# Java加密解密全解

原创 2016年08月05日 11:31:18

1 sha加密：

安全哈希算法（Secure Hash Algorithm）主要适用于数字签名标准（Digital Signature Standard DSS）里面定义的数字签名算法（Digital Signature Algorithm DSA）。对于长度小于2^64位的消息，SHA1会产生一个160位的消息摘要。该算法经过加密专家多年来的发展和改进已日益完善，并被广泛使用。该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。散列函数值可以说是对明文的一种“[指纹](http://baike.baidu.com/view/5628.htm)”或是“摘要”所以对散列值的[数字签名](http://baike.baidu.com/view/7626.htm)就可以视为对此明文的数字签名。

[安全散列算法](http://baike.baidu.com/view/1984487.htm)SHA

（Secure Hash Algorithm，SHA)

是美国国家标准技术研究所发布的国家标准FIPS PUB 180，最新的标准已经于2008年更新到FIPS PUB 180-3。其中规定了SHA-1，SHA-224，SHA-256，SHA-384，和SHA-512这几种[单向散列算法](http://baike.baidu.com/view/4481025.htm)。SHA-1，SHA-224和SHA-256适用于长度不超过2^64二进制位的消息。SHA-384和SHA-512适用于长度不超过2^128二进制位的消息。

### 散列算法

散列是信息的提炼，通常其长度要比信息小得多，且为一个固定长度。加密性强的散列一定是不可逆的，这就意味着通过散列结果，无法推出任何部分的原始信息。任何输入信息的变化，哪怕仅一位，都将导致散列结果的明显变化，这称之为雪崩效应。散列还应该是防冲突的，即找不出具有相同散列结果的两条信息。具有这些特性的散列结果就可以用于验证信息是否被修改。

[单向散列函数](http://baike.baidu.com/view/6321298.htm)一般用于产生[消息摘要](http://baike.baidu.com/view/2396437.htm)，密钥加密等，常见的有：

l MD5（Message Digest Algorithm 5）：是RSA数据安全公司开发的一种[单向散列算法](http://baike.baidu.com/view/4481025.htm)。

l SHA（Secure Hash Algorithm）：可以对任意长度的数据运算生成一个160位的数值；

SHA-1

在1993年，安全散列算法（SHA）由美国国家标准和技术协会（NIST)提出，并作为联邦信息处理标准（FIPS PUB 180）公布；1995年又发布了一个修订版FIPS PUB 180-1，通常称之为SHA-1。SHA-1是基于MD4算法的，并且它的设计在很大程度上是模仿MD4的。现在已成为公认的最安全的散列算法之一，并被广泛使用。

### 原理

SHA-1是一种[数据加密算法](http://baike.baidu.com/view/878529.htm)，该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。

[单向散列函数](http://baike.baidu.com/view/6321298.htm)的安全性在于其产生散列值的操作过程具有较强的单向性。如果在输入序列中嵌入密码，那么任何人在不知道密码的情况下都不能产生正确的散列值，从而保证了其安全性。SHA将输入流按照每块512位（64个字节）进行分块，并产生20个字节的被称为信息认证代码或信息摘要的输出。

该算法输入[报文](http://baike.baidu.com/view/175122.htm)的长度不限，产生的输出是一个160位的[报文摘要](http://baike.baidu.com/view/4971320.htm)。输入是按512 位的分组进行处理的。SHA-1是不可逆的、防冲突，并具有良好的雪崩效应。

通过散列算法可实现[数字签名](http://baike.baidu.com/view/7626.htm)实现，数字签名的原理是将要传送的明文通过一种函数运算（Hash）转换成报文摘要（不同的明文对应不同的报文摘要），报文摘要加密后与明文一起传送给接受方，接受方将接受的明文产生新的报文摘要与发送方的发来报文摘要解密比较，比较结果一致表示明文未被改动，如果不一致表示明文已被篡改。

MAC （信息认证代码）就是一个散列结果，其中部分输入信息是密码，只有知道这个密码的参与者才能再次计算和验证MAC码的合法性。

### SHA-1与MD5的比较

因为二者均由MD4导出，SHA-1和MD5彼此很相似。相应的，他们的强度和其他特性也是相似，但还有以下几点不同：

l 对强行攻击的安全性：最显著和最重要的区别是SHA-1摘要比MD5摘要长32 位。使用强行技术，产生任何一个[报文](http://baike.baidu.com/view/175122.htm)使其摘要等于给定报摘要的难度对MD5是2^128数量级的操作，而对SHA-1则是2^160数量级的操作。这样，SHA-1对强行攻击有更大的强度。

l 对密码分析的安全性：由于MD5的设计，易受密码分析的攻击，SHA-1显得不易受这样的攻击。

l 速度：在相同的硬件上，SHA-1的运行速度比MD5慢。

JAVA 已经实现了 SHA-256 和 SHA-512 两种 Hash 算法

利用 java.security.MessageDigest 调用已经集成的 Hash 算法

创建 Encrypt 对象，并调用 SHA256 或者 SHA512 并传入要加密的文本信息，分别得到 SHA-256 或 SHA-512 两种被加密的 hash 串。

若要改为 MD5 算法，修改传入参数 strType 为 "MD5" 即可得到 MD5 加密功能。

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. /\*\*
2. \* @file Encrypt.java
3. \* @date 2016年8月5日
4. \* @version 3.4.1
5. \*
6. \* Copyright (c) 2013 Sihua Tech, Inc. All Rights Reserved.
7. \*/
8. package encrypt;
10. /\*\*
11. \*
12. \*
13. \* @author chengjian.he
14. \* @version  3.4, 2016年8月5日 上午10:05:37
15. \* @since
16. \*/
18. import java.security.MessageDigest;
19. import java.security.NoSuchAlgorithmException;
21. public class Encrypt
22. {
24. /\*\*
25. \* 传入文本内容，返回 SHA-256 串
26. \*
27. \* @param strText
28. \* @return
29. \*/
30. public String SHA256(final String strText)
31. {
32. return SHA(strText, "SHA-256");
33. }
35. /\*\*
36. \* 传入文本内容，返回 SHA-512 串
37. \*
38. \* @param strText
39. \* @return
40. \*/
41. public String SHA512(final String strText)
42. {
43. return SHA(strText, "SHA-512");
44. }
46. /\*\*
47. \* 字符串 SHA 加密
48. \*
49. \* @param strSourceText
50. \* @return
51. \*/
52. private String SHA(final String strText, final String strType)
53. {
54. // 返回值
55. String strResult = null;
57. // 是否是有效字符串
58. if (strText != null && strText.length() > 0)
59. {
60. try
61. {
62. // SHA 加密开始
63. // 创建加密对象 并傳入加密類型
64. MessageDigest messageDigest = MessageDigest.getInstance(strType);
65. // 传入要加密的字符串
66. messageDigest.update(strText.getBytes());
67. // 得到 byte 類型结果
68. byte byteBuffer[] = messageDigest.digest();
70. // 將 byte 轉換爲 string
71. StringBuffer strHexString = new StringBuffer();
72. // 遍歷 byte buffer
73. for (int i = 0; i < byteBuffer.length; i++)
74. {
75. String hex = Integer.toHexString(0xff & byteBuffer[i]);
76. if (hex.length() == 1)
77. {
78. strHexString.append('0');
79. }
80. strHexString.append(hex);
81. }
82. // 得到返回結果
83. strResult = strHexString.toString();
84. }
85. catch (NoSuchAlgorithmException e)
86. {
87. e.printStackTrace();
88. }
89. }
91. return strResult;
92. }
94. public static void main(String args[]){
95. Encrypt ey = new Encrypt();
96. System.out.println(ey.SHA("ILoveYou", "MD5"));//62accaf23ac9a73c0b28765b7dfaf75a
97. }
98. }

2 Base64

Base64是网络上最常见的用于传输8Bit[字节代码](http://baike.baidu.com/view/185293.htm)的编码方式之一，大家可以查看RFC2045～RFC2049，上面有MIME的详细规范。Base64编码可用于在[HTTP](http://baike.baidu.com/view/9472.htm)环境下传递较长的标识信息。例如，在Java Persistence系统Hibernate中，就采用了Base64来将一个较长的唯一[标识符](http://baike.baidu.com/view/390932.htm)（一般为128-bit的UUID）编码为一个字符串，用作HTTP[表单](http://baike.baidu.com/view/296684.htm)和HTTP GET URL中的参数。在其他应用程序中，也常常需要把二进制[数据编码](http://baike.baidu.com/view/575608.htm)为适合放在URL（包括隐藏[表单域](http://baike.baidu.com/view/8676899.htm)）中的形式。此时，采用Base64编码具有不可读性，即所编码的数据不会被人用肉眼所直接看到。

**然而，标准的Base64并不适合直接放在URL里传输，因为URL编码器会把标准Base64中的“/”和“+”字符变为形如“%XX”的形式，而这些“%”号在存入数据库时还需要再进行转换，因为ANSI SQL中已将“%”号用作通配符。**

**为解决此问题，可采用一种用于URL的改进Base64编码，它不仅在末尾填充'='号，并将标准Base64中的“+”和“/”分别改成了“-”和“\_”，这样就免去了在URL编解码和数据库存储时所要作的转换，避免了编码信息长度在此过程中的增加，并统一了数据库、表单等处对象**[**标识符**](http://baike.baidu.com/view/390932.htm)**的格式。**

**另有一种用于正则表达式的改进Base64变种，它将“+”和“/”改成了“!”和“-”，因为“+”,“\*”以及前面在IRCu中用到的“[”和“]”在正则表达式中都可能具有特殊含义。**

**此外还有一些变种，它们将“+/”改为“\_-”或“.\_”（用作编程语言中的标识符名称）或“.-”（用于XML中的Nmtoken）甚至“\_:”（用于XML中的Name）。**

**其他应用**

**Mozilla Thunderbird和Evolution用Base64来保密电子邮件密码**

**Base64 也会经常用作一个简单的“加密”来保护某些数据，而真正的加密通常都比较繁琐。**

**垃圾讯息传播者用Base64来避过反垃圾邮件工具，因为那些工具通常都不会翻译Base64的讯息。**

**在LDIF档案，Base64用作编码字串。**

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. /\*\*
2. \* @file Base64.java
3. \* @date 2016年8月5日
4. \* @version 3.4.1
5. \*
6. \* Copyright (c) 2013 Sihua Tech, Inc. All Rights Reserved.
7. \*/
8. package encrypt;
10. import java.io.UnsupportedEncodingException;
12. import Decoder.BASE64Decoder;
13. import Decoder.BASE64Encoder;
15. /\*\*
16. \*
17. \*
18. \* @author chengjian.he
19. \* @version  3.4, 2016年8月5日 上午10:32:23
20. \* @since   Yeexun 3.4
21. \*/
23. public class Base64 {
24. // 加密
25. public String getBase64(String str) {
26. byte[] b = null;
27. String s = null;
28. try {
29. b = str.getBytes("utf-8");
30. } catch (UnsupportedEncodingException e) {
31. e.printStackTrace();
32. }
33. if (b != null) {
34. s = new BASE64Encoder().encode(b);
35. }
36. return s;
37. }
39. // 解密
40. public String getFromBase64(String s) {
41. byte[] b = null;
42. String result = null;
43. if (s != null) {
44. BASE64Decoder decoder = new BASE64Decoder();
45. try {
46. b = decoder.decodeBuffer(s);
47. result = new String(b, "utf-8");
48. } catch (Exception e) {
49. e.printStackTrace();
50. }
51. }
52. return result;
53. }
55. public static void main(String args[]){
56. Base64 b6 = new Base64();
57. System.out.println(b6.getBase64("ILoveYou"));
58. System.out.println(b6.getFromBase64(b6.getBase64("ILoveYou")));
59. }
60. }

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. SUxvdmVZb3U=
2. ILoveYou

3 Base64Encoder

一直以来Base64的加密解密都是使用sun.misc包下的BASE64Encoder及BASE64Decoder的sun.misc.BASE64Encoder/BASE64Decoder类。这人个类是sun公司的内部方法，并没有在java api中公开过，不属于JDK标准库范畴，但在JDK中包含了该类，可以直接使用。

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. /\*\*
2. \* @file Base64Encoder.java
3. \* @date 2016年8月5日
4. \* @version 3.4.1
5. \*
6. \* Copyright (c) 2013 Sihua Tech, Inc. All Rights Reserved.
7. \*/
8. package encrypt;
10. /\*\*
11. \*
12. \*
13. \* @author chengjian.he
14. \* @version  3.4, 2016年8月5日 上午10:44:22
15. \* @since   Yeexun 3.4
16. \*/
17. public class Base64Encoder {
18. public static String getBASE64(String s) {
19. if (s == null)
20. return null;
21. return (new sun.misc.BASE64Encoder()).encode(s.getBytes());
22. }
23. // 将 BASE64 编码的字符串 s 进行解码   解密
24. public static String getFromBASE64(String s) {
25. if (s == null)
26. return null;
27. sun.misc.BASE64Decoder decoder = new sun.misc.BASE64Decoder();
28. try {
29. byte[] b = decoder.decodeBuffer(s);
30. return new String(b);
31. } catch (Exception e) {
32. return null;
33. }
34. }
35. public static String mTOa(Object ming){
36. return Base64Encoder.getBASE64(Base64Encoder.getBASE64(Base64Encoder.getBASE64((String)ming)));
37. }
38. public static String aTOm(String an){
39. return Base64Encoder.getFromBASE64(Base64Encoder.getFromBASE64(Base64Encoder.getFromBASE64(an)));
40. }
41. public static void main(String[] args) {
42. String a = mTOa("100000.89".toString());
43. System.out.println(a);//加密
44. System.out.println(aTOm(a));//解密
45. }
46. }

4 RSA

RSA[公钥](http://baike.baidu.com/view/355291.htm)[加密算法](http://baike.baidu.com/view/155969.htm)是1977年由[罗纳德·李维斯特](http://baike.baidu.com/view/2838988.htm)（Ron Rivest）、[阿迪·萨莫尔](http://baike.baidu.com/view/10475292.htm)（Adi Shamir）和[伦纳德·阿德曼](http://baike.baidu.com/view/11580236.htm)（Leonard Adleman）一起提出的。1987年首次公布，当时他们三人都在麻省理工学院工作。RSA就是他们三人姓氏开头字母拼在一起组成的。

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绝大多数密码攻击，已被ISO推荐为公钥[数据加密标准](http://baike.baidu.com/view/1519129.htm)。

今天只有短的RSA钥匙才可能被强力方式解破。到2008年为止，世界上还没有任何可靠的攻击RSA算法的方式。只要其钥匙的长度足够长，用RSA加密的信息实际上是不能被解破的。但在[分布式计算](http://baike.baidu.com/view/30655.htm)和[量子计算机](http://baike.baidu.com/view/18645.htm)理论日趋成熟的今天，RSA加密安全性受到了挑战。

RSA算法基于一个十分简单的数论事实：将两个大质数相乘十分容易，但是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。

## 缺点

编辑

1）产生密钥很麻烦，受到素数产生技术的限制，因而难以做到一次一密。

2）安全性，RSA的安全性依赖于大数的因子分解，但并没有从理论上证明破译RSA的难度与大数分解难度等价，而且密码学界多数人士倾向于因子分解不是NP问题。现今，人们已能分解140多个十进制位的大素数，这就要求使用更长的密钥，速度更慢；另外，人们正在积极寻找攻击RSA的方法，如选择密文攻击，一般攻击者是将某一信息作一下伪装（Blind），让拥有私钥的实体签署。然后，经过计算就可得到它所想要的信息。实际上，攻击利用的都是同一个弱点，即存在这样一个事实：乘幂保留了输入的乘法结构：

（XM)d = Xd \*Md mod n

前面已经提到，这个固有的问题来自于公钥密码系统的最有用的特征--每个人都能使用公钥。但从算法上无法解决这一问题，主要措施有两条：一条是采用好的公钥协议，保证工作过程中实体不对其他实体任意产生的信息解密，不对自己一无所知的信息签名；另一条是决不对陌生人送来的随机文档签名，签名时首先使用One-Way Hash Function对文档作HASH处理，或同时使用不同的签名算法。除了利用公共模数，人们还尝试一些利用解密指数或φ（n）等等攻击.

3）速度太慢，由于RSA 的分组长度太大，为保证安全性，n 至少也要 600 bits以上，使运算代价很高，尤其是速度较慢，较对称密码算法慢几个数量级；且随着大数分解技术的发展，这个长度还在增加，不利于数据格式的标准化。SET(Secure Electronic Transaction）协议中要求CA采用2048比特长的密钥，其他实体使用1024比特的密钥。为了速度问题，人们广泛使用单，[公钥](http://baike.baidu.com/view/355291.htm)密码结合使用的方法，优缺点互补：单钥密码加密速度快，人们用它来加密较长的文件，然后用RSA来给文件密钥加密，极好的解决了单钥密码的密钥分发问题。

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. /\*\*
2. \* @file RSATool.java
3. \* @date 2016年8月5日
4. \* @version 3.4.1
5. \*
6. \* Copyright (c) 2013 Sihua Tech, Inc. All Rights Reserved.
7. \*/
8. package encrypt;
10. /\*\*
11. \*
12. \*
13. \* @author chengjian.he
14. \* @version  3.4, 2016年8月5日 上午10:51:35
15. \* @since   Yeexun 3.4
16. \*/
17. import java.io.FileInputStream;
18. import java.io.FileNotFoundException;
19. import java.io.FileOutputStream;
20. import java.io.IOException;
21. import java.io.ObjectInputStream;
22. import java.io.ObjectOutputStream;
23. import java.security.Key;
24. import java.security.KeyPair;
25. import java.security.KeyPairGenerator;
26. import java.security.NoSuchAlgorithmException;
27. import java.security.interfaces.RSAPrivateKey;
28. import java.security.interfaces.RSAPublicKey;
30. import javax.crypto.Cipher;
31. import javax.crypto.NoSuchPaddingException;
33. public class RSATool {
35. public static void makekeyfile(String pubkeyfile, String privatekeyfile)
36. throws NoSuchAlgorithmException, FileNotFoundException, IOException {
37. // KeyPairGenerator类用于生成公钥和私钥对，基于RSA算法生成对象
38. KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
39. // 初始化密钥对生成器，密钥大小为1024位
40. keyPairGen.initialize(1024);
41. // 生成一个密钥对，保存在keyPair中
42. KeyPair keyPair = keyPairGen.generateKeyPair();
44. // 得到私钥
45. RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
47. // 得到公钥
48. RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
50. // 生成私钥
51. ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(
52. privatekeyfile));
53. oos.writeObject(privateKey);
54. oos.flush();
55. oos.close();
57. oos = new ObjectOutputStream(new FileOutputStream(pubkeyfile));
58. oos.writeObject(publicKey);
59. oos.flush();
60. oos.close();
62. System.out.println("make file ok!");
63. }
65. /\*\*
66. \*
67. \* @param k
68. \* @param data
69. \* @param encrypt
70. \*            1 加密 0解密
71. \* @return
72. \* @throws NoSuchPaddingException
73. \* @throws Exception
74. \*/
75. public static byte[] handleData(Key k, byte[] data, int encrypt)
76. throws Exception {
78. if (k != null) {
80. Cipher cipher = Cipher.getInstance("RSA");
82. if (encrypt == 1) {
83. cipher.init(Cipher.ENCRYPT\_MODE, k);
84. byte[] resultBytes = cipher.doFinal(data);
85. return resultBytes;
86. } else if (encrypt == 0) {
87. cipher.init(Cipher.DECRYPT\_MODE, k);
88. byte[] resultBytes = cipher.doFinal(data);
89. return resultBytes;
90. } else {
91. System.out.println("参数必须为: 1 加密 0解密");
92. }
93. }
94. return null;
95. }
97. public static void main(String[] args) throws Exception {
99. String pubfile = "d:/temp/pub.key";
100. String prifile = "d:/temp/pri.key";
102. makekeyfile(pubfile, prifile);
104. ObjectInputStream ois = new ObjectInputStream(new FileInputStream(pubfile));
105. RSAPublicKey pubkey = (RSAPublicKey) ois.readObject();
106. ois.close();
108. ois = new ObjectInputStream(new FileInputStream(prifile));
109. RSAPrivateKey prikey = (RSAPrivateKey) ois.readObject();
110. ois.close();
112. // 使用公钥加密
113. String msg = "~O(∩\_∩)O哈哈~";
114. String enc = "UTF-8";
116. // 使用公钥加密私钥解密
117. System.out.println("原文: " + msg);
118. byte[] result = handleData(pubkey, msg.getBytes(enc), 1);
119. System.out.println("加密: " + new String(result, enc));
120. byte[] deresult = handleData(prikey, result, 0);
121. System.out.println("解密: " + new String(deresult, enc));
123. msg = "嚯嚯";
124. // 使用私钥加密公钥解密
125. System.out.println("原文: " + msg);
126. byte[] result2 = handleData(prikey, msg.getBytes(enc), 1);
127. System.out.println("加密: " + new String(result2, enc));
128. byte[] deresult2 = handleData(pubkey, result2, 0);
129. System.out.println("解密: " + new String(deresult2, enc));
131. }
132. }

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. make file ok!
2. 原文: ~O(∩\_∩)O哈哈~
3. 加密: �A N�ډB�����ym��r�C��ʇ�������U

5 AES对称加密

加密技术可以分为对称与非对称两种.

对称加密,解密,即加密与解密用的是同一把秘钥,常用的对称加密技术有DES,AES等

而非对称技术,加密与解密用的是不同的秘钥,常用的非对称加密技术有RSA等

为什么要有非对称加密,解密技术呢

假设这样一种场景A要发送一段消息给B,但是又不想以明文发送,所以就需要对消息进行加密.如果采用对称加密技术,那么加密与解密用的是同一把秘钥.除非B事先就知道A的秘钥,并且保存好.这样才可以解密A发来的消息.

由于对称技术只有一把秘钥,所以秘钥的管理是一个很麻烦的问题.而非对称技术的诞生就解决了这个问题.非对称加密与解密使用的是不同的秘钥,并且秘钥对是一一对应的,即用A的私钥加密的密文只有用A的公钥才能解密.

这样的话,每个人都有两把秘钥,私钥和公钥,私钥是只有自己才知道的,不能告诉别人,而公钥是公开的,大家都可以知道.这样,当A想要发送消息给B的时候,只需要用B的公钥对消息进行加密就可以了,由于B的私钥只有B才拥有,所以A用B的公钥加密的消息只有B才能解开.而B想更换自己的秘要时也很方便,只须把公钥告诉大家就可以了.

那么,既然非对称加密如此之好,对称加密就没有存在的必要了啊,其实不然,由于非对称加密算法的开销很大,所以如果直接以非对称技术来加密发送的消息效率会很差.那么怎么办呢?解决的办法也很简单,就是把对称加密技术与非对称加密技术结合起来使用.

还是这个例子:A要发送一个消息给B.

一,A先生成一个对称秘钥,这个秘钥可以是随机生成的,

二,A用B的公钥加密第一步生成的这个对称秘钥

三,A把加密过的对称秘钥发给B

四,A用第一步生成的这个对称秘钥加密实际要发的消息

五,A把用对称秘钥加密的消息发给B

对于B

他先收到A发来的对称秘钥,这个秘钥是用B的公钥加密过的,所以B需要用自己的私钥来解密这个秘钥

然后B又收到A发来的密文,这时候用刚才解密出来的秘钥来解密密文

这样子的整个过程既保证了安全,又保证了效率.

接下来是Java实现:

我这个Java实现使用的是AES的对称加密和RSA的非对称加密(DES的对称加密实现方法和AES的是一样的,但是由于DES算法本身有缺陷,容易被破解,所以现在多用其升级版AES对称加密)

**[java]** [view plain](http://blog.csdn.net/sinat_29581293/article/details/52126519) [copy](http://blog.csdn.net/sinat_29581293/article/details/52126519)

1. /\*\*
2. \* @file AES.java
3. \* @date 2016年8月5日
4. \* @version 3.4.1
5. \*
6. \* Copyright (c) 2013 Sihua Tech, Inc. All Rights Reserved.
7. \*/
8. package encrypt;
10. import java.io.File;
11. import java.io.FileInputStream;
12. import java.io.FileOutputStream;
13. /\*\*
14. \*
15. \*
16. \* @author chengjian.he
17. \* @version  3.4, 2016年8月5日 上午11:35:13
18. \* @since   Yeexun 3.4
19. \*/
20. import java.io.IOException;
21. import java.io.InputStream;
22. import java.io.ObjectInputStream;
23. import java.io.OutputStream;
24. import java.security.InvalidKeyException;
25. import java.security.Key;
26. import java.security.NoSuchAlgorithmException;
27. import java.security.SecureRandom;
28. import java.security.interfaces.RSAPrivateKey;
29. import java.security.interfaces.RSAPublicKey;
31. import javax.crypto.BadPaddingException;
32. import javax.crypto.Cipher;
33. import javax.crypto.IllegalBlockSizeException;
34. import javax.crypto.KeyGenerator;
35. import javax.crypto.NoSuchPaddingException;
36. import javax.crypto.ShortBufferException;
38. public class AES {
40. private Key key;
42. /\*\*
43. \* 生成AES对称秘钥
44. \* @throws NoSuchAlgorithmException
45. \*/
46. public void generateKey() throws NoSuchAlgorithmException {
47. KeyGenerator keygen = KeyGenerator.getInstance("AES");
48. SecureRandom random = new SecureRandom();
49. keygen.init(random);
50. this.key = keygen.generateKey();
51. }

54. /\*\*
55. \* 加密
56. \* @param in
57. \* @param out
58. \* @throws InvalidKeyException
59. \* @throws ShortBufferException
60. \* @throws IllegalBlockSizeException
61. \* @throws BadPaddingException
62. \* @throws NoSuchAlgorithmException
63. \* @throws NoSuchPaddingException
64. \* @throws IOException
65. \*/
66. public void encrypt(InputStream in, OutputStream out) throws InvalidKeyException, ShortBufferException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException, NoSuchPaddingException, IOException {
67. this.crypt(in, out, Cipher.ENCRYPT\_MODE);
68. }
70. /\*\*
71. \* 解密
72. \* @param in
73. \* @param out
74. \* @throws InvalidKeyException
75. \* @throws ShortBufferException
76. \* @throws IllegalBlockSizeException
77. \* @throws BadPaddingException
78. \* @throws NoSuchAlgorithmException
79. \* @throws NoSuchPaddingException
80. \* @throws IOException
81. \*/
82. public void decrypt(InputStream in, OutputStream out) throws InvalidKeyException, ShortBufferException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException, NoSuchPaddingException, IOException {
83. this.crypt(in, out, Cipher.DECRYPT\_MODE);
84. }
86. /\*\*
87. \* 实际的加密解密过程
88. \* @param in
89. \* @param out
90. \* @param mode
91. \* @throws IOException
92. \* @throws ShortBufferException
93. \* @throws IllegalBlockSizeException
94. \* @throws BadPaddingException
95. \* @throws NoSuchAlgorithmException
96. \* @throws NoSuchPaddingException
97. \* @throws InvalidKeyException
98. \*/
99. public void crypt(InputStream in, OutputStream out, int mode) throws IOException, ShortBufferException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException {
100. Cipher cipher = Cipher.getInstance("AES");
101. cipher.init(mode, this.key);
103. int blockSize = cipher.getBlockSize();
104. int outputSize = cipher.getOutputSize(blockSize);
105. byte[] inBytes = new byte[blockSize];
106. byte[] outBytes = new byte[outputSize];
108. int inLength = 0;
109. boolean more = true;
110. while (more) {
111. inLength = in.read(inBytes);
112. if (inLength == blockSize) {
113. int outLength = cipher.update(inBytes, 0, blockSize, outBytes);
114. out.write(outBytes, 0, outLength);
115. } else {
116. more = false;
117. }
118. }
119. if (inLength > 0)
120. outBytes = cipher.doFinal(inBytes, 0, inLength);
121. else
122. outBytes = cipher.doFinal();
123. out.write(outBytes);
124. out.flush();
125. }
127. public void setKey(Key key) {
128. this.key = key;
129. }
131. public Key getKey() {
132. return key;
133. }
135. public static void main(String[] args) throws Exception {
136. AES aes = new AES();
137. aes.generateKey();
138. File file = new File("D:/aa.jpg");
139. FileInputStream in = new FileInputStream(file);
140. File file1 = new File("D:/temp/pub.key");
141. FileOutputStream out = new FileOutputStream(file1);
142. aes.encrypt(in, out);
143. aes.decrypt(in, out);
144. }
146. }



加密，是以某种特殊的算法改变原有的信息数据，使得未授权的用户即使获得了已加密的信息，但因不知解密的方法，仍然无法了解信息的内容。大体上分为双向加密和单向加密，而双向加密又分为对称加密和非对称加密(有些资料将加密直接分为对称加密和非对称加密)。   
  
双向加密大体意思就是明文加密后形成密文，可以通过算法还原成明文。而单向加密只是对信息进行了摘要计算，不能通过算法生成明文，单向加密从严格意思上说不能算是加密的一种，应该算是摘要算法吧。具体区分可以参考：   
(本人解释不清呢 …… )   
<http://security.group.iteye.com/group/wiki/1710-one-way-encryption-algorithm>   
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. }

2. 3DES又称Triple DES，是DES加密算法的一种模式，它使用3条56位的密钥对3DES   
数据进行三次加密。数据加密标准（DES）是美国的一种由来已久的加密标准，它使用对称密钥加密法，并于1981年被ANSI组织规范为ANSI X.3.92。DES使用56位密钥和密码块的方法，而在密码块的方法中，文本被分成64位大小的文本块然后再进行加密。比起最初的DES，3DES更为安全。 　　   
3DES（即Triple DES）是DES向AES过渡的加密算法（1999年，NIST将3-DES指定为过渡的加密标准），是DES的一个更安全的变形。它以DES为基本模块，通过组合分组方法设计出分组加密算法，其具体实现如下：   
设Ek()和Dk()代表DES算法的加密和解密过程，K代表DES算法使用的密钥，P代表明文，C代表密文，   
这样， 　　   
3DES加密过程为：C=Ek3(Dk2(Ek1(P)))   
3DES解密过程为：P=Dk1((EK2(Dk3(C)))

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 EncrypDES3 {
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 EncrypDES3() 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("DESede");
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("DESede");
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. EncrypDES3 des = new EncrypDES3();
79. String msg ="郭XX-搞笑相声全集";
80. byte[] encontent = des.Encrytor(msg);
81. byte[] decontent = des.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));
86. }
88. }

3. 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>   
  
(三)、题外话 MySQL加密解密函数   
MySQL有两个函数来支持这种类型的加密，分别叫做ENCODE()和DECODE()。   
下面是一个简单的实例：

Mysql代码  

1. mysql> INSERT INTO users (username,password) VALUES ('joe',ENCODE('guessme','abr'));
3. Query OK， 1 row affected （0.14 sec）

其中，Joe的密码是guessme，它通过密钥abracadabra被加密。要注意的是，加密完的结果是一个二进制字符串，如下所示：   
  
提示：虽然ENCODE()和DECODE()这两个函数能够满足大多数的要求，但是有的时候您希望使用强度更高的加密手段。在这种情况下，您可以使用AES\_ENCRYPT()和AES\_DECRYPT()函数，它们的工作方式是相同的，但是加密强度更高。   
  
  
单向加密与双向加密不同，一旦数据被加密就没有办法颠倒这一过程。因此密码的验证包括对用户输入内容的重新加密，并将它与保存的密文进行比对，看是否匹配。一种简单的单向加密方式是MD5校验码。MySQL的MD5（）函数会为您的数据创建一个“指纹”并将它保存起来，供验证测试使用。下面就是如何使用它的一个简单例子：

Mysql代码  

1. mysql> INSERT INTO users (username,password) VALUES ('joe',MD5('guessme'));
3. Query OK， 1 row affected （0.00 sec）

或者，您考虑一下使用ENCRYPT（）函数，它使用系统底层的crypt（）系统调用来完成加密。这个函数有两个参数：一个是要被加密的字符串，另一个是双（或者多）字符的“salt”。它然后会用salt加密字符串；这个salt然后可以被用来再次加密用户输入的内容，并将它与先前加密的字符串进行比对。下面一个例子说明了如何使用它：

Mysql代码  

1. mysql> INSERT INTO users (username,password) VALUES('joe'， ENCRYPT('guessme','ab'));
3. Query OK， 1 row affected （0.00 sec）

提示：ENCRYPT()只能用在UNIX、LINIX系统上，因为它需要用到底层的crypt()库。   
  
二、单向加密(信息摘要)   
Java一般需要获取对象MessageDigest来实现单项加密(信息摘要)。   
1. MD5 即Message-Digest Algorithm 5（信息-摘要算法 5），用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译摘要算法、哈希算法），主流编程语言普遍已有MD5实现。将数据（如汉字）运算为另一固定长度值，是杂凑算法的基础原理，MD5的前身有MD2、MD3和MD4。MD5的作用是让大容量信息在用数字签名软件签署私人密钥前被"压缩"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的十六进制数字串）。   
除了MD5以外，其中比较有名的还有sha-1、RIPEMD以及Haval等

Java代码  

1. import java.security.MessageDigest;
2. import java.security.NoSuchAlgorithmException;
4. public class EncrypMD5 {
6. public byte[] eccrypt(String info) throws NoSuchAlgorithmException{
7. //根据MD5算法生成MessageDigest对象
8. MessageDigest md5 = MessageDigest.getInstance("MD5");
9. byte[] srcBytes = info.getBytes();
10. //使用srcBytes更新摘要
11. md5.update(srcBytes);
12. //完成哈希计算，得到result
13. byte[] resultBytes = md5.digest();
14. return resultBytes;
15. }

18. public static void main(String args[]) throws NoSuchAlgorithmException{
19. String msg = "郭XX-精品相声技术";
20. EncrypMD5 md5 = new EncrypMD5();
21. byte[] resultBytes = md5.eccrypt(msg);
23. System.out.println("密文是：" + new String(resultBytes));
24. System.out.println("明文是：" + msg);
25. }
27. }

2. SHA 是一种数据加密算法，该算法经过加密专家多年来的发展和改进已日益完善，现在已成为公认的最安全的散列算法之一，并被广泛使用。该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。散列函数值可以说时对明文的一种“指纹”或是“摘要”所以对散列值的数字签名就可以视为对此明文的数字签名。

Java代码  

1. import java.security.MessageDigest;
2. import java.security.NoSuchAlgorithmException;
4. public class EncrypSHA {
6. public byte[] eccrypt(String info) throws NoSuchAlgorithmException{
7. MessageDigest md5 = MessageDigest.getInstance("SHA");
8. byte[] srcBytes = info.getBytes();
9. //使用srcBytes更新摘要
10. md5.update(srcBytes);
11. //完成哈希计算，得到result
12. byte[] resultBytes = md5.digest();
13. return resultBytes;
14. }
16. /\*\*
17. \* @param args
18. \* @throws NoSuchAlgorithmException
19. \*/
20. public static void main(String[] args) throws NoSuchAlgorithmException {
21. String msg = "郭XX-精品相声技术";
22. EncrypSHA sha = new EncrypSHA();
23. byte[] resultBytes = sha.eccrypt(msg);
24. System.out.println("明文是：" + msg);
25. System.out.println("密文是：" + new String(resultBytes));
27. }
29. }

附件中是以上几种的源代码，附带额外的两种使用方式。   
  
增加一种关于文件的哈希算法源代码：

Java代码  

1. import java.io.FileInputStream;
2. import java.io.InputStream;
3. import java.security.MessageDigest;
5. public class FileHashUtil {
7. public static final char[] hexChar = {
8. '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };
9. public static final String[] hashTypes = new String[] { "MD2", "MD5", "SHA1", "SHA-256", "SHA-384", "SHA-512" };
11. public void MD5File(String fileName) throws Exception{
12. //String fileName = args[0];
13. System.out.println("需要获取hash的文件为：　" + fileName);
14. java.util.List<MessageDigest> mds = new java.util.ArrayList<MessageDigest>();
15. for (String hashType : hashTypes) {
16. MessageDigest md = MessageDigest.getInstance(hashType);
17. mds.add(md);
18. }
19. InputStream fis = null;
20. try {
21. fis = new FileInputStream(fileName);
22. byte[] buffer = new byte[1024];
23. int numRead = 0;
24. while ((numRead = fis.read(buffer)) > 0) {
25. for (MessageDigest md : mds) {
26. md.update(buffer, 0, numRead);
27. }
28. }
29. } catch (Exception ex) {
30. ex.printStackTrace();
31. } finally {
32. if (fis != null) {
33. fis.close();
34. }
35. }
36. for (MessageDigest md : mds) {
37. System.out.println(md.getAlgorithm() + " == " + toHexString(md.digest()));
38. }
39. }

42. public static void main(String[] args) throws Exception {
43. String[] fileName = new String[] {"D:/hapfish/ShellFolder.java","D:/hapfish/ShellFolder - 副本.java",
44. "E:/ShellFolder - 副本.java","E:/ShellFolder.txt","D:/hapfish/ShellFolder.jpg",
45. "E:/ShellFolder增加字符.txt","D:/hapfish/birosoft.jar"};
46. FileHashUtil files  = new FileHashUtil();
47. for(int i=0;i<fileName.length;i++){
48. files.MD5File(fileName[i]);
49. }

52. }
54. public static String toHexString(byte[] b) {
55. StringBuilder sb = new StringBuilder(b.length \* 2);
56. for (int i = 0; i < b.length; i++) {
57. sb.append(hexChar[(b[i] & 0xf0) >>> 4]);
58. sb.append(hexChar[b[i] & 0x0f]);
59. }
60. return sb.toString();
61. }
63. }

运行说明

说明代码  

1. "D:/hapfish/ShellFolder.java",
2. "D:/hapfish/ShellFolder - 副本.java",
3. "E:/ShellFolder - 副本.java",
4. "E:/ShellFolder.txt",
5. "D:/hapfish/ShellFolder.jpg",
6. 以上五个文件是同一文件经过复制、改扩展名的，最后计算哈希结果是一致的。
8. "E:/ShellFolder增加字符.txt" 增加了几个字符串，就不一样了
10. "D:/hapfish/birosoft.jar" 完全不相关的另外一个文件

运行结果：

Java代码  

1. 需要获取hash的文件为：　D:/hapfish/ShellFolder.java
2. MD2 == 3a755a99c5e407005cd45ebd856b4649
3. MD5 == 5d08d440fa911d1e418c69a90b83cd86
4. SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63
5. SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99
6. SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6
7. SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458
8. 需要获取hash的文件为：　D:/hapfish/ShellFolder - 副本.java
9. MD2 == 3a755a99c5e407005cd45ebd856b4649
10. MD5 == 5d08d440fa911d1e418c69a90b83cd86
11. SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63
12. SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99
13. SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6
14. SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458
15. 需要获取hash的文件为：　E:/ShellFolder - 副本.java
16. MD2 == 3a755a99c5e407005cd45ebd856b4649
17. MD5 == 5d08d440fa911d1e418c69a90b83cd86
18. SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63
19. SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99
20. SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6
21. SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458
22. 需要获取hash的文件为：　E:/ShellFolder.txt
23. MD2 == 3a755a99c5e407005cd45ebd856b4649
24. MD5 == 5d08d440fa911d1e418c69a90b83cd86
25. SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63
26. SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99
27. SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6
28. SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458
29. 需要获取hash的文件为：　D:/hapfish/ShellFolder.jpg
30. MD2 == 3a755a99c5e407005cd45ebd856b4649
31. MD5 == 5d08d440fa911d1e418c69a90b83cd86
32. SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63
33. SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99
34. SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6
35. SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458
36. 需要获取hash的文件为：　E:/ShellFolder增加字符.txt
37. MD2 == f2717c24c6c0e110457bd17221c9ca6c
38. MD5 == c49e353a7c4c26bd7ccb5e90917c230f
39. SHA1 == 477c8a9e465bfaa4be42d35c032a17f7e6b42b97
40. SHA-256 == 9fa18adaf242ebcdc6563922d84c2a163c82e1a24db2eb2b73978ed1f354a8a3
41. SHA-384 == 4eee8f8e6d64d21c15dc01fa049f4d12a3b8e1d94d87763fe0bea75ab5ea8432fa8251289ece45ee39fe3d36b3c3020c
42. SHA-512 == e852ec0ff77250be497389d2f5a1818c18bb66106b9905c4ee26fe0d256eb3b77e0ce9a28a84e4b67e4332ba37ec3aa7518148e3a682318c0fc34c391f45c201
43. 需要获取hash的文件为：　D:/hapfish/birosoft.jar
44. MD2 == 38c5e1404718916dec59c33cafc909b3
45. MD5 == dc3e2cc4fb3949cf3660e0f5f8c3fba3
46. SHA1 == cde3dc25498afc5a563af0bb0eb54dc45f71bb28
47. SHA-256 == adf6a961c70c6ea677dff066fc5d896fb0beb4dd442ca0eb619ae1d1b04291e5
48. SHA-384 == fe7c6b754893c53ebd82bb53703fb5cc32115c9a38f98072f73def90729b271ee3c5c78e258bd9ff5ee5476193c2178b
49. SHA-512 == a15376f327256a6e049dfbdc5c2ad3a98bffccc6fa92ee01ff53db6b04471ca0f45ca28f76ff4a6911b57825afa046671299141f2499d71f1dac618c92385491

最后，把运行结果贴出来有点占空间，主要为了说明表述自己的猜想。一般来说同一哈希算法对同一文件(镜像、扩展名被修改)所产生的结果应该是一致的。   
  
因此有个猜想，在baidu文库、腾讯的群共享上传时，先会判断是否有相同文件，从某种可能上来说也采用了对文件的哈希算法，毕竟从本地运算一个哈希算法后获得的数值要比把整个文件传过去比较实惠得多。而且字符串的比较也是很方便的。   
  
对于某一种哈希算法，存在一种可能：就是两个不同的文件，计算出来的哈希值可能是一样的。当然为了保险，可以用两种甚至更多的哈希算法，只有在每种算法获得的哈希值都相同时，才能判断是同一个文件。   
如果我们也对用户上传的文件进行哈希计算的话，就可以节省资源，同样的文件按理说可以减少上传次数……