

密码学基础

实 验 报 告



专业：信息安全
班级：1504201
学号：150120526
姓名：殷悦

哈尔滨工业大学（威海）

实验 1 实现 DES 加密算法

姓名	殷悦	院系	计算机学院	学号	150120526
任课教师	刘杨	指导教师	刘杨		
实验地点	研究院中 403	实验时间	2017-10-9		
实验课表现	出勤、表现得分		实验报告得分	实验总分	
	操作结果得分				

一、实验目的

- 1、掌握 DES 加、解密算法的原理和过程；
- 2、能够编写程序实现 DES 加、解密，以及 DES 的应用。

二、实验内容

一. 发展历史

DES 算法为密码体制中的对称密码体制，又被称为美国数据加密标准，是 1972 年美国 IBM 公司研制的对称密码体制加密算法。明文按 64 位进行分组，密钥长 64 位，密钥事实上是 56 位参与 DES 运算（第 8、16、24、32、40、48、56、64 位是校验位，使得每个密钥都有奇数个 1）分组后的明文组和 56 位的密钥按位替代或交换的方法形成密文组的加密方法。

二. DES 加密标准

DES 是对二元数字分组加密的分组密码算法，分组长度为 64 比特。每 64 位明文加密成 64 位密文，没有数据压缩和扩展，密钥长度为 56 比特，若输入 64 比特，则第 8，16，24，32，40，48，56，64 为奇偶校验位，所以，实际密钥只有 56 位。DES 算法完全公开，其保密性完全依赖密钥。

三. DES 算法原理

(一)加密过程

DES 使用 64 为密钥匙，把 64 位的明文输入块变为 64 位的密文输出块，将数据输出分为 L0、R0 两部分，每部分各长 32 位，使用以下置换表：

58,50,12,34,26,18,10,2,60,52,44,36,28,20,12,4,
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,

将输入的第 58 位换到第一位，第 50 位换到第 2 位，一直到最后，输出新结果

L0、R0 则是换位输出后的两部分，L0 是输出的左 32 位，R0 是右 32 位，例：设置换前的输入值为 D1D2D3……D64，则经过初始置换后的结果为：L0=D550……D8；R0=D57D49…D7。

经过 26 次迭代运算后，得到 L16、R16，将此作为输入，进行逆置换，即得到密文输出。逆置换正好是初始置的逆运算，例如，第 1 位经过初始置换后，处于第 40 位，而通过逆置换，又将第 40 位换回到第 1 位，其逆置换规则如下表所示：

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,

放大换位表

32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10,11,
12,13,12,13,14,15,16,17,16,17,18,19,20,21,20,21,
22,23,24,25,24,25,26,27,28,29,28,29,30,31,32, 1,

单纯换位表

16,7,20,21,29,12,28,17, 1,15,23,26, 5,18,31,10,
2,8,24,14,32,27, 3, 9,19,13,30, 6,22,11, 4,25,

在 $f(R_i, K_i)$ 算法描述图中, $S_1, S_2 \dots S_8$ 为选择函数, 其功能是把 6bit 数据变为 4bit 数据。下面给出选择函数 $S_i (i=1, 2, \dots, 8)$ 的功能表:

选择函数 S_i

S_1 :

14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13,

S_2 :

15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9,

S_3 :

10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12,

S_4 :

7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14,

S_5 :

2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3,

S_6 :

12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13,

S_7 :

4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12,
S8:
13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11,

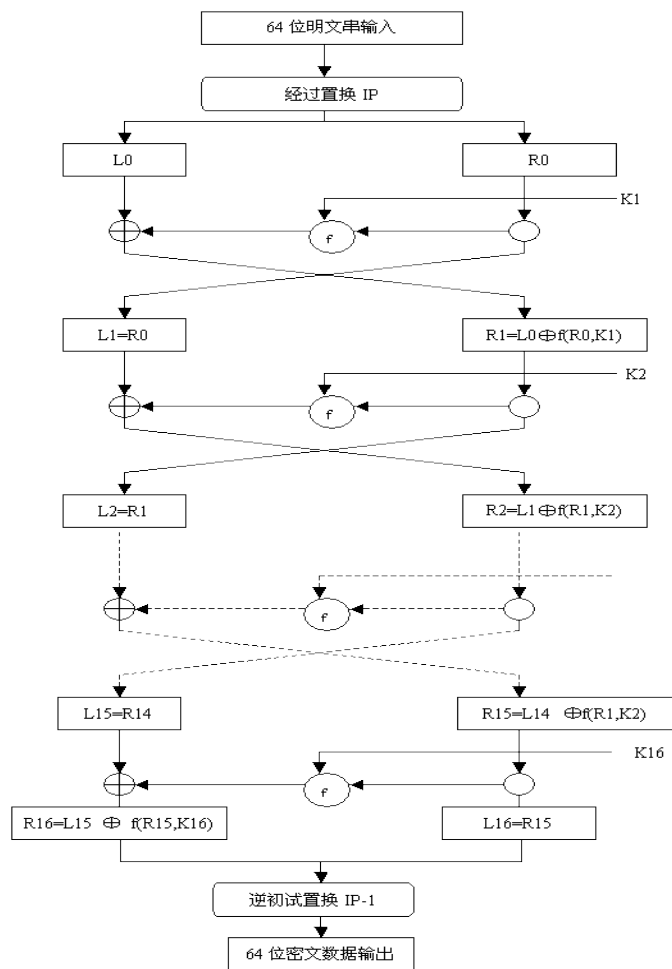


图 2-1 加密流程图

(二)子密钥 Ki(48bit)的生成算法

初始 Key 值为 64 位，但 DES 算法规定，其中第 8、16、.....64 位是奇偶校验位，不参与 DES 运算。故 Key 实际可用位数便只有 56 位。即：经过缩小选择换位表 1 的变换后，Key 的位数由 64 位变成了 56 位，此 56 位分为 C0、D0 两部分，各 28 位，然后分别进行第 1 次循环左移，得到 C1、D1，将 C1（28 位）、D1（28 位）合并得到 56 位，再经过缩小选择换位 2，从而便得到了密钥 K0（48 位）。依此类推，便可得到 K1、K2、.....、K15，不过需要注意的是，16 次循环左移对应的左移位数要依据下述规则进行：

循环左移位数 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1

以上介绍了 DES 算法的加密过程。DES 算法的解密过程是一样的，区别仅仅在于第一次迭代时用于子密钥 K15，第二次 K14、……，最后一次用 K0，算法本身并没有任何变化。

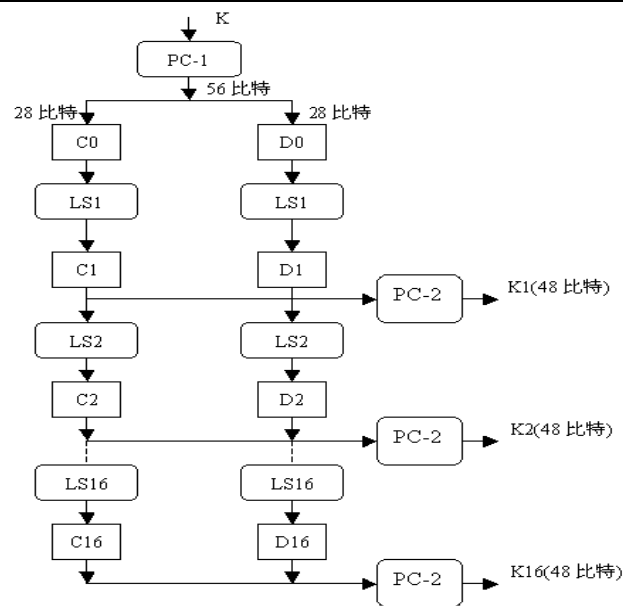


图 2-2 子密钥生成流程图

(三)解密

DES 的解密过程和 DES 的加密过程完全类似，只不过将 16 圈的子密钥序列 K_1, K_2, \dots, K_{16} 的顺序倒过来。即第一圈用第 16 个子密钥 K_{16} ，第二圈用 K_{15} ，其余类推。

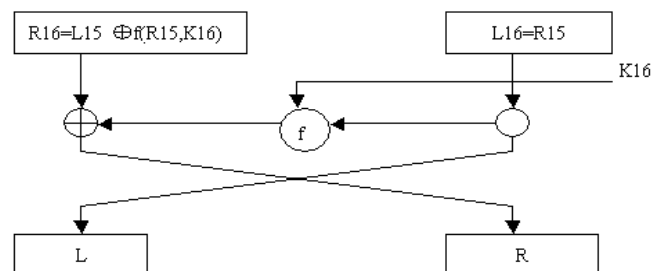


图 2-3 解密流程图

四．代码讲解

使用三个头文件，time.h 在生成随机数的时候使用

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
static unsigned char ip[64]={//Initial permutation IP
```

```
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
    }

static unsigned char ip1[64]={// initial replacement IP reverse IP-1
    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
}

static unsigned char pc1[56]={// key replacement 1
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
}

static unsigned char pc2[48]={// key replacement 2
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
}

static unsigned char e[48]={// Extended replacement E
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,

```

```

24, 25, 26, 27, 28, 29,
28, 29, 30, 31, 32, 1
}

static unsigned char lcircle[16]={// the number of rounds to be rounded
1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
}

static unsigned char s[8][4][16]={// S table
// S1
14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,
// S2
15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,
// S3
10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,
// S4
7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,
// S5
2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,
// S6
12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,
// S7
4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,

```

```

// S8
13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11
}

static unsigned char P[32]={// replace P
    16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
    2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25
}

#define max 100
unsigned char firstkey[32]; //password
unsigned char random_key[64]; // Random sequence for CBC encryption
unsigned char r_key[64]; // random sequence
unsigned char temp_key[64]; // temporary key store
unsigned char mw[8]; // Random sequence for CBC encryption
unsigned char m[64]; // plaintext / ciphertext array
unsigned char charkey[8]; // char type key
unsigned char ejzkey[64]; // binary key
unsigned char ER[48]; // 32 bits extended by 32 bits later
unsigned char dmw[8]={0} // encoded plaintext / ciphertext
unsigned char lskey[56]; // temporary storage key
unsigned char childkey[16][56]; // 16 subkeys

FILE* fptr=NULL, *dfptr;
int size;
int mode_cbc;
unsigned char* L, *R;

void Thekey(unsigned char* firstkey, unsigned char* bkey); // generate a 64-bit key based on the
password
void encode(); // encryption
void decode(); // decryption
void getbit(unsigned char* m, unsigned char* mw); /// char to binary
void replace(unsigned char* a, unsigned char* b, int n); // replace function
void circle(unsigned char* key, int n); // left shift function
void yihuo(unsigned char* a, unsigned char* b, int n); // exclusive OR function
void S_box(unsigned char* a); // S box
void getchars(unsigned char* a, unsigned char* mw); //binary to char
int add(unsigned char* a); //summation function
void getdkey(unsigned char* key); //produces 16 subkeys
void enrepeat(); // encryption loop

```



```

void derepeat();// decrypt the loop
void getbmphead(FILE* fptr, FILE* dfptr);// read the BMP header
void randomkey(unsigned char* random_key); // Generates a 64-bit random sequence

```

这个函数对密码进行处理，如果密码超过 8 比特，超出的部分删除，如果少于 8 比特，则进行重复补充至 8 位

```

void Thekey(unsigned char* firstkey, unsigned char* bkey)// generate a key greater than 64 bits,
less than completion
{
    int i, j=0;
    for(i=0; i<8; i++)
    {
        if(firstkey[j]!='\n')
        {
            j=0;
            i--;
        }
        else
        {
            bkey[i]=firstkey[j];
            j++;
        }
    }
    return ;
}

```

这个函数对照片进行加密，可以选择使用 CBC 模式或者 ECB 模式

```

void encode() // encryption
{
    int i, n, j=0;
    while(ftell(fptr)<size)
    {
        fread(&mw[0], 1, 8, fptr);
        getbit(m, mw); // convert plaintext to binary
        if(j==0)
        {
            for(i=0; i<64; i++)
                r_key[i]=random_key[i];
        }
        if(mode_cbc)//1.CBC 0.ECB
            yihuo(m, r_key, 64);
        replace(m, ip, n=64);// initial replacement of plaintext
        L=&m[0]; // clear text is divided into left and right parts
    }
}

```

```

        R=&m[32];
        enrepeat(); // enter the sixteenth iteration
        replace(L, ip1, 64); // reverse initial permutation
        for(i=0;i<64;i++)
            r_key[i]=L[i];
        getchars(L, dmw); // convert the binary ciphertext into char
        fwrite(&dmw[0], 1, 8, dfptr); // Write the ciphertext to the file
        j++;
    }
    return;
}

```

这个函数对图片进行解密

```

void decode() // decryption
{
    int i, n, s=0, j=0;

    while((s=ftell(fp))<size)
    {
        fread(&mw[0], 1, 8, fp);
        getbit(m, mw); // convert ciphertext to binary
        L=&m[0]; // ciphertext is divided into left and right parts
        R=&m[32];
        if(j==0)
        {
            for(i=0;i<64;i++)
                r_key[i]=random_key[i];
        }
        for(i=0;i<64;i++)
            temp_key[i]=L[i];
        replace(m, ip, n=64); // the initial replacement of ciphertext
        derepeat();
        replace(L, ip1, 64); // reverse initial permutation
        if(mode_cbc)
            yihuo(L, r_key, 64);
        for(i=0;i<64;i++)
            r_key[i]=temp_key[i];
        getchars(L, dmw); // convert binary plaintext into char
        fwrite(&dmw[0], 1, 8, dfptr); // Write the plaintext data in the array to the file
        j++;
    }
    return;
}

```

这个函数进行加密时多次轮秘钥置换

```

void enrepeat()
{
    int i, n, q;
    for(q=0;q<16;q++)//Enter 16 iterations
    {
        for(i=0;i<56;i++)
            lskey[i]=childkey[q][i];
        replace(lskey, pc2, n=48);// The key is compressed to 48 bits
        for(i=0;i<32;i++)
            ER[i]=R[i];
        replace(ER, e, n=48);// expand the right half to 48 bits

        yihuo(ER, lskey, n=48); // The right half is XORed with the key (48 bits)
        S_box(ER);// enter the S-box operation
        replace(ER, P, n=32); // P replace the right half

        yihuo(ER, L, n=32); // left and right XOR
        for(i=0;i<32;i++) // swap around
        {
            L[i]=R[i];
            R[i]=ER[i];
        }
        if(q==15) // 16 times and then exchange again
        {
            for(i=0;i<32;i++)
            {
                ER[i]=R[i];
                R[i]=L[i];
                L[i]=ER[i];
            }
        }
    }
}

```

这个函数进行解密时多次轮密钥置换

```

void derepeat()
{
    int i, n, q, j=0;
    for(q=0;q<16;q++) // Enter the 16 rounds of iterations
    {
        for(i=0;i<56;i++)
            lskey[i]=childkey[15-q][i];
        replace(lskey, pc2, n=48); //The key is compressed to 48 bits
    }
}

```

```

    for(i=0;i<32;i++)
        ER[i]=R[i];
    replace(ER, e, n=48); // expand the right half to 48 bits

    yihuo(ER, lkey, n=48); // The right half is XORed with the key (48 bits)
    S_box(ER); // enter the S-box operation
    replace(ER, P, n=32); // P replace the right half
    yihuo(ER, L, n=32); // left and right XOR
    for(i=0;i<32;i++) // swap around
    {
        L[i]=R[i];
        R[i]=ER[i];
    }
    if(q==15) // 16 times and then exchange again
    {
        for(i=0;i<32;i++)
        {
            ER[i]=R[i];
            R[i]=L[i];
            L[i]=ER[i];
        }
    }
}

```

```

void getdkey(unsigned char* key)
{
    int i, j;
    for(i=0;i<16;i++)
    {
        circle(key, i);
        for(j=0;j<56;j++)
        {
            childkey[i][j]=key[j]; // get 16 subkeys
        }
    }
    return;
}

```

这个函数将字节转换成二进制 0 或者 1 的字符

```

void getbit(unsigned char* m, unsigned char* mw)
{
    int i, j;
    unsigned char and[8]={128, 64, 32, 16, 8, 4, 2, 1}

```

```

for(i=0;i<8;i++)
{
    for(j=0;j<8;j++)
    {
        if((mw[i]&&[j])==0)//with the method to remove the corresponding binary
            m[i*8+j]=0;
        else
            m[i*8+j]=1;
    }
}
return;
}

```

这个函数将 Key 进行置换

```
void replace(unsigned char* a, unsigned char* b, int n)
```

```

{
    int i;
    unsigned char c[max];//replace
    for(i=0;i<n;i++)
        c[i]=a[b[i]-1];
    for(i=0;i<n;i++)
        a[i]=c[i];
    return;
}

```

这个函数将数据进行左位移

```
void circle(unsigned char* key, int n)
```

```

{
    int i, j, k;
    switch(lcircle[n])// key left loop: the first 28 bits to the left and the last 28
    {
        case 1:
            j=key[0];
            for(i=0;i<27;i++)
                key[i]=key[i+1];
            key[27]=j;
            j=key[28];
            for(i=28;i<55;i++)
                key[i]=key[i+1];
            key[55]=j;
            break;
        case 2:
            j=key[0];
            k=key[1];

```

```

        for(i=0;i<26;i++)
            key[i]=key[i+2];
        key[26]=j;
        key[27]=k;
        j=key[28];
        k=key[29];
        for(i=28;i<54;i++)
            key[i]=key[i+2];
        key[54]=j;
        key[55]=k;
        break;
    }
    return;
}

```

这个函数将字符串的每个 0 或 1 的自符进行与或操作

```

void yihuo(unsigned char* a, unsigned char* b, int n)
{
    int i;
    for(i=0;i<n;i++)
        a[i]=a[i]^b[i];
    return;
}

```

```

void S_box(unsigned char* a)
{
    int i, j;
    unsigned char b[8][6];
    unsigned char c[8];
    for(i=0;i<8;i++) //divide the 48 bits into 8 groups of six
    {
        for(j=0;j<6;j++)
            b[i][j]=a[i*6+j];
    }
    for(i=0;i<8;i++)
    {
        c[i]=s[i] [ b[i][0]*2+b[i][5] ] [ b[i][1]*8+b[i][2]*4+b[i][3]*2+b[i][4] ];//find the
        corresponding decimal number in table S
    }
    for(i=0;i<8;i++)//convert decimal to binary
    {
        for(j=3;j>=0;j--)
        {
            a[i*4+j]=c[i]%2;

```

```

        c[i]/=2;
    }
}
return;
}

void getchars(unsigned char* a, unsigned char* dmw)
{
    int i;
    for(i=0;i<8;i++)
        dmw[i]=add(&a[i*8]); //binary conversion to decimal char
    return;
}

int add(unsigned char* a)
{
    return a[0]*128+a[1]*64+a[2]*32+a[3]*16+a[4]*8+a[5]*4+a[6]*2+a[7];
}

void getbmphead(FILE* fptr, FILE* dfptr)
{
    int i;
    char c;
    for(i=0;i<55;i++)
    {
        c=fgetc(fptr);
        fputc(c, dfptr);
    }
    return ;
}

void randomkey(unsigned char* random_key)
{
    int i;
    srand(time(0));
    for(i=0;i<64;i++)
        random_key[i]=rand()%2;
    return;
}

int main()
{
    int i, n;

```

```

char filename[256]={ 'p','a','s','s','w','d','.','b','m','p',0}, c;
int function,ret;
printf("Please choose the function:\n");
printf("1.picture encrypt\t2.login\n");
scanf("%d",&function);
getchar();
function--;
if(function)
{
    printf("1.Login\t2.Set password\n");
    ret=scanf("%d",&function);
    //if(ret!=2)fflush(stdin);
    getchar();
} else
{
    for(i=0;i<256;i++)
        filename[i]=0;
    printf("please type the picture name that you want to encrypt:\n");
    gets(filename);
}
fptr=fopen(filename, "rb");
if(fptr==NULL)
{
    printf("Cant open the file\n");
    return 0;
}
fseek(fptr,0,SEEK_END);
size=ftell(fptr);
rewind(fptr);
randomkey(random_key);

printf("please type the password\n");
i=0;
firstkey[i]=getchar();
while(firstkey[i]!='\n')
{
    i++;
    firstkey[i]=getchar();
}
if(firstkey[0]!='\n')
{
    printf("illegal password");
    return 0;
}

```



```

Thekey(firstkey, charkey);
getbit(ejzkey, charkey); //convert the key to binary
replace(ejzkey, pc1, n=56); //key is replaced by 56 bits
getdkey(ejzkey); // get 16 subkeys
if(function==0){
    printf("Choose the encrypt mode:\t0.EBC\t1.CBC\n");
    scanf("%d",&mode_cbc);
}
else
    mode_cbc=1;
if(function==0 || function==2)
{

    if(mode_cbc!=0 && mode_cbc!=1)
    {
        printf("illegal choice\n");
    }
    printf("encrypting...\n");
    dfptr=fopen("encipher.bmp", "wb+");
    getbmphead(fptr, dfptr); // read the file header
    encode(); //encryption
    printf("Finish.The new file is encipher.bmp\n");
    fclose(fptr);
    fclose(dfptr);
}
if(function==0 || function==1)
{
    printf("decryteing...\n");
    fptr=fopen("encipher.bmp", "rb");
    dfptr=fopen("cipher.bmp", "wb+");
    if(fptr==NULL || dfptr==NULL){
        printf("Fail to open the file\n");
    }
    getbmphead(fptr, dfptr);
    decode(); //decrypt
    printf("Finish.The new file is cipher.bmp\n");
    //fclose(fptr);
    if(function==1){
        fptr=fopen("password","rb");
        dfptr=fopen("cipher.bmp","rb");
        char cmp,cmp2;
        do{
            printf("123\n");
            cmp=fgetc(fptr);

```

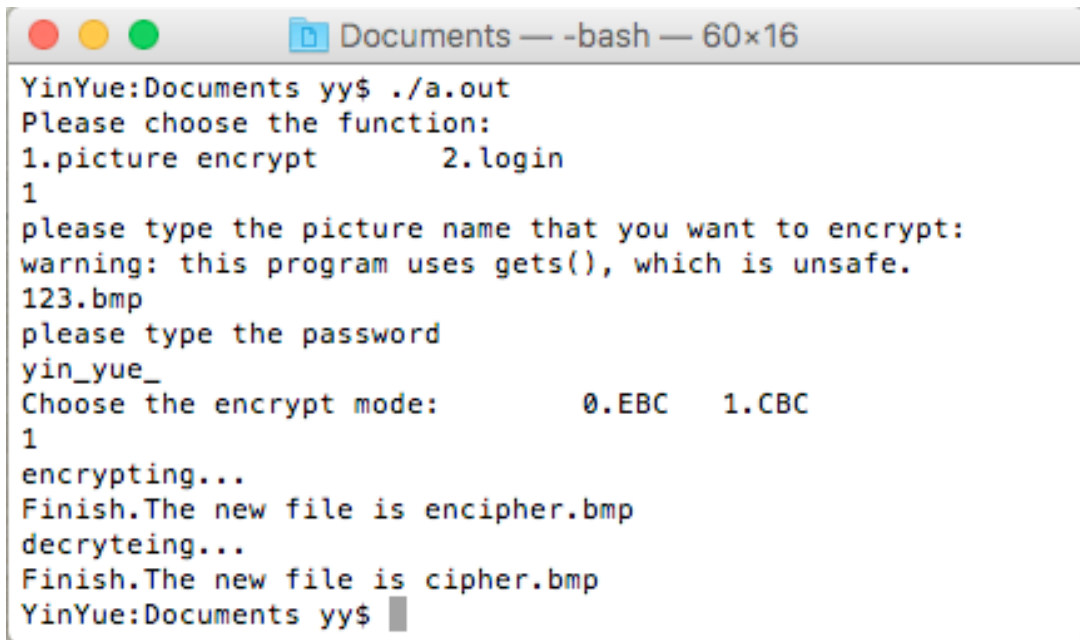
```

        cmp2=fgetc(dfptr);
        printf("\n1");
        if(cmp!=cmp2){
            printf("Fail to login,please check the password and try again\n");
            break;
        }
    }while(cmp!=EOF||cmp2!=EOF);
    if(cmp==cmp2)
        printf("Login successfully\n");
    fclose(fptr);
    fclose(dfptr);
}
}
return 0;
}

```

三、实验结果

图 3-1 图片加密程序运行操作



```

Documents — -bash — 60×16
YinYue:Documents yy$ ./a.out
Please choose the function:
1.picture encrypt      2.login
1
please type the picture name that you want to encrypt:
warning: this program uses gets(), which is unsafe.
123.bmp
please type the password
yin_yue_
Choose the encrypt mode:      0.EBC    1.CBC
1
encrypting...
Finish.The new file is encipher.bmp
decryteing...
Finish.The new file is cipher.bmp
YinYue:Documents yy$ █

```

图 3-2 原始 BMP 图片

```
YinYue:密码学 yy$ gcc -o DES DES.c
YinYue:密码学 yy$ ./DES
p:46181 q:9767
e:65537d:151813113pub key:(451049827,65537)
sec key:(451049827,151813113)
warning: this program uses gets(), which is unsafe.
please input the text you want to encrypt:Hello,I'm YinYue,My college is HIT(WH)
Hello,I'm YinYue,My college is HIT(WH)After the encrypt,the ciphertext is following
183087439      418373270      331941913      331941913
208477970      75829488      240612115      74586160 30806765
9      145173482      364152199      214078409      19676787
7      364152199      127279086      418373270      75829488
      205420693      328735048      145173482      21780739
5      208477970      331941913      331941913      41837327
0      25579702      418373270      145173482      21407840
9      33982361      145173482      183087439      24061211
5      95035621      69423203      345024067      18308743
9      391648175
After the decrypt,the plaintext is following
Hello,I'm YinYue,My college is HIT(WH)
YinYue:密码学 yy$
```

图 3-3 使用 CBC 模式加密后的图片



图 3-4 使用 EBC 模式加密后的图片：

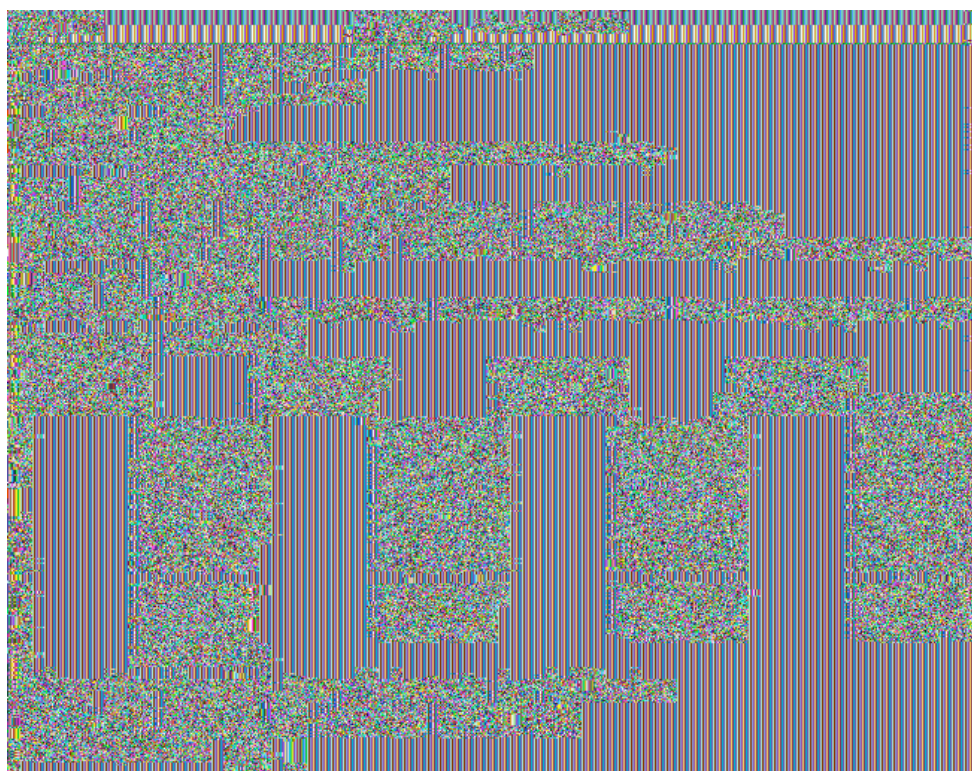


图 3-5 登录认证：

说明：第一次运行程序，设置密码，第二次运行程序执行登录过程

```
Documents — -bash — 68x23
YinYue:Documents yy$ vim shiyan2.c
YinYue:Documents yy$ gcc shiyan2.c
YinYue:Documents yy$ ./a.out
Please choose the function:
1.picture encrypt      2.login
2
1.Login 2.Set password
2
please type the password
yin_yue_
encrypting...
Finish.The new file is encipher.bmp
YinYue:Documents yy$ ./a.out
Please choose the function:
1.picture encrypt      2.login
2
1.Login 2.Set password
1
please type the password
yin_yue_
decryteing...
Finish.The new file is cipher.bmp
Login successfully
```

四、 实验中遇到的问题总结

1.输入后回车，下一行未接收输入，直接进行下一步

解决方法：回车后，缓冲区还保留有换行符，因此直接略过下个字符串接收，使用 `getchar();` 或者 `fflush(stdin);`可清空缓冲区残留的换行符

2.加密后的文件打不开

解决方案：不能对图片直接加密，图片的头部不能加密，需要单独保留出来

3.图片格式结构体在 Windows 下编译成功，在 Mac OS 和 Linux 下无法找到

解决方案：windows.h 包含了图片头的库函数，而 Linux 和 Mac OS 下则没有，需要自行编写图片头

4.加密汉字后，再解密，解密后的内容变成乱码

解决方案：加密过程中，加密后的某个字符结果为 0，从而字符串被截断，后面的过程未参与解密，导致了解密后文本乱码问题

5.在 Windows 下加密解密的汉字正常，在 Mac OS 和 Linux 下解密结果为乱码

解决方案：加密和解密的编码要保证一致

实验 2：实现 RSA 密码体制

姓名	殷悦	院系	计算机	学号	150120526
任课教师	刘杨	指导教师	刘杨		
实验地点	研究院中 403	实验时间	2017-10-09		
实验课表现	出勤、表现得分		实验报告得分		实验总分
	操作结果得分				

一、实验目的

- 1、编写程序构造一 RSA 密钥；
- 2、编写程序实现快速指数算法；
- 3、编写程序生成大素数；
- 4、实现 RSA 密码体制。

二、实验内容

一. 发展历史

1976年，两位美国计算机学家Whitfield Diffie 和 Martin Hellman，提出了一种崭新构思，可以在不直接传递密钥的情况下，完成解密。这被称为"Diffie-Hellman密钥交换算法"

1977年，三位数学家Rivest、Shamir 和 Adleman 设计了一种算法，可以实现非对称加密。这种算法用他们三个人的名字命名，叫做RSA算法。

RSA算法基于一个十分简单的数论事实：将两个大质数相乘十分容易，但是想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

RSA公开密钥密码体制。公开密钥密码体制就是使用不同的加密密钥与解密密钥，是一种“由已知加密密钥推导出解密密钥在计算上是不可行的”密码体制。

在公开密钥密码体制中，加密密钥（即公开密钥）PK是公开信息，而解密密钥（即秘密密钥）SK是需要保密的。加密算法E和解密算法D也都是公开的。虽然解密密钥SK是由公开密钥PK决定的，但却不能根据PK计算出SK。

RSA加密

$$\text{密文} = \text{明文}^E \pmod{N}$$

从通式可知，只要知道E和N任何人都可以进行RSA加密了，所以说E、N是RSA加密的密钥，也就是说E和N的组合就是公钥，我们用(E,N)来表示公钥

$$\text{公钥} = (E, N)$$

二. RSA加解密原理

$$\text{明文} = \text{密文}^D \pmod N$$

也就是说对密文进行D次方后除以N的余数就是明文，这就是RSA解密过程。知道D和N就能进行解密密文了，所以D和N的组合就是私钥

$$\text{私钥} = (D, N)$$

从上述可以看出RSA的加密方式和解密方式是相同的

加密是求“E次方的mod N”;解密是求“D次方的mod N”

E是加密(Encryption)的首字母
D是解密(Decryption)的首字母
N是数字(Number)的首字母。

公钥	(E, N)
私钥	(D, N)
密钥对	(E, D, N)
加密	密文 = 明文 ^E mod N
解密	明文 = 密文 ^D mod N

生成密钥对

公钥是 (E, N)，私钥是 (D, N) 所以密钥对即为 (E, D, N)

密钥对生成步骤如下：

- >求N
- >求L (L为中间过程的中间数)
- >求E
- >求D

1.求N

准备两个质数p, q。这两个数不能太小，太小则会容易破解，将p乘以q就是N

$$N = p * q$$

2.求L

L 是 p-1 和 q-1的最小公倍数，可用如下表达式表示 $L = \text{lcm}(p-1, q-1)$

3.求D

数D是由数E计算出来的。D、E和L之间必须满足以下关系：

$$1 < D < L$$

$$E * D \bmod L = 1$$

只要D满足上述2个条件，则通过E和N进行加密的密文就可以用D和N进行解密。
 简单地讲条件2是为了保证密文解密后的数据就是明文。
 现在私钥自然也已经生成了，密钥对也就自然生成了。

4.求N

$$N = p * q ; p, q \text{ 为质数}$$

求 $L = \text{lcm}(p-1, q-1)$; L为p-1、q-1的最小公倍数

求 $1 < E < L, \text{gcd}(E, L) = 1$; E, L最大公约数为1 (E和L互质)

求 $1 < D < L, E * D \bmod L = 1$

具体实践:

5.1 求N

我们准备两个很小对质数，

$$p = 17$$

$$q = 19$$

$$N = p * q = 323$$

5.2 求L

$$L = \text{lcm}(p-1, q-1) = \text{lcm}(16, 18) = 144$$

144为16和18的最小公倍数

5.3 求E

求E必须要满足2个条件: $1 < E < L, \text{gcd}(E, L) = 1$

即 $1 < E < 144, \text{gcd}(E, 144) = 1$

E和144互为质数，5显然满足上述2个条件

故 $E = 5$

此时公钥 $= (E, N) = (5, 323)$

5.4 求D

求D也必须满足2个条件: $1 < D < L, E * D \bmod L = 1$

即 $1 < D < 144, 5 * D \bmod 144 = 1$

显然当 $D = 29$ 时满足上述两个条件

$$1 < 29 < 144$$

$$5 * 29 \bmod 144 = 145 \bmod 144 = 1$$

此时私钥 $= (D, N) = (29, 323)$

5.5 加密

准备的明文必须时小于N的数，因为加密或者解密都要mod N其结果必须小于N

假设明文 = 123

则

$$\text{密文} = \text{明文}^E \bmod N = 123^5 \bmod 323 = 225$$

5.6 解密

$$\text{明文} = \text{密文}^D \bmod N = 225^{29} \bmod 323 = 123$$

解密后的明文为123。

三. 代码说明

包含三个头文件，time.h 用于生成随机数时的种子函数

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

为了保证计算不会越界，计算时采用无符号 long long 型，长度为 64 位

```
typedef unsigned long long ull;
```

gcd 用于计算两个数的最大公约数，这里采用欧几里得的辗转相除法。

辗转相除法就是把上一轮有余数的除法计算中，除数变为下一轮计算的被除数，余数变为下一轮计算的除数，一直这样计算下去，直到最后一次计算余数为零，在最后一轮计算中的被除数，即为所求的最大公约数。

举例： 105 和 85 的最大公约数

第一轮计算 $105 \div 85 = 1 \dots 20$

第二轮计算 $85 \div 20 = 4 \dots 5$

第三轮计算 $20 \div 5 = 4$

第三轮没有余数，因此 105 和 85 的最大公约数就是第三轮计算的被除数

5.

```
ull gcd(ull a, ull b){
    int t;
    while(b){
        t=a;
        a=b;
        b=t%b;
    }
    return a;
}
```

产生密钥对

```
void generateKey(ull p,ull q,ull* e,ull* d){
```

产生公钥:

在这里, e 任意取, 但是需要满足 e 和 $(p-1) * (q-1)$ 的最大公因数为 1

```
//generatePubKey
```

```
int r;
```

```
(*e)=2;
```

```
r=(p-1)*(q-1);
```

```
(*e)=65537;
```

```
while((*e)<1||(*e)>r||gcd((*e),r)!=1){
```

```
    (*e)++;
```

当 e 等于 -1 是说明 e 已经达到最大值溢出

```
    if((*e)==-1){
```

```
        printf("error,can't find e,overflow\n");
```

```
        return;
```

```
    }
```

```
}
```

```
printf("e:%llu",*e);
```

产生私钥:

在这里, d 需要满足 e 乘 d 除以 r 的余数为 1

```
//generateSecKey
```

```
*d=1;
```

```
while((*e)*(*d)%r!=1){
```

```
    *d=rand()%r;
```

```
    //(*d)++;
```

```
    //printf("d:%llu",*d);
```

当 d 等于 -1 是说明 e 已经达到最大值溢出

```
    if((*d)==-1){
```

```
        printf("error,can't find b,overflow\n");
```

```
        return;
```

```
    }
```

```
}
```

```
printf("d:%llu",*d);
```

```
}
```

```
/*
```

```
ull MulMod(ull a,ull b,ull mod){
```

```
    int num1[64],num2[64],res[128];
```

```
    int opr1,opr2,i;
```

```
    ull o1,o2;
```

```
    for(opr1=0,opr2=0;opr1<64;opr1++){
```

```

        num1[opr1]=0;
        num2[opr1]=0;
        res[opr2++]=0;
        res[opr2+0
o1=1;o2=1;
for(opr1=0;opr1<64;opr1++){
    num1[opr1]=(a&o1)?1:0;
    o1<=<=1;
}
for(opr2=0;opr2<64;opr2++){
    num2[opr2]=(a&o2)?1:0;
    o2<=<=1;
}
for(opr1=0;opr1<64;opr1++){
    for(opr2=0;opr2<64;opr2++){
        if(num1[opr1]&num2[opr2])
            res[opr1+opr2-1]++;
    }
    for(i=0;i<128;i++){
        if(res[i]>1){
            res[i]=0;res[i+1]++;
        }
    }
}

o2=1;
for(i=0;i<64;i++){
    if(res[i])
        o1+=o2;
    o2<=<=1;
}
return o1;
}
*/
/*
ull powAndMod(ull x,ull m,ull mod){
    ull y;
    y=1;
    while(m!=0){
        if(m%2==1){
            y=(x*y)%mod;
            m--;
        }else{
            x=(x*x)%mod;

```

```

        m/=2;
    }
}
return y;
}

```

*/

快速指数求余

朴素模幂运算是非常的无法容忍的，模运算是非常的消耗内存资源的，在计算的次数非常的大的时候，我们是没办法忍受这种时间耗费的

$(a*b)\%c=(a\%c)*(b\%c)\%c$ 这个是成立的，也是我们实现快速幂的基础

对于任何一个整数的模幂运算 $a^b\%c$

对于 b 我们可以拆成二进制的形式 $b=b_0+b_1*2+b_2*2^2+\dots+b_n*2^n$

这里我们的 b_0 对应的是 b 二进制的第一位

那么我们的 a^b 运算就可以拆解成

$a^{b_0}*a^{b_1*2}*\dots*a^{(b_n*2^n)}$

对于 b 来说，二进制位不是 0 就是 1，那么对于 b_x 为 0 的项我们的计算结果是 1 就不用考虑了，我们真正想要的其实是 b 的非 0 二进制位

那么假设除去了 b 的 0 的二进制位之后我们得到的式子是

$a^{(b_x*2^x)}*\dots*a^{(b_n*2^n)}$

这里我们再应用我们一开始提到的公式，那么我们的 $a^b\%c$ 运算就可以转化为 $(a^{(b_x*2^x)\%c}*\dots*(a^{(b_n*2^n)\%c}))\%c$

```

ull quickMod(ull a,ull b,ull mod){
    //a^10%c=a^(1*2^3+0*2^2+1*2^1+0*2^0)%c=((a^(2^3)%c)*(a^(2^1)%c
))%c
    ull aus=1;
    while(b){
        if(b&0x1)
            aus=aus*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return aus;
}

```

欧拉定理：

对于和 m 互素的 x ，有 $x^{\phi(m)} \equiv 1 \pmod{m}$

证明：

设所有 n 以下和 n 互质的数依次为 $X_1, X_2, \dots, X_{\phi(n)}$

设 k 为一个与 n 互质的数，那么设 $A=\{kX_1, kX_2, \dots, kX_{\phi(n)}\}$

那么 A 中没有两个数模 n 同余

证明：假设 $ak \equiv bk \pmod{n}$

那么有 $ak - bk = nq$ ，即 $(a-b)k = nq$ ，所以左式模 n 为 0

然而 k 与 n 互质, $1 < (a-b) < n$, 所以 $(a-b)$ 也模 n 不等于 0

那么显然上式不成立

得证

A 中所有数的余数都与 n 互质

证明: 假设 $\gcd(kX_i \bmod n, n) = r$

那么 $kX_i = qn + pr$

那么 kX_i 也有因子 r , 那么 kX_i 与 n 不互质, 显然不可能

得证

那么由以上两个结论可知 A 中的数模 n 的余数应该与 $X_1, X_2, \dots, X_{\phi(n)}$ 唯一对应。

即 $X_1 * X_2 * \dots * X_{\phi(n)} \equiv kX_1 * kX_2 * \dots * kX_{\phi(n)} \pmod{n}$

也就是说 $0 \equiv (k^{\phi(n)} - 1) * X_1 * X_2 * \dots * X_{\phi(n)} \pmod{n}$

显然 $X_1 * X_2 * \dots * X_{\phi(n)}$ 是与 n 互质的, 所以 $k^{\phi(n)} - 1 \equiv 0 \pmod{n}$

$k^{\phi(n)} \equiv 1 \pmod{n}$

得证

费马小定理:

特别的, 当 p 为素数时, x 无法被 p 整除, $\phi(p) = p-1$, 于是便有费马小定理 $X^{p-1} \equiv 1 \pmod{p}$

在 p 是素数时, 对任意正整数 x 都有 $X^p \equiv X \pmod{p}$

于是对于 a 的逆元 x , 有 $ax \equiv 1 \pmod{m}$, 对于 a, m 互素且 m 为素数时, 有 $x = a^{m-2}$, 于是我们可以通过快速幂快速求出 a 的逆元。

另外, 借助素数筛, 我们还可以很快的求出 $1-n$ 的欧拉函数值。每当我们找到一个素数, 就把他的倍数的欧拉函数值乘上 $(p-1)/p$ 。

而且, 借助费马小定理我们可以实现对除法取模。

```
int fermatPrime(ull num){
    //a^(n-1)=1(mod n)
    int i;
    for(i=0;i<5;i++){
        if(quickMod(1+rand()%(num-1),num-1,num)!=1)
            break;
    }
    if(i==5)return 1;
    return 0;
}
```

```
void generateBigPrime(ull* p, ull* q){
    int i;
    ull temp;
    srand((unsigned)time(NULL));
    do{
        *p=0;
        for(i=0;i<1;i++){
            temp=rand()%65535;
```

```

        temp<=16*i;
        *p|=temp;
    }
//    printf("p:%llu\tq:%llu\n",*p,*q);
}while(!fermatPrime(*p));

do{
    *q=0;
    for(i=0;i<1;i++){
        temp=rand()%65535;
        temp<=16*i;
        *q|=temp;
    }
}while(p==q||!fermatPrime(*q));
printf("p:%llu\tq:%llu\n",*p,*q);
}

```

加密函数

逐个字节对该数进行加密，执行密文=明文^E(mod N)

```

void enc(char plaintext[],ull ciphertext[],ull n,ull e){
    int i;
    for(i=0;plaintext[i]!=0;i++){
        ciphertext[i]=quickMod((ull)plaintext[i],e,n);
        printf("%c",plaintext[i]);
    }
    ciphertext[i]=0;
}

```

解密函数

逐个字节对该数进行解密，执行明文=密文^D(mod N)

```

void dec(ull ciphertext[],char plaintext[],ull n,ull d){
    int i;
    for(i=0;ciphertext[i]!=0;i++){
        plaintext[i]=(char)quickMod(ciphertext[i],d,n);
        ciphertext[i]=0;
    }
}

```

打印大数串

```

void printfx(ull ciphertext[]){
    int i;
    for(i=0;ciphertext[i]!=0;i++){
        printf("%llu\t",ciphertext[i]);
    }
}

```

```

    }
    printf("\n");
}

int main(){
    /* prime p q
    * n=p*q r=(p-1)(q-1)
    * e<r gcd(e,r)=1 (e*d)mod r=1
    * pub key (n,e) sec key (n,d)
    * plaintext A ciphertext B
    * A=B^e(mod n) B=A^d(mod n)
    * e d can be exchanged
    * */
    ull p=0,q=0;
    ull e,d;
    /*
    p=1725580387;
    q=1816490293;
    e=65537;
    d=5685353;
    */
    char text[1000];
    ull textx[1000];
    generateBigPrime(&p,&q);
    generateKey(p,q,&e,&d);
    //printf("p:%llu\tq:%llu\te:%llu\td:%llu\n",p,q,e,d);

    printf("pub key:(%llu,%llu)\nsec
key:(%llu,%llu)\n",p*q,e,p*q,d);

    printf("please input the text you want to encrypt:");
    gets(text);
    enc(text,textx,p*q,e);
    printf("After the encrypt,the ciphertext is following\n");
    printfx(textx);
    dec(textx,text,p*q,d);
    printf("After the decrypt,the plaintext is following\n");
    printf("%s\n",text);
    return 0;
}

```

三、实验结果

```
YinYue:密码学 yy$ gcc -o DES DES.c
YinYue:密码学 yy$ ./DES
YinYue:密码学 yy$ ./DES
YinYue:密码学 yy$ ./a.out
p:46181 q:9767
e:65537d:151813113pub key:(451049827,65537)
sec key:(451049827,151813113)
warning: this program uses gets(), which is unsafe.
please input the text you want to encrypt:Hello,I'm YinYue,My college is HIT(WH)
Hello,I'm YinYue,My college is HIT(WH)After the encrypt,the ciphertext is following
183087439      418373270      331941913      331941913
208477970      75829488      240612115      74586160 30806765
9      145173482      364152199      214078409      19676787
7      364152199      127279086      418373270      75829488
      205420693      328735048      145173482      21780739
5      208477970      331941913      331941913      41837327
0      25579702      418373270      145173482      21407840
9      33982361      145173482      183087439      24061211
5      95035621      69423203      345024067      18308743
9      391648175
After the decrypt,the plaintext is following
Hello,I'm YinYue,My college is HIT(WH)
YinYue:密码学 yy$
```

四、实验中遇到的问题总结

- 1.p, q 值过大, 导致加密后数据溢出 (使用 unsigned long long)
- 2.加密后的数据用 char 类型存储溢出 (char 类型太小, 改用 int 或更大的数据类型存储加密结果更好)
- 3.scanf 遇到空格直接返回结果, 没有获取空格后的字符串, 无法一次读取学号+空格+姓名 (改用 gets 函数)
scanf 函数与 gets 函数冲突
(这是因为二者使用的结束标记不同。输入字符串时, scanf()遇到空格、回车、Tab 结束, 但在缓冲区中还留着这些结束符, 此后如果使用 gets()想去获取下一行字符串, 它碰到的却是前面遗留下来的回车(或者回车之前还有空格等空白符), 那么这次 gets()就直接失效了, 解决方法: 用一句 while(getchar()!='\n'); 来处理掉缓冲区里的回车换行符, 或者改用 cin 函数)
- 4.p, q 值过小, 导致 N 过小, 加密后的字符无法被解密还原为原文
(这是因为公式 $\text{mod } N$, 结果的范围从 $0 \sim N-1$, 所以假如 N 小于原文的值则解密将出错, 解决方法为扩大 p, q 取值, 所以 p, q 取 8bit 范围的数可能会出错, 范围要扩大到涵盖原文取值, 建议取值为逼近 8bit 的素数)