Lab2——分组密码算法DES

学号：2013921
姓名：周延霖
年级：2020级
专业：信息安全

一、实验内容说明

1、实验目的

通过用DES算法对实际的数据进行加密和解密来深刻了解DES的运行原理。

2、实验要求

1. 分别实现DES的加密和解密，提交程序代码和执行结果。
2. 在检验雪崩效应中，要求至少改变明文和密文中各八位，给出统计结果并计算出平均值。

3、实验步骤

1. **算法分析：** 对课本中DES算法进行深入分析，对初始置换、E扩展置换，S盒代换、轮函数、密钥生成等环节要有清晰的了解，并考虑其每一个环节的实现过程。
2. DES实现程序的总体设计：在第一步的基础上，对整个DES加密函数的实现进行总体设计，考虑数据的存储格式，参数的传递格式，程序实现的总体层次等，画出程序实现的流程图。
3. 在总体设计完成后，开始具体的编码，在编码过程中，注意要尽量使用高效的编码方式。
4. 利用3中实现的程序，对DES的密文进行雪崩效应检验。即固定密钥，仅改变明文中的一位，统计密文改变的位数；固定明文，仅改变密钥中的一位，统计密文改变的位数。

二、实验环境

- 操作系统：`macOS Montery 12.4`
- 软件系统：`Xcode`
- 编译工具：`Apple clang version 13.1.6 (clang-1316.0.21.2.5)`
- 编程语言：`C++`

三、实验过程

本次实验首先翻阅课本，对理论课上的知识进行回顾，然后设计整个实验的流程图以及各个结构体和函数的大致思路，然后进行具体代码的编写实现，以下为具体过程：

1、流程分析

DES分析

可以将其分为以下几步：

- 子秘钥生成
- 初始置换 IP
- 密码函数 f
- 尾置换 IP-1

考虑到我们的存储格式，由于我们进行的大部分是位运算，为了更加方便的进行代码编程，我们首先考虑将数据格式转为bit格式，这里我们使用bitset<64>。

用户输入的秘钥（key）、明文（origin）、密文（secret）均设定为变量类型为bitset<64>全局变量。

第一步.数据处理

首先我们编写处理函数，void getbit(string a, bitset<64>& temp)能将我们输入的十六进制字符串转换为bitset<64>,而void getHex(bitset<64> b,string &s)则起到相反的作用，我们将数据处理完之后，存入全局变量中。

第二步.子秘钥生成

首先在我们进行加解密之前，我们要生成好16组48位秘钥。

接下来我们已经获取到了秘钥key，我们通过函数 void getKeys()将生成的子秘钥存到bitset<48> sonKey[16]中。

getKeys()函数流程如下：

- 去掉奇偶校验位64->56，并根据压缩置换表pc_1进行置换，但我们可以注意到，压缩置换表中已经省去了奇偶校验位，所以我们可以跳过去掉奇偶校验位的过程，直接按照pc_1表进行压缩置换，置换过程如下：
  - **由于以后置换全部采取这种方法，所以后续置换就不详细展示**

```
for (int i = 0; i < 56; i++)
{
    firstKey[55 - i] = key[64 - PC_1[i]];
}
```

- 循环十六次
  - 从sonKey[16]中取出一组，将该组56位子秘钥分为高低位两部分，分别根据移位表进行左移调用leftShift(bitset<28> k, int shift 左移位数)，再将位移后的两部分重新拼接，再对照着PC_2表，压缩置换为48位，我们将压缩后的秘钥重新放回sonKey[16]

第三步.DES加密

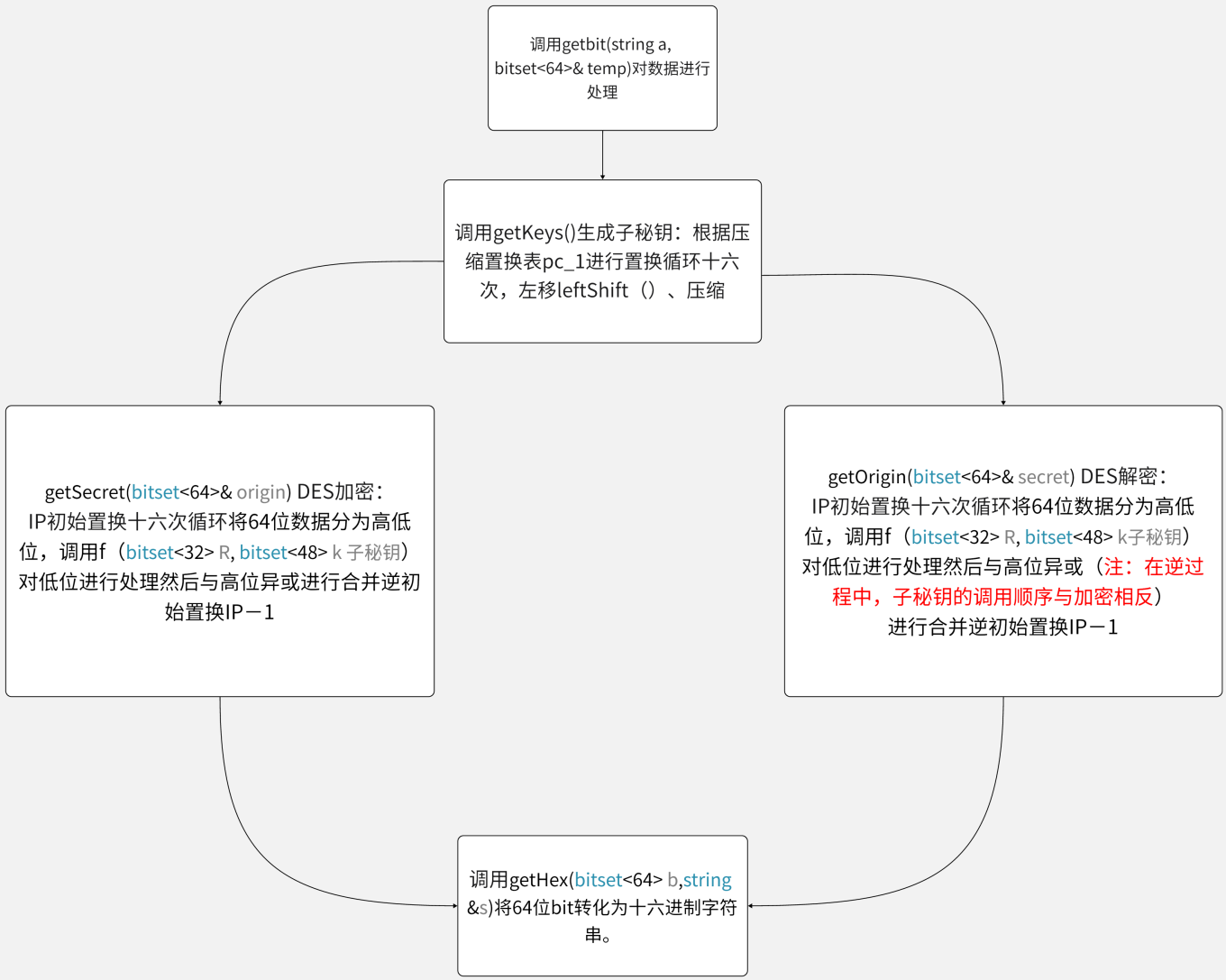
1. IP初始置换，按照IP表对明文进行置换
2. 将64位数据分为高低位，调用`bitset<32> f(bitset<32> R, bitset<48> k)`对低位进行处理，得到的结果与高位进行异或，然后互换新的高低位位置，进行十六次迭代
3. 进行合并，注意合并时原本高低位位置发生调换
4. 逆初始置换IP－1

重要函数bitset<32> f(bitset<32> R, bitset<48> k 子秘钥)流程:

- 根据表E将32扩展为48
- 与子秘钥异或运算
- 查找S_BOX置换表，分成8组，每个6位，其中最高位与最低位组成行号，剩下4位组成列号
- P-1置换

流程图

- DES密码流程图如下图所示：



2、代码实现

各类的结构体（即书本上出现的置换表以及各种扩展和置换运算）已经预先定义

各类预处理函数

- 首先是二进制与字符的互相转换

```
// 将string类型转换为bits类型
void getbit(string a, bitset<64>& temp)
{
    int num = 63;
    for (int i = 2; i < a.length(); i++)
    {
```

```
        if (a[i] <= '9')
        {
            for (int j = 0; j < 4; j++)
            {
                temp[num--] = HexToBit[a[i] - 48][j];
            }
        }
        else
        {
            for (int j = 0; j < 4; j++)
            {
                temp[num--] = HexToBit[a[i] - 65 + 10][j];
            }
        }
    }
}

// 将bits类型转换为string类型
void getHex(bitset<64> b,string &s)
{
    s[0] = '0';
    s[1] = 'x';
    int num = 2;
    for (int i = 63; i >= 0; i-=4)
    {
        int a = b[i] * 8 + b[i - 1] * 4 + b[i - 2] * 2 + b[i - 3];
        if (a > 9)
            s[num++] = 'A' + a - 10;
        else
            s[num++] = '0' + a;
    }
}
```

- 接下来是左循环移位函数

```
// 左循环移位
bitset<28> leftShift(bitset<28> k, int shift)
{
    bitset<28> tmp = k;
    for (int i = 27; i >= 0; --i)
    {
        if (i - shift < 0)
            k[i] = tmp[i - shift + 28];
        else
            k[i] = tmp[i - shift];
    }
    return k;
}
```

- 然后是F函数的运算，包括S盒代换、扩展运算、子密钥的异或以及P置换操作

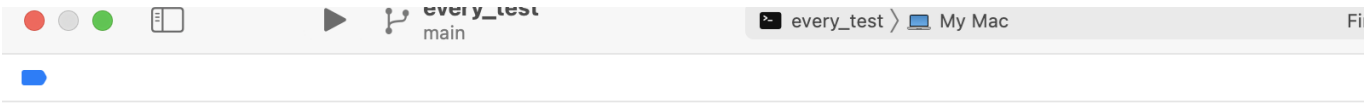
```
// 轮函数的计算并返回一个32bits的值
bitset<32> f(bitset<32> R, bitset<48> k)
{
    // S盒代换时需要用到的计数器
    int temp = 0;

    // 预定义扩展运算的结果
    bitset<48> expandR;

    // 进行E扩展运算
    for (int i = 0; i < 48; ++i)
    {
        expandR[47 - i] = R[32 - E[i]];
    }

    // 与子密钥进行异或运算
    expandR = expandR ^ k;

    // 进行S盒代换操作
    bitset<32> output;
    for (int i = 0; i < 48; i = i + 6)
    {
        int row = expandR[47 - i] * 2 + expandR[47 - i - 5];
        int col = expandR[47 - i - 1] * 8 + expandR[47 - i - 2] * 4 +
expandR[47 - i - 3] * 2 + expandR[47 - i - 4];
        int num = S_BOX[i / 6][row][col];
        bitset<4> binary(num);
        output[31 - temp] = binary[3];
        output[31 - temp - 1] = binary[2];
        output[31 - temp - 2] = binary[1];
        output[31 - temp - 3] = binary[0];
        temp += 4;
    }

    // 进行P置换操作
    bitset<32> tmp = output;
    for (int i = 0; i < 32; ++i)
    {
        output[31 - i] = tmp[32 - P[i]];
    }

    return output;
}
```

加密过程

- 加密函数首先进行IP初始置换，按照IP表对明文进行置换
- 然后将64位数据分为高低位，调用`bitset<32> f(bitset<32> R, bitset<48> k)`对低位进行处理，得到的结果与高位进行异或，然后互换新的高低位位置，进行十六次迭代
- 接下来进行合并，注意合并时原本高低位位置发生调换
- 最后进行逆初始置换IP－1，具体函数如下：

```
// 加密函数
bitset<64> getSecret(bitset<64>& origin)
{
    bitset<64> secret;
    bitset<64> firstInput;
    bitset<32> high;
    bitset<32> low;

    // 初始置换IP
    for (int i = 0; i < 64; i++)
    {
        firstInput[63 - i] = origin[64 - IP[i]];
    }

    // 将其分为左右两部分
    for (int i = 32; i < 64; i++)
    {
        high[i - 32] = firstInput[i];
    }
    for (int i = 0; i < 32; i++)
    {
        low[i] = firstInput[i];
    }

    // 进行16轮的循环
    for (int round = 0; round < 16; round++)
    {
        bitset<32> nextHigh;
        nextHigh = low;
        low = high ^ f(low, sonKey[round]);
        high = nextHigh;
    }

    // 左右交换
    for (int i = 0; i < 32; i++)
    {
        secret[i] = high[i];
        secret[i+32] = low[i];
    }

    // 逆初始置换
    firstInput = secret;
    for (int i = 0; i < 64; i++)
    {
        secret[63 - i] = firstInput[64 - IP_1[i]];
    }

    return secret;
}
```

- 最后的加密输出结果如下：




```
================================================================================
粥小霖的DES加密器，你可以输入以下数字进行相应操作：
0.检测雪崩
1.DES加密
2.DES解密
================================================================================
1
--------------------------------------------------------------------------------
`加密`:
明文：0x0000000000000000
密钥：0x10316E028C8F3B4A
正确结果：0x82DCBAFBDEAB6602
加密结果：0x82DCBAFBDEAB6602
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
`加密`:
明文：0x95F8A5E5DD31D900
密钥：0x0101010101010101
正确结果：0x8000000000000000
加密结果：0x8000000000000000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
`加密`:
明文：0xDD7F121CA5015619
密钥：0x0101010101010101
正确结果：0x4000000000000000
加密结果：0x4000000000000000
--------------------------------------------------------------------------------
Program ended with exit code: 0
```

可以看到加密结果与正确结果一样

> 解密过程

- 解密函数与加密过程类似，只是子秘钥的调用顺序与加密相反，具体函数如下：

```
// 解密函数
bitset<64> getOrigin(bitset<64>& secret)
{
    bitset<64> origin;
    bitset<64> firstInput;
    bitset<32> high;
    bitset<32> low;

    // 初始置换IP
    for (int i = 0; i < 64; i++)
        firstInput[63 - i] = secret[64 - IP[i]];

    // 将其分为左右两部分
    for (int i = 32; i < 64; i++)
        high[i - 32] = firstInput[i];
    for (int i = 0; i < 32; i++)
        low[i] = firstInput[i];
```

```
    // 进行16轮的循环
    bitset<32> nextHigh;
    for (int round = 0; round < 16; round++)
    {
        nextHigh = low;
        low = high ^ f(low, sonKey[15 - round]);
        high = nextHigh;
    }

    // 左右交换
    for (int i = 0; i < 32; i++)
    {
        origin[i] = high[i];
        origin[i+32] = low[i];
    }

    // 逆初始置换
    firstInput = origin;
    for (int i = 0; i < 64; i++)
        origin[63 - i] = firstInput[64 - IP_1[i]];

    return origin;
}
```

- 最后的解密输出结果如下：

可以看到解密结果与正确结果一样

> 雪崩效应

- 在输入为0时进行雪崩效应
- 先固定密钥，每次改变一个明文，进行八次，观察更改的结果的不同位数
- 然后固定明文，每次改变一个密钥，进行八次，观察更改的结果的不同位数，具体函数如下：

```
if (a == 0)
    {
        cout <<
"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++" << endl;
        string a_pre = "0x95F8A5E5DD31D900";
        string b_pre = "0x0101010101010101";

        int num[8] = {0,0,0,0,0,0,0,0};
        int num_pre = 0;

        // 需要改变的明文
        string a[8] = {
"0x95F8A5E5DD31D901","0x95F8A5E5DD31D902","0x95F8A5E5DD31D904","0x95F8A5E5
```

```
DD31D908",

"0x95F8A5E5DD31D910","0x95F8A5E5DD31D920","0x95F8A5E5DD31D940","0x95F8A5E5
DD31D980"
        };
        string b[8] = {
"0x0101010101010111","0x0101010101011101","0x0101010101110101","0x01010101
11010101",

"0x0101011101010101","0x0101110101010101","0x0111010101010101","0x11010101
01010101"
        };

        cout << "接下来每次改变一位明文，观察结果" << endl;
        for(int w = 0;w < 8;w++)
        {
            // 正确的密文
            string c = "0x8000000000000000";

            getbit(a[w], origin);
            getbit(c, first);
            getbit(b_pre, key);

            // 获取密钥
            getKeys();

            // 进行加密
            result = getSecret(origin);
            cout << "原始明文: " << a_pre << endl << "改后明文: " << a[w] <<
endl;
            cout << "原始结果: " << first << endl << "改后结果: " << result <<
endl;
            for (int i = 0; i < 64; i++)
            {
                if (first[i] != result[i])
                {
                    num[w]++;
                }
            }
            cout << "不同的数字位数: " << num[w] << endl;
        }
        for(int w = 0;w < 8;w++)
        {
            num_pre += num[w];
        }
        cout << "改变明文获得不同的数字位数的平均数为: " << double(num_pre) / 8
<< endl;

        // 重新初始化，开始更改密钥
        num_pre = 0;
        for(int w = 0;w < 8;w++)
        {
            num[w] = 0;
        }
```

```
        cout << "-------------------------------------------------------------
-------------------------" << endl;

        cout << "接下来每次改变一位密钥，观察结果" << endl;
        for(int w = 0;w < 8;w++)
        {
            // 正确的密文
            string c = "0x8000000000000000";

            getbit(a_pre, origin);
            getbit(c, first);
            getbit(b[w], key);

            // 获取密钥
            getKeys();

            // 进行加密
            result = getSecret(origin);
            cout << "原始密钥: " << b_pre << endl << "改后密钥: " << b[w] <<
endl;
            cout << "原始结果: " << first << endl << "改后结果: " << result <<
endl;
            for (int i = 0; i < 64; i++)
            {
                if (first[i] != result[i])
                {
                    num[w]++;
                }
            }
            cout << "不同的数字位数: " << num[w] << endl;
        }
        for(int w = 0;w < 8;w++)
        {
            num_pre += num[w];
        }
        cout << "改变明文获得不同的数字位数的平均数为: " << double(num_pre) / 8
<< endl;


        cout <<
"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++" << endl;
    }
```

- 最后的雪崩效应的输出结果如下：

改变明文：

使秘钥不变，每次仅更改一位明文，重复八次，平均每次有35.125位发生变化

改变密钥：

使明文不变，每次仅更改一位密钥，重复八次，平均每次有33.5位发生变化

综上可以看出即使小小的改动，加密结果也会发生很大的变化

四、总结与展望

1、总结

本次是密码学的第二次实验，在这次实验中对理论课上讲解的DES加解密进行编程，使得对于这种密码算法的原理和攻击方法更加的了解，也对密码学方面的编程更加的熟悉。

2、展望

在这次实验后，对密码学方面的知识更加的期待，也对这些原理和攻击方法更加的感兴趣，期待这学期自己更好的发展，万事胜意、心想事成。