ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

MẬT MÃ VÀ AN NINH MẠNG (CO3069)

BÀI TẬP LỚN
HIỆN THỰC HỆ MÃ RSA

Lớp L01 - HK231

Giảng viên hướng dẫn: Nguyễn Cao Đạt
Sinh viên thực hiện: Nguyễn Văn Khánh Nhân 2010480
Tô Hòa 1910198
Trần Nguyễn Nam Anh 2110758
Lâm Phạm Trọng Phúc 2011844

Thành phố Hồ Chí Minh - 09/2023

Danh sách hình vẽ

MỤC LỤC

I Giới thiệu

1 Tổng quan đề tài

Trong đề tài này, nhóm tác giả hiện thực hệ mã RSA trên Java. Trong hiện thực RSA, giả sử các số nguyên tố lớn ít nhất phải 500 bits (có thể lớn hơn) và các hàm được yêu cầu hiện thực gồm:

- Tìm số nguyên tố lớn khi cho số lượng bit của số nguyên tố lớn cần tìm.
- Tính ước số lớn nhất khi cho hai số nguyên lớn.
- Tính toán khóa giải mã d khi cho khoá mã hóa e và hai số nguyên tố lớn.
- Tạo bộ khóa ngẫu nhiên khi cho 2 số nguyên tố lớn.
- Mã hóa khi cho thông điệp và khóa mã hóa e và n.
- Giải mã khi cho thông điệp mã hóa và khóa giải mã d và n.

2 Cấu trúc bài báo cáo

Bài báo cáo gồm 5 chương, với các nội dung chính được trình bày như sau:

- **Chương 1 - Giới thiệu:** Trình bày tổng quan về đề tài và cấu trúc của bài báo cáo.
- **Chương 2 - Cơ sở lý thuyết:** Trình bày các cơ sở lý thuyết liên quan đến đề tài.
- **Chương 3 - Phân tích và thiết kế:** Trình bày các bước phân tích và thiết kế dựa trên các yêu cầu của đề tài.
- **Chương 4 - Hiện thực và đánh giá:** Trình bày phần hiện thực và kết quả đánh giá.
- **Chương 5 - Kết luận:** Trình bày những việc làm được và những việc chưa làm được, ưu và nhược điểm, hướng phát triển của đề tài.

II Cơ sở lý thuyết

1 Cơ sở toán học

1.1 Phép chia lấy dư và phép modular

Division Algorithm: Cho số nguyên dương n và số nguyên không âm a, nếu ta chia a cho n, ta sẽ thu được số nguyên thương q và số nguyên dư r tuân theo mối quan hệ sau:

$$a = qn + r \qquad 0 \leq r < n; q = \lfloor a/n \rfloor \tag{1}$$

trong đó $\lfloor x \rfloor$ là só nguyên lớn nhất nhỏ hơn hoặc bằng x.

Modular Arithmetic: Nếu a là số nguyên và n là số nguyên dương, chúng ta xác định $a \ mod \ n$ là phần dư khi chia a cho n. Số nguyên n được gọi là **modulus (số chia)**. Do đó với bất kì số nguyên a, ta có thể viết lại phương trình (1) như sau:

$$a = \lfloor a/n \rfloor n + (a \mod n)$$

Hai số nguyên a và b được gọi là **đồng dư modulo** n nếu $(a \bmod n) = (b \bmod n)$. Ký hiệu: $a \equiv b \pmod{n}$. Nếu $a \equiv 0 \pmod{n}$ thì $n \mid a$.

Các tính chất của phép đồng dư:

- $a \equiv b \pmod{n}$ nếu $n|(a-b)$
- $a \equiv b \pmod{n}$ dẫn đến $b \equiv a \pmod{n}$
- $a \equiv b \pmod{n}$ và $b \equiv c \pmod{n}$ dẫn đến $a \equiv c \pmod{n}$

Các quy tắc cho phép tính thường gồm cộng, trừ, nhân cũng được áp dụng trong số học modulo (Modular arithmetic):

- $(a \bmod n) + (b \bmod n) \bmod n = (a+b) \bmod n$
- $(a \bmod n)$ - $(b \bmod n) \bmod n = (a-b) \bmod n$
- $(a \bmod n)$ x $(b \bmod n) \bmod n = (axb) \bmod n$

1.2 Ước số chung lớn nhất (GCD)

Số b khác 0 được xác định là thừa số của a nếu $a = mb$ với m là một số bất kì, trong đó a, b và m đều là số nguyên. $GCD(a,b)$ là ký hiệu để chỉ ước số chung lớn nhất của a và b, là số nguyên k lớn nhất thỏa $k \mid a$ và $k \mid b$:

$$GCD(a,b) = max[k, k \mid a \ \ \& \ \ k \mid b]$$

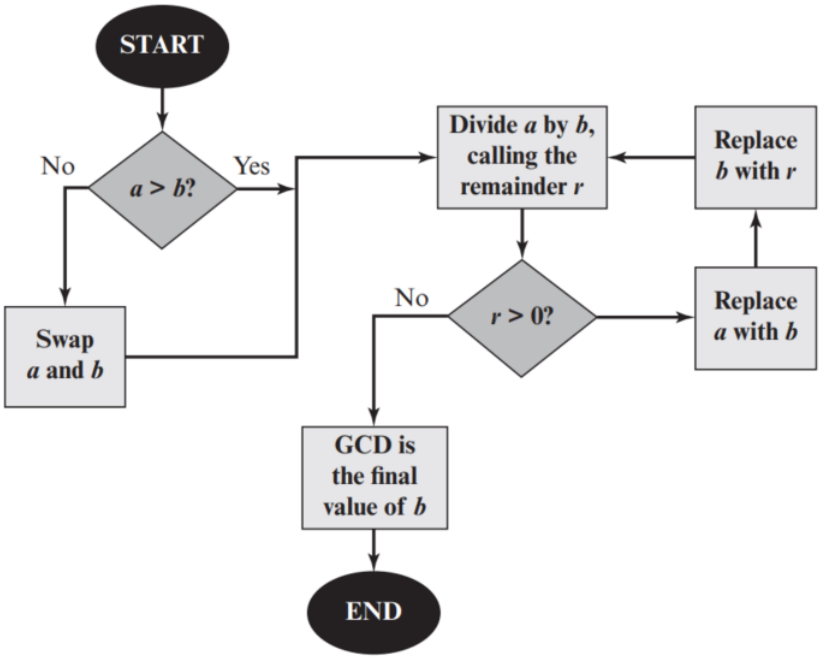
Lưu ý:

- GCD(0,0) = 0
- GCD(a,0) = |a|

Hai số nguyên a, b được gọi là hai số nguyên tố cùng nhau (relatively prime) nếu và chỉ nếu $GCD(a, b) = 1$.

Thuật toán Euclid: là thuật toán được xây dựng để tính GCD của hai số nguyên, là số lớn nhất có thể chia được bởi hai số nguyên đó với số dư bằng không. Thuật toán này có ý nghĩa quan trọng trong mật mã học. Các bước thực hiện thuật toán Euclid như sau:

- **Đầu vào:** Cho hai số nguyên dương a và b, với $a \geq b$.
- **Bước 1:** Chia a cho b và lấy phần dư, ký hiệu là r.
- **Bước 2:** Nếu r = 0, thì GCD(a, b) = b và thuật toán kết thúc.
- **Bước 3:** Nếu $r \neq 0$, thì gán a = b và b = r, sau đó quay lại Bước 1.
- Lặp lại các Bước 1 đến 3 cho đến khi r = 0.
- Kết quả cuối cùng là GCD(a, b), tức là ước chung lớn nhất của a và b.



Hình 1: Thuật toán Euclid

1.3 Mô đun nghịch đảo

Thuật toán Euclid mở rộng: quan trọng cho các tính toán sau này trong lĩnh vực của các trường hữu hạn và trong các thuật toán mã hóa như RSA. Với mối số nguyên a, b, thuật toán Euclid mở rộng không chỉ tính ra GCD là d mà còn tìm ra hai số nguyên x và y thỏa:

$$ax + by = d = GCD(a, b) \tag{2}$$

Extended Euclidean Algorithm			
Calculate	**Which satisfies**	**Calculate**	**Which satisfies**
$r_{-1} = a$		$x_{-1} = 1; y_{-1} = 0$	$a = ax_{-1} + by_{-1}$
$r_0 = b$		$x_0 = 0; y_0 = 1$	$b = ax_0 + by_0$
$r_1 = a \bmod b$ $q_1 = \lfloor a/b \rfloor$	$a = q_1 b + r_1$	$x_1 = x_{-1} - q_1 x_0 = 1$ $y_1 = y_{-1} - q_1 y_0 = -q_1$	$r_1 = ax_1 + by_1$
$r_2 = b \bmod r_1$ $q_2 = \lfloor b/r_1 \rfloor$	$b = q_2 r_1 + r_2$	$x_2 = x_0 - q_2 x_1$ $y_2 = y_0 - q_2 y_1$	$r_2 = ax_2 + by_2$
$r_3 = r_1 \bmod r_2$ $q_3 = \lfloor r_1/r_2 \rfloor$	$r_1 = q_3 r_2 + r_3$	$x_3 = x_1 - q_3 x_2$ $y_3 = y_1 - q_3 y_2$	$r_3 = ax_3 + by_3$
. . .	. . .	. . .	. . .
$r_n = r_{n-2} \bmod r_{n-1}$ $q_n = \lfloor r_{n-2}/r_{n-1} \rfloor$	$r_{n-2} = q_n r_{n-1} + r_n$	$x_n = x_{n-2} - q_n x_{n-1}$ $y_n = y_{n-2} - q_n y_{n-1}$	$r_n = ax_n + by_n$
$r_{n+1} = r_{n-1} \bmod r_n = 0$ $q_{n+1} = \lfloor r_{n-1}/r_n \rfloor$	$r_{n-1} = q_{n+1} r_n + 0$		$d = \gcd(a, b) = r_n$ $x = x_n; y = y_n$

Hình 2: Thuật toán Euclid mở rộng

Modulo nghịch đảo: Xét số nguyên dương n. Xét các số nguyên trên tập Z_n. Với số nguyên a, ta gọi nghịch đảo modulo a (modular multiplicative inverse) của a là a^{-1} là số nguyên thỏa $a \times a^{-1} \equiv \pmod{n}$. Tuy nhiên a^{-1} chỉ tồn tại nếu và chỉ nếu a và n là hai số nguyên tố cùng nhau $GCD(a, n) = 1$.

Cách tính modulo nghịch đảo dựa vào thuật toán Euclid mở rộng: Từ thuật toán Euclid mở rộng đã tìm hiểu ở trên, nếu ta có $GCD(a, n) = 1$, ta luôn tìm được 2 số nguyên x và y thỏa mãn $x \times x + n \times y = 1$. Vì ta đang xét $\pmod{n}$ nên từ đó ta có $a \times x \equiv 1 \pmod{n}$. Do đó x chính là a^{-1}.

1.4 Lý thuyết số nguyên tố

1.4.a Định lý Fermat nhỏ

Định lý Fermat nói rằng: Nếu p là số nguyên tố và a là một số nguyên dương không chia hết cho p thì

$$a^{p-1} \equiv 1(mod\ p) \tag{3}$$

1.4.b Định lý Euler

Hàm tổng của Euler: Trước khi trình bày định lý Euler, chúng ta cần giới thiệu một đại lượng quan trọng trong lý thuyết số, được gọi là hàm Euler's totient. Hàm này, được viết là $\phi(n)$, được định nghĩa là số lượng số nguyên dương nhỏ hơn n và nguyên tố cùng nhau với n. Theo quy ước, $\phi(1) = 1$

Định lý Euler: Định lý Euler phát biểu rằng với mọi a và n nguyên tố cùng nhau:

$$a^{\phi(n)} \equiv 1(mod\ n) \tag{4}$$

1.4.c Thuật toán Miller–Rabin

Thuật toán Miller–Rabin - Thuật toán do Miller và Rabin phổ biến được sử dụng để kiểm tra tính nguyên tố của một số lớn.

Chi tiết về thuật toán: nếu n là số nguyên tố, thì hoặc phần tử đầu tiên trong danh sách các dư số, hoặc phần dư $(a^q, a^{2q}, ..., a^{2^{k-1}q}, a^{2^k q})$ modulo n bằng 1; hoặc có một phần tử nào đó trong danh sách bằng (n - 1); nếu không, n là hợp số (nghĩa là không phải là số nguyên tố). Tuy nhiên, nếu điều kiện được thỏa mãn, điều đó không chắc chắn rằng n là số nguyên tố.

2 Thuật toán RSA

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n)=(p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Hình 3: Thuật toán RSA

2.1 Khóa công khai và khóa bí mật

- Alice chọn hai số nguyên tố ngẫu nhiên lớn p và q và tính n = p * q.
- Alice tính giá trị của hàm Euler của n, ký hiệu là $\phi(n)$, là số các số nguyên tố cùng nhau với n trong khoảng từ 1 đến n. Nó được tính bằng công thức $\phi(n)$ = (p-1) * (q-1).
- Alice chọn số nguyên e sao cho 1 < e < $\phi(n)$ và e là số nguyên tố cùng nhau với $\phi(n)$. e được chọn là khóa công khai của Alice và được công khai cho mọi người.
- Alice tính d (khóa bí mật) bằng cách tìm giá trị d sao cho (e * d) % $\phi(n)$ = 1.
- Khóa công khai của Alice là (e, n) và khóa bí mật của Alice là (d, n).

2.2 Mã hóa

- Bob muốn gửi một tin nhắn cho Alice. Bob chuyển đổi tin nhắn thành một số nguyên M, trong đó 0 <= M < n.

- Bob sử dụng khóa công khai của Alice (e, n) để mã hóa tin nhắn. Bob tính C = M^e mod n.
- Bob gửi giá trị C cho Alice.

2.3 Giải mã

- Alice nhận được giá trị mã hóa C từ Bob.
- Alice sử dụng khóa bí mật của mình (d, n) để giải mã tin nhắn. Alice tính M = C^d mod n.
- Alice thu được giá trị ban đầu của tin nhắn M.

III Phân tích và thiết kế

1 Những vấn đề cần giải quyết

- **Hiệu suất:** Thuật toán RSA, đặc biệt là trong việc sinh số nguyên tố lớn và các phép tính mod lớn, có thể gây ra hiệu suất không hiệu quả đối với các khối dữ liệu lớn.
- **Tạo khóa không hiệu quả:** Quá trình tạo khóa hiện tại trong lớp *RSACryptoSystem* có thể không tối ưu đối với độ dài bit lớn. Phương pháp *StrongPrimeGenerator* và các bước tạo khóa có thể cần tối ưu hóa để có hiệu suất tốt hơn.
- **Bảo mật**: Sử dụng ngẫu nhiên *SecureRandom* để tạo số nguyên tố và khóa, nhưng không có đánh giá cụ thể về độ mạnh của số nguyên tố được tạo. Việc sử dụng *SecureRandom* là quan trọng để tạo số nguyên tố ngẫu nhiên và khác biệt, nhưng việc này có thể đôi khi không đảm bảo tính ngẫu nhiên tuyệt đối. Cần kiểm tra cẩn thận về việc sử dụng nguồn số ngẫu nhiên để đảm bảo tính bảo mật của thuật toán.
- **Quản lý file:** Các phương thức đọc và ghi file có thể cần xử lý các tình huống ngoại lệ và quản lý nguồn tài nguyên hiệu quả hơn. Cách xử lý khóa (key) và tệp tin (file) có thể cần được cải thiện để đảm bảo an toàn và linh hoạt hơn. Hiện tại, việc lưu khóa vào các tệp DER không được mã hóa có thể là vấn đề bảo mật.
- **Chuẩn hóa dữ liệu:** Chương trình có thể không xử lý một số tình huống, chẳng hạn như dữ liệu đầu vào không hợp lệ hoặc độ dài khóa không phù hợp.
- **Xử lý lỗi:** Thiếu khả năng xử lý lỗi toàn diện đối với các thao tác tệp, phân tích cú pháp khóa và các lỗi thời gian chạy tiềm ẩn khác. Xử lý lỗi mạnh mẽ nên được thực hiện để nâng cao độ tin cậy của hệ thống.

2 Cần hiện thực lại những gì

- Cải thiện hiệu suất bằng cách tối ưu hóa thuật toán và xử lý các trường hợp đặc biệt, chẳng hạn như việc sử dụng kỹ thuật tinh chỉnh độ dài khóa dựa trên kích thước dữ liệu đầu vào.
- Đánh giá cường độ bảo mật của số nguyên tố được tạo bằng cách sử dụng các thuật toán kiểm tra số nguyên tố mạnh mẽ hơn. Kiểm tra và cải thiện cách thức

sinh số nguyên tố và quản lý ngẫu nhiên để đảm bảo tính bảo mật và đồng nhất

- Cải thiện quản lý nguồn tài nguyên, xử lý ngoại lệ một cách chặt chẽ và xem xét khả năng cải thiện hiệu suất. Xử lý các trường hợp dữ liệu không hợp lệ và đảm bảo tính chuẩn hóa của dữ liệu đầu vào.
- Cân nhắc sử dụng các chuẩn và phương pháp lưu trữ an toàn hơn cho khóa, chẳng hạn như lưu trữ chúng trong keystore bảo vệ mật khẩu.

3 Mô hình/Kiến trúc của chương trình

Cấu trúc lớp (Class) hiện tại có tính mô đun và có tổ chức. Mỗi class (*MyMath, PrivateKey, PublicKey, StrongPrimeGenerator và RSACryptoSystem*) có một trách nhiệm rõ ràng. Việc phân chia trách nhiệm được duy trì tốt.

3.1 Thiết kế hệ thống

1. *Tìm số nguyên tố lớn:* Nhận đầu vào là số bit mong muốn của số nguyên tố và trả về một số nguyên tố lớn có số bit được yêu cầu.
2. *Tính GCD:* Nhận hai số nguyên lớn làm đầu vào và trả về ước chung lớn của chúng.
3. *Tính khóa giải mã:* Nhận khóa mã hóa e và hai số nguyên tố lớn và trả về khóa giải mã d.
4. *Tạo cặp khóa ngẫu nhiên:* Nhận vào hai số nguyên tố lớn và trả về cặp khóa công khai và riêng tư.
5. *Mã hóa và Giải mã:* Nhận vào thông điệp và khóa mã hóa (e, n) hoặc thông điệp mã hóa và khóa giải mã (d, n). Trả về thông điệp được mã hóa hoặc giải mã.

3.2 Các bước thực hiện

1. *Tìm số nguyên tố lớn:* Sử dụng thuật toán tìm số nguyên tố với số bit được đặt ra.
2. *Tìm ước chung lớn nhất (GCD):* Sử dụng thuật toán Euclid mở rộng.
3. *Tính khóa giải mã:* Sử dụng mở rộng Euclid để tính nghịch đảo modulo.
4. *Tạo cặp khóa ngẫu nhiên:* Sử dụng các hàm đã triển khai trước đó để tạo khóa.

5. *Mã hóa và giải mã:* Sử dụng các hàm mũ modulo và phép toán số nguyên lớn để thực hiện mã hóa và giải mã.

4 Những chức năng

- **Chức năng tạo khóa:** Hàm *generateKeys* trong *RSACryptoSystem* chịu trách nhiệm tạo khóa công khai và riêng tư. Nó hiệu quả sử dụng class *StrongPrimeGenerator* để có được số nguyên tố mạnh.
- **Chức năng mã hóa và giải mã:** Các hàm mã hóa và giải mã trong *RSACryptoSystem* hiệu quả sử dụng quy luật mũ modul (*MyMath.modPow*) để thực hiện mã hóa và giải mã, tương ứng.
- **I/O file:** Các hàm để lưu và phân tích khóa từ và đến các tệp (*saveKeys, printPublicKey, printPrivateKey*) hoạt động. Tuy nhiên, việc thêm nhiều xử lý lỗi mạnh mẽ cho các hoạt động tệp sẽ cải thiện tổng độ ổn định của hệ thống.

IV Hiện thực và đánh giá

1 Hiện thực

1.1 Class MyMath

Class này dùng để chứa các hàm thể hiện các phép toán cơ bản

Một số hàm tiêu biểu

- **Hàm BigInteger MyMath.modPow(BigInteger a, BigInteger b, BigInteger n)**:

  Tính $a^b \mod n$.

- **Hàm BigInteger MyMath.gcd(BigInteger a, BigInteger b):** Tìm ước chung lớn nhất của hai số.

- **Hàm BigInteger[] MyMath.gcdExtended(BigInteger a, BigInteger b):** Tìm ước chung lớn nhất mở rộng và các hệ số x, y sao cho $ax + by = GCD(a, b)$.

- **Hàm BigInteger MyMath.modInverse(BigInteger e, BigInteger phi):** Tìm nghịch đảo modulo của một số

- **Hàm boolean MyMath.isProbablePrime(BigInteger n, int k):** Kiểm tra xem một số có là số nguyên tố xác suất hay không. Với k là số lần chạy test Miller-Rabin

- **Hàm boolean MyMath.fermatTestBase(BigInteger n, BigInteger a):** Kiểm tra điều kiện của phương pháp kiểm tra Fermat

- **Hàm boolean MyMath.millerRabinTest(BigInteger n, BigInteger k)** : Kiểm tra điều kiện của phương pháp kiểm tra Miller-Rabin

- **Các hàm tạo số ngẫu nhiên**

  - `randomBigInteger(BigInteger min, BigInteger max)`: Tạo số nguyên ngẫu nhiên trong khoảng `min` đến `max`.
  - `randomRange(BigInteger min, BigInteger max)`: Tạo số nguyên ngẫu nhiên trong khoảng `min` đến `max` (sử dụng RNG `rng`).
  - `randomBigInteger(int bitLength)`: Tạo số nguyên ngẫu nhiên có độ dài bit là `bitLength`.

- `randomSecondBigInteger(BigInteger prime1, int bitLength)`: Tạo số nguyên ngẫu nhiên lớn hơn nếu `prime1` $<$ `range / 2`; ngược lại, bé hơn.
- `randomPrime(int bitLength)`: Tạo số nguyên tố ngẫu nhiên có độ dài bit là `bitLength`.
- `gordonStrongPrime(int bitLen)`: Tạo số nguyên tố mạnh sử dụng thuật toán Gordon.
- `maurerAlgorithm(int bitLength)`: Tạo số nguyên tố sử dụng thuật toán Maurer.
- `randomStrongPrime(int N)`: Tạo số nguyên tố mạnh có độ dài `N`.

1.2 Class StrongPrime

Class có chứa hàm **BigInteger StrongPrimeGenerator.generate(int bitLength)** này dùng để sinh số nguyên tố lớn.

1.3 Class RSAsystem

Class này hiện thực các hàm để mã hóa và giải mã thông điệp theo hệ mã RSA.
Một số hàm tiêu biểu

- **Hàm BigInteger RSACryptoSystem.encrypt(BigInteger message)**: Để tính toán mã hóa $C = M^e \mod n$.
- **Hàm BigInteger RSACryptoSystem.decrypt(BigInteger encryptedMessage)**: Để tính toán giải mã $M = C^d \mod n$.
- **Hàm void RSACryptoSystem.generateKeys(int bitLength)**: Sinh ra cặp key mới.
- **Hàm void RSACryptoSystem.startEncrypt(String[] args)**: Mã hóa theo file và key đưa vào.
- **Hàm void RSACryptoSystem.startDecrypt(String[] args)**: Giải mã theo file và key đưa

1.4 Class App

Là class chứa hàm **void App.main(String[] args)** để chạy chương trình. Cụ thể được trình bày trong README.doc

2 Đánh giá

2.1 Ưu điểm

- **Tính chính xác và an toàn**

1. **Về việc sử dụng thư viện BigInteger:**

   Tính chính xác và an toàn là hai yếu tố quan trọng khi thực hiện các phép toán liên quan đến số nguyên tố lớn, như trong thuật toán RSA. Để đảm bảo tính chính xác, nhóm đã áp dụng một phương pháp hiệu quả bằng cách sử dụng lớp BigInteger trong ngôn ngữ lập trình Java.

   Lớp BigInteger cung cấp một cơ chế linh hoạt để thực hiện các phép toán trên số nguyên với độ chính xác tùy ý. Việc này tránh được những vấn đề có thể phát sinh từ sự tràn số hoặc mất chính xác khi thực hiện các phép toán với các số nguyên lớn. Điều này là quan trọng đặc biệt khi làm việc với các số nguyên tố lớn, nơi sự chính xác là yếu tố quyết định.

   Trong bối cảnh của thuật toán RSA, nơi yêu cầu về tính chính xác và an toàn là rất cao, việc sử dụng lớp BigInteger không chỉ giúp giải quyết những thách thức về tính toán mà còn tăng cường khả năng duy trì và quản lý các thông tin quan trọng liên quan đến khóa và phép toán mật mã. Điều này đóng góp vào việc xây dựng một hệ thống RSA mạnh mẽ, đáng tin cậy và đáp ứng được các yêu cầu cao cấp của an toàn thông tin.

   Bên cạnh đó, việc hiện thực lại các chức năng như lũy thừa, ước số chung lớn, kiểm tra số nguyên tố thay vì sử dụng hiện thực có sẵn trong "BigInteger" có thể gặp một số rủi ro về mặt hiệu suất và tính an toàn của thuật toán.

2. **Về việc sinh khóa:**

   Về quá trình sinh khóa trong hàm generateKeys, nhóm đã triển khai một phương pháp cực kỳ tỉ mỉ và hiệu quả để đảm bảo tính an toàn cao và khả năng chống lại các tấn công đối với việc sử dụng số nguyên tố yếu. Quy trình này bao gồm sự kết hợp của hai thuật toán quan trọng: thuật toán tạo số nguyên tố mạnh mẽ Maurer và kiểm tra số nguyên tố Miller-Rabin.

   Thuật toán tạo số nguyên tố mạnh mẽ (Maurer): Đầu tiên, nhóm sử dụng thuật toán Maurer, một thuật toán độc đáo và tiên tiến, để sinh ra một số nguyên tố mạnh mẽ. Sự mạnh mẽ ở đây không chỉ xuất phát từ việc số đó là số nguyên tố, mà còn từ tính chất rằng (p-1)/2 cũng là số nguyên tố, làm

tăng độ phức tạp của quá trình phân tích số nguyên tố, đồng thời cung cấp một cơ sở an toàn vững chắc.

Kiểm tra số nguyên tố Miller-Rabin: Để đảm bảo tính chắc chắn và tin cậy của số nguyên tố tạo ra, nhóm tiếp tục với bước kiểm tra số nguyên tố Miller-Rabin. Qua nhiều vòng lặp của thuật toán này với các số nguyên tố nhỏ, nhóm xác minh rằng số nguyên tố có độ tin cậy cao, ngăn chặn hiệu ứng của các tấn công như tấn công giả mạo Miller-Rabin.

Những bước này không chỉ đảm bảo tính chính xác trong việc sinh khóa mà còn nâng cao đáng kể độ an toàn của hệ thống trước các mối đe dọa tiềm ẩn và tấn công có thể xảy ra đối với khóa dựa trên số nguyên tố. Điều này làm tăng khả năng bảo vệ của thuật toán RSA, là một thành phần quan trọng trong cấu trúc an ninh của hệ thống.

- **Tính mở rộng**

  Tính mở rộng của mã nguồn của chương trình được xây dựng với sự tổ chức cực kỳ linh hoạt, sử dụng các class như "PublicKey" và "PrivateKey" để biểu diễn các khái niệm quan trọng. Sự phân chia này không chỉ giúp tạo cấu trúc rõ ràng mà còn tăng tính dễ bảo trì, giúp nhóm phát triển và duy trì mã nguồn một cách hiệu quả.

  Chương trình không chỉ hạn chế việc giao tiếp với người dùng thông qua giao diện đồ họa, mà còn hỗ trợ các tùy chọn dòng lệnh. Điều này mang lại ưu điểm lớn về tự động hóa và linh hoạt trong quá trình sử dụng chương trình. Việc có thể thực hiện các quy trình tự động từ dòng lệnh là một ưu điểm quan trọng, đặc biệt là trong các kịch bản tự động hóa và quy trình liên quan đến bảo mật.

  Tổng thể, chương trình không chỉ đáp ứng tốt các yêu cầu hiện tại mà còn có khả năng mở rộng linh hoạt để đáp ứng các yêu cầu và yêu cầu mới trong tương lai. Các khía cạnh như tính tương thích dòng lệnh và cấu trúc mã nguồn có thể dễ dàng được mở rộng và điều chỉnh để đáp ứng những thách thức mới và mở rộng chức năng của chương trình.

- **Về hiệu suất**

  1. Tích hợp Bảo mật cao và Sử dụng SecureRandom: Việc sử dụng thư viện SecureRandom không chỉ giúp chương trình tạo số nguyên ngẫu nhiên một cách an toàn mà còn đóng góp tích cực đến việc cải thiện tính bảo mật chung

của hệ thống. Mỗi khía cạnh của quá trình này được xem xét kỹ lưỡng để đảm bảo rằng dữ liệu được tạo ra và sử dụng đều đảm bảo an toàn.

2. Xử lý Hiệu quả với File và Dữ liệu Lớn: Chương trình của nhóm không chỉ giới hạn ở khả năng mã hóa và giải mã, mà còn tận dụng khả năng xử lý hiệu quả với file và dữ liệu lớn. Việc đọc và ghi từng phần của các tệp được thực hiện một cách thông minh để giảm bớt gánh nặng đối với bộ nhớ, đồng thời tối ưu hóa hiệu suất của chương trình.
3. Hỗ trợ Chức năng Phức tạp và Tính Mở rộng: Chương trình không chỉ giới thiệu đến người dùng những tính năng bảo mật mạnh mẽ mà còn hỗ trợ nhiều chức năng phức tạp. Từ việc tạo khóa đến quản lý khóa công khai và riêng, cũng như mã hóa và giải mã tệp tin, chương trình được xây dựng với tính mở rộng cao, đáp ứng linh hoạt đối với các yêu cầu và kịch bản sử dụng đa dạng của người dùng.

- **Việc tổ chức mã nguồn**

  Trong quá trình làm bài, nhóm đã đặc biệt chú trọng đến việc tổ chức mã nguồn sao cho nó không chỉ là hiệu quả mà còn dễ quản lý và tái sử dụng. Dưới đây là một số chi tiết về cách nhóm tổ chức mã nguồn và quản lý dự án:

  Chia thành Các Class Riêng Biệt: Để đảm bảo sự rõ ràng và dễ quản lý, nhóm đã chia chương trình thành nhiều class độc lập như "App", "MyMath", "PrivateKey", "PublicKey", và "RSACryptoSystem". Mỗi class này tập trung vào một phần nhỏ cụ thể của hệ thống, tạo điều kiện cho việc tái sử dụng mã nguồn một cách linh hoạt và hiệu quả.

  Sử Dụng Phương Thức Nhỏ: nhóm đã tích hợp việc sử dụng các phương thức nhỏ để thực hiện các chức năng cụ thể. Điều này không chỉ giúp mã nguồn trở nên rõ ràng và dễ đọc mà còn tạo ra khả năng tái sử dụng mã nguồn cao. Các phương thức này tập trung vào nhiệm vụ cụ thể, tối ưu hóa sự hiểu rõ và bảo trì.

  Cấu Trúc Logic với Các Class Phân Loại: Các class như "PrivateKey", "PublicKey", và "RSACryptoSystem" không chỉ đơn thuần là đại diện cho các khái niệm trong mã nguồn mà còn tạo ra một tổ chức logic tốt. Cấu trúc này rõ ràng phản ánh cấu trúc của hệ thống RSA, làm cho quá trình theo dõi mã nguồn, đặc biệt là trong quá trình mã hóa và giải mã, trở nên dễ dàng và có tổ chức.

2.2 Nhược điểm và hướng cải thiện

1. **Về tính bảo mật**

   Một trong những nhược điểm quan trọng là thiếu quá trình kiểm tra lỗi đầu vào. Quá trình này không chỉ đơn giản là một bước kiểm tra đơn thuần, mà là một yếu tố quan trọng đối với bảo mật hệ thống. Điều này có thể tạo điều kiện thuận lợi cho những lỗ hổng bảo mật nếu người dùng nhập vào dữ liệu không hợp lệ hoặc có độ dài không đúng.

   Với việc thiếu kiểm tra lỗi đầu vào, chương trình trở nên dễ bị tấn công bởi các kỹ thuật như injection, nơi kẻ tấn công có thể chèn dữ liệu độc hại để thực hiện các hành động không mong muốn. Điều này có thể dẫn đến việc lợi dụng chương trình để thực hiện các hành động như đánh cắp thông tin, thực hiện các cuộc tấn công khác, hoặc đơn giản là làm gián đoạn hoạt động của hệ thống.

   Đặc biệt, khi xử lý các thông tin nhạy cảm như khóa mã hóa, việc kiểm soát và chặn các đầu vào không hợp lệ trở nên vô cùng quan trọng. Một việc làm thiếu sót trong việc kiểm tra lỗi đầu vào có thể dẫn đến những hậu quả nghiêm trọng đối với tính bảo mật của hệ thống, tăng nguy cơ bị tấn công và lộ thông tin quan trọng.

   Một khuyết điểm đáng chú ý khác là sự thiếu sót trong việc kiểm tra các key đặc biệt. Chương trình không thực hiện quá trình kiểm tra cẩn thận đối với các key có giá trị đặc biệt, chẳng hạn như key có giá trị là 0 hay 1. Điều này mở ra khả năng xuất hiện vấn đề lớn trong quá trình mã hóa và giải mã.

   Việc không kiểm tra đối với các key đặc biệt có thể dẫn đến các tình huống không mong muốn khi sử dụng hệ thống. Chẳng hạn, key có giá trị là 0 có thể tạo ra mã hóa không hiệu quả, vì mọi thông điệp mã hóa đều sẽ trở thành 0. Ngược lại, key có giá trị là 1 cũng có thể tạo ra mã hóa dễ bị dự đoán, làm giảm khả năng bảo mật của hệ thống.

   Quá trình kiểm tra các key đặc biệt là quan trọng để đảm bảo tính độc lập và ngẫu nhiên của key, từ đó tăng cường độ an toàn của hệ thống mã hóa. Thiếu sót này có thể dẫn đến việc môi trường mã hóa trở nên dễ dàng bị tấn công và làm suy giảm khả năng bảo mật chung của dự án.

2. **Việc xử lý ngoại lệ chưa tốt**:

Xử lý ngoại lệ chưa được thực hiện một cách hiệu quả. Hiện tại, chương trình chỉ đơn giản là in ra stack trace khi phát sinh lỗi mà không kèm theo một thông điệp lỗi rõ ràng, điều này có thể gây khó khăn cho người dùng trong việc hiểu vấn đề xảy ra.

Việc chỉ in ra stack trace mà không cung cấp một thông điệp lỗi mô tả chính xác và dễ hiểu có thể làm tăng khả năng khó hiểu vấn đề. Người dùng không nhận được đủ thông tin để đánh giá và xác định lý do của lỗi, điều này có thể dẫn đến thời gian dễ mắc kẹt trong việc gỡ lỗi và sửa chữa.

Để cải thiện trải nghiệm người dùng và quản lý lỗi một cách hiệu quả hơn, nên xử lý ngoại lệ sao cho chương trình không chỉ in ra stack trace mà còn kèm theo thông điệp lỗi chi tiết. Thông điệp này nên được thiết kế sao cho dễ hiểu, chính xác và cung cấp hướng dẫn rõ ràng về cách khắc phục vấn đề. Điều này sẽ giúp người dùng và nhà phát triển dễ dàng xác định và sửa lỗi một cách nhanh chóng, giảm thiểu thời gian mất mát trong quá trình phát triển và triển khai ứng dụng.

3. **Về hiệu suất**

   Vẫn tồn tại một số điểm cần cải thiện để đạt được hiệu suất tối ưu hóa hơn. Đặc biệt, khi xử lý các tệp tin lớn hoặc key lớn, chưa có cơ chế tối ưu hiệu suất đặc biệt cho những tình huống này.Việc xử lý các tệp tin lớn đòi hỏi một chiến lược đặc biệt để giảm thiểu tải công việc tính toán và tối ưu hóa bộ nhớ. Mặt khác, khi làm việc với key lớn, việc tối ưu hiệu suất là một thách thức khác cần được xem xét. Có thể xem xét sự áp dụng các kỹ thuật như tinh chỉnh tham số thuật toán hoặc sử dụng các thư viện tối ưu hóa để cải thiện tốc độ xử lý.

   Để giải quyết vấn đề này, có thể xem xét triển khai cơ chế tối ưu hiệu suất chuyên biệt, chẳng hạn như sử dụng các phương pháp chia nhỏ tệp tin hoặc phân mảnh key để giảm thiểu áp lực tính toán trong quá trình thực thi. Các chiến lược như lập lịch tác vụ xử lý lớn vào những khoảng thời gian ít bận rộn cũng có thể giúp tối ưu hóa hiệu suất mà không làm ảnh hưởng đến trải nghiệm người dùng.

   Qua đó, việc xác định và triển khai các cơ chế tối ưu hiệu suất sẽ đảm bảo rằng chương trình có khả năng xử lý hiệu quả trong mọi tình huống, đồng thời cung cấp trải nghiệm người dùng mượt mà và không gặp trở ngại khi làm việc với dữ liệu lớn hay key có kích thước đáng kể.

4. **Chưa có chức năng quản lý Key**

Sự thiếu hụt chức năng quản lý key trong chương trình là một hạn chế đáng kể và có thể tạo ra những thách thức trong việc quản lý và duy trì hệ thống mật mã. Hiện tại, người dùng không có khả năng tạo mới, lưu trữ, hay thậm chí xóa các key một cách thuận tiện thông qua giao diện của chương trình.

Quản lý key là một phần quan trọng của quá trình triển khai và duy trì hệ thống mật mã an toàn. Nhu cầu phải thường xuyên thay đổi key, tạo mới key cho các mục đích cụ thể, hay thậm chí quản lý quyền truy cập vào key là những yếu tố quan trọng mà chương trình hiện tại chưa đáp ứng.

Vấn đề này có thể tạo ra những khó khăn trong việc quản lý bảo mật và thực hiện các biện pháp bảo mật nhất quán. Đối với các tổ chức hoặc người dùng cá nhân quản lý nhiều key, khả năng tạo và lưu trữ chúng một cách an toàn và tiện lợi là rất quan trọng.

Đề xuất cải thiện chương trình bằng cách thêm vào các chức năng quản lý key, bao gồm khả năng tạo mới key, lưu trữ chúng một cách an toàn trong hệ thống, và thậm chí cả khả năng xóa key khi chúng không còn cần thiết. Điều này sẽ cung cấp cho người dùng một phương tiện hiệu quả để quản lý và duy trì hệ thống mật mã của mình một cách chặt chẽ và linh hoạt.

5. **Về khả năng mở rộng mã nguồn**

Hạn chế về kích thước key trong chương trình là một vấn đề quan trọng cần xem xét và cải thiện. Hiện tại, chương trình chỉ hỗ trợ key size cố định là 1024 bits hoặc 2048 bits. Điều này mang lại một số hạn chế đáng kể đối với khả năng linh hoạt và mở rộng của chương trình.

Trong thực tế, yêu cầu bảo mật ngày càng gia tăng, và việc sử dụng các key có kích thước lớn hơn thường được khuyến khích để ngăn chặn các cuộc tấn công mạng ngày càng phức tạp. Tuy nhiên, với chương trình hiện tại, người dùng đối mặt với sự hạn chế đối với việc sử dụng các key có kích thước khác nhau ngoài 1024 bits hoặc 2048 bits.

Điều này có thể tạo ra một thách thức đối với những người dùng có nhu cầu sử dụng các key có kích thước khác nhau tùy thuộc vào ngữ cảnh cụ thể của họ. Đối với những hệ thống yêu cầu mức độ bảo mật cao hơn, việc hỗ trợ key size lớn hơn là cực kỳ quan trọng để đảm bảo tính toàn vẹn và an toàn của thông tin.

Để khắc phục vấn đề này, có thể xem xét việc cải thiện chương trình để hỗ trợ một

loạt các key sizes, bao gồm cả những key có kích thước lớn hơn để đáp ứng các yêu cầu bảo mật hiện đại. Điều này sẽ cung cấp cho người dùng sự linh hoạt hơn khi triển khai chương trình trong các môi trường yêu cầu mức độ bảo mật cao.

V Kết luận

Trong quá trình thực hiện đề tài "Hiện Thực Hệ Mã RSA", nhóm đã tiếp cận một loạt các khía cạnh của mật mã và bảo mật thông tin. Dưới đây là một phân tích chi tiết về những điểm quan trọng mà nhóm rút ra từ dự án này.

Nhóm không chỉ áp dụng lý thuyết mật mã vào thực tế mà còn hiểu rõ về nguyên lý hoạt động của thuật toán RSA. Việc này đã mở ra cánh cửa cho nhóm khám phá sâu hơn về cách mà mật mã khóa công khai có thể được tích hợp và áp dụng hiệu quả trong thế giới thực. Nhóm nhận thức được sự phức tạp và sức mạnh của thuật toán này mà còn thấu hiểu được vai trò to lớn mà nó đóng trong lĩnh vực bảo mật thông tin. Thuật toán RSA không chỉ là một công cụ mã hóa khóa công khai phổ biến, mà còn là biểu tượng của sự an toàn và tin cậy trong việc bảo vệ thông tin quan trọng.

Một trong những điểm quan trọng nhất của RSA là khả năng sử dụng cặp khóa công khai và khóa riêng tư, mỗi khóa đều có những chức năng đặc biệt. Khóa công khai được chia sẻ rộng rãi và dùng để mã hóa dữ liệu, trong khi khóa riêng tư được giữ chặt và chỉ dùng để giải mã. Sự kết hợp của hai loại khóa này tạo nên một hệ thống bảo mật mạnh mẽ, nơi mà nguyên tắc của số nguyên tố lớn đóng vai trò quan trọng. Điều này giúp RSA chống lại nhiều loại tấn công phổ biến, bao gồm cả việc phân tích theo dõi và tìm kiếm khóa. Thuật toán Maurer và kiểm tra số nguyên tố Miller-Rabin trong quá trình sinh khóa là những lớp bảo vệ bổ sung, tăng cường độ an toàn và khả năng chống lại các tấn công liên quan đến số nguyên tố yếu.

Nhóm đã thực hiện quá trình sinh khóa trong hệ mã RSA dựa trên kiến thức lý thuyết vững chắc về số nguyên tố, thuật toán tạo số nguyên tố và phương pháp kiểm tra số nguyên tố. Điều này không chỉ là một thách thức mà còn là cơ hội mở rộng hiểu biết của nhóm về ứng dụng thực tế của các khái niệm này. Đồng thời tổ chức mã nguồn sao cho nó không chỉ đơn giản là một hiện thực cơ bản của hệ mã RSA mà còn có tính mở rộng và khả năng bảo trì. Điều này giúp chương trình có thể phục vụ nhiều mục đích và dễ dàng tích hợp với các phần mềm bảo mật khác nhau trong tương lai. Nhóm cũng đã chú ý đến hiệu suất và an toàn của chương trình bằng cách sử dụng thư viện BigInteger trong Java và tích hợp thuật toán Maurer cùng kiểm tra số nguyên tố Miller-Rabin. Những cải tiến này đã nâng cao độ chính xác và độ an toàn của quá trình sinh khóa và sử dụng khóa.

Tóm lại, đề tài này không chỉ giúp nhóm ứng dụng kiến thức lý thuyết mà còn mở rộng cái nhìn về mật mã và bảo mật thông tin. Nhóm hy vọng rằng sau bài tập này,

những thành viên trong nhóm sẽ có thêm kỹ năng và hiểu biết về lĩnh vực quan trọng này.

Tài liệu tham khảo

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.

[2] E. Maiwald, "Network Security: A Beginner's Guide," McGraw-Hill, 2013.

[3] W. Stallings, "Cryptography and Network Security: Principles and Practice, Global Edition," Pearson, 2017.

[4] Wikipedia, "RSA (cryptosystem)," https://en.wikipedia.org/wiki/RSA_(cryptosystem).

[5] Wikipedia, "Strong Prime," https://en.wikipedia.org/wiki/Strong_prime.

[6] Wikipedia, "Miller–Rabin primality test," https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test.

[7] GeeksforGeeks - Java Program to Implement the RSA Algorithm, https://www.geeksforgeeks.org/java-program-to-implement-the-rsa-algorithm/

[8] GitHub - RSA Algorithm in Java, https://github.com/topics/rsa-algorithm?l=java

[9] Baeldung - RSA in Java, https://www.baeldung.com/java-rsa

[10] Medium - RSA Encryption with Java Example, https://medium.com/@asoldan1459/rsa-encryption-with-java-example-c28d042817cb

[11] DevGlan - RSA Encryption and Decryption in Java, https://www.devglan.com/java8/rsa-encryption-decryption-java

Phụ lục

Phụ lục A: Ngôn ngữ lập trình và Môi trường Phát triển

Ngôn ngữ lập trình sử dụng trong dự án là Java và môi trường phát triển được sử dụng là VSCode.

Phụ lục B: Kiến thức Toán về Module và Xử lý Số Nguyên Tố Lớn

Trình bày chi tiết kiến thức toán liên quan đến module và xử lý số nguyên tố lớn, những khái niệm cơ bản và công thức được áp dụng trong triển khai hệ thống mã hóa RSA.

Phụ lục C: Mã Hóa RSA - Mã Nguồn Đầy Đủ

Nơi cung cấp toàn bộ mã nguồn của hệ thống mã hóa RSA, triển khai bằng ngôn ngữ lập trình Java.