**[Java加密解密简单实现](http://blog.csdn.net/wang_zhou_jian/article/details/5635659)**

加密算法有很多种：这里只大约列举几例：

1:消息摘要：（数字指纹）：既对一个任意长度的一个数据块进行计算，产生一个唯一指纹。MD5/SHA1  
发送给其他人你的信息和摘要,其他人用相同的加密方法得到摘要，最后进行比较摘要是否相同。  
  
2:单匙密码体制:DES:比较简便高效,密钥简短，加解密速度快，破译极其困难,但其安全性依赖于密匙的安全性。  
DES（Data Encryption Standard）是发明最早的最广泛使用的分组对称加密算法。DES算法的入口参数有三个：Key、Data、Mode。其中Key为8个字节共64 位，是DES算法的工作密钥；Data也为8个字节64位，是要被加密或被解密的数据；Mode为DES的工作方式，有两种：加密或解密  
  
3: 数字签名:就是信息发送者用其私钥对从所传报文中提取出的特征数据（或称数字指纹）进行RSA算法操作，以保证发信人无法抵赖曾发过该信息（即不可抵赖 性），同时也确保信息报文在经签名后末被篡改（即完整性）。当信息接收者收到报文后，就可以用发送者的公钥对数字签名进行验证。  
代表：DSA  
  
4:非对称密匙密码体制（公匙体系）：加密密匙不同于解密密匙，加密密匙公之于众，谁都可以使用，解密密匙只有解密人自己知道。代表：RSA

下面是对上面几个例子进行的简单实现：

**package** com.train.encrypt;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.security.KeyPair;

**import** java.security.KeyPairGenerator;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.security.SecureRandom;

**import** java.security.Signature;

**import** javax.crypto.Cipher;

**import** javax.crypto.KeyGenerator;

**import** javax.crypto.SecretKey;

/\*\*

\* 加密解密

\*

\* **@author** shy.qiu

\* **@since** http://blog.csdn.net/qiushyfm

\*/

**public** **class** CryptTest {

/\*\*

\* 进行MD5加密

\*

\* **@param** info

\* 要加密的信息

\* **@return** String 加密后的字符串

\*/

**public** String encryptToMD5(String info) {

**byte**[] digesta = **null**;

**try** {

// 得到一个md5的消息摘要

MessageDigest alga = MessageDigest.*getInstance*("MD5");

// 添加要进行计算摘要的信息

alga.update(info.getBytes());

// 得到该摘要

digesta = alga.digest();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

}

// 将摘要转为字符串

String rs = byte2hex(digesta);

**return** rs;

}

/\*\*

\* 进行SHA加密

\*

\* **@param** info

\* 要加密的信息

\* **@return** String 加密后的字符串

\*/

**public** String encryptToSHA(String info) {

**byte**[] digesta = **null**;

**try** {

// 得到一个SHA-1的消息摘要

MessageDigest alga = MessageDigest.*getInstance*("SHA-1");

// 添加要进行计算摘要的信息

alga.update(info.getBytes());

// 得到该摘要

digesta = alga.digest();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

}

// 将摘要转为字符串

String rs = byte2hex(digesta);

**return** rs;

}

// //////////////////////////////////////////////////////////////////////////

/\*\*

\* 创建密匙

\*

\* **@param** algorithm

\* 加密算法,可用 DES,DESede,Blowfish

\* **@return** SecretKey 秘密（对称）密钥

\*/

**public** SecretKey createSecretKey(String algorithm) {

// 声明KeyGenerator对象

KeyGenerator keygen;

// 声明 密钥对象

SecretKey deskey = **null**;

**try** {

// 返回生成指定算法的秘密密钥的 KeyGenerator 对象

keygen = KeyGenerator.*getInstance*(algorithm);

// 生成一个密钥

deskey = keygen.generateKey();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

}

// 返回密匙

**return** deskey;

}

/\*\*

\* 根据密匙进行DES加密

\*

\* **@param** key

\* 密匙

\* **@param** info

\* 要加密的信息

\* **@return** String 加密后的信息

\*/

**public** String encryptToDES(SecretKey key, String info) {

// 定义 加密算法,可用 DES,DESede,Blowfish

String Algorithm = "DES";

// 加密随机数生成器 (RNG),(可以不写)

SecureRandom sr = **new** SecureRandom();

// 定义要生成的密文

**byte**[] cipherByte = **null**;

**try** {

// 得到加密/解密器

Cipher c1 = Cipher.*getInstance*(Algorithm);

// 用指定的密钥和模式初始化Cipher对象

// 参数:(ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE,UNWRAP\_MODE)

c1.init(Cipher.***ENCRYPT\_MODE***, key, sr);

// 对要加密的内容进行编码处理,

cipherByte = c1.doFinal(info.getBytes());

} **catch** (Exception e) {

e.printStackTrace();

}

// 返回密文的十六进制形式

**return** byte2hex(cipherByte);

}

/\*\*

\* 根据密匙进行DES解密

\*

\* **@param** key

\* 密匙

\* **@param** sInfo

\* 要解密的密文

\* **@return** String 返回解密后信息

\*/

**public** String decryptByDES(SecretKey key, String sInfo) {

// 定义 加密算法,

String Algorithm = "DES";

// 加密随机数生成器 (RNG)

SecureRandom sr = **new** SecureRandom();

**byte**[] cipherByte = **null**;

**try** {

// 得到加密/解密器

Cipher c1 = Cipher.*getInstance*(Algorithm);

// 用指定的密钥和模式初始化Cipher对象

c1.init(Cipher.***DECRYPT\_MODE***, key, sr);

// 对要解密的内容进行编码处理

cipherByte = c1.doFinal(hex2byte(sInfo));

} **catch** (Exception e) {

e.printStackTrace();

}

// return byte2hex(cipherByte);

**return** **new** String(cipherByte);

}

// /////////////////////////////////////////////////////////////////////////////

/\*\*

\* 创建密匙组，并将公匙，私匙放入到指定文件中

\*

\* 默认放入mykeys.bat文件中

\*/

**public** **void** createPairKey() {

**try** {

// 根据特定的算法一个密钥对生成器

KeyPairGenerator keygen = KeyPairGenerator.*getInstance*("DSA");

// 加密随机数生成器 (RNG)

SecureRandom random = **new** SecureRandom();

// 重新设置此随机对象的种子

random.setSeed(1000);

// 使用给定的随机源（和默认的参数集合）初始化确定密钥大小的密钥对生成器

keygen.initialize(512, random);// keygen.initialize(512);

// 生成密钥组

KeyPair keys = keygen.generateKeyPair();

// 得到公匙

PublicKey pubkey = keys.getPublic();

// 得到私匙

PrivateKey prikey = keys.getPrivate();

// 将公匙私匙写入到文件当中

doObjToFile("mykeys.bat", **new** Object[] { prikey, pubkey });

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

}

}

/\*\*

\* 利用私匙对信息进行签名 把签名后的信息放入到指定的文件中

\*

\* **@param** info

\* 要签名的信息

\* **@param** signfile

\* 存入的文件

\*/

**public** **void** signToInfo(String info, String signfile) {

// 从文件当中读取私匙

PrivateKey myprikey = (PrivateKey) getObjFromFile("mykeys.bat", 1);

// 从文件中读取公匙

PublicKey mypubkey = (PublicKey) getObjFromFile("mykeys.bat", 2);

**try** {

// Signature 对象可用来生成和验证数字签名

Signature signet = Signature.*getInstance*("DSA");

// 初始化签署签名的私钥

signet.initSign(myprikey);

// 更新要由字节签名或验证的数据

signet.update(info.getBytes());

// 签署或验证所有更新字节的签名，返回签名

**byte**[] signed = signet.sign();

// 将数字签名,公匙,信息放入文件中

doObjToFile(signfile, **new** Object[] { signed, mypubkey, info });

} **catch** (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 读取数字签名文件 根据公匙，签名，信息验证信息的合法性

\*

\* **@return** true 验证成功 false 验证失败

\*/

**public** **boolean** validateSign(String signfile) {

// 读取公匙

PublicKey mypubkey = (PublicKey) getObjFromFile(signfile, 2);

// 读取签名

**byte**[] signed = (**byte**[]) getObjFromFile(signfile, 1);

// 读取信息

String info = (String) getObjFromFile(signfile, 3);

**try** {

// 初始一个Signature对象,并用公钥和签名进行验证

Signature signetcheck = Signature.*getInstance*("DSA");

// 初始化验证签名的公钥

signetcheck.initVerify(mypubkey);

// 使用指定的 byte 数组更新要签名或验证的数据

signetcheck.update(info.getBytes());

System.***out***.println(info);

// 验证传入的签名

**return** signetcheck.verify(signed);

} **catch** (Exception e) {

e.printStackTrace();

**return** **false**;

}

}

/\*\*

\* 将二进制转化为16进制字符串

\*

\* **@param** b

\* 二进制字节数组

\* **@return** String

\*/

**public** String byte2hex(**byte**[] b) {

String hs = "";

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.toUpperCase();

}

/\*\*

\* 十六进制字符串转化为2进制

\*

\* **@param** hex

\* **@return**

\*/

**public** **byte**[] hex2byte(String hex) {

**byte**[] ret = **new** **byte**[8];

**byte**[] tmp = hex.getBytes();

**for** (**int** i = 0; i < 8; i++) {

ret[i] = *uniteBytes*(tmp[i \* 2], tmp[i \* 2 + 1]);

}

**return** ret;

}

/\*\*

\* 将两个ASCII字符合成一个字节； 如："EF"--> 0xEF

\*

\* **@param** src0

\* byte

\* **@param** src1

\* byte

\* **@return** byte

\*/

**public** **static** **byte** uniteBytes(**byte** src0, **byte** src1) {

**byte** \_b0 = Byte.*decode*("0x" + **new** String(**new** **byte**[] { src0 }))

.byteValue();

\_b0 = (**byte**) (\_b0 << 4);

**byte** \_b1 = Byte.*decode*("0x" + **new** String(**new** **byte**[] { src1 }))

.byteValue();

**byte** ret = (**byte**) (\_b0 ^ \_b1);

**return** ret;

}

/\*\*

\* 将指定的对象写入指定的文件

\*

\* **@param** file

\* 指定写入的文件

\* **@param** objs

\* 要写入的对象

\*/

**public** **void** doObjToFile(String file, Object[] objs) {

ObjectOutputStream oos = **null**;

**try** {

FileOutputStream fos = **new** FileOutputStream(file);

oos = **new** ObjectOutputStream(fos);

**for** (**int** i = 0; i < objs.length; i++) {

oos.writeObject(objs[i]);

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**try** {

oos.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 返回在文件中指定位置的对象

\*

\* **@param** file

\* 指定的文件

\* **@param** i

\* 从1开始

\* **@return**

\*/

**public** Object getObjFromFile(String file, **int** i) {

ObjectInputStream ois = **null**;

Object obj = **null**;

**try** {

FileInputStream fis = **new** FileInputStream(file);

ois = **new** ObjectInputStream(fis);

**for** (**int** j = 0; j < i; j++) {

obj = ois.readObject();

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**try** {

ois.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** obj;

}

/\*\*

\* 测试

\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

CryptTest jiami = **new** CryptTest();

// 执行MD5加密"Hello world!"

System.***out***.println("Hello经过MD5:" + jiami.encryptToMD5("Hello"));

// 生成一个DES算法的密匙

SecretKey key = jiami.createSecretKey("DES");

// 用密匙加密信息"Hello world!"

String str1 = jiami.encryptToDES(key, "Hello");

System.***out***.println("使用des加密信息Hello为:" + str1);

// 使用这个密匙解密

String str2 = jiami.decryptByDES(key, str1);

System.***out***.println("解密后为：" + str2);

// 创建公匙和私匙

jiami.createPairKey();

// 对Hello world!使用私匙进行签名

jiami.signToInfo("Hello", "mysign.bat");

// 利用公匙对签名进行验证。

**if** (jiami.validateSign("mysign.bat")) {

System.***out***.println("Success!");

} **else** {

System.***out***.println("Fail!");

}

}

}

用到的重要的类  
javax.crypto.KeyGenerator  
public final SecretKey generateKey()生成一个密钥  
public static final KeyGenerator getInstance(String algorithm) 返回生成指定算法的秘密密钥的KeyGenerator对象。  
javax.crypto 接口 SecretKey  
javax.crypto.Cipher 此类为加密和解密提供密码功能。它构成了 Java Cryptographic Extension (JCE) 框架的核心   
public final void init(int opmode,Key key)  
public final byte[] doFinal(byte[] input) 按单部分操作加密或解密数据，或者结束一个多部分操作  
java.security.KeyPairGenerator  
static KeyPairGenerator getInstance(String algorithm)   
回生成指定算法的 public/private 密钥对的 KeyPairGenerator 对象。  
java.security.Signature  
使用 Signature 对象签名数据或验证签名包括以下三个阶段：  
1：初始化，使用   
初始化验证签名的公钥（请参见 initVerify），或使用   
初始化签署签名的私钥（也可以选择“安全随机数生成器”）initSign(PrivateKey)和initSign(PrivateKey, SecureRandom)）。   
2：更新  
根据初始化类型，这可更新要签名或验证的字节。请参见 update 方法。  
3：签署或验证所有更新字节的签名。请参见 sign 方法和 verify 方法。

****一、双向加密****   
****(一)、对称加密****   
采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。   
需要对加密和解密使用相同密钥的加密算法。由于其速度，对称性加密通常在消息发送方需要加密大量数据时使用。对称性加密也称为密钥加密。   
所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥是控制加密及解密过程的指令。   
  
算法是一组规则，规定如何进行加密和解密。因此对称式加密本身不是安全的。 　　   
常用的对称加密有：DES、IDEA、RC2、RC4、SKIPJACK、RC5、AES算法等   
  
对称加密一般java类中中定义成员

**Java代码**



1. //KeyGenerator 提供对称密钥生成器的功能，支持各种算法
2. **private** KeyGenerator keygen;
3. //SecretKey 负责保存对称密钥
4. **private** SecretKey deskey;
5. //Cipher负责完成加密或解密工作
6. **private** Cipher c;
7. //该字节数组负责保存加密的结果
8. **private** **byte**[] cipherByte;

在构造函数中初始化

**Java代码**



1. Security.addProvider(**new** com.sun.crypto.provider.SunJCE());
2. //实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)
3. keygen = KeyGenerator.getInstance("DES");//
4. //生成密钥
5. deskey = keygen.generateKey();
6. //生成Cipher对象,指定其支持的DES算法
7. c = Cipher.getInstance("DES");

****1. DES****算法为密码体制中的对称密码体制，又被成为美国数据加密标准，是1972年美国IBM公司研制的对称密码体制加密算法。 明文按64位进行分组, 密钥长64位，密钥事实上是56位参与DES运算（第8、16、24、32、40、48、56、64位是校验位， 使得每个密钥都有奇数个1）分组后的明文组和56位的密钥按位替代或交换的方法形成密文组的加密方法。

**Java代码**



1. **import** java.security.InvalidKeyException;
2. **import** java.security.NoSuchAlgorithmException;
3. **import** java.security.Security;
5. **import** javax.crypto.BadPaddingException;
6. **import** javax.crypto.Cipher;
7. **import** javax.crypto.IllegalBlockSizeException;
8. **import** javax.crypto.KeyGenerator;
9. **import** javax.crypto.NoSuchPaddingException;
10. **import** javax.crypto.SecretKey;
12. **public** **class** EncrypDES {
14. //KeyGenerator 提供对称密钥生成器的功能，支持各种算法
15. **private** KeyGenerator keygen;
16. //SecretKey 负责保存对称密钥
17. **private** SecretKey deskey;
18. //Cipher负责完成加密或解密工作
19. **private** Cipher c;
20. //该字节数组负责保存加密的结果
21. **private** **byte**[] cipherByte;
23. **public** EncrypDES() **throws** NoSuchAlgorithmException, NoSuchPaddingException{
24. Security.addProvider(**new** com.sun.crypto.provider.SunJCE());
25. //实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)
26. keygen = KeyGenerator.getInstance("DES");
27. //生成密钥
28. deskey = keygen.generateKey();
29. //生成Cipher对象,指定其支持的DES算法
30. c = Cipher.getInstance("DES");
31. }
33. /\*\*
34. \* 对字符串加密
35. \*
36. \* @param str
37. \* @return
38. \* @throws InvalidKeyException
39. \* @throws IllegalBlockSizeException
40. \* @throws BadPaddingException
41. \*/
42. **public** **byte**[] Encrytor(String str) **throws** InvalidKeyException,
43. IllegalBlockSizeException, BadPaddingException {
44. // 根据密钥，对Cipher对象进行初始化，ENCRYPT\_MODE表示加密模式
45. c.init(Cipher.ENCRYPT\_MODE, deskey);
46. **byte**[] src = str.getBytes();
47. // 加密，结果保存进cipherByte
48. cipherByte = c.doFinal(src);
49. **return** cipherByte;
50. }
52. /\*\*
53. \* 对字符串解密
54. \*
55. \* @param buff
56. \* @return
57. \* @throws InvalidKeyException
58. \* @throws IllegalBlockSizeException
59. \* @throws BadPaddingException
60. \*/
61. **public** **byte**[] Decryptor(**byte**[] buff) **throws** InvalidKeyException,
62. IllegalBlockSizeException, BadPaddingException {
63. // 根据密钥，对Cipher对象进行初始化，DECRYPT\_MODE表示加密模式
64. c.init(Cipher.DECRYPT\_MODE, deskey);
65. cipherByte = c.doFinal(buff);
66. **return** cipherByte;
67. }
69. /\*\*
70. \* @param args
71. \* @throws NoSuchPaddingException
72. \* @throws NoSuchAlgorithmException
73. \* @throws BadPaddingException
74. \* @throws IllegalBlockSizeException
75. \* @throws InvalidKeyException
76. \*/
77. **public** **static** **void** main(String[] args) **throws** Exception {
78. EncrypDES de1 = **new** EncrypDES();
79. String msg ="郭XX-搞笑相声全集";
80. **byte**[] encontent = de1.Encrytor(msg);
81. **byte**[] decontent = de1.Decryptor(encontent);
82. System.out.println("明文是:" + msg);
83. System.out.println("加密后:" + **new** String(encontent));
84. System.out.println("解密后:" + **new** String(decontent));
85. }
87. }

****AES****密码学中的高级加密标准（Advanced Encryption Standard，AES），又称  高级加密标准   
Rijndael加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于2001年11月26日发布于FIPS PUB 197，并在2002年5月26日成为有效的标准。2006年，高级加密标准已然成为对称密钥加密中最流行的算法之一。 　　该算法为比利时密码学家Joan Daemen和Vincent Rijmen所设计，结合两位作者的名字，以Rijndael之命名之，投稿高级加密标准的甄选流程。（Rijdael的发音近于 "Rhinedoll"。）

**Java代码**



1. **import** java.security.InvalidKeyException;
2. **import** java.security.NoSuchAlgorithmException;
3. **import** java.security.Security;
5. **import** javax.crypto.BadPaddingException;
6. **import** javax.crypto.Cipher;
7. **import** javax.crypto.IllegalBlockSizeException;
8. **import** javax.crypto.KeyGenerator;
9. **import** javax.crypto.NoSuchPaddingException;
10. **import** javax.crypto.SecretKey;
12. **public** **class** EncrypAES {
14. //KeyGenerator 提供对称密钥生成器的功能，支持各种算法
15. **private** KeyGenerator keygen;
16. //SecretKey 负责保存对称密钥
17. **private** SecretKey deskey;
18. //Cipher负责完成加密或解密工作
19. **private** Cipher c;
20. //该字节数组负责保存加密的结果
21. **private** **byte**[] cipherByte;
23. **public** EncrypAES() **throws** NoSuchAlgorithmException, NoSuchPaddingException{
24. Security.addProvider(**new** com.sun.crypto.provider.SunJCE());
25. //实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)
26. keygen = KeyGenerator.getInstance("AES");
27. //生成密钥
28. deskey = keygen.generateKey();
29. //生成Cipher对象,指定其支持的DES算法
30. c = Cipher.getInstance("AES");
31. }
33. /\*\*
34. \* 对字符串加密
35. \*
36. \* @param str
37. \* @return
38. \* @throws InvalidKeyException
39. \* @throws IllegalBlockSizeException
40. \* @throws BadPaddingException
41. \*/
42. **public** **byte**[] Encrytor(String str) **throws** InvalidKeyException,
43. IllegalBlockSizeException, BadPaddingException {
44. // 根据密钥，对Cipher对象进行初始化，ENCRYPT\_MODE表示加密模式
45. c.init(Cipher.ENCRYPT\_MODE, deskey);
46. **byte**[] src = str.getBytes();
47. // 加密，结果保存进cipherByte
48. cipherByte = c.doFinal(src);
49. **return** cipherByte;
50. }
52. /\*\*
53. \* 对字符串解密
54. \*
55. \* @param buff
56. \* @return
57. \* @throws InvalidKeyException
58. \* @throws IllegalBlockSizeException
59. \* @throws BadPaddingException
60. \*/
61. **public** **byte**[] Decryptor(**byte**[] buff) **throws** InvalidKeyException,
62. IllegalBlockSizeException, BadPaddingException {
63. // 根据密钥，对Cipher对象进行初始化，DECRYPT\_MODE表示加密模式
64. c.init(Cipher.DECRYPT\_MODE, deskey);
65. cipherByte = c.doFinal(buff);
66. **return** cipherByte;
67. }
69. /\*\*
70. \* @param args
71. \* @throws NoSuchPaddingException
72. \* @throws NoSuchAlgorithmException
73. \* @throws BadPaddingException
74. \* @throws IllegalBlockSizeException
75. \* @throws InvalidKeyException
76. \*/
77. **public** **static** **void** main(String[] args) **throws** Exception {
78. EncrypAES de1 = **new** EncrypAES();
79. String msg ="郭XX-搞笑相声全集";
80. **byte**[] encontent = de1.Encrytor(msg);
81. **byte**[] decontent = de1.Decryptor(encontent);
82. System.out.println("明文是:" + msg);
83. System.out.println("加密后:" + **new** String(encontent));
84. System.out.println("解密后:" + **new** String(decontent));
85. }
87. }

****(二)、非对称加密****   
1976年，美国学者Dime和Henman为解决信息公开传送和密钥管理问题，提出一种新的密钥交换协议，允许在不安全的媒体上的通讯双方交换信息，安全地达成一致的密钥，这就是“公开密钥系统”。相对于“对称加密算法”这种方法也叫做“非对称加密算法”。 与对称加密算法不同，非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥   
（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。   
  
****1. RSA**** 公钥加密算法是1977年由Ron Rivest、Adi Shamirh和LenAdleman在（美国麻省理工学院）开发的。RSA取名来自开发他们三者的名字。RSA是目前最有影响力的公钥加密算法，它能够抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

**Java代码**



1. **import** java.security.InvalidKeyException;
2. **import** java.security.KeyPair;
3. **import** java.security.KeyPairGenerator;
4. **import** java.security.NoSuchAlgorithmException;
5. **import** java.security.interfaces.RSAPrivateKey;
6. **import** java.security.interfaces.RSAPublicKey;
8. **import** javax.crypto.BadPaddingException;
9. **import** javax.crypto.Cipher;
10. **import** javax.crypto.IllegalBlockSizeException;
11. **import** javax.crypto.NoSuchPaddingException;
13. **public** **class** EncrypRSA {
15. /\*\*
16. \* 加密
17. \* @param publicKey
18. \* @param srcBytes
19. \* @return
20. \* @throws NoSuchAlgorithmException
21. \* @throws NoSuchPaddingException
22. \* @throws InvalidKeyException
23. \* @throws IllegalBlockSizeException
24. \* @throws BadPaddingException
25. \*/
26. **protected** **byte**[] encrypt(RSAPublicKey publicKey,**byte**[] srcBytes) **throws** NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException{
27. **if**(publicKey!=**null**){
28. //Cipher负责完成加密或解密工作，基于RSA
29. Cipher cipher = Cipher.getInstance("RSA");
30. //根据公钥，对Cipher对象进行初始化
31. cipher.init(Cipher.ENCRYPT\_MODE, publicKey);
32. **byte**[] resultBytes = cipher.doFinal(srcBytes);
33. **return** resultBytes;
34. }
35. **return** **null**;
36. }
38. /\*\*
39. \* 解密
40. \* @param privateKey
41. \* @param srcBytes
42. \* @return
43. \* @throws NoSuchAlgorithmException
44. \* @throws NoSuchPaddingException
45. \* @throws InvalidKeyException
46. \* @throws IllegalBlockSizeException
47. \* @throws BadPaddingException
48. \*/
49. **protected** **byte**[] decrypt(RSAPrivateKey privateKey,**byte**[] srcBytes) **throws** NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException{
50. **if**(privateKey!=**null**){
51. //Cipher负责完成加密或解密工作，基于RSA
52. Cipher cipher = Cipher.getInstance("RSA");
53. //根据公钥，对Cipher对象进行初始化
54. cipher.init(Cipher.DECRYPT\_MODE, privateKey);
55. **byte**[] resultBytes = cipher.doFinal(srcBytes);
56. **return** resultBytes;
57. }
58. **return** **null**;
59. }
61. /\*\*
62. \* @param args
63. \* @throws NoSuchAlgorithmException
64. \* @throws BadPaddingException
65. \* @throws IllegalBlockSizeException
66. \* @throws NoSuchPaddingException
67. \* @throws InvalidKeyException
68. \*/
69. **public** **static** **void** main(String[] args) **throws** NoSuchAlgorithmException, InvalidKeyException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {
70. EncrypRSA rsa = **new** EncrypRSA();
71. String msg = "郭XX-精品相声";
72. //KeyPairGenerator类用于生成公钥和私钥对，基于RSA算法生成对象
73. KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
74. //初始化密钥对生成器，密钥大小为1024位
75. keyPairGen.initialize(1024);
76. //生成一个密钥对，保存在keyPair中
77. KeyPair keyPair = keyPairGen.generateKeyPair();
78. //得到私钥
79. RSAPrivateKey privateKey = (RSAPrivateKey)keyPair.getPrivate();
80. //得到公钥
81. RSAPublicKey publicKey = (RSAPublicKey)keyPair.getPublic();
83. //用公钥加密
84. **byte**[] srcBytes = msg.getBytes();
85. **byte**[] resultBytes = rsa.encrypt(publicKey, srcBytes);
87. //用私钥解密
88. **byte**[] decBytes = rsa.decrypt(privateKey, resultBytes);
90. System.out.println("明文是:" + msg);
91. System.out.println("加密后是:" + **new** String(resultBytes));
92. System.out.println("解密后是:" + **new** String(decBytes));
93. }
95. }

****2. DSA****  
Digital Signature Algorithm (DSA)是Schnorr和ElGamal签名算法的变种，被美国NIST作为DSS(DigitalSignature Standard)。(感觉有点复杂，没有附代码)   
详见[http://63938525.iteye.com/blog/1051565](http://63938525.iteye.com/blog/1051565" \t "/home/ryan/Documents\\x/_blank)