# 实验目的

通过实验，让学生充分理解和掌握 DES、RSA算法的原理。

# 实验内容

1. 根据RSA和DES算法的原理实现对文本内容的加密与解密；
2. 使用socket编写加密通信程序，加解密算法使用DES算法，通过RSA算法自动分配密钥，并可正确加解密消息；
3. 利用Wireshark等抓包软件抓取通信过程中的数据包，并进行分析。

# 实验原理

## （1）RSA算法原理

RSA算法它是第一个既能用于数据加密也能用于数字签名的公钥算法。它易于理解和操作，也很流行。算法的名字以发明者的名字命名：Ron Rivest, Adi Shamir 和Leonard Adleman。但RSA的安全性一直未能得到理论上的证明。它经历了各种攻击，至今未被完全攻破。

RSA算法的步骤如下：

1 ）选择两质数 p、q。

2 ）计算 n = p\*q。

3 ）计算 n 的欧拉函数 φ(n)=(p-1)\*(q-1)。

4 ）选择整数 e ，使e 与φ(n)互质，且 1<e<φ(n)。

5 ）计算 d ，使d\*e=1 mod φ(n)。

其中，公钥 KU ＝{e,n} ，私钥 KR ＝{d,n} 。

利用RSA 加密，首先需将明文数字化。对于明文块M 和密文块 C 加/解密的形式如下：

加密：C=M^e mod n。

解密：M＝C^d mod n。

为了更详细地说明这个算法，下面我们举个实例：

取两个质数p=11，q=13，p和q的乘积为n=p×q=143，算出欧拉函数φ(n)=(p-1)×(q-1)=120；再选取一个与φ(n)=120互质的数，例如e=7，则公开密钥=（n，e）=（143，7）。对于这个e值，可以算出其逆：d=103。因为e×d=7×103=721，满足e×d mod φ(n) =1；即721 mod 120=1成立。则秘密密钥=（n，d）=（143，103）。

注：可参考[链接](https://blog.csdn.net/ws_kfxd/article/details/86426306)。

## （2）DES算法原理

DES( Data Encryption Standard)算法，于1977年得到美国政府的正式许可，是一种用56位密钥来加密64位数据的分组加密算法。虽然56位密钥的DES算法已经风光不在,而且常有用DES加密的明文被破译的报道,但是了解一下昔日美国的标准加密算法总是有益的,而且目前DES算法得到了广泛的应用,在某些场合,她仍然发挥着余热。

图1展示了DES的加密过程：

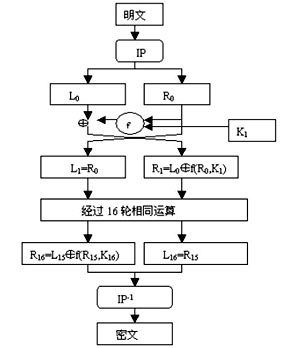
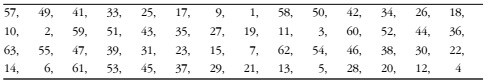


图1 DES的加密过程

DES算法具体通过对明文进行一系列的排列和替换操作来将其加密。过程的关键就是从给定的初始密钥中得到16个子密钥的函数。要加密一组明文，每个子密钥按照顺序（1-16）以一系列的位操作施加于数据上，每个子密钥一次，一共重复16次。每一次迭代称之为一轮。要对密文进行解密可以采用同样的步骤，只是子密钥是按照逆向的顺序（16-1）对密文进行处理。

DES算法的第一步就是从初始密钥中计算得出16个子密钥。图示2展示了这个过程。DES使用一个56位的初始密钥，但是这里提供的是一个64位的值，这是因为在硬件实现中每8位可以用于奇偶校验，在软件实现中多出的位只是简单的忽略掉。要获得一个56位的密钥，可以执照表1的方式执行密钥转换。解释一下表1，按照从左往右从上往下的方式看，表格中每个位置P包含初始密钥中位在转换后的密钥中所占的位置。比如，初始密钥中的第57位就是转换后的密钥中的第1位，而初始密钥中的第49位则变成转换后的密钥中的第2位，以此类推。（数据位的计数顺序按照从左到右从1开始的）。

表1 DES中密钥的转换表



将密钥转换为56位后，接下来计算子密钥。首先，将56位的密钥分为两个28位的组。然后，针对每个子密钥，根据子密钥的序列值（也就是16个子密钥中的第几个）旋转这两组值（旋转的位数见表2），然后重新合并。之后，再按照表3所示对重组后的密钥进行置换，使56位的子密钥缩小为48位（注意表3只有48位，丢弃了8位）。这个排列过程就称为置换选择。

针对16个子密钥，每个子密钥重复一次该过程。这里的目的是保证将初始密钥中的不同位在每一轮排列后应用于加密的数据上。

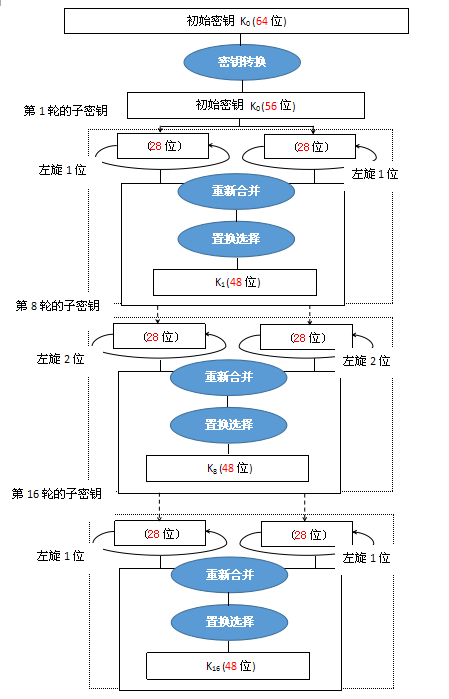
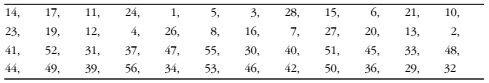


图2 DES计算子密钥的过程

表2 针对DES子密钥每一轮的旋转次数

https://images2018.cnblogs.com/blog/1281268/201807/1281268-20180721074351257-192188283.jpg

表3 DES子密钥的置换选择



经过上述过程，我们已经准备好了子密钥。接着就可以加密和解密数据块了。图3展示了这个过程：

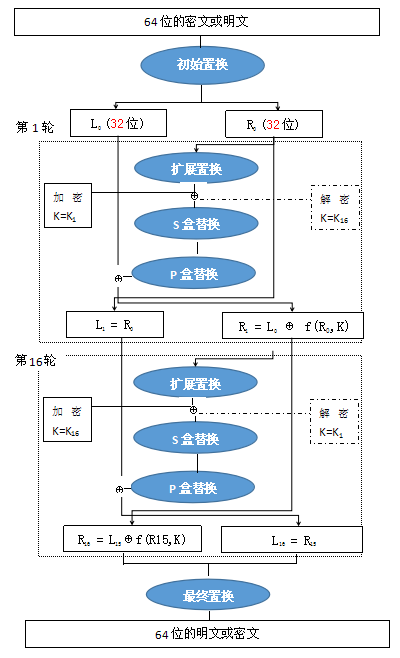
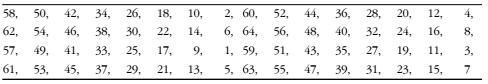


图3 DES中加密和解密数据块

从表4所示的方式置换64位的数据块开始，该置换过程称为初始置换。该过程并不会增加DES的安全性，但这种做法在16位和32位的总线出现之前将使得数据更容易加载到DES芯片中。虽然这种处理已经不合时宜，但该置换过程仍然保留以满足DES标准。

表4 DES中数据的初始置换



经过初始置换后，64位的数据块分为两个32位的组，和。

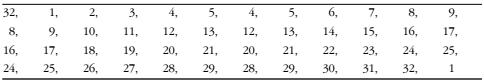
完成初始置换后，数据块将重复执行16轮一系列的操作。每一轮操作(i)的目的是计算出和 ，这些结果将用在下一轮操作中直到最终得到数据和。

每一轮以和开始，然后根据表5所示进行扩展置换，将从32位扩展到48位。该置换的主要目的是在加密数据的过程中制造一些雪崩效应，使用数据块中的1位将在下一步操作中影响更多位，从而产生扩散效果。

一旦扩展置换完成，计算出48位的结果值与这一轮子密钥的异或值（，符号计为）。这将产生48位的中间值，记为。

如果将计为扩展置换的结果，则本轮到目前为止的操作可以表示为：

表5 DES中数据块的扩展置换



下一步， 需要通过8个单独的S盒执行8次替换操作。每个S盒（j）从 的6j 到 6j+6 的位置取出6位，并为其在表6中查出1个4位的值，将该值写到缓冲区的4j位置处（如图4）。

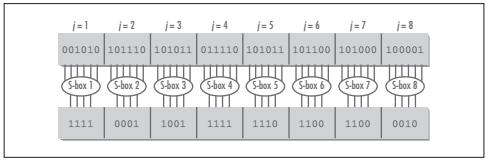
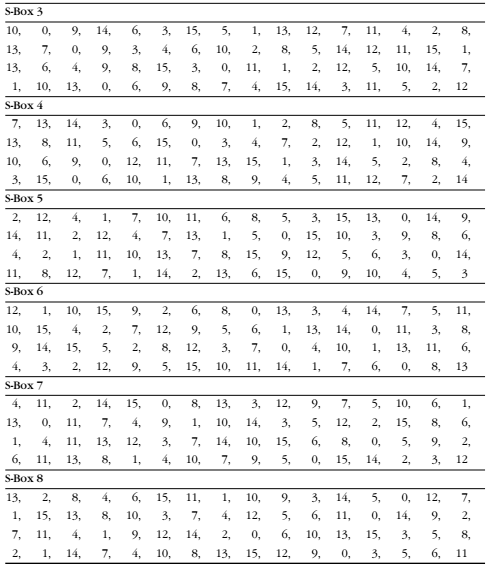


图4 DES中的8个S盒

读表6，查找S盒（j）。通过前面取出的6位值，根据第1位和最后1位组成的2位值找到表6中的行号，而根据中间剩下的4位来确定表6中的列号。比如，在图4中，中的第3个6位组是101011。因此，在表6中查找到的第3个S盒是9。因为行号等于= 3，列号等于= 5（查表时从索引0开始计数）。S盒为数据增加了不确定性，除了给DES带来安全性外，没什么特别的。

表6 DES中数据块的S盒替换



一旦完成了S盒替换，得到的结果又变为一个32位的值。接下来再通过P盒来置换。如下表7所示：

表7 DES中数据块的P盒置换

https://images2018.cnblogs.com/blog/1281268/201807/1281268-20180721091607493-694625718.png

到目前为止，我们把这一轮的操作想象为一个函数，一般记为*f*。如果代表中的第个6位组，代表第*j*个S盒，而P代表P盒置换，则该函数可以定义为：

每一轮的最后一个操作是计算 *f* 的32位结果值与传入本轮操作的原始数据的左分组之间的异或值。

一旦完成，将左右两个分组交换然后开始下一轮。

在最后一轮中，不用交换左右分组。

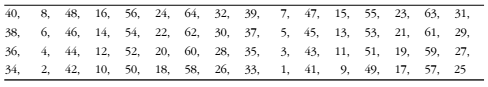
把所有的步骤连起来，在每一轮中计算和的步骤可以精确表示为：

当全部的16轮操作都结束后，将最后的右分组和最后剩下的左分组连接起来，组成一个64位的分组。

最后一步是将按照表8所示的置换进行置换。简而言之，就是撤消之前的初始置换。

加密数据时，最终结果就是一个64位的密文，而当解密数据时，最终结果就是64位的明文了。

表8 DES中数据块的最终置换



注：可参考[链接](https://www.cnblogs.com/idreamo/p/9333753.html)

## 加密通信设计

* 1. 通信过程如图5所示：

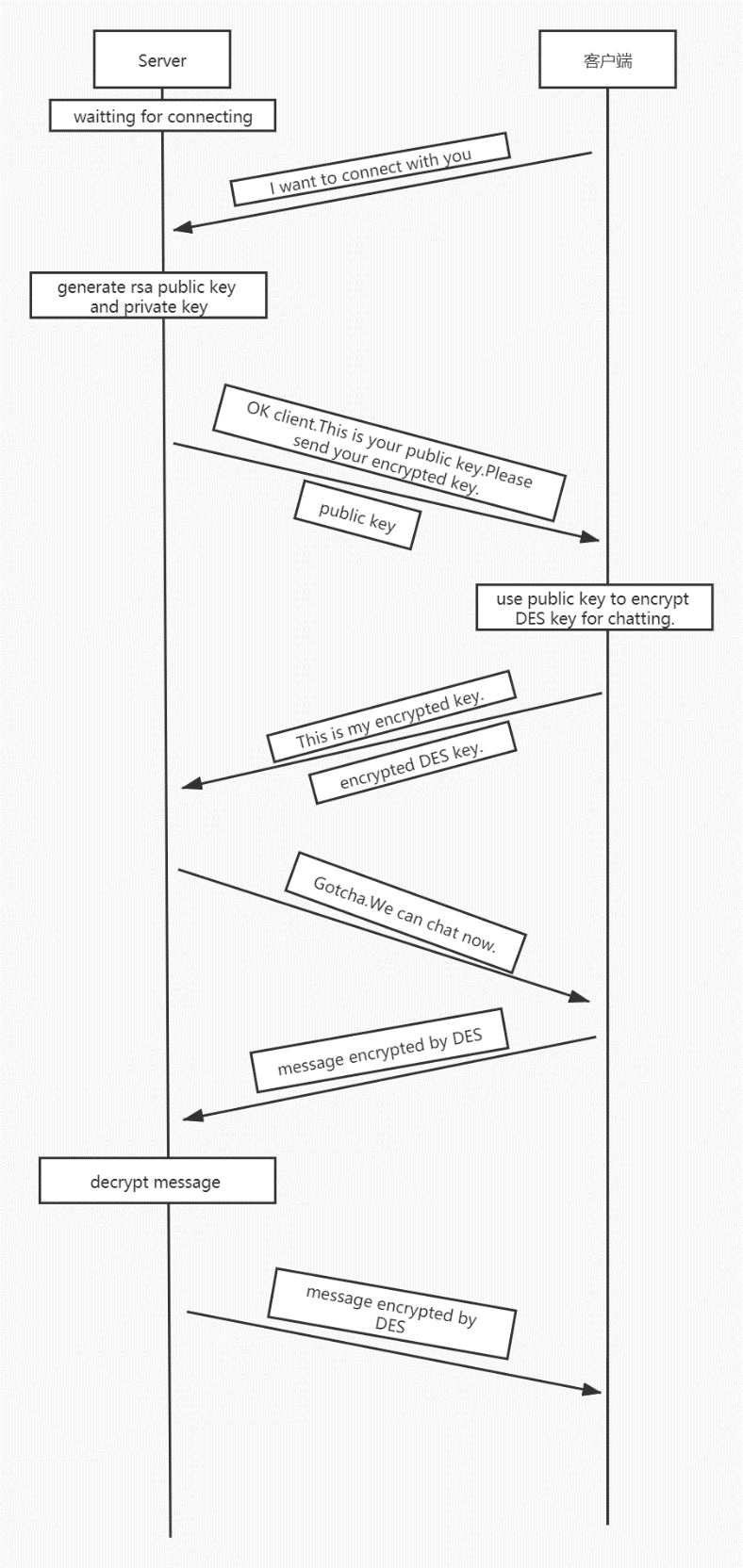


图5 加密通信流程图

* 1. Server端一直等待Client的连接。

3）当一个client需要连接server时，向server发送“Hello server, I want to connect with you”。

4）server接收到消息后，开始为这个client生成一个RSA的公私钥对，并将公钥发送给client，并告诉client“OK client. This is your public key. Please send your encrypted key.”

5）client接收到消息后，将自己的DES的密钥用server给的公钥进行加密，并将加密的结果发送给server，告诉server“This is my encrypted key.”

6）server接收到client发送的加密后的DES密钥，用RSA私钥对此进行解密，从而得到client的DES密钥，告诉client“Gotcha. We can chat now.”

7）client接收到server可以聊天的消息后，便可以开始和server发送消息进行交流，并且每个消息都会用自己的DES进行加密，然后server接收到消息以后可以使用DES密钥进行解密。

# 四、实验环境

（1）python 3运行环境

（2）Wireshark等抓包工具

# 五、实验步骤

（1）安装pycryptodome：pip3 install pycryptodome。pycryptodome是python一个强大的加密算法库，可以实现常见的单向加密、对称加密、非对称加密和流加密算法。如果通过pip3安装pycryptodome时网络速度很慢，可以通过在终端输入：

pip3 config set global.index-url <https://pypi.tuna.tsinghua.edu.cn/simple>

指令来将软件源更换为清华源，也可以自行配置为国内的其它源，如阿里云，中国科技大学，豆瓣等。

（2）安装Wireshark抓包工具，直接去[官网](https://www.wireshark.org/download.html)下载安装包安装即可。

（3）使用提供的DES加密程序（也可以自己实现），实现对文本数据的加密，要求输出加密后的密文和解密后的明文。

（4）使用Crypto生成私钥和公钥：

from Crypto import Random

from Crypto.PublicKey import RSA

random\_generator = Random.new().read

rsa = RSA.generate(2048, random\_generator)

# 生成私钥

private\_key = rsa.exportKey()

print(private\_key.decode('utf-8'))

# 生成公钥

public\_key = rsa.publickey().exportKey()

print(public\_key.decode('utf-8'))

​

with open('rsa\_private\_key.pem', 'wb')as f:

f.write(private\_key)

with open('rsa\_public\_key.pem', 'wb')as f:

f.write(public\_key)

（5）使用公钥对数据进行加密，再使用私钥对加密后的数据进行解密，要求输出加密后的密文和解密后的明文：

def get\_key(key\_file):

with open(key\_file) as f:

data = f.read()

key = RSA.importKey(data)

return key

def encrypt\_data(msg):

public\_key = get\_key('rsa\_public\_key.pem')

cipher = PKCS1\_cipher.new(public\_key)

encrypt\_text = base64.b64encode(cipher.encrypt(bytes(msg.encode("utf8"))))

return encrypt\_text.decode('utf-8')

def decrypt\_data(encrypt\_msg):

private\_key = get\_key('rsa\_private\_key.pem')

cipher = PKCS1\_cipher.new(private\_key)

back\_text = cipher.decrypt(base64.b64decode(encrypt\_msg), 0)

return back\_text.decode('utf-8')

（6）根据设计好的流程实现加密通信程序，主要使用的是websockets库进行socket连接，辅以asyncio库用来进行线程等待。

（7）server端和client端的输出结果类似下图所示：

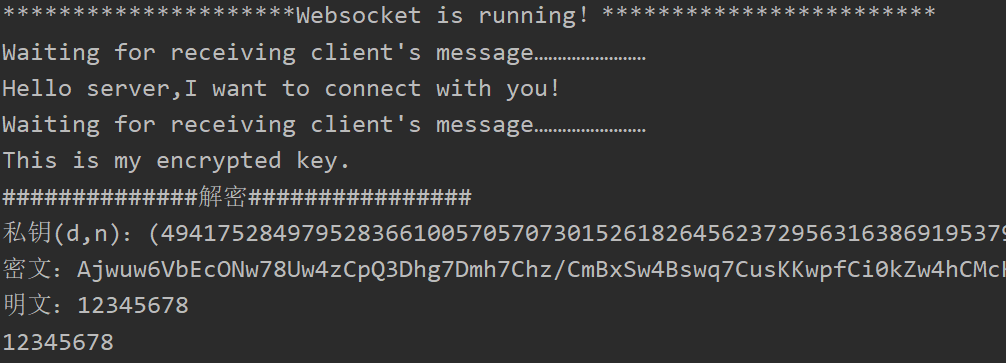


图6 server端输出示例

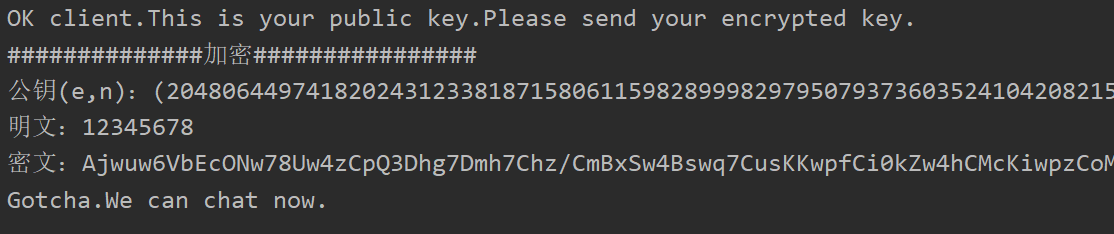


图7 client端输出示例

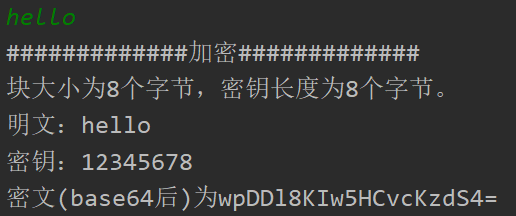


图8 client向server发送hello

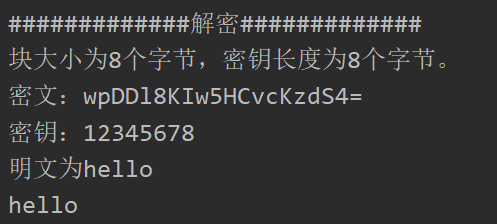


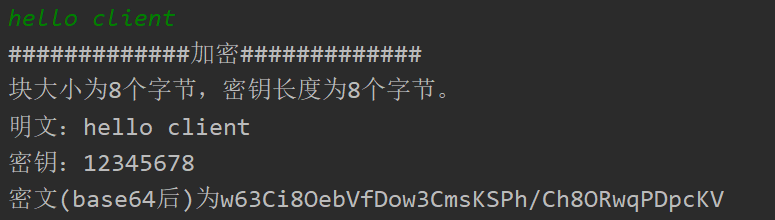
图9 server接收到消息  


图10 server向client发送hello client

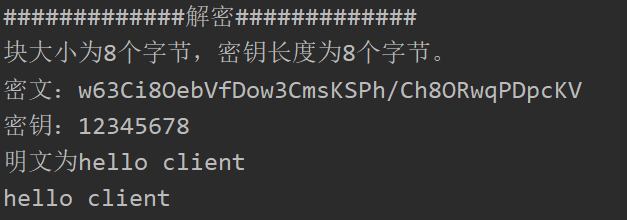
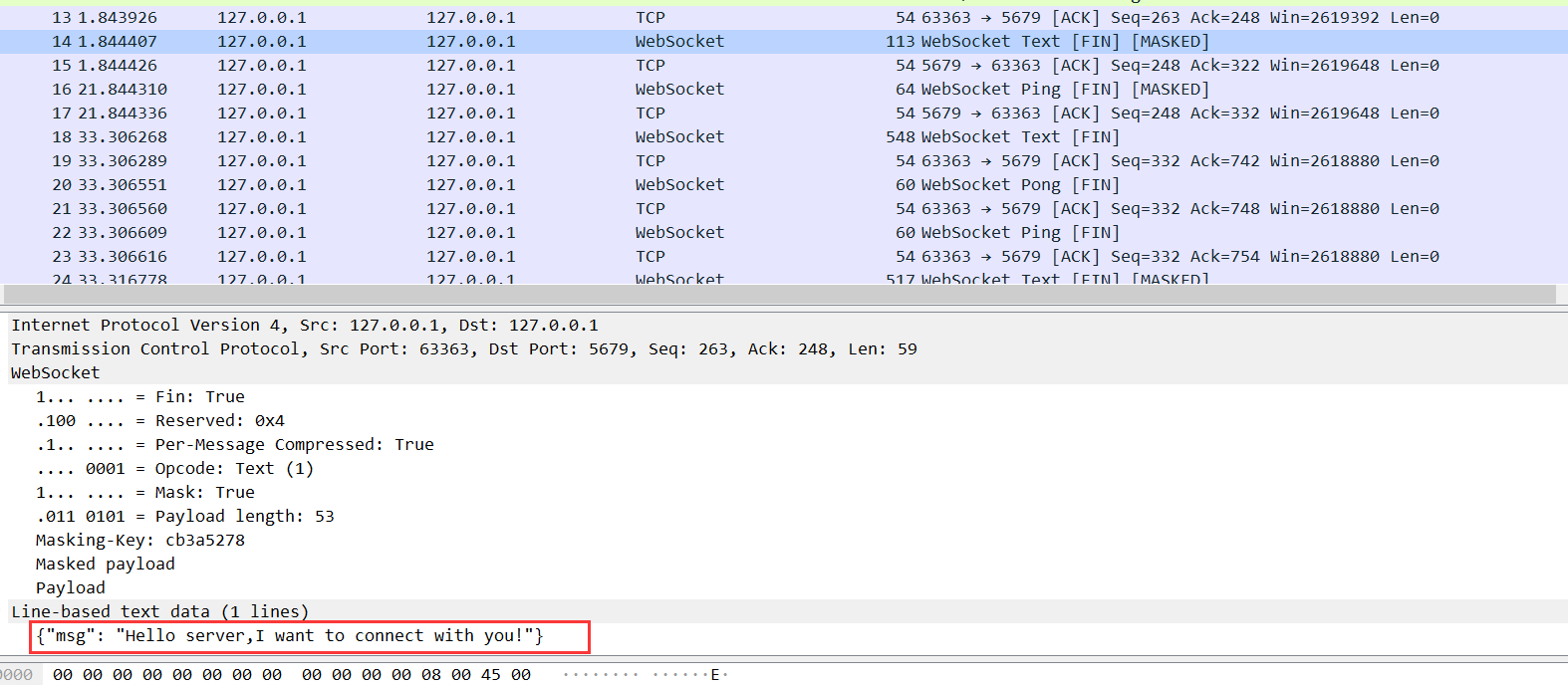
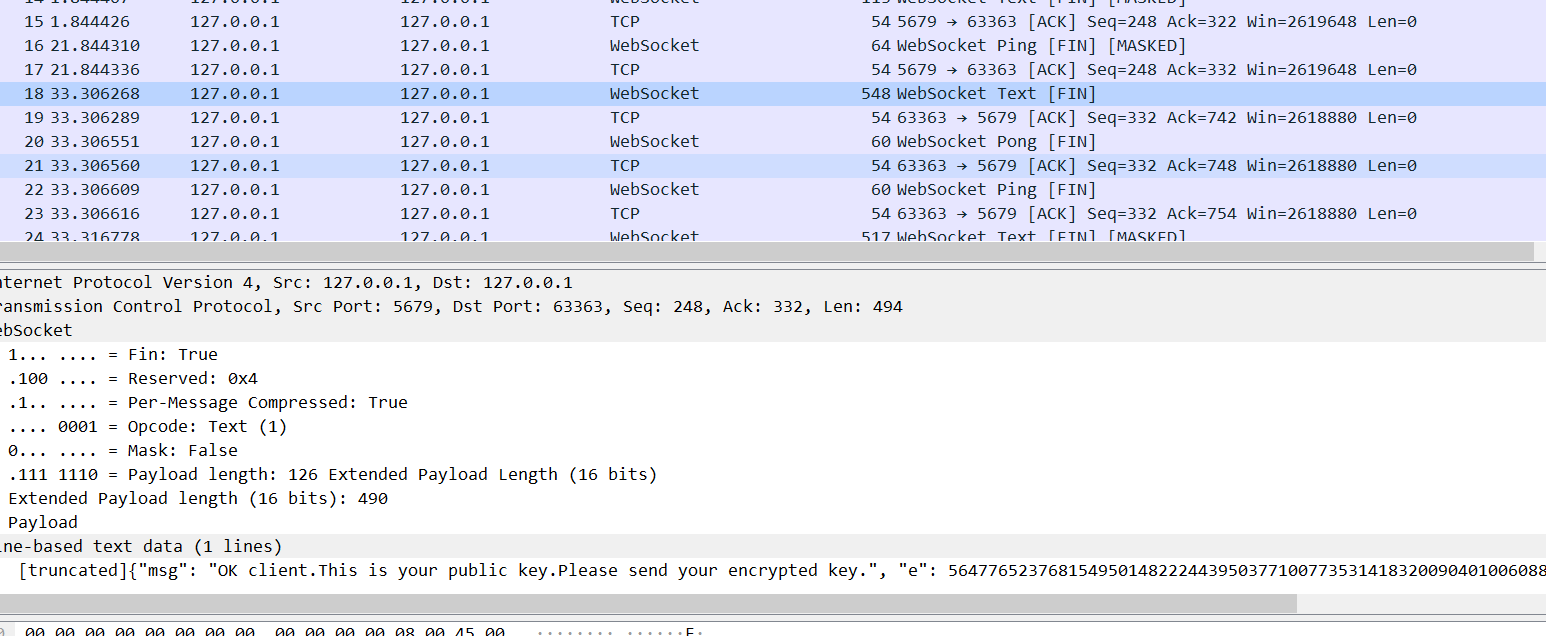
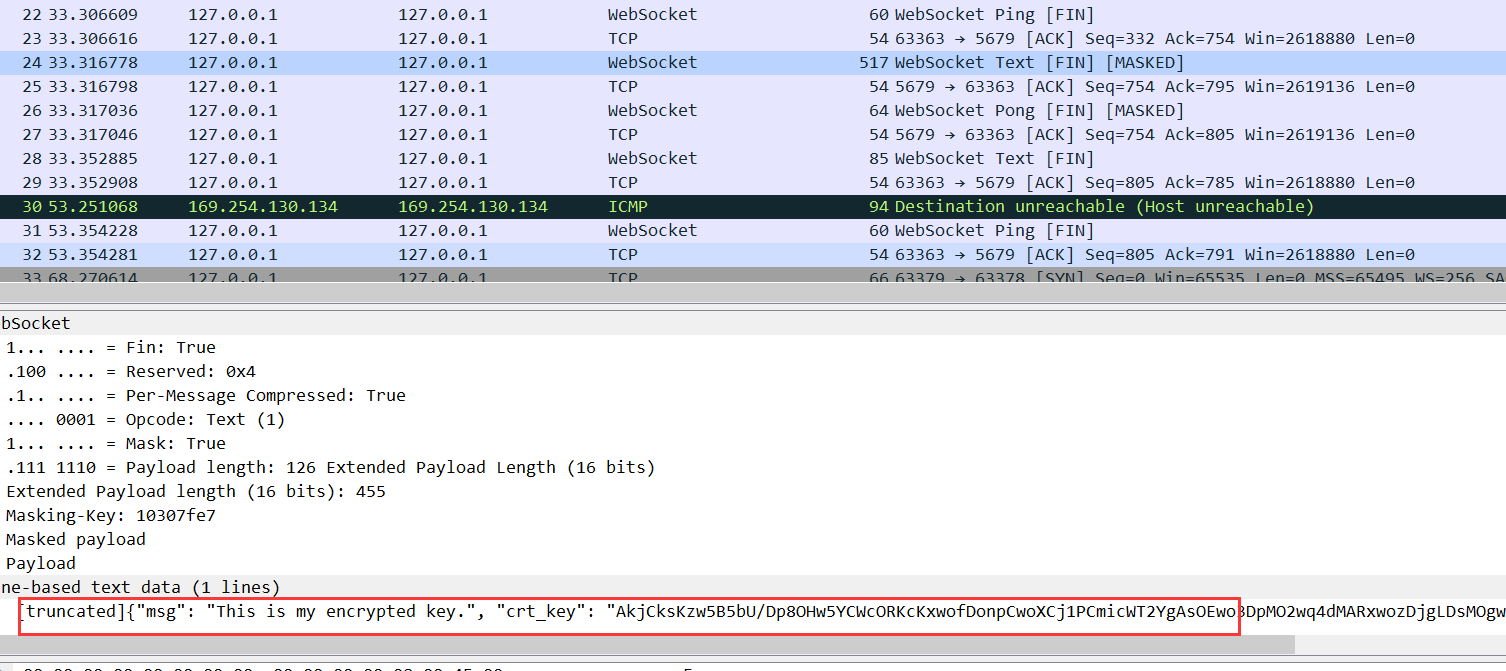


图11 client接收消息

（8）使用Wireshark抓包结果如下：







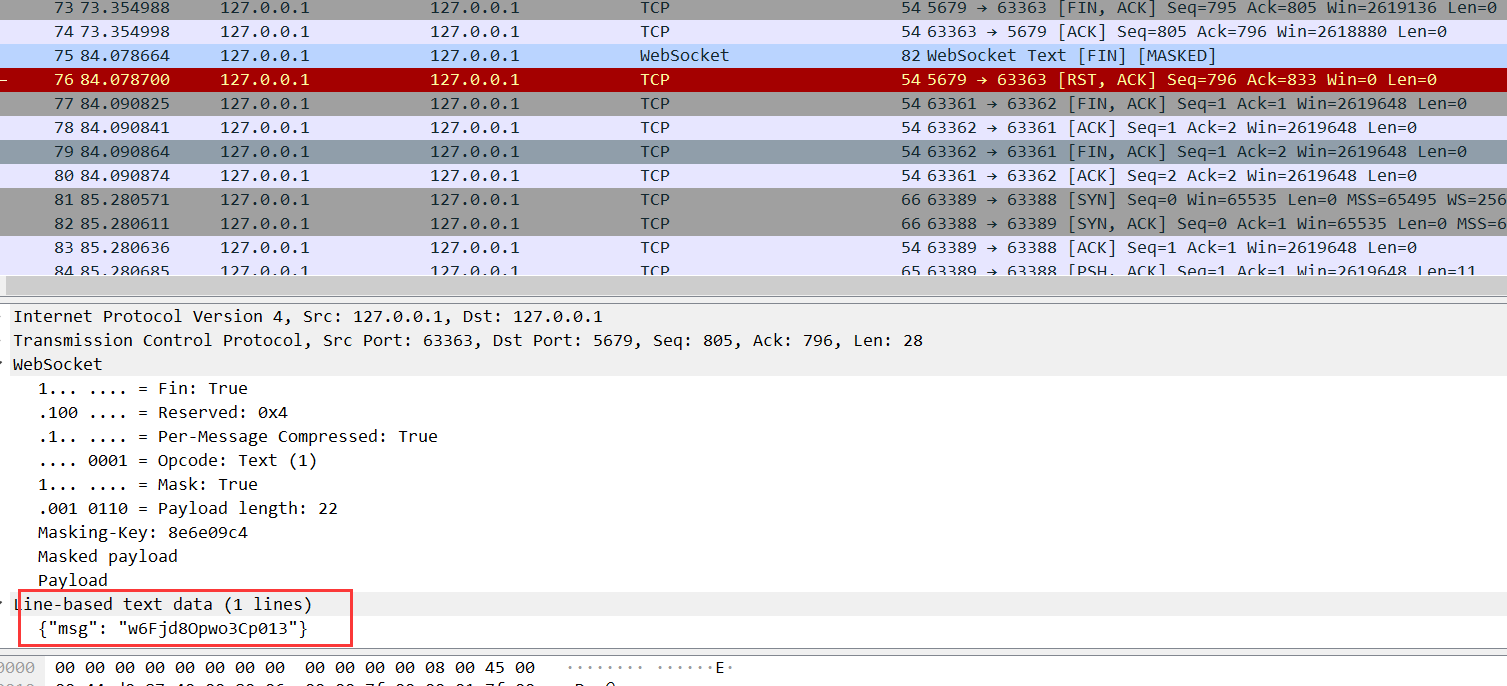


图12 抓包结果