



南開大學
Nankai University

南 开 大 学

计 算 机 学 院

网络安全技术实验报告

基于 DES 加密的 TCP 聊天程序

朱浩泽 1911530

年级：2019 级

专业：计算机科学与技术

班级：计算机科学与技术 2 班

2022 年 3 月 22 日

目录

一、 实验目的	1
二、 实验内容	1
三、 实验步骤及实验结果	1
(一) 实验步骤	1
1. 实验环境	1
2. 代码实现	1
(二) 实验结果	6
四、 实验遇到的问题及其解决方法	9
五、 实验结论	10

一、 实验目的

1. 理解 DES 加密原理
2. 理解 TCP 协议的工作原理
3. 掌握 Linux 下基于 socket 的编程方法

二、 实验内容

1. 利用 socket 编写一个 TCP 聊天程序
2. 通信内用经过 DES 加密与解密

三、 实验步骤及实验结果

(一) 实验步骤

1. 实验环境

macOS 12.3(基于 unix), C++11, Cmake

2. 代码实现

- DES 部分包括两个文件 DES.h 和 DES.cpp 是对于 DES 模块的声明与定义。在这一部分，我们定义了 CDesOperator 类

```
1 class CDesOperate{
2 private:
3     //recursive key 16
4     ULONG32 m_arrOutKey[16][2];
5     //initial key
6     ULONG32 m_arrBufKey[2];
7
8     //execute whole action
9     INT32 HandleData(ULONG32 *left, ULONG8 choice);
10    //execute 16 round without IP
11    INT32 MakeData(ULONG32 *left, ULONG32 *right, ULONG32 number);
12    //generate 1 recursive key
13    INT32 MakeKey(ULONG32 *keyleft, ULONG32 *keyright, ULONG32 number);
14    //generate 16 recursive key
15    INT32 MakeFirstKey(ULONG32 *keyP);
16
17 public:
18     CDesOperate();
19     INT32 Encry(char *pPlaintext, int nPlaintextLength,
20 char *pCipherBuffer, int &nCipherBufferLength, char *pKey, int nKeyLength);
21     INT32 Decry(char *pCipher, int nCipherBufferLength,
22 char *pPlaintextBuffer, int &nPlaintextBufferLength, char *pKey, int
    nKeyLength);
23 };
```

其中, m_arrOutKey 中存储生成的子密钥, m_arrBufKey 存储原始密钥。

makeFirstKey: 生成初始密钥并通过调用 makeKey 生成所有密钥, 其主要思想是将有效的 56bit 进行置换选择, 结果等分为左右各 28bit, 再进行循环左移, 左移后将两个部分合并为 56bit, 再从中选 48bit 作为此轮迭代的密钥, 共生成 16 个 48 位的密钥。其主要代码如下:

```

1 INT32 CDesOperate::MakeFirstKey(ULONG32 *keyP) {
2     uint32_t tempKey[2]={0};
3     uint32_t*pFirstKey=(uint32_t*)m_arrBufKey;
4     uint32_t*pTempKey=(uint32_t*)tempKey;
5     memset((uint8_t*)m_arrBufKey, 0, sizeof(m_arrBufKey));
6     memcpy((uint8_t*)&tempKey, (uint8_t*)keyP,8);
7     memset((uint8_t*)m_arrOutKey, 0, sizeof(m_arrOutKey));
8     for(int j = 0; j < 28; j++) {
9         //循环28次 64---->56 但还是要用2个32位来存储
10        if(keyleft[j] > 32)
11        {
12            //第一个32位
13            if(pTempKey[1]&pc_by_bit[keyleft[j]-1]) {
14                //第一次出现这种pc_by_bit[],此后涉及到选取特定的位都将用到
15                pFirstKey[0] |= pc_by_bit[j];
16                //其实原理很简单 先判断一下要选取的bit数组对应的位是否为1
17            }
18            //通过与上0x80000000(1000 0000 0000 0000...)等只有一bit为1的数即可判断
19        }
20        //再将相应的位 置1通过或上0x80000000(1000 0000 0000 0000...)等只有一bit为1的数即可
21        else {
22            if(pTempKey[0] & pc_by_bit[keyleft[j] - 1])
23            {
24                pFirstKey[0] |= pc_by_bit[j];
25            }
26        }
27        if(keyright[j] > 32) {
28            //第二个32位
29            if(pTempKey[1] & pc_by_bit[keyright[j] - 1]) {
30                pFirstKey[1] |= pc_by_bit[j];
31            }
32        }
33        else {
34            if(pTempKey[0] & pc_by_bit[keyright[j] - 1])
35            {
36                pFirstKey[1] |= pc_by_bit[j];
37            }
38        }
39    }
40    for(int j = 0; j < 16; j++) {
41        MakeKey(&pFirstKey[0],&pFirstKey[1],j); //firstKey已形成, 循环调用
42        oneStepOfMakeSubKe()形成子密钥
43    }
44    return SUCCESS;
45 }

```

HandleData 是数据加密或解密, MakeData 是数据加密或解密的每轮迭代。这两个函数是 DES 数据加密运算的主要部分, 也分为 16 轮迭代, 其主要过程是先将明文分成 64bit 的

数据块，不够 64 位的用 0 补齐；每一轮中对每一个 64bit 的数据块，首先进行初始换位，并将数据块分为 32bit 的两部分；保持左部不变，将右部由 32 位扩展为 48 位与该轮的密钥进行异或操作；对新的 48 位进行压缩操作（S 盒）输出 32 位的压缩后的数据；对新的 32 位的数据进行一次置换操作；把左右部分进行异或作为右半部分，最原始的右边作为左半部分，最后进行逆初始置换。其主要实现代码如下：

```

1 INT32 CDesOperate::HandleData(ULONG32 *left, ULONG8 choice) {
2     uint32_t *right = &left[1] ;
3     uint32_t tmpbuf[2] = { 0 };
4     for (int j = 0 ; j < 64 ; j++)
5     {
6         if (j < 32)
7         {
8             if (pc_first[j] > 32)
9             {
10                if (*right & pc_by_bit[pc_first[j]-1])
11                {
12                    tmpbuf[0] |= pc_by_bit[j] ;
13                }
14            }
15            else
16            {
17                if (*left & pc_by_bit[pc_first[j]-1])
18                {
19                    tmpbuf[0] |= pc_by_bit[j] ;
20                }
21            }
22        }
23        else
24        {
25            if (pc_first[j] > 32) {
26                if (*right & pc_by_bit[pc_first[j]-1]) {
27                    tmpbuf[1] |= pc_by_bit[j] ;
28                }
29            }
30            else {
31                if (*left & pc_by_bit[pc_first[j]-1]) {
32                    tmpbuf[1] |= pc_by_bit[j] ;
33                }
34            }
35        }
36    }
37    *left = tmpbuf[0];
38    *right = tmpbuf[1];
39    tmpbuf[0]=0;
40    tmpbuf[1]=0; //重新置零!
41
42    switch (choice)
43    {
44    case 0:
45        for(int num=0;num<16;num++)//16轮迭代,加密
46        {
47            MakeData(left,right,(uint32_t)num);
48        }
49        break;

```

```

50 case 1:
51     for(int num=15;num>=0;num--)//16轮迭代, 解密
52     {
53         MakeData(left,right,(uint32_t)num);
54     }
55     break;
56 default:
57     break;
58 }
59
60 INT32 temp;
61 temp = *left;
62 *left = *right;
63 *right = temp;//交换左右!
64
65 for (int j = 0 ; j < 64 ; j++) {
66     if (j < 32 )
67     {
68         if ( pc_last[j] > 32) {
69             if (*right & pc_by_bit[pc_last[j]-1]) {
70                 tmpbuf[0] |= pc_by_bit[j] ;
71             }
72         }
73         else {
74             if (*left & pc_by_bit[pc_last[j]-1]) {
75                 tmpbuf[0] |= pc_by_bit[j];
76             }
77         }
78     }
79     else {
80         if (pc_last[j] > 32) {
81             if (*right&pc_by_bit[pc_last[j]-1]) {
82                 tmpbuf[1] |= pc_by_bit[j];
83             }
84         }
85         else {
86             if (*left&pc_by_bit[pc_last[j]-1]) {
87                 tmpbuf[1] |= pc_by_bit[j] ;
88             }
89         }
90     }
91 }
92 *left = tmpbuf[0] ;
93 *right = tmpbuf[1];
94
95 return true;
96 }

```

在 Encry 函数中调用各个函数对 DES 加密和 DES 解密。

```

1 INT32 CDesOperate::Encry(char *pPlaintext, int nPlaintextLength, char *
  pCipherBuffer, int &nCipherBufferLength, char *pKey, int nKeyLength) {
2     //首先检查初始密钥长度, 若正确, 则创建 16 轮迭代的密钥。
3     if(nKeyLength != 8) {
4         return 0;
5     }

```

```

6   MakeFirstKey((uint32_t *)pKey);
7
8   //由于加解密均要以 32bit 为单位进行操作,故需要计算相关参数,以确定加密的循环次数以及密文缓冲区是否够用,确定后将需要加密的明文格式化到新分配的缓冲区内。
9   int nLenthofLong = ((nPlaintextLength+7)/8)*2;
10  if(nCipherBufferLength<nLenthofLong*4) {
11      //out put buffer is not enough
12      nCipherBufferLength=nLenthofLong*4;
13  }
14  memset(pCipherBuffer,0,nCipherBufferLength);
15  uint32_t *pOutPutSpace = (uint32_t *)pCipherBuffer;
16  uint32_t * pSource;
17  if(nPlaintextLength != sizeof(uint32_t)*nLenthofLong) {
18      pSource= new uint32_t[nLenthofLong];
19      memset(pSource,0,sizeof(uint32_t)*nLenthofLong);
20      memcpy(pSource,pPlaintext,nPlaintextLength);
21  }
22  else {
23      pSource= (uint32_t *)pPlaintext;
24  }
25
26  //开始对明文进行加密,加密后将之前分配的缓冲区从内存中删除。
27  uint32_t gp_msg[2] = {0,0};
28  for (int i=0;i<(nLenthofLong/2);i++)
29  {
30      gp_msg[0] = pSource [2*i];
31      gp_msg[1] = pSource [2*i+1];
32      HandleData(gp_msg,(uint8_t)0);
33      pOutPutSpace[2*i] = gp_msg[0];
34      pOutPutSpace[2*i+1] = gp_msg[1];
35  }
36  if(pPlaintext!=(char *) pSource)
37  {
38      delete []pSource;
39  }
40
41  return SUCCESS;
42 }

```

- 基于 tcp 的客户端工作流程主要是先利用 socket 建立流式套接字,返回套接字号,然后利用 connect 函数,发送请求将套接字 s 与服务器连接,接着利用 send()/recv(),在 ns 上完成与服务器的加密数据交互,最后 close() 关闭套接字,其主要代码如下

```

1   std::cout << "Please input the server address:" << std::endl;
2   char strIpAddr[16];
3   std::cin >> strIpAddr;
4   int nConnectSocket, nLength;
5   struct sockaddr_in sDestAddr;
6   if((nConnectSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
7       perror("Socket");
8       exit(errno);
9   }
10  sDestAddr.sin_family = AF_INET;
11  sDestAddr.sin_port = htons(6060);
12  sDestAddr.sin_addr.s_addr = inet_addr(strIpAddr);

```

```

13 if(connect(nConnectSocket, (struct sockaddr*)&sDestAddr, sizeof(sDestAddr)) != 0)
14 {
15     perror("Connect");
16     exit(errno);
17 }
18 else {
19     std::cout << "Connect Success!" << std::endl;
20     std::cout << "Begin to chat.." << std::endl;
21     char *temp = "benbenmi";
22     SecretChat(nConnectSocket, strIpAddr, temp);
23 }
24 close(nConnectSocket);

```

- 基于 tcp 的服务端过程是首先利用 socket() 建立流式套接字，返回套接字号，通过 bind()，将套接字与本地地址绑定，开启 Listen()，服务器开始监听准备接受连接，服务器进入阻塞状态，循环等待客户端连接；建立连接后，accept() 返回新的套接字号 ns，send()/recv()，在 ns 上完成与客户端的加密数据交互，close() 关闭套接字 ns，其主要代码如下

```

1  std::cout << "Listening..." << std::endl;
2  int nListenSocket, nAcceptSocket;
3  socklen_t nLength;
4  struct sockaddr_in sLocalAddr, sRemoteAddr;
5  bzero(&sLocalAddr, sizeof(sLocalAddr));
6  sLocalAddr.sin_family = PF_INET;
7  sLocalAddr.sin_port = htons(6060);
8  sLocalAddr.sin_addr.s_addr = INADDR_ANY;
9  if ((nListenSocket = socket(PF_INET, SOCK_STREAM, 0)) == -1)
10 {
11     perror("socket");
12     exit(1);
13 }
14
15 if(bind(nListenSocket, (struct sockaddr*) &sLocalAddr, sizeof(struct sockaddr))
16     == -1) {
17     perror("bind");
18     exit(1);
19 }
20
21 if(listen(nListenSocket, 5) == -1) {
22     perror("listen");
23     exit(1);
24 }
25
26 nAcceptSocket = accept(nListenSocket, (struct sockaddr*) &sRemoteAddr, &nLength);
27 close(nListenSocket);
28 std::cout << "server: got connection from " << inet_ntoa(sRemoteAddr.sin_addr) <<
29     ", port " << ntohs(sRemoteAddr.sin_port) << ", socket " << nAcceptSocket <<
30     std::endl;
31 SecretChat(nAcceptSocket, inet_ntoa(sRemoteAddr.sin_addr), "benbenmi");
32 close(nAcceptSocket);

```

(二) 实验结果

行代码程序，打开服务端进行监听


```
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 6 !2 6
./main
Client or Server?
s
Listening...
```

客户端上线，连接成功

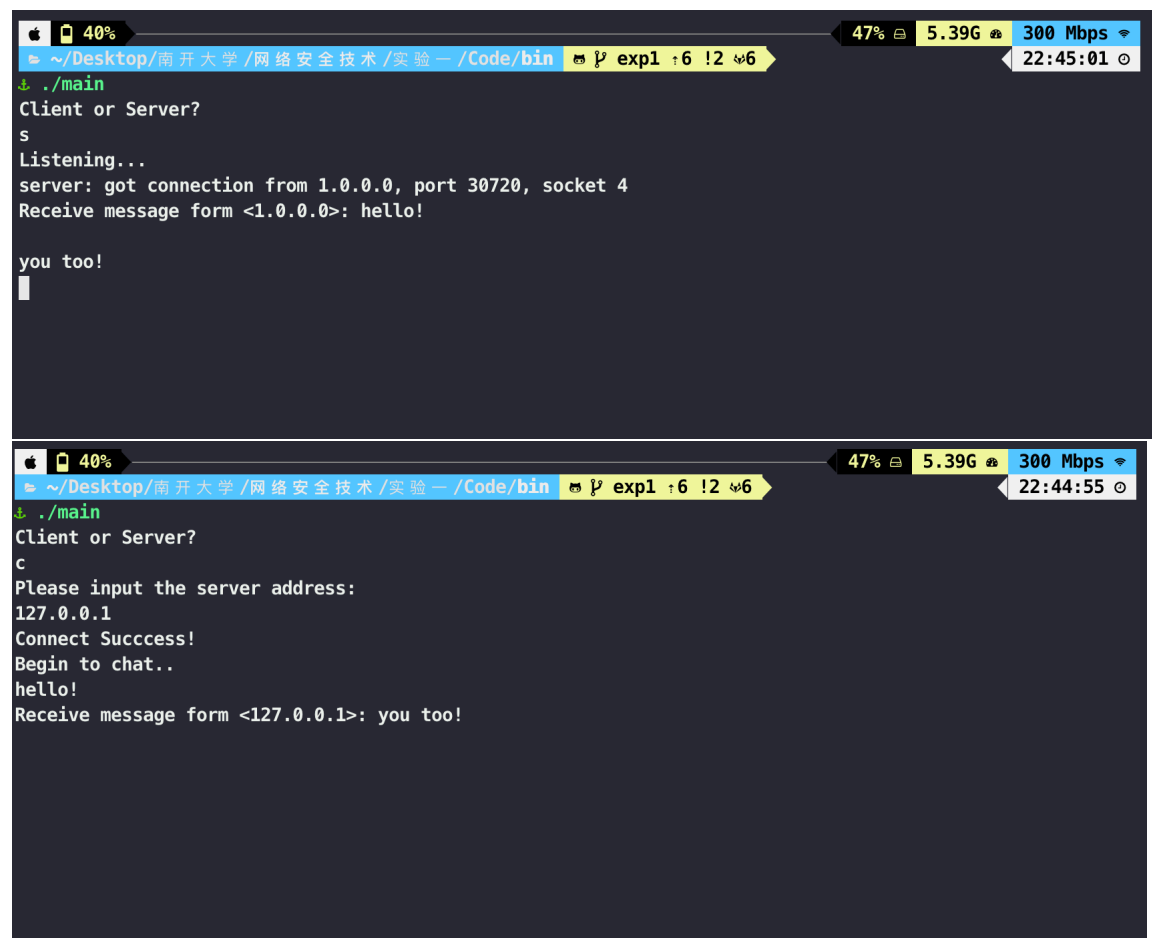
```
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 6 !2 6
./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Begin to chat..
```

客户端向服务端发送消息并成功收到

```
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 6 !2 6
./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Begin to chat..
hello!

~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 6 !2 6
./main
Client or Server?
s
Listening...
server: got connection from 1.0.0.0, port 30720, socket 4
Receive message form <1.0.0.0>: hello!
```

服务端向客户端发送消息并成功收到



```
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 +6 !2 6 47% 5.39G 300 Mbps 22:45:01
./main
Client or Server?
s
Listening...
server: got connection from 1.0.0.0, port 30720, socket 4
Receive message form <1.0.0.0>: hello!

you too!

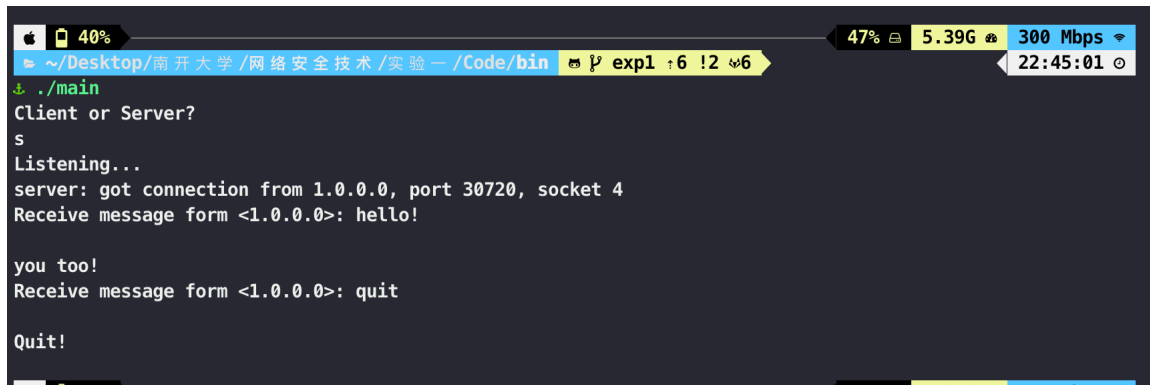
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 +6 !2 6 47% 5.39G 300 Mbps 22:44:55
./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Begin to chat..
hello!
Receive message form <127.0.0.1>: you too!
```

退出程序



```
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 +6 !2 6 47% 5.39G 300 Mbps 22:44:55
./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Begin to chat..
hello!
Receive message form <127.0.0.1>: you too!

quit
Quit!
```



```
~/Desktop/南开大学/网络安全技术/实验一/Code/bin exp1 ↑6 !2 ↵6 47% 5.39G 300 Mbps 22:45:01
./main
Client or Server?
s
Listening...
server: got connection from 1.0.0.0, port 30720, socket 4
Receive message form <1.0.0.0>: hello!

you too!
Receive message form <1.0.0.0>: quit

Quit!
```

四、 实验遇到的问题及其解决方法

首先是编译的问题，由于这次没有使用 Xcode 这种集成终端，而是选择完全在 iTerm 2 和 vim 上编写代码，所以在编译时也遇到了一定的问题，如何将多个文件编译成一个可执行文件，故学习使用了 Cmake 软件，由于项目架构如下

```
1  └─ include
2  |   └─ DES.h
3  |   └─ head.h
4  └─ src
5     └─ DES.cpp
6     └─ main.cpp
```

故编写 CmakeList.txt 文件如下

```
1  cmake_minimum_required (VERSION 3.0)
2
3  project (demo)
4
5  set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
6
7  aux_source_directory (src SRC_LIST)
8
9  include_directories (include)
10
11 # add_executable (main ${SRC_LIST})
12
13 add_executable(main ${PROJECT_SOURCE_DIR}/src/main.cpp ${PROJECT_SOURCE_DIR}/src/DES.
    cpp)
```

成功自动生成了 Makefile，通过 make 命令可以直接编译，make clean 可以删除可执行文件，编译问题得到解决。

其次是在传输时出现了乱码，这种问题在计算机网络课程的实验上也出现过，当时是在手动实现滑动窗口时出现过，但这次是直接使用 TCP 程序，所以传输不可能出问题，所以一定是出现在了加密解密算法之中。经过仔细的 debug，最终发现由于在单轮迭代中对左右两部分进行了交换，但实际上最后一轮是不需要交换的，所以在完成 16 轮迭代后应该再交换回来。这个问题导致了乱码，发现之后进行了修改，最终实验成功。

五、 实验结论

经过本次实验我了解了 DES 算法，DES 加密解密算法比较简单，实验指导书上给的也比较详细，按照指导书上的步骤很快便实现了。但是在目前学习的阶段，是第一次在非底层的实验上与比特位打交道，也是在区块链学习后第一次系统的接触密码学，并对这一领域有了更好的认识 and 了解。

这次实验也是第一次在 Linux 系统上进行 socket 编程，与 Windows 平台上区别不大，但是还是有略微的差别，但考虑到很多服务器也是基于 Linux 的，所以这一部分的学习实用性很强。其次，这是第一次在 Linux 系统上进行非集成终端或框架使用的编程，学会了使用 Cmake 工具生成 Makefile，这对以后的学习有着极大的帮助。

NIUB