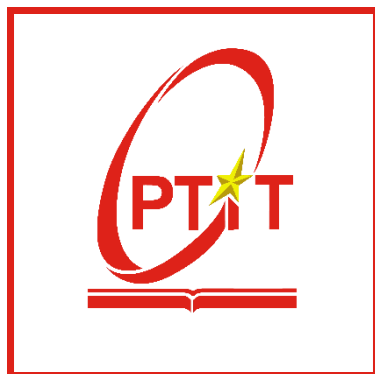


**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**  
**KHOA AN TOÀN THÔNG TIN**

---



**THỰC TẬP CƠ SỞ**  
**Bài 16: Lập trình thuật toán mật mã học**

Sinh viên	Nguyễn Duy Đạt
MSV	B21DCAT056
Giảng viên	Vũ Minh Mạnh

## Môn học Thực tập cơ sở

### Bài 16: Lập trình thuật toán mật mã học

#### I. Lý thuyết

##### 1. Lập trình với số lớn

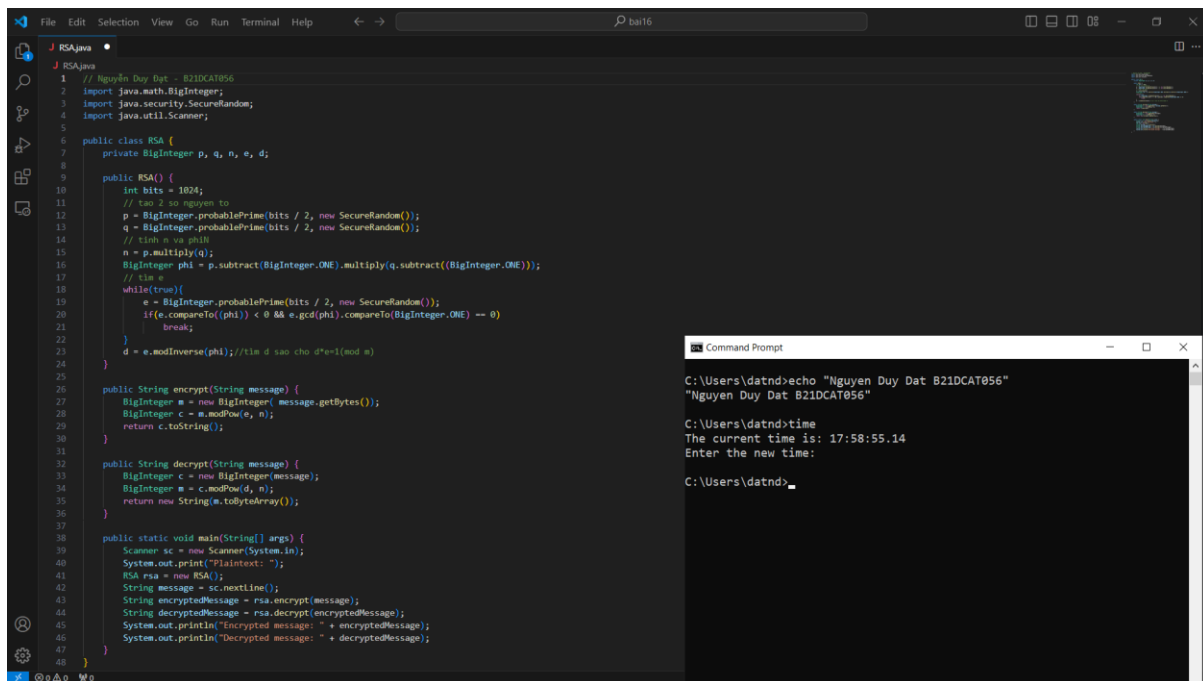
- Việc lập trình với những số có độ dài hàng nghìn bit là rất khó khăn. Thay vì tự viết hàm tính toán các số lớn, sinh viên sử dụng class BigInteger có sẵn trong Java chuyên để xử lý các số lớn.
- Class BigInteger cũng cung cấp những phép toán cơ bản như cộng add(), trừ subtract(), nhân multiply(), chia divide(), giúp việc tính toán các phép toán cơ bản dễ dàng hơn.
- Ngoài ra, BigInteger còn cung cấp hàm lũy thừa lấy phần dư modpow() hay hàm nghịch đảo modulo modInverse() giúp việc lập trình mã hoá và giải mã RSA dễ dàng hơn.

##### 2. Giải thuật mã khóa công khai RSA

- Thuật toán mã hoá RSA là thuật toán mã hoá khóa công khai được sử dụng rộng rãi để truyền dữ liệu an toàn. Thuật toán mã hoá RSA được phát triển bởi Rivest, Shamir, Adleman. Quy trình mã hoá của RSA được công khai năm 1977. Độ an toàn của RSA liên hệ chặt chẽ với độ khó của bài toán phân tích nhân tử của một số rất lớn thành hai thừa số nguyên tố. Hiện nay vẫn chưa có siêu máy tính nào có thể giải bài toán này với thời gian chấp nhận được, nhưng trong tương lai với máy tính lượng tử có thể sẽ khả thi.
- Quy trình mã hoá:
  - + Chọn hai số nguyên tố lớn  $p$  và  $q$  và tính  $N = pq$ . Cần chọn  $p$  và  $q$  sao cho  $M < 2^{(i-1)} < N < 2^i$
  - + Tính  $\Phi(n) = (p - 1)(q - 1)$
  - + Tìm một số  $e$  sao cho:  $\{e \text{ và } \Phi(n) \text{ là 2 số cùng nhau và } 0 < e < \Phi(n)\}$
  - + Tìm một số  $d$  sao cho:  $e \cdot d \equiv 1 \pmod{\Phi(n)}$  (hay:  $d = e^{-1} \pmod{\Phi(n)}$ )
  - + Chọn khóa công khai  $K1$  là cặp  $(e, N)$ , khóa riêng  $K2$  là cặp  $(d, N)$ .
  - + Mã hoá  $C = M^e \pmod{N}$ , hoặc  $C = M^d \pmod{N}$  nếu mã hoá chứng thực.
  - + Giải mã  $M = C^d \pmod{N}$ , hoặc  $M = C^e \pmod{N}$  nếu chứng thực.

## II. Thực hành

- Code thuật toán mã hóa RSA bằng Java

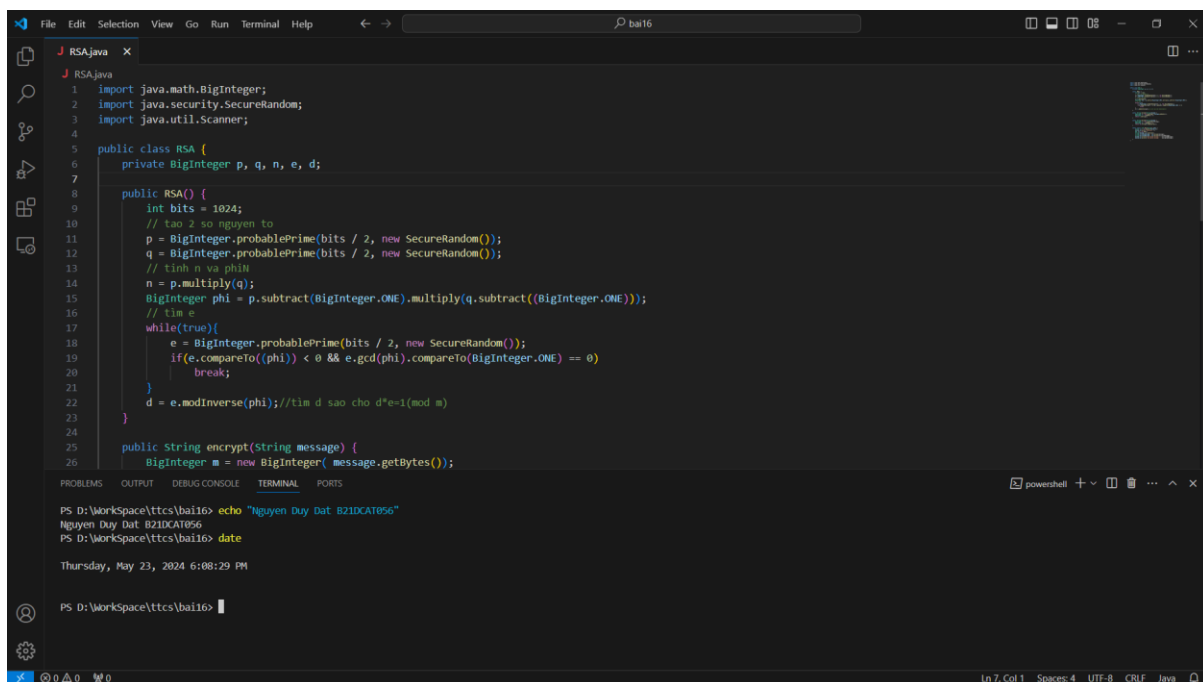


The screenshot shows an IDE with a file named `RSA.java`. The code implements the RSA algorithm. It includes imports for `java.math.BigInteger`, `java.security.SecureRandom`, and `java.util.Scanner`. The `RSA` class has private fields `p`, `q`, `n`, `e`, and `d`. The `init` method generates two large prime numbers `p` and `q`, calculates `n = p * q`, `phi = (p-1) * (q-1)`, and finds a value `e` such that `gcd(e, phi) == 1`. It then calculates `d = e.modInverse(phi)`. The `encrypt` method takes a message string, converts it to bytes, and then to a `BigInteger` `m`. It calculates `c = m.modPow(e, n)` and returns `c.toString()`. The `decrypt` method takes an encrypted string, converts it to a `BigInteger` `c`, and then calculates `m = c.modPow(d, n)`, returning `m.toString()`. The `main` method uses a `Scanner` to read a message, encrypts it, and then decrypts it, printing the results. A `Command Prompt` window is open, showing the execution of the program. It displays the current time and the encrypted message.

```
1 // Nguyễn Duy Đạt - B21DCAT056
2 import java.math.BigInteger;
3 import java.security.SecureRandom;
4 import java.util.Scanner;
5
6 public class RSA {
7     private BigInteger p, q, n, e, d;
8
9     public RSA() {
10         int bits = 1024;
11         // tạo 2 số nguyên tố
12         p = BigInteger.probablePrime(bits / 2, new SecureRandom());
13         q = BigInteger.probablePrime(bits / 2, new SecureRandom());
14         // tính n và phi
15         n = p.multiply(q);
16         BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
17         // tìm e
18         while(true){
19             e = BigInteger.probablePrime(bits / 2, new SecureRandom());
20             if(e.compareTo(phi) < 0 && e.gcd(phi).compareTo(BigInteger.ONE) == 0)
21                 break;
22         }
23         d = e.modInverse(phi); // tìm d sao cho d*e=1(mod n)
24     }
25
26     public String encrypt(String message) {
27         BigInteger m = new BigInteger(message.getBytes());
28         BigInteger c = m.modPow(e, n);
29         return c.toString();
30     }
31
32     public String decrypt(String message) {
33         BigInteger c = new BigInteger(message);
34         BigInteger m = c.modPow(d, n);
35         return new String(m.toByteArray());
36     }
37
38     public static void main(String[] args) {
39         Scanner sc = new Scanner(System.in);
40         System.out.print("Plaintext: ");
41         RSA rsa = new RSA();
42         String message = sc.nextLine();
43         String encryptedMessage = rsa.encrypt(message);
44         String decryptedMessage = rsa.decrypt(encryptedMessage);
45         System.out.println("Encrypted message: " + encryptedMessage);
46         System.out.println("Decrypted message: " + decryptedMessage);
47     }
48 }
```

```
C:\Users\datnd>echo "Nguyễn Duy Đạt B21DCAT056"
"Nguyễn Duy Đạt B21DCAT056"
C:\Users\datnd>time
The current time is: 17:58:55.14
Enter the new time:
C:\Users\datnd>
```

- Sử dụng thư viện `BigInteger` trong java để triển khai mã hóa RSA:
- + `p`, `q` sẽ là 2 số nguyên tố ngẫu nhiên
- + Hàm `modInverse()`: tính nghịch đảo của `e` trong modulo  $\Phi(n)$  - Thuật toán mã hóa và giải mã: thông tin mã hóa cần phải được chuyển thành dạng byte vì các thông tin được truyền thường ở dạng chuỗi ký tự

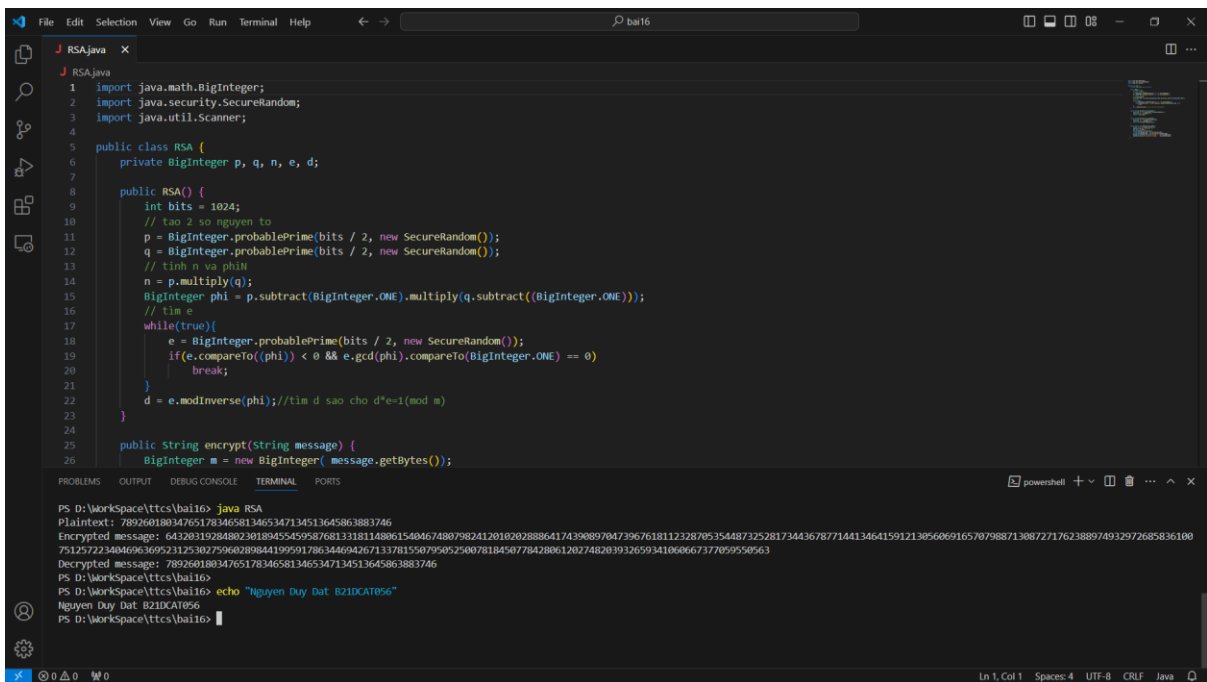


The screenshot shows the same `RSA.java` code in the IDE. Below the code editor, there is a `TERMINAL` window showing the execution of the program. It displays the current time and the encrypted message.

```
1 import java.math.BigInteger;
2 import java.security.SecureRandom;
3 import java.util.Scanner;
4
5 public class RSA {
6     private BigInteger p, q, n, e, d;
7
8     public RSA() {
9         int bits = 1024;
10         // tạo 2 số nguyên tố
11         p = BigInteger.probablePrime(bits / 2, new SecureRandom());
12         q = BigInteger.probablePrime(bits / 2, new SecureRandom());
13         // tính n và phi
14         n = p.multiply(q);
15         BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
16         // tìm e
17         while(true){
18             e = BigInteger.probablePrime(bits / 2, new SecureRandom());
19             if(e.compareTo(phi) < 0 && e.gcd(phi).compareTo(BigInteger.ONE) == 0)
20                 break;
21         }
22         d = e.modInverse(phi); // tìm d sao cho d*e=1(mod n)
23     }
24
25     public String encrypt(String message) {
26         BigInteger m = new BigInteger(message.getBytes());
```

```
PS D:\Workspace\ttcs\bai16> echo "Nguyễn Duy Đạt B21DCAT056"
Nguyễn Duy Đạt B21DCAT056
PS D:\Workspace\ttcs\bai16> date
Thursday, May 23, 2024 6:08:29 PM
PS D:\Workspace\ttcs\bai16>
```

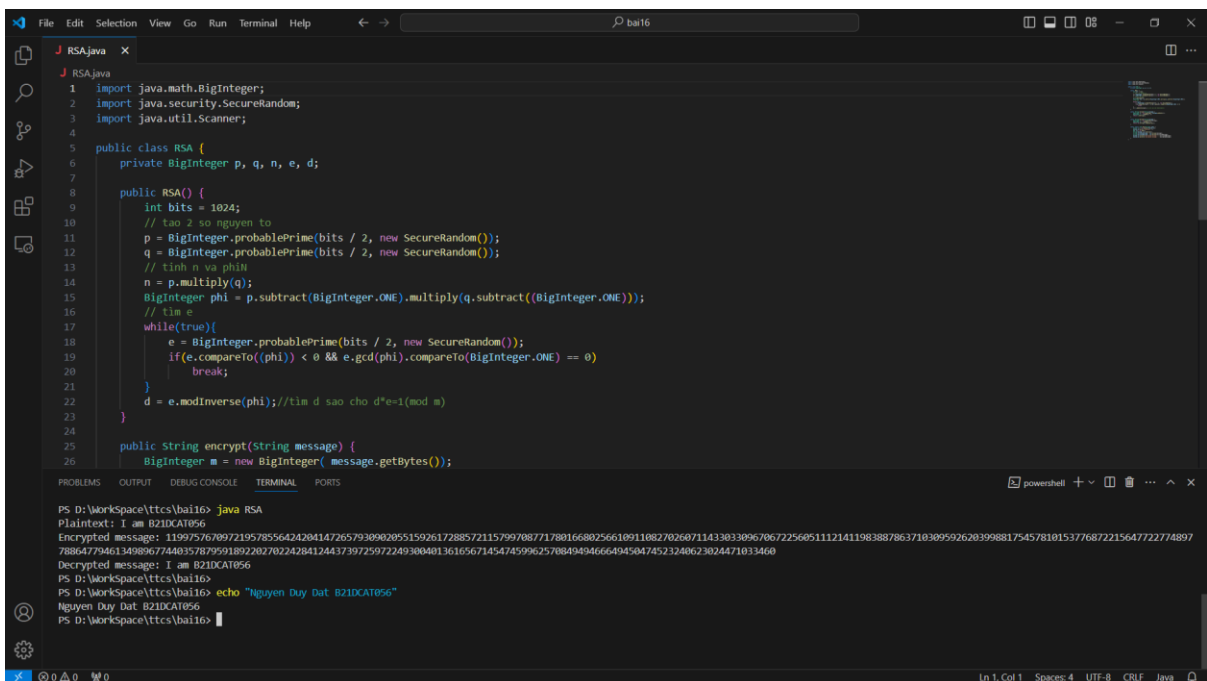
## - Thử nghiệm với số lớn



```
1 import java.math.BigInteger;
2 import java.security.SecureRandom;
3 import java.util.Scanner;
4
5 public class RSA {
6     private BigInteger p, q, n, e, d;
7
8     public RSA() {
9         int bits = 1024;
10        // tao 2 so nguyen to
11        p = BigInteger.probablePrime(bits / 2, new SecureRandom());
12        q = BigInteger.probablePrime(bits / 2, new SecureRandom());
13        // tinh n va phi
14        n = p.multiply(q);
15        BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
16        // tim e
17        while(true){
18            e = BigInteger.probablePrime(bits / 2, new SecureRandom());
19            if(e.compareTo(phi) < 0 && e.gcd(phi).compareTo(BigInteger.ONE) == 0)
20                break;
21        }
22        d = e.modInverse(phi); //tim d sao cho d*e=1(mod m)
23    }
24
25    public String encrypt(String message) {
26        BigInteger m = new BigInteger( message.getBytes());
```

```
PS D:\Workspace\ttcs\bai16> java RSA
Plaintext: 789260180347651283465813465347134513645863883746
Encrypted message: 64328319284802301894554595876813318114896154046748079824120182028886417439889784739676181123287852544873252817344367877144134641591213056069165707988713087271762388974932972685836100
751257223404696369523125302759602898441995917863446942671337815507950525007818450778428061202748203932659341060667377095959063
Decrypted message: 789260180347651283465813465347134513645863883746
PS D:\Workspace\ttcs\bai16>
PS D:\Workspace\ttcs\bai16> echo "Nguyen Duy Dat B21DCAT056"
Nguyen Duy Dat B21DCAT056
PS D:\Workspace\ttcs\bai16>
```

## - Thử nghiệm mã hóa và giải mã chuỗi ký tự: “I am B21DCAT056”



```
1 import java.math.BigInteger;
2 import java.security.SecureRandom;
3 import java.util.Scanner;
4
5 public class RSA {
6     private BigInteger p, q, n, e, d;
7
8     public RSA() {
9         int bits = 1024;
10        // tao 2 so nguyen to
11        p = BigInteger.probablePrime(bits / 2, new SecureRandom());
12        q = BigInteger.probablePrime(bits / 2, new SecureRandom());
13        // tinh n va phi
14        n = p.multiply(q);
15        BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
16        // tim e
17        while(true){
18            e = BigInteger.probablePrime(bits / 2, new SecureRandom());
19            if(e.compareTo(phi) < 0 && e.gcd(phi).compareTo(BigInteger.ONE) == 0)
20                break;
21        }
22        d = e.modInverse(phi); //tim d sao cho d*e=1(mod m)
23    }
24
25    public String encrypt(String message) {
26        BigInteger m = new BigInteger( message.getBytes());
```

```
PS D:\Workspace\ttcs\bai16> java RSA
Plaintext: I am B21DCAT056
Encrypted message: 11997576709721957855642420414726579309620951592617288572115799708771780166802566109110827026071143303309670672256051112141198388786371030959262039988175457810153776872215647722774897
7886477946134989677440357879591892202702242841244373972597224930040136165671454745996257084949466649450474523240623024471033460
Decrypted message: I am B21DCAT056
PS D:\Workspace\ttcs\bai16>
PS D:\Workspace\ttcs\bai16> echo "Nguyen Duy Dat B21DCAT056"
Nguyen Duy Dat B21DCAT056
PS D:\Workspace\ttcs\bai16>
```