

ChinaSEI系列讲义(By郭克华)

Java 加密解密方法大全

ChinaSEI 软工学苑
.com

如果有文字等小错，请多包涵。在不盈利的情况下，欢迎免费传播。

版权所有:郭克华

本讲义经过修正、扩充，由清华大学出版社出版。

详细可查询 <http://www.china-pub.com/51834>

http://product.dangdang.com/product.aspx?product_id=20862469

【1】加密概述

【1-1】加密的应用

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。数据加密技术已经广泛应用于因特网电子商务、手机网络和银行自动取款机等领域。加密系统中有如下重要概念：

- 1：明文。被隐蔽的消息称作明文（plaintext）。
- 2：密文。隐蔽后的消息称作密文（ciphertext）。
- 3：加密。将明文变换成密文的过程称作加密(encryption)。
- 4：解密。由密文恢复出原明文的过程称作解密(decryption)。
- 5：敌方。主要指非授权者，通过各种办法，窃取机密信息。
- 6：被动攻击。获密文进行分析，这类攻击称作被动攻击(passive attack)。
- 7：主动攻击。非法入侵者(tamper)采用篡改、伪造等手段向系统注入假消息，称为主动攻击(active attack)。
- 8：加密算法。对明文进行加密时采用的算法。
- 9：解密算法。对密文进行解密时采用算法。
- 10：加密密钥和解密密钥。加密算法和解密算法的操作通常是在一组密钥（key）的控制下进行的，分别称为加密密钥(encryption key)和解密密钥(decryption key)。

在加密系统中，加密算法和密钥是最重要的两个概念。在这里需要对加密算法和密钥进行一个解释。以最简单的“恺撒加密法”为例。

《高卢战记》有描述恺撒曾经使用密码来传递信息，即所谓的“恺撒密码”。它是一种替代密码，通过将字母按顺序推后 3 位起到加密作用，如将字母 A 换作字母 D，将字母 B 换作字母 E。如“China”可以变为“Fklqd”；解密过程相反。

在这个简单的加密方法中，“向右移位”，可以理解为加密算法；“3”可以理解为加密密钥。对于解密过程，“向左移位”，可以理解为解密算法；“3”可以理解为解密密钥。显然，密钥是一种参数，它是在明文转换为密文或将密文转换为明文的算法中输入的数据。

恺撒加密法的安全性来源于两个方面：第一，对加密算法的隐藏；第二，对密钥的隐蔽。单单隐蔽加密算法以保护信息，在学界和业界已有相当讨论，一般认为是不够安全的。公开的加密算法是给黑客长年累月攻击测试，对比隐蔽的加密算法要安全多。一般说来，加密之所以安全，是因为其加密的密钥的隐藏，并非加密解密算法的保密。而流行的一些加密解密算法一般是完全公开的。敌方如果取得已加密的数据，即使得知加密算法，若没有密钥，也不能进行解密。

【1-2】常见的加密算法

加密技术从本质上说是对信息进行编码和解码的技术。加密是将可读信息（明文）变为代码形式（密文）；解密是加密的逆过程，相当于将密文变为明文。加密算法有很多种，一般可分为对称加密、非对称加密和单向加密三类算法。

对称加密算法应用较早，技术较为成熟。其过程如下：

- 1: 发送方将明文和加密密钥一起经过加密算法处理, 变成密文, 发送出去。
- 2: 接收方收到密文后, 使用加密密钥及相同算法的逆算法对密文解密, 恢复为明文。

在对称加密算法中, 双方使用的密钥相同, 要求解密方事先必须知道加密密钥。其特点是算法公开、计算量小、加密速度快、加密效率高。不足之处是, 通信双方都使用同样的密钥, 安全性得不到保证。此外, 用户每次使用该算法, 需要保证密钥的唯一性, 使得双方所拥有的密钥数量很大, 密钥管理较为困难。对称加密算法中, 目前流行的算法有: DES、3DES 和 IDEA 等, 美国国家标准局倡导的 AES 即将作为新标准取代 DES。

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)。每个人拥有这两个密钥, 公开密钥对外公开, 私有密钥不公开。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。

非对称加密算法的基本过程是:

- 1: 通信前, 接收方随机生成的公钥, 发送给发送方, 自己保留私钥。
- 2: 发送方利用接收方的公钥加密明文, 使其变为密文。
- 3: 接收方收到密文后, 使用自己的私钥解密密文。

广泛应用的非对称加密算法有 RSA 算法和美国国家标准局提出的 DSA。非对称加密算法的保密性比较好, 它消除了最终用户交换密钥的需要, 但加密和解密花费时间长、速度慢, 它不适合于对文件加密而只适用于对少量数据进行加密。

另一类是单向加密算法。该算法在加密过程中不需要使用密钥, 输入明文后由系统直接经过加密算法处理成密文, 密文无法解密。只有重新输入明文, 并经过同样的加密算法处理, 得到相同的密文并被系统重新识别后, 才能真正解密。该方法计算复杂, 通常只在数据量有限的情形下使用, 如广泛应用在计算机系统口令加密。近年来, 单向加密的应用领域正在逐渐增大。应用较多单向加密算法的有 RSA 公司发明的 MD5 算法和美国国家安全局(NSA) 设计, 美国国家标准与技术研究院(NIST) 发布 SHA (Secure Hash Algorithm, 安全散列算法)。

大多数语言体系 (如.net、Java) 都具有相关的 API 支持各种加密算法。本章以 Java 语言为例来阐述加密解密过程, 这些算法在其他语言中的实现, 读者可以参考相关资料。

【2】实现对称加密

如前所述, 对称加密算法过程中, 发送方将明文和加密密钥一起经过加密算法处理, 变成密文, 发送出去; 接收方收到密文后, 使用加密密钥及相同算法的逆算法对密文解密, 恢复为明文。双方使用的密钥相同, 要求解密方事先必须知道加密密钥。从这里可以得出几个结论:

- 1: 加密时使用什么密钥, 解密时必须使用相同密钥, 否则无法解密。
- 2: 对同样的信息, 不同的密钥, 加密结果和解密结果理论上不相同。

本节介绍三种流行的对称加密算法: DES、3DES 和 AES。

【2-1】用Java实现DES

DES 是数据加密标准 (Data Encryption Standard) 的简称, 出自 IBM 的研究工作,

并在 1977 年被美国政府正式采纳。它是使用较为广泛的密钥系统，最初开发 DES 是嵌入硬件中，DES 特别是在保护金融数据的安全，如自动取款机中，使用较多。

在 DES 中，使用了一个 56 位的密钥以及附加的 8 位奇偶校验位，产生最大 64 位的分组大小。加密过程中，将加密的文本块分成两半。使用子密钥对其中一半应用循环功能，然后将输出与另一半进行“异或”运算；接着交换这两半。循环往复。DES 使用 16 个循环，但最后一个循环不交换。

攻击 DES，一般只能使用穷举的密钥搜索，即重复尝试各种密钥直到有一个符合为止。如果 DES 使用 56 位的密钥，则可能的密钥数量是 2^{56} 个，穷举难度较大。IBM 曾对 DES 拥有几年的专利权，但在 1983 年到期。

关于 DES 的其他信息，可以参考相关资料。

在对称加密中，解密和加密的密钥一定要相同。以下代码是用 Java 语言实现将一个字符串“郭克华_安全编程技术”先加密，然后用同样的密钥解密的过程。不过，由于本书不是讲解某种语言本身，所以在这里略过 Java 加密体系的讲解，在代码中如果出现新的 API，读者可以参考 Java 文档。

P12_01.java

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import java.security.NoSuchAlgorithmException;
import java.security.Security;

public class P12_01
{
    //KeyGenerator提供对称密钥生成器的功能，支持各种算法
    private KeyGenerator keygen;
    //SecretKey负责保存对称密钥
    private SecretKey deskey;
    //Cipher负责完成加密或解密工作
    private Cipher c;
    //该字节数组负责保存加密的结果
    private byte[] cipherByte;

    public P12_01()
    {
        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*/
```

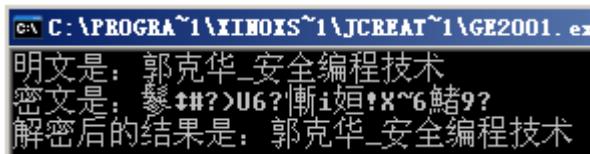
```

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
{
    P12_01 p12_01 = new P12_01();
    String msg = "郭克华_安全编程技术";
    System.out.println("明文是: " + msg);
    byte[] enc = p12_01.createEncryptor(msg);
    System.out.println("密文是: " + new String(enc));
    byte[] dec = p12_01.createDecryptor(enc);
    System.out.println("解密后的结果是: " + new String(dec));
}
}

```

运行，界面如下：



在不同的情况下，密文的内容会不一样。因为 KeyGenerator 每次生成的密钥是随机的，

这很容易理解，否则 DES 算法就没有安全性可言了。

【2-2】用Java实现 3DES

3DES，即三重 DES，是 DES 的加强版，也是 DES 的一个更安全的变形。它使用 3 条 56 位（共 168 位）的密钥对数据进行三次加密，一般情况下，提供了较为强大的安全性。实际上，3DES 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。

3DES 以 DES 为基本模块，通过组合分组方法设计出分组加密算法。令 $E_k()$ 和 $D_k()$ 表示 DES 算法的加密和解密过程， K 表示 DES 算法使用的密钥， P 表示明文， C 表示密文。3DES 的具体实现过程如下：

1：加密过程： $C = E_{k3}(D_{k2}(E_{k1}(P)))$

2：解密过程为： $P = D_{k1}((E_{k2}(D_{k3}(C)))$

从上述过程可以看出， K_1 、 K_2 、 K_3 决定了算法的安全性。若三个密钥互不相同，本质上就相当于用一个长为 168 位的密钥进行加密。若数据对安全性要求不高， K_1 可等于 K_3 。在这种情况下，密钥的有效长度为 112 位。

在 Java 的加密体系中，使用 3DES 非常简单，程序结构和使用 DES 时相同，只不过在初始化时将算法名称由“DES”改为“DESede”即可。

下列程序基本原理和 P12_01 相同，只不过将加密和解密过程写在一起。

P12_02.java

```
import java.security.Security;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class P12_02
{
    public static void main(String[] args) throws Exception
    {
        //KeyGenerator提供对称密钥生成器的功能，支持各种算法
        KeyGenerator keygen;
        //SecretKey负责保存对称密钥
        SecretKey deskey;
        //Cipher负责完成加密或解密工作
        Cipher c;
        Security.addProvider(new com.sun.crypto.provider.SunJCE());
        //实例化支持3DES算法的密钥生成器，算法名称用DESede
        keygen = KeyGenerator.getInstance("DESede");
        //生成密钥
        deskey = keygen.generateKey();
```

```

//生成Cipher对象, 指定其支持3DES算法
c = Cipher.getInstance("DESede");

String msg = "郭克华_安全编程技术";
System.out.println("明文是: " + msg);

//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 = msg.getBytes();
//加密, 结果保存进enc
byte[] enc = c.doFinal(src);
System.out.println("密文是: " + new String(enc));

//根据密钥, 对Cipher对象进行初始化,ENCRYPT_MODE表示加密模式
c.init(Cipher.DECRYPT_MODE, deskey);
//解密, 结果保存进dec
byte[] dec = c.doFinal(enc);
System.out.println("解密后的结果是: " + new String(dec));
}
}

```

运行, 效果如下:



〔2-3〕 用Java实现AES

AES 在密码学中是高级加密标准 (Advanced Encryption Standard) 的缩写, 该算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。最近, 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。

AES 算法又称 Rijndael 加密法, 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 命名。AES 是美国国家标准技术研究所 NIST 旨在取代 DES 的 21 世纪的加密标准。

AES 算法将成为美国新的数据加密标准而被广泛应用在各个领域中。尽管人们对 AES 还有不同的看法, 但总体来说, AES 作为新一代的数据加密标准汇聚了强安全性、高性能、高效率、易用和灵活等优点。AES 设计有三个密钥长度: 128, 192, 256 位, 相对而言, AES 的 128 密钥比 DES 的 56 密钥强得多。AES 算法主要包括三个方面: 轮变化、圈数和密钥扩展。关于其具体实现, 读者可以参考密码学书籍。

以下代码是用 Java 语言实现 AES 算法, 将一个字符串“郭克华_安全编程技术”先加密, 然后用同样的密钥解密。在代码中如果出现新的 API, 读者可以参考 Java 文档。

P12_03.java

```
import java.security.Security;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

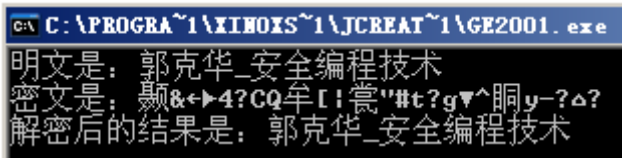
public class P12_03
{
    public static void main(String[] args) throws Exception
    {
        //KeyGenerator提供对称密钥生成器的功能，支持各种算法
        KeyGenerator keygen;
        //SecretKey负责保存对称密钥
        SecretKey deskey;
        //Cipher负责完成加密或解密工作
        Cipher c;
        Security.addProvider(new com.sun.crypto.provider.SunJCE());
        //实例化支持AES算法的密钥生成器，算法名称用AES
        keygen = KeyGenerator.getInstance("AES");
        //生成密钥
        deskey = keygen.generateKey();
        //生成Cipher对象，指定其支持AES算法
        c = Cipher.getInstance("AES");

        String msg = "郭克华_安全编程技术";
        System.out.println("明文是： " + msg);

        //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 = msg.getBytes();
        //加密，结果保存进enc
        byte[] enc = c.doFinal(src);
        System.out.println("密文是： " + new String(enc));

        //根据密钥，对Cipher对象进行初始化,ENCRYPT_MODE表示加密模式
        c.init(Cipher.DECRYPT_MODE, deskey);
        //解密，结果保存进dec
        byte[] dec = c.doFinal(enc);
        System.out.println("解密后的结果是： " + new String(dec));
    }
}
```

}
运行，效果如下：



【3】实现非对称加密

在非对称加密算法过程中，接收方产生一个公开密钥和一个私有密钥，前者公开。发送方将明文用接收方的公开密钥进行处理，变成密文，发送出去；接收方收到密文后，使用自己的私有密钥对密文解密，恢复为明文。在这种通信过程中，密钥由接收方产生，公开密钥公开，私有密钥保密。这里面有几个特点：

- 1：加密时使用的公开密钥，解密时必须使用对应的私有密钥，否则无法解密。
- 2：对同样的信息，可以用私有密钥加密，用公开密钥解密；也可以用公开密钥加密，用私有密钥解密。但是在应付窃听上，后者用得较多。

本节介绍 2 种流行的非对称加密算法：RSA 和 DSA。

【3-1】用Java实现RSA

RSA算法出现于 20 世纪 70 年代年，它是第一个既能用于数据加密也能用于数字签名的算法。它易于理解 and 操作，也很流行。算法的名字以发明者的名字命名：Ron Rivest、AdiShamir 和Leonard Adleman。不过，RSA的安全性一直未能得到理论上的证明。RSA是被研究得最广泛的公钥算法，从提出到现在已近二十年，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的公钥方案之一。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。

RSA 的缺点主要有：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：分组长度太大，为保证安全性，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。

RSA 的安全性依赖于大数分解。公钥和私钥都是两个大素数的函数。据猜测，从一个密钥和密文推断出明文的难度等同于分解两个大素数的积。

关于 RSA 算法的描述，读者可以参考相关文献。RSA 可用于数字签名，具体操作时考虑到安全性和信息量较大等因素，一般可以先作 HASH 运算。

如前所述，RSA 的安全性依赖于大数分解，但是否等同于大数分解一直未能得到理论上的证明，因为没有证明破解 RSA 就一定需要作大数分解。由于进行的都是大数计算，使得 RSA 最快的情况也比 DES 慢很多倍，无论是软件还是硬件实现。速度一直是 RSA 的缺陷。一般来说只用于少量数据加密。

以下代码是用 Java 语言实现 RSA 算法，将一个字符串“郭克华_安全编程技术”先加密，然后用同样的密钥解密。在代码中如果出现新的 API，读者可以参考 Java 文档。

P12_04.java

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import javax.crypto.Cipher;

public class P12_04
{
    //RSA加密解密
    public static void main(String[] args)
    {
        try
        {
            P12_04 p12_04 = new P12_04();
            String msg = "郭克华_安全编程技术";
            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;
}

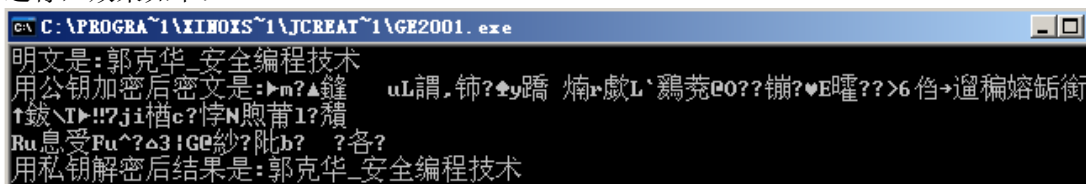
protected byte[] decrypt(RSAPrivateKey privateKey, byte[] encBytes)
{
    if (privateKey != null)
    {
        try
        {
            Cipher cipher = Cipher.getInstance("RSA");
            //根据私钥，对Cipher对象进行初始化
            cipher.init(Cipher.DECRYPT_MODE, privateKey);
            //解密，结果保存进resultBytes
            byte[] decBytes = cipher.doFinal(encBytes);
            return decBytes;
        }
        catch (Exception e)
```

```

    {
        e.printStackTrace();
    }
}
return null;
}
}
}

```

运行，效果如下：



```

C:\PROGRA~1\XINHOIS~1\JCREAT~1\GE2001.exe
明文是:郭克华_安全编程技术
用公钥加密后密文是:
用私钥解密后结果是:郭克华_安全编程技术

```

【3-2】DSA算法

数字签名算法(Digital Signature Algorithm ,DSA)，也是一种非对称加密算法，被美国 NIST 作为数字签名标准(DigitalSignature Standard, DSS)。DSA 是基于整数有限域离散对数难题的，其安全性与 RSA 相比差不多，其原理和 RSA 类似。DSA 一般应用于数字签名中，在后面的章节将会讲解。

【4】实现单向加密

单向加密算法，又称为不可逆加密算法，在加密过程中不需要使用密钥，明文由系统加密处理成密文，密文无法解密。一般适合于验证，在验证过程中，重新输入明文，并经过同样的加密算法处理，得到相同的密文并被系统重新验证。该方法计算复杂，通常只在数据量有限的情形下使用，如广泛应用在计算机系统口令加密。该算法有如下特点：

- 1：加密算法对同一消息反复执行该函数总得到相同的密文。
- 2：加密算法生成的密文是不可预见的，密文看起来和明文没有任何关系。
- 3：明文的任何微小变化都会对生成的密文产生很大的影响。
- 4：具有不可逆性。即通过密文要得到明文，理论上是不可行的。

本节介绍 2 种流行的单向加密算法：MD5 和 SHA。

【4-1】用Java实现MD5

MD5 的全称是Message-digest Algorithm 5（信息-摘要算法），用于确保信息传输完整一致。在 90 年代初由MIT Laboratory for Computer Science和RSA Data Security Inc,的Ronald L. Rivest开发出来，经MD2 和MD4 发展而来。它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大整数）。不管是MD2、MD3、MD4 还是MD5，它们都需要获得一个随机长度的信息并产生一个 128 位的信息摘要。

MD5 最广泛被用于各种软件的密码认证和钥匙识别上，比如软件的序列号，

20 世纪 90 年代, Rivest在已有算法基础上, 开发出技术上更为趋近成熟的MD5 算法。虽然MD5 比MD4 稍微慢一些, 但却更为安全。MD5 用的是哈希函数, 其典型应用是对一段信息产生信息摘要, 以防止被篡改。比如, 在UNIX下有很多软件在下载的时候都有一个文件名相同, 文件扩展名为.md5 的文件, 在这个文件中通常只有一行文本, 大致结构如:

MD5 (tanajiya.tar.gz) = 0ca175b9c0f726a831d895e269332461

这就是 tanajiya.tar.gz 文件的数字签名。MD5 将整个文件当作一个大文本信息, 通过其单向的字符串变换算法, 产生了这个唯一的 MD5 信息摘要。如同地球上任何人都有自己独一无二的指纹, 与之类似, MD5 可以为任何文件(不管其大小、格式、数量)产生一个同样独一无二的“数字指纹”, 如果任何人对文件做了任何改动, 其 MD5 值也就是对应的“数字指纹”都会发生变化。

我们常常在某些软件下载站点的某软件信息中看到其 MD5 值, 它的作用就在于我们可以在下载该软件后, 对下载回来的文件用专门的软件(如 Windows MD5 Check 等)做一次 MD5 校验, 以确保我们获得的文件与该站点提供的文件为同一文件。利用 MD5 算法来进行文件校验的方案被大量应用到软件下载站、论坛数据库、系统文件安全等方面。

举个例子, 你将一段话写在一个叫 readme.txt 文件中, 并对这个 readme.txt 产生一个 MD5 的值并记录在案, 然后你可以传播这个文件给别人, 别人如果修改了文件中的任何内容, 你对这个文件重新计算 MD5 时就会发现(两个 MD5 值不相同)。如果再有一个第三方的认证机构, 用 MD5 还可以防止文件作者的“抵赖”, 这就是所谓的数字签名应用。

MD5 还广泛用于操作系统的登陆认证上, 如 Unix、各类 BSD 系统登录密码、数字签名等诸多方。如在 UNIX 系统中用户的密码是以 MD5 (或其它类似的算法)经 Hash 运算后存储在文件系统中。当用户登录的时候, 系统把用户输入的密码进行 MD5 Hash 运算, 然后再去和保存在文件系统中的 MD5 值进行比较, 进而确定输入的密码是否正确。通过这样的步骤, 系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。这可以避免用户的密码被具有系统管理员权限的用户知道。MD5 将任意长度的“字节串”映射为一个 128bit 的大整数, 并且是通过该 128bit 反推原始字符串是困难的, 换句话说就是, 即使你看到源程序和算法描述, 也无法将一个 MD5 的值变换回原始的字符串, 从数学原理上说, 是因为原始的字符串有无穷多个, 这有点象不存在反函数的数学函数。所以, 要遇到了 md5 密码的问题, 比较好的办法是: 你可以用这个系统中的 md5()函数重新设一个密码, 如 admin, 把生成的一串密码的 Hash 值覆盖原来的 Hash 值就行了。

MD5 算法简要的叙述为: MD5 以 512 位分组来处理输入的信息, 且每一分组又被划分为 16 个 32 位子分组, 经过了一系列的处理后, 算法的输出由四个 32 位分组组成, 将这四个 32 位分组级联后将生成一个 128 位散列值。相关信息大家可以参考相应文献。

2004 年 8 月 17 日的美国加州圣巴巴拉的国际密码学会议 (Crypto'2004) 上, 来自中国山东大学的王小云教授做了破译MD5、HAVAL-128、MD4 和RIPEMD算法的报告, 公布了MD系列算法的破解结果。破解MD5 之后, 2005 年 2 月, 王小云教授又破解了另一国际密码SHA-1。因为SHA-1 在美国等国际社会有更加广泛的应用, 密码被破的消息一出, 在国际社会的反响可谓石破天惊。换句话说, 王小云的研究成果表明了从理论上讲电子签名可以伪造, 必须及时添加限制条件, 或者重新选用更为安全的密码标准, 以保证电子商务的安全。

针对王小云教授等破译的以 MD5 为代表的 Hash 函数算法的报告, 美国国家技术与标

准局（NIST）于 2004 年 8 月 24 日发表专门评论，评论的主要内容为：“在最近的国际密码学会议（Crypto 2004）上，研究人员宣布他们发现了破解数种 HASH 算法的方法，其中包括 MD4, MD5, HAVAL-128, RIPEMD 还有 SHA-0。分析表明，于 1994 年替代 SHA-0 成为联邦信息处理标准的 SHA-1 的减弱条件的变种算法能够被破解；但完整的 SHA-1 并没有被破解，也没有找到 SHA-1 的碰撞。研究结果说明 SHA-1 的安全性暂时没有问题，但随着技术的发展，技术与标准局计划在 2010 年之前逐步淘汰 SHA-1，换用其他更长更安全的算法

以下代码是用 Java 语言实现 MD5 算法，将一个字符串“郭克华_安全编程技术”加密。在代码中如果出现新的 API，读者可以参考 Java 文档。

P12_07.java

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class P12_05
{
    // 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 = "郭克华_安全编程技术";
        System.out.println("明文是: " + msg);
    }
}
```

```

P12_05 p12_05 = new P12_05();
byte[] resultBytes = p12_05.encrypt(msg);
String result = new String(resultBytes);
System.out.println("密文是: " + result);
}
}

```

运行，效果如下：

反复运行，效果一样。

〔4-2〕用Java实现SHA

安全散列算法(Secure Hash Algorithm, SHA)是美国国家标准和技术局发布的国家标准 FIPS PUB 180-1，一般称为 SHA-1。其对长度不超过 264 二进制位的消息产生 160 位的消息摘要输出。

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，然后以一种单向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”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。

以下代码是用 Java 语言实现 SHA 算法，将一个字符串“郭克华_安全编程技术”进行加密。

P12_06.java

```

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class P12_06
{
    //SHA加密
    public static void main(String[] args) throws NoSuchAlgorithmException
    {
        try
        {
            String msg = "郭克华_安全编程技术";
            System.out.println("明文是: " + msg);
            MessageDigest md5 = MessageDigest.getInstance("SHA");

```



```

        byte[] srcBytes = msg.getBytes();
        md5.update(srcBytes);
        byte[] resultBytes= md5.digest();
        String result = new String(resultBytes);
        System.out.println("密文是: " + result);
    }
    catch(NoSuchAlgorithmException e)
    {
        e.printStackTrace();
    }
}
}

```

运行，效果如下：



反复运行，效果一样。

【4-3】用Java实现消息验证码

单向加密的结果也叫做消息摘要，可以较好地验证数据的完整性，因为不同的数据加密得到的结果不同。利用 MD5 算法生成消息摘要，可以验证数据是否被修改，但是，如果在数据传递过程中，窃取者将数据窃取出来，并且修改数据，再重新生成一次摘要，将改后的数据和重新计算的摘要发送给接收者，接收者仍然无法判断内容是否被修改过了，因为在验证阶段，根据接收的数据计算的摘要和窃取者修改数据之后计算出来的摘要是相同的。因此，为了确保安全性，有时需要知道发送者身份，消息验证码在一定程度上可以实现这个功能，以保证安全性。

消息验证码和 MD5/SHA1 算法不同的地方是：在生成摘要时，发送者和接收者都拥有一个共同的密钥。这个密钥可以是通过对称密码体系生成的，事先被双方共有，只有同样的密钥才能生成同样的消息验证码。Java 里面可以较好地完成这个功能。以下代码是用 Java 语言实现 HMAC/MD5 算法，将一个字符串“郭克华_安全编程技术”进行加密。

P12_07.java

```

import javax.crypto.KeyGenerator;
import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

public class P12_07
{
    public static void main(String[] args)

```

```

{
    //要计算消息验证码的字符串
    String str="郭克华_安全编程技术";
    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();
    }
}
}

```

运行，效果如下：



该程序反复运行，效果不一样，因为每次生成的 DES 密钥不一样。

在实际操作的过程中，DES 密钥是保存在文件中或者数据库中，然后从其中取出，这样保证得到的结果是一样的。如果敌方得不到密钥，则无法生成正确的消息验证码。

【练习】

- 1: 任写一个文本文件，将其内容用 DES、AES 方式加密然后解密。

2: 任写一个文本文件，将其内容用 **RSA** 方法加密然后解密。

3: 任写一个文本文件，将其内容用 **MD5** 算法进行加密，然后修改这个文本文件，再加密，比较两次的密文。

4: 编写一个“软件加密器”。打开一个 **Java** 界面，必须首先选择一个破解文件，如果能够找到正确的破解文件，该 **Java** 界面才能打开。