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 = Ek(M) = IP-1·W·T16·T15·…·T1·IP(M)

* M为算法输入的64位明文块；
* Ek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；
* T1，T2，…，T16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 = Dk(C) = IP-1·W·T1·T2·…·T16·IP(C)

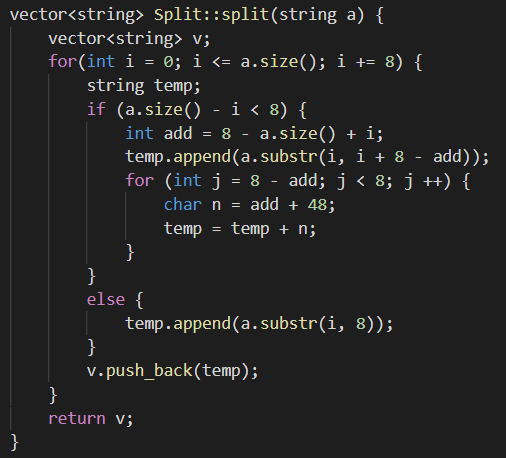
### 模块分解：

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。

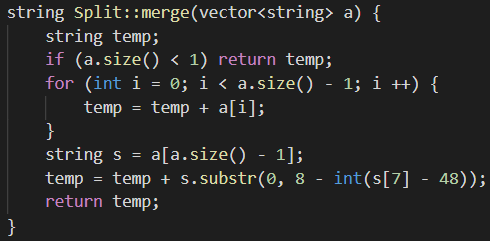
1. **信息空间**

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。

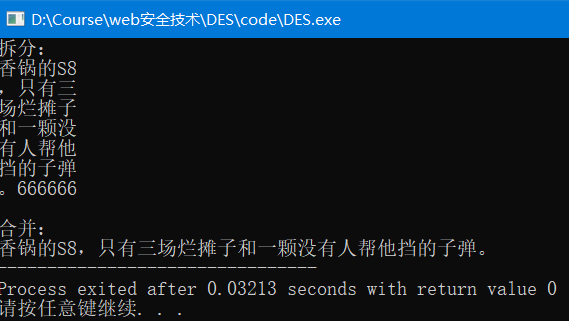
分割实现：



合并实现：

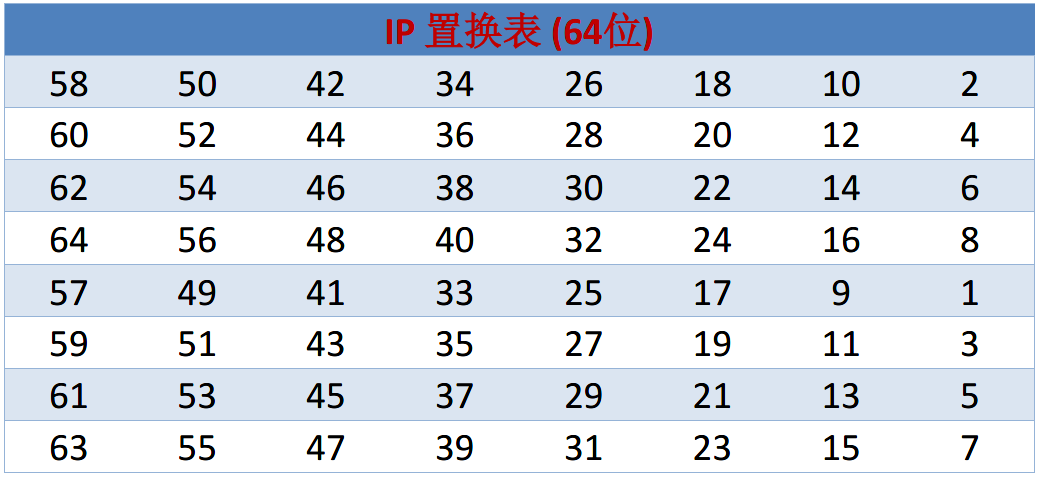


测试效果：



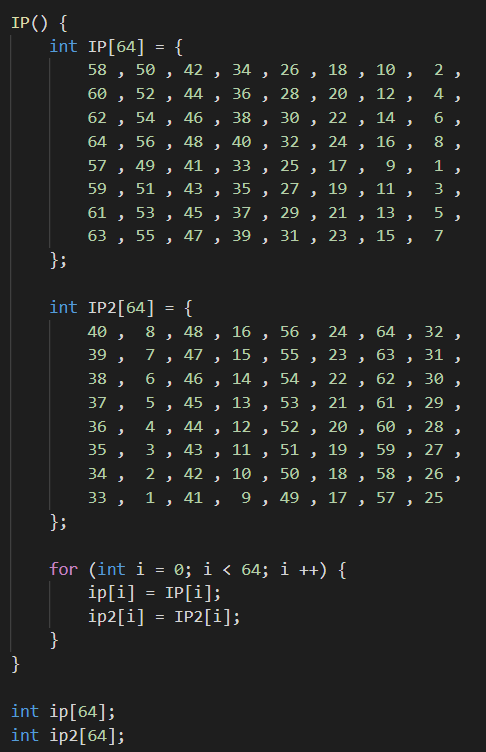
1. **IP置换**

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M0 = IP(M) = L0R0，这里L0 和R0分别是M0 的前32位和后32位。下表给出IP 置换后的下标编号序列。

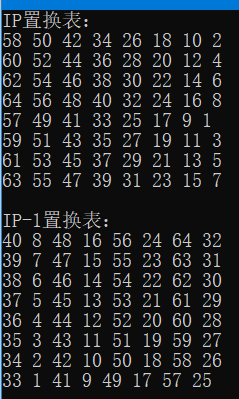


这里的IP置换表的第a个数字，相当于新数组第a个位，其中数字i相对于旧数组的num[i]。即表中数字相当于这个位置的数字在旧数组中的下标。因此将此表录入数组，然后即可根据下标置换。这里为了方便转换操作，将64位的8字节字符串换成64字节对应其二进制值。

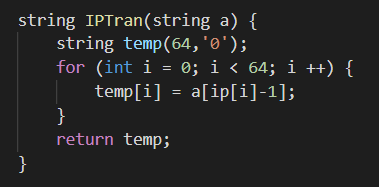
建立IP与IP-1置换表：



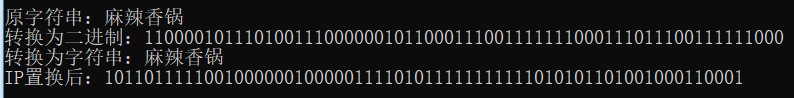
置换表打印检验：



二进制码置换代码：



转换效果：



1. **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Ri-1作E-扩展，成为48位的串E(Ri-1)

(2)将E(Ri-1)和长度为48位的子密钥Ki 作48位二进制串按位异或运算，Ki 由密钥K 生成

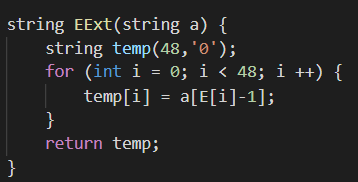
(3)将(2)得到的结果平均分成8个分组，每个分组长度6位。各个分组分别经过8个不同的S-盒进行6-4 转换，得到8个长度分别为4位的分组

(4)将(3)得到的分组结果顺序连接得到长度为32位的串

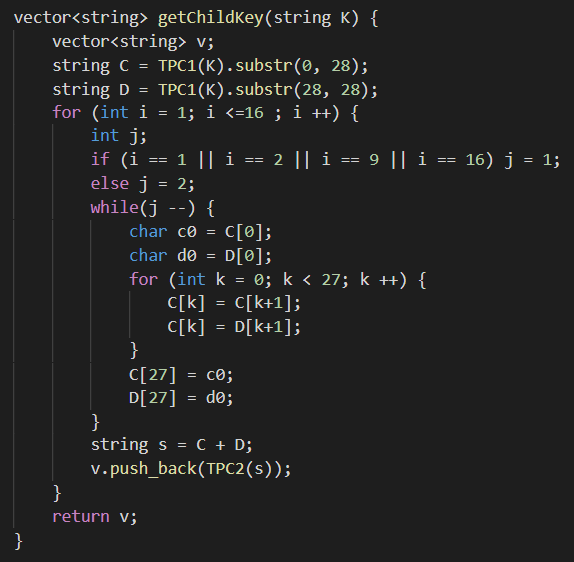
(5)将(4)的结果经过P-置换，得到的结果作为轮函数f(Ri-1, Ki) 的最终32位输出。

首先需要声明定义一个数组作为E扩展的参照表（代码略）

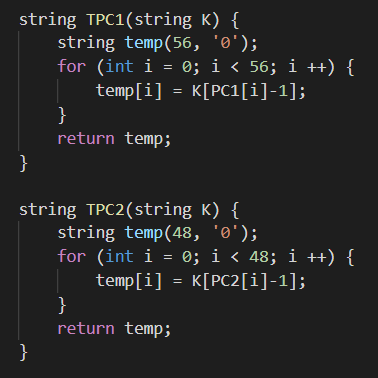
E-扩展的代码与IP置换相同原理：



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K1到K16：

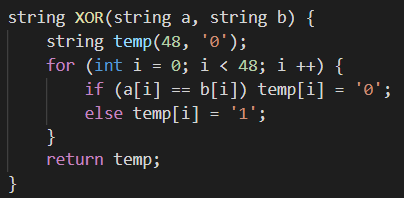


需要先对原密钥PC1置换，分成两段后分别移位后再合并再PC2置换：

（PC1与PC2的置换表略）

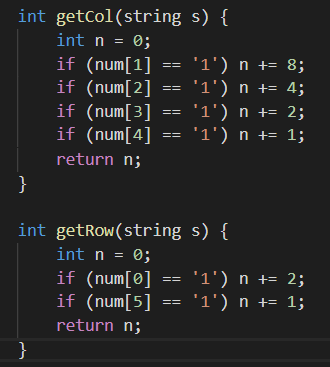
上面获得子密钥的函数应在迭代中调用后再将其中一个子密钥传入轮函数。

异或函数：

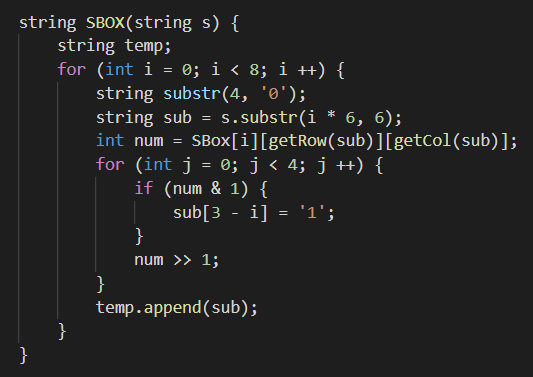


后面的任务是将异或后的串分成8段，每段6位分别通过各自的SBOX后变成4位再合并为32位。

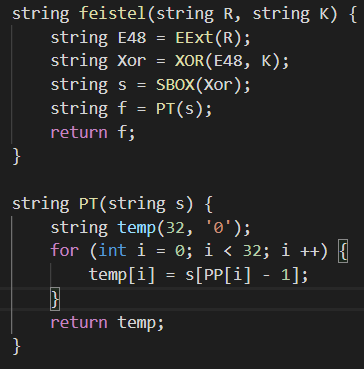
首先定义获得行列号的函数：



然后让串经过box（BOX的8个二维数组的定义略）：

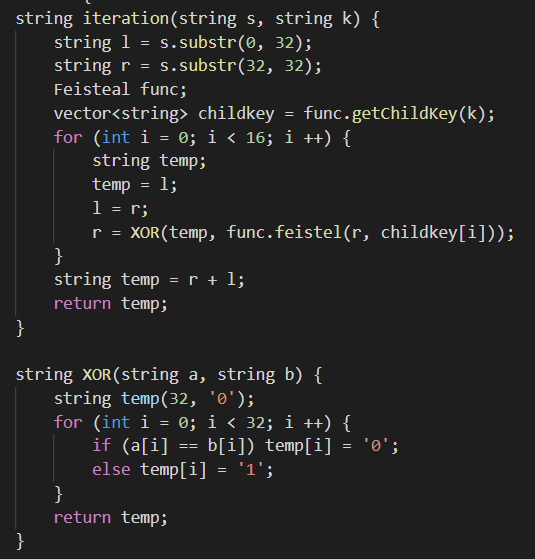


最后让串经过P-置换后返回，则得到轮函数的返回值：



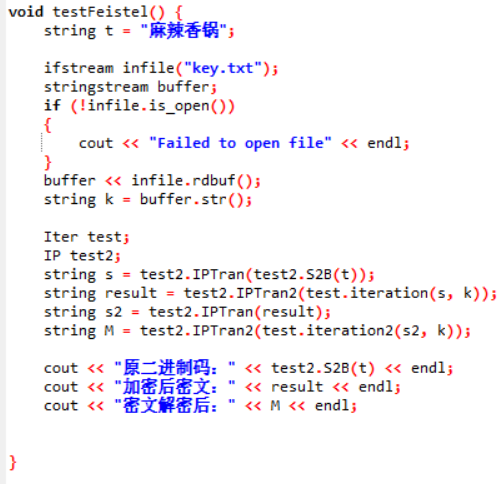
1. **16次迭代**

16次迭代，交换，轮函数调用，异或，最后的W置换：



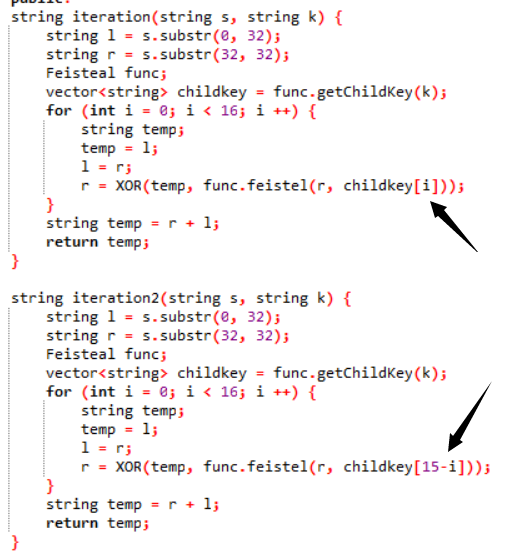
由于轮函数模块与迭代关系紧密，两者交替调用，而且16个子密钥在每一步迭代中分别作为每次调用轮函数的参数，因此迭代与轮函数都写完后才可以开始进行测试。

测试代码：

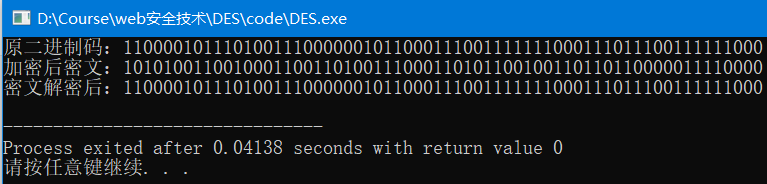


这里直接将8字节t作为明文，key.txt内为已设置好的64位密钥

IPTran将其IP置换，interation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））。



最终测试结果如下：



### 数据结构：

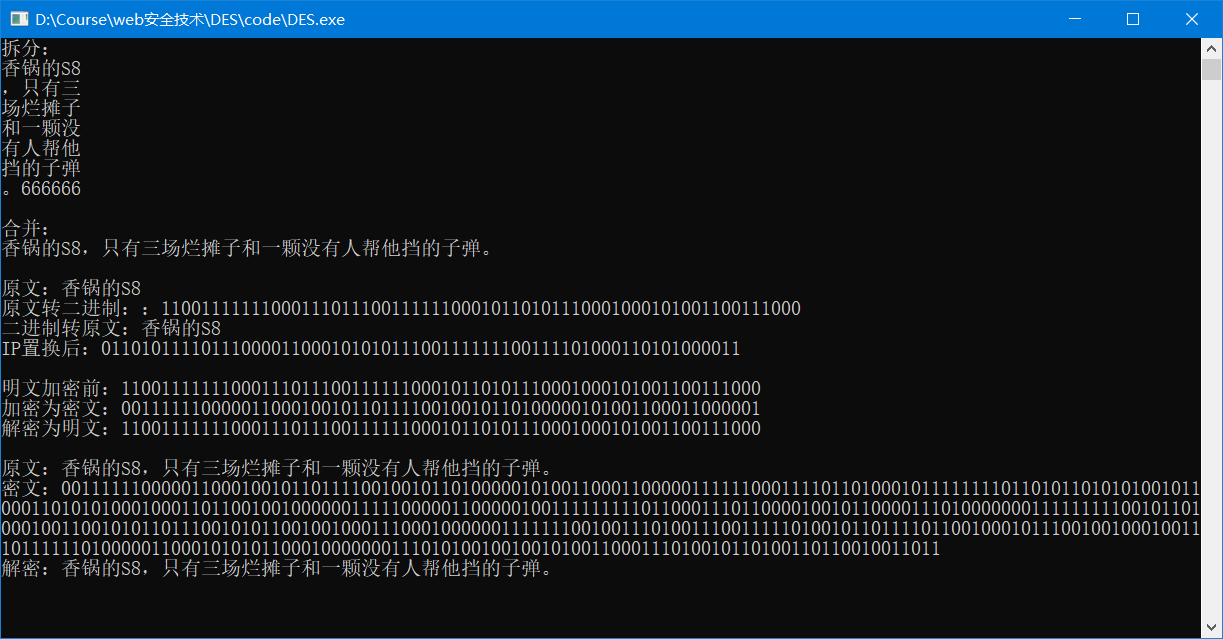
|  |  |
| --- | --- |
| IP置换表 | 64位一维数组 |
| IP-1置换表 | 64位一维数组 |
| E-扩展规则（比特-选择表） | 48位一维数组 |
| S-盒 | 8×4×16三维数组 |
| P-置换表 | 32位一维数组 |
| PC-1置换表 | 56位一维数组 |
| PC-2压缩置换表 | 48位一维数组 |
| 子密钥 | 数量16的元素为string的vector容器 |

### 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：

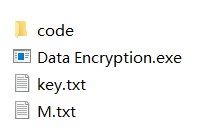
该测试文件exe为 ../code/DES.exe。



根目录中的code文件内为测试用的代码与所有代码文件。

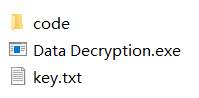
同时分别编写了加密程序与解密程序供校验。

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：



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。