DES 加密算法设计实验报告

郭雄锋 15331092

1. 算法原理概述

块加密

将明文M 分割成M1、M2 … Mn区段，对每一个区段资料应用相 同的演算法则和钥匙，数学表示为:

$$E(M,K) = E(M_1,K)E(M_2,K)\ldots E(M_n,K)$$

对称加密

对称加密(也叫私钥制加密) 指加密和解密使用相同密钥的加 密算法，有时又叫传统密码算法。
对称密码系统的加密密钥能够从解密密钥中推算出来，同时,解密密钥也可以从加密密钥中推算出来。在大多数的对称算法中，采用相同的加密密钥和解密密钥，所以也称这种加密算法为秘密密钥算法或单密钥算法.

DES 加密概述

1. DES 是一种典型的块加密方法：它以64位为分组长度，64 位一组的明文作为算法的输入，通过一系列复杂的操作， 输出同样64位长度的密文。
2. DES 使用加密密钥定义变换过程，因此算法认为只有持有 加密所用的密钥的用户才能解密密文。
3. DES 的采用64位密钥，但由于每8位中的最后1位用于奇偶 校验，实际有效密钥长度为56位。密钥可以是任意的56位 的数，且可随时改变。其中极少量的数被认为是弱

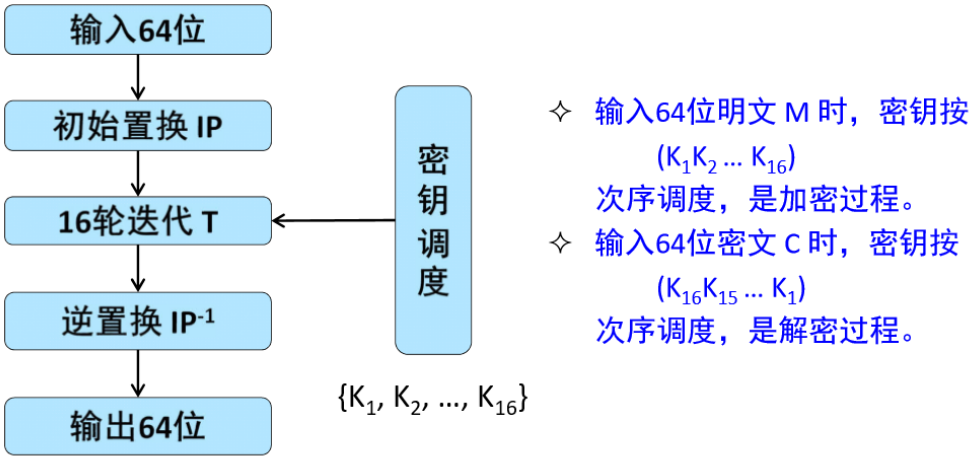
密钥，但能容易地避开它们。所有的保密性依赖于密钥。

4. DES 算法的基本过程是换位和置换。

以上内容摘自老师课件，并经过了个人整理。

DES基本过程

- **DES 算法概要**
  - DES 算法的总体结构 — *Feistel* 结构



2. 总体结构

现在来介绍我设计的DES加密系统的总体结构。**首先注意到，DES的加密有以下特性**：

1. DES的加解密过程应该作为一个系统。 以便于使用。
2. 密钥的生成过程与加密过程是严格分离的。
3. 密钥的生成过程是十分繁琐的。每一个密钥都需要前一个经过迭代才能得到。
4. 对于每一个DES的加解密系统，过程都是一致的，只是因为不同的密钥而产生不同的加解密结果。
5. DES的一个巧妙之处在于其加解密过程完全一致，只是需要密钥的顺序刚好相反。
6. DES涉及到许多置换，而这些置换都是已经被确定的。

   现在，根据上面提到的有关特性，我们来设计DES加密系统：
   - 首先，对于第1点，这就要求我们考虑用OOP思想去构建一个完整的系统，而不是只是用一个函数去完成这一目的。 事实上尽管DES的加密思想十分简单，但是代码实现起来略显繁琐。 写成若干个函数的拼接将使整个程序显得混乱且结构不清晰。
   - 对于第2，3点，既然key和DES并不是双向相关，那么我们可以考虑将key单独提取出来，做成一个模块。
   - 对于第4点，再次印证key与DES是单向相关的。 故可以考虑在DES模块中加入一个key模块作为其成员。
   - 对于第5点，我们可以将加解密过程设计为DES模块中的两个方法，而不必做两个不同的系统。

- 对于第6点， 由于有大量的常量。 我们可以考虑将这些量以单独文件保存，以头文件的方式引用。 以避免代码显得臃肿。

基于以上分析， 我给出我的总体结构：

1. 文件结构
   一共包含4个文件， 分别是`main.cpp, DES.cpp, key.cpp, helper.cpp`
   功能分别为：

   - `main.cpp`: 主函数， 负责调用与测试DES的加解密过程是否正确。
   - `DES.cpp`: 包含DES模块的文件，主要用于进行对于特定的key的加密解密过程。
   - `key.cpp`: 包含key模块的文件， 主要用于对传入的64bit key生成16个48bit 的key序列。
   - `helper.cpp`: 包含各种常数以及一些需要用到的辅助函数的模块。

2. 类层次间的引用联系。

   - `main` 主要调用 `DES` 和 `key` 的模块， 负责初始化key并将之用于初始化DES。
   - `DES` 类中包含一个 `key` 模块， 并且会部分调用 `helper` 中的数据与辅助函数。
   - `key` 类主要就是通过调用 `helper` 中的数据与模块， 来生成对应的密钥序列。

3. 模块分解

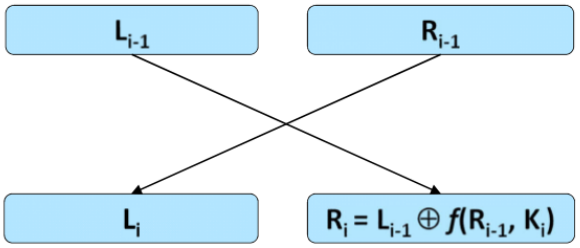
下面开始介绍我的各个模块中的具体结构。

`main` 主要用于测试， 故不多说了。
`helper` 中， 主要存放的是用于进行DES过程的特定置换以及S_BOX之类的常数数据和通用的（即不属于DES或key模块的方法）一些辅助函数。
主要需要介绍的是 `DES` 模块和 `key` 模块中的成员及方法。

- `DES` 模块
  DES 模块主要就是负责整个DES加密或解密的过程。通过上面的DES的过程图可以看出， 整个过程大致可以分为如下几个步骤：
  1. 初始置换init
  2. 16次迭代T
  3. 初始置换的逆置换。

所以DES模块里有对应的三个方法， 来执行这三个步骤。
而在这三个步骤中， 显然的， 2的任务最为繁重。 故还需继续分解。

- **DES 算法概要**
  - 迭代 T
    - ✧ 根据 L_0R_0 按下述规则进行16次迭代，即
      $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, i = 1 .. 16.
    - ✧ 这里 ⊕ 是32位二进制串按位异或运算，f 是 Feistel 轮函数
    - ✧ 16个长度为48bit的子密钥 K_i (i = 1 .. 16) 由密钥 K 生成
    - ✧ 16次迭代后得到 $L_{16}R_{16}$
    - ✧ 左右交换输出 $R_{16}L_{16}$



可以看到这个迭代过程的核心就是`Feistal`轮函数的作用。所以自然`Feistal`函数也要作为一个方法， 来便于迭代时的反复调用。
然后对于`Feistal`函数， 仍是一个较复杂的过程，因此继续像这样分解下去。又得到三个子过程：

1. `E_expand()`
2. `S_Box_Transform()`
3. `P_Hash()`
   分解到这一步后， 每个子函数都执行的是相对简单的任务。实现起来都较为轻松了。
   上面对应的是加密过程。 而解密过程同样适用。所以把上面总的步骤做成一个函数`process()`来调用。并向`DES`添加上`code()`和`decode()`方法，分别都在方法内调用`process()`来处理。 只不过在这之前两者要先对`Key`做不同处理罢了。
   由于`DES`需要对外显示的就是加密，解密两个过程。 因此除了`code()`，`decode()`两个方法之外，其他的都应作为private权限设置。

- `Key`模块
  Key模块相对于DES而言较为简单。首先 考虑key的外部可以访问的是Key的哪些内容， 即我们需要调用key的哪些方法与数据。
  首先确定的是我们需要从Key中得到16个48 bit的子密钥。所以肯定要用一个数组来存放key，并且权限是public。然而在加解密过程中， 我们还需要将key的顺序reverse，故应还要一个`reverse()`方法。注意到我们还需了解当前存放的16个key是否之前已经被reverse过， 所以还需要一个`bool`来保存当前key数组是否已经被reverse过了。
  综上， `Key`模块需要包含：
  1. key数组。
  2. `reverse()`方法。
  3. `isReversed`变量。

由于生成key数组也是一个复杂迭代过程， 因此还需要进一步对其进行分解。
分解的方法与`DES`完全相同。就是把若干个步骤拆分成若干个函数， 使得每个函数的功能清晰。
得到key数组的一个总的方法是`getSubkey()`, 在其中调用了如下几个分解出来的子方法：

1. `PC_1Hash()`:用于做PC1置换
2. `PC_2Hash()`:用于做PC2置换
3. `leftShift()` ：用于每轮迭代时对L, R两部分进行 循环左移。

4. 数据结构

由于前面解释的比较详细了。这部分可能写的比较简略。

1. 首先对于每个模块， 都是用的C++的`class`
2. 然后对于其中的各种置换表， 我都是用的一维数组来存放的。（S_BOX除外）
   之所以用一维数组， 显然是为了便于查询。 对于查找某个位置的对应置换，只需要下标寻址即可。
3. 对于S_BOX， 因为它需要通过行列不同编号来确定， 因此对于8个S_BOX， 用二维数组存放较为合适。
4. DES过程中会涉及大量的位运算。 对于这些变量， 统统采用C++STL中的`bitset`来保存。十分方便。

5. 类C语言算法过程。

这部分主要以部分代码为主。附带部分简略解释。
本代码已经过少量测试， 编译通过， 测试数据均已核对正确。
首先放`Key`模块内容：

```
    #include <bitset>
    #include <algorithm>
    #include<iostream>
    #include "helper.cpp"
using namespace std;

class Key{
    public :
        bitset<48> k[16];
        Key() {

        }
        Key(string & s) {
            getSubKey(bit_form(s));
            isreversed = false;
```

```
        }
        void Reverse() {
            // Used for decode.
            for (int i = 0; i < 8; ++i) {
                swap(k[i], k[15 - i]);
            }
            isreversed = !isreversed;
        }
        bool isreversed;
    private:
        bitset<56> PC_1Hash(bitset<64> &key) {

             bitset<56> ans;
             for (int i = 0; i < 56; ++i) {
                ans[i] = key[PC_1[i] - 1];//diff, -1是因为从0开始计数
的。
             }
             return ans;
        }
        bitset<48> PC_2Hash(bitset<56> &key) {

             bitset<48> ans;

             for (int i = 0; i < 48; ++i) {
                ans[i] = key[PC_2[i] - 1];
             }
             return ans;
        }
        void leftShift(bitset<28> &i, int n) {
            i = (i << n) | (i >> 28 - n); //
        }
        void getSubKey(bitset<64> init_key) {
            // 64bit 原始key
            bitset<56> valid_key;
            int j = 0;
            valid_key = PC_1Hash(init_key);
            bitset<28> C, D;
            for (int i = 0; i < 28; ++i) {
                C[i] = valid_key[i];//C[i]高位
                D[i] = valid_key[i + 28];
            }
            for(int i = 0; i < 16; ++i) {
                if (i + 1 == 1 || i + 1 == 2 || i + 1 == 9 || i + 1
== 16) {
                    leftShift(C, 1);
                    leftShift(D, 1);
                }
                else {
                    leftShift(C, 2);
```

```
                    leftShift(D, 2);
                }
                //concat Ci, Di
                bitset<56>tmp;
                for (int j = 0; j < 28; ++j) {
                    tmp[j] = C[j];
                    tmp[28 + j] = D[j];
                }
                k[i] = PC_2Hash(tmp);
            }
        }
    };
```

这部分主要过程即为`getSubKey()`，即首先进行`PC_1Hash()`，分离C, D两部分进行循环左移迭代。 最后再进行一次`PC_2Hash()`即得到密钥组保存在`bitset<48> k[16];`中。
下面是`DES`模块：

```
#include "key.cpp"
#include <iostream>
class Des {
public:
    Des(Key k) {
        this->key = k;
    }
    bitset<64> code(string s) {

        if (key.isreversed) key.Reverse();
        return process(bit_form(s));
    }
    bitset<64> decode(bitset<64> c) {
        if (!key.isreversed) key.Reverse();
        return process(c);
    }
private:
    bitset<64> process(bitset<64> s) {
        //Init Permutation.
        bitset<64> init = IP(s);
        //Iteration.
        //swap L16R16 -> R16L16
        bitset<64> it = Iter(init);

        //Init Permutaion Inverse.
        return IP_Inv(it);
    }
    bitset<48> E_expand(bitset<32> & R) {
```

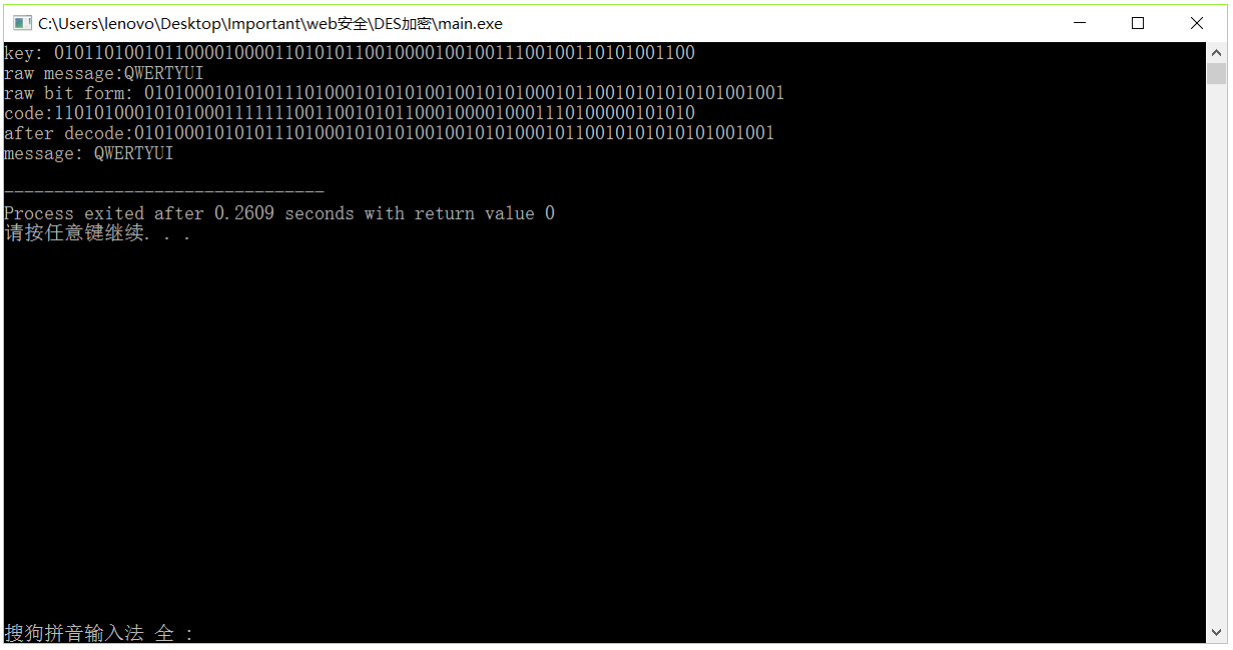
```
        bitset<48> ans;
        for (int i = 0; i < 48; ++i) {
            ans[i] = R[E[i] - 1];
        }
        return ans;
    }
    bitset<32> PHash(bitset<32> s) {

        bitset<32> ans;
        for (int i = 0; i < 32; ++i) {
            ans[i] = s[P[i] - 1];
        }
        return ans;
    }
    bitset<4> S_Box(bitset<6> s, int i) {
        int row = s[0] * 2 + s[5], col = s[1] * 8 + s[2] * 4 + s[3]
* 2 + s[4];// row = s0s5, col = s1s2s3s4
        return bitset<4>(S_BOX[i][row][col]);
    }
    bitset<32> S_Box_Transform(bitset<48> s) {
        bitset<32> ans;
        for (int i = 0; i < 8; ++i) {
            bitset<6> tmp;
            //得到分组
            for (int j = 0; j < 6; ++j) {
                tmp[j] = s[i * 6 + j];
            }
            //对每组转换
            bitset<4> t = S_Box(tmp, i);
            //写入输出bitset
            for (int j = 0; j < 4; ++j) {
                ans[4 * i + j] = t[j];
            }
        }
        return ans;
    }
    bitset<32> Feistal(bitset<32> R, bitset<48> k) {
        bitset<48> exR = E_expand(R);
        bitset<48> ans = exR ^ k;
        //分组， 64转换。合并。
        return PHash(S_Box_Transform(ans));//S_BOX 之后再用P_置换打
乱。
    }
    bitset<64> IP(bitset<64> &s) {
        bitset<64> ans;
        for (int i = 0; i < 64; ++i) {
            ans[i] = s[IP[i] - 1];
        }
```

```
        //cout << "IP transform: " << ans << endl;
        return ans;
    }
    bitset<64> Iter(bitset<64> &s) {
        bitset<32> L, R;
        //get L0, R0.
        for (int i = 0; i < 32; ++i) {
            L[i] = s[i];
            R[i] = s[i + 32];
        }
        //Iter main process
        for (int i = 0; i < 16; ++i) {
            bitset<32>tmp = R;
            R = L ^ Feistal(R, key.k[i]);
            L = tmp;

        }
        bitset<64> ans;
        //这里，反向。
        for (int i = 0; i < 32; ++i) {
            ans[i] = R[i];
            ans[i + 32] = L[i];
        }
        return ans;
    }
    bitset<64> IP_Inv(bitset<64> & s) {
        bitset<64> ans;
        for (int i = 0; i < 64; ++i) {
            ans[i] = s[IP_Inv[i] - 1];
        }
        return ans;
    }
    Key key;
};
```

详细的过程已经在上面解释过了。就不多解释了。下面是`main`的测试：

```cpp
#include<iostream>
#include <bitset>
#include <algorithm>
#include <fstream>
#include "DES.cpp"
using namespace std;

int main() {
    string s_key = "ZXCVBNML";
    Key k(s_key);
    cout << "key: " << bit_form(s_key) << endl;
    Des des(k);
    string raw_msg = "QWERTYUI";
    cout << "raw message:" << raw_msg << endl;
    cout << "raw bit form: " << bit_form(raw_msg) << endl;
    auto trans_info = des.code(raw_msg);
    cout << "code:" << trans_info << endl;
    bitset<64> decode_msg = des.decode(trans_info);
    cout << "after decode:" << decode_msg << endl;
    cout << "message: " << str(decode_msg) << endl;
}
```

直接贴一张运行截图：



可以看到， 经过加密解密过程后， 恢复为了明文。

6. 其他部分代码

由于这次作业只能交一个pdf， 因此无法把代码打包上传。我把整个项目放到了github上，以下是地址：
https://github.com/guoxiongfeng/Web-Security-DES

下载后，直接编译运行`main.cpp`即可。