóa bí mật: Cặp số nguyên (N, d).

* 1. **Chứng minh**
* Chọn 2 số nguyên tố khác nhau p và q.
* Tính n=p∗q. Độ dài của n (tính theo bit) chính là độ dài của khóa. Hiện nay người ta khuyến cáo sử dụng khóa có độ dài tối thiếu 2048 bit.
* Tính φ(n)=φ(p)φ(q)=(p−1)(q−1)=n−(p+q−1). Trong đó φ(n) là phi hàm Euler. φ(n)= số lượng số nguyên dương nhỏ hơn n mà nguyên tố cùng nhau với n.
* Chọn một số nguyên e thỏa mãn 1<e<φ(n)1và gcd(e,φ(n))=1.
* Tính ra d≡e−1(modφ(n))

**Chứng minh**

Để chứng minh tính đúng đắn của thuật toán sinh khóa trên, ta cần chứng minh rằng med ≡ m(modn) với n=pq

***1. Trường hợp***gcd(n,m)=1

Ta có ed≡1(modφ(n))

Do e và d nguyên dương nên ed=1+hφ(n) với h là một số nguyên không âm.

Do đó med=m1+hφn=m(mφ(n))h

Theo định lý Euler,

aφ(n) ≡ 1 (mod n) ∀a,n:gcd(a,n)=1 nên m(mφ(n))h≡m(1)h≡m(modn)

***2. Trường hợp***gcd(n,m)≠1

Theo định lý phần dư Trung Hoa (Chinese Remainder Theorem), nếu p, q nguyên tố cùng nhau thì:

x ≡ y (mod p) và x ≡ y (mod q) ⇒ x ≡ y (mod pq)

Do vậy ta cần chứng mình 2 mệnh đề sau:

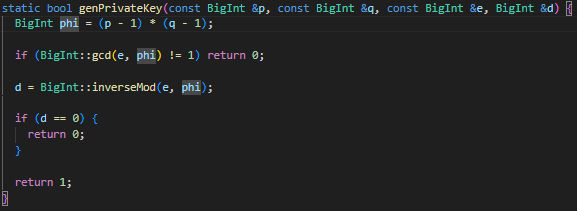
* med ≡ m (mod p)
* med ≡ m (mod q)

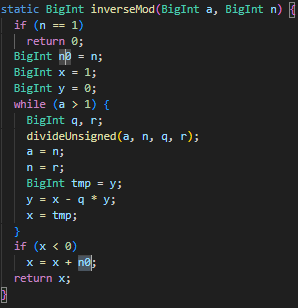
Vì gcd(m,n) ≠ 1⇒ gcd(m,n) = p hoặc gcd(m,n) = q.

Không mất tính tổng quát, giả sử gcd(m,n) = p. Ta có:

* gcd(m,p) = p ⇒ m ≡ 0 (mod p) ⇒med ≡ 0 ≡ m (mod p)
* gcd(m,q) = 1 ⇒ med ≡ m(mod q) (chứng mình tương tự trong trường hợp 1)
  1. **Điểm mấu chốt mà bạn nghĩ là quan trọng nhất/tâm đắt nhất/đóng góp nhất của bản thân cho bài giải**

Áp dụng các thuật toán vào việc sinh khóa





* 1. **Khó khăn gặp phải khi thực hiện**

- Việc chọn p và q, hai số nguyên tố lớn, đủ ngẫu nhiên và an toàn là một thách thức. Nếu chúng ta chọn số nguyên tố quá nhỏ, hệ mã RSA có thể trở nên dễ bị tấn công.

- Tính toán giá trị Euler(n) có thể trở nên phức tạp nếu p và q quá lớn. Việc chọn một số e sao cho nó và Euler(n) là nguyên tố cùng nhau cũng đòi hỏi sự cẩn trọng.

- Tìm kiếm giá trị d sao cho (d \* e) mod Euler(n) = 1 có thể tốn nhiều thời gian, đặc biệt là khi Euler(n) là một số lớn.

- Bảo mật của hệ mã RSA phụ thuộc vào sự khó khăn của việc phân tích nguyên tố lớn. Tuy nhiên, các tiến triển trong lĩnh vực toán học và máy tính có thể làm giảm độ an toàn của hệ thống, đặc biệt là nếu sử dụng khóa ngắn hoặc nếu có các phương pháp tấn công mới được phát triển.

- Việc tính toán trên các số nguyên tố lớn có thể tốn nhiều tài nguyên tính toán, đặc biệt là khi áp dụng cho các hệ thống có độ dài khóa lớn.