

DES 算法的程序设计与实现

蔡梓珩 16340008

算法原理概述

DES 是块加密。将需要加密的明文以 64 位一组的长度分组，通过对每一组进行初始置换、利用密钥 16 轮迭代、交换置换和逆置换得到 64 位密文。利用数学可证得，同样的对密文做相同操作可得到明文（迭代次序相反）。

总体结构

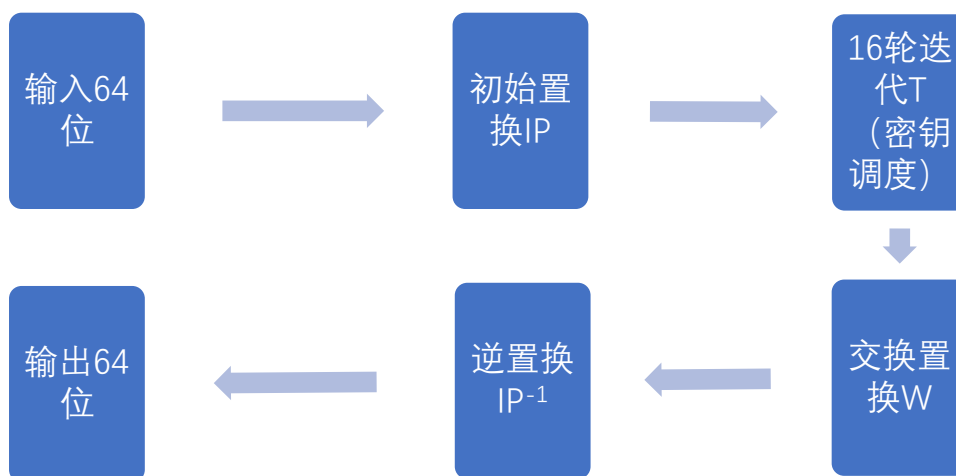
加密过程：

$$C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \cdots T_1 \cdot IP(M)$$

- M 为算法输入的 64 位明文块；
- E_k 描述以 K 为密钥的加密函数，由连续的过程复合构成；
- IP 为 64 位初始置换；
- T_1, T_2, \dots, T_{16} 是一系列的迭代变换；
- W 为 64 位置换，将输入的高 32 位和低 32 位交换后输出；
- IP^{-1} 是 IP 的逆置换；
- 是 IP 的逆置换。

解密过程：

$$M = D_k(C) = IP^{-1} \cdot W \cdot T_1 \cdot T_2 \cdots T_{16} \cdot IP(C)$$



模块分解：

模块将结合代码共同解释，所截图代码为编程阶段成果，有可能会在后面的测试中因为发现有 bug 而有改动，以文件中所附带代码为准。

1. 信息空间

首先需要将明文分割成等长 64 位的字符串。最后一组若不足 8 字节则补全（用要补全的个数取值），若刚好为 64 位，则再添加一个一个分组，里面 8 个字节都取 8。这里编了一个 Split 类用于分割以及合并明文。

分割实现：

```
vector<string> Split::split(string a) {
    vector<string> v;
    for(int i = 0; i <= a.size(); i += 8) {
        string temp;
        if (a.size() - i < 8) {
            int add = 8 - a.size() + i;
            temp.append(a.substr(i, i + 8 - add));
            for (int j = 8 - add; j < 8; j++) {
                char n = add + 48;
                temp = temp + n;
            }
        }
        else {
            temp.append(a.substr(i, 8));
        }
        v.push_back(temp);
    }
    return v;
}
```

合并实现：

```
string Split::merge(vector<string> a) {
    string temp;
    if (a.size() < 1) return temp;
    for (int i = 0; i < a.size() - 1; i++) {
        temp = temp + a[i];
    }
    string s = a[a.size() - 1];
    temp = temp + s.substr(0, 8 - int(s[7] - 48));
    return temp;
}
```

测试效果：

```
D:\Course\web安全技术\DES\code\DES.exe
拆分：
香锅的S8
，只有三
场烂摊子
和一颗没
有人帮他
挡的子弹
。666666

合并：
香锅的S8，只有三场烂摊子和一颗没有人帮他挡的子弹。
-----
Process exited after 0.03213 seconds with return value 0
请按任意键继续. . .
```

2. IP 置换

给定给定 64 位明文块 M ，通过一个固定的初始置换 IP 来重排 M 中的二进制位，得到二进制串 $M_0 = IP(M) = L_0R_0$ ，这里 L_0 和 R_0 分别是 M_0 的前 32 位和后 32 位。下表给出 IP 置换后的下标编号序列。

IP 置换表 (64位)							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

这里的 IP 置换表的第 a 个数字，相当于新数组第 a 个位，其中数字 i 相对于旧数组的 $num[i]$ 。即表中数字相当于这个位置的数字在旧数组中的下标。因此将此表录入数组，然后即可根据下标置换。这里为了方便转换操作，将 64 位的 8 字节字符串换成 64 字节对应其二进制值。

建立 IP 与 IP^{-1} 置换表：

```

IP() {
    int IP[64] = {
        58 , 50 , 42 , 34 , 26 , 18 , 10 , 2 ,
        60 , 52 , 44 , 36 , 28 , 20 , 12 , 4 ,
        62 , 54 , 46 , 38 , 30 , 22 , 14 , 6 ,
        64 , 56 , 48 , 40 , 32 , 24 , 16 , 8 ,
        57 , 49 , 41 , 33 , 25 , 17 , 9 , 1 ,
        59 , 51 , 43 , 35 , 27 , 19 , 11 , 3 ,
        61 , 53 , 45 , 37 , 29 , 21 , 13 , 5 ,
        63 , 55 , 47 , 39 , 31 , 23 , 15 , 7
    };

    int IP2[64] = {
        40 , 8 , 48 , 16 , 56 , 24 , 64 , 32 ,
        39 , 7 , 47 , 15 , 55 , 23 , 63 , 31 ,
        38 , 6 , 46 , 14 , 54 , 22 , 62 , 30 ,
        37 , 5 , 45 , 13 , 53 , 21 , 61 , 29 ,
        36 , 4 , 44 , 12 , 52 , 20 , 60 , 28 ,
        35 , 3 , 43 , 11 , 51 , 19 , 59 , 27 ,
        34 , 2 , 42 , 10 , 50 , 18 , 58 , 26 ,
        33 , 1 , 41 , 9 , 49 , 17 , 57 , 25
    };

    for (int i = 0; i < 64; i ++) {
        ip[i] = IP[i];
        ip2[i] = IP2[i];
    }
}

int ip[64];
int ip2[64];

```

置换表打印检验：

IP置换表:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP-1置换表:

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

二进制码置换代码:

```
string IPTran(string a) {  
    string temp(64,'0');  
    for (int i = 0; i < 64; i++) {  
        temp[i] = a[ip[i]-1];  
    }  
    return temp;  
}
```

转换效果:

原字符串: 麻辣香锅
转换为二进制: 1100001011101001110000001011000111001111111000111011100111111000
转换为字符串: 麻辣香锅
IP置换后: 101101111100100000010000011110101111111111010101101001000110001

3. 轮函数

轮函数是迭代中的一步, 因此在写迭代之前需要先返程轮函数的实现。轮函数主要有以下运算内容:

(1)将长度为 32 位的串 R_{i-1} 作 E-扩展, 成为 48 位的串 $E(R_{i-1})$

(2)将 $E(R_{i-1})$ 和长度为 48 位的子密钥 K_i 作 48 位二进制串按位异或运算, K_i 由密钥 K 生成

(3)将(2)得到的结果平均分成 8 个分组, 每个分组长度 6 位。各个分组分别经过 8 个不同的 S-盒进行 6-4 转换, 得到 8 个长度分别为 4 位的分组

(4)将(3)得到的分组结果顺序连接得到长度为 32 位的串

(5)将(4)的结果经过 P-置换, 得到的结果作为轮函数 $f(R_{i-1}, K_i)$ 的最终 32 位输出。

首先需要声明定义一个数组作为 E 扩展的参照表（代码略）

E-扩展的代码与 IP 置换相同原理：

```
string EExt(string a) {  
    string temp(48, '0');  
    for (int i = 0; i < 48; i++) {  
        temp[i] = a[E[i]-1];  
    }  
    return temp;  
}
```

接下来需要将扩展后的串与子密钥异或运算。子密钥需要一次性生成 K_1 到 K_{16} ：

```
vector<string> getChildKey(string K) {  
    vector<string> v;  
    string C = TPC1(K).substr(0, 28);  
    string D = TPC1(K).substr(28, 28);  
    for (int i = 1; i <= 16; i++) {  
        int j;  
        if (i == 1 || i == 2 || i == 9 || i == 16) j = 1;  
        else j = 2;  
        while(j--) {  
            char c0 = C[0];  
            char d0 = D[0];  
            for (int k = 0; k < 27; k++) {  
                C[k] = C[k+1];  
                C[k] = D[k+1];  
            }  
            C[27] = c0;  
            D[27] = d0;  
        }  
        string s = C + D;  
        v.push_back(TPC2(s));  
    }  
    return v;  
}
```

需要先对原密钥 PC1 置换，分成两段后分别移位后再合并再 PC2 置换：

```

string TPC1(string K) {
    string temp(56, '0');
    for (int i = 0; i < 56; i++) {
        temp[i] = K[PC1[i]-1];
    }
    return temp;
}

string TPC2(string K) {
    string temp(48, '0');
    for (int i = 0; i < 48; i++) {
        temp[i] = K[PC2[i]-1];
    }
    return temp;
}

```

(PC1 与 PC2 的置换表略)

上面获得子密钥的函数应在迭代中调用后再将其中一个子密钥传入轮函数。

异或函数：

```

string XOR(string a, string b) {
    string temp(48, '0');
    for (int i = 0; i < 48; i++) {
        if (a[i] == b[i]) temp[i] = '0';
        else temp[i] = '1';
    }
    return temp;
}

```

后面的任务是将异或后的串分成 8 段，每段 6 位分别通过各自的 SBOX 后变成 4 位再合并为 32 位。

首先定义获得行列号的函数：

```

int getCol(string s) {
    int n = 0;
    if (num[1] == '1') n += 8;
    if (num[2] == '1') n += 4;
    if (num[3] == '1') n += 2;
    if (num[4] == '1') n += 1;
    return n;
}

int getRow(string s) {
    int n = 0;
    if (num[0] == '1') n += 2;
    if (num[5] == '1') n += 1;
    return n;
}

```

然后让串经过 box (BOX 的 8 个二维数组的定义略):

```

string SBOX(string s) {
    string temp;
    for (int i = 0; i < 8; i++) {
        string substr(4, '0');
        string sub = s.substr(i * 6, 6);
        int num = SBox[i][getRow(sub)][getCol(sub)];
        for (int j = 0; j < 4; j++) {
            if (num & 1) {
                sub[3 - j] = '1';
            }
            num >> 1;
        }
        temp.append(sub);
    }
}

```

最后让串经过 P-置换后返回，则得到轮函数的返回值：


```

string feistel(string R, string K) {
    string E48 = EExt(R);
    string Xor = XOR(E48, K);
    string s = SBOX(Xor);
    string f = PT(s);
    return f;
}

string PT(string s) {
    string temp(32, '0');
    for (int i = 0; i < 32; i++) {
        temp[i] = s[PP[i] - 1];
    }
    return temp;
}

```

4. 16 次迭代

16 次迭代，交换，轮函数调用，异或，最后的 W 置换：

```

string iteration(string s, string k) {
    string l = s.substr(0, 32);
    string r = s.substr(32, 32);
    Feistel func;
    vector<string> childkey = func.getChildKey(k);
    for (int i = 0; i < 16; i++) {
        string temp;
        temp = l;
        l = r;
        r = XOR(temp, func.feistel(r, childkey[i]));
    }
    string temp = r + l;
    return temp;
}

string XOR(string a, string b) {
    string temp(32, '0');
    for (int i = 0; i < 32; i++) {
        if (a[i] == b[i]) temp[i] = '0';
        else temp[i] = '1';
    }
    return temp;
}

```

由于轮函数模块与迭代关系紧密，两者交替调用，而且 16 个子密钥在每一步迭代中分别作为每次调用轮函数的参数，因此迭代与轮函数都写完后才可以开始进行测试。

测试代码：

```
void testFeistel() {
    string t = "麻辣香锅";

    ifstream infile("key.txt");
    stringstream buffer;
    if (!infile.is_open())
    {
        cout << "Failed to open file" << endl;
    }
    buffer << infile.rdbuf();
    string k = buffer.str();

    Iter test;
    IP test2;
    string s = test2.IPTran(test2.S2B(t));
    string result = test2.IPTran2(test.iteration(s, k));
    string s2 = test2.IPTran(result);
    string M = test2.IPTran2(test.iteration2(s2, k));

    cout << "原二进制码: " << test2.S2B(t) << endl;
    cout << "加密后密文: " << result << endl;
    cout << "密文解密后: " << M << endl;
}
```

这里直接将 8 字节 t 作为明文，key.txt 内为已设置好的 64 位密钥
IPTran 将其 IP 置换，iteration 将其迭代 16 次并 W 置换，IPTran2 将其逆置换。得到密文 result。然后对 result 作同样的操作（iteration 变为 iteration2，即迭代顺序相反（指子密钥调用））。

```

// DES.cpp
string iteration(string s, string k) {
    string l = s.substr(0, 32);
    string r = s.substr(32, 32);
    Feistel func;
    vector<string> childkey = func.getChildKey(k);
    for (int i = 0; i < 16; i++) {
        string temp;
        temp = l;
        l = r;
        r = XOR(temp, func.feistel(r, childkey[i]));
    }
    string temp = r + l;
    return temp;
}

string iteration2(string s, string k) {
    string l = s.substr(0, 32);
    string r = s.substr(32, 32);
    Feistel func;
    vector<string> childkey = func.getChildKey(k);
    for (int i = 0; i < 16; i++) {
        string temp;
        temp = l;
        l = r;
        r = XOR(temp, func.feistel(r, childkey[15-i]));
    }
    string temp = r + l;
    return temp;
}

```

最终测试结果如下：

```

D:\Course\web安全技术\DES\code\DES.exe
原二进制码: 1100001011101001110000001011000111001111111000111011100111111000
加密后密文: 1010100110010001100110100111000110101100100110110110000011110000
密文解密后: 1100001011101001110000001011000111001111111000111011100111111000
-----
Process exited after 0.04138 seconds with return value 0
请按任意键继续. . .

```

数据结构：

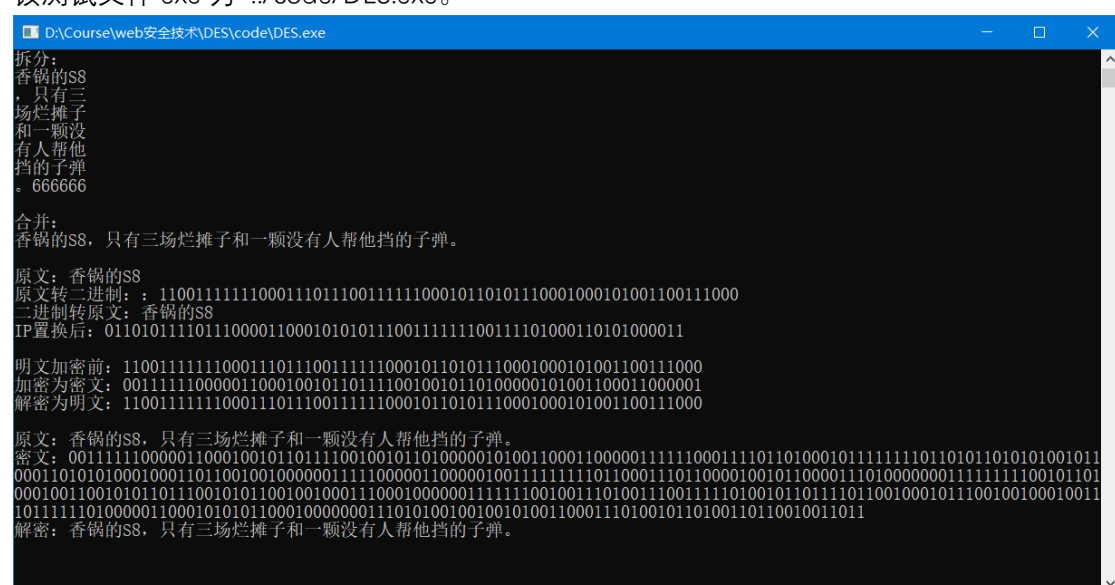
IP 置换表	64 位一维数组
IP ⁻¹ 置换表	64 位一维数组
E-扩展规则（比特-选择表）	48 位一维数组
S-盒	8×4×16 三维数组
P-置换表	32 位一维数组
PC-1 置换表	56 位一维数组
PC-2 压缩置换表	48 位一维数组
子密钥	数量 16 的元素为 string 的 vector 容器

C 语言源代码：

已将代码按模块分解在上面贴出解释功能，全部源代码可在附带文件中看到。由于自顶向下编程，模块较多，则不在文档内贴出。关于算法的主要代码都可在上文的模块分解中结合模块查看。

编译运行结果：

该测试文件 exe 为 ../code/DES.exe。



```
D:\Course\web安全技术\DES\code\DES.exe
拆分：
香锅的S8
，只有三
场烂摊子
和一颗没
有人帮他
挡的子弹
。666666

合并：
香锅的S8，只有三场烂摊子和一颗没有人帮他挡的子弹。

原文：香锅的S8
原文转二进制：：1100111111100011101110011111100010110101110001000101001100111000
二进制转原文：香锅的S8
IP置换后：0110101111011100001100010101011100111111100111101000110101000011

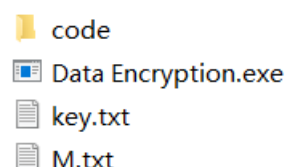
明文加密前：1100111111100011101110011111100010110101110001000101001100111000
加密为密文：00111111100000110001001011011110010010110100000101001100011000001
解密为明文：1100111111100011101110011111100010110101110001000101001100111000

原文：香锅的S8，只有三场烂摊子和一颗没有人帮他挡的子弹。
密文：00111111100000110001001011011110010010110100000100011000110000011111100011110110100010111111101101011010101001011
0001101010100010001101100100100000011111000001100000100111111110110001110110000100101100001110100000001111111100101101
000100110010101101110010111001001000111000100000011111100100111010011100111110100101101110110010001011100100100010011
1011111101000001100010101011000100000001110101001001010011000111010010110100110100110100110100110110010011011
解密：香锅的S8，只有三场烂摊子和一颗没有人帮他挡的子弹。
```

根目录中的 code 文件内为测试用的代码与所有代码文件。




同时分别编写了加密程序与解密程序供校验。

在地址 ../明文加密/ 中，可以看到以下四个文件：



其中 code 文件夹内有加密程序的相关代码与编译程序。key.txt 为 64 位密钥，M.txt 为需要加密的原文。若我们运行 Data Encryption.exe，该目录下会生成一个 C.txt，为二进制密文。

在地址 ../密文解密/ 中，可以看到以下三个文件：

 code
 Data Decryption.exe
 key.txt

其中 code 文件夹内有解密程序的相关代码与编译程序。实际上三个 code 文件夹中的代码文件都大部分相同，只有编写 main 函数的那个 cpp 文件内容不一样。key.txt 为 64 位密钥。这里的密钥与明文加密文件夹中的密钥相同，我们可以将上面运行加密程序后生成的二进制密文复制到该目录下，运行 Data Decryption.exe，该目录下会生成一个 M.txt，为解密后的原文。

若需要校验，可以修改加密文件夹中的 M 中的原文，运行加密程序，得到对应密文，复制密文到解密文件夹下运行解密程序，得到原文验证。同样可以修改 key（需要确保为 64 为二进制字符串），来得到不同的密文测试。若有对应的测例，可以以此得到密文对照密文是否符合算法运算结果。这里通过两个文件夹来模拟加密端和解密端两端。

总而言之，想直接查看设好的测例的结果，可直接运行 `../code/DES.exe`，想测试加密解密效果则按照上文操作即可。