SHANDONG UNIVERSITY

密码工程第三次实验报告

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

1 实验原理

1.1 SIMON 分组密码

SIMON 是一种硬件实现友好的轻量化分组密码, 其分组长度和密钥长度灵活多样, 适用范围广.

表 1: SIMON family

block size 2n	key size mn	word size n	key words m	const seq	rounds T
32	64	16	4	z0	32
48	72	24	3	z0	36
	96	4	z1	36	
64	96	32	3	z2	42
	128	4	z3	44	
96	96	48	2	z2	52
	144	3	z3	54	
128	128	64	2	z2	68
	192	3	z3	69	
	256	4	z4	72	

本文选择 SIMON128/128 进行硬件实现.

1.2 SIMON 轮函数

SIMON 密码的轮函数只存在如下三种组件, 逻辑简单, 易于硬件实现.

- $\oplus$, 按位异或.
- &, 按位与.
- S^j, 循环左移 j bit.

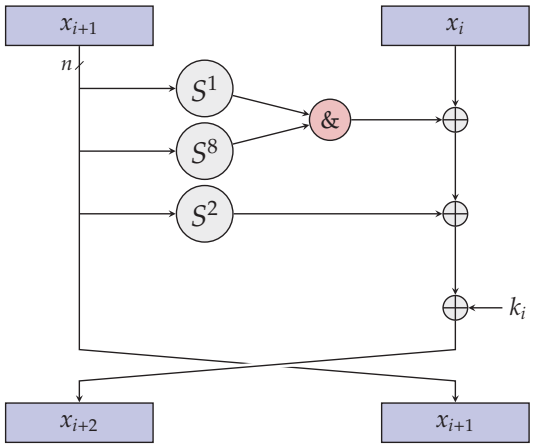

图 1: SIMON 轮函数

算法伪代码如下:

Algorithm 1 RoundFunction

Input:

x, y

1: **for** $i = 0 \cdots 67$ **do**
2: $tmp = x.$
3: $x = y \oplus (Sx \& S^8 x) \oplus S^2 x \oplus k[i].$
4: $y = tmp.$
5: **end for**

1.3 SIMON 密钥生成

SIMON 密钥拓展同样只用到了上述三种组件, 便于硬件实现. 首先将初始向量 IV 加载到寄存器中, 每轮更新寄存器 k_i 和 k_{i+1}, 以寄存器 k_i 中的值作为当前轮的轮密钥.

电路原理图如下:

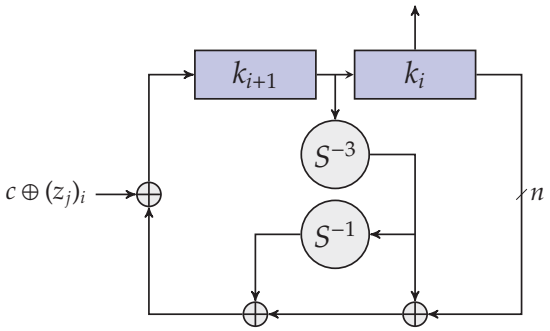

图 2: SIMON 密钥拓展

其中, 对于 SIMON128/128, 参数 z_j 选择 62bit 长的

$$z_2 = 10101111011100000011010010011000101000010001111110010110110011.$$

算法伪代码如下:

Algorithm 2 KeySchele

Input:

$k[i], 0 \le i \le 67$

1: **for** $i = 2 \cdots 67$ **do**
2: $tmp = S^{-3} k[i-1].$
3: $tmp = tmp \oplus S^{-1} tmp.$
4: $k[i] = k[i-2] \oplus tmp \oplus z_2[(i-2) mod 62] \oplus 3.$
5: **end for**

2 实验过程

2.1 模块输入输出

设定模块输入输出. 明文和密钥分别对应输入 plain 和 key. clk 信号控制加密速度, 每当 clk 经过上升沿时进行一次完整的 SIMON 加密. 加密后 128bit 密文输出到 cipehr. 输出 valid 表示是否已完成加密.

```
module Grain128(
  input clk,              // 输入时钟
  input [0:127] key,      // 输入key
  input [0:127] plain,    // 输入明文
  output[0:127] cipher,   // 输出密文
  output reg valid        // 输出是否有效
  );
```

2.2 轮函数

设定 128bit 寄存器 state, 用以保存每轮过论函数的结果, 初始值赋为明文, 经 68 次轮函数后值即为密文. 使用逻辑电路表示轮函数, 计算新的 state, 并拆分为两部分 L 和 R. 在每次过轮函数时, 利用一 64bit 临时寄存器更新 state 状态, 更新 68 次后得到最后的密文.

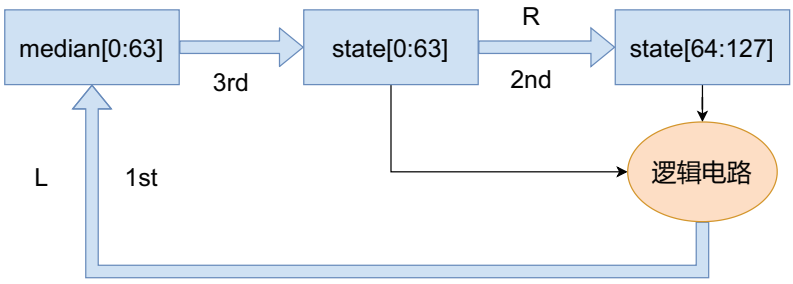

图 3: SIMON 轮函数

```
reg [0:127]state;
wire[0:63]L;
wire[0:63]R;
reg [0:63]median;
assign R=state[0:63];
assign L=({state[1:63],state[0]}
        &{state[8:63],state[0:7]})
        ^({state[2:63],state[0:1]})
        ^(state[64:127])
        ^(ki);
assign cipher=state;
integer i ;
always @(posedge clk) begin
        #10;
        valid=0;
```

```
16          {ki1,ki}=key;
17          state=plain;
18          for (i=0; i<68; i=i+1) begin
19                  #10 ;
20                  median=L;
21                  state[64:127]=R;
22                  state[0:63]=L;
23                  //......
24          end
25          valid=1;
26  end
```

2.3 密钥生成

2.3.1 针对硬件实现简化

原算法中使用 62bit 长的常二进制串 z_2, 在第 i 轮密钥拓展中使用

$$z_2[(i-2)\ mod\ 62],\ \ 2 \le i \le 67.$$

即

$$z_2[0], z_2[1], \cdots, z_2[60], z_2[61], z_2[0], z_2[1], z_2[2], z_2[3].$$

为简化设计并减少门电路开销, 本文将 62bit 长的 z_2 拓展到 66bit 长, 在原 z_2 基础上添加了 z_2 的前 4bit. 同时为了程序一般适用性, 又添加了 2bit 0 无效数据.

$$z_2 \Leftarrow \{z_2, z_2[0:3], 0b00\}$$

设定 68bit 寄存器"z2" 保存该常量.

2.3.2 轮密钥更新

设定 3 个 64bit 长寄存器 ki,ki1 和 ki2_reg, 前两者用来保存当前轮和下一轮的轮密钥, 而 ki2_reg 用以暂存数据更新前两者. 每次执行一轮轮函数更新 state 后, 还需要更新轮密钥 ki 和 ki1. 因此使用逻辑电路提前计算出下下轮的轮密钥, 即 ki2_wire, 在进行一轮时先更新 ki2_reg<=ki2_wire, 再依次更新 ki 和 ki1.

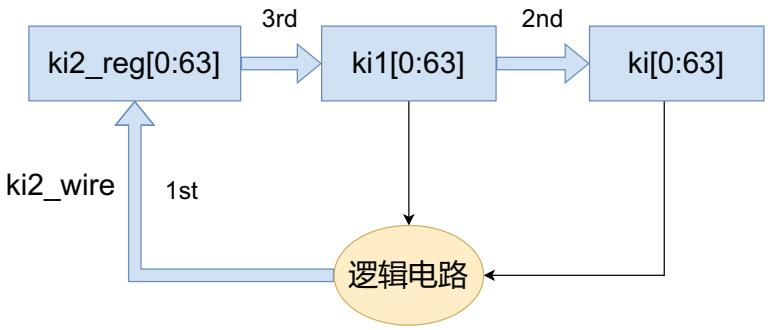

图 4: SIMON 密钥拓展

```
reg [0:67]z2=128'b10101111011100000011010010011000/
                  10100001000111111001011011001 1_1010_00;
reg [0:63]ki;
reg [0:63]ki1;
reg [0:63]ki2_reg;
reg zimm;
wire[0:63]temp;
wire[0:63]ki2_wire;
assign temp=({ki1[61:63],ki1[0:60]});
assign ki2_wire={temp[63],temp[0:62]}^temp
                ^(¬ki)
                ^{{62{1'b0}},2'b11}
                ^{{63{1'b0}},zimm};
integer i ;
always @(posedge clk) begin
        valid=0;     #10;
        {ki1,ki}=key;
        state=plain; #10;
        for (i=0; i<68; i=i+1) begin
                //......
                zimm=z2[i];#10 ;
                ki2_reg=ki2_wire;
                ki=ki1;
                ki1=ki2_reg;
        end
        valid=1;
end
```

2.4 完整实现

对于硬件实现, 密钥拓展和轮函数应同步进行, 以此节约硬件开销并提高运行效率.

首先定义好所有的连线 (wire 变量) 及其连续赋值, 硬件中连线的传播速度可近似认为无限快, 无需考虑等待.

随后我们只需考虑 always 语句中的过程性赋值, 因为寄存器赋值操作需要一定时间完成, 所以我们需要仔细考虑语句之间的同步问题. 在程序中我们使用时延操作确保硬件时序同步.

每当检测到'clk' 信号的上升沿,always 块内语句开始执行. 首先初始化寄存器 ki,ki1 和 state, 将初始向量 IV 和明文 plain 分别赋值到对应寄存器内. 随后执行 for 循环, 每次循环会先执行轮函数更新密文, 再对密钥进行更新. 经过 68 次循环后,state 寄存器中保存的值即为密文, 同时标志信号'valid' 会置为 1 表明加密结束.

综上给出完整代码:

```
//orgaworl@outlook.com
module Grain128(
input clk,          // 输入时钟
input [0:127] key, // 输入key
input [0:127] plain,    // 输入IV
output[0:127] cipher,// 输出密钥流
output reg valid       // 密钥流是否有效
```

```
);
// 1. 密钥生成部分
reg [0:67]z2=128'b10101111011100000011010010011000101000010001111110010110110011101000;
reg [0:63]ki;
reg [0:63]ki1;
reg [0:63]ki2_reg;
reg zimm;
wire[0:63]temp;
wire[0:63]ki2_wire;
assign temp=({ki1[61:63],ki1[0:60]});
assign ki2_wire={temp[63],temp[0:62]}^temp
                ^(¬ki)
                ^{{62{1'b0}},2'b11}
                ^{{63{1'b0}},zimm};

// 2. 轮函数部分
reg init=1'b0;
reg [0:127]state;
wire[0:63]L;
wire[0:63]R;
reg [0:63]median;
assign R=state[0:63];
assign L={{state[1:63],state[0]}
        &{state[8:63],state[0:7]})
        ^({state[2:63],state[0:1]})
        ^(state[64:127])
        ^(ki);
assign cipher=state;
// 3. 进行68轮更新
integer i ;
always @(posedge clk) begin
        // 3.1 初始化
        #10;
        valid=0;
        {ki1,ki}=key;
        state=plain;
        #10;
        for (i=0; i<68; i=i+1) begin
                // 3.2 先进行密文更新
                #10 ;
                median=L;
                state[64:127]=R;
                state[0:63]=L;
                // 3.3 再进行密钥更新
                zimm=z2[i];#10 ;
                ki2_reg=ki2_wire;#30 ;
                ki=ki1;#10 ;
                ki1=ki2_reg;#20 ;
        end
        valid=1;
end
endmodule
```

3 实验结果

已知 SIMON 算法的一对明密文及其加密密钥如下, 可用以验证 SIMON 硬件实现的正确性.

表 2: 测试实例

block size	128
key size	128
key	0x0f0e0d0c0b0a09080706050403020100
plaintext	0x6373656420737265c6c657661727420
ciphertext	0x49681b1e1e54fe3f65aa832af84e0bbc

编译并运行 SIMON 模块的 testench, 得到波形图如下所示.

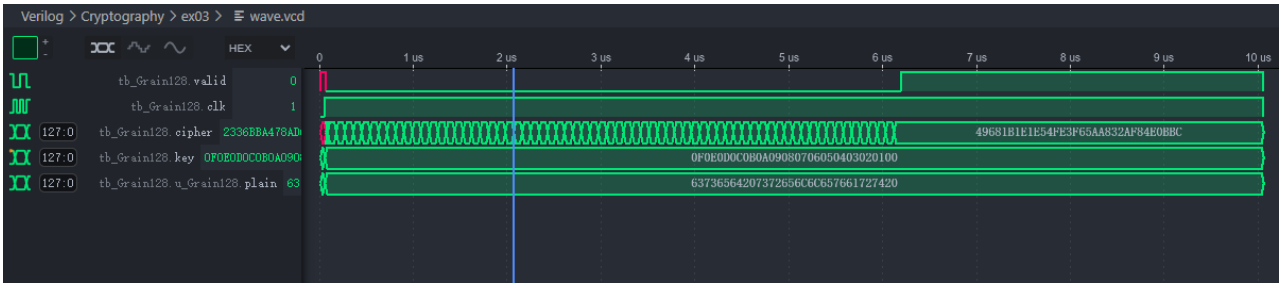

图 5: 波形图

观察硬件实现波形图可知, 当 clk 信号为上升沿时模块开始进行一次完整的 SIMON128/128 加密, 直到 valid 信号变为 1 说明加密结束.

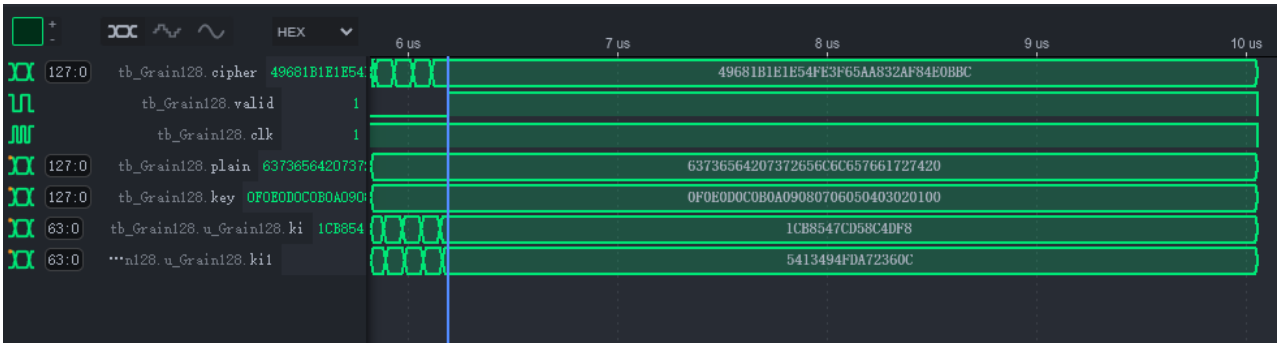

图 6: 波形图

此时查看 cipher 输出的值, 发现为 0x49681b1e1e54fe3f65aa832af84e0bbc, 和理论值相同.

参考文献

[1] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith and L. Wingers, "The SIMON and SPECK lightweight block ciphers," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2015, pp. 1-6, doi: 10.1145/2744769.2747946.

A code

module code.

```
1   //orgaworl@outlook.com
2   module Grain128(
3   input clk,          // 输入时钟
4   input [0:127] key, // 输入key
5   input [0:127] plain,   // 输入IV
6   output[0:127] cipher,// 输出密钥流
7   output reg valid       // 密钥流是否有效
8   );
9
10  // 1. 密钥生成部分
11  reg ...
        [0:67]z2=128'b10101111011100000011010010011000101000010001111110010110110011101000;
12  reg [0:63]ki;
13  reg [0:63]ki1;
14  reg [0:63]ki2_reg;
15  reg zimm;
16  wire[0:63]temp;
17  wire[0:63]ki2_wire;
18  assign temp=({ki1[61:63],ki1[0:60]});
19  assign ki2_wire={temp[63],temp[0:62]}^temp
20                  ^(¬ki)
21                  ^{{62{1'b0}},2'b11}
22                  ^{{63{1'b0}},zimm};
23
24  // 2. 轮函数部分
25  reg init=1'b0;
26  reg [0:127]state;
27  wire[0:63]L;
28  wire[0:63]R;
29  reg [0:63]median;
30  assign R=state[0:63];
31  assign L=({state[1:63],state[0]}
32          &{state[8:63],state[0:7]})
33          ^({state[2:63],state[0:1]})
34          ^(state[64:127])
35          ^(ki);
36  assign cipher=state;
37  // 3. 进行68轮更新
38  integer i ;
39  always @(posedge clk) begin
40          // 3.1 初始化
41          #10;
42          valid=0;
43          {ki1,ki}=key;
```

```
            state=plain;
            #10;
            for (i=0; i<68; i=i+1) begin
                    // 3.2 先进行密文更新
                    #10 ;
                    median=L;
                    state[64:127]=R;
                    state[0:63]=L;
                    // 3.3 再进行密钥更新
                    zimm=z2[i];#10 ;
                    ki2_reg=ki2_wire;#30 ;
                    ki=ki1;#10 ;
                    ki1=ki2_reg;#20 ;
            end
            valid=1;
    end
    endmodule
```

testbench code.

```
    //orgaworl@outlook.com
    `timescale  1ns / 1ps
    `include"SIMON.v"
    module tb_Grain128;
    parameter PERIOD  = 10;
    reg   clk                                  = 0 ;
    reg   [0:127]  key                         = 0 ;
    reg   [0:127]  plain                       = 0 ;
    wire  [0:127]  cipher                      ;
    wire  valid                                ;
    Grain128  u_Grain128 (
        .clk                    ( clk             ),
        .key                    ( key     [0:127] ),
        .plain                  ( plain   [0:127] ),

        .cipher                 ( cipher  [0:127] ),
        .valid                  ( valid           )
    );
    initial begin
        $dumpfile("wave.vcd");
        $dumpvars;
        clk=0;#50
        key=128'h0f0e0d0c0b0a09080706050403020100;
        plain=128'h63736564207372656c6c657661727420;
        clk=1;
        #10000;

        // clk=0;#50
        // key=128'h0f0e0d0c0b0a09080706050403020100;
        // plain=128'h63736564207372656c6c657661727420;
        // clk=1;#2500;
        $finish;
    end
    endmodule
```