**MỤC LỤC**

[1 CHƯƠNG I: GIỚI THIỆU 3](#_Toc494491052)

[1.1 Yêu cầu: 3](#_Toc494491053)

[1.2 Nội dung 4](#_Toc494491054)

[2 CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 4](#_Toc494491055)

[2.1 Tìm số nguyên tố lớn khi cho số lượng bit của số nguyên tố lớn cần tìm. 4](#_Toc494491056)

[2.2 Tính ước số lớn nhất khi cho hai số nguyên lớn. 4](#_Toc494491057)

[2.3 Giải thuật RSA 4](#_Toc494491058)

[2.4 Thiết kế hệ thống 5](#_Toc494491059)

[3 CHƯƠNG 3: HIỆN THỰC VÀ ĐÁNH GIÁ HỆ THỐNG 6](#_Toc494491060)

[3.1 Hiện thực lớp PrimeNumber 6](#_Toc494491061)

[3.2 Hiện thực lớp RSA 7](#_Toc494491062)

[3.3 Hiện thực RSAGenerator 9](#_Toc494491063)

[3.4 Hiện thực RSACryptor 11](#_Toc494491064)

[3.5 Đánh giá 12](#_Toc494491065)

[4 Kết luận 12](#_Toc494491066)

# CHƯƠNG I: GIỚI THIỆU

## Yêu cầu:

Trong bài tập lớn này, bạn cần phải hiện thực hệ mã RSA trên Java/C/C++. Bạn không được dùng các hiện thực RSA đã có từ web hay trong Java. Những gì bạn có thể dùng là:

* + - * Java có lớp BigInteger được xây dựng sẳn.
      * C++ có thư viện [NTL](http://shoup.net/ntl/)(Library for doing Number Theory) or [GMP](http://directory.fsf.org/GNU/gmp.html) (the GNU Multiple Precision Arithmetic Library).
      * Bạn có thể dùng các hiện thực big-integer để quản lý dữ liệu của bạn và thực hiện phép toán **mod** nhưng không được dùng các phương thức đã hiện thực (gcd, power, tìm số nguyên tố, ..). Như vậy bạn phải tự hiện thực các phương thức này.
      * Bạn có thể dùng hàm an toàn đang tồn tại để tạo ra các số ngẫu nhiên lớn. Ví dụ Java cung cấp các công cụ để tạo số ngẫu nhiên trong java.util.random hay java.security.SecureRandom. Tương tự C++ có rand() và srand() để tạo số ngẫu nhiên.

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 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.

## Nội dung

+ Phân tích yêu cầu

+ Hiện thực hệ thống

+ Đánh giá

# CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

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

+ Sử dụng thư viện BigInteger

+ Dùng thư viện SecureRandom để sinh ra số lớn ngẫu nhiên với độ dài bit được cho

+ Dùng giải thuật Miller Rabin để kiểm tra tính nguyên tố

## Tính ước số lớn nhất khi cho hai số nguyên lớn.

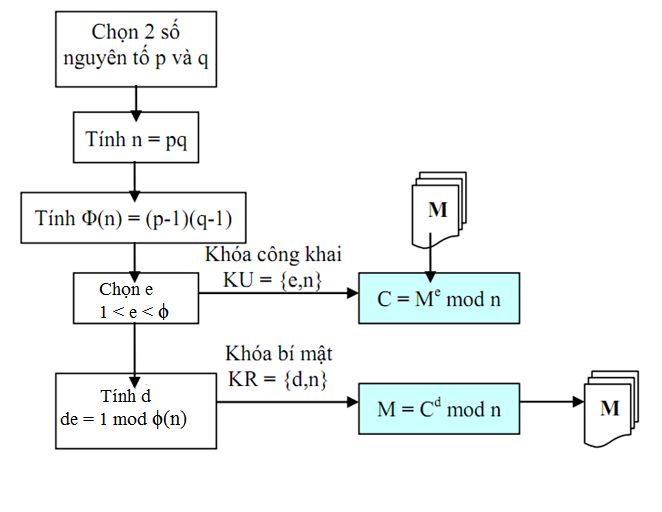
+ Dùng giải thuật Euclid

**public** **static** **int** gcd**(int** a, **int** b**)** *//valid for positive integers.*  
**{**  
 **while(**b > **0)**  
 **{**  
 **int** c = a % b;  
 a = b;  
 b = c;  
 **}**  
 **return** a;  
**}**

+ Áp dụng với thư viện BigInteger

## Giải thuật RSA

* + - * 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.

******

1. Chọn 2 số nguyên tố lớn khác nhau p, q thỏa mãn điều kiện 
2. Tính tích của nó 
3. Tính giá trị hàm Phi Euler của n: .
4. Chọn số nguyên d, sao cho và gcd(d,)=1.
5. Tính giá trị e thỏa mãn điều kiện:.

**Khóa công khai** bao gồm: n và e. **Khóa mật:** d còn p,q và thường là xóa sau khi tính toán khóa.

## Thiết kế hệ thống

+ Thư viện PrimeNumber để sinh số nguyên tố

+ Thư viện RSA để xử lý tạo khóa, mã hóa, giải mã

+ Giao diện RSAGenerator để người dùng thao tác sinh RSA Key Pair

+ Giao diện RSACryptor để người dùng mã hóa, giải mã

# CHƯƠNG 3: HIỆN THỰC VÀ ĐÁNH GIÁ HỆ THỐNG

## Hiện thực lớp PrimeNumber

**Method genFixOddBigInteger**

+ Khai báo: public static BigInteger genFixOddBigInteger(int bitLength)

+ Chức năng: Sinh ra số ngẫu nhiên lẻ BigInteger với độ dài bitLength cho trước

**Method isPrimeBigByMillerRabin**

+ Khai báo: public static boolean isPrimeBigByMillerRabin(BigInteger n, int k)

+ Chức năng: Kiểm tra một số BigInteger theo giải thuật Miller Rabin với độ chắc chắn k

**Method randomRangeBig**

+ Khai báo: public static BigInteger randomRangeBig(BigInteger bottom, BigInteger top)

+ Chức năng: dùng thư viện SecureRandom và BigInteger để sinh ra số BigInteger ngẫu nhiên x thỏa mãn x>= bottom và x <= top.

**Method gcd**

+ Khai báo: public BigInteger gcd(BigInteger a, BigInteger b)

+ Chức năng: Hàm tìm ước số chung lớn nhất cho 2 số BigInteger a, b

**Method modPowBig**

+ Khai báo: public static BigInteger modPowBig(BigInteger base, BigInteger exponent, final BigInteger modulo)

+ Chức năng: Tính result = base ^ exponent mod modulo

+ Tối ưu dùng giải thuật Exponentiation By Squaring, còn gọi là Binary Exponentiation

**Method genPrime**

+ Khai báo: public static BigInteger genPrime(int bitLength,int k)

+ Chức năng: sinh ra số nguyên tố, dùng kết hợp với hàm kiểm tra tính nguyên tố isPrimeBigByMillerRabin

**Method logProgress**

+ Khai báo: public static void logProgress(String signal)

+ Chức năng: gởi thông tin xử lý đến giao diện cho người dùng theo dõi tiến trình tính toán

**Method genFixOddBigInteger**

+ Khai báo: public static BigInteger genFixOddBigInteger(int bitLength)

+ Chức năng: sinh ra ngẫu nhiên số BigInteger lẻ với độ dài bitLength được cho

**Method driverTestIsPrimeBigByMillerRabin**

+Khai báo: public static void driverTestIsPrimeBigByMillerRabin()

+ Chức năng: test thử giải thuật kiểm tra số nguyên tố dùng MillerRabin

**Method setLogger**

+ Khai báo: public static void setLogger(JTextArea console)

+ Chức năng: Thiết lập đối tượng xuất thông tin xử lý, giúp người dùng theo dõi quá trình tính toán.

## Hiện thực lớp RSA

**Method generateKeyPair**

+ Khai báo: public void generateKeyPair()

+ Chức năng: sinh ra cặp khóa public key (e,n) và private key (d,n);

* Với độ dài bitLength cho trước, ta sẽ tạo ra 2 số nguyên tố p, q; N= p\*q;
* Tạo ra e với kích thước bitLength/2; d= e mod phi(N)
* Xác định chunkSize: là kích thước khi cắt file plaintext ra thành từng đoạn trước khi mã hóa; Lý do là C=M^e mod N; với kích thước bitLength của e, N giới hạn, thì chỉ cho phép mã hóa M nhỏ hơn không gian N;
* Xác định blockSize: là kích thước để mã hóa, giải mã; Lý do là hàm mod kết hợp với thư viện BigInteger trong java trả về một số có độ dài bit trong 1 số trường hợp tăng 1, nên để an toàn trong quá trình mã hóa giải mã, ta tăng blockSize= chunkSize+1;

**Method logProgress**

+Khai báo: public static void logProgress(String signal)

+ Chức năng: gửi thông tin quá trình xử lý đến console hoặc đối tượng giao diện giúp người dùng theo dõi dễ dàng

**Method encryptBlock**

+ Khai báo: public byte[] encryptBlock(byte[] message,int byteSize)

+ Chức năng: mã hóa từng khối byte[] với kích thước byteSize, kết quả cũng trả về byte[]; Bởi vì khóa mã hóa có kích thước byteSize

**Method decryptBlock**

+ Khai báo: public byte[] decryptBlock(byte[] message, int byteSize)

+ Chức năng: giải mã từng khối byte[] với kích thước byteSize, kết quả cũng trả về byte[]; Bởi vì khóa mã hóa có kích thước byteSize

**Method encryptFile**

+ Khai báo: public void encryptFile(String InputFile, String OutputFile)

+ Chức năng: sử dụng hàm encryptBlock để mã hóa file

**Method decryptFile**

+ Khai báo: public void decryptFile(String InputFile, String OutputFile)

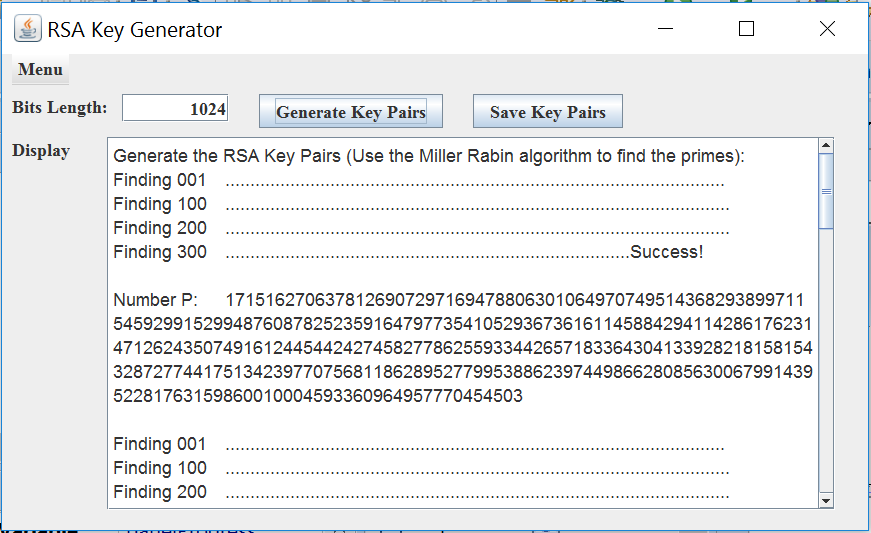
+ Chức năng: sử dụng hàm decryptBlock để mã hóa file

**Method getKeyPair**

+ Khai báo: public String getKeyPair()

+ Chức năng: Trả về thông tin RSA Key Pair

## Hiện thực RSAGenerator



Chức năng chính là sinh ra RSA Key Pair

**Method genKeyPair**

+ Khai báo: public void genKeyPair()

+ Chức năng: Tạo một Thread để chạy giải thuật sinh RSA Key Pair. Do nếu không tạo Thread thì đối tượng giao diện trong Java sẽ dừng khi đang xử lý giải thuật RSA.

**Method operateFile**

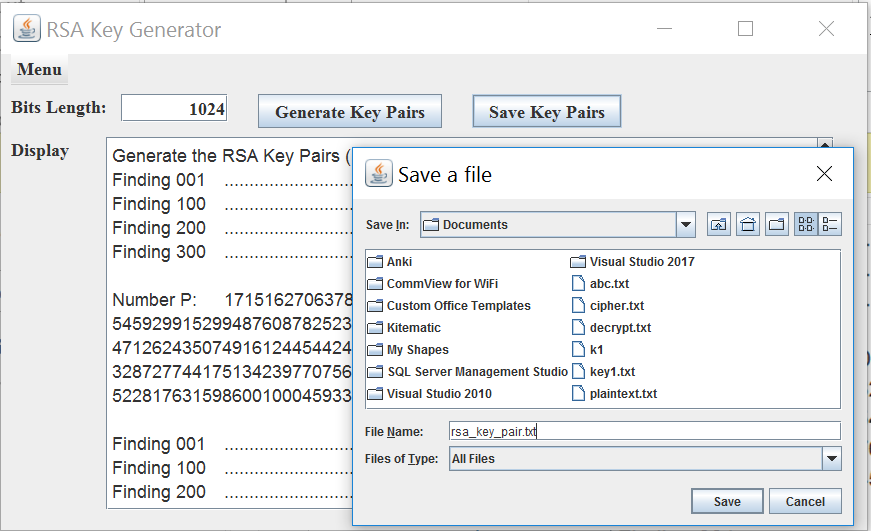
+ Khai báo: protected void operateFile(String title, int type, String data)

+ Chức năng: Mở Dialog cho phép chọn đường dẫn trước khi save khóa

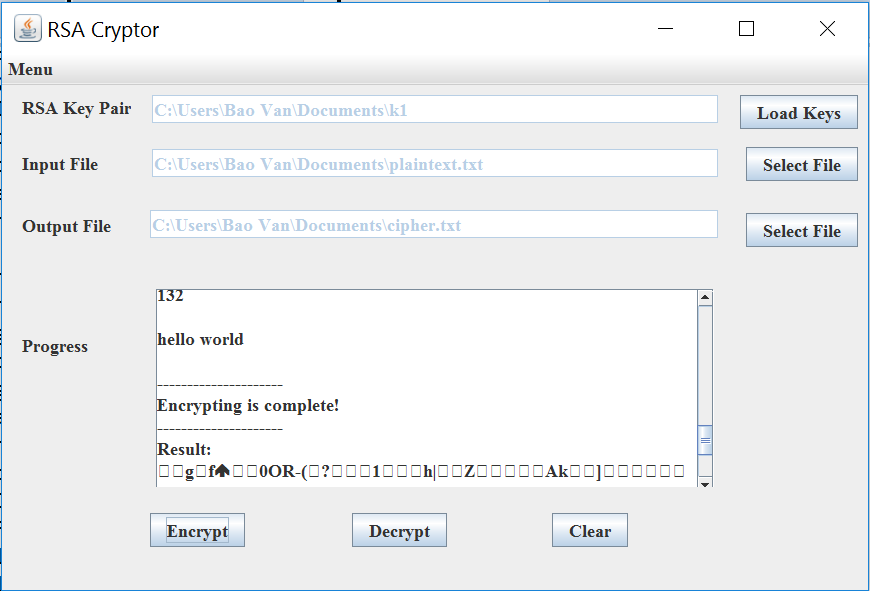
**Method saveKeyPair**

+ Khai báo: public void saveKeyPair()

+ Chức năng: Save RSA Key Pair sang file



## Hiện thực RSACryptor



Cho phép người dùng load RSA Key Pair File.

Chọn File plaintext, ciphertext, tiến hành Mã hóa, giải mã.

**Method loadKeyPairFile**

+ Khai báo: protected void loadKeyPairFile()

+ Chức năng: Mở Dialog cho phép người dùng load RSA KeyPair File

**Method parseKeyPair**

+ Khai báo: protected void parseKeyPair(File file)

+ Chức năng: lấy các thông tin puplic key (e,N) private key( d,N) để tạo đối tượng RSA, chuẩn bị tiến hành mã hóa hoặc giải mã.

**Method selectInputFile**

+ Khai báo: protected void selectInputFile()

+ Chức năng: Cho phép load file Input

**Method selectOutputFile**

+ Khai báo: protected void selectOutputFile()

+ Chức năng: Khai báo đường dẫn lưu file kết quả

## Đánh giá

* Hàm sinh số nguyên tố có thể sinh số nguyên tố dài 1024 bit trong 10s với độ chắc chắn k=10.
* Hàm sinh số nguyên tố có thể sinh số nguyên tố dài 2048 bit trong 79s với độ chắc chắn k=10.
* Hàm modPow được tối ưu với giải thuật Exponentiation By Squaring.
* Lớp RSA là thư viện cung cấp các phương thức cho phép tạo khóa, mã hóa, giải mã RSA
* Lớp RSAGenerator cung cấp giao diện để người dùng sinh khóa, lưu khóa sang file
* Lớp RSACryptor cung cấp giao diện cho phép sử dụng khóa, mã hóa và giải mã file.

# Kết luận

Đã hoàn thành đầy đủ các yêu cầu của Assignment 2

|  |  |  |
| --- | --- | --- |
| **Yêu cầu** | **Đánh giá** | **Triển khai** |
| * + - * Tìm số nguyên tố lớn khi cho số lượng bit của số nguyên tố lớn cần tìm. | Hoàn thành | PrimeNumber.genPrime();  Test với hàm driverGenPrime(); |
| * + - * Tính ước số lớn nhất khi cho hai số nguyên lớn. | Hoàn thành | PrimeNumber.gcd() |
| * + - * Tính toán khóa giải mã d khi cho khoá mã hóa e và hai số nguyên tố lớn. | Hoàn thành | RSA.generateKeyPair() |
| * + - * Tạo bộ khóa ngẫu nhiên khi cho 2 số nguyên tố lớn. | Hoàn thành | RSA.generateKeyPair() |
| * + - * Mã hóa khi cho thông điệp và khóa mã hóa e và n. | Hoàn thành | RSA.encryptFile() |
| * + - * Giải mã khi cho thông điệp mã hóa và khóa giải mã d và n. | Hoàn thành | RSA.decryptFile() |

**Tài liệu tham khảo**

[1]. Bài giảng An ninh mạng. ThS. Nguyễn Cao Đạt. email: E-mail: [dat@cse.hcmut.edu.vn](mailto:dat@cse.hcmut.edu.vn) - ĐH Bách Khoa TPHCM

[2]. Cryptography And Network Security 4th Edition 2005

[3]. Giải thuật tìm số nguyên tố Miller Rabin

<http://mersennewiki.org/index.php/Miller-Rabin_pseudoprimality_test>

<http://www.sanfoundry.com/java-program-miller-rabin-primality-test-algorithm/>

[4]. Giải thuật tính modPow: b^e mod m

<http://en.wikipedia.org/w/index.php?title=Modular_exponentiation&oldid=517653653#Right-to-left_binary_method>

<https://en.wikipedia.org/wiki/Exponentiation_by_squaring>

<https://en.wikipedia.org/w/index.php?title=Modular_exponentiation&oldid=517653653#Right-to-left_binary_method>

[5]. Tham khảo tính save file trong java

<https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html>

**Phụ lục**

Môi trường phát triển: IDE Eclipse JAVA EE Version: Neon.2 Release (4.6.2)

Thư viện JAVA:

+ Thư viện hỗ trợ random: java.security.SecureRandom

+Xử lý số Integer lớn: java.math.BigInteger

+Tạo giao diện GUI với JFC/Swing: javax.swing

+ Xử lý Files and I/O: java.io