

SHANGHAI JIAO TONG UNIVERSITY



【计算机通信网课程大作业】

网络发包器

学 生 1: 吴怡琳

学 生 2: 徐民凯

专 业: 信息安全

指导教师: 姚立红

1. **项目概述**

**1.1 功能要求**

根据大作业要求ppt，本次所需要完成的网络发包器需要满足以下要求：

1、**基本功能**：

构建如下的网络报文，将其发送到网络上，观察实际的网络通信效果。

**（1）IP报文**

包括：版本号、头部长度、上层协议、校验和、生存时间、IP源地址、IP目的地址等。

**（2）TCP报文**

包括：源端口、目的端口、序列号、偏移量、窗口、校验和等。

**（3）UDP报文**

包括：源端口、目的端口、长度、校验和等。

2、**构建ARP报文**

包括：硬件类型、协议类型、硬件地址长度、协议地址长度、操作类型、源物理地址、源IP地址、目的物理地址、目的IP地址等。

3、**构建ICMP报文**

包括：类型、代码、校验和等。

4、**支持PAP文件**

**1.2 项目要求**

具体项目要求如下：

1. 可以两人合作完成；

2. 可使用C/C++/C#/Java/Python等语言，Windows/Linux平台均可，可借助WinPcap、SharpPcap、LibPcap、 Libnet等类库；

3. 通过网络发包器，构建完整的TCP、UDP报文，进行实际的网络访问，应返回正确的结果；

4. 需要实现友好的用户界面。

**1.3 项目成果**

根据要求，大作业仅实现功能1最高分A-，实现全部功能最高分A+。**我们的发包器：**

1. **实现了作业中要求的全部功能（IP, UDP, TCP, ARP, ICMP的发送，PCAP文件的读写支持以及bpf（包过滤））；**
2. **考虑到网络空间安全专业内容，我们增加了密码学工具箱（RSA, AES, HASH, Digital Signature），方便对报文数据的各种密码学操作，为数据在网络上的传输提供一定的安全性保障；**
3. **我们为用户提供了详细的说明文档，以方便用户的使用。**

**我们的项目完整地实现了上述功能。具体情况将在下文中阐述，这里不再赘述。**

**1.4 项目环境**

本次项目的开发情况如下所示：

|  |  |
| --- | --- |
| 开发人员 | 吴怡琳（516021910334），徐民凯（516021910714） |
| 开发平台 | Win10-x64 |
| 开发语言 | Python 2.7 |
| 开发IDE | PyCharm |
| UI设计工具 | PyQt5 |
| 抓包工具 | WireShark |
| 使用库 | PyQt5, PyCrypto, Socket, Scapy, Struct等 |

**1.5 项目分工**

项目具体分工如下：

**后端搭建搭建：由两人共同完成；**

**GUI：吴怡琳负责各个密码学工具、菜单栏、界面转换以及部分帮助窗口；徐民凯负责各个网络报文的发送、PCAP文件的读写、部分帮助窗口。相应部分的前后端连接与报告撰写分别由两人完成。**

**两人各自的具体工作，将在后面后端与前端的技术细节分两个部分进行详述。**



Fig1.3.1项目文件列表

**二、成果展示**

**2.1 网络包构造与发送**

**2.1.1 IP发包**

在上方文本框内输入对应信息，点击“生成数据包”按钮，即会在下方文本框内输出要发送的包格式，点击“发送数据包”按钮完成发包，下方文本框中出现Send a packet！字样，发包完成。

输入出现问题时，下方文本框中显示Input error！

发包只支持IPv4协议，不支持IPv6。

具体选择项信息：（\*代表必选项）

**\*源IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*目的IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*数据**：IP包负载（payload），输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串，如0123456789abcdef。

**MF**：IP标志位最低位，表示是否还有更多分片。选中为1，不选为0。

**DF**：IP标志位中间位，表示是否允许分片。选中为1，不选为0。

**版本号**：IP版本号，4bit，数字int格式，默认为4。

**服务号**：IP服务号，8bit，数字int格式，默认为0。

**标识**：IP标识位，当IP需要分片时依此确定同属一个IP报文，16bit，数字int格式，默认为0x0001。

**片偏移**：片偏移量，标示分片的IP包在原IP报文中的位置，13bit，数字int格式，默认为0。

**生存时间**：8bit，数字int格式，默认为64。

**上层协议**：标示该IP包携带何种协议数据，8bit，数字int格式，默认为6（TCP协议）。

**校验和**：IP包校验和，16bit，十六进制数字形式，在点击“生成数据包”后会填入正确校验和，不支持个人输入修改。

**可选头部**：输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串。

下面给出一个示例，以下是一个IP包的构造过程：

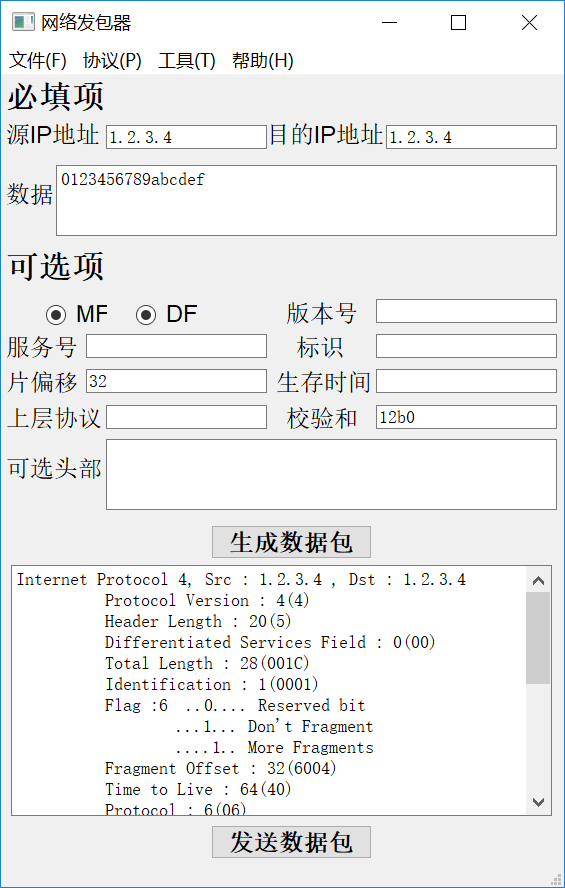


Fig 2.1.1.1 IP包的构造

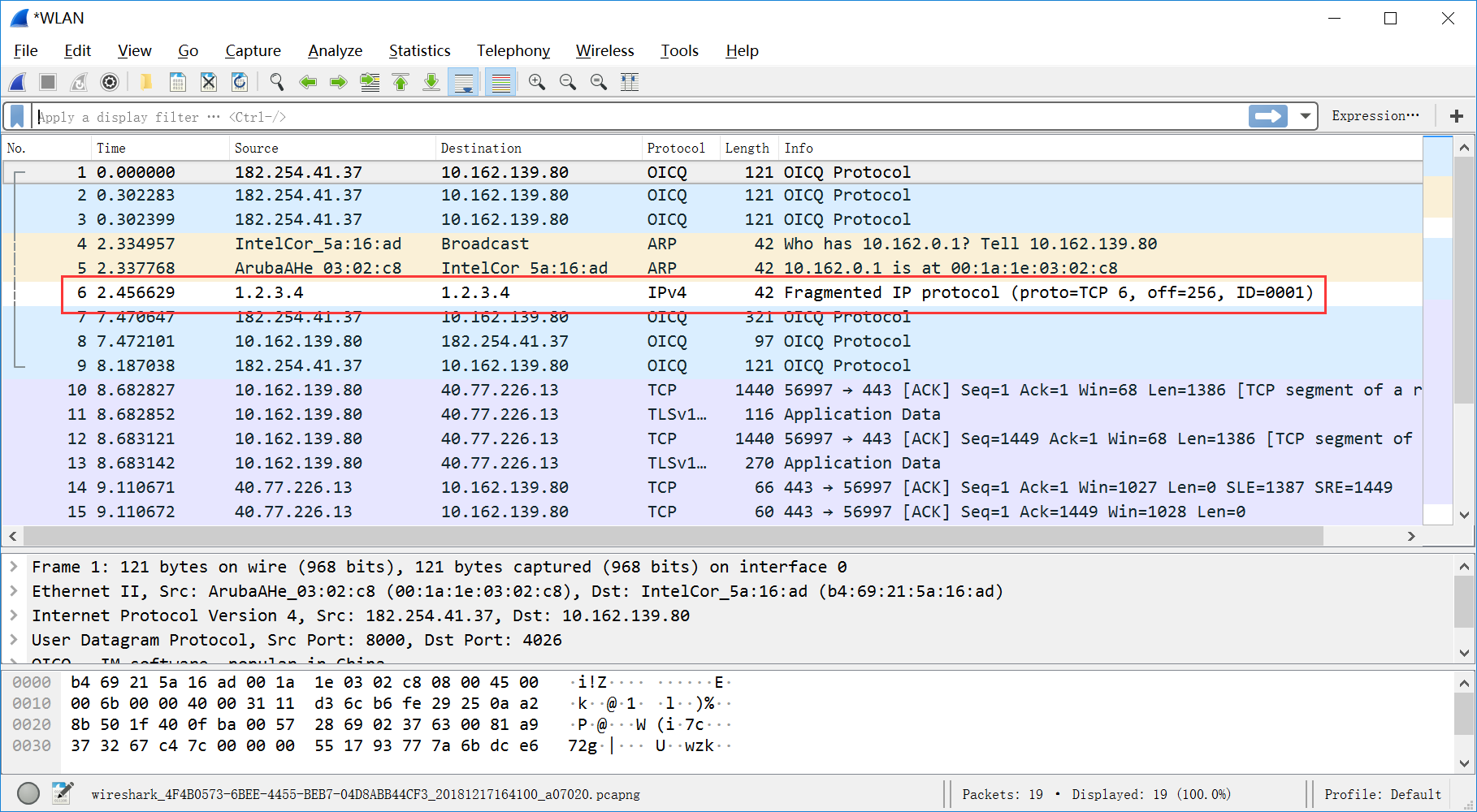


Fig 2.1.1.2 IP包实际发送结果

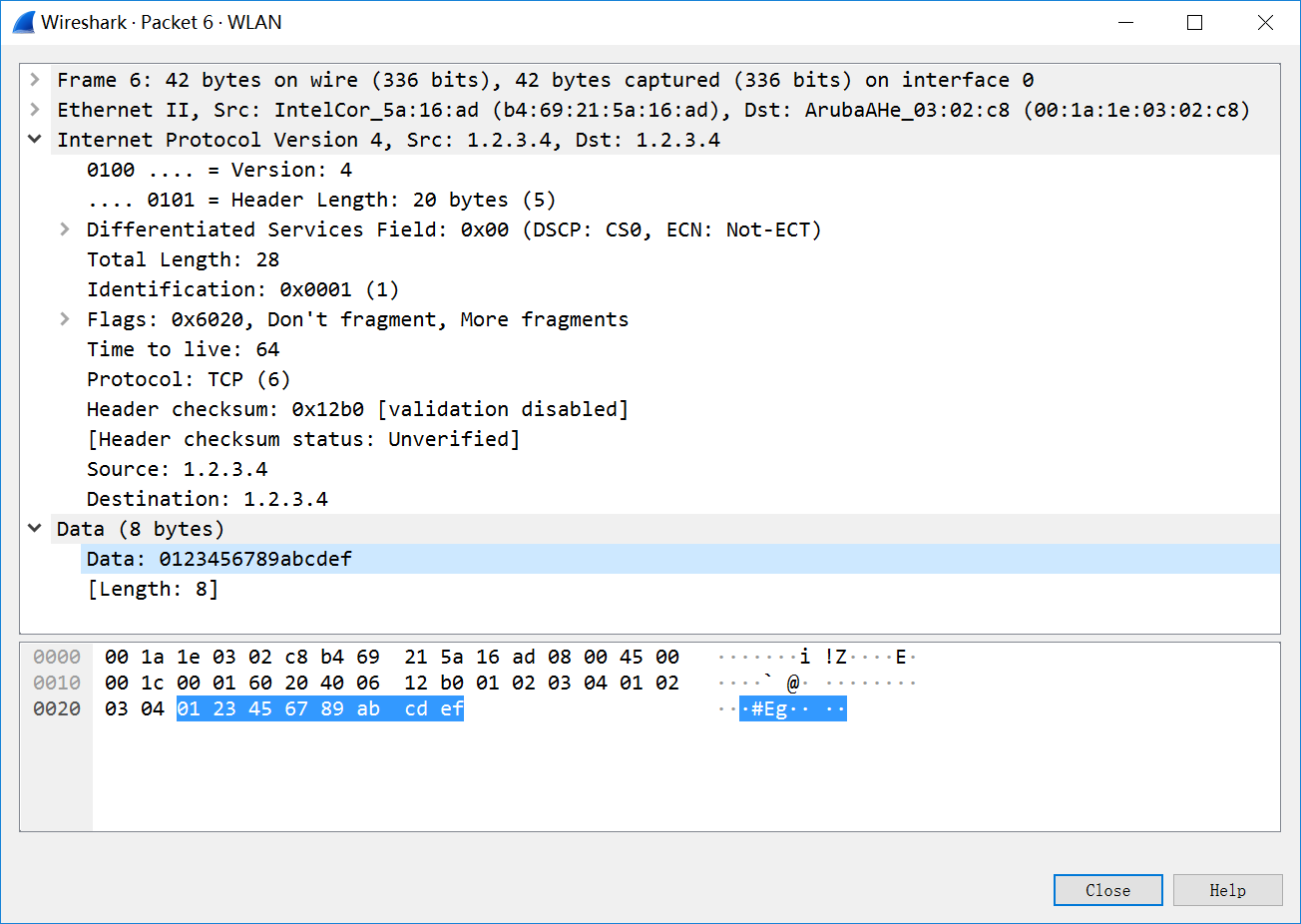


Fig 2.1.1.3 IP包实际抓包结果

**2.1.2 TCP发包**

在上方文本框内输入对应信息，点击“生成数据包”按钮，即会在下方文本框内输出要发送的包格式，点击“发送数据包”按钮完成发包，下方文本框中出现Send a packet！字样，发包完成。

输入出现问题时，下方文本框中显示Input error！

具体选择项信息：（\*代表必选项）

**\*源IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*目的IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*源端口**：16bit，数字int格式，如8080。

**\*目的端口**：16bit，数字int格式，如8080。

**\*数据**：TCP包负载（payload），输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串，如0123456789abcdef。

**CWR**：标志位标志位，表示拥塞窗口减少。选中为1，不选为0。

**ECE**：标志位标志位，表示显式拥塞提醒回应。选中为1，不选为0。

**URG**：标志位紧急标志位，表示高优先级数据包。选中为1，不选为0。

**ACK**：标志位应答标志位。选中为1，不选为0。

**PSH**：标志位推送标志位，表示立即转发。选中为1，不选为0。

**RST**：标志位复位标志位，表示中断连接。选中为1，不选为0。

**SYN**：标志位开始标志位，表示开始会话请求。选中为1，不选为0。

**FIN**：标志位结束标志位，表示结束会话。选中为1，不选为0。

**序列号**：32bit，数字int格式，默认为0。

**确认号**：32bit，数字int格式，默认为0。

**报头长度**：4bit，数字int格式，默认为5。

**窗口大小**：发送窗口大小，16bit，数字int格式，默认为65523。

**校验和**：16bit，十六进制数字形式，在点击“生成数据包”后会填入正确校验和，不支持个人输入修改。

**紧急指针**：URG=1时有效，与序列号相加标示紧急数据最后一个字节的序号，16bit，数字int格式，默认为0。

**可选头部**：输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串。

下面给出一个示例，以下是一个TCP包的构造过程：

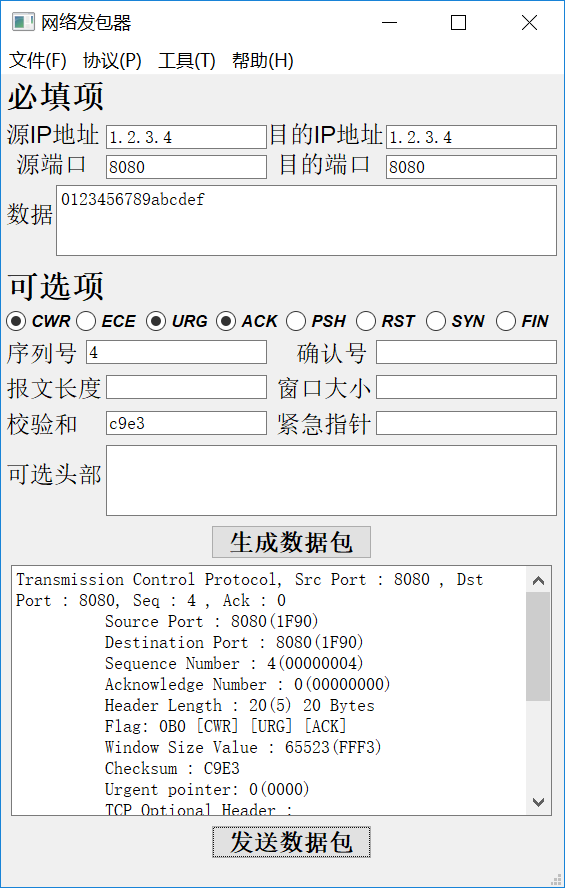


Fig 2.1.2.1 TCP包的构造

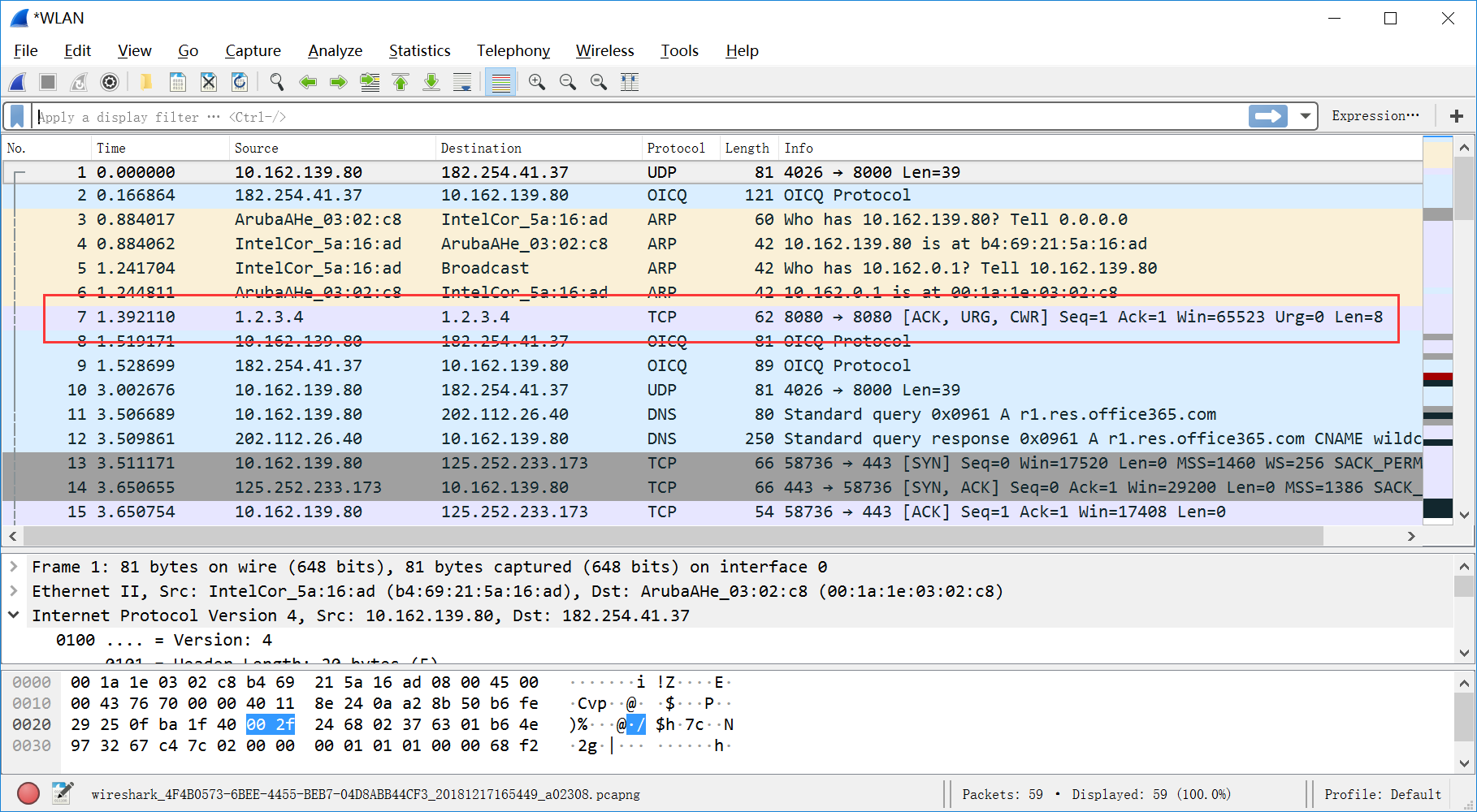


Fig 2.1.2.2 TCP包实际发送结果

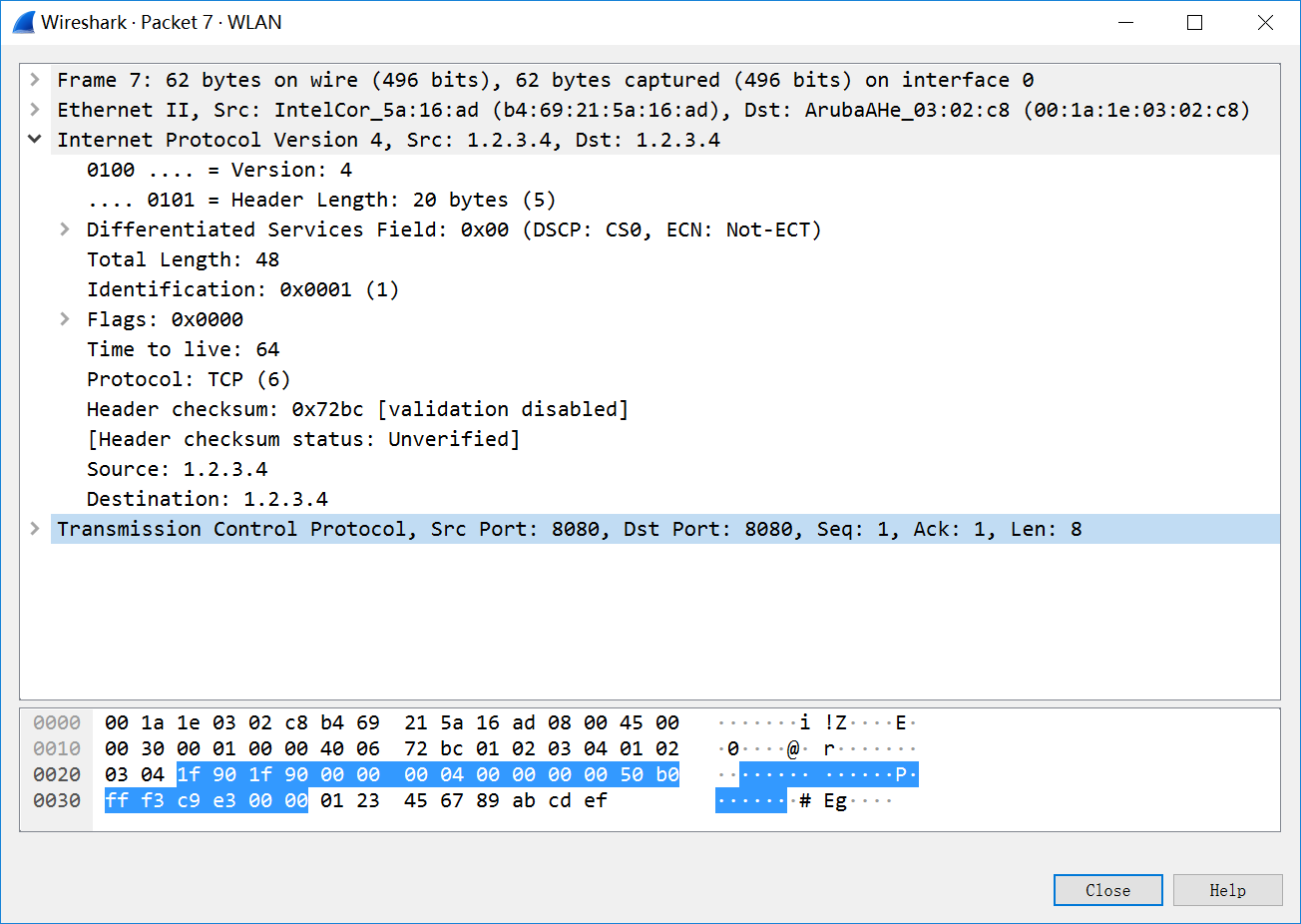


Fig 2.1.2.3 TCP包实际抓包结果

**2.1.3 UDP发包**

在上方文本框内输入对应信息，点击“生成数据包”按钮，即会在下方文本框内输出要发送的包格式，点击“发送数据包”按钮完成发包，下方文本框中出现Send a packet！字样，发包完成。

输入出现问题时，下方文本框中显示Input error！

具体选择项信息：（\*代表必选项）

**\*源IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*目的IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*源端口**：16bit，数字int格式，如8080。

**\*目的端口**：16bit，数字int格式，如8080。

**\*数据**：UDP包负载（payload），输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串，如0123456789abcdef。

**校验和**：16bit，十六进制数字形式，在点击“生成数据包”后会填入正确校验和，不支持个人输入修改。

下面给出一个示例，以下是一个UDP包的构造过程：

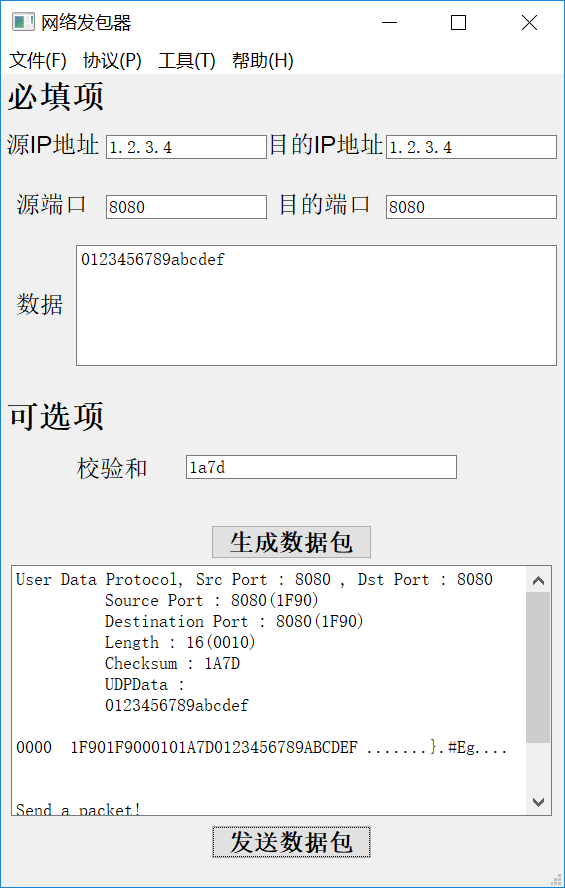


Fig 2.1.3.1 UDP包的构造

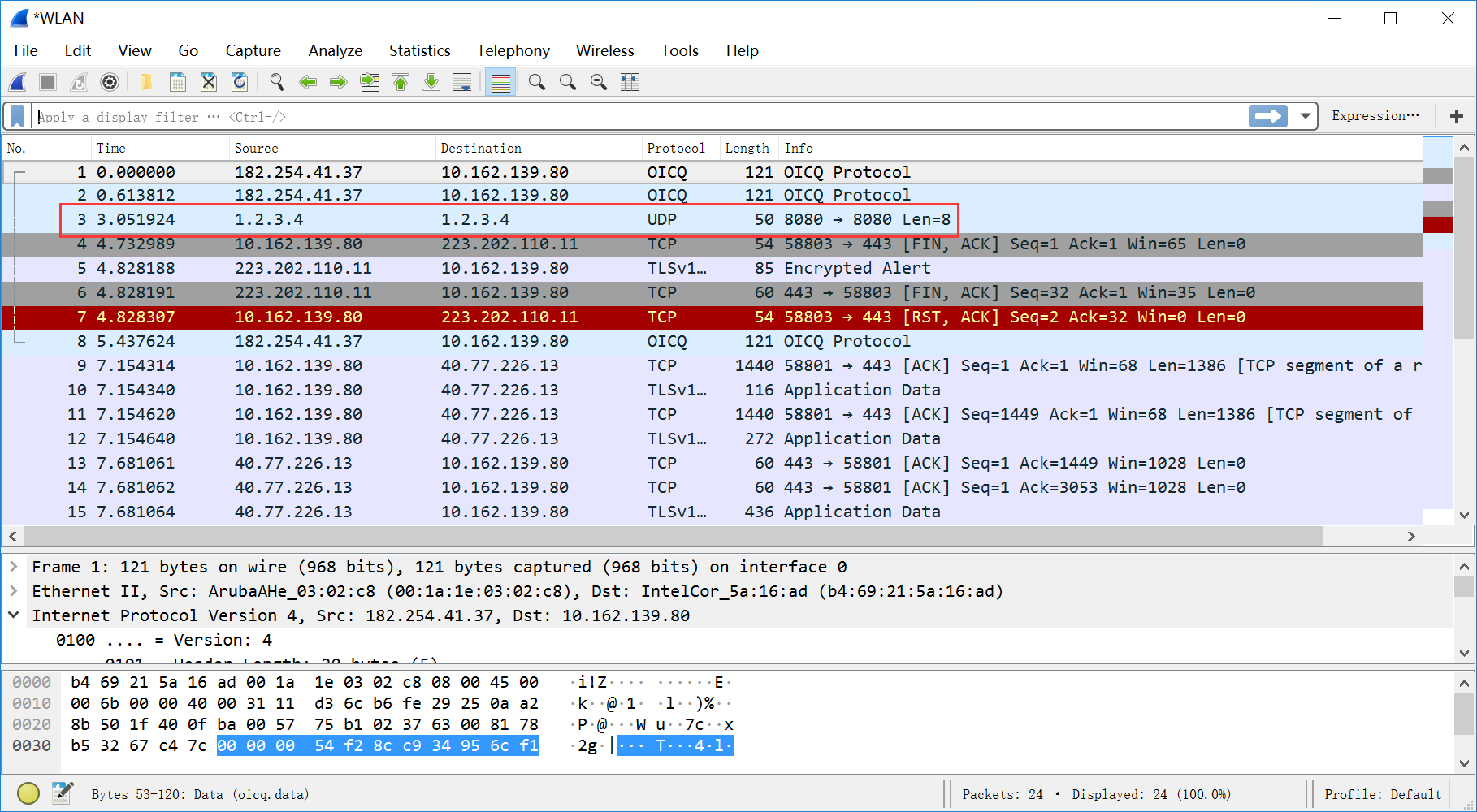


Fig 2.1.3.2 UDP包实际发送结果

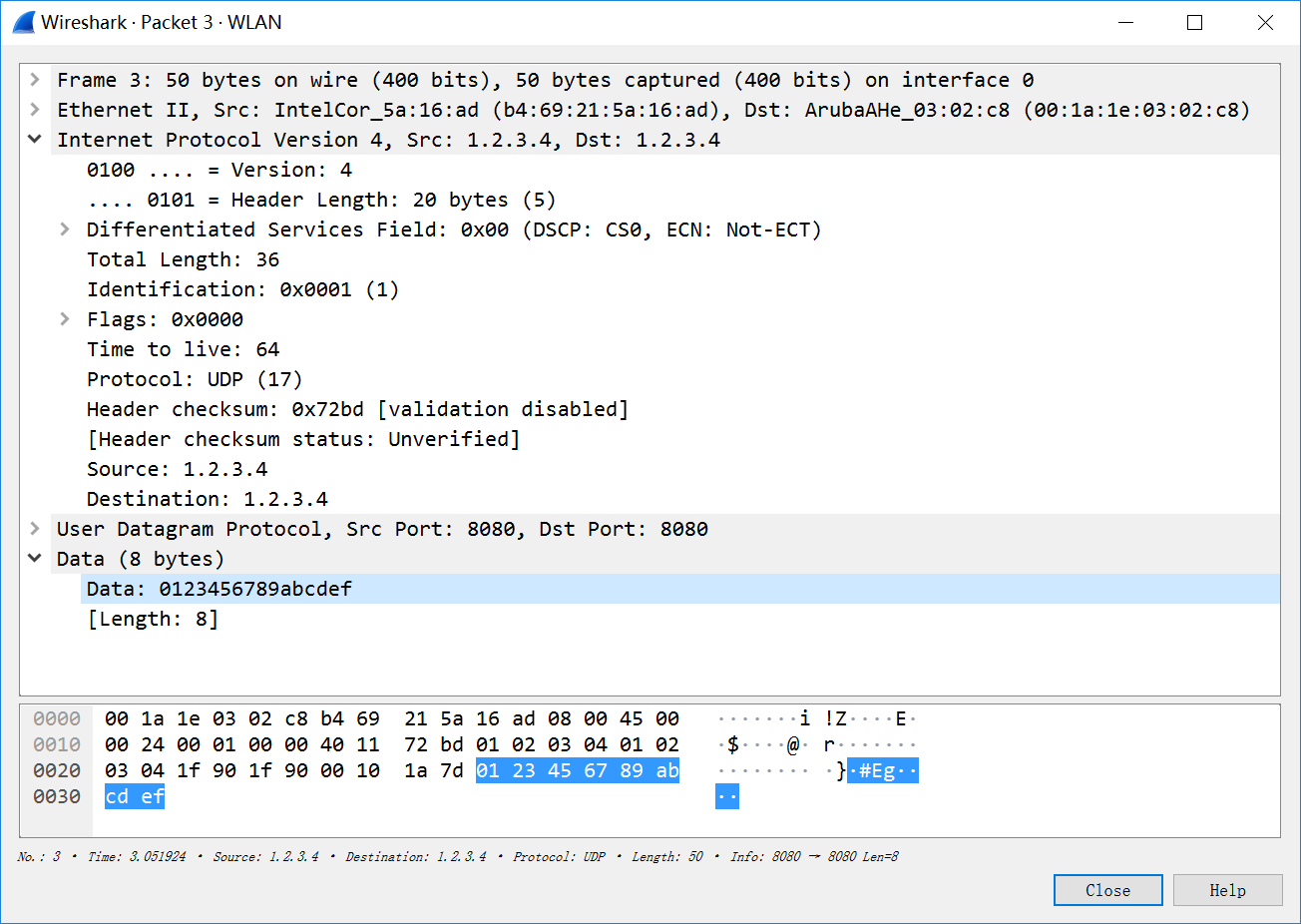


Fig 2.1.3.3 UDP包实际抓包结果

**2.1.4 ARP发包**

在上方文本框内输入对应信息，点击“生成数据包”按钮，即会在下方文本框内输出要发送的包格式，点击“发送数据包”按钮完成发包，下方文本框中出现Send a packet！字样，发包完成。

输入出现问题时，下方文本框中显示Input error！

具体选择项信息：（\*代表必选项）

**\*源IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*目的IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*源Mac地址**：48bit标准Mac地址格式，如00:ff:ba:42:75:aa。

**\*目的Mac地址**：48bit标准Mac地址格式，如00:ff:ba:42:75:aa。。

**\*ARP操作码**：16bit,数字int格式，1表示ARP请求包，2表示ARP应答包，默认为1。

**硬件类型**：16bit，数字int格式，默认为1。

**硬件类型长度**：8bit，数字int格式，默认为6。

**上层协议类型**：16bit，数字int格式，默认为0x0800。

**上层协议类型长度**：8bit，数字int格式，默认为4。

下面给出一个示例，以下是一个ARP包的构造过程：

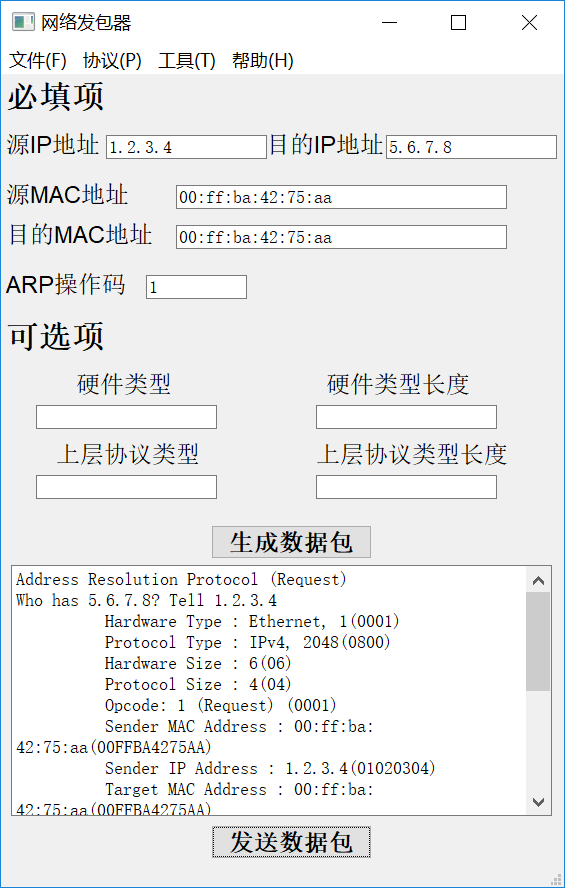


Fig 2.1.4.1 ARP包的构造

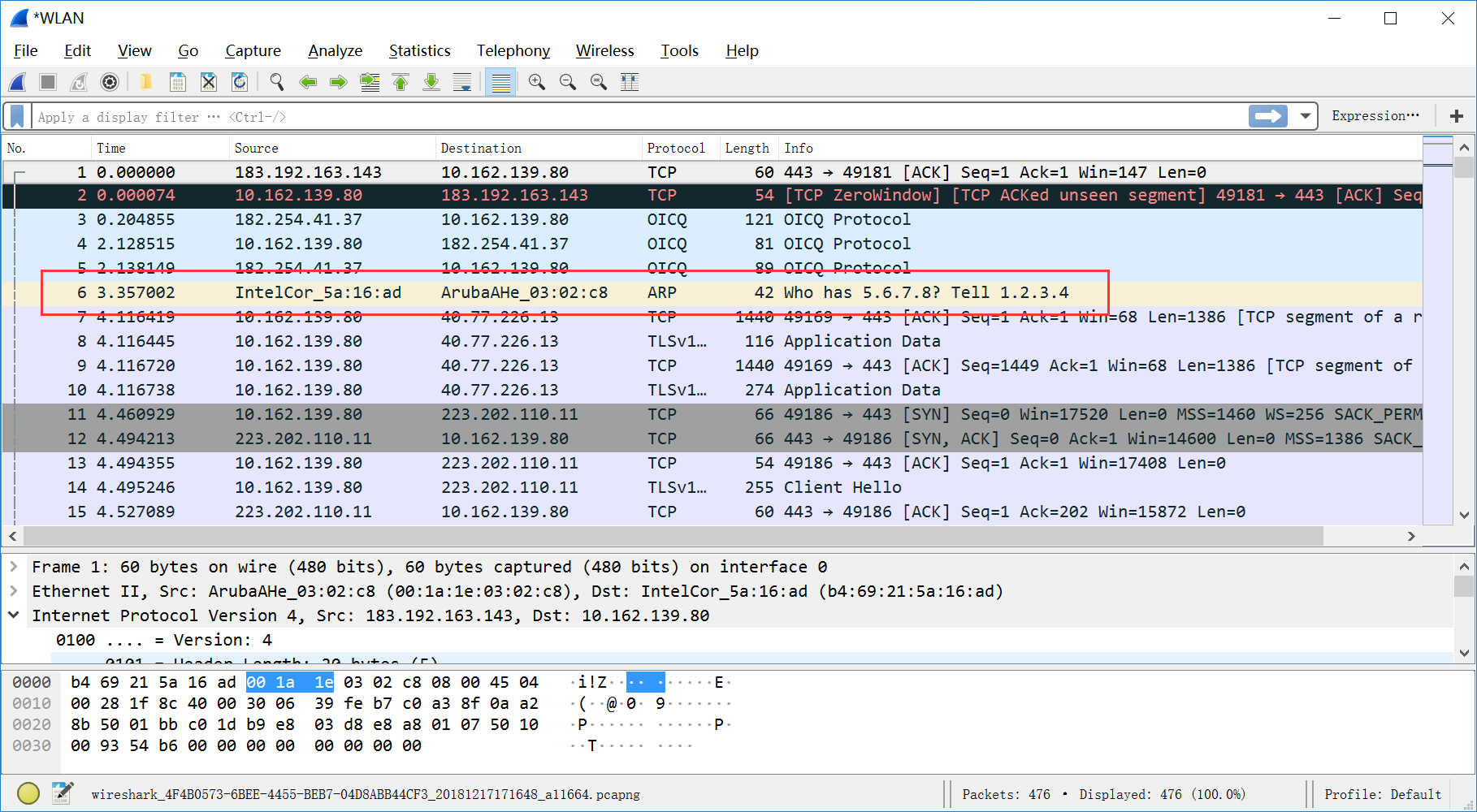


Fig 2.1.4.2 ARP包实际发送结果

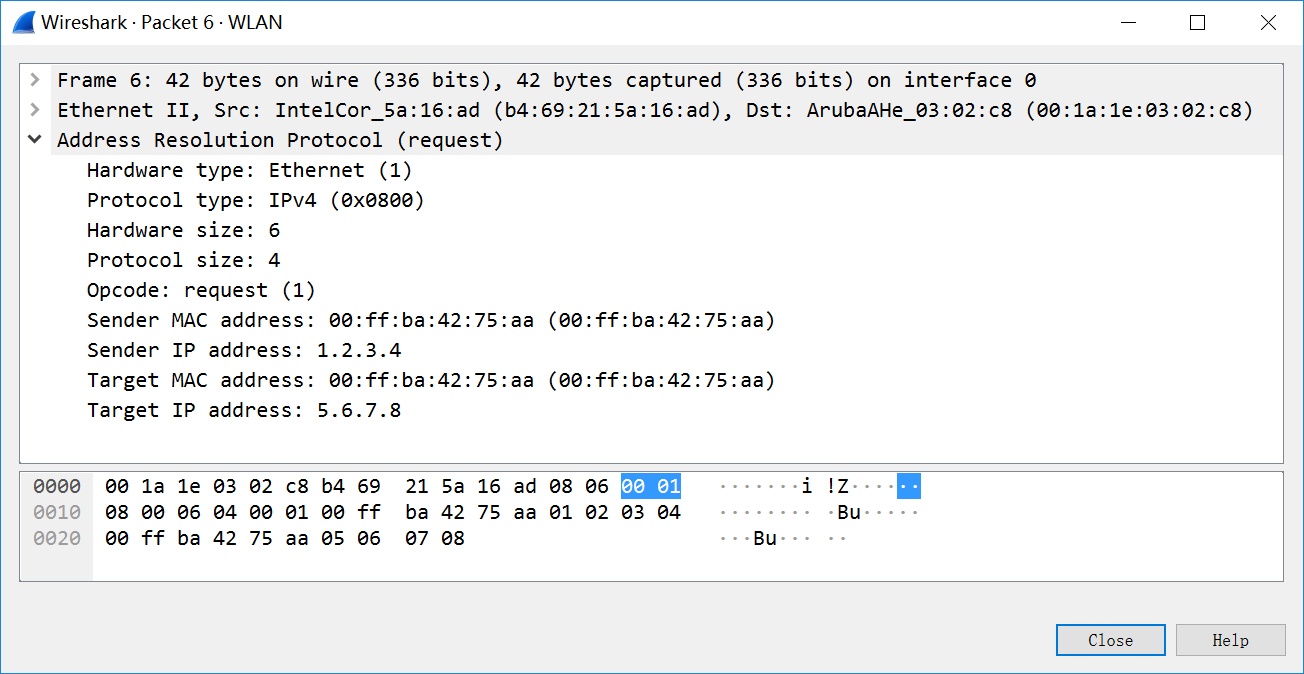


Fig 2.1.4.3 ARP包实际抓包结果

**2.1.5 ICMP发包**

在上方文本框内输入对应信息，点击“生成数据包”按钮，即会在下方文本框内输出要发送的包格式，点击“发送数据包”按钮完成发包，下方文本框中出现Send a packet！字样，发包完成。

输入出现问题时，下方文本框中显示Input error！

具体选择项信息：（\*代表必选项）

**\*源IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*目的IP地址**：地址格式为IPv4格式，如1.2.3.4。

**\*报文格式**：ICMP报文作用和格式，8bit，数字int格式，默认为8。

**\*报文类型**：详细说明ICMP报文类型，8bit，数字int格式，默认为0。

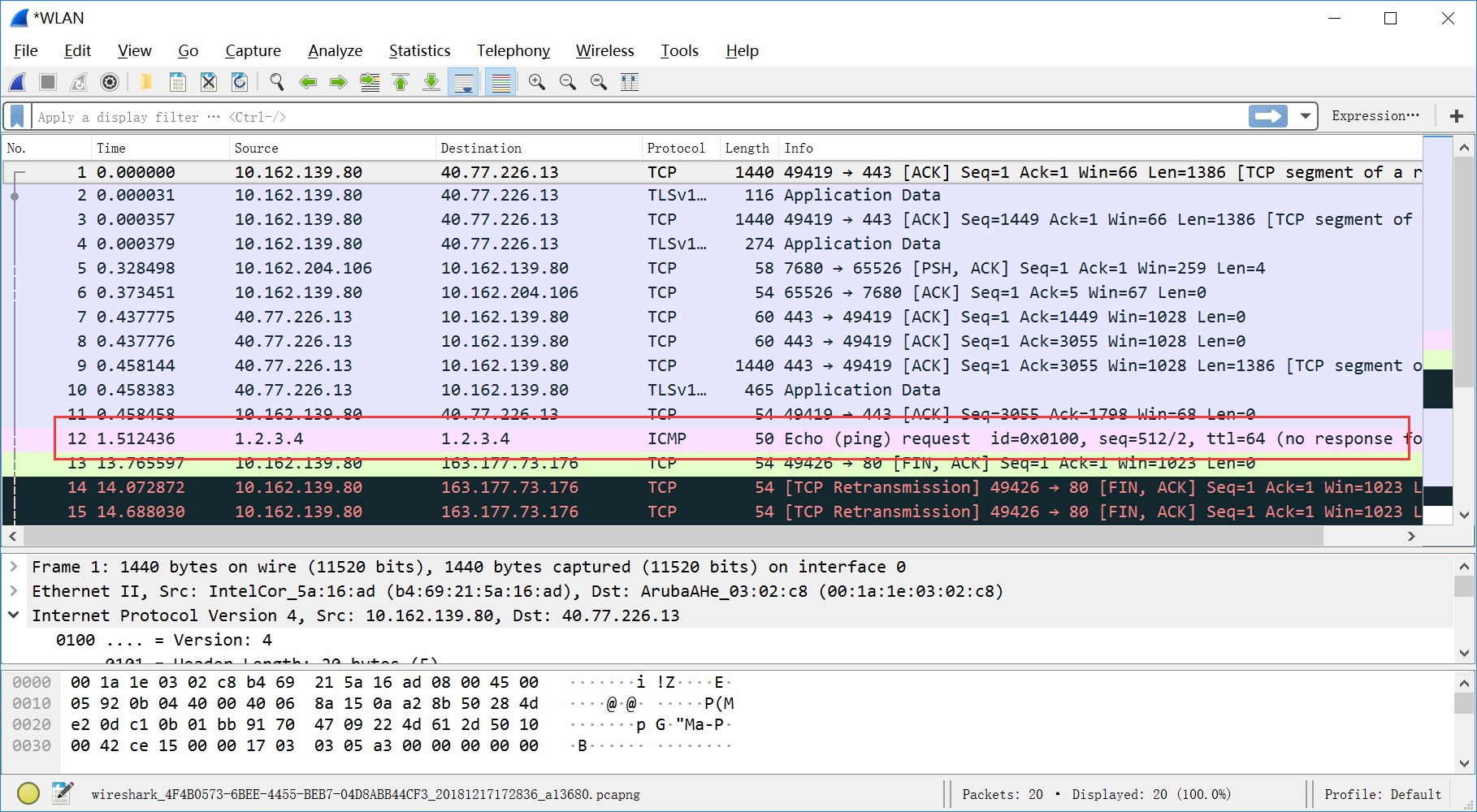
**\*数据**：TCP包负载（payload），输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串，如0123456789abcdef。

**可选头部**：输入限定为由0-9 a-f A-F组成的字符串，即十六进制数据字符串。

下面给出一个示例，以下是一个ICMP包的构造过程（这是个Ping包）：



Fig 2.1.5.1 ICMP包的构造

Fig 2.1.5.2 ICMP包实际发送结果

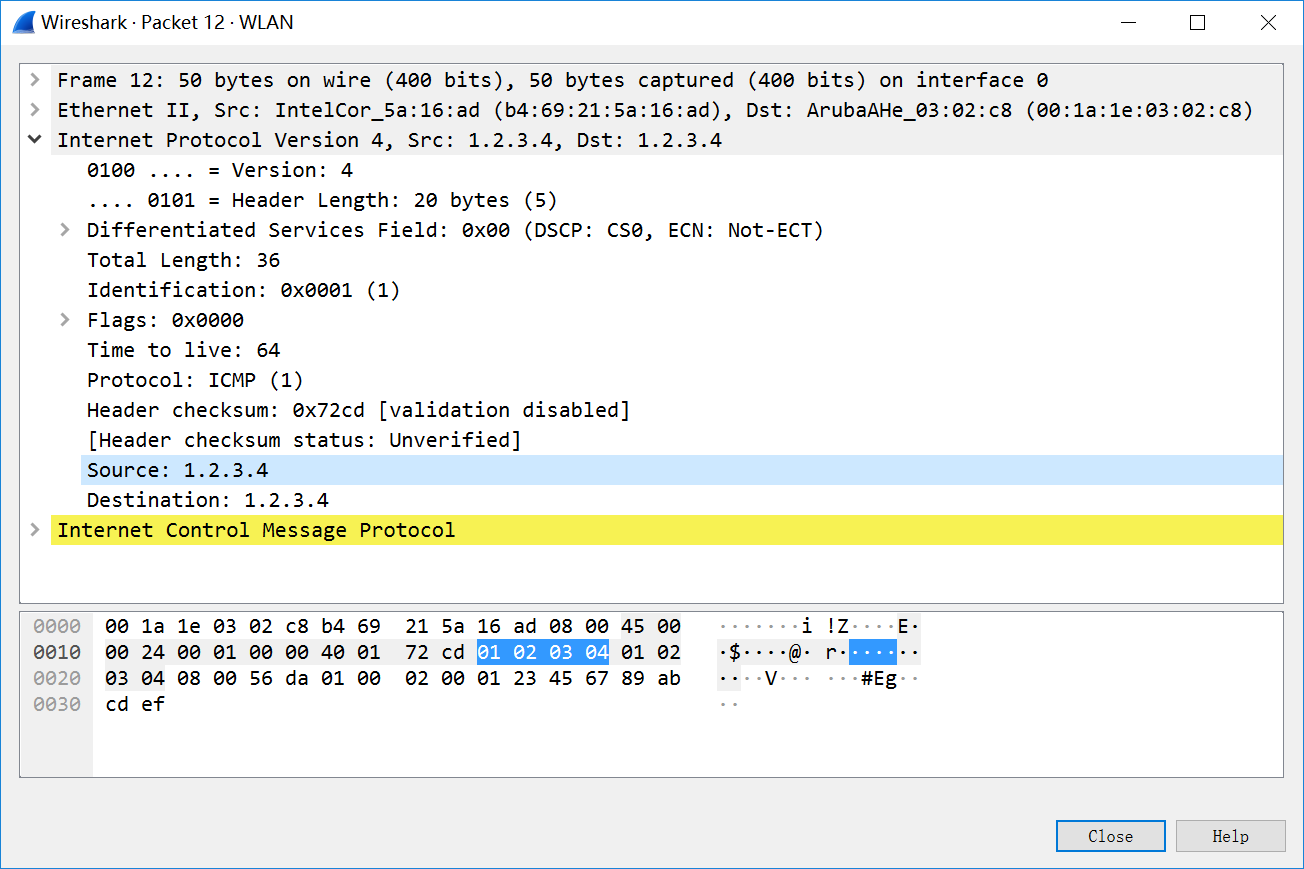


Fig 2.1.5.3 ICMP包实际抓包结果

**2.2 数据包以PCAP形式保存**

该功能用于将已经生成的数据包存到本地PCAP文件中，保存方式为“添加”模式，即在本地PCAP文件末尾插入需要保存的数据包，不影响PCAP文件原本的数据。

点击菜单栏的“文件(F)”，在下拉菜单中选择“以PCAP形式保存”，弹出对话框，点击“...”按钮后可选择本地文件，选择后输入框出现文件地址，点击“保存”按钮即可插入需要保存的数据包。

保存成功后，出现“保存成功！”字样。

**注意**：该功能只适用于通过发包界面“生成数据包”后，保存生成的数据包。

下面以2.1.1中生成的5个网络包为例，展示这个过程：

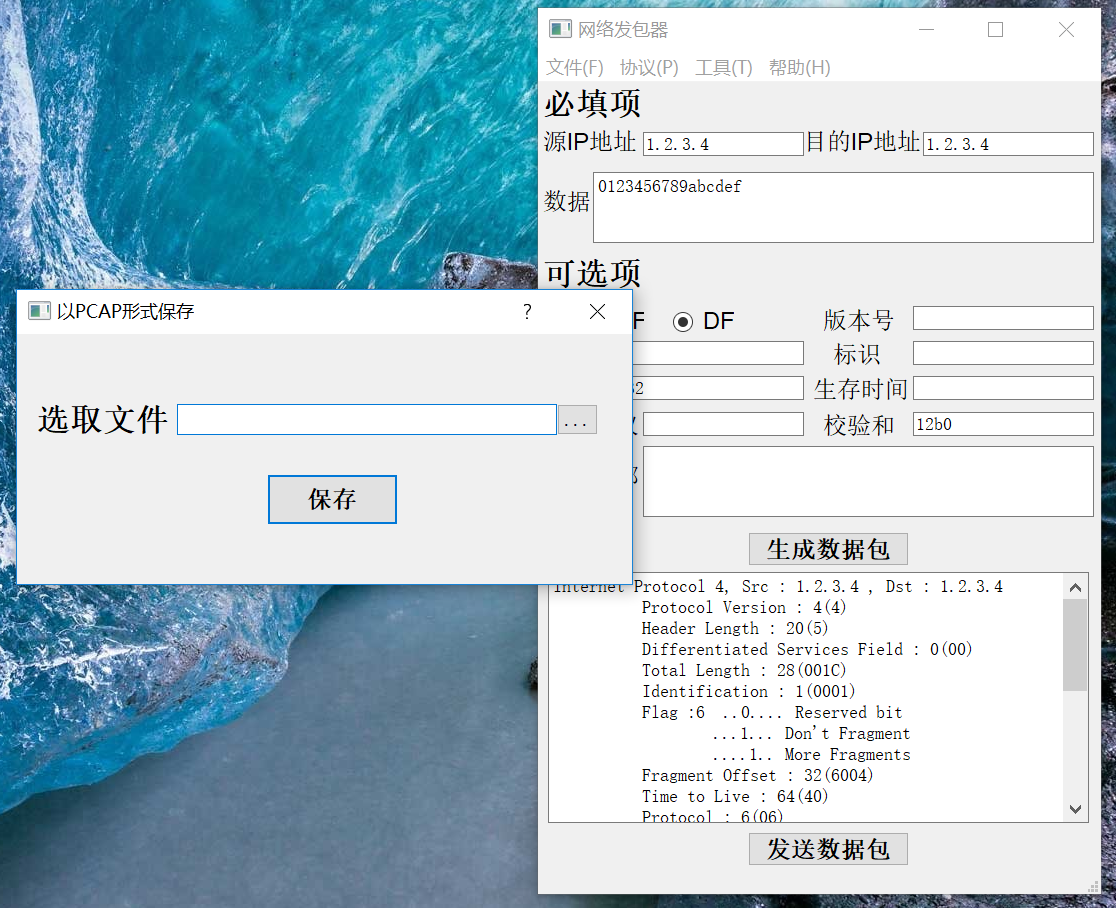


Fig 2.2.1 选择PCAP文件

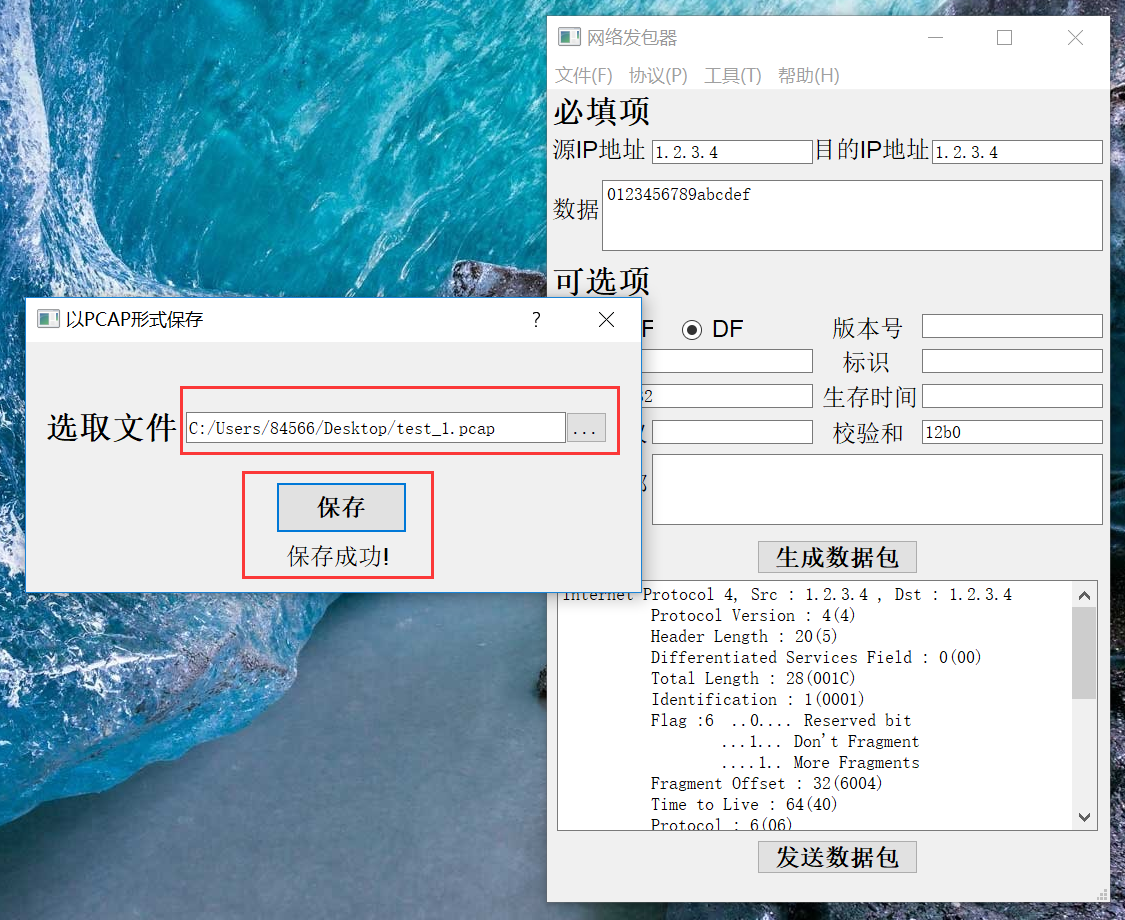


Fig 2.2.2 写入PCAP文件

如此一来，把2.1中5个数据包全部写入了test\_1.pcap这个文件中。这里我们如果用wireshark将test\_1.pcap文件打开，那么可以看到，wireshark能够识别我们写入的这五个数据包。这说明我们的写入操作是成功的。

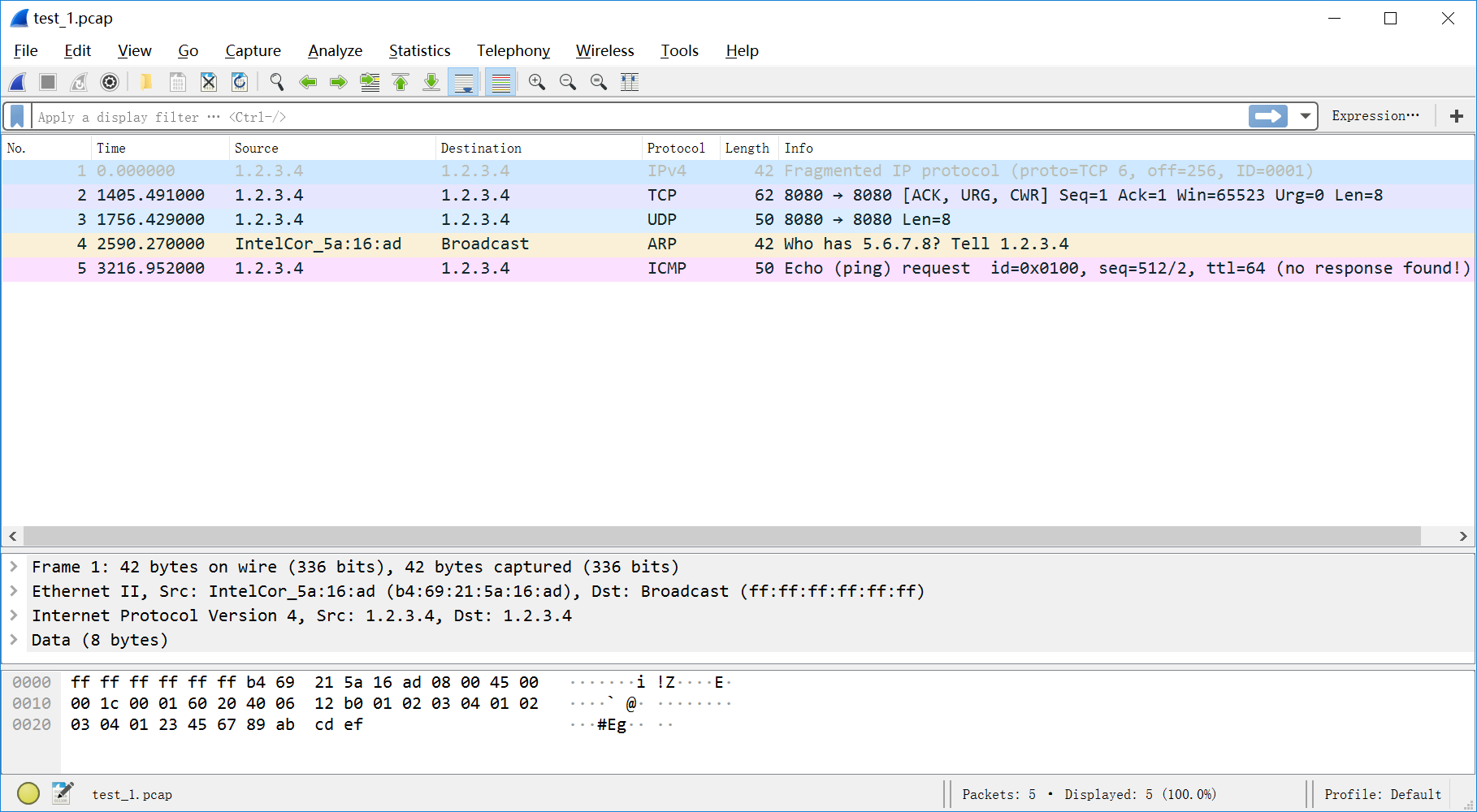


Fig 2.2.3 wireshark中对于2.1中五个数据包的展示

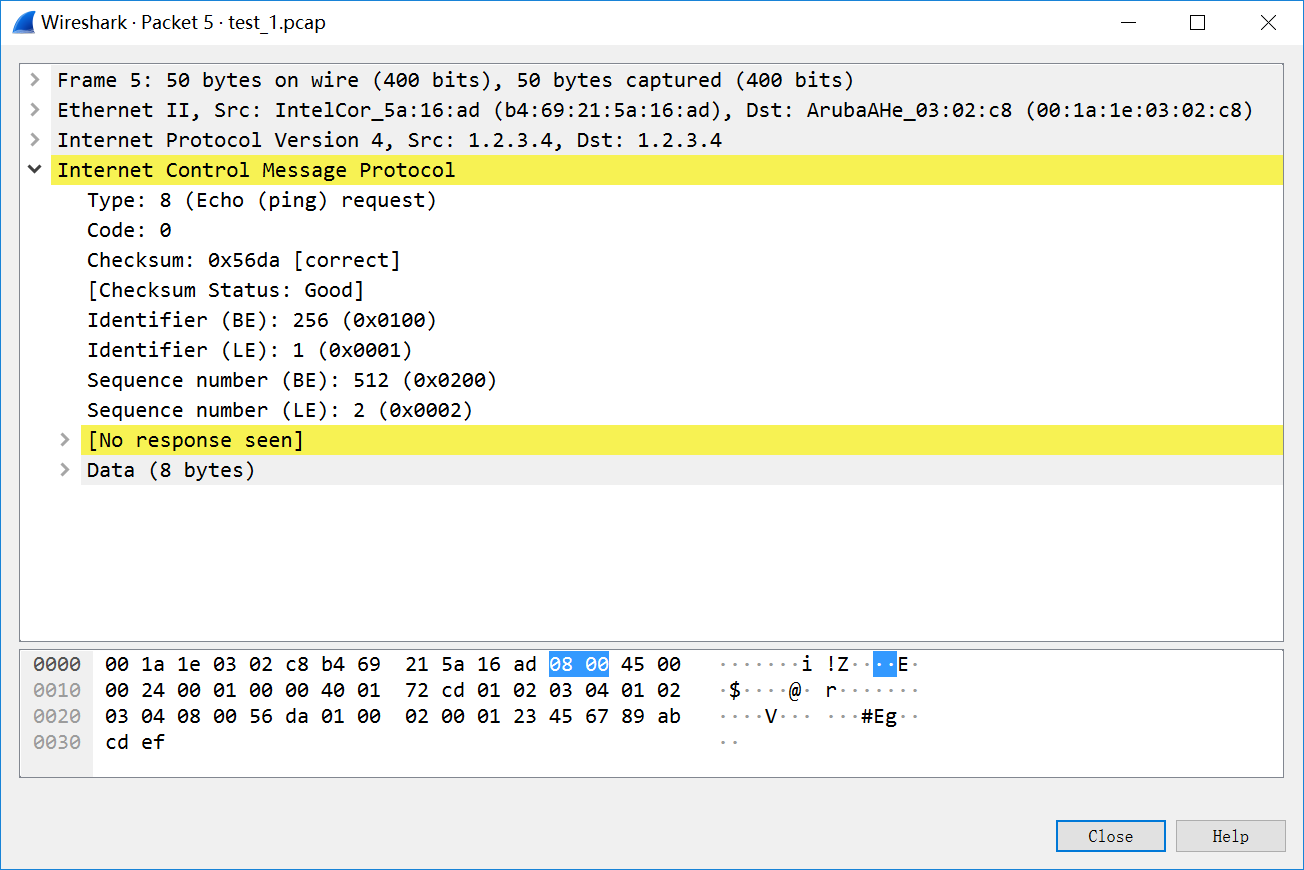


Fig 2.2.4 wireshark中对于2.1.5中ICMP数据包的展示

**2.3 数据包从PCAP文件读取（并发送）**

该功能用于读取本地PCAP文件中的数据包，进行一定程度的包过滤，并支持发送从本地PCAP文件中选中的数据包。

1、点击菜单栏的“文件(F)”，在下拉菜单中选择“从PCAP文件读取”，弹出对话框。

2、点击“文件：”后面的“...”按钮后可选择要读取的本地PCAP文件，选择后输入框出现文件地址。

3、下面的“包过滤”各项是过滤选项，选中想要过滤的选项，有的需要填入过滤条件，过滤条件的格式和发包界面需要的格式相同。

4、过滤选项完成后，点击“查看滤过的数据报”按钮，下方框中出现选中PCAP文件中满足过滤要求的数据包的序号和摘要。

5、在下方输入要发送的数据包的序号（过滤后输入框中出现的序号），点击右侧“确认”按钮，右端输入框中出现数据包完整格式，点击“发送数据包”即可发送。

6、发送成功后，出现Send a packet！字样。

**注意**：这里过滤要求采取的是并集。

**注意**：这里DATA的过滤要求采取的是子串，即找到password等关键词。

下面是一个读取文件的例子。可以看到，我们读入了刚才的test\_1.pcap文件，并且在过滤要求中选择了UDP和ICMP，于是我们成功将刚才写入的五个数据包中的UDP和ICMP包导入了进来，下方框中是它们的简要叙述，包括了基本信息与时间戳以及序号。我们选择序号为4的ICMP包，因而在下方填写4，点击右箭头，将网络包具体情况输出出来，如下图所示：

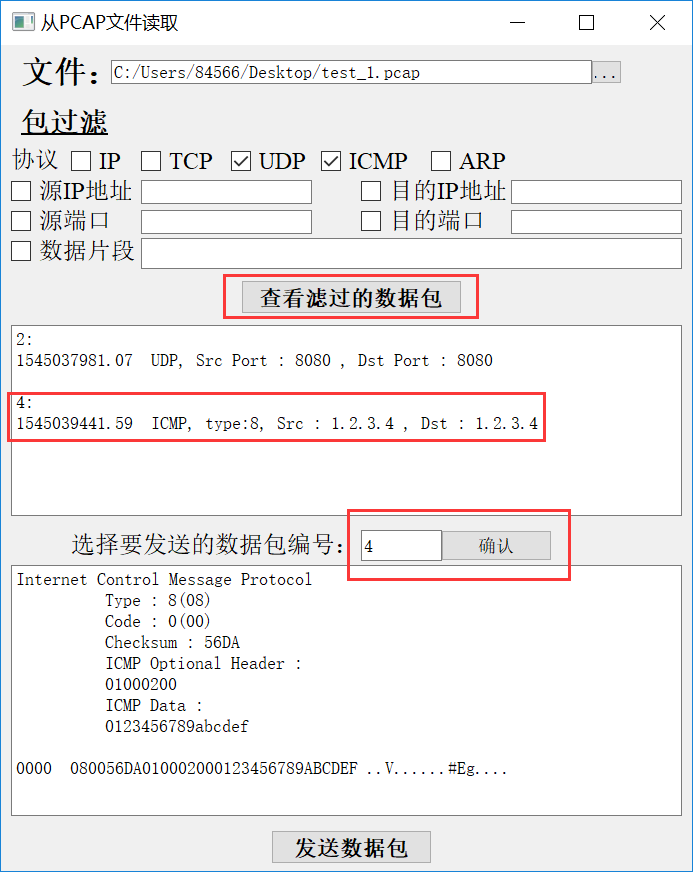


Fig 2.3.1 读取PCAP并过滤UDP与TCP包

在测试过程中我们发现，我们保存自己生成的数据包应为“添加”模式，所形成的PCAP文件与实际的PCAP文件并不相同。我们通过wireshark抓取了一段时间内的数据包，保存为“test\_2.pcap”文件，进行更为详细的测试。

我们读入了test\_2.pcap文件，并且在过滤要求中选择了TCP和ARP，设置目的IP 地址为224.0.0.1，成功将文件中目的IP 地址为224.0.0.1的TCP和ARP包导入了进来，下方框中是它们的简要叙述，包括了基本信息与时间戳以及序号。

我们选择序号为63的ARP包，因而在下方填写63，点击“确认”，将网络包具体情况输出出来，如下图所示：

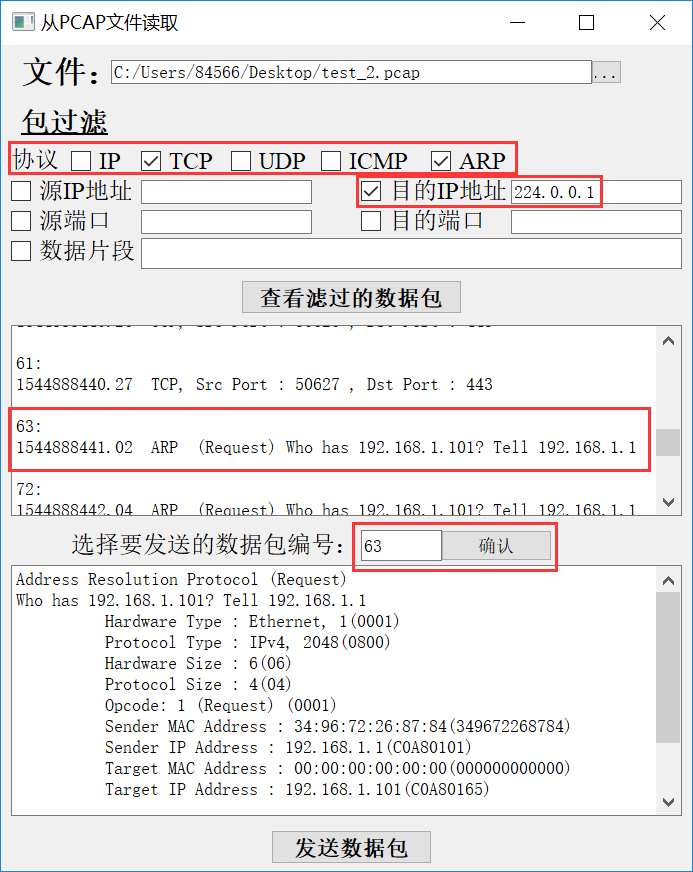


Fig 2.3.2 读取PCAP并过滤目的地址

此时我们点击“发送数据包”，就可以把这个数据包给发出去了。发包情况和前面是一样的，用wireshark可以抓到，这里不再赘述了。

**2.4 密码工具**

这是自主增加的内容，方便对数据包的数据进行各种简单密码处理。所有出现的加密数据均可通过复制成为报文数据。

**2.4.1 AES加密**

这个工具可以进行AES加密解密处理。打开对话框后，填入AES加密需要的密钥和IV，上方的框中输入要加密的数据，点击“加密”按钮，下方即出现加密后的数据。下方的框中输入加密后的数据，点击“解密”按钮，上方的框中即出现解密的数据。当想要重新输入需要加密或者需要解密的数据时，可以点击“清零”按钮，清零数据框。

**注意**： （1）使用CBC模式。

（2）key必须是16，24，32byte的hex格式字符串数据。

（3）IV必须是16 byte的hex格式字符串数据。

（4）如果出现错误的key/IV/输入数据，文本框中会报错。

下面是三个使用实例：



Fig 2.4.1.1 AES样例(AES Key\_16byte)

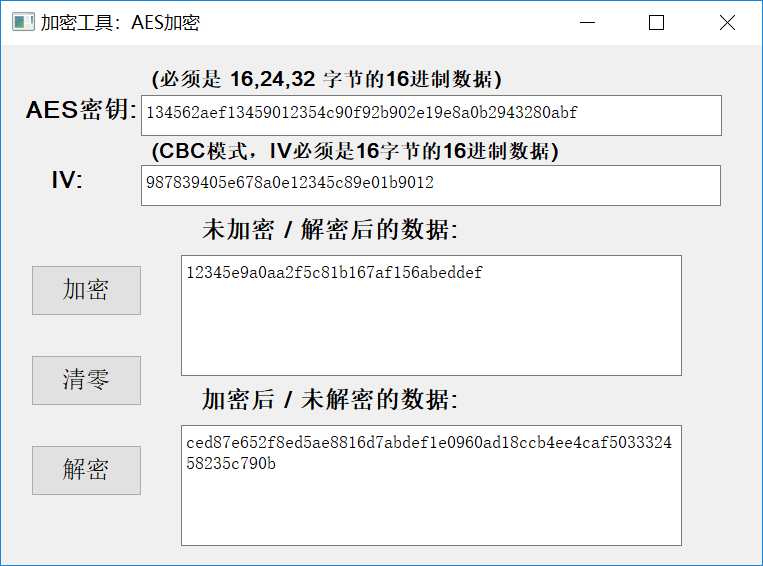


Fig 2.4.1.2 AES样例（AES Key\_24byte）

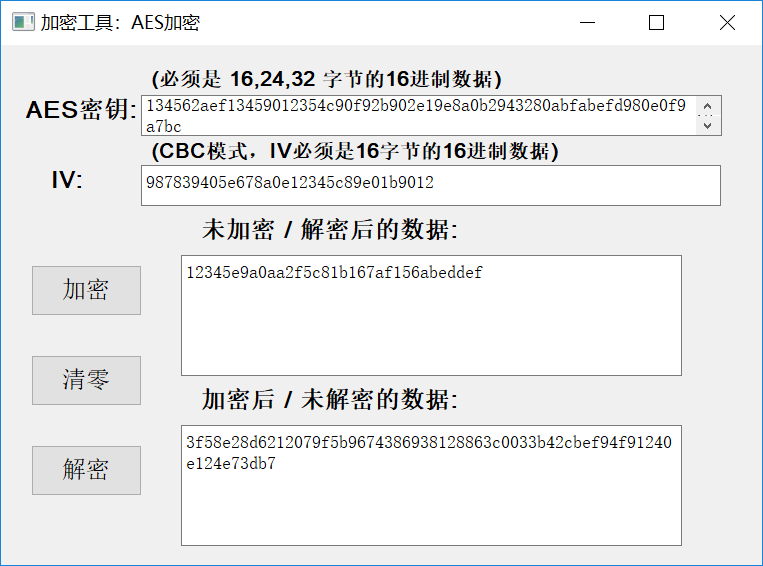


Fig 2.4.1.3 AES样例（AES Key\_32byte）

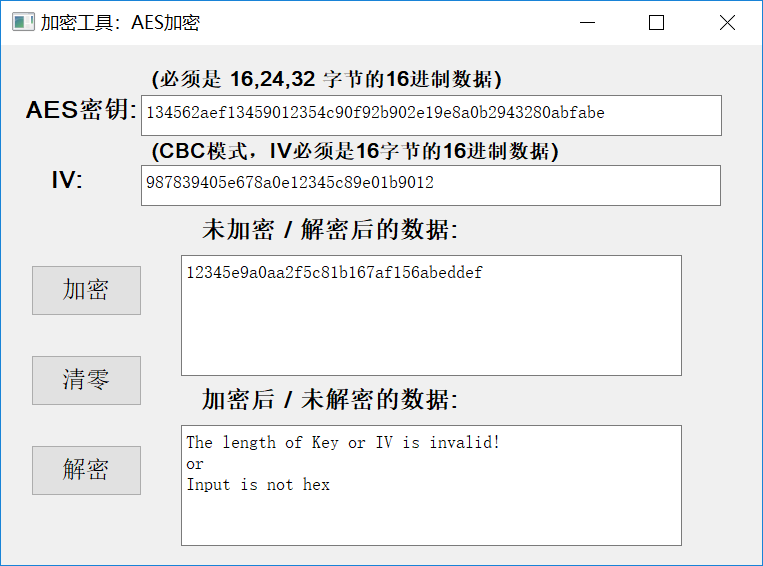


Fig 2.4.1.4 AES样例（AES密钥长度错误时加密报错）



Fig 2.4.1.5 AES样例（AES密钥长度错误时解密报错）

**2.4.2 Hash(SHA256)加密**

这个工具对数据进行Hash处理，使用的算法是SHA256算法。在左侧框中输入数据（可以为任意字符），点击“哈希加密”按钮，右侧即出现经过Hash加密后的结果。因为Hash不能反向解密，所以此部分没有解密功能。当想要重新输入需要加密或者需要解密的数据时，可以点击“清零”按钮，清零数据框。

下面是一个使用实例：

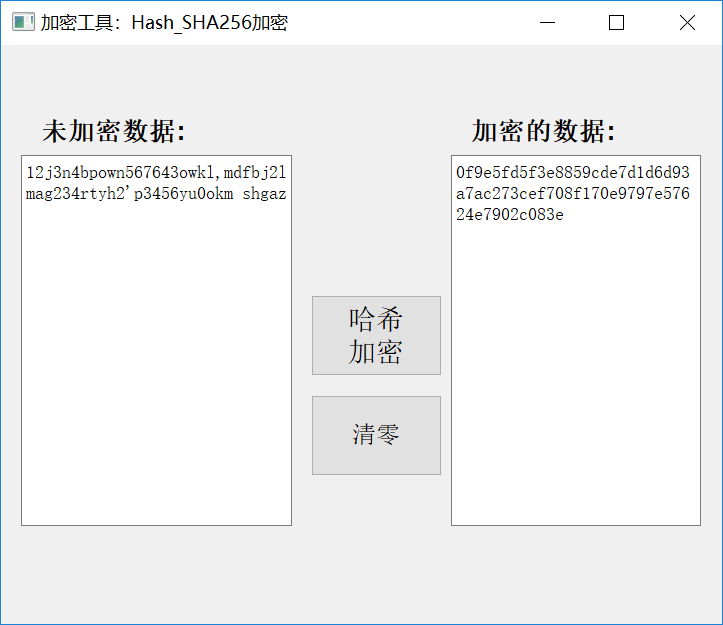


Fig 2.4.2.1 SHA256示例

**2.4.3 RSA密钥**

这个工具用于产生进行RSA加密和数字签名的公钥和私钥文件。输入要保存为的文件名（如“xx”），点击Create RSA Key File按钮，会在程序文件目录下生成名为“xx-public.pem”的公钥文件和名为“xx-private.pem”的私钥文件。

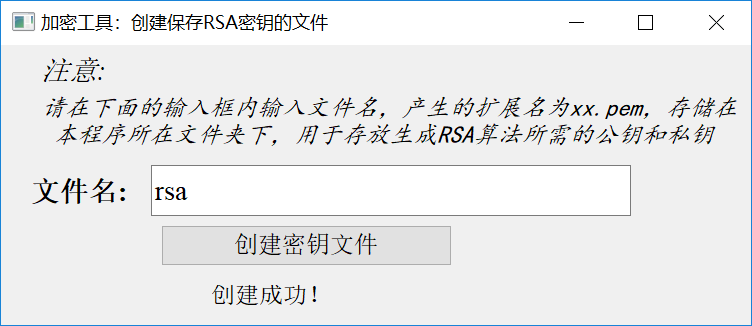


Fig 2.4.3.1 RSA密钥文件生成示例



Fig 2.4.3.2 RSA密钥文件生成结果

**2.4.4 RSA加密**

这个工具用于通过RSA密钥生成的公私钥文件进行加密解密。根据要求选择本地文件中的公私钥文件。上方的框中为要加密的数据，点击“加密”后下方出现加密后数据。下方框中为加密后的数据，点击“解密”后左侧出现解密后的数据。当想要重新输入需要加密或者需要解密的数据时，可以点击“清零”按钮，清零数据框。

**注意**：输入数据必须是hex格式字符串数据。

下面是一个实例：

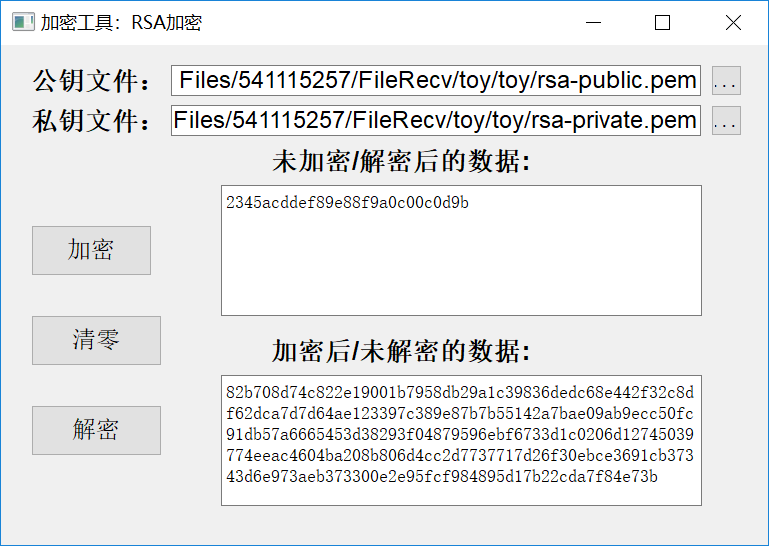


Fig 2.4.4.1 RSA密钥加密结果

**2.4.5 RSA数字签名**

这个工具用于通过RSA密钥生成的公私钥文件进行数字签名和签名验证。根据要求选择本地文件中的公私钥文件。左侧框中为要签名的数据，点击“转换”后右侧出现数字签名。在左侧存在签名前的数据且右侧存在数字签名时，点击“检查”按钮，判断签名正确性。若正确右侧框中出现Signature Check correct!字样，错误出现Signature Check wrong!字样。当想要重新输入需要数字签名的数据时，可以点击“清零”按钮，清零数据框。

**注意**：“检查”按钮必须同时有签名前的数据和数字签名才能起作用。

**注意**：输入数据必须是hex格式字符串数据。

下面是一个使用实例：



Fig 2.4.5.1 RSA密钥签名结果

**2.4.6 16进制与字符串转换**

这个工具用于对数据进行十六进制hex字符串和普通字符串之间的相互转换。左侧输入十六进制数据hex字符串，点击“16进制到字符串”按钮，右侧出现对应的字符串数据。右侧输入字符串数据，点击“字符串到16进制”按钮，左侧出现对应的十六进制hex字符串。当想要重新输入16进制数据或者字符串数据时，可以点击“清零”按钮，清零数据框。

下面是一个使用实例：



Fig 2.4.6.1 16进制数据与普通字符串的转化

**2.5 用户帮助与制作人员**

本发包器拥有极其详细的用户帮助说明，并且对制作人员有一个展示页面（详情），如下图所示：

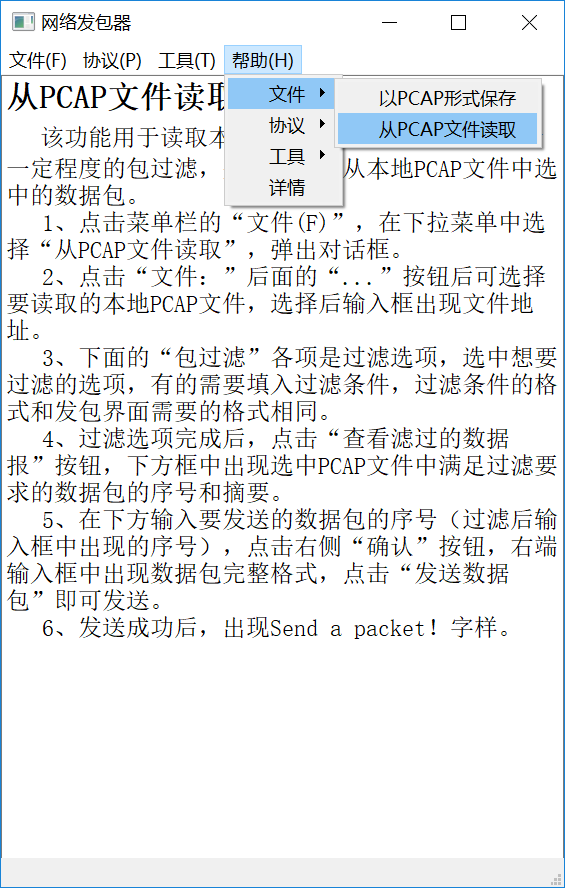


Fig 2.5.1 帮助文档1

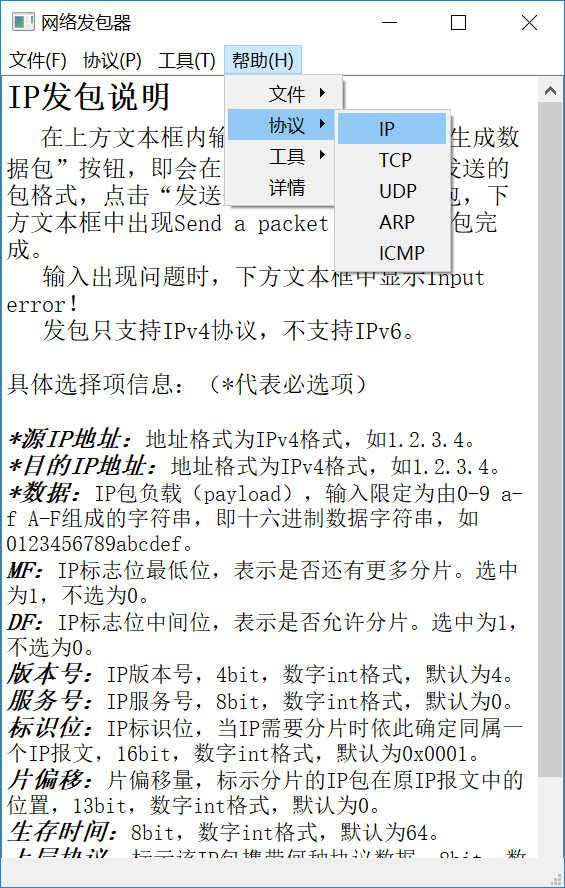


Fig 2.5.2 帮助文档2

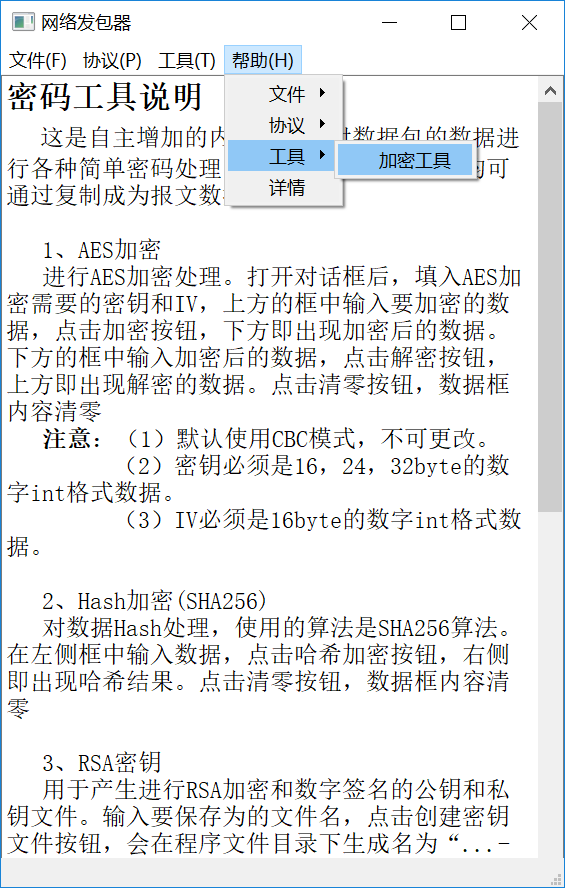


Fig 2.5.3 帮助文档3

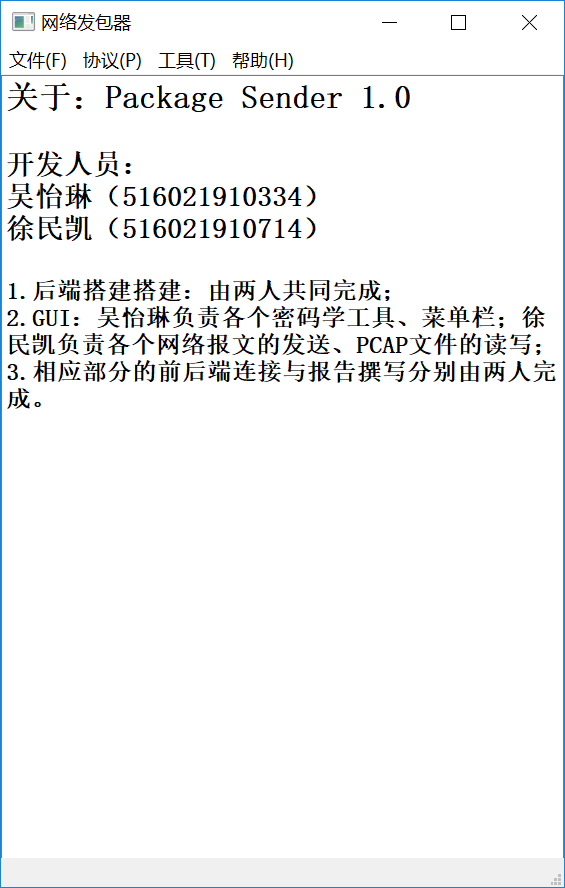


Fig 2.5.4 制作人员

**三、遇到的问题以及解决方法**

本次大作业完成过程中遇到了诸多问题，以下进行举例说明：

**1. 发包库的选择**

对于发包库而言，我们有两个选择：socket与scapy。通过上网查询了很多资料，我们发现：

1. socket对于TCP和ARP的支持比较差，特别是TCP，socket的内核甚至会自行把我们发出去的TCP包给reset掉，没有网络通信的效果；
2. 相比之下，Scapy不仅对windows有很好的支持，并且有着从二层和三层发包两重功能，非常适合发包器这一应用。

因此，我们最终采用了scapy库进行发包。

**2. PyQt5安装问题**

PyQt5官方提供的只有Python3版本，网上的Python2.7安装资料几乎没有，我们尝试下载源码自主编译但最终失败。最终，我们找到直接通过github源码安装的方法，详细过程见附录。

**3. 发包函数的选择**

如上所述，scapy可以实现两层的发包，其中：sendp是scapy中负责二层发包的函数，而send是负责三层发包的函数。在作业过程中我们发现，sendp始终没有达到正确的发包效果，wireshark无法抓到生成的数据包，而在使用send函数后却可以正常发包。经过网上查阅资料，我们判断，这可能与windows上的interface有关。通过对系统的相关配置，我们最终实现了使用sendp发包。但考虑到方便用户的使用，我们最终还是全部采用了send函数，从三层发包，使得所有机器都能正常运行程序。

**4. 测试发包的问题**

在结束代码工作之后，我们尝试了使用所写的发包器进行测试，但每次都在点击“发送数据包”之后因为某些问题而闪退。这个闪退并没有错误提示，只有process exit with code 1的字样。我们进行了很多尝试，最终也没能找到问题所在。

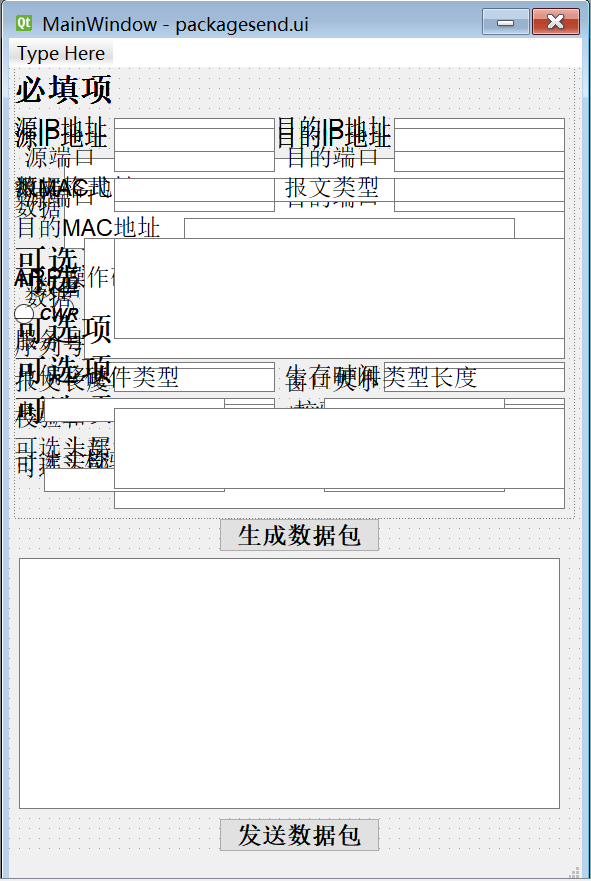
最终，我们决定暂时不管这个闪退的问题，而先用wireshark测试发包是否成功。在使用wireshark时，我们发现wireshark没能找到wlan，上网查找信息是因为某项网络服务没有打开。为此，我们在windows设备管理器的“查看-显示隐藏的设备”一栏NetGroup Packet Filter Driver 右键属性—驱动程序—启动类型，修改类型为“系统”；而后在cmd下输入net start npf，打开网络抓包服务。而这个发包器却解决了闪退问题。最终我们查明，闪退的原因来自于网络服务没有打开，而不是因为代码的问题。

**5. 密码工具实现的问题**

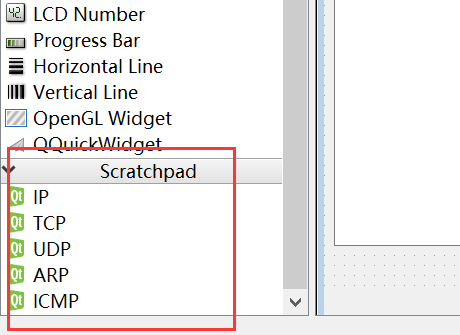
这里我们采用了PyCrypto来实现加密解密签名哈希等功能。PyCrypto安装起来也有一点麻烦。在Settings的Project Interpreter里直接添加相应的库pycrypto,但是在安装成功后，直接import Crypto的时候总是报错，说module不存在，后来发现安装完成后在\Lib\site-packages下的文件夹默认为crypto,将首字母改成大写之后解决了问题。后来在写密码工具的时候，也碰到了一堆问题，这里举例一个。譬如在写AES加密的时候，PyCrypto的AES函数要求输入数据的字节数必须是16的倍数，但是又不帮你做padding的工作。因此如何填充，并且填充加密解密之后如何把填充去掉，这个都得自己去看文件和标准。因此虽然说是调用一个库，其实一点都不简单，问题还是很多。好在最后都能解决。

**6. Qt Designer设计问题**

在Qt Designer中进行UI设计时，有时由于多个界面需要相互覆盖，很难进行设计，如下图所示：



针对这个问题，我们在实际操作过程中，发现Qt Designer可以将设计好的QWidge保存为模板形式进行暂存。通过这一功能，我们先将不同的QWidge设计好，最后加入到主界面中。模板功能如下所示：



**7. 调试过程中出错界面闪退问题**

在运行过程中，点击一些触发按钮后，经常由于写错函数或输入不符合要求等各种各样的问题的出现，导致连接到的函数由于输入参数出错等问题报错，而PyQt的界面本身又不会有错误信息显示在控制台上，而是直接将界面闪退。例如，在设计UI时，我们将某处的UDP误打为UCP，使得与UDP相关的操作都会闪退。我为此花费了很多时间修改UDP的pack与send部分，但都无果。最终，在设计的UI部分发现了错误，使得UDP部分的功能得以正常运行。

**8. slot and signal 问题**

GUI界面的控件布局还是相对简单的，直接通过external tools打开qt designer，选中左侧工具栏中的对应控件拖动它们到相应的位置就可实现界面设计，但是比较困难的是设计界面不同部分信号与槽的连接，通过图形化的界面可以选择已有的函数进行触发连接，但是如果要添加自定义的函数触发，还是需要通过将.ui文件转化为.py文件之后，修改py文件中的xxx.clicked.connect.triggered()实现连接，一开始我们试图直接使用图形化界面完成设计，后来发现在.py文件里编程设计是必不可少的。

**9. 选协议问题**

因为我们将协议的选项放在菜单栏的二级菜单中，通过QAction触发连接show()来显示协议界面，所以当某个协议的界面被触发时，我们只能够看到某个填写协议的画面被显示出来，但是内部是不知道哪一个协议被触发了。但是在点击生成数据包的时候，由于要和后端连接，需要判断当前的数据对应的是哪个协议，因此要设置一个变量self.Index,用0-4来分别对应ip、tcp、udp、arp、icmp,触发时将index设置为对应值，生成数据包时根据index的值产生对应协议的报文。

**10. 保存密钥的文件格式问题**

关于如何存放RSA的公钥和私钥，一开始我们以为生成了之后随便存储在类似于txt格式的文件中即可，方便后面读取。后来经过baidu和google之后，终于找到了合适的存储方式，也对公钥和私钥的生成有了更深的理解。PEM是OpenSSL和许多其他SSL工具的标准格式，OpenSSL使用PEM文件格式存储证书和密钥。PEM文件是Base64编码的证书。PEM证书通常用于web服务器，因为他们可以通过一个简单的文本编辑器，很容易地转换成可读的数据。通常当一个PEM编码在文本编辑器中打开文件,它会包含不同的页眉和页脚。

-----BEGIN CERTIFICATE REQUEST----- and -----END CERTIFICATEREQUEST-----

CSR(证书签名请求)

-----BEGIN RSA PRIVATE KEY----- and -----END RSA PRIVATEKEY-----

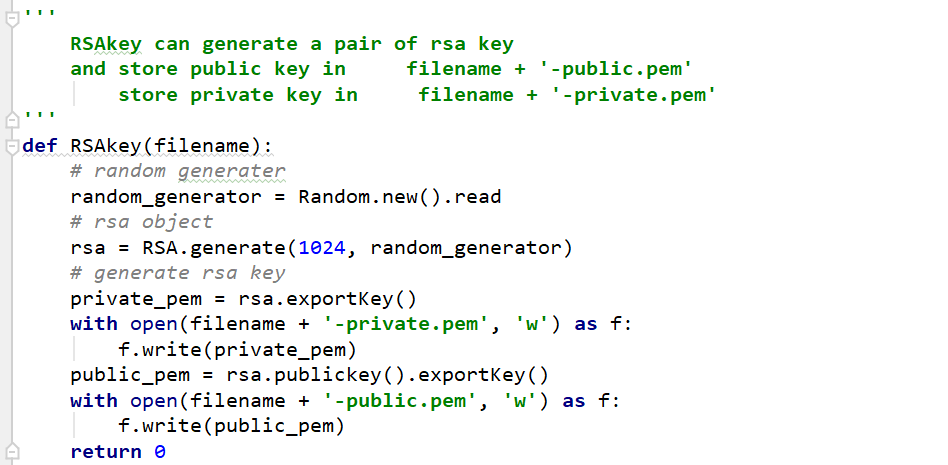
私钥

-----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----

证书文件

PKCS #8: Private-Key Information Syntax（语法） Standard（标准）

通过以下函数就可以轻松读取和保存。



**四、体会与建议**

此次计算机网络的大作业对我们来说确实是一个要求非常高的项目，虽然过程总是充满了痛苦和de不完的bug，但是最后实现了想要的功能之后还是非常有成就感的。无论是从实践的角度还是理论的角度，此次大作业我们都有很大的收获：

首先是实践的方面。基本上在此之前虽然写过很多实现不同功能的代码，但大部分都是通过程序实现某一功能，很少遇到需要设计GUI，将前端和后端结合起来，实现一个可以集合多种复杂功能的项目。而且开发这种大项目主要还是使用linux平台，平台上各种附加的包安装简便，环境与支持也更适合进行开发。这次因为要尝试GUI，所以决定在 windows平台上试一试，但是因为对GUI也不太了解，所以无形中增加了不少难度。果然在配置环境和调试上花了很多精力。但幸运地是scapy对windows有很好的支持，利用scapy能够比较顺利地编程实现相关功能。特定版本pyQt5的安装、配置与使用也花了不少时间，在后面调试代码的过程中也是错误不断。但我们最终在反复试错中找到了最后的解决方法，比较全面地实现了我们所设想的功能。

然后是理论知识方面。对于计算机网络中各种概念的理解更加深刻了。之前在计算机通信网课程中，像各个协议以及协议头中的各个部分，虽然学习的时候知道是什么，但概念还是很容易混淆。这次大作业，对加深对于计算机通信网知识的理解帮助非常大。编写生成包与发送包的函数需要对所有协议类型有着非常清晰深刻的认识。通过自己不断试错、不断寻找资料来学习，最终看到代码正确运行，心里确实充满了成就感。通过自己的学习思考和动手实践，对计算机网络中概念的理解自然也更加透彻。虽然老师之前也提过wireshark抓包软件，自己也尝试过用wireshark来抓包，但是展开抓到的包的内容还是不懂如何获得有用的信息，和上课的理论知识还是无法具体结合起来，总觉得协议太多，又分不清彼此的区别，但是这次大作业之后，自己设置发包器，通过数据和报头的设置，很多报文的内容自然就理解了。由此可见，即使是理论学习，实践操作绝对是能让理论知识最为巩固和理解深刻的步骤。

最后是关于课程的一些建议。这门课的大作业还是很有实践意义的，毕竟在这么短的时间内，每个小组只有1-2个人的情况下，完成这样一个工作量很大的项目对我们来说都是锻炼。可选的七个项目的内容确实是很有意思的，选择的范围也很广，也基本涵盖了我们这学期的所学内容，所以大家完全可以针对自己感兴趣的模块进行选择。从应用层的聊天程序到较为底层的设计都有涉及，然而因为大作业基本都堆积在最后一个月里，而且这个项目又是所有大作业里最复杂的，所以虽然是两个人合作，实际上也花了非常多的时间。我们不仅有很多个周末各自写代码线上讨论交流，还有很多次一起在图书馆讨论难以线上沟通的问题。从后端的分工到GUI的整体设计，这期间都有意见分歧和遇到棘手问题的时候，好在两个人一起合作完成总是比一个人要效率更高，不仅是工作量的分配更加合理，讨论和交流的过程中对方的意见可以更好地改进项目的不足，更及时地发现问题。当然实践都是需要花费大量时间的，虽然这次总体完成了大作业的要求，但是需要完善的地方还有很多，比如GUI的设计可以更加美观一些，可以增加另外的一些功能比如增加查看日志功能（可以查询之前发过的报文信息）。如果大作业可以适当减少广度而提高对深度的要求，使得我们可以将某一功能做得更加完善，我认为最后项目的质量可能会更高，对相应问题的理解也会更加的深入。

最后，感谢老师和助教一个学期的辛勤付出，也希望本组人员能够在接下来的考试中取得好成绩！

**附录：技术细节**

**一、GUI技术细节**

**1.1 PyQt5的安装**

因为我们的发包库只支持Python2.7，所以我们选择此版本Python作为作业的编译环境。因此，在考虑绘制GUI时，我们选择了Python下支持较好且效果很棒的Python GUI库PyQt。但PyQt的最新版本PyQt5虽然支持Python2.7，但官方只给出了Python3的安装工具。

经过我们的查找，我们发现了github上开源的通过PyPI下载PyQt for Python 2.7 64-bit on Windows的方法。在Python2.7上安装PyQt5的方法为：

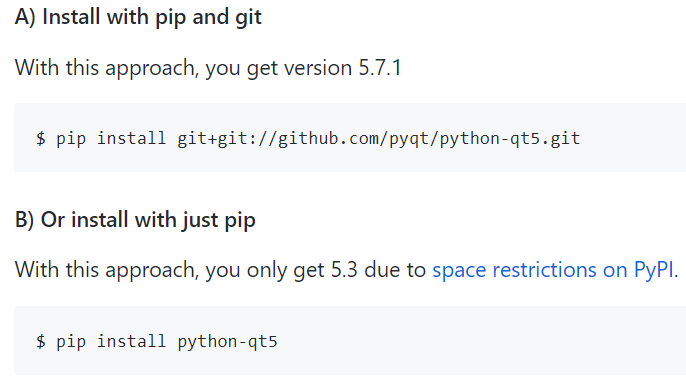


Fig1.1.1 Unofficial PyQt5 via PyPI for Python 2.7 64-bit on Windows

**1.2 配置python的External Tools**

由于我们选择的作业编译环境是Python，使用的IDE是PyCharm，所以我们可以直接将Qt\_Designer和PyUic添加到External Tools里，这样方便直接调用，进入Settings->Tools，点击上方的“+”，先设置PyUIC可以将.ui文件转化为.py文件，program一栏填写本电脑安装的python.exe的地址，其它栏如图所示；然后添加Qt Designer，program一栏分别填写本电脑安装的PyQt5目录下designer.exe，其余项的填写如图所示。

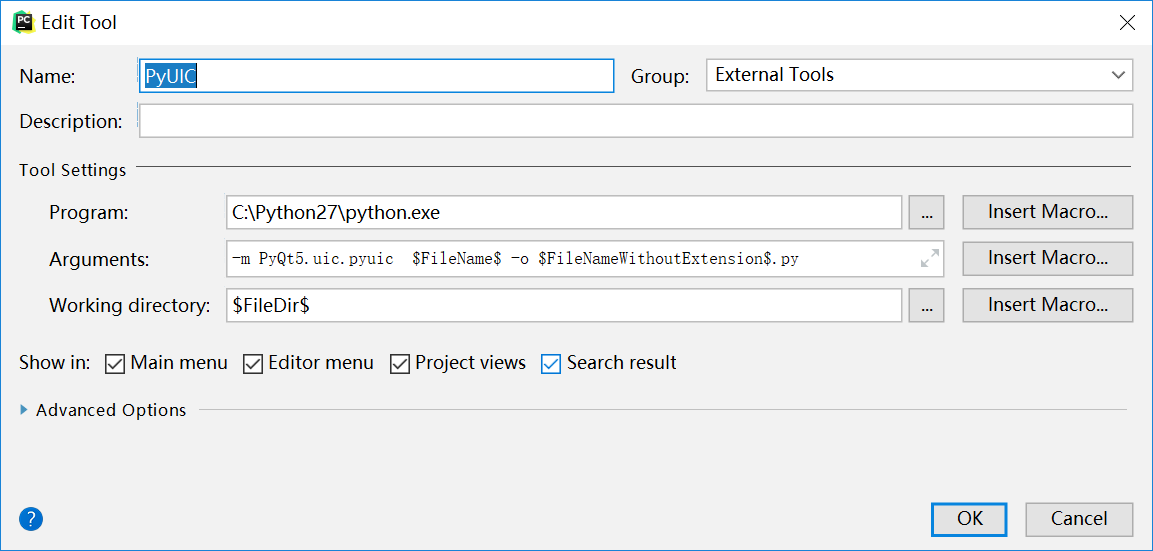


Fig 1.2.1 PyUIC的设置

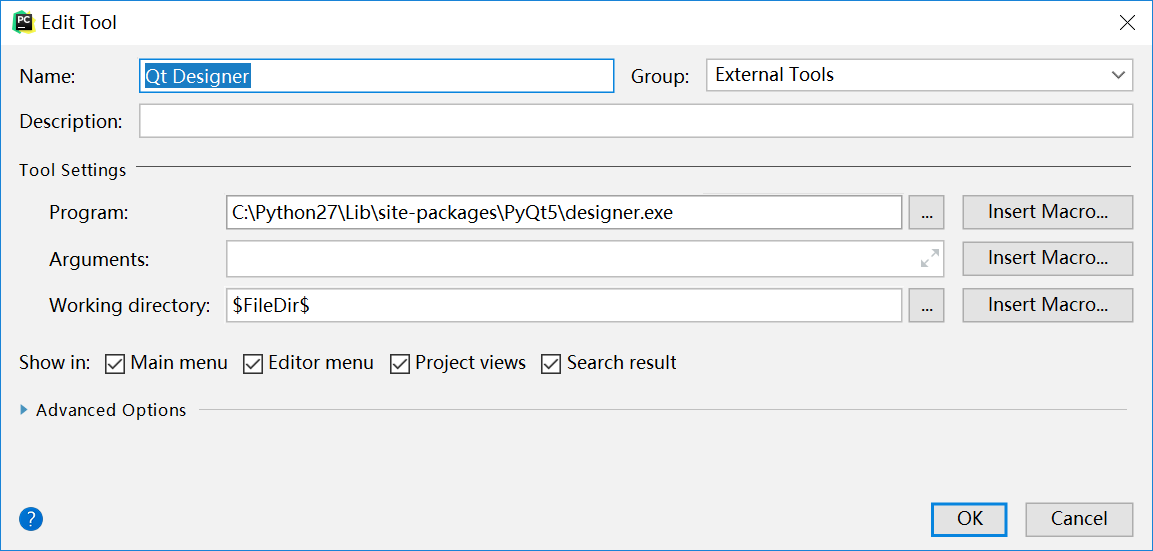


Fig 1.2.2 Qt Designer的设置

**1.3 使用Qt Designer设计界面**

安装完成PyQt5后，可以通过PyQt5自带的Qt Designer来设计用户界面。Qt Designer可以通过Python2.7文件夹下\Lib\site-packages\PyQt5\designer.exe打开。因为我们使用的IDE是PyCharm，比较方便的方法是将该designer.exe文件添加为PyCharm的外部工具（External Tools）。

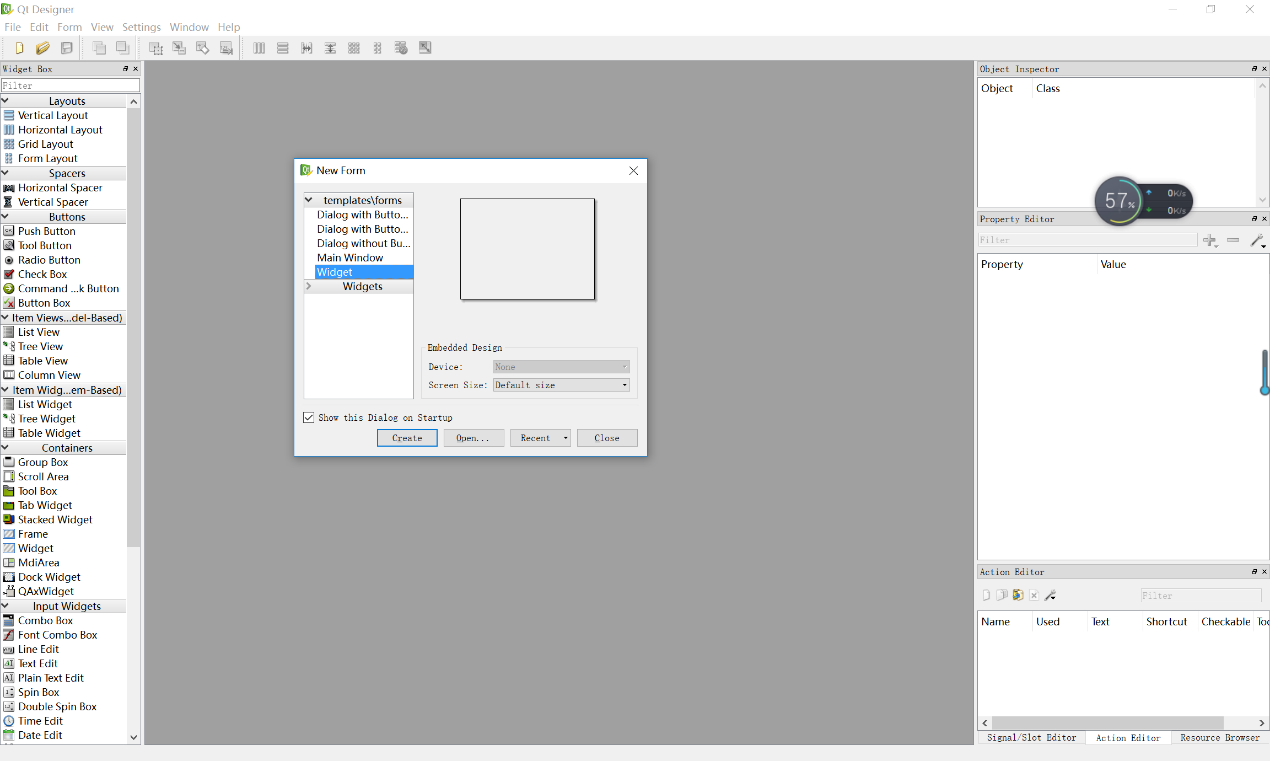


Fig1.3.1 Qt Designer打开界面

Qt Designer需要新建一个界面（MainWindow或Widget），即可在界面上进行拖放左侧如Buttons、Item Views等各种所需要的控件，并对其进行基础设置。由此可以设计出所需要的界面。所设置的界面将会保存为.ui格式。

在本次大作业中使用到的控件以及相应设置方法、控件和程序逻辑的连接方面的内容会在后面详细描述。

**1.4 利用Pyuic将.ui文件转换为.py文件**

PyQt5自带有批处理工具Pyuic.bat，主要功能为将Qt Designer设计好的.ui文件转换为可以通过Python进行操控的.py文件。右键选中.ui文件然后选择External Tools中的PyUIC，就可将.ui文件转化为.py文件

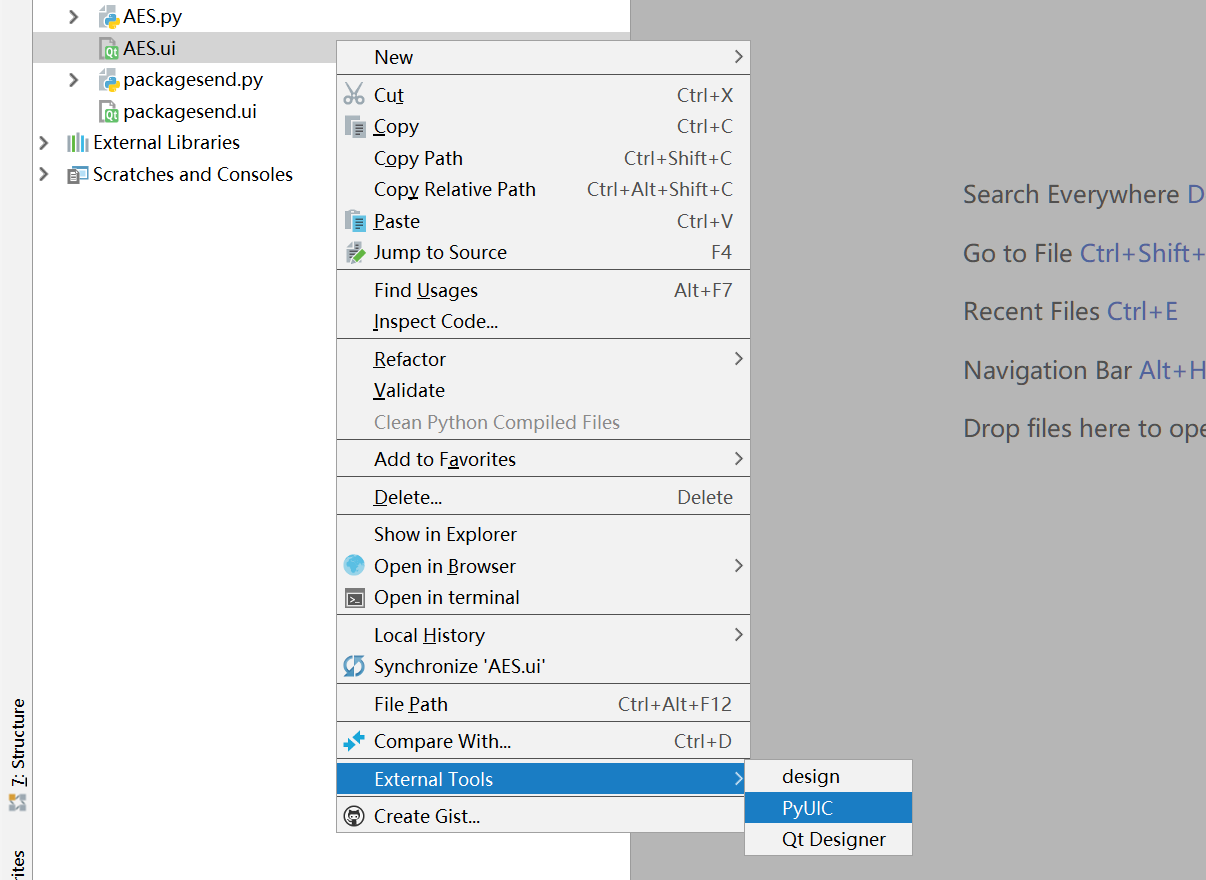


Fig1.4.1 ui文件转化为py文件

可在ui文件同一目录下生成同名的py文件。

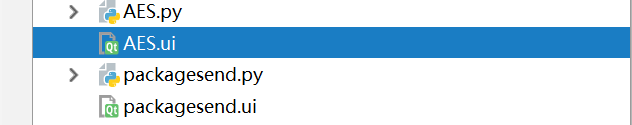


Fig1.4.2 ui文件和同名py文件

**1.5 使用的PyQt控件及其设置方法**

下面是设计过程中TCP发包界面的ui文件图，该界面涵盖了所有使用的控件。我将以此为基础对其加以介绍。

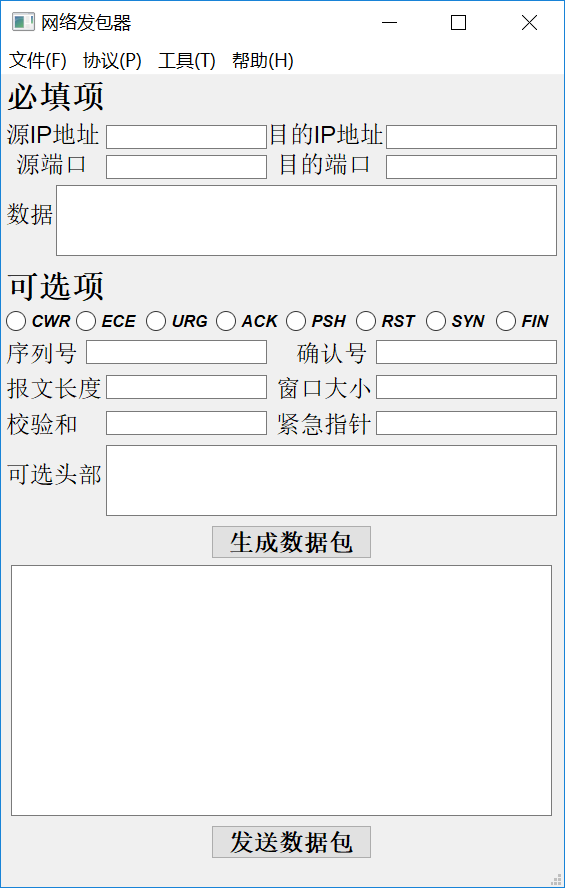


Fig1.5.1 TCP发包界面ui文件图

**1.5.1 QMainWindow**

主界面，最初新建MainWindow可得，Fig1.4.1图片中所有的控件均在此上面绘制，其与其他控件的从属关系见下图。



Fig1.5.2 QMainWindow与其他控件的从属关系

主要设置为在Qt Designer右侧属性中设置其objectName。其大小可直接通过拖动改变，也可以在属性栏geometry中的Width和Height上设置。

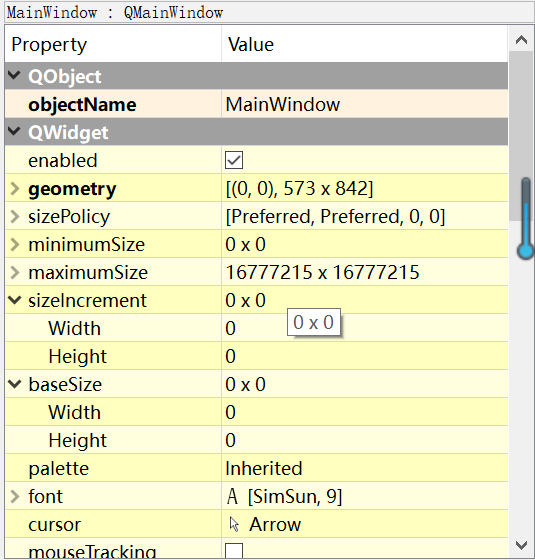


Fig1.5.3 QMainWindow属性栏

**1.5.2 QWidget**

界面。可以单独成界面，也可以放在其他QMainWindow和QWidget中作为容器界面。设置和QMainWindow基本相同。支持函数setVisible(bool)设置是否可见。大作业中主要用于设置除发包界面外的其他功能界面以及发包界面中不同协议的切换。

**1.5.3 QLabel**

标签。Ui界面中的所有介绍性的文字，如协议、可选项等，都由此设置。双击即可修改内容，其字体可在属性栏font一项中设置。支持函数setVisible(bool)设置是否可见。

**1.5.4 QLineEdit**

最简单的文本输入框。输入文本居上下方向的中间，不自动换行，不支持滚轮操作。长输入需要使用方向键往前查找。支持函数setText(str)修改显示内容为str，text()获取输入框中的内容。Ui界面中短输入框均为QLineEdit。

**1.5.5 QTextEdit**

文本输入框。初始光标在左上角，自动换行，输入高度超过输入框高时出现上下滚轮，适合输入和显示较长的内容。支持函数setText(str)修改显示内容为str，toPlainText()获取输入框内容。Ui界面中长输入框均为QTextEdit。

**1.5.6 QRadioButton**

即Fig1.4.1中CWR、ECE等上面的可选小圆圈。默认autoExclusive为True，即同一界面下的QRadioButton互斥，不能同时勾选。需要在右侧属性栏中将autoExclusive一项中的对勾去掉。支持函数isChecked()判断是否被选中。

**1.5.7 QPushButton**

Ui界面中的方框形按钮。基本设置同上，双击可修改按钮文字，右侧属性栏可修改objectName、font等基本属性。按钮被按下后，触发triggered，可通过函数triggered.connect(function)将触发后调用的函数更改为function，需要注意的是该function不能有输入参数。

**1.5.8 QMenuBar**

菜单栏。可以直接在Qt Designer中输入菜单栏各项内容。需要注意的是菜单栏中的内容为QMenu类型，没有找到直接点击该类型时连接到其他函数的方法。

**1.5.9 QMenu**

菜单栏中的内容，Ui界面中的文件、帮助、协议等。不能直接触发其他函数。其下面的内容为QAction类型。

**1.5.10 QAction**

菜单栏二级子选项的类型，如加密工具下的AES加密、RSA密钥等。可通过函数triggered.connect(function)将触发后调用的动作更改为function函数执行的操作。大作业中主要用于弹出新界面。

**1.5.11 QTextBrowser**

文本浏览控件，用在了帮助界面中，在Qt Designer中即可方便更改，支持HTML格式从而可以呈现更丰富的文本内容。支持只读设置，可滑动查看，不同帮助页面通过setVisible（）来实现切换。

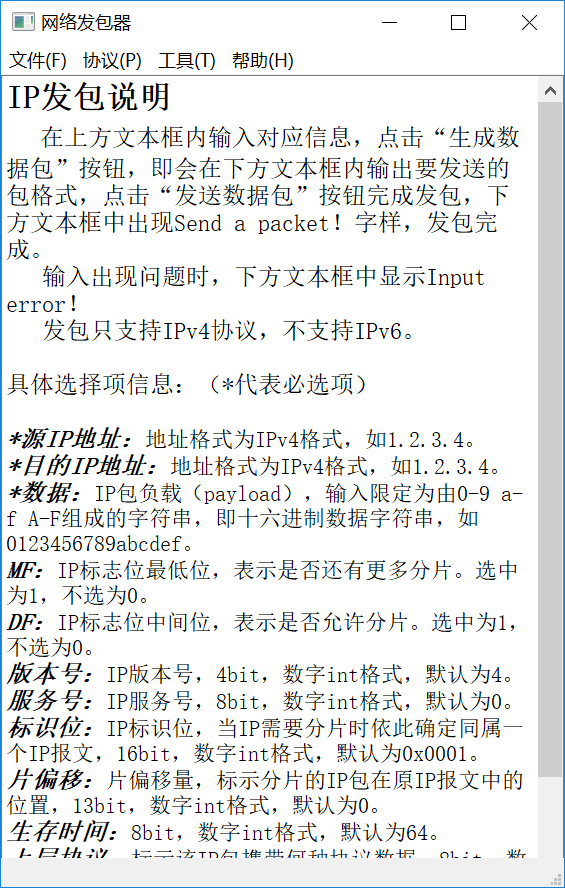


Fig1.5.4帮助界面中的QTextBrowser

**1.5.12 QFileDialog**

文件选择对话框，可以通过它选取本地文件，使用方法在后面部分会描述。

1.6 **py文件下界面展示和界面转换**

将ui文件转换为py文件后，生成的py文件中是一个类，类中的setupUI函数包括了所有在Qt Designer中进行的设计操作的Python代码，（除了from scapy.all import \*

import SockIcmp,SockArp,SockUdp,SockTcp,SockIp,SockPcap为后来添加)如下所示：

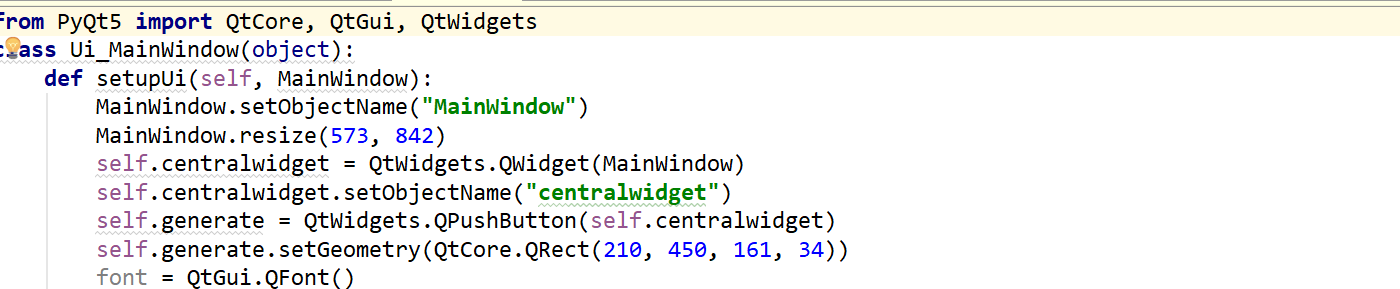


Fig 1.6.1 生成的py文件的类

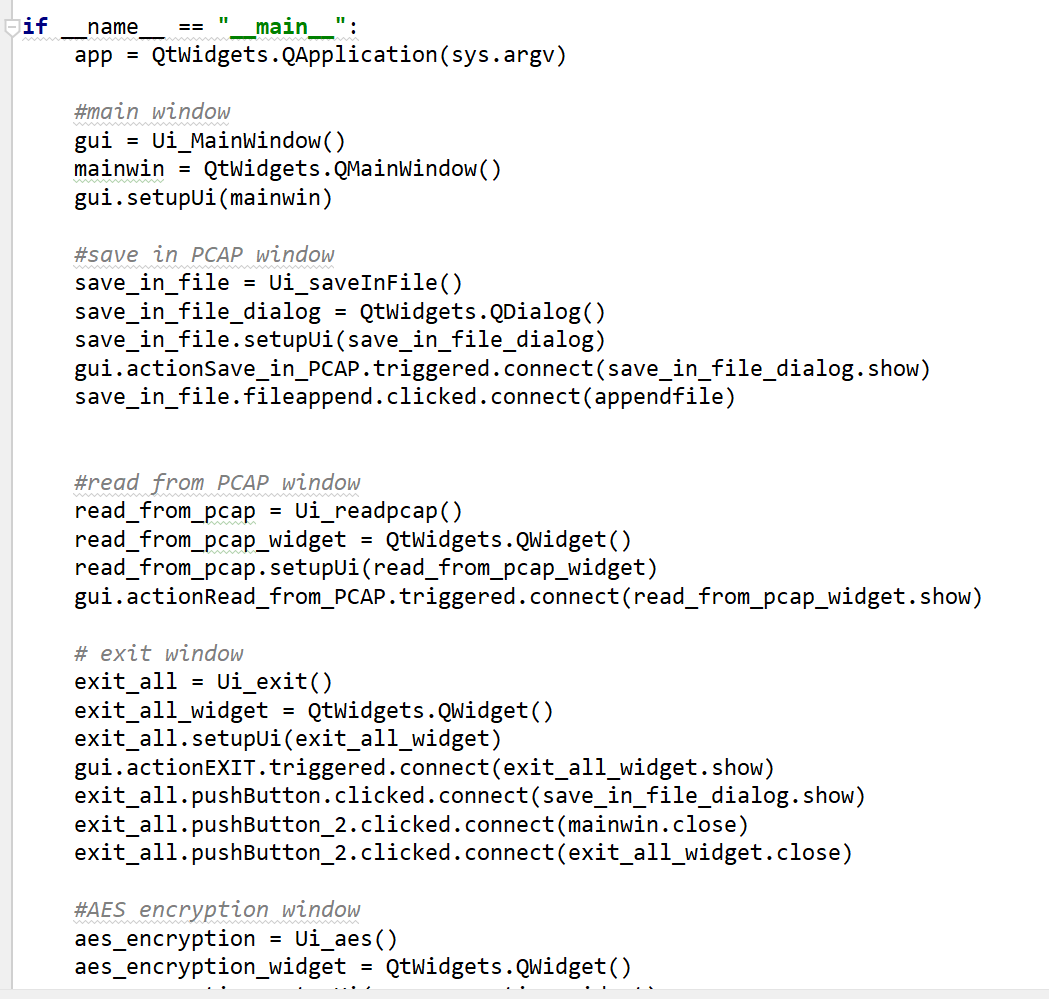
在该文件中要产生相应的界面，需要添加如下语句：



Fig 1.6.2 py文件产生界面的代码

第一句QApplication一句新建一个进程，然后初始化界面的类Ui\_MainWindow，同时通过QMainWindow生成一个主界面。setupUI的过程既是初始化界面类的过程，同时也将界面的各项操作绘制在了生成的主页面MainWindow上，利用show()函数即可显示出来，并在界面上进行各项操作。当点击右上角的关闭按钮时，运行sys.exit释放进程。

本次大作业中其他界面可以通过主界面弹出，其方法如下：

Fig 1.6.3 界面转换（节选）

基本思路为：首先将所有界面都绘制在其所设置的QMainWindow或者QWidget上，然后将各自的show函数连接到主界面（上图中主界面命名为gui）菜单栏二级子选项的QAction上，这样点击菜单栏选择项时就会触发show函数，从而将新界面显示出来。

**1.7 随下拉菜单更改变换显示内容**

本次大作业中实现了下图所示的功能：



Fig1.7.1 IP协议界面情况

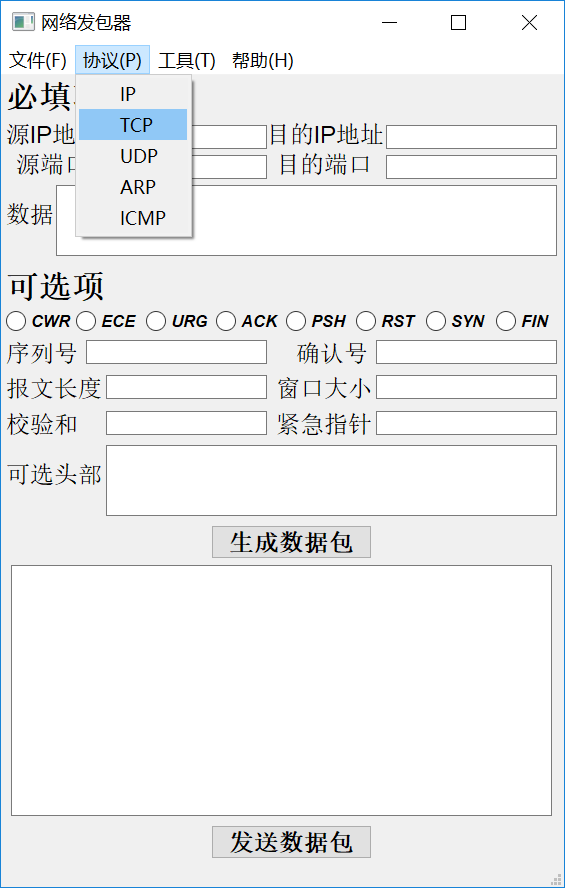


Fig1.7.2 TCP协议界面情况

当菜单栏中协议一栏的二级菜单所选项发生改变时，在同一界面上会显示不同的内容，主要通过QWidget的setVisible(bool)来实现，具体代码如下：

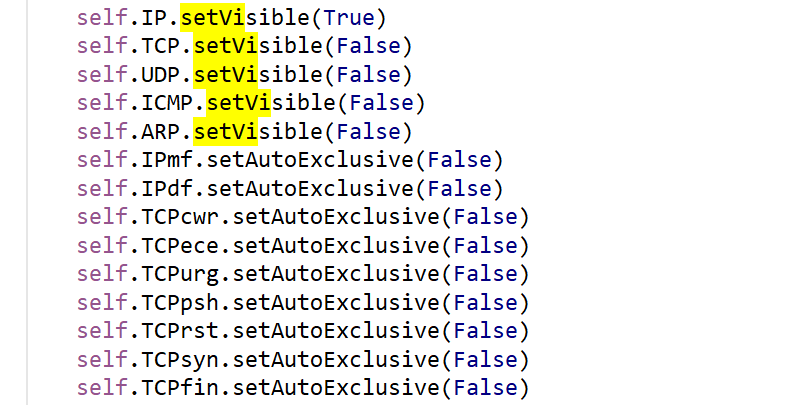


Fig1.7.3 界面初始化，只显示IP Widget

当每一个菜单二级选项的action被触发后对应的内容就会被显示出来。

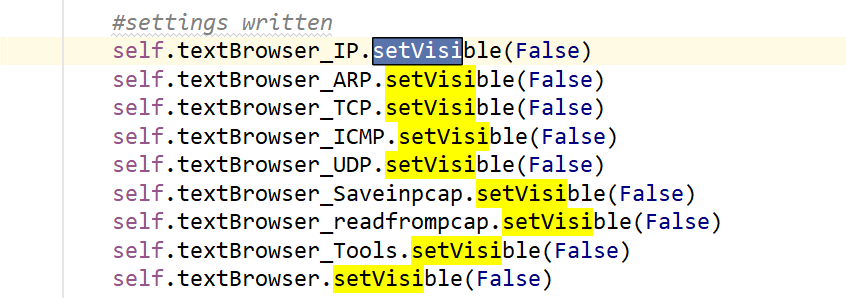


Fig1.7.4 下拉菜单协议的二级菜单选项改变触发不同协议的显示

因为帮助窗口的内容也是在主页面显示，所以在把某一个特定的协议置为visible的同时，需要把其他协议的内容和所有帮助窗口的内容置为invisible。以IP\_change为例，只把self.IP.setVisible改为True,其余均为False,另外将Index的值置为0，表明当前为IP协议模式，方便后面报文的产生。Index从0到4分别表示:IP, TCP, UDP, ARP, ICMP

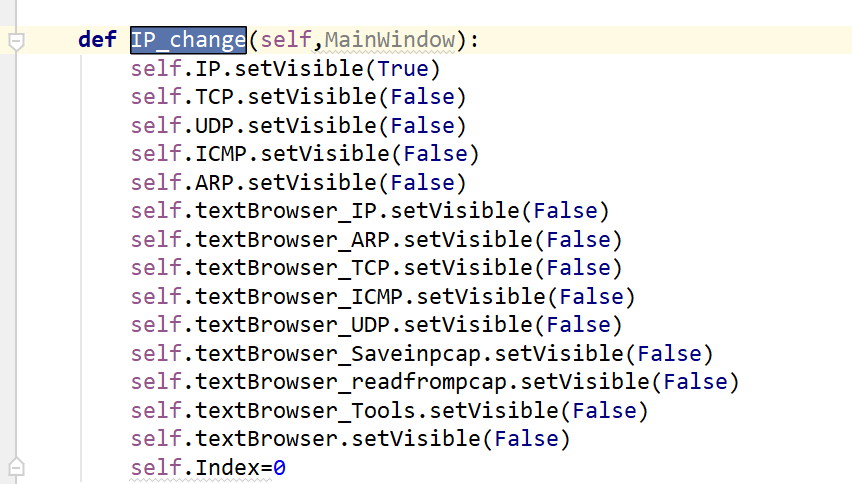


Fig1.7.5 IP\_change函数，根据当前index记录所选协议的情况

**1.8 主界面生成数据包按钮控制**

当填好生成数据包的各项要求后，点击”生成数据包“按钮，会在主界面下方输出该数据包的格式，该功能的实现如下：



代码1.8.1 生成数据包触发连接



Fig 1.8.2 根据不同协议生成数据包

下面以IP包为例，其生成函数如下：

Fig1.8.3 IP包构建

在Fig1.7.3中通过调用逻辑函数SockIp.sockIP来构建IP包，实现了界面和逻辑的结合。同时可以看出根据不同的可选项输入情况会修改IP包的各项信息。

产生的Packet作为了该类的一个子属性，这样写的优势是在写函数时不需要考虑函数返回值的问题，同时又很方便地和发包函数等其他函数相结合。

但同时由于调用了程序逻辑函数，一旦输入的情况有误或者不符合要求，就会导致程序逻辑函数报错从而整个界面闪退，无法继续。因此在Fig1.7.3中可以看到使用了Python的异常处理机制try…except…else。正常情况函数运行在try结构体内，报错则转向except部分，try运行完若无错误则运行else部分。

在Fig1.7.3中使用了setText(str)函数来通过QTextEdit进行显示IP包的情况，其中packet.decode部分是数据包的结构，hexdump部分是数据包的16进制情况。

同时实现了当生成包时将运算正确的校验和显示在Checksum位置上。正确校验和的输出是下面几句代码：



Fig1.8.4 正确校验和(Checksum)的输出

之所以这样处理是因为直接使用hex函数的话输出的结果与一般数据包的输出结果有微小的差异，一个相当于大数端，另一个相当于小数端，这一部分是对其进行了一次转换，使得下方文本框中的数据包的校验和和显示在Checksum部分的校验和相同。

**1.9 主界面发送数据报按钮实现发包**

这一部分主要应用了程序逻辑函数，方法如下：

Fig1.9.1发送数据包按钮连接

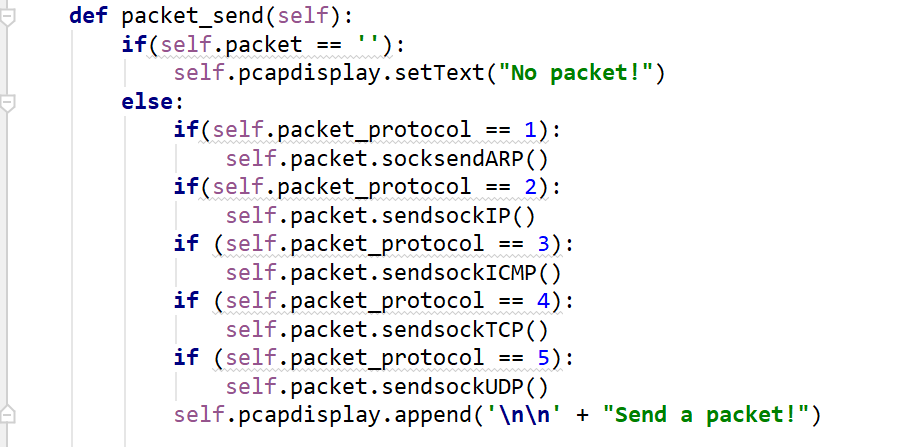
****

Fig1.9.2 发包函数

**1.10 PCAP保存数据包**

首先是Save in PCAP界面可以选取本地文件，其效果如下图：

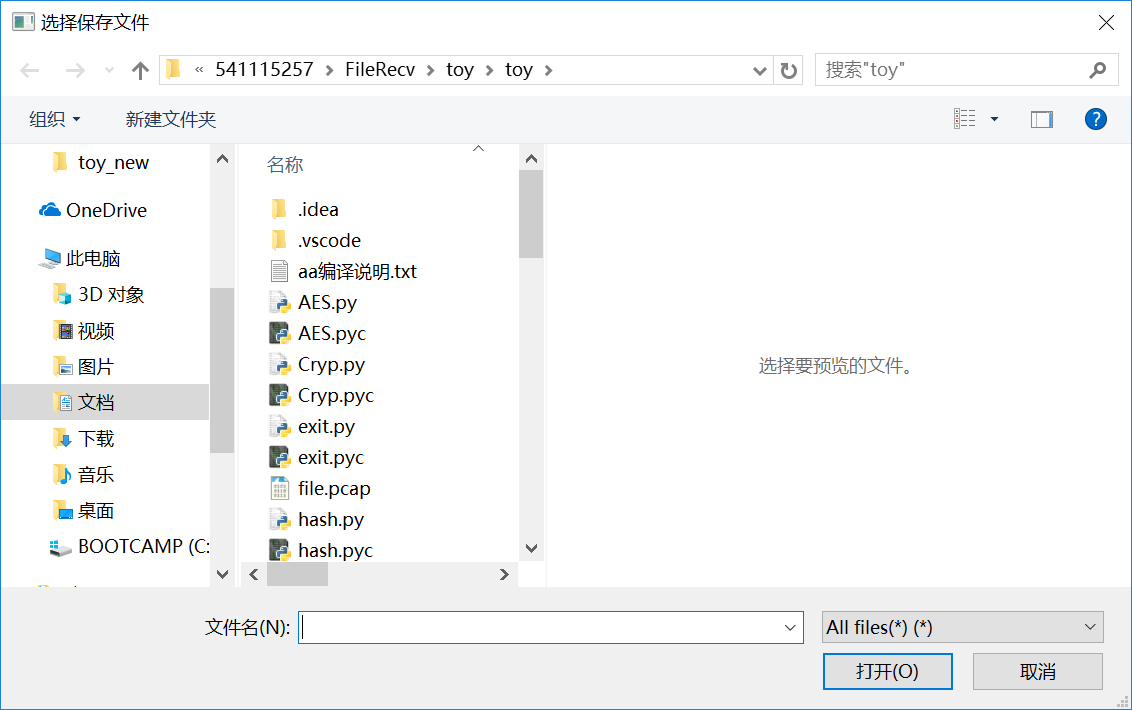


Fig1.10.1 PCAP保存选取文件

该功能的实现依赖于QFileDialog这一PyQt5自带的文件选择对话框，其实现代码如下：



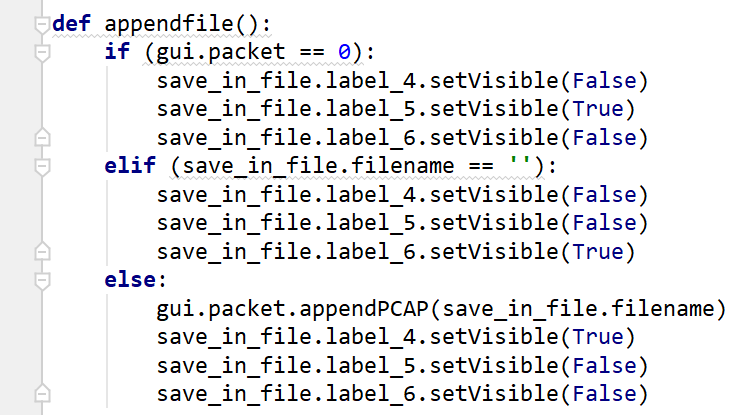
代码1.10.2 打开选取文件对话框

该函数第一项是新的对话框建构在哪里，QFileDialog只能建构在QMainWindow和QWidget上，而Save in PCAP本身是QDialog，因此将该参数取None，可正常运行。后面的参数从前往后分别代表对话框title、对话框默认文件位置（空为程序文件当前位置）和支持文件类型的下拉菜单项。

因为不能直接将主界面的子属性packet传入到Save in PCAP界面，因此保存数据包的操作是在main.py中进行的，因为main.py可以对该文件中生成的类的属性进行操作。实现如下：



Fig1.10.3 保存函数连接



代码1.10.4 保存函数

保存函数中设置的label的可见情况是用来显示是否成功保存。

**1.11 PCAP读取操作**

由于PCAP读取操作中的文件读取和数据包发送的方法和之前介绍的方法相同，这里不再赘述，只着重介绍一下其中的过滤操作的实现。

由于函数连接的情况和之前一样，这里同样不再介绍。Filter按钮连接到的函数如下所示：

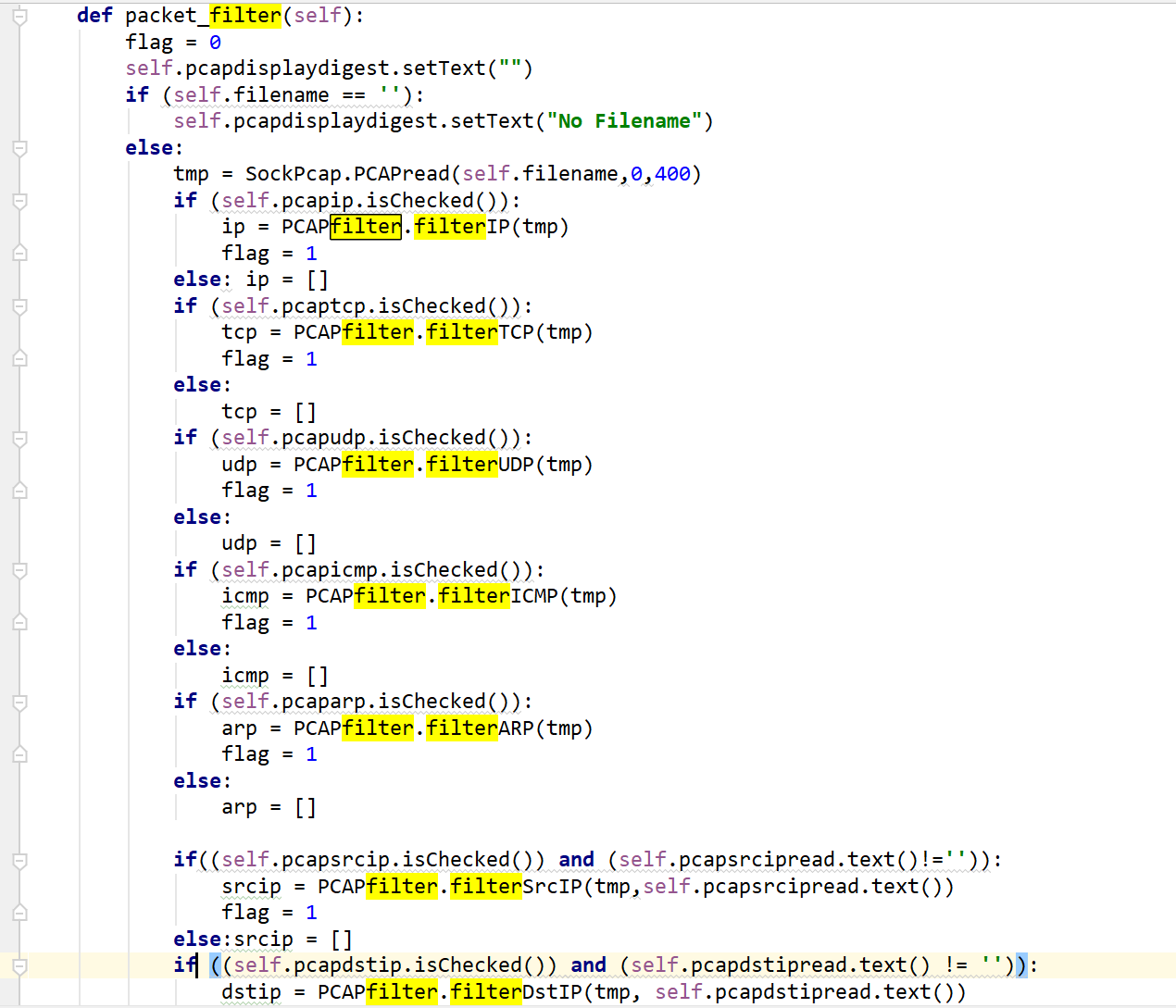
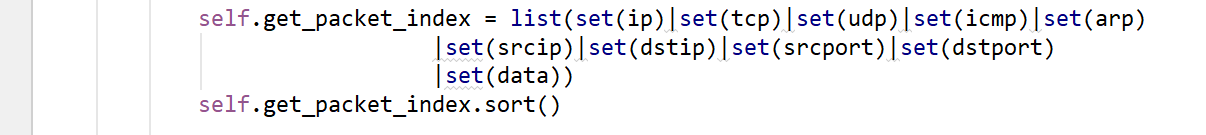


Fig1.11.1 过滤函数：协议过滤部分（节选）



Fig1.11.2 过滤函数：内容过滤部分（节选）

Fig1.11.3 过滤函数：最终过滤结果序号

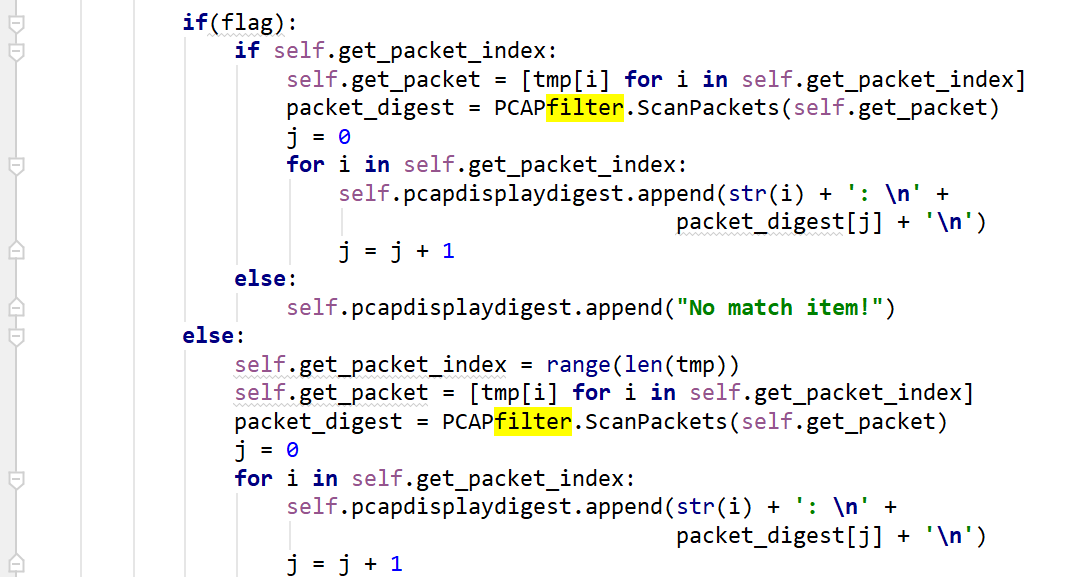


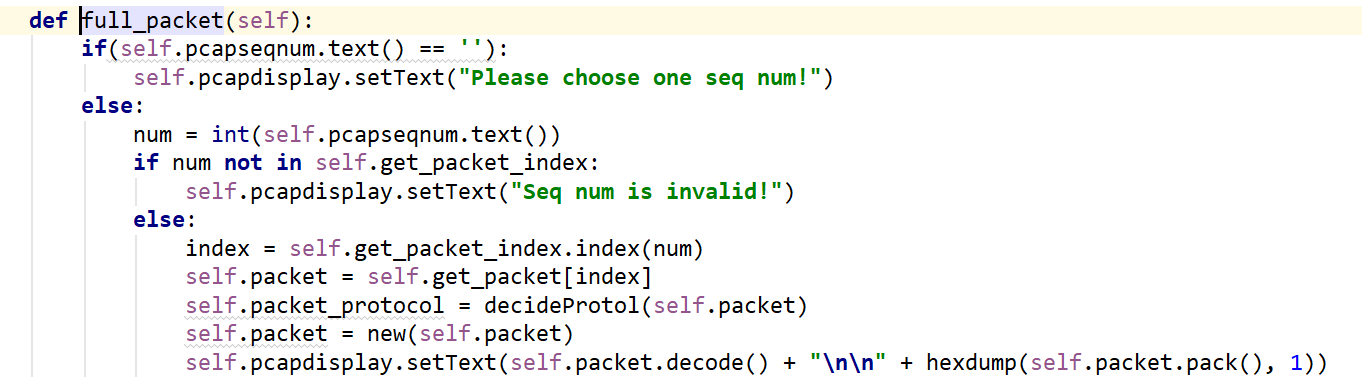
Fig1.11.4 过滤函数：最终过滤结果输出

Fig1.10.1和Fig1.10.2都是根据逻辑函数进行过滤的操作，每个过滤返回的结果为满足条件的数据包在PCAP文件的所有数据包中的位置，即其index。

Fig1.10.3中利用集合set，首先将所有过滤结果的list变为set取并集，得到所有要显示的数据包的index。同时由于set特有的性质，在最终的get\_packet\_index中没有相同元素。

Fig1.10.4是根据逻辑函数ScanPackets输出符合条件的数据包的摘要。Flag变量在这里的作用是判断是否选取了过滤项，如果根本没有选取过滤项，则输出文件中所有数据包的摘要。如果选取之后没有匹配结果，则输出No match item！

根据所选择的序号生成原数据包的实现方法如下：

Fig1.11.5 由序号生成原数据包并显示

**1.12 AES加密**

由于Crypto Tools中的工具，包括AES加密、Hash(SHA256)加密、RSA加密、RSA数字签名和16进制与字符串转换在界面与逻辑连接方面大致相同，这里只介绍AES加密的实现方法。

加密按钮实现如下：

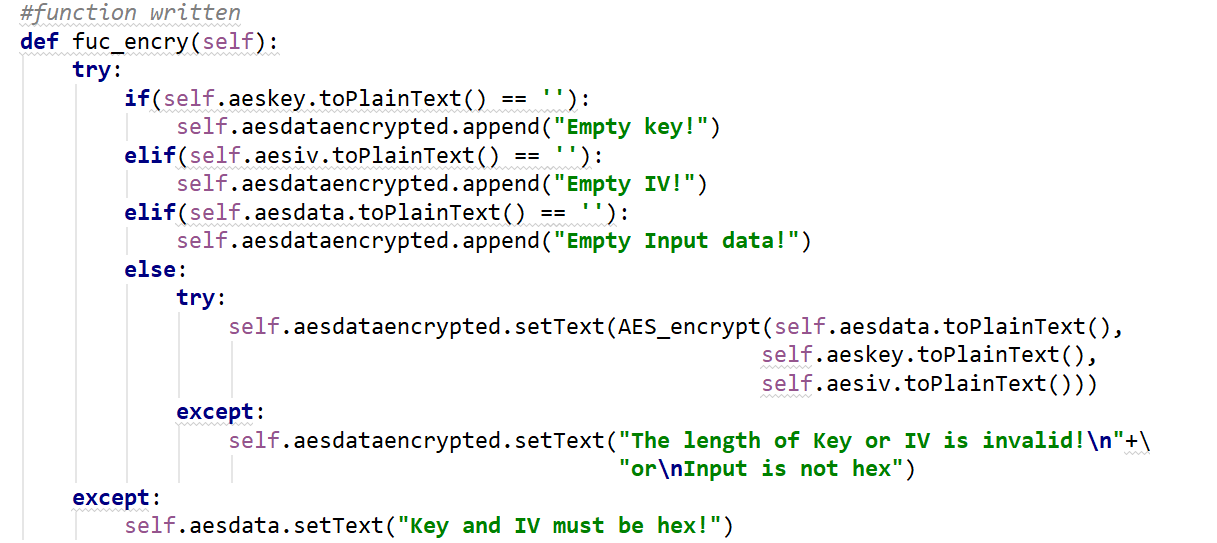


Fig1.12.1 AES加密

解密按钮实现如下：

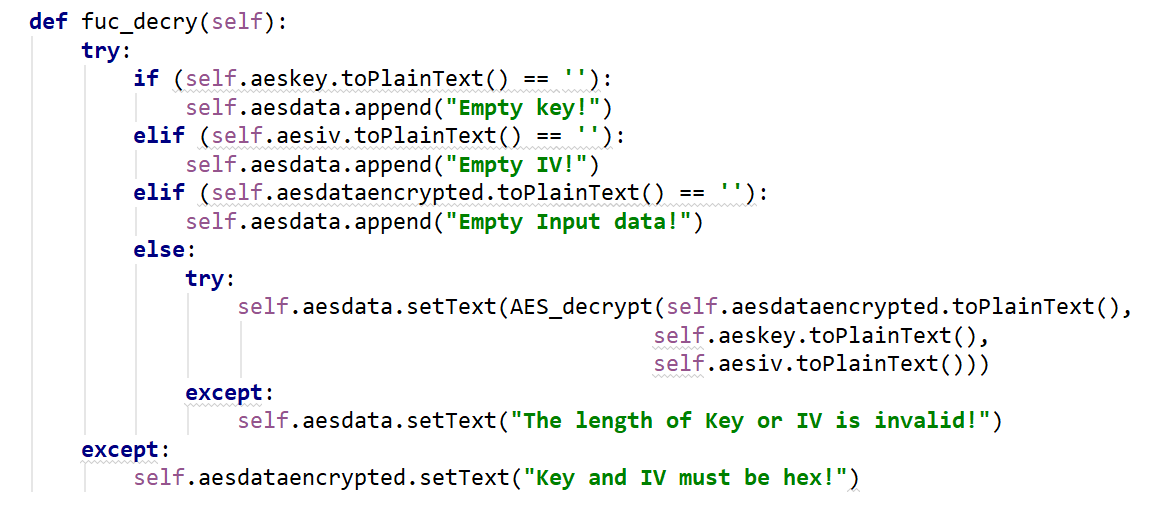


Fig1.12.2 AES解密

二者实现过程基本相同。

在处理过程中可以看到有很多判断条件是否符合要求的语句，是因为AES加密对于加密的要求比较严格。为了防止出现闪退的情况，使用了二层的异常处理机制。最外面一层是用来处理输入的Key和IV不是十六进制比特流形式的情况的，而里面一层处理的是Key和IV的长度不满足要求以及要加密的数据不是十六进制比特流的问题。

**1.13 RSA密钥**

点击按钮后调用函数如下：

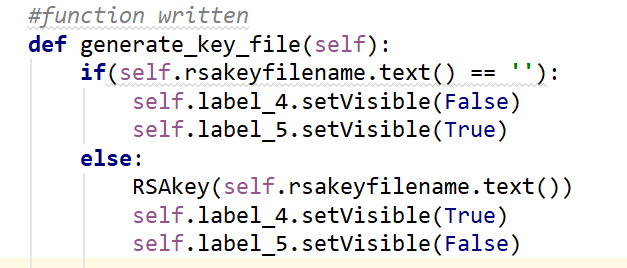


Fig1.13.1 生成RSA密钥文件

由此可见，主要是调用了程序逻辑函数。

**1.14 帮助界面**

Help是菜单栏界面主要利用了之前提到的根据下拉菜单项修改显示内容的方法和QTextBrowser丰富且不可更改的控件来完成功能。因为只是显示功能，所以并没有连接任何函数。因为帮助一栏的信息显示在主页面，所以默认设置全部为Invisible

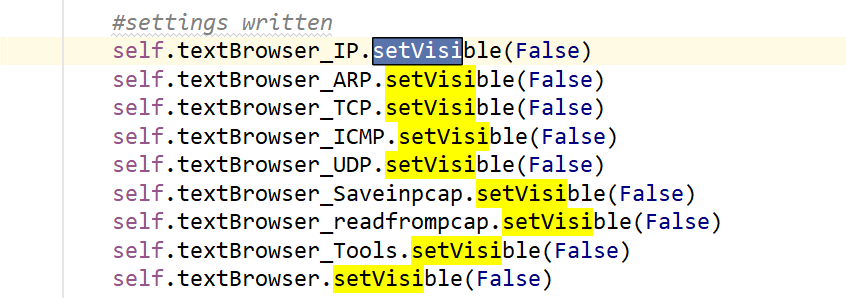


Fig 1.14.1 帮助页面的默认设置

不同的页面通过Action的触发来显示：

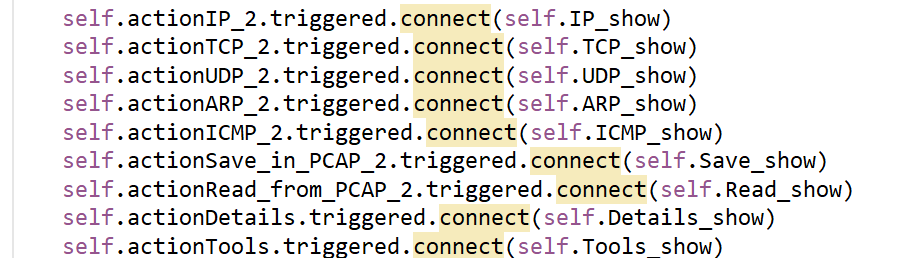


Fig1.14.2 帮助页面触发控制

以IP\_show函数举例，就是把IP相关内容解释所在的textBrowser设为可见，其余均设为不可见。

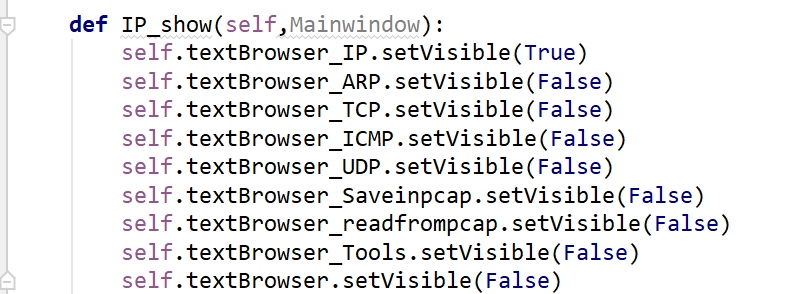


Fig1.14.3 IP\_show 函数

**二、发包后端技术细节**

**2.1 网络包构造与处理**

本数据包发包器中，我们自己实现了类对网络包进行了封装，而后使用scapy的send函数将已经构造完成的网络包直接发送。

对于五种报文，我们展示出类的\_\_init\_\_函数。五种数据包的类如下所示：

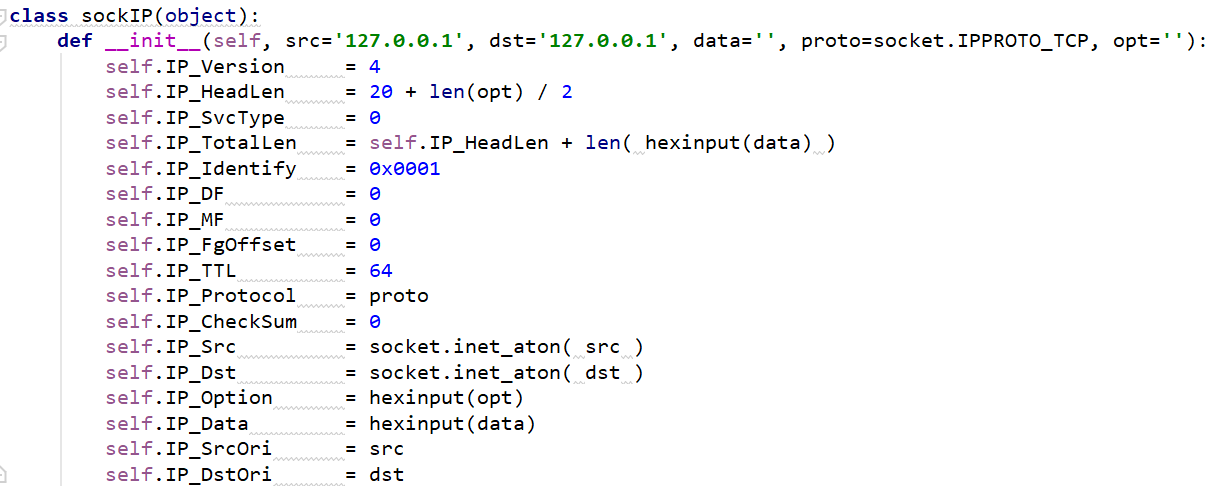


Fig2.1.1 IP数据包的构造类

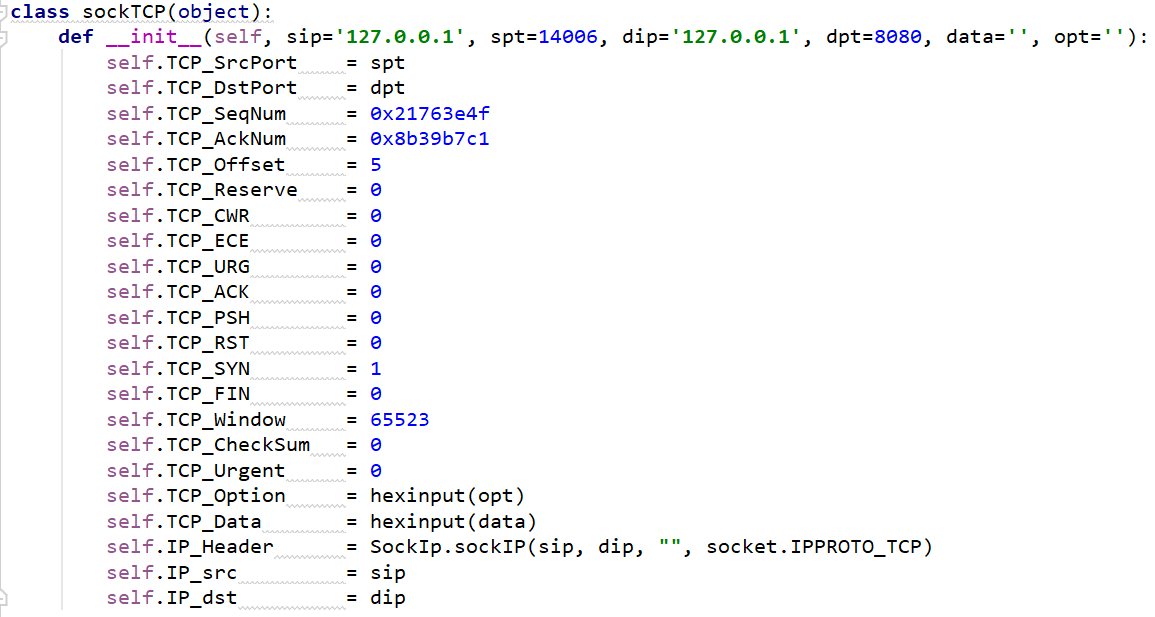


Fig2.1.2 TCP数据包的构造类

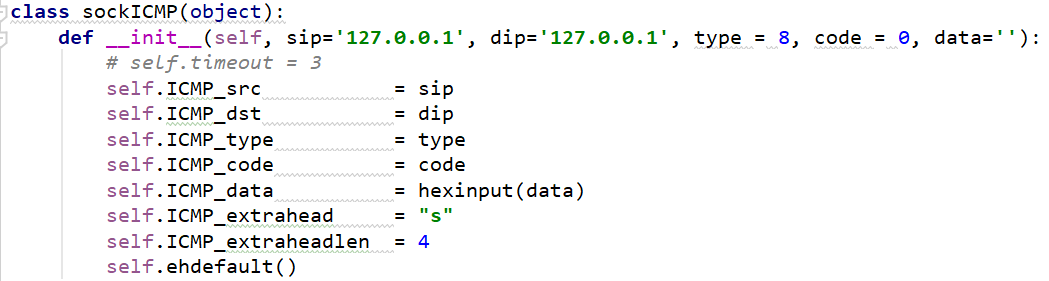


Fig2.1.3 ICMP数据包的构造类

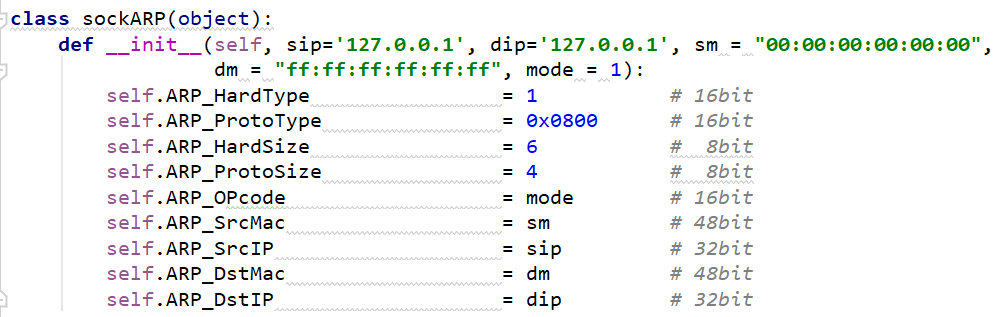


Fig2.1.4 ARP数据包的构造类

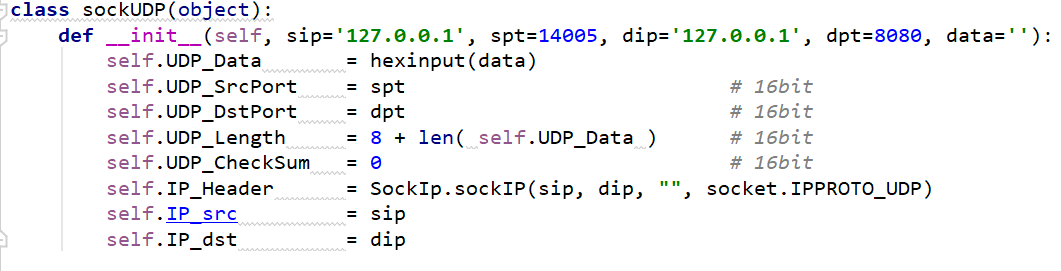


Fig2.1.5 UDP数据包的构造类

用户输入其试图构造的数据包的相关信息之后，点击“生成数据包”，后端即会调用对应的类，并生成一个类对象，这个类的属性和用户输入的信息一一对应。在获得这样的类之后，我们采用python的struct库，基于数据包的格式要求，将上述信息序列化为一个能发送的网络包。

我们以IP报文为例，详述一下这个过程。IP包序列化函数pack如下所示：

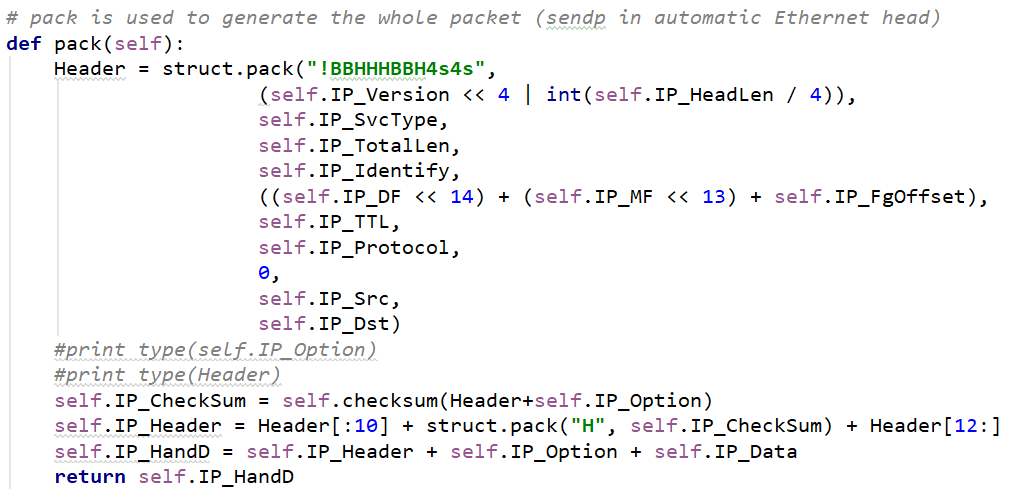


Fig2.1.6 IP数据包序列化

python的struct的pack函数，可以根据指定的格式产生数据包。例如：IP\_Protocol表示IP数据包的负载的协议，这个数在类中是一个整数型，而在数据报文中应当是用8bit来表示；这里IP\_Protocol对应的格式为“B”，这个字母表示8bit无符号数，那么最后正好能把整数变成一个8bit无符号数。通过这样的方式，数据包头部即可构造出来。

需要注意的部分是校验和：IP的校验和只要求IP头部（包括可选头），而TCP等的校验和不仅要求伪头，还要把数据部分和头部放在一起校验。在初始序列化的过程中，校验和会被设置为0，而之后再对生成完的序列进行校验和计算，再把校验和填到原来是0的那个位置。校验和的计算函数如下所示：

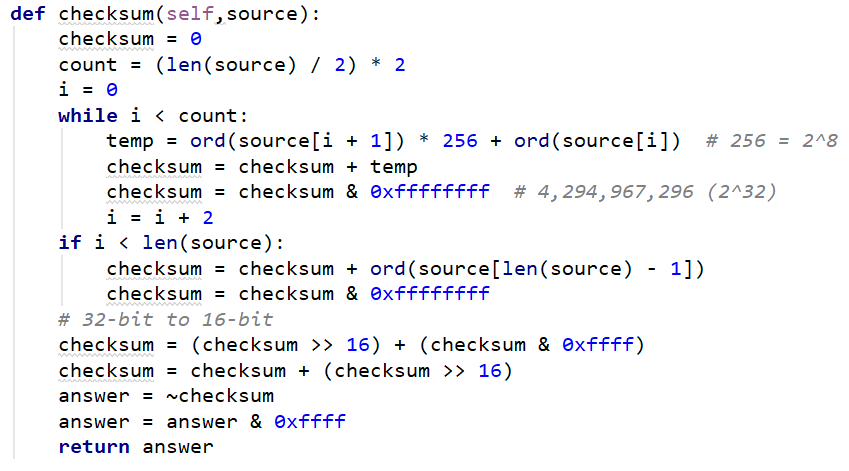


Fig2.1.7 IP数据包校验和产生函数

这里，我们把TCP的数据包序列化过程展示出来以稍加对比。可以看出，两者有明显的区别，但本质作用还是相同的：

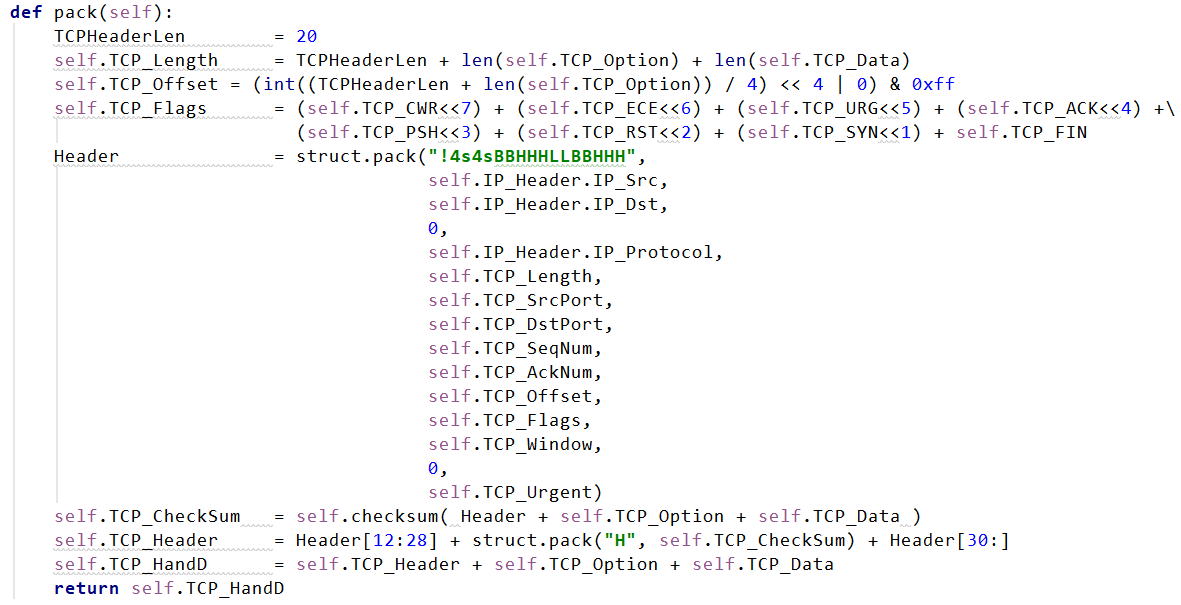


Fig2.1.7 TCP数据包序列化

另外需要做出说明的是数据包展示部分。我们试图展示与wireshark一样随光标移动动态对应的十六进制序列，但最终由于时间和能力的限制，只能尽可能达到效果。完成这项功能的函数为decode。我们用IP数据包来做展示，如下图所示：

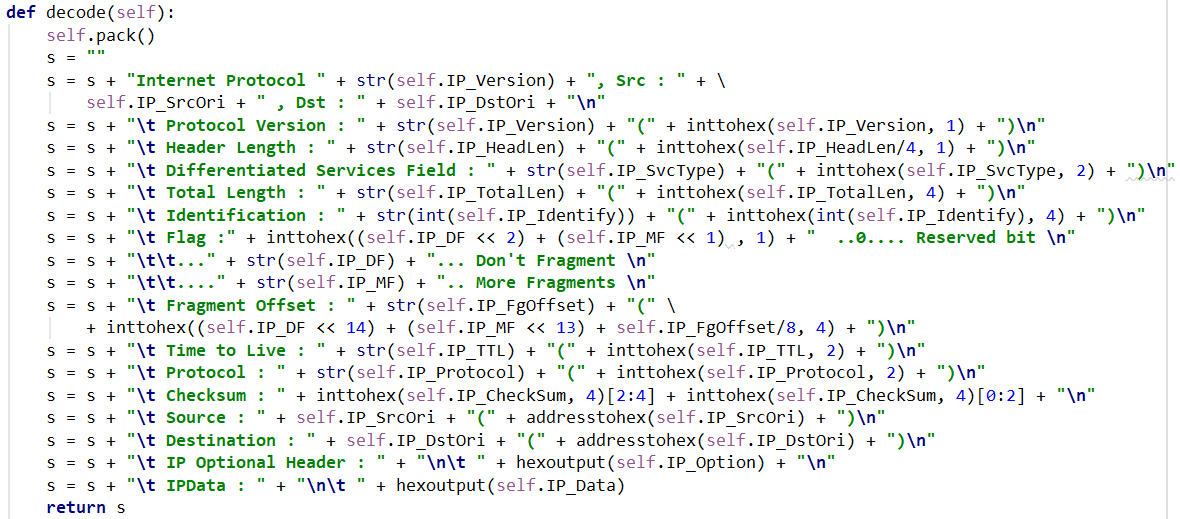


Fig2.1.8 IP数据包展示函数decode

上述函数运行结果会在用户点击“生成数据包”后显示在下方文本框中，具体生成结果见“效果展示”一章的内容。这里为了说明，简单地print出来仅作简短解说。



Fig2.1.9 IP数据包decode函数运行结果

这里的展示不考虑下层数据（对IP来说，序列展示从IP层开始，而不考虑Ethernet头）。这里，对于每一项数据，都列出了其意义，并且在冒号后面写出了其具体数值。在具体数值后面的括号内，则显示出了其在hex序列中对应的十六进制单元。在最下方，我们采用经典的hexdump格式输出这个数据包，提供了更直观的展示，尽可能包含了wireshark的主要展示功能。

最后，我们利用scapy提供的三层发包函数send构建了一个函数，能将数据包发送出去。此外，我们自己提供了将生成的数据包导入（本质上是添加）到某个pcap文件中的函数。这两个函数也集成在类之中，如下图所示：

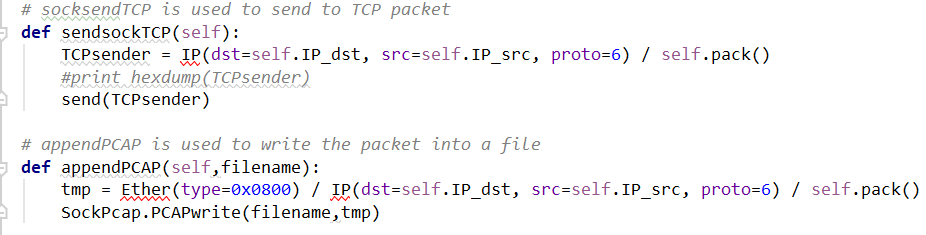


Fig2.1.10 TCP数据包发送函数sendsockTCP与写入PCAP函数appendPCAP

**2.2 PCAP读写支持**

PCAP读写主要采用scapy提供的读写功能。但其给予的只是较为底层的读写，我们在此基础上进行了诸扩展，最终实现了对于PCAP文件的良好支持。

首先，我们利用PcapWriter与PcapReader进行较为底层的读写。这里我们对于数据包进行了处理，最终形成了适用于本项目的读写函数。集成后，我们只需要输入数据包的list（每一个元素都是一个pack过的数据包，通过前面所述的类生成）和文件名，就可以在该文件中写入这些数据包。同样，给予一个文件名和起始锚点，就可以读出数据包的字节序列。具体实现如下：

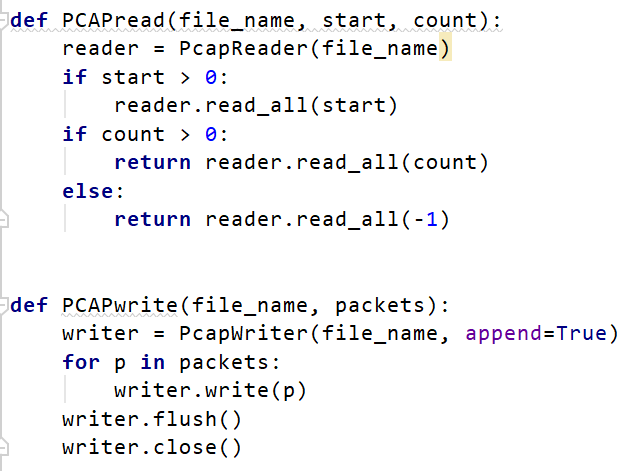


Fig2.2.1 PCAP读写函数（数据包字节流的读写）

相比写入数据包，读取数据包相对繁琐。PCAP文件本质上只帮助用户存储字节流，即：在写入PCAP文件之后，我们再次将数据包读出时，只会是字节流的形式，而并不存在之前我们所构造的类。为了满足可用性，我们必须将字节流进行解析。在这里，我们需要把所有字节信息转换成类元素，相当于实现了Sniffer的功能，工作量确实较大。其实现过程如下：

首先，先从字节流中读出协议类型（ICMP, TCP, UDP, 其他IP, ARP）：

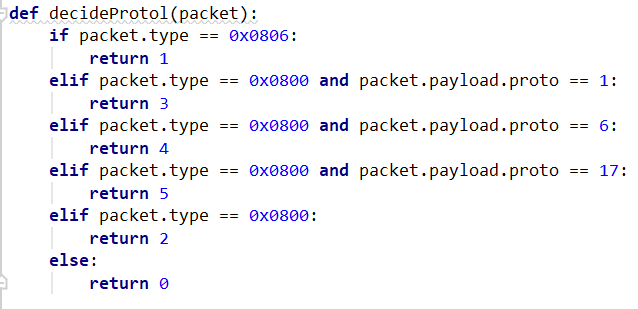


Fig2.2.2协议判断函数decideProtol

然后，我们就可以按照不同的协议类型，进行不同的类生成，这里我们给出TCP的代码示例：



Fig2.2.3 根据字节流生成sockTCP类的对象 new TCP

在生成自定义的对象sockTCP之后，对于这个数据包的处理就可以完全建立在我们之前的sockTCP类上了。但此外，我们还需要进行一些其他对于数据包列表的操作。例如，我们希望能够将所有数据包的摘要展示出来。这个功能可以通过ScanPackets完成：

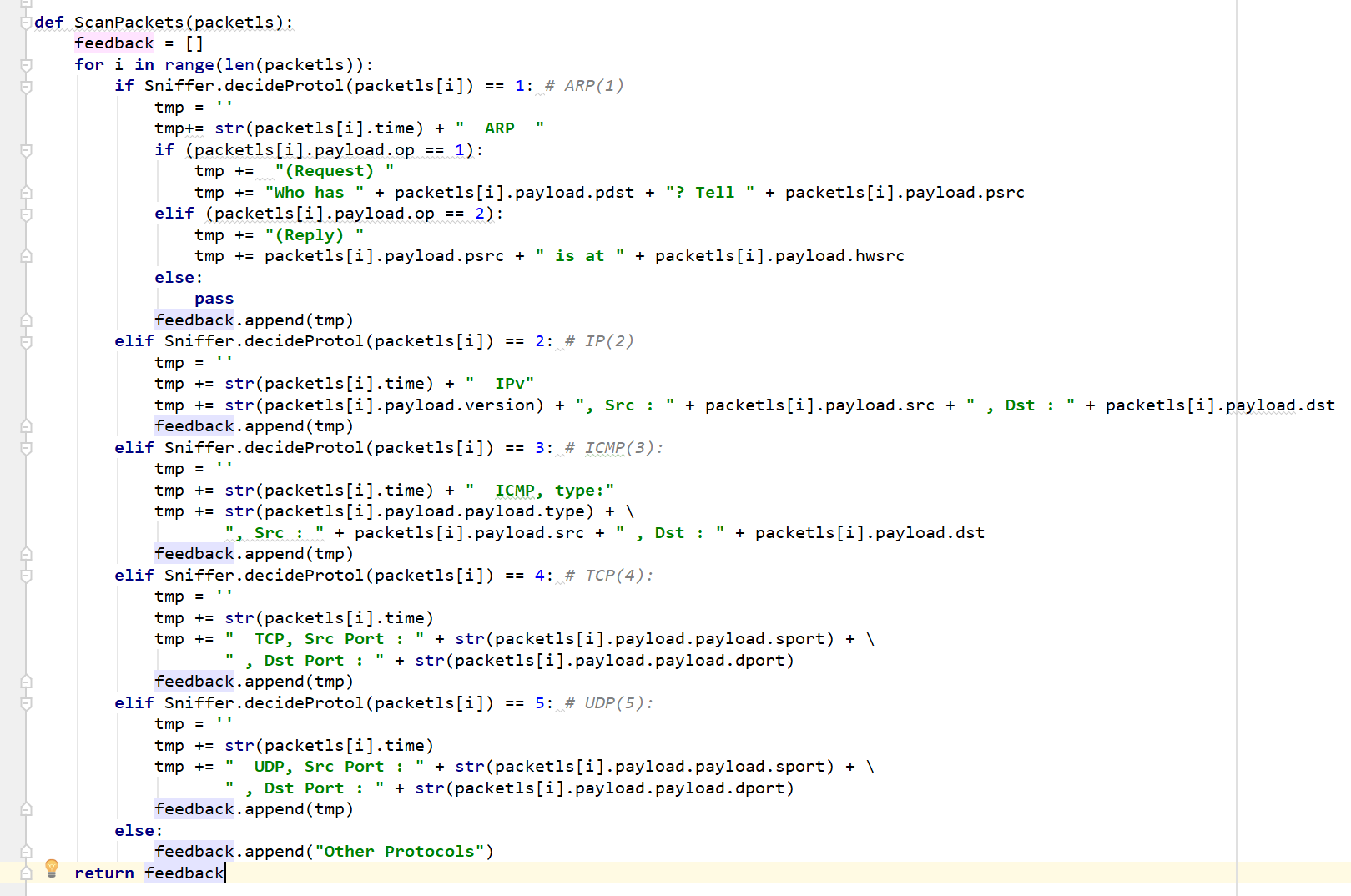


Fig2.2.4展示数据包序列的摘要：ScanPackets

这个函数得到的列表输出结果如下图所示。我们基本上将时间和数据包的内容大致展示给用户，方便用户快速寻找数据包。

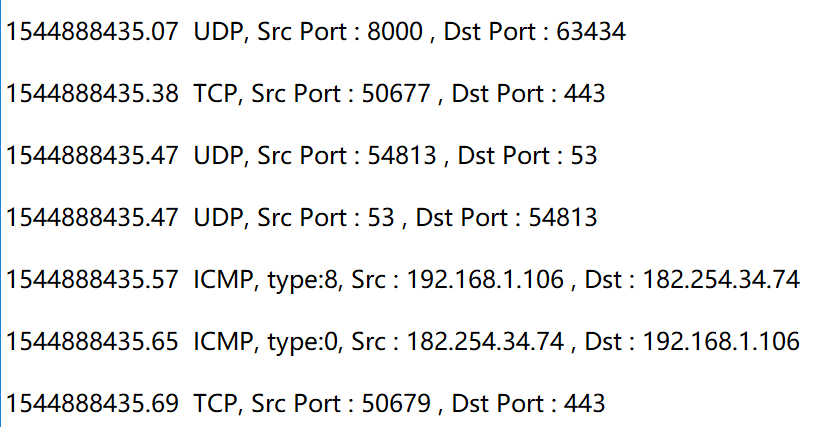


Fig2.2.5展示数据包序列的摘要ScanPackets运行结果实例

**2.3 数据包过滤**

对于PCAP中读取的数据包，我们往往并不需要全部，而是希望找出我们所需要的部分数据包。这里我们提供了完整的过滤功能，具体函数与功能见下表：

|  |  |
| --- | --- |
| 函数名 | 过滤效果 |
| filterDstIP(packetls,dstip) | 得到所有目标ip为dstip的包 |
| filterSrcIP(packetls,srcip) | 得到所有源ip为srcip的包 |
| filterSrcPort(packetls,srcport) | 得到所有源端口为srcport的包 |
| filterDstPort(packetls,dstport) | 得到所有目的端口为dstport的包 |
| filterARP(packetls) | 得到所有ARP包 |
| filterIP(packetls) | 得到所有IP包（除了ICMP,TCP,UDP外） |
| filterICMP(packetls) | 得到所有ICMP包 |
| filterTCP(packetls) | 得到所有TCP包 |
| filterUDP(packetls) | 得到所有UDP包 |
| filterData(packetls,target) | 得到所有数据（payload）中包含子串target的包 |

这里给出filterData的实现代码：

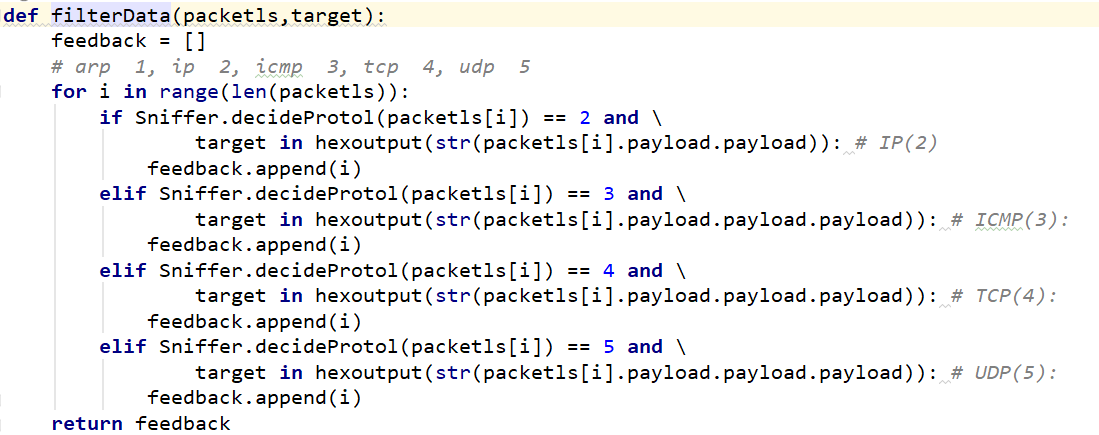


Fig2.3.1 filterData的实现代码

**2.4 密码学小工具**

我们的发包器不仅将大作业要求的基本功能全部实现（IP, UDP, TCP, ARP, ICMP的发送，以及PCAP支持，包过滤），还考虑到我们来自网络空间安全专业，因而增加了密码学工具箱，支持报文payload数据的各种密码学操作（RSA, AES, HASH, Digital Signature），从而可以为数据在网络上的传输提供另一层的保障。

在此之前先说明一点，即是通过我们的密码学工具出来的结果，一定是hex字符串格式，如“eeff78FFAB891F”。之所以是这样的输出，原因在于我们生成的包的data域支持的是hex字符串。这样，工具与发包器得到了极好的结合。

接下来我们说一下各密码体系的具体实现。我们主要使用了PyCrypto库。

1. Hash（这里采用SHA256）

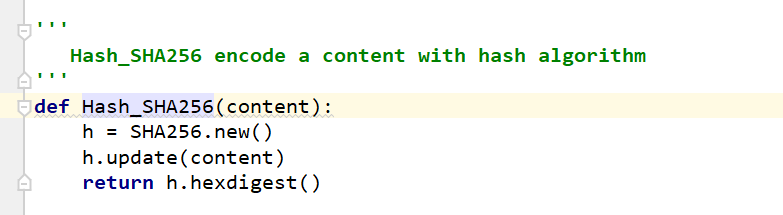


Fig 2.4.1 SHA256实现

2. AES（支持16,24,32byte的密钥，默认为CBC模式）

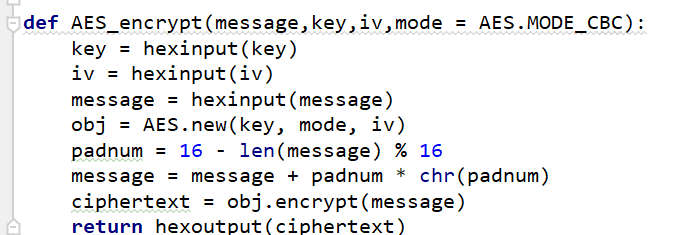


Fig 2.4.2 AES加密的实现

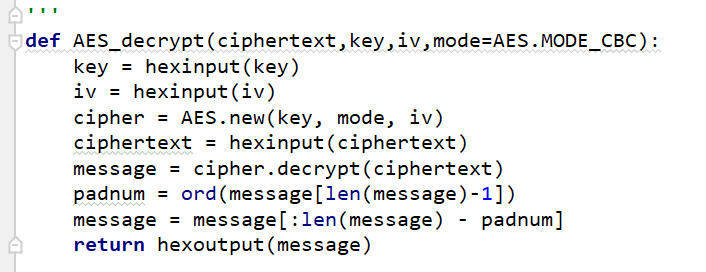


Fig 2.4.3 AES解密的实现

3. RSA密钥生成

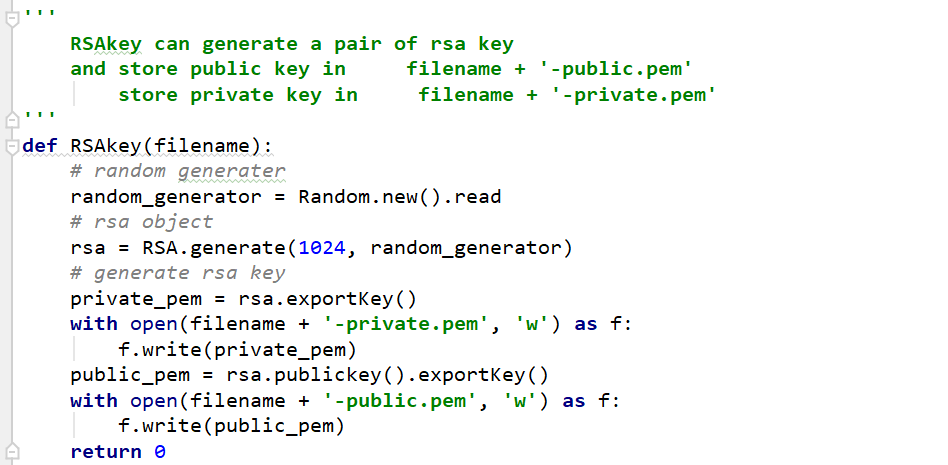


Fig2.4.4 RSA密钥生成实现代码

1. RSA加密（密钥为文件内导入）



Fig 2.4.4 RSA加密与解密实现代码

1. 基于RSA的数字签名与签名验证（密钥为文件内导入）

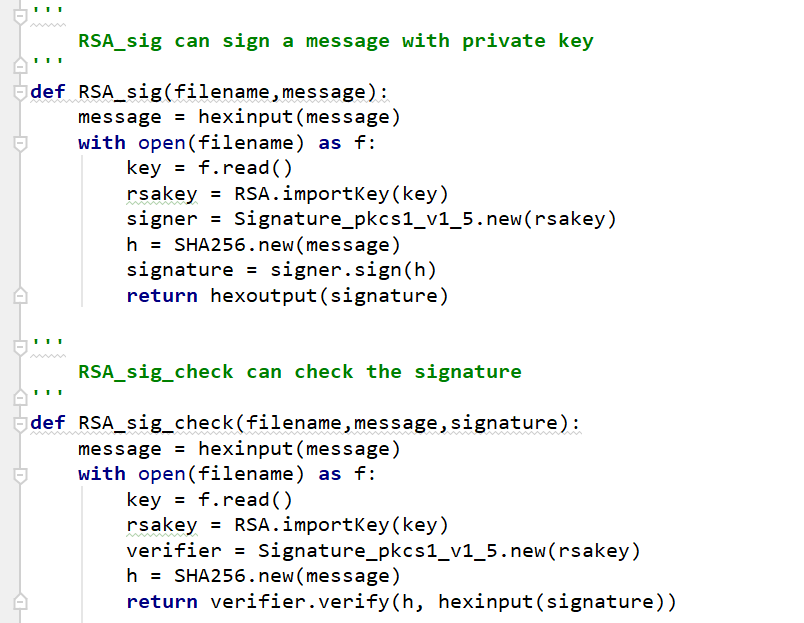


Fig2.4.5 基于RSA的数字签名与签名验证实现代码