南　开　大　学

网络空间安全学院

密码学实验报告二

Lab2 分组密码算法 DES

学号：　2013484

姓名：　张世伟

年级：　2020 级

专业：　信息安全-法学

2022 年 11 月 24 日

摘　要

本文 C++ 语言实现了 DES 算法，并检测计算了雪崩效应

关键词: C++，DES，雪崩效应

目　录

摘要 **I**

第 1 节 DES 加解密算法 **1**

A 附录 1 **14**

第 1 节　DES 加解密算法

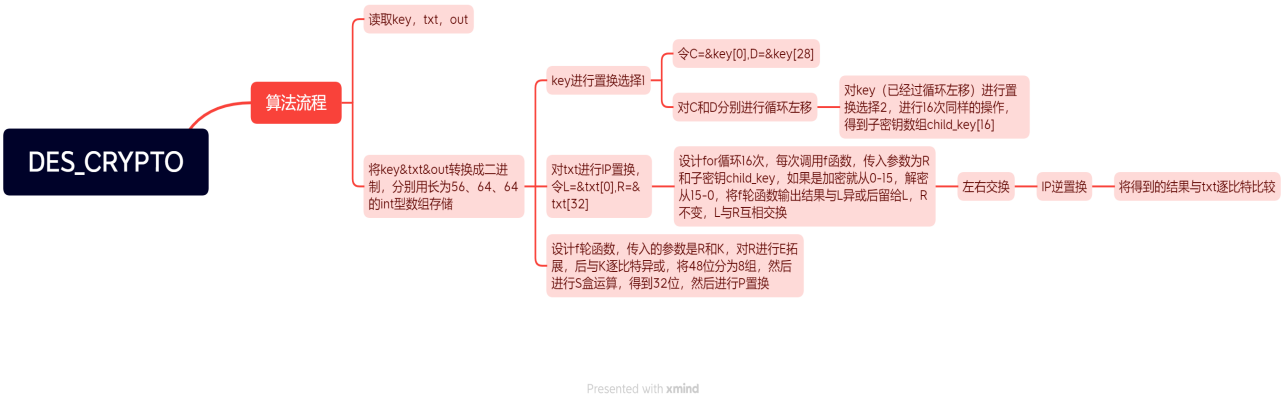

图 1.1: DES 算法流程图

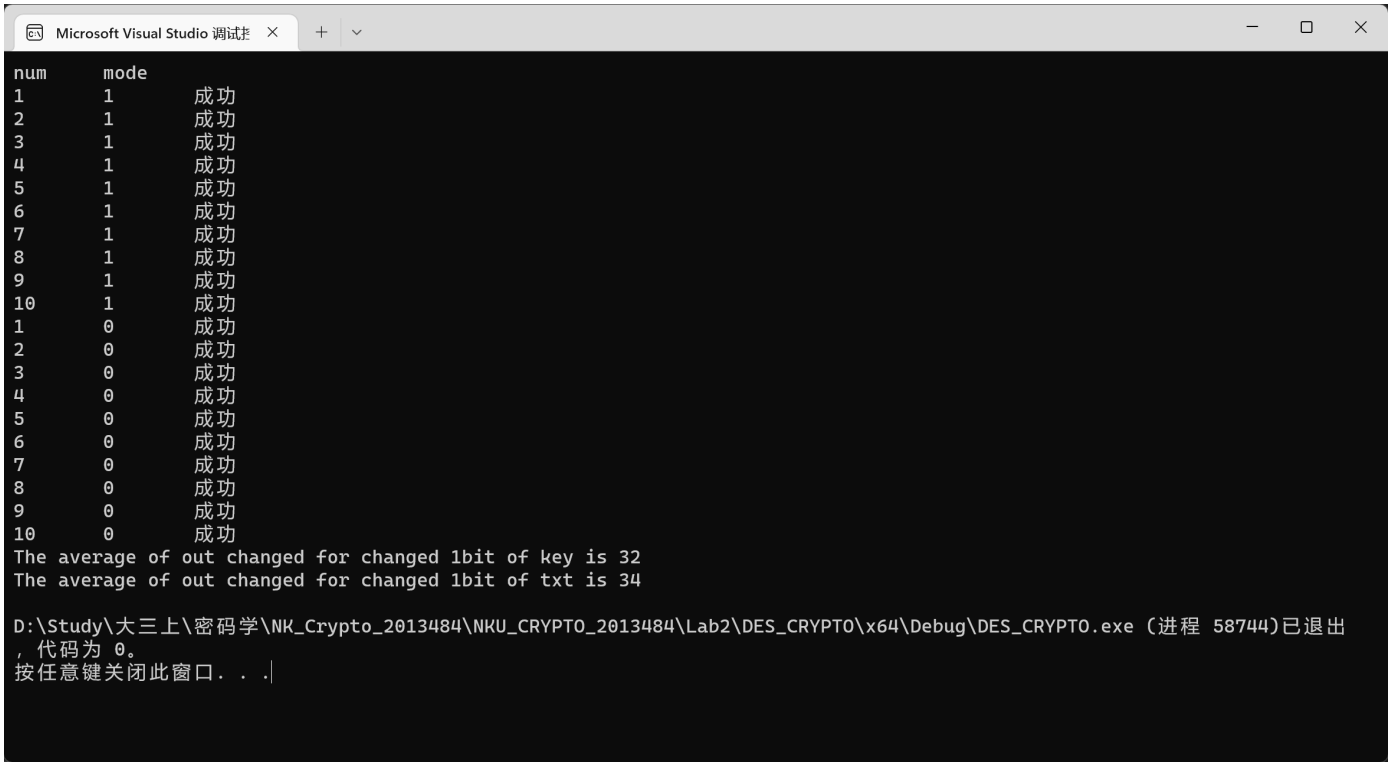

图 1.2: 输出结果演示

20 组样例全部正确，改变 key1 位，8 次得到的平均值是 32，改变 txt1 位，8 次计算得到的平均值是 34

接下来概括性写一下中间生成结果（以第一组数据为例）

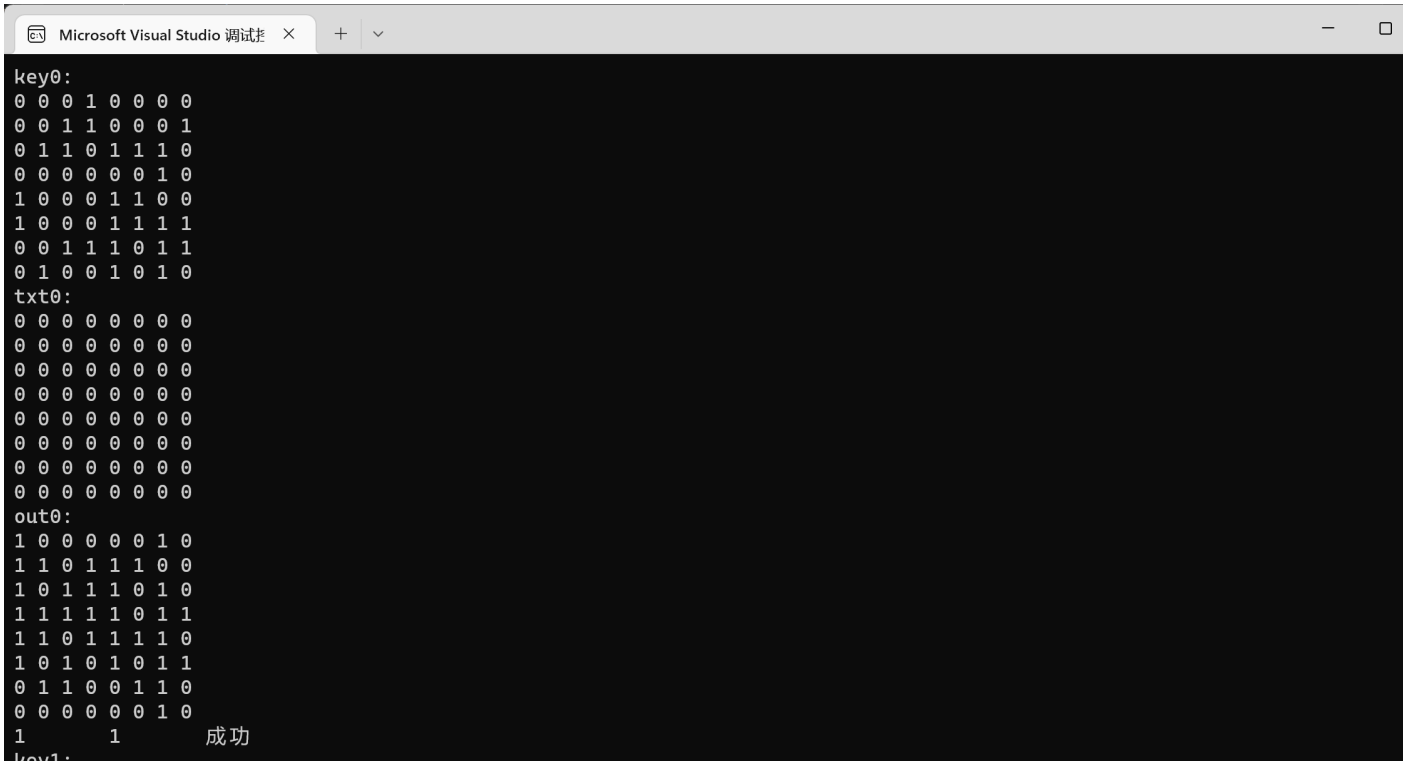

图 1.3: 读入的第一组数据

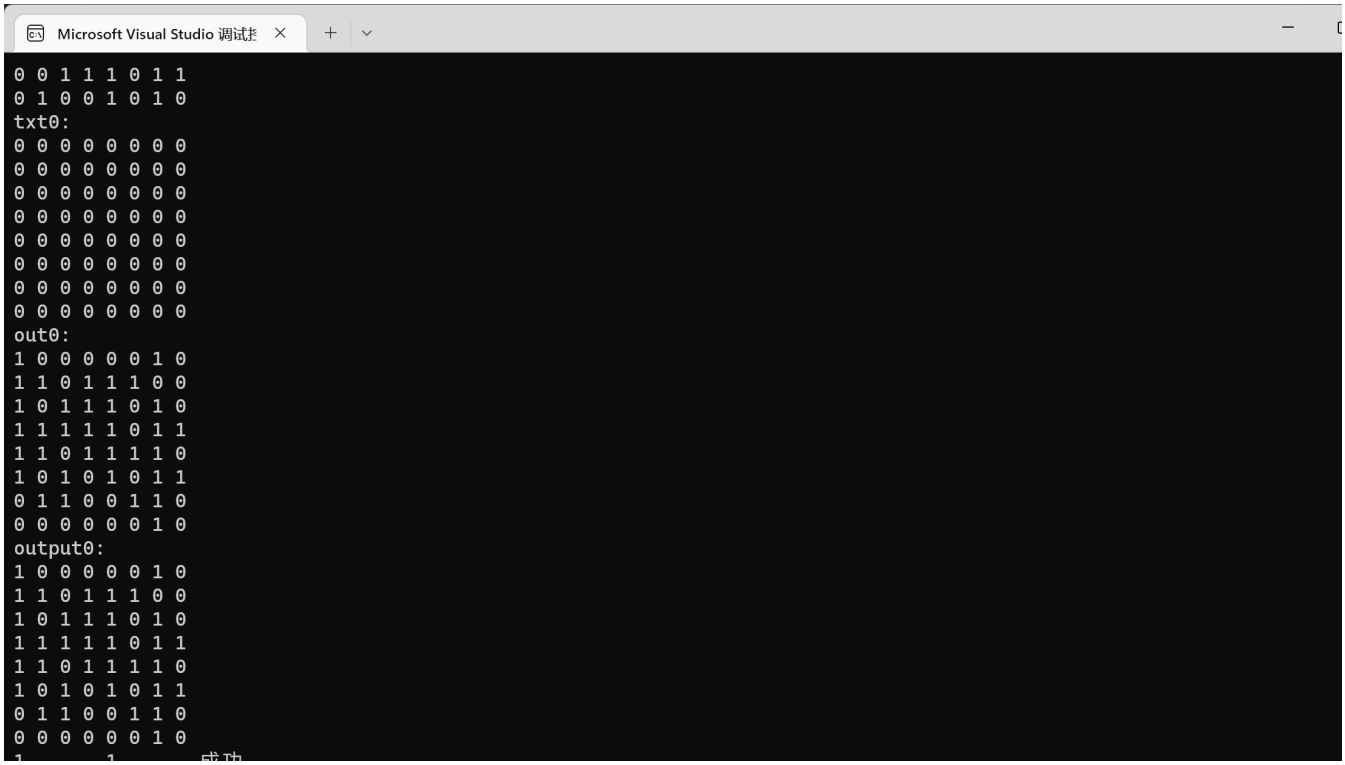

图 1.4: 第一组计算结果与标准结果一致

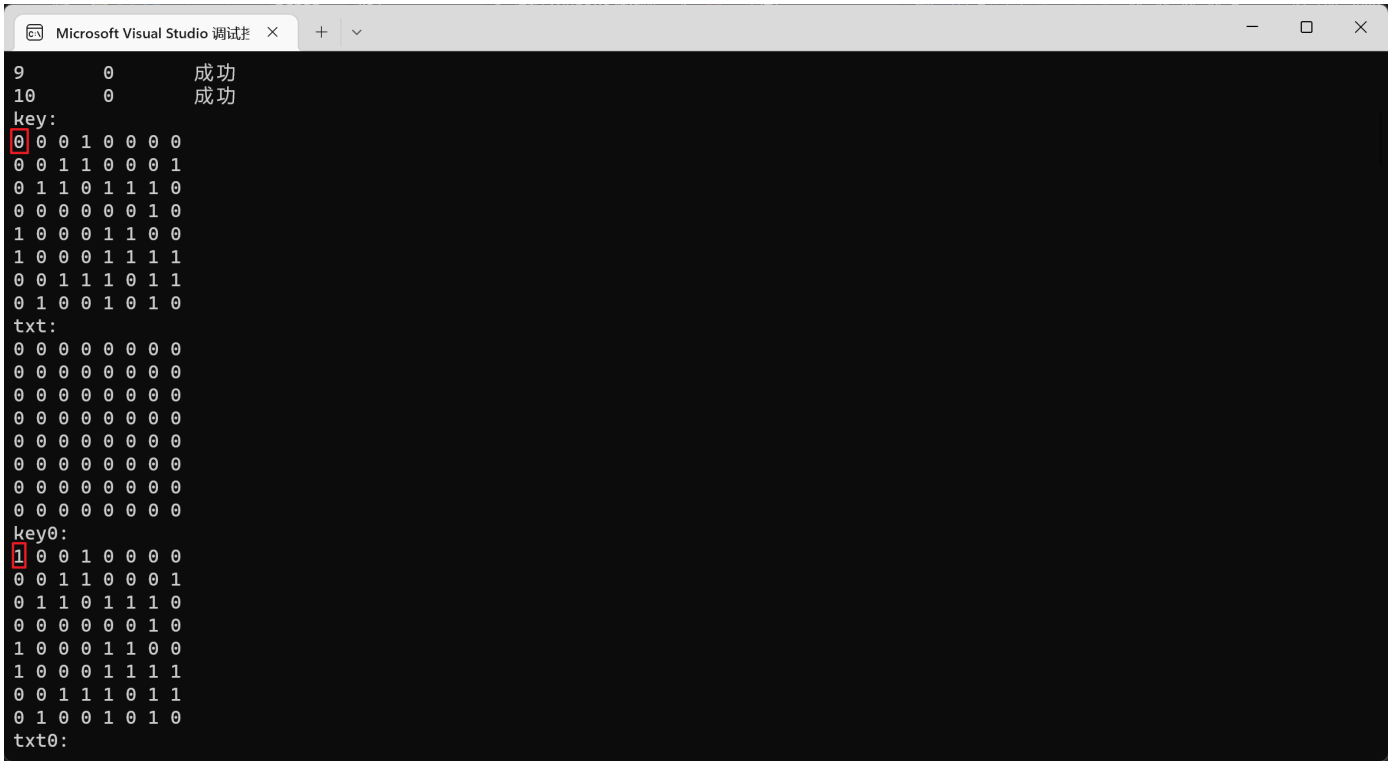

图 1.5: 计算雪崩效应，改变 key 的一位，txt 不变

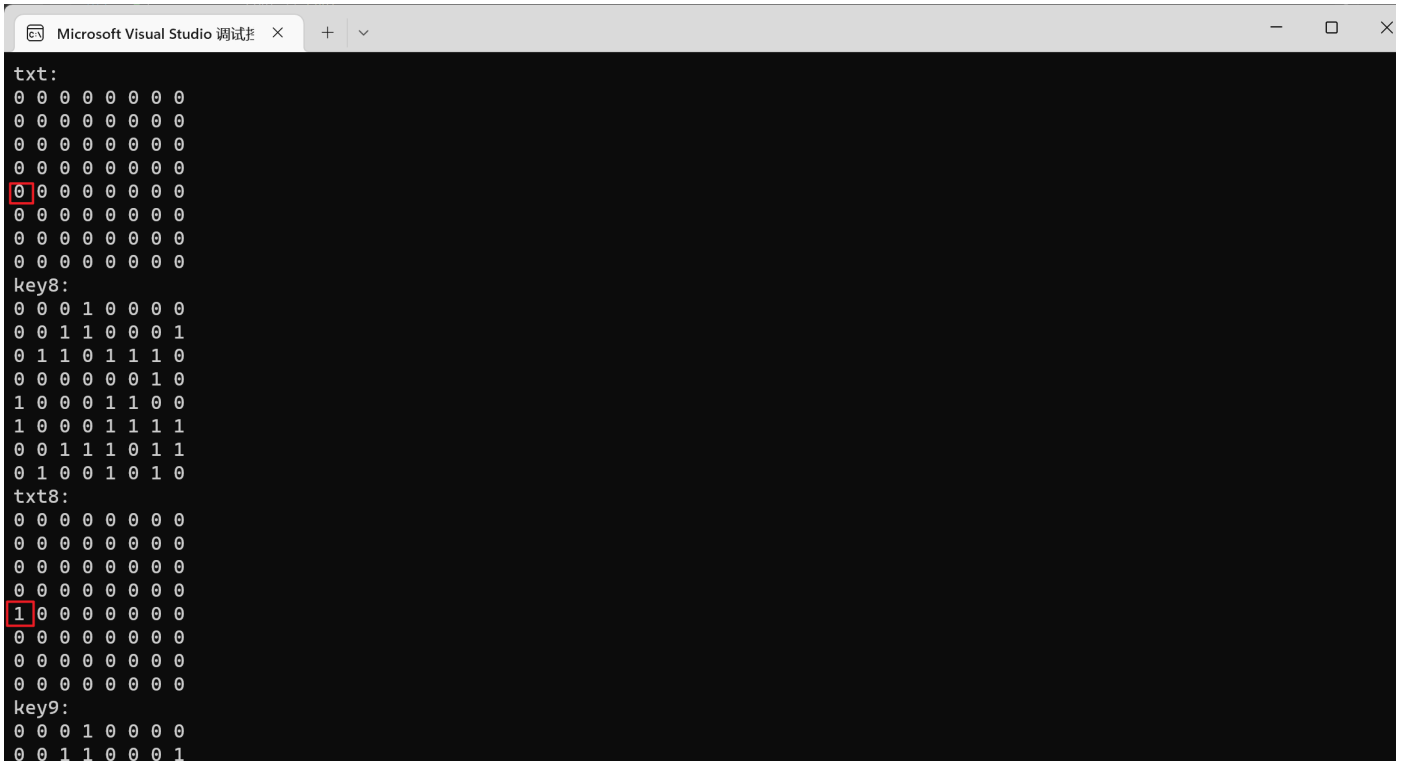

图 1.6: 计算雪崩效应，改变 txt 的一位，key 不变

```cpp
// DES_CRYPTO.cpp :
#include <iostream>
#include"des_test_data.h"
using namespace std;

int* child_key[16];//储存子密钥
//为了代码精简删掉表项，详情请见DES_CRYPTO.sln
const int IP_table[64] = {  };

const int IP_1_table[64] = {
};
const int LS_table[16] = {
};
const int PC_1_table[56] = {};
const int PC_2_table[48] = { };
const int E_table[48] = { };
const int S_Box_table[32][16] = {  };
const int P_table[32] =
{
};

int* int2dec(int a)//a是int型，比如140
{
        int* b = new int[2];
        if (!a)
        b[0] = b[1] = 0;
        else
        {
                b[1] = a % 16;
                b[0] = a / 16;
        }
        return b;
}

void DEC2BIN(int a, int* b, int k)
{//将10进制数转换成2进制数，并用长度为4的int数组保存
```

```
37          for (int i = 0; i < 8; i++)
38          b[k + i] = 0;
39          int* c = new int[2];
40          c = int2dec(a);
41          int temp;
42          for (int i = 0; i < 2; i++)
43          {
44                  if (!i)
45                  temp = k + 3;
46                  else
47                  temp = k + 7;
48                  while (c[i])
49                  {
50                          int j = c[i] % 2;
51                          c[i] /= 2;
52                          b[temp--] = j;
53                  }
54          }
55  }
56
57  int* IP_replace(int* text)//IP置换
58  {
59          int* temp = new int[64];
60          for (int i = 0; i < 64; i++)
61          {
62                  temp[i] = text[IP_table[i] - 1];
63          }
64          return temp;
65  }
66
67  int* IP_1_replace(int* text)//IP逆置换
68  {
69          int* temp = new int[64];
70          for (int i = 0; i < 64; i++)
71          {
72                  temp[i] = text[IP_1_table[i] - 1];
```

```
73          }
74          return temp;
75  }
76
77  int* PC_1_replace(int* text)//压缩置换PC-1
78  {
79          int k = 0;
80          int* temp = new int[56];
81          for (int i = 0; i < 56; i++)
82          {
83                  temp[i] = text[PC_1_table[i] - 1];
84          }
85          return temp;
86  }
87
88  int* PC_2_replace(int* text)//压缩置换PC-2
89  {
90          int* temp = new int[48];
91          for (int i = 0; i < 48; i++)
92          {
93                  temp[i] = text[PC_2_table[i] - 1];
94          }
95          return temp;
96  }
97
98  void LS(int* text,int K_num)
99  {//循环左移,K_num是第几个子密钥
100         for (int j = 0; j < LS_table[K_num]; j++)
101         {
102                 int* L = &text[0];
103                 int* R = &text[28];
104                 int temp1_1, temp1_2;
105                 temp1_1 = L[0], temp1_2 = R[0];
106                 for (int i = 0; i < 27; i++)
107                 {
108                         L[i] = L[i + 1];
```

```
109                         R[i] = R[i + 1];
110                 }
111                 L[27] = temp1_1;
112                 R[27] = temp1_2;
113         }
114 }
115
116 int* f(int* R, int* K)
117 {
118         int* temp = new int[48];
119         for (int i = 0; i < 48; i++)
120         {
121                 temp[i] = R[E_table[i] - 1];//E拓展
122                 if (temp[i] == K[i])//E拓展后与子密钥异或
123                 temp[i] = 0;
124                 else
125                 temp[i] = 1;
126         }
127         int* tmp = new int[32];
128         for (int i = 0; i < 32; i++)
129         tmp[i] = 0;
130         for (int s = 0; s < 8; s++)//s代表盒号
131         {
132                 int k = s * 6;
133                 //i, j用来定位在盒里的坐标s[i][j]
134                 int i= temp[k] * 2 + temp[k + 5];
135                 int j = temp[k + 1] * 8 + temp[k + 2] * 4
136                  + temp[k + 3] * 2 + temp[k + 4];
137                 int num = S_Box_table[s * 4 + i][j];
138                 int N = s * 4 + 3;
139                 while (num)
140                 {
141                         int i = num % 2;
142                         num /= 2;
143                         tmp[N--] = i;
144                 }
```

```
145             }
146             int* tmpp = new int[32];
147             for (int i = 0; i < 32; i++)
148             {
149                     tmpp[i] = tmp[P_table[i] - 1];
150             }
151             return tmpp;
152     }
153
154     int main()
155     {
156             //保存第0组数据，用于后续雪崩检验
157             int key0[64], txt0[64], out0[64];
158             cout << "num" << "\tmode" << endl;
159             //20组数据，前10组为加密，后10组为解密
160             for (int i = 0; i < 20; i++)
161             {
162                     int k = 0;
163                     int* key = new int[64];//储存密钥
164                     int* txt = new int[64];//储存txt
165                     int* output = new int[64];//DES计算的结果
166                     int* out = new int[64];//正确的结果
167
168
169     //将key,txt,out转换成2进制，共64位，存储在长位64的int型数组里面
170                     for (int j = 0; j < 8; j++)
171                     {
172                             DEC2BIN((int)cases[i].key[j], key, k);
173                             DEC2BIN((int)cases[i].txt[j], txt, k);
174                             DEC2BIN((int)cases[i].out[j], out, k);
175                             k += 8;
176                     }
177                     //保存第0组的数据，用于后续雪崩效应的计算
178                     if (i == 0)
179                     {
180                             for (int i = 0; i < 64; i++)
```

```
181                         {
182                                 key0[i] = key[i];
183                                 txt0[i] = txt[i];
184                         }
185                 }
186                 int* temp = PC_1_replace(key);//压缩置换1
187                 for (int i = 0; i < 16; i++)
188                 {
189                         //循环左移
190                         LS(temp, i);
191                         //压缩置换2
192                         child_key[i] = PC_2_replace(temp);
193                 }
194                 txt = IP_replace(txt);//IP置换
195                 int* L = &txt[0];
196                 int* R = &txt[32];
197                 int j;//j用来区分加密和解密使用的子密钥不同
198                 if (cases[i].mode)
199                 {
200                         j = 0;
201                 }
202                 else
203                         j = 15;
204                 for (int k = 0; k < 16; k++)//16轮
205                 {
206                         int* tmpp;
207                         if (cases[i].mode)
208                         {//加密，子密钥正序使用0-15
209                                 tmpp = f(R, child_key[j++]);
210                         }
211                         else//解密，子密钥逆序使用15-0
212                                 tmpp = f(R, child_key[j--]);
213                         for (int i = 0; i < 32; i++)
214                         {//Ri=Li-1+f(Ri-1,Ki)
215                                 if (tmpp[i] == L[i])
216                                         L[i] = 0;
```

```
217                                 else
218                                         L[i] = 1;
219                         }
220                         int* temp = L;
221                         L = R;
222                         R = temp;
223                 }
224                 for (int i = 0; i < 32; i++)//左右交换
225                 {
226                         swap(L[i], R[i]);
227                 }
228                 output = IP_1_replace(txt);//IP逆置换
229                 if (i == 0)
230                 {
231                         for (int i = 0; i < 64; i++)
232                         {
233                                 out0[i] = output[i];
234                         }
235                 }
236         //用来判断计算得到的output与标准答案out是否一致
237                 bool t = true;
238                 for (int k = 0; k < 64; k++)
239                 {
240                         if (out[k] != output[k])
241                         {
242                                 t = false;//不一致
243                                 break;
244                         }
245                 }
246                 if (t)
247                         cout << cases[i].num << "\t" <<
248                          cases[i].mode << "  \t成功" << endl;
249                 else
250                         cout << cases[i].num << "\t" <<
251                         cases[i].mode << "  \t失败" << endl;
252         }
```

```
253
254         //测试雪崩效应，选取第一组数据
255         int num = 0;
256 //计算16次，前8次计算改变1bit密钥，不改变明文
257 //后8次计算改变1bit明文，不改变密钥
258         for (int i = 0; i < 16; i++)
259         {
260                 if (i % 8 == 0)
261                 num = 0;
262                 int* testkey = new int[64];
263                 int* testtxt = new int[64];
264                 int* testout = new int[64];
265                 if (i < 8)
266                 {
267                         for (int j = 0; j < 64; j++)
268                         {
269                                 if (j == i * 8)
270                                 {
271                                         testkey[j] = (key0[j] + 1) % 2;
272                                 }
273                                 else
274                                 {
275                                         testkey[j] = key0[j];
276                                 }
277                                 testtxt[j] = txt0[j];
278                         }
279                 }
280                 else
281                 {
282                         for (int j = 0; j < 64; j++)
283                         {
284                                 if (j == i * 4)
285                                 {
286                                         testtxt[j] = (txt0[j] + 1) % 2;
287                                 }
288                                 else
```

```
289                                 {
290                                         testtxt[j] = txt0[j];
291                                 }
292                                 testkey[j] = key0[j];
293                         }
294                 }
295                 int* temp = PC_1_replace(testkey);
296                 for (int i = 0; i < 16; i++)
297                 {
298                         LS(temp, i);
299                         child_key[i] = PC_2_replace(temp);
300                 }
301                 testtxt = IP_replace(testtxt);
302                 int* L = &testtxt[0];
303                 int* R = &testtxt[32];
304                 for (int k = 0; k < 16; k++)
305                 {
306                         int* tmpp;
307                         tmpp = f(R, child_key[k++]);
308                         for (int i = 0; i < 32; i++)
309                         {
310                                 if (tmpp[i] == L[i])//Ri=Li-1+f(Ri-1,Ki)
311                                 L[i] = 0;
312                                 else
313                                 L[i] = 1;
314                         }
315                         int* temp = L;
316                         L = R;
317                         R = temp;
318                 }
319                 for (int i = 0; i < 32; i++)//左右交换
320                 {
321                         swap(L[i], R[i]);
322                 }
323                 testout = IP_1_replace(testtxt);
324                 for (int i = 0; i < 64; i++)
```

```
325                     {
326                             if (out0[i] != testout[i])
327                             num++;
328                     }
329                     if (i == 7)
330                             cout << "The average of out changed
331                             for changed 1bit of key is " <<
332                              float(num / 8) << endl;
333                     if (i == 15)
334                             cout << "The average of out changed
335                             for changed 1bit of txt is " <<
336                              float(num / 8) << endl;
337             }
338 }
```