|  |  |
| --- | --- |
| **计算机网络安全实验报告** | |
| 点到点加密通讯 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
| **姓名** | 田丰瑞 |
| **班级** | 软件73班 |
| **学号** | 2172213528 |
| **电话** | 18744296191 |
| **Email** | [tianfr@stu.xjtu.edu.cn](mailto:tianfr@stu.xjtu.edu.cn?subject=软件73%20田丰瑞) |
| **日期** | 2020-5-14 |

# 声明

网络安全实验的所有代码已经全部开源于我的GitHub（仅开源代码部分），项目地址：

<https://github.com/tianfr/Internet-Security-ExpCode/>

# 点到点加密通讯实验

## 实验要求

使用非对称加密算法设计一个点到点加密通讯软件，要求实现完整性和鉴别。

## 设计思路

首先设计一个点到点通讯软件，然后在软件通讯时利用一系列加密算法加密。

### 点到点通讯软件

#### 功能性需求

该软件需实现包括但不限于以下功能：

**用户注册：**用户通过点击注册按钮进入注册界面，通过输入一次用户名两次用户密码实现用户注册。

**用户登录：**用户通过输入合法用户名和密码实现用户登录，并进入软件主页面。

**实时查询在线用户：**登录用户在软件界面可实时刷新查看在线用户人员。

**选择聊天用户：**登录用户可在软件界面选择想聊天的用户人员。

**向指定用户发送消息：**用户可在软件主界面向用户发送消息。

**向指定用户发送文件：**用户可在软件主界面向指定用户发送本地文件。

**实时接收用户发送消息：**软件主界面实时显示其他用户发送的消息。

**实时接收用户发送文件：当其他用户向本用户发送文件时，软件主界面实时提示查收文件。**

**消息保存：用户可以保存聊天记录。**

#### **加密通讯需求**

保证消息完整性、机密性、防重放

加密关键技术：RSA非对称加密，Hash算法完整性检查，DES对称加密算法，挑战应答机制

CA认证机构，数字信封，公钥证书

在通讯过程中，服务器充当一个CA，起到分发公钥证书和密钥的过程。

公钥证书制作流程

通讯整体流程如下：

**前提条件：每个客户端有CA的公钥，CA维护所有用户的公钥，每个客户端有自己的公私钥。**

##### 连接建立开始——公钥传递

假设客户端A想与A通讯，首先客户端A向服务器发出请求申请公钥证书请求包括一段随机数作为挑战应答的内容，服务器返回一个用A的公钥加密好B的公钥证书，A用自己的私钥解密并校验解密后的挑战应答结果，无误后读取解密内容获得B的公钥，A再将自己的公钥证书和随机数作为挑战应答整合一起用hash创建校验码，最后用B的公钥加密后发给B，B得到A发过来的包后先用自己的私钥解密，然后校验哈希函数，准确无误后从数据中获得A的公钥，并用A的公钥加密挑战应答结果后发给A，A收到后用自己的私钥解开比对挑战应答结果，无误证明连接建立。



图1：用户A与用户B的连接建立

##### 消息传输——a与b之间即时消息传递

A将发送的消息用自己的私钥签名后创建hash校验码，将校验码加到消息后面再用B的公钥加密后发送给B，B收到消息后用自己的私钥解密后，用hash函数校验完整性，无误后用A的公钥验证签名，签名无误后读取A的消息，AB之间消息传递完成。

表1：即时消息报文传输协议

|  |  |  |  |
| --- | --- | --- | --- |
| **即时消息报文协议** | | | |
| **数据内容** | **数据处理** | | |
| HASH校验码 |  |  | 接收方公钥加密 |
| 接收方用户地址 | HASH256 |  |
| 发送方用户地址 |
| 发送时间 |
| 即时消息内容 | 用发送方私钥加密后压缩 |



图2：即时消息传输示例图

##### 消息传输——a与b之间文件传递

A首先创建一个随机的DES密钥，然后将文件内容及信息，时间戳和A的签名用zip压缩后再用DES密钥加密文件内容，并用B的公钥加密DES密钥创建数字信封，将加密文件信息和数字信封整合到一起利用socket发送给B，B用自己的私钥解开DES密钥，然后解压缩后用DES密钥解密，验证签名后读取文件信息，至此，文件传输完成。

表2：传输文件协议

|  |  |  |
| --- | --- | --- |
| **传输文件协议** | | |
| **数据内容** | **数据处理** | |
| HASH校验码 | HASH256 | |
| 接收方用户地址 |
| 发送方用户地址 |
| DES密钥 | 发送方私钥签名，接收方公钥加密 |  |
| 发送时间 |  |
| 文件内容 | zip压缩后用随机DES密钥加密 |



图3：文件传输示意图

## 原理简介

### **非对称算法加密通讯**

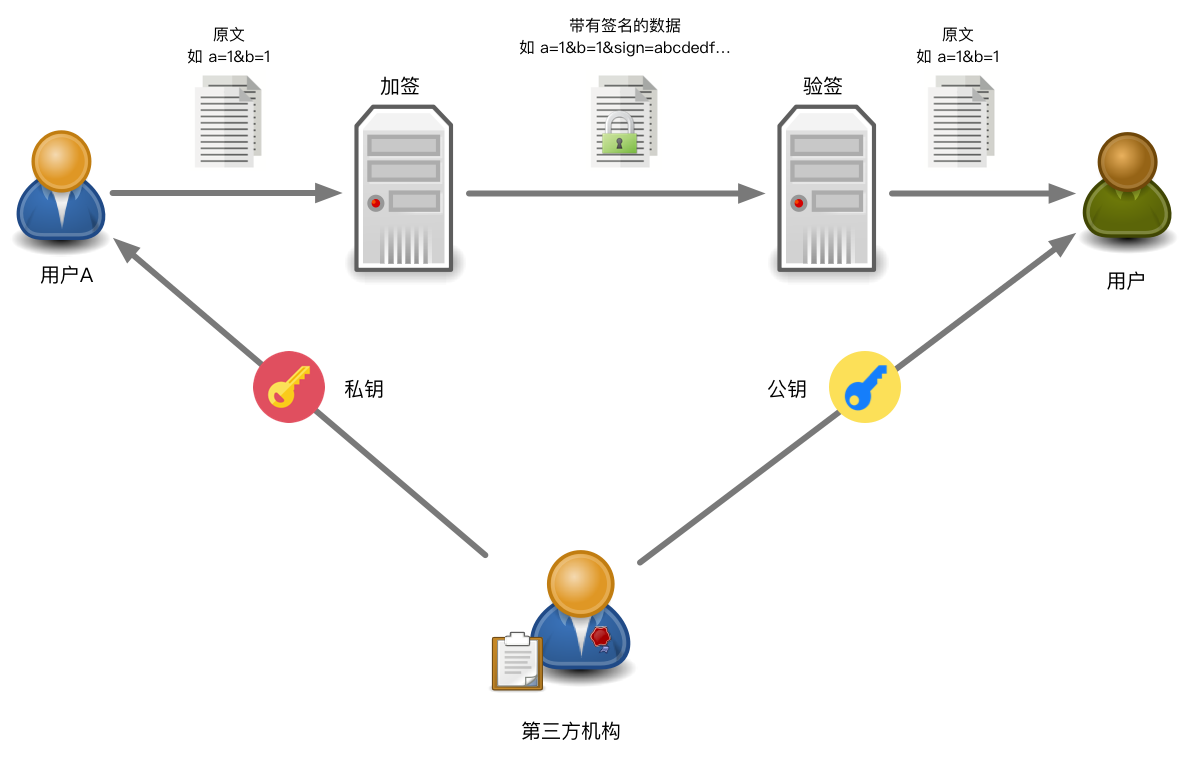
非对称加密算法是一种密钥的保密方法。

非对称加密算法需要两个密钥：**公开密钥（publickey:简称公钥）和私有密钥（privatekey:简称私钥）**。公钥与私钥是一对，如果用公钥对数据进行加密，只有用对应的私钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。 非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将公钥公开，需要向甲方发送信息的其他角色(乙方)使用该密钥(甲方的公钥)对机密信息进行加密后再发送给甲方；甲方再用自己私钥对加密后的信息进行解密。甲方想要回复乙方时正好相反，使用乙方的公钥对数据进行加密，同理，乙方使用自己的私钥来进行解密。

另一方面，甲方可以使用自己的私钥对机密信息进行签名后再发送给乙方；乙方再用甲方的公钥对甲方发送回来的数据进行验签。

甲方只能用其私钥解密由其公钥加密后的任何信息。 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。

非对称密码体制的特点：算法强度复杂、安全性依赖于算法与密钥但是由于其算法复杂，而使得加密解密速度没有对称加密解密的速度快。对称密码体制中只有一种密钥，并且是非公开的，如果要解密就得让对方知道密钥。所以保证其安全性就是保证密钥的安全，而非对称密钥体制有两种密钥，其中一个是公开的，这样就可以不需要像对称密码那样传输对方的密钥了。这样安全性就大了很多。



### 点到点通讯原理

在电信中，点到点连接是指两个节点或端点之间的通信连接。

一个例子是电话呼叫，其中一个电话相互连接，一个呼叫者所说的只能被另一个呼叫者听到。 这与点对多点或广播连接形成对比，其中许多节点可以接收由一个节点发送的信息。 点对点通信链路的其他示例是租用线路，微波中继链路和双向无线电。

该术语还用于计算机网络和计算机体系结构中，以指代仅链接两个计算机或电路的线路或其它连接，而不是诸如可连接许多通信设备的总线或交叉开关的其它网络拓扑。

点对点有时缩写为P2P。 P2P的这种使用不同于指向用于文件共享网络的对等的P2P。

### Python socket编程

Python 提供了两个级别访问的网络服务。：

* 低级别的网络服务支持基本的 Socket，它提供了标准的 BSD Sockets API，可以访问底层操作系统Socket接口的全部方法。
* 高级别的网络服务模块 SocketServer， 它提供了服务器中心类，可以简化网络服务器的开发。

#### 什么是 Socket?

Socket又称"套接字"，应用程序通常通过"套接字"向网络发出请求或者应答网络请求，使主机间或者一台计算机上的进程间可以通讯。

#### socket()函数

Python 中，我们用 socket（）函数来创建套接字，语法格式如下：

socket.socket([family[, type[, proto]]])

#### 参数

* family: 套接字家族可以使AF\_UNIX或者AF\_INET
* type: 套接字类型可以根据是面向连接的还是非连接分为SOCK\_STREAM或SOCK\_DGRAM
* protocol: 一般不填默认为0.

#### Socket 对象(内建)方法

| **函数** | **描述** |
| --- | --- |
| 服务器端套接字 | |
| s.bind() | 绑定地址（host,port）到套接字， 在AF\_INET下,以元组（host,port）的形式表示地址。 |
| s.listen() | 开始TCP监听。backlog指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为1，大部分应用程序设为5就可以了。 |
| s.accept() | 被动接受TCP客户端连接,(阻塞式)等待连接的到来 |
| 客户端套接字 | |
| s.connect() | 主动初始化TCP服务器连接，。一般address的格式为元组（hostname,port），如果连接出错，返回socket.error错误。 |
| s.connect\_ex() | connect()函数的扩展版本,出错时返回出错码,而不是抛出异常 |
| 公共用途的套接字函数 | |
| s.recv() | 接收TCP数据，数据以字符串形式返回，bufsize指定要接收的最大数据量。flag提供有关消息的其他信息，通常可以忽略。 |
| s.send() | 发送TCP数据，将string中的数据发送到连接的套接字。返回值是要发送的字节数量，该数量可能小于string的字节大小。 |
| s.sendall() | 完整发送TCP数据，完整发送TCP数据。将string中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。 |
| s.recvfrom() | 接收UDP数据，与recv()类似，但返回值是（data,address）。其中data是包含接收数据的字符串，address是发送数据的套接字地址。 |
| s.sendto() | 发送UDP数据，将数据发送到套接字，address是形式为（ipaddr，port）的元组，指定远程地址。返回值是发送的字节数。 |
| s.close() | 关闭套接字 |
| s.getpeername() | 返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。 |
| s.getsockname() | 返回套接字自己的地址。通常是一个元组(ipaddr,port) |
| s.setsockopt(level,optname,value) | 设置给定套接字选项的值。 |
| s.getsockopt(level,optname[.buflen]) | 返回套接字选项的值。 |
| s.settimeout(timeout) | 设置套接字操作的超时期，timeout是一个浮点数，单位是秒。值为None表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如connect()） |
| s.gettimeout() | 返回当前超时期的值，单位是秒，如果没有设置超时期，则返回None。 |
| s.fileno() | 返回套接字的文件描述符。 |
| s.setblocking(flag) | 如果flag为0，则将套接字设为非阻塞模式，否则将套接字设为阻塞模式（默认值）。非阻塞模式下，如果调用recv()没有发现任何数据，或send()调用无法立即发送数据，那么将引起socket.error异常。 |
| s.makefile() | 创建一个与该套接字相关连的文件 |

### DES加密简介

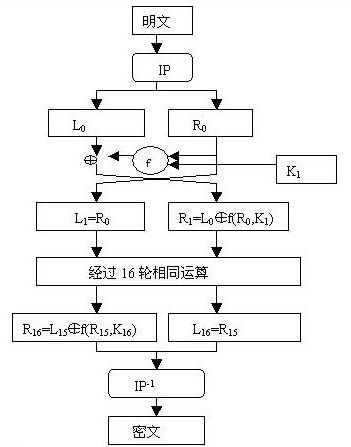
#### 1.DES算法简介

DES算法为密码体制中的对称密码体制，又被称为美国数据加密标准。

DES是一个分组加密算法，典型的DES以64位为分组对数据加密，加密和解密用的是同一个算法。

密钥长64位，密钥事实上是56位参与DES运算（第8、16、24、32、40、48、56、64位是校验位，使得每个密钥都有奇数个1），分组后的明文组和56位的密钥按位替代或交换的方法形成密文组。

DES算法的主要流程如下图所示，本文按照流程依次介绍每个模块。



#### 2.IP置换

IP置换目的是将输入的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。

置换规则如下表所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

表中的数字代表新数据中此位置的数据在原数据中的位置，即原数据块的第58位放到新数据的第1位，第50位放到第2位，……依此类推，第7位放到第64位。置换后的数据分为L0和R0两部分，L0为新数据的左32位，R0为新数据的右32位。

要注意一点，位数是从左边开始数的，即最0x0000 0080 0000 0002最左边的位为1，最右边的位为64。

#### 3.密钥置换

不考虑每个字节的第8位，DES的密钥由64位减至56位，每个字节的第8位作为奇偶校验位。产生的56位密钥由下表生成（注意表中没有8,16,24，32,40,48,56和64这8位）：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 58 | 50 | 42 | 34 | 26 | 18 |
| 10 | 2 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | 60 | 52 | 44 | 36 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 | 62 | 54 | 46 | 38 | 30 | 22 |
| 14 | 6 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | 28 | 20 | 12 | 4 |

在DES的每一轮中，从56位密钥产生出不同的48位子密钥，确定这些子密钥的方式如下：

1).将56位的密钥分成两部分，每部分28位。

2).根据轮数，这两部分分别循环左移1位或2位。每轮移动的位数如下表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 轮数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 位数 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

移动后，从56位中选出48位。这个过程中，既置换了每位的顺序，又选择了子密钥，因此称为压缩置换。压缩置换规则如下表（注意表中没有9，18，22，25，35，38，43和54这8位）：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14 | 17 | 11 | 24 | 1 | 5 | 3 | 28 | 15 | 6 | 21 | 10 |
| 23 | 19 | 12 | 4 | 26 | 8 | 16 | 7 | 27 | 20 | 13 | 2 |
| 41 | 52 | 31 | 37 | 47 | 55 | 30 | 40 | 51 | 45 | 33 | 48 |
| 44 | 49 | 39 | 56 | 34 | 53 | 46 | 42 | 50 | 36 | 29 | 32 |

置换方法同上，此处省略。

#### 4.E扩展置换

扩展置置换目标是IP置换后获得的右半部分R0，将32位输入扩展为48位(分为4位×8组)输出。

扩展置换目的有两个：生成与密钥相同长度的数据以进行异或运算；提供更长的结果，在后续的替代运算中可以进行压缩。

扩展置换原理如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

表中的数字代表位，两列黄色数据是扩展的数据，可以看出，扩展的数据是从相邻两组分别取靠近的一位，4位变为6位。靠近32位的位为1，靠近1位的位为32。表中第二行的4取自上组中的末位，9取自下组中的首位。

我们举个例子看一下(虽然扩展置换针对的是**上步IP置换中的R0，但为便于观察扩展，这里不取R0举例**)：

输入数据0x1081 1001，转换为二进制就是0001 0000 1000 0001B，按照上表扩展得下表

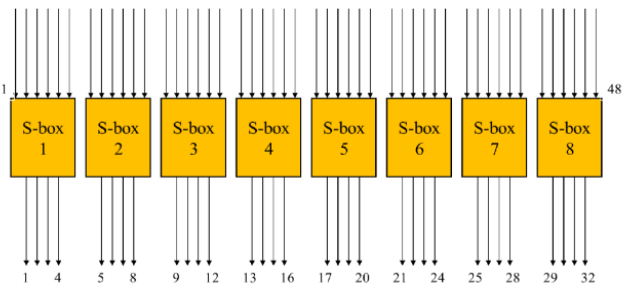
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |

表中的黄色数据是从临近的上下组取得的，二进制为1000 1010 0001 0100 0000 0010 1000 1010 0000 0000 0000 0010B，转换为十六进制0x8A14 028A 0002。

扩展置换之后，右半部分数据R0变为48位，与密钥置换得到的轮密钥进行异或。

#### 5.S盒代替

压缩后的密钥与扩展分组异或以后得到48位的数据，将这个数据送人S盒，进行替代运算。替代由8个不同的S盒完成，每个S盒有6位输入4位输出。48位输入分为8个6位的分组，一个分组对应一个S盒，对应的S盒对各组进行代替操作。



一个S盒就是一个4行16列的表，盒中的每一项都是一个4位的数。S盒的6个输入确定了其对应的输出在哪一行哪一列，输入的高低两位做为行数H，中间四位做为列数L，在S-BOX中查找第H行L列对应的数据(<32)。

8个S盒如下：

S盒1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

S盒2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |

S盒3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |

S盒4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 19 |
| 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |

S盒5

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 5 | 8 | 3 | 15 | 13 | 0 | 14 | 9 |
| 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 13 | 3 | 9 | 8 | 6 |
| 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

S盒6

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |

S盒7

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |

S盒8

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

例如，假设S盒8的输入为110011，第1位和第6位组合为11，对应于S盒8的第3行；第2位到第5位为1001，对应于S盒8的第9列。S盒8的第3行第9列的数字为12，因此用1100来代替110011。注意，S盒的行列计数都是从0开始。

代替过程产生8个4位的分组，组合在一起形成32位数据。

S盒代替时DES算法的关键步骤，所有的其他的运算都是线性的，易于分析，而S盒是非线性的，相比于其他步骤，提供了更好安全性。

#### 6.P盒置换

S盒代替运算的32位输出按照P盒进行置换。该置换把输入的每位映射到输出位，任何一位不能被映射两次，也不能被略去，映射规则如下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 |
| 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 |
| 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

表中的数字代表原数据中此位置的数据在新数据中的位置，即原数据块的第16位放到新数据的第1位，第7位放到第2位，……依此类推，第25位放到第32位。

例如0x10A1 0001进行P盒置换后变为0x8000 0886。

0x10A1 0001表现为表的形式（第一位位于左上角）原来为

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

经P盒变换后为

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

即1000 0000 0000 0000 0000 1000 1000 0110B，十六进制为0x8000 0886。

最后，P盒置换的结果与最初的64位分组左半部分L0异或，然后左、右半部分交换，接着开始另一轮。

#### 7.IP-1末置换

末置换是初始置换的逆过程，DES最后一轮后，左、右两半部分并未进行交换，而是两部分合并形成一个分组做为末置换的输入。末置换规则如下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 |
| 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

置换方法同上，此处省略。

经过以上步骤，就可以得到密文了。

# 点到点加密软件安全性分析

## AES公私钥加密安全性分析

### RSA算法加密强度

目前密码的破译主要有2种方法．方法之一是密钥的穷尽搜索，其破译方法是尝试所有可能的密钥组合．虽然大多数的密钥尝试都是失败的，但最终有一个密钥让破译者得到原文，这个过程称为密钥的穷尽搜索．方法之二是密码分析．由于RSA算法在加密和解密过程都是用指数计算，其计算工作量巨大，用穷尽搜索法进行破译是根本不可能的．因此要对RSA算法加密后的信息进行破译只能采用密码分析法，用密码分析法攻击RSA密码系统，途径之一是直接计算“规的e次方根”，但目前还没有解决这一问题的算法，这个问题是现实不可计算的问题；途径之二[4]是想办法计算出d，欲得到d，可考虑从以下3个方面人手．

### 大数因子分解的难度

## DES算法传输文件安全性分析

### 密钥分发

DES是对称的分组密码算法，对称的分组密码算法最主要的问题是：由于加解密双方都要使用

相同的密钥，因此在发送、接收数据之前，必须完成密钥的分发，因而密钥的分发便成了该加密体系中的最薄弱H风险最大的环节。

而在本次实验中， 作者使用RSa非对称加密算法来分发密钥，因此如果想直接破译密钥的难度转成了破译RSA加密算法的难度。关于Rsa加密算法的破译难度可以参考上节。由于RSa密钥破译的困难性， 本次用DES加密的文件传输最薄弱的环节——密钥分发得以保障。

### des算法漏洞

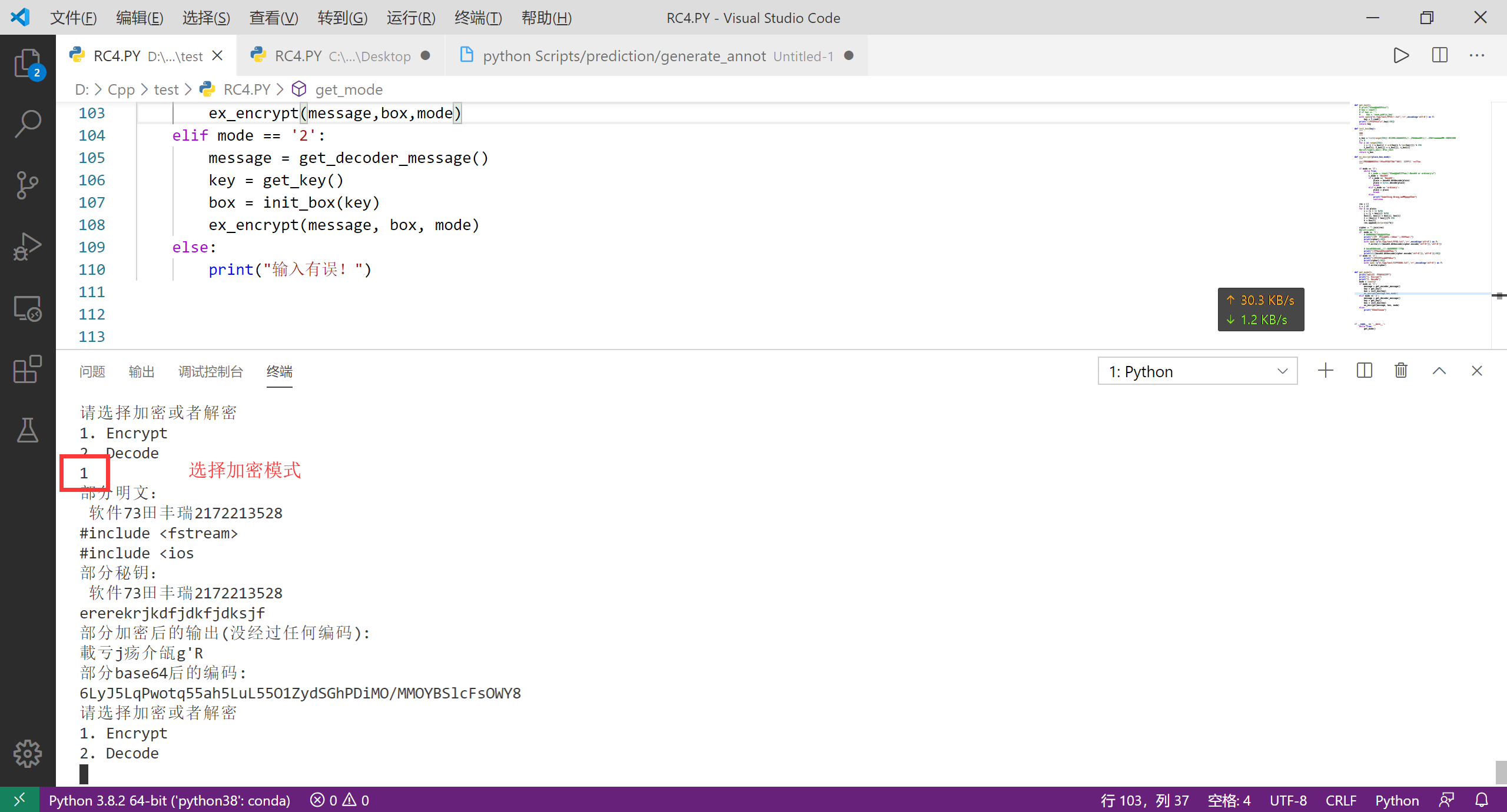
由DES算法我们可以看到，DES算法中只用到64位密钥中的其中65位而第8，16，24……64位 8个位并未参与DES运算.这一点向我们提出了一个应用上的要求即DES的安全性是基于除 了8，16，24……64位外的其余 56位的组合变化才得以保证的。因此,在实际应用中我们应避 开使用第8，16，24……64位作为DES密钥的有效数据位，而使用其它的 56位作为有效数据位。只有这样才能保证 DES算法安全可靠地发挥作用。如果不了解这一点把密钥8，16，24……64位作为有效数据位使用，将不能保证DES加密数据的安全性，对运用DES来达到保密作用的系统将产生数据被破译的危险。

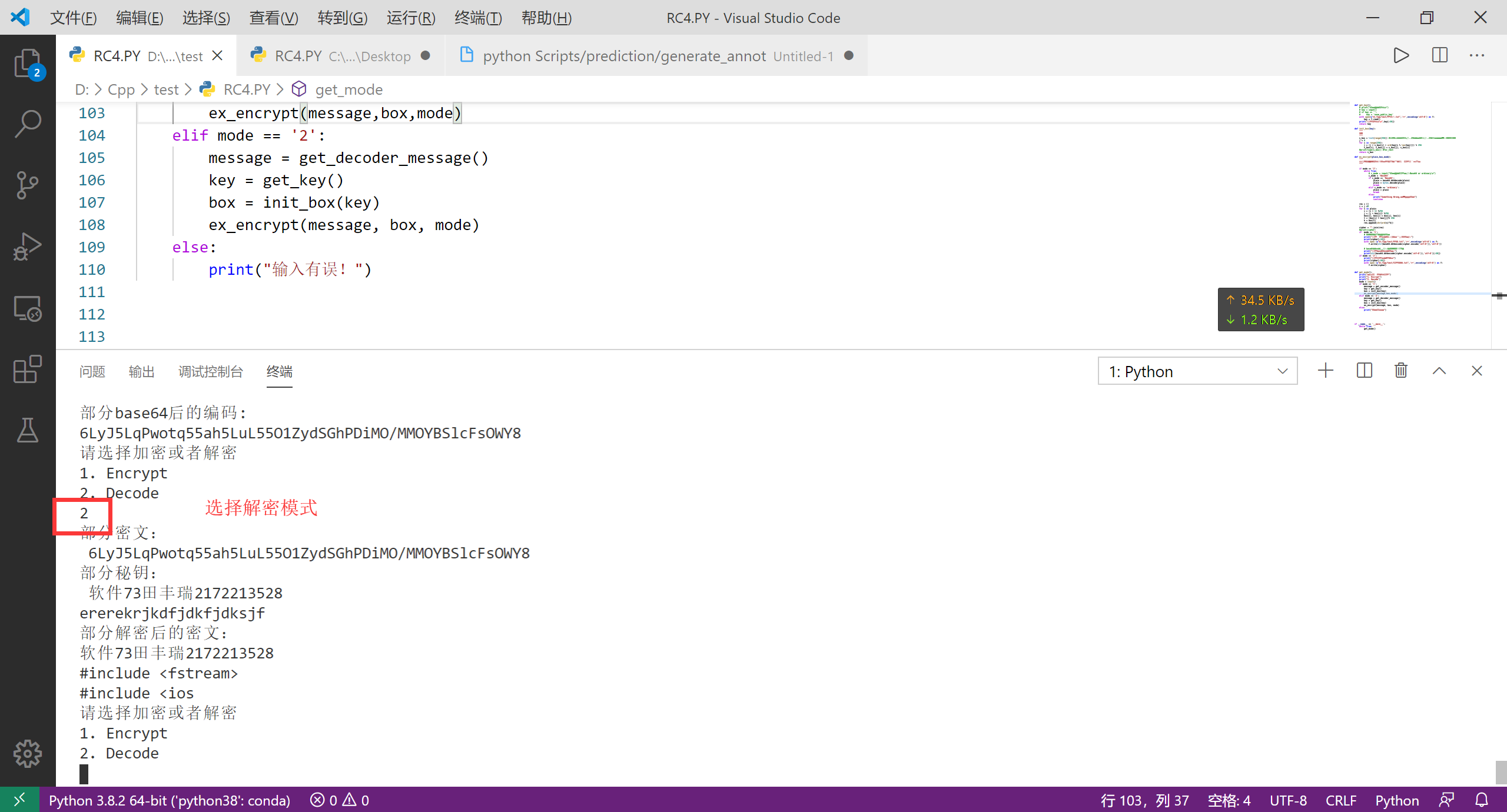
基于以上的问题，我们就不能用汉字作为密钥。因为汉字由两个字节组成，每个字节的 ASCⅡ码 都大于127，转换成二进制就是8位，不能加奇偶校验位，而且如果去掉第8，16，24……64位就会丢失密钥，而且不同的汉字可能实际上是相同的密钥。

# 运行结果

本次使用的明文为我之前写过的一段程序，明文开头有我的姓名学号标识。

### 程序执行结果

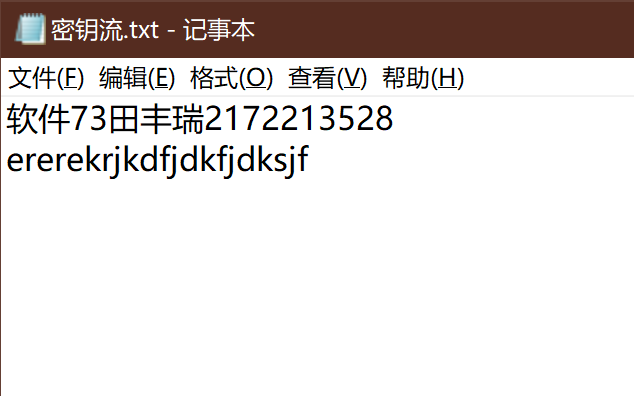




### 明文

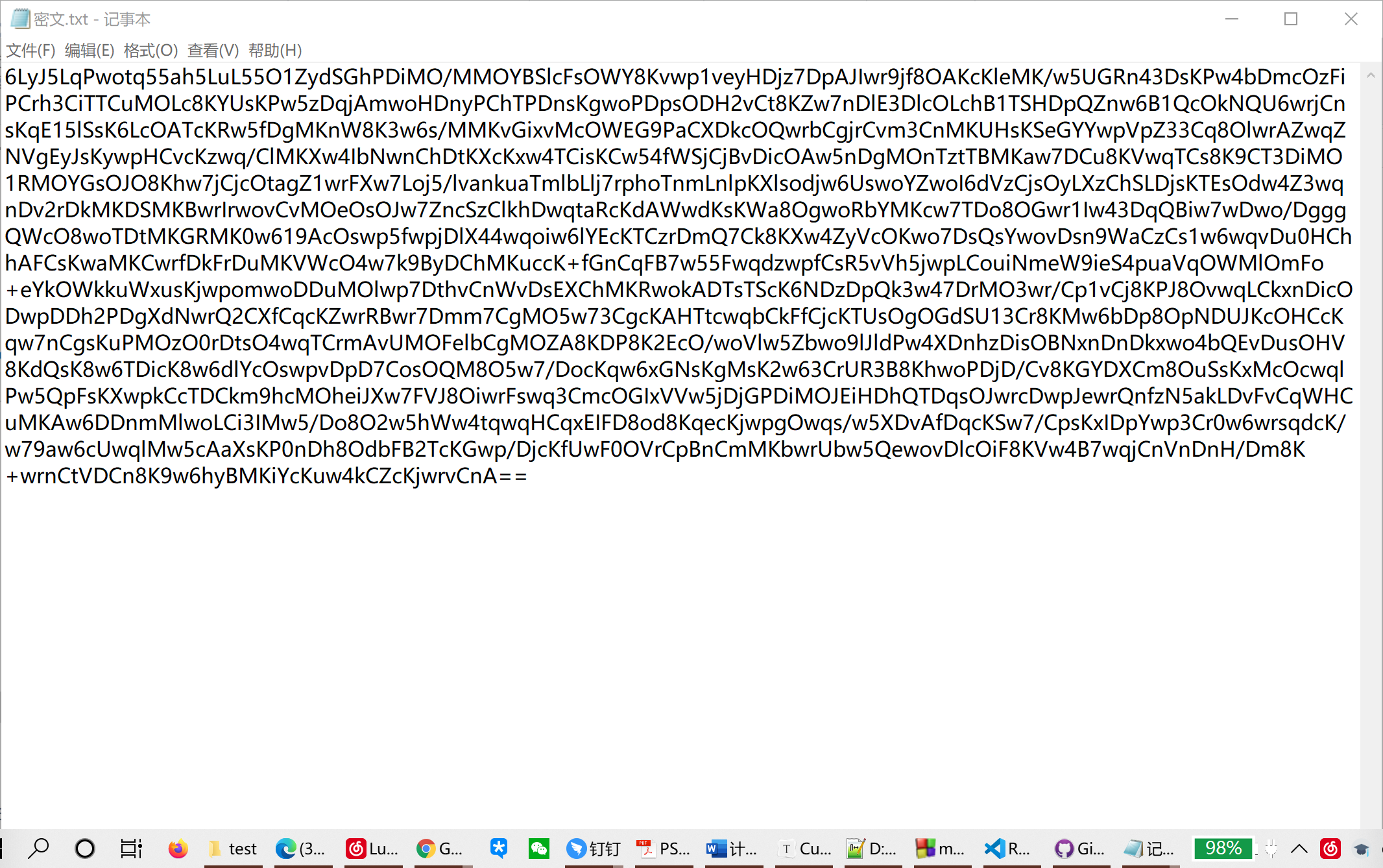


### 密钥流



### 密文

密文已经转换为base64编码格式保证可读性。



### 解密文件



可见已经解密成功。

# 总结

本次实验加深了我对RC4的理解。我个人认为RC4最有意思的地方在于其可通过简单的交换实现加密和解密，其没有用到很深的数学知识。本次实验是我在网络安全课上做的第一个实验，让我对网络安全领域有一个初步的认识。在实现RC4时候由于我搞清楚了RC4的整个流程，因此我并没有用网络现成的代码来实现，而是自己写了一个RC4的函数来实现其算法过程。