

PRACTICA DE JAVA DE ENCRIPCIÓN ASIMETRICA

Enunciat

En aquesta segona pràctica provarem d'encriptar i desencriptar arxius amb programes escrits en Java. En el moodle del mòdul 9 trobaràs arxius per crear claus RSA, per encriptar i desencriptar, sino et funcionen, busca'ls per internet.

Pràctica

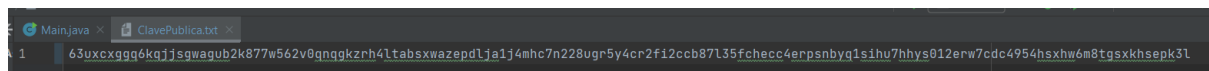
1. Busca per internet codi font en Java per crear claus pública i privada.

CODIGO DE CREACIÓN DE CLAVES, ENCRIPCIÓN Y DESENCRIPTACIÓN EN EL EJERCICIO 3

2. Executa el programa i fes una demostració amb captures de:

- Generació de clau publica

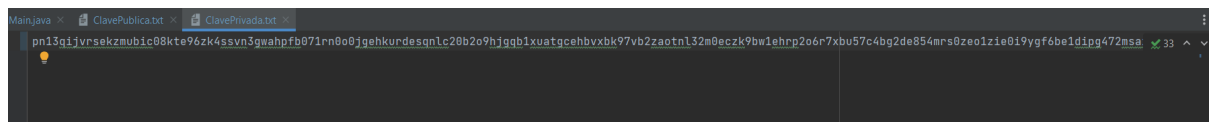
EN EL FICHERO DADO EN MAIN SE GENERA LA CLAVE :



```
Main.java x  ClavePublica.txt x
63uxcxggg6kqjjsqwgub2k877w562v0qnggkzrh4ltabsxwazepdlja1j4mhc7n228ugr5y4cr2fi2ccb87l35fchecc4erpsnbyq1sihu7hhys012erw7cdc4954hsxhw6m8tgsxkhsepk3l
```

- Generació de clau privada

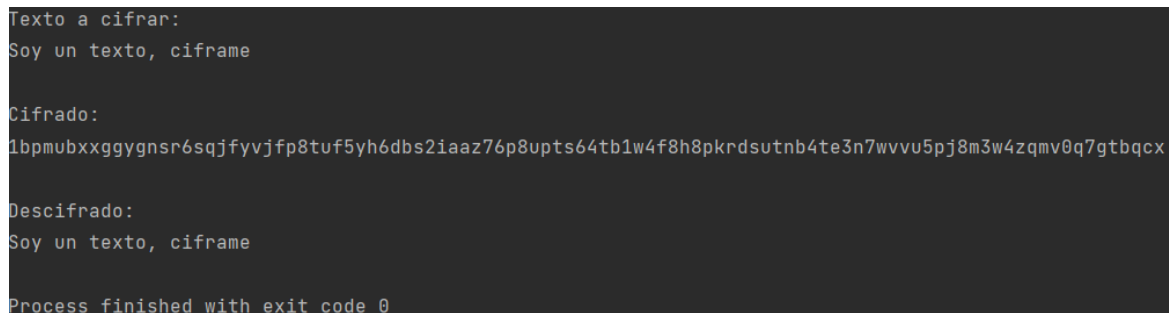
EN EL FICHERO DADO EN MAIN SE GENERA LA CLAVE:



```
Main.java x  ClavePublica.txt x  ClavePrivada.txt x
pn13giivrsekzmublc08kte96zk4ssvn3gwahpfb071rn00jgehkundesqnlc20b2o9hjgqb1xuatqcehbvxk97vb2zaotnl32m0ecz9bw1ehrp2o6r7xbu57c4bg2de854mrs0zeo1zie0i9ygf6be1dipg472msa 33 ^
```

- Encriptació i desencriptació d'un text o arxiu

lo hace todo la main con la clase rsa mostradas en el ejercicio 3



```
Texto a cifrar:
Soy un texto, ciframe

Cifrado:
1bpmubxxggygnsr6sqjfyvjfp8tuf5yh6dbs2iaaz76p8upts64tb1w4f8h8pkrdsutnb4te3n7wvvu5pj8m3w4zqmv0q7gtbqcx

Descifrado:
Soy un texto, ciframe

Process finished with exit code 0
```

3. Llista el codi font i afegeix comentaris de que fan les diferents parts.

clase con todas las funciones utilizadas para la creacion de claves, encriptación y desencriptación:

```

package CriptografiaAsimetrica;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.io.Writer;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Arrays;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

public class Rsa {

    public PrivateKey PrivateKey = null;
    public PublicKey PublicKey = null;

    public Rsa() {

    }

    public void setPrivateKeyString(String key) throws
    NoSuchAlgorithmException, InvalidKeySpecException {
        byte[] encodedPrivateKey = stringToBytes(key);

        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PKCS8EncodedKeySpec privateKeySpec = new
        PKCS8EncodedKeySpec(encodedPrivateKey);
        PrivateKey privateKey =
        keyFactory.generatePrivate(privateKeySpec);
        this.PrivateKey = privateKey;
    }

```

```
    public String getPrivateKeyString() {  
        PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new  
PKCS8EncodedKeySpec(this.PrivateKey.getEncoded());  
        return bytesToString(pkcs8EncodedKeySpec.getEncoded());  
    }
```

```
    public void setPublicKeyString(String key) throws  
NoSuchAlgorithmException, InvalidKeySpecException {  
        byte[] encodedPublicKey = stringToBytes(key);
```

```
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");  
        X509EncodedKeySpec publicKeySpec = new  
X509EncodedKeySpec(encodedPublicKey);  
        PublicKey publicKey =  
keyFactory.generatePublic(publicKeySpec);  
        this.PublicKey = publicKey;  
    }
```

```
    public String getPublicKeyString() {  
        X509EncodedKeySpec x509EncodedKeySpec = new  
X509EncodedKeySpec(this.PublicKey.getEncoded());  
        return bytesToString(x509EncodedKeySpec.getEncoded());  
    }
```

```
    public void genKeyPair(int size) throws  
NoSuchAlgorithmException, NoSuchPaddingException,  
InvalidKeyException,  
IllegalBlockSizeException, BadPaddingException {
```

```
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");  
        kpg.initialize(size);  
        KeyPair kp = kpg.genKeyPair();
```

```
        PublicKey publicKey = kp.getPublic();  
        PrivateKey privateKey = kp.getPrivate();
```

```
        this.PrivateKey = privateKey;  
        this.PublicKey = publicKey;  
    }
```

```
    public String Encrypt(String plain)  
        throws NoSuchAlgorithmException,  
NoSuchPaddingException, InvalidKeyException,  
IllegalBlockSizeException,  
BadPaddingException, InvalidKeySpecException,  
UnsupportedEncodingException, NoSuchProviderException {
```

```

        byte[] encryptedBytes;

        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, this.PublicKey);
        encryptedBytes = cipher.doFinal(plain.getBytes());

        return bytesToString(encryptedBytes);
    }

    public String Decrypt(String result) throws
NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException,
        IllegalBlockSizeException, BadPaddingException {

        byte[] decryptedBytes;

        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, this.PrivateKey);
        decryptedBytes = cipher.doFinal(stringToBytes(result));
        return new String(decryptedBytes);
    }

    public String bytesToString(byte[] b) {
        byte[] b2 = new byte[b.length + 1];
        b2[0] = 1;
        System.arraycopy(b, 0, b2, 1, b.length);
        return new BigInteger(b2).toString(36);
    }

    public byte[] stringToBytes(String s) {
        byte[] b2 = new BigInteger(s, 36).toByteArray();
        return Arrays.copyOfRange(b2, 1, b2.length);
    }

    public void saveToDiskPrivateKey(String path) throws
IOException {
        try {
            Writer out = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(path), "UTF-8"));
            out.write(this.getPrivateKeyString());
            out.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public void saveToDiskPublicKey(String path) {

```

```

        try {
            Writer out = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(path), "UTF-8"));
            out.write(this.getPublicKeyString());
            out.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

```

    public void openFromDiskPublicKey(String path)
        throws IOException, NoSuchAlgorithmException,
InvalidKeySpecException {
        String content = this.readFileAsString(path);
        this.setPublicKeyString(content);
    }
}

```

```

    public void openFromDiskPrivateKey(String path)
        throws IOException, NoSuchAlgorithmException,
InvalidKeySpecException {
        String content = this.readFileAsString(path);
        this.setPrivateKeyString(content);
    }
}

```

```

    private String readFileAsString(String filePath) throws
IOException {
        StringBuffer fileData = new StringBuffer();
        BufferedReader reader = new BufferedReader(new
FileReader(filePath));
        char[] buf = new char[1024];
        int numRead = 0;
        while ((numRead = reader.read(buf)) != -1) {
            String readData = String.valueOf(buf, 0, numRead);
            fileData.append(readData);
        }
        reader.close();
        return fileData.toString();
    }
}

```

```

}

```

CLASE MAIN:

```

import CriptografiaAsimetrica.Rsa;

```

```

public class Main {

```

```

    public static void main(String[] args) throws Exception {

```

//Definimos un texto a cifrar

```
String ciframe = "Soy un texto, ciframe";
```

```
System.out.println("Texto a cifrar: ");
```

```
System.out.println(ciframe);
```

```
Rsa rsa = new Rsa();
```

//Generamos un par de claves

//Admite claves de 512, 1024, 2048 y 4096 bits

```
rsa.genKeyPair(512);
```

```
String file_private = "./ClavePrivada";
```

```
String file_public = "./ClavePublica";
```

```
rsa.saveToDiskPrivateKey(file_private);
```

```
rsa.saveToDiskPublicKey(file_public);
```

//Ciframos y e imprimimos, el texto cifrado

//es devuelto en la variable secure

```
String secure = rsa.Encrypt(ciframe);
```

```
System.out.println("\nCifrado:");
```

```
System.out.println(secure);
```

//Le pasamos el texto cifrado (secure) y nos

//es devuelto el texto ya descifrado (unsecure)

```
String unsecure = rsa2.Decrypt(secure);
```

```
//Imprimimos
```

```
System.out.println("\nDescifrado:");
```

```
System.out.println(unsecure);
```

```
}
```



4. Indica el protocolo de xifratge que emprà i les longituds de les claus.

Usa el protocolo RSA y la longitud de las claves es 512 bits