网 络 安 全 技 术

实 验 报 告

学　　院：网络空间安全学院
年　　级：2021 级
专　　业：信息安全、法学双学位
学　　号：2113203
姓　　名：付政烨
手 机 号：15719340667
完成日期：2024 年 3 月 29 日

目　录

目录..... I
一、　实验目的..... 1
二、　实验内容..... 1
三、　实验步骤及实验结果..... 1
（一）　DES 加密与解密..... 1
1. 加密操作..... 2
2. 解密操作..... 3
3. F 轮函数..... 4
4. 子密钥生成..... 6
5. 基础加密辅助函数..... 9
（二）　基于 DES 的客户端与服务端通信..... 9
1. 客户端..... 9
2. 服务端..... 10
（三）　实验结果..... 11
四、　实验遇到的问题及其解决方法..... 12
（一）　TCP 套接字编程中的连接问题..... 12
（二）　DES 加密/解密过程中的数据对齐问题..... 12
（三）　编译问题及 Makefile 的编写..... 12
五、　实验结论..... 13

一、 实验目的

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。

二、 实验内容

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。

三、 实验步骤及实验结果

(一) DES 加密与解密

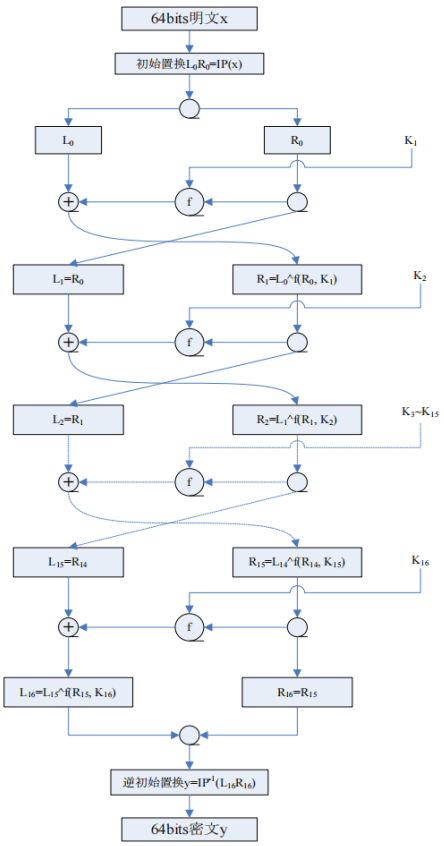

图 3.1 DES 加密流程图

1. 加密操作

(1) 原理简介

DES 加密算法是一种基于对称密钥的加密方法，广泛应用于各种安全通信中。它将 64 位的明文输入转换为 64 位的密文输出，通过一个 56 位的密钥控制加密过程。DES 的核心是一个复杂的转换过程，包括初始置换、16 轮的 Feistel 结构处理以及最终的逆初始置换。每一轮的处理都依赖于一个从原始密钥生成的 48 位子密钥。

加密过程的核心是 Feistel 网络，它将数据分为左右两半，然后通过一系列的操作将右半部分转换并与左半部分进行混合。这种结构的关键特点是加密和解密过程几乎相同，只需将子密钥的使用顺序逆转即可实现解密。

(2) 代码介绍

- **密钥检查与转换**: 在加密操作开始之前，首先检查密钥的长度是否为 8 个字符（64 位），然后将密钥从字符串形式转换为布尔向量形式，以便进行后续的加密处理。
- **子密钥生成**：通过 gen_subkey 函数，基于输入的 64 位密钥生成 16 个 48 位的子密钥。这些子密钥将在后续的加密过程中的每一轮中被使用。

```
vector<vector<bool>> sub_key = gen_subkey(bin_key);
```

- **明文转换**：将明文从字符串形式转换为二进制形式，以便后续进行加密处理。

```
vector<vector<bool>> bin_raw_text = encry_str2bool(raw_text);
```

- **加密过程**：加密操作对每一个明文块进行处理。encry_process 函数接收一个明文块和子密钥集合，执行 DES 加密过程，返回加密后的密文块。

```
for (int i = 0; i < bin_raw_text.size(); i++) {
    vector<bool> temp_res = encry_process(bin_raw_text[i], sub_key);
    bin_code_text.push_back(temp_res);
}
```

在加密的每一轮中，首先对数据进行初始置换，然后进行 16 轮的处理，每一轮中将左半部分直接成为新的右半部分，而新的左半部分则通过将原右半部分进行扩展、与子密钥混合、S-盒处理、P 置换后，与原左半部分进行异或操作得到。这些操作完成后，将处理后的数据进行逆初始置换，得到最终的密文数据。

```
vector<bool> temp_step1 = init_replacement_IP(input, INIT_REPLACE_IP);
...
for (int i = 0; i < 16; i++) {
    vector<bool> f_func_res = f_func(step1_r, sub_key[i]);
    vector<bool> xor_res = XOR(f_func_res, step1_l);
    step1_l = step1_r;
    step1_r = xor_res;
}
...
step3 = init_replacement_IP(step3, INVERSE_REPLACE_IP);
```

2. 解密操作

(1) 原理简介

DES 加密算法的解密过程基本上是加密过程的逆操作。首先对密文进行初始置换，然后通过 16 轮的处理，每一轮都使用子密钥（与加密过程中使用的顺序相反），最后进行一次逆初始置换，得到明文。

(2) 代码介绍

- **密钥长度检查和转换**：在解密操作开始之前，首先检查密钥长度是否为 8 个字符（即 64 位），然后将密钥从字符串形式转换为布尔向量形式，以便后续处理。
- **子密钥生成**：gen_subkey 函数根据输入的 64 位密钥生成 16 个子密钥，每个子密钥用于加密过程中的一轮。

```
vector<vector<bool>> sub_key = gen_subkey(bin_key);
```

- **密文转换**：将密文从字符串形式转换为二进制形式，便于后续的解密处理。

```
vector<vector<bool>> bin_code_text = decry_str2bool(cipherText);
```

- **解密过程**：解密操作对每一个密文块进行处理。decry_process 函数接收一个密文块和子密钥集合，执行 DES 解密过程，返回解密后的明文块。

```
for (int i = 0; i < bin_code_text.size(); i++) {
    vector<bool> temp_res = decry_process(bin_code_text[i], sub_key);
    bin_raw_text.push_back(temp_res);
}
```

- **解密过程的细节**：在解密的每一轮中，首先对数据进行初始置换，然后进行 16 轮的处理，每一轮中将右半部分通过扩展、与子密钥混合、S-盒处理、P 置换后，与左半部分进行异或操作。这些操作的顺序和加密时相反，使用的子密钥顺序也相反。最后，将处理后的数据进行逆初始置换，得到最终的明文数据。

```
vector<bool> temp_step1 = init_replacement_IP(input, INIT_REPLACE_IP);
...
for (int i = 0; i < 16; i++) {
    vector<bool> f_func_res = f_func(step1_l, sub_key[15 - i]);
    vector<bool> xor_res = XOR(f_func_res, step1_r);
    step1_r = step1_l;
    step1_l = xor_res;
}
...
step3 = init_replacement_IP(step3, INVERSE_REPLACE_IP);
```

3. F 轮函数

(1) 原理简介

在 DES 加密算法中，f 函数是加密过程的核心部分，它负责将半块（32 位）的数据通过一系列复杂的变换转换为另一半块的数据。f 函数的设计使得 DES 算法的加密过程具有高度的复杂性和安全性。f 函数包括以下四个步骤：扩展置换

(E 盒)、与子密钥进行异或操作（密钥加法)、S 盒替代（选择压缩运算）和 P 盒置换（置换运算)。

(2) 代码介绍

- **扩展置换（E 盒)**：E 盒将 32 位的输入扩展到 48 位。这是通过重复使用部分位实现的，目的是为了让每一位都能受到来自子密钥的影响。E 盒操作利用 des_E_box 数组来决定如何从 32 位输入中选择和重复位以产生 48 位输出。

```
vector<bool> CDesOperate::E_Box(vector<bool> input) {
    vector<bool> e_box_output;
    for (int i = 0; i < 48; i++) {
        e_box_output.push_back(input[des_E_box[i] - 1]);
    }
    return e_box_output;
}
```

- **密钥加法**：扩展后的 48 位与本轮的 48 位子密钥进行异或运算。这一步骤将密钥的信息混入到数据中。输入与子密钥进行异或运算，实现简单，直接调用 XOR 函数处理扩展后的 48 位数据和 48 位子密钥。

```
vector<bool> CDesOperate::key_add(vector<bool> input, vector<bool> key) {
    return XOR(input, key);
}
```

- **S 盒替代（选择压缩运算)**：异或运算的结果被分为 8 个 6 位的分组，每个分组通过一个 S 盒转换为 4 位。每个 S 盒都是不同的，可以将 6 位的输入映射到 4 位的输出，提供了非线性变换的特性。这个过程将 48 位输入分为 8 个 6 位的分组，每组数据通过对应的 S 盒转换成 4 位，总共得到 32 位的输出。

```
vector<bool> CDesOperate::select_comp_operation(vector<bool> input) {
    ...
    for (int i = 0; i < 8; i++) {
```

```
        int temp_int = des_S_box[i][binary2dec(input_group[i])];
        vector<bool> temp_bool = dec2binary(temp_int, 4);
        res.insert(res.end(), temp_bool.begin(), temp_bool.end());
    }
    return res;
}
```

- **P 盒置换（置换运算）**：S 盒的输出被重新组合为 32 位，然后通过 P 盒进行置换。P 盒的目的是将 S 盒的输出进一步散列，增强加密的复杂度和安全性。使用 P 盒（由 rep_P 数组定义）对 S 盒的输出进行置换，生成最终的 32 位输出。

```
vector<bool> CDesOperate::replace_operation(vector<bool> input) {
    vector<bool> result;
    for (int i = 0; i < 32; i++) {
        result.push_back(input[rep_P[i] - 1]);
    }
    return result;
}
```

4. 子密钥生成

(1) 原理简介

DES 加密算法的一个核心组成部分是其子密钥生成机制。在 DES 算法中，从 64 位的初始密钥中生成 16 个 48 位的子密钥，这些子密钥将在加密或解密的各个轮次中使用。子密钥生成过程涉及到几个关键步骤，包括奇偶校验位的设置、PC-1 置换、循环左移以及 PC-2 置换。

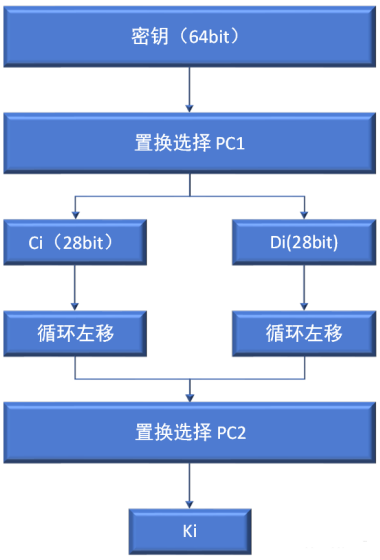

图 3.2 密钥生成流程图

(2) 代码介绍

- **奇偶校验位设置**：DES 要求 64 位密钥的每 8 位中的第 8 位为奇偶校验位，以确保每 8 位中有奇数个 1。这是密钥预处理的一部分，用于检查密钥的完整性。代码通过计算每 8 位中 1 的个数，并据此设置每个字节的第 8 位，以满足奇数个 1 的要求。

```
for (int i = 0; i < 8; i++) {
    int cnt = 0;
    for (int j = 0; j < 7; j++) {
        if (init_key[i * 8 + j]) {
            cnt++;
        }
    }

    if (cnt & 1 == 0) {
        init_key[i * 8 + 7] = true;
    }
    else {
        init_key[i * 8 + 7] = false;
    }
}
```

- **PC-1 置换**：第一步是对 64 位密钥进行 PC-1 置换，这一过程将 64 位密钥缩减为 56 位，去除了每 8 位中的奇偶校验位。这 56 位被分为两个 28 位的部分进行后续操作。通过使用预定义的 PC-1 置换表，从 64 位密钥中选择 56 位，并将其分为两个 28 位的部分。

```
vector<vector<bool>> key_PC1(2);
for (int i = 0; i < 28; i++) {
    key_PC1[0].push_back(init_key[key_PC_1[i] - 1]);
    key_PC1[1].push_back(init_key[key_PC_2[i] - 1]);
}
```

- **循环左移**：对这两个 28 位的部分进行循环左移操作。在 16 轮生成子密钥的过程中，每轮的左移次数是预先定义好的，这个过程有助于增加密钥的复杂度。根据预定义的左移表，对这两个 28 位的部分进行循环左移操作。

```
for (int i = 0; i < 16; i++) {
    key_PC1[0] = left_shift ( left_shift_table [i], key_PC1[0]);
    key_PC1[1] = left_shift ( left_shift_table [i], key_PC1[1]);
     ...
}
```

- **PC-2 置换**：最后，将经过左移的 56 位密钥进行 PC-2 置换，缩减为 48 位，作为一轮的子密钥。这一过程在 16 轮中重复执行，生成 16 个不同的子密钥。

```
vector<vector<bool>> key_PC2;
for (int i = 0; i < 16; i++) {
    vector<bool> temp_key_PC2;
    for (int j = 0; j < 48; j++) {
        temp_key_PC2.push_back(key_left_shift[i][key_choose[j] - 1]);
    }
    key_PC2.push_back(temp_key_PC2);
}
```

5. 基础加密辅助函数

在加密算法的实现过程中，特别是在对称密钥加密算法如 DES 中，经常需要执行一些基础的操作，包括左移、进制转换和异或操作。这些操作虽然基础，但在加密和解密过程中起着至关重要的作用。左移操作通常用于密钥生成过程中，进制转换在密钥或数据的处理中频繁使用，而异或操作是 Feistel 网络的核心组成部分，用于数据的混淆和解混淆（详见源代码）。

（二） 基于 DES 的客户端与服务端通信

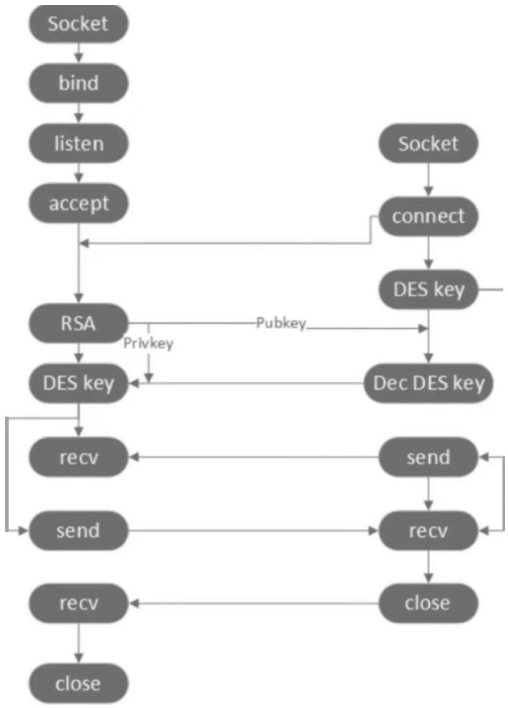

图 3.3 基于 DES 的客户端与服务端通信架构图

1. 客户端

客户端程序的主要功能是向服务端发起连接请求，发送加密消息，并接收来自服务端的加密消息。它首先创建一个 TCP 套接字，然后连接到服务端指定的 IP 地址和端口号。一旦连接建立，客户端可以发送加密的消息给服务端，并能够接收并解密来自服务端的加密消息。

- **连接建立**：负责设置服务器的地址和端口，然后使用 connect 函数尝试建立到服务端的连接。

```
struct sockaddr_in server_addr;
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_port = (tcp_port);
server_addr.sin_family = AF_INET;
...
if (connect(tcp_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) <
    0) {...}
```

- **消息加密与发送**：在发送消息之前，使用 encry_operation 方法对消息进行加密。加密成功后，通过 send 函数将加密后的消息发送到服务端。

```
string encry_res;
if (des.encry_operation(client_msg, DES_KEY, encry_res) != 0) {...}
...
if (send(tcp_socket, client_msg, strlen(client_msg), 0) < 0) {...}
```

- **消息接收与解密**：客户端使用 recv 函数接收来自服务端的加密消息，并使用 decry_operation 方法进行解密。

```
string encry_res;
if (des.encry_operation(client_msg, DES_KEY, encry_res) != 0) {...}
...
if (send(tcp_socket, client_msg, strlen(client_msg), 0) < 0) {...}
```

2. 服务端

服务端程序监听来自客户端的连接请求。一旦接收到请求，它将接收客户端发送的加密消息，解密这些消息，然后可能回复一个加密的响应。服务端能够同时处理多个客户端的连接请求。

- **监听和接收连接**：服务端首先创建一个 TCP 套接字，然后绑定到一个本地地址和端口上，并开始监听连接请求。使用 accept 函数等待和接收来自客户端的连接请求。

```
int tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
...
```

```
if ( listen (tcp_socket, max_line) < 0) {...}
...
int cur_client = accept(tcp_socket, (struct sockaddr*)&client_addr, &length);
```

- **消息接收与解密**：当接收到客户端的消息时，服务端使用 recv 函数接收加密的消息，并使用 decry_operation 方法进行解密。

```
int client_msg_len = recv( cur_client , client_msg, sizeof(client_msg), 0);
...
if (des.decry_operation(client_msg, DES_KEY, decry_res) != 0) {...}
```

- **消息加密与发送**：服务端通过 encry_operation 方法对响应消息进行加密，然后使用 send 函数将加密的消息发送回客户端。

```
int client_msg_len = recv( cur_client , client_msg, sizeof(client_msg), 0);
...
if (des.decry_operation(client_msg, DES_KEY, decry_res) != 0) {...}
```

（三） 实验结果

图 3.4 服务端

图 3.5 客户端

四、 实验遇到的问题及其解决方法

（一） TCP 套接字编程中的连接问题

问题：客户端尝试连接到服务端时可能会遇到的问题包括网络不可达、服务端未监听指定端口或服务端地址不正确等。

解决方法：首先，确保服务端程序已经启动并且正在监听正确的端口。其次，检查客户端和服务端的 IP 地址和端口号设置是否正确。最后，使用网络工具（如 ping 命令）检查网络连通性。如果问题依旧存在，可以考虑检查防火墙设置是否阻止了连接。

（二） DES 加密/解密过程中的数据对齐问题

问题：由于 DES 算法处理的数据块大小是固定的（64 位），在加密或解密时可能会遇到输入数据长度不是 64 位倍数的情况。

解决方法：可以使用填充（Padding）技术来处理这一问题。常见的填充方法包括 PKCS#5/PKCS#7 填充，即在数据的末尾添加足够数量的字节，使得总长度达到 64 位的倍数。每个添加的字节的值等于添加的总字节数。在解密时，再根据填充规则去除这些填充字节，恢复原始数据。

（三） 编译问题及 Makefile 的编写

问题：在项目的初期阶段没有编写 Makefile，导致无法有效地编译和链接多个源文件，从而无法生成可执行文件。手动编译每个文件并尝试链接它们是一个繁琐且容易出错的过程。

解决方法：参考之前课程实验的 Makefile 编写了一个适用于当前项目的 Makefile。Makefile 通过定义规则来自动化编译和链接过程，极大地提高了开发效率。以下是一个简化版的 Makefile 示例，用于编译和链接一个基于 DES 加密算法的聊天程序（详见源代码）。

五、 实验结论

本次实验深入探究了 DES 加解密原理和 TCP 协议的应用，通过构建一个基于 DES 加密的客户端-服务端通信程序，实践了理论知识与编程技能的结合。在实验中，我们首先掌握了 DES 加密算法的工作流程，包括明文的初始置换、16 轮 Feistel 结构处理、最终的逆置换，以及解密过程几乎是这一过程的逆向操作。通过实现客户端与服务端之间的加密通信，加深了对 TCP 协议——一种可靠的、面向连接的通信协议—的理解，体会到了其在网络通信中确保数据有序性和可靠性的重要性。此外，实验过程中的 Socket 编程练习不仅加强了对网络编程接口的理解，也提升了解决实际问题的能力。

实验过程中遇到的挑战和问题促使我们不断查询资料、探索解决方案，从中体会到了不断学习新知识、新技术的重要性，尤其是在信息安全领域。与此同时，团队合作的经历让我认识到集思广益解决问题的力量，以及有效沟通的必要性。通过本次实验，不仅对 DES 算法和 TCP 协议有了更深入的理解，也认识到了理论与实践相结合学习方法的有效性，为今后的学习和研究奠定了坚实的基础。

参 考 文 献

[1] 吴功宜. 网络安全高级软件编程技术 [M]. 清华大学出版社,2010.