第三次编程作业实验报告

刘曼姝 1901210656 2019.11.5

一、 实验目的与要求：

计算 ZUC 的 S0 与 S1 盒的差分分布表（DDT）与线性近似表（LAT），并回答为什么形成密文需要最后一轮的密钥混合（可以 Basic SPN 为例）。

二、 实验设备、系统环境与软件：

Windows 10 64 位操作系统的笔记本电脑，安装了 Visual Studio 2019。

三、 实验原理：

1、 ZUC 算法的 S0 与 S1 盒[1]

ZUC 即祖冲之序列密码算法，是中国自主设计的加密算法，其逻辑上分为上中下层，见图 1。上层是 16 级线性反馈移位寄存器（LFSR）；中层是比特重组（BR）；下层是非线性函数 F。

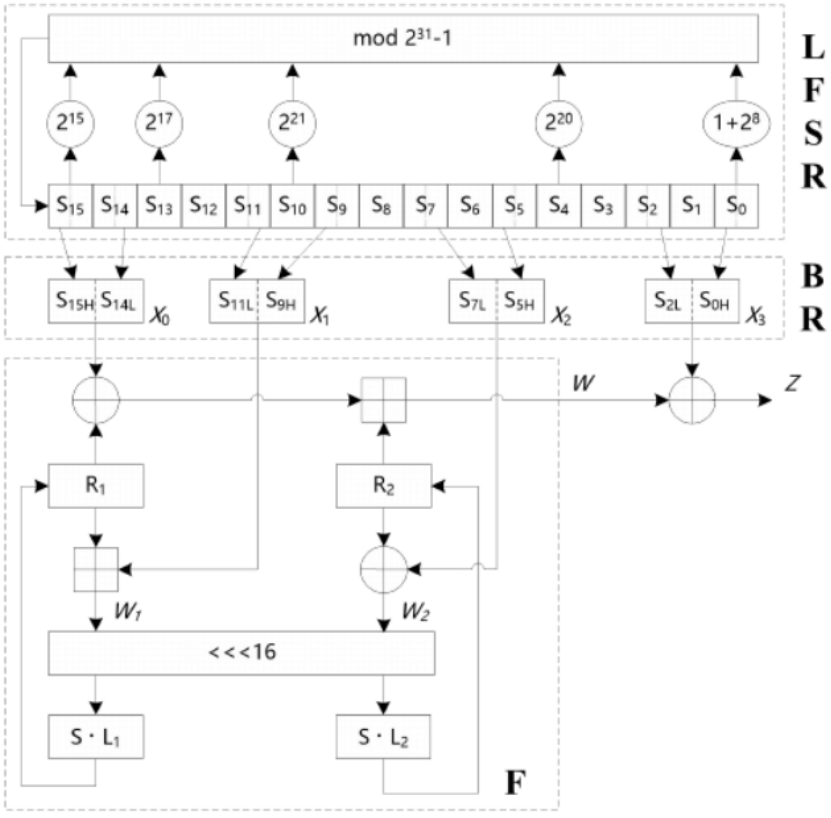


图 1 祖冲之算法结构图

其中，在非线性函数 F 部分，有 32bit 的 S 盒变换。32bit 的 S 盒由 4 个小的 8x8 的 S 盒并置而成，即 S=(S0, S1, S2, S3)，其中 S0=S2，S1=S3。S0 和 S1 的定义分别见表 A.1 和表 A.2。设 S0 或 S1 的 8bit 输入为 x。将 x 视作两个 16 进制数的连接，即 x=h||l，则表 A.1 或表 A.2 中第 h 行第 l 列交叉的元素即为 S0 或 S1 的输出 S0(x)或 S1(x)。设 S 盒的 32bit 输入 X 和 32bit 输出 Y 分别为：$X=x_0||x_1||x_2||x_3$，$Y=y_0||y_1||y_2||y_3$，其中 x_i 和 y_i 均为 8bit 字节，i=0, 1, 2, 3，则有 $y_i=Si(x_i)$，i=0, 1, 2, 3。

表 A.1 S_0 盒

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	3E	72	5B	47	CA	E0	00	33	04	D1	54	98	09	B9	6D	CB
1	7B	1B	F9	32	AF	9D	6A	A5	B8	2D	FC	1D	08	53	03	90
2	4D	4E	84	99	E4	CE	D9	91	DD	B6	85	48	8B	29	6E	AC
3	CD	C1	F8	1E	73	43	69	C6	B5	BD	FD	39	63	20	D4	38
4	76	7D	B2	A7	CF	ED	57	C5	F3	2C	BB	14	21	06	55	9B
5	E3	EF	5E	31	4F	7F	5A	A4	0D	82	51	49	5F	BA	58	1C
6	4A	16	D5	17	A8	92	24	1F	8C	FF	D8	AE	2E	01	D3	AD
7	3B	4B	DA	46	EB	C9	DE	9A	8F	87	D7	3A	80	6F	2F	C8
8	B1	B4	37	F7	0A	22	13	28	7C	CC	3C	89	C7	C3	96	56
9	07	BF	7E	F0	0B	2B	97	52	35	41	79	61	A6	4C	10	FE
A	BC	26	95	88	8A	B0	A3	FB	C0	18	94	F2	E1	E5	E9	5D
B	D0	DC	11	66	64	5C	EC	59	42	75	12	F5	74	9C	AA	23
C	0E	86	AB	BE	2A	02	E7	67	E6	44	A2	6C	C2	93	9F	F1
D	F6	FA	36	D2	50	68	9E	62	71	15	3D	D6	40	C4	E2	0F
E	8E	83	77	6B	25	05	3F	0C	30	EA	70	B7	A1	E8	A9	65
F	8D	27	1A	DB	81	B3	A0	F4	45	7A	19	DF	EE	78	34	60

表 A.2 S_1 盒

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	55	C2	63	71	3B	C8	47	86	9F	3C	DA	5B	29	AA	FD	77
1	8C	C5	94	0C	A6	1A	13	00	E3	A8	16	72	40	F9	F8	42
2	44	26	68	96	81	D9	45	3E	10	76	C6	A7	8B	39	43	E1
3	3A	B5	56	2A	C0	6D	B3	05	22	66	BF	DC	0B	FA	62	48
4	DD	20	11	06	36	C9	C1	CF	F6	27	52	BB	69	F5	D4	87
5	7F	84	4C	D2	9C	57	A4	BC	4F	9A	DF	FE	D6	8D	7A	EB
6	2B	53	D8	5C	A1	14	17	FB	23	D5	7D	30	67	73	08	09
7	EE	B7	70	3F	61	B2	19	8E	4E	E5	4B	93	8F	5D	DB	A9
8	AD	F1	AE	2E	CB	0D	FC	F4	2D	46	6E	1D	97	E8	D1	E9
9	4D	37	A5	75	5E	83	9E	AB	82	9D	B9	1C	E0	CD	49	89

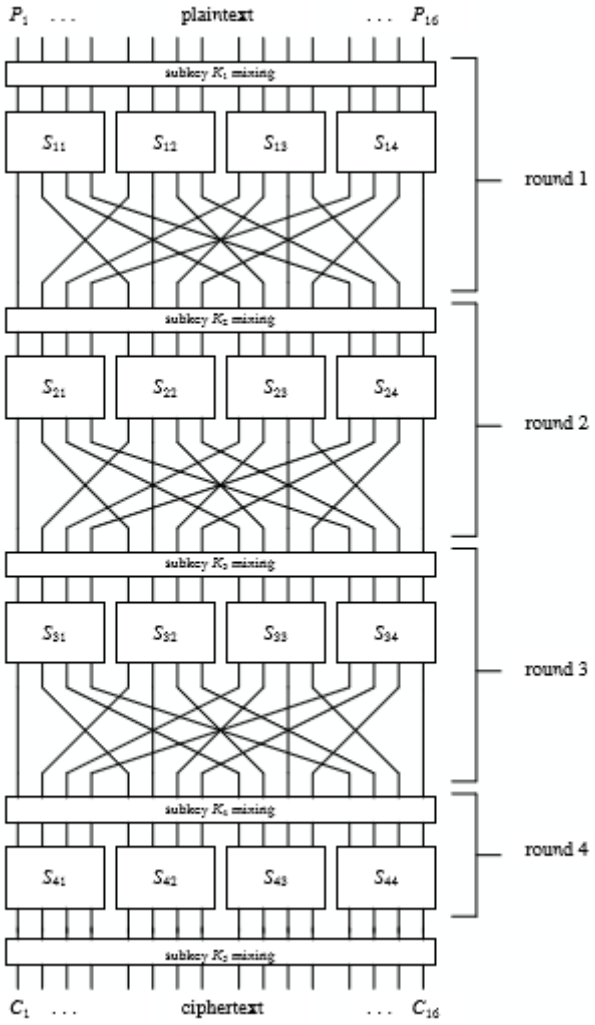
表 A.2 S_1 盒（续）

A	01	B6	BD	58	24	A2	5F	38	78	99	15	90	50	B8	95	E4
B	D0	91	C7	CE	ED	0F	B4	6F	A0	CC	F0	02	4A	79	C3	DE
C	A3	EF	EA	51	E6	6B	18	EC	1B	2C	80	F7	74	E7	FF	21
D	5A	6A	54	1E	41	31	92	35	C4	33	07	0A	BA	7E	0E	34
E	88	B1	98	7C	F3	3D	60	6C	7B	CA	D3	1F	32	65	04	28
F	64	BE	85	9B	2F	59	8A	D7	B0	25	AC	AF	12	03	E2	F2

注：S_0 盒和 S_1 盒数据均为十六进制表示。

2、 Basic SPN[2]

SPN（substitution-permutation network），即“代换-置换网络”，过程如下图所示，它是一类特殊的迭代密码。代换-置换网络的轮函数包括三个变换：代换、置换、密钥混合。需要注意的是，其最后一轮需要密钥混合，即在 4 轮的 Basic SPN 中，共需要 5 次密钥混合。

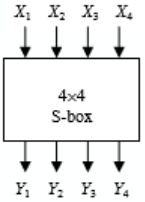


3、 差分分布表 DDT

在差分密码分析中，对于输入为 $X = [X_1\ X_2 \ldots X_n]$，输出为 $Y = [Y_1\ Y_2 \ldots Y_n]$ 的分组密码，设 $\Delta X = [\Delta X_1\ \Delta X_2 \ldots \Delta X_n]$，$\Delta Y = [\Delta Y_1\ \Delta Y_2 \ldots \Delta Y_n]$，其中 $\Delta X = X' \oplus X''$（X_i'和 X_i''代表 X′和 X″的第 i 位），$\Delta Yi = Yi' \oplus Yi''$（同理），称（ΔX，ΔY）为差分。在理想的随机密码中，在给定特定输入差 ΔX 的情况下发生特定输出差 ΔY 的概率为 $1/2^n$，其中 n 是 X 的位数，而差分密码分析就是利用某些明文差分的高概率，并将差分引入密码的最后一轮。差分密码分析属于选择明文攻击（CPA），攻击者需要选择 X'和 X''= X'+ΔX。

为了方便差分分析，引入差分分布表 DDT，其第 A+1 行代表输入差分 $\Delta X = X' \oplus X'' = A$、第 B+1 列代表输出差分 $\Delta Y = Y' \oplus Y'' = B$、第 A+1 行与第 B+1 列相交位置的值代表 $\Delta X = X' \oplus X'' = A$ 时，$\Delta Y = Y' \oplus Y'' = B$ 出现的频次。根据上文可以推出，越理想的随机密码，DDT 中应该包含越多的 0 值；而 DDT 中的非 0 值则表示存在某些明文差分对应的密文差分是高概率的，可能会被攻击者利用。

例：Basic SPN 的 DDT：



input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Table 1. S-box Representation (in hexadecimal)

X	*Y*
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

		Output Difference															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Difference	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
	2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
	3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
	4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
	5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
	6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
	7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
	8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
	9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
	A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
	B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
	C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
	D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
	E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
	F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

4、 线性近似表 LAT

在线性密码分析中，要确定出现概率较高或较低的如下形式的表达式：

$$X_{i_1} \oplus X_{i_2} \oplus ... \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus ... \oplus Y_{j_v} = 0$$

此表达式表示了 u 个输入 bit 和 v 个输出 bit 的“异或和”，从堆积原理得出，要进行线性密码分析，就需要计算（每种）该形式表达式出现概率与 1/2 的差（即线性概率偏差，可正可负），若存在较高的线性概率偏差，则说明输入输出之间以较高的概率具有线性特征。线性密码分析属于已知明文攻击（KPA），攻击者需要具有一组明文-密文对。

为了方便线性分析，引入线性近似表 LAT，其第 A+1 行代表 u 个分析的输入 bit 均为 1 时有 $X_{i1} \oplus X_{i2} \oplus \cdots \oplus X_{iu}$=A、第 B+1 列代表 v 个分析的输出 bit 均为 1 时有 $Y_{j1} \oplus Y_{j2} \oplus \cdots \oplus Y_{jv}$=B、第 A+1 行与第 B+1 列相交位置的值代表在每个输入/输出 bit 变化时满足表达式：

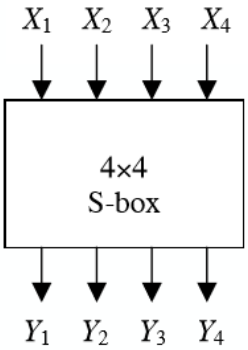
$$X_{i_1} \oplus X_{i_2} \oplus ... \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus ... \oplus Y_{j_v} = 0$$

的情况出现的频次与256/2=128的差值。根据上文可以推出，越理想的随机密码，LAT 中应该包含越多的 0 值；而 LAT 中的非 0 值则表示存在某些如下形式的表达式：

$$X_{i_1} \oplus X_{i_2} \oplus \ldots \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus \ldots \oplus Y_{j_v} = 0$$

出现概率较高或较低，可能会被攻击者利用。

例：Basic SPN 的 LAT：



$$X_2 \oplus X_3 = Y_1 \oplus Y_3 \oplus Y_4 .$$

X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4	$X_2 \oplus X_3$	$Y_1 \oplus Y_3 \oplus Y_4$	$X_1 \oplus X_4$	Y_2	$X_3 \oplus X_4$	$Y_1 \oplus Y_4$
0	0	0	0	1	1	1	0	0	0	0	1	0	1
0	0	0	1	0	1	0	0	0	0	1	1	1	0
0	0	1	0	1	1	0	1	1	0	0	1	1	0
0	0	1	1	0	0	0	1	1	1	1	0	0	1
0	1	0	0	0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	1	1	1	1	1	1	1	1	0
0	1	1	0	1	0	1	1	0	1	0	0	1	0
0	1	1	1	1	0	0	0	0	1	1	0	0	1
1	0	0	0	0	0	1	1	0	0	1	0	0	1
1	0	0	1	1	0	1	0	0	0	0	0	1	1
1	0	1	0	0	1	1	0	1	1	1	1	1	0
1	0	1	1	1	1	0	0	1	1	0	1	0	1
1	1	0	0	0	1	0	1	1	1	1	1	0	1
1	1	0	1	1	0	0	1	1	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	1	1	0	1	1	1	0	0	0	1	0	1

X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4	$X_2 \oplus X_3$	$Y_1 \oplus Y_3 \oplus Y_4$	$X_1 \oplus X_4$	Y_2	$X_3 \oplus X_4$	$Y_1 \oplus Y_4$
0	0	0	0	1	1	1	0	0	0	0	1	0	1
0	0	0	1	0	1	0	0	0	0	1	1	1	0
0	0	1	0	1	1	0	1	1	0	0	1	1	0
0	0	1	1	0	0	0	1	1	1	1	0	0	1
0	1	0	0	0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	1	1	1	1	1	1	1	1	0
0	1	1	0	1	0	1	1	0	1	0	0	1	0
0	1	1	1	1	0	0	0	0	1	1	0	0	1
1	0	0	0	0	0	1	1	0	0	1	0	0	1
1	0	0	1	1	0	1	0	0	0	0	0	1	1
1	0	1	0	0	1	1	0	1	1	1	1	1	0
1	0	1	1	1	1	0	0	1	1	0	1	0	1
1	1	0	0	0	1	0	1	1	1	1	1	0	1
1	1	0	1	1	0	0	1	1	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	1	1	0	1	1	1	0	0	0	1	0	1

Table 3. Sample Linear Approximations of S-box

Input Sum	Output Sum 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	−2	−2	0	0	−2	+6	+2	+2	0	0	+2	+2	0	0
2	0	0	−2	−2	0	0	−2	−2	0	0	+2	+2	0	0	−6	+2
3	0	0	0	0	0	0	0	0	+2	−6	−2	−2	+2	+2	−2	−2
4	0	+2	0	−2	−2	−4	−2	0	0	−2	0	+2	+2	−4	+2	0
5	0	−2	−2	0	−2	0	+4	+2	−2	0	−4	+2	0	−2	−2	0
6	0	+2	−2	+4	+2	0	0	+2	0	−2	+2	+4	−2	0	0	−2
7	0	−2	0	+2	+2	−4	+2	0	−2	0	+2	0	+4	+2	0	+2
8	0	0	0	0	0	0	0	0	−2	+2	+2	−2	+2	−2	−2	−6
9	0	0	−2	−2	0	0	−2	−2	−4	0	−2	+2	0	+4	+2	−2
A	0	+4	−2	+2	−4	0	+2	−2	+2	+2	0	0	+2	+2	0	0
B	0	+4	0	−4	+4	0	+4	0	0	0	0	0	0	0	0	0
C	0	−2	+4	−2	−2	0	+2	0	+2	0	+2	+4	0	+2	0	−2
D	0	+2	+2	0	−2	+4	0	+2	−4	−2	+2	0	+2	0	0	+2
E	0	+2	+2	0	−2	−4	0	+2	−2	0	0	−2	−4	+2	−2	0
F	0	−2	−4	−2	−2	0	+2	0	0	−2	+4	−2	−2	0	+2	0

Table 4. Linear Approximation Table

四、 实验步骤与设计思路：

使用 Visual Studio 2019，基于 C 语言编写程序如下：

1、DDT 计算的实现

（1）定义并初始化 S0 盒、S1 盒与 DDT：

```
//S0盒
u8 S0[256] =
{
    0x3e, 0x72, 0x5b, 0x47, 0xca, 0xe0, 0x00, 0x33, 0x04, 0xd1, 0x54, 0x98, 0x09, 0xb9, 0x6d, 0xcb,
    0x7b, 0x1b, 0xf9, 0x32, 0xaf, 0x9d, 0x6a, 0xa5, 0xb8, 0x2d, 0xfc, 0x1d, 0x08, 0x53, 0x03, 0x90,
    0x4d, 0x4e, 0x84, 0x99, 0xe4, 0xce, 0xd9, 0x91, 0xdd, 0xb6, 0x85, 0x48, 0x8b, 0x29, 0x6e, 0xac,
    0xcd, 0xc1, 0xf8, 0x1e, 0x73, 0x43, 0x69, 0xc6, 0xb5, 0xbd, 0xfd, 0x39, 0x63, 0x20, 0xd4, 0x38,
    0x76, 0x7d, 0xb2, 0xa7, 0xcf, 0xed, 0x57, 0xc5, 0xf3, 0x2c, 0xbb, 0x14, 0x21, 0x06, 0x55, 0x9b,
    0xe3, 0xef, 0x5e, 0x31, 0x4f, 0x7f, 0x5a, 0xa4, 0x0d, 0x82, 0x51, 0x49, 0x5f, 0xba, 0x58, 0x1c,
    0x4a, 0x16, 0xd5, 0x17, 0xa8, 0x92, 0x24, 0x1f, 0x8c, 0xff, 0xd8, 0xae, 0x2e, 0x01, 0xd3, 0xad,
    0x3b, 0x4b, 0xda, 0x46, 0xeb, 0xc9, 0xde, 0x9a, 0x8f, 0x87, 0xd7, 0x3a, 0x80, 0x6f, 0x2f, 0xc8,
    0xb1, 0xb4, 0x37, 0xf7, 0x0a, 0x22, 0x13, 0x28, 0x7c, 0xcc, 0x3c, 0x89, 0xc7, 0xc3, 0x96, 0x56,
    0x07, 0xbf, 0x7e, 0xf0, 0x0b, 0x2b, 0x97, 0x52, 0x35, 0x41, 0x79, 0x61, 0xa6, 0x4c, 0x10, 0xfe,
    0xbc, 0x26, 0x95, 0x88, 0x8a, 0xb0, 0xa3, 0xfb, 0xc0, 0x18, 0x94, 0xf2, 0xe1, 0xe5, 0xe9, 0x5d,
    0xd0, 0xdc, 0x11, 0x66, 0x64, 0x5c, 0xec, 0x59, 0x42, 0x75, 0x12, 0xf5, 0x74, 0x9c, 0xaa, 0x23,
    0x0e, 0x86, 0xab, 0xbe, 0x2a, 0x02, 0xe7, 0x67, 0xe6, 0x44, 0xa2, 0x6c, 0xc2, 0x93, 0x9f, 0xf1,
    0xf6, 0xfa, 0x36, 0xd2, 0x50, 0x68, 0x9e, 0x62, 0x71, 0x15, 0x3d, 0xd6, 0x40, 0xc4, 0xe2, 0x0f,
    0x8e, 0x83, 0x77, 0x6b, 0x25, 0x05, 0x3f, 0x0c, 0x30, 0xea, 0x70, 0xb7, 0xa1, 0xe8, 0xa9, 0x65,
    0x8d, 0x27, 0x1a, 0xdb, 0x81, 0xb3, 0xa0, 0xf4, 0x45, 0x7a, 0x19, 0xdf, 0xee, 0x78, 0x34, 0x60
};

//S1盒
u8 S1[256] =
{
    0x55, 0xc2, 0x63, 0x71, 0x3b, 0xc8, 0x47, 0x86, 0x9f, 0x3c, 0xda, 0x5b, 0x29, 0xaa, 0xfd, 0x77,
    0x8c, 0xc5, 0x94, 0x0c, 0xa6, 0x1a, 0x13, 0x00, 0xe3, 0xa8, 0x16, 0x72, 0x40, 0xf9, 0xf8, 0x42,
    0x44, 0x26, 0x68, 0x96, 0x81, 0xd9, 0x45, 0x3e, 0x10, 0x76, 0xc6, 0xa7, 0x8b, 0x39, 0x43, 0xe1,
    0x3a, 0xb5, 0x56, 0x2a, 0xc0, 0x6d, 0xb3, 0x05, 0x22, 0x66, 0xbf, 0xdc, 0x0b, 0xfa, 0x62, 0x48,
    0xdd, 0x20, 0x11, 0x06, 0x36, 0xc9, 0xc1, 0xcf, 0xf6, 0x27, 0x52, 0xbb, 0x69, 0xf5, 0xd4, 0x87,
    0x7f, 0x84, 0x4c, 0xd2, 0x9c, 0x57, 0xa4, 0xbc, 0x4f, 0x9a, 0xdf, 0xfe, 0xd6, 0x8d, 0x7a, 0xeb,
    0x2b, 0x53, 0xd8, 0x5c, 0xa1, 0x14, 0x17, 0xfb, 0x23, 0xd5, 0x7d, 0x30, 0x67, 0x73, 0x08, 0x09,
    0xee, 0xb7, 0x70, 0x3f, 0x61, 0xb2, 0x19, 0x8e, 0x4e, 0xe5, 0x4b, 0x93, 0x8f, 0x5d, 0xdb, 0xa9,
    0xad, 0xf1, 0xae, 0x2e, 0xcb, 0x0d, 0xfc, 0xf4, 0x2d, 0x46, 0x6e, 0x1d, 0x97, 0xe8, 0xd1, 0xe9,
    0x4d, 0x37, 0xa5, 0x75, 0x5e, 0x83, 0x9e, 0xab, 0x82, 0x9d, 0xb9, 0x1c, 0xe0, 0xcd, 0x49, 0x89,
    0x01, 0xb6, 0xbd, 0x58, 0x24, 0xa2, 0x5f, 0x38, 0x78, 0x99, 0x15, 0x90, 0x50, 0xb8, 0x95, 0xe4,
    0xd0, 0x91, 0xc7, 0xce, 0xed, 0x0f, 0xb4, 0x6f, 0xa0, 0xcc, 0xf0, 0x02, 0x4a, 0x79, 0xc3, 0xde,
    0xa3, 0xef, 0xea, 0x51, 0xe6, 0x6b, 0x18, 0xec, 0x1b, 0x2c, 0x80, 0xf7, 0x74, 0xe7, 0xff, 0x21,
    0x5a, 0x6a, 0x54, 0x1e, 0x41, 0x31, 0x92, 0x35, 0xc4, 0x33, 0x07, 0x0a, 0xba, 0x7e, 0x0e, 0x34,
    0x88, 0xb1, 0x98, 0x7c, 0xf3, 0x3d, 0x60, 0x6c, 0x7b, 0xca, 0xd3, 0x1f, 0x32, 0x65, 0x04, 0x28,
    0x64, 0xbe, 0x85, 0x9b, 0x2f, 0x59, 0x8a, 0xd7, 0xb0, 0x25, 0xac, 0xaf, 0x12, 0x03, 0xe2, 0xf2
};
```

```
//DDT（统计数组）
int total[256][256] = { 0 };
```

（2）计算 DDT 的表项：

遍历 256 个输入差分和 256 个输出差分。假设当遍历到的输入差分 ΔX= X′⊕X″=A 时，输出差分 ΔY= Y′⊕Y″=S0(X′)⊕S0(X″)（或 S1(X′)⊕S1(X″)）=B，则将 DDT 的第 A+1 行、第 B+1 列的值自增 1，即 total[A][B]++。

```
int main() {
    int i, j;
    u8 ua = 0x00,ub = 0x00; //输入
    for (i = 0;i < 256;i++) {
        for (j = 0;j < 256;j++) {
            u8 va = S1[ua & 0xFF];
            u8 vb = S1[ub & 0xFF]; //输出
            total[i][va ^ vb]++; //统计
            ua += 1;
            ub = ua ^ i;
        }
        ua = 0x00;
        ub = 0x00 ^ (i+1);
    }
```

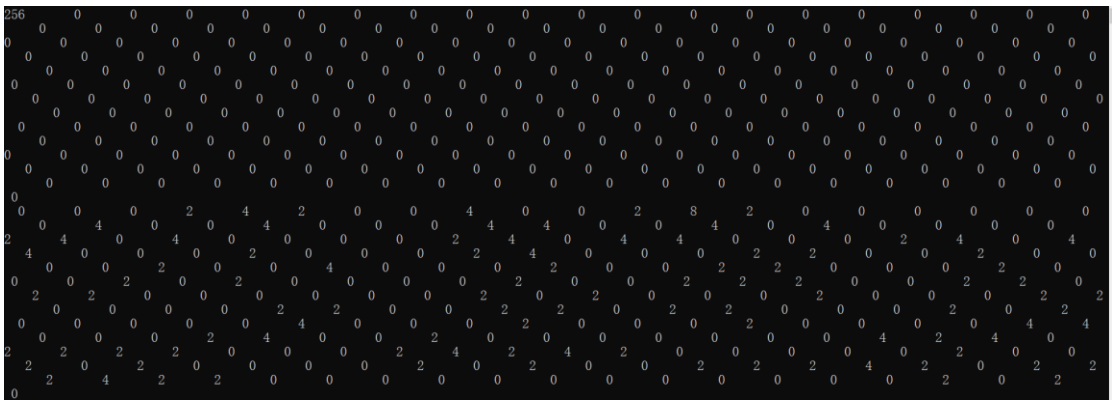
（3）打印 DDT 结果：

将上一步中统计好的 DDT（total）打印输出：

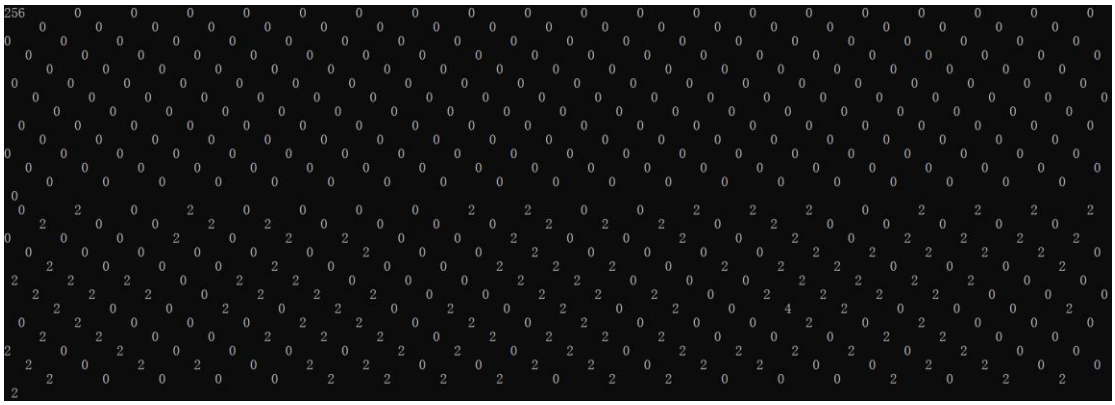
```
    u8 tou = 0x00;
    //打印结果
    for (int k = 0;k < 256;k++) {
        printf("%3.2x\t", tou);
        tou++;
    } //表头
    printf("\n");
    for (i = 0;i < 256;i++) {
        for (int j = 0;j < 256;j++) {
            printf("%3d\t", total[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

输出结果如下（部分）：

S0：



S1：



导入到 Excel 软件中如下所示（部分）：

S0：

↓Input Di	→Output I	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	14	15	16	17	18	19	1a
0	256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	2	4	2	0	0	4	0	0	2	8	2	0	0	0	0	0	0	4	0	0	4	0	0	0	4	4
2	2	0	2	0	0	0	2	0	2	4	2	4	2	0	2	0	0	0	0	0	0	2	0	0	4	2	4	0	0	0
3	3	0	0	0	0	0	0	0	0	0	0	4	0	4	0	0	0	0	0	0	0	4	0	0	4	0	0	0	0	0
4	4	0	2	4	2	4	0	0	0	4	2	0	2	4	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	0	2	0	0	0	0	0	2	0	0	0	2	4	2	0	0	0	2	0	0	6	0	0	0	0	0	2	0	0
6	6	0	0	0	0	0	0	0	2	0	2	0	0	0	0	4	0	0	4	4	0	0	0	0	2	0	2	8	0	0
7	7	0	0	4	0	0	0	0	0	0	0	2	0	2	0	0	0	2	0	4	0	4	4	2	0	0	0	0	0	0
8	8	0	4	4	4	4	4	0	4	0	4	0	4	0	4	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2	0	0	4	2	0	0	0	2	8	0	0
A	10	0	2	0	0	0	2	4	0	0	2	4	0	0	2	0	0	0	0	2	0	0	0	2	0	0	0	2	0	0
B	11	0	4	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0
C	12	0	0	0	0	0	0	4	0	0	4	0	4	0	0	0	0	0	0	0	0	4	0	4	0	0	0	0	0	0
D	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4	6	4	0	0	0	0	6	0	2	0	0	0
E	14	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	0	0	0	2	4	0	4	2	0	2	0	0	0	2
F	15	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	4	4	0	4	0	0	4	2	0	2	0	0	2
10	16	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
11	17	0	2	0	2	0	0	4	0	0	2	0	2	0	4	0	0	0	0	2	0	0	0	2	0	0	0	2	0	0
12	18	0	2	0	2	0	2	0	0	4	2	4	0	0	2	0	2	0	2	0	0	0	0	0	0	4	0	4	0	0	2
13	19	0	0	0	0	0	0	0	0	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	4
14	20	0	2	4	2	4	0	0	0	4	2	0	2	4	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	21	0	2	4	2	4	0	0	0	0	2	0	2	0	0	4	0	4	0	0	0	0	0	0	4	0	0	0	0	0	4
16	22	0	0	0	0	4	0	0	2	0	2	0	0	0	0	4	4	0	0	4	0	0	2	0	2	0	0	0	4	0
17	23	0	2	0	0	0	0	0	2	4	0	0	0	0	0	0	0	0	0	8	0	0	0	8	4	0	2	4	2	0
18	24	0	4	4	4	4	4	0	4	0	4	0	4	0	4	0	4	8	0	0	0	0	0	0	0	0	0	0	0	0	0
19	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2	0	0	4	2	0	0	0	2	8	0	0	0

S1：

↓Input Di	→Output I	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	14	15	16	17	18	19	1a
0	256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	2	0	2	0	0	0	0	2	2	0	0	2	2	2	0	2	2	2	2	2	0	0	2	2	0	0	0	0	2
2	2	0	2	0	2	2	2	2	0	0	2	0	0	0	0	2	2	2	0	0	0	0	0	0	2	2	0	2	0	2	0
3	3	0	2	2	2	0	2	0	0	2	2	2	2	0	0	0	0	2	0	0	2	0	0	0	0	2	2	2	0	2	2
4	4	0	0	0	0	0	2	0	0	0	2	2	0	2	0	0	2	2	0	0	2	2	2	0	2	0	0	0	2	0	0
5	5	0	0	0	2	2	2	2	2	2	0	0	0	0	2	0	0	2	0	2	2	2	2	0	2	0	2	2	2	0	0
6	6	0	2	0	2	0	2	0	2	0	2	0	0	2	2	0	0	2	2	4	0	2	2	2	0	2	0	2	2	2	0
7	7	0	0	0	0	2	0	2	0	0	0	0	2	2	2	2	2	0	0	2	0	0	0	2	2	0	0	0	2	2	0
8	8	0	2	0	0	0	0	2	2	0	2	0	0	0	0	0	0	2	0	2	2	0	0	0	2	0	2	2	0	2	2
9	9	0	2	0	0	0	0	0	0	0	0	0	2	0	2	2	2	0	2	0	2	0	2	0	2	0	2	2	2	2	2
A	10	0	0	2	2	0	2	0	2	0	2	2	2	0	0	0	2	0	2	0	0	2	2	2	2	2	2	2	2	2	2
B	11	0	2	0	2	0	2	0	2	2	0	2	2	2	0	2	4	2	0	0	2	0	2	0	0	0	2	0	0	2	2
C	12	0	0	2	0	2	2	2	0	2	2	0	2	2	0	2	0	2	0	0	0	2	2	0	0	0	2	0	2	0	2
D	13	0	2	0	0	0	0	2	2	0	0	2	0	0	2	2	0	2	2	0	2	0	2	2	0	0	2	2	0	0	2
E	14	0	0	0	2	2	2	2	2	0	2	0	0	0	2	0	0	2	0	0	2	2	0	0	0	2	2	0	2	2	2
F	15	0	0	0	2	2	2	0	0	2	0	0	0	2	2	2	0	2	2	2	0	0	0	0	2	0	2	0	0	2	2
10	16	0	2	0	0	0	2	0	2	0	0	0	2	0	0	2	2	2	0	0	0	0	2	0	0	0	0	2	0	0	2
11	17	0	0	0	2	0	2	0	0	2	0	2	2	0	0	0	0	0	0	2	2	0	0	0	2	2	2	2	0	2	0
12	18	0	0	2	0	0	0	0	0	2	0	2	2	0	0	0	0	2	2	0	0	2	2	0	0	2	0	2	2	2	2
13	19	0	0	0	0	2	0	0	2	2	2	0	2	0	2	0	2	0	2	0	2	2	0	2	0	2	0	0	0	2	0
14	20	0	2	2	0	0	2	0	0	0	2	0	2	0	2	0	0	0	2	2	2	2	0	0	0	2	0	2	0	2	2
15	21	0	2	0	0	0	2	2	0	0	0	2	0	0	0	2	0	0	0	0	2	2	2	0	2	2	2	0	0	2	0
16	22	0	0	2	0	2	0	0	0	0	0	2	0	2	0	0	0	0	2	0	0	2	0	0	2	0	0	2	2	2	2
17	23	0	0	2	0	2	2	2	0	2	0	2	2	2	2	2	0	0	0	2	2	0	0	0	0	2	2	0	0	2	2
18	24	0	0	0	2	0	0	2	0	2	0	0	0	0	2	0	2	0	2	0	2	2	2	2	2	0	0	0	0	0	2
19	25	0	2	0	0	2	2	2	0	0	0	0	2	2	2	0	0	2	2	2	0	2	0	2	2	0	0	2	2	0	2

2、LAT 计算的实现

（1）定义并初始化 S0 盒、S1 盒与 LAT：

同四、1、DDT 计算的实现中的操作。

（2）定义字节内的 bit 异或函数，以方便计算 $X_{i1}⊕X_{i2}⊕\cdots⊕X_{iu}$、$Y_{j1}⊕Y_{j2}⊕\cdots⊕Y_{jv}$：

```
//字节内的比特异或
int bitxor (u8 ua) {
    int res = 0;
    for (int m = 0;m < 8;m++) {
        res = res^(ua%2);
        ua=ua >> 1;
    }
    return res;
}
```

（3）计算 LAT 的表项：

遍历 256 个输入情况和 256 个输出情况。假设遍历到的如下表达式：

$$X_{i_1} \oplus X_{i_2} \oplus ... \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus ... \oplus Y_{j_v} = 0$$

当其输入 bit 均为 1 时有 $X_{i1}⊕X_{i2}⊕\cdots⊕X_{iu}=A$、输出 bit 均为 1 时有 $Y_{j1}⊕Y_{j2}⊕\cdots⊕Y_{jv}=B$、而在每个输入/输出 bit 变化时，若满足：

$$X_{i_1} \oplus X_{i_2} \oplus ... \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus ... \oplus Y_{j_v} = 0$$

则将 LAT 的第 A+1 行、第 B+1 列的值自增 1，即 total[A][B]++。

```
int main() {
    int i, j, k;
    u8 ua = 0x00;
    for (i = 0;i < 256;i++) {
        for (j = 0;j < 256;j++) {
            for (k = 0;k < 256;k++) {
                if (bitxor(ua & i) == bitxor((S1[ua & 0xFF]) & j))total[i][j]++;
                ua += 1;
            }
            ua = 0x00;
        }
    }
```

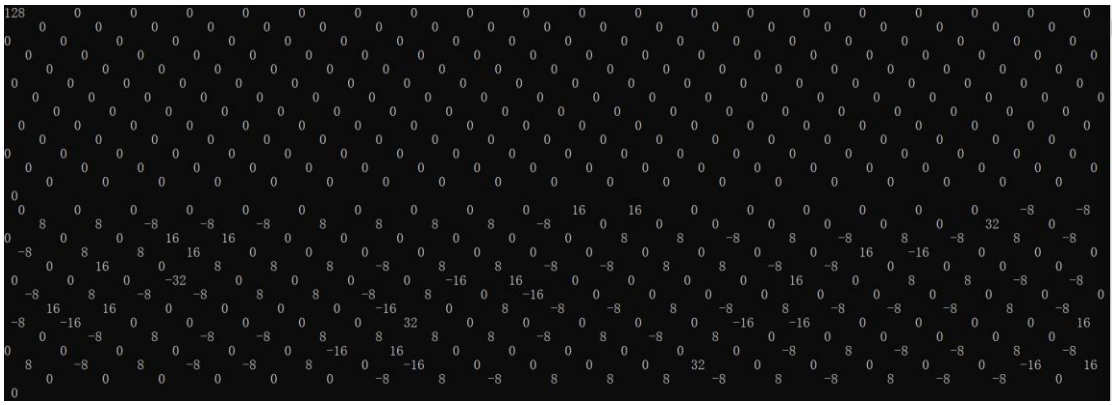
（3）打印 DDT 结果：

将上一步中统计好的 LAT（total）减去 256/2=128 后打印输出：

```
    for (i = 0;i < 256;i++) {
        for (j = 0;j < 256;j++) {
            printf("%3d\t", total[i][j]-128);
        }
        printf("\n");
    }
    return 0;
}
```

输出结果如下（部分）：

S0：



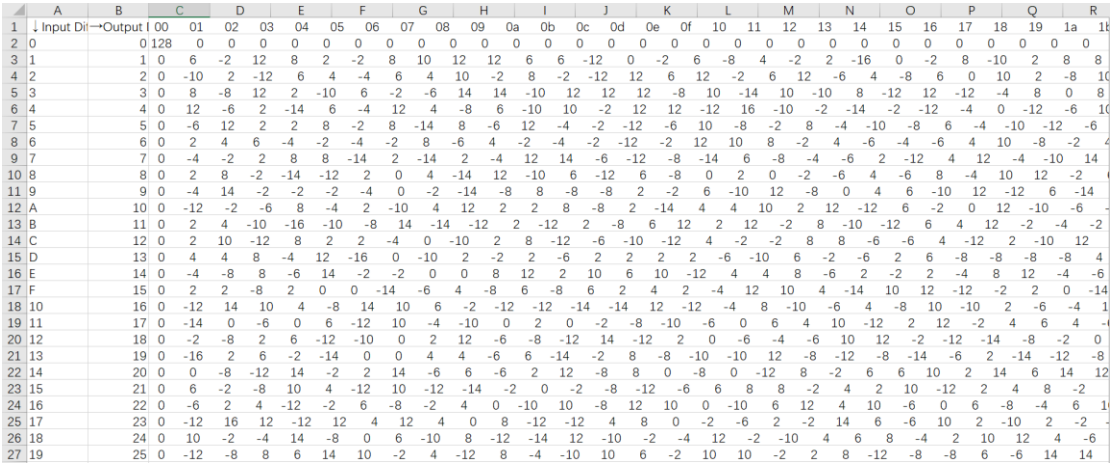
S1：

导入到 Excel 软件中如下所示（部分）：

S0：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	↓Input Dif	→Output I	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
2	0	0	128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	0	16	16	0	0	0	0	0
4	2	2	0	0	0	0	0	0	-8	8	0	0	8	-8	16	-16	8	8
5	3	3	0	0	0	0	0	0	8	8	0	0	8	-8	0	0	8	-8
6	4	4	0	0	0	-16	-16	0	0	0	0	0	-8	-8	-8	8	16	-16
7	5	5	0	0	16	0	0	0	-16	16	0	0	-8	-8	8	8	0	0
8	6	6	0	0	0	-16	0	-8	8	0	0	0	0	8	-8	8	8	0
9	7	7	0	0	0	0	0	0	8	8	0	0	-16	16	8	8	-8	8
10	8	8	0	0	-32	0	0	0	0	0	0	0	8	8	-8	8	8	-8
11	9	9	0	0	0	0	0	0	0	0	-16	-16	8	8	-8	-8	8	8
12	A	10	0	0	0	0	0	0	-8	8	-16	16	0	0	8	-8	0	0
13	B	11	0	0	0	0	0	0	-8	-8	0	0	0	0	-8	-8	0	0
14	C	12	0	0	-16	0	0	-16	16	16	0	0	0	0	0	0	8	-8
15	D	13	0	0	0	-16	0	0	0	0	0	0	0	0	0	0	-8	-8
16	E	14	0	0	0	0	0	-16	8	-8	0	0	-8	8	0	0	0	0
17	F	15	0	0	0	0	0	0	8	8	0	0	-8	8	16	16	16	-16
18	10	16	0	0	0	0	-8	-8	0	0	8	-8	-8	0	-8	8	0	-8
19	11	17	0	0	0	0	0	0	-8	-8	0	0	8	0	0	0	0	8
20	12	18	0	32	0	-32	8	-8	8	-8	0	0	0	8	8	-8	0	8
21	13	19	0	0	0	0	0	0	0	-16	-24	-24	0	8	0	0	16	-8
22	14	20	0	32	0	32	-8	-8	0	0	0	0	0	8	0	0	-8	0
23	15	21	0	0	0	0	0	0	-8	8	8	8	16	-8	8	8	8	0
24	16	22	0	0	0	0	-8	-24	8	8	-8	8	-8	0	16	-16	8	16
25	17	23	0	0	0	0	0	16	0	0	0	-8	16	8	8	-8	0	0
26	18	24	0	0	0	0	-8	24	0	0	8	-8	0	8	0	0	-8	0
27	19	25	0	0	0	0	0	0	-8	-8	0	0	0	-8	-8	-8	-8	0

S1：



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
↓Input Di	→Output I	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	14	15	16	17	18	19	1a	1b
0	0	128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	6	-2	12	8	2	-2	8	10	12	12	6	6	-12	0	-2	6	-8	4	-2	2	-16	0	-2	8	-10	2	8	8
2	2	0	-10	2	-12	6	4	-4	6	4	10	-2	8	-2	-12	12	6	12	-2	6	12	-6	4	-8	6	0	10	2	-8	10
3	3	0	8	-8	12	2	-10	6	-2	-6	14	14	-10	12	12	12	-8	10	-14	10	-10	8	-12	12	-12	-4	8	0	8	
4	4	0	12	-6	2	-14	6	-4	12	4	-8	6	-10	10	-2	12	12	-12	16	-10	-2	-14	-2	-12	-4	0	-12	-6	10	
5	5	0	-6	12	2	2	8	-2	8	-14	8	-6	12	-4	-2	-12	-6	10	-8	-2	8	-4	-10	-8	6	-4	-10	-12	-6	
6	6	0	2	4	6	-4	-2	-4	-2	8	-6	4	-2	-4	-2	-12	-2	12	10	8	-2	4	-6	-4	-6	4	10	-8	-2	4
7	7	0	-4	-2	2	8	8	-14	2	-14	2	-4	12	14	-6	-12	-8	-14	6	-8	-4	-6	2	-12	4	12	-4	-10	14	
8	8	0	2	8	-2	-14	-12	2	0	4	-14	12	-10	6	-12	6	-8	0	2	0	-2	-6	4	-6	8	-4	10	12	-2	
9	9	0	-4	14	-2	-2	-2	-4	0	-2	-14	-8	8	-8	-8	2	-2	6	-10	12	-8	0	4	6	-10	12	-12	6	-14	
A	10	0	-12	-2	-6	8	-4	2	-10	4	12	2	2	8	-8	2	-14	4	4	10	2	12	-12	6	-2	0	12	-10	-6	
B	11	0	2	4	-10	-16	-10	-8	14	-14	-12	2	-12	2	-8	6	12	2	12	-2	8	-10	-12	6	4	12	-2	-4	-2	
C	12	0	2	10	-12	8	2	2	-4	0	-10	2	8	-12	-6	-10	-12	4	-2	-2	8	8	-6	-6	4	-12	2	-10	12	
D	13	0	4	4	8	-4	12	-16	0	-10	2	-2	2	-6	2	2	2	2	-6	-10	6	-2	-6	2	6	-8	-8	-8	-8	4
E	14	0	-4	-8	8	-6	14	-2	-2	0	0	8	12	2	10	6	10	-12	4	4	8	-6	2	-2	2	-4	8	12	-4	-6
F	15	0	2	2	-8	2	0	0	-14	-6	4	-8	6	-8	6	2	4	2	-4	12	10	4	-14	10	12	-12	-2	2	0	-14
10	16	0	-12	14	10	4	-8	14	10	6	-2	-12	-12	-14	-14	12	-12	-4	8	-10	-6	4	-8	10	-10	2	-6	-4	1	
11	17	0	-14	0	-6	0	6	-12	10	-4	-10	0	2	0	-2	-8	-10	-6	0	6	4	10	-12	2	12	-2	4	6	4	
12	18	0	-2	-8	2	6	-12	-10	0	2	12	-6	-8	-12	14	-12	2	0	-6	-4	-6	10	12	-2	-12	-14	-8	-2	0	
13	19	0	-16	2	6	-2	-14	0	0	4	4	-6	6	-14	-2	8	-8	-10	-10	12	-8	-12	-8	-14	-6	2	-14	-12	-8	
14	20	0	0	-8	-12	14	-2	2	14	-6	6	-6	2	12	-8	8	0	-8	0	-12	8	-2	6	6	10	2	14	6	14	12
15	21	0	6	-2	-8	10	4	-12	10	-12	-14	-2	0	-2	-8	-12	-6	6	8	8	-2	4	2	10	-12	2	4	8	-2	
16	22	0	-6	2	4	-12	-2	6	-8	-2	4	0	-10	10	-8	12	10	0	-10	6	12	4	10	-6	0	6	-8	-4	6	1
17	23	0	-12	16	12	-12	12	4	12	4	0	8	-12	-12	4	8	0	-2	-6	2	-2	14	6	-6	10	2	-10	2	-2	
18	24	0	10	-2	-4	14	-8	0	6	-10	8	-12	-14	12	-10	-2	-4	12	-2	-10	4	6	8	-4	2	10	12	4	-6	
19	25	0	-12	-8	8	6	14	10	-2	4	-12	8	-4	-10	10	6	-2	10	10	-2	2	8	-12	-8	-8	6	-6	14	14	

五、 实验结果分析：

1、 对 ZUC 算法 S0 与 S1 盒的 DDT 与 LAT 的分析

表格	S0, DDT		S1, DDT		S0, LAT		S1, LAT	
表中元素的绝对值及其相应个数分布	值	个数	值	个数	值	个数	值	个数
	256	1	256	1	128	1	128	1
	8	304	4	255	32	160	16	1275
	6	560	2	32130	24	288	14	4080
	4	6576	0	33150	16	7232	12	9180
	2	16592	-	-	8	31200	10	6120
	0	41503	-	-	4	0	8	8670
	-	-	-	-	2	0	6	10200
	-	-	-	-	0	26655	4	9180
	-	-	-	-	-	-	2	12240
	-	-	-	-	-	-	0	4590

从表中非零元素个数来看，S0 的 DDT 和 LAT 中非零元素的个数比 S1 的 DDT 和 LAT 中非零元素的个数都少；从表中元素的绝对值大小来看，S0 的 DDT 和 LAT 中元素的绝对值偏大，S1 的 DDT 和 LAT 中元素的绝对值偏小。

2、 Basic SPN 加密需要最后一轮密钥混合的原因

设在对 Basic SPN 的差分分析中，对应输入差分 $\Delta X= X' \oplus X''$的、用这一轮的密钥 K 加密前的明文差分为 $\Delta P= P' \oplus P''$，则 $\Delta X= (P' \oplus K) \oplus (P'' \oplus K)= P' \oplus P''$，

因此可以看出，在计算 ΔX 时，密钥没有起作用；故如果使用和轮数相等个数的密钥，则在最后一轮的输出处很容易根据计算好的差分分布表 DDT 进行攻击，而在 SPN 的最后一轮异或轮密钥的操作，会使得攻击者先需要猜测最后一轮的轮密钥 K_R，在获得倒数第二轮的密文输出 C'以后，再检验输入输出的差分特征，如果正确则才能判断 K_R 猜测成功，虽然攻击者不需要穷举 K_R 的全部位，但是还是需要穷举所有的差分路径最后一轮传递到的 bit 所在块，增加了攻击的开销。

（课件上的另一个问题：在 SPN 的最后一轮不需要 P 操作的原因是因为 P 操作对抵抗差分密码分析没有提供任何安全性。）

六、实验中的问题与总结：

本次实验我基于 C 语言编程，计算出了 ZUC 的 S0 与 S1 盒的差分分布表（DDT）与线性近似表（LAT），并分析了 Basic SPN 为什么形成密文需要最后一轮的密钥混合。实验中遇到的主要问题是：由于 ZUC 的 S0 与 S1 盒的 DDT 和 LAT 是 256x256 的比较大的表格，因此不太容易输出得比较清晰和整齐，在导入 Excel 后稍微解决了一些问题，但是还是没有达到很好的效果；另一个是在计算 LAT 的时候，一开始没有考虑到需要字节内部的运算，后来发现需要一个这样的运算，因此添加了一个字节内的比特异或“bitxor()”函数。

【参考资料】

[1] https://wenku.baidu.com/view/c5c1ca5c0640be1e650e52ea551810a6f424c84e.html （ZUC 密码算法）.

[2] https://baike.baidu.com/item/SPN/6532168?fr=aladdin （SPN-百度百科）.