华中科技大学软件学院

实验报告书

课程名：《信息系统安全》

题 目： Des加密，公钥加密，数据包捕获

班 级： 软工1503班

学 号： U201517045

姓 名： 忻犁

评语：

成绩： 指导教师：

批阅时间： 年 月 日

目录

[实验1.1 对称分组密码实验 2](#_Toc516223901)

[1.实验目的（DES和AES算法二选一） 2](#_Toc516223902)

[2.实验内容 2](#_Toc516223903)

[3.算法实现过程 2](#_Toc516223904)

[4.运行截图 7](#_Toc516223905)

[实验1.2公钥密码实验 12](#_Toc516223906)

[1. 实验目的 12](#_Toc516223907)

[2.实验内容 12](#_Toc516223908)

[3.算法实现 12](#_Toc516223909)

[4.运行截图 15](#_Toc516223910)

[实验2 数据包捕获和分析 18](#_Toc516223911)

[1. 实验目的 18](#_Toc516223912)

[2. 实验内容 18](#_Toc516223913)

[3.实验步骤 19](#_Toc516223914)

[3.3 对数据包进行分析 23](#_Toc516223915)

[3.4 对QQ官网登录界面的数据包分析 24](#_Toc516223916)

## 实验1.1 对称分组密码实验

### 1.实验目的（DES和AES算法二选一）

1.理解对称加密算法的原理和特点

2.理解DES算法的加密原理和使用模式

3理解AES算法的加密原理和使用模式

### 2.实验内容

1．DES加密解密字符串

2．DES算法CBC模式加密文件

3．AES加密解密字符串

4．AES算法CBC模式加密文件

### 3.算法实现过程

这一次的实验Des算法我没有用现成的算法，而是自己按照算法原理写了一遍，下面介绍具体的实现过程。

#### ****3.1.所需参数****

key：8个字节共64位的工作密钥

data：8个字节共64位的需要被加密或被解密的数据

在实际应用中需要加密的明文应该是不限定字长的，所以按照8个8个将明文分组之后，最后的不足8位部分按照末尾补0的方式补全。

#### 3.2处理数据

Des算法在加密，解密过程中都是对二进制数据进行运算，所以需要将输入的字符串类型的数据转换成二进制形式。

String str=Integer.*toBinaryString*(temp[i]);

**int** binaryint=Integer.*parseInt*(str);

result+=String.*format*("%08d", binaryint);

这边的代码将string类型的数据转换成对应的二进制形式并且高位补0，一个字符对应8位的二进制码。

#### 3.3初始置换

DES算法使用64位的密钥key将64位的明文输入块变为64位的密文输出块，并把输出块分为L0、R0两部分，每部分均为32位。

具体的置换规则就不列出了，就是一个置换的矩阵。而且置换算法的实现也没有什么特别的地方，置换完成之后将左，右半部分存储起来，作为后面16轮迭代的输入。

**for**(**int** i=0;i<64;++i){

result[i]=arr[*replacement*[i]-1];

}

**for**(**int** i=0;i<32;++i){

*left*[i]=result[i];

*right*[i]=result[i+32];

}

这边的置换比较简单，单纯的从置换矩阵找对应的数据赋值就可以了。

#### 3.4迭代过程

这一步是Des算法的核心，Des算法经过16轮的迭代，将密钥和明文结合起来，最后得到一个加密后的64位数据。

每轮迭代的过程可以表示如下：

Ln = R(n - 1)；

Rn = L(n - 1)⊕f(Rn-1,kn-1)

可以看到，每一次迭代，左半部分直接赋值为上一次迭代的右半部分，而右半部分则需要用上一轮的右半部分和密钥生成的每一轮的子密钥进行一个f运算再和上一轮的左半部分异或得到结果。

为了实现这个迭代过程，需要将密钥进行处理，得到每一轮对应的子密钥。

###### 3.4.1 子密钥生成

首先需要将密钥数据（64位）经过一个置换矩阵降至56位，再将得到的56位矩阵分成C0和D0。

**for**(**int** i=0;i<28;++i){

*C0*[i]=arr[*pc\_1*[i]-1];

}

**for**(**int** i=0;i<28;++i){

*D0*[i]=arr[*pc\_1*[i+28]-1];

}

这一步做完之后，就得到了第一次生成子密钥的输入，之后需要做的是循环左移，将每一轮得到的Ci,Di循环左移。每一轮对应的迭代的循环左移位数为：



也就是说，第1，2，9，16轮循环需要循环左移一位，其余的轮次循环左移两位。

**if**(round==0||round==1||round==8||round==15){ //如果是第1，2，9，16轮就循环左移1位

**int** temp1=*C0*[0],temp2=*D0*[0];

**for**(**int** i=0;i<27;++i){

*C0*[i]=*C0*[i+1];

*D0*[i]=*D0*[i+1];

}

*C0*[27]=temp1;

*D0*[27]=temp2;

}

**else**{ //其余循环左移2位

**int** temp1=*C0*[0],temp2=*C0*[1],temp3=*D0*[0],temp4=*D0*[1];

**for**(**int** i=0;i<26;++i){

*C0*[i]=*C0*[i+2];

*D0*[i]=*D0*[i+2];

}

*C0*[26]=temp1;

*C0*[27]=temp2;

*D0*[26]=temp3;

*D0*[27]=temp4;

}

循环左移完成之后，将得到的Ci,Di合并（56位），再经过置换表（48位）得到48位子密钥Ki（48位）。

###### 3.4.2扩展置换E

可以看到上一步得到的子密钥是48位的数据，所以需要将右半部分数据Ri（32位）同样扩展到48位。



**for**(**int** i=0;i<48;++i){

*R0*[i]=r[*rule\_e*[i]-1];

}

可以看到，这边重复了一些位，然后将右半部分扩展到了48位。

###### 3.4.3 S盒计算

首先需要得到S盒的输入，也就是上一步扩展的右半部分与之前得到的子密钥异或。

**for**(**int** i=0;i<48;++i){

*xorsresult*[i]=r[i]^k[i];

}

然后将得到的48位结果分成8组，每一组6位进行S盒计算。计算规则是将6位中的首位和末位结合作为S盒的行数，中间四位作为列数，然后根据行数，列数找出对应的输出（4位）。这样将每一组的输出合并之后，得到32位的结果。

**for**(**int** i=0;i<8;++i){

row=xor[i\*6]\*2+xor[i\*6+5];

column=xor[i\*6+1]\*8+xor[i\*6+2]\*4+xor[i\*6+3]\*2+xor[i\*6+4];

result=*S*[i][row\*16+column];

*Sresult*[i\*4]=result/8;

*Sresult*[i\*4+1]=(result-*Sresult*[i\*4]\*8)/4;

*Sresult*[i\*4+2]=(result-*Sresult*[i\*4]\*8-*Sresult*[i\*4+1]\*4)/2;

*Sresult*[i\*4+3]=result-*Sresult*[i\*4]\*8-*Sresult*[i\*4+1]\*4-*Sresult*[i\*4+2]\*2;

}

这边也没有特别的难的地方，只要是一些二进制和十进制的转换。

###### 3.4.4 P盒计算

将得到的32位S盒输出作为P盒计算的输入，P盒的计算是经过一个置换表，然后将置换结果与数据左半部分异或，然后左右部分交换，最后进入下一轮迭代。（最后一次迭代不需要交换左右部分）。

**for**(**int** i=0;i<32;++i){

*Presult*[i]=s[*P*[i]-1];

}

**for**(**int** i=0;i<32;++i){

l[i]=l[i]^p[i];

}

**for**(**int** i=0;i<32;++i){

temp=l[i];

l[i]=r[i];

r[i]=temp;

}

三个循环实现了置换，异或，交换。然后将进入下一轮迭代。

###### 3.4.5 迭代过程整合

将每一步的迭代过程实现之后，需要将迭代整合成一个函数，方便调用。

**public** **static** **void** f(**int** round,**int** [] key){ //16轮的迭代

**if**(round<16){

*createkey*(key,round);

*expand*(*right*);

*xorS*(*R0*,*subkey*[round]);

*boxS*(*xorsresult*);

*boxP*(*Sresult*);

*xorP*(*left*,*Presult*);

*exchange*(*left*,*right*);

round++;

*f*(round,key);

}

**else** **if**(round==16){ //16轮完成之后将左右互换之后做逆置换

*exchange*(*left*,*right*);

*inversement*(*left*,*right*);

}

}

可以看到，每一轮的迭代需要依次经过子密钥生成，扩展，异或，S盒计算，P盒计算。

#### 3.5 逆置换

这一步的结果就是密文了，经过16轮的迭代，得到了最后的数据左半部分L16和右半部分R16。将这两部分合并之后作为逆置换的输入，逆置换通过一个置换表（与初始置换相反）得到密文（64位）。

**for**(**int** i=0;i<32;++i){

temp[i]=l[i];

temp[i+32]=r[i];

}

**for**(**int** i=0;i<64;++i){

*ciphertext*[i]=temp[*inverse*[i]-1];

}

由于输出是16进制的，所以还需要做一步将二进制密文转换为16进制密文。

**for**(**int** j=0;j<16;++j){

**for**(**int** i=0;i<4;++i){

sum+=Math.*pow*(2,3-i)\*t[i+j\*4];

}

result+=*hexStr*.charAt(sum);

sum=0;

}

这一步完成之后，加密过程就大功告成了。

###### 3.6 加密算法整合

最后一步，将加密算法整个整合成一个函数。

这里还需要考虑一个问题，就是输入的明文并不是正好的8个字符，所以需要先将字符串按照8个一组切割。

**for**(i=0;i<t.length()/8;++i){

text=t.substring(8\*i,8\*i+8);

*s*=i\*8+8;

*replace*(*toBinary*(text));

*f*(0,*toBinary*(k));

result+=*binstrtostr*(*ciphertext*);

Demo.*progress*(*s*);

}

**if**(t.length()%8!=0){

text=t.substring(8\*i,t.length());

*s*=t.length();

*replace*(*toBinary*(text));

*f*(0,*toBinary*(k));

result+=*binstrtostr*(*ciphertext*);

Demo.*progress*(*s*);

}

前面一个循环是循环读取8个字符，然后加密，并且将加密的进度传到进度条显示。后面一个循环处理最后一次的数据。

###### 3.7 DES解密

解密的算法与加密一致，唯一的不同在于每一轮迭代用到的子密钥是按照相反的顺序，就是第一轮用K16，第二轮用K15…..

所以实现起来很快，首先将16进制的密文转换成二进制格式。

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

sum=*hexStr*.indexOf(temp[i]);

result[i\*4]=sum/8;

result[i\*4+1]=(sum-result[i\*4]\*8)/4;

result[i\*4+2]=(sum-result[i\*4]\*8-result[i\*4+1]\*4)/2;

result[i\*4+3]=sum-result[i\*4]\*8-result[i\*4+1]\*4-result[i\*4+2]\*2;

}

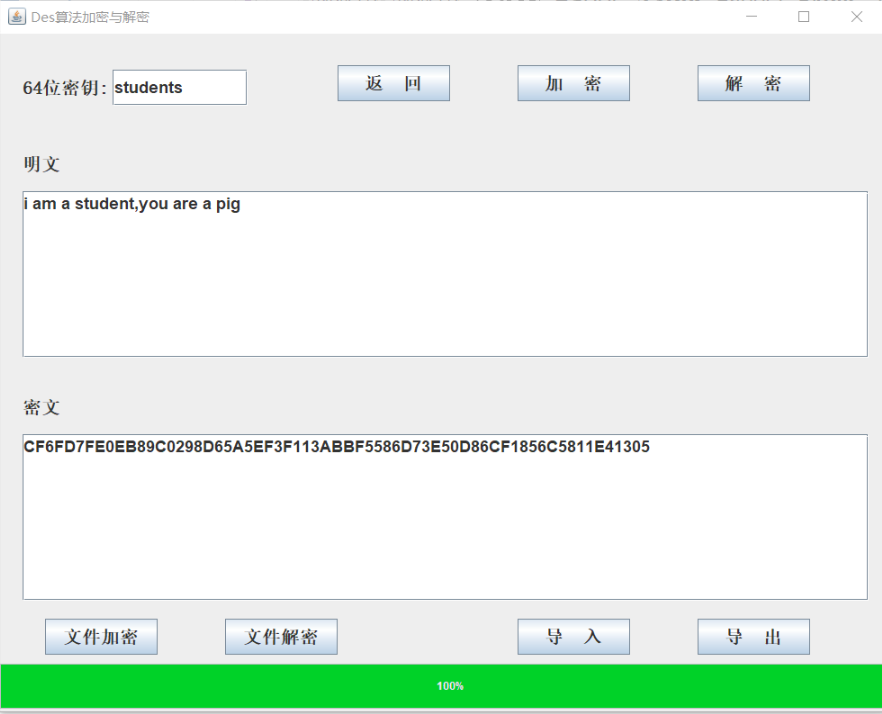
然后只需要修改f函数中子密钥部分，其余都和加密一样，所以就不列举代码了。

### 4.运行截图

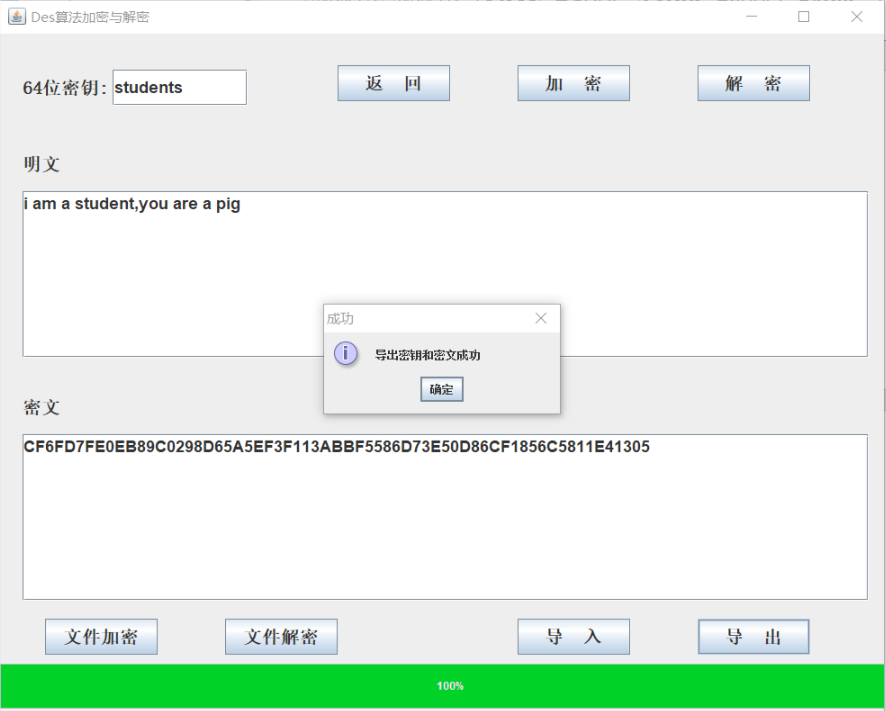
首先进入主界面，我这边整合了三个实验，所以选择Des加密。



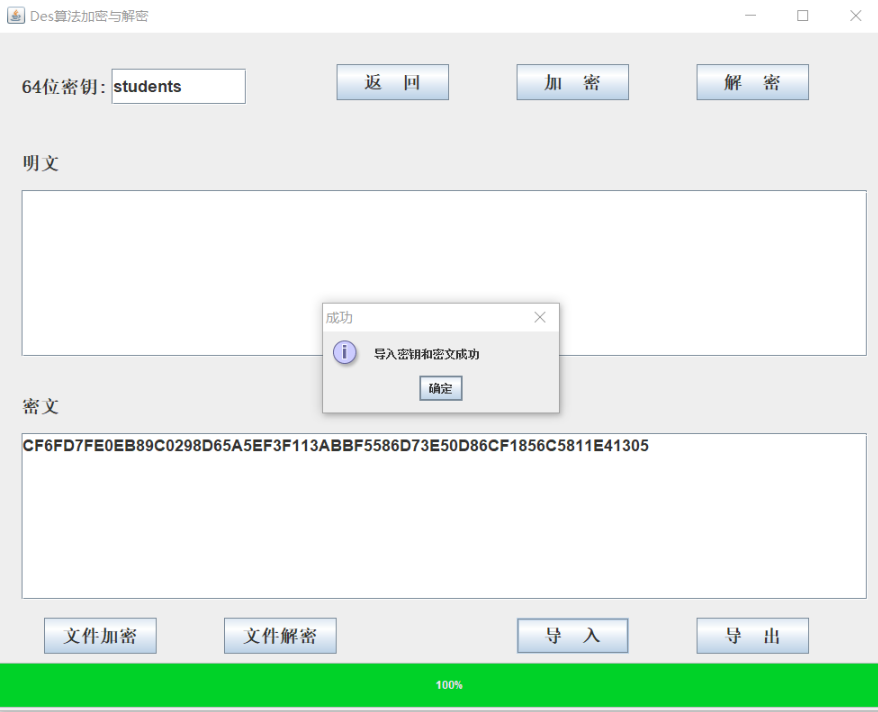
进入了Des加密界面之后，在64位密钥区域输入密钥“students”，然后输入明文”I am a student,you are a pig”。点击加密，得到密文。



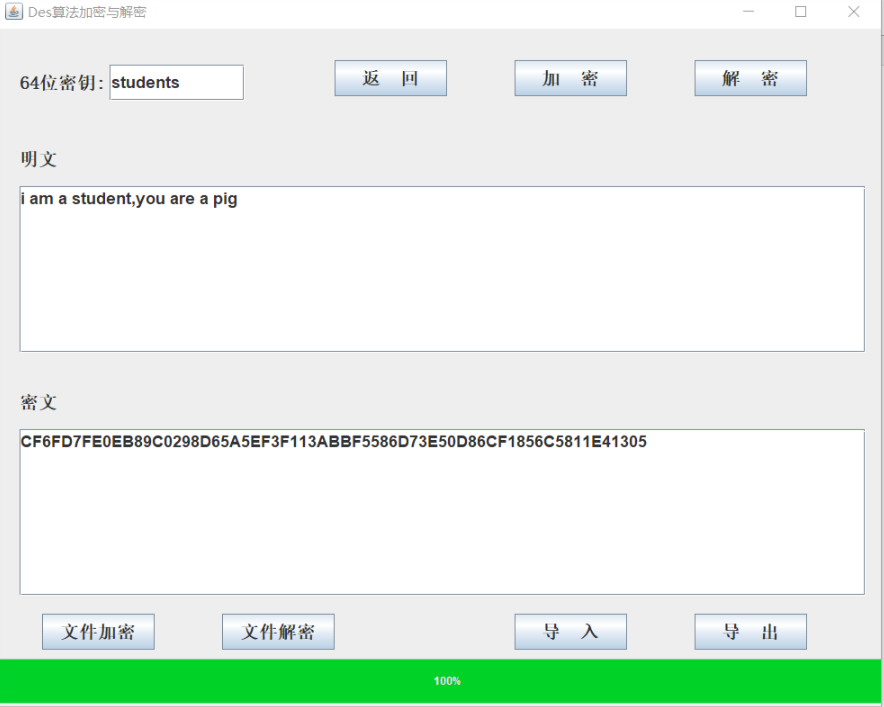
点击导出，将得到的密文导出到对应文件（ciphertext.txt），并将密钥导出到对应文件(key.txt)。



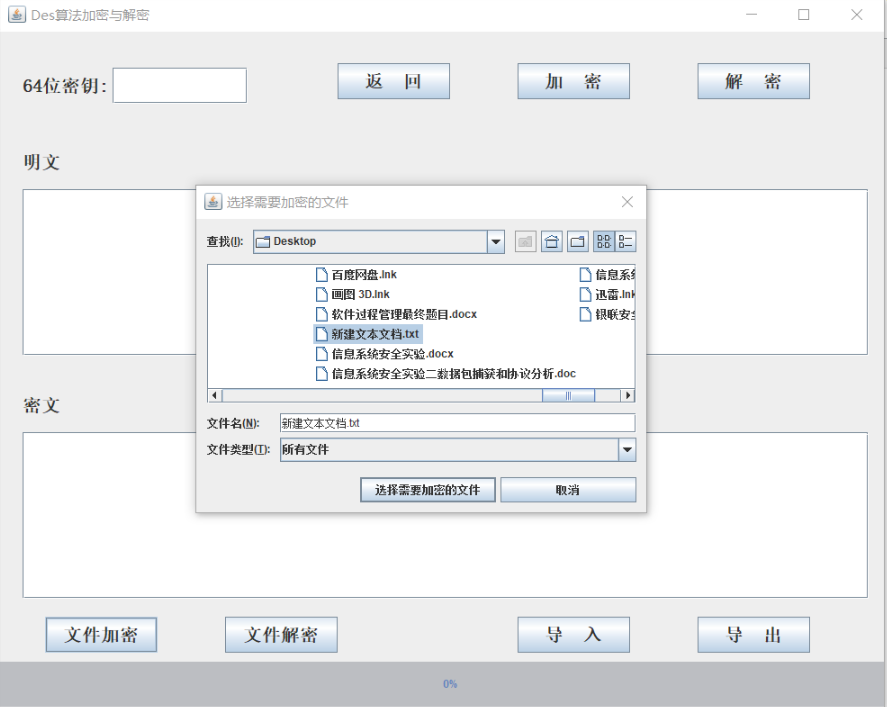
再点击导入，将ciphertext.txt和key.txt中的内容导入到密文区域和密钥区域。



点击解密，得到明文。



点击文件加密，选取准备好的英语作文。



输入密钥”students”，点击加密，得到密文。



同样导出密文之后，再导入密文，点击解密，得到明文。



## 实验1.2公钥密码实验

### 实验目的

1.了解非对称加密机制

2.理解RSA算法的加密原理

### 2.实验内容

1．RSA生成公私钥及加密解密过程演示

2．RSA加密解密

### 3.算法实现

公钥加密算法主要是一个次方的运算，需要用到一些技巧去简化次方数。

#### 3.1所需参数

1.素数p,素数q和随机数e。用来生成公钥和私钥。

2.需要加密的明文，需要解密的密文。

注意p,q需要是素数，所以要在一开始判断一下。

**for**(**int** i=2;i<(**int**)Math.*sqrt*(p)+1;++i){

**if**(p%i==0){

i=(**int**)Math.*sqrt*(p)+1;

**return** **true**;

}

}

**return** **false**;

#### 3.2 生成公钥，私钥

1.公钥n和私钥n等于p\*q。

2.ф(n)等于(p-1)\*(q-1)。

3.公钥e,私钥d 满足 (e\*d)mod(ф(n))=1。

前两个实现都不难，第三步的实现需要一点技巧。

假设(e\*d)=(ф(n)\*N)+1。N从1开始遍历，直到找到为止。

*n*=p\*q;

*fn*=(p-1)\*(q-1);

**int** i=1;

**for**(i=1;(*fn*\*i+1)%*e*!=0;i++){

}

*d*=(*fn*\*i+1)/*e*;

经过计算后得到了公钥，私钥。

#### 3.3 公钥加密（演示）

演示部分的加密只需要加密一个数字，所以只需要计算下列算式：

密文c=(m^e)mod(n)。

这边主要的难点就是m^e这个数字肯定是超过了整型的范围，所以需要进行降次，再进行取模运算。

借助取模运算的分配律：(a\*b) mod c=((a mod c)\*(b mod c))mod c

这样可以运用递归的方法把次方降下来。

例如(40^100)mod(200)

可以分解为(40^50mod200 \*40^50mod200)mod200

再降次

((40^25mod200\*40^25mod200)mod200 \*(40^25mod200\*40^25mod200)mod200)mod200

……

直到什么此方呢？

考虑到ascii码最大的数字是127，127^4是在int范围内的，所以降到4此方以内就可以运算了。

**if**(ei<=4){ //直到4次方以内

**return** (**int**)(Math.*pow*(mi, ei)%ni);

}

**else**{ //每次将次方减半

**return** (*encryption*(mi,ni,ei/2)\**encryption*(mi,ni,ei-ei/2))%ni;

}

经过运算之后，就得到了加密得到的密文。

#### 3.4公钥解密（演示）

公钥解密类似公钥加密，不同点在于算是变为了：

明文m=c^d mod n。

还是用到前面的原理，把次方降下来，然后运算。

**if**(di<=4){ //直到4次方以内

**return** (**int**)(Math.*pow*(ci, di)%ni);

}

**else**{ //每次将次方减半

**return** (*encryption*(ci,ni,di/2)\**encryption*(ci,ni,di-di/2))%ni;

}

经过运算之后，得到了密文解密的明文。

#### 3.5公钥加密

正式的公钥加密需要对输入的字符串进行加密。这与公钥加密演示版本只有一点区别，就是在一开始多一步对数据的处理，将输入的字符串逐个转换成对应的整型，然后分别加密。

**public** **static** String encrypt(String m,**int** n,**int** e){ //公钥加密，将输入的字符串分解成字符再进行加密

String result="";

**char**[] temp=m.toCharArray();

**int** sum;

**int** res;

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

sum=(**int**)temp[i];

res=*encryption*(sum,n,e);

result+=String.*valueOf*(res)+" ";

}

**return** result;

}

这样公钥加密就完成了。

#### 3.6公钥解密

公钥解密和演示版本同样区别在于一开始的数据分析。需要将密文按照空格分开，也就是逐字符读取，依次将读到的字符合成，直到读到一个空格，就把得到的字符串解密。

**public** **static** String decrypt(String c,**int** n,**int** d){ //公钥加密，将输入的字符串分解成字符再进行加密

String result="";

**char**[] temp=c.toCharArray();

String s="";

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

**if**(temp[i]!=' '){ //如果不是空格，九八字符串起来

s+=temp[i];

}

**else**{ //如果是空格，就把串成的字符串作为密文

**int** sum=Integer.*parseInt*(s);

result+=(**char**)*decryption*(sum,n,d);

//result+=String.valueOf(decryption(sum,n,d))+" ";

s="";

}

}

**return** result;

}

最后的输出也应该转换成字符的形式，对应明文的输入。

### 4.运行截图

首先进入主界面，点击公钥加密（演示）。



输入素数p 13 ,素数 q 17。选择e 143。点击生成公钥。



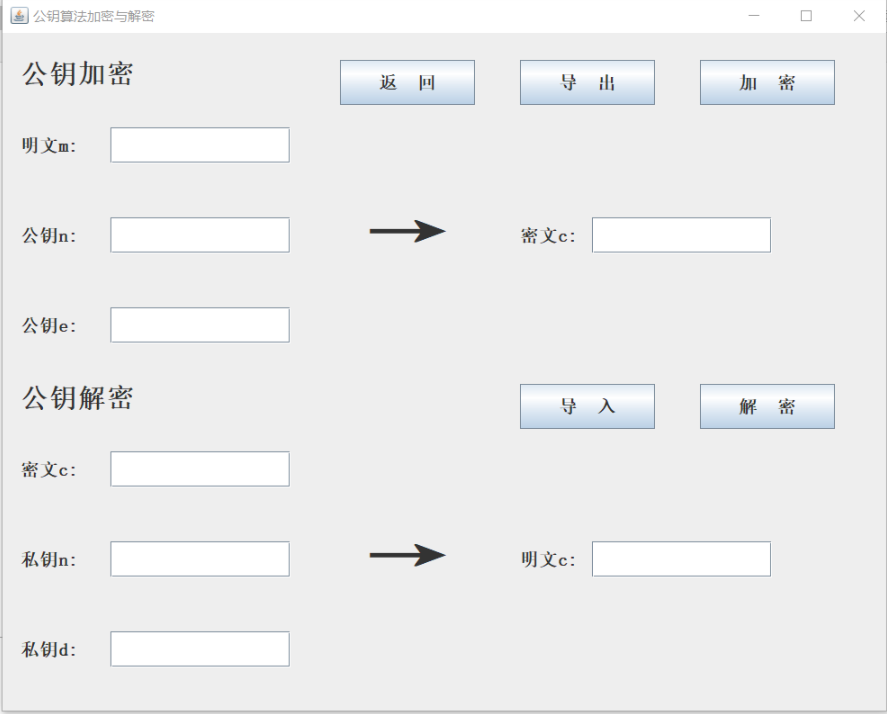
将得到的公钥，填入公钥加密的公钥n，公钥e中。填写明文40，点击加密。



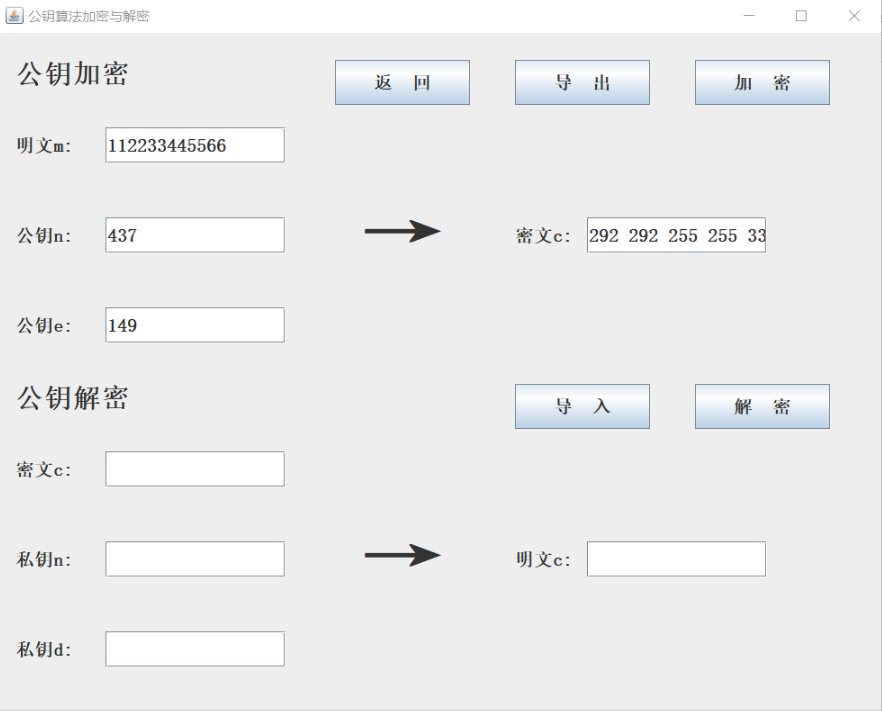
输入密文105，填入前面的得到的私钥，点击解密。



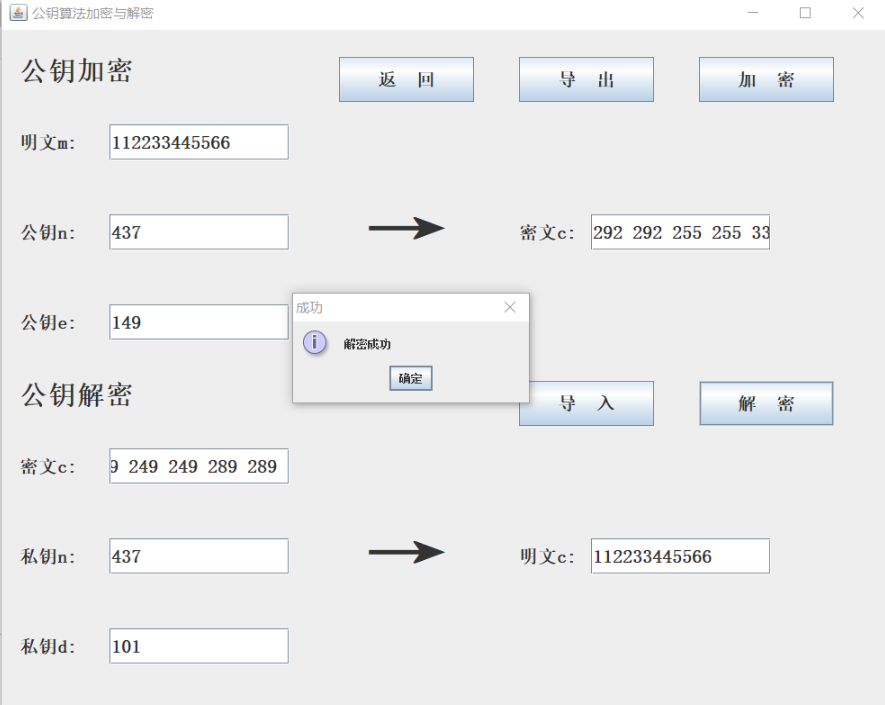
点击返回，回到主界面，点击公钥加密与解密。



输入明文“112233445566”，公钥n 437,公钥e 149。点击加密。



点击导出，将密文导出到publiccipher.txt。再点击导入，从对应文件中读取密文到密文c，输入私钥d=101,n=437，点击解密。



检查，明文与预期一致。

## 实验2 数据包捕获和分析

### 实验目的

熟悉网络数据包捕获与分析工具SNIFFER（或协议分析工具wireshark）的操作和使用方法，掌握数据包捕获和分析的基本过程；掌握协议过滤器的设定方法，能够捕获并分析常见的网络层和运输层数据包。

### 实验内容

使用协议分析工具wireshark，实验内容如下：

1.用Wireshark软件抓取本地PC的数据包，并观察其主要使用了哪些网络协议。

2.查找资料，了解相关网络协议的提出背景，帧格式，主要功能等。

3.根据所获数据包的内容分析相关协议，从而加深对常用网络协议理解。

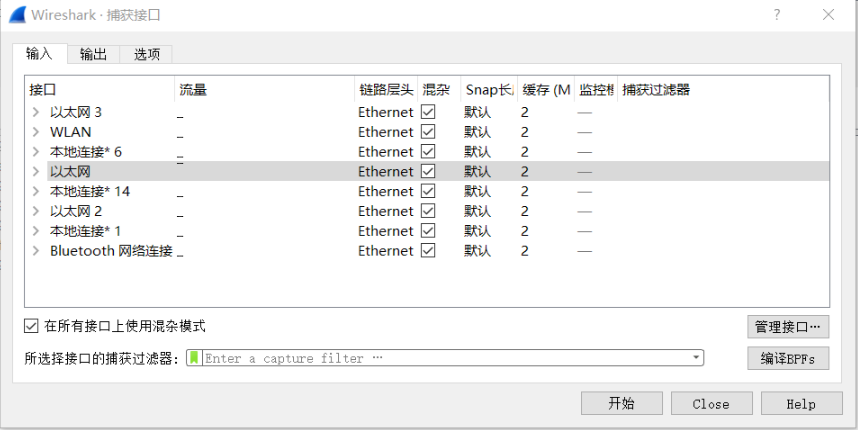
4.尝试对PC版QQ、微信、邮箱或网上银行登陆过程进行协议分析，看能否获得登陆的帐户名、口令以及传输的数据信息，分析它们是否以及采用了哪些安全算法或机制。

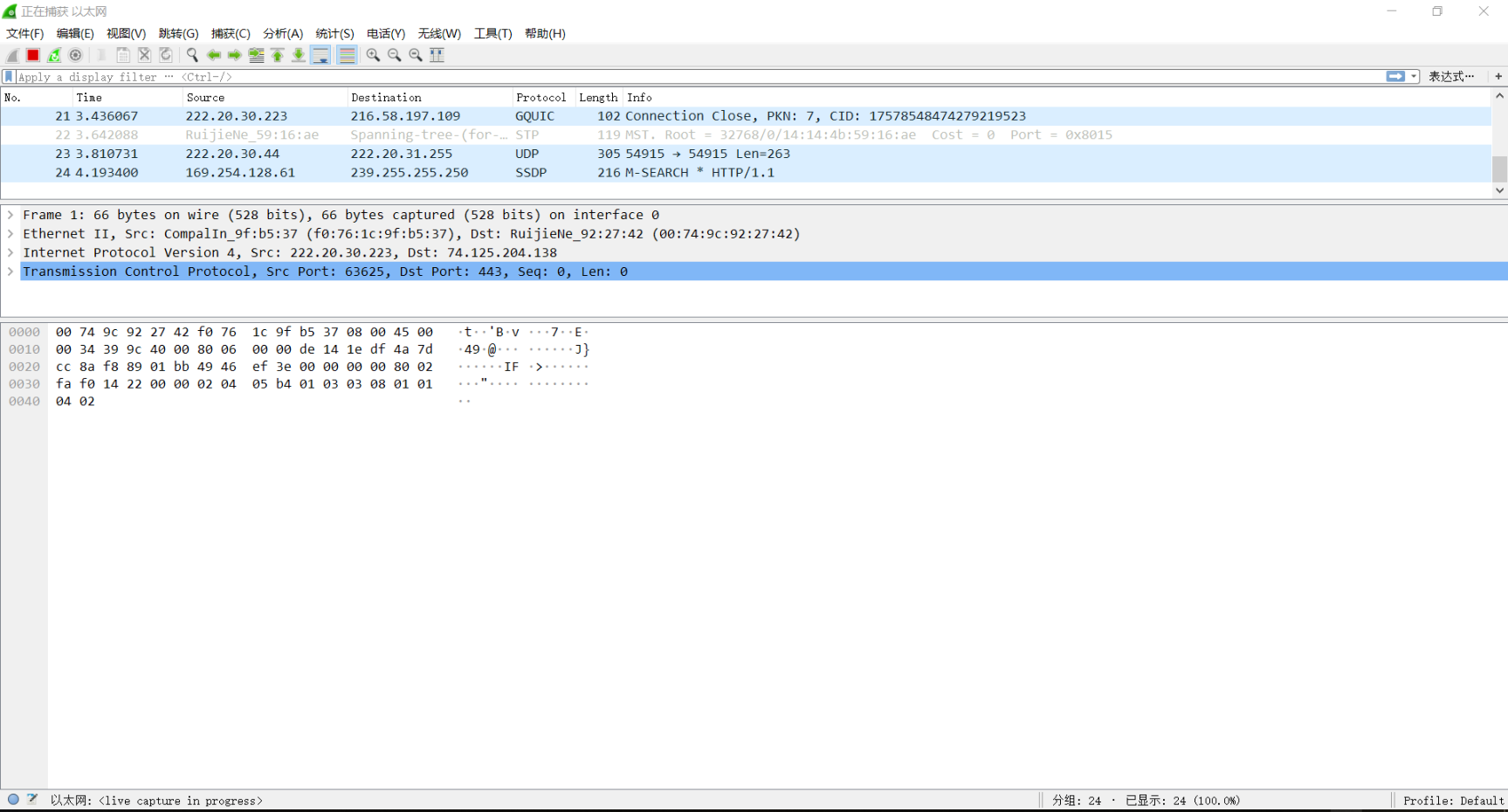
### 3.实验步骤

所有实验内容使用wireshark进行实验。

#### 3.1 抓取数据包及分析协议

首先打开安装好的wireshark，点击捕获，开始。然后回弹出一个捕获接口，选择需要抓包的接口，开始抓包。





可以看到上面的数据代表了抓取的数据包，上面有时间，源地址，目的地址，使用的协议，长度和一些信息。

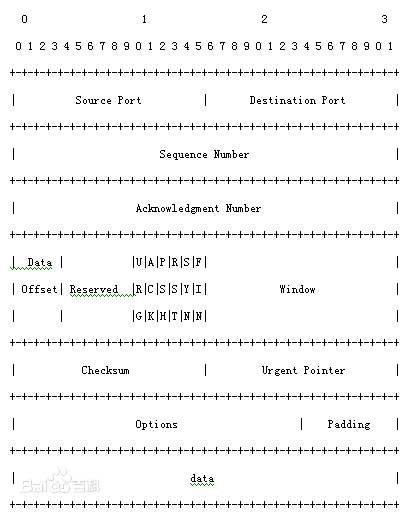
通过分析协议这一栏，可以看到大致使用了TCP,GQUIC,ICMPv6,UDP,SSDP,DNS,ARP这些协议。

#### 3.2 相关协议分析

相关协议分析如下：

###### 3.2.1 TCP协议

TCP（Transmission Control Protocol [传输控制协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE/9727741)）是一种面向连接的、可靠的、基于字节流的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82/4329536)通信协议。它是由IETF的RFC 793定义。它的帧格式如下图。



TCP 协议的功能主要是当应用层向TCP层发送用于网间传输的、用8位字节表示的[数据流](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%B5%81)，TCP则把数据流分割成适当长度的报文段，最大传输段大小（[MSS](https://baike.baidu.com/item/MSS/3567770" \t "_blank)）通常受该计算机连接的网络的数据链路层的最大传送单元（[MTU](https://baike.baidu.com/item/MTU" \t "_blank)）限制。之后TCP把数据包传给IP层，由它来通过网络将包传送给接收端实体的TCP层。

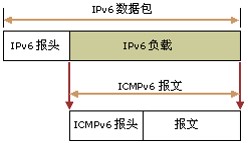
###### 3.2.2 QUIC协议

QUIC（Quick UDP Internet Connection）是谷歌制定的一种基于UDP的低时延的互联网传输层协议。在2016年11月国际互联网工程任务组(IETF)召开了第一次QUIC工作组会议，受到了业界的广泛关注。这也意味着QUIC开始了它的标准化过程，成为新一代传输层协议。

QUIC很好地解决了当今传输层和应用层面临的各种需求，包括处理更多的连接，安全性，和低延迟。QUIC融合了包括TCP，TLS，HTTP/2等协议的特性，但基于UDP传输。QUIC的一个主要目标就是减少连接延迟，当客户端第一次连接服务器时，QUIC只需要1RTT（Round-Trip Time）的延迟就可以建立可靠安全的连接,相对于TCP+TLS的1-3次RTT要更加快捷。之后客户端可以在本地缓存加密的认证信息，在再次与服务器建立连接时可以实现0-RTT的连接建立延迟。QUIC同时复用了HTTP/2协议的多路复用功能（Multiplexing），但由于QUIC基于UDP所以避免了HTTP/2的线头阻塞（Head-of-Line Blocking）问题。因为QUIC基于UDP，运行在用户域而不是系统内核，使得QUIC协议可以快速的更新和部署，从而很好地解决了TCP协议部署及更新的困难。

###### 3.2.3 ICMPv6协议

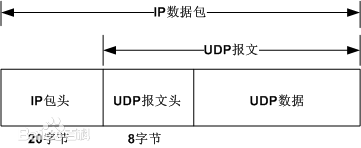
ICMPv6（Internet Control Managemet Protocol Version 6），即互联网控制信息协议版本六。ICMPv6为了与[IPv6](https://baike.baidu.com/item/IPv6)配套使用而开发的互联网控制信息协议。它的帧格式如下。



互联网控制信息协议是[IP协议](https://baike.baidu.com/item/IP%E5%8D%8F%E8%AE%AE)的一个重要组成部分。ICMPv6向源节点报告关于目的地址传输IPv6包的错误和信息，具有差错报告、网络诊断、邻节点发现和多播实现等功能。在[IPv6](https://baike.baidu.com/item/IPv6" \t "_blank)中，ICMPv6实现[IPv4](https://baike.baidu.com/item/IPv4" \t "_blank)中[ICMP](https://baike.baidu.com/item/ICMP" \t "_blank)、[ARP](https://baike.baidu.com/item/ARP" \t "_blank)和[IGMP](https://baike.baidu.com/item/IGMP" \t "_blank)的功能。

###### 3.2.4 UDP协议

UDP 是User Datagram Protocol的简称， 中文名是用户数据报协议，是[OSI](https://baike.baidu.com/item/OSI)（Open System Interconnection，[开放式系统互联](https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E5%BC%8F%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%81%94/562749" \t "_blank)） 参考模型中一种无连接的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)协议。它的帧格式如下。



UDP是[OSI](https://baike.baidu.com/item/OSI" \t "_blank)参考模型中一种无连接的传输层协议，它主要用于不要求分组顺序到达的传输中，分组传输顺序的检查与排序由应用层完成 [2]  ，提供面向事务的简单不可靠信息传送服务。UDP 协议基本上是[IP协议](https://baike.baidu.com/item/IP%E5%8D%8F%E8%AE%AE" \t "_blank)与上层协议的接口。UDP协议适用[端口](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3" \t "_blank)分别运行在同一台设备上的多个[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)。

###### 3.2.5 SSDP协议

SSDP，即简单服务发现协议（SSDP，Simple Service Discovery Protocol），是一种应用层协议，是构成通用即插即用(UPnP)技术的核心协议之一。

SSDP简单服务发现协议提供了在局部网络里面发现设备的机制。控制点（也就是接受服务的客户端）可以通过使用简单服务发现协议，根据自己的需要查询在自己所在的局部网络里面提供特定服务的设备。设备（也就是提供服务的[服务器端](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8%E7%AB%AF)）也可以通过使用简单服务发现协议，向自己所在的局部网络里面的控制点声明它的存在。

###### 3.2.6 DNS协议

DNS（Domain Name System，[域名系统](https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E7%B3%BB%E7%BB%9F/2251573" \t "_blank)），万维网上作为域名和[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80" \t "_blank)相互映射的一个[分布式数据库](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%95%B0%E6%8D%AE%E5%BA%93/1238109)，能够使用户更方便的访问[互联网](https://baike.baidu.com/item/%E4%BA%92%E8%81%94%E7%BD%91)，而不用去记住能够被机器直接读取的IP数串。

每个[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80)都可以有一个主机名，主机名由一个或多个字符串组成，字符串之间用小数点隔开。有了主机名，就不要死记硬背每台IP设备的IP地址，只要记住相对直观有意义的主机名就行了。这就是DNS协议的功能。

通过[主机名](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA%E5%90%8D)，最终得到该主机名对应的IP地址的过程叫做域名解析（或主机名解析）。在解析域名时，可以首先采用静态域名解析的方法，如果[静态](https://baike.baidu.com/item/%E9%9D%99%E6%80%81" \t "_blank)域名解析不成功，再采用动态域名解析的方法。可以将一些常用的域名放入静态域名解析表中，这样可以大大提高域名解析效率。

###### 3.2.7 ARP协议

地址解析协议，即ARP（Address Resolution Protocol），是根据[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80" \t "_blank)获取[物理地址](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%9C%B0%E5%9D%80/2129)的一个[TCP/IP协议](https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE)。

|  |  |  |
| --- | --- | --- |
| **ARP报文格式** | | |
| 硬件类型 | | 协议类型 |
| 硬件地址长度 | 协议长度 | 操作类型 |
| 发送方硬件地址（0-3字节） | | |
| 发送方硬件地址（4-5字节） | | 发送方IP地址（0-1字节） |
| 发送方IP地址（2-3字节） | | 目标硬件地址（0-1字节） |
| 目标硬件地址（2-5字节） | | |
| 目标IP地址（0-3字节） | | |

[OSI模型](https://baike.baidu.com/item/OSI%E6%A8%A1%E5%9E%8B)把网络工作分为七层，IP地址在OSI模型的第三层，MAC地址在第二层，彼此不直接打交道。在通过[以太网](https://baike.baidu.com/item/%E4%BB%A5%E5%A4%AA%E7%BD%91" \t "_blank)发送IP数据包时，需要先封装第三层（32位IP地址）、第二层（48位MAC地址）的报头，但由于发送时只知道目标IP地址，不知道其MAC地址，又不能跨第二、三层，所以需要使用地址解析协议。使用地址解析协议，可根据网络层IP数据包包头中的IP地址信息解析出目标硬件地址（MAC地址）信息，以保证通信的顺利进行。

### 3.3 对数据包进行分析

抓包结果整个窗口被分成三部分：

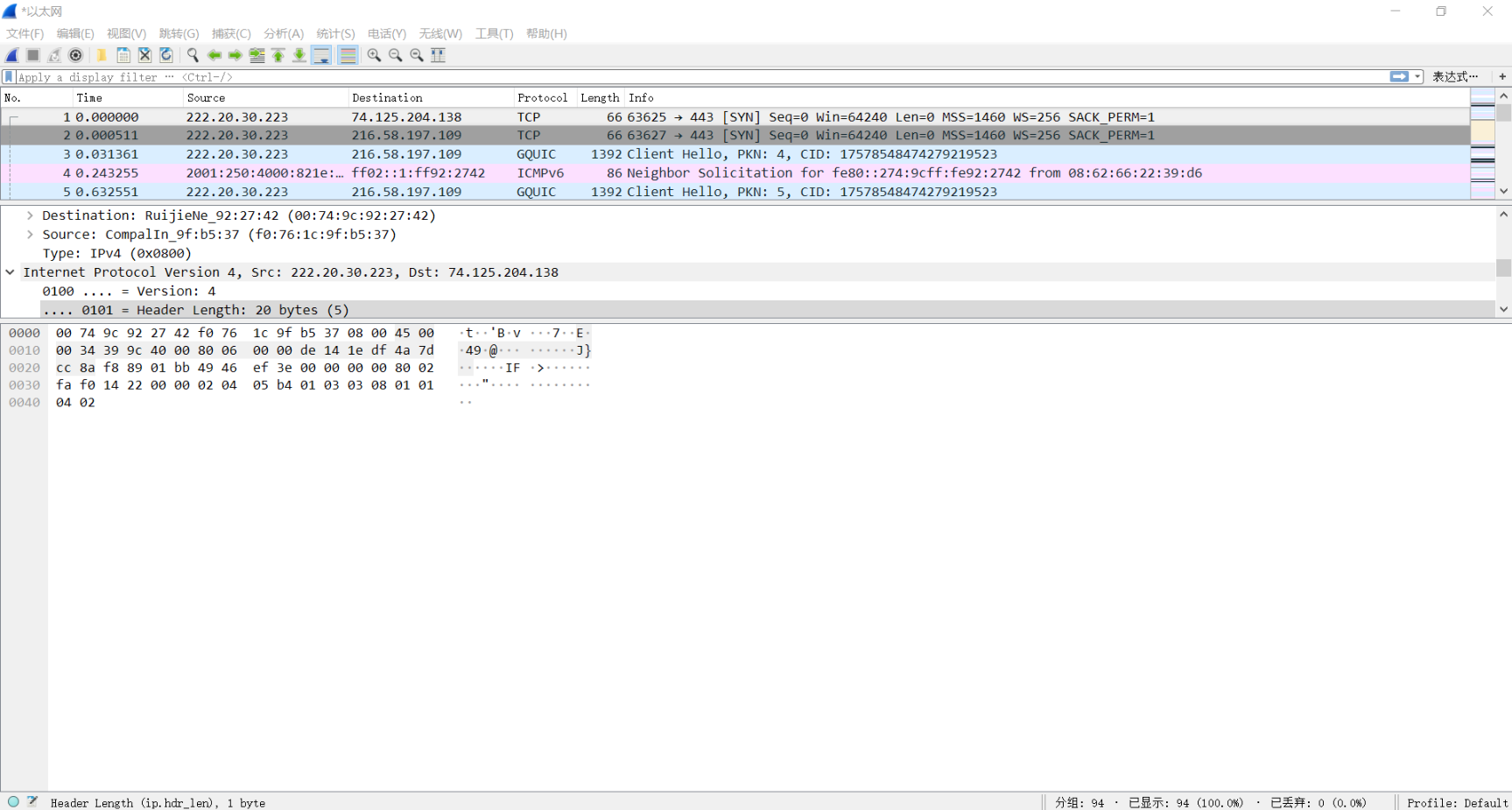
1、最上面为数据包列表，用来显示截获的每个数据包的总结性信息；

2、中间为协议树，用来显示选定的数据包所属的协议信息；

3、最下边是以十六进制形式表示的数据包内容，用来显示数据包在物理层上传输时的最终形式。

数据包列表中，第一列是编号（如第1个包），第二列是截取时间（0.000000），第三列source是源地址（115.155.39.93），第四列destination是目的地址（115.155.39.112），第五列protocol是这个包使用的协议（这里是UDP协议），第六列info是一些其它的信息，包括源端口号和目的端口号（源端口：58459，目的端口：54062）。

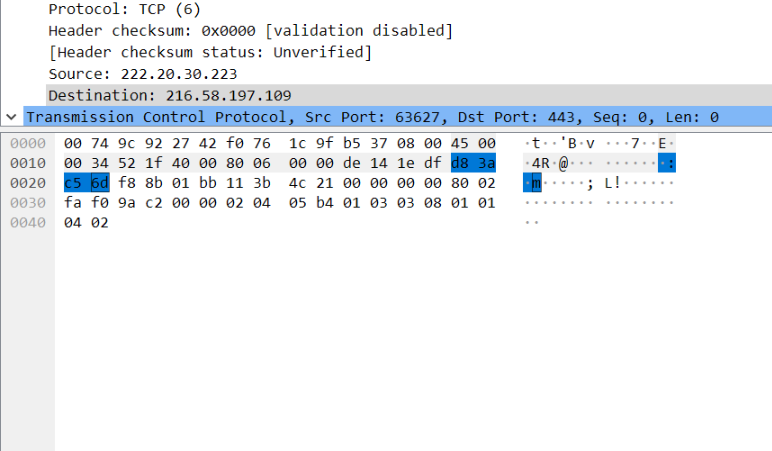
协议树可以得到被截获数据包的更多信息，如主机的MAC地址（Ethernet II）、IP地址（Internet protocol）、UDP端口号（user datagram protocol）以及UDP协议的具体内容（data）。



例如上面展示的抓到的第二个数据包，首先通过最上面的数据包列表可以了解到抓取的时间是0.000511，源地址是222.20.30.223，目的地址是216.58.197.109，协议是TCP，长度是66，另外还有它的一些信息。

再看中间部分，中间显示了协议的信息，上面可以获取一些更多的信息，比如说主机的mac地址，IP地址等内容。

再看底下的部分，可以看到数据包的内容，点击任意的内容可以看到它对应的协议树中的部分。



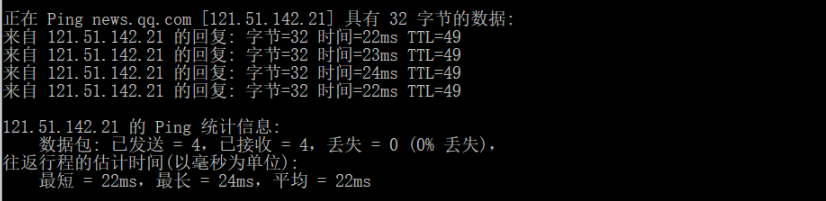
比如这边的d8 3a c5 6d 对应的就是协议书中的destination也就是目的地址。

### 3.4 对QQ官网登录界面的数据包分析

###### 3.4.1 获取IP地址

首先第一步需要获取[www.qq.com的IP](http://www.qq.com的IP)地址，直接在命令提示符输入ping [www.qq.com](http://www.qq.com)

得到一串信息。

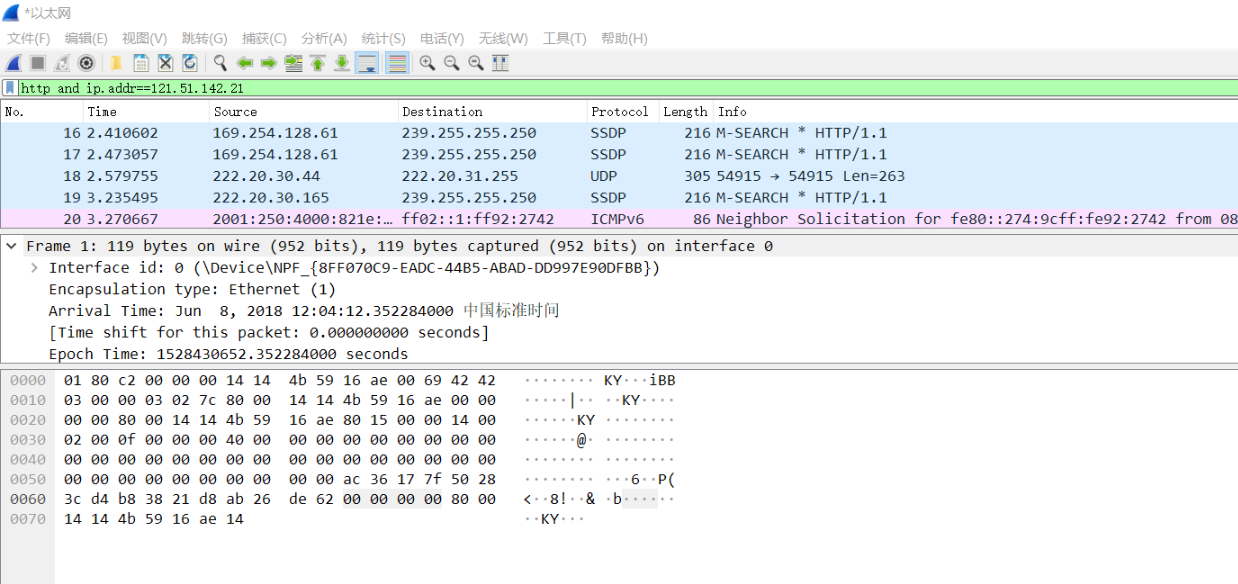


其中的121.51.142.21就是IP地址了。这边的IP地址主要用于后面获取需要分析的数据包，不然茫茫多的数据包中间很难找到需要分析的QQ登录的数据包。

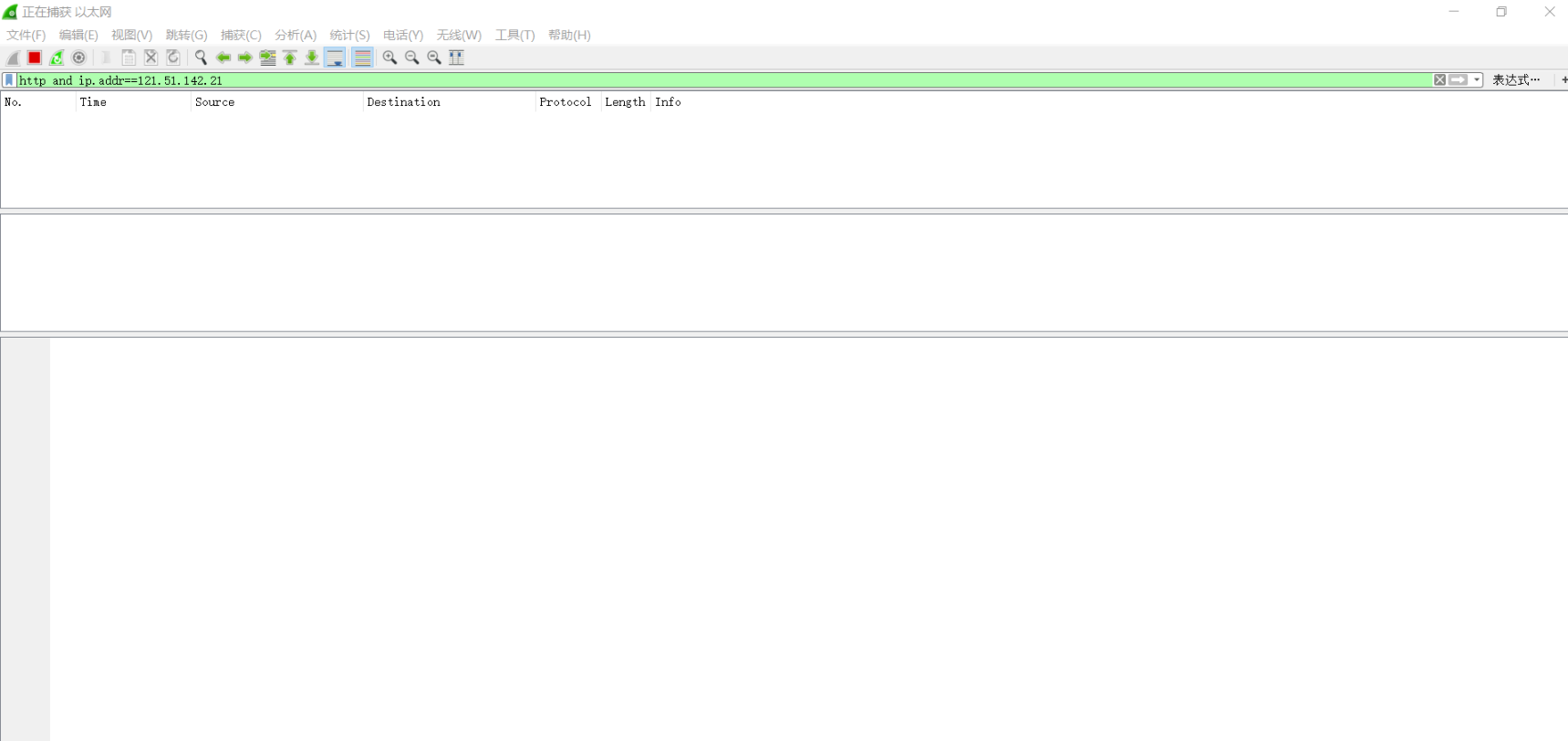
###### 3.4.2 设置过滤器

这一步需要先开始抓包，然后暂停，将IP地址填入到过滤器中，然后继续抓包。

填写格式为http and ip.addr==需要的IP地址。

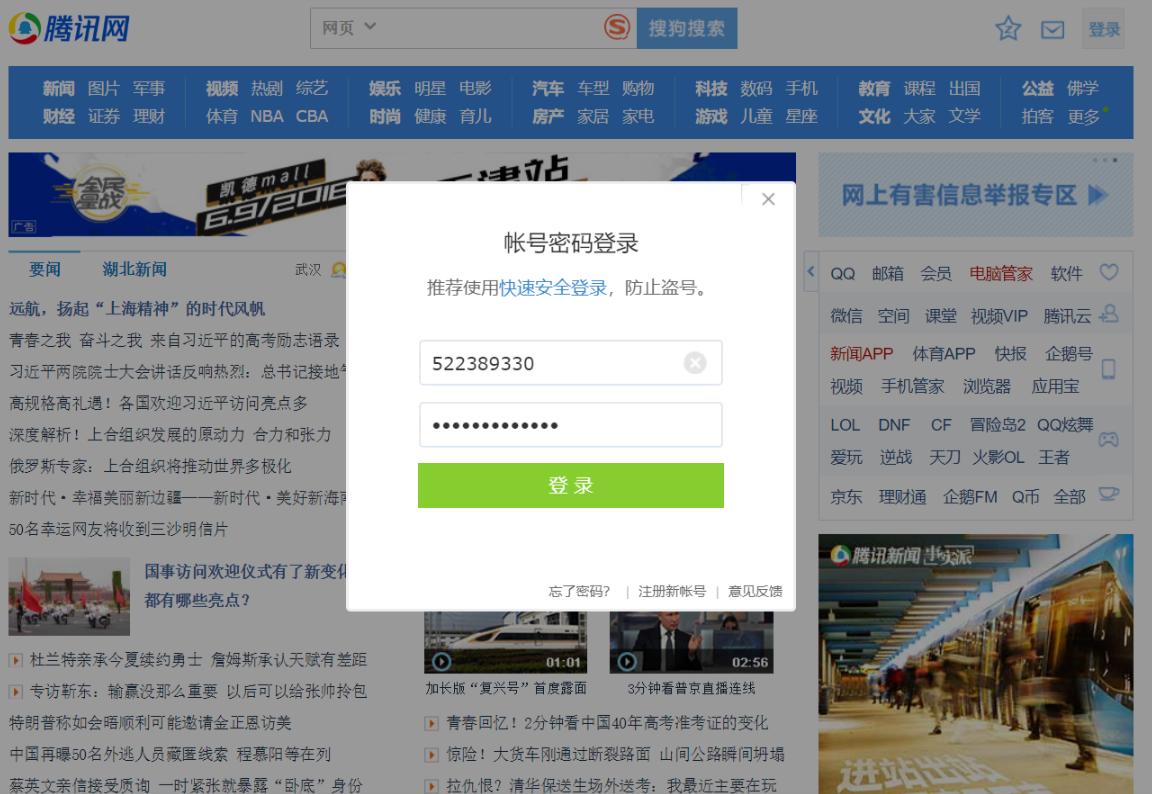


然后可以看到现在没有任何关于IP为121.51.142.21的数据包。

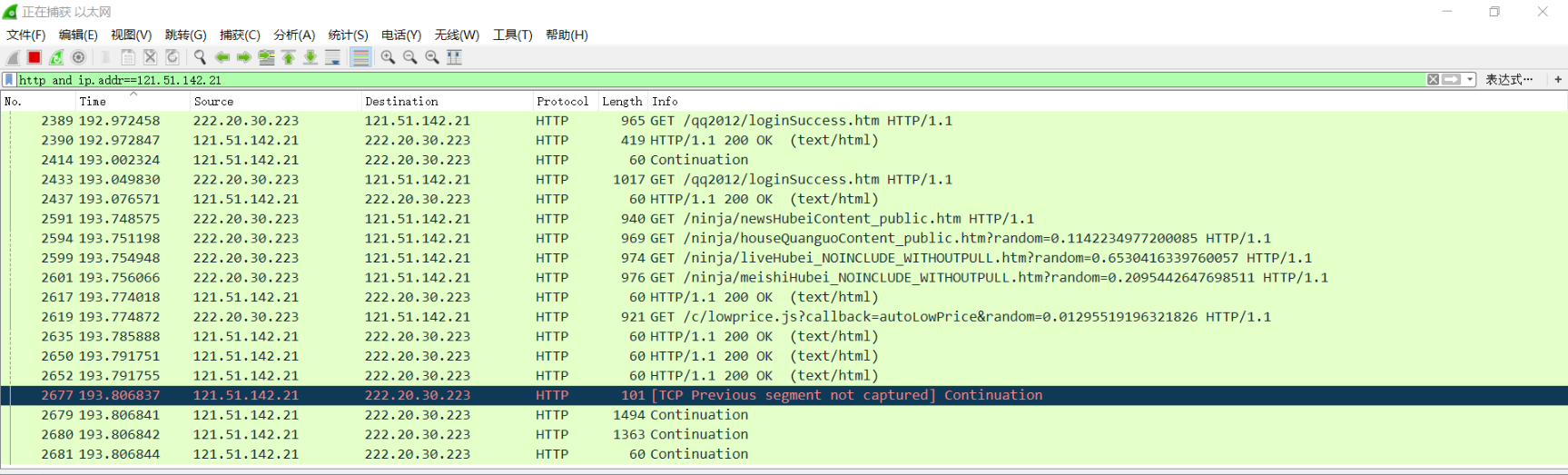


###### 3.4.3 输入登陆信息

在QQ登陆界面输入登录信息。



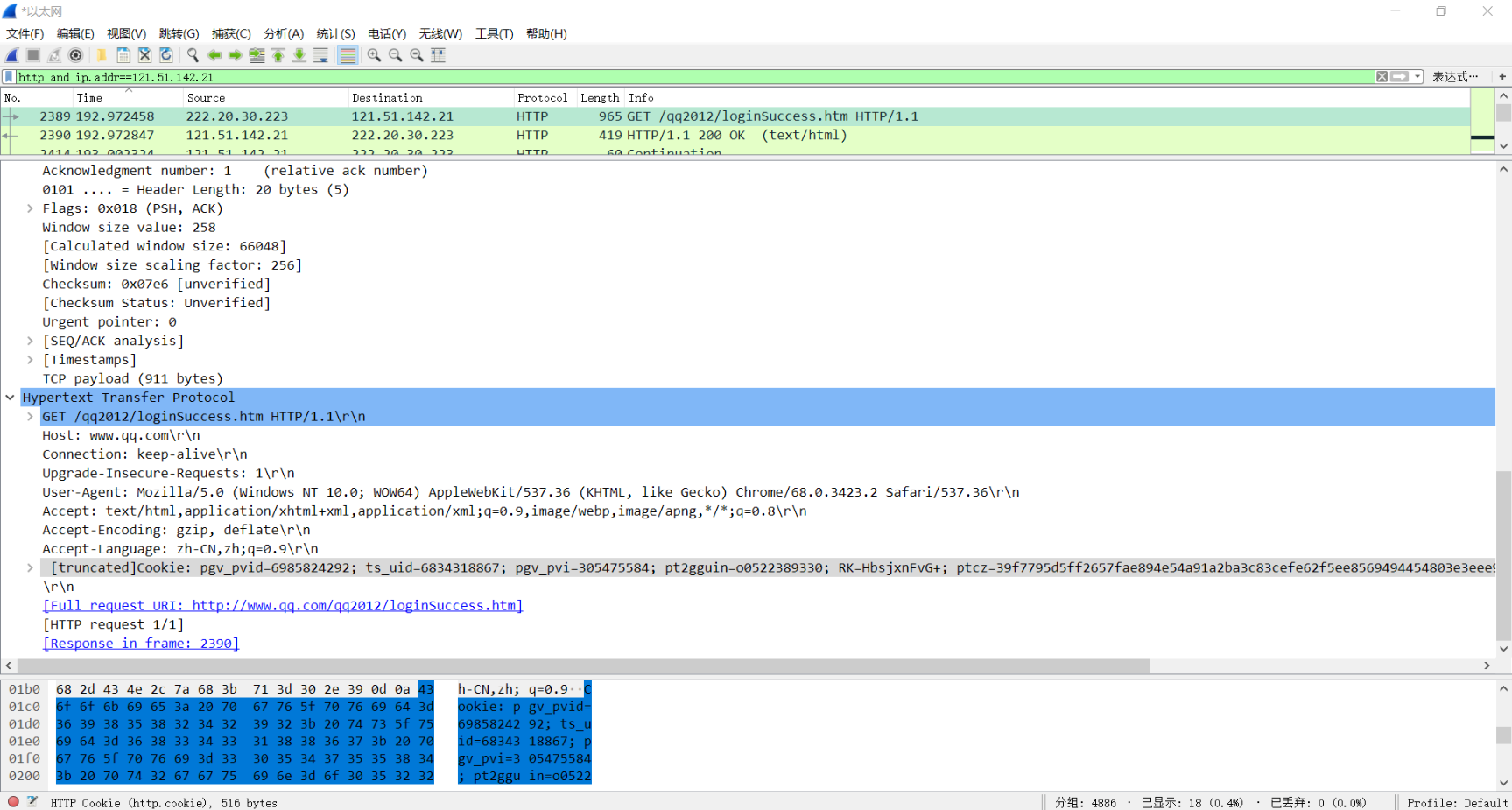
点击登录后，立即查看wireshark。



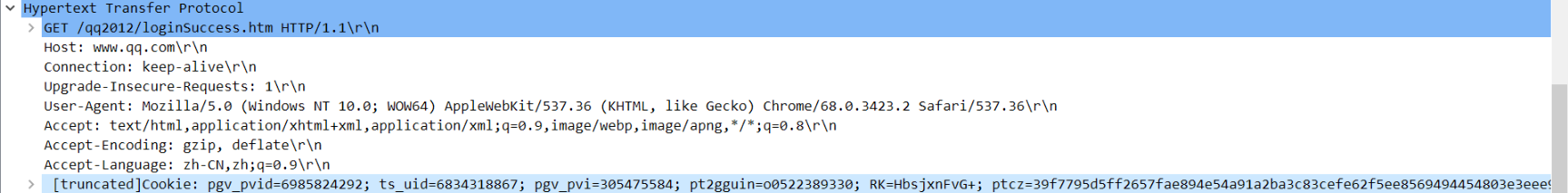
发现多了这么多的数据包，然后找到用户名密码所在的那个数据包。

通过逐个打开数据包分析协议树信息，大致确认在第一个数据包内。

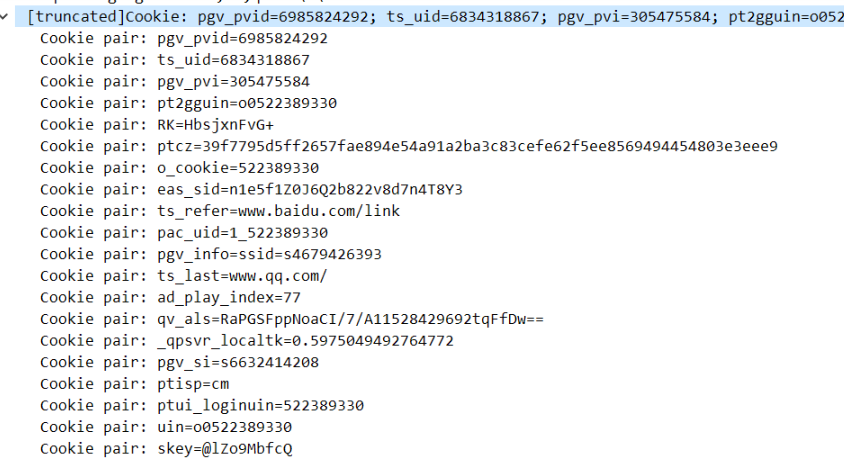
打开第一个数据包的协议树，分析协议。



由于我自己知道自己的用户名，所以很快就找到了用户名密码所在的位置。



再继续展开查看



这边就可以比较清楚地看到，其中用户名没有做加密处理，就是522389330。而密码部分做了加密处理，所以我也不太清楚大致哪一行是密码部分。

现在的QQ加密技术已经十分完善，很难得到加密前的密码信息。