网络安全技术

实
验
报
告

学　　院　网安学院

年　　级　2021

班　　级　信息安全一班

学　　号　2111252

姓　　名　李佳豪

手机号　13191110713

2024 年 3 月 20 日

目录

一、实验目的

DES（Data Encryption Standard）算法是一种用 56 位有效密钥来加密 64 位数据的对称 分组加密算法，该算法流程清晰，已经得到了广泛的应用，算是应用密码学中较为基础的加 密算法。TCP（传输控制协议）是一种面向链接的、可靠的传输层协议。TCP 协议在网络层 IP 协议的基础上，向应用层用户进程〃供可靠的、全双工的数据流传输。

本章训练的目的如下。 ①理解 DES 加解密原理。 ②理解 TCP 协议的工作原理。 ③掌握 linux 下基于 socket 的编程方法。 本章训练的要求如下。 ①利用 socket 编写一个 TCP 聊天程序。 ②通信内容经过 DES 加密与解密。

二、实验内容

学习 **DES** 加密算法，理解**每一处**实现细节，并且使用 C++实现算法；此外复习 **SOCKET** 编程，在 windows 平台使用 socket 套接字构建 TCP 流传输实现点对点通讯，通讯内容经过**自己实现的 DES** 加密，最后设计并编写必要、美观的用户界面进行测试。

三、实验步骤及实验结果

实验步骤

1、DES 加、解密模块

(1) 密钥扩展

通过 `DesTransform` 表执行 P 置换，获得一个 56 位的密钥，之后将 56 位的密钥分为两个 28 位的组。然后，针对每个子密钥，根据子密钥的序列值（也就是 16 个子密钥中的第几个）旋转这两组值，然后重新合并。之后，再按照 `Despermuted` 对重组后的密钥进行置换，使 56 位的子密钥缩小为 48 位。

```
1.  void Key_exp(uint64_t* K, uint64_t* Ki) {
2.      uint32_t l; uint32_t r;  uint64_t _K = *K;
```

```
    Permute(K, &_K, DesTransform, 64, 55);
    for (int i = 0;i < 16;i++) {
        l = (_K >> 28) & 0xfffffff;
        r = _K & 0xfffffff;
        Rotate(&l);
        Rotate(&r);

        if (DesRotations[i] == 2){
            Rotate(&l);
            Rotate(&r);
        }
        _K = (uint64_t(l) << 28 | r);
     Permute(&_K, Ki + i, Despermuted,56 , 47);
    }
}
```

(2) P 置换：

```
void Permute(uint64_t* src, uint64_t* dst, const uint8_t* table,int src_len ,int dst_len) {
    uint64_t _t = 0;
    for (int i = 0; i <= dst_len; i++) {
        _t |= ((*src >> (src_len - (table[dst_len - i])) & 0b1) << i);
    memcpy(dst, &_t, 8);
}
```

(3) S 置换： 即通过将 48 位数，依次取 6 位进行查表替换。因此 while 每一次循环，将替换后的 4 位拼接到结果_t 中。

```
void Subtitute(uint64_t* src, uint64_t* dst) {
    // input 6*8  output 4*8
    int i=0, row, col;
    uint8_t t;
    uint64_t _t=0;
    while (i < 8){
        t = *src >> (i * 6) & 0b111111;
        row = (t >> 5) << 1 | (t & 0b1);
        col = t >> 1 & 0b1111;
        _t |= (uint64_t(S_Box[7-i][row][col]) << (i * 4));
        i++;
    }
    memcpy(dst, &_t,8);}
```

2、利用 SOCKET 实现用户点对点连接

Socket.h 需要包含应用程序的 socket 连接工作；

作为 Server 时，listen_socket 初始化后监听本地 PORT，remote 负责获得 connect 的 socket。

作为 Client 时，remote 负责向 Server 发起连接。

socket.h

```
#ifndef HEADER_SOCKEH
#define HEADER_SOCKEH
#define PORT 8999

#include<winsock2.h>
#include <WS2tcpip.h>
#pragma comment(lib,"ws2_32.lib")

extern bool FLag_Recv;
extern void appendMess(std::string s);

extern SOCKET listen_socket;
extern SOCKET remote_sock;

void StartServ(bool mode);

void RecvFromRemote(int id);

#endif // HEADER_SOCKEH
```

3、编写用户界面

Thread 中调用 ThreadView，将每隔 1s 刷新聊天区，及时绘制新的消息。

```
 thread  t1 = std::thread(ThreadView, 1);
void ThreadView(int id) {
    cout << "\u001b[50D\u001b[28B" << "\u001b[1m\u001b[36mINPUT:" << "\u001b[0m\n";
    while (Flag_View) {
        DrawInterface();
        sleep_for(std::chrono::milliseconds(1000));
    }
    cout << "View Exit" << endl;
}

```

```
void DrawInterface() {

    saveCursorPosition();
    cout << "\u001b[50A\u001b[50D";    //将光标input上一行 清除聊天
    clearLines(25);

    // 绘制界面
    cout << "\u001b[36m+" << setw(60) << setfill('-') << "+\n";
    cout << "\u001b[36m\u001b[1m\u001b[4m>>ChatRoom" << "\u001b[0m Time: " << time2s(getCurrentTime()) << endl;
    cout << "\u001b[36m+" << setw(60) << setfill('-') << "+\n" << "\u001b[0m";

    //输出message
    int i = 0;
    for (auto it = messBuffer.begin(); it != messBuffer.end(); ++it, ++i) {
        cout << (*it).toString();
        if (i > MAX_MESS_NUM)
            break;
    }
    cout << "\u001b[36m+" << setw(60) << setfill('-') << "+\n";

    restoreCursorPosition();// 恢复光标位置
}
```

4、设计 pad 方案，利用加密模块对传输内容加密。

(1) Pad 方案，向数据块的末尾添加若干字节，使整个数据块的长度能够被块大小整除。

```
// 使用 PKCS#7 填充算法填充字符串
std::string padString(const std::string& input, size_t block_size) {
    std::string result = input;
    size_t pad_len = (block_size - (input.length() & 0x8)) & 0x8
    result.append(pad_len, static_cast<char>(pad_len));
    return result;
}
```

(2) 加密和解密：

```
string Decrypy(const string& plain) {
    // plain 是 8 位对齐的
```

```
    string rs = "";
    uint64_t t = 0;
    vector<std::string> blocks = splitIntoBlocks(plain, block_size);
  //依次解密
   for (const auto& block : blocks) {
        Decrypy(block2ull(block), &t);
        int idx = 56;

        //64 位转 char*
        for (int i = 0;i < 8;i++) {
            if ((t >> idx & 0xff) < 8)
                return rs;
            rs += char(t >> idx & 0xff);
            idx -= 8;
        }
    }
    return rs;
}

string Encrypy(const string& plain) {

    string padded_string = padString(plain, block_size);
     // 同加密 ...
}
```

5、最终整合

将所用模块汇总，使用虚拟机模拟 Server，本地模拟 Client，即可进行双方点对点通讯。

```
  int main(){

    set_key(0x02140316);
    StartServ(0);

    thread t0 = std::thread(RecvFromRemote, 0);
    thread  t1 = std::thread(ThreadView, 1);
    thread  t2 = std::thread(ThreadInput, 2);

    t0.join();
    t1.join();
    t2.join();
```

```
13.
14.     WSACleanup();
15.     cout << "Exit\n";
16. }
```

实验结果：

1、DES 模块：

测试样例：

```
1.      uint64_t plain(0x223ffeffaabbcc21);
2.      uint64_t* dst = new uint64_t(0);
3.
4.      cout << "\u001b[31m\u001b[1m" << "Plain: " << to_binary(plain,
    64) << "\u001b[0m" << endl;
5.
6.      uint64_t* K = new uint64_t(0x4142434445464748);
7.      cout << "K_-: " << to_binary(*K, 64) << endl;
8.
9.      bool En = 1;
10.
11.     uint64_t  Ki[16];
12.     Key_exp(K, Ki);
13.     for (int i = 0;i < 16;i++)
14.         cout << "K_" << i << " " << to_hex(Ki[i]) << endl;
15.
16.     Des_Encrypy(plain, dst, Ki, En);
17.     cout << "\u001b[32m\u001b[1m"<<"加
    密: " << to_binary(*dst, 64) << "\u001b[0m" << endl;
18.
19.     uint64_t* ddst = new uint64_t(0);
20.     Des_Encrypy(*dst, ddst, Ki, !En);
21.     cout << "\u001b[31m\u001b[1m"<<"解
    密: " << to_binary(*ddst, 64)<<"\u001b[0m" << endl;
```

结果如下：红色部分是加密前和解密后的结果，可以完全对应

2、Padding & 加密模块

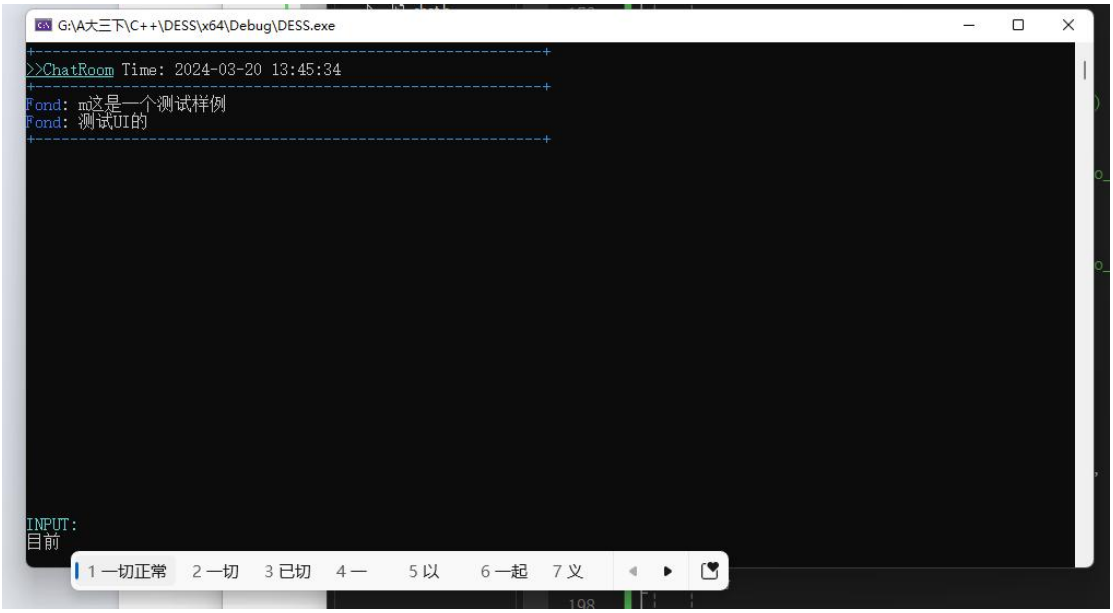
测试样例

```
1.  int main(){
2.    set_key(0x02140316);
3.    extern size_t block_size;
4.
5.    std::string plain_string = "这是一个测试 pad 结果和加解密是否正确的样例";
6.    cout << "\u001b[32mPlain_string: " << plain_string<<"\u001b[0m\n" << endl;
7.
8.    string padded_string = padString(plain_string, block_size);
9.    cout << "\u001b[35mPadding_string: " << plain_string << "\u001b[0m\n" << endl;
10.
11.   string r = Encrypy(padded_string);
12.   cout << "\u001b[36mEncrypy_string: " << r << "\u001b[0m\n" << endl;
13.
14.   string r_ = Decrypy(r);
15.   cout << "\u001b[32mDecrypt_string: " << r_ << "\u001b[0m\n" << endl;
16. }
```

结果展示： pad 结果和解密结果完全正确。

3、CHAT UI 界面

UI 展示当前时间、用户名和聊天内容，最下方为输入区

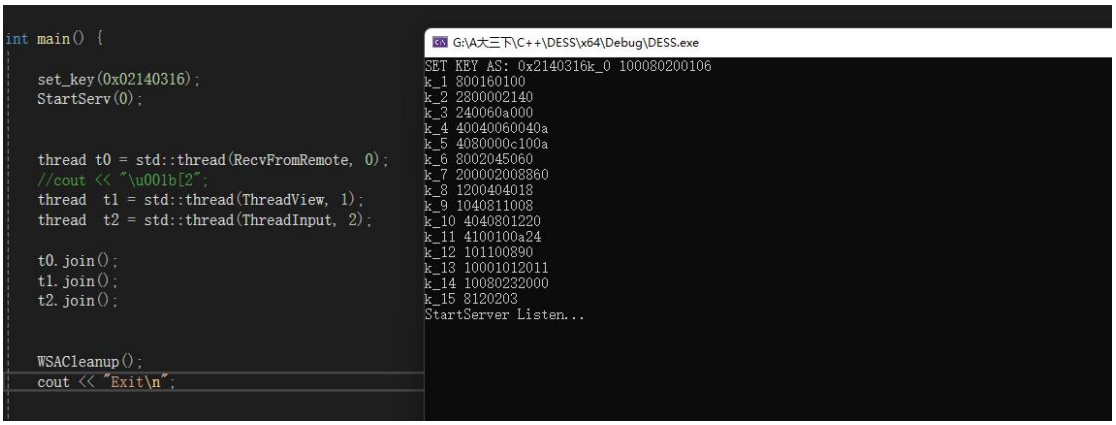


4、程序演示

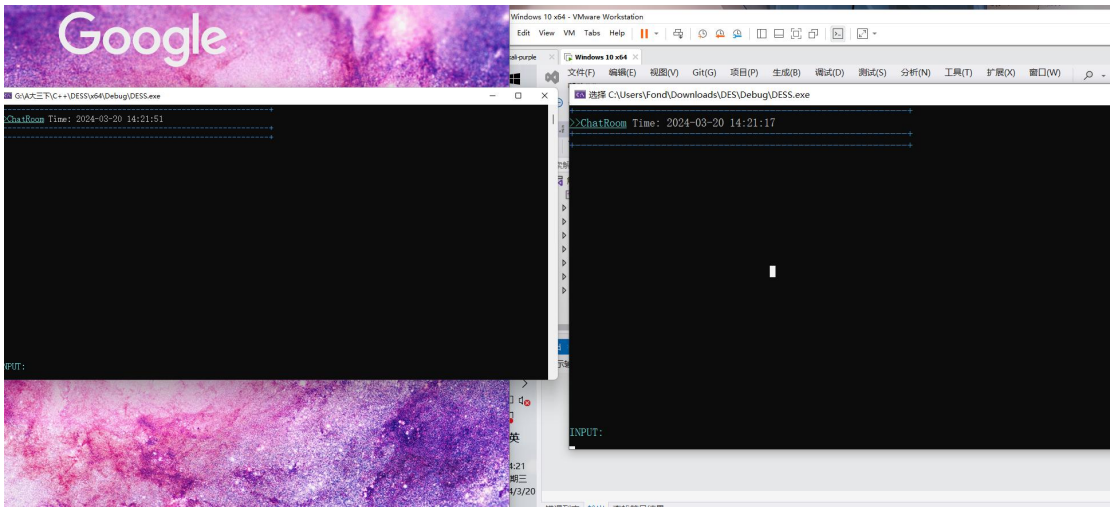
Server：windows11 本机

Client：windows10 虚拟机

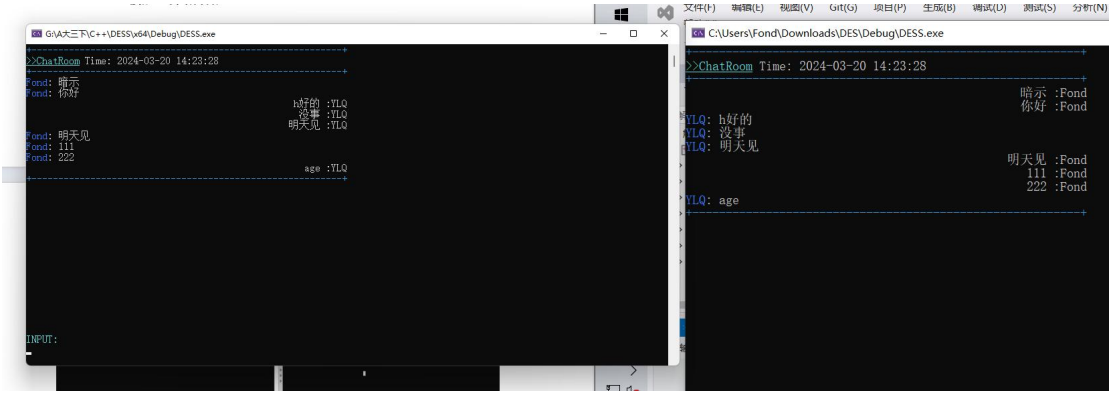
Server 开启监听：



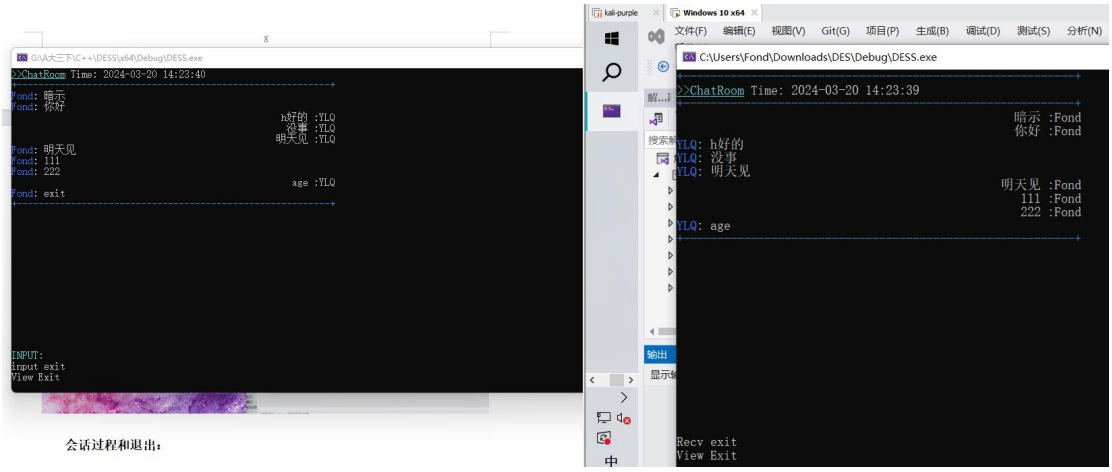
◆ **Client 连接后，会话开始：**



◆ **会话过程：**



◆ **[Exit] 退出：**



会话过程和退出：

四、实验遇到的问题及其解决方法

问题一：

实现 DES 加解密模块时，由于算法中存在 28、48、56 位等数据，而 C++ 提供的只有 8、16、32 和 64 位，采取**什么数据结构**存储中间数据首先需要考虑。

解决方案：

我处理方法是：没有超过位的数据利用 32 位的 uint32_t 存储，而超过 32 位的使用 uint64_t 存储。

问题二：

DES 加密的输入一定是 8 字节，但一次传输的数据不是 8BYTE 对齐的，需

要进行 pad，但 pad 后加密的密文当传输给对方，对方解密后还需要将 pad 的字符截断。如何 pad 以及将 pad str 解密？

解决方案：

pad 的方法按照 PKCS#7 填充算法填充字符串，即 pad 使用字符的 ASCII 码为 （8 - input.size()%8），当对方解密时只要发现当前解析的字符 ASCII 值在 0-8 之间就知道这是 pad 的结果。

问题三：

由于我在 DES 加密设计的 API 的输入是 uint64_t，但是 app 传入的是 string，那么将依次从 string 中截取 8 个字符、将其转换为 64 位的 uint64_t 即可。
例如下写法：

```
1.  uint64_t rs = 0;
2.  int idx = 56;
3.      for (char c : substr){
4.          rs |= uint64_t(c) << idx;
5.          idx -= 8;
6.      }
7.  return rs;
```

但实际上上面的转换无法满足要求，最后加密结果是错误的。

解决方案：

经过很久的 debug，发现这是由于使用 char 类型 扩展为 uint64_t ，其最高位会作为符号位拼接导结果的最高位中。因此需要将 char 先转为 u_char 再扩展为 uint64_t。

因此第四行应该使用如下写法：

```
4. rs |= uint64_t(uint8_t(c)) << idx;
```

问题四：

终端设计用户界面时，需求有如下：实时接受对方消息显示在界面；有输入

栏，并在输入后按回车发送。前者要求一个线程持续刷新界面、绘制新消息，而后者则需要另一个线程接受输入，并将当前输入的内容保留在界面，但终端只有一个，刷新界面不能将当前输入刷新。

即如何刷新解密但是还可以在解密中保持用户输入状态。

解决方案：

在刷新线程需要刷新时，先保存当前输入最后一个字符的光标坐标，然后执行刷新，刷新时控制只重新绘制聊天信息界面，而保留输入状态，刷新结束恢复光标坐标，从而达到刷新界面但是还能保存用户输入状态。

五、实验结论

1. 之前我们是学过密码学这门课的，相比当今强大的算力，DES 的 64 位密钥长度已经不能提供足够的安全性。也因此我们之前并没有过多了解 DES。这次作业从 0 开始复现，不仅是简单的实现功能，更要保证时算法设计的空间复杂度、时间复杂度都尽可能小，更能让我对 DES 有深的理解。
2. 这次作业在设计用户界面时，重点研究如何解决多线程共享一个终端，并且保证终端显示的正确。经过探索也确定了改变 cursor 位置，实现两个线程对终端不同区域的绘制，简单实现了我的目的。也因此看了很多 ANSI 转义序列的知识。
3. 总之这是一个比较综合的作业，代码从头开始写用了整整一天的时间，DEBUG 又花了半天时间，无论是 coding 方面还是相关知识方面的收获都很大。