2111252 李佳豪 信息安全

密码第二次实验，APN加密以及线性攻击

1、加密

代码思路，根据书中提供的伪代码进行即可；

- `int get_ki(int k, int i)` 根据密钥扩展算法返回第i+1次的Key

```
int get_ki(int k, int i) {
    int rs = ((k << (i * 4) & 0xFFFF0000 ) >> 16 );
   // cout << "K_" << i << ": " << toBinary(rs) << endl;
    return rs;
}
```

- `int encrypto(int x)` 输入是一个16位整型x，输出即为SPN加密结果，相关参数均与题目要求一致;

  核心部分（K异或、sBox、pBox）操作列出

```
for (int n = 0;n < 3;n++) {
     int kn = get_ki(K, n);
     int u = w ^ kn;
        // S盒子
     int v = 0;
     for (int m = 0;m < 16;m += 4) {
         int vm = u >> m & 0xF;
         v |= (SBox[vm] << m);
     }
       //P
     w = 0;
     for (int m = 15;m >= 0;m--) {
         int v_bit = v & 0x1;
         v = v >> 1;
         w |= v_bit << (16 - PBox[m]);
     }
    }
    int kn = get_ki(K, 3);
    int u = w ^ kn;
    ... ...
     //再进行一次S与疑惑第五轮Key即可,最后返回下述的y是加密后的16位
    y = v ^ kn;
```

2、SPN线性攻击

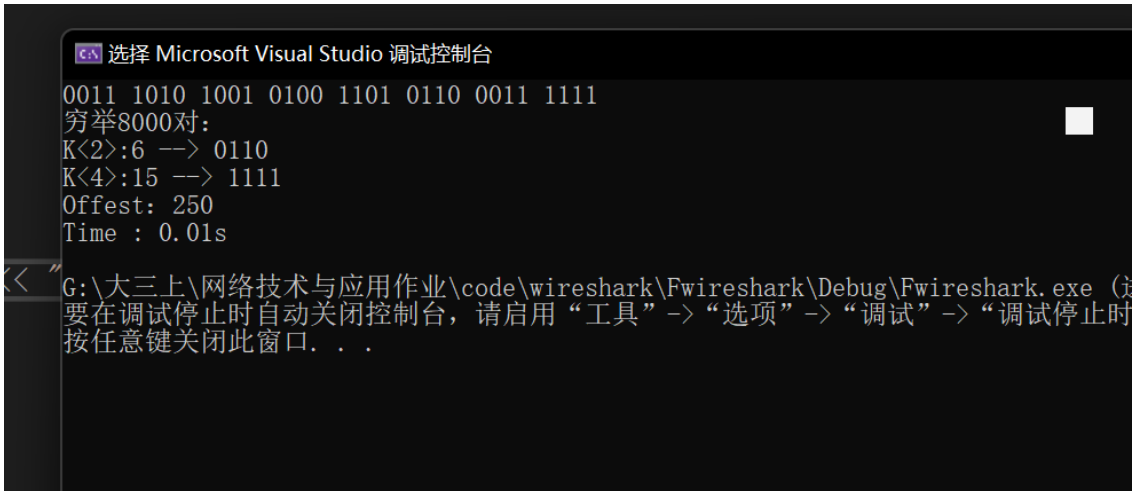
思路：首先根据课本提供的线性攻击链和伪代码，构造8000对明密对，遍历所有可能的第五轮第二、四组密钥，Count攻击链结果为0的值，最后取其Count的偏移最大值，即完成一半的破击任务；之后利用得到的一半第五轮密钥，构造新的攻击链使用同样思路破解第一、三组密钥；

- 使用rand种子和SPN加密算法函数，构造8000对明密对

```
struct Dpair {
    int plain = 0;
    int encry = 0;

};  // 这是空存储明密对的映射

srand(time(NULL));
for (int i = 0;i < Maxsize;i++) {
    ps[i].plain = rand() & 0xFFFF;
    ps[i].encry = encrypto(ps[i].plain);
}
```

- 根据伪代码遍历

```
  int count[16][16] = { 0 };

  for (int p_i = 0;p_i < ps.size();p_i++) {
       Dpair p = ps[p_i];
        for (int i = 0;i < 16;i++) {
             for (int j = 0;j < 16;j++) {

                 int v2 = i ^ ((p.encry & 0x0F00) >> 8);
                 int v4 = j ^ (p.encry & 0x000F);
                 //cout << toBinary(p.encry, 16) << " " << toBinary(v2,4) <<
endl;
                 int u2 = S_Box[v2];
                 int u4 = S_Box[v4];
                 //这里对应伪代码一长串的异或攻击链
                 int z = ((p.plain >> 11) ^ (p.plain >> 9) ^ (p.plain >> 8)
^
                          (u2 >> 2) ^ (u2) ^ (u4 >> 2) ^ u4) & 0x1;
                 if (!z)
                    count[i][j] += 1;
             }
        }
 }
//开始找最大 _i返回最后的K<2> K<4>  _count记录最大值
 int _i = -1, _j = -1;
 int _count = -1;

 for (int i = 0;i < 16;i++) {
    for (int j = 0;j < 16;j++) {
       count[i][j] = abs(count[i][j] - Maxsize / 2);
        if (count[i][j] > _count)
         {
            _i = i; _j = j;
             _count = count[i][j];
          }
     }
 }
```



- 接着使用其他构造链:

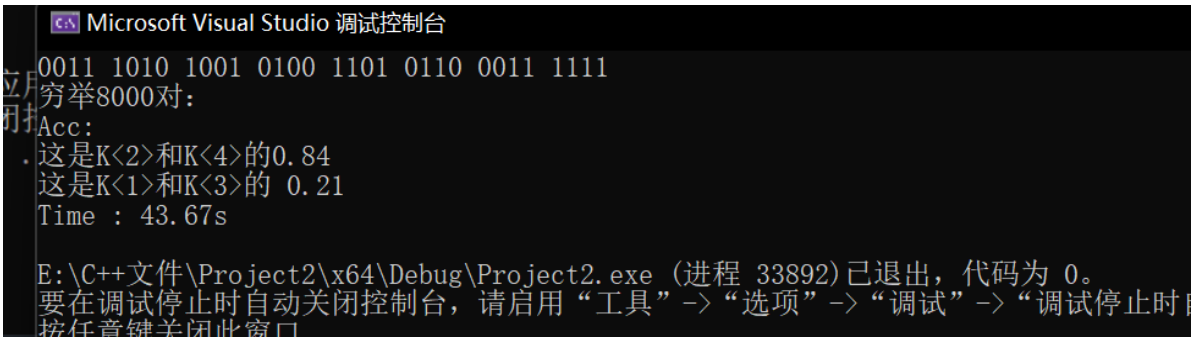
  查找相关资料得知，由于关于K<1> 和 K<3>构造链结果偏移量都较小，选择如下两个构造链值相加。

$$x_0 \oplus x_1 \oplus x_4 \oplus u_1^4 \oplus u_5^4 \oplus u_9^4 \oplus u_{13}^4$$
$$x_9 \oplus x_{10} \oplus x_{12} \oplus u_3^4 \oplus u_7^4 \oplus u_{11}^4 \oplus u_{15}^4$$

  核心代码，最后计算2000次，查看K<1>和 K<3>的准确率并不高

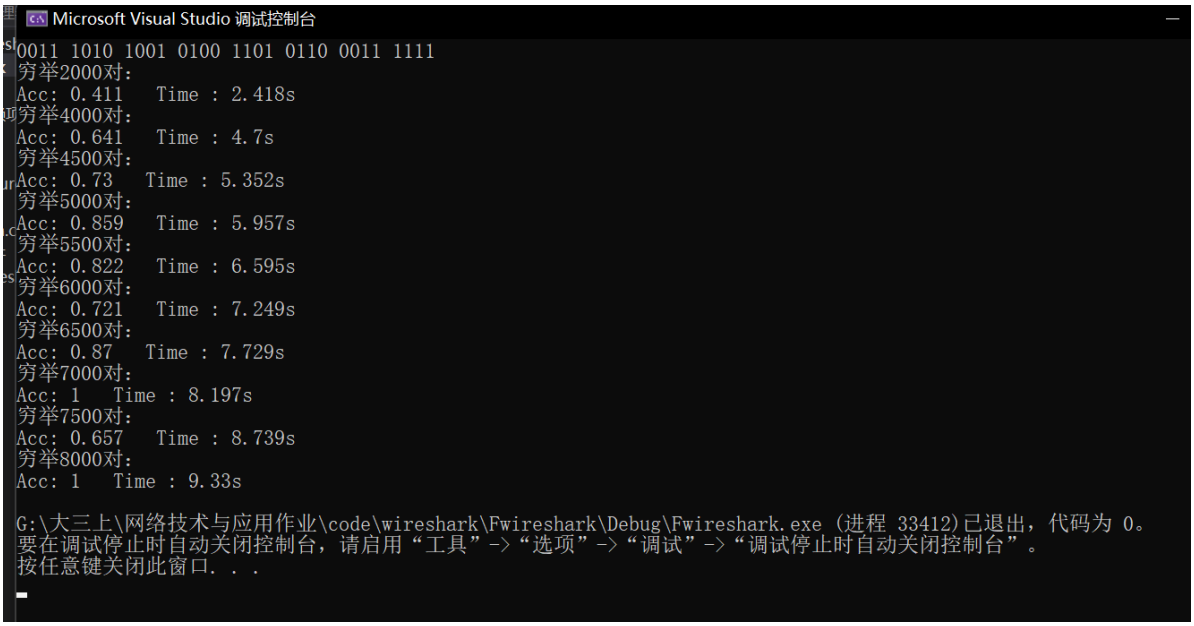
```
 int count1[16][16] = { 0 };
 int count2[16][16] = { 0 };
     for (Dpair p : ps) {
         int  y = p.encry;
         int  x = p.plain;
         for (int i = 0;i < 16;i++) {
             for (int j = 0;j < 16;j++) {
                 int u1 = S_Box[(y & 0x000F) ^ i];
                 int u2 = S_Box[(y & 0x00F0) >> 4 ^ _i];
                 int u3 = S_Box[(y & 0x0F00) >> 8 ^ j];
                 int u4 = S_Box[(y & 0xF000) >> 12 ^ _j];

// k<1>
                int z = ((x >> 15) ^ (x >> 14) ^ (x >> 12) ^ (u1 >> 3) ^ (u2
>> 3) ^ (u3 >> 3) ^ (u4 >> 3)) & 0x1;
                if (!z)
                    count1[i][j]++;
                    z = ((x >> 7) ^ (x >> 6) ^ (x >> 4) ^ (u1 >> 1) ^ (u2 >>
1) ^ (u3 >> 1) ^ (u4 >> 1)) & 0x1;
                if (!z)
                   ount2[i][j]++;
                }
            }
       }

for (int i = 0; i < 16; i++)
    for (int j = 0; j < 16; j++)
         count[i][j] = abs(count1[i][j]) +abs(count2[i][j]);;

_count = 0;
int _k1 = 0, _k3 = 0;
 for (int i = 0;i < 16;i++) {
```

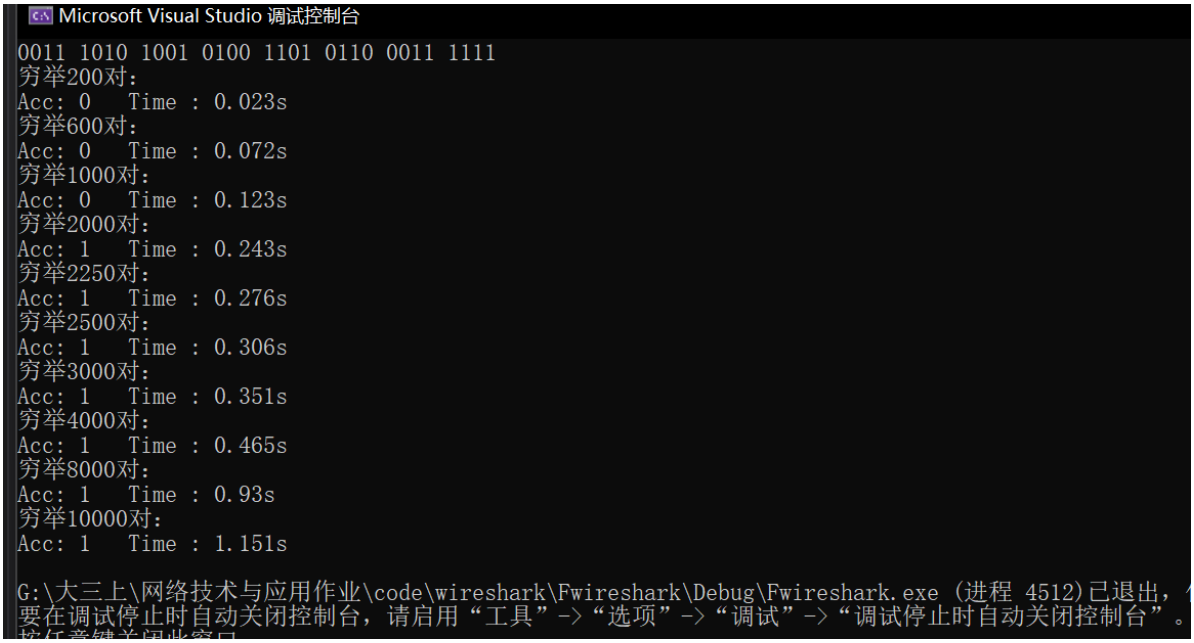
```
        for (int j = 0;j < 16;j++) {
            if (count[i][j] > _count){
                _k1 = i;
                _k3 = j;
                _count = count[i][j];
            }
        }
    }
```



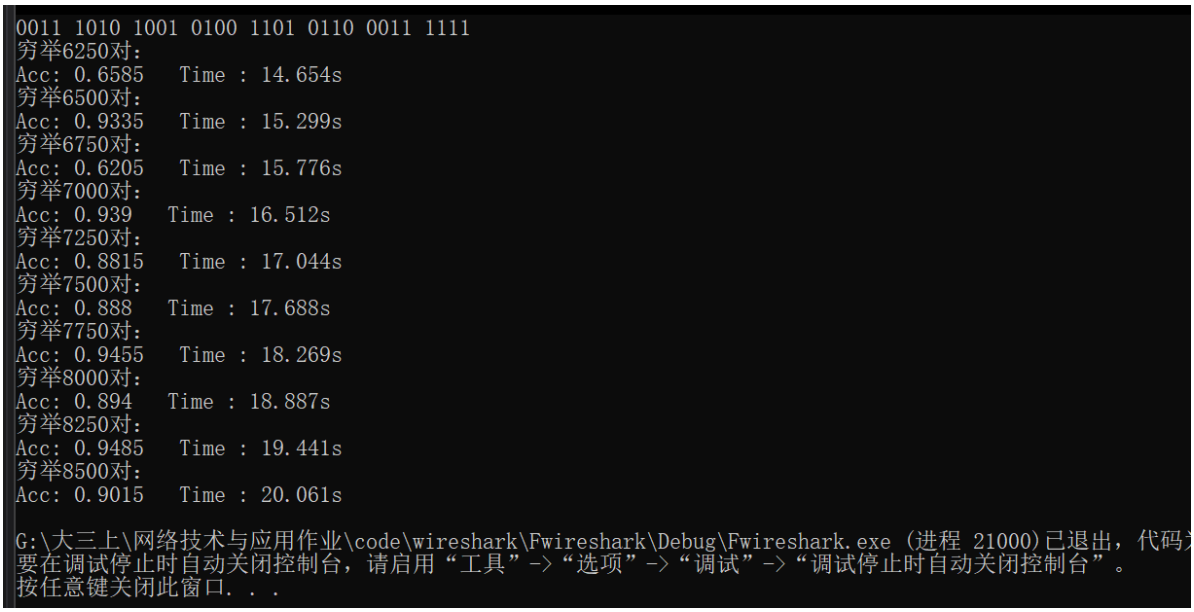
3、枚举明密对数量的影响

思路，遍历明密对数量从1000到10000，对于每一个数量使用不同随机数种子生成不同明密对，重复100次，观察其攻击的成功率和时间；多次运行查看结果，发现其突变大致6500-8000之间





之后在6500-8000内，每确定一个数量遍历次数提高到2000次，使攻击成功率更准确。进一步观察其成功率



结论

- 经过多次测试，在明密文对数到达7500-8200这个范围时，攻击成功率已经稳定在0.9左右，这样符合了课本给出的8000这个数值，但偶尔会在较低明密对数时（6000-7000）也会有很高的准确率，这里我想或许跟随机数种子有关；
- 观察时间，发现对数在200 - 2000时，增加几乎呈线性，但随着对数增多，时间增长速度减缓。

4、附录

完整代码如下，下面代码生成完整密钥，目录代码是测试明秘对数量影响的:

```
#include<iostream>
#include<cstring>
#include<vector>
#include <ctime>

using namespace std;
```

```
const int K = 0b00111010100101001101011000111111;

int SBox[16] = { 0xE, 0x4, 0xD, 0x1, 0x2, 0xF, 0xB, 0x8, 0x3, 0xA, 0x6, 0xC,
0x5, 0x9, 0x0, 0x7 };
int S_Box[16] = { 0xE, 0x3, 0x4, 0x8, 0x1, 0xC, 0xA, 0xF, 0x7, 0xD, 0x9, 0x6,
0xB, 0x2, 0x0, 0x5 };
int PBox[16] = { 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16 };

int count2 = 0;


struct Dpair {
    int plain = 0;
    int encry = 0;

};


string toBinary(int n, int l) {
    string result;
    int i = 0;
    while (i < l) {
        result.insert(result.begin(), '0' + (n & 1));
        ++i;
        if (i % 4 == 0 && i != l)
            result.insert(result.begin(), ' ');
        n >>= 1;
    }

    return result.empty() ? "0" : result;
}
string toBinary(int n) {
    return toBinary(n, 16);
}
int get_ki(int k, int i) {
    int rs = ((k << (i * 4) & 0xFFFF0000) >> 16);
    // cout << "K_" << i << ": " << toBinary(rs) << endl;
    return rs;
}
int stoii(char* chr, int l) {
    int rs = 0;
    for (int i = 0;i < l;i++)
        rs = 2 * rs + chr[i] - '0';
    return rs;

}
int encrypto(int x) {

    int w = x;
    // cout << toBinary(x)<<endl;
    for (int n = 0;n < 3;n++) {
        int kn = get_ki(K, n);
        int u = w ^ kn;
        //cout << "u_" << n << ": " << toBinary(u) << endl;
```

```
        // S
        int v = 0;
        for (int m = 0;m < 16;m += 4) {
            int vm = u >> m & 0xF;
            v |= (SBox[vm] << m);

        }
        //cout <<"v_"<<n<<": " << toBinary(v)<<endl;

        //P
        w = 0;
        for (int m = 15;m >= 0;m--) {
            int v_bit = v & 0x1;
            v = v >> 1;
            w |= v_bit << (16 - PBox[m]);
        }
        // cout << "w_" << n << ": " << toBinary(w) << endl;
    }
    int kn = get_ki(K, 3);
    int u = w ^ kn;
    //  cout << "u_3: " << toBinary(u) << endl;

    // S
    int v = 0;
    for (int m = 0;m < 16;m += 4) {

        int vm = u >> m & 0xF;
        v |= (SBox[vm] << m);
    }


    kn = get_ki(K, 4);
    int y = 0;
    y = v ^ kn;

    //cout << toBinary(y);
    return y;
}

int main() {
    cout << toBinary(K, 32) << endl;
    int Maxsize = 8000;

    int k2 = 6;
    int k4 = 15;
    int k1 = 13;
    int k3 = 3;


    int si_ls[10] = { 8000 };
    for (int idx = 0;idx < 1;idx++) {
        Maxsize = si_ls[idx];
        vector<Dpair> ps(Maxsize);
        cout << "穷举" << Maxsize << "对: " << endl;
```

```
        clock_t start = clock();
        double a = 0;
        double b = 0;
        for (int epoch = 0;epoch < 2000;epoch++) {


            srand(static_cast<unsigned int>(time(nullptr)));
            for (int i = 0;i < Maxsize;i++) {
                ps[i].plain = rand() & 0xFFFF;
                ps[i].encry = encrypto(ps[i].plain);
            }


            int count[16][16] = { 0 };

            for (int p_i = 0;p_i < ps.size();p_i++) {
                Dpair p = ps[p_i];

                for (int i = 0;i < 16;i++) {
                    for (int j = 0;j < 16;j++) {

                        int v2 = i ^ ((p.encry & 0x0F00) >> 8);
                        int v4 = j ^ (p.encry & 0x000F);
                        //cout << toBinary(p.encry, 16) << " " << toBinary(v2,4)
<< endl;
                        int u2 = S_Box[v2];
                        int u4 = S_Box[v4];
                        int z = ((p.plain >> 11) ^ (p.plain >> 9) ^ (p.plain >>
8) ^ (u2 >> 2) ^ (u2) ^ (u4 >> 2) ^ u4) & 0x1;
                        if (!z)

                            count[i][j] += 1;
                    }

                }
            }
            int _i = -1, _j = -1;
            int _count = -1;

            for (int i = 0;i < 16;i++) {
                for (int j = 0;j < 16;j++) {
                    count[i][j] = abs(count[i][j] - Maxsize / 2);
                    if (count[i][j] > _count)
                    {
                        _i = i;
                        _j = j;
                        _count = count[i][j];
                    }
                }
            }
           // cout << "K<2>:" << _i<<" --> " << toBinary(_i, 4) << "\n" <<
"K<4>:" << _j <<" --> " << toBinary(_j, 4) << " " << "\nOffest: " << _count <<
endl;
```

```
            int count1[16][16] = { 0 };
            int count2[16][16] = { 0 };
            for (Dpair p : ps) {
                int  y = p.encry;
                int  x = p.plain;
                for (int i = 0;i < 16;i++) {
                    for (int j = 0;j < 16;j++) {
                        int u1 = S_Box[(y & 0x000F) ^ i];
                        int u2 = S_Box[(y & 0x00F0) >> 4 ^ _i];
                        int u3 = S_Box[(y & 0x0F00) >> 8 ^ j];
                        int u4 = S_Box[(y & 0xF000) >> 12 ^ _j];

                        // k<1>
                        int z = ((x >> 15) ^ (x >> 14) ^ (x >> 12) ^ (u1 >> 3) ^
(u2 >> 3) ^ (u3 >> 3) ^ (u4 >> 3)) & 0x1;
                        if (!z)
                            count1[i][j]++;


                        z = ((x >> 7) ^ (x >> 6) ^ (x >> 4) ^ (u1 >> 1) ^ (u2 >>
1) ^ (u3 >> 1) ^ (u4 >> 1)) & 0x1;
                        if (!z)
                            count2[i][j]++;


                    }
                }
            }

            for (int i = 0; i < 16; i++)
                for (int j = 0; j < 16; j++)
                {

                    count[i][j] = abs(count1[i][j]) +abs(count2[i][j]);;
                }
            //  count[i][j] = abs(count1[i][j]) + abs(count2[i][j]);


            _count = 0;
            int _k1 = 0, _k3 = 0;
            for (int i = 0;i < 16;i++) {
                for (int j = 0;j < 16;j++) {
                    //count[i][j] = abs(count[i][j] - Maxsize);
                    if (count[i][j] > _count)
                    {
                        _k1 = i;
                        _k3 = j;
                        _count = count[i][j];
                    }
                }
            }
           //cout << "K<1>:" << _i << " " << "K<3>:" << _j << " " << "共出现" <<
_count << endl;
```

```
          // cout << "K<1>:" << _k1 << " --> " << toBinary(_k1, 4) << "\n" <<
"K<3>:" << _k3 << " --> " << toBinary(_k3, 4) << " " << "\nOffest: " << _count <<
endl;
          // cout << "K<2>:" << _i << " --> " << toBinary(_i, 4) << "\n" <<
"K<4>:" << _j << " --> " << toBinary(_j, 4) << " " << "\nOffest: " << _count <<
endl;

            if (_i == k2 && _j == k4 )
                a += 1;

            if (_i == k2 && _j == k4)
                b += 1;
        }

        clock_t end = clock();
        double elapsed_secs = double(end - start) / CLOCKS_PER_SEC;
        //cout <<"Time : " << elapsed_secs << "s" << endl;
        cout << "Acc: " << "\n这是K<2>和K<4>的" << a / 2000 << "\n这是K<1>和K<3>的
" << b / 8000 << "\nTime : " << elapsed_secs << "s" << endl;
    }
    return 0;
}
```