IT 8761

Security Labaratory

3122171041 48

Sharon Julia. S

18.12.2020

AN

Aim:

To develop a java program to implement the RSA algorithm with a module to check the p and q as prime numbers using Miller Rabin primality checking algorithm.

Procedure:

Algorithm - RSA

i) generate 2 large random prime p and q using Miller Rabin algorithm

ii) Compute the product n = pq

iii) compute phi = (p-1)(q-1)

iv) choose an integer e, $1 < e < phi$, such that $gcd(e, phi) = 1$

v) compute secret exponent d, $1 < d < phi$, such that $ed \equiv 1 \pmod{phi}$

vi) public key is (n, e) and private key is (d, p, q)

Encryption procedure for RSA:

Sender does

i) Obtain the recipient B's public key (n, e)

ii) Represents plaintext message as a positive integer m, $1 < m$

iii) Computes cipher text $C = m^e \bmod n$

iv) Display cipher text C

Decryption procedure for RSA:

i) input the ciphertext

ii) Use private key (n,d) to compute

$m = c^d \bmod n$

iii) extract plaintext from message m

iv) Display plaintext

Algorithm - Miller Rabin

i) Find integers k, q with $k > 0$, q odd so that $(n-1) = 2^k q$

ii) select random int a, $1 < a < n-1$

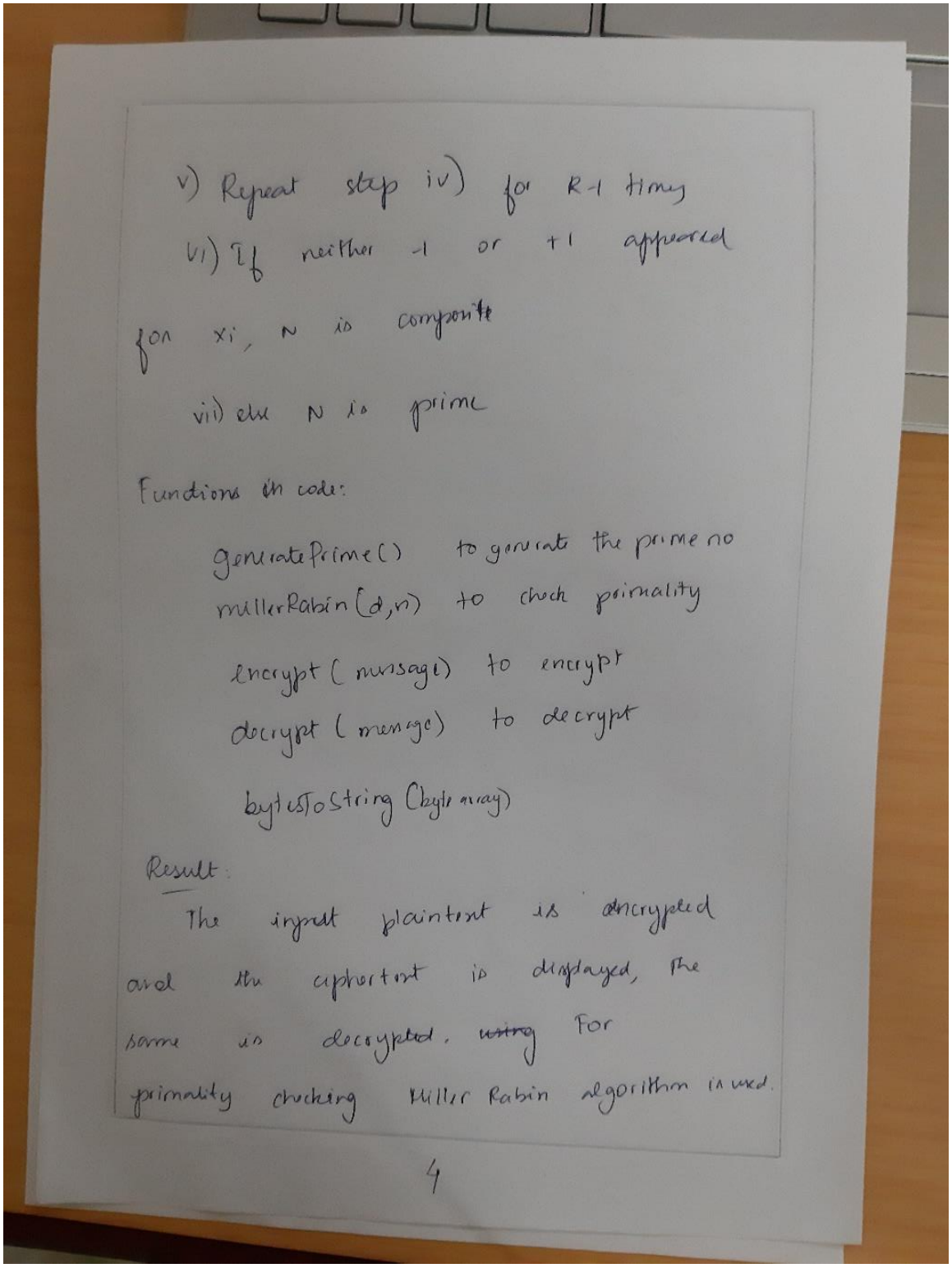
iii) if $a^q \bmod n = 1$ then inconclusive

iv) for k time do

if $a^{2^j q} \bmod n = n-1$

return inconclusive

v) Display or return composite



CODE:

```
import java.io.IOException;
import java.math.BigInteger;
import java.util.*;
```

```
class Main {
  private BigInteger p;
  private BigInteger q;
  private BigInteger N;
  private BigInteger phi;
  private BigInteger e;
  private BigInteger d;
  private int bitlength=64;
  private Random random;
  private Random r;
  static BigInteger ZERO= BigInteger.ZERO;
  static BigInteger ONE=BigInteger.ONE;
  static BigInteger big2=new BigInteger("2");
  static BigInteger big3=new BigInteger("3");
  static BigInteger big4=new BigInteger("4");
  int noOfIterations=5;

  void generatePrime(){
    byte[] b=new byte[bitlength/8];
    random.nextBytes(b);
    p=new BigInteger(b);
    while (!isPrime(p,noOfIterations)){
      random.nextBytes(b);
      q=new BigInteger(b);
    }
  }

  boolean millerRabin(BigInteger d,BigInteger n){
    BigInteger maxLimit=n.subtract(big2);
    BigInteger minLimit=big2;
    BigInteger bigInteger=maxLimit.subtract(minLimit);
    int len=maxLimit.bitLength();
    BigInteger a=new BigInteger(len,random);
    if (a.compareTo(minLimit)<0) a=a.add(minLimit);
    if (a.compareTo(bigInteger)>=0) a=a.mod(bigInteger).add(minLimit);
    //
    if (a.compareTo(bigInteger)>=0) a=a.mod(bigInteger).add(minLimit);
    BigInteger x=a.modPow(d,n);

    if (x.compareTo(ONE)==0 || x.compareTo(n.subtract(ONE))==0) return true;

    while(d.compareTo(n.subtract(ONE))!=0){
      x=x.multiply(x).mod(n);
      d=d.multiply(big2);
      if (x.compareTo(ONE)==0) return false;
      if (x.compareTo(n.subtract(ONE))==0) return true;
    }
    return false;
```

```
}
boolean isPrime(BigInteger n,int k){
  if (n.compareTo(ONE)<=0 || n.compareTo(big4)==0)
  return false;
  if (n.compareTo(big3)<=0)
  return true;
  BigInteger d=n.subtract(ONE);
  while(d.mod(big2).signum()==0) d=d.divide(big2);
  for (int i=0;i<k;i++) if (!millerRabin(d,n))
  return false;

  return true;
}

public Main(){
  r=new Random();
  random=new Random();
  p=BigInteger.probablePrime(bitlength,r);
  q=BigInteger.probablePrime(bitlength, r);

  System.out.println("p:"+p+"\nq:"+q);
  System.out.println("checking primality of p and q");

  if (isPrime(p,10)){
    System.out.println("verified p as prime");

  }
  if (isPrime(q,10)){
    System.out.println("Verified q as prime");
  }

  N=p.multiply(q);
  System.out.println("n :"+N);
  phi=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
  System.out.println("\nEuler totient(phi) :"+phi+"\n");
  e=BigInteger.probablePrime(bitlength,r);
  System.out.println("e:"+e+"\n");
  while(phi.gcd(e).compareTo(BigInteger.ONE)>0 && e.compareTo(phi)<0){
    e.add(BigInteger.ONE);
  }
  d=e.modInverse(phi);

}

private static String bytesToString(byte[] encrypted){
  String test="";
  for (byte b:encrypted){
```

```
      test+=Byte.toString(b);
    }
    return test;
  }

  //encrypt
  public byte[] encrypt(byte[] message){
    return (new BigInteger(message)).modPow(e,N).toByteArray();
  }

  public byte[] decrypt(byte[] message){
    return (new BigInteger(message)).modPow(d,N).toByteArray();

  }
  public static void main(String[] args) {
    Main rsa=new Main();
    Scanner in=new Scanner(System.in);
    String plaintext;

    System.out.println("Enter plain text");
    plaintext=in.next();

    byte[] encrypted=rsa.encrypt(plaintext.getBytes());

    byte[] decrypted=rsa.decrypt(encrypted);

    System.out.println("Decrypted Bytes:"+bytesToString(decrypted));

    System.out.println("Decrypted plaintext:"+new String(decrypted));

  }
}
```

OUTPUT:

javac -classpath .:/run_dir/junit-4.12.jar:target/dependency/* -d . Main.java

 java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main

p:16694898077293859083

q:13432161899971119203

checking primality of p and q

verified p as prime

Verified q as prime

n :224248573877727667116335523446477270849

Euler totient(phi) :22424857387772766708620846346921229256 4

e:17106681859251098167

Enter plain text

millerrabin

Decrypted Bytes:109105108108101114114979810511 0

Decrypted plaintext:millerrabin