密码学课程实验报告

书面作业二

学　　院　网络空间安全学院

专　　业　信息安全

学　　号　2213041

姓　　名　李雅帆

班　　级　信安班

一、请利用线性密码分析确定 SPN 分组加密算法的（轮）密钥。

1. 得到 8000 对明文密文。

先随机生成 8000 条明文，再通过之前写的 SPN 加密算法（进行了一些修改）得到对应的密文，得到测试数据集 SPNtest.txt。

（1）随机生成 8000 条明文源码部分:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <bitset>
using namespace std;

int main() {
    ofstream outputFile("test.txt");
    srand(static_cast<unsigned int>(time(0)));
    for (int i = 0; i < 8000; i++) {
        // Generate a random 16-bit binary number
        bitset<16> randomBits(rand() % (1 << 16));
        outputFile << randomBits.to_string() << endl;
    }
    outputFile.close();
    return 0;
}
```

（2）得到对应密文源码部分：

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
using namespace std;

// S 替换
void Sbox(char A[]) {
    const char* B[] = {
        "0000", "0001", "0010", "0011",
        "0100", "0101", "0110", "0111",
        "1000", "1001", "1010", "1011",
```

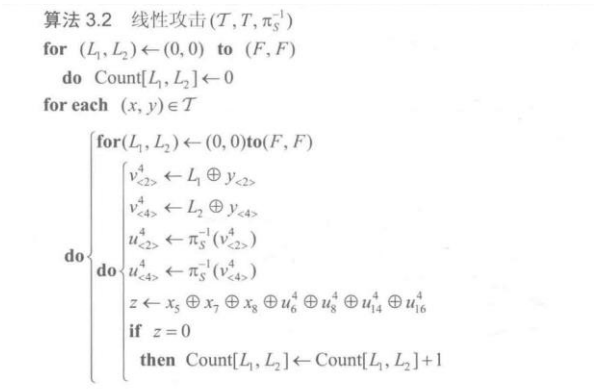
```
        "1100", "1101", "1110", "1111"
    };
    int sboxMap[] = { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 };
    char temp[5] = "";
    memcpy(temp, A, 4);
    temp[4] = '\0';

    for (int i = 0; i < 16; i++) {
        if (strncmp(temp, B[i], 4) == 0) {
            memcpy(A, B[sboxMap[i]], 4);
            A[4] = '\0';
            return;
        }
    }
}

// P 置换
void Pbox(char A[]) {
    char P[17];
    memcpy(P, A, 16);
    P[16] = '\0';
    A[0] = P[0];  A[1] = P[4];  A[2] = P[8];  A[3] = P[12];
    A[4] = P[1];  A[5] = P[5];  A[6] = P[9];  A[7] = P[13];
    A[8] = P[2];  A[9] = P[6];  A[10] = P[10];  A[11] = P[14];
    A[12] = P[3];  A[13] = P[7];  A[14] = P[11];  A[15] = P[15];
    A[16] = '\0';
}

// 异或
void XOR(char A[], const char K[]) {
    for (int i = 0; i < 16; i++) {
        A[i] = ((A[i] - '0') ^ (K[i] - '0')) + '0';
    }
    A[16] = '\0';
}

// SPN 加密
void SPN(char x[], const char keys[][17], int Nr) {
    char state[17];
    memcpy(state, x, 16);
    state[16] = '\0';

    // 前 Nr-1 轮加密
    for (int round = 0; round < Nr - 1; round++) {
```

```
        // 与密钥异或
        XOR(state, keys[round]);
        // S 盒替换
        for (int i = 0; i < 16; i += 4) {
            char temp[5];
            memcpy(temp, &state[i], 4);
            temp[4] = '\0';
            Sbox(temp);
            memcpy(&state[i], temp, 4);
        }
        // 置换
        Pbox(state);
    }

    // 第 Nr 轮
    XOR(state, keys[Nr - 1]);

    // 最终的 S 盒替换
    for (int i = 0; i < 16; i += 4) {
        char temp[5];
        memcpy(temp, &state[i], 4);
        temp[4] = '\0';
        Sbox(temp);
        memcpy(&state[i], temp, 4);
    }
    // 最后与最终轮密钥异或
    XOR(state, keys[Nr]);
    // 输出加密结果
    cout << state << endl;
}

int main() {
    const char key[33] = "00111010100101001101011000111111";
    char k[33];
    memcpy(k, key, 32);
    k[32] = '\0';

    // 生成轮密钥 (4 轮，每轮 16 位)
    char keys[5][17];
    for (int i = 0; i < 5; i++) {
        memcpy(keys[i], &k[i * 4], 16);
        keys[i][16] = '\0';
    }
```

```
    ifstream inputFile("D:/fan/homework3/test.txt");
    ofstream outputFile("SPNtest.txt");
    string plaintext;

    while (getline(inputFile, plaintext)) {
        plaintext.resize(16, '0');
        char x[17];
        memcpy(x, plaintext.c_str(), 16);
        x[16] = '\0';

        SPN(x, keys, 4);
        outputFile << plaintext << endl << x << endl;
    }

    inputFile.close();
    outputFile.close();
    return 0;
}
```

(3)对测试数据集进行线性密码分析，尝试得到轮密钥。

①伪代码



算法 3.2　线性攻击 $(\mathcal{T}, T, \pi_S^{-1})$

```
for (L1, L2) ← (0,0) to (F, F)
  do Count[L1, L2] ← 0
for each (x, y) ∈ T
  do { for (L1, L2) ← (0,0) to (F, F)
       do { v4<2> ← L1 ⊕ y<2>
            v4<4> ← L2 ⊕ y<4>
            u4<2> ← π_S^{-1}(v4<2>)
            u4<4> ← π_S^{-1}(v4<4>)
            z ← x5 ⊕ x7 ⊕ x8 ⊕ u4_6 ⊕ u4_8 ⊕ u4_14 ⊕ u4_16
            if z = 0
              then Count[L1, L2] ← Count[L1, L2] + 1
```



```
max ← −1
for (L1, L2) ← (0,0) to (F, F)
  do { Count[L1, L2] ← |Count[L1, L2] − T/2|
       if Count[L1, L2] > max
         then { max ← Count[L1, L2]
                max key ← (L1, L2)
output(maxkey)
```

L1 和 L2 是密钥的两个部分，Count[L1, L2]是一个用于存储统

计计数的表格。这里通过遍历所有可能的密钥组合 (L1, L2)，为每个可能的密钥对 (L1, L2) 准备一个计数器，用于后续统计。

遍历所有的明文密文对 (x, y)，T 表示收集到的明文和对应的密文对的集合。v1 和 v2 是通过将密钥部分 L1 和 L2 与密文的一部分进行异或操作得到的中间值；u1 和 u2 是对中间值 v1 和 v2 应用逆 S 盒得到的值。

将明文部分 x5, x7, x8 和密钥相关部分 u1, u2 及其他的中间值通过异或运算得到结果 z。如果 z 的结果为 0，则认为该密钥猜测 (L1, L2) 是正确的，增加 Count[L1, L2] 的值。对每一组 (L1, L2) 密钥猜测，统计多少次能得到 z = 0。

初始化 max 为 -1，遍历所有可能的密钥对 (L1, L2)。对每一个密钥组合 (L1, L2)，计算 |Count[L1, L2] - T/2|。

T 是明文密文对的总数，T/2 是期望的统计值，越接近这个值，说明该密钥猜测越有可能。如果 Count[L1, L2] 超过当前最大值 max，则更新最大值 max 并记录当前的密钥对 (L1, L2) 为 maxkey。

②代码部分

```
int BinToDec(int n[4])
{
    int s = 8 * n[0] + 4 * n[1] + 2 * n[2] + n[3];
    return s;
}

const char* B[] =
{
    "0000", "0001", "0010", "0011",
    "0100", "0101", "0110", "0111",
    "1000", "1001", "1010", "1011",
    "1100", "1101", "1110", "1111"
```

```
};
void DecToBin(int n, int result[4])
{
    const char* binaryString = B[n];
    for (int i = 0; i < 4; i++)
    {
        result[i] = binaryString[i] - '0';
    }
}
```

BinToDec 函数将 4 位的二进制数组 n[4] 转换成相应的十进制整数；DecToBin 函数的作用是将一个十进制数 n 转换为一个 4 位二进制数组。

```
void Inverse(int middle[4], int mmiddle[4])
{
    int l = BinToDec(middle);
    int index = -1;
    for (int p = 0; p < 16; p++)
    {
        if (sbox[p] == l)
        {
            index = p;
            break;
        }
    }
    if (index != -1)
    {
        DecToBin(index, mmiddle);
    }
    else
    {
        for (int i = 0; i < 4; i++)
        {
            mmiddle[i] = 0;
        }
    }
}
```

Inverse 函数实现了 S 盒的逆变换。middle[4] 表示输入的 4 位二进制数，首先将其转换为十进制数，然后在 sbox[] 中找到其对应

的逆映射索引 index。最后再将找到的 index 转换为二进制形式存入 mmiddle[4]。

```
void read()
{
    ifstream in("D:/fan/homework3/SPNtest.txt");
    if (!in) {
        cerr << "无法打开文件!" << endl;
        return;
    }
    for (int j = 0; j < T; j++)
    {
        char xx[17], yy[17];
        for (int i = 1; i < 17; i++)
        {
            in >> xx[i];
            X[j][i] = xx[i] - '0';
        }
        for (int i = 1; i < 17; i++)
        {
            in >> yy[i];
            Y[j][i] = yy[i] - '0';
        }
    }
    in.close();
}
```

read 函数负责读取文件 SPNtest.txt，并将文件中的内容读入到二维数组 X 和 Y 中，每个 X[j][i] 和 Y[j][i] 都是一个长度为 16 的二进制向量。

```
void SPNLine()
{
    for (int l = 0; l < T; l++)
    {
        for (int i = 1; i <= 16; i++)
        {
            x[i] = X[l][i];
            y[i] = Y[l][i];
        }
        for (int i = 1; i <= 16; i++)
        {
```

```
            DecToBin(i - 1, L1);
            for (int j = 1; j <= 16; j++)
            {
                DecToBin(j - 1, L2);
                for (int k = 1; k <= 4; k++)
                {
                    v[4 + k] = L1[k - 1] ^ y[4 + k];
                }
                for (int k = 1; k <= 4; k++)
                {
                    v[12 + k] = L2[k - 1] ^ y[12 + k];
                }
                for (int a = 5; a <= 8; a++)
                {
                    vteam2[a - 5] = v[a];
                }
                for (int b = 13; b <= 16; b++)
                {
                    vteam4[b - 13] = v[b];
                }
                Inverse(vteam2, uteam2);
                Inverse(vteam4, uteam4);
                int z = x[5] ^ x[7] ^ x[8] ^ uteam2[1] ^ uteam2[3] ^ uteam4[1] ^
uteam4[3];
                if (z == 0)
                {
                    Count[i][j] += 1;
                }
            }
        }
    }
    int max = -1;
    for (int i = 1; i <= 16; i++)
    {
        DecToBin(i - 1, L1);
        for (int j = 1; j <= 16; j++)
        {
            DecToBin(j - 1, L2);
            Count[i][j] = abs(Count[i][j] - T / 2);
            if (Count[i][j] > max)
            {
                max = Count[i][j];
                for (int l = 0; l < 4; l++)
                {
```

```
                    K[1] = L1[1];
                    K[4 + 1] = L2[1];
                }
            }
        }
    }
    for (int i = 0; i < 8; i++)
    {
        cout << K[i];
    }
    cout << endl;
}
```

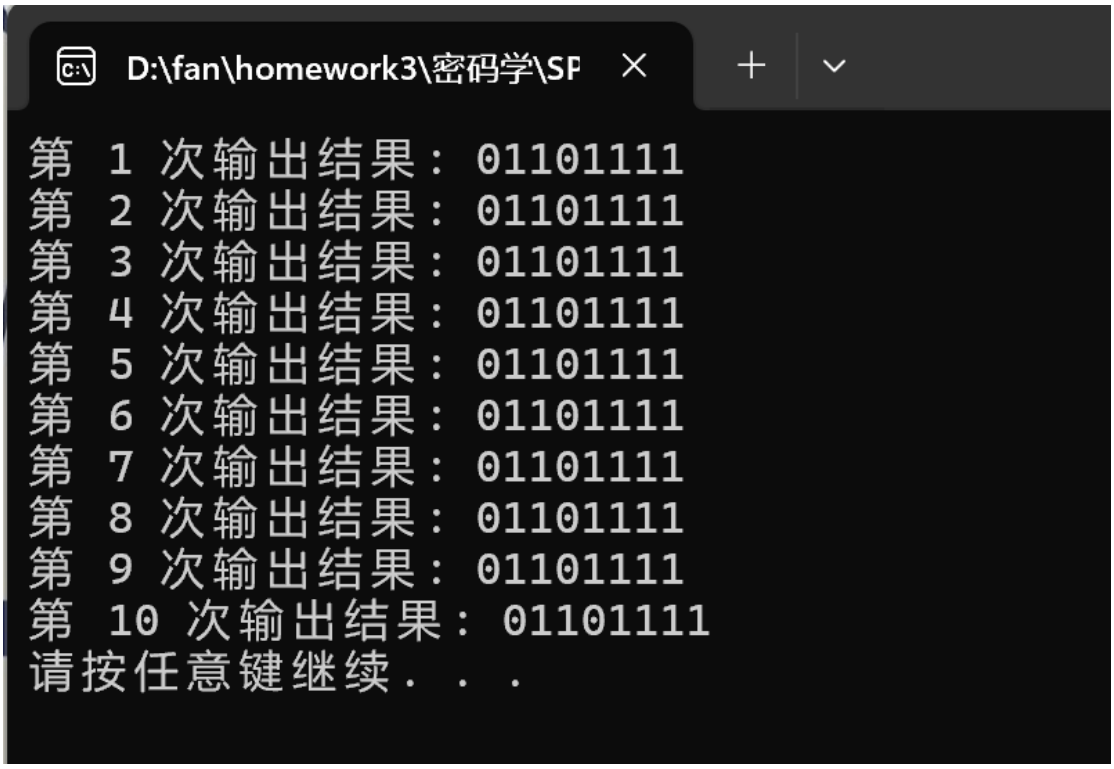
SPNLine 函数负责进行线性密码分析。对于每个明文和密文对，尝试不同的键猜测，通过 S 盒的逆变换来检查猜测密钥是否正确，记录出现的情况并统计次数，最终选择出现次数最多的猜测作为密钥，并输出。

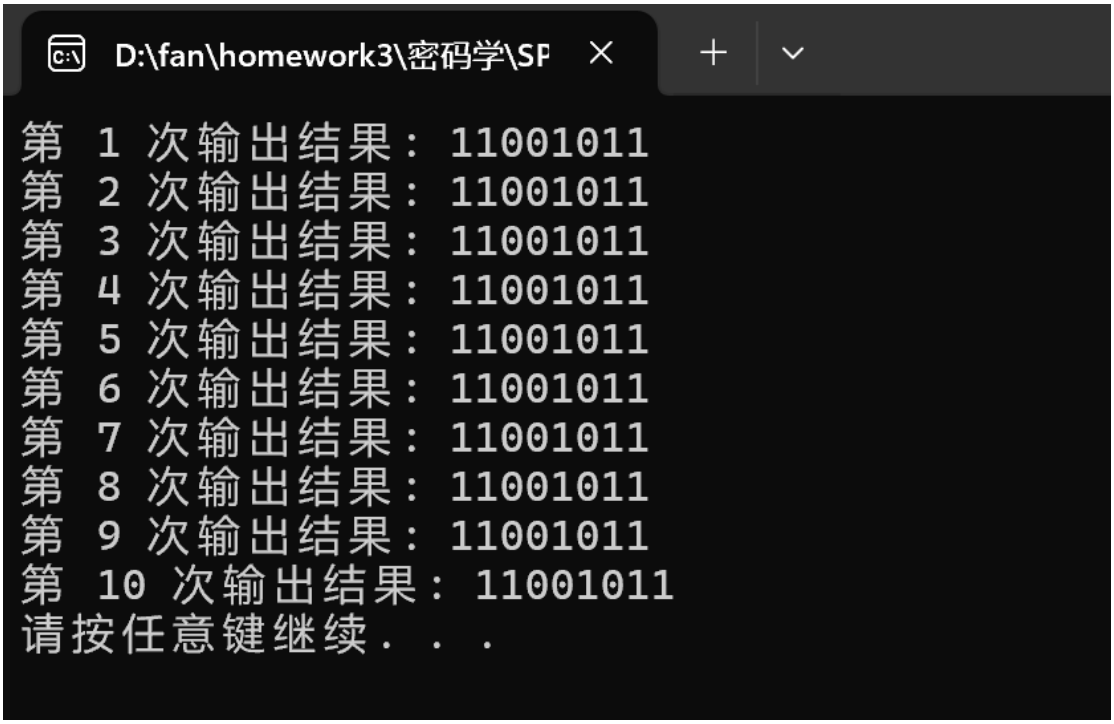
```
int main()
{
    read();

    for (int run = 1; run <= 10; run++)
    {

        for (int i = 1; i <= 16; i++)
        {
            for (int j = 1; j <= 16; j++)
            {
                Count[i][j] = 0;
            }
        }
        cout << "第 " << run << " 次输出结果: ";
        SPNLine();
    }
    system("pause");
    return 0;
}
```

main 函数进行 10 次运行，每次重置计数数组 Count，并调用 SPNLine 进行线性分析，打印每次攻击的结果。

③结果

用不同的密钥得到了两个测试集，进行了两次攻击，均得到了正确的结果。





二、请计算国密分组加密算法 SM4 的 SBox 差分分析表。

S 盒中数据均采用 16 进制表示。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

例：输入‘ef’，则经 S 盒后的值为表中第 e 行和第 f 列的值，*Sbox*(‘ef’)= ‘84’。

1.源代码

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

const int SBox[16][16] = {
    {0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2, 0x28,
0xfb, 0x2c, 0x05},
    {0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26, 0x49,
0x86, 0x06, 0x99},
    {0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43, 0xed,
0xcf, 0xac, 0x62},
    {0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa, 0x75,
0x8f, 0x3f, 0xa6},
    {0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19, 0xe6,
0x85, 0x4f, 0xa8},
    {0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b, 0x70,
0x56, 0x9d, 0x35},
    {0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b, 0x01,
```

```
0x21, 0x78, 0x87},
    {0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7, 0xa0,
0xc4, 0xc8, 0x9e},
    {0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce, 0xf9,
0x61, 0x15, 0xa1},
    {0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30, 0xf5,
0x8c, 0xb1, 0xe3},
    {0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab, 0x0d,
0x53, 0x4e, 0x6f},
    {0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72, 0x6d,
0x6c, 0x5b, 0x51},
    {0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41, 0x1f,
0x10, 0x5a, 0xd8},
    {0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12, 0xb8,
0xe5, 0xb4, 0xb0},
    {0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09, 0xc5,
0x6e, 0xc6, 0x84},
    {0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e, 0xd7,
0xcb, 0x39, 0x48}
};

 void Differ() {
    const int SBoxSize = 256;
    int DDT[SBoxSize][SBoxSize] = { 0 };

    int flatSBox[SBoxSize];
    for (int i = 0; i < 16; ++i)
        for (int j = 0; j < 16; ++j)
            flatSBox[i * 16 + j] = SBox[i][j];

    for (int x1 = 0; x1 < SBoxSize; ++x1) {
        for (int deltaX = 0; deltaX < SBoxSize; ++deltaX) {
            int x2 = x1 ^ deltaX;
            int deltaY = flatSBox[x1] ^ flatSBox[x2];
            DDT[deltaX][deltaY]++;
        }
    }

    ofstream outputFile("Difference.txt");
    outputFile << "差分分析表" << endl;
    for (int i = 0; i < SBoxSize; ++i) {
        for (int j = 0; j < SBoxSize; ++j) {
            outputFile << setw(3) << DDT[i][j] << " ";
        }
```

```
        outputFile << "\n";
    }
    outputFile.close();
}
int main() {
    Differ();
    return 0;
}
```

（1）定义 S-Box：使用一个 16x16 的二维数组存储 S-Box，包含 256 个 16 进制值。

（2）初始化差分分布表：创建一个 256x256 的二维数组 DDT，用来记录每对输入差分 deltaX 和输出差分 deltaY 的出现次数，初始化为 0。

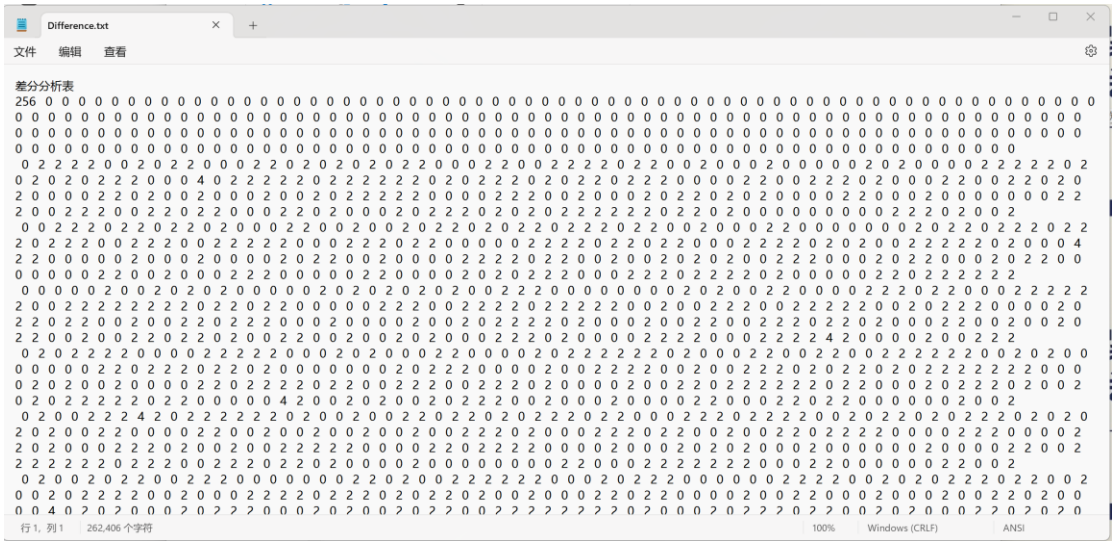
（3）扁平化 S-Box：将二维的 S-Box 转换为一维数组 flatSBox。

（4）计算差分分布表：使用嵌套循环遍历所有可能的输入差分 deltaX 和输入值 x1。对于每一对(x1, deltaX)，先计算 x2 为 x1 与 deltaX 的异或（XOR）结果，再使用扁平化的 S-Box 获取 flatSBox[x1]和 flatSBox[x2]，计算输出差分 deltaY，最后在 DDT 中对应的位置 DDT[deltaX][deltaY]的值加 1，记录该输入输出差分组合的出现次数。

（5）在 Difference.txt 中输出结果。

2. 结果

输出一个 256×256 的矩阵，这里截图一部分，Difference.txt 文件中是输出结果，已经发送到邮箱。


Difference.txt
文件 编辑 查看
差分分析表
256
行 1，列 1
262,406 个字符
100%
Windows (CRLF)
ANSI