实验报告

【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】

1. 语言：C
2. 平台：clion 2021.2 版本

【实验内容】

一、BBS 伪随机数生成算法

1. 算法流程

1. 选择两个大素数p,q, 满足：

$$p \equiv q \equiv 3(mod\,4)$$

2. 同时令：

$$n = pq$$

3. 产生一个随机数s，满足：

$$(s, n) = 1$$

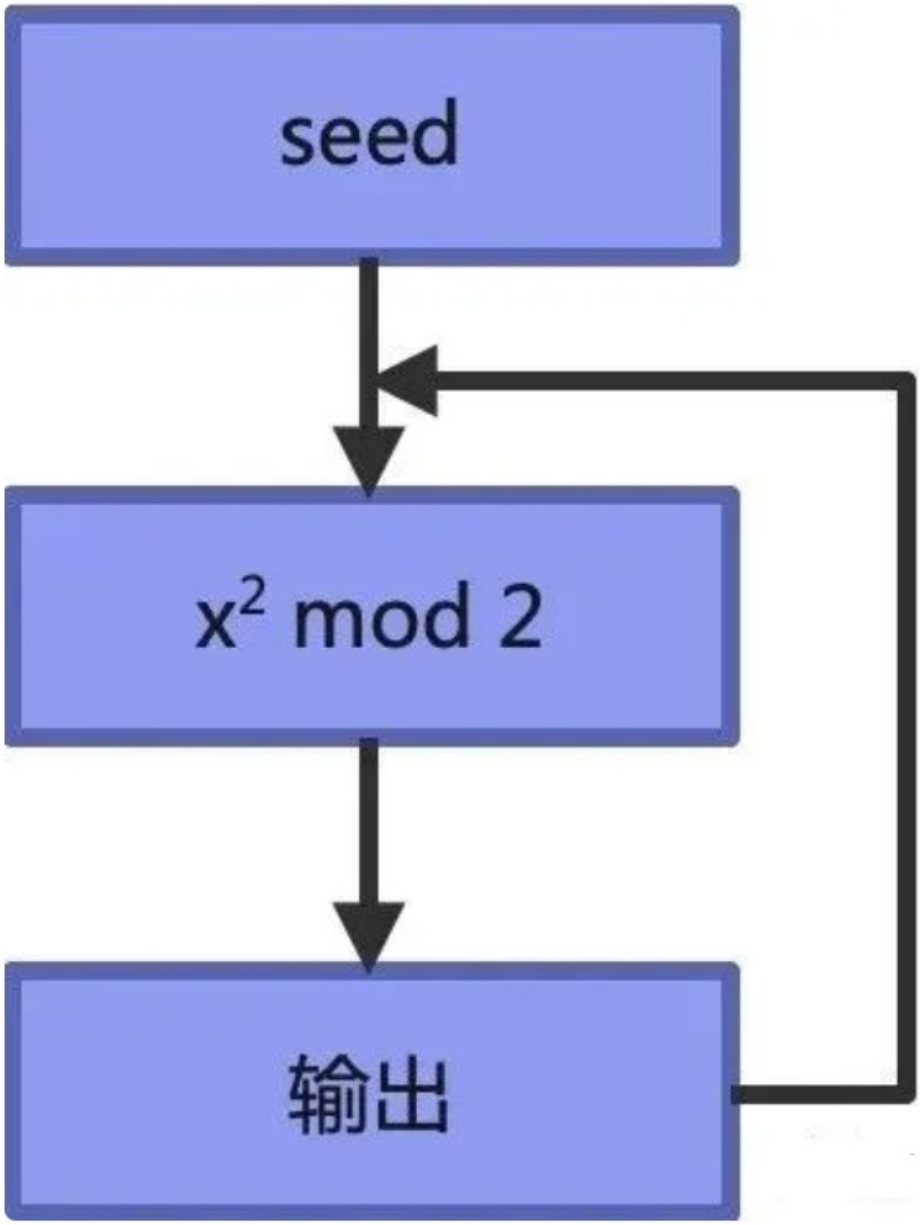
4. 随后按照如下算法产生位B_i序列：

$$\begin{aligned}
&1: X_0 = s^2 \,mod\, n \\
&2: for\ i = 1\ to\ \infty \\
&3: \quad X_i = (X_{i-1})^2\ mod\ n \\
&4: \quad B_i = X_i\ mod\ 2
\end{aligned}$$

- **流程图：**

1. **BBS 伪随机数生成算法流程图：**

生成器结构

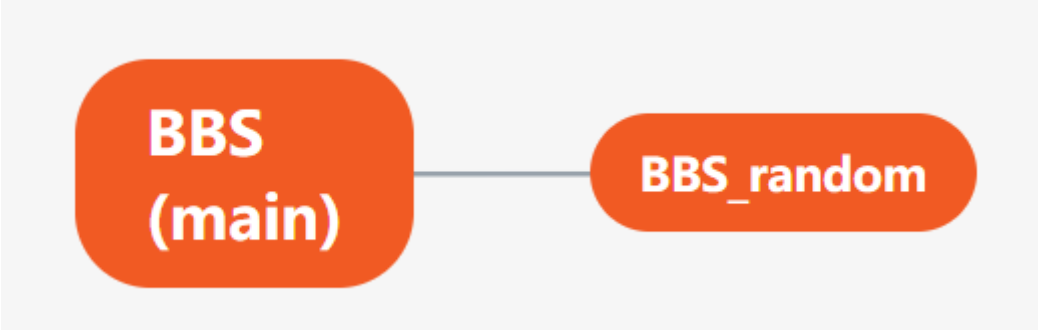


- **伪代码：**

1. **伪代码：**

$$
\begin{aligned}
&Algorithm \quad BBS\_random\\
&Input: p, q, s\\
&output: \{B_0, \ldots, B_n\}\\
&1: X_0 = s^2 \, mod \, n\\
&2: for \; i = 1 \; to \; n\\
&3: \quad X_i = (X_{i-1})^2 \; mod \; n\\
&4: \quad B_i = X_i \; mod \; 2\\
&5: return \; \{B_0, \ldots, B_n\}
\end{aligned}
$$

- **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\BBS.exe"
128
222165882977239361939728588835061906049
811911977174302056128488503423240581
125679355090102963147251551176497957611014591246
304206124949867201243173227033338624577
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\BBS.exe"
10
290245406325850415858814760645754379607
139185043753526822918040071236437427801
479174373179062226102966029950357353939062937622
590
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\BBS.exe"
20
188356838742927928098180427092862499797
240080138223277870155085310285290218427
138703847279577214114984762023857751525098361l146
421378
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\BBS.exe"
128
107571070039009448816193583052040328949
546940289893798617644794375759966711193
577865590625744890672079647166268106427594200590
200916045019258624445255024137375751322
Process finished with exit code 0
```

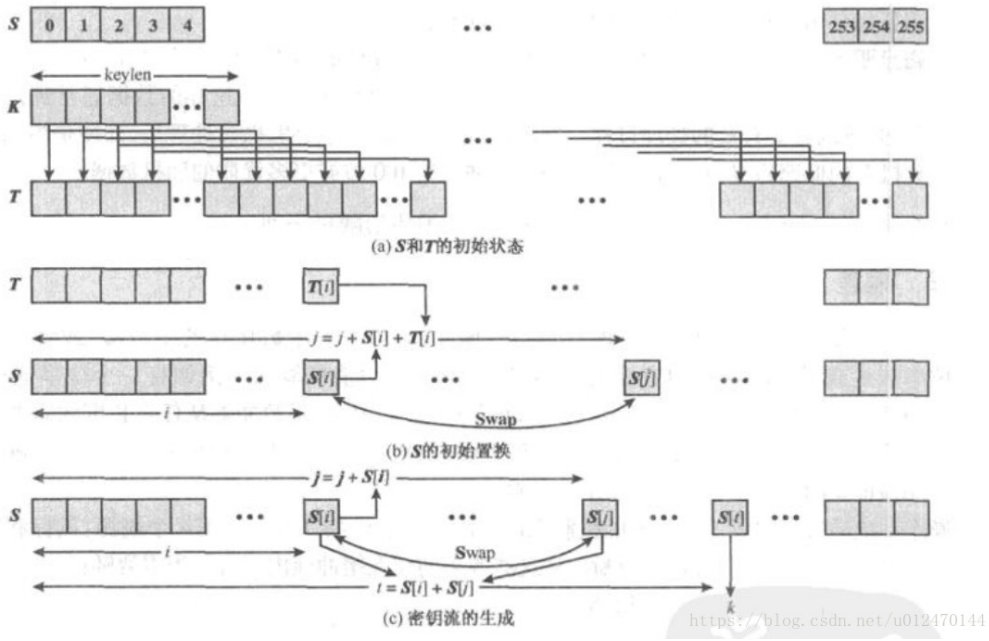
```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\BBS.exe"
128
229441290505062992396765077701994260367
150060395801617986397147392247163582581
142109613173842923360554061740667073013211500690l
296007184477070248406118829362621811463
Process finished with exit code 0
```

二、BBS 伪随机数生成算法

1. 算法流程

○ 流程图：

1. 流程总览：



加密流程分为三步，先对S，T初始化，再对S作初始置换，最后生成密钥流。

○ 伪代码：

1. 伪代码：

总伪代码：

算法 2 RC4算法

输入： $k, plain$

输出： $cipher$

```
function RC4(k, plain)
    S, T ← Initialization(k)
    S ← IP_S(S, T)
    cipher ← RC4_crypto(S, plain)
    return cipher
end function
```

三步对应的伪代码：

算法 3 初始化S，T

输入： K

输出： S, T

```
function INITIALISATION(K)
    for i = 0 → i = 255 do
        S_i ← i
        T_i ← K_{i mod keylen}
    end for
    return S, T
end function
```

算法 4 S初始置换

输入： S, T
输出： S

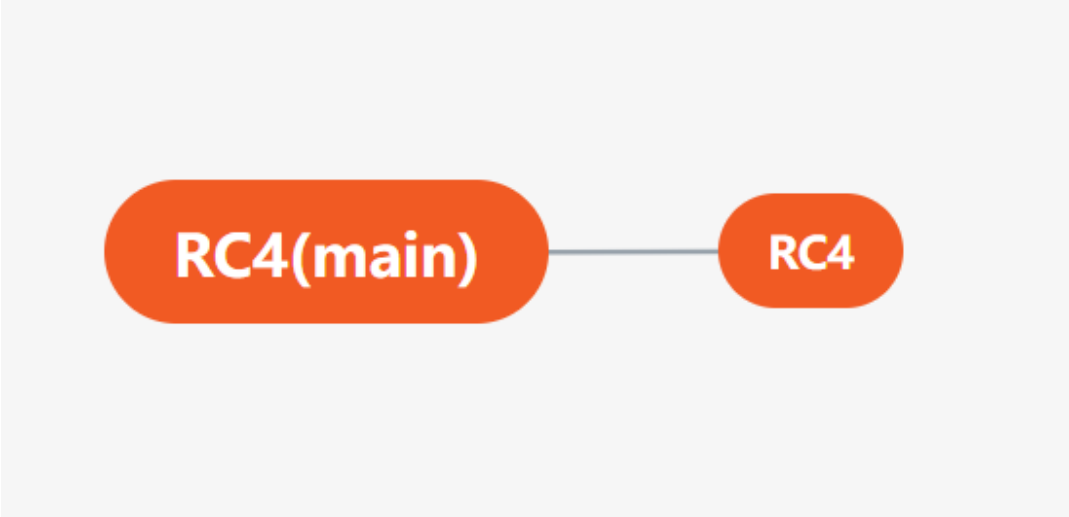
```
function IPS(S, T)
    j ← 0
    for i = 0 → i = 255 do
        j ← (j + S_i + T_i) mod 256
        swap(S_i, S_j)
    end for
    return S
end function
```

算法 5 RC4加密

输入： $S, plain$
输出： $cipher$

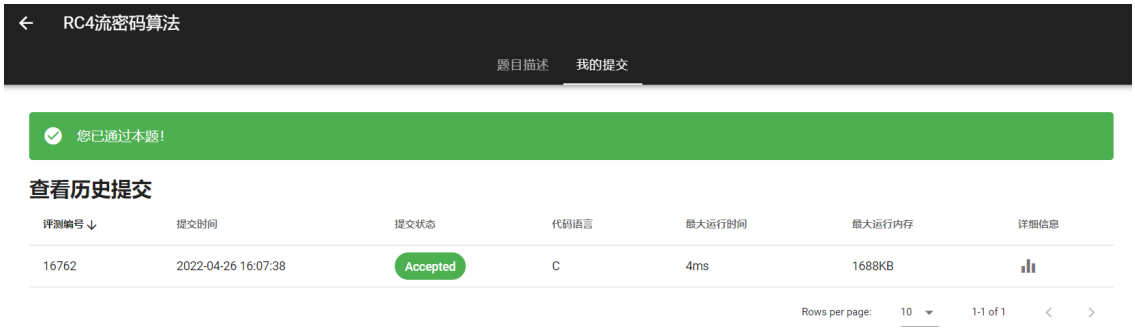
```
function RC4CRYPTO(S, plain)
    plist ← plain(pos : pos + 2)
    for item ∈ plist do
        ks ← item
        i ← (i + 1) mod 256
        j ← (j + S_i) mod 256
        swap(S_i, S_j)
        t ← (S_i, S_j) mod 256
        ks ← ks ⊕ t
        cipher ← cipher + hex(ks)
    end for
    return cipher
end function
```

- **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：



三、梅森旋转算法

1. **算法流程**

   - **流程图：**

     1. **流程总览：**

        如图，我们先输入一个种子$seed$将其作为$MT[0]$的值，根据递推公式求出梅森旋转链。之后遍历旋转链，对每一个节点作旋转处理，最后再对旋转所得结果进行处理，输出。

   - **伪代码：**

     1. **各分函数伪代码：**

        初始化种子函数如下：

算法 7 seedinit

输入： $seed$

输出： $index$

```
1: function SEEDINIT(seed)
2:     MT[0] ← seed
3:     index ← n
4:     for i = 0 → i = n − 1 do
5:         if i = 0 then
6:             continue
7:         end if
8:         tmp = f ∗ (MT[i − 1] ⊕ (MT[i − 1] >> (w − 2))) + i
9:         MT[i] = tmp&d
10:    end for
11:    return index
12: end function
```

在extract函数中涉及到的twist函数伪代码如下：

算法 8 twist

输入： $\emptyset$

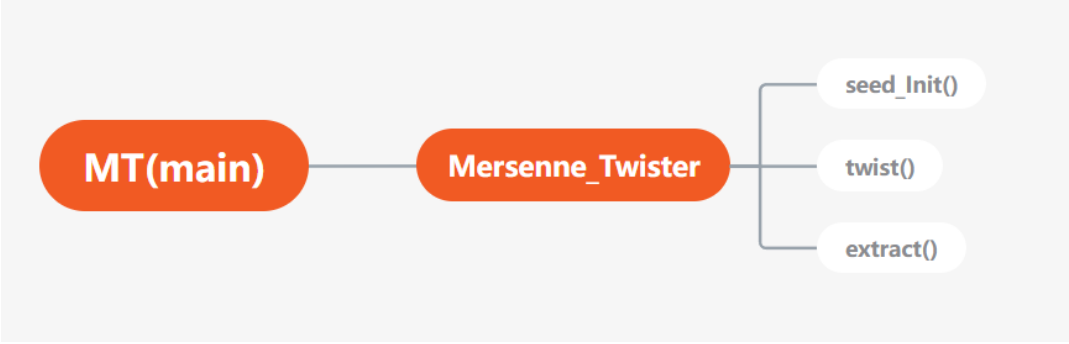
输出： $index$

```
1: function TWIST(∅)
2:     lowermask = (1 << r) − 1
3:     uppermask =! lowermask
4:     for i = 0 → i = n − 1 do
5:         x = (MT[i] & uppermask) + (MT[(i + 1) mod n] & lowermask)
6:         xA = x >> 1
7:         if x mod 2 ! = 0 then
8:             xA = xA ⊕ a
9:         end if
10:    end for
11:    index = 0
12:    return index
13: end function
```

函数总流程图如下：

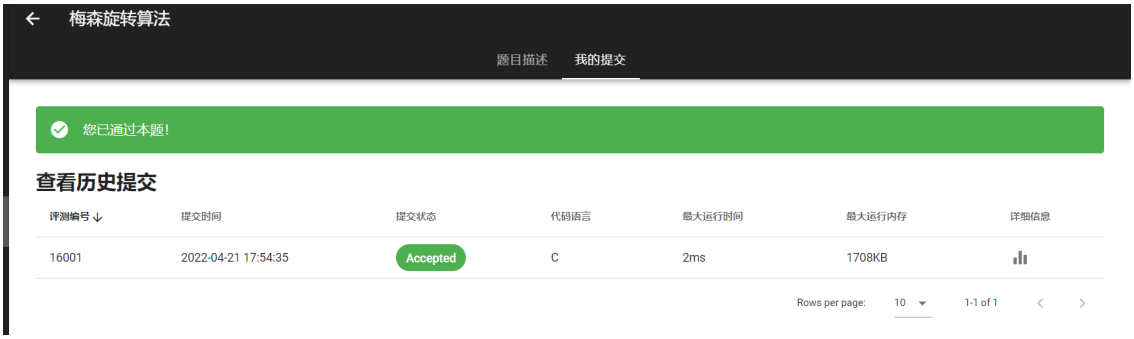
$$
\begin{aligned}
&Algorithm \quad Mersenne\_Twister\\
&Input: seed, num\\
&output: ans[num]\\
&1: seed_Init(seed)\\
&2: twist()\\
&3: for\ i = 1\ to\ num\\
&4: \quad ans[i] \leftarrow extract()\\
&5: return\ ans[num]
\end{aligned}
$$

- **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

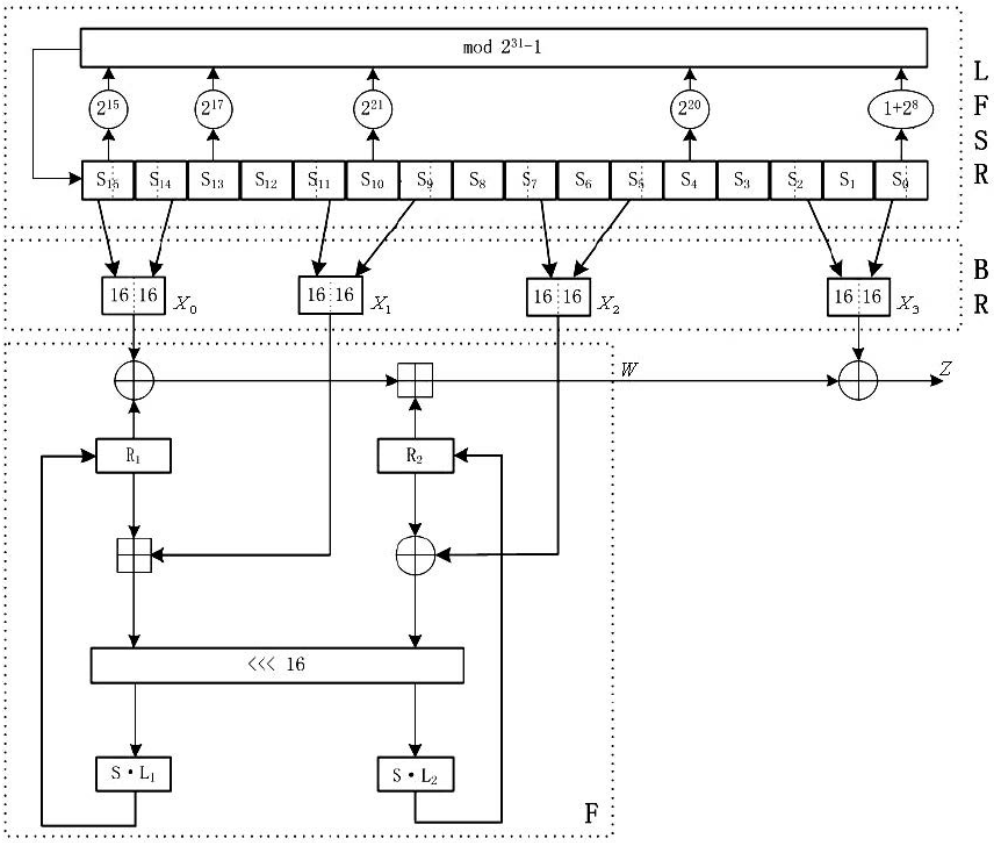


四、ZUC-128（祖冲之算法）

1. **算法流程**

   - **流程图：**

     1. **流程总览：**



   - **伪代码：**

     1. **伪代码：**

        1. 总算法代码：

4.6.1 初始化阶段

首先把 128 比特的初始密钥 k 和 128 比特的初始向量 iv 按照 4.5 节密钥装入方法装入到 LFSR 的存放器单元变量 $s_0, s_1, \ldots, s_{15}$ 中，作为 LFSR 的初态，并置 32 比特记忆单元变量 R_1 和 R_2 为全 0。然后执行下述操作：

重复执行下述过程 32 次：

(1) BitReconstruction()；

(2) $W= F(X_0, X_1, X_2)$；

(3) LFSRWithInitialisationMode ($W >> 1$)。

4.6.2 工作阶段

首先执行如下过程一次，并将 F 的输出 W 舍弃：

(1) BitReconstruction()；

(2) $F (X_0, X_1, X_2)$；

(3) LFSRWithWorkMode()。

然后进入密钥输出阶段。在密钥输出阶段，每运行一个节拍，执行如下过程一次，并输出一个 32 比特的密钥字 Z：

(1) BitReconstruction()；

(2) Z = F (X0, X1, X2) ⊕ X3；

(3) LFSRWithWorkMode()。

2. LFSR_init_Mode伪代码：

LFSRWithInitialisationMode(u)

{

(1) $v = 2^{15}s_{15} +2^{17} s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1 + 2^8)s_0 \bmod (2^{31}-1)$；

(2) $s_{16}=(v+u) \bmod (2^{31}-1)$；

(3) 如果 $s_{16}=0$，如此置 $s_{16}=2^{31}-1$；

(4) $(s_1, s_2, \ldots, s_{15}, s_{16})$ $(s_0, s_1, \ldots, s_{14}, s_{15})$。

}

3. LFSR_work_Mode伪代码：

LFSRWithWorkMode()

{

(1) $s_{16} = 2^{15} s_{15} +2^{17}s_{13} + 2^{21} s_{10} + 2^{20}s_4 + (1 + 2^8)s_0 \bmod (2^{31}-1)$；

(2) 如果 $s_{16}=0$，如此置 $s_{16}=2^{31}-1$；

(3) $(s_1, s_2, \ldots, s_{15}, s_{16})$ $(s_0, s_1, \ldots, s_{14}, s_{15})$。

}

4. BR伪代码：

BitReconstruction()

{

(1) $X_0 = s_{15H} \| s_{14L}$;

(2) $X_1 = s_{11L} \| s_{9H}$;

(3) $X_2 = s_{7L} \| s_{5H}$;

(4) $X_3 = s_{2L} \| s_{0H}$。

}

5. 非线性函数F代码：

F 包含 2 个 32 比特记忆单元变量 R_1 和 R_2。

F 的输入为 3 个 32 比特字 X_0、X_1、X_2，输出为一个 32 比特字 W。F 的计算过程如下：

F (X_0, X_1, X_2)

{

(1) $W = (X_0 \oplus R_1) \boxplus R_2$;

(2) $W_1 = R_1 \boxplus X_1$;

(3) $W_2 = R_2 \oplus X_2$;

(4) $R_1 = S(L_1(W_{1L} \| W_{2H}))$;

(5) $R_2 = S(L_2(W_{2L} \| W_{1H}))$。

}

其中 S 为 32 比特的 S 盒变换，定义在附录 A 中给出；L_1 和 L_2 为 32 比特线性变换，定义如下：

$$L_1(X) = X \oplus (X <<< 2) \oplus (X <<< 10) \oplus (X <<< 18) \oplus (X <<< 24),$$

$$L_2(X) = X \oplus (X <<< 8) \oplus (X <<< 14) \oplus (X <<< 22) \oplus (X <<< 30)。$$

6. 密钥装入函数：

密钥装入过程将 128 比特的初始密钥 k 和 128 比特的初始向量 iv 扩展为 16 个 31 比特字作为 LFSR 寄存器单元变量 $s_0, s_1, \cdots, s_{15}$ 的初始状态。设 k 和 iv 分别为

$$k_0 \| k_1 \| \cdots\cdots \| k_{15}$$

和

$$iv_0 \| iv_1 \| \cdots\cdots \| iv_{15},$$

其中 k_i 和 iv_i 均为 8 比特字节，$0 \leqslant i \leqslant 15$。密钥装入过程如下：

(1) D 为 240 比特的常量，可按如下方式分成 16 个 15 比特的子串：

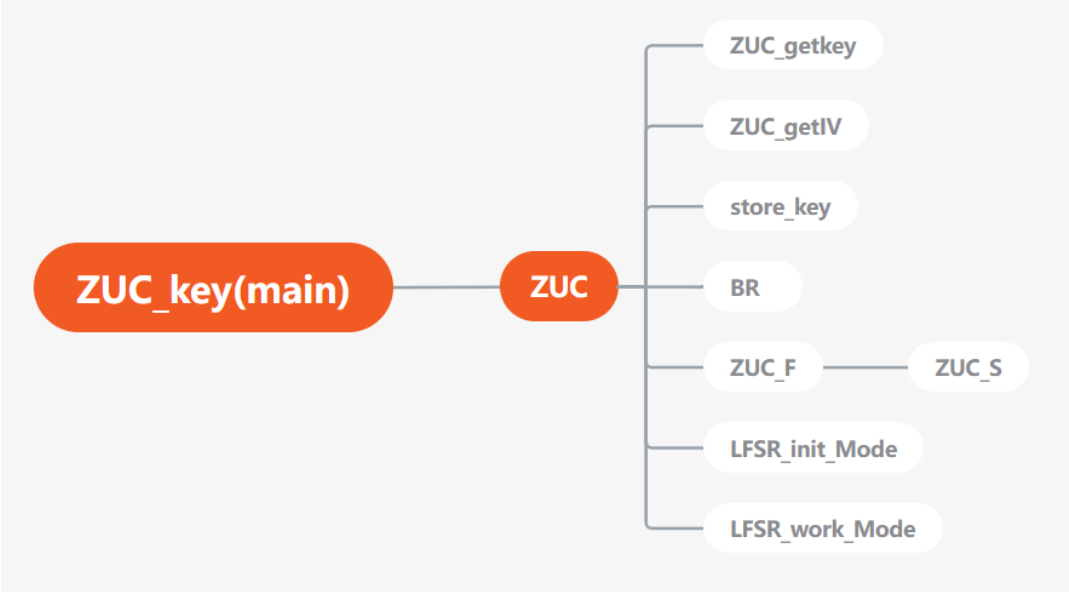
$$D = d_0 \| d_1 \| \cdots\cdots \| d_{15},$$

其中：

$$d_0 = 100010011010111_2,$$
$$d_1 = 010011010111100_2,$$
$$d_2 = 110001001101011_2,$$
$$d_3 = 001001101011110_2,$$
$$d_4 = 101011110001001_2,$$
$$d_5 = 011010111100010_2,$$
$$d_6 = 111000100110101_2,$$
$$d_7 = 000100110101111_2,$$
$$d_8 = 100110101111000_2,$$
$$d_9 = 010111100010011_2,$$
$$d_{10} = 110101111000100_2,$$
$$d_{11} = 001101011110001_2,$$
$$d_{12} = 101111000100110_2,$$
$$d_{13} = 011110001001101_2,$$
$$d_{14} = 111100010011010_2,$$
$$d_{15} = 100011110101100_2.$$

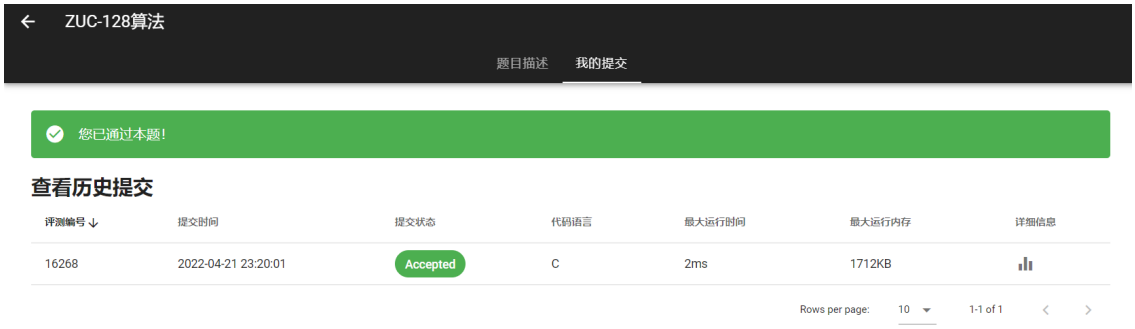
(2) 对 $0 \leqslant i \leqslant 15$，有 $s_i = k_i \| d_i \| iv_i$。

- **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

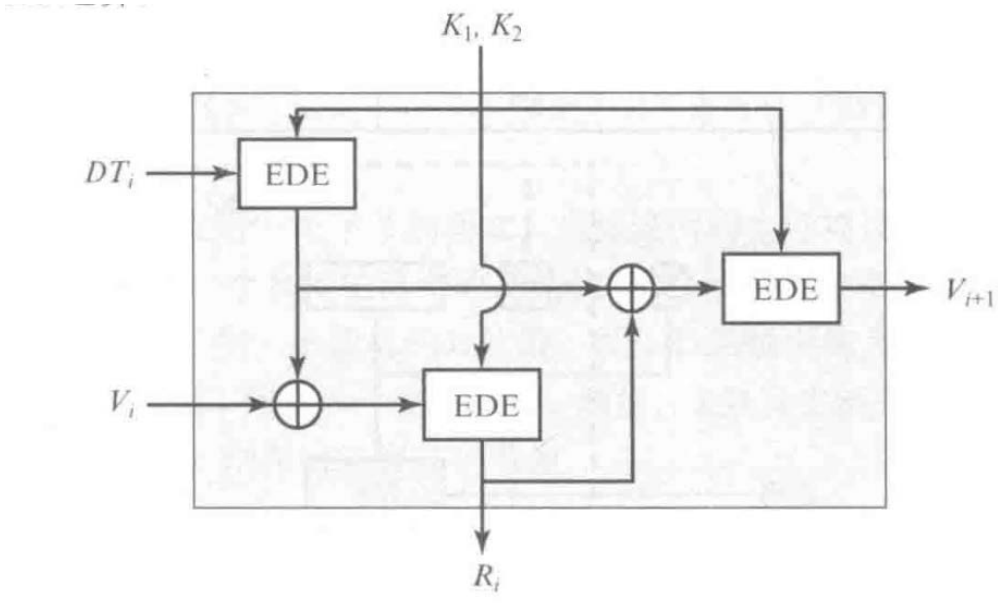


五、ANSI X9.17伪随机数发生器

1. **算法流程**

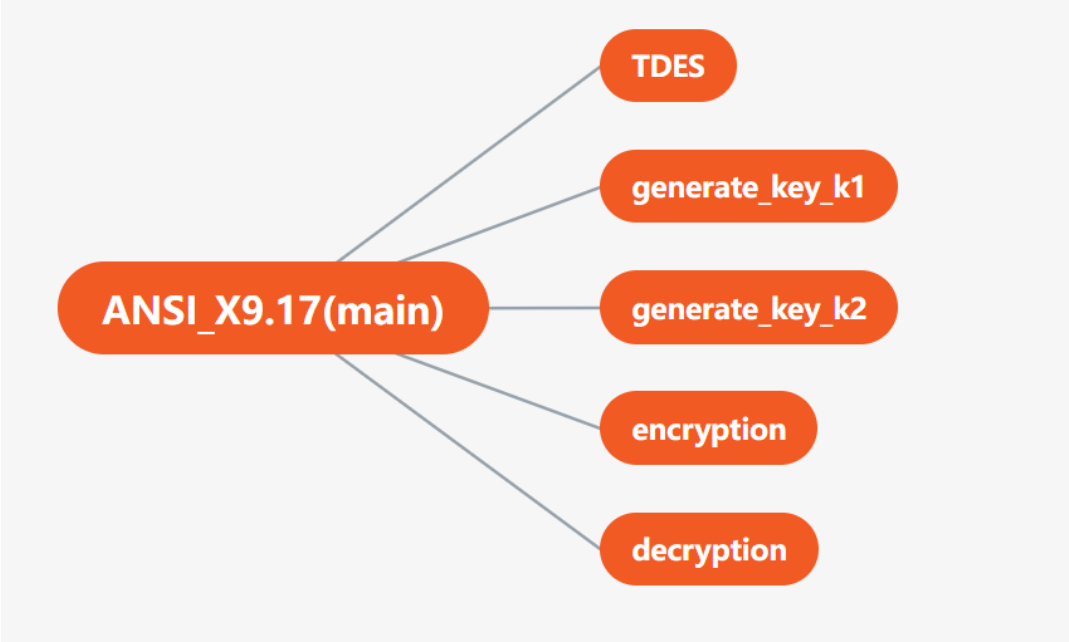
   - **流程图：**

     1. **流程总览：**



流程较为简单，其中嵌套了3个3DES函数来完成生成过程。

   - **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

【选做】ANSI X9.17伪随机发生器

题目描述 我的提交

您已通过本题!

查看历史提交

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
16962	2022-04-27 22:47:53	Accepted	C	20ms	1704KB	

Rows per page: 10 1-1 of 1