

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) 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。

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

```
1 void substitution(int* a, int* b, int* s)
2 {
3     int T[4] = {0};
4     int k = 0;
5     for (int i = 0; i < 4; i++)
6     {
7         T[i] = a[i * 4] * pow(2, 3) + a[i * 4 + 1] * pow(2, 2) + a[i * 4 +
8         2] * pow(2, 1) + a[i * 4 + 3];
9         int temp = s[T[i]];
10        int j = (i * 4) + 3;
11        while (temp > 0)
12        {
13            b[j] = temp % 2;
14            temp = temp / 2;
15            j--;
16        }
17        while (j >= (i * 4))
18        {
19            b[j] = 0;
20            j--;
21        }
22    }
```

- 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。

```
1 void Permutation(int* a, int* b, int* p)
2 {
3     for (int i = 0; i < 16; i++) b[p[i]] = a[i];
4 }
```

- 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。

```
1 void SPN(int* x, int* y, int* s, int* p, int* key)
2 {
3     int w[16], u[16], v[16], k[16];
4     for (int i = 0; i < 16; i++) w[i] = x[i];
5     for (int i = 0; i < 3; i++)
6     {
7         for (int j = 0; j < 16; j++) k[j] = key[4 * i + j];
8         for (int j = 0; j < 16; j++) u[j] = w[j] ^ k[j];
9         Substitution(u, v, s);
10        Permutation(v, w, p);
11    }
12    for (int j = 0; j < 16; j++) k[j] = key[4 * 3 + j];
13    for (int j = 0; j < 16; j++) u[j] = w[j] ^ k[j];
14    Substitution(u, v, s);
15    for (int j = 0; j < 16; j++) k[j] = key[4 * 4 + j];
16    for (int j = 0; j < 16; j++) y[j] = v[j] ^ k[j];
17 }
```

- 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。

```

1  int main()
2  {
3      string X, Key;
4      cin >> X >> Key;
5      int len = x.length();
6      for (int i = 0; i < len; i++) x[i] = x[i] - '0';
7      len = Key.length();
8      for (int i = 0; i < len; i++) key[i] = Key[i] - '0';
9      SPN(x, y, S, P, key);
10     for (int i = 0; i < 16; i++) cout << y[i];
11     return 0;
12 }

```

- 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

```

Microsoft Visual Studio 调试
0010011010110111
00111010100101001101011000111111
1011110011010110
E:\刘修铭\南开大学\个人材料\课程\2023-2024 第
)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“
按任意键关闭此窗口。 . .

```

2. OJ系统

评测记录
#237650

测试点详情

AC 0ms/1652K	AC 0ms/1672K	AC 0ms/1684K	AC 0ms/1696K	AC 0ms/1708K	AC 0ms/1720K	AC 0ms/1736K
AC 0ms/1748K	AC 0ms/1760K	AC 0ms/1772K	AC 0ms/1784K			

提交的代码

```

#include <iostream>
#include <string>
#include <cmath>
using namespace std;

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];

void Substitution(int *a, int *b, int *s);
void Permutation(int *a, int *b, int *p);
void SPN(int *x, int *y, int *s, int *p, int *key);

int main()

```

提交属性

编号: 237650
 用户: [u:00005] @is2112492
 题目: SPN Encryption
 时间: 2023/10/7 17:14:46
 公开: 否
 语言: C++
 用时: 0 ms
 内存: 1784 KiB
 编译: 查看编译信息
 判定: 答案正确
 分数: 100分

二、线性攻击算法

(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00111010100101001101011000111111 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。

```
1 import subprocess
2 import random
3 from tqdm import tqdm
4
5 def generate_random_binary_string(length=16):
6     return ''.join(random.choice('01') for _ in range(length))
7
8 executable_file_path = "./exe/SPN.exe"
9 num_pairs = 16000
10
11 with open("./data/pairs.txt", "w") as file:
12     for _ in tqdm(range(num_pairs), desc="Generating Pairs", unit="pair"):
13         input_data = generate_random_binary_string()
14         result = subprocess.run([executable_file_path],
15                                 input=input_data.encode(), stdout=subprocess.PIPE, stderr=subprocess.PIPE)
16         output_data = result.stdout.decode().strip()
17
18         file.write(input_data + "\n")
19         file.write(output_data + "\n")
20
21 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。

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

```
1 void DecimalToBinary(int decimal, int* binary, int num)
2 {
3     int i = num - 3; //最低索引位
4     while (decimal > 0)
5     {
6         binary[num] = decimal % 2;
7         decimal /= 2;
```

```

8         num--;
9     }
10    while (num >= i)//空位补0
11    {
12        binary[num] = 0;
13        num--;
14    }
15 }

```

- 下面便是 main 函数部分。

- 首先对 s 盒进行逆运算，存入数组 s1。

```

1  int s1[16] = {};//S盒的逆
2  for (int i = 0; i < 16; i++)
3  {
4      s1[s[i]] = i;
5  }

```

- 接着使用文件流读入已经得到的数据集。

```

1  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。

```

1  for (int j = 0; j < 16; j++)
2  {
3      for (int k = 0; k < 16; k++)
4      {
5          //for(L_1,L_2) <- (0,0) to (F,F)
6          DecimalToBinary(j, L1, 3);
7          DecimalToBinary(k, L2, 3);
8
9          //L_1与y_{<2>}异或
10         v[4] = L1[0] ^ y[4];
11         v[5] = L1[1] ^ y[5];
12         v[6] = L1[2] ^ y[6];
13         v[7] = L1[3] ^ y[7];
14         //L_2与y_{<4>}异或
15         v[12] = L2[0] ^ y[12];
16         v[13] = L2[1] ^ y[13];
17         v[14] = L2[2] ^ y[14];
18         v[15] = L2[3] ^ y[15];
19
20         int temp1 = v[4] * pow(2, 3) + v[5] * pow(2, 2) + v[6] * pow(2,
1) + v[7] * pow(2, 0);
21         int temp2 = s1[temp1];//S盒的逆运算
22         DecimalToBinary(temp2, u, 7);
23
24         temp1 = v[12] * pow(2, 3) + v[13] * pow(2, 2) + v[14] * pow(2,
1) + v[15] * pow(2, 0);

```

```

25     temp2 = S1[temp1]; //S盒的逆运算
26     DecimalToBinary(temp2, u, 15);
27
28     int z = x[4] ^ x[6] ^ x[7] ^ u[5] ^ u[7] ^ u[13] ^ u[15];
29
30     if (z == 0) Count[j][k]++;
31 }
32 }

```

- 最后分析比较计数器的值，输出最大可能的轮密钥。

```

1  int max = -1;
2  int LL1 = 0, LL2 = 0; //记录最大的Count[i][j]对应的L1和L2
3  for (int i = 0; i < 16; i++)
4  {
5      for (int j = 0; j < 16; j++)
6      {
7          Count[i][j] = abs(Count[i][j] - n / 2);
8          if (Count[i][j] > max)
9          {
10             max = Count[i][j];
11             LL1 = i;
12             LL2 = j;
13         }
14     }
15 }
16
17 cout << "maxkey:" << endl;
18 DecimalToBinary(LL1, L1, 3);
19 for (int i = 0; i < 4; i++)
20 {
21     cout << L1[i];
22 }
23 cout << " ";
24 DecimalToBinary(LL2, L2, 3);
25 for (int i = 0; i < 4; i++)
26 {
27     cout << L2[i];
28 }

```

(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。

```
请输入需要分析的明文对个数：
8000
maxkey:
0110 1111
time:
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。

```
请输入需要分析的明文对个数：
4000
maxkey:
0110 1111
time:
272263 微秒
```

- 使用2000对时也能得到正确结果，用时约为135626微秒。

```
请输入需要分析的明文对个数：
2000
maxkey:
0110 1111
time:
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。

```
请输入需要分析的明文对个数：
801
maxkey:
0110 1111
time:
54401 微秒
```

- 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。