信息安全第一次作业 简单DES的实现

数据科学与计算机学院 17343094 彭湃

本次的作业是实现简单的`DES`算法，我们下面将会从`算法概述`、`总体结构`、`模块分解`、`数据结构`、`源代码`、`运行结果`六个方面完成我们的程序设计和分析。

一、算法概述

- `DES`是一种典型的快加密算法，它采用64位作为分组的长度，每次输入64位一组的明文，经过相关操作，输出64位的密文。
- `DES`是采用密钥定义变换过程的，所以理论上来说，只有持有加密所用密钥的用户才能对密文进行解密。
- `DES`的密钥确实也是34位，但是它的有效长度只有56位，因为每8位就会有一位用于奇偶校验（定义这一位是最后一位）。密钥可以是任意的56位数字，且可以随时改变。
- 说白了，其实`DES`算法的基本操作就是置换和换位。

1.信息空间

信息空间由{0,1}组成的字符串构成。无论是原始明文，还是密文，亦或是密钥，都是64位（8个字节）。

如果原始明文不足8个字节的时候，此时我们需要进行填充。即：

- 如果最后的明文不够8个字节时，则在末尾以字节填满。填的数据是补上的字节数。
- 如果最后的明文刚好分组完全的时候，则在末尾填充上8个字节（增加一个新的分组），且数据都是08。（我们理解成我们加了8个字节）

- 分组结构：
  - 明文：$M = m_1 m_2 m_3 \ldots\ldots m_{64}$，　其中$m_i \in \{0,1\}$
  - 密文：$C = c_1 c_2 c_3 \ldots\ldots c_{64}$，　其中$c_i \in \{0,1\}$
- 密钥：$K = k_1 k_2 k_3 \ldots\ldots k_{64}$，　其中$k_i \in \{0,1\}$　但是k_8、K_16、$\ldots\ldots k_{64}$是校验位

2.加密/解密过程

加密过程，简而言之可以用下面的公式来解释。

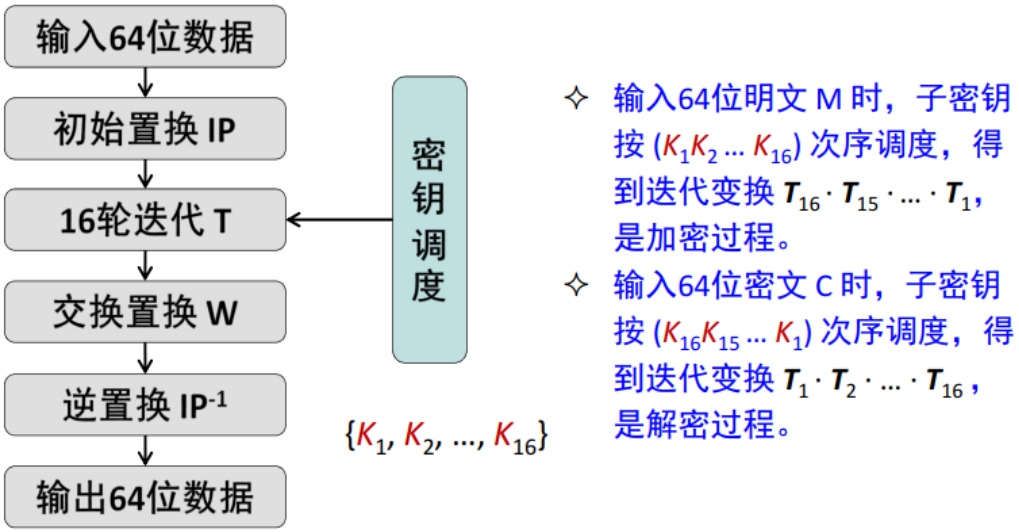
$$C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \ldots\ldots \cdot T_1 \cdot IP(M)$$

- $E_k(M)$ 是描述以`k`为密钥的加密函数。
- `M` 是初始的64位明文.
- `C`是最后输出的密文块。
- `IP` 是初始的64位置换。
- $T_{16} \ldots\ldots T_1$是一系列的迭代变换。
- `W` 是一个置换操作，将输入的高32位和低32位交换后输出 。
- IP^{-1} 是`IP`的逆置换.

解密过程的公式可以归纳成是：

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

事实上，加密和解密的其它步骤都是相同的，只有在进行16轮迭代的时候会有差别，即在于子密钥调度的顺序，一个是按照$K_1 K_2 K_3 \ldots\ldots K_{16}$次序调度，得到迭代变换$T_{16} \ldots\ldots T_1$，而另外一个恰恰相反。



二、总体结构

没有采用头文件、主文件分离的方法来写，所有程序都放在一个c++文件了。

`Des.cpp`：可以按照功能将其简单的分为以下几个大部分：

- 表格区：包含`IP_Table`、`E_Table`、`P_Table`、`PC1_Table`、`PC2_Table`、`IPinverse_Table`、`S1-8_Box`。

- 主体功能函数区：`DES` 算法函数的具体实现部分，如`初始置换IP` 、`迭代` 、`E拓展`等功能的实现。
- main函数区：其实叫`UI` 区也许更合适些。

三、模块分解

整个`DES`算法，我觉得可以大致分为四个模块，分别是`IP置换`、`16轮迭代`、`设置子密钥`、`IP逆置换` 。其中`设置子密钥` 和 `轮迭代与轮函数`则是关键，也最为复杂。

轮迭代与轮函数:

- 高32位和低32位置换
- `Fesitel` 函数
    - E-拓展
    - 子密钥异或
    - S-转换
    - P-置换

设置子密钥:

- PC1置换
- 二进制串循环左移
- PC2压缩置换

下面我们将结合我们的代码进行具体分析。

1.IP置换

这个过程其实就是我们按照特定的表(即`IP_Table`)，对我们的明文块进行重排，表格老师的ppy上面已经给出了，就是复制进程序的时候一个数一个数后面打逗号比较麻烦。

```
const static int IP_Table[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
};
```

然后我们将输入的64位明文块按照这个规则表变换成输出块。其中，前32位我们存进`left`，后32位我们存进`right` 。即$M_0 = IP(M) = L_0R_0$.

```
int temp[64];
for(int i = 0; i < 64; i++)
{
    temp[i] = input[IP_Table[i] - 1];
}
for(int i = 0; i < 64; i++)
{
```

```
    if(i<32)
    {
        left[i] = temp[i];
    }
    else
    {
        right[i-32] = temp[i];
    }
}
```

2.设置子密钥

我们根据给定的64位的密钥，生成16个48位的子密钥，供我们后面的`Fesitel`函数调用。

- 因为64位中，每八位的最后一位便是校验位，所以实际上起作用的只有56位，我们对这56位进行`PC-1`置换，得到C_0和D_0，其中C_0是前28位，D_0是后28位。 因为ppt上已经给了现成的`PC-1`置换表，所以我们按照它的格式来设置就好。

```
int temp[56];
for(int i = 0; i < 56; i++)
{
    temp[i] = input[PC1_Table[i] - 1];
}
for(int i = 0; i < 56; i++)
{
    if(i < 28)
    {
        c[i] = temp[i];
    }
    else
    {
        d[i-28] = temp[i];
     }
}
```

- 然后我们需要计算$LS_i(C_{i-1})$和$LS_i(D_{i-1})$。当$i = 1, 2, 9, 16$的时候，我们将其循环左移一个位置，其他情况，我们将其循环左移两个位置。

```
void LeftShift(const int input[28], int output[28], int counter)
{
    for(int i = 0; i < 28; i++)
    {
        output[i] = input[(i + counter) % 28];
    }
}
```

```
for(int i = 1; i <= 16; i++)
{
    if(i == (1 || 2 || 9 || 16))
    {
        leftCount += 1;
    }
    else
    {
        leftCount += 2;
    }
    LeftShift(C, Ci[i - 1], leftCount);
    LeftShift(D, Di[i - 1], leftCount);
}
```

- 现在C_i和D_i 均是28位，我们将56位的C_iD_i进行 `PC-2` 压缩置换，得到48位的K_i. 按照老师给定的 `PC2_Table` 来就好。

```
for(int i = 0; i < 56; i++)
{
    if(i < 28)
    {
        temp[i] = c[i];
    }
    else
    {
        temp[i] = d[i-28];
    }
}
for(int i = 0; i < 48; i++)
{
    output[i] = temp[PC2_Table[i]-1];
}
```

最后我们在`void subKey(const int input[64], int output[16][48])` 中将其整合，子密钥就生成好啦。

3.轮迭代与轮函数

- **轮迭代**：首先我们需要按照给定好的规则对L_i和R_i 进行16次的轮迭代。

  $L_i = R_{i-1}, R_i = L_{i-1} \bigoplus f(R_{i-1}, k_i)$

  这里面的k_i由密钥K生成。

```
for(int i = 1; i <= 16; i++)
{
    for(int j = 0; j < 32; j++)
    {
        L[i][j] = R[i - 1][j];
    }
    int temp[32] = {0};
    //轮函数处理
    Feistel(R[i - 1], subkeys[i - 1], temp);
    //异或处理
    XOR(temp, L[i - 1], R[i], 32);
}
```

16次迭代后我们会得到$L_{16}R_{16}$，此时我们需要将其左右交换，得到$R_{16}L_{16}$.

```
int temp[32] = {0};
for(int i = 0; i < 32; i++)
{
    temp[i] = L[16][i];
    L[16][i] = R[16][i];
    R[16][i] = temp[i];
}
```

- **轮函数**:

  我们先对串做`E-拓展`，使其从32位变成48位，得到$E(R_{i-1})$。按照给出来的`E-Table`操作即可。

```
for(int i = 0; i < 48; i++)
{
    output[i] = input[E_Table[i] - 1];
}
```

然后我们将上面一步得到的$E(R_{i-1})$和子密钥进行异或操作。

```
XOR(temp, subkey, temp2, 48);
```

我们将上面一步得到的结果平均分成8个组，每个组6位，然后对应$s_1 \ldots\ldots s_8$进行6 -> 4的操作，最后变成了8个四位的分组。并且将其顺序连接起来成为串。

```
void S(const int input[48], int output[32])
{
  int a1[6], a2[6], a3[6], a4[6], a5[6], a6[6], a7[6], a8[6];
  int b1[4], b2[4], b3[4], b4[4], b5[4], b6[4], b7[4], b8[4];
  for(int i = 0; i < 6; i++)
  {
    a1[i] = input[i];
    a2[i] = input[i + 6];
    a3[i] = input[i + 2 * 6];
    a4[i] = input[i + 3 * 6];
    a5[i] = input[i + 4 * 6];
    a6[i] = input[i + 5 * 6];
    a7[i] = input[i + 6 * 6];
    a8[i] = input[i + 7 * 6];
```

```
  }
  int n, m;
  n = (a1[0] << 1) + a1[5];
  m = (a1[1] << 3) + (a1[2] << 2) + (a1[3] << 1) + a1[4];

  n = (a2[0] << 1) + a2[5];
  m = (a2[1] << 3) + (a2[2] << 2) + (a2[3] << 1) + a2[4];

  n = (a3[0] << 1) + a3[5];
  m = (a3[1] << 3) + (a3[2] << 2) + (a3[3] << 1) + a3[4];

  n = (a4[0] << 1)+ a4[5];
  m = (a4[1] << 3) + (a4[2] << 2) + (a4[3] << 1) + a4[4];

  n = (a5[0] << 1) + a5[5];
  m = (a5[1] << 3) + (a5[2] << 2) + (a5[3] << 1) + a5[4];

  n = (a6[0] << 1) + a6[5];
  m = (a6[1] << 3) + (a6[2] << 2) + (a6[3] << 1) + a6[4];

  n = (a7[0] << 1) + a7[5];
  m = (a7[1] << 3) + (a7[2] << 2) + (a7[3] << 1) + a7[4];

  n = (a8[0] << 1) + a8[5];
  m = (a8[1] << 3)+ (a8[2] << 2) + (a8[3] << 1) + a8[4];

  for (int i = 0; i < 4; i++)
  {
    b1[3 - i] = (S1_Box[n][m] >> i) & 1;
    b2[3 - i] = (S2_Box[n][m] >> i) & 1;
    b3[3 - i] = (S3_Box[n][m] >> i) & 1;
    b4[3 - i] = (S4_Box[n][m] >> i) & 1;
    b5[3 - i] = (S5_Box[n][m] >> i) & 1;
    b6[3 - i] = (S6_Box[n][m] >> i) & 1;
    b7[3 - i] = (S7_Box[n][m] >> i) & 1;
    b8[3 - i] = (S8_Box[n][m] >> i) & 1;
    }

  for(int i = 0; i < 4; i++)
  {
    output[i] = b1[i];
    output[i + 4] = b2[i];
    output[i + 2 * 4] = b3[i];
    output[i + 3 * 4] = b4[i];
    output[i + 4 * 4] = b5[i];
    output[i + 5 * 4] = b6[i];
    output[i + 6 * 4] = b7[i];
    output[i + 7 * 4] = b8[i];
  }
}
```

最后进行`P-压缩置换`就可以啦。也有现成的`P_Table`.跟上面的差不多，这里就不多说了。

4.IP逆变换

最后一步，逆置换就可以完成我们的加密过程。

```
for (int i = 0; i < 64; i++)
{
    output[i] = input[IPInverse_Table[i] - 1];
}
```

```
IPInverse(output_1, output);
```

四、数据结构

1.创建明文、密文与密钥

```
int CipherText[64] = {0};
char PlainText[9] = {0};
char key[9] = { 0 };
```

2.加密测试部分

```
cout << "please enter your plaintext!(must be eight number or character)" <<
endl;
cout << "Plaintext: ";
cin >> PlainText;
cout << "please enter your secret_key!(must be eight number or character)"
<<endl;
cout<< "SecretKey: ";
cin >> key;

cout<<endl;
cout<<"Encryption in progress"<<endl;
cout<<"----------------------"<<endl;
encryption(PlainText, key, CipherText);
cout<<"your ciphertext is: "<<endl;
for (int i = 0; i < 64; i++)
{
    cout<<CipherText[i];
    if ((i + 1) % 8 == 0)
    {
        cout<<endl;
    }
}
```

3.解密验证部分

```
cout << endl << "Decryption in progress: " <<endl;
cout<<"----------------------"<<endl;
Decryption(CipherText, key, PlainText);
cout<<"your plaintext is:"<<endl;
for (int i = 0; i<8; i++)
{
    cout<<PlainText[i];
}
cout<<endl<<endl;
```

4.相关表格

```
const static int IPInverse_Table[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
};
……
```

五、c++源代码

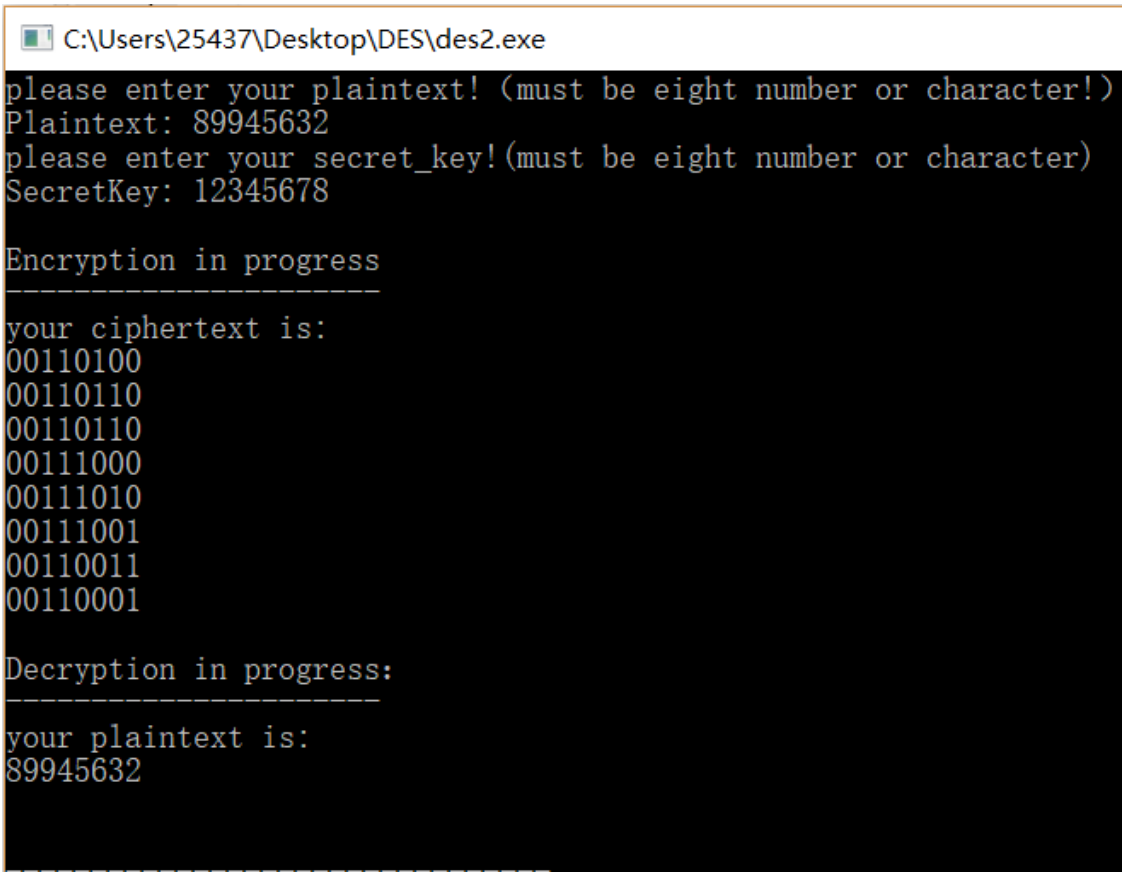
其实主要部分的代码在模块分解的时候已经说的很清晰了，详细的`des.cpp`可见我们的附件。

- `src` : des2.cpp 源代码
- `bin` : des2.exe 执行文件

六、运行结果

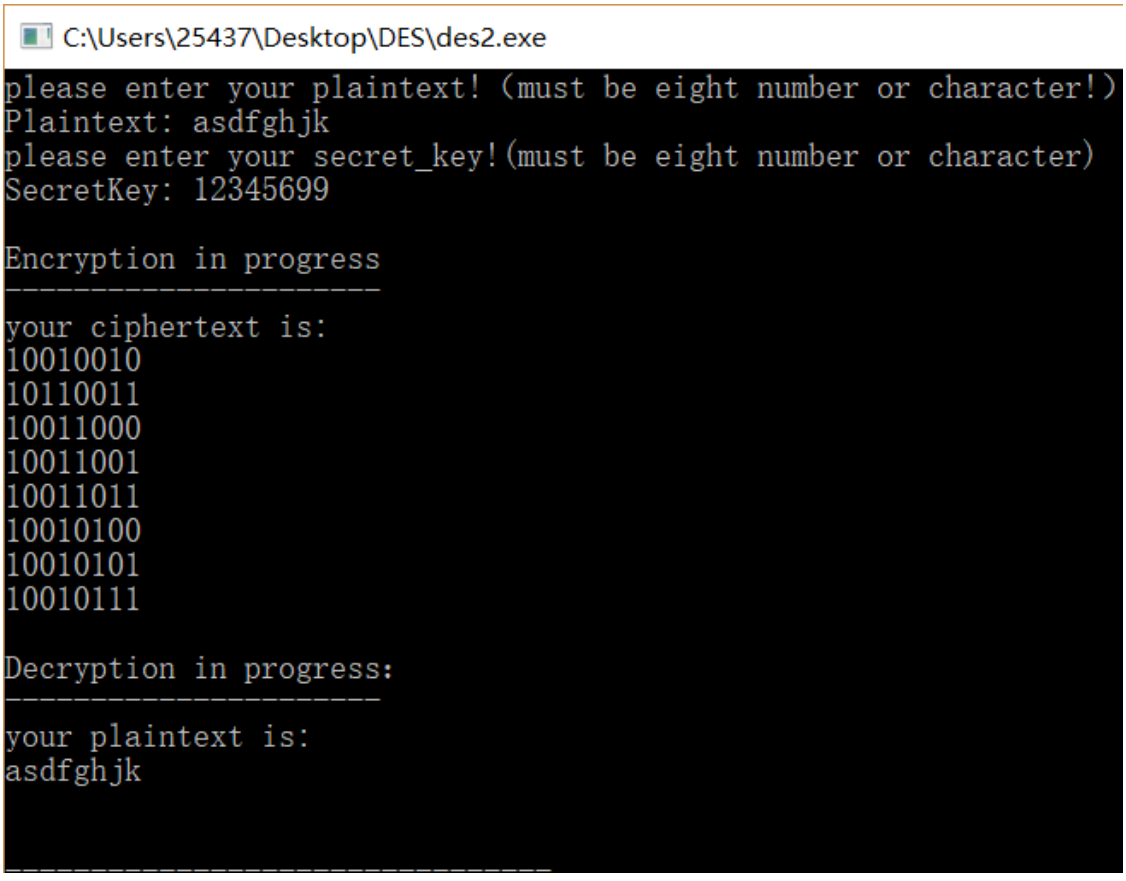
为了方便起见，我们限定我们输入的明文和密钥必须都是8位。否则会直接退出。

- 测试明文是数字的情况



经解密对照，可发现我们的加密过程是正确的。

- 测试明文是字母的情况：



经比照，正确无误。

在写这个程序的过程中，参考了不少优秀的博客和代码，在此一并感谢。