МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

ОТЧЕТ

по лабораторной работе №5

по дисциплине

«Средства и методы защиты информации»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ Капранов С. Н.\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ Сорокин Е. А. \_\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_\_ 18-В1 \_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

**Задание №5**

Реализовать алгоритм шифрования данных «Система омофонов».

**Теория**

**RSA** (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи.

**Алгоритм**

Алгоритм создания открытого и секретного ключей

RSA-ключи генерируются следующим образом:

1) выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое);

2) вычисляется их произведение n=pq, которое называется модулем;

3) вычисляется значение функции Эйлера от числа n: φ(n)=(p-1)(q-1);

4) выбирается целое число e (1<e<φ(n)), взаимно простое со значением функции φ(n);

число e называется открытой экспонентой (англ. public exponent);

обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые из чисел Ферма: 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием быстрого возведения в степень, будет меньше;

слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA;

5) вычисляется число d, мультипликативно обратное к числу e по модулю φ(n), то есть число, удовлетворяющее сравнению:

de≡1 (mod φ(n))

(число d называется секретной экспонентой; обычно оно вычисляется при помощи расширенного алгоритма Евклида);

6) пара (e,n) публикуется в качестве открытого ключа RSA (англ. RSA public key);

7) пара (d,n) играет роль закрытого ключа RSA (англ. RSA private key) и держится в секрете.

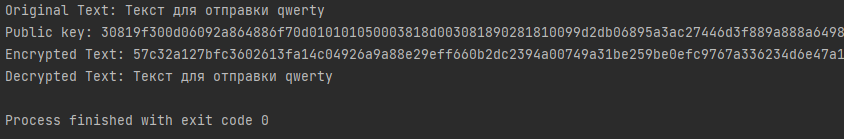
Для шифрования, данные должны быть представлены в виде чисел от 0 до n-1. Обозначим такое число за m.

Для шифрования необходимо возвести это число в степень e по модулю n:

c = me (mod n)

Для расшифровки необходимо шифрованное число возвести в степень d по модулю n:

m = cd (mod n)

**Пример работы программы**

import java.io.\*;  
import java.security.\*;  
import java.security.spec.EncodedKeySpec;  
import java.security.spec.InvalidKeySpecException;  
import java.security.spec.PKCS8EncodedKeySpec;  
import java.security.spec.X509EncodedKeySpec;  
  
import javax.crypto.Cipher;  
  
public class CryptoUtil {  
  
 public static final String ALGORITHM = "RSA";  
 public static final String PRIVATE\_KEY\_FILE = "private.key";  
 public static final String PUBLIC\_KEY\_FILE = "public.key";  
  
 public static void generateKey() {  
 try {  
 final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);  
 keyGen.initialize(1024, new SecureRandom());  
 final KeyPair key = keyGen.generateKeyPair();  
  
 File privateKeyFile = new File(PRIVATE\_KEY\_FILE);  
 File publicKeyFile = new File(PUBLIC\_KEY\_FILE);  
  
 if (privateKeyFile.getParentFile() != null) {  
 privateKeyFile.getParentFile().mkdirs();  
 }  
 privateKeyFile.createNewFile();  
  
 if (publicKeyFile.getParentFile() != null) {  
 publicKeyFile.getParentFile().mkdirs();  
 }  
 publicKeyFile.createNewFile();  
  
 BufferedWriter pubOut = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(publicKeyFile)));  
 pubOut.write(byte2Hex(key.getPublic().getEncoded()));  
 pubOut.flush();  
 pubOut.close();  
  
 BufferedWriter privOut = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(privateKeyFile)));  
 privOut.write(byte2Hex(key.getPrivate().getEncoded()));  
 privOut.flush();  
 privOut.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 public static byte[] encrypt(String text, PublicKey key) {  
 byte[] cipherText = null;  
 try {  
 final Cipher cipher = Cipher.getInstance(ALGORITHM);  
 cipher.init(Cipher.ENCRYPT\_MODE, key);  
 cipherText = cipher.doFinal(text.getBytes());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return cipherText;  
 }  
  
 public static String decrypt(byte[] text, PrivateKey key) {  
 byte[] dectyptedText = null;  
 try {  
 final Cipher cipher = Cipher.getInstance(ALGORITHM);  
 cipher.init(Cipher.DECRYPT\_MODE, key);  
 dectyptedText = cipher.doFinal(text);  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 return new String(dectyptedText);  
 }  
  
 private static byte[] fileToKey(String file) throws IOException {  
 BufferedReader pubIn = new BufferedReader(new InputStreamReader(new FileInputStream(file)));  
 StringBuilder sb = new StringBuilder();  
 String tmp;  
 do {  
 tmp = pubIn.readLine();  
 if (tmp != null) sb.append(tmp);  
 } while (tmp != null);  
 return hex2Byte(sb.toString());  
 }  
  
 private static PublicKey restorePublic() throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {  
 KeyFactory keyFactory = KeyFactory.getInstance(ALGORITHM);  
 EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(fileToKey(PUBLIC\_KEY\_FILE));  
 return keyFactory.generatePublic(publicKeySpec);  
 }  
  
 private static PrivateKey restorePrivate() throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {  
 KeyFactory keyFactory = KeyFactory.getInstance(ALGORITHM);  
 EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(fileToKey(PRIVATE\_KEY\_FILE));  
 return keyFactory.generatePrivate(privateKeySpec);  
 }  
  
 public static void main(String[] args) {  
  
 try {  
 if (!new File(PRIVATE\_KEY\_FILE).exists() || !new File(PUBLIC\_KEY\_FILE).exists()) {  
 generateKey();  
 }  
  
 final String originalText = "Текст для отправки qwerty";  
 final byte[] encryptedText = encrypt(originalText, restorePublic());  
  
 String plainText = decrypt(encryptedText, restorePrivate());  
  
 System.out.println("Original Text: " + originalText);  
 System.out.println("Public key: " + byte2Hex(restorePublic().getEncoded()));  
 System.out.println("Encrypted Text: " + byte2Hex(encryptedText));  
 System.out.println("Decrypted Text: " + plainText);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static String byte2Hex(byte b[]) {  
 java.lang.String hs = "";  
 java.lang.String stmp = "";  
 for (int n = 0; n < b.length; n++) {  
 stmp = java.lang.Integer.toHexString(b[n] & 0xff);  
 if (stmp.length() == 1)  
 hs = hs + "0" + stmp;  
 else  
 hs = hs + stmp;  
 }  
 return hs.toLowerCase();  
 }  
  
 public static byte hex2Byte(char a1, char a2) {  
 int k;  
 if (a1 >= '0' && a1 <= '9') k = a1 - 48;  
 else if (a1 >= 'a' && a1 <= 'f') k = (a1 - 97) + 10;  
 else if (a1 >= 'A' && a1 <= 'F') k = (a1 - 65) + 10;  
 else k = 0;  
 k <<= 4;  
 if (a2 >= '0' && a2 <= '9') k += a2 - 48;  
 else if (a2 >= 'a' && a2 <= 'f') k += (a2 - 97) + 10;  
 else if (a2 >= 'A' && a2 <= 'F') k += (a2 - 65) + 10;  
 else k += 0;  
 return (byte) (k & 0xff);  
 }  
  
 public static byte[] hex2Byte(String str) {  
 int len = str.length();  
 if (len % 2 != 0) return null;  
 byte r[] = new byte[len / 2];  
 int k = 0;  
 for (int i = 0; i < str.length() - 1; i += 2) {  
 r[k] = hex2Byte(str.charAt(i), str.charAt(i + 1));  
 k++;  
 }  
 return r;  
 }  
}