密码学实验报告

2112492 刘修铭 信息安全专业

https://github.com/lxmliu2002/Cryptography

一、SPN加密算法

（一）代码文件

- SPN.cpp

（二）核心代码解读

按照教材中伪代码，分块实现了SPN加密算法。下面将对核心代码进行分块说明：

- 由于给定明文或密钥均为确定的16或32位，故而在全局部分直接定义了三个数组用来保存明文、密文与密钥；同时全局部分定义了`S`盒与`P`盒数据。

```
int S[16] = { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 };
int P[16] = { 0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15 };
int x[16], y[16], key[32];
```

- 接着定义了`S`盒函数，接受三个整数数组作为参数：`a`、`b`和`s`。
  - 将数组`a`中的值转换成十进制后，参考`S`盒中的数据进行替换，接着再转换成对应的二进制数，从而完成`S`盒函数的相关功能。

```
void Substitution(int* a, int* b, int* s)
{
    int T[4] = {0};
    int k = 0;
    for (int i = 0; i < 4; i++)
    {
        T[i] = a[i * 4] * pow(2, 3) + a[i * 4 + 1] * pow(2, 2) + a[i * 4 + 2] * pow(2, 1) + a[i * 4 + 3];
        int temp = s[T[i]];
        int j = (i * 4) + 3;
        while (temp > 0)
        {
            b[j] = temp % 2;
            temp = temp / 2;
            j--;
        }
        while (j >= (i * 4))
        {
            b[j] = 0;
            j--;
        }
```

```
    }
}
```

- 下面定义了一个`P`盒函数，接受三个整数数组作为参数：`a`、`b`和`p`。
  - 执行置换操作，将输入数组`a`中的元素按照给定的置换表`p`的顺序重新排列，并将结果存储在数组`b`中。

```
void Permutation(int* a, int* b, int* p)
{
    for (int i = 0; i < 16; i++) b[p[i]] = a[i];
}
```

- 然后将以上功能串联到一块，编写了`SPN`加密函数。
  - 首先将输入的`x`数组值传给`w`作为初始值。
  - 接着进行三次循环。
    - 先计算出每一次循环需要的轮密钥。
    - 然后进行异或运算。
    - 接着分别进行`S`盒与`P`盒进行非线性处理。
  - 然后计算出下一轮的轮密钥、进行异或运算、`S`盒非线性处理。
  - 最后再计算轮密钥与异或运算，将结果保存到数组`y`中。

```
void SPN(int* x, int* y, int* s, int* p, int* key)
{
    int w[16], u[16], v[16], k[16];
    for (int i = 0; i < 16; i++) w[i] = x[i];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 16; j++) k[j] = key[4 * i + j];
        for (int j = 0; j < 16; j++) u[j] = w[j] ^ k[j];
        Substitution(u, v, s);
        Permutation(v, w, p);
    }
    for (int j = 0; j < 16; j++) k[j] = key[4 * 3 + j];
    for (int j = 0; j < 16; j++) u[j] = w[j] ^ k[j];
    Substitution(u, v, s);
    for (int j = 0; j < 16; j++) k[j] = key[4 * 4 + j];
    for (int j = 0; j < 16; j++) y[j] = v[j] ^ k[j];
}
```

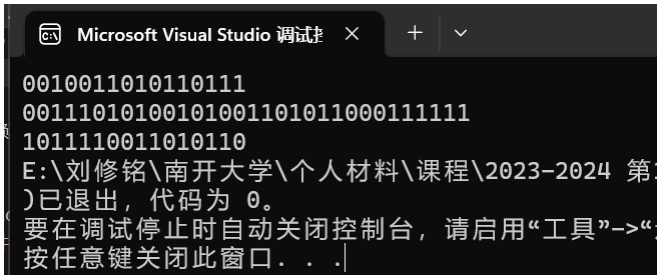
- 最后是`main`函数。
  - 读取输入字符串`X`和密钥字符串`Key`，将它们转换成整数数组`x`和`key`，然后调用之前提供的`SPN`函数来进行加密，并输出加密后的结果。

```cpp
int main()
{
    string X, Key;
    cin >> X >> Key;
    int len = X.length();
    for (int i = 0; i < len; i++) x[i] = X[i] - '0';
    len = Key.length();
    for (int i = 0; i < len; i++) key[i] = Key[i] - '0';
    SPN(x, y, S, P, key);
    for (int i = 0; i < 16; i++) cout << y[i];
    return 0;
}
```

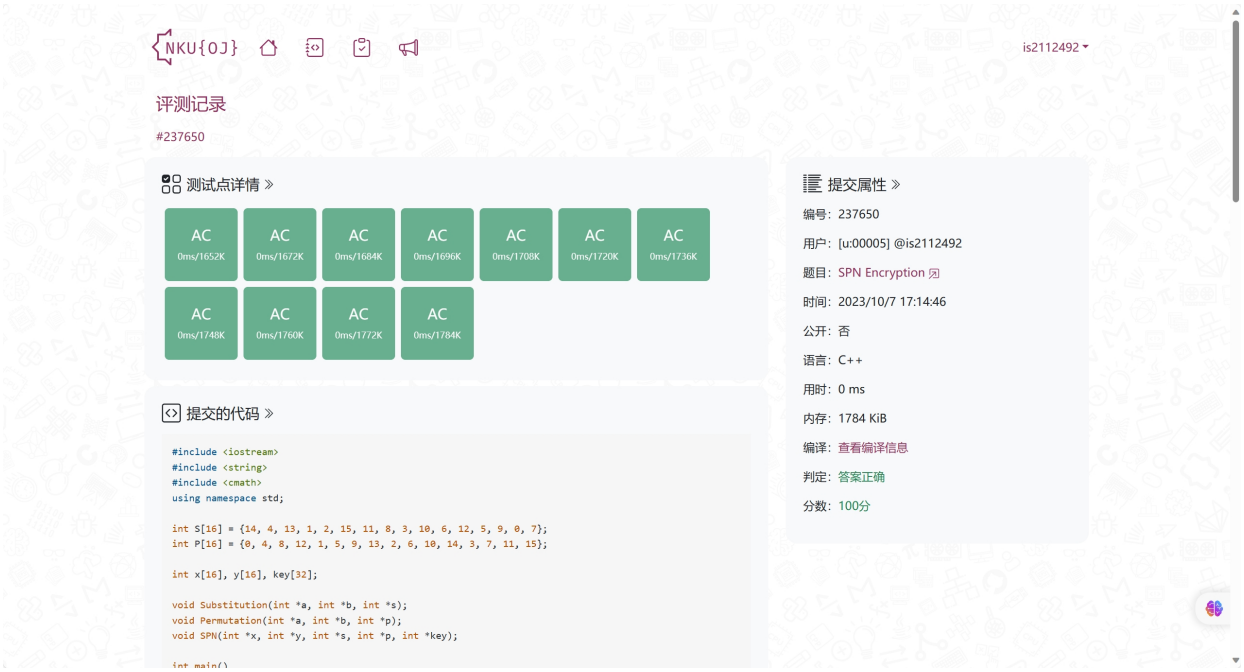
- SPN解密算法的过程通常是加密过程的逆操作。要实现SPN的解密算法，进行相关操作的逆操作即可实现，如逆置换、逆替代、逆异或等。

（二）运行结果

1. 测试样例



2. OJ系统



二、线性攻击算法

（一）代码文件

- SPN.ipynb
- SPN.cpp
- Liner.cpp
  - 不建议直接运行exe文件，会导致路径错误。

（一）线性攻击算法实现

按照给定伪代码实现算法即可。下面对核心代码进行说明：

- 随机生成16000个16位二进制串，将其放入前面得到的SPN加密算法中，以`0011101010010100110101100011111`为密钥进行加密得到密文，然后将明密文对写入`pairs.txt`文件，得到本次实验的数据集。该密钥K_5轮密钥为`1101011000111111`。

```python
import subprocess
import random
from tqdm import tqdm

def generate_random_binary_string(length=16):
    return ''.join(random.choice('01') for _ in range(length))

executable_file_path = "./exe/SPN.exe"
num_pairs = 16000

with open("./data/pairs.txt", "w") as file:
    for _ in tqdm(range(num_pairs), desc="Generating Pairs", unit="pair"):
        input_data = generate_random_binary_string()
        result = subprocess.run([executable_file_path],
input=input_data.encode(), stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        output_data = result.stdout.decode().strip()

        file.write(input_data + "\n")
        file.write(output_data + "\n")

print(f"Generated {num_pairs} pairs and saved to pairs.txt.")
```

- 全局部分定义了`S`盒、明文与密文数组及计数器。

```cpp
int S[16] = { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 };
int x[16] = {};//明文
int y[16] = {};//密文
int Count[16][16] = {};//计数器
```

- 接着定义了一个函数`DecimalToBinary`，用于将十进制数转换成四位二进制，并将其按照给定的索引位存入数组。

```cpp
void DecimalToBinary(int decimal, int* binary, int num)
{
    int i = num - 3;//最低索引位
    while (decimal > 0)
    {
        binary[num] = decimal % 2;
        decimal /= 2;
        num--;
    }
    while (num >= i)//空位补0
    {
        binary[num] = 0;
        num--;
    }
}
```

- 下面便是`main`函数部分。
  - 首先对`S`盒进行逆运算，存入数组`S1`。

```cpp
int S1[16] = {};//S盒的逆
for (int i = 0; i < 16; i++)
{
    S1[S[i]] = i;
}
```

  - 接着使用文件流读入已经得到的数据集。

```cpp
ifstream input(".\\data\\pairs.txt");
```

  - 然后遍历`(0,0) to (F,F)`，按照算法内容进行计算，得到对应的计数器。

```cpp
for (int j = 0; j < 16; j++)
{
    for (int k = 0; k < 16; k++)
    {
        //for(L_1,L_2) <- (0,0) to (F,F)
        DecimalToBinary(j, L1, 3);
        DecimalToBinary(k, L2, 3);

        //L_1与y_{<2>}异或
        v[4] = L1[0] ^ y[4];
        v[5] = L1[1] ^ y[5];
        v[6] = L1[2] ^ y[6];
        v[7] = L1[3] ^ y[7];
        //L_2与y_{<4>}异或
        v[12] = L2[0] ^ y[12];
        v[13] = L2[1] ^ y[13];
        v[14] = L2[2] ^ y[14];
        v[15] = L2[3] ^ y[15];
```

```
        int temp1 = v[4] * pow(2, 3) + v[5] * pow(2, 2) + v[6] * pow(2, 1) + v[7] * pow(2, 0);
        int temp2 = S1[temp1];//S盒的逆运算
        DecimalToBinary(temp2, u, 7);

        temp1 = v[12] * pow(2, 3) + v[13] * pow(2, 2) + v[14] * pow(2, 1) + v[15] * pow(2, 0);
        temp2 = S1[temp1];//S盒的逆运算
        DecimalToBinary(temp2, u, 15);

        int z = x[4] ^ x[6] ^ x[7] ^ u[5] ^ u[7] ^ u[13] ^ u[15];

        if (z == 0) Count[j][k]++;
    }
}
```

- 最后分析比较计数器的值，输出最大可能的轮密钥。

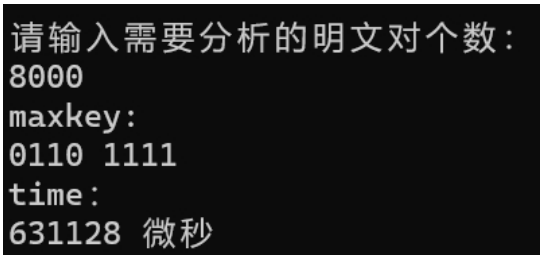
```
int max = -1;
int LL1 = 0, LL2 = 0;//记录最大的Count[i][j]对应的L1和L2
for (int i = 0; i < 16; i++)
{
    for (int j = 0; j < 16; j++)
    {
        Count[i][j] = abs(Count[i][j] - n / 2);
        if (Count[i][j] > max)
        {
            max = Count[i][j];
            LL1 = i;
            LL2 = j;
        }
    }
}

cout << "maxkey:" << endl;
DecimalToBinary(LL1, L1, 3);
for (int i = 0; i < 4; i++)
{
    cout << L1[i];
}
cout << " ";
DecimalToBinary(LL2, L2, 3);
for (int i = 0; i < 4; i++)
{
    cout << L2[i];
}
```

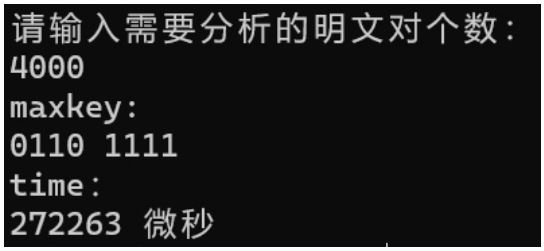
（二）密钥分析

该密钥K_5轮密钥为`1101011000111111`，则$L_1 = 0110$，$L_2 = 1111$。

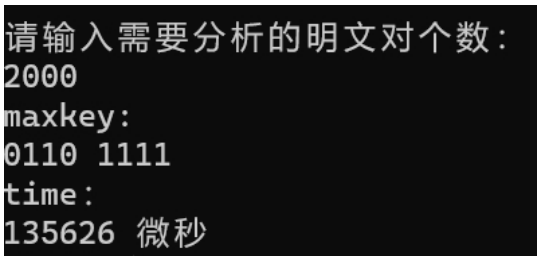
- 当使用8000对明密文对进行攻击时，可以看到输出结果正确，说明攻击成功，用时约631128微秒。



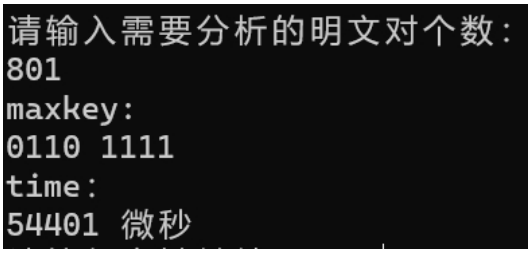
- 下面尝试减少明密文对数量，测试攻击成功率。
  - 使用4000对时也能得到正确结果，用时约为272263微秒。



  - 使用2000对时也能得到正确结果，用时约为135626微秒。



  - 经过测试，对于该组密钥，最少使用801组即可攻击成功（由于概率问题，每次攻击需要的数目可能会上下波动，该数值仅对下面图片负责）。



- 修改传入密钥，得到新的数据集，测试攻击成功率。
  - 密钥为
    - `00000000000000000000000000000000`
      - 其轮密钥为`00000000`
    - `11111111111111111111111111111111`
      - 其轮密钥为`11111111`
    - `01010101010101010101010101010101`
      - 其轮密钥为`01010101`
    - `10101010101010101010101010101010`
      - 其轮密钥为`10101010`
  - 经过测试，当明密文对数为4000时成功率已经足够高，以上五组攻击均能成功。

```
Key:00000000000000000000000000000000
maxkey:0000 0000
Key:11111111111111111111111111111111
maxkey:1111 1111
Key:01010101010101010101010101010101
maxkey:0101 0101
Key:10101010101010101010101010101010
maxkey:1010 1010
```

- 在2得到第2、4部分轮密钥后，对于剩余的八位轮密钥进行分析。
    - 选择新的线性分析链，在第2、4部分密钥已知的基础上分析出第1、3部分的密钥。接着在已知起始密钥低16位的基础上，穷举高16位密钥对给出的8000个明密文对进行验证。由于在加密过程中进行了5轮的S代换和P置换，导致相同明文在不同密钥下得到相同密文的概率极低，因此在实际验证过程中并不需要验证8000个明密文对是否对应，而仅需验证3个即可判断出密钥是否合适。
    - 由于虽然可以选取到仅包含第5轮第1、3部分的线性分析链，但是由于偏差不大，因此代表性较差，可能导致对于1、3部分密钥可能性较大部分的遍历过多，而导致时间开销较大。

三、差分攻击算法

为了进行效率对比，本人也实现了差分攻击算法。

（一）代码文件

- SPN.ipynb
- Filter.cpp
- Differential.cpp
    - 不建议直接运行exe文件，会导致路径错误。

（一）差分攻击算法实现

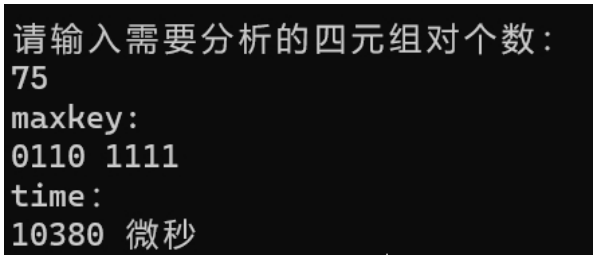
- 首先使用python随机生成16000个16位二进制数，并与`0000101100000000`进行异或计算，将结果保存至`xor.txt`中。
- 接着使用`SPN`算法对上面的二进制串进行加密并过滤得到满足$y_{<1>} = (y_{<1>})^* \ \&\& \ y_{<3>} = (y_{<3>})^*$的“正确对”,将结果保存至`xor_result.txt`中。

```
if (y1[0] == y2[0] && y1[1] == y2[1] && y1[2] == y2[2] && y1[3] == y2[3] &&
y1[8] == y2[8] && y1[9] == y2[9] && y1[10] == y2[10] && y1[11] == y2[11])
{
    for (int j = 0; j < 16; j++) output << x1[j];
    output << endl;
    for (int j = 0; j < 16; j++) output << y1[j];
    output << endl;
    for (int j = 0; j < 16; j++) output << x2[j];
    output << endl;
    for (int j = 0; j < 16; j++) output << y2[j];
    output << endl;
}
```

- 接着按照伪代码，完成算法即可。此处代码较为繁琐，不予以展示，完整代码请看附件。

（二）结果分析

- 当输入值为75时，可以看到攻击成功。



- 经过测试，差分攻击最少借助10组“正确对”即可实现攻击（由于概率问题，每次攻击需要的数目可能会上下波动，该数值仅对下面图片负责）。