

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



MẬT MÃ & AN NINH MẠNG (CO3069) - L01

Bài tập lớn:

HIỆN THỰC HỆ MÃ RSA TRÊN JAVA

Giảng viên hướng dẫn: ThS. Nguyễn Cao Đạt

Nhóm sinh viên: Chế Quang Huy - 1812340

Trần Văn Tâm - 1713057

Phạm Thế Khang - 1812558

Võ Phạm Long Huy - 1812438

Tp. Hồ Chí Minh, Tháng 5/2023

Mục lục

1	Giới thiệu	4
1.1	Đề tài	4
1.2	Mục tiêu cần thực hiện	4
2	Phân tích công nghệ được sử dụng	5
2.1	Cơ sở lý thuyết	5
2.2	Mã hóa RSA hoạt động	6
2.2.1	Hàm Trapdoor	6
2.2.2	Tạo số nguyên tố	6
2.2.3	Tạo khóa	6
2.2.4	Mã hóa	7
2.2.5	Giải hóa	7
2.3	Kiểm tra Miller - Rabin	8
2.3.1	Tiêu chuẩn kiểm tra $Q(n, a)$	8
2.3.2	Số giả nguyên tố	9
2.3.3	Giải thuật kiểm tra Miller - Rabin	9
2.3.4	Kiểm tra Miller - Rabin lặp	10
3	Hiện thực và đánh giá hệ thống	11
3.1	Hiện thực	11
3.1.1	Các hàm thực hiện	11
3.1.2	Demo	19
3.2	Đánh giá	20
4	Kết luận	21
4.1	Chương trình đã làm được	21
4.2	Chương trình chưa làm được	21
4.3	Ưu điểm	21
4.4	Nhược điểm	21
5	Tài liệu tham khảo	22

1 Giới thiệu

1.1 Đề tài

Hiện thực hệ mã RSA trên Java chỉ dùng:

- Java có lớp BigInteger được xây dựng sẵn.
- C++ có thư viện NTL (Library for doing Number Theory) or GMP (the GNU Multiple Precision Arithmetic Library).

Trong hiện thực RSA của bạn, giả sử các số nguyên tố lớn ít nhất phải 500 bits (nhưng có thể lớn hơn) và bạn phải viết các hàm sau:

- 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 khóa 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 .

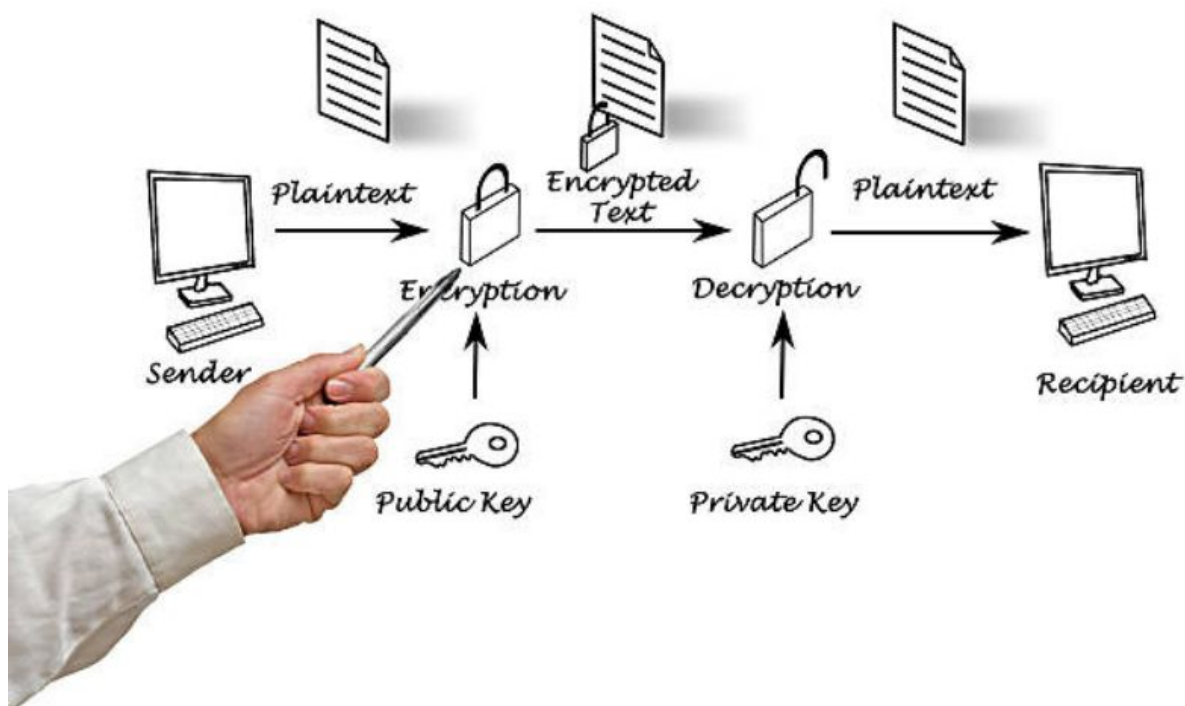
1.2 Mục tiêu cần thực hiện

- Tìm hiểu về hệ mã RSA
- Tìm hiểu và cách dùng các thư viện NTL (Library for doing Number Theory) or GMP (the GNU Multiple Precision Arithmetic Library).
- Xem lại các phương thức (GCD, Power, tìm số nguyên tố,...)
- Nghiên cứu về cách tạo bộ khóa, mã hóa và giải mã thông điệp cụ thể.

2 Phân tích công nghệ được sử dụng

2.1 Cơ sở lý thuyết

RSA là một thuật toán mật mã hóa khóa công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hóa. Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khóa công cộng. RSA đang được sử dụng phổ biến trong thương mại điện tử và được cho là đảm bảo an toàn với điều kiện độ dài khóa đủ lớn.



Hình II.1: Mô hình RSA

Một giải pháp để ngăn chặn kẻ nghe trộm và truy cập nội dung tin nhắn là mã hóa nó. Về cơ bản, điều này có nghĩa là thêm một mã vào tin nhắn, nó sẽ làm cho tin nhắn thành một mớ hỗn độn. Nếu mã của bạn đủ phức tạp, thì những người duy nhất có thể truy cập vào thư gốc là những người có quyền truy cập vào mã.

Theo mã hóa RSA, các tin nhắn được mã hóa bằng một mã gọi là public key, mã này có thể được chia sẻ công khai. Do một số đặc thù tính toán khoa học khác biệt của thuật toán RSA, một khi một thông điệp đã được mã hóa bằng public key, nó chỉ có thể được giải mã bằng một key được gọi là private key. Mỗi người dùng RSA có cặp key bao gồm public key và private key của riêng họ. Private key cần được giữ bí mật.

2.2 Mã hóa RSA hoạt động

Chúng bao gồm hàm Trapdoor, tạo số nguyên tố, hàm phi Carmichael. Và các quy trình riêng biệt liên quan đến việc tính toán các public key và private key được sử dụng trong quá trình mã hóa và giải mã.

2.2.1 Hàm Trapdoor

Hàm Secure Trapdoor là một hàm Trapdoor trong đó F là hàm một chiều, tức là có thể dễ dàng tính toán $y = F(pk, x) \forall x \in X$ nhưng không thể tính được x từ y mà không có khóa bí mật sk hay xác suất tính được x từ y mà không có khóa bí mật sk là không đáng kể.

2.2.2 Tạo số nguyên tố

Các chức năng của hàm Trapdoor đã được đề cập ở trên tạo cơ sở cho cách thức hoạt động của các lược đồ mã hóa public key và private key. Các thuộc tính của chúng cho phép chia sẻ public key mà không gây nguy hiểm cho tin nhắn hoặc tiết lộ private key. Chúng cũng cho phép dữ liệu được mã hóa bằng một key theo cách mà chỉ có thể giải mã bằng key khác.

Các số nguyên tố trong RSA cần phải rất lớn và cũng tương đối xa nhau. Các số nhỏ hoặc gần nhau sẽ dễ bị bẻ khóa hơn nhiều. Mặc dù vậy, ví dụ của chúng tôi sẽ sử dụng các số nhỏ hơn để làm cho mọi thứ dễ theo dõi và tính toán hơn.

Giả sử kiểm tra tính nguyên tố cho các số nguyên tố mà chúng ta đã sử dụng ở trên, 907 và 773. Bước tiếp theo là khám phá module (n), sử dụng công thức sau:

$$n = p \times q$$

$$\text{Where } p = 907 \text{ and } q = 773$$

Therefore:

$$n = 907 \times 773$$

$$n = 701,111$$

2.2.3 Tạo khóa

Giả sử An và Bình cần trao đổi thông tin bí mật thông qua một kênh không an toàn (ví dụ như Internet). Với thuật toán RSA, An đầu tiên cần tạo ra cho mình cặp khóa gồm khóa công khai và khóa bí mật theo các bước sau:

- Chọn 2 số nguyên tố lớn p và q với $p \neq q$, lựa chọn ngẫu nhiên và độc lập.
- Tính: $n = pq$
- Tính: giá trị hàm số Euler $\phi(n) = (p - 1)(q - 1)$

- Chọn một số tự nhiên e sao cho $1 < e < \phi(n)$ và là số nguyên tố cùng nhau với $\phi(n)$.
- Tính: d sao cho $de \equiv 1 \pmod{\phi(n)}$.

Một số lưu ý:

- Các số nguyên tố thường được chọn bằng phương pháp thử xác suất.
- Các bước 4 và 5 có thể được thực hiện bằng giải thuật Euclid mở rộng (xem thêm số học môđun).
- Bước 5 có thể viết cách khác:
Tìm số tự nhiên x , sao cho $d = \frac{x(p-1)(q-1) + 1}{e}$ cũng là số tự nhiên.
Khi đó sử dụng giá trị $d \bmod (p-1)(q-1)$

Khóa công khai bao gồm:

- n : môđun.
- e : số mũ công khai (cũng gọi là số mũ mã hóa).

Khóa bí mật bao gồm:

- n : môđun, xuất hiện cả trong khóa công khai và khóa bí mật.
- d : số mũ bí mật (cũng gọi là số mũ giải mã).

Một dạng khác của khóa bí mật bao gồm:

- p và q : hai số nguyên tố chọn ban đầu.
- $d \bmod (p-1)$ và $d \bmod (q-1)$: thường được gọi là d_{mp1} và d_{mq1} .
- $(1/q) \bmod p$: thường được gọi là i_{qmp} .

2.2.4 Mã hóa

Giả sử Bình muốn gửi đoạn thông tin M cho An. Đầu tiên Bình chuyển M thành một số $m < n$ theo một hàm có thể đảo ngược (từ m có thể xác định lại M) được thỏa thuận trước. Quá trình này được mô tả ở phần *#Chuyển đổi văn bản rõ*.

Lúc này Bình có m và biết n cũng như e do An gửi. Bình sẽ tính c là bản mã hóa của m theo công thức:

$$c = m^e \bmod n$$

2.2.5 Giải hóa

An nhận c từ Bình và biết khóa bí mật d . An có thể tìm được m từ c theo công thức sau:

$$m = c^d \bmod n$$

Biết m , An tìm lại M theo phương pháp đã thỏa thuận trước. Quá trình giải mã hoạt động vì ta có:

$$c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$

Do $ed \equiv 1 \pmod{p-1}$ và $ed \equiv 1 \pmod{q-1}$, theo Định lý Fermat nhỏ nên:

$$m^{ed} \equiv m \pmod{p}$$

$$m^{ed} \equiv m \pmod{q}$$

Do p và q là hai số nguyên tố cùng nhau, áp dụng định lý số dư Trung Quốc, ta có:

$$m^{ed} \equiv m \pmod{pq}$$

$$c^d \equiv m \pmod{n}$$

2.3 Kiểm tra Miller - Rabin

Kiểm tra Miller - Rabin là một thuật toán xác suất để kiểm tra tính nguyên tố cũng như các thuật toán kiểm tra tính nguyên tố: Kiểm tra Fermat và Kiểm tra Solovay - Strassen.

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 $a \in A \subset \mathbb{N}$ và kiểm tra xem chúng có đúng với số n muốn kiểm tra và một số $a \in A$ được chọn ngẫu nhiên hay không. Nếu mệnh đề $Q(p, a)$ không đúng, tất yếu n không phải là số nguyên tố, còn nếu $Q(p, a)$ 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.

2.3.1 Tiêu chuẩn kiểm tra $Q(n, a)$

Trước hết là một bổ đề về căn bậc hai của đơn vị trong trường hữu hạn \mathbb{Z}_p , trong đó p là số nguyên tố. Chắc chắn rằng 1 và -1 luôn là các căn bậc hai của 1 theo module 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 module p .

Khi đó:

$$x^2 \equiv 1 \pmod{p}$$

$$x^2 - 1 \equiv 0 \pmod{p}$$

$$(x - 1)(x + 1) \equiv 0 \pmod{p}$$

Từ đó, $x - 1$ hoặc $x + 1$ là chia hết cho p .

Bây giờ giả sử p là một số nguyên tố lẻ, khi đó $p - 1$ là một số chẵn và ta có thể viết $p - 1$ dưới dạng $2^s \cdot m$, 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 $\{1, 2, \dots, p - 1\}$.

Xét dãy số $x_k = a^{2^k \cdot m}$ với $k = 0, 1, 2, \dots, s$. Khi đó $x_k = (x_{k-1})^2$, với $k = 1, 2, \dots, s$ và $x_s = a^{p-1}$

Từ định lý Fermat nhỏ:

$$a^{p-1} \equiv 1 \pmod{p} \text{ hay}$$

$$x_s \equiv 1 \pmod{p} \text{ hay}$$

$$x_{s-1}^2 \equiv 1 \pmod{p}$$

Nếu $x_{s-1} \equiv -1 \pmod{p}$ thì ta dừng lại.

Nếu $x_{s-1} \equiv 1 \pmod{p}$ thì ta tiếp tục với x_{s-2} , sau một số hữu hạn bước.

- hoặc ta có một chỉ số k , $0 \leq k \leq s-1$ sao cho $x_k \equiv -1 \pmod{p}$,
- hoặc tới $k = 0$ ta vẫn có $x_k \equiv 1 \pmod{p}$.

Ta có mệnh đề $Q(p, a)$ như sau:

Nếu p là số nguyên tố lẻ và $p-1 = 2^s \cdot m$ thì $\forall a : 0 < a < p-1$:

- hoặc $x_k = a^{2^k \cdot m} \equiv 1 \pmod{p}$, $\forall k = 0, 1, 2, \dots, s$
- hoặc tồn tại k : $0 \leq k \leq s$ sao cho $x_k = a^{2^k \cdot m} \equiv -1 \pmod{p}$.

2.3.2 Số giả nguyên tố

- Theo định lý Fermat nhỏ, với số nguyên tố p ta có $\forall a \in \{1, 2, \dots, p-1\} : a^{p-1} \equiv 1 \pmod{p}$
- Định nghĩa: Hợp số n thỏa mãn $a^{n-1} \equiv 1 \pmod{n}$ 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 $\forall a \in \{1, 2, \dots, n\} : \text{GCD}(a, n) = 1$ được gọi là số Carmichael.
- Định nghĩa: Hợp số n được gọi là số giả nguyên tố mạnh Fermat cơ sở a nếu nó thỏa mãn mệnh đề $Q(n, a)$.

2.3.3 Giải thuật kiểm tra Miller - Rabin

Định lý: Nếu n là hợp số dương lẻ thì trong các số $a \in \{2, \dots, n-1\}$ tồn tại không quá $\frac{n-1}{4}$ 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 $\frac{1}{4}$, nghĩa là $P(B/A) \leq \frac{1}{4}$. Tuy nhiên để tính xác suất sai của kiểm tra Miller - Rabin cần tính xác suất điều kiện $P(A/B)$. Dựa trên định lý về ước lượng số các số nguyên tố ta đưa ra ước lượng:

$$P(A) \approx 1 - \frac{2}{\ln n} \approx \frac{\ln n - 2}{\ln n}$$

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

$$P(A/B) = \frac{P(B|A) * P(A)}{P(B)} = \frac{P(B|A) * P(A)}{P(B|A) * P(A) + P(B|\neg A) * P(\neg A)}$$

$P(A)$ đã biết ở trên, $P(B|A) \leq \frac{1}{4}$, còn $P(B|\neg A) = 1$ vì khi n là số nguyên tố thì chắc chắn mệnh đề $Q(n, a)$ là đúng và $P(\neg A) = 1 - P(A) = \frac{2}{\ln n}$

Vì vậy:

$$P(A/B) = \frac{P(B|A) * P(A)}{P(B|A) * P(A) + P(\neg A)} \approx \frac{P(B|A) * (\ln n - 2)}{P(B|A) * (\ln n - 2) + 2}$$

2.3.4 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 $P(B|A) \leq \frac{1}{4^k}$ khi thay vào công thức với 50 lần thử nếu cả 50 lần, phép thử đều "dương tính" thì xác suất sai giảm xuống chỉ còn là một số rất nhỏ không vượt quá 9.10^{-29} .

3 Hiện thực và đánh giá hệ thống

3.1 Hiện thực

3.1.1 Các hàm thực hiện

Hàm `static BigInteger GCD(BigInteger a, BigInteger b)`: dùng để tìm ước chung lớn nhất. Ước chung lớn nhất (GCD, viết tắt của từ Greatest Common Divisor) của hai hay nhiều số là số nguyên dương lớn nhất mà là ước chung (Common Divisor) của tất cả các số đó.

Hàm được hiện thực bằng giải thuật Euclid, lấy hai số chia lấy dư của nhau cho đến khi tìm được số dư là 0 thì trả về số bị chia.

```
static BigInteger GCD(BigInteger a, BigInteger b) {  
    while (true) {  
        if (a.compareTo(BigInteger.ZERO) == 0) return b;  
        if (b.compareTo(BigInteger.ZERO) == 0) return a;  
        if (a.compareTo(b) == 1) {  
            a = a.mod(b);  
        } else {  
            b = b.mod(a);  
        }  
    }  
}
```

Hàm `static BigInteger Pow(BigInteger a, BigInteger b, BigInteger n)`: dùng để tìm $a^b \bmod n$, đối với 2 số a, b thuộc tập $[1, 2^{512}]$ thì việc tính trước a^b rồi mới lấy dư n là việc không thể do chi phí tính toán là rất lớn và tốn rất nhiều tài nguyên bộ nhớ.

Do đó hàm được hiện thực theo các bước:

- Nhận giá trị cơ số a , mũ b , số chia lấy dư n .
- Đặt $res = 1$
- Mỗi vòng lặp (lặp cho đến khi $b = 1$)

– Nếu b chẵn:

$$* b = b/2$$

$$* a = a^2 \bmod n$$

– Nếu b lẻ:

$$* b = (b - 1)/2$$

$$* a = a^2 \bmod n$$

$$* res = res \times a \bmod n$$

- Trả về res là kết quả của $a^b \bmod n$

```
static BigInteger Pow(BigInteger a, BigInteger b, BigInteger n) {
    BigInteger two = new BigInteger("2");
    if (b.compareTo(BigInteger.ZERO) == 0) {
        return BigInteger.ONE;
    }
    BigInteger res = Pow(a, b.divide(two), n);
    if (b.mod(two).compareTo(BigInteger.ZERO) == 0) {
        return res.multiply(res).mod(n);
    }
    return res.multiply(res).mod(n).multiply(a).mod(n);
}
```

Hàm `static BigInteger[] Inverse(BigInteger a, BigInteger b)`: dùng để trả về trả về x, y sao cho $a * x - b * y = 1$ với $x \leq b$ và $y \leq a$ và x, y là nguyên tố cùng nhau.

Việc tìm x, y được hiện thực bằng giải thuật Euclid mở rộng. Cho a, b :

- Nếu $b = 1$: trả về cặp $\{x, y\}$ là $\{1, a - 1\}$
- Nếu $a = 0$: trả về cặp $\{x, y\}$ là $\{1, 0\}$
- Nếu $a > b$: Dùng ý tưởng đệ quy dựa vào tính chất:
 - Nếu tìm được cặp x_1, y_1 thỏa $b.x_1 + (a \bmod b).y_1 = g$ (g bất kỳ) thì x, y sẽ có giá trị là:

$$x = y_1$$
$$y = x_1 - y_1 \cdot \lfloor a/b \rfloor$$

```
static BigInteger[] Inverse(BigInteger a, BigInteger b) {
    if (b.compareTo(BigInteger.ONE) == 0) {
        return new BigInteger[] {
            BigInteger.ONE,
            a.subtract(BigInteger.ONE)
        };
    }
    if (a.compareTo(BigInteger.ONE) == 0) {
        return new BigInteger[] {
            BigInteger.ONE,
            BigInteger.ZERO
        };
    }
    if (a.compareTo(b) != -1) {
```

```
        BigInteger[] vals = Inverse(a.mod(b), b);
        return new BigInteger[] {
            vals[0],
            a.divide(b).multiply(vals[0]).add(vals[1])
        };
    }
    BigInteger[] vals = Inverse(a, b.mod(a));
    return new BigInteger[] {
        b.divide(a).multiply(vals[1]).add(vals[0]),
        vals[1]
    };
}
```

Hàm `static BigInteger PhiInv(BigInteger e, BigInteger p, BigInteger q)`:

Tìm nghịch đảo modulo $\phi(n)$ của e gọi là d với $n = pq$, và giá trị $\phi(n) = (p - 1) * (q - 1)$.

Trong hiện thực mã hóa RSA thì:

- p, q là hai giá trị nguyên tố lớn được chọn ngẫu nhiên (512, 1024 hoặc 2048 bit).
- $\phi(n)$ là số nguyên tố cùng nhau với n nằm trong đoạn $[1, n - 1]$.
- e là một giá trị được chọn sao cho thỏa tính chất nguyên tố cùng nhau với $\phi(n)$
- d là nghịch đảo modulo $\phi(n)$ của e .
- Khóa công khai là (e, n) khóa bí mật là (d, n) .

```
static BigInteger PhiInv(BigInteger e, BigInteger p, BigInteger q) {
    BigInteger pminus1 = p.subtract(BigInteger.ONE);
    BigInteger qminus1 = q.subtract(BigInteger.ONE);
    BigInteger phi = pminus1.multiply(qminus1);
    BigInteger[] arr = Inverse(e, phi);
    BigInteger d = arr[0];
    return d;
}
```

Hàm `static BigInteger RandomBigInteger(BigInteger n)`:

Tìm một số nguyên ngẫu nhiên nhỏ hơn n cho trước.

```
static BigInteger RandomBigInteger(BigInteger m, BigInteger n) {
    Random randNum = new Random();
    int len = n.bitLength();
```

```
BigInteger res = new BigInteger(len, randNum);  
while ((res.compareTo(n) != -1) || (res.compareTo(m) == -1)) {  
    randNum = new Random();  
    res = new BigInteger(len, randNum);  
}  
return res;  
}
```

Hàm kiểm Miller - Rabin nhận vào n , a . Trong đó n là giá trị cần kiểm là số nguyên hay không, a là một giá trị được chọn để kiểm thử tính chất (để kiểm tra n có phải nguyên tố sẽ cần có nhiều giá trị a).

Hàm được hiện thực theo các bước:

- Nhận input là n với a .
- Tính S , d thỏa $2^S \cdot d = n - 1$ và d lẻ.
- Gán $x = \frac{n-1}{2^S}$
- Lặp với i từ 0 đến $n-1$
 - $y = x^2 \bmod n$
 - Kiểm tra nếu bản thân $y = 1 \bmod n$ mà $x \neq 1 \bmod n$ (tức $x = -1 \bmod n$) và $x \neq n-1$ thì hàm Miller - Rabin trả kết sai, n chắc chắn không phải là số nguyên tố.
 - Không thì gán $x = y$ và tiếp tục vòng lặp.
- Kết thúc vòng lặp kiểm tra $y \neq -1$ thì trả về false còn nếu $y = 1$ thì tức là n có 25% khả năng là số nguyên tố a là căn bậc 2^S của $1 \bmod n$ và a là một giả nguyên tố mạnh Ferma.

```
static boolean Miller_Rabin (BigInteger n, BigInteger a) {  
    BigInteger nminus1 = n.subtract(BigInteger.ONE);  
    BigInteger two = new BigInteger("2");  
    BigInteger copy = n.subtract(BigInteger.ONE);  
    int s = 0;  
    while (copy.mod(two).compareTo(BigInteger.ZERO) == 0) {  
        s++;  
        copy = copy.divide(two);  
    }  
    BigInteger y = BigInteger.ONE;  
    BigInteger d = nminus1.divide(two.pow(s));  
    BigInteger x = Pow(a, d, n);  
    for (int i = 0; i < s; i++) {  
        y = Pow(x, two, n);  
    }
```

```
        if ((y.compareTo(BigInteger.ONE) == 0) && (x.compareTo(BigInteger.ONE) != 0) &&
(x.compareTo(nminus1) != 0)){
            return false;
        }
        x = y;
    }
    if (y.compareTo(BigInteger.ONE) != 0) {
        return false;
    }
    return true;
}
```

Hàm kiểm tra số nguyên tố sử dụng giải thuật Miller - Rabin. Hàm hiện thực gồm các bước:

- Nhận n là số nguyên tố cần kiểm tra.
- Lặp 15 lần
 - Chọn một số a bất kỳ nhỏ hơn n .
 - Kiểm tra nếu $Q(n, a)$ trả về false hoặc a và n không nguyên tố cùng nhau thì trả về kết quả n là hợp số.

Nếu cả 15 lần chọn a mà Miller - Rabin đều trả về true thì xác suất n là số nguyên tố rất cao. Tuy nhiên, để chắc chắn hơn thì ta cũng sẽ kiểm tra thử xem n có chia hết cho các số nguyên tố nhỏ hơn 200 hay không.

```
static boolean isPrime(BigInteger n) {
    boolean bool = true;
    int t = 15;
    BigInteger a = new BigInteger("1");
    BigInteger one = a;

    for (int i = 0; i < t; i++) {
        a = RandomBigInteger(one, n);
        if (GCD(a, n).compareTo(BigInteger.ONE) != 0) {
            return false;
        }
        bool = Miller_Rabin(n, a);
        if (bool == false) {
            return bool;
        }
    }
}
```

```
String[] prime = { "2", "3", "5", "7", "11", "13", "17", "19", "23", "29", "31", "37",  
"41", "43", "47", "53",  
"59", "61", "67", "71", "73", "79", "83", "91", "101", "103", "107", "109", "113",  
"127", "131", "137", "139", "149", "151", "157", "163" };  
for (int i = 0; i < 37; i++) {  
    a = new BigInteger(prime[i]);  
    if (n.mod(a).compareTo(BigInteger.ZERO) == 0) {  
        return false;  
    }  
}  
return bool;  
}
```

Hàm `static BigInteger getPrime(int bit)`: có tác dụng tạo một số nguyên tố ngẫu nhiên với độ dài *bit* nhất định.

- Chọn một số random gán vào *res*.
- Lặp cho đến khi *res* chứa số nguyên tố.
 - Nếu *res* chia 6 dư 1 thì gán $res = res + 4$
 - Nếu không thì gán $res = res + 2$

```
static BigInteger getPrime(int bit) {  
    Random randNum = new Random();  
    BigInteger res = new BigInteger(bit, randNum);  
    BigInteger two = new BigInteger("2");  
    BigInteger four = new BigInteger("4");  
    BigInteger six = new BigInteger("6");  
  
    boolean t = isPrime(res);  
    if (res.mod(two).compareTo(BigInteger.ZERO) == 0) {  
        res = res.add(BigInteger.ONE);  
    }  
    while (t == false) {  
        if (res.mod(six).compareTo(BigInteger.ONE) == 0) {  
            res = res.add(four);  
        } else {  
            res = res.add(two);  
        }  
    }  
}
```



```
t = isPrime(res);  
}  
return res;  
}
```

Hàm `static BigInteger getStrongPrime(int bit, BigInteger e)`: dùng để tạo 1 số nguyên tố mạnh ngẫu nhiên và thỏa tính chất nguyên tố cùng nhau với khóa *private* e (để hiện thực giải thuật RSA). Định nghĩa số nguyên tố mạnh p nhóm lựa chọn để hiện thực, thỏa các tính chất:

- $p = 1 \bmod p_0$ với p_0 là một số nguyên tố lớn.
- $p = -1 \bmod p_1$ với p_1 là một số nguyên tố lớn.

p_0 và p_1 trong hiện thực là những số nguyên tố ngẫu nhiên 128 bit. Các bước hiện thực gồm:

- Sinh một số nguyên lớn 1024 bits gán vào res
- Sinh 2 giá trị p_0 và p_1 là những số nguyên tố 128 bit sao cho p_0 khác p_1
- Tính inv_1 , inv_2 lần lượt là nghịch đảo modulo của $p_1 \% p_0$, $p_0 \% p_1$
- Gán $crt_1 = inv_1 * p_1$ vậy crt_1 thỏa:

$$\begin{aligned} - crt_1 &= 1 \bmod p_0 \\ - crt_1 &= 0 \bmod p_1 \end{aligned}$$

- Gán $crt_2 = (p_1 - inv_2) * p_0$ vậy crt_2 thỏa:

$$\begin{aligned} - crt_2 &= -1 \% p_1 \\ - crt_2 &= 0 \% p_0 \end{aligned}$$

- Biến đổi res :

$$\begin{aligned} - increment &= 2 * p_0 * p_1 \\ - crt &= crt_1 + crt_2 + increment \\ - res &= res + (crt - res \% increment) \% increment \end{aligned}$$

- Lúc này res là một tổng có chứa crt , chỉ mới là một số có khả năng thỏa tính chất $crt = 1 \bmod p_0$, $crt = -1 \bmod p_1$
- Chạy vòng lặp, trong mỗi lần lặp kiểm tra nếu res chưa thỏa tính chất một số nguyên tố mạnh và nguyên tố cùng nhau với e thì ta gán $res = res + 4 * p_0 * p_1$. Sau nhiều lần lặp thì sẽ tìm được một giá trị nguyên tố và số nguyên tố đó cũng sẽ thỏa tính chất của một số nguyên tố mạnh.

```
static BigInteger[] getStrongPrime(int bit, BigInteger e) {
    BigInteger two = new BigInteger("2");
    BigInteger lo = new BigInteger( "94807519081091767274915958072241511435209662959" +
        "194056066952784574407133867106560704159305496966045119348817003874896742069226" +
        "49072631363246958077487951940"
    );
    BigInteger hi = two.pow(512).subtract(BigInteger.ONE);
    BigInteger res = RandomBigInteger(lo, hi);
    boolean result = false;
    BigInteger p0 = getPrime(128);
    while (e.mod(p0).compareTo(BigInteger.ZERO) == 0) {
        p0 = getPrime(128);
    }
    BigInteger p1 = getPrime(128);
    while ((p1.compareTo(p0) == 0) || (e.mod(p1).compareTo(BigInteger.ZERO) == 0)) {
        p1 = getPrime(128);
    }
    p1 = p1.multiply(two);

    BigInteger increment = p0.multiply(p1);

    BigInteger[] inv1 = Inverse(p1, p0);
    BigInteger crt1 = inv1[0];
    crt1 = crt1.multiply(p1);

    BigInteger[] inv2 = Inverse(p0, p1);
    BigInteger crt2 = p1.subtract(inv2[0]);
    crt2 = crt2.multiply(p0);

    BigInteger crt = crt1.add(crt2).add(increment);
    BigInteger resmod = res.mod(increment);
    res = res.add(crt.subtract(resmod).mod(increment));
    increment = increment.multiply(two);

    while (true) {
        boolean possible_prime = true;
        if (possible_prime) {
            if (GCD(e, res.subtract(BigInteger.ONE)).compareTo(BigInteger.ONE) != 0) {
                possible_prime=false;
            }
        }
    }
}
```

```
        if (possible_prime) {
            result = isPrime(res);
            if (result == true) {
                break;
            }
        }
        res = res.add(increment);
    }
    return new BigInteger[] {res, p0, p1.divide(two)};
}
```

Hàm mã hóa và giải mã

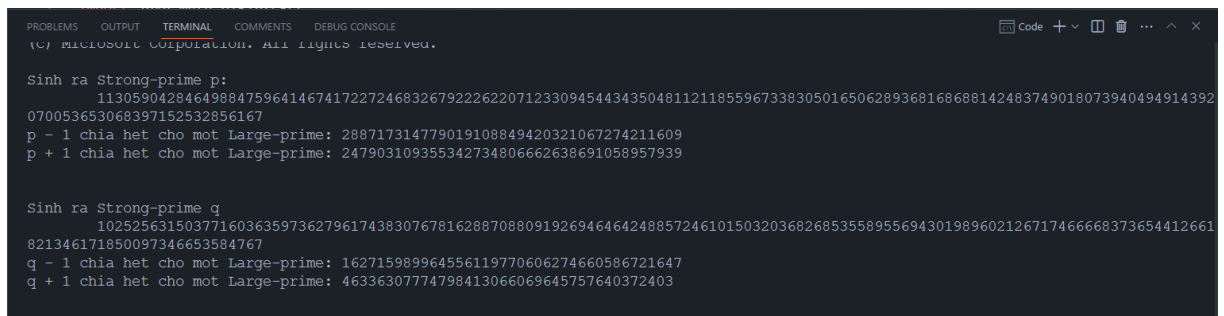
```
static BigInteger Encrypt(BigInteger m, BigInteger e, BigInteger p, BigInteger q) {
    BigInteger N = p.multiply(q);
    return Pow(m, e, N);
}

static BigInteger Decrypt(BigInteger c, BigInteger d, BigInteger p, BigInteger q) {
    BigInteger N = p.multiply(q);
    return Pow(c, d, N);
}
```

3.1.2 Demo

Toàn bộ Source code đã được upload lên Github:

<https://github.com/chequanghuyca/MatmaAnninhmang-RSA/tree/main/RSA>



```
PROBLEMS OUTPUT TERMINAL COMMENTS DEBUG CONSOLE
(c) MICROSOFT CORPORATION. All rights reserved.

Sinh ra Strong-prime p:
11305904284649884759641467417227246832679222622071233094544343504811211855967338305016506289368168688142483749018073940494914392
070053653068397152532856167
p - 1 chia het cho mot Large-prime: 288717314779019108849420321067274211609
p + 1 chia het cho mot Large-prime: 247903109355342734806662638691058957939

Sinh ra Strong-prime q
10252563150377160363597362796174383076781628870880919269464642488572461015032036826853558955694301989602126717466668373654412661
821346171850097346653584767
q - 1 chia het cho mot Large-prime: 162715989964556119770606274660586721647
q + 1 chia het cho mot Large-prime: 46336307774798413066069645757640372403
```

Để minh họa cho thuật toán, ta tiến hành tạo các số ngẫu nhiên có giá trị lớn (512 bit), với p , q là số nguyên tố, thông điệp (message) là một số ngẫu nhiên. Sử dụng thuật toán mã hóa của RSA ta tiến hành mã hóa, sau đó lại giải mã thông điệp đó ra. Toàn bộ thông tin được in lên màn hình.

```
Message:
4316255604127700698734231157493611606377884139360531158805619010504868247328849991339717780935631739666650647601839359632392400559804798
5709649165222900724175920723877370472813470677231324067206520156261909081504679991458020057548097694896318621148937631100669558388461074
61031449470257519338430320931569082

Encryption:
1117909871578594468857904463880961160420245366347638109763022940700939762399019586925411523236462818130229811842620555759245418889329315
4774924289898580037900725410161502669897058800300140372105619879766832769929242714679740529205677772270263084709836808933122056902429488
9613633211453531110247234263266708232

Inverse modulo phi:
1153162935041294839160043373828995812125182112214842226653748648369530113413812364172018365800945445223265352131714450528235154364870681
1987260001657021515922601536904094039910520969466876837863433015181856671398882447338085911222142052993457439675106576755971408993834547
393856533366810940162199497866665491

Decryption:
4316255604127700698734231157493611606377884139360531158805619010504868247328849991339717780935631739666650647601839359632392400559804798
5709649165222900724175920723877370472813470677231324067206520156261909081504679991458020057548097694896318621148937631100669558388461074
61031449470257519338430320931569082

c:\Users\Huy Che\MatmaAnninhmang-RSA\RSA\src>
```

3.2 Đánh giá

Hệ thống đáp ứng đầy đủ nhu cầu cơ bản của mã hóa RSA, với cặp khóa là số nguyên tố cùng nhau lớn 512 bits. Tốc độ tạo khóa, mã hóa và giải mã diễn ra rất nhanh.

4 Kết luận

4.1 Chương trình đã làm được

- 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 khóa 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 .

4.2 Chương trình chưa làm được

Nhận vào thông điệp và khóa từ bên ngoài.

4.3 Ưu điểm

- Số dòng code ít (khoảng 300 dòng).
- Sử dụng rất ít các thư viện có sẵn của Java.
- Tốc độ thực thi tương đối nhanh.

4.4 Nhược điểm

- Chưa cung cấp cho người dùng khả năng tương tác từ các thiết bị ngoại vi.
- Chưa thể sử dụng như một module tích hợp vào chương trình khác.
- Độ phức tạp giải thuật còn khá cao ($O(n^2)$ ở hàm kiểm tra số nguyên tố khi sử dụng hàm Miller - Rabin có chứa vòng lặp, bên trong 1 vòng lặp khác).

5 Tài liệu tham khảo

References

- [1] Wikipedia - RSA. (cryptosystem)
<https://en.wikipedia.org/wiki/RS>
- [2] Vinh Phạm - RSA là gì? Cách thức hoạt động của mã hóa RSA.
<https://bizflycloud.vn/tin-tuc/rsa-la-gi-20220506153312797.htm>
- [3] manhhomienbienthuy - Hệ mã RSA và chữ ký số.
<https://viblo.asia/p/he-ma-hoa-rsa-va-chu-ky-so-6J3ZgkgMZmB>

– HẾT –