# JAVA 上加密算法的实现用例

原文链接：<https://www.ibm.com/developerworks/cn/java/l-security/>

## 第 1 章基础知识

### 1.1. 单钥密码体制

单钥密码体制是一种传统的加密算法，是指信息的发送方和接收方共同使用同一把密钥进行加解密。

通常 , 使用的加密算法 比较简便高效 , 密钥简短，加解密速度快，破译极其困难。但是加密的安全性依靠密钥保管的安全性 , 在公开的计算机网络上安全地传送和保管密钥是一个严峻的问题，并且如果在多用户的情况下密钥的保管安全性也是一个问题。

单钥密码体制的代表是美国的 DES

### 1.2. 消息摘要

一个消息摘要就是一个数据块的数字指纹。即对一个任意长度的一个数据块进行计算，产生一个唯一指印（对于 SHA1 是产生一个 20 字节的二进制数组）。

消息摘要有两个基本属性：

* 两个不同的报文难以生成相同的摘要
* 难以对指定的摘要生成一个报文，而由该报文反推算出该指定的摘要

代表：美国国家标准技术研究所的 SHA1 和麻省理工学院 Ronald Rivest 提出的 MD5

### 1.3. Diffie-Hellman 密钥一致协议

密钥一致协议是由公开密钥密码体制的奠基人 Diffie 和 Hellman 所提出的一种思想。

先决条件 , 允许两名用户在公开媒体上交换信息以生成"一致"的 , 可以共享的密钥

代表：指数密钥一致协议 (Exponential Key Agreement Protocol)

### 1.4. 非对称算法与公钥体系

1976 年，Dittie 和 Hellman 为解决密钥管理问题，在他们的奠基性的工作"密码学的新方向"一文中，提出一种密钥交换协议，允许在不安全的媒体上通过通讯双方交换信息，安全地传送秘密密钥。在此新思想的基础上，很快出现了非对称密钥密码体制，即公钥密码体制。在公钥体制中，加密密钥不同于解密密钥，加密密钥公之于众，谁都可以使用；解密密钥只有解密人自己知道。它们分别称为公开密钥（Public key）和秘密密钥（Private key）。

迄今为止的所有公钥密码体系中，RSA 系统是最著名、最多使用的一种。RSA 公开密钥密码系统是由 R.Rivest、A.Shamir 和 L.Adleman 俊教授于 1977 年提出的。RSA 的取名就是来自于这三位发明者的姓的第一个字母

### 1.5. 数字签名

所谓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。

在数字签名中有重要作用的数字指纹是通过一类特殊的散列函数（HASH 函数）生成的，对这些 HASH 函数的特殊要求是：

1. 接受的输入报文数据没有长度限制；
2. 对任何输入报文数据生成固定长度的摘要（数字指纹）输出
3. 从报文能方便地算出摘要；
4. 难以对指定的摘要生成一个报文，而由该报文反推算出该指定的摘要；
5. 两个不同的报文难以生成相同的摘要

代表：DSA

## 第 2 章在 JAVA 中的实现

### 2.1. 相关

Diffie-Hellman 密钥一致协议和 DES 程序需要 JCE 工具库的支持 , 可以到 <http://java.sun.com/security/index.html> 下载 JCE, 并进行安装。简易安装把 jce1.2.1\lib 下的所有内容复制到 %java\_home%\lib\ext 下 , 如果没有 ext 目录自行建立 , 再把 jce1\_2\_1.jar 和 sunjce\_provider.jar 添加到 CLASSPATH 内 , 更详细说明请看相应用户手册

### 2.2. 消息摘要 MD5 和 SHA 的使用

使用方法 :

首先用生成一个 MessageDigest 类 , 确定计算方法

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

添加要进行计算摘要的信息

alga.update(myinfo.getBytes());

计算出摘要

byte[] digesta=alga.digest();

发送给其他人你的信息和摘要

其他人用相同的方法初始化 , 添加信息 , 最后进行比较摘要是否相同

algb.isEqual(digesta,algb.digest())

相关 AIP

java.security.MessageDigest 类

static getInstance(String algorithm)

返回一个 MessageDigest 对象 , 它实现指定的算法

参数 : 算法名 , 如 SHA-1 或 MD5

void update (byte input)

void update (byte[] input)

void update(byte[] input, int offset, int len)

添加要进行计算摘要的信息

byte[] digest()

完成计算 , 返回计算得到的摘要 ( 对于 MD5 是 16 位 ,SHA 是 20 位 )

void reset()

复位

static boolean isEqual(byte[] digesta, byte[] digestb)

比效两个摘要是否相同

代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | import java.security.\*;   public class myDigest {    public static void main(String[] args)  {      myDigest my=new myDigest();      my.testDigest();    }    public void testDigest()    {     try {       String myinfo="我的测试信息";      //java.security.MessageDigest alg=java.security.MessageDigest.getInstance("MD5");        java.security.MessageDigest alga=java.security.MessageDigest.getInstance("SHA-1");        alga.update(myinfo.getBytes());        byte[] digesta=alga.digest();        System.out.println("本信息摘要是 :"+byte2hex(digesta));        // 通过某中方式传给其他人你的信息 (myinfo) 和摘要 (digesta) 对方可以判断是否更改或传输正常        java.security.MessageDigest algb=java.security.MessageDigest.getInstance("SHA-1");        algb.update(myinfo.getBytes());        if (algb.isEqual(digesta,algb.digest())) {           System.out.println("信息检查正常");         }         else          {            System.out.println("摘要不相同");           }     }     catch (java.security.NoSuchAlgorithmException ex) {       System.out.println("非法摘要算法");     }    }    public String byte2hex(byte[] b) // 二行制转字符串      {       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;         if (n<b.length-1)  hs=hs+":";        }       return hs.toUpperCase();      }   } |

### 2.3. 数字签名 DSA

1. 对于一个用户来讲首先要生成他的密钥对 , 并且分别保存

生成一个 KeyPairGenerator 实例

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | java.security.KeyPairGenerator keygen=java.security.KeyPairGenerator.getInstance("DSA");    //如果设定随机产生器就用如相代码初始化  SecureRandom secrand=new SecureRandom();  secrand.setSeed("tttt".getBytes()); // 初始化随机产生器  keygen.initialize(512,secrand);     // 初始化密钥生成器    //否则  keygen.initialize(512);    //生成密钥公钥 pubkey 和私钥 prikey  KeyPair keys=keygen.generateKeyPair(); // 生成密钥组  PublicKey pubkey=keys.getPublic();  PrivateKey prikey=keys.getPrivate();    //分别保存在 myprikey.dat 和 mypubkey.dat 中 , 以便下次不在生成  //( 生成密钥对的时间比较长  java.io.ObjectOutputStream out=new java.io.ObjectOutputStream(      new java.io.FileOutputStream("myprikey.dat"));  out.writeObject(prikey);  out.close();  out=new java.io.ObjectOutputStream(new java.io.FileOutputStream("mypubkey.dat"));  out.writeObject(pubkey);  out.close(); |

1. 用他私人密钥 (prikey) 对他所确认的信息 (info) 进行数字签名产生一个签名数组

从文件中读入私人密钥 (prikey)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | java.io.ObjectInputStream in=new java.io.ObjectInputStream(      new java.io.FileInputStream("myprikey.dat"));  PrivateKey myprikey=(PrivateKey)in.readObject();  in.close();  初始一个 Signature 对象 , 并用私钥对信息签名  java.security.Signature signet=java.security.Signature.getInstance("DSA");  signet.initSign(myprikey);  signet.update(myinfo.getBytes());  byte[] signed=signet.sign();    把信息和签名保存在一个文件中 (myinfo.dat)  java.io.ObjectOutputStream out=new java.io.ObjectOutputStream(      new java.io.FileOutputStream("myinfo.dat"));  out.writeObject(myinfo);  out.writeObject(signed);  out.close();  把他的公钥的信息及签名发给其它用户 |

1. 其他用户用他的公共密钥 (pubkey) 和签名 (signed) 和信息 (info) 进行验证是否由他签名的信息

读入公钥  
java.io.ObjectInputStream in=new java.io.ObjectInputStream(new java.io.FileInputStream("mypubkey.dat"));  
PublicKey pubkey=(PublicKey)in.readObject();  
in.close();

读入签名和信息  
in=new java.io.ObjectInputStream(new java.io.FileInputStream("myinfo.dat"));  
String info=(String)in.readObject();  
byte[] signed=(byte[])in.readObject();  
in.close();

初始一个 Signature 对象 , 并用公钥和签名进行验证  
java.security.Signature signetcheck=java.security.Signature.getInstance("DSA");  
signetcheck.initVerify(pubkey);  
signetcheck.update(info.getBytes());  
if (signetcheck.verify(signed)) { System.out.println("签名正常");}

对于密钥的保存本文是用对象流的方式保存和传送的 , 也可可以用编码的方式保存 . 注意要  
import java.security.spec.\*  
import java.security.\*

具休说明如下

* + public key 是用 X.509 编码的 , 例码如下 :

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | byte[] bobEncodedPubKey=mypublic.getEncoded(); // 生成编码    // 传送二进制编码    // 以下代码转换编码为相应 key 对象    X509EncodedKeySpec bobPubKeySpec = new X509EncodedKeySpec(bobEncodedPubKey);    KeyFactory keyFactory = KeyFactory.getInstance("DSA");    PublicKey bobPubKey = keyFactory.generatePublic(bobPubKeySpec); |

* + 对于 Private key 是用 PKCS#8 编码 , 例码如下 :

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | byte[] bPKCS=myprikey.getEncoded();    // 传送二进制编码    // 以下代码转换编码为相应 key 对象    PKCS8EncodedKeySpec priPKCS8=new PKCS8EncodedKeySpec(bPKCS);    KeyFactory keyf=KeyFactory.getInstance("DSA");    PrivateKey otherprikey=keyf.generatePrivate(priPKCS8); |

1. 常用 API

java.security.KeyPairGenerator 密钥生成器类  
public static KeyPairGenerator getInstance(String algorithm) throws NoSuchAlgorithmException  
以指定的算法返回一个 KeyPairGenerator 对象  
参数 : algorithm 算法名 . 如 :"DSA","RSA"

public void initialize(int keysize)

以指定的长度初始化 KeyPairGenerator 对象 , 如果没有初始化系统以 1024 长度默认设置

参数 :keysize 算法位长 . 其范围必须在 512 到 1024 之间，且必须为 64 的倍数

public void initialize(int keysize, SecureRandom random)  
以指定的长度初始化和随机发生器初始化 KeyPairGenerator 对象  
参数 :keysize 算法位长 . 其范围必须在 512 到 1024 之间，且必须为 64 的倍数  
random 一个随机位的来源 ( 对于 initialize(int keysize) 使用了默认随机器

public abstract KeyPair generateKeyPair()  
产生新密钥对

java.security.KeyPair 密钥对类  
public PrivateKey getPrivate()  
返回私钥

public PublicKey getPublic()  
返回公钥

java.security.Signature 签名类  
public static Signature getInstance(String algorithm) throws NoSuchAlgorithmException  
返回一个指定算法的 Signature 对象  
参数 algorithm 如 :"DSA"

public final void initSign(PrivateKey privateKey)  
throws InvalidKeyException  
用指定的私钥初始化  
参数 :privateKey 所进行签名时用的私钥

public final void update(byte data)  
throws SignatureException  
public final void update(byte[] data)  
throws SignatureException  
public final void update(byte[] data, int off, int len)  
throws SignatureException  
添加要签名的信息

public final byte[] sign()  
throws SignatureException  
返回签名的数组 , 前提是 initSign 和 update

public final void initVerify(PublicKey publicKey)  
throws InvalidKeyException  
用指定的公钥初始化  
参数 :publicKey 验证时用的公钥

public final boolean verify(byte[] signature)  
throws SignatureException  
验证签名是否有效 , 前提是已经 initVerify 初始化  
参数 : signature 签名数组

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126 | import java.security.\*;  import java.security.spec.\*;  public class testdsa {     public static void main(String[] args) throws java.security.NoSuchAlgorithmException,       java.lang.Exception {         testdsa my=new testdsa();         my.run();   }   public void run()   {   // 数字签名生成密钥   // 第一步生成密钥对 , 如果已经生成过 , 本过程就可以跳过 ,   // 对用户来讲 myprikey.dat 要保存在本地   // 而 mypubkey.dat 给发布给其它用户    if ((new java.io.File("myprikey.dat")).exists()==false) {        if (generatekey()==false) {            System.out.println("生成密钥对败");            return;           };         }  // 第二步 , 此用户  // 从文件中读入私钥 , 对一个字符串进行签名后保存在一个文件 (myinfo.dat) 中  // 并且再把 myinfo.dat 发送出去  // 为了方便数字签名也放进了 myifno.dat 文件中 , 当然也可分别发送   try {   java.io.ObjectInputStream in=new java.io.ObjectInputStream(       new java.io.FileInputStream("myprikey.dat"));   PrivateKey myprikey=(PrivateKey)in.readObject();   in.close();  // java.security.spec.X509EncodedKeySpec pubX509=  //   new java.security.spec.X509EncodedKeySpec(bX509);  //java.security.spec.X509EncodedKeySpec pubkeyEncode=  //   java.security.spec.X509EncodedKeySpec   String myinfo="这是我的信息";    // 要签名的信息   // 用私钥对信息生成数字签名   java.security.Signature signet=java.security.Signature.getInstance("DSA");   signet.initSign(myprikey);   signet.update(myinfo.getBytes());   byte[] signed=signet.sign();  // 对信息的数字签名   System.out.println("signed( 签名内容 )="+byte2hex(signed));  // 把信息和数字签名保存在一个文件中   java.io.ObjectOutputStream out=new java.io.ObjectOutputStream(       new java.io.FileOutputStream("myinfo.dat"));   out.writeObject(myinfo);   out.writeObject(signed);   out.close();   System.out.println("签名并生成文件成功");   }   catch (java.lang.Exception e) {     e.printStackTrace();     System.out.println("签名并生成文件失败");   };   // 第三步   // 其他人通过公共方式得到此户的公钥和文件   // 其他人用此户的公钥 , 对文件进行检查 , 如果成功说明是此用户发布的信息 .   //   try {    java.io.ObjectInputStream in=new java.io.ObjectInputStream(        new java.io.FileInputStream("mypubkey.dat"));    PublicKey pubkey=(PublicKey)in.readObject();    in.close();    System.out.println(pubkey.getFormat());    in=new java.io.ObjectInputStream(new java.io.FileInputStream("myinfo.dat"));    String info=(String)in.readObject();    byte[] signed=(byte[])in.readObject();    in.close();   java.security.Signature signetcheck=java.security.Signature.getInstance("DSA");   signetcheck.initVerify(pubkey);   signetcheck.update(info.getBytes());   if (signetcheck.verify(signed)) {   System.out.println("info="+info);    System.out.println("签名正常");   }   else  System.out.println("非签名正常");   }   catch (java.lang.Exception e) {e.printStackTrace();};   }   // 生成一对文件 myprikey.dat 和 mypubkey.dat--- 私钥和公钥 ,   // 公钥要用户发送 ( 文件 , 网络等方法 ) 给其它用户 , 私钥保存在本地   public boolean generatekey()   {     try {   java.security.KeyPairGenerator keygen =       java.security.KeyPairGenerator.getInstance("DSA");  // SecureRandom secrand=new SecureRandom();  // secrand.setSeed("tttt".getBytes()); // 初始化随机产生器  // keygen.initialize(576,secrand);     // 初始化密钥生成器   keygen.initialize(512);   KeyPair keys=keygen.genKeyPair();  //  KeyPair keys=keygen.generateKeyPair(); // 生成密钥组   PublicKey pubkey=keys.getPublic();   PrivateKey prikey=keys.getPrivate();   java.io.ObjectOutputStream out=new java.io.ObjectOutputStream(       new java.io.FileOutputStream("myprikey.dat"));   out.writeObject(prikey);   out.close();   System.out.println("写入对象 prikeys ok");   out=new java.io.ObjectOutputStream(       new java.io.FileOutputStream("mypubkey.dat"));    out.writeObject(pubkey);    out.close();    System.out.println("写入对象 pubkeys ok");    System.out.println("生成密钥对成功");    return true;   }   catch (java.lang.Exception e) {    e.printStackTrace();    System.out.println("生成密钥对失败");    return false;    };   }   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;        if (n<b.length-1)  hs=hs+":";       }      return hs.toUpperCase();     }  } |

### 2.4. DESede/DES 对称算法

首先生成密钥 , 并保存 ( 这里并没的保存的代码 , 可参考 DSA 中的方法 )

KeyGenerator keygen = KeyGenerator.getInstance(Algorithm);

SecretKey deskey = keygen.generateKey();

用密钥加密明文 (myinfo), 生成密文 (cipherByte)

Cipher c1 = Cipher.getInstance(Algorithm);

c1.init(Cipher.ENCRYPT\_MODE,deskey);

byte[] cipherByte=c1.doFinal(myinfo.getBytes());

传送密文和密钥 , 本文没有相应代码可参考 DSA

.............

用密钥解密密文

c1 = Cipher.getInstance(Algorithm);

c1.init(Cipher.DECRYPT\_MODE,deskey);

byte[] clearByte=c1.doFinal(cipherByte);

相对来说对称密钥的使用是很简单的 , 对于 JCE 来讲支技 DES,DESede,Blowfish 三种加密术

对于密钥的保存各传送可使用对象流或者用二进制编码 , 相关参考代码如下

|  |  |
| --- | --- |
| 1  2  3  4  5 | SecretKey deskey = keygen.generateKey();  byte[] desEncode=deskey.getEncoded();  javax.crypto.spec.SecretKeySpec destmp =      new javax.crypto.spec.SecretKeySpec(desEncode,Algorithm);  SecretKey mydeskey=destmp; |

相关 API

KeyGenerator 在 DSA 中已经说明 , 在添加 JCE 后在 instance 进可以如下参数

DES,DESede,Blowfish,HmacMD5,HmacSHA1

javax.crypto.Cipher 加 / 解密器

|  |  |
| --- | --- |
| 1  2  3 | public static final Cipher getInstance(java.lang.String transformation)                                  throws java.security.NoSuchAlgorithmException,                                         NoSuchPaddingException |

返回一个指定方法的 Cipher 对象

参数 :transformation 方法名 ( 可用 DES,DESede,Blowfish)

public final void init(int opmode, java.security.Key key)  
throws java.security.InvalidKeyException

用指定的密钥和模式初始化 Cipher 对象

参数 :opmode 方式 (ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE,UNWRAP\_MODE)

key 密钥

|  |  |
| --- | --- |
| 1  2  3  4 | public final byte[] doFinal(byte[] input)                       throws java.lang.IllegalStateException,                              IllegalBlockSizeException,                              BadPaddingException |

对 input 内的串 , 进行编码处理 , 返回处理后二进制串 , 是返回解密文还是加解文由 init 时的 opmode 决定

注意 : 本方法的执行前如果有 update, 是对 updat 和本次 input 全部处理 , 否则是本 inout 的内容

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | /\*  安全程序 DESede/DES 测试   \*/   import java.security.\*;   import javax.crypto.\*;   public class testdes {   public static void main(String[] args){      testdes my=new testdes();      my.run();    }   public  void run() {   // 添加新安全算法 , 如果用 JCE 就要把它添加进去   Security.addProvider(new com.sun.crypto.provider.SunJCE());   String Algorithm="DES"; // 定义 加密算法 , 可用 DES,DESede,Blowfish   String myinfo="要加密的信息";     try {     // 生成密钥     KeyGenerator keygen = KeyGenerator.getInstance(Algorithm);     SecretKey deskey = keygen.generateKey();     // 加密     System.out.println("加密前的二进串 :"+byte2hex(myinfo.getBytes()));     System.out.println("加密前的信息 :"+myinfo);     Cipher c1 = Cipher.getInstance(Algorithm);     c1.init(Cipher.ENCRYPT\_MODE,deskey);     byte[] cipherByte=c1.doFinal(myinfo.getBytes());      System.out.println("加密后的二进串 :"+byte2hex(cipherByte));     // 解密     c1 = Cipher.getInstance(Algorithm);     c1.init(Cipher.DECRYPT\_MODE,deskey);     byte[] clearByte=c1.doFinal(cipherByte);     System.out.println("解密后的二进串 :"+byte2hex(clearByte));     System.out.println("解密后的信息 :"+(new String(clearByte)));    }     catch (java.security.NoSuchAlgorithmException e1) {e1.printStackTrace();}     catch (javax.crypto.NoSuchPaddingException e2) {e2.printStackTrace();}     catch (java.lang.Exception e3) {e3.printStackTrace();}    }   public String byte2hex(byte[] b) // 二行制转字符串      {       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;         if (n<b.length-1)  hs=hs+":";        }       return hs.toUpperCase();      }   } |

### 2.5. Diffie-Hellman 密钥一致协议

公开密钥密码体制的奠基人 Diffie 和 Hellman 所提出的 "指数密钥一致协议"(Exponential Key Agreement Protocol), 该协议不要求别的安全性 先决条件 , 允许两名用户在公开媒体上交换信息以生成"一致"的 , 可以共享的密钥。在 JCE 的中实现用户 alice 生成 DH 类型的密钥对 , 如果长度用 1024 生成的时间请 , 推荐第一次生成后保存 DHParameterSpec, 以便下次使用直接初始化 . 使其速度加快

|  |  |
| --- | --- |
| 1  2  3  4 | System.out.println("ALICE: 产生 DH 对 ...");   KeyPairGenerator aliceKpairGen = KeyPairGenerator.getInstance("DH");   aliceKpairGen.initialize(512);   KeyPair aliceKpair = aliceKpairGen.generateKeyPair(); |

alice 生成公钥发送组 bob

|  |  |
| --- | --- |
| 1 | byte[] alicePubKeyEnc = aliceKpair.getPublic().getEncoded(); |

bob 从 alice 发送来的公钥中读出 DH 密钥对的初始参数生成 bob 的 DH 密钥对

注意这一步一定要做 , 要保证每个用户用相同的初始参数生成的

|  |  |
| --- | --- |
| 1  2  3  4 | DHParameterSpec dhParamSpec = ((DHPublicKey)alicePubKey).getParams();    KeyPairGenerator bobKpairGen = KeyPairGenerator.getInstance("DH");    bobKpairGen.initialize(dhParamSpec);    KeyPair bobKpair = bobKpairGen.generateKeyPair(); |

bob 根据 alice 的公钥生成本地的 DES 密钥

|  |  |
| --- | --- |
| 1  2  3  4 | KeyAgreement bobKeyAgree = KeyAgreement.getInstance("DH");    bobKeyAgree.init(bobKpair.getPrivate());    bobKeyAgree.doPhase(alicePubKey, true);    SecretKey bobDesKey = bobKeyAgree.generateSecret("DES"); |

bob 已经生成了他的 DES 密钥 , 他现把他的公钥发给 alice,

|  |  |
| --- | --- |
| 1 | byte[] bobPubKeyEnc = bobKpair.getPublic().getEncoded(); |

alice 根据 bob 的公钥生成本地的 DES 密钥

|  |  |
| --- | --- |
| 1  2  3  4  5 | ,,,,,, 解码  KeyAgreement aliceKeyAgree = KeyAgreement.getInstance("DH");  aliceKeyAgree.init(aliceKpair.getPrivate());  aliceKeyAgree.doPhase(bobPubKey, true);  SecretKey aliceDesKey = aliceKeyAgree.generateSecret("DES"); |

bob 和 alice 能过这个过程就生成了相同的 DES 密钥 , 在这种基础就可进行安全能信

****常用 API****

java.security.KeyPairGenerator 密钥生成器类  
public static KeyPairGenerator getInstance(String algorithm)  
throws NoSuchAlgorithmException  
以指定的算法返回一个 KeyPairGenerator 对象  
参数 : algorithm 算法名 . 如 : 原来是 DSA, 现在添加了 DiffieHellman(DH)

public void initialize(int keysize)  
以指定的长度初始化 KeyPairGenerator 对象 , 如果没有初始化系统以 1024 长度默认设置  
参数 :keysize 算法位长 . 其范围必须在 512 到 1024 之间，且必须为 64 的倍数  
注意 : 如果用 1024 生长的时间很长 , 最好生成一次后就保存 , 下次就不用生成了

public void initialize(AlgorithmParameterSpec params)  
throws InvalidAlgorithmParameterException  
以指定参数初始化

javax.crypto.interfaces.DHPublicKey  
public DHParameterSpec getParams()  
返回  
java.security.KeyFactory

public static KeyFactory getInstance(String algorithm)  
throws NoSuchAlgorithmException  
以指定的算法返回一个 KeyFactory  
参数 : algorithm 算法名 :DSH,DH

public final PublicKey generatePublic(KeySpec keySpec)  
throws InvalidKeySpecException  
根据指定的 key 说明 , 返回一个 PublicKey 对象

java.security.spec.X509EncodedKeySpec  
public X509EncodedKeySpec(byte[] encodedKey)  
根据指定的二进制编码的字串生成一个 key 的说明  
参数 :encodedKey 二进制编码的字串 ( 一般能过 PublicKey.getEncoded() 生成 )  
javax.crypto.KeyAgreement 密码一至类

public static final KeyAgreement getInstance(java.lang.String algorithm)  
throws java.security.NoSuchAlgorithmException  
返回一个指定算法的 KeyAgreement 对象  
参数 :algorithm 算法名 , 现在只能是 DiffieHellman(DH)

public final void init(java.security.Key key)  
throws java.security.InvalidKeyException  
用指定的私钥初始化  
参数 :key 一个私钥

public final java.security.Key doPhase(java.security.Key key,  
boolean lastPhase)  
throws java.security.InvalidKeyException,  
java.lang.IllegalStateException  
用指定的公钥进行定位 ,lastPhase 确定这是否是最后一个公钥 , 对于两个用户的  
情况下就可以多次定次 , 最后确定  
参数 :key 公钥  
lastPhase 是否最后公钥

public final SecretKey generateSecret(java.lang.String algorithm)  
throws java.lang.IllegalStateException,  
java.security.NoSuchAlgorithmException,  
java.security.InvalidKeyException  
根据指定的算法生成密钥  
参数 :algorithm 加密算法 ( 可用 DES,DESede,Blowfish)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 | \*/   import java.io.\*;   import java.math.BigInteger;   import java.security.\*;   import java.security.spec.\*;   import java.security.interfaces.\*;   import javax.crypto.\*;   import javax.crypto.spec.\*;   import javax.crypto.interfaces.\*;   import com.sun.crypto.provider.SunJCE;   public class testDHKey {      public static void main(String argv[]) {      try {          testDHKey my= new testDHKey();          my.run();      } catch (Exception e) {          System.err.println(e);      }      }      private void run() throws Exception {          Security.addProvider(new com.sun.crypto.provider.SunJCE());      System.out.println("ALICE: 产生 DH 对 ...");      KeyPairGenerator aliceKpairGen = KeyPairGenerator.getInstance("DH");          aliceKpairGen.initialize(512);      KeyPair aliceKpair = aliceKpairGen.generateKeyPair(); // 生成时间长          // 张三 (Alice) 生成公共密钥 alicePubKeyEnc 并发送给李四 (Bob) ,          // 比如用文件方式 ,socket.....      byte[] alicePubKeyEnc = aliceKpair.getPublic().getEncoded();         //bob 接收到 alice 的编码后的公钥 , 将其解码      KeyFactory bobKeyFac = KeyFactory.getInstance("DH");      X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec  (alicePubKeyEnc);      PublicKey alicePubKey = bobKeyFac.generatePublic(x509KeySpec);          System.out.println("alice 公钥 bob 解码成功");       // bob 必须用相同的参数初始化的他的 DH KEY 对 , 所以要从 Alice 发给他的公开密钥 ,           // 中读出参数 , 再用这个参数初始化他的 DH key 对           // 从 alicePubKye 中取 alice 初始化时用的参数      DHParameterSpec dhParamSpec = ((DHPublicKey)alicePubKey).getParams();      KeyPairGenerator bobKpairGen = KeyPairGenerator.getInstance("DH");      bobKpairGen.initialize(dhParamSpec);      KeyPair bobKpair = bobKpairGen.generateKeyPair();          System.out.println("BOB: 生成 DH key 对成功");      KeyAgreement bobKeyAgree = KeyAgreement.getInstance("DH");      bobKeyAgree.init(bobKpair.getPrivate());          System.out.println("BOB: 初始化本地 key 成功");          // 李四 (bob) 生成本地的密钥 bobDesKey      bobKeyAgree.doPhase(alicePubKey, true);      SecretKey bobDesKey = bobKeyAgree.generateSecret("DES");      System.out.println("BOB: 用 alice 的公钥定位本地 key, 生成本地 DES 密钥成功");          // Bob 生成公共密钥 bobPubKeyEnc 并发送给 Alice,          // 比如用文件方式 ,socket....., 使其生成本地密钥      byte[] bobPubKeyEnc = bobKpair.getPublic().getEncoded();          System.out.println("BOB 向 ALICE 发送公钥");           // alice 接收到 bobPubKeyEnc 后生成 bobPubKey           // 再进行定位 , 使 aliceKeyAgree 定位在 bobPubKey      KeyFactory aliceKeyFac = KeyFactory.getInstance("DH");      x509KeySpec = new X509EncodedKeySpec(bobPubKeyEnc);      PublicKey bobPubKey = aliceKeyFac.generatePublic(x509KeySpec);         System.out.println("ALICE 接收 BOB 公钥并解码成功");   ;      KeyAgreement aliceKeyAgree = KeyAgreement.getInstance("DH");      aliceKeyAgree.init(aliceKpair.getPrivate());          System.out.println("ALICE: 初始化本地 key 成功");      aliceKeyAgree.doPhase(bobPubKey, true);          // 张三 (alice) 生成本地的密钥 aliceDesKey      SecretKey aliceDesKey = aliceKeyAgree.generateSecret("DES");          System.out.println("ALICE: 用 bob 的公钥定位本地 key, 并生成本地 DES 密钥");          if (aliceDesKey.equals(bobDesKey)) System.out.println("张三和李四的密钥相同");         // 现在张三和李四的本地的 deskey 是相同的所以 , 完全可以进行发送加密 , 接收后解密 , 达到         // 安全通道的的目的          /\*           \* bob 用 bobDesKey 密钥加密信息           \*/      Cipher bobCipher = Cipher.getInstance("DES");      bobCipher.init(Cipher.ENCRYPT\_MODE, bobDesKey);          String bobinfo= "这是李四的机密信息";          System.out.println("李四加密前原文 :"+bobinfo);      byte[] cleartext =bobinfo.getBytes();      byte[] ciphertext = bobCipher.doFinal(cleartext);          /\*           \* alice 用 aliceDesKey 密钥解密           \*/      Cipher aliceCipher = Cipher.getInstance("DES");      aliceCipher.init(Cipher.DECRYPT\_MODE, aliceDesKey);      byte[] recovered = aliceCipher.doFinal(ciphertext);          System.out.println("alice 解密 bob 的信息 :"+(new String(recovered)));      if (!java.util.Arrays.equals(cleartext, recovered))          throw new Exception("解密后与原文信息不同");      System.out.println("解密后相同");      }   } |

## 第 3 章小结

在加密术中生成密钥对时，密钥对的当然是越长越好，但费时也越多，请从中从实际出发选取合适的长度，大部分例码中的密钥是每次运行就从新生成，在实际的情况中是生成后在一段时间保存在文件中，再次运行直接从文件中读入，从而加快速度。当然定时更新和加强密钥保管的安全性也是必须的。