ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO KỸ THUẬT:

MẬT MÃ VÀ AN NINH MẠNG

BÀI TẬP LỚN SỐ 2

GVHD: Nguyễn Hữu Hiếu

SV thực hiện: Nguyễn Tất Chung 1810055

Phạm Nguyễn Thái Khương 1812694

Nguyễn Thành Phát 1810425

Tp. Hồ Chí Minh, Tháng 06/2021

Table of Contents

[CHƯƠNG 1 - GIỚI THIỆU 3](#_Toc73975183)

[Giới thiệu 3](#_Toc73975184)

[Cơ sở lý thuyết: 4](#_Toc73975185)

[a) Tạo khóa: 5](#_Toc73975186)

[b) Mã hóa 5](#_Toc73975187)

[c) Giải mã 5](#_Toc73975188)

[CHƯƠNG 2 - PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 6](#_Toc73975189)

[1. Xây dựng hệ thống tính toán với số lớn: 6](#_Toc73975190)

[2. Tìm số nguyên tố lớn khi đã cho số bit của số nguyên tố lớn cần tìm. 7](#_Toc73975191)

[3. Tìm ước số lớn nhất khi cho hai số nguyên lớn. 8](#_Toc73975192)

[4. Tính toán khoá giải mã d khi cho khoá mã hoá e và hai số nguyên tố lớn. 8](#_Toc73975193)

[5. Mã hoá khi cho thông điệp và khoá mã hoá e và n. 8](#_Toc73975194)

[6. Giải mã khi cho thông điệp mã hoá và khoá giải mã d và n. 8](#_Toc73975195)

[7. Phân tích và thiết kế. 9](#_Toc73975196)

[CHƯƠNG 3 - HIỆN THỰC VÀ ĐÁNH GIÁ HỆ THỐNG 9](#_Toc73975197)

[1. Class Utils: 9](#_Toc73975198)

[2. Class RSA: 9](#_Toc73975199)

[3. Class Main: 10](#_Toc73975200)

[4. Đánh giá hệ thống: 10](#_Toc73975201)

[CHƯƠNG 4 - KẾT LUẬN 10](#_Toc73975202)

[PHỤ LỤC 10](#_Toc73975203)

# CHƯƠNG 1 - GIỚI THIỆU

## Giới thiệu

Yêu cầu chương trình/hệ thống: hiện thực hệ mã RSA trên Java/C/C++, không được dùng các hiện thực RSA đã có từ web hay trong Java. Những gì có thể dùng gồm:

* Lớp BigInteger được xây dựng sẵn của Java.
* Thư viện NTL(Library for doing Number Theory) hoặc GMP(the GNU Multiple Precision Arithmetic Library).
* Có thể dùng các hiện thực big-integer để quản lý quản lý dữ liệu của bạn và thực hiện phép tính mod nhưng không được dùng các phương thức đã hiện thực (gcd, power, tìm số nguyên tố, …). Phải tự hiện thực các phương thức này.
* Có thể dùng các hàm an toàn đang tồn tại để tạo ra số ngẫu nhiên lớn. Vd Java cung cấp các công cụ để tạo số ngẫu nhiên trong java.util.random hay java.security.SecureRandom. Tương tự C++ có hàm rand() và srand() để tạo số ngẫu nhiên.

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

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

Nhóm chọn hiện thực hệ mã RSA trên Java.

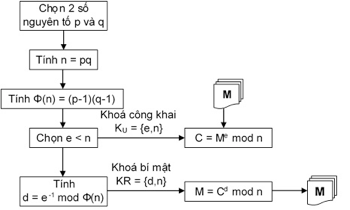
* Phân tích và thiết kế hệ thống: xác định các hàm, class cần hiện thực và một số hàm phụ hỗ trợ cho hàm chính.
* Hiện thực và đánh giá hệ thống: các class, phương thức của mỗi class
* Kết luận: một số chức năng hệ thống đã làm được.

## Cơ sở lý thuyết:

Trong mật mã học, 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.

Thuật toán RSA có hai khóa: khóa công khai và khóa bí mật (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. 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.

Ta có thể mô phỏng trực quan một hệ mật mã khoá công khai như sau: Bình muốn gửi cho An một thông tin mật mà Bình muốn duy nhất An có thể đọc được. Để làm được điều này, An gửi cho Bình một chiếc hộp có khóa đã mở sẵn và giữ lại chìa khóa. Bình nhận chiếc hộp, cho vào đó một tờ giấy viết thư bình thường và khóa lại (như loại khoá thông thường chỉ cần sập chốt lại, sau khi sập chốt khóa ngay cả Bình cũng không thể mở lại được-không đọc lại hay sửa thông tin trong thư được nữa). Sau đó Bình gửi chiếc hộp lại cho An. An mở hộp với chìa khóa của mình và đọc thông tin trong thư. Trong ví dụ này, chiếc hộp với khóa mở đóng vai trò khóa công khai, chiếc chìa khóa chính là khóa bí mật.



**Sơ đồ biểu diễn thuật toán mã hóa RSA**

*(Nguồn:* PKCS #1 v2.2: RSA Cryptography Standard, *RSA Laboratories)*

*Các đặc điểm chính trong thuật toán mã hóa RSA*

Thuật toán RSA được thiết kế dựa trên độ khó của bài toán phân tích ra thừa số nguyên tố trên tập số nguyên *Zn*.

Cho số nguyên dương n = p \* q, với p, q là 2 số nguyên tố rất lớn (ít nhất 100 ký số). Khi biết *n*, muốn tìm *p, q* thì phải giải bài toán phân tích ra thừa số nguyên tố, công việc này đòi hỏi phải thực hiện một số lượng các phép tính vô cùng lớn.

### a) Tạo khóa:

- Tạo ngẫu nhiên 2 số nguyên tố *p, q* khác nhau và rất lớn (có số ký tự ít nhất là 100), sau đó tính: *n = p \* q*; *Ф(n) = (p -1) \* (q -1)*.

- Chọn ngẫu nhiên 1 số *e* sao cho *1 < e < Ф(n)*, với *e* là số nguyên tố cùng nhau với *Ф(n)*.

- Tính số nghịch đảo *d* của *e* đối với *Ф(n)*: *1 < d < Ф(n), ed = 1(mod Ф(n))*.

Ở đây *d* là số mũ bí mật; *e* là số mũ công khai.

- Cặp *KU = {e, n}* được gọi là khoá công khai.

- Cặp *KR = {d, n}* được gọi là khoá bí mật.

### b) Mã hóa

Khi Bên gửi muốn gửi thông điệp *M* cho Bên nhận với yêu cầu cần bảo mật thông tin. Bên gửi yêu cầu Bên nhận gửi khoá công khai *KU = {e, n}*. Bên gửi dùng khoá công khai *KU* này để mã hoá thông điệp *M* thành *C* theo công thức: *C = Me  mod n*, sau đó Bên gửi gửi *C* cho Bên nhận.

### c) Giải mã

Để giải mã bản mã *C*, Bên nhận dùng khóa bí mật *KR = {d, n}* để có thể khôi phục lại dữ liệu gốc ban đầu do Bên gửi gửi đến thông qua phép toán *M = Cd mod n*.

# CHƯƠNG 2 - PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

### 1. Xây dựng hệ thống tính toán với số lớn:

Trong dự án này, nhóm chúng em đã sử dụng thư viện BigInteger để quản lý số lớn.

Những phương thức của thư viện BigInteger đã được sử dụng trong dự án này là:

* add: Cộng hai số nguyên
* and: Phép ‘Or’ hai số nguyên
* compareTo: so sánh hai số nguyên
* divide: Chia hai số nguyên
* mod: Module hai số nguyên
* multiply: Nhân hai số nguyên
* shiftRight: Dịch phải số nguyên

Ngoài ra, để sử dụng cho việc tính toán, chúng em đã hiện thực thêm những phương thức sau:

* power(x, y): Tính toán x^y
* powerMod(x, y, z): Tính toán (x^y) mod z

Sử dụng phương pháp Right-to-left binary [1].

Mã giả:

**function** modular\_pow(base, exponent, modulus) **is**

**if** modulus = 1 **then**

**return** 0

[Assert](https://en.wikipedia.org/wiki/Assertion_(computing)) :: (modulus - 1) \* (modulus - 1) does not overflow base

result := 1

base := base **mod** modulus

**while** exponent > 0 **do**

**if** (exponent **mod** 2 == 1) **then**

result := (result \* base) **mod** modulus

exponent := exponent >> 1

base := (base \* base) **mod** modulus

**return** result

* gcd(a, b): Tính toán ƯCLN của a và b [2].

Sử dụng Euclidean algorithm.

Mã giả:

**function** gcd(a, b)

**while** a ≠ b

**if** a > b−

a := a − b

**else**

b := b − a

**return** a

* gcdExtended(a, b): Tính a^(-1) mod b

Sử dụng Extended Euclidean Algorithm [3].

Mã giả:

**function** inverse(a, n)

t := 0; newt := 1

r := n; newr := a

**while** newr ≠ 0 **do**

quotient := r **div** newr

(t, newt) := (newt, t − quotient × newt)

(r, newr) := (newr, r − quotient × newr)

**if** r > 1 **then**

**return** "a is not invertible"

**if** t < 0 **then**

t := t + n

**return** t

### 2. Tìm số nguyên tố lớn khi đã cho số bit của số nguyên tố lớn cần tìm.

Để hiện thực hàm tìm số nguyên tố, nhóm chúng em đã hiện thực ba thành phần con:

* randomGeneration(n): sinh ra một số nguyên dương n-bit

Sử dụng thư viện java.security.SecureRandom để sinh ra số ngẫu nhiên.

* getLowLevelPrime(n): sinh ra một số có thể là nguyên tố n-bit

Ngẫu nhiên sinh ra một số, sau đó chia nó với 100 số nguyên tố đầu tiên.

Nếu thỏa được không chia hết ở bước trên thì trả về kết quả.

* isMillerRabinPassed(number): kiểm tra số nguyên tố bằng Miller–Rabin primality test [4].

Mã giả:

**Input #1**: *n* > 3, an odd integer to be tested for primality

**Input #2**: *k*, the number of rounds of testing to perform

**Output**: “*composite*” if *n* is found to be composite, “*probably prime*” otherwise

write *n* as 2*r*·*d* + 1 with *d* odd (by factoring out powers of 2 from *n* − 1)

WitnessLoop: **repeat** *k* **times**:

pick a random integer *a* in the range [2, *n* − 2]

*x* ← *ad* mod *n*

**if** *x* = 1 or *x* = *n* − 1 **then**

**continue** WitnessLoop

**repeat** *r* − 1 **times**:

*x* ← *x*2 mod *n*

**if** *x* = *n* − 1 **then**

**continue** WitnessLoop

**return** “*composite*”

**return** “*probably prime*”

### 3. Tìm ước số lớn nhất khi cho hai số nguyên lớn.

Sử dụng hàm gcd(a, b) đã nêu ở trên mục 1.

### 4. Tính toán khoá giải mã d khi cho khoá mã hoá e và hai số nguyên tố lớn.

Khi cho hai số nguyên tố p, q lớn, bước đầu ta tính toán ø(n)= (p - 1)(q - 1).

Sau đó tìm e thỏa 1 < e < ø(n) và gcd(e, ø(n)) = 1.

Để tìm d thỏa ed = 1 mod ø(n) (1)

Ta có (1) ⇔ d = e^-1 mod ø(n).

Áp dụng gcdExtended(e, ø(n)) đã nêu ở mục 1. Ta tìm được d.

### 5. Mã hoá khi cho thông điệp và khoá mã hoá e và n.

Để mã hóa thông điệp m. Ta sử dụng hàm powerMod(m, e, n) để mã hóa.

### 6. Giải mã khi cho thông điệp mã hoá và khoá giải mã d và n.

Để giải mã thông điệp m. Ta sử dụng hàm powerMod(m, e, n) để giải mã.

### 7. Phân tích và thiết kế.

Để hiện thực được bài toán trên, chúng em quyết định hiện thực hai class:

* Utils: bao gồm những phương thức được đề cập ở mục 1, 2, 3 và 4.

Đó là những hàm dùng để hỗ trợ tính toán hoặc không liên quan nhiều đến RSA.

* RSA: bao gồm những phương thức được đề cập ở mục 5 và 6.

Gồm những hàm cần thiết cho việc hiện thực RSA.

* Main: Gồm những hàm sử dụng cả hai class Utils và RSA.

# CHƯƠNG 3 - HIỆN THỰC VÀ ĐÁNH GIÁ HỆ THỐNG

### 1. Class Utils:

Bao gồm những phương thức sau:

* splitMessages: Chia đôi mỗi message trong danh sách nhằm giảm kích thước mỗi message.
* bigIntegerToString: Chuyển message dạng BigInteger thành chuỗi.
* power(x, y): Tính toán x^y.
* powerMod(x, y, z): Tính toán (x^y) mod z
* gcd(a, b): Tính toán ƯCLN của a và b.
* gcdExtended(a, b): Tính a^(-1) mod b.
* randomGeneration(n): sinh ra một số nguyên dương lẻ n-bit
* getLowLevelPrime(n): sinh ra một số có thể là nguyên tố n-bit
* isMillerRabinPassed(number): Kiểm tra số nguyên tố
* eGeneration(ø(n)): Tạo ra e
* calculateD(e, ø(n)): Tính d

### 2. Class RSA:

Bao gồm những phương thức sau:

* initPublicKey: Khởi tạo khóa công khai
* initPrivateKey: Khởi tạo khóa bí mật
* encrypt: Mã hóa chuỗi số
* encryptMessage: Mã hóa thông điệp
* encryptFile: Mã hóa file
* decrypt: Giải mã
* sign: Ký lên chuỗi số
* signMessage: Ký lên thông điệp
* signFile: Ký lên file
* verify: Xác minh
* fileToDecimal: Chuyển file chưa mã hóa sang chuỗi số
* readFromFile: Chuyển file đã mã hóa sang chuỗi số
* writeBigIntegerToFile: Ghi chuỗi số vào file
* writeStringToFile: Ghi string vào file

### 3. Class Main:

Bao gồm những phương thức sau:

* generateKey: Tạo bộ khóa công khai và bí mật, sau đó ghi vào file
* enc\_file: Mã hóa file
* dec\_file: Giải mã file
* sign\_file: Ký file
* verify\_file: Xác minh file

### 4. Đánh giá hệ thống:

Hệ thống đã đạt yêu cầu.

# CHƯƠNG 4 - KẾT LUẬN

Hệ thống đã làm tốt yêu cầu được đưa ra

# PHỤ LỤC

Ứng dụng được phát triển bằng Java, sử dụng JDK 11.0

Các thư viện đã được sử dụng:

* java.math.BigInteger
* java.util.List
* java.util.ArrayList
* java.io
* java.nio.charset.Charset
* java.security.SecureRandom

Mã nguồn mở được sử dụng:

* [Implementation of RSA encryption algorithm - Rafael M. Pestano](https://github.com/rmpestano/RSA/)

Tài liệu tham khảo:

[[1] Right-to-left binary Method](https://en.wikipedia.org/wiki/Modular_exponentiation#Right-to-left_binary_method)

[[2] Euclidean algorithm](https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations)

[[3] Extended Euclidean algorithm](https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm#Computing_multiplicative_inverses_in_modular_structures)

[[4] Miller–Rabin primality test](https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test)