

## 实验报告（流密码）

### 【实验目的】

- 1、了解常用的流密码算法，并对其进行实现；
- 2、了解常用的伪随机数生成算法，并对其进行实现。

### 【实验环境】

- Python 3.9.7 64-bit

### 【实验内容】

## 一、BBS 伪随机发生器

### 1、算法流程

- $p \equiv q \equiv 3 \pmod{4}$ ;  $n = p \times q$ ;  $\gcd(s, n) = 1$
- 令  $X_0 = s^2 \pmod{n}$
- **for**  $i \leftarrow 1$  **to**  $len$ :  $X_i = X_{i-1}^2 \pmod{n}$ ;  $B_i = X_i \pmod{2}$

### 2、测试样例及结果截图

```
128
22216588297723936193972858835061906049
811911977174302056128488503423240581
125679355090102963147251551176497957611014591246
304206124949867201243173227033338624577

Process finished with exit code 0
```

```
10
290245406325850415858814760645754379607
139185043753526822918040071236437427801
479174373179062226102966029950357353939062937622
590

Process finished with exit code 0
```

```
20
188356838742927928098180427092862499797
240080138223277870155085310285290218427
1387038472795772141149847620238577515250983611146
421378

Process finished with exit code 0
```

### 3、讨论与思考

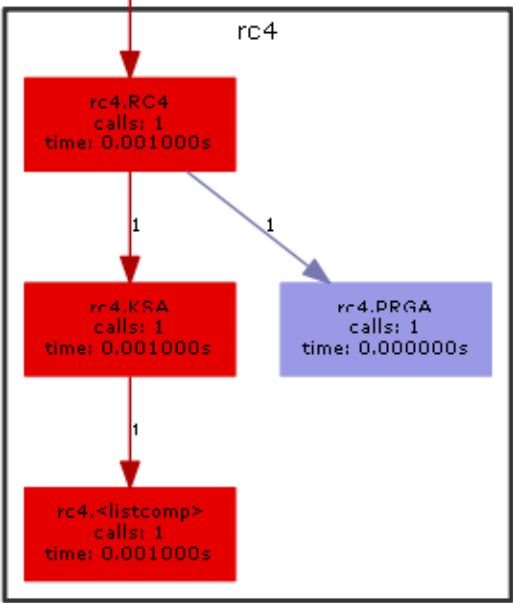
- BBS 发生器是产生安全伪随机数的普遍算法，是特意构造算法中密码强度有最强公开证明的一个，被称为密码安全伪随机位发生器（CSPRBG）。
- BBS 之所以被称为安全伪随机位生成器，是因为它能经受续位测试。续位测试是指给定序列的最开始  $k$  位，没有任何有效算法可以产生超过  $1/2$  的概率预测出第  $k+1$

- 位。因此对于实际应用，这个序列是不可预测的。
- BBS 的安全性是基于对 $n$ 的因子分解的困难性上的。

二、RC4 流密码算法

1、算法流程

函数调用关系图如下：



函数名	函数功能
KSA()	密钥调度算法
PRGA()	伪随机生成算法

2、测试样例及结果截图

```
0x95f6
0xd675322f52e216ad
0x4fb36276bae193fc
Process finished with exit code 0
```

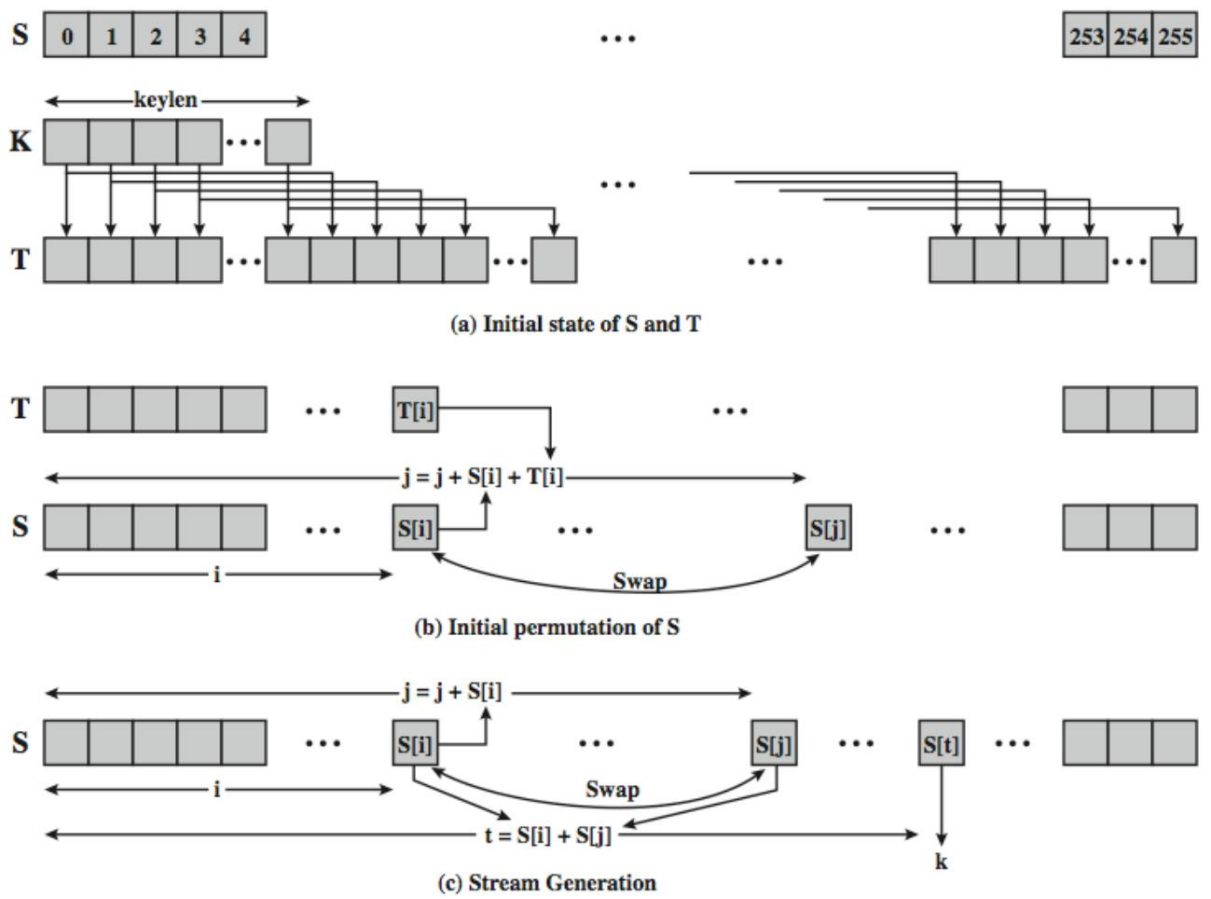
```
0xe78a3bcf43b517ca
0xf9f8ded7de8072ee
0x90401d2ab6abc3a
Process finished with exit code 0
```

```
0x86c005ae8d42a1e8b012ce710e38d35672d6b4b41249975f69
0x646d0ac40f7d0594
0xdec4d5929f3344ef
Process finished with exit code 0
```

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存	详细信息
18982	2022-05-03 22:09:54	Accepted	Python	72ms	9220KB	

3、讨论与思考

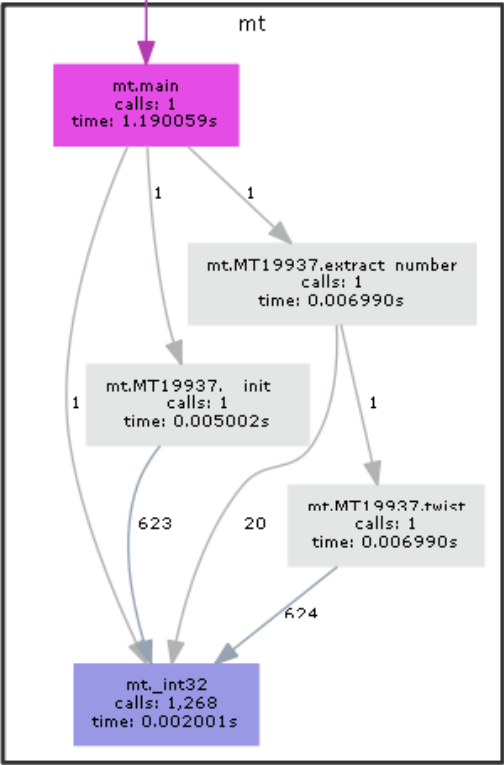
- 通过输入的密钥 $k$ 初始化非线性 $S$ 表；
- 对 $S$ 表进行非线性变换，得到密钥流（实质上是生成伪随机数）；
- 明文与密钥流异或得到密文。
- RC4 作为流密码的实质在于 PRNG 算法的流性质。




### 三、梅森旋转算法

#### 1、算法流程

函数调用关系图如下：



## 2、测试样例及结果截图（结果过于冗杂，仅输出 OJ 截图）

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存	详细信息
18983	2022-05-03 22:10:07	Accepted	Python	44ms	8636KB	

## 3、讨论与思考

- 梅森旋转算法分为三个阶段。
- 第一阶段：通过种子和递推式得到基础的梅森旋转链；
- 第二阶段：对旋转链进行旋转（实质上是线性置换再异或一个由该位生成的数）；
- 第三阶段：从旋转后的链提取出随机数。

## 四、ZUC-128 算法

## 1、算法流程

伪代码：

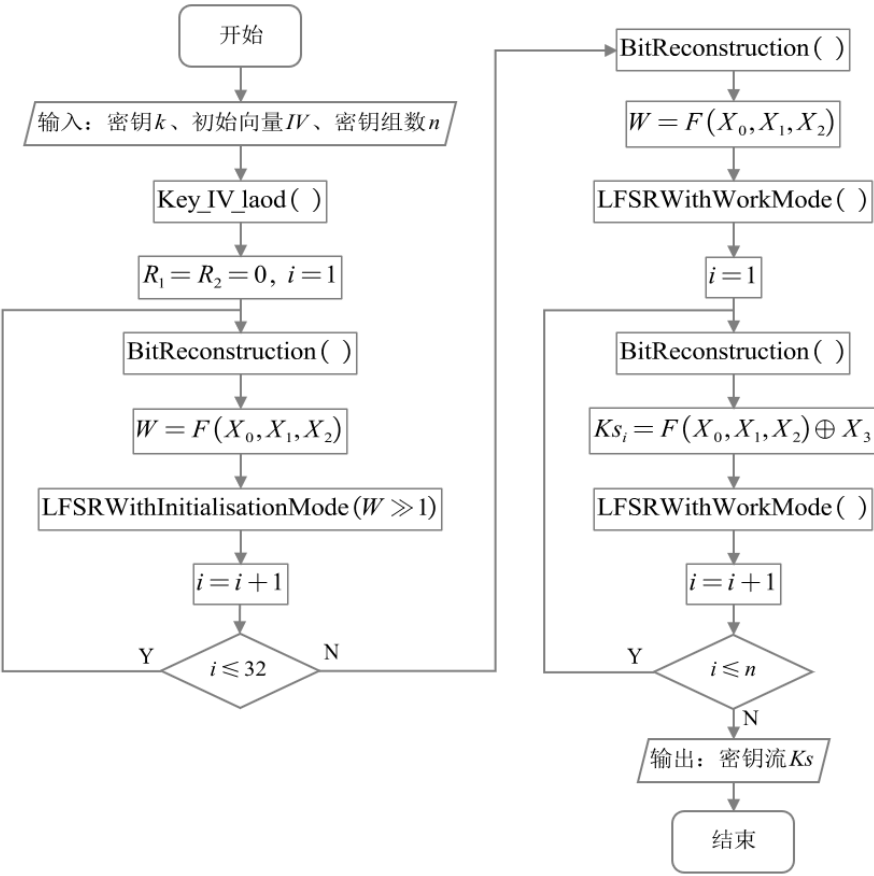
**Algorithm 1:** ZUC — 128**Input:** key  $k$ , initial vector  $IV$ , number of key groups  $n$ **Output:** keystreams  $Ks$ 

```

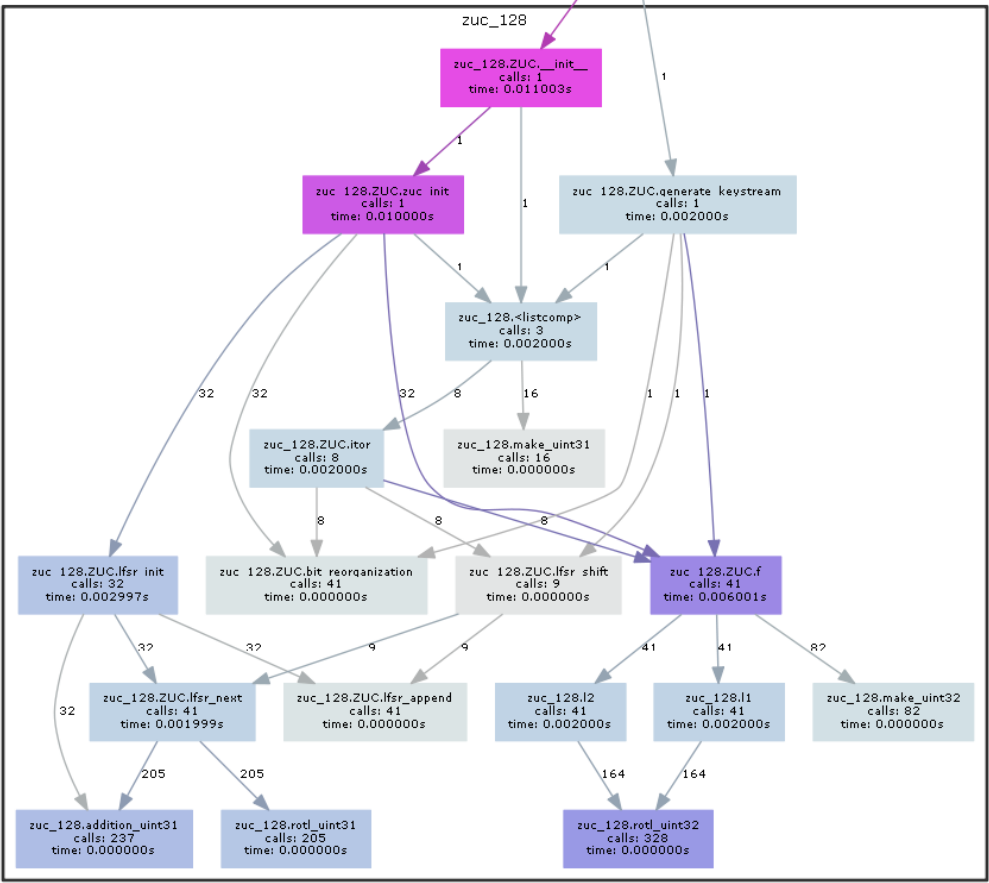
1. function "ZUC_128( $k, IV, n$ )"
2.    /* 初始化操作 */
3.    Key_IV_load( )    // 密钥装入
4.     $R_1 \leftarrow 0; R_2 \leftarrow 0$ 
5.    for  $i \leftarrow 1$  to 32 do:
6.        BitReconstruction( )    // 比特重组
7.         $W \leftarrow F(X_0, X_1, X_2)$     // 非线性函数
8.        LFSRWithInitialisationMode( $W \gg 1$ )    // LFSR初始化模式
9.    end for
10.   /* 生成密钥流 */
11.   set  $Ks[n]$     // 设置  $n$  长列表储存密钥流
12.   BitReconstruction( )
13.    $W \leftarrow F(X_0, X_1, X_2)$ 
14.   LFSRWithWorkMode( )    // LFSR工作模式
15.   for  $i \leftarrow 1$  to  $n$  do:
16.       BitReconstruction( )
17.        $Ks[i] \leftarrow F(X_0, X_1, X_2) \oplus X_3$ 
18.       LFSRWithWorkMode( )
19.   end for
20.   return  $Ks$ 
21. end function

```

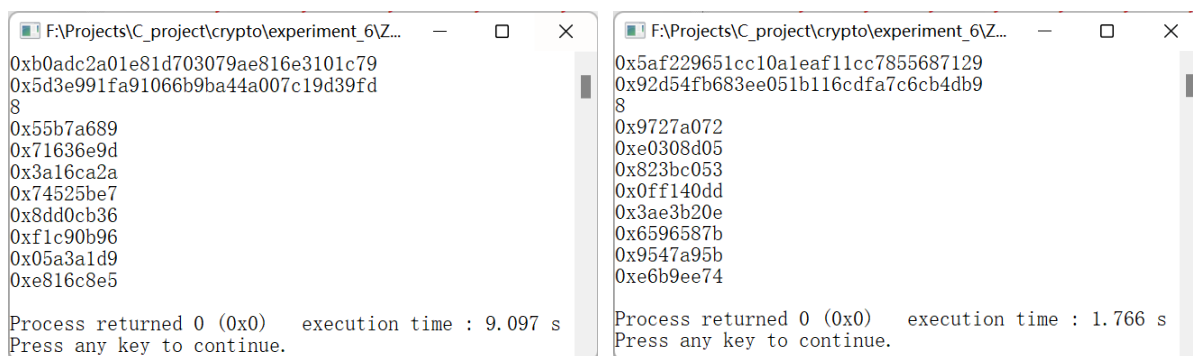
流程图：



函数调用关系图如下：

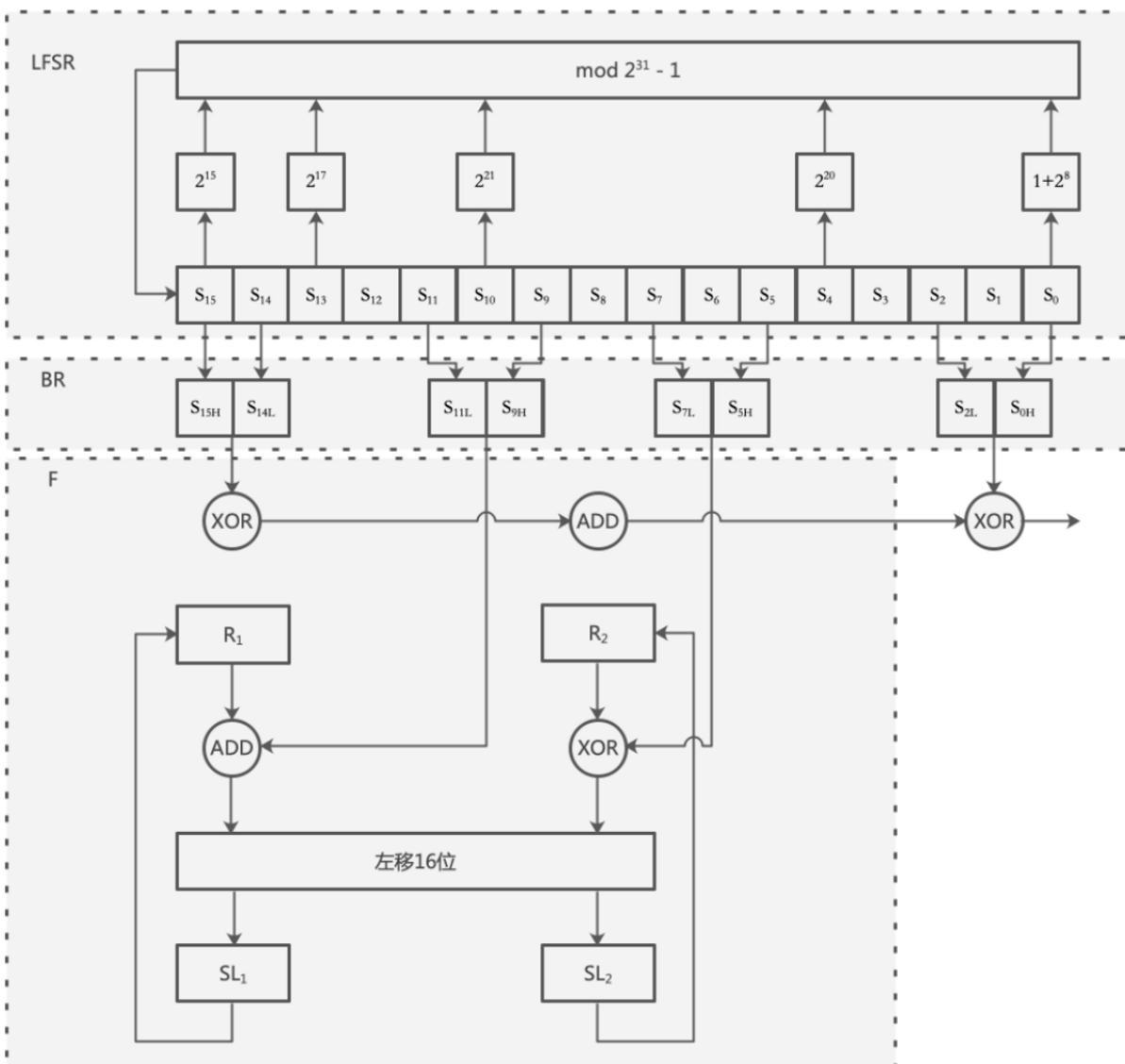


## 2、测试样例及结果截图



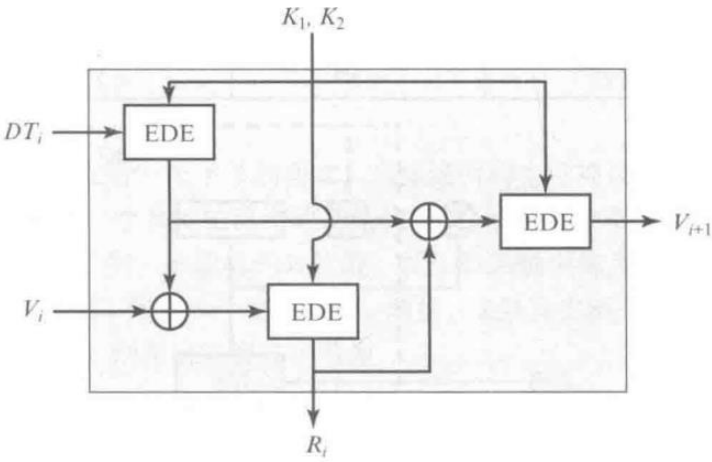
## 3、讨论与思考

- 祖冲之加密算法的执行分为两个阶段：初始化阶段和工作阶段；
- 第一阶段，密钥和初始向量进行初始化，即不产生输出；
- 第二阶段，工作阶段，每一个时钟产生一个 32 bit 密钥输出。
- LFSR 的初始化模式有输入 ( $W \gg 1$ )，工作模式没有输入。



五、【选做】ANSI X9.17 伪随机发生器

1、算法流程



2、测试样例及结果截图（结果过于冗杂，仅展示 OJ 截图）

评测编号 ↓	提交时间	提交状态	代码语言	最大运行 时间	最大运行内存	详细信息
18986	2022-05-03 22:10:48	Accepted	C	19ms	1712KB	

3、讨论与思考

- ANSI X9.17 标准是美国国家标准化协会制订的金融机构密钥管理规范；ANSI（美国国家标准化协会）负责金融安全的小组是 ASC X9 和 ASC X12。其中 ASC X9 负责制定金融业务标准，ASC X12 负责制定商业交易标准。其中 ANSI X9.17 标准制定于 1985 年，对金融机构的密钥管理进行了规范。

**【收获与建议】**

## 1、收获

- ~~加深了对流密码的理解。~~
- ~~提高了对 Python 和 C 语言的熟练度。~~
- ~~巩固了排版能力。~~
- 收获了劳累与繁忙。

## 2、建议

- 无。

**【思考题】**

## 1、思考 RC4 算法中什么样的密钥属于弱密钥。

RC4 存在弱密钥使得其在初始置换后  $S$  表的顺序不变，即密钥失去作用。初始置换伪代码：

```
1.  $j \leftarrow 0$ 
2. for  $i \leftarrow 1$  to 256 do
3.      $j \leftarrow (j + S_i + S_j) \bmod 256$ 
4.      $S_i, S_j \leftarrow S_j, S_i$ 
5. end for
```

已知  $S$  中的元素不会重复，若使交换  $S_i, S_j$  的顺序后  $S$  的顺序并未发生改变，则  $i = j$ ，即  $i = (j + S_i + S_j) \bmod 256$ ，这里我们可以只推导其中的一组弱密钥，则可以认为  $i = j + S_i + S_j$ 。则：

$$\begin{aligned} i = 0 &\Rightarrow j + S_0 + T_0 = 0 \Rightarrow T_0 = 0 \\ i = 1 &\Rightarrow j + S_1 + T_1 = 1 \Rightarrow T_1 = 0 \\ i = 2 &\Rightarrow j + S_2 + T_2 = 2 \Rightarrow T_2 = 255 \\ &\dots \\ i = 255 &\Rightarrow j + S_{255} + T_{255} = 255 \Rightarrow T_{255} = 2 \end{aligned}$$

由于  $T$  是由初始密钥  $K$  推导出来的，则可以认为

$$K_0 = 0, K_1 = 0, K_2 = 1, \dots, K_{255} = 2$$

是 RC4 的一组弱密钥。