

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	1
二、 实验内容	1
三、 实验步骤及实验结果	1
实验步骤	1
1、 DES 加、解密模块	1
2、 利用 SOCKET 实现用户点对点连接	3
3、 编写用户界面	3
4、 设计 pad 方案，利用加密模块对传输内容加密。	4
5、 最终整合	5
实验结果:	6
1、 DES 模块:	6
2、 Padding & 加密模块	7
3、 CHAT UI 界面	8
4、 程序演示	9
四、 实验遇到的问题及其解决方法	10
问题一:	10
问题二:	10
问题三:	11
问题四:	11
五、 实验结论	12

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，向应用层用户进程提供可靠的、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;
```

```

3.     Permute(K, &_K, DesTransform, 64, 55);
4.     for (int i = 0; i < 16; i++) {
5.         l = (_K >> 28) & 0xffffffff;
6.         r = _K & 0xffffffff;
7.         Rotate(&l);
8.         Rotate(&r);
9.
10.        if (DesRotations[i] == 2){
11.            Rotate(&l);
12.            Rotate(&r);
13.        }
14.        _K = (uint64_t(l) << 28 | r);
15.        Permute(&_K, Ki + i, Despermuted, 56, 47);
16.    }
17. }

```

(2) P 置换:

```

1. void Permute(uint64_t* src, uint64_t* dst, const uint8_t* table, int src_len, int dst_len) {
2.     uint64_t _t = 0;
3.     for (int i = 0; i <= dst_len; i++) {
4.         _t |= ((*src >> (src_len - (table[dst_len - i])) & 0b1) < < i);
5.         memcpy(dst, &_t, 8);
6.     }

```

(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。

```

1. void Subtitute(uint64_t* src, uint64_t* dst) {
2.     // input 6*8 output 4*8
3.     int i=0, row, col;
4.     uint8_t t;
5.     uint64_t _t=0;
6.     while (i < 8){
7.         t = *src >> (i * 6) & 0b111111;
8.         row = (t >> 5) << 1 | (t & 0b1);
9.         col = t >> 1 & 0b1111;
10.        _t |= (uint64_t(S_Box[7-i][row][col]) << (i * 4));
11.        i++;
12.    }
13.    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
1. #ifndef HEADER_SOCKEH
2. #define HEADER_SOCKEH
3. #define PORT 8999
4.
5. #include<winsock2.h>
6. #include <WS2tcpip.h>
7. #pragma comment(lib, "ws2_32.lib")
8.
9. extern bool Flag_Recv;
10. extern void appendMess(std::string s);
11.
12. extern SOCKET listen_socket;
13. extern SOCKET remote_sock;
14.
15. void StartServ(bool mode);
16.
17. void RecvFromRemote(int id);
18.
19. #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。

```
1. thread t1 = std::thread(ThreadView, 1);
1. void ThreadView(int id) {
2.     cout << "\u001b[50D\u001b[28B" << "\u001b[1m\u001b[36mINPUT:"
    << "\u001b[0m\n";
3.     while (Flag_View) {
4.         DrawInterface();
5.         sleep_for(std::chrono::milliseconds(1000));
6.     }
7.     cout << "View Exit" << endl;
8. }
9.
```

```

10. void DrawInterface() {
11.
12.     saveCursorPosition();
13.     cout << "\u001b[50A\u001b[50D";    //将光标input 上一行 清除聊天
14.     clearLines(25);
15.
16.     // 绘制界面
17.     cout << "\u001b[36m+" << setw(60) << setfill('-') << "+\n";
18.     cout << "\u001b[36m\u001b[1m\u001b[4m>>ChatRoom" << "\u001b[0
    m Time: " << time2s(getCurrentTime()) << endl;
19.     cout << "\u001b[36m+" << setw(60) << setfill('-') << "+\n" <<
        "\u001b[0m";
20.
21.     //输出 message
22.     int i = 0;
23.     for (auto it = messBuffer.begin(); it != messBuffer.end(); ++
        it, ++i) {
24.         cout << (*it).toString();
25.         if (i > MAX_MESS_NUM)
26.             break;
27.     }
28.     cout << "\u001b[36m+" << setw(60) << setfill('-') << "+\n";
29.
30.     restoreCursorPosition();// 恢复光标位置
31. }

```

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。

```

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

```

- (2) 加密和解密:

```

1. string Decrypy(const string& plain) {
2.     // plain 是 8 位对齐的

```

```

3.     string rs = "";
4.     uint64_t t = 0;
5.     vector<std::string> blocks = splitIntoBlocks(plain, block_size);
6.     // 依次解密
7.     for (const auto& block : blocks) {
8.         Decrypy(block2ull(block), &t);
9.         int idx = 56;
10.
11.         //64 位转 char*
12.         for (int i = 0; i < 8; i++) {
13.             if ((t >> idx & 0xff) < 8)
14.                 return rs;
15.             rs += char(t >> idx & 0xff);
16.             idx -= 8;
17.         }
18.     }
19.     return rs;
20. }
21.
22. string Encrypy(const string& plain) {
23.
24.     string padded_string = padString(plain, block_size);
25.     // 同加密 ...
26. }

```

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。

```

1.     int main(){
2.
3.         set_key(0x02140316);
4.         StartServ(0);
5.
6.         thread t0 = std::thread(RecvFromRemote, 0);
7.         thread t1 = std::thread(ThreadView, 1);
8.         thread t2 = std::thread(ThreadInput, 2);
9.
10.        t0.join();
11.        t1.join();
12.        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(0x223ffefffaabbcc21);
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_Encrypt(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_Encrypt(*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


```
Microsoft Visual Studio 调试控制台
Plain: 00100010 00111111 11111110 11111111 10101010 10111011 11001100 00100001
K_0: 01000001 01000010 01000011 01000100 01000101 01000110 01000111 01001000
K_0 a09242132a82
K_1 a01252102307
K_2 245250b60084
K_3 651504023c3
K_4 e415136a009
K_5 f4109621542
K_6 b01890ca12a
K_7 190889645c40
K_8 1908884a9840
K_9 10288cc0c538
K_10 102c04091e08
K_11 402c24d85030
K_12 c0a424014a2c
K_13 c08622903890
K_14 e09222a10235
K_15 a09222a34280
加密: 00010001 10110011 01110000 10001111 11000110 00010100 01010001 01000101
解密: 00100010 00111111 11111110 11111111 10101010 10111011 11001100 00100001
G:\A大三下\C++\DESS\x64\Debug\DESS.exe (进程 3184)已退出, 代码为 0。
按任意键关闭此窗口. . .
```

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" << e
   ndl;
13.
14.    string r_ = Decrypy(r);
15.    cout << "\u001b[32mDecrypt_string: " << r_ << "\u001b[0m\n" <<
   endl;
16. }
```

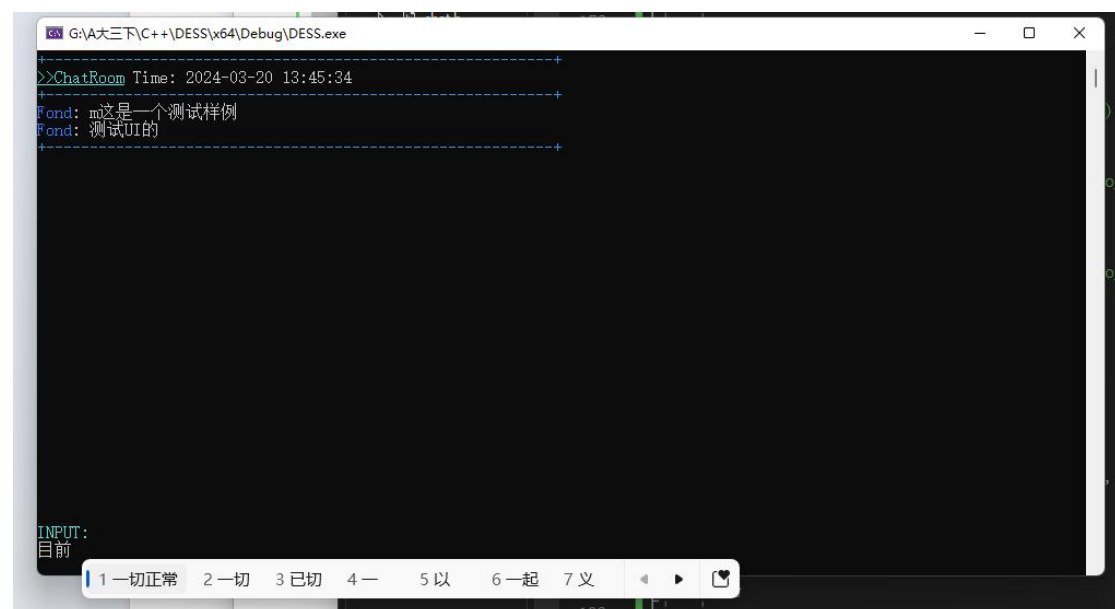
结果展示： pad 结果和解密结果完全正确。

```
Microsoft Visual Studio 调试控制台
SET KEY AS: 0x2140316k_0 100080200106
k_1 800160100
k_2 2800002140
k_3 240060a000
k_4 40040060040a
k_5 4080000c100a
k_6 8002045060
k_7 200002008860
k_8 1200404018
k_9 1040811008
k_10 4040801220
k_11 4100100a24
k_12 101100890
k_13 10001012011
k_14 10080232000
k_15 8120203
Plain_string: 这是一个测试pad结果和加解密是否正确的样例
Padding_string: 这是一个测试pad结果和加解密是否正确的样例
Encrypt_string: 蛛醅?t骛#单媒&>昆崑)狎霸av纒R勵璽?
v? 2什
Decrypt_string: 这是一个测试pad结果和加解密是否正确的样例

G:\A大三下\C++\DESS\x64\Debug\DESS.exe (进程 36640)已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
```

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

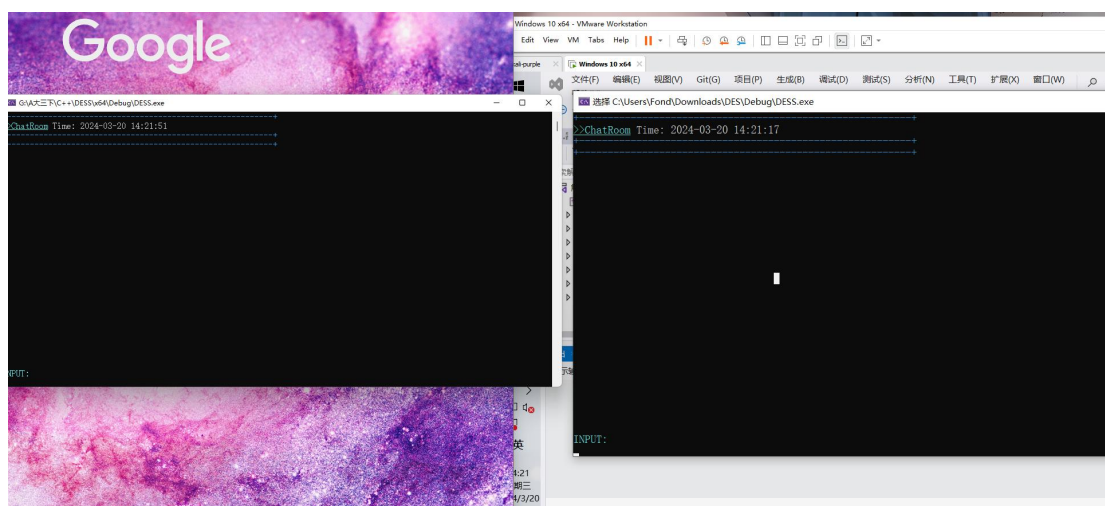
Client: windows10 虚拟机

Server 开启监听:

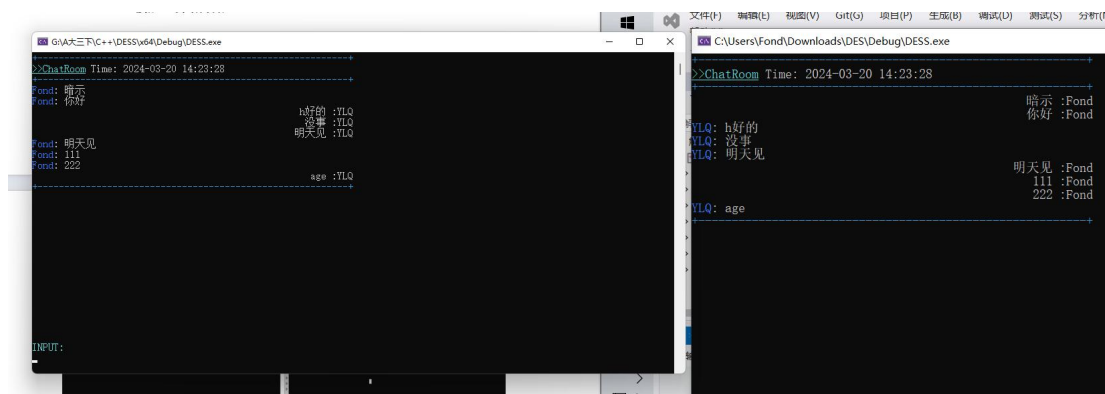
```
int main() {  
    set_key(0x02140316);  
    StartServ(0);  
  
    thread t0 = std::thread(RecvFromRemote, 0);  
    //cout << "\u001b[2";  
    thread t1 = std::thread(ThreadView, 1);  
    thread t2 = std::thread(ThreadInput, 2);  
  
    t0.join();  
    t1.join();  
    t2.join();  
  
    WSACleanup();  
    cout << "Exit\n";  
}
```

```
G:\A大三下\C++\DESS\Debug\DESS.exe  
SET KEY AS: 0x2140316k_0 100080200106  
k_1 800160100  
k_2 2800002140  
k_3 240060a000  
k_4 40040060040a  
k_5 4080000c100a  
k_6 8002045060  
k_7 200002008860  
k_8 1200404018  
k_9 1040811008  
k_10 4040801220  
k_11 4100100a24  
k_12 101100890  
k_13 10001012011  
k_14 10080232000  
k_15 8120203  
StartServer Listen...
```

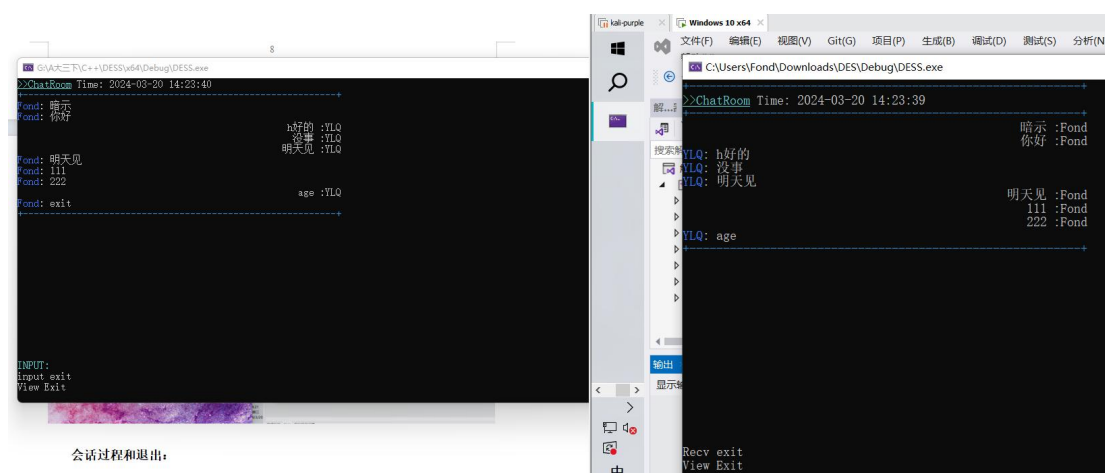
◆ Client 连接后，会话开始:



◆ 会话过程:



◆ [Exit] 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 $(8 - \text{input.size()} \% 8)$ ，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。