实验1 Java环境下的加解密实验

1. 实验名称

Java环境下的加解密

1. 实验目的
2. 掌握常用的对称加密算法DES的工作原理
3. 掌握常用的消息摘要算法MD5的工作原理。
4. Java语言编程实现DES和MD5算法
5. Java框架中JCA、JCE、JSSE的使用
6. 实验环境

Eclipse

1. 实验步骤

**（1）DES算法工作原理**

DES算法的流程如图1所示：有16个相同的处理过程，称为“回次”(round)，并在首尾各有一次置换，称为IP与FP（或称IP-1，FP为IP的反函数（即IP“撤销”FP的操作，反之亦然）。

在主处理回次前，数据块被分成两个32位的半块，并被分别处理；这种交叉的方式被称为费斯妥结构。费斯妥结构保证了加密和解密过程足够相似，唯一的区别在于子密钥在解密时是以反向的顺序应用的，而剩余部分均相同。这样的设计大大简化了算法的实现，尤其是硬件实现，因为没有区分加密和解密算法的需要。

图中的⊕符号代表异或（XOR）操作。“F函数”将数据半块与某个子密钥进行处理。然后，一个F函数的输出与另一个半块异或之后，再与原本的半块组合并交换顺序，进入下一个回次的处理。在最后一个回次完成时，两个半块不必交换顺序，这是费斯妥结构的一个特点，以保证加解密的过程相似。

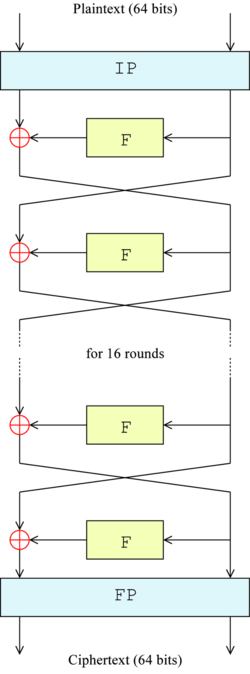


图1 DES算法总体流程图

图2中显示了Feistel函数（F函数）的过程。其每次对半块（32位）进行操作，并包括四个步骤：

（1）扩张—用扩张置换（图中的E）将32位的半块扩展到48位，其输出包括8个6位的块，每块包含4位对应的输入位，加上两个邻接的块中紧邻的位。

（2）与密钥混合—用异或操作将扩张的结果和一个子密钥进行混合。16个48位的子密钥—每个用于一个回次的F变换—是利用密钥调度从主密钥生成的（见下文）。

（3）S盒—在与子密钥混合之后，块被分成8个6位的块，然后使用“S盒”，或称“置换盒”进行处理。8个S盒的每一个都使用以查找表方式提供的非线性的变换将它的6个输入位变成4个输出位。S盒提供了DES的核心安全性—如果没有S盒，密码会是线性的，很容易破解。

（4）置换—最后，S盒的32个输出位利用固定的置换，“P置换”进行重组。这个设计是为了将每个S盒的4位输出在下一回次的扩张后，使用4个不同的S盒进行处理。

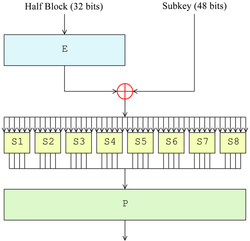


图2 DES的Feistel函数（F函数）

S盒、P置换和E扩张各自满足了克劳德·香农在1940年代提出的实用密码所需的必要条件，“混淆和扩散”。

图3显示了加密过程中的密钥调度—产生子密钥的算法。首先，使用选择置换1（PC-1）从64位输入密钥中选出56位的密钥—剩下的8位要么直接丢弃，要么作为奇偶校验位。然后，56位分成两个28位的半密钥；每个半密钥接下来都被分别处理。在接下来的回次中，两个半密钥都被左移1或2位（由回次数决定），然后通过选择置换2（PC-2）产生48位的子密钥—每个半密钥24位。移位（图中由<<标示）表明每个子密钥中使用了不同的位，每个位大致在16个子密钥中的14个出现。

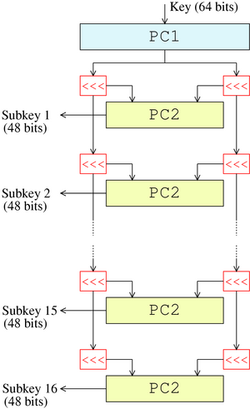


图3 DES的密钥生成

解密过程中，除了子密钥输出的顺序相反外，密钥调度的过程与加密完全相同。

**（2）MD5算法工作原理**

MD5的算法工作原理如图4所示：

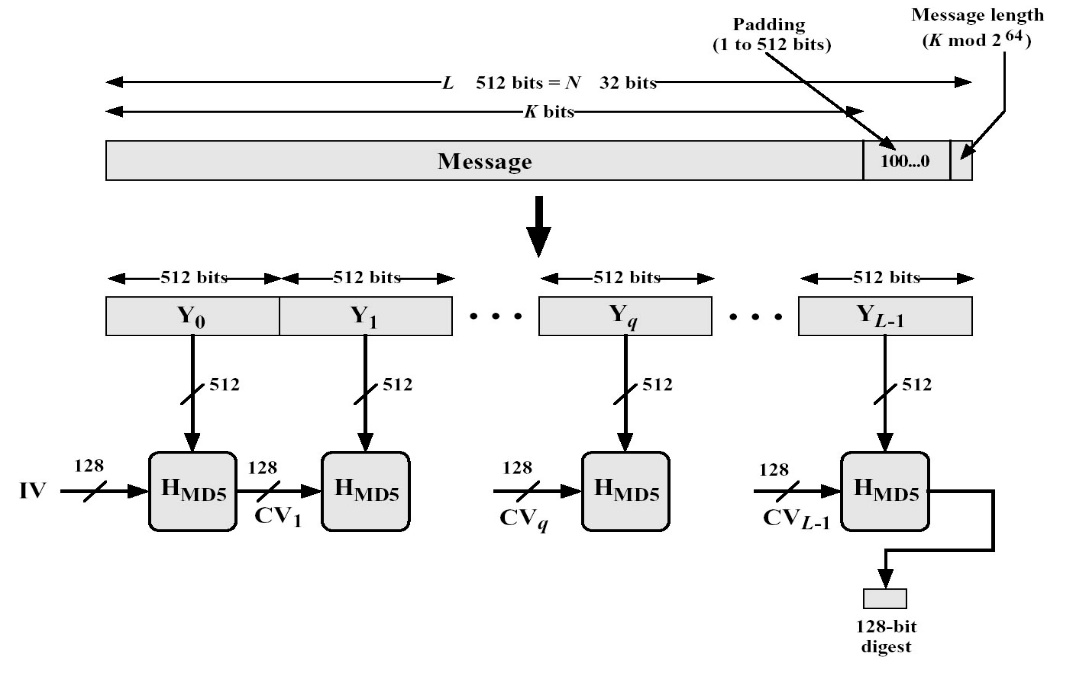


图4 MD5算法原理图

其基本操作过程如下：

（1）对明文输入按512bit分组，填充使其成为512bit的整数倍，且最后一组的后64bit用来表示按时消息的长在mod下的值K。填充位数为1~512bit，填充数字图样为（100…0），得Y0,Y1,…YL-1。其中，YL为512bit，按字记消息长为N=L\*16。即信息有L组，每组16个字，每字32bit，M0=[0，1，…，N-1]

（2） 每轮输出为128bit，可用下述四个32bit字：A，B，C，D表示，其初始存数以十六进制表示为：A=01234567，B=89ABCDEF，C=FEDCBA98，D=76543210。

（3）HMD-5的运算。对512bit（16-字）组进行运算，Yq表示输入的第q组512bit数据，在各轮中参加运算。T[1,2,…64]为64个元素表，分四组参与不同轮的计算。T[i]是232\*abs(sin(i))的整数部分，i是弧度。可用32bit二元数表示，T是32bit的随机数源。



图5 HMD5算法原理



一般128位的MD5散列被表示为32位十六进制数字

**（3）Java框架中JCA、JCE、JSSE的使用**

采用JCA和JCE框架使用DES、RSA、MD5以及消息验证码算法对“Don't tell anybody!!”信息进行处理，验证其与1，2步中的结果是否一致。

1. 实验结果及截图

**（1）DES算法使用Java语言实现**

初始子密钥：12345678，通过密钥扩展生成16个48bit的子密钥，用于16个圈函数中。

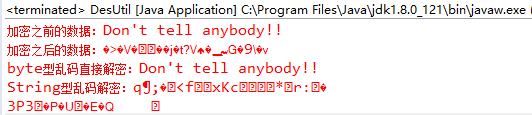
DES算法加密的明文：Don't tell anybody!!

DES算法加密的密文：�>�V�\_\_��j�t?V\_�؄G�9\�v

DES算法加密的byte[]型密文直接解密：Don't tell anybody!!

DES算法加密的密文先转为String再解密：q¶;\_�\_<f��xKc����\*�r:\_�3P3\_�P�U\_�E�Q \_

结果截图：

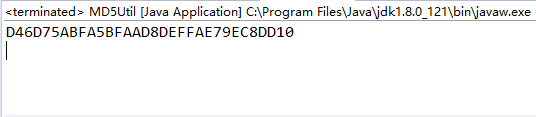


**（2）MD5算法使用Java语言实现**

MD5算法加密的明文：Don't tell anybody!!

MD5算法加密的密文：D46D75ABFA5BFAAD8DEFFAE79EC8DD10

结果截图：



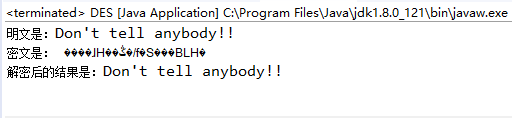
**（3）Java框架中JCA、JCE、JSSE的使用**

**1.使用JCA和JCE框架实现DES算法**

DES算法加密的明文：Don't tell anybody!!

DES算法加密的密文： ����ɺH��ݣ�/f�S���BLH�

结果截图：



**2.使用JCA和JCE框架实现RSA算法**

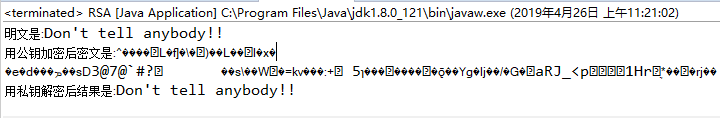
RSA算法加密的明文：Don't tell anybody!!

RSA算法用公钥加密的密文：^����\_L�f]�\�\_)��L�\_�\_I�x�

�e�d���ܡ��sD3@7@`#?� ��s\��W\_�=kv�\_��:+\_ 5ɿ���\_����\_�ǭ��Yg�Ij��/�G�\_aRJ\_<p����1Hr�֛\*��\_�rj��

RSA算法用私钥解密的结果：Don't tell anybody!!

结果截图：



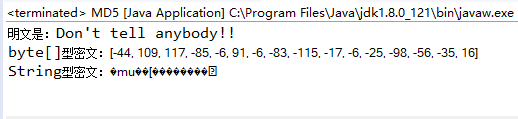
**3.使用JCA和JCE框架实现MD5算法**

MD5算法加密的明文：Don't tell anybody!!

MD5算法加密的byte[]型密文：[-44, 109, 117, -85, -6, 91, -6, -83, -115, -17, -6, -25, -98, -56, -35, 16]

MD5算法加密的String型密文：�mu��[��������\_

结果截图：

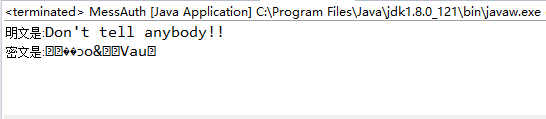


**4.使用JCA和JCE框架实现消息验证码算法**

消息验证码算法加密的明文：Don't tell anybody!!

消息验证码算法加密的密文：\_\_��כo&��V\_au�

结果截图：



1. 实验部分代码
2. **使用JCA和JCE框架实现DES算法**

|  |
| --- |
| **import** java.security.NoSuchAlgorithmException;  **import** java.security.Security;  **import** javax.crypto.Cipher;  **import** javax.crypto.KeyGenerator;  **import** javax.crypto.NoSuchPaddingException;  **import** javax.crypto.SecretKey;  **public** **class** DES {  **private** KeyGenerator keygen; // 提供对称密钥生成器的功能，支持各种算法  **private** SecretKey deskey;  **private** Cipher c;  **private** **byte**[] cipherByte;  **public** DES() {  Security.*addProvider*(**new** com.sun.crypto.provider.SunJCE());  **try** {  // 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)  keygen = KeyGenerator.*getInstance*("DES");  // 生成密钥  deskey = keygen.generateKey(); // 生成Cipher对象，指定其支持DES算法  c = Cipher.*getInstance*("DES");  }  **catch** (NoSuchAlgorithmException ex) {  ex.printStackTrace();  }  **catch** (NoSuchPaddingException ex) {  ex.printStackTrace();  }  }  /\* 对字符串str加密 \*/  **public** **byte**[] createEncryptor(String str) {    **try** {  // 根据密钥，对Cipher对象进行初始化,ENCRYPT\_MODE表示加密模式  c.init(Cipher.***ENCRYPT\_MODE***, deskey);  **byte**[] src = str.getBytes(); // 加密，结果保存进cipherByte  cipherByte = c.doFinal(src);  }  **catch** (java.security.InvalidKeyException ex) {  ex.printStackTrace();  }  **catch** (javax.crypto.BadPaddingException ex) {  ex.printStackTrace();  }  **catch** (javax.crypto.IllegalBlockSizeException ex) {  ex.printStackTrace();  }  **return** cipherByte;  }  /\* 对字节数组buff解密 \*/  **public** **byte**[] createDecryptor(**byte**[] buff) {  **try** {  // 根据密钥，对Cipher对象进行初始化,ENCRYPT\_MODE表示解密模式  c.init(Cipher.***DECRYPT\_MODE***, deskey); // 得到明文，存入cipherByte字符数组  cipherByte = c.doFinal(buff);  }  **catch** (java.security.InvalidKeyException ex) {  ex.printStackTrace();  }  **catch** (javax.crypto.BadPaddingException ex) {  ex.printStackTrace();  }  **catch** (javax.crypto.IllegalBlockSizeException ex) {  ex.printStackTrace();  }  **return** cipherByte;  }  **public** **static** **void** main(String[] args) **throws** Exception {  DES des = **new** DES();  String msg = "Don't tell anybody!!";  System.***out***.println("明文是：" + msg);  **byte**[] enc = des.createEncryptor(msg);  System.***out***.println("密文是：" + **new** String(enc));  **byte**[] dec = des.createDecryptor(enc);  System.***out***.println("解密后的结果是：" + **new** String(dec));  }  } |

1. **使用JCA和JCE框架实现RSA算法**

|  |
| --- |
| **import** java.security.KeyPair;  **import** java.security.KeyPairGenerator;  **import** java.security.interfaces.RSAPrivateKey;  **import** java.security.interfaces.RSAPublicKey;  **import** javax.crypto.Cipher;  **public** **class** RSA {  //RSA加密解密  **public** **static** **void** main(String[] args) {  **try** {  RSA p12\_04 = **new** RSA();  String msg = "Don't tell anybody!!";  System.***out***.println("明文是:" + msg);  //KeyPairGenerator 类用于生成公钥和私钥对，基于RSA算法生成对象  KeyPairGenerator keyPairGen = KeyPairGenerator.*getInstance*("RSA");  //初始化密钥对生成器,密钥大小为1024位  keyPairGen.initialize(1024);  //生成一个密钥对，保存在keyPair中  KeyPair keyPair = keyPairGen.generateKeyPair();  // 得到私钥  RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();  //得到公钥  RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();  //用公钥加密  **byte**[] srcBytes = msg.getBytes();  **byte**[] resultBytes = p12\_04.encrypt(publicKey, srcBytes);  String result = **new** String(resultBytes);  System.***out***.println("用公钥加密后密文是:" + result);  //用私钥解密  **byte**[] decBytes = p12\_04.decrypt(privateKey,resultBytes);  String dec = **new** String(decBytes);  System.***out***.println("用私钥解密后结果是:" + dec);  } **catch** (Exception e) {  e.printStackTrace();  }  }    **protected** **byte**[] encrypt(RSAPublicKey publicKey, **byte**[] srcBytes) {  **if** (publicKey != **null**) {  **try** {  //Cipher负责完成加密或解密工作，基于RSA  Cipher cipher = Cipher.*getInstance*("RSA");  //根据公钥，对Cipher对象进行初始化  cipher.init(Cipher.***ENCRYPT\_MODE***, publicKey);  // 加密，结果保存进resultBytes  **byte**[] resultBytes = cipher.doFinal(srcBytes);  **return** resultBytes;  } **catch** (Exception e) {  e.printStackTrace();  }  }  **return** **null**;  }    **protected** **byte**[] decrypt(RSAPrivateKey privateKey, **byte**[] encBytes) {  **if** (privateKey != **null**) {  **try** {  Cipher cipher = Cipher.*getInstance*("RSA");  //根据私钥，对Cipher对象进行初始化  cipher.init(Cipher.***DECRYPT\_MODE***, privateKey);  //解密，结果保存进resultBytes  **byte**[] decBytes = cipher.doFinal(encBytes);  **return** decBytes;  } **catch** (Exception e) {  e.printStackTrace();  }  }  **return** **null**;  }  } |

1. **使用JCA和JCE框架实现MD5算法**

|  |
| --- |
| **import** java.security.MessageDigest;  **import** java.security.NoSuchAlgorithmException;  **import** java.util.Arrays;  **public** **class** MD5 {  // MD5加密  **public** **byte**[] encrypt(String msg) {  **try** {  //根据MD5算法生成MessageDigest对象  MessageDigest md5 = MessageDigest.*getInstance*("MD5");  **byte**[] srcBytes = msg.getBytes();  //使用srcBytes更新摘要  md5.update(srcBytes);  //完成哈希计算,得到result  **byte**[] resultBytes= md5.digest();  **return** resultBytes;  } **catch**(NoSuchAlgorithmException e) {  e.printStackTrace();  }  **return** **null**;  }    **public** **static** **void** main(String[] args) {  String msg = "Don't tell anybody!!";  System.***out***.println("明文是：" + msg);  MD5 md5 = **new** MD5();  **byte**[] resultBytes = md5.encrypt(msg);  String result = **new** String(resultBytes);  System.***out***.println("byte[]型密文：" + Arrays.*toString*(resultBytes));  System.***out***.println("String型密文：" + result);  }  } |

**4.使用JCA和JCE框架实现消息验证码算法**

|  |
| --- |
| **import** javax.crypto.KeyGenerator;  **import** javax.crypto.Mac;  **import** javax.crypto.SecretKey;  **import** javax.crypto.spec.SecretKeySpec;  **public** **class** MessAuth {  **public** **static** **void** main(String[] args) {  //要计算消息验证码的字符串  String str="Don't tell anybody!!";  System.***out***.println("明文是:" + str);  **try** {  //用DES算法得到计算验证码的密钥  KeyGenerator keyGen=KeyGenerator.*getInstance*("DESede");  SecretKey key=keyGen.generateKey();  **byte**[] keyByte=key.getEncoded();  //生成MAC对象  SecretKeySpec SKS=**new** SecretKeySpec(keyByte,"HMACMD5");  Mac mac=Mac.*getInstance*("HMACMD5");  mac.init(SKS);  //传入要计算验证码的字符串  mac.update(str.getBytes("UTF8"));  //计算验证码  **byte**[] certifyCode=mac.doFinal();  System.***out***.println("密文是:" + **new** String(certifyCode));  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |