



SHANDONG UNIVERSITY

---

## 密码工程第二次实验报告

---

姓名: 谢子洋

学院: 网络空间安全学院 (研究院)

专业: 网络空间安全

学号: 202100460116

2023 年 11 月 28 日

# 目录

<b>1</b>	<b>实验原理</b>	<b>2</b>
1.1	Grain128 生成密钥流 . . . . .	2
1.2	Grain128 初始化 . . . . .	3
<b>2</b>	<b>实验过程</b>	<b>4</b>
2.1	模块输入输出 . . . . .	4
2.2	寄存器设初值 . . . . .	4
2.3	算法初始化 . . . . .	5
2.4	产生加密流 . . . . .	6
2.5	完整代码 . . . . .	7
<b>3</b>	<b>实验结果</b>	<b>8</b>
	<b>参考文献</b>	<b>10</b>
<b>A</b>	<b>Code</b>	<b>11</b>

# 1 实验原理

## 1.1 Grain128 生成密钥流

Grain128 流密码主要由三部分组成, 分别为 NFSR, LFSR, 输出函数.

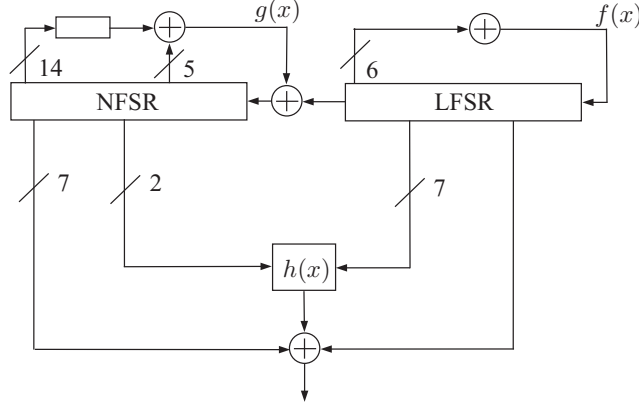


图 1: Grain128

1) NFSR 由  $b_i, b_{i+1}, \dots, b_{i+127}$  组成, 其非线性反馈生成式为:

$$\begin{aligned}
 b_{i+128} = & s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} \\
 & + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} + \\
 & + b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + \\
 & + b_{i+68}b_{i+84}.
 \end{aligned} \tag{1.1}$$

2) LFSR 由  $s_i, s_{i+1}, \dots, s_{i+127}$  组成, 其线性反馈生成式为:

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}. \tag{1.2}$$

3) 函数  $h(x)$  定义为

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8. \tag{1.3}$$

其中  $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$  依次对应  $b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79}, s_{i+95}$

4) 输出函数定义为:

$$z_i = \sum_{j \in \mathcal{A}} b_{i+j} + h(x) + s_{i+93}. \tag{1.4}$$

其中  $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$

## 1.2 Grain128 初始化

Grain128 算法在产生有效密钥流之前需要先进行初始化: 将 **key** 和 **IV** 加载到寄存器并进行 256 轮更新, 过程中输出函数要反馈到 **LFSR** 和 **NFSR**.

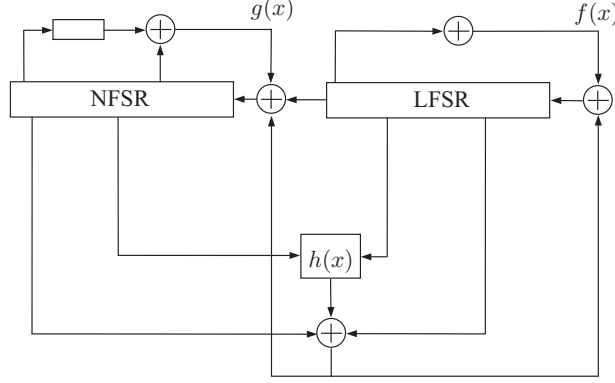


图 2: Grain128

1) 初始化寄存器初值将 **key** 各 bit 以  $k_i, 0 \leq i \leq 127$  表示, 将 **IV** 各 bit 以  $IV_i, 0 \leq i \leq 95$  表示.

对于 128bit 的 NFSR, 令

$$b_i = k_i, 0 \leq i \leq 127 \quad (1.5)$$

对于 128bit 的 LFSR, 令

$$s_i = \begin{cases} IV_i, & 0 \leq i \leq 95 \\ 1, & 96 \leq i \leq 127 \end{cases} \quad (1.6)$$

2) 进行初始化循环

在加载 **key** 和 **IV** 比特后, 密码算法进行 256 次而不产生任何密钥流。输出函数被反馈到 **LFSR** 和 **NFSR**, 并与输入异或产生新 bit.

for  $i = 0$  to 255

$$NFSR \leftarrow \{b_{i+1}, b_{i+2}, \dots, b_{i+128} \oplus z_i\} \quad (1.7)$$

$$LFSR \leftarrow \{s_{i+1}, s_{i+2}, \dots, s_{i+128} \oplus z_i\}$$

## 2 实验过程

### 2.1 模块输入输出

输入时钟 `clk` 控制密钥流生成速度, 每当时钟上升沿模块产生新密钥 `bit`. 输入 `key` 和 `IV` 唯一确定密钥流. 因为算法存在初始化阶段, 因此并非任意时刻的密钥流都是可用的, 本文通过添加有效标志 `valid` 输出加以区分, 只有当 `valid` 为 1 时输出密钥流才是有效的.

```
1 module Grain128(  
2     input clk,           // 输入时钟  
3     input [0:127] key,  // 输入key  
4     input [0:95] IV,    // 输入IV  
5     output wire stream, // 输出密钥流  
6     output valid        // 密钥流是否有效  
7 );  
8 endmodule
```

### 2.2 寄存器设初值

设计两个 128bit 寄存器分别代表 NFSR 和 LFSR。

```
1 reg [0:127]NFSR;  
2 reg [0:127]LFSR;  
3 wire[0:127]b,b_128;  
4 wire[0:127]s,s_128;  
5 assign b=NFSR;  
6 assign s=LFSR;
```

Grain128 流密码算法要求 NFSR 和 LFSR 在算法开始时分别设定规定初始值, 因此我们在 `initial` 语句块中为寄存器设定初始值, 并设定延时保证寄存器初始化完全.

```
1 initial begin  
2     #10;  
3     NFSR[0:127] =key[0:127];  
4     LFSR[0:95]  =IV[0:95];  
5     LFSR[96:127]={32{1'b1}};  
6 end
```

## 2.3 算法初始化

设定寄存器初始值过程中时序电路不可运行, 否则会导致算法流程错误.

因此使用阻塞语句 `wait()` 延后时序电路的执行, 保证在寄存器赋完初值后时序电路才开始运行.

```
1 reg init=1'b0;
2 initial begin
3     .....
4     init=1'b1;
5 end
6 always @(posedge clk) begin
7     wait (init) ;
8     .....
9 end
```

寄存器设定初始值后需要进行 256 轮以初始化, 过程中算法输出值要与寄存器输入值异或, 并作为新的输入更新寄存器. 因此本文添加寄存器"count" 用以计数, 初值设为 256, 每经一个时钟上升沿将该寄存器数值减 1, 直到寄存器数值为 0.

初期"valid" 为 0 表示输出密钥流无效, 每轮输出的 bit 要与反馈函数输出异或, 作为寄存器的新输入. 当"count" 寄存器数值为 0 时, 数字电路完成初始化, 输出信号"valid" 变为 1, 开始输出有效密钥流.

```
1 reg [8:0] count; //计数
2 assign valid=(~(count[0]|count[1]|count[2]|
3               count[3]|count[4]|count[5]|
4               count[6]|count[7]|count[8]));
5 initial begin
6     .....
7     count[8:0]=9'b1_0000_0000;
8 end
9 always @(posedge clk) begin
10     if(~valid) count=count-1;
11     .....
12     NFSR[127]=b_128^(valid?1'b0:resultBit);
13     LFSR[127]=s_128^(valid?1'b0:resultBit);
14 end
```

## 2.4 产生加密流

将电路设计抽象并简化, 使用软件绘制如下设计图.

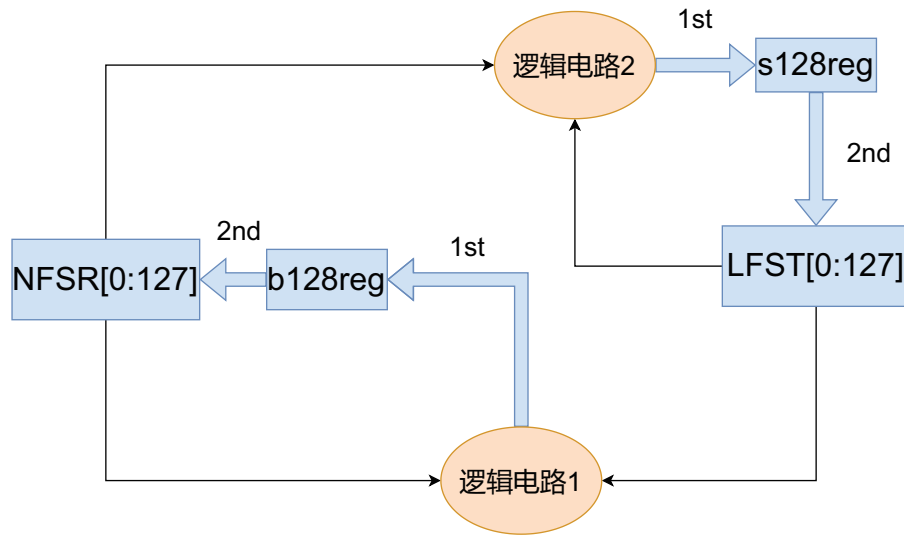


图 3: Grain128 内部更新设计

除 NFSR 和 LFSR 使用寄存器外, 其他部分均可用逻辑电路表示. 因此可使用 assign 语句将电路中所有逻辑电路表示出来.

```

1 assign b=NFSR;
2 assign s=LFSR;
3 assign b_128=s[0]^b[0]^b[26]^b[56]^b[91]^b[96]
4         ^ (b[3]&b[67]) ^ (b[11]&b[13]) ^ (b[17]&b[18])
5         ^ (b[27]&b[59]) ^ (b[40]&b[48]) ^ (b[61]&b[65])
6         ^ (b[68]&b[84]);
7 assign s_128=s[0]^s[7]^s[38]^s[70]^s[81]^s[96];
8 assign x={b[12],s[8],s[13],s[20],b[95],s[42],s[60],s[79],s[95]};
9 assign hx=(x[0]&x[1])^(x[2]&x[3])^(x[4]&x[5])^(x[6]&x[7])^(x[0]&x[4]&x[8]);
10 assign resultBit=hx^s[93]^b[2]^b[15]^b[36]^b[45]^b[64]^b[73]^b[89];
11 assign stream=resultBit&valid;

```

模块输入包含时钟 clk, 每当检测到 clk 的上升沿就进行一次寄存器的更新.

```

1 always @(posedge clk) begin
2     NFSR[0:126]=NFSR[1:127];
3     LFSR[0:126]=LFSR[1:127];
4     NFSR[127]=b_128^(valid?1'b0:resultBit);
5     LFSR[127]=s_128^(valid?1'b0:resultBit);
6 end

```

## 2.5 完整代码

最后给出完整代码:

```
1 module Grain128(  
2     input clk, // 输入时钟  
3     input [0:127] key,  
4     input [0:95] IV,  
5     output wire stream,  
6     output valid  
7 );  
8 reg init=1'b0;  
9 reg [8:0] count; // 计数  
10 reg [0:127] NFSR;  
11 reg [0:127] LFSR;  
12 wire [0:127] b, b_128;  
13 wire [0:127] s, s_128;  
14 wire [0:8] x;  
15 wire hx, resultBit;  
16  
17 assign b=NFSR;  
18 assign s=LFSR;  
19 assign b_128=s[0]^b[0]^b[26]^b[56]^b[91]^b[96]  
20         ^ (b[3]&b[67]) ^ (b[11]&b[13]) ^ (b[17]&b[18])  
21         ^ (b[27]&b[59]) ^ (b[40]&b[48]) ^ (b[61]&b[65])  
22         ^ (b[68]&b[84]);  
23 assign s_128=s[0]^s[7]^s[38]^s[70]^s[81]^s[96];  
24 assign x={b[12],s[8],s[13],s[20],b[95],s[42],s[60],s[79],s[95]};  
25 assign hx=(x[0]&x[1])^(x[2]&x[3])^(x[4]&x[5])^(x[6]&x[7])^(x[0]&x[4]&x[8]);  
26 assign resultBit=hx^s[93]^b[2]^b[15]^b[36]^b[45]^b[64]^b[73]^b[89];  
27 assign stream=resultBit&valid;  
28 assign valid=(~(count[0]|count[1]|count[2]|  
29             count[3]|count[4]|count[5]|  
30             count[6]|count[7]|count[8]));  
31  
32 initial begin  
33     #10;  
34     NFSR[0:127] =key[0:127];  
35     LFSR[0:95]  =IV[0:95];  
36     LFSR[96:127]={32{1'b1}};  
37     count[8:0]=9'b1_0000_0000;  
38     init=1'b1;  
39 end  
40 always @(posedge clk) begin  
41     wait (init) ;  
42     if(~valid) count=count-1;  
43     NFSR[0:126]=NFSR[1:127];  
44     LFSR[0:126]=LFSR[1:127];  
45     NFSR[127]=b_128^(valid?1'b0:resultBit);  
46     LFSR[127]=s_128^(valid?1'b0:resultBit);  
47 end  
48 endmodule
```



### 3 实验结果

论文 [1] 中给出了部分测试实例, 具体数值如下.

表 1: 测试实例

Key	00000000000000000000000000000000
IV	00000000000000000000000000000000
Keystream	0fd9deefeb6fad437bf43fce35849cfe
Key	0123456789abcdef123456789abcdef0
IV	0123456789abcdef12345678
Keystream	db032aff3788498b57cb894fffb6bb96

本文以此作为测试数据并编写了 Grain128 模块的 testbench, 具体代码如下.

```

1  `timescale 1ns / 1ps
2  `include "Grain128.v"
3  module tb_Grain128;
4      parameter PERIOD = 10;
5      reg clk = 0;
6      reg [0:127] key = 0;
7      reg [0:95] IV = 0;
8      wire stream;
9      wire valid;
10     Grain128 u_Grain128 (
11         .clk ( clk ),
12         .key ( key [0:127] ),
13         .IV ( IV [0:95] ),
14
15         .stream ( stream ),
16         .valid ( valid )
17     );
18     always #30 clk = ~clk;
19     initial begin
20         $dumpfile("wave.vcd");
21         $dumpvars;
22         clk=0;
23
24         key[0:127]={128{1'b0}};
25         IV[0:95]={96{1'b0}};
26         #30000;
27
28         // key[0:127]=128'h0123456789abcdef123456789abcdef0;
29         // IV[0:95]=96'h0123456789abcdef12345678;
30         // #30000;
31
32         $finish;
33     end
34 endmodule

```

运行 testbench, 得到测试结果如下:

### (1) 测试数据 1.

设定 key 和 IV 为全 0, 编译 testbench 并运行, 输出波形图如下:

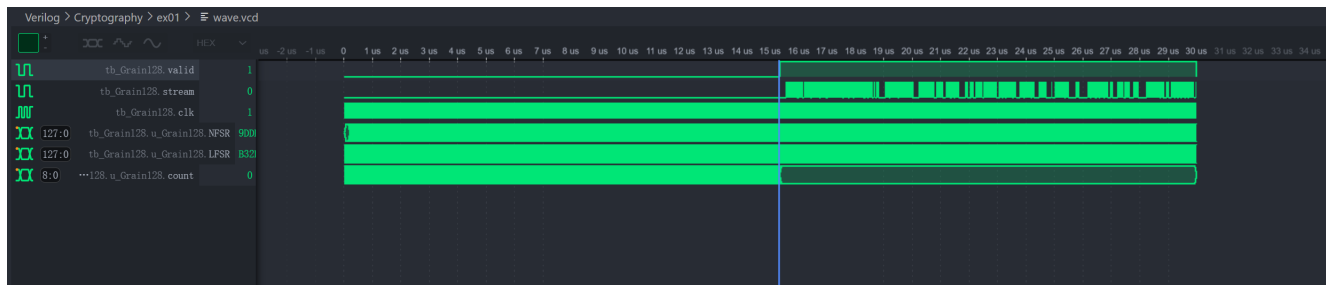


图 4: 波形图

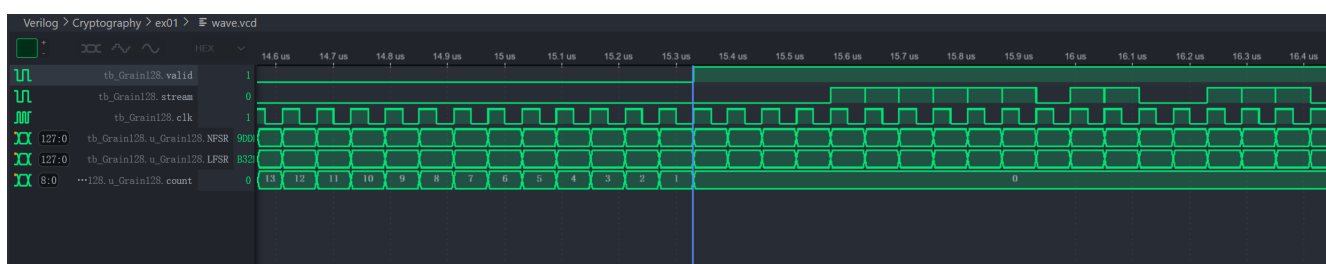


图 5: 波形图

初始条件下 valid 为 0, 表明该时段算法正在初始化. 可观察到当 count 寄存器数值变为 0 时, valid 变为 1, 表明该时段正在产生有效的密钥流. 观察"stream" 变量, 可知生成的密钥流为:

0b 0000 1111 1101 1001...

与正确结果保持一致.

### (2) 测试数据 2.

设定

Key=0123456789abcdef123456789abcdef0

IV =0123456789abcdef12345678

编译并运行 testbench, 得到波形图如下

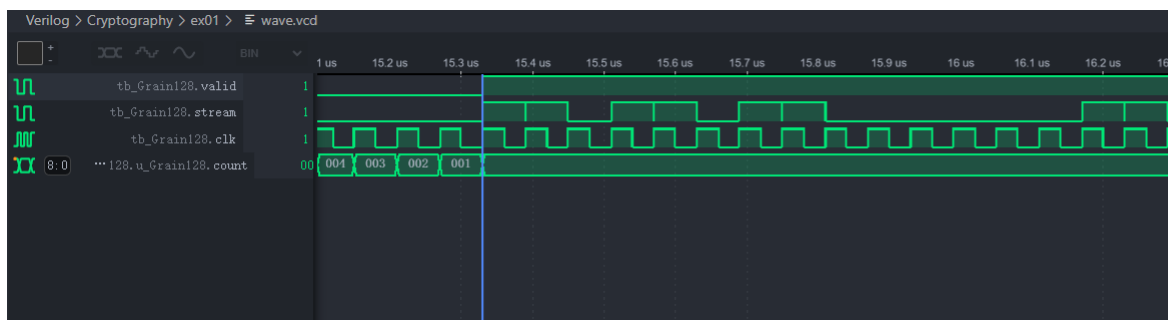


图 6: 波形图

观察可知产生的密钥流为:

*0b* 1101 1011 0000 0011...

与正确结果一致.

## 参考文献

- [1] M. Hell, T. Johansson, A. Maximov and W. Meier, "A Stream Cipher Proposal: Grain-128," 2006 IEEE International Symposium on Information Theory, Seattle, WA, USA, 2006, pp. 1614-1618, doi: 10.1109/ISIT.2006.261549.

## A Code

module.

```
1  module Grain128(  
2      input clk, // 输入时钟  
3      input [0:127] key,  
4      input [0:95] IV,  
5      output wire stream,  
6      output valid  
7  );  
8      reg init=1'b0;  
9      reg [8:0] count;//计数  
10  
11     reg [0:127]NFSR;  
12     reg [0:127]LFSR;  
13     wire[0:127]b,b_128;  
14     wire[0:127]s,s_128;  
15     wire [0:8]x;  
16     wire hx,resultBit;  
17  
18     assign b=NFSR;  
19     assign s=LFSR;  
20     assign b_128=s[0]^b[0]^b[26]^b[56]^b[91]^b[96]  
21         ^ (b[3]&b[67])^(b[11]&b[13])^(b[17]&b[18])  
22         ^ (b[27]&b[59])^(b[40]&b[48])^(b[61]&b[65])  
23         ^ (b[68]&b[84]);  
24     assign s_128=s[0]^s[7]^s[38]^s[70]^s[81]^s[96];  
25     assign x={b[12],s[8],s[13],s[20],b[95],s[42],s[60],s[79],s[95]};  
26     assign hx=(x[0]&x[1])^(x[2]&x[3])^(x[4]&x[5])^(x[6]&x[7])^(x[0]&x[4]&x[8]);  
27     assign resultBit=hx^s[93]^b[2]^b[15]^b[36]^b[45]^b[64]^b[73]^b[89];  
28     assign stream=resultBit&valid;  
29     assign valid=(~(count[0]|count[1]|count[2]|  
30         count[3]|count[4]|count[5]|  
31         count[6]|count[7]|count[8]));  
32  
33     initial begin  
34         #10;  
35         NFSR[0:127] =key[0:127];  
36         LFSR[0:95] =IV[0:95];  
37         LFSR[96:127]={32{1'b1}};  
38         count[8:0]=9'b1_0000_0000;  
39         init=1'b1;  
40     end  
41     always @(posedge clk) begin  
42         wait (init) ;  
43         if(~valid) count=count-1;  
44         NFSR[0:126]=NFSR[1:127];  
45         LFSR[0:126]=LFSR[1:127];  
46         NFSR[127]=b_128^(valid?1'b0:resultBit);  
47         LFSR[127]=s_128^(valid?1'b0:resultBit);  
48     end  
49 endmodule
```

testbench.

```
1  `timescale 1ns / 1ps
2  `include "Grain128.v"
3  module tb_Grain128;
4      parameter PERIOD = 10;
5      reg    clk                      = 0 ;
6      reg    [0:127] key              = 0 ;
7      reg    [0:95] IV               = 0 ;
8      wire   stream                  ;
9      wire   valid                   ;
10     Grain128 u_Grain128 (
11         .clk          ( clk          ),
12         .key           ( key          [0:127] ),
13         .IV            ( IV          [0:95] ),
14
15         .stream        ( stream       ),
16         .valid         ( valid        )
17     );
18     always#30 clk=~clk;
19     initial begin
20         $dumpfile("wave.vcd");
21         $dumpvars;
22         clk=0;
23
24         key[0:127]={128{1'b0}};
25         IV[0:95]={96{1'b0}};
26         #30000;
27
28         // key[0:127]=128'h0123456789abcdef123456789abcdef0;
29         // IV[0:95]=96'h0123456789abcdef12345678;
30         // #30000;
31
32         $finish;
33     end
34 endmodule
```