江 蘇 大 學

JIANGSU UNIVERSITY

网络与系统安全课程设计

学院名称：计算机科学与通信工程学院

专业班级：信息安全 1501

学生姓名：沈鑫楠

学生学号：3150604028

教师姓名：李晓薇

完成日期：2018 年 7 月 3 日

目录

1.课程设计目的

本课程设计要求学生在已有理论学习的基础上，通过应用所学习到的相关知识，解决一些实际安全应用问题。从而，真正理解和掌握网络与系统安全相关理论知识。

2.需求分析

本课程设计是基于 OpenSSL 的编程，首先需要采用 openssl 生成证书 ca、client、server，其中 ca 为根证书。生成证书之后，需要采用 openssl 库进行编程实现简单的加密的点对点通信。实现简单通信之后，需要利用抓包软件捕获通信过程的相关数据，从而证明通信过程是安全的。

2.1 功能性需求分析

系统应分为客户端与服务端，服务端应先启用并开始监听是否有数据，客户端应后启用。一旦启用便能够随时地发送与接收数据，在发送和接收数据的过程中，数据应按照 openssl 证书进行 ssl 保密通信，防止第三方进行监听，进行通信的证书应该由根证书 ca 来签发，使用证书前应首先验证证书的有效性，防止证书被非可信人员篡改。在验证证书过后，就可以进行会话通信了，此时两方都应该随时监听对方是否有数据发送，一旦收到对方发来的数据，应该立即显示在屏幕上。

同时，两方也需要检测键盘是否有输入，一旦键盘有完整的一行输入，就应该立即向另一方发送数据。

2.2 软硬件以及其他需求分析

首先，由于该系统由服务端和客户端两个程序组成，需要在两台计算机上运行，因此，两台能够连入互联网的计算机是必须的。其次，由于本系统采用 Python2.7 进行编写，因此两台计算机应该装有 Python2.7 运行环境并安装好 openssl 的相关库以便进行运行。此外，在运行程序之前，要确保这两台计算机彼此能够相互访问，并分别留有两个空闲的、不被其他程序占用的端口来进行通信，其中一个端口用于发送数据，另一个端口用于接收数据。

2.3 其他非功能性需求分析

由于本系统需要两个计算机之间进行相互通信，因此应该保证两台计算机的配置足够以运行通信程序，防止卡顿和中断的问题。因为这个通信系统需要利用网络进行通信，两台计算机需要事先配置好网卡，并设置好防火墙的过滤规则，防止发送的数据被防火墙阻挡。

3.方案设计

3.1 概要设计

为了达到上述的功能性以及非功能性需求，需要利用 openssl 生成证书，然后利用证书进行通信。通信系统分为客户端和服务端两个部分，其中客户端中有发送和接收两个线程，服务端也有发送和接收两个线程，数据交换的过程如图 3-1 所示。

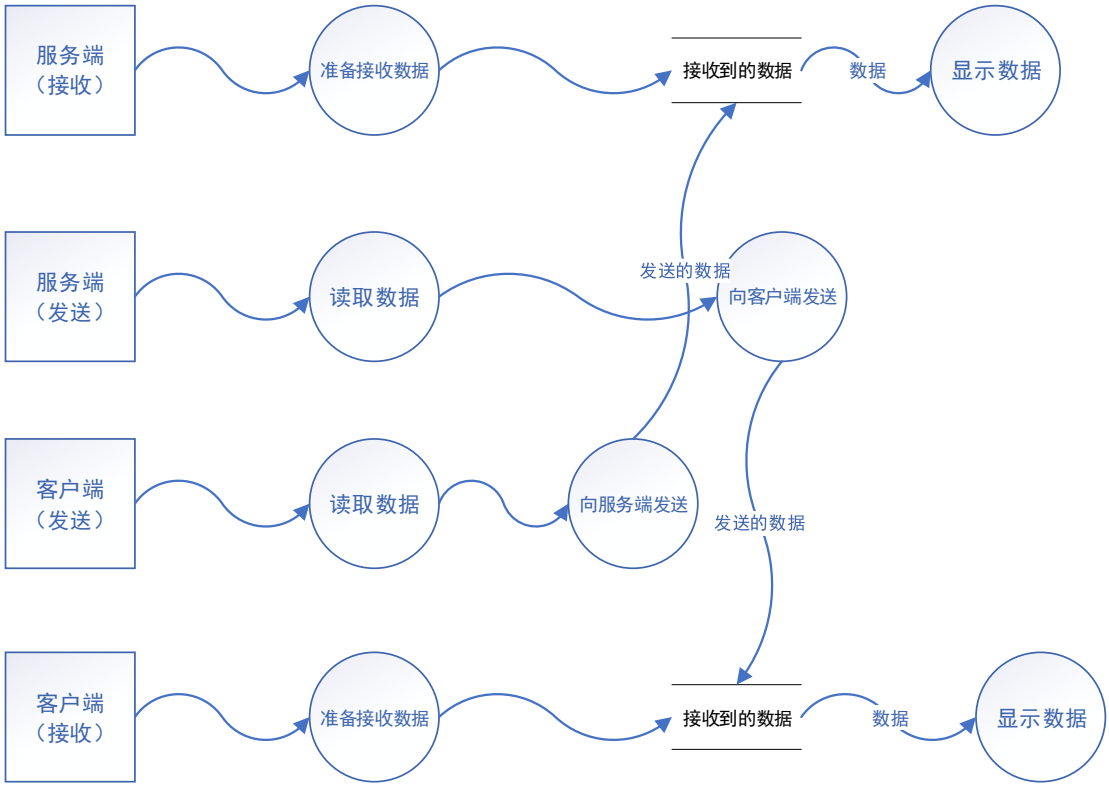

图 3-1 服务端与客户端通信的数据流图

3.2 详细设计

为了实现点对点的通信，应编写客户端和服务端两个程序。针对这两个程序，应该采用如下的策略：

（1）服务端：

服务端应该开辟两个线程，其中一个线程用于接收数据，一个线程用于发送数据，在开辟这两个线程之前，应该首先做一些初始化的工作，比如导入并验证证书，发送和接受的套接字的初始化。需要注意的是，应该首先初始化接受套接字，因为此时客户端还未启动，先初始化发送套接字会产生一些错误和问题，比如目标不可达或发送错误等。在初始化完成之后发送线程应该随时从键盘读取数据然后进行发送，接收线程应该随时读取接收缓冲区来判断是否接收到新的数据，并把新接收到的数据显示在屏幕上，像这样双方发送和接受才得以完成。

（2）客户端

同样地，客户端也应该开辟两个线程，其中一个线程用于接收数据，一个线程用于发送数据。在开辟这两个线程之前，应该首先做一些初始化的工作，比如导入并验证证书，发送和接受的套接字的初始化。与服务端不同的是，在服务端完成接受套接字的初始化后，客户端应该首先对发送套接字进行初始化，以保证后续发送能够正常进行，然后再进行接收套接字的初始化，这样客户端和服务端的双方通信得以正常展开。在初始化完成之后发送线程应该随时从键盘读取数据然后进行发送，接收线程应该随时读取接收缓冲区来判断是否接收到新的数据，并把新接收到的数据显示在屏幕上，如此客户端和服务端才能够正常地发送数据。

4.可行性分析

4.1 硬件环境

主机为 Intel 系列 CPU，内存为 8GB，虚拟机内存为 3GB，可以在主机上运行服务端程序，在虚拟机上运行客户端程序，或者也可以采用两台内存至少为 3GB 的主机连入同一局域网进行通信。

4.2 软件环境

主机和虚拟机均运行 Windows 系统，并且安装有 Python 环境，因此采用 Python 语言进行编程是可行的。此外，Python 语言中有 Openssl 以及 socket 的相关库函数，可以利用这些提供的函数进行证书加密传输，因此使用 Python 来设计局域网点对点基于证书认证的加密通信程序是较为合适和便捷的一种方案。

5.系统实现

5.1 系统实现方法

5.1.1 数字证书申请

申请 ca、client、server 三个证书，并将 ca 作为根证书，client 和 server 由 ca 进行颁发

在任意一台安装有 openssl 的主机上进行如下操作：

①首先要生成服务器端的私钥(serverkey.pem 文件):

openssl genrsa -des3 -out serverkey.pem 1024

②openssl req -new -key serverkey.pem -out servercsr.pem

生成 Certificate Signing Request（CSR）,生成的 servercsr.pem 文件交给 CA 签名后形成服务端自己的证书.屏幕上将有提示,依照其指示一步一步输入要 求的个人信息即可.

③对客户端也作同样的命令生成私钥及公钥文件:

openssl genrsa -des3 -out clientkey.pem 1024

openssl req -new -key clientkey.pem -out clientcsr.pem

④CSR 文件必须有 CA 的签名才可形成证书.此处我们自己做 CA。

首先生成 CA 的私钥文件:

openssl genrsa -des3 -out cakey.pem 1024

在生成 CA 自签名的证书:

openssl req -new -x509 -key cakey.pem -out cacrt.pem -days 365

⑤用生成的 CA 的证书为刚才生成的 servercsr.pem,clientcsr.pem 文件签名:

openssl ca -in servercsr.pem -out servercrt.pem -cert cacrt.pem -keyfile cakey.pem

openssl ca -in clientcsr.pem -out clientcrt.pem -cert cacrt.pem -keyfile cakey.pem

5.1.2 程序的详细实现

①服务端实现

服务端开启发送线程和接收线程，发送线程用于发送数据，接收线程用于接收数据。在初始化阶段需要验证 client 和 server 证书，并且建立发送和接收两个套接字，其中在建立发送套接字时需要初始化（socket）、连接（connect），在建立接受套接字时需要初始化（socket）、绑定（bind）、监听（listen）、接受连接（accept）。初始化完毕后，就可以开启（start）两个线程，接收线程循环接收数据（recv），并显示在屏幕上，发送线程循环读取键盘上的输入，并且进行发送（send），在循环中不断进行发送和接收，从而达到点对点安全通信的目的。具体实现的流程图如图 5-1、5-2 所示。

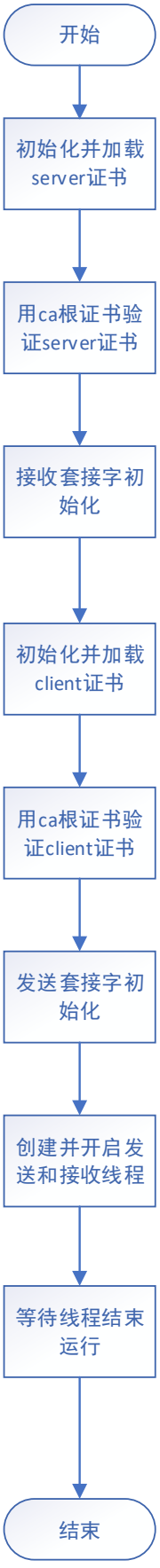

图 5-1 服务端主线程流程图

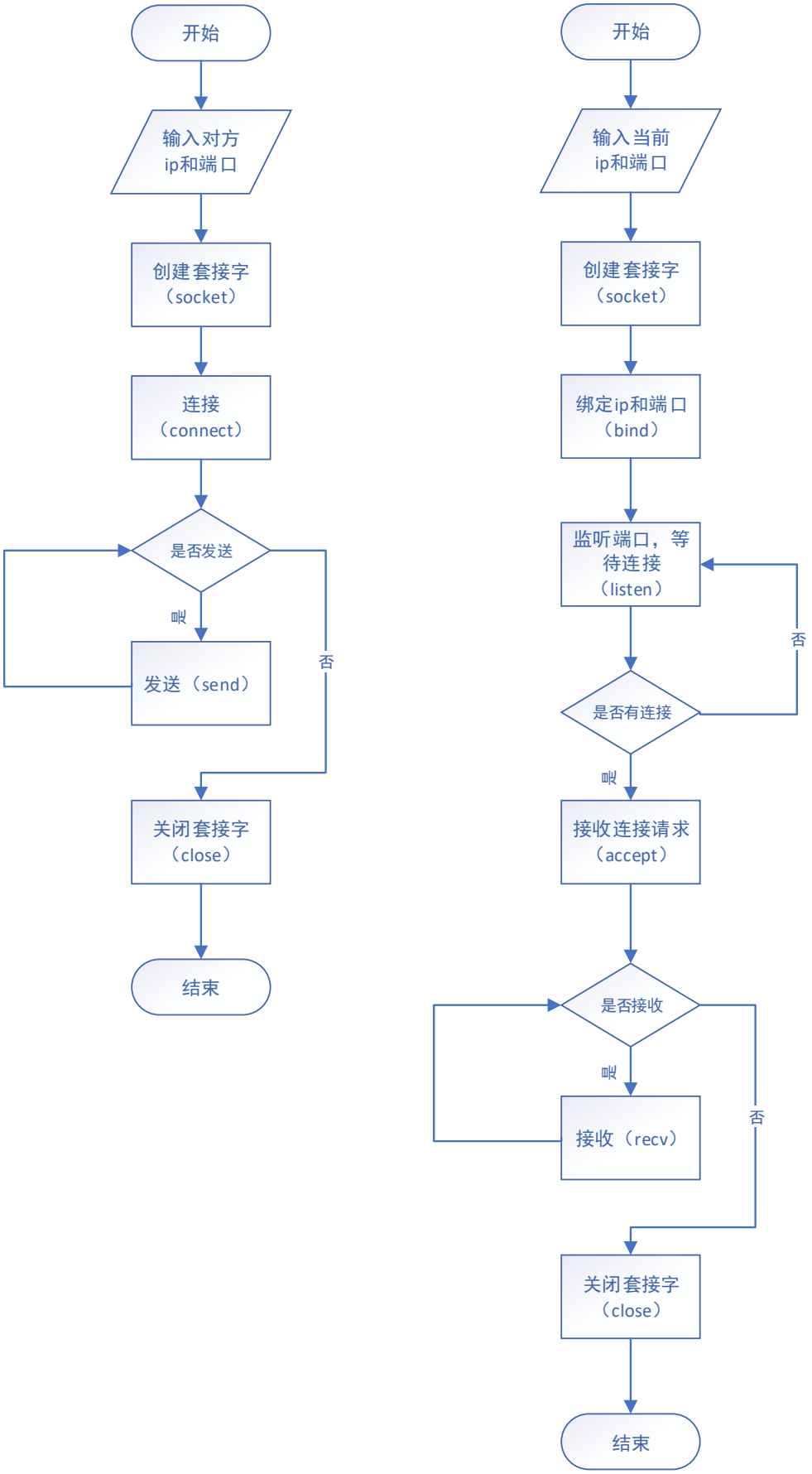

图 5-2 服务端发送线程和接收线程流程图

②客户端实现

与服务端类似，客户端也需要开启发送线程和接收线程，其中发送线程用于发送数据，接收线程用于接收数据。在初始化阶段需要用 ca 根证书验证 client 和 server 证书，并且建立发送和接收两个套接字，其中在建立发送套接字时需要初始化（socket）、连接（connect），在建立接受套接字时需要初始化（socket）、绑定（bind）、监听（listen）、接受连接（accept）。初始化完毕后，就可以开启（start）两个线程，接收线程循环接收数据（recv），并显示在屏幕上，发送线程循环读取键盘上的输入，并且进行发送（send），在循环中不断进行发送和接收，从而达到与服务端进行安全通信的目的。具体实现的流程图如图 5-3、5-4 所示。

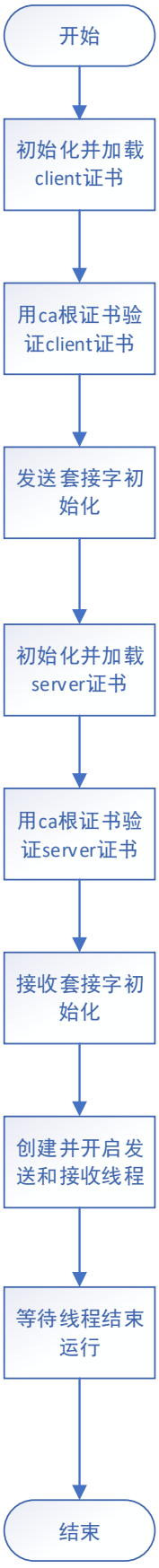

图 5-3 客户端主线程流程图

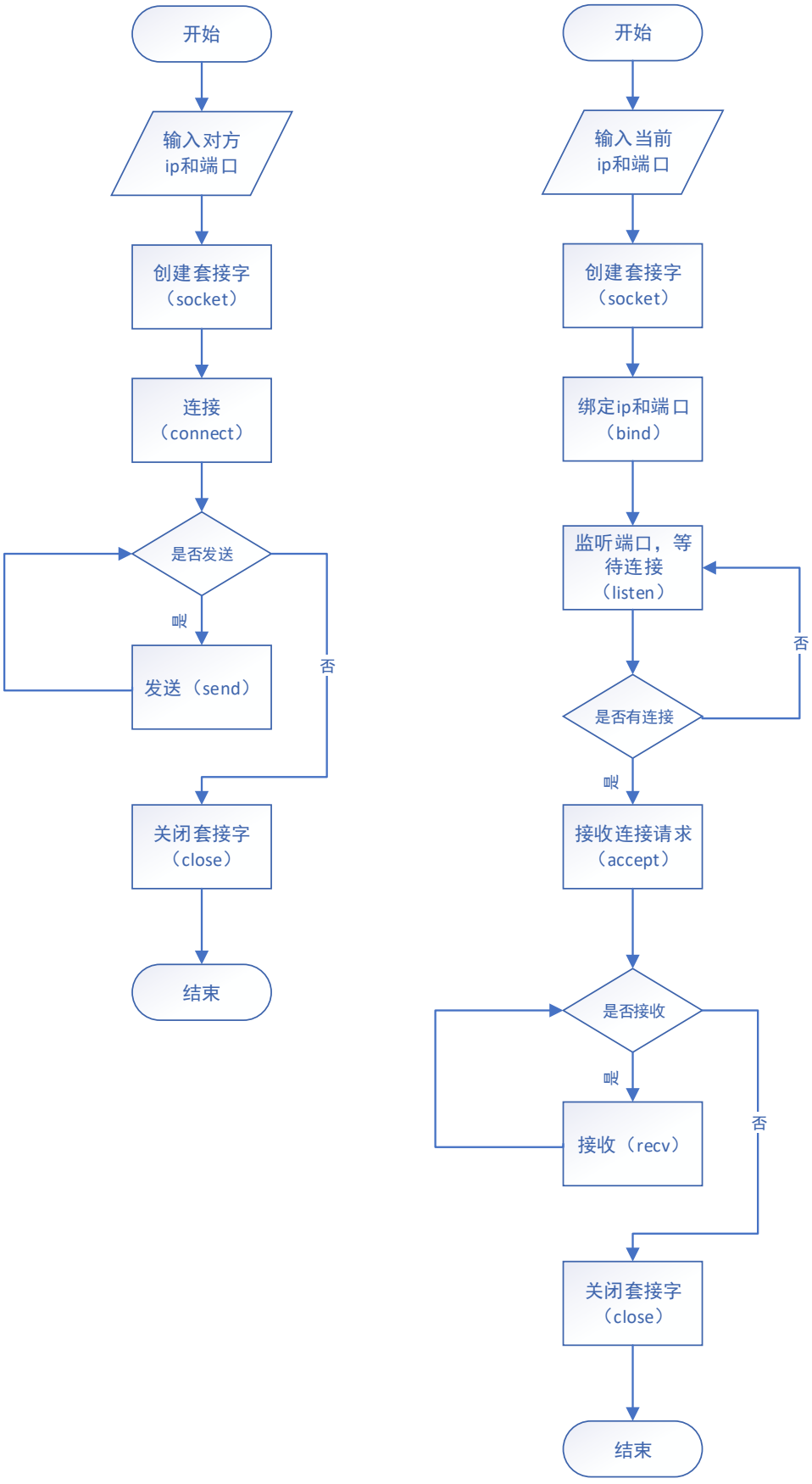

图 5-4 客户端发送线程和接收线程流程图

5.2 系统实现结果

5.2.1 数字证书申请结果

openssl 证书申请结果如图 5-5、5-6、5-7 所示

图 5-5 CA 根证书申请结果

图 5-6 server 证书申请结果

图 5-7 client 证书申请结果

5.2.2 程序运行结果

经过编程实现，客户端和服务端都能够向对方发送数据，对方也能够按时接收到相关的数据，结果如图 5-8、5-9 所示。

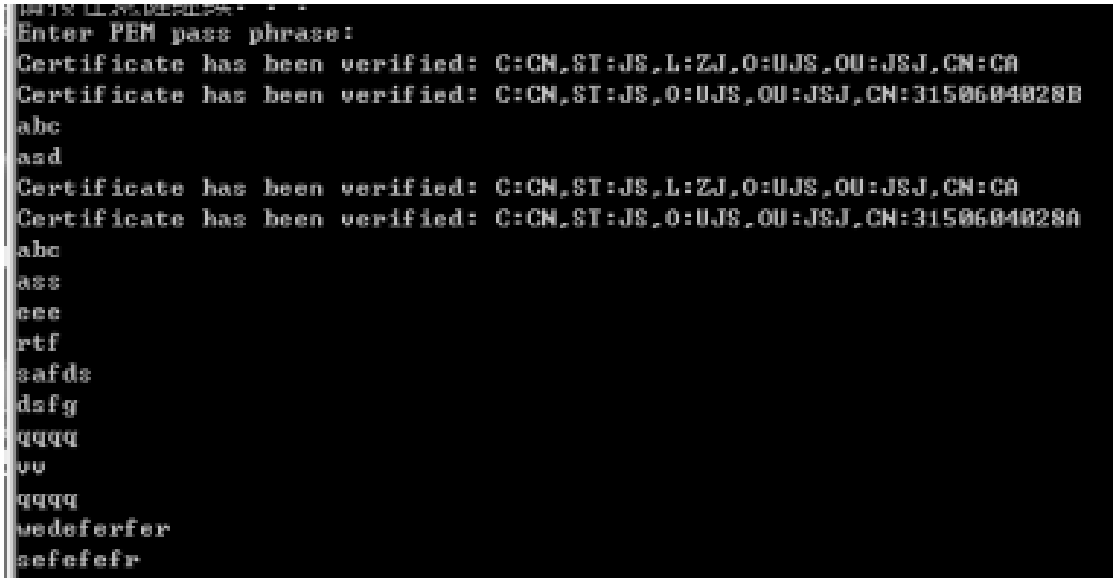

图 5-8 服务端运行结果

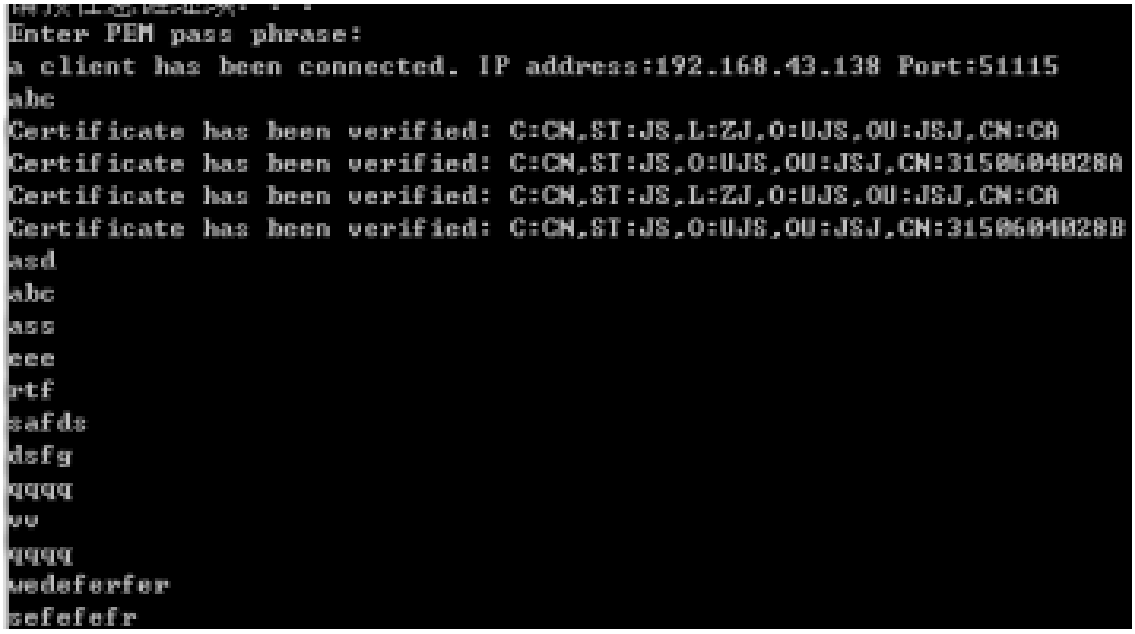

图 5-9 客户端运行结果

5.2.3 抓包分析

对数据进行抓包，如图 5-10 所示。

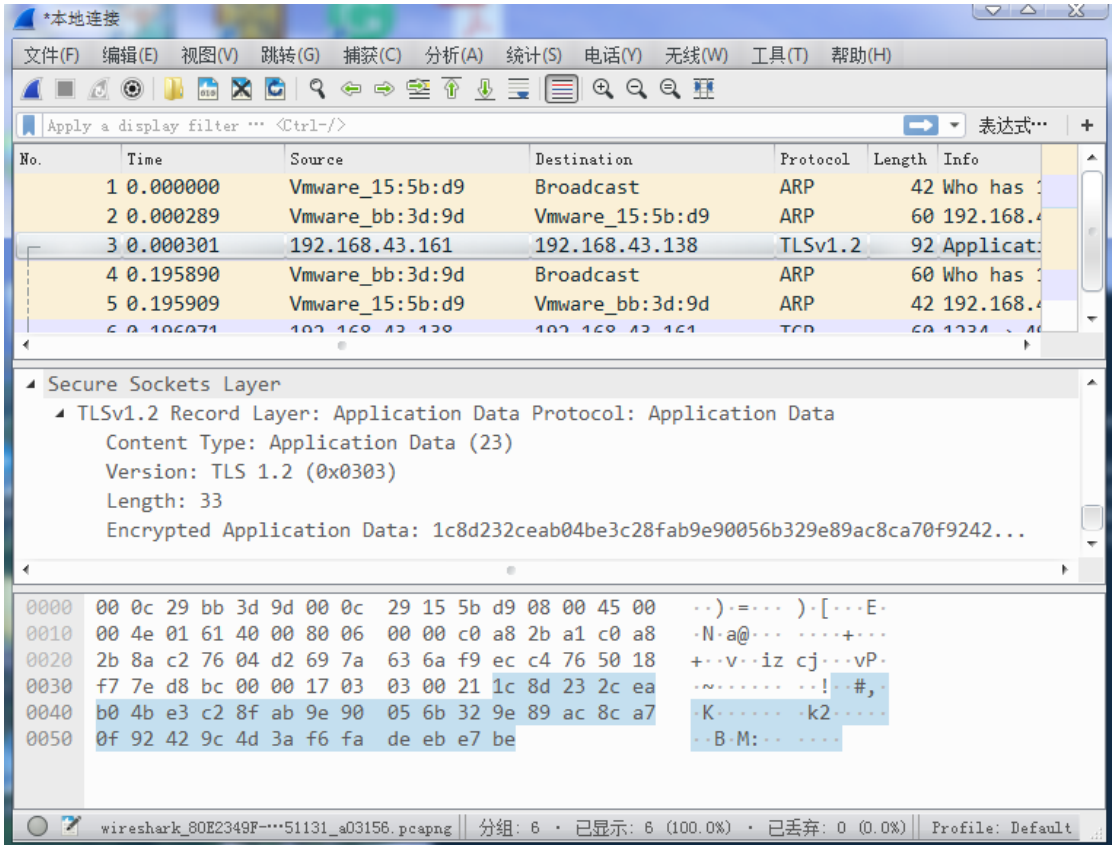

图 5-10 通信过程抓包分析

可以看到数据进行过加密，无法捕获明文数据。

6.总结

通过这次课程设计，我对网络安全有了一个全新的认识，并能能够较好地将网络安全的知识运用到实际的开发中去，做成一个有实际运用价值的加密通信系统。此外，这次课程设计也让我懂得了设计和开发时应该具有条理性，有条理地从服务端的设计到客户端的设计以及通信功能的实现，从完成基本的通信功能到利用证书进行加密通信。这

次课程设计也锻炼了我解决问题的能力，一开始我并不能熟练地操作openssl 库，不太会用 python 语言进行 socket 开发。面对这些问题，我查阅了相关书籍和网上的相关资料，找到了正确处理这些问题的方法，从而很好地运用了 python 语言结合 openssl 库进行开发。运用ssl 进行通讯既有优点也有不足，它实现了保密通信，能够有效地防止中间人攻击窃取数据，然而利用这种方式进行通讯花费巨大，且对服务器的资源占用较大，容易拖慢服务器的运行速度。

这次课程设计也让我明白了服务端与客户端的开发同样重要。因为只有服务端和客户端都能够完美地运行，整个系统才能实现必要的功能。除此之外，这次课程设计也让我领会到了理论知识和实际操作的重要性，学好网络安全的相关理论知识才能对通信系统有一个良好的设计。然而，仅仅局限于理论知识是远远不够的，把所学的理论知识运用到实际开发中来才能发挥价值。因此，这次我把平时学到的网络安全的理论知识运用到了点对点通信系统的实际开发中，进行了一个系统性的设计。

经过一个星期的设计，虽然过程并不是一帆风顺的，但是苦尽甘来，我最终也顺利完成了设计。无论如何，我最终设计出了令人满意的基于数字证书验证的通信系统。这次课程设计给我带来的不仅仅是知识上的收获，还有精神上的财富。我通过这次课程设计懂得了要把平时学到的理论知识运用到实际的开发和设计环节中来，并在开发和设计中不断发现和解决问题，不断做自我修正，为以后的开发作铺垫。

附：重要程序清单（含注释）

服务端代码：

```
import socket,sys,os,select,time,threading,msvcrt
from OpenSSL import *
def verifycert_func(connection_test, certificate_test, error_number, depth, ok):
    # 验证证书有效性
    certstr=certificate_test.get_subject()
    tempstr=str(certstr)
    tempstr=tempstr[19:-2]
    tempstr=tempstr.replace('/',',')
    tempstr=tempstr.replace('=',':')
    print 'Certificate has been verified: ' + tempstr
    return ok
ip=raw_input("Enter IP address:")
port=raw_input("Enter port number:")
os.system("pause")
threadLock=threading.Lock()
# 初始化并加载证书
receiving_context = SSL.Context(SSL.SSLv23_METHOD)
receiving_context.set_options(SSL.OP_NO_SSLv2)
```

```
receiving_context.set_verify(SSL.VERIFY_PEER|SSL.VERIFY_FAIL_IF_NO
_PEER_CERT, verifycert_func) # Demand a certificate
receiving_context.use_privatekey_file ('serverkey.pem')
receiving_context.use_certificate_file('servercrt.pem')
receiving_context.load_verify_locations('cacrt.pem')
# 设置接收套接字
socket_r = SSL.Connection(receiving_context,
socket.socket(socket.AF_INET, socket.SOCK_STREAM))
socket_r.bind((ip, int(port)))
socket_r.listen(10)
connection_, address_ = socket_r.accept()
print 'a client has been connected. IP address:'+str(address_[0])+'
Port:'+str(address_[1])
ip_=raw_input("Enter IP address:")
port_=raw_input("Enter port number:")
os.system("pause")
# 初始化并加载证书
sending_context = SSL.Context(SSL.SSLv23_METHOD)
sending_context.set_verify(SSL.VERIFY_PEER, verifycert_func) #
Demand a certificate
sending_context.use_privatekey_file ('clientkey.pem')
```

```
sending_context.use_certificate_file('clientcrt.pem')
sending_context.load_verify_locations('cacrt.pem')
# 设置发送套接字
socket_s = SSL.Connection(sending_context,
socket.socket(socket.AF_INET, socket.SOCK_STREAM))
socket_s.connect((ip_, int(port_)))


#发送线程
class Sending_Thread(threading.Thread):
    def __init__(self,threadID,name,counter):
        threading.Thread.__init__(self)
        self.threadID=threadID
        self.name=name
        self.counter=counter

    def run(self):

        while 1:
            sending()
```

```
#接收线程
class Receiving_Thread(threading.Thread):
    def __init__(self,threadID,name,counter):
        threading.Thread.__init__(self)
        self.threadID=threadID
        self.name=name
        self.counter=counter

    def run(self):



        receiving()



#用于发送的函数
def sending():
    sending_message = sys.stdin.readline()#读入输入的数据

    if sending_message=='q\n':
```

```
            return
        try:
            socket_s.send(sending_message)#发送数据
        except SSL.Error:
            print 'link has been disconnected.'
            return
#用于接收的函数
def receiving():
    while 1:
        try:
            received_message = connection_.recv(1024)#接收数据
            sys.stdout.write(received_message)#显示
            sys.stdout.flush()
        except SSL.Error:
            print 'link has been disconnected.'
            return
#主程序中开启发送和接收线程
receiving_T=Receiving_Thread(1,"receive",1)
sending_T=Sending_Thread(2,"send",2)
receiving_T.start()
sending_T.start()
```

客户端代码：

```
import socket,sys,os,select,time,threading,msvcrt
from OpenSSL import *
def verifycert_func(connection_test, certificate_test, error_number, depth, ok):
    # 验证证书有效性
    certstr=certificate_test.get_subject()
    tempstr=str(certstr)
    tempstr=tempstr[19:-2]
    tempstr=tempstr.replace('/',',')
    tempstr=tempstr.replace('=',':')
    print 'Certificate has been verified: ' + tempstr
    return ok
ip=raw_input("Enter IP address:")
port=raw_input("Enter port number:")
os.system("pause")
threadLock=threading.Lock()
# 初始化并加载证书
sending_context = SSL.Context(SSL.SSLv23_METHOD)
sending_context.set_verify(SSL.VERIFY_PEER, verifycert_func) # Demand a certificate
```

```
sending_context.use_privatekey_file ('clientkey.pem')
sending_context.use_certificate_file('clientcrt.pem')
sending_context.load_verify_locations('cacrt.pem')
# 设置发送套接字
socket_s = SSL.Connection(sending_context,
socket.socket(socket.AF_INET, socket.SOCK_STREAM))
socket_s.connect((ip, int(port)))

ip_=raw_input("Enter IP address:")
port_=raw_input("Enter port number:")
os.system("pause")



# 初始化并加载证书
receiving_context = SSL.Context(SSL.SSLv23_METHOD)
receiving_context.set_options(SSL.OP_NO_SSLv2)
receiving_context.set_verify(SSL.VERIFY_PEER|SSL.VERIFY_FAIL_IF_NO
_PEER_CERT, verifycert_func) # Demand a certificate
receiving_context.use_privatekey_file ('serverkey.pem')
receiving_context.use_certificate_file('servercrt.pem')
```

```
receiving_context.load_verify_locations('cacrt.pem')
# 设置接收套接字
socket_r = SSL.Connection(receiving_context,
socket.socket(socket.AF_INET, socket.SOCK_STREAM))
socket_r.bind((ip_, int(port_)))
socket_r.listen(10)
connection_, address_ = socket_r.accept()
print 'a client has been connected. IP address:'+str(address_[0])+'
Port:'+str(address_[1])

#发送线程
class Sending_Thread(threading.Thread):
    def __init__(self,threadID,name,counter):
        threading.Thread.__init__(self)
        self.threadID=threadID
        self.name=name
        self.counter=counter

    def run(self):

        while 1:
```

```
            sending()


#接收线程
class Receiving_Thread(threading.Thread):
    def __init__(self,threadID,name,counter):
        threading.Thread.__init__(self)
        self.threadID=threadID
        self.name=name
        self.counter=counter

    def run(self):




        receiving()



#用于发送的函数
def sending():
    sending_message = sys.stdin.readline()     #读入输入的数据
```

```
    if sending_message=='q\n':
        return
    try:
        socket_s.send(sending_message)#发送数据
    except SSL.Error:
        print 'link has been disconnected.'
        return
#用于接收的函数
def receiving():
    while 1:
        try:
            received_message = connection_.recv(1024)#接收数据
            sys.stdout.write(received_message)#显示
            sys.stdout.flush()
        except SSL.Error:
            print 'link has been disconnected.'
            return
#主程序中开启发送和接收线程
receiving_T=Receiving_Thread(1,"receive",1)
sending_T=Sending_Thread(2,"send",2)
```

```
receiving_T.start()

sending_T.start()
```