Bài Lab 7: Thuật Toán RSA

- Thuật toán RSA:
  - RSA là một thuật toán mật mã hoá khoá công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hoá.Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khoá công cộng
  - Giả sử Alice và Bob cần trao đổi thông tin bí mật thông qua một kênh không an toàn (ví dụ như Internet). Với thuật toán RSA, Alice đầu tiên cần tạo ra cho mình cặp khóa gồm khóa công khai và khóa bí mật theo các bước sau
    - 1. Chọn 2 số nguyên tố lớn p và q với p≠ q, lựa chọn ngẫu nhiên và độc lập.
    - 2. Tính: n = p.q
    - 3. Tính: giá trị hàm số Ơle.
    - $\phi(n) = (p-1)(q-1)$
    - 4. Chọn một số tự nhiên *e* sao cho
    - $1 < e < \phi(n)$
    - và là số nguyên tố cùng nhau với . $\phi(n)$
    - 5. Tính: *d* sao cho.
    - $de \equiv 1 \pmod{\phi(n)}$
  - **Khóa công khai** bao gồm:
  - *n*, môđun, và
  - *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, và
  - *d*, số mũ bí mật (cũng gọi là *số mũ giải mã*).
- Ví dụ: Giả sử Bob muốn gửi đoạn thông tin *M* cho Alice. Đầu tiên Bob 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.

Lúc này Bob có *m* và biết *n* cũng như *e* do Alice gửi. Bob sẽ tính *c* là ản mã hóa của *m* theo công thức:

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

Cuối cùng Bos gửi c cho ALice .

- Alice nhận *c* từ Bob và biết khóa bí mật *d*. Alice có thể tìm được *m* từ *c* theo công thức sau:

  $m \equiv c^d \bmod n$

Biết *m*, Alice 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: $m^{ed} \equiv m \pmod p$

Và $m^{ed} \equiv m \pmod q$

Do *p* và *q* là hai số nguyên tố cùng nhau, ta có: $m^{ed} = m \pmod{p\,q}$

hay: $c^d = m \pmod n$

- Ví dụ 2:
  - Ta chọn hai số nguyên tố p và q ,với p= 5 và p = 7
  - Tính n = p*q = 5*7 = 35.
  - Z = (p- 1)*(q-1) = (5-1)(7-1) = 24
  - Tiếp đến chọn e thoả 1< e< n
  - -> chọn e= 5.
  - Tìm d ,sao cho e*d -1 chia hết cho z (24)
  - -> chọn d = 29.
  - Do đó: Pulic key =(n,e) =(35,5)
  - Private key = (n,d) =(35,29)
  - Mã hoá chuỗi sau:secure
    - Ta có bảng sau

Nội dung bị mã hoá	$M = c^d \bmod n$	Dữ liệu gốc
24	19	S
10	5	E
33	3	C
21	21	u
23	18	R
10	5	e

Lớp RSA cài đặt thuật toán mã hóa bất đối xứng RSA như sau:

```
import java.math.BigInteger ;
import java.util.Random ;
import java.io.* ;
public class RSA
{
	/**
	 * Bit length of each prime number.
	 */
	int primeSize ;

	/**
	 * Two distinct large prime numbers p and q.
	 */
	BigInteger p, q ;

	/**
	 * Modulus N.
	 */
	BigInteger N ;

	/**
	 * r = ( p - 1 ) * ( q - 1 )
	 */
	BigInteger r ;

	/**
	 * Public exponent E and Private exponent D
	 */
	BigInteger E, D ;


	public RSA(){

	}

	/**
	 * Constructor.
	 *
```

```
	 * @param   primeSize         Bit length of each prime number.
	 */

	public RSA( int primeSize )
	{
		this.primeSize = primeSize ;

		// Generate two distinct large prime numbers p and q.
		generatePrimeNumbers() ;

		// Generate Public and Private Keys.
		generatePublicPrivateKeys() ;
	}


	/**
	 * Generate two distinct large prime numbers p and q.
	 */
	public void generatePrimeNumbers()
	{
		p = BigInteger.probablePrime(primeSize /2, random);


		do
		{
			q = BigInteger.probablePrime(primeSize /2, random);
		}
		while( q.compareTo( p ) == 0 ) ;
	}


	/**
	 * Generate Public and Private Keys.
	 */
	public void generatePublicPrivateKeys()
	{
		// N = p * q
		N = p.multiply( q ) ;


		// r = ( p - 1 ) * ( q - 1 )
		r = p.subtract( BigInteger.valueOf( 1 ) ) ;
		r = r.multiply( q.subtract( BigInteger.valueOf( 1 ) ) ) ;


		// Choose E, coprime to and less than r
		do
		{
			E = new BigInteger( 2 * primeSize, new Random() ) ;
		}
		while( ( E.compareTo( r ) != -1 ) ||
		    ( E.gcd( r ).compareTo( BigInteger.valueOf( 1 ) ) != 0 ) ) ;


		// Compute D, the inverse of E mod r
		D = E.modInverse( r ) ;
	}
```

```
	/**
	 * Encrypts the plaintext (Using Public Key).
	 *
	 * @param   message String containing the plaintext message to
	 * be encrypted.
	 * @return  The ciphertext as a BigInteger array.
	 */
	public BigInteger[] encrypt( String message )
	{
		int i ;
		byte[] temp = new byte[1] ;


		byte[] digits = message.getBytes() ;

		BigInteger[] bigdigits = new BigInteger[digits.length] ;

		for( i = 0 ; i < bigdigits.length ; i++ )
		{
			temp[0] = digits[i] ;
			bigdigits[i] = new BigInteger( temp ) ;
		}

		BigInteger[] encrypted = new BigInteger[bigdigits.length] ;

		for( i = 0 ; i < bigdigits.length ; i++ )
			encrypted[i] = bigdigits[i].modPow( E, N ) ;


		return( encrypted ) ;
	}

	public BigInteger[] encrypt( String message,BigInteger userD,BigInteger
userN)
		{
			int i ;
			byte[] temp = new byte[1] ;


			byte[] digits = message.getBytes() ;

			BigInteger[] bigdigits = new BigInteger[digits.length] ;

			for( i = 0 ; i < bigdigits.length ; i++ )
			{
				temp[0] = digits[i] ;
				bigdigits[i] = new BigInteger( temp ) ;
			}

			BigInteger[] encrypted = new BigInteger[bigdigits.length] ;

			for( i = 0 ; i < bigdigits.length ; i++ )
				encrypted[i] = bigdigits[i].modPow( userD, userN ) ;
```

```
                return( encrypted ) ;
        }
        /**
         * Decrypts the ciphertext (Using Private Key).
         *
         * @param   encrypted         BigInteger array containing
         * the ciphertext to be decrypted.
         * @return  The decrypted plaintext.
         */
        public String decrypt( BigInteger[] encrypted,BigInteger D,BigInteger N
)
        {
                int i ;


                BigInteger[] decrypted = new BigInteger[encrypted.length] ;

                for( i = 0 ; i < decrypted.length ; i++ )
                        decrypted[i] = encrypted[i].modPow( D, N ) ;

                char[] charArray = new char[decrypted.length] ;

                for( i = 0 ; i < charArray.length ; i++ )
                        charArray[i] = (char) ( decrypted[i].intValue() ) ;


                return( new String( charArray ) ) ;
        }


        /**
         * Get prime number p.
         *
         * @return  Prime number p.
         */
        public BigInteger getp()
        {
                return( p ) ;
        }


        /**
         * Get prime number q.
         *
         * @return  Prime number q.
         */
        public BigInteger getq()
        {
                return( q ) ;
        }


        /**
         * Get r.
         *
         * @return  r.
         */
```

```
	public BigInteger getr()
	{
		return( r ) ;
	}


	/**
	 * Get modulus N.
	 *
	 * @return  Modulus N.
	 */
	public BigInteger getN()
	{
		return( N ) ;
	}


	/**
	 * Get Public exponent E.
	 *
	 * @return  Public exponent E.
	 */
	public BigInteger getE()
	{
		return( E ) ;
	}


	/**
	 * Get Private exponent D.
	 *
	 * @return  Private exponent D.
	 */
	public BigInteger getD()
	{
		return( D ) ;
	}



	/**
	 * RSA Main program for Unit Testing.
	 */
	public static void main( String[] args ) throws IOException
	{
		/*if( args.length != 1 )
		{
			System.out.println( "Syntax: java RSA PrimeSize" ) ;
			System.out.println( "e.g. java RSA 8" ) ;
			System.out.println( "e.g. java RSA 512" ) ;

			System.exit( -1 ) ;
		}

		// Get bit length of each prime number
		int primeSize = Integer.parseInt( args[0] ) ;*/
		int primeSize =8;
```

```
            // Generate Public and Private Keys

            RSA rsa = new RSA( primeSize ) ;

            System.out.println( "Key Size: [" + primeSize + "]" );
            System.out.println( "" ) ;

            System.out.println( "Generated prime numbers p and q" );
            System.out.println( "p: [" + rsa.getp().toString( 16
).toUpperCase() + "]" );
            System.out.println( "q: [" + rsa.getq().toString( 16
).toUpperCase() + "]" );
            System.out.println( "" ) ;

            System.out.println( "The public key is the pair (N, E) which will
be published." ) ;
            System.out.println( "N: [" + rsa.getN().toString( 16
).toUpperCase() + "]" ) ;
            System.out.println( "E: [" + rsa.getE().toString( 16
).toUpperCase() + "]" ) ;
            System.out.println( "" ) ;

            System.out.println( "The private key is the pair (N, D) which
will be kept private." ) ;
            System.out.println( "N: [" + rsa.getN().toString( 16
).toUpperCase() + "]" ) ;
            System.out.println( "D: [" + rsa.getD().toString( 16
).toUpperCase() + "]" ) ;
            System.out.println( "" ) ;


            // Get message (plaintext) from user
            System.out.println( "Please enter message (plaintext):" ) ;
            String plaintext = ( new BufferedReader( new InputStreamReader(
System.in ) ) ).readLine() ;
            System.out.println( "" ) ;

            // Encrypt Message
            BigInteger[] ciphertext = rsa.encrypt( plaintext ) ;

            System.out.print( "Ciphertext: [" ) ;
            for( int i = 0 ; i < ciphertext.length ; i++ )
            {
                  System.out.print( ciphertext[i].toString( 16
).toUpperCase() ) ;

                  if( i != ciphertext.length - 1 )
                        System.out.print( " " ) ;
            }
            System.out.println( "]" ) ;
            System.out.println( "" ) ;

            RSA rsa1 = new RSA(8);

            String recoveredPlaintext = rsa1.decrypt( ciphertext
,rsa.getD(),rsa.getN()) ;
```

```
            System.out.println( "Recovered plaintext: [" + recoveredPlaintext
+ "]" ) ;
        }
}
```

- Bài tập hướng dẫn: Mã hóa và giải mã văn bản sử dụng thuật toán RSA.

```
public class PwdEncryption{

public static void main(String args[]){
    Scanner in = new Scanner(System.in);
        String nhash;
        BigInteger[] ciphertext = null;
        BigInteger n = null;
        BigInteger d = null;
        String password="";
        System.out.println("Enter text to be encrypted:");
        password=in.nextLine();

        System.out.println("Password (Input) : "+ password);
        //8FD1 5FB3 5057 5FB3 65E63AB879713AB8


        RSA rsa = new RSA( 8 ) ;
                n=rsa.getN();
                d=rsa.getD();
                ciphertext = rsa.encrypt(password) ;



                StringBuffer bf = new StringBuffer();
                for( int i = 0 ; i < ciphertext.length ; i++ )
                {
                        bf.append( ciphertext[i].toString( 16 ).toUpperCase() ) ;
```

```
                if( i != ciphertext.length - 1 )
                        System.out.print( " " ) ;
        }


        String message=bf.toString();
        System.out.println();
        System.out.println("Encrypted Message : "+message);
    3D1D2E

        String dhash  =  rsa.decrypt( ciphertext ,d,n) ;
        System.out.println();
        System.out.println("Decrypted Message : "+dhash);     }
                                                             }
```

- Kết quả chương trình:

```
Enter text to be encrypted:
Hutech technology
Password (Input) : Hutech technology

Encrypted Message : 3874721B80BE3AE0396F7FF7FCD80BE3AE0396F7FF4FE4365978F4365986B9DA9

Decrypted Message : Hutech technology
BUILD SUCCESSFUL (total time: 23 seconds)
```

- Bài Tập NC : Viết chương trình mã hóa và giải mã sử dụng thuật toán RSA. Yêu cầu chương trình:
  - Thiết kế form chứa thông tin sau: Họ và tên đầy đủ, địa chỉ địa chỉ email, số điện thoại, nhập password có ký tự đặc biệt.
  - Viết hàm xử lý mã hóa và giải mã .