密码学实验报告二

DES算法

实验流程：

首先我们对DES大致流程做一个分析，其大致分为如下几步

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>。

所以我们的DES程序整体算法流程

注：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。

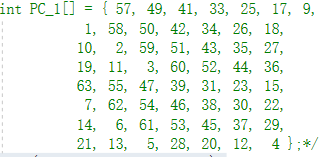
第二步，子秘钥生成

首先在我们进行加解密之前，我们要生成好16组48位秘钥。

首先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

1. 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



for (int i = 0; i < 56; i++)

{

firstKey[55 - i] = key[64 - PC\_1[i]];

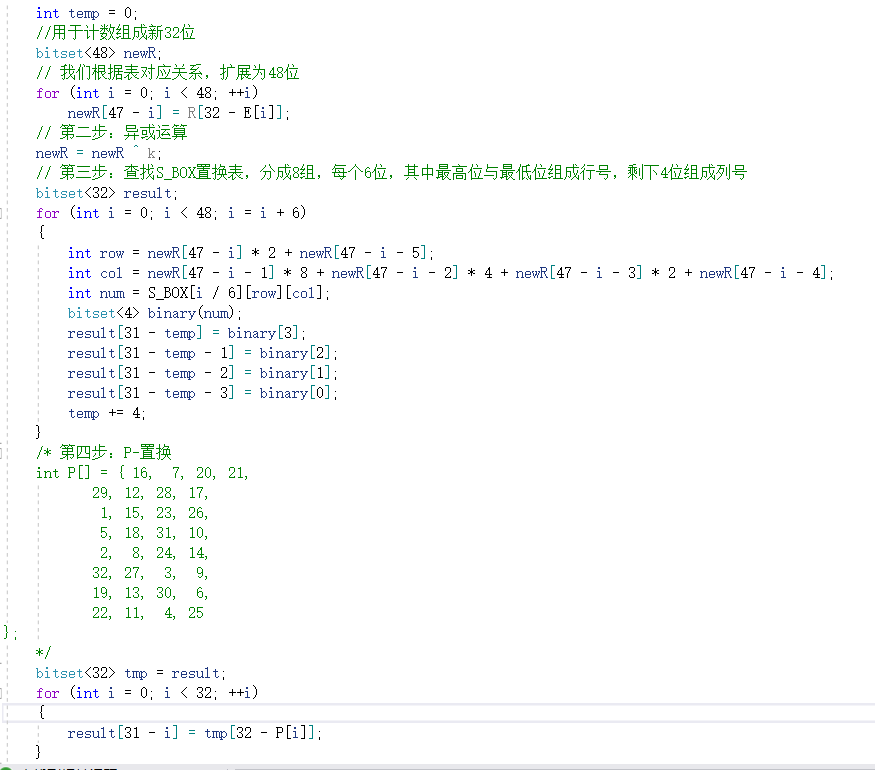
}由于以后置换全部采取这种方法，所以后续置换就不详细展示

1. 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



调用getHex(bitset<64> b,string &s)将64位bit转化为十六进制字符串。

getOrigin(bitset<64>& secret) DES解密：

IP初始置换

十六次循环

将64位数据分为高低位，调用f（bitset<32> R, bitset<48> k子秘钥）对低位进行处理然后与高位异或（注：在逆过程中，子秘钥的调用顺序与加密相反）

进行合并

逆初始置换IP－1

getSecret(bitset<64>& origin) DES加密：

IP初始置换

十六次循环

将64位数据分为高低位，调用f（bitset<32> R, bitset<48> k 子秘钥）对低位进行处理然后与高位异或

进行合并

逆初始置换IP－1

调用getKeys()生成子秘钥：

根据压缩置换表pc\_1进行置换

循环十六次，左移leftShift（）、压缩

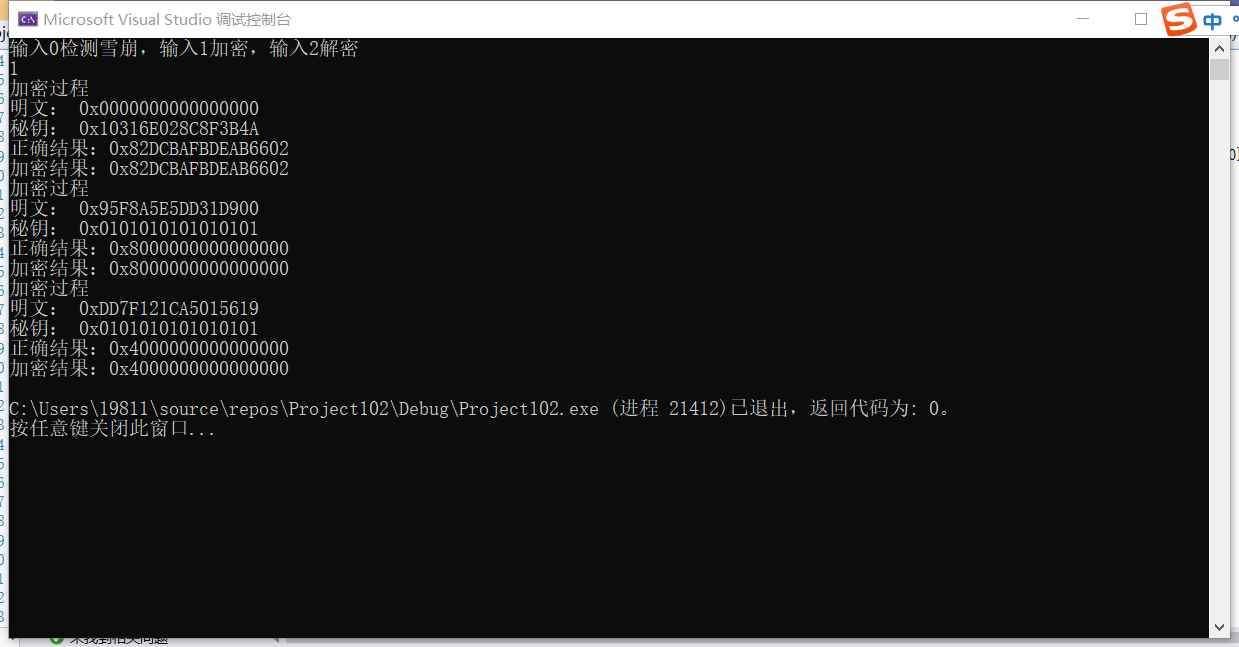
调用getbit(string a, bitset<64>& temp)对数据进行处理

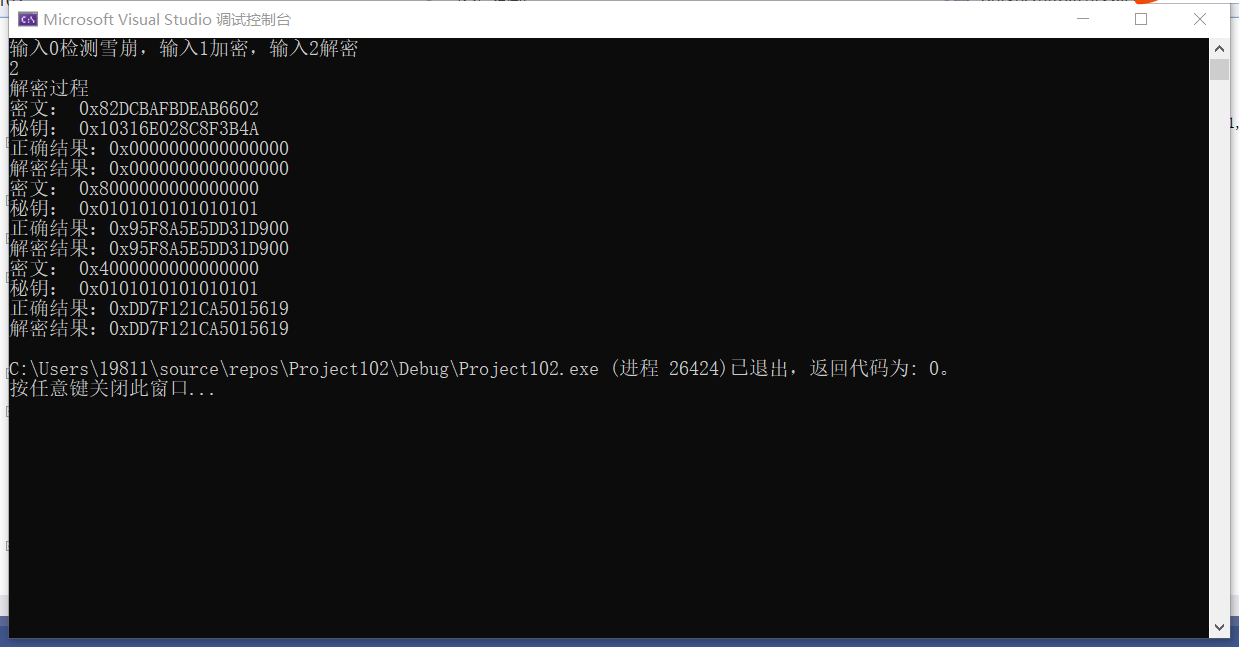
重要函数bitset<32> f(bitset<32> R, bitset<48> k 子秘钥)流程

1. 根据表E将32扩展为48
2. 与子秘钥异或运算
3. 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

4、P-1置换

加解密结果





雪崩效应

我们拿上图中展示的数据进行雪崩效应检测，首先我们使秘钥不变，每次仅更改一位明文，重复八次，得到如下结果

原始结果1000000000000000000000000000000000000000000000000000000000000000

改后结果0001111110011101011101101111111000000010011101110010110011000100

改后结果0001101111000100100010100111001011111110000001000010001110110100

改后结果0001101001110111010010101011101111000001100101111011110100001010

改后结果1110110001101001100111101100100111110101110111000001110101000101

改后结果1111010101001011010111010111110011110000011010010011111110100111

改后结果1001010011101001011110111011110100001110110101111100110111111101

改后结果0011100010110111100010011000001111100110011001110111000101011110

改后结果1110001101101100011000101101011001111010111000110110100010010100

平均每次有35位发生变化

原始结果1000000000000000000000000000000000000000000000000000000000000000

改后结果1000000000000000000000000000000000000000000000000000000000000000

改后结果0101010101000010100101100000110101001111111101000110101110011111

改后结果1101010100011111110010011111010111111101100111110001110010101011

改后结果0000000110111110110000001011010101010101100111110010111110111011

改后结果1000000000000000000000000000000000000000000000000000000000000000

改后结果0000000110011110000000101011010110110000100110100100010111000001

改后结果1100010100011001010111011011111011010000001011111001110100000001

改后结果0110000111000011111110101000011101110111000100101110001101001110

平均后25次

值得注意的是我们发现对于这个实验样本，当我们把秘钥0x0101010101010101其中的1改成0时，加密结果并没有发生变化，可能是由于我们不小心删除的是奇偶校验位。