《密码学》课程设计

密码学课程组*

2024 年 6 月 27 日

[1]课程主页:

http://uisu.gitee.io/lxf/moderncrypto.html

https://yes-uisu.github.io/lxf/moderncrypto.html

目录

1 课程任务

1.1 任务概述

设计一个点对点的 IP 网络保密通信软件，此软件可以按 client 模式启动，也可以按 server 模式启动，如果按 client 模式启动，需要输入其所要连接的 server 端 ip 地址，如果按照 server 端启动，则不需要。通信采用 TCP 连接的方式，使用端口号为 27013。

此程序使用方法为：

secretsender options1 [ip-address] [options2] [data]

options1 :

- -c :c 表示 client，表示程序按 client 模式启动，[ip-address] 表示程序启动后连接的 server 的 ip，连接成功后将 [data] 发给 server，并且将发送内容的字符数和内容在屏幕上输出，如果数据需要加密传输，还需输出加密后的数据 (十六进制的方式)。
- -s :s 表示 server，表示程序按 server 模式启动。按 server 模式启动时，无需输入 [ip-address] 和 [data]。当有 client 端连接 server 后，server 会将 client 发送来的字符串的内容在屏幕上输出，如数据加了密，则需解密输出。

当 options1 参数为 c 时，才会有 [ip-address] [options2] [data] 参数。

[ip-address] 是 server 端的 IPv4 地址，如果 server 和 client 在同一台机器上运行，ip 地址为 127.0.0.1。

[options2] :

- -n :n 表示 none，表示发送的数据并没有加密，按照明文发送。如果是 server 端，则无需此参数。
- -m :m 表示发送的数据加密发送。如果是 server 端，则无需此参数。

[data] 是 client 端要发送给 server 的明文字符串。如果是 server 端，则无需此参数。

1.2 程序实现的其他要求

1. 程序使用 C 语言实现。
2. 在进行加密传输时，密钥交换使用 Diffie-Hellman 密钥交换协议。此程序只是示例，并不考虑其安全强度问题，所以 Diffie-Hellman 的共享参数取的很小，分别为 $p = 23, a = 5$。
3. 双方协商后获得密钥为 K,我们使用移位加密 $c = m+K'(mod\, CodeSpace)$，移位加密的密钥 $K' = K(mod\ p)$，$CodeSpace$ 是编码空间的大小，如果编码不是从 0 开始，注意加密前要进行一个简单的对齐映射，解密后还要映射到明文空间。
4. 明文空间为 ASCII 码，我们不考虑 ASCII 码最后一个"DEL" 不可见字符编码，取 $CodeSpace = 127$。
5. 为了统一要求，可发送的字符数 (包括空格) 应不少于 300。
6. 程序输出的提示信息要求，见测试用例，需在相同情况下，与测试用例保持一致。

1.3 应用层通信协议

1.3.1 协议设计

我们虽然已经明确了使用 Diffie-Hellman 来进行密钥的协商，但是在代码实现前还是需要进行细化设计，图1是我们给的一个协议设计示例 (这个示例只是一个示范，大家可以在实现时设计你自己的协议，例如可以把发送的数据也设计到此协议中)，此协议只是用来做密钥协商的，在传输数据时，直接使用 Socket 进行传输。

协议说明：

- 公共参数: 就是传输的 $p = 23, a = 5$。
- 确认公共参数: 发送此数据包表示收到发来的公共参数。
- Y_A:$Y_A = a^A mod\ p$
- Y_B:$Y_B = a^B mod\ p$
- Bye: 表示结束此次应用层会话。

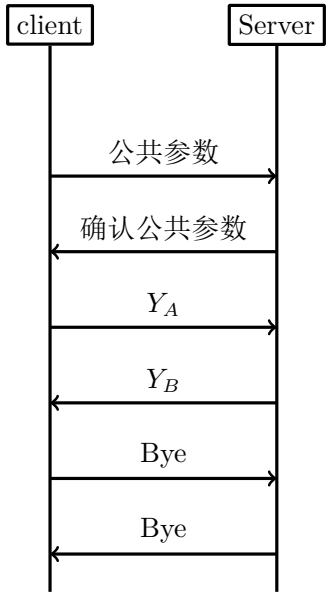

图 1: Diffie-Hellman 应用层协议设计

我们可以把公共参数写到程序中，也可以配置参数的形式写到配置文件中，但是这两种方法都不够灵活，所以在此我们采用通信中双方传输公共参数的方式。

1.3.2 数据结构设计

协议的 C 语言参考定义为：

```
1
2         #define MsgVersion 1
3         enum MsgType{
4                 SENDPAR=1,
5                 PAROK,
6                 MyNum,
7                 Bye
8         }
9         typedef struct _DH_PROTOCOL
10        {
```

```
11                  unsigned char version; //版本
12                  enum MsgType type;    //消息类型
13                  unsigned short p; //参数p
14                  unsigned short a; //参数a
15                  unsigned short YAB; //共享给对方的数值
16          }DH_Protocol;
```

关于数据结构如何使用的参考示例代码见https://gitee.com/uisu/C_samplecode/tree/master/DUstruct,或,https://github.com/yes-uisu/C_samplecode/tree/master/DUstruct。

1.4 再说明

1.4.1 需求分析——需要足够重视

对软件需求做出一个清晰、准确、详尽的描述，使得需求方和实现方能够达成共识，是软件设计中非常重用的一个环节，在软件工程中我们称为需求分析。

在我们以往的练习中，我就如同一个需求方，给大家描述你们程序要做什么，但是大家也应该已经感受到了，要想最终达成“我和你”对于这个软件完成后应该是什么样子，对于这么小的软件，你在具体实现中，就会发现很多需求不明确，所以说清楚需求不是件容易的事情。

这里的“不容易”是多方面的，比如我们对描述这件事情所用的“术语系统”首先要统一，才好交流，这很类似客户具有很强的业务背景，而我们的产品经理除了具备系统知识 (因为他要和开发人员交流，把客户需求传递给开发人员)，同时还要有强的业务背景的原因。所以我们可以看到，在软件工程的课程里面有专门的需求分析的讲解，同时也有专门的软件需求分析课程、专著以及专用软件，有兴趣的同学可以去了解了解。

所以不论采用什么样的方法和方式，最终不要忘记“需求分析”要解决的问题是什么——“把要做得事情无歧义、完整地描述出来”。

1.4.2 需求的一种描述方法——以明文发送用例为例

为了让大家对此任务的需求有个直观了解，下面我给出程序执行时几种情况示例 (这个类似原型法的需求描述方式)。

客户端和服务端在同一个操作系统上启动[1]。

首先启动服务端，执行过程为：

```
>secretsender -s
waiting connected...
```

服务端启动后等待客户端的连接。

然后以发送明文方式启动客户端，并且准备在连接成功后给服务端发送字符串“student”，执行过程为：

```
>secretsender -c 127.0.0.1 -n ”student”
connect 127.0.0.1: ok
send data <student> : ok
```

在启动客户端后，客户端首先连接服务端，成功后显示 ok，然后发送明文数据给服务端，发送成功后显示 ok，然后客户端程序终止。服务端在整个过程中会有新的输出，服务端的界面表现为：

```
>secretsender -s
waiting connected...ok
received data <student>
```

1.4.3 需求到代码的设计

需求和你的代码实现相比，她是系统高一层的抽象。这句话怎么理解，我们举个例子。

比如此课程设计中要求“客户端在执行时，能够把以参数形式传给程序的字符串，发送给服务端 (我们先假设以明文传送)”。

对于这个需求，你要设计他的具体实现，很多同学觉得在写设计时没有什么好设计的，感觉需求不都说了吗，其实你在设计代码实现这个需求实现时，你是要有很多代码层需要考虑的问题，你没有意识到这个设计过程的存在，你觉得没什么好写的，有时候是因为你没有意识到可以有不同的实现方法、路径，而每种实现路径或方法，都是有其优劣的，你在设计时，是需要去判断、思考、取舍，下面我们简单分析这个需求，理解以上文字。

1. 以参数形式传到程序中的字符串，需要存到一个字符串数组里，显然这个字符串是变长的，那么我们是定义一个动态变长的？还是静态的？

[1]刚开始写的时候准备写“在同一台物理机上”，但是觉得这样不够严谨，因为有些同学可能装了虚拟机，如果一个在宿主机上启动，一个在虚拟机上启动，显然在网络上，这是两个主机，IP 地址是不一样的，所以就改为” 在同一个操作系统上” 了。

假设在这一步选择静态的，那好，这个静态数组长度一旦确定，是不是就限制了你系统传输信息在长度方面的能力？那定多长，既能满足用户需求[2]，也能不浪费系统太多资源。

2. 在确定好可以传输的最大字符串长度后，仅接着你要明确，socket 是否可以传输这么长的字符串？怎知道 socket 的 send 能传多长的字符串？可以看 send 的定义[3]，从这个定义我们可以看到可传输的最长数据应该 int 的最大值？确实是这样吗？因为微软的库不开源，所以我们看不到 send 具体实现，只能从定义上推测，好了，如果你觉得是这样，就这样写了，也许就为你的程序埋下一个 bug。因为可能 send 在具体实现时不是按 int 的最大值去实现的，当然，这只是一个可能，目前你并不能排除这种可能的存在，我们假设这种可能发生了，这就是一个隐藏的 bug，如果你的测试用例考虑了这种边界条件，这个问题可以在测试阶段发现，如果没有考虑这样的测试用例，那么这个 bug 就带入了程序的发布版。当然你也可以在设计阶段排除掉上面说的那种可能，你只要编写一段程序就能验证在你的那个开发环境下，是否存在这个发送长度的限制，这也就是我们在设计一个系统时，为什么要去写很多小的程序去验证一些函数和一些方法的原因之一。

3. 假设 client 端通过 socket 发送这么长的字符串没有问题了，那么 server 端接收是否有问题？同样，如果你在 server 端也选择静态数组方式来存储接收到的字符串，这个字符串定义多大，你是要根据 client 的实现来定义的？如果定义小了，也是一个隐藏的 bug，如果定义大了造成浪费。

4. 再多说一点，我们从 client 和 server 端在这个数组定义上要一致，

[2]如何确定是否满足了客户需求？产品经理是要去和客户去沟通确认的，需求必需经过客户认可，否则，而且在过程管理中，通常是需要客户签字确认的，因为需求的变更将会使得开发成本不可控 (时间进度、人员等待)

[3]int WSAAPI send(SOCKET s, const char *buf, int len, int flags);

s：A descriptor identifying a connected socket.

buf：A pointer to a buffer containing the data to be transmitted.

len：The length, in bytes, of the data in buffer pointed to by the buf parameter.

flags：A set of flags that specify the way in which the call is made. This parameter is constructed by using the bitwise OR operator with any of the following values.

Return value:If no error occurs, send returns the total number of bytes sent, which can be less than the number requested to be sent in the len parameter. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code can be retrieved by calling WSAGetLastError.

可以体会代码组织架构的设计，你可以体会为什么通常的 IDE 会有 workspace 和 project，为什么你看到一些参考代码会有很多常量宏的定义放在头文件里。

1.4.4 概要设计，详细设计——设计本身的层层深化

我们先从需求看，需求中要求可以选择加密传输，针对这个需求，设计者要考虑用什么加密方法？密钥怎么传？

我们作为设计者，在没有充分考虑加密强度或者安全性的前提下，选择了移位加密的方法，并且密钥的协商采用 Diffie-Hellman 协议。第一层设计完成。

基于第一层的设计，我们接着要再去深化或细化考虑。

Diffie-Hellman 的公共参数怎么共享？我们采用在网络上传输，当然你还可以有很多种选择，比如公共参数是两个确定的数，公共参数存在在系统的配置文件中，公共参数可以从一个共享服务器上获得，等等，每种都有其优缺点，选择每一种，你后续的深化设计都会面临不同的问题。

我们选择在网上传输，并且传输的是两个数，那么就会涉及到如何设计传输的数据结构 (其实一个数也涉及到数据结构)，结合整个交互过程会有不同类型的消息，综合考虑，需要详细的定义应用层协议的数据结构。

到这里，大家应该能够体会到在系统设计时，在需要做信息交换时，标准定义的重要性了，如果没有一个标准的定义，是无法达到互联互通的，甚至虽然标准定义了，但是由于实现的复杂度很高等原因，这个标准也很难推广，实际种不乏这样的列子。

2 验收测试用例

首先需要明确的是，这个测试用例是老师在验收时需要执行的测试用例，并没有要求你们 (程序设计者) 也用这个测试用例来验证你们的实现是否正确，通常来说，测试者和代码设计者会独立去设计各自的测试用例，对于一些大型项目，会利用自动化测试工具，当有内容变化时，需要自动执行这些测试用例来验证修改对于项目的改变是正向可控的。

在我们程序验收时，给出的这些测试用例是测试的基线测试用例。

另外，我们要注意到，当测试人员拿到需求时，其已经可以设计系统级的测试用例。在概要设计阶段和模块设计阶段，又为测试人员设计子系统、

模块黑/白盒测试用例提供依据。

2.1 用例编号说明

本任务书中的所有用例编号形式为：TC-**-@@，其中 TC 表示 test cases，** 是所测功能编号，比如 SP 表示“发送明文”的功能（send plaintext），@@ 表示此功能测试的序号，比如 1，表示 1 号测试用例。

本文档测试用例涉及到的功能有框架、发送明文和发送密文，功能编号分别为 FR、SP 和 SC。

2.2 框架验收

在框架源程序中，除了有主函数外，还应有根据程序功能设计的函数，并且不少于三个函数。函数应该按要求有注释，并且在程序中有调用。

框架程序只需实现根据命令行参数进行的程序执行过程的控制逻辑，并不需要有具体功能实现，但是在注释部分要说明此处执行的功能。

框架程序可以编译为可执行程序，根据输入的参数，能够进行正确的流程控制和函数[4]调用，并能通过信息的输出，证明过程的正确性。

2.2.1 用例名：无参数 (编号：TC-FR-1)

在终端中输入及显示：

```
>secretsender
Parameter error.
>
```

2.2.2 用例名：启动服务端 (编号：TC-FR-2)

在终端中输入及显示：

```
>secretsender -s
waiting connected...
```

程序处于循环等待中，可以在任务管理器中关闭此程序。

[4]这里的函数只要定义出来，打印输出相应的测试信息即可

2.2.3 用例名：客户端明文传输 (编号：TC-FR-3)

在终端中输入及显示：

```
>secretsender -c 127.0.0.1 -n ”abc”
connect 127.0.0.1: ok
Data length:3
send data <abc> : ok
>
```

2.2.4 用例名：客户端密文传输 (编号：TC-FR-4)

在终端中输入及显示：

```
>secretsender -c 127.0.0.1 -m ”ab”
connect 127.0.0.1: ok
key=3
Data length:1
Plain <ab>
Cypher(hex) <64 65>
send data Hex: <64 65> : ok
>
```

注意 key=3 这个可以在程序中直接给表示协商后密钥的变量赋值 3。

此处要求能够实现根据密钥值进行加解密操作，但是在这个测试用例里面，只测试了加密。

2.3 发送明文

2.3.1 用例名：发送一个字符 (编号：TC-SP-1)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
received data <a>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -n ”a”
connect 127.0.0.1: ok
Data length:1
send data <a> : ok
>
```

2.3.2 用例名：发送 3 个字符 (编号：TC-SP-2)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
received data <abc>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -n ”abc”
connect 127.0.0.1: ok
Data length:3
send data <abc> : ok
>
```

2.3.3 用例名：发送字母数字 (编号：TC-SP-3)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
received data <abc908ww>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -n ”abc908ww”
connect 127.0.0.1: ok
Data length:8
send data <abc908ww> : ok
>
```

2.3.4 用例名：发送长串 (编号：TC-SP-4)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
received data <when day is done. If the day is done. If birds sing to more. If the wind has flagged tired. Then draw the veil of darkness thick upon me.>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -n ”when day is done. If the day is done. If birds sing to more. If the wind has flagged tired. Then draw the veil of darkness thick upon me.”
connect 127.0.0.1: ok
Data length:137
send data <when day is done. If the day is done. If birds sing to more. If the wind has flagged tired. Then draw the veil of darkness thick upon me.> : ok
>
```

2.3.5 用例名：发送 0 长串 (编号：TC-SP-5)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后的完整显示：

```
>secretsender -s
waiting connected...
```

服务端一直处于等待连接状态。

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -n ""
Error:Data length is zero.
>
```

提示参数错误后，客户端退出。

2.4 发送密文

假设密文发送时协商计算而来的密钥为“3”，加密过程为 $c = m + 3 \pmod{127}$。

我们在设计一些测试用例的时候，可以利用其他工具生成我们的测试数据，比如，本用例中加密后的十六进制数据，假设密钥为 3，可以利用 sagemath 生成：

```
a="abc"
t="Hex:"
for i in a:
    c=mod(ord(i)+3,127)
    t=t+hex(int(c))[2:]
print(t)
```

也可以利用原生 python 生成：

```
a="abc"
t="Hex:"
m="Plain:"
for i in a:
    c=ord(i)+3 % 127
    m=m+hex(int(ord(i)))[2:]
    t=t+hex(int(c))[2:]
print(m)
print(t)
```

2.4.1 用例名：发送一个字符 (编号：TC-SC-1)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
key=3
received cipher(hex) <64>
plain text <a>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -m ”a”
connect 127.0.0.1: ok
key=3
Data length:1
Plain <a>
Cypher(hex) <64>
send data Hex: <64> : ok
>
```

2.4.2 用例名：发送 3 个字符 (编号：TC-SC-2)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
key=3
received cipher(hex) <64 65 66>
plain text <abc>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -m ”abc”
connect 127.0.0.1: ok
key=3
Data length:3
Plain <abc>
Cypher(hex) <64 65 66>
send data Hex:<64 65 66> : ok
>
```

2.4.3 用例名：发送字母数字 (编号：TC-SC-3)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
key=3
received cipher(hex) <64 65 66 3c 33 3b 7a 7a>
plain text <abc908ww>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -m ”abc908ww”
connect 127.0.0.1: ok
key=3
Data length:8
Plain <abc908ww>
Cypher(hex) <64 65 66 3c 33 3b 7a 7a>
send data Hex:<64 65 66 3c 33 3b 7a 7a> : ok
>
```

2.4.4 用例名：发送长串 (编号：TC-SC-4)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端启动后，等待连接，客户端启动发送数据后，服务端的完整显示：

```
>secretsender -s
waiting connected...ok
key=3
received cipher(hex) <7a 6b 68 71 23 67 64 7c 23 6c 76 23 67 72 71
68 31 23 4c 69 23 77 6b 68 23 67 64 7c 23 6c 76 23 67 72 71 68 31
23 4c 69 23 65 6c 75 67 76 23 76 6c 71 6a 23 77 72 23 70 72 75 68
31 23 4c 69 23 77 6b 68 23 7a 6c 71 67 23 6b 64 76 23 69 6f 64 6a
6a 68 67 23 77 6c 75 68 67 31 23 57 6b 68 71 23 67 75 64 7a 23 77
6b 68 23 79 68 6c 6f 23 72 69 23 67 64 75 6e 71 68 76 76 23 77 6b
6c 66 6e 23 78 73 72 71 23 70 68 31>
plain text <when day is done. If the day is done. If birds sing to
more. If the wind has flagged tired. Then draw the veil of darkness
thick upon me.>
>
```

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -m ”when day is done. If the day is done. If birds sing to more. If the wind has flagged tired. Then draw the veil of darkness thick upon me.”
connect 127.0.0.1: ok
Data length:137
Plain <when day is done. If the day is done. If birds sing to more. If the wind has flagged tired. Then draw the veil of darkness thick upon me.>
Cypher(hex) <7a 6b 68 71 23 67 64 7c 23 6c 76 23 67 72 71 68 31 23 4c 69 23 77 6b 68 23 67 64 7c 23 6c 76 23 67 72 71 68 31 23 4c 69 23 65 6c 75 67 76 23 76 6c 71 6a 23 77 72 23 70 72 75 68 31 23 4c 69 23 77 6b 68 23 7a 6c 71 67 23 6b 64 76 23 69 6f 64 6a 6a 68 67 23 77 6c 75 68 67 31 23 57 6b 68 71 23 67 75 64 7a 23 77 6b 68 23 79 68 6c 6f 23 72 69 23 67 64 75 6e 71 68 76 76 23 77 6b 6c 66 6e 23 78 73 72 71 23 70 68 31>
send data Hex:<7a 6b 68 71 23 67 64 7c 23 6c 76 23 67 72 71 68 31 23 4c 69 23 77 6b 68 23 67 64 7c 23 6c 76 23 67 72 71 68 31 23 4c 69 23 65 6c 75 67 76 23 76 6c 71 6a 23 77 72 23 70 72 75 68 31 23 4c 69 23 77 6b 68 23 7a 6c 71 67 23 6b 64 76 23 69 6f 64 6a 6a 68 67 23 77 6c 75 68 67 31 23 57 6b 68 71 23 67 75 64 7a 23 77 6b 68 23 79 68 6c 6f 23 72 69 23 67 64 75 6e 71 68 76 76 23 77 6b 6c 66 6e 23 78 73 72 71 23 70 68 31> : ok
>
```

2.4.5 用例名：发送 0 长串 (编号：TC-SC-5)

环境要求：此用例执行时，服务端和客户端应该启动在同一个操作系统中。

服务端输入及显示：

```
>secretsender -s
waiting connected...
```

服务端一直处于等待连接状态。

客户端输入及显示：

```
>secretsender -c 127.0.0.1 -m ""
Error:Data length is zero.
>
```

提示参数错误后，客户端退出。

3 程序提交要求

原始仓库地址`https://gitee.com/uisu/secretsender`，这是一个公开的开源仓库，首先根据发送的链接获得仓库权限，然后 fork 仓库。

按阶段提交程序，特别注意不要提交多了，比如提交程序框架时，并不需要具体实现明文发送，如果实现了明文发送，反而在 code review 时是扣分项。

每次提交时，最后更新到你的仓库后，首先要按要求根据不同阶段打一个“标签”，在打标签时，系统同时会形成当前仓库的一个压缩包，打完标签后，可以向我发送 PR 请求，注意在请求中说明你是交哪一个阶段的程序，我会直接下载你的“标签”对应的压缩包，所以只要你打完标签，就可以更新你的仓库了，不会影响我下载的内容。

特别强调，没有 Tag，没有 PR 均视为未交此阶段作业!!!

3.1 程序注释要求

请按照理论课编程中的注释要求进行代码注释。注释分数会在 code review 分数中体现。

4 文档要求

文档格式见“现代密码学课程设计报告模板”，有 MS word 和 wps 两种模板。文档总分为 30 分，评分规则见表1。

表 1: 文档评分规则

评分项	分值	A(5 分)	B(2∼ 4)	C(0 ∼ 2)
设计需求	5	清晰准确完整	较完整	描述不清
设计与实现	10	清晰准确完整	较完整	描述不清
测试	5	清晰准确完整	较完整	描述不清
抓包分析	5	清晰准确完整	较完整	描述不清
总结	5	清晰准确完整	较完整	描述不清

表 2: 验收阶段及标准

阶段	截止日期	验收内容	分值	超时处理	备注
P1	第二天 10:00	框架验收	10	超时满分降为 5	Tag 为 v1.0
P2	第三天 10:00	发送明文	30	超时满分降为 25	Tag 为 v2.0
P3	第四天 8:00	发送密文	30	超时满分降为 25	Tag 为 v3.0
P4	第六天 8:00	文档	30	超时 0 分	

[1] 作业的提交时间均以收到 gitee 上的 Pull Request 为准。

[2] 作业内容以从 gitee 上下载的为准。

5 评分标准

评分分为四个阶段，每个阶段验收要求见表2。

在发送明文和发送密文验收时，每个用例执行正确得 5 分，代码 Review 5 分。这是基线评估，然后会去做一些重点检查，作为扣分项，比如代码文件、函数的描述等和一些其他的程序测试用例。

6 问题引导示例

需求： **编译后的程序是个命令行程序，执行时需要带参数。**

问题引出：

1. 如何编写带参数的命令行程序？如何获得这些参数？
2. 如何解析判读这些参数？
3. 如何获得要发送的字符串，并且去掉“”？

7 参考信息

7.1 TCP/IP 示例程序

参考示例代码见https://gitee.com/uisu/C_samplecode/tree/master/TCPSenRev。

7.2 Wireshark

Wireshark 是一款常用的网络数据包分析软件，网站为https://www.wireshark.org, 软件可以免费下载使用。本课程设计会用这款软件进行数据包的分析。

7.3 Rawcap

引入此抓包程序的原因是 wireshark 在利用 winpcap 无法抓取本地环路的数据包，Rawcap 网址https://www.netresec.com/index.ashx?page=RawCap。如何实时将 Rawcap 抓取的数据包送入 wireshark 进行处理，可以参见 Rawcap 网站上的说明。在设计完此任务后，我在进行任务验证时使用的方法是用 rawcap 存为 pcap 文件，然后用 wireshark 打开分析。