# 加密与解密

## 流密码

同步流密码

自同步流密码

## 分组密码

分组密码多应用于网络加密，是对此密码体质中发展最为完善的密码体制。

分组密码对**固定长度**的一组明文进行加密，这一固定长度称为**分组长度**。

分组长度是分组密码的一个参数，它与分组算法的安全性成正比，其取值范围取决于实际引用的环境。

为保证分组算法的安全性，分组长度越长越好，分组长度越长，密码解析越困难；

为保证分组算法的实用性，分组长度越短越好，分组长度越短，越便于操作和运算；

分组长度的设定需要权衡分组算方法的安全性与实用性，一般设置为56位。但随着密码学的发展，分组长度只有56位的分组密码已经不能确保算方法的安全性。目前。分组密码多选择128位作为算方法的分组长度。

### 加密过程

对一个 长度为 n 的 明文分组进行加密操作，相应地产生一个n位的密文分组，由此可见，不同的n位明晚分组共有2的n次方个。每一个不同的n位明文分组都应该产生一个唯一的密文分组，加密过程对应的变换成为**可逆变换**或**非奇异变换**。

代表性的分组加密算法有DES、AES等。

### 设计原则

1. 不可破译原则

理论上不可破译

实际不可破译

扩散原则

混乱原则

密码体质过于复杂导致对密码分析者来说是无法利用的

如何衡量一个密码的安全性？

密码体制的破译所需要的时间和费用超出了现有的资源和能力

密码体制的破译所需要的时间超过了该体制所保护信息的有效时间

密码体制的破译所需要的费用超过了该体制所保护信息的价值

1. 针对实现的设计原则

软件实现的设计原则

硬件实现的设计原则

### 分组密码的工作模式

以DES算法为例，DES算法根据其加密算法所定义的明文分组的大小（56位），将数据分割成若干56位的加密区块，再以加密区块为单位，分别进行加密处理。最后剩下的不足一个区块的大小，称为短块，短块的吃力方法有填充法，流密码加密法，密码挪用技术。

1. 电子密码本模式（ECB）
2. 密文链接模式（CBC）
3. 密文反馈模式（CFB）
4. 输出反馈模式（OFC）
5. 计数器模式（CTR）

## 对称加密算法

加密与解密使用同一个密钥。

发送消息

密钥加密消息

密文

接收消息

密钥加密消息

密文

网络传输

密钥

### 对称加密算法族谱简介

在对称加密算法中，DES算法最具有代表性，堪称规范。DESede是DES算法的变种；AES算法则作为DES算法的替代品；而IDEA算法作为一种强加密算法，成为电子邮件加密软件PGP(Pretty Good Privacy)的核心算法之一。

在Java实现层面上，DES、DESede、AES和IDEA这4种算法略有不同。

DES和DESede算法在使用密钥材料还原密钥时，建议使用各自相应的密钥材料实现类（xxxSpec）完成相应转换操作。

AES算法在使用密钥材料还原密钥时，则需要使用一般密钥材料实现类（SecretKeySpec类）完成相应转换操作。其他对称加密算法可参照该方式实现，如RC2、RC4......

IDEA算法实现JDK 7 为提供，可使用Bouncy Castle提供支持。

### 数据加密标准——DES

DES算法和DESede算法统称DES系列算法。诸多对称加密算法发展源于DES算法的研究。

1998年，实用化DES算法破译机的出现宣告DES算法已不具备安全性。以当今计算机技术能力，经DES算法加密的数据在24小时内可能被破解。由此，DES算法正式退出历史舞台。

但DES算法对密码学领域贡献巨大。各种对称加密算法均由研究DES算法发展而来。升值很多硬件芯片本身具备DES加密实现。

其地位堪比C语言在计算机语言中的地位。

作者在与合作公司在构建加密通信模块时，通常会指定这样便于书写的密钥，并将其书写在合同书上。

#### 代码示例

import static org.junit.Assert.assertEquals;

import java.security.InvalidKeyException;

import java.security.Key;

import java.security.NoSuchAlgorithmException;

import java.security.NoSuchProviderException;

import java.security.spec.InvalidKeySpecException;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.DESKeySpec;

import org.bouncycastle.util.encoders.Base64;

import org.junit.Test;

/\*\*

\* DES安全编码组件

\*

\* 注意,密钥生成算法和加密/解密算法很有可能采用不同算法。

\*/

public class DESCoder {

@Test

public void printBouncy() {

Provider provider = Security.getProvider("BC");

Set<Service> services = provider.getServices();

for (Service service : services) {

System.out.println(service.getClassName() + " " +service.getAlgorithm());

}

}

@Test

public void test() throws NoSuchAlgorithmException, NoSuchProviderException, InvalidKeyException, NoSuchPaddingException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException {

String string = "Hello world";

byte[] data = string.getBytes();

System.out.println("原文:\t" + string);

// 生成密钥

byte[] key = DESCoder.initKey();

System.out.println("密钥:\t" + Base64.toBase64String(key));

// 加密

byte[] encrypt = DESCoder.encrypt(data, key);

System.out.println("加密:\t" + Base64.toBase64String(encrypt));

// 解密

byte[] decrypt = DESCoder.decrypt(encrypt, key);

String output = new String(decrypt);

System.out.println("解密:\t" + output);

assertEquals(string, output);

}

public static final String KEY\_ALGORITHM = "DES";

/\*\*

\* 加密/解密算法/工作模式/填充方式

\*/

public static final String CIPHER\_ALGORITHM = "DES";

/\*\*

\* 转换密钥

\* @param key 二进制密钥

\* @return

\* @throws NoSuchAlgorithmException

\* @throws InvalidKeySpecException

\* @throws InvalidKeyException

\*/

public static Key toKey(byte[] key) throws NoSuchAlgorithmException, InvalidKeySpecException, InvalidKeyException {

// 实例化DES密钥材料

DESKeySpec dks = new DESKeySpec(key);

// 实例化秘密密钥工厂

SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(KEY\_ALGORITHM);

// 生成秘密密钥

SecretKey secretKey = keyFactory.generateSecret(dks);

return secretKey;

}

/\*\*

\* 解密

\* @param data

\* @param key

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidKeySpecException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public static byte[] decrypt(byte[] data, byte[] key) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException {

// 还原密钥

Key k = toKey(key);

Cipher cipher = Cipher.getInstance(CIPHER\_ALGORITHM);

// 初始化

cipher.init(Cipher.DECRYPT\_MODE, k);

// 执行操作

return cipher.doFinal(data);

}

/\*\*

\* 加密

\* @param data

\* @param key

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidKeySpecException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public static byte[] encrypt(byte[] data, byte[] key) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException {

// 还原密钥

Key k = toKey(key);

// 实例化

Cipher cipher = Cipher.getInstance(CIPHER\_ALGORITHM);

cipher.init(Cipher.ENCRYPT\_MODE, k);

return cipher.doFinal(data);

}

/\*\*

\* 初始化密钥

\* @return

\* @throws NoSuchProviderException

\* @throws NoSuchAlgorithmException

\*/

public static byte[] initKey() throws NoSuchAlgorithmException, NoSuchProviderException {

/\*\*

\* 若要使用 64位密钥 则使用Bouncy Castle 密钥

\*/

KeyGenerator kg = KeyGenerator.getInstance(CIPHER\_ALGORITHM, "BC");

kg.init(64);

SecretKey key = kg.generateKey();

return key.getEncoded();

}

}

### 三重DES加密 —— DESede

DES算法改良。DESede算法对DES算法针对其密钥长度较短（DES三倍）和迭代次数较少等问题做了相应改进。但其处理速度较慢，密钥计算时间较长，加密效率较高。

### 高级数据加密标准——AES

由于DES算法漏洞，DESede算法低效，AES算法作为替代品应运而生。

其具有128位分组长度，支持128位、192位、256位。NIST又选中MARS、RC6、Rijndael、Serpent、Twofish作为候选算法。

目前采用AES算法能够有效抵御已知的针对DES算法的所有攻击方法，部份分差分攻击、相关密钥攻击等。至今AES破译的官方报到。

AES算法因密钥建立时间短、灵敏性好、内存需求低等优点，在各领域得到广泛的研究与应用。

其常用于UMTS(Uniersal Mobile Telecommunications System 通用移动系统)。基于SSH协议的软件也采用了AES算法。在一些无线路由器中也采用AES算法构建加密协议。

#### 示例

package org.hc.learning.algorithm.对称加密;

import static org.junit.Assert.assertEquals;

import java.security.InvalidKeyException;

import java.security.Key;

import java.security.NoSuchAlgorithmException;

import java.security.NoSuchProviderException;

import java.security.Provider;

import java.security.Security;

import java.security.Provider.Service;

import java.security.spec.InvalidKeySpecException;

import java.util.Set;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import org.bouncycastle.util.encoders.Base64;

import org.junit.Test;

public class AESCoder {

@Test

public void listService() {

Provider provider = Security.getProvider("BC");

Set<Service> services = provider.getServices();

for (Service service : services) {

// System.out.println(service.getClassName() + " " +service.getAlgorithm());

if (service.getAlgorithm().contains("AES")) {

System.out.print(service.getType());

System.out.print("\t");

}

System.out.println(service.getAlgorithm());

}

}

@Test

public void test() throws NoSuchAlgorithmException, NoSuchProviderException, InvalidKeyException, NoSuchPaddingException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException {

String string = "Hello world";

byte[] data = string.getBytes();

System.out.println("原文:\t" + string);

// 生成密钥

byte[] key = AESCoder.initKey();

System.out.println("密钥:\t" + Base64.toBase64String(key));

// 加密

byte[] encrypt = AESCoder.encrypt(data, key);

System.out.println("加密:\t" + Base64.toBase64String(encrypt));

// 解密

byte[] decrypt = AESCoder.decrypt(encrypt, key);

String output = new String(decrypt);

System.out.println("解密:\t" + output);

assertEquals(string, output);

}

public static final String KEY\_ALGORITHM = "AES";

/\*\*

\* 加密/解密算法/工作模式/填充方式

\*/

/\*public static final String CIPHER\_ALGORITHM = "DES/ECB/PKCS5Padding";\*/

public static final String CIPHER\_ALGORITHM = "AES";

/\*\*

\* 转换密钥

\* @param key 二进制密钥

\* @return

\* @throws NoSuchAlgorithmException

\* @throws InvalidKeySpecException

\* @throws InvalidKeyException

\*/

public static Key toKey(byte[] key) throws NoSuchAlgorithmException, InvalidKeySpecException, InvalidKeyException {

SecretKey secretKey = new SecretKeySpec(key, KEY\_ALGORITHM);

return secretKey;

}

/\*\*

\* 解密

\* @param data

\* @param key

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidKeySpecException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\* @throws NoSuchProviderException

\*/

public static byte[] decrypt(byte[] data, byte[] key) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException, NoSuchProviderException {

// 还原密钥

Key k = toKey(key);

Cipher cipher = Cipher.getInstance(CIPHER\_ALGORITHM, "BC");

// 初始化

cipher.init(Cipher.DECRYPT\_MODE, k);

// 执行操作

return cipher.doFinal(data);

}

/\*\*

\* 加密

\* @param data

\* @param key

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidKeySpecException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\* @throws NoSuchProviderException

\*/

public static byte[] encrypt(byte[] data, byte[] key) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException, NoSuchProviderException {

// 还原密钥

Key k = toKey(key);

// 实例化

Cipher cipher = Cipher.getInstance(CIPHER\_ALGORITHM, "BC");

cipher.init(Cipher.ENCRYPT\_MODE, k);

return cipher.doFinal(data);

}

/\*\*

\* 初始化密钥

\* @return

\* @throws NoSuchProviderException

\* @throws NoSuchAlgorithmException

\*/

public static byte[] initKey() throws NoSuchAlgorithmException, NoSuchProviderException {

/\*\*

\* 若要使用 64位密钥 则使用Bouncy Castle 密钥

\*/

KeyGenerator kg = KeyGenerator.getInstance(CIPHER\_ALGORITHM, "BC");

// 128 192

kg.init(128);

SecretKey key = kg.generateKey();

return key.getEncoded();

}

}

### 国际数据加密标准——IDEA

在NIST发布征集AES算法以前，就已经有人寻找DES算法的替代算法。IDEA的提出者并未在原有DES算法上进行改进，而是另辟蹊径寻求突破性解决方案。

IDEA算法使用长度为128的密钥，数据块大小为64位。理论上讲，IDEA属于“强加密”算法。至今未出现对该算法的有效攻击。

IDEA于美国之外提出，避开了美国法律对加密技术的诸多限制。

IDEA算法是目前较为常用的电子邮件加密算法之一。

#### 示例

public class IDEACoder {

@Test

public void listService() {

Provider provider = Security.getProvider("BC");

Set<Service> services = provider.getServices();

for (Service service : services) {

// System.out.println(service.getClassName() + " " +service.getAlgorithm());

if (service.getAlgorithm().contains("AES")) {

System.out.print(service.getType());

System.out.print("\t");

}

System.out.println(service.getAlgorithm());

}

}

@Test

public void test() throws NoSuchAlgorithmException, NoSuchProviderException, InvalidKeyException, NoSuchPaddingException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException {

String string = "Hello world";

byte[] data = string.getBytes();

System.out.println("原文:\t" + string);

// 生成密钥

byte[] key = IDEACoder.initKey();

System.out.println("密钥:\t" + Base64.toBase64String(key));

// 加密

byte[] encrypt = IDEACoder.encrypt(data, key);

System.out.println("加密:\t" + Base64.toBase64String(encrypt));

// 解密

byte[] decrypt = IDEACoder.decrypt(encrypt, key);

String output = new String(decrypt);

System.out.println("解密:\t" + output);

assertEquals(string, output);

}

public static final String KEY\_ALGORITHM = "IDEA";

/\*\*

\* 加密/解密算法/工作模式/填充方式

\*/

/\*public static final String CIPHER\_ALGORITHM = "DES/ECB/PKCS5Padding";\*/

public static final String CIPHER\_ALGORITHM = "IDEA";

/\*\*

\* 转换密钥

\* @param key 二进制密钥

\* @return

\* @throws NoSuchAlgorithmException

\* @throws InvalidKeySpecException

\* @throws InvalidKeyException

\*/

public static Key toKey(byte[] key) throws NoSuchAlgorithmException, InvalidKeySpecException, InvalidKeyException {

SecretKey secretKey = new SecretKeySpec(key, KEY\_ALGORITHM);

return secretKey;

}

/\*\*

\* 解密

\* @param data

\* @param key

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidKeySpecException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\* @throws NoSuchProviderException

\*/

public static byte[] decrypt(byte[] data, byte[] key) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException, NoSuchProviderException {

// 还原密钥

Key k = toKey(key);

Cipher cipher = Cipher.getInstance(CIPHER\_ALGORITHM, "BC");

// 初始化

cipher.init(Cipher.DECRYPT\_MODE, k);

// 执行操作

return cipher.doFinal(data);

}

/\*\*

\* 加密

\* @param data

\* @param key

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidKeySpecException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\* @throws NoSuchProviderException

\*/

public static byte[] encrypt(byte[] data, byte[] key) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException, BadPaddingException, NoSuchProviderException {

// 还原密钥

Key k = toKey(key);

// 实例化

Cipher cipher = Cipher.getInstance(CIPHER\_ALGORITHM,"BC");

cipher.init(Cipher.ENCRYPT\_MODE, k);

return cipher.doFinal(data);

}

/\*\*

\* 初始化密钥

\* @return

\* @throws NoSuchProviderException

\* @throws NoSuchAlgorithmException

\*/

public static byte[] initKey() throws NoSuchAlgorithmException, NoSuchProviderException {

/\*\*

\* 若要使用 64位密钥 则使用Bouncy Castle 密钥

\*/

KeyGenerator kg = KeyGenerator.getInstance(CIPHER\_ALGORITHM, "BC");

// 64 128

kg.init(128);

SecretKey key = kg.generateKey();

return key.getEncoded();

}

}

### 基于口令加密——PBE

PBE(Password Based Encryption 基于口令加密) 算法是一种基于口令的加密算法，其特点在于口令由用户自己掌管，采用随机数（盐）杂凑多重加密等方法保证数据的安全性。

PEC算法没有密钥的概念，密钥在其他对称加密算法中是经过算法计算得出的，PBE算法中则用口令替代了密钥。

密钥长短影响算法安全性，但不方便记忆。即便是我们将Base64编码转换为可见字符，长密钥一样不容易记忆。因此，在这情况下密钥是需要存储的，口令则不然。而口令是便于记忆的一种凭证。

PBE算法并没有构建真正的加密/解密算法，而是对我们已知的对称加密算法进行了包装。其实使用PBE算法对数据加密/解密操作时，其实是使用了DES或AES等对称加密算法做了相应操作。

既然PBE算法使用了我们较为常用的对称加密算法，那就无法回避密钥的问题。口令并不能替代密钥，密钥是经过加密算法算计出来的，但口令本身不能很长。单纯的口令很容易通过穷举攻击方式破译，这就引入了“盐”。“盐”能够阻止字典攻击或预先计算的攻击，它本身是一个随机信息，相同的随即信息极不可能使用两次。将盐附加在口令上，通过消息摘要算法经过迭代计算获得构建密钥/初始化向量的基本材料，使得破译加密信息的难度较大。

#### 模型分析

甲乙双方作为消息传递放，甲作为发送方，乙作为接收方。假设甲乙双方已经完成消息传递前已商定的加密算法和迭代次数。

1. 由消息传递双方约定口令，这里由甲方构建口令。
2. 由口令构建者公布口令，这里由甲方将口令公布给乙方。
3. 由口令构建者构建本次消息传的使用的盐，这里由甲方构建盐。
4. 由消息发送方使用口令，盐对数据加密，这里由甲方对数据加密。
5. 由消息发送方将盐、加密数据发送给消息接收者，这里由甲方将盐、加密数据发送给乙方。
6. 由消息接收方使用盐、口令对加密数据解密，这里由乙方完成数据解密。

#### 示例

public class PBECoder {

@Test

public void listService() {

Provider provider = Security.getProvider("BC");

Set<Service> services = provider.getServices();

for (Service service : services) {

System.out.println(service.getClassName() + " " +service.getAlgorithm());

}

}

@Test

public void test() throws InvalidKeyException, NoSuchAlgorithmException, InvalidKeySpecException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidAlgorithmParameterException {

String data = "Helloworld";

System.out.println("原文 :" + data);

byte[] bytes = data.getBytes();

String password = "qwerty";

byte[] salt = PBECoder.initSalt();

System.out.println("盐 :" + Base64.toBase64String(salt));

// 加密

byte[] encrypt = PBECoder.encrypt(bytes, password, salt);

System.out.println("加密 :" + Base64.toBase64String(encrypt));

// 解密

byte[] decrypt = PBECoder.decrypt(encrypt, password, salt);

String output = new String(decrypt);

System.out.println("解密 :" + output);

assertEquals(data, output);

}

public static final String ALGORITHM = "PBEWITHMD5andDES";

public static final int ITERATION\_COUNT = 100;

public static byte[] initSalt() {

// 实例化安全随机数

SecureRandom random = new SecureRandom();

// 产出盐

return random.generateSeed(8);

}

/\*\*

\* 转换密钥

\* @param password

\* @return

\* @throws NoSuchAlgorithmException

\* @throws InvalidKeySpecException

\*/

public static Key toKey(String password) throws NoSuchAlgorithmException, InvalidKeySpecException {

// 密钥材料转换

PBEKeySpec keySpec = new PBEKeySpec(password.toCharArray());

// 实例化

SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(ALGORITHM);

// 生成密钥

SecretKey secretKey = keyFactory.generateSecret(keySpec);

return secretKey;

}

/\*\*

\* 加密

\* @param data

\* @param password

\* @param salt

\* @return

\* @throws InvalidKeySpecException

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws BadPaddingException

\* @throws IllegalBlockSizeException

\* @throws InvalidAlgorithmParameterException

\* @throws InvalidKeyException

\*/

public static byte[] encrypt(byte[] data, String password, byte[] salt) throws NoSuchAlgorithmException, InvalidKeySpecException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException, InvalidAlgorithmParameterException {

Key key = toKey(password);

PBEParameterSpec paramSpec = new PBEParameterSpec(salt, ITERATION\_COUNT);

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.ENCRYPT\_MODE, key, paramSpec);

return cipher.doFinal(data);

}

/\*\*

\* 解密

\* @param data

\* @param password

\* @param salt

\* @return

\* @throws NoSuchAlgorithmException

\* @throws InvalidKeySpecException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws InvalidAlgorithmParameterException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public static byte[] decrypt(byte[] data, String password, byte[] salt) throws NoSuchAlgorithmException, InvalidKeySpecException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException{

// 转换密钥

Key key = toKey(password);

PBEParameterSpec paramSpec = new PBEParameterSpec(salt, ITERATION\_COUNT);

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.DECRYPT\_MODE, key, paramSpec);

return cipher.doFinal(data);

}

}

## 非对称加密算法RSA

加密解密使用一对钥匙，称为公钥与私钥。而使用私钥加密的数据，只有对应的公约才能解密，而使用公钥加密的数据，只能使用对应私钥才能解密。

发送消息

接收消息

公钥 -> 加密算法

密文

密文

私钥 -> 解密算法

由于 非对称加密算法 效率较低，故可以通过 非对称加密算法 传送 对称加密算法 的密钥。

详细应用请看 网络协议

# Base64

最初用于电子邮件系统，因为早期的电子邮件由于历史原因至允许传输ASCII码，若传输非ASCII码邮件用户将收到一堆乱码，后演变为显示传递URL参数的一种编码方法，时称“Url Base64”

20180918

挺容易理解的算法，主要用于处理非常见字符于互联网上传递。实现可以参考本人git\_houcheng\code\Java\hc\hc-learning\src\main\java\org\hc\learning\algorithm\base64下代码。

## Java实现

private final static String mapper = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

/\*\*

\* 实现思路:

\* 先将 数组data 中所以数据复制到 clone 并且补充足够的长度给 clone

\* 然后对 clone 中的数据进行加密

\* @param data

\* @return

\*/

public static String encode(byte[] data){

// 声明

int len = data.length / 3 \* 4 ;

int mod = data.length % 3 ; // 余数

if( mod %3 != 0){

len += 4;

}

int lenBytes = len/ 4 \* 3;

byte[] clone = new byte[ lenBytes ];

char[] chars = new char[ len ];

// 复制

for( int i=0 ; i<lenBytes ; i++ ){

if( i < data.length ){

clone[i] = data[i];

}else{

clone[i] = 0 ;

}

}

// 加密核心

int count = 0 ;

for( int i=0 ; i<clone.length ; i+=3 ){

chars[count++] = mapper.charAt( ( clone[i] >>2 ) % 64 );

chars[count++] = mapper.charAt( ( clone[i] <<6 >>2 ) %64 + ( clone[i+1] >>4 ) %64 ); // 不能将 %64提取,移位操作非常规运算

chars[count++] = mapper.charAt( ( clone[i+1] <<4 >>2 ) %64 + ( clone[i+2] >>6 ) %64 );

chars[count++] = mapper.charAt( ( clone[i+2] <<2 >>2 ) %64 );

}

return mod != 0 ? new String(chars, 0 , chars.length- ( 3 - mod)) : new String(chars) ;

}

public static String decode( byte[] data ){

char c ;

byte[] ori = new byte[ data.length + 2 ]; // 原数据

char[] chars = new char[ (data.length +2 ) /3 \*4 ];

for ( int i=0 ; i<ori.length ; i++ ){ // 将加密数据 首先解析 为 对应表中的数字信息

if ( i < data.length ){

c = (char) data[i];

ori[i] = (byte) mapper.indexOf( c );

}else{

ori[i] = 0;

}

}

// 在通过这些数字信息 解析 为 原信息

int count = 0 ;

for( int i=0 ; i<ori.length ; i+=4){

chars[count++] = (char) ( (ori[i] \*4 ) + ( ori[i+1] \*4 /64 ) ) ;

chars[count++] = (char) ( (ori[i+1] %8 \* 16 ) + ( ori[i+2] \*4 /16 ) );

chars[count++] = (char) ( (ori[i+2] %2 \* 64 ) + ( ori[i+3] ) );

}

return new String( chars );

}

public static void main(String[] args) {

byte[] data = "Helloworld111".getBytes();

byte[] encode = Base64.encode( data );

for( byte b : data ){

System.out.print( b + " " );

}

System.out.println();

System.out.println( new String(encode) );

String dataEncode = encode( data );

System.out.println( dataEncode );

String decode = decode( dataEncode.getBytes() );

System.out.println( decode );

}

# 散列函数

又名哈希函数、消息摘要函数、单向函数、杂凑函数。

散列函数主要用于验证数据完整性，是 数据签名 的核心技术。散列函数常见算法有 MD（消息摘要算法）、SHA（安全散列算法）、Mac（消息认证码算法）

## 场景

在下载软件时，偶尔也有从网上下载到破损文件的经历。情况严重时，还可能从某软件的官网上下载到被纂改的软件。

若要验证所下载的文件和官方提供文件是否一致，就引入了数据完整性验证的问题。

而消息摘要算法成了校验数据完整性的主要手段。

消息摘要算法最初用来构建数字签名。

数字签名操作中，签名操作其实是变相的使用消息摘要算法获得的数字指纹，而验证操作则是验证其数字指纹是否相符合。这也是为什么当山东大学王小云教授通过碰撞算法破解MD5和SHA算法后，使得数字签名在理论上被伪造成为可能。

此外用于校验数据完整性的算法还有CRC32、RipeMD系列等。

20180918

在实现注册/登录场景时，恐怕需要在数据库中使用一个字段以保存所使用算法，避免需要替换算法时无法确定用户当前密码所使用的算法。

## MD算法系列

## SHA算法系列

SHA算法基于MD4算法实现，作为MD算法的继任者成为了新一代的消息摘要算法代表。SHA算法与MD算法不同之处主要在于摘要长度，SHA算法摘要更长，安全性更高。

SHA-0

SHA-1

SHA-2 ( 以下4种算法统称 SHA-2 )

SHA-224

SHA-256

SHA-384

SHA-512

## MAC算法系列

## 其他消息摘要算法

## 循环冗余校验算法——CRC算法

### 场景

压缩软件WinRAR，在亚索文件列表中能看到一个标有“CRC32”字样的内容，“奇偶校验码”、“循环冗余校验码”和CRC32都是同一套东西，它们和CRC有着千丝万缕的联系。

CRC算法并不属于加密算法范畴。

CRC可以根据数据产生简短的固定位数的一种散列函数，主要用来校验数据传输或保存后出现的错误。生成的散列值在传输或储存之前计算出来并附加到数据后面。

在通信领域广泛地用于实现差错控制。

### 实例

package org.hc.learning.algorithm.消息摘要;

import static org.junit.Assert.assertEquals;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.security.DigestInputStream;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import org.apache.commons.codec.digest.DigestUtils;

import org.bouncycastle.util.encoders.Hex;

import org.junit.Test;

public class CRC文件校验 {

@Test

public void testByMessageDigest() {

// 文件路径

String path = "F:\\vs\_community\_CHS\_\_149203781.1488612268.exe";

// 构建文件输入流

FileInputStream fis = null;

try {

fis = new FileInputStream(new File(path));

} catch (FileNotFoundException e) {

e.printStackTrace();

}

// 初始化MessageDigest, 并指定MD5算法

DigestInputStream dis = null;

try {

dis = new DigestInputStream(fis, MessageDigest.getInstance("MD5"));

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

// 流缓冲大小

int buf = 1024;

// 缓冲字节数组

byte[] buffer = new byte[buf];

// 当读到值大于-1就继续读

int read = 0;

try {

read = dis.read(buffer, 0, buf);

} catch (IOException e) {

e.printStackTrace();

}

while(read > -1) {

try {

read = dis.read(buffer, 0, buf);

} catch (IOException e) {

e.printStackTrace();

}

}

// 关闭流

try {

dis.close();

} catch (IOException e) {

e.printStackTrace();

}

// 获得MessageDigest

MessageDigest md = dis.getMessageDigest();

// 摘要处理

byte[] b = md.digest();

String md5Hex = Hex.toHexString(b);

System.out.println(md5Hex);

assertEquals("比如软件下载官网提供的消息摘要",md5Hex);

}

@Test

public void testByDigestUtils() throws Exception {

// 文件路径

String path = "F:\\vs\_community\_CHS\_\_149203781.1488612268.exe";

FileInputStream fis = new FileInputStream(new File(path));

String md5Hex = DigestUtils.md5Hex(fis);

System.out.println(md5Hex);

fis.close();

assertEquals("比如软件下载官网提供的消息摘要", md5Hex);

}

}

以上两种对文件校验的方法效率相近，但apache提供的common-codec对以上操作进行了封装，可用更少代码量进行校验。

<!-- 编解码器 -->

<!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

</dependency>

# 定位模拟法

'''

Created on 2018年8月25日

@author: Administrator

'''

'''

#https://zhuanlan.zhihu.com/p/38850888

title:

一副从1到n的牌，

每次从牌堆顶取一张放桌子上，

再取一张放牌堆底，

直到手机没牌，

最后桌子上的牌是从1到n有序，

设计程序，输入n，输出牌堆的顺序数组

'''

#弃牌 排序

# 1 2 3 4 5

# table | cards

#[] | [1,5,2,4,3]

#[1 ] | [5,2,4,3] -> [2,4,3,5]

#[1,2] | [4 , 3, 5] -> [3,5,4]

#[1,2,3] | [4,5] -> [5]

#[1,2,3,4] | [5]

#[1,2,3,4,5] | []

#排序 加牌

#[1,2,3,4,5] | []

#[1,2,3,4] | [] -> [5]

#[1,2,3] | [5] -> [4,5]

#[1,2] | [5,4 -> [3,5,4]

#[1] | [4,3,5] -> [2,4,3,5]

#[] | [5,2,4,3] -> [1,5,2,4,3]

# 结果数组 1 2 3 4 5 值

# 定位数组 1 3 5 4 2 位置

# 原数组 1 5 2 4 3

# 定位模拟法

# 将 等长的自然数数组 作为定位数组

# 按照变换规则将定位数组变换n次，得到结果数组进行n次变换后的位置

# 将结果数组 根据 定位数组 变换位置 得到原数组

n = 6

list\_result = list( range(1, n ) ) #声明一个结果数组

print( "结果数组:\t\t"+str( list\_result ) )

list\_location = list( range(1, n ) ) #初始化 定位数组

print( "初始定位数组:\t"+str(list\_location) )

#对 定位数组 按照规则 变换

count = 0

length = len(list\_location)

while count < length-1 : #根据变换规则进行变换

temp = list\_location[count+1]

list\_location.remove(temp)

list\_location.append(temp)

count=count+1

print( "变换后定位数组:\t"+str(list\_location ))

list\_ori = []

count = 1

while count <= length :

list\_ori.append( list\_result[ list\_location.index( count )] )

count = count + 1

print("原数组:\t\t" + str(list\_ori))

# 如果能令变换规则更加抽象，事情大概会更简单