**DES算法详细设计**

1. **算法原理概述**

数据加密标准 (DES，Data Encryption Standard) 是一种使 用密钥加密的块密码，1976年被美国国家标准局 (NBS, National Bureau of Standards，1988年改名为 NIST) 确定为联邦信息处理标准 (FIPS)，随后在国际上获得广泛采用。

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。

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。

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。

DES 算法的基本过程是换位和置换。

**2. 总体结构**

**2.1基本信息**

设信息空间由{0, 1}组成的字符串构成，明文信息和经过DES加密的密文信息是64位的分组，密钥也是64位。

明文：

密文：

密钥：

◌ 除去k8, k16, …, k64 共8位奇偶校验位，起作用的仅为56位。

加密过程

 ,其中为初始置换，是的逆，是一系列的迭代变换。

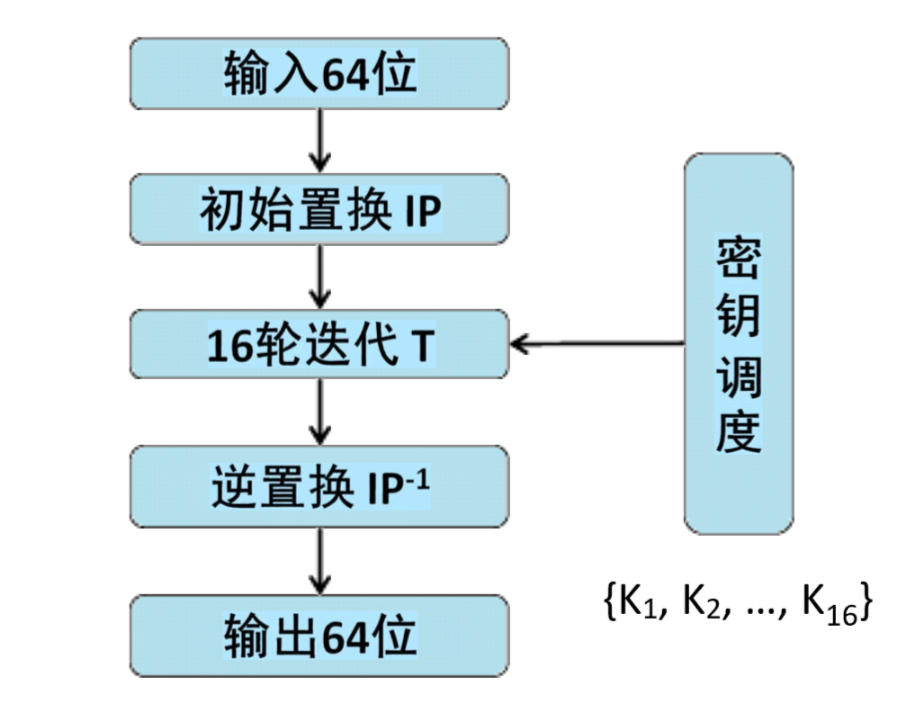
解密过程

 

**2.2 Feistel结构**

输入64位明文M 时，密钥按次序调度，是加密过程。

输入64位密文C 时，密钥按次序调度，是解密过程。



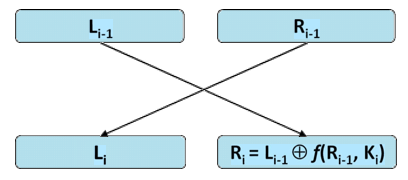
1. 输入64bit明文
2. 初始置换IP

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 和分别是R0 的前32位和后32位。下表给出IP 置换后的下标编号序列。



64bit看作线性表，用下标位置表示排列结果

1. 迭代T

根据L0R0按下述规则进行16次迭代，即

Li = Ri-1, Ri = Li-1 *f*(Ri-1, Ki), 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Ki (i = 1 .. 16)由密钥K生成

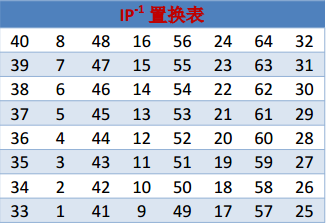
16次迭代后得到L16R16

左右交换输出R16L16

1. 逆置换IP-1

对迭代T输出的二进制串R16L16使用初始置换的逆置换IP-1得到密文C，

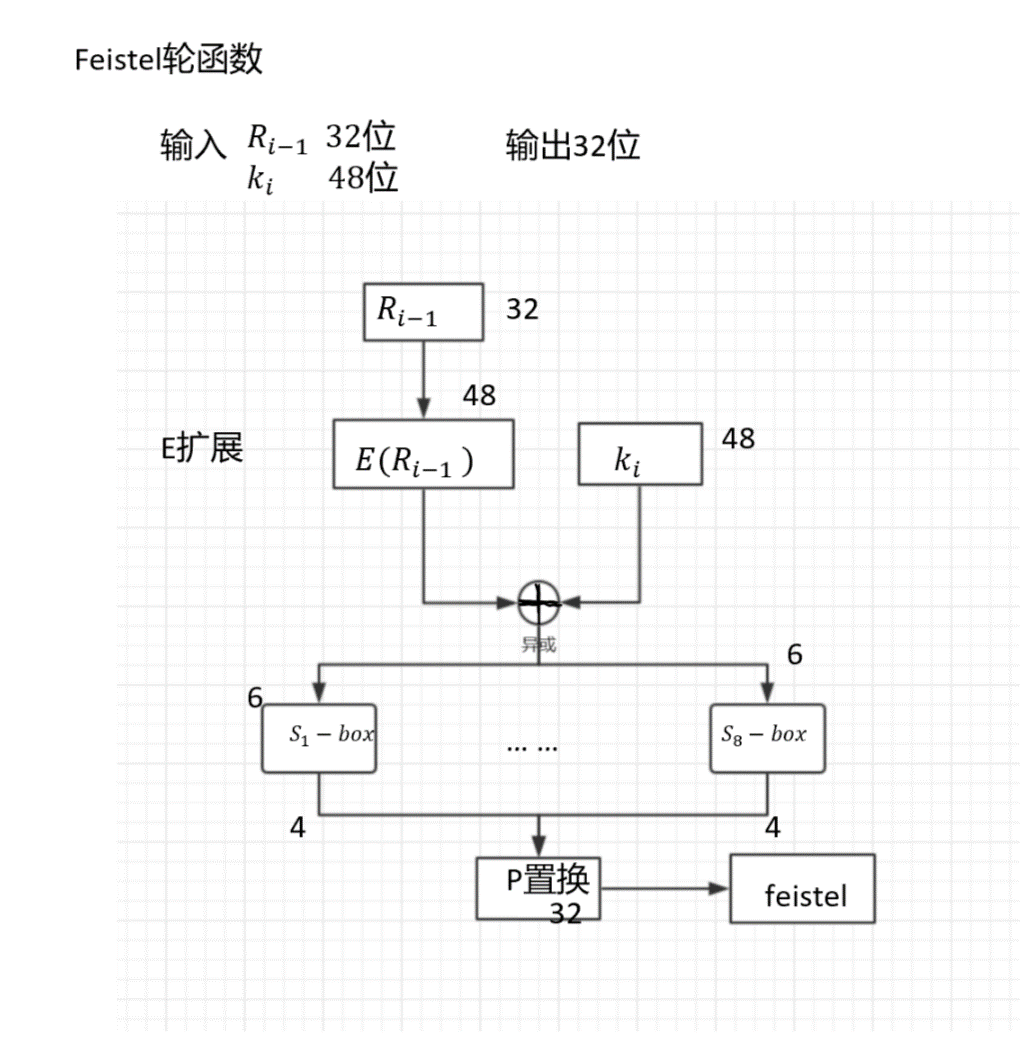
即：C = IP-1 (R16L16).





1. 输出64位密文C

注1： 迭代T中使用Feistel轮函数 f(Ri-1, Ki)详解



1. 将长度为32位的串 Ri-1作 E-扩展，成为48位的串 E(Ri-1)

E-扩展规则：



1. 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
2. 将 (2)得到的结果平均分成8个分组 *(*每个分组长度6位*)*，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

二进制6-4 转换机制：S-盒

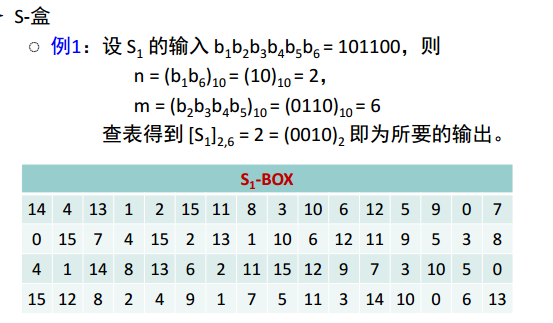
S-盒是一类选择函数，用于二进制6-4转换。Feistel 轮函数使用8个S-盒S1, …, S8，每个S-盒是一个4行(编号0-3)、16 列(编号0-15)的表，表中元素是一个4位二进制数的十进制表示，取值在0-15之间。

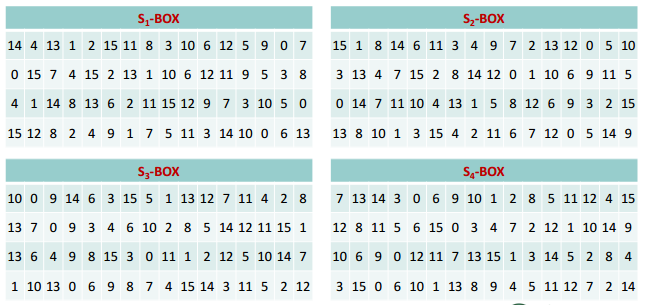
设Si 的6位输入为b1b2b3b4b5b6，则:

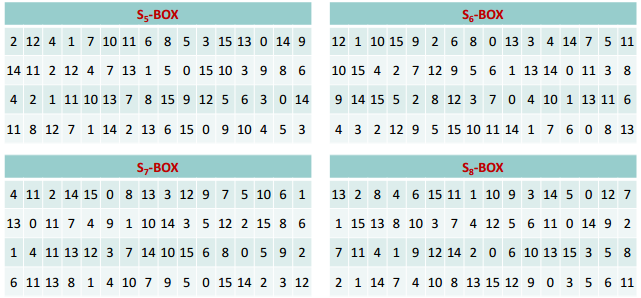
n = (b1b6)10 确定行号

m = (b2b3b4b5)10 确定列号

[Si]n,m 元素的值的二进制形式即为所要的Si的输出





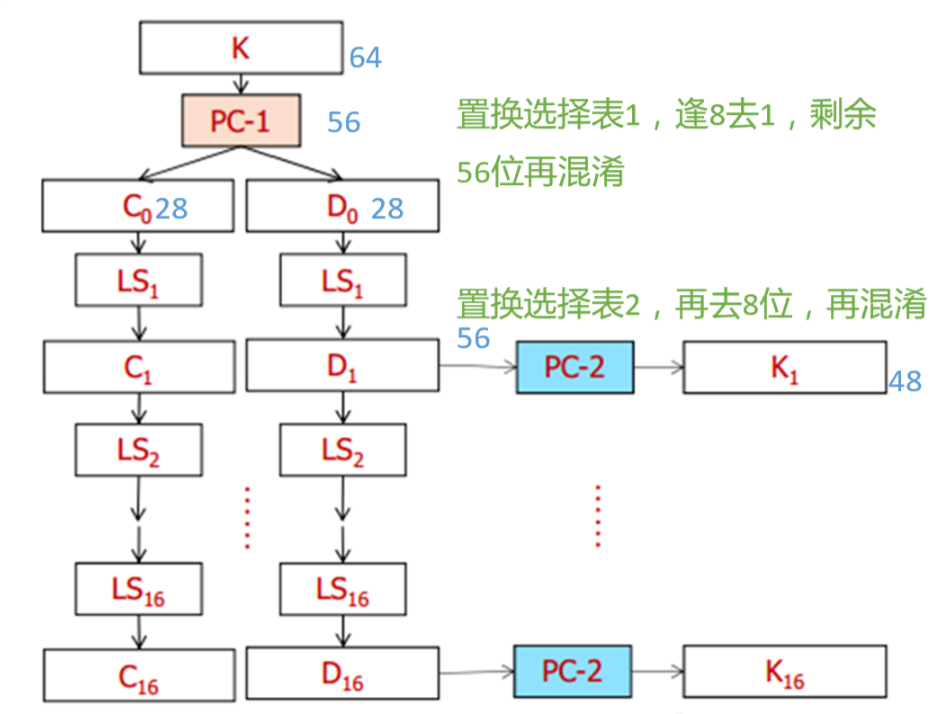


1. 将 (3) 得到的分组结果合并得到长度为32位的串
2. 将 (4) 的结果经过 P*-*置换，得到轮函数 f(Ri-1, Ki)的最终结果

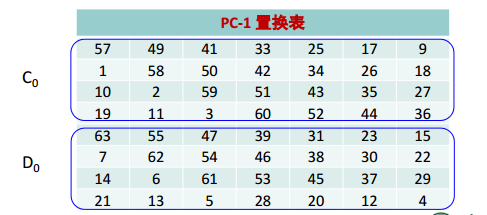


注2：迭代T中Feistel轮函数 f(Ri-1, Ki)子密钥Ki生成

子密钥生成过程根据给定的64位密钥K生成Feistel 轮函数的每轮中使用的子密钥Ki



1. 对K的56个非校验位实行置换PC-1，得到C0D0，其中C0 和D0 分别由PC-1置换后的前28位和后28位组成。i = 1

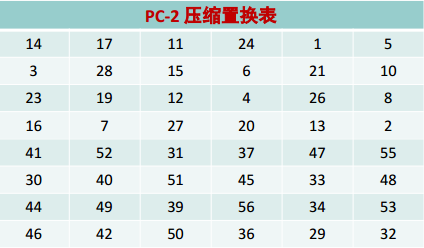


1. 计算Ci = LSi(Ci-1) 和Di = LSi(Di-1)

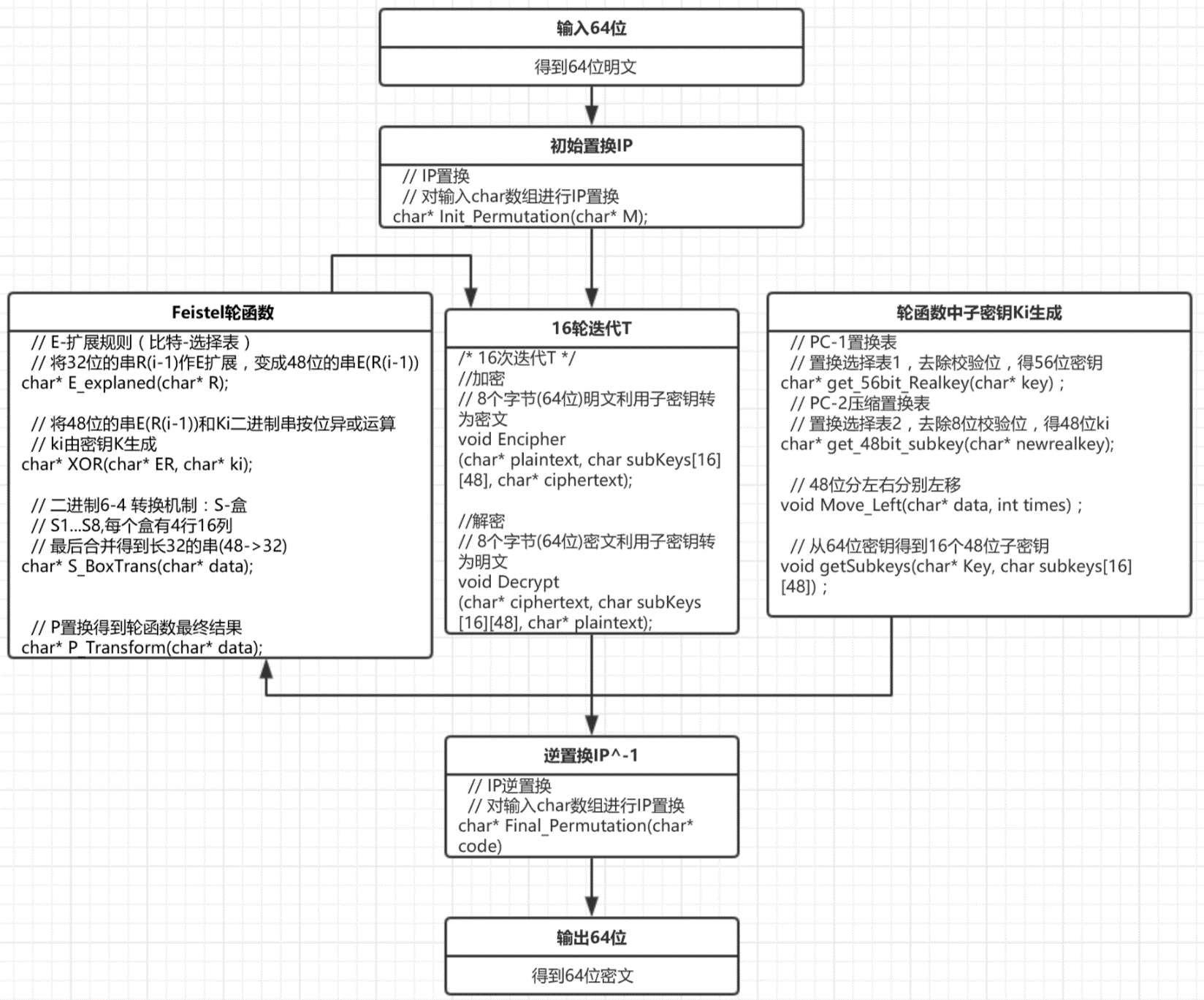
当i =1, 2, 9, 16时，LSi (A)表示将二进制串A循环左移一个位置；否则循环左移两个位置。

1. 对56位的CiDi 实行PC-2压缩置换，得到48位的Ki。i = i+1。

PC-2压缩置换：从56位的CiDi 中去掉第9, 18, 22, 25, 35, 38,43, 54位，将剩下的48位按照PC-2置换表作置换，得到Ki。



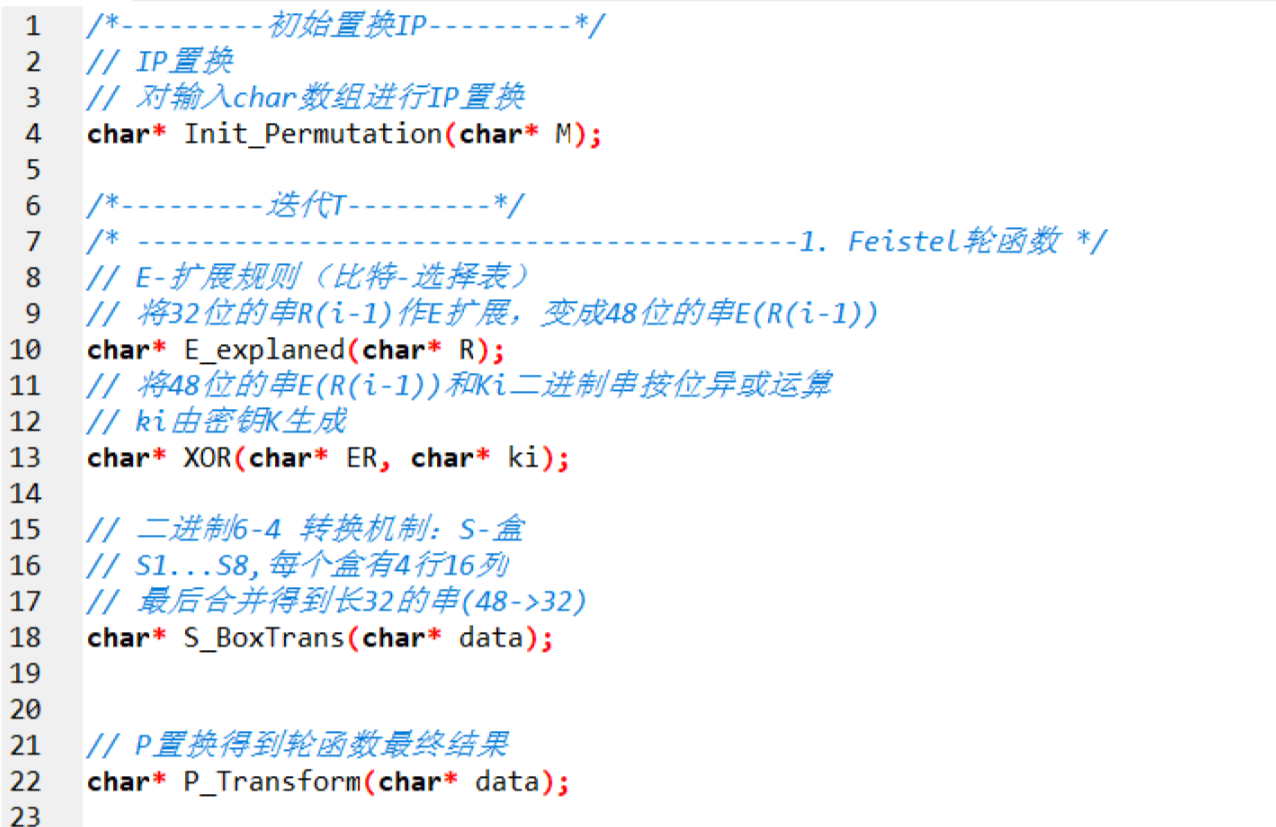
1. 如果已经得到K16，密钥调度过程结束；否则转(2)。
2. **模块分解**



划分为如图几个主要模块

模块函数定义如下：

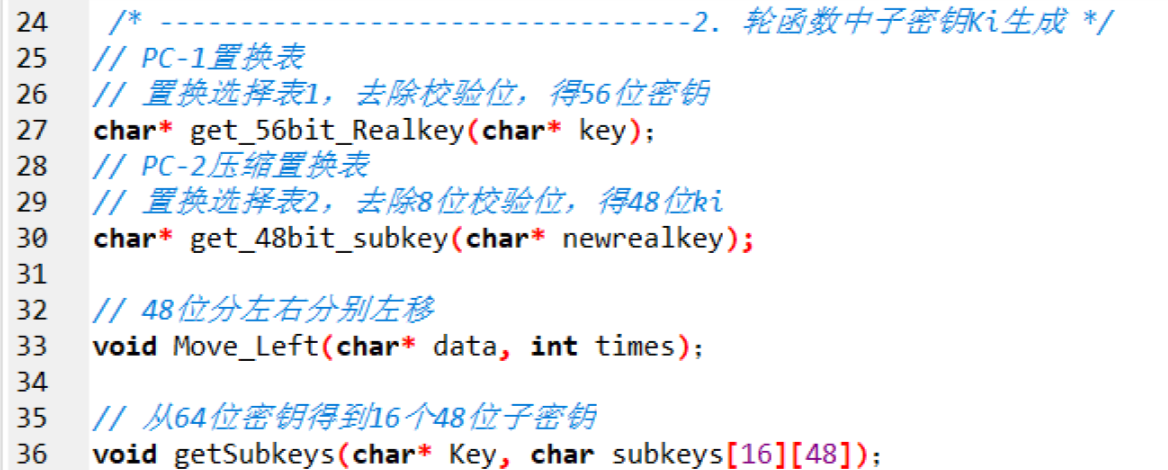
1. 初始置换IP



1. Feistel轮函数



1. 轮函数中子密钥Ki生成



1. 逆置换IP^-1

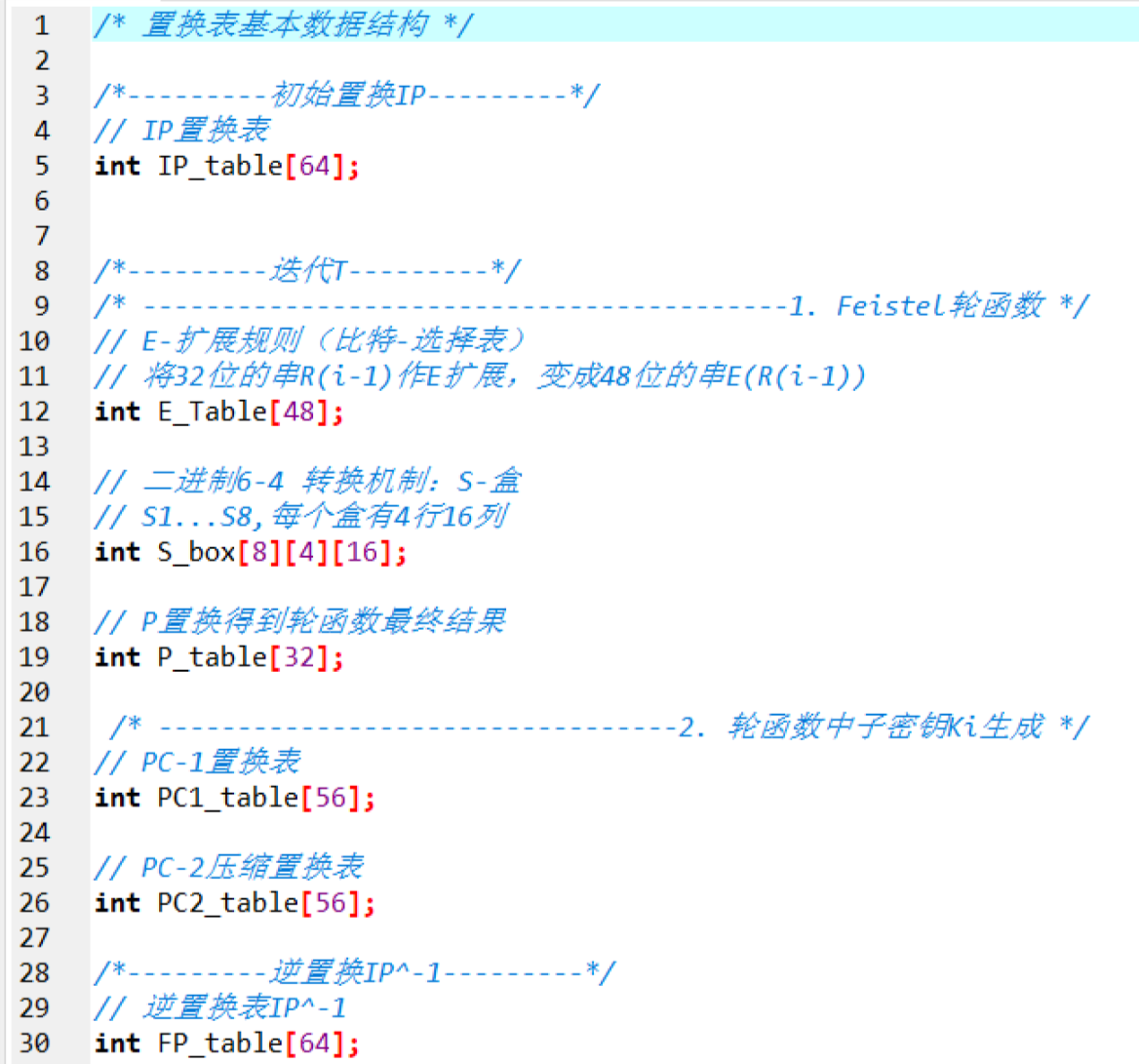


1. **数据结构**

线性表是最常用且最简单的一种数据结构，它是n个数据元素的有限序列。使用数组存储线性表的元素，即用一组连续的存储单元依次存储线性表的数据元素。

用简单的数组来存储所用到的置换表和明文密文及中间密钥等基本数据。

1. 置换表基本存储结构



1. 其他数据存储

均采用char数组存储。

1. **类C语言算法过程（为便于观看转为markdown显示再截图如下）**

计算机生成了可选文字:
[
关闭
]
•
内
容
目
录
1.
/*---------
初始置换
IP---------*/
2.
// IP
置换
3.
// 
对输入
char
数组进行
IP
置换
4.
char
*
Init_Permutation
(
char
*
 M
)
{
5.
char
*
 IP_result 
=
new
char
[
64
];
6.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
7.
        IP_result
[
i
]
=
 M
[
IP_table
[
i
]];
8.
}
9.
return
 IP_result
;
10.
}
11.
12.
13.
/*---------
迭代
T---------*/
14.
/* -----------------------------------------1. Feistel
轮函数
 */
15.
// E-
扩展规则（比特
-
选择表）
16.
// 
将
32
位的串
R(i-1)
作
E
扩展，变成
48
位的串
E(R(i-1))
17.
char
*
 E_explaned
(
char
*
 R
)
{
18.
char
*
 E_result 
=
new
char
[
48
];
19.
for
(
int
 i 
=
0
;
 i 
<
48
;
i
++)
{
20.
        E_result
[
i
]
=
 R
[
E_table
[
i
]];
21.
}
22.
return
 E_result
;
23.
}
24.
// 
将
48
位的串
E(R(i-1))
和
Ki
二进制串按位异或运算
25.
// ki
由密钥
K
生成
26.
char
*
 XOR
(
char
*
 ER
,
char
*
 ki
)
{
27.
char
*
 XOR_result 
=
new
char
[
48
];
28.
for
(
int
 i 
=
0
;
 i 
<
48
;
 i
++)
{
29.
        XOR_result
[
i
]
=
 ER
[
i
]^
ki
[
i
];
30.
}
31.
return
 XOR_result
;
32.
}
33.
34.
35.
// 
二进制
6-4 
转换机制：
S-
盒
36.
// S1...S8,
每个盒有
4
行
16
列
37.
// 
最后合并得到长
32
的串
(48->32)
38.
char
*
 S_BoxTrans
(
char
*
 data
)
{
39.
char
*
 result 
=
new
char
[
32
];
40.
for
(
int
 i 
=
0
;
 i 
<
8
;
 i
++)
{
41.
int
 nindex 
=
 i 
*
6
;
42.
int
 mindex 
=
 i 
<<
2
;
43.
// n
行
m
列
44.
// n=(b1b6)10
45.
// m=(b2b3b4b5)10
46.
int
 n 
=
(
data
[
nindex
]
<<
1
)
+
 data
[
nindex
+
5
];
47.
int
 m 
=
(
data
[
nindex
+
1
]
<<
3
)
+
(
data
[
nindex
+
2
]
<<
2
)
48.
+
(
data
[
nindex
+
3
]
<<
1
)
+
 data
[
nindex
+
4
]
；
49.
// 
从
sbox
中选取对应值
50.
int
 num 
=
 S_box
[
i
][
n
][
m
];
51.
52.
// 
转成
4
位二进制
53.
        result
[
mindex
]
=
(
num
&
0x08
)
>>
3
;
54.
        result
[
mindex
+
1
]
=
(
num
&
0x04
)
>>
2
;
55.
        result
[
mindex
+
2
]
=
(
num
&
0x02
)
>>
1
;
56.
        result
[
mindex
+
3
]
=
 num
&
0x01
;
57.
}
58.
return
 result
;
59.
}
60.
61.
// P
置换得到轮函数最终结果
62.
char
*
 P_Transform
(
char
*
 data
)
{
63.
char
*
 result 
=
new
char
[
32
];


计算机生成了可选文字:
[
关闭
]
•
内
容
目
录
1.
/*---------
初始置换
IP---------*/
2.
// IP
置换
3.
// 
对输入
char
数组进行
IP
置换
4.
char
*
Init_Permutation
(
char
*
 M
)
{
5.
char
*
 IP_result 
=
new
char
[
64
];
6.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
7.
        IP_result
[
i
]
=
 M
[
IP_table
[
i
]];
8.
}
9.
return
 IP_result
;
10.
}
11.
12.
13.
/*---------
迭代
T---------*/
14.
/* -----------------------------------------1. Feistel
轮函数
 */
15.
// E-
扩展规则（比特
-
选择表）
16.
// 
将
32
位的串
R(i-1)
作
E
扩展，变成
48
位的串
E(R(i-1))
17.
char
*
 E_explaned
(
char
*
 R
)
{
18.
char
*
 E_result 
=
new
char
[
48
];
19.
for
(
int
 i 
=
0
;
 i 
<
48
;
i
++)
{
20.
        E_result
[
i
]
=
 R
[
E_table
[
i
]];
21.
}
22.
return
 E_result
;
23.
}
24.
// 
将
48
位的串
E(R(i-1))
和
Ki
二进制串按位异或运算
25.
// ki
由密钥
K
生成
26.
char
*
 XOR
(
char
*
 ER
,
char
*
 ki
)
{
27.
char
*
 XOR_result 
=
new
char
[
48
];
28.
for
(
int
 i 
=
0
;
 i 
<
48
;
 i
++)
{
29.
        XOR_result
[
i
]
=
 ER
[
i
]^
ki
[
i
];
30.
}
31.
return
 XOR_result
;
32.
}
33.
34.
35.
// 
二进制
6-4 
转换机制：
S-
盒
36.
// S1...S8,
每个盒有
4
行
16
列
37.
// 
最后合并得到长
32
的串
(48->32)
38.
char
*
 S_BoxTrans
(
char
*
 data
)
{
39.
char
*
 result 
=
new
char
[
32
];
40.
for
(
int
 i 
=
0
;
 i 
<
8
;
 i
++)
{
41.
int
 nindex 
=
 i 
*
6
;
42.
int
 mindex 
=
 i 
<<
2
;
43.
// n
行
m
列
44.
// n=(b1b6)10
45.
// m=(b2b3b4b5)10
46.
int
 n 
=
(
data
[
nindex
]
<<
1
)
+
 data
[
nindex
+
5
];
47.
int
 m 
=
(
data
[
nindex
+
1
]
<<
3
)
+
(
data
[
nindex
+
2
]
<<
2
)
48.
+
(
data
[
nindex
+
3
]
<<
1
)
+
 data
[
nindex
+
4
]
；
49.
// 
从
sbox
中选取对应值
50.
int
 num 
=
 S_box
[
i
][
n
][
m
];
51.
52.
// 
转成
4
位二进制
53.
        result
[
mindex
]
=
(
num
&
0x08
)
>>
3
;
54.
        result
[
mindex
+
1
]
=
(
num
&
0x04
)
>>
2
;
55.
        result
[
mindex
+
2
]
=
(
num
&
0x02
)
>>
1
;
56.
        result
[
mindex
+
3
]
=
 num
&
0x01
;
57.
}
58.
return
 result
;
59.
}
60.
61.
// P
置换得到轮函数最终结果
62.
char
*
 P_Transform
(
char
*
 data
)
{
63.
char
*
 result 
=
new
char
[
32
];


计算机生成了可选文字:
[
关闭
]
•
内
容
目
录
1.
/*---------
初始置换
IP---------*/
2.
// IP
置换
3.
// 
对输入
char
数组进行
IP
置换
4.
char
*
Init_Permutation
(
char
*
 M
)
{
5.
char
*
 IP_result 
=
new
char
[
64
];
6.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
7.
        IP_result
[
i
]
=
 M
[
IP_table
[
i
]];
8.
}
9.
return
 IP_result
;
10.
}
11.
12.
13.
/*---------
迭代
T---------*/
14.
/* -----------------------------------------1. Feistel
轮函数
 */
15.
// E-
扩展规则（比特
-
选择表）
16.
// 
将
32
位的串
R(i-1)
作
E
扩展，变成
48
位的串
E(R(i-1))
17.
char
*
 E_explaned
(
char
*
 R
)
{
18.
char
*
 E_result 
=
new
char
[
48
];
19.
for
(
int
 i 
=
0
;
 i 
<
48
;
i
++)
{
20.
        E_result
[
i
]
=
 R
[
E_table
[
i
]];
21.
}
22.
return
 E_result
;
23.
}
24.
// 
将
48
位的串
E(R(i-1))
和
Ki
二进制串按位异或运算
25.
// ki
由密钥
K
生成
26.
char
*
 XOR
(
char
*
 ER
,
char
*
 ki
)
{
27.
char
*
 XOR_result 
=
new
char
[
48
];
28.
for
(
int
 i 
=
0
;
 i 
<
48
;
 i
++)
{
29.
        XOR_result
[
i
]
=
 ER
[
i
]^
ki
[
i
];
30.
}
31.
return
 XOR_result
;
32.
}
33.
34.
35.
// 
二进制
6-4 
转换机制：
S-
盒
36.
// S1...S8,
每个盒有
4
行
16
列
37.
// 
最后合并得到长
32
的串
(48->32)
38.
char
*
 S_BoxTrans
(
char
*
 data
)
{
39.
char
*
 result 
=
new
char
[
32
];
40.
for
(
int
 i 
=
0
;
 i 
<
8
;
 i
++)
{
41.
int
 nindex 
=
 i 
*
6
;
42.
int
 mindex 
=
 i 
<<
2
;
43.
// n
行
m
列
44.
// n=(b1b6)10
45.
// m=(b2b3b4b5)10
46.
int
 n 
=
(
data
[
nindex
]
<<
1
)
+
 data
[
nindex
+
5
];
47.
int
 m 
=
(
data
[
nindex
+
1
]
<<
3
)
+
(
data
[
nindex
+
2
]
<<
2
)
48.
+
(
data
[
nindex
+
3
]
<<
1
)
+
 data
[
nindex
+
4
]
；
49.
// 
从
sbox
中选取对应值
50.
int
 num 
=
 S_box
[
i
][
n
][
m
];
51.
52.
// 
转成
4
位二进制
53.
        result
[
mindex
]
=
(
num
&
0x08
)
>>
3
;
54.
        result
[
mindex
+
1
]
=
(
num
&
0x04
)
>>
2
;
55.
        result
[
mindex
+
2
]
=
(
num
&
0x02
)
>>
1
;
56.
        result
[
mindex
+
3
]
=
 num
&
0x01
;
57.
}
58.
return
 result
;
59.
}
60.
61.
// P
置换得到轮函数最终结果
62.
char
*
 P_Transform
(
char
*
 data
)
{
63.
char
*
 result 
=
new
char
[
32
];


计算机生成了可选文字:
64.
for
(
int
 i 
=
0
;
 i 
<
32
;
 i
++)
{
65.
        result
[
i
]
=
 data
[
P_table
[
i
]];
66.
}
67.
return
 result
;
68.
}
69.
70.
71.
72.
73.
74.
// 
置换选择表
1
，去除
8
位校验位，得
56
位密钥
75.
char
*
 get_56bit_Realkey
(
char
*
 key
)
{
76.
char
*
 realkey 
=
new
char
[
56
];
77.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
78.
// 
逢
8
去
1
79.
if
((
i
+
1
)%
8
!=
0
)
{
80.
            realkey 
=
 key
[
PC1_table
[
i
]];
81.
}
82.
}
83.
return
 realkey
;
84.
}
85.
86.
// 
置换选择表
2
，去除
8
位校验位，得
48
位
ki
87.
char
*
 get_48bit_subkey
(
char
*
 newrealkey
)
{
88.
char
*
 subkey 
=
new
char
[
48
];
89.
for
(
int
 i 
=
0
;
 i 
<
56
;
 i
++)
{
90.
// 
去除
9
，
18
，
22
，
25
，
35
，
38
，
43
，
54
位
91.
if
(
i 
!=
9
&&
 i 
!=
18
&&
 i 
!=
22
&&
 i 
!=
25
92.
&&
 i 
!=
35
&&
 i 
!=
38
&&
 i 
!=
43
&&
 i 
!=
54
)
{
93.
            subkey 
=
 newrealkey
[
PC2_table
[
i
]];
94.
}
95.
}
96.
return
 subkey
;
97.
}
98.
99.
100.
// i=1,2,9,16
时，循环左移一个位置，否则循环左移两个位置
101.
int
 movetoleft
[
16
]
=
{
1
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
};
102.
// 48
位分左右分别左移
103.
void
Move_Left
(
char
*
 data
,
int
 times
)
{
104.
char
*
 savedata 
=
new
char
[
56
];
105.
// 
保存移位数据
106.
    memcpy
(
savedata
,
 data
,
 times
);
107.
    memcpy
(
savedata
+
times
,
 data
+
28
,
 times
);
108.
109.
// 
前
28
位
110.
    memcpy
(
data
,
 data
+
times
,
28
-
times
);
111.
    memcpy
(
data
+
28
-
times
,
 savedata
,
 times
);
112.
// 
后
28
位
113.
    memcpy
(
data
+
28
,
 data
+
28
+
times
,
28
-
times
);
114.
    memcpy
(
data
+
56
-
times
,
 savedata
+
times
,
 times
);
115.
116.
117.
}
118.
119.
120.
// 
从
64
位密钥得到
16
个子密钥
121.
void
 getSubkeys
(
char
*
Key
,
char
 subkeys
[
16
][
48
])
{
122.
char
*
 realkey 
=
new
char
[
56
];
123.
// PC-1
置换
124.
    realkey 
=
 get_56bit_Realkey
(
key
);
125.
// 16
次循环迭代
126.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++)
{
127.
// 
左移
movetoleft[i]
位
128.
Move_Left
(
realkey
,
 movetoleft
[
i
]);
129.
// // PC-2
置换，得到子密钥
130.
        subkeys
[
i
]
=
 get_48bit_subkey
(
realkey
);
131.
}
132.
}
133.
134.
135.
// IP
逆置换
136.
// 
对输入
char
数组进行
IP
置换
137.
char
*
Final_Permutation
(
char
*
 code
)
{
138.
char
*
 FP_result 
=
new
char
[
64
];
139.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
140.
        FP_result
[
i
]
=
 code
[
FP_table
[
i
]];
141.
}
计算机生成了可选文字:
64.
for
(
int
 i 
=
0
;
 i 
<
32
;
 i
++)
{
65.
        result
[
i
]
=
 data
[
P_table
[
i
]];
66.
}
67.
return
 result
;
68.
}
69.
70.
71.
72.
73.
74.
// 
置换选择表
1
，去除
8
位校验位，得
56
位密钥
75.
char
*
 get_56bit_Realkey
(
char
*
 key
)
{
76.
char
*
 realkey 
=
new
char
[
56
];
77.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
78.
// 
逢
8
去
1
79.
if
((
i
+
1
)%
8
!=
0
)
{
80.
            realkey 
=
 key
[
PC1_table
[
i
]];
81.
}
82.
}
83.
return
 realkey
;
84.
}
85.
86.
// 
置换选择表
2
，去除
8
位校验位，得
48
位
ki
87.
char
*
 get_48bit_subkey
(
char
*
 newrealkey
)
{
88.
char
*
 subkey 
=
new
char
[
48
];
89.
for
(
int
 i 
=
0
;
 i 
<
56
;
 i
++)
{
90.
// 
去除
9
，
18
，
22
，
25
，
35
，
38
，
43
，
54
位
91.
if
(
i 
!=
9
&&
 i 
!=
18
&&
 i 
!=
22
&&
 i 
!=
25
92.
&&
 i 
!=
35
&&
 i 
!=
38
&&
 i 
!=
43
&&
 i 
!=
54
)
{
93.
            subkey 
=
 newrealkey
[
PC2_table
[
i
]];
94.
}
95.
}
96.
return
 subkey
;
97.
}
98.
99.
100.
// i=1,2,9,16
时，循环左移一个位置，否则循环左移两个位置
101.
int
 movetoleft
[
16
]
=
{
1
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
};
102.
// 48
位分左右分别左移
103.
void
Move_Left
(
char
*
 data
,
int
 times
)
{
104.
char
*
 savedata 
=
new
char
[
56
];
105.
// 
保存移位数据
106.
    memcpy
(
savedata
,
 data
,
 times
);
107.
    memcpy
(
savedata
+
times
,
 data
+
28
,
 times
);
108.
109.
// 
前
28
位
110.
    memcpy
(
data
,
 data
+
times
,
28
-
times
);
111.
    memcpy
(
data
+
28
-
times
,
 savedata
,
 times
);
112.
// 
后
28
位
113.
    memcpy
(
data
+
28
,
 data
+
28
+
times
,
28
-
times
);
114.
    memcpy
(
data
+
56
-
times
,
 savedata
+
times
,
 times
);
115.
116.
117.
}
118.
119.
120.
// 
从
64
位密钥得到
16
个子密钥
121.
void
 getSubkeys
(
char
*
Key
,
char
 subkeys
[
16
][
48
])
{
122.
char
*
 realkey 
=
new
char
[
56
];
123.
// PC-1
置换
124.
    realkey 
=
 get_56bit_Realkey
(
key
);
125.
// 16
次循环迭代
126.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++)
{
127.
// 
左移
movetoleft[i]
位
128.
Move_Left
(
realkey
,
 movetoleft
[
i
]);
129.
// // PC-2
置换，得到子密钥
130.
        subkeys
[
i
]
=
 get_48bit_subkey
(
realkey
);
131.
}
132.
}
133.
134.
135.
// IP
逆置换
136.
// 
对输入
char
数组进行
IP
置换
137.
char
*
Final_Permutation
(
char
*
 code
)
{
138.
char
*
 FP_result 
=
new
char
[
64
];
139.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
140.
        FP_result
[
i
]
=
 code
[
FP_table
[
i
]];
141.
}
计算机生成了可选文字:
64.
for
(
int
 i 
=
0
;
 i 
<
32
;
 i
++)
{
65.
        result
[
i
]
=
 data
[
P_table
[
i
]];
66.
}
67.
return
 result
;
68.
}
69.
70.
71.
72.
73.
74.
// 
置换选择表
1
，去除
8
位校验位，得
56
位密钥
75.
char
*
 get_56bit_Realkey
(
char
*
 key
)
{
76.
char
*
 realkey 
=
new
char
[
56
];
77.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
78.
// 
逢
8
去
1
79.
if
((
i
+
1
)%
8
!=
0
)
{
80.
            realkey 
=
 key
[
PC1_table
[
i
]];
81.
}
82.
}
83.
return
 realkey
;
84.
}
85.
86.
// 
置换选择表
2
，去除
8
位校验位，得
48
位
ki
87.
char
*
 get_48bit_subkey
(
char
*
 newrealkey
)
{
88.
char
*
 subkey 
=
new
char
[
48
];
89.
for
(
int
 i 
=
0
;
 i 
<
56
;
 i
++)
{
90.
// 
去除
9
，
18
，
22
，
25
，
35
，
38
，
43
，
54
位
91.
if
(
i 
!=
9
&&
 i 
!=
18
&&
 i 
!=
22
&&
 i 
!=
25
92.
&&
 i 
!=
35
&&
 i 
!=
38
&&
 i 
!=
43
&&
 i 
!=
54
)
{
93.
            subkey 
=
 newrealkey
[
PC2_table
[
i
]];
94.
}
95.
}
96.
return
 subkey
;
97.
}
98.
99.
100.
// i=1,2,9,16
时，循环左移一个位置，否则循环左移两个位置
101.
int
 movetoleft
[
16
]
=
{
1
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
};
102.
// 48
位分左右分别左移
103.
void
Move_Left
(
char
*
 data
,
int
 times
)
{
104.
char
*
 savedata 
=
new
char
[
56
];
105.
// 
保存移位数据
106.
    memcpy
(
savedata
,
 data
,
 times
);
107.
    memcpy
(
savedata
+
times
,
 data
+
28
,
 times
);
108.
109.
// 
前
28
位
110.
    memcpy
(
data
,
 data
+
times
,
28
-
times
);
111.
    memcpy
(
data
+
28
-
times
,
 savedata
,
 times
);
112.
// 
后
28
位
113.
    memcpy
(
data
+
28
,
 data
+
28
+
times
,
28
-
times
);
114.
    memcpy
(
data
+
56
-
times
,
 savedata
+
times
,
 times
);
115.
116.
117.
}
118.
119.
120.
// 
从
64
位密钥得到
16
个子密钥
121.
void
 getSubkeys
(
char
*
Key
,
char
 subkeys
[
16
][
48
])
{
122.
char
*
 realkey 
=
new
char
[
56
];
123.
// PC-1
置换
124.
    realkey 
=
 get_56bit_Realkey
(
key
);
125.
// 16
次循环迭代
126.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++)
{
127.
// 
左移
movetoleft[i]
位
128.
Move_Left
(
realkey
,
 movetoleft
[
i
]);
129.
// // PC-2
置换，得到子密钥
130.
        subkeys
[
i
]
=
 get_48bit_subkey
(
realkey
);
131.
}
132.
}
133.
134.
135.
// IP
逆置换
136.
// 
对输入
char
数组进行
IP
置换
137.
char
*
Final_Permutation
(
char
*
 code
)
{
138.
char
*
 FP_result 
=
new
char
[
64
];
139.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
140.
        FP_result
[
i
]
=
 code
[
FP_table
[
i
]];
141.
}


计算机生成了可选文字:
142.
return
 FP_result
;
143.
}
144.
145.
146.
char
*
 charTobit
(
char
*
 text
);
147.
char
*
 bitTochar
(
char
*
 bits
);
148.
//
加密
149.
// 8
个字节
(64
位
)
明文利用子密钥转为密文
150.
void
Encipher
(
char
*
 plaintext
,
char
 subKeys
[
16
][
48
],
char
*
 ciphertext
)
{
151.
char
*
 plain 
=
new
char
[
64
];
152.
char
*
 afterIP 
=
new
char
[
64
];
153.
char
*
Left
=
new
char
[
48
];
154.
char
*
Right
=
new
char
[
48
];
155.
char
*
 eRight 
=
new
char
[
48
];
156.
char
*
 xor_result 
=
new
char
[
48
];
157.
char
*
 s_result 
=
new
char
[
32
];
158.
char
*
 f_result 
=
new
char
[
32
];
159.
char
*
Ri
=
new
char
[
32
];
160.
char
*
 afterFP 
=
new
char
[
64
];
161.
162.
    plain 
=
 charTobit
(
plaintext
);
163.
//
初始
IP  
164.
    afterIP 
=
Init_Permutation
(
plain
);
165.
166.
//16
轮迭代
167.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++){
168.
// 
左半部分
169.
        memcpy
(
Left
,
 afterIP
,
32
);
170.
// 
右半部分
171.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
172.
//
右半部分
E
扩展置换，
32
位
->48
位
173.
        eRight 
=
 E_explaned
(
Right
);
174.
//
将右半部分与子密钥进行异或操作
175.
        xor_result 
=
 XOR
(
eRight
,
 subkeys
[
i
]);
176.
//
异或结果进入
Sbox
，输出
32
位结果
177.
        s_result 
=
 S_BoxTrans
(
xor_result
);
178.
//P
置换得到轮函数最终结果
179.
        f_result 
=
 P_Transform
()
180.
//
明文左半部分与轮函数结果进行异或
181.
Ri
=
 XOR
(
Left
,
f_result
);
182.
// Li
和
Ri
交换
183.
if
(
i 
<
15
){
184.
Swap
(
Right
,
Ri
);
185.
}
186.
        memcpy
(
afterIP
,
Right
,
32
);
187.
        memcpy
(
afterIP
+
32
,
Ri
,
32
);
188.
}
189.
//
逆初始置换（
IP^-1
置换）
190.
    afterFP 
=
Final_Permutation
(
afterIP
);
191.
    ciphertext 
=
 bitTochar
(
afterFP
);
192.
}
193.
194.
//
解密
195.
// 8
个字节
(64
位
)
密文利用子密钥转为明文
196.
void
Decrypt
(
char
*
 ciphertext
,
char
 subKeys
[
16
][
48
],
char
*
 plaintext
)
{
197.
char
*
 cipher 
=
new
char
[
64
];
198.
char
*
 afterIP 
=
new
char
[
64
];
199.
char
*
Left
=
new
char
[
48
];
200.
char
*
Right
=
new
char
[
48
];
201.
char
*
 eRight 
=
new
char
[
48
];
202.
char
*
 xor_result 
=
new
char
[
48
];
203.
char
*
 s_result 
=
new
char
[
32
];
204.
char
*
 f_result 
=
new
char
[
32
];
205.
char
*
Ri
=
new
char
[
32
];
206.
char
*
 afterFP 
=
new
char
[
64
];
207.
208.
    cipher 
=
 charTobit
(
ciphertext
);
209.
//
初始
IP  
210.
    afterIP 
=
Init_Permutation
(
cipher
);
211.
212.
//16
轮迭代
213.
for
(
int
 i 
=
15
;
 i 
>=
0
;
 i
--){
214.
// 
左半部分
215.
        memcpy
(
Left
,
 afterIP
,
32
);
216.
// 
右半部分
217.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
218.
//
右半部分
E
扩展置换，
32
位
->48
位
219.
        eRight 
=
 E_explaned
(
Right
);
计算机生成了可选文字:
64.
for
(
int
 i 
=
0
;
 i 
<
32
;
 i
++)
{
65.
        result
[
i
]
=
 data
[
P_table
[
i
]];
66.
}
67.
return
 result
;
68.
}
69.
70.
71.
72.
73.
74.
// 
置换选择表
1
，去除
8
位校验位，得
56
位密钥
75.
char
*
 get_56bit_Realkey
(
char
*
 key
)
{
76.
char
*
 realkey 
=
new
char
[
56
];
77.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
78.
// 
逢
8
去
1
79.
if
((
i
+
1
)%
8
!=
0
)
{
80.
            realkey 
=
 key
[
PC1_table
[
i
]];
81.
}
82.
}
83.
return
 realkey
;
84.
}
85.
86.
// 
置换选择表
2
，去除
8
位校验位，得
48
位
ki
87.
char
*
 get_48bit_subkey
(
char
*
 newrealkey
)
{
88.
char
*
 subkey 
=
new
char
[
48
];
89.
for
(
int
 i 
=
0
;
 i 
<
56
;
 i
++)
{
90.
// 
去除
9
，
18
，
22
，
25
，
35
，
38
，
43
，
54
位
91.
if
(
i 
!=
9
&&
 i 
!=
18
&&
 i 
!=
22
&&
 i 
!=
25
92.
&&
 i 
!=
35
&&
 i 
!=
38
&&
 i 
!=
43
&&
 i 
!=
54
)
{
93.
            subkey 
=
 newrealkey
[
PC2_table
[
i
]];
94.
}
95.
}
96.
return
 subkey
;
97.
}
98.
99.
100.
// i=1,2,9,16
时，循环左移一个位置，否则循环左移两个位置
101.
int
 movetoleft
[
16
]
=
{
1
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
,
2
,
2
,
2
,
2
,
2
,
2
,
1
};
102.
// 48
位分左右分别左移
103.
void
Move_Left
(
char
*
 data
,
int
 times
)
{
104.
char
*
 savedata 
=
new
char
[
56
];
105.
// 
保存移位数据
106.
    memcpy
(
savedata
,
 data
,
 times
);
107.
    memcpy
(
savedata
+
times
,
 data
+
28
,
 times
);
108.
109.
// 
前
28
位
110.
    memcpy
(
data
,
 data
+
times
,
28
-
times
);
111.
    memcpy
(
data
+
28
-
times
,
 savedata
,
 times
);
112.
// 
后
28
位
113.
    memcpy
(
data
+
28
,
 data
+
28
+
times
,
28
-
times
);
114.
    memcpy
(
data
+
56
-
times
,
 savedata
+
times
,
 times
);
115.
116.
117.
}
118.
119.
120.
// 
从
64
位密钥得到
16
个子密钥
121.
void
 getSubkeys
(
char
*
Key
,
char
 subkeys
[
16
][
48
])
{
122.
char
*
 realkey 
=
new
char
[
56
];
123.
// PC-1
置换
124.
    realkey 
=
 get_56bit_Realkey
(
key
);
125.
// 16
次循环迭代
126.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++)
{
127.
// 
左移
movetoleft[i]
位
128.
Move_Left
(
realkey
,
 movetoleft
[
i
]);
129.
// // PC-2
置换，得到子密钥
130.
        subkeys
[
i
]
=
 get_48bit_subkey
(
realkey
);
131.
}
132.
}
133.
134.
135.
// IP
逆置换
136.
// 
对输入
char
数组进行
IP
置换
137.
char
*
Final_Permutation
(
char
*
 code
)
{
138.
char
*
 FP_result 
=
new
char
[
64
];
139.
for
(
int
 i 
=
0
;
 i 
<
64
;
 i
++)
{
140.
        FP_result
[
i
]
=
 code
[
FP_table
[
i
]];
141.
}


计算机生成了可选文字:
142.
return
 FP_result
;
143.
}
144.
145.
146.
char
*
 charTobit
(
char
*
 text
);
147.
char
*
 bitTochar
(
char
*
 bits
);
148.
//
加密
149.
// 8
个字节
(64
位
)
明文利用子密钥转为密文
150.
void
Encipher
(
char
*
 plaintext
,
char
 subKeys
[
16
][
48
],
char
*
 ciphertext
)
{
151.
char
*
 plain 
=
new
char
[
64
];
152.
char
*
 afterIP 
=
new
char
[
64
];
153.
char
*
Left
=
new
char
[
48
];
154.
char
*
Right
=
new
char
[
48
];
155.
char
*
 eRight 
=
new
char
[
48
];
156.
char
*
 xor_result 
=
new
char
[
48
];
157.
char
*
 s_result 
=
new
char
[
32
];
158.
char
*
 f_result 
=
new
char
[
32
];
159.
char
*
Ri
=
new
char
[
32
];
160.
char
*
 afterFP 
=
new
char
[
64
];
161.
162.
    plain 
=
 charTobit
(
plaintext
);
163.
//
初始
IP  
164.
    afterIP 
=
Init_Permutation
(
plain
);
165.
166.
//16
轮迭代
167.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++){
168.
// 
左半部分
169.
        memcpy
(
Left
,
 afterIP
,
32
);
170.
// 
右半部分
171.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
172.
//
右半部分
E
扩展置换，
32
位
->48
位
173.
        eRight 
=
 E_explaned
(
Right
);
174.
//
将右半部分与子密钥进行异或操作
175.
        xor_result 
=
 XOR
(
eRight
,
 subkeys
[
i
]);
176.
//
异或结果进入
Sbox
，输出
32
位结果
177.
        s_result 
=
 S_BoxTrans
(
xor_result
);
178.
//P
置换得到轮函数最终结果
179.
        f_result 
=
 P_Transform
()
180.
//
明文左半部分与轮函数结果进行异或
181.
Ri
=
 XOR
(
Left
,
f_result
);
182.
// Li
和
Ri
交换
183.
if
(
i 
<
15
){
184.
Swap
(
Right
,
Ri
);
185.
}
186.
        memcpy
(
afterIP
,
Right
,
32
);
187.
        memcpy
(
afterIP
+
32
,
Ri
,
32
);
188.
}
189.
//
逆初始置换（
IP^-1
置换）
190.
    afterFP 
=
Final_Permutation
(
afterIP
);
191.
    ciphertext 
=
 bitTochar
(
afterFP
);
192.
}
193.
194.
//
解密
195.
// 8
个字节
(64
位
)
密文利用子密钥转为明文
196.
void
Decrypt
(
char
*
 ciphertext
,
char
 subKeys
[
16
][
48
],
char
*
 plaintext
)
{
197.
char
*
 cipher 
=
new
char
[
64
];
198.
char
*
 afterIP 
=
new
char
[
64
];
199.
char
*
Left
=
new
char
[
48
];
200.
char
*
Right
=
new
char
[
48
];
201.
char
*
 eRight 
=
new
char
[
48
];
202.
char
*
 xor_result 
=
new
char
[
48
];
203.
char
*
 s_result 
=
new
char
[
32
];
204.
char
*
 f_result 
=
new
char
[
32
];
205.
char
*
Ri
=
new
char
[
32
];
206.
char
*
 afterFP 
=
new
char
[
64
];
207.
208.
    cipher 
=
 charTobit
(
ciphertext
);
209.
//
初始
IP  
210.
    afterIP 
=
Init_Permutation
(
cipher
);
211.
212.
//16
轮迭代
213.
for
(
int
 i 
=
15
;
 i 
>=
0
;
 i
--){
214.
// 
左半部分
215.
        memcpy
(
Left
,
 afterIP
,
32
);
216.
// 
右半部分
217.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
218.
//
右半部分
E
扩展置换，
32
位
->48
位
219.
        eRight 
=
 E_explaned
(
Right
);


计算机生成了可选文字:
142.
return
 FP_result
;
143.
}
144.
145.
146.
char
*
 charTobit
(
char
*
 text
);
147.
char
*
 bitTochar
(
char
*
 bits
);
148.
//
加密
149.
// 8
个字节
(64
位
)
明文利用子密钥转为密文
150.
void
Encipher
(
char
*
 plaintext
,
char
 subKeys
[
16
][
48
],
char
*
 ciphertext
)
{
151.
char
*
 plain 
=
new
char
[
64
];
152.
char
*
 afterIP 
=
new
char
[
64
];
153.
char
*
Left
=
new
char
[
48
];
154.
char
*
Right
=
new
char
[
48
];
155.
char
*
 eRight 
=
new
char
[
48
];
156.
char
*
 xor_result 
=
new
char
[
48
];
157.
char
*
 s_result 
=
new
char
[
32
];
158.
char
*
 f_result 
=
new
char
[
32
];
159.
char
*
Ri
=
new
char
[
32
];
160.
char
*
 afterFP 
=
new
char
[
64
];
161.
162.
    plain 
=
 charTobit
(
plaintext
);
163.
//
初始
IP  
164.
    afterIP 
=
Init_Permutation
(
plain
);
165.
166.
//16
轮迭代
167.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++){
168.
// 
左半部分
169.
        memcpy
(
Left
,
 afterIP
,
32
);
170.
// 
右半部分
171.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
172.
//
右半部分
E
扩展置换，
32
位
->48
位
173.
        eRight 
=
 E_explaned
(
Right
);
174.
//
将右半部分与子密钥进行异或操作
175.
        xor_result 
=
 XOR
(
eRight
,
 subkeys
[
i
]);
176.
//
异或结果进入
Sbox
，输出
32
位结果
177.
        s_result 
=
 S_BoxTrans
(
xor_result
);
178.
//P
置换得到轮函数最终结果
179.
        f_result 
=
 P_Transform
()
180.
//
明文左半部分与轮函数结果进行异或
181.
Ri
=
 XOR
(
Left
,
f_result
);
182.
// Li
和
Ri
交换
183.
if
(
i 
<
15
){
184.
Swap
(
Right
,
Ri
);
185.
}
186.
        memcpy
(
afterIP
,
Right
,
32
);
187.
        memcpy
(
afterIP
+
32
,
Ri
,
32
);
188.
}
189.
//
逆初始置换（
IP^-1
置换）
190.
    afterFP 
=
Final_Permutation
(
afterIP
);
191.
    ciphertext 
=
 bitTochar
(
afterFP
);
192.
}
193.
194.
//
解密
195.
// 8
个字节
(64
位
)
密文利用子密钥转为明文
196.
void
Decrypt
(
char
*
 ciphertext
,
char
 subKeys
[
16
][
48
],
char
*
 plaintext
)
{
197.
char
*
 cipher 
=
new
char
[
64
];
198.
char
*
 afterIP 
=
new
char
[
64
];
199.
char
*
Left
=
new
char
[
48
];
200.
char
*
Right
=
new
char
[
48
];
201.
char
*
 eRight 
=
new
char
[
48
];
202.
char
*
 xor_result 
=
new
char
[
48
];
203.
char
*
 s_result 
=
new
char
[
32
];
204.
char
*
 f_result 
=
new
char
[
32
];
205.
char
*
Ri
=
new
char
[
32
];
206.
char
*
 afterFP 
=
new
char
[
64
];
207.
208.
    cipher 
=
 charTobit
(
ciphertext
);
209.
//
初始
IP  
210.
    afterIP 
=
Init_Permutation
(
cipher
);
211.
212.
//16
轮迭代
213.
for
(
int
 i 
=
15
;
 i 
>=
0
;
 i
--){
214.
// 
左半部分
215.
        memcpy
(
Left
,
 afterIP
,
32
);
216.
// 
右半部分
217.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
218.
//
右半部分
E
扩展置换，
32
位
->48
位
219.
        eRight 
=
 E_explaned
(
Right
);
计算机生成了可选文字:
142.
return
 FP_result
;
143.
}
144.
145.
146.
char
*
 charTobit
(
char
*
 text
);
147.
char
*
 bitTochar
(
char
*
 bits
);
148.
//
加密
149.
// 8
个字节
(64
位
)
明文利用子密钥转为密文
150.
void
Encipher
(
char
*
 plaintext
,
char
 subKeys
[
16
][
48
],
char
*
 ciphertext
)
{
151.
char
*
 plain 
=
new
char
[
64
];
152.
char
*
 afterIP 
=
new
char
[
64
];
153.
char
*
Left
=
new
char
[
48
];
154.
char
*
Right
=
new
char
[
48
];
155.
char
*
 eRight 
=
new
char
[
48
];
156.
char
*
 xor_result 
=
new
char
[
48
];
157.
char
*
 s_result 
=
new
char
[
32
];
158.
char
*
 f_result 
=
new
char
[
32
];
159.
char
*
Ri
=
new
char
[
32
];
160.
char
*
 afterFP 
=
new
char
[
64
];
161.
162.
    plain 
=
 charTobit
(
plaintext
);
163.
//
初始
IP  
164.
    afterIP 
=
Init_Permutation
(
plain
);
165.
166.
//16
轮迭代
167.
for
(
int
 i 
=
0
;
 i 
<
16
;
 i
++){
168.
// 
左半部分
169.
        memcpy
(
Left
,
 afterIP
,
32
);
170.
// 
右半部分
171.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
172.
//
右半部分
E
扩展置换，
32
位
->48
位
173.
        eRight 
=
 E_explaned
(
Right
);
174.
//
将右半部分与子密钥进行异或操作
175.
        xor_result 
=
 XOR
(
eRight
,
 subkeys
[
i
]);
176.
//
异或结果进入
Sbox
，输出
32
位结果
177.
        s_result 
=
 S_BoxTrans
(
xor_result
);
178.
//P
置换得到轮函数最终结果
179.
        f_result 
=
 P_Transform
()
180.
//
明文左半部分与轮函数结果进行异或
181.
Ri
=
 XOR
(
Left
,
f_result
);
182.
// Li
和
Ri
交换
183.
if
(
i 
<
15
){
184.
Swap
(
Right
,
Ri
);
185.
}
186.
        memcpy
(
afterIP
,
Right
,
32
);
187.
        memcpy
(
afterIP
+
32
,
Ri
,
32
);
188.
}
189.
//
逆初始置换（
IP^-1
置换）
190.
    afterFP 
=
Final_Permutation
(
afterIP
);
191.
    ciphertext 
=
 bitTochar
(
afterFP
);
192.
}
193.
194.
//
解密
195.
// 8
个字节
(64
位
)
密文利用子密钥转为明文
196.
void
Decrypt
(
char
*
 ciphertext
,
char
 subKeys
[
16
][
48
],
char
*
 plaintext
)
{
197.
char
*
 cipher 
=
new
char
[
64
];
198.
char
*
 afterIP 
=
new
char
[
64
];
199.
char
*
Left
=
new
char
[
48
];
200.
char
*
Right
=
new
char
[
48
];
201.
char
*
 eRight 
=
new
char
[
48
];
202.
char
*
 xor_result 
=
new
char
[
48
];
203.
char
*
 s_result 
=
new
char
[
32
];
204.
char
*
 f_result 
=
new
char
[
32
];
205.
char
*
Ri
=
new
char
[
32
];
206.
char
*
 afterFP 
=
new
char
[
64
];
207.
208.
    cipher 
=
 charTobit
(
ciphertext
);
209.
//
初始
IP  
210.
    afterIP 
=
Init_Permutation
(
cipher
);
211.
212.
//16
轮迭代
213.
for
(
int
 i 
=
15
;
 i 
>=
0
;
 i
--){
214.
// 
左半部分
215.
        memcpy
(
Left
,
 afterIP
,
32
);
216.
// 
右半部分
217.
        memcpy
(
Right
,
afterIP
+
32
,
32
);
218.
//
右半部分
E
扩展置换，
32
位
->48
位
219.
        eRight 
=
 E_explaned
(
Right
);


计算机生成了可选文字:
220.
//
将右半部分与子密钥进行异或操作
221.
        xor_result 
=
 XOR
(
eRight
,
 subkeys
[
i
]);
222.
//
异或结果进入
Sbox
，输出
32
位结果
223.
        s_result 
=
 S_BoxTrans
(
xor_result
);
224.
//P
置换得到轮函数最终结果
225.
        f_result 
=
 P_Transform
()
226.
//
明文左半部分与轮函数结果进行异或
227.
Ri
=
 XOR
(
Left
,
f_result
);
228.
// Li
和
Ri
交换
229.
if
(
i 
<
15
){
230.
Swap
(
Right
,
Ri
);
231.
}
232.
        memcpy
(
afterIP
,
Right
,
32
);
233.
        memcpy
(
afterIP
+
32
,
Ri
,
32
);
234.
}
235.
//
逆初始置换（
IP^-1
置换）
236.
    afterFP 
=
Final_Permutation
(
afterIP
);
237.
    plaintext 
=
 bitTochar
(
afterFP
);
238.
}
◦
◾
未分类
 1 
◾
欢迎使用
 Cmd Markdown 
编辑阅读器
搜索我的文稿标题，
 * 
显示全部
如何分类
◦
下载客户端
◦
变更历史
◦
关注开发者
◦
报告问题，建议
◦
联系我们
◦
Web 
端版本
•
•
•
•
×
正在加载文章图片，请稍等片刻
... 
添加新批注
保存
取消
保存
取消
修改
保存
取消
删除
•
私有
•
公开
•
删除
查看更早的
 5 
条回复
回复批注
