**云南大学数学与与统计学院**

**上机实践报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：信息论基础实验 | **年级**：2013 | **上机实践成绩**： |
| **指导教师**：陆正福 | **姓名**：金洋 |  |
| **上机实践名称**：分组密码实验 | **学号**：20131910023 | **上机实践日期**：2016/7/3 |
| **上机实践编号**：No.10 | **组号**： | **上机实践时间**： **13:32** |

**一、实验目的**

理解分组密码体制

**二、实验内容**

1. DES密码体制的实现与分析
2. AES密码体制的实现与分析

要求：

（1） 实现密码体制。

（2） 任取一段输入数据作为明文，计算明文熵；将密码系统作用于明文，得到密文，计算密文熵。比较明文熵和密文熵。

（3）改变明文1bit，观察密文的变化。改变密钥1bit，观察密文的变化。

（4）改变密文1bit，观察解密后的明文变化。

（5）分析（2）-（4）中的实验现象和原因。

**三、实验环境**

1. 个人计算机，任意可以完成实验的平台，如Java平台、Python语言、R语言、Matlab平台、Magma平台等。
2. 对于信息与计算科学专业的学生，建议选择Java、Python、R等平台。
3. 对于非信息与计算科学专业的学生，建议选择Matlab、Magma等平台。

**四、实验记录与实验结果分析**

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。**如果涉及实验结果截屏，应选择白底黑字。**）

1. DES密码体制的实现与分析

DES算法全称为Data Encryption Standard，即数据加密算法，它是IBM公司于1975年研究成功并公开发表的。DES算法的入口参数有三个：Key、Data、Mode。其中Key为8个字节共64位，是DES算法的工作密钥；Data也为8个字节64位，是要被加密或被解密的数据；Mode为DES的工作方式,有两种：加密或解密。

算法原理：DES算法把64位的明文输入块变为64位的密文输出块，它所使用的密钥也是64位，其算法主要分为两步：

①初始置换：其功能是把输入的64位数据块按位重新组合,并把输出分为L0、R0两部分，每部分各长32位，其置换规则为将输入的第58位换到第一位，第50位换到第2位……依此类推,最后一位是原来的第7位。L0、R0则是换位输出后的两部分，L0是输出的左32位，R0是右32位，例：设置换前的输入值为D1D2D3……D64，则经过初始置换后的结果为:L0=D58D50……D8；R0=D57D49……D7。

②逆置换：经过16次迭代运算后，得到L16、R16,将此作为输入，进行逆置换，逆置换正好是初始置换的逆运算，由此即得到密文输出。

（1） 实现密码体制。

**DES.java**

**package** IT10;

**import** java.security.InvalidKeyException;

**import** java.security.Key;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.SecureRandom;

**import** java.security.spec.InvalidKeySpecException;

**import** javax.crypto.Cipher;

**import** javax.crypto.SecretKey;

**import** javax.crypto.SecretKeyFactory;

**import** javax.crypto.spec.DESKeySpec;

**import** org.apache.commons.codec.binary.Base64;

**import** java.util.Scanner;

**import** com.sun.prism.impl.BaseMesh;

**public** **class** DES {

//算法名称

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

//算法名称/加密模式/填充方式

//DES共有四种工作模式-->>ECB：电子密码本模式、CBC：加密分组链接模式、CFB：加密反馈模式、OFB：输出反馈模式

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

/\*\*

\*

\* 生成密钥key对象

\* **@param** KeyStr 密钥字符串

\* **@return** 密钥对象

\* **@throws** InvalidKeyException

\* **@throws** NoSuchAlgorithmException

\* **@throws** InvalidKeySpecException

\* **@throws** Exception

\*/

**private** **static** SecretKey keyGenerator(String keyStr) **throws** Exception {

**byte** input[] = *HexString2Bytes*(keyStr);

DESKeySpec desKey = **new** DESKeySpec(input);

//创建一个密匙工厂，然后用它把DESKeySpec转换成

SecretKeyFactory keyFactory = SecretKeyFactory.*getInstance*("DES");

SecretKey securekey = keyFactory.generateSecret(desKey);

**return** securekey;

}

**private** **static** **int** parse(**char** c) {

**if** (c >= 'a') **return** (c - 'a' + 10) & 0x0f;

**if** (c >= 'A') **return** (c - 'A' + 10) & 0x0f;

**return** (c - '0') & 0x0f;

}

// 从十六进制字符串到字节数组转换

**public** **static** **byte**[] HexString2Bytes(String hexstr) {

**byte**[] b = **new** **byte**[hexstr.length() / 2];

**int** j = 0;

**for** (**int** i = 0; i < b.length; i++) {

**char** c0 = hexstr.charAt(j++);

**char** c1 = hexstr.charAt(j++);

b[i] = (**byte**) ((*parse*(c0) << 4) | *parse*(c1));

}

**return** b;

}

/\*\*

\* 加密数据

\* **@param** data 待加密数据

\* **@param** key 密钥

\* **@return** 加密后的数据

\*/

**public** **static** String encrypt(String data, String key) **throws** Exception {

Key deskey = *keyGenerator*(key);

// 实例化Cipher对象，它用于完成实际的加密操作

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

SecureRandom random = **new** SecureRandom();

// 初始化Cipher对象，设置为加密模式

cipher.init(Cipher.***ENCRYPT\_MODE***, deskey, random);

**byte**[] results = cipher.doFinal(data.getBytes());

// 该部分是为了与加解密在线测试网站（http://tripledes.online-domain-tools.com/）的十六进制结果进行核对

**for** (**int** i = 0; i < results.length; i++) {

System.***out***.print(results[i] + " ");

}

System.***out***.println();

// 执行加密操作。加密后的结果通常都会用Base64编码进行传输

**return** Base64.*encodeBase64String*(results);

}

/\*\*

\* 解密数据

\* **@param** data 待解密数据

\* **@param** key 密钥

\* **@return** 解密后的数据

\*/

**public** **static** String decrypt(String data, String key) **throws** Exception {

Key deskey = *keyGenerator*(key);

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

//初始化Cipher对象，设置为解密模式

cipher.init(Cipher.***DECRYPT\_MODE***, deskey);

// 执行解密操作

**return** **new** String(cipher.doFinal(Base64.*decodeBase64*(data)));

}

**public** **static** **void** main(String[] args) **throws** Exception {

System.***out***.println("请输入明文:");

Scanner input=**new** Scanner(System.***in***);

String message=input.next();

System.***out***.println("原文: " + message);

System.***out***.println("明文信息熵为： "+ ENTROPY.*entropy*(message));

String key = "A1B2C3D4E5F60708";

System.***out***.println("密钥："+key);

String encryptData = *encrypt*(message, key);

System.***out***.println("加密后: " + encryptData);

System.***out***.println("密文信息熵为： "+ ENTROPY.*entropy*(encryptData));

System.***out***.println("请输入密文：");

encryptData=input.next();

String decryptData = *decrypt*(encryptData, key);

System.***out***.println("解密后: " + decryptData);

}

}

计算熵的部分：

**Node.java**

**package** IT10;

**public** **class** Node {

**private** **double** pr;

**private** **char** al;

**public** **void** setp(**double** p) {

**this**.pr = p;

}

**public** **void** setalpha(**char** a) {

**this**.al = a;

}

**public** **double** getp() {

**return** pr;

}

**public** **char** getalpha() {

**return** al;

}

**public** Node(**double** p, **char** alpha) {

**this**.pr = p;

**this**.al = alpha;

}

}

**ENTROPY.java**

**package** IT10;

**import** java.util.ArrayList;

**public** **class** ENTROPY {

**public** **static** **double** entropy(String message) {

ArrayList<Node> array = **new** ArrayList<Node>();

array.clear();

**double** num = message.length();

**for** (**int** i = 0; i < num; i++) {

**boolean** flag\_exit = **true**;

**for** (**int** j = 0; j < array.size(); j++) {

**if** (array.get(j).getalpha() == message.charAt(i)) {

flag\_exit = **false**;

array.get(j).setp(array.get(j).getp() + 1 / num);

}

}

**if** (flag\_exit)

array.add(**new** Node(1 / num, message.charAt(i)));

}

**double** entropy = 0;

**for** (**int** i = 0; i < array.size(); i++) {

**double** p1 = array.get(i).getp();

entropy += (-p1 \* (Math.*log*(p1) / Math.*log*(2)));

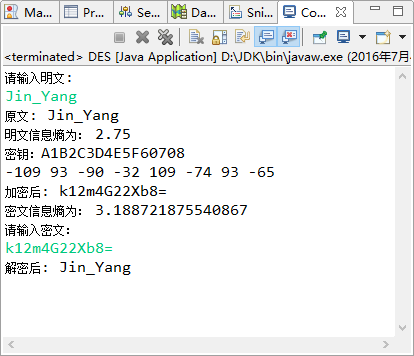
}

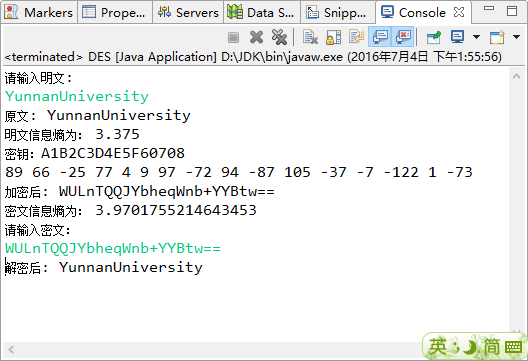
**return** entropy;

}

}

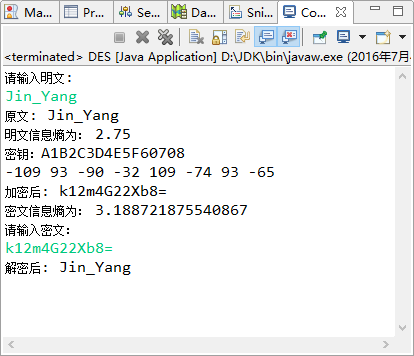
1. 任取一段输入数据作为明文，计算明文熵；将密码系统作用于明文，得到密文，计算密文熵。比较明文熵和密文熵。



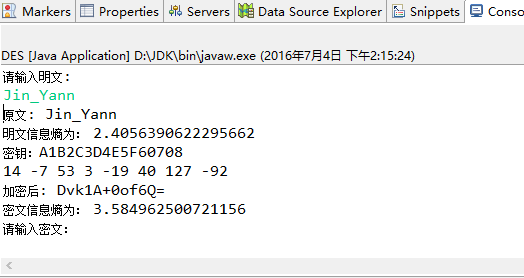


一般来说，明文熵和密文熵相比，后者的熵值更大；

1. 改变明文1bit，观察密文的变化。改变密钥1bit，观察密文的变化。

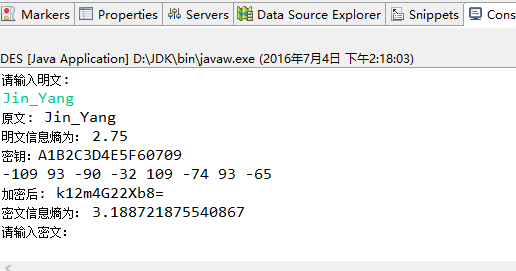


改变明文1 bit后，



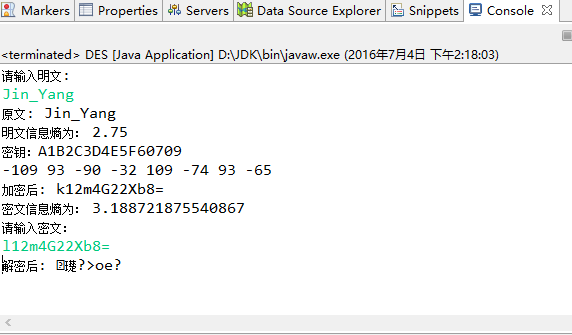
密文完全变化；

改变密钥1bit后（由 key = "A1B2C3D4E5F60708" 变为key = "A1B2C3D4E5F60709";）：



密文没有变化。

1. 改变密文1bit，观察解密后的明文变化。



解密后明文完全变化。

（5）分析（2）-（4）中的实验现象和原因。

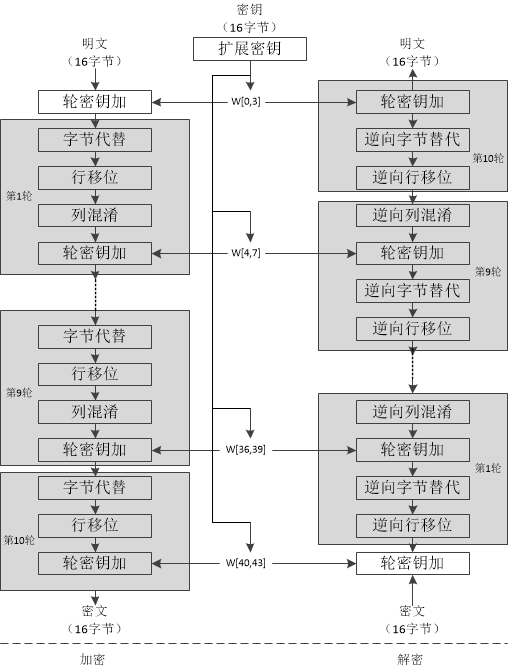
加密会增加数据的冗余这会导致密文的熵变大；且由信源绝对信息率定义为，信源的近似信息率定义为，可得明文熵，密文熵，后者显然更大；

改变密钥1bit后（由 key = "A1B2C3D4E5F60708" 变为key = "A1B2C3D4E5F60709";）：但是密文结果没有变化，这是比较疑问的地方；

无论是加密还是解密，当输入的信息发生变动，由于在加解密算法都要对数据进行多轮分块组合，即使输入的异差很小，经过运算后的结果将大不相同。

1. AES密码体制的实现与分析

　AES加解密的流程图如下：



1. 实现密码体制。

**AESTest.java**

**package** IT10;

**import** java.io.UnsupportedEncodingException;

**import** java.security.InvalidKeyException;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.SecureRandom;

**import** java.util.Scanner;

**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;

**public** **class** AESTest {

**public** **static** **void** main(String[] args) {

System.***out***.println("请输入明文:");

Scanner input=**new** Scanner(System.***in***);

String message=input.next();

System.***out***.println("原文: " + message);

System.***out***.println("明文信息熵为： "+ ENTROPY.*entropy*(message));

String key = "A1B2C3D4E5F60709";

System.***out***.println("密钥："+key);

String encryptData = *encrypt*(message, key);

System.***out***.println("加密后: " + encryptData);

System.***out***.println("密文信息熵为： "+ ENTROPY.*entropy*(encryptData));

System.***out***.println("请输入密文：");

String newEncryptData=input.next();

String decryptData = *decrypt*(newEncryptData, key);

System.***out***.println("解密后: " + decryptData);

}

/\*\*

\* 加密

\*

\* **@param** content

\* 待加密内容

\* **@param** key

\* 加密的密钥

\* **@return**

\*/

**public** **static** String encrypt(String content, String key) {

**try** {

KeyGenerator kgen = KeyGenerator.*getInstance*("AES");

kgen.init(128, **new** SecureRandom(key.getBytes()));

SecretKey secretKey = kgen.generateKey();

**byte**[] enCodeFormat = secretKey.getEncoded();

SecretKeySpec secretKeySpec = **new** SecretKeySpec(enCodeFormat, "AES");

Cipher cipher = Cipher.*getInstance*("AES");

**byte**[] byteContent = content.getBytes("utf-8");

cipher.init(Cipher.***ENCRYPT\_MODE***, secretKeySpec);

**byte**[] byteRresult = cipher.doFinal(byteContent);

StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < byteRresult.length; i++) {

String hex = Integer.*toHexString*(byteRresult[i] & 0xFF);

**if** (hex.length() == 1) {

hex = '0' + hex;

}

sb.append(hex.toUpperCase());

}

**return** sb.toString();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (UnsupportedEncodingException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

}

**return** **null**;

}

/\*\*

\* 解密

\*

\* **@param** content

\* 待解密内容

\* **@param** key

\* 解密的密钥

\* **@return**

\*/

**public** **static** String decrypt(String content, String key) {

**if** (content.length() < 1)

**return** **null**;

**byte**[] byteRresult = **new** **byte**[content.length() / 2];

**for** (**int** i = 0; i < content.length() / 2; i++) {

**int** high = Integer.*parseInt*(content.substring(i \* 2, i \* 2 + 1), 16);

**int** low = Integer.*parseInt*(content.substring(i \* 2 + 1, i \* 2 + 2), 16);

byteRresult[i] = (**byte**) (high \* 16 + low);

}

**try** {

KeyGenerator kgen = KeyGenerator.*getInstance*("AES");

kgen.init(128, **new** SecureRandom(key.getBytes()));

SecretKey secretKey = kgen.generateKey();

**byte**[] enCodeFormat = secretKey.getEncoded();

SecretKeySpec secretKeySpec = **new** SecretKeySpec(enCodeFormat, "AES");

Cipher cipher = Cipher.*getInstance*("AES");

cipher.init(Cipher.***DECRYPT\_MODE***, secretKeySpec);

**byte**[] result = cipher.doFinal(byteRresult);

**return** **new** String(result);

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

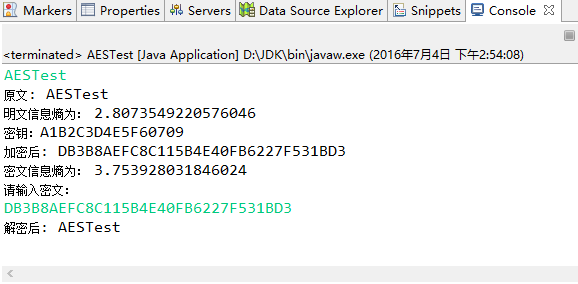
}

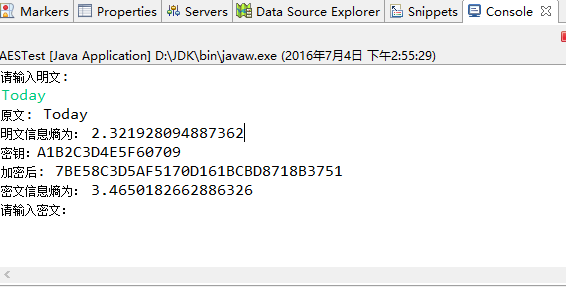
**return** **null**;

}

}

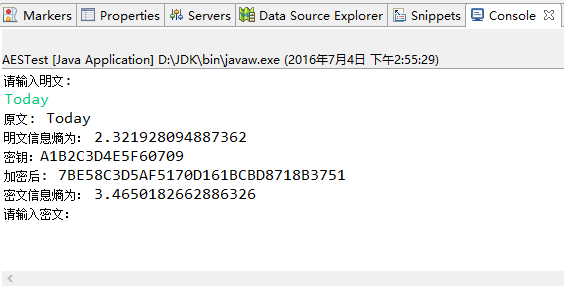
1. 任取一段输入数据作为明文，计算明文熵；将密码系统作用于明文，得到密文，计算密文熵。比较明文熵和密文熵。

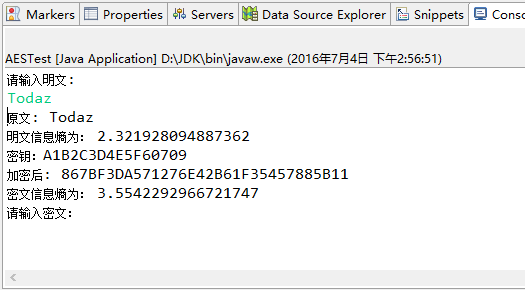


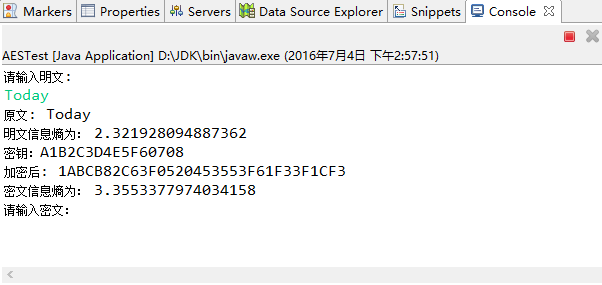


均为明文熵<密文熵。

1. 改变明文1bit，观察密文的变化。改变密钥1bit，观察密文的变化。





改变明文1bit，密文完全变化；

改变密钥1bit，密文完全变化。

**五、实验体会**

**（请认真填写自己的真实体会）**

分析（2）-（4）中的实验现象和原因：

加密会增加数据的冗余这会导致密文的熵变大；且由信源绝对信息率定义为，信源的近似信息率定义为，可得明文熵，密文熵，后者显然更大；

改变密钥1bit后（由 key = "A1B2C3D4E5F60708" 变为key = "A1B2C3D4E5F60709";）：但是密文结果没有变化，这是比较疑问的地方；

无论是加密还是解密，当输入的信息发生变动，由于在加解密算法都要对数据进行多轮分块组合，即使输入的异差很小，经过运算后的结果将大不相同。

**六、参考文献**

1. Thomas M. Cover, Joy A. Thomas.   Elements of Information Theory (2nd Edition) [M]. John Wiley & Sons, Inc.

**2. 阿蜜果. [常用加密算法的Java实现总结(二)——对称加密算法DES、3DES和AES](http://www.blogjava.net/amigoxie/archive/2014/07/06/415503.html) [EB/OL]. http://www.blogjava.net/amigoxie/archive/2014/07/06/415503.html, 2014-07-06.**