**文件安全传输系统**

**设计与实现**

|  |  |
| --- | --- |
| 分组序号 | 第5组 |
| 课题名称 | 文件安全传输系统的设计 |
| 分组成员 | 汪 聪 4052014088 |
|  | 闫 朔 4052014082 |
|  | 何祖鑫 4052014015 |
|  | 吴 洋 40520140 |

目录

**一、课题任务1**

1.1开发环境1

1.2主要任务1

**二、总体设计1**

2.1程序框架1

2.2设计流程2

2.3验证模型3

2.4传输模型3

**三、详细设计4**

3.1基础协议4

3.2组件实现5

3.3用户界面16

**四、课题总结19**

4.1遇到的问题及解决方法19

4.2实验分工20

4.3心得体会21

一、课题任务

1.1开发环境

VisualStdio2017 .Net4.5 WPF

1.2主要任务

实现通过网络进行文件加密传输，要求进行密钥协商，及大文件的传输，程序使用C#进行开发，有良好的用户界面。

二、总体设计

2.1程序框架

2.2设计流程

安全文件传输系统

数据绑定

前后端链接

安全传输功能整合

WPF前端开发

传输组件的实现

安全组件的实现

密钥协商

协议设计

加密解密

算法选择与实现

传输协议设计

界面设计

2.3验证模型

发送随机数作为挑战值

验证方

受验方

由随机数生成密钥并加密应答值

解密应答比对应答是否有效，发送结果

认证成功则使用共享密钥建立安全链路

2.4传输模型

接受方

发送方

传输连接的建立

发送文件请求

是否接收

开始发送

Block[0]

Block[1]

Block[2]

…

Block[n]

接收完毕请求校验

校验并告知结果

断开连接

三、详细设计

3.1基础协议

⑴基于CHAP的密钥协商

由传输请求方发起请求，接收方发起挑战，挑战值为随机生成的6位整数。该挑战值即为密钥生成密钥，生成算法为：

请求方收到挑战值后，对其用预协商的算法进行变换，同时生成密钥，变换算法为：

之后生成响应值：

接收方收到回应后同本地计算的预期回应值比对，相同则验证成功，同时将*Key*作为传输密钥。

⑵基于AES的文件分块加密

文件传输采用AES加密，密钥由基于CHAP的密钥协商生成，使用*System.Security.Cryptography.AesCryptoServiceProvider*生成AES服务实例，配置

*AES.Key = Encoding.ASCII.GetBytes(Key);*

*AES.IV = Encoding.ASCII.GetBytes(Key.Substring(0, 16));*

*AES.Padding = PaddingMode.Zeros;*

保存加密服务实例。

文件加密采用分块加密，设置块缓冲区的大小为*File\_Buffer\_Size*（默认为1MB），每次对一个文件块进行加密，最后一块若需填充则已设置的填充方式为准。

⑶自定义的传输控制协议

由发送方发起文件传输请求*Transmit\_Request*；

接受方选择是否接收*Transmit\_Allow*或*Transmit\_Cancel*；

发送方发送完毕后等待接收方的接收完成；

接收方接收并解密完成后计算文件*MD5*值*Transmit\_Over*；

发送方收到接收完毕的消息及MD5值后，比对本地文件的MD5值，告知接收端*VerifySuccess*或*VerifyFailed*；

3.2组件实现

⑴认证协议的实现

服务端发起挑战部分代码：

//生成一个100000-999999随机数作为挑战值challenge\_value

//要求对方收到挑战值后，对称密钥加密回送应答

int challenge\_value = new *Random*().**Next**(100000, 999999);

textBox\_challengeValue.Text = challenge\_value.**ToString**();

//用string储存密钥

*StringBuilder* sBuilder = new *StringBuilder*();

byte[] hash;

string Key;

using (*MD5* md5Hash = MD5.**Create**())

{

//生成密钥

//Hash((1-challenge\_value).ToString())作为预定密钥

hash = md5Hash.**ComputeHash**(Encoding.UTF8.**GetBytes**((1 - challenge\_value).**ToString**()));

foreach (byte b in hash)

{

//哈希值本身为16字节，AES的密钥需要32字节，将其转换为小写的16进制

sBuilder.**Append**(b.**ToString**("x2"));

}

Key = sBuilder.**ToString**();

textBox\_key.Text = sBuilder.**ToString**();

//计算响应值

//Hash(Hash(challenge\_value.ToString()))为应答

hash = md5Hash.**ComputeHash**(Encoding.UTF8.**GetBytes**(challenge\_value.**ToString**()));

hash = md5Hash.**ComputeHash**(hash);

}

//计算响应密文(期望响应)

AES = DataCrypto.**GenAesCryptoServiceProvider**(Key);

//字符串采用base64编码

ExpertResponse = Convert.**ToBase64String**(DataCrypto.**Encrypt**(hash, AES));

textBox\_expertResponse.Text = ExpertResponse;

//生成挑战消息并发送

*Authen\_Message* am = new *Authen\_Message*(Status\_Flag.Start\_Challenge, DateTime.Now, challenge\_value.**ToString**());

socketConn.**Send**(**Message2Byte**(am));

客户端响应挑战部分代码：

//生成密钥，过程与服务端一致

//收到的挑战值显示于textBox\_challengeValue

*StringBuilder* sBuilder = new *StringBuilder*();

byte[] hash;

using (*MD5* md5Hash = MD5.**Create**())

{

hash = md5Hash.**ComputeHash**(Encoding.UTF8.**GetBytes**((1 - challenge\_value).**ToString**()));

foreach (byte b in hash)

{

sBuilder.**Append**(b.**ToString**("x2"));

}

Key = sBuilder.**ToString**();

textBox\_key.Text = sBuilder.**ToString**();

}

*////////////////////////////////////////////////////////////////////////////////////////////////////*

//生成响应值

*StringBuilder* sBuilder = new *StringBuilder*();

byte[] hash;

using (*MD5* md5Hash = MD5.**Create**())

{

//计算响应值

hash = md5Hash.**ComputeHash**(Encoding.UTF8.**GetBytes**(textBox\_challengeValue.Text));

hash = md5Hash.**ComputeHash**(hash);

}

//生成AES服务实例

AES = DataCrypto.**GenAesCryptoServiceProvider**(Key);

//计算预期响应

string Response = Convert.**ToBase64String**(DataCrypto.**Encrypt**(hash, AES));

textBox\_genResponse.Text = Response;

*Authen\_Message* am = new *Authen\_Message*(Status\_Flag.Response\_Challenge, DateTime.Now, Response);

socketConn.**Send**(**Message2Byte**(am));

服务器端验证响应部分代码：

//如果收到的响应值与预期响应值相同则认证成功，否则失败

if (ExpertResponse == am.Extend)

{

//生成认证成功的消息并发送

*Authen\_Message* au = new *Authen\_Message*(Status\_Flag.Authen\_Success, DateTime.Now, null);

socketConn.**Send**(**Message2Byte**(au));

//标记为已认证

IsAuthenticated = true;

//开一个传输控制线程，进入传输控制阶段

*Thread* trd = new *Thread*(TransControl);

trd.IsBackground = true;

trd.**Start**();

break;

}

else

{

//生成认证失败的消息并发送

*Authen\_Message* au = new *Authen\_Message*(Status\_Flag.Authen\_Failed, DateTime.Now, null);

socketConn.**Send**(**Message2Byte**(au));

}

客户端收到认证成功消息后进入传输控制阶段，失败则退出验证阶段。

⑵AES加密类的构造

//有32字节的密钥(字符串形式)生成AES服务提供者实例

public static *AesCryptoServiceProvider* **GenAesCryptoServiceProvider**(string Key)

{

*AesCryptoServiceProvider* AES = new *AesCryptoServiceProvider*();

//配置密钥

AES.Key = Encoding.ASCII.**GetBytes**(Key);

//配置向量为密钥的前16字节

AES.IV = Encoding.ASCII.**GetBytes**(Key.**Substring**(0, 16));

//配置填充模式为填充0

AES.Padding = PaddingMode.Zeros;

return AES;

}

//加密明文字节数组，返回密文字节数组，参数为明文与AES实例

public static byte[] **Encrypt**(byte[] cipher, *AesCryptoServiceProvider* AES)

{

//初始化一个流存放密文

*MemoryStream* ms = new *MemoryStream*();

//创建加密转换实例

*ICryptoTransform* AESencrypt = AES.**CreateEncryptor**();

//构建加密流

*CryptoStream* cryptostream = new *CryptoStream*(ms, AESencrypt, CryptoStreamMode.Write);

//将明文字节数组写入加密流

cryptostream.**Write**(cipher, 0, cipher.Length);

//将缓冲区内数据立即加密，并清除缓冲区

cryptostream.**FlushFinalBlock**();

//将密文由流转为字节数组

byte[] code = ms.**ToArray**();

//关闭流

cryptostream.**Close**();

ms.**Close**();

return code;

}

//解密密文字节数组，返回明文字节数组，参数为密文与AES实例

public static byte[] **Decrypt**(byte[] code, *AesCryptoServiceProvider* AES)

{

*MemoryStream* ms = new *MemoryStream*();

//创建解密转换实例

*ICryptoTransform* AESdecrypt = AES.**CreateDecryptor**();

*CryptoStream* cryptostream = new *CryptoStream*(ms, AESdecrypt, CryptoStreamMode.Write);

cryptostream.**Write**(code, 0, code.Length);

cryptostream.**FlushFinalBlock**();

byte[] cipher = ms.**ToArray**();

cryptostream.**Close**();

ms.**Close**();

return cipher;

}

⑶传输协议的实现

消息标志设置

public enum Status\_Flag

{

Start\_Challenge = 1,

Response\_Challenge,

Authen\_Success,

Authen\_Failed,

Time\_Out,

Transmit\_Request,

Transmit\_Allow,

Transmit\_Cancel,

Transmit\_Over,

VerifySuccess,

VerifyFailed

}

认证消息结构

public struct Authen\_Message

{

//消息标志

public *Status\_Flag* Flag;

//时间戳

public *DateTime* Time;

//强制填充字符串至256位，保证包长一致，方便消息数据的封包解包

[*MarshalAs*(UnmanagedType.ByValTStr, SizeConst = 256)]

//扩展字段

public string Extend;

//构造函数

public **Authen\_Message**(*Status\_Flag* flag, *DateTime* t, string str)

{

this.Flag = flag;

this.Time = t;

this.Extend = str;

}

//返回结构信息的方法

public string **MessageInfo**()

{

return "{\n FLAG : " + Flag.**ToString**() + "\n TIME : " + Time.**ToLongTimeString**() + "\n Exten : " + Extend + "\n}";

}

}

传输控制消息结构

public struct TransCtrl\_Message

{

public *Status\_Flag* Flag;

public *DateTime* Time;

[*MarshalAs*(UnmanagedType.ByValTStr, SizeConst = 256)]

//文件名

public string FileName;

//文件长度

public long FileLength;

public **TransCtrl\_Message**(*Status\_Flag* flag, *DateTime* t, string file\_naem, long file\_length)

{

this.Flag = flag;

this.Time = t;

this.FileName = file\_naem;

this.FileLength = file\_length;

}

public string **MessageInfo**()

{

return "{\n FLAG : " + Flag.**ToString**() + "\n TIME : " + Time.**ToLongTimeString**() + "\n FileName : " + FileName + "\n FileLength : " + FileLength.**ToString**() + "\n}";

}

}

传输控制协议部分代码

//如果对方接收文件

if (tm.Flag == Status\_Flag.Transmit\_Allow)

{

//开辟线程发送文件

*ParameterizedThreadStart* pts = new *ParameterizedThreadStart*(SendFile);

*Thread* threadWatch = new *Thread*(pts);

threadWatch.IsBackground = true;

// SendFileInfo为待发送文件的文件路径及长度

threadWatch.**Start**(SendFileInfo);

}

//如果对方取消接收文件

if (tm.Flag == Status\_Flag.Transmit\_Cancel)

{

this.Dispatcher.**Invoke**(new *Action*(() =>

{

**WriteLog**("对方拒绝接收");

snackbar.MessageQueue.**Enqueue**("对方拒绝接收");

}));

//跳出接收消息的循环

break;

}

//如果对方接收到文件并解密完毕

if (tm.Flag == Status\_Flag.Transmit\_Over)

{

//计算校验值

string file\_hash = null;

using (*MD5* md5Hash = MD5.**Create**())

{

*FileStream* fs = new *FileStream*(SendFileInfo.FileName, FileMode.Open, FileAccess.Read);

byte[] hash = md5Hash.**ComputeHash**(fs);

file\_hash = Convert.**ToBase64String**(hash);

}

// Transmit\_Over消息中的FileName字段用于储存哈希值

//如果本地计算的哈希值与对方哈希值一致，则校验成功

if (file\_hash == tm.FileName)

{

//回送校验成功的消息

*TransCtrl\_Message* tcm = new *TransCtrl\_Message*(Status\_Flag.VerifySuccess, DateTime.Now, null, 0);

socketConn.**Send**(**Message2Byte**(tcm));

}

//反之则不成功

else

{

//回送消息

*TransCtrl\_Message* tcm = new *TransCtrl\_Message*(Status\_Flag.VerifyFailed, DateTime.Now, null, 0);

socketConn.**Send**(**Message2Byte**(tcm));

}

文件发送方法部分代码

//文件信息结构，包含文件路径和长度

*File\_Info* fi = (*File\_Info*)fileParam;

socketSendFile = new *Socket*(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

//传输端口默认为8089,接收端监听此端口，发送端一旦接入即开始发送

*IPEndPoint* iep = new *IPEndPoint*(((*IPEndPoint*)socketConn.RemoteEndPoint).Address, 8089);

try

{

socketSendFile.**Connect**(iep);

}

Catch

{

//异常处理

//return;

}

//定义变量用来保存当前已接收的数据量

long offset = 0;

while (true)

{

//读取文件块

byte[] Read\_buffer = **FileRead**(fi.FileName, offset, File\_Buffer\_Size);

int length = Read\_buffer.Length;

offset += length;

//加密文件块

byte[] Send\_Buffer = DataCrypto.**Encrypt**(Read\_buffer, AES);

try

{

length = socketSendFile.**Send**(Send\_Buffer);

}

catch (*Exception* ex)

{

//网络异常则停止发送

break;

}

//如果超过或等于文件长度则停止发送

if (offset >= fi.FileLength)

{

break;

}

//回收内存

System.GC.**Collect**();

}

接收文件部分代码

*File\_Info* fi = (*File\_Info*)fileParam;

//开始监听文件传送端口

*Socket* socket = new *Socket*(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

*IPEndPoint* iep = new *IPEndPoint*(((*IPEndPoint*)socketConn.LocalEndPoint).Address, 8089);

socket.**Bind**(iep);

socket.**Listen**(1);

try

{

socketRecFile = socket.**Accept**();

//停止对端口的监听

socket.**Close**();

}

catch

{

}

//开一个线程进行解密，参数为文件信息

*ParameterizedThreadStart* pts = new *ParameterizedThreadStart*(DecryptData);

*Thread* trd = new *Thread*(pts);

trd.IsBackground = true;

trd.**Start**(fi);

//计算加密填充后的长度

long file\_length = fi.FileLength + ((fi.FileLength % 16 == 0) ? 0 : (16 - fi.FileLength % 16));

long offset = 0;

while (true)

{

int length = -1;

byte[] buffer = new byte[File\_Buffer\_Size];

try

{

length = socketRecFile.**Receive**(buffer);

}

catch (*Exception* ex)

{

//网络异常则停止接收

break;

}

//设置读写锁进入写，将禁止所有读锁

\_rwlock.**EnterWriteLock**();

**FileWrite**(fi.FileName + ".tmp", offset, length, buffer);

\_rwlock.**ExitWriteLock**();

offset += length;

//如果已经接收完毕，便停止接收

if (offset >= file\_length)

{

//重置套接字

try

{

socketRecFile.**Close**();

}

catch (*Exception* ex)

{

}

socketRecFile = null;

}));

break;

}

System.GC.**Collect**();

}

解密文件部分代码

*File\_Info* fi = (*File\_Info*)fileParam;

//填充后的长度

long file\_length = fi.FileLength + ((fi.FileLength % 16 == 0) ? 0 : (16 - fi.FileLength % 16));

long read\_offset = 0;

long write\_offset = 0;

//需要读取的长度

int read\_size = File\_Buffer\_Size;

//需要写入的长度

int write\_size = File\_Buffer\_Size;

while (true)

{

//如果是最后一个分块则修改size

if (read\_offset + File\_Buffer\_Size > file\_length)

{

read\_size = (int)(file\_length % (long)File\_Buffer\_Size);

write\_size = (int)(fi.FileLength % (long)File\_Buffer\_Size);

}

//开始接收

while (true)

{

//如果不足一块则等待

if (!File.**Exists**(fi.FileName + ".tmp") ||

new *FileInfo*(fi.FileName + ".tmp").Length < read\_offset + read\_size)

{

continue;

}

break;

}

\_rwlock.**EnterReadLock**();

byte[] Read\_Buffer = **FileRead**(fi.FileName + ".tmp", read\_offset, read\_size);

\_rwlock.**ExitReadLock**();

read\_offset += read\_size;

byte[] Write\_Buffer = null;

try

{

Write\_Buffer = DataCrypto.**Decrypt**(Read\_Buffer, AES);

}

catch (*Exception* ex)

{

//终止传输

try

{

socketRecFile.**Shutdown**(SocketShutdown.Both);

socketRecFile.**Close**();

socketRecFile = null;

}

catch (*Exception* e)

{

**WriteLog**(e.**ToString**());

**WriteLog**("停止接收 " + new *FileInfo*(fi.FileName).Name);

snackbar.MessageQueue.**Enqueue**("停止接收" + new *FileInfo*(fi.FileName).Name);

}

break;

}

//写入解密后的文件

**FileWrite**(fi.FileName, write\_offset, write\_size, Write\_Buffer);

write\_offset += write\_size;

if (write\_offset >= fi.FileLength)

{

//删除tmp文件

File.**Delete**(fi.FileName + ".tmp");

//计算校验值

string file\_hash = null;

using (*MD5* md5Hash = MD5.**Create**())

{

*FileStream* fs = new *FileStream*(fi.FileName, FileMode.Open, FileAccess.Read);

byte[] hash = md5Hash.**ComputeHash**(fs);

file\_hash = Convert.**ToBase64String**(hash);

}

//回送消息

*TransCtrl\_Message* tm = new *TransCtrl\_Message*(Status\_Flag.Transmit\_Over, DateTime.Now, file\_hash, 0);

socketConn.**Send**(**Message2Byte**(tm));

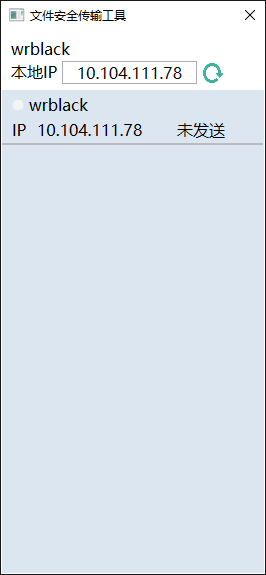
break;

}

}

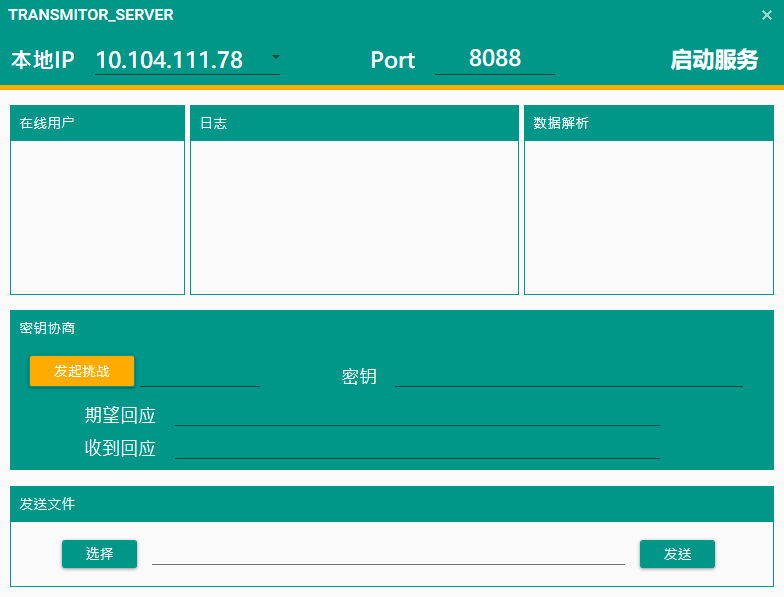
3.3界面设计

⑴P2P版本的界面设计



用户使用时不区分客户端服务端，后台自动完成在线列表的维护，每个用户的信息用一个用户控件UserInfo\_Control 显示，双击该控件选择文件发送至该控件对应的远端用户。收到用户的发送请求时，用户控件会调整背景色为橙色提示用户接收，双击控件即可选择保存位置。传送进度由控件底部的线式进度条显示。

⑵演示版界面设计

服务端界面

演示程序分为服务端和客户端两部分，P2P程序同时扮演两个角色。服务端负责验证与控制部分套接字的绑定与监听，用户可以选择监听地址与端口，已连接的将加入在线用户列表，日志负责记录相关操作，数据解析窗口提供接收数据的可视化。客户端连接后，服务端可以发起挑战，此时将生成挑战值、密钥和期望回应，收到用户的回应将显示在文本框中，用户可以比对，系统将自动比对并将比对结果输出至日志，认证成功后即可互相发送文件。发送过程中的进度显示与发送按钮的右侧，为环形进度条。接收过程的进度条置于窗体上方，黄色线条形进度条。

客户端界面



客户端需要主动连接服务端，可以自己填写要连接的地址与端口。连接结束后将等待服务端发起挑战，收到挑战后，用户可以点击计算密钥生成密钥，点击生成响应将计算响应并加密发送至服务端，验证成功后，即可收发文件。

四、课题总结

4.1遇到的问题及解决方法

⑴P2P版本中的在线列表如何创建开始时给我们出了一个难题，最初提出假设一个中央服务器用于保存用户的信息，但这样的话软件就无法实现纯P2P设计了。后来我们借鉴了飞秋的工作方式，在程序中加入了在线信息控制协议，通过广播通知的方式告知各个客户端上线信息，解决了这个问题。

⑵加密后数据扩充的问题惊呆了我们，我们发现在使用AES加密的过程中，即使数据是16字节的整数倍，密文也回比明文多出16字节。经过调试测试我们发现.NET默认的AES填充方式为PSCK2，这种填充模式会加入校验，导致密文数据的增加。后来改用零填充解决了该问题。

⑶数据半包粘包的问题是本次开发过程中最大的收获。我们的程序在单机回环地址测试时能够正常运行，结果一到了网络环境就出现解密错误的问题，经过调试发现，网络环境下由于一系列原因会导致数据无法按照我们预想的一次性接收整个数据包，至此发现了半包粘包问题。最初我们采用循环获取，取整处理的方式解决了这个问题，可是这种方法传输效率不高，后来我们采用了写临时文件的方式持续接收，提供了较高的传输效率，同时也减小了CPU与内存的占用率，实际测试的效果表明可以满足日常需求。

4.2实验分工

安全文件传输系统

数据绑定

前后端链接

安全传输功能整合

传输组件的实现

WPF前端开发

安全组件的实现

传输协议设计

密钥协商

协议设计

加密解密

算法选择与实现

界面设计

汪 聪 总体设计、前端开发、组件链接

闫 朔 网络构建、相关网络问题的调试

吴 洋 加密解密类的实现

何祖鑫 密钥协商、传输控制协议的设计

各司其职，同心协力，共同进步

4.3心得体会

汪聪：本次开发过程让我收获良多，我对于解决方案的设计又有了更深的认识，以后再进行复杂项目的研发时我就会严格做好工程设计，对各组件进行详细的定义与设计，对于系统内各部分的交互也会在最开始的时候进行设计，防止在项目实现阶段出现不好解决的问题，为系统预留拓展接口便于扩充功能。

另外，对于Socket通信程序的设计，我又有了更多的认识和经验，相信在以后的项目实战过程中能够更加熟练。

最后，本次开发过程是极其煎熬的，我在前阶段由于参加了一个嵌入式物联网的比赛一直加班熬夜。后期开发时间紧了，我又连续奋战，没有想到我能够挺过来，意志力、吃苦能力、钻研能力都得到的较大的磨练与提升，相信这段日子会给我的大学回忆留下深刻的一笔。

闫朔：通过本学期对C#学习，我深入理解了socket网络通信的基本原理，包括TCP层面的网络通信，socket的粘包、半包问题，关于学术上的问题报告上面已经说了很多了，我更想谈的是这段实验期间我的所感所想：

经过一系列高强度的网络编程我也学到了分析代码的技巧，虽然经历了很多次的失败，但我们都没有放弃，因为我们是一个团队，每个人都应该履职尽责，与此同时我也深深的体会到了一名程序员的艰辛，可喜的是本学期C#实验让我真正的学到了很多东西，也体会到了最终胜利的喜悦。我还深深记得教员对我们的基本要求，第一:提高每个人动手能力;第二:提高团队协作能力;第三:提高大家的设计能力;第四:提高大家持之以恒，永不言败的精神。我想我基本做到了，在这里也真心感谢教员的良苦用心。总之，本学期的C#课程对我来说意义很大，相信经过了此次学习一定会为我以后的学习生活产生巨大帮助。

吴洋：这是我第一次参加这么巨大的项目，这次实验给我留下了很深的印象。

刚开始我们组长给我分配任务时我感觉很难，做的时候出现很多错误而且我是照着资料书打代码还有错误，那时的感觉是迷茫的。后来好不容易调试出来了，拿给组长看却不符合要求，我只能是从头再来。后来经过网上查找资料，和组长组员的讨论之下最终达到了要求。

无论是学习还是做项目的过程中，耐心都要非常重要。有时出现错误，上网查资料，自己又想了很久，又找组员讨论还是不能解决时就会有不想做的念头。在后来的探讨中就慢慢的喜欢上这个项目，觉得它很有趣，当在做项目成功的实现它功能时，心理就特别高兴，就好想再做几个功能。但是事实不是一下就能成功的实现它的功能所以耐心非常重要。

最重要的一点是我们做项目时，我们要有团队的合作精神。只有这样我们才能很好的完成一个好的项目。没有一个共同的工作目标一切都很难实现，集体主义的具体表现就是团队精神，在我们工作中离不开团队精神的指引，团队离不开一个统一的领导和统一的指挥，才能保证我们工作的顺利开展和成绩的取得，这就是团队精神的巨大的作用。我最大的愿望就是几个想学习的同学在一起讨论问题，只有这样才能进步更快，才能学到更多的知识。

何祖鑫：心得体会：经过本学期的学习，较上学期而言，自己接触的更加深入了。谈到总结与收获，我想大致几方面：

1. 自己团队协作能力有了很大的提高。此次课程的，由汪聪作为总工程师，我们其余三人各自负责自己的模块完成。在实践之前，我们在协议选择，算法选择，都经过了大量的讨论，在充分考虑安全强度与运行效率，以及实现可行性等综合因素，我们团队作出了最优选择。在这其中，我非常感谢其余组员给予自己的帮助。

2. 动手实践能力有了提高。虽然此次课程的完成，代码部分主要由其余同学负责完成，但是自己在这其中也有跟进，主要是协议实现上，虽然代码量很少，但是对于C#的编译有了一定的新认识，比如：以前C语言时候，对密码算法的实现需要自己编写，而在C#中却只需要引用相应的类就可以了。

3. 对理论知识到应用层的实现，有了深入认识。本学期的考核，我们团队所需要的完成的是，对大文件的安全传输。 以前，自己对于信息安全的一些技术，只是停留在理论学习。而在应用上的实现，却是从未有过尝试。尤其是在此次两个协议的设计中，MD5算法结果为16字节，而AES加密却是需要32字节，当时就这么一个简单的问题，自己却是难以解决，最后汪聪却只是简单的作了一个转化就完成了。这对自己的触动十分的大。