

# 密码学实验报告

姓名：吴静  
学号：2113285  
专业：信息安全

## 一、SPN加密算法

### 代码

首先是全局变量，定义了s盒和p盒，还有一个输出函数：

```
1 char s[17] = { 'E', '4', 'D', '1', '2', 'F', 'B', '8', '3', 'A', '6', 'C', '5',  
  '9', '0', '7', '\0' };  
2 int p[17] = { 0, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16 };  
3  
4 void output(char* s);
```

其次是main函数：

```
1 int main() {  
2     char x[17];  
3     char k[33];  
4     cout << "Please input x:" << endl;  
5     input_x(x);  
6     cout << "Please input k:" << endl;  
7     input_k(k);  
8     char temp_w[17];  
9     char temp_k[17];  
10    char temp_u[17];  
11    char temp_v[17];  
12    memcpy(temp_w, x, 17);  
13    for (int r = 1; r < 5; r++) { //表示轮数  
14        cout << endl << "round" << r << ":" << endl;  
15        memcpy(temp_k, k + 4 * (r - 1), 16);  
16        temp_k[16] = '\0';  
17        cout << "k:";  
18        output(temp_k);  
19        get_u(temp_u, temp_w, temp_k);  
20        get_v(temp_v, temp_u);  
21        if(r!=4)  
22            get_w(temp_w, temp_v);  
23    }  
24    cout << endl << "round5:" << endl;  
25    memcpy(temp_k, k + 16, 16);  
26    temp_k[16] = '\0';  
27    cout << "k:";  
28    output(temp_k);  
29    char y[17];
```

```

30 |     get_y(y, temp_v, temp_k);
31 | }
32 |

```

main 函数主要实现的功能是输入16位明文，通过调用各种函数对输入的明文进行加密。

对于生成密文的过程，按照伪代码进行编程即可，算法如下：

---

算法 3.1  $\text{SPN}(x, \pi_s, \pi_p, (K^1, \dots, K^{Nr+1}))$

$$w^0 \leftarrow x$$

**for**  $r \leftarrow 1$  **to**  $Nr-1$

$$\left\{ \begin{array}{l} u^r \leftarrow w^{r-1} \oplus K^r \\ \textbf{for } i \leftarrow 1 \textbf{ to } m \\ \textbf{do } \left\{ \begin{array}{l} v_{<i>}^r \leftarrow \pi_s(u_{<i>}^r) \\ w^r \leftarrow (v_{\pi_p(1)}^r, \dots, v_{\pi_p(\ell_m)}^r) \end{array} \right. \end{array} \right.$$

$$u^{Nr} \leftarrow w^{Nr-1} \oplus K^{Nr}$$

**for**  $i \leftarrow 1$  **to**  $m$

$$\textbf{do } v_{<i>}^{Nr} \leftarrow \pi_s(u_{<i>}^{Nr})$$

$$y \leftarrow v^{Nr} \oplus K^{Nr+1}$$

**output**(y)

---

首先\$w\_0\$为明文  $x$ ，接着在每一个轮次下面计  $u, v$  和下一轮的轮密钥  $k$ ；其中包括s盒置换和p盒置换，最后计算  $y$ 。

部分函数解释如下：

get\_u(temp\_u, temp\_w, temp\_k); 用于计算w和k的异或，最后存入u数组中：

```

1 | void get_u(char* u, char* w, char* K) {
2 |     for (int i = 0; i < 16; i++)
3 |         u[i] = xor_x1x2(w[i], K[i]);
4 |     u[16] = '\0';
5 |     cout << "u:";
6 |     output(u);
7 | }

```

get\_v(temp\_v, temp\_u); 用于计算v的值，是通过u的s置换得到的：

```

1 | void get_v(char* v, char* u) {
2 |     for (int i = 0; i < 4; i++) {
3 |         char u_content[5];
4 |         memcpy(u_content, u + 4 * i, 4);
5 |         u_content[4] = '\0';
6 |         int decnum = BintoDec(u_content, 4);
7 |         char tempptr = s[decnum];
8 |         HextoBin(v + i * 4, tempptr);
9 |         //在四次循环后完成v的转化
10 |    }

```

```

11     v[16] = '\0';
12     cout << "v:";
13     output(v);
14 }

```

get\_w(temp\_w, temp\_v); 用于获取当前轮次下的轮密钥:

```

1 void get_w(char* w, char* v) {
2     for (int i = 0; i < 16; i++) {
3         w[i] = *(v + p[i + 1] - 1);
4         //v下标从1开始计数,但是实际存是从0开始,所以要减1
5         //p的下标也从1开始有意义,所以要加1
6     }
7     w[16] = '\0';
8     cout << "w:";
9     output(w);
10 }

```

get\_y(y, temp\_v, temp\_k); 则是计算最后的y值, 计算方法和u一样:

```

1 void get_y(char* y, char* v, char* k) {
2     for (int i = 0; i < 16; i++)
3         y[i] = xor_x1x2(v[i], k[i]);
4     y[16] = '\0';
5     cout << endl << "y:";
6     output(y);
7 }

```

另外还有两个进制转换函数 BintoDec 和 HextoBin, 分别为将二进制转换为十进制和将十六进制转换为二进制:

```

1 int BintoDec(char* s, int length) {
2     int dec = 0;
3     for (int i = 0; i < length; i++) {
4         if (s[i] == '0')
5             continue;
6         dec += pow(2, length - 1 - i);
7     }
8     return dec;
9 }
10
11 void HextoBin(char* binaryStr, char s) {
12     int text;
13     if (s <= 'Z' && s >= 'A')
14         text = s - 'A' + 10;
15     else
16         text = s - '0';
17
18     for (int i = 3; i >= 0; i--) {
19         if (text % 2)

```

```

20         binaryStr[i] = '1';
21     else
22         binaryStr[i] = '0';
23     text /= 2;
24 }
25 binaryStr[4] = '\0';
26 }

```

## 运行结果截图

```

Please input x:
0010011010110111
Please input K:
00111010100101001101011000111111

round1:
k:0011 1010 1001 0100
u:0001 1100 0010 0011
v:0100 0101 1101 0001
w:0010 1110 0000 0111

round2:
k:1010 1001 0100 1101
u:1000 0111 0100 1010
v:0011 1000 0010 0110
w:0100 0001 1011 1000

round3:
k:1001 0100 1101 0110
u:1101 0101 0110 1110
v:1001 1111 1011 0000
w:1110 0100 0110 1110

round4:
k:0100 1101 0110 0011
u:1010 1001 0000 1101
v:0110 1010 1110 1001

round5:
k:1101 0110 0011 1111
y:1011 1100 1101 0110

```

## 二、数据集的准备与获取

### 思路

由于要进行SPN的线性攻击算法，所以要准备许多数据集，这些数据集是使用同一个密钥加密的明文和密文，因为数量太大，所以我们采用随机数随机生成明文的方式进行，再用SPN加密算法加密随机生成的明文，并将明文和密文一起存入一个新文件 `example.txt` 中方便调用。

于是将明文的获取方式变成随机数的输入，用户所需要输入的只有生成明文-密文对的数量num。


于是便有了main函数中随机数的产生以及num的输入。

```
1  int main() {
2      //随机数的产生
3      random_device rd;
4      mt19937 gen(rd());
5      uniform_int_distribution<int> distribution(0, 1);
6
7      int num;
8      cout << "Please input k:" << endl;
9      char k[33];
10     input_K(k);
11
12     //输入明文-密文对数目
13     cout << "please input number:" << endl;
14     cin >> num;
15
16     //打开文件，准备进行写入
17     outputfile.open("example.txt");
18     for(int ii=0;ii<num;ii++)
19         //循环，表示对每一个生成的明文进行加密
20         {
21             char x[17];
22             cout << "Please input x:" << endl;
23             for (int i = 0; i < 16; i++)
24                 x[i] = distribution(gen) + '0';
25             //明文采用随机数生成的方式
26             x[16] = '\0';
27             //input_x(x);
28             cout << x << endl;
29             outputfile << x << endl;
30             //向文件中写入随机数生成的明文
31
32             .....
33             (中间部分和前面SPN攻击算法的一样，不过多赘述)
34             .....
35         }
36 }
```

在获得密文y后写入文件中。

至此，我们可以得到特定密钥生成的一定数量的明文-密文对数目，在文件example.txt中，方便线性攻击时使用。

## 运行结果

 example.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

---

1010001001011000  
0011110111010010  
0010010001000010  
0000110010111000  
0011000011100011  
0110110111101000  
1011011000001010  
1011000001110110  
0110111110110100  
0100010110000000  
1100001100000101  
0011101111001001  
1111111000000100  
0101000010100010  
1101110001100000  
0110001010110101  
0101110100101100  
0011001010010011

以上为 example.txt 部分截图。

### 三、 SPN的线性攻击算法

---

#### 代码文件

先看伪代码：

---

算法 3.2 线性攻击 ( $T, T, \pi_S^{-1}$ )

```

for ( $L_1, L_2$ )  $\leftarrow (0, 0)$  to ( $F, F$ )
  do Count[ $L_1, L_2$ ]  $\leftarrow 0$ 
  for each ( $x, y$ )  $\in T$ 
    {
    for ( $L_1, L_2$ )  $\leftarrow (0, 0)$  to ( $F, F$ )
      {
       $v_{<2>}^4 \leftarrow L_1 \oplus y_{<2>}$ 
       $v_{<4>}^4 \leftarrow L_2 \oplus y_{<4>}$ 
       $u_{<2>}^4 \leftarrow \pi_S^{-1}(v_{<2>}^4)$ 
       $u_{<4>}^4 \leftarrow \pi_S^{-1}(v_{<4>}^4)$ 
       $z \leftarrow x_5 \oplus x_7 \oplus x_8 \oplus u_6^4 \oplus u_8^4 \oplus u_{14}^4 \oplus u_{16}^4$ 
      if  $z = 0$ 
        then Count[ $L_1, L_2$ ]  $\leftarrow$  Count[ $L_1, L_2$ ] + 1
      }
    }
  max  $\leftarrow -1$ 
  for ( $L_1, L_2$ )  $\leftarrow (0, 0)$  to ( $F, F$ )
    {
    Count[ $L_1, L_2$ ]  $\leftarrow$  |Count[ $L_1, L_2$ ] -  $T/2$ |
    do if Count[ $L_1, L_2$ ] > max
      then {
      max  $\leftarrow$  Count[ $L_1, L_2$ ]
      maxkey  $\leftarrow (L_1, L_2)$ 
      }
    }
  output(maxkey)

```

---

接下来我将根据伪代码的部分解释main函数：

main 函数：

```

1  int main() {
2      string x;
3      string y;
4      streampos lastPos = 0;
5      int num;
6      cout << "Please input num:" << endl;
7      cin >> num;
8      for (int i = 0; i < 16; i++) {
9          if (s[i] >= '0' && s[i] <= '9')
10             s1[s[i] - '0'] = i;
11          else
12             s1[s[i] - 'A' + 10] = i;
13      }
14      for (int i = 0; i < 16; i++)
15          for (int j = 0; j < 16; j++)
16             Count[i][j] = 0;
17
18      ifstream inputfile;
19      inputfile.open("example.txt");
20      for(int i=0;i<num;i++)
21      {
22          inputfile.seekg(lastPos);
23          getline(inputfile, x);

```

```

24     getline(inputfile, y);
25     lastPos = inputfile.tellg();
26     linear(x, y);
27 }
28
29 int max = -1;
30 int maxkey_1 = 0;
31 int maxkey_2 = 0;
32 for (int i = 0; i < 16; i++) {
33     for (int j = 0; j < 16; j++) {
34         Count[i][j] = abs(Count[i][j] - num / 2);
35         if (Count[i][j] > max) {
36             max = Count[i][j];
37             maxkey_1 = i;
38             maxkey_2 = j;
39         }
40     }
41 }
42 int maxkey_L1[4];
43 int maxkey_L2[4];
44 DectoBin(maxkey_L1, maxkey_1);
45 DectoBin(maxkey_L2, maxkey_2);
46 cout << "maxkey:" << endl;
47 for (int i = 0; i < 4; i++)
48     cout << maxkey_L1[i];
49 cout << ' ';
50 for (int i = 0; i < 4; i++)
51     cout << maxkey_L2[i];
52 cout << ' ';
53
54 }

```

1. 首先当然要依次读取数据集中的数据，总共要读取 `num` 次，为了在循环中每次都能读取到上一次读取到的数据的后面，我们加了一个定位符 `lastPos`，首先赋值为 0，然后在每次读取数据后都更新定位符，以方便下一次读取数据时不读取重复。
2. 首先求出 S 置换的逆置换 `s1`。
3. 初始化计数器 `Count`。
4. 依次读取文件中的明文和密文，存入 `x` 和 `y` 中。
5. 通过 `linear` 函数进行计数器的计数。

先来看计数器计数的算法实现：

```

1 void linear(string x,string y) {
2
3     for (int i = 0; i < 16; i++) {
4         for (int j = 0; j < 16; j++) {
5             DectoBin(L1, i);
6             DectoBin(L2, j);
7

```



```

8         v[4] = L1[0] ^ (y[4] - '0');
9         v[5] = L1[1] ^ (y[5] - '0');
10        v[6] = L1[2] ^ (y[6] - '0');
11        v[7] = L1[3] ^ (y[7] - '0');
12
13        v[12] = L2[0] ^ (y[12] - '0');
14        v[13] = L2[1] ^ (y[13] - '0');
15        v[14] = L2[2] ^ (y[14] - '0');
16        v[15] = L2[3] ^ (y[15] - '0');
17
18        int len1 = v[4] * pow(2, 3) + v[5] * pow(2, 2) + v[6] * pow(2, 1) +
v[7] * pow(2, 0);
19        DectoBin(u+4, s1[len1]);
20        //cout << "len1:" << len1 << " s1[len1]:" << s1[len1] << " u:" <<
u[4] << u[5] << u[6] << u[7] << endl;
21
22        int len2 = v[12] * pow(2, 3) + v[13] * pow(2, 2) + v[14] * pow(2,
1) + v[15] * pow(2, 0);
23        DectoBin(&(u[12]), s1[len2]);
24        //cout << "len2:" << len2 << " s1[len2]:" << s1[len2] << " u:" <<
u[12] << u[13] << u[14] << u[15] << endl;
25
26        int z = (x[4]-'0') ^ (x[6]-'0') ^ (x[7]-'0') ^ u[5] ^ u[7] ^ u[13]
^ u[15];
27        //cout << "z:" << z << endl;
28
29        if (z == 0)
30            Count[i][j]++;
31    }
32 }
33 }

```

以下为部分解释：

1. 首先遍历 (0,0) 到 (F,F)，利用双层循环分别给 L1 和 L2 赋值，由于在之后的运算中，L1 和 L2 是以四位二进制数出现的，所以这里首先要注意预处理数据：将十六进制数转化为四位二进制数

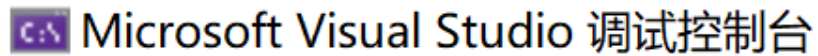
DectoBin：

```

1 void DectoBin(int* binaryStr, int text) {
2     //把十进制转化为四位二进制数
3
4     for (int i = 3; i >= 0; i--) {
5         if (text % 2)
6             binaryStr[i] = 1;
7         else
8             binaryStr[i] = 0;
9         text /= 2;
10    }
11 }

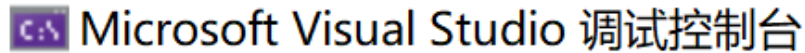
```





```
Please input num:
9000
maxkey:
0000 0000
```

密钥为 1111111111111111111111111111111111:



```
Please input num:
2000
maxkey:
1111 1111
```

等等，以上均运行成功。

**注意作业中给的文件中,**

- example.txt 包含了密钥为 00111010100101001101011000111111 的16000组明文-密文对;
- example1.txt 包含了密钥为 00000000000000000000000000000000 的16000组明文-密文对;
- example2.txt 包含了密钥为 11111111111111111111111111111111 的10000组明文-密文对;