说明文档

1. 指令步骤详细说明

主要实现的是 Simon64/128 和 Speck64/128，两种算法明文都是 64 位，分成左右两个部分，各 32 位。下文中的所有操作对象都是针对 32 位。

两种算法都用到异或（XOR）、按位与（AND），循环左移 1 位（S^1）、循环左移 2 位（S^2）、循环左移 8 位（S^8）在 Simon 加密中用到，循环左移 3 位（S^3）、循环右移 8 位（S^{-8}）、模 2^n 加在 Speck 加密中用到，Simon 密钥扩展用到循环右移 1 位（S^{-1}）、循环右移 3 位（S^{-3}）。

1.1 Bitwise XOR

1 个 cycle 可以完成 2 个字节的异或操作。完成 32 位异或操作，需要 4 条指令，4 cycles，8 字节 flash。

```
eor r8,  r0;
eor r9,  r1;
eor r10, r2;
eor r11, r3 ;
```

图1. 32 位异或操作

```
and r12, r3;
and r13, r0;
and r14, r1;
and r15, r2;
```

图2. 32 位与操作

1.2 Bitwise AND

完成 32 位与操作，需要 4 条指令，4 cycles，8 字节 flash。

1.3 Left circular shift, S^j, by j bits

1) S^1：循环左移 1 位可以通过逻辑左移（LSL）、循环左移（ROL）和带进位加法（ADC）实现。

LSL：寄存器最高位进入到标志寄存器的 C 位，低位向高位移动，最低位补 0。

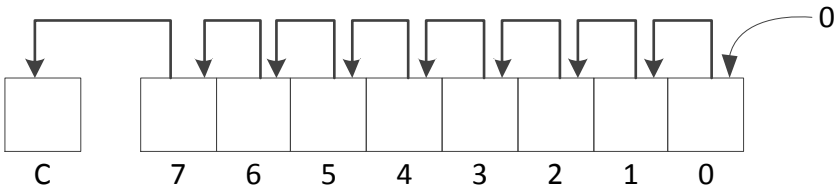

图3. LSL

ROL：最高位进入到标志寄存器的 C 位，低位向高位移动，C 位进入最低位。

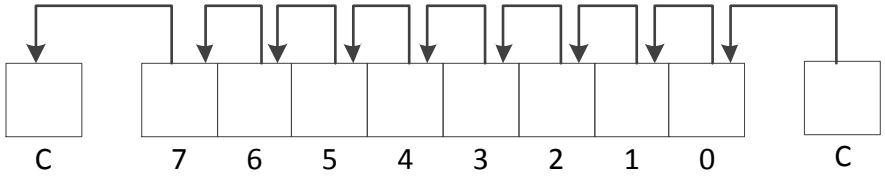

图4. ROL

ADC：标志寄存器的 C 位的值加到两个操作数的最低位。

通过一个 LSL、3 个 ROL 和一个 ADC 操作，可以实现 32 bits 循环左移 1 位。一共需要 5 条指令，5 个 cycles，10 字节 flash。

```
lsl r4;
rol r5;
rol r6;
rol r7;
adc r4, zero;
```

图5. S^1

2) S^2：通过 2 次 S^1 实现，共需要 10 条指令，10 个 cycles，20 字节 flash。

3) S^3：通过 3 次 S^1 实现，共需要 15 条指令，15 个 cycles，30 字节 flash；还可以通过乘法指令实现。

表 1 是实现[Y3,Y2,Y1,Y0]循环左移 3 bits 的过程，移位后的结果保存在[X3,X2,X1,X0]中。

表1. 乘法实现 S^3

operation			cycles
(R1, R0)	←	MUL(Y0, 8)	2
(X1, X0)	←	(R1, R0)	1
(R1, R0)	←	MUL(Y2, 8)	2
(X3, X2)	←	(R1, R0)	1
(R1, R0)	←	MUL(Y1, 8)	2
X1	←	X1 EOR R0	1
X2	←	X2 EOR R1	1
(R1, R0)	←	MUL(Y3, 8)	2
X3	←	X3 EOR R0	1
X0	←	X0 EOR R1	1

图 6 是通过乘法指令实现 S^3 的示意图。

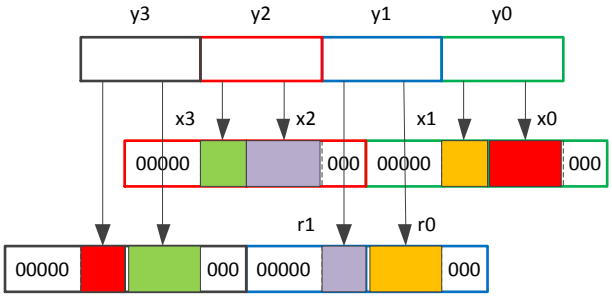

图6. 乘法实现 S^3 示意图

使用乘法指令实现一共需要 10 条指令，14 个 cycles，20 字节 flash。相比直接通过 3 次 S^1 实现，执行时间和执行数目都要少。但由于乘法指令的结果只能保持在 R1:R0，因此这两个寄存器不能用于其它用途。同时，乘法指令执行后的结果并不是在原寄存器中，如果要保持乘法指令带来的优势，则必须通过循环展开抵消这种错位，这样代码量会有所增加。

4) S^8

理论上实现循环左移 8 bits 需要 5 条移位指令。但由于 Atmega128 的寄存器都是 8 位的，在实际的运算中可以错位选择相应的寄存器参与运算，而不需要通过真正的移位操作。因此，循环左移 8 位不需要执行时间。

1.4 Right circular shift, S^{-j}, by j bits

1) S^{-1}：循环右移 1 位可以通过逻辑右移（LSR）、循环右移（ROR）、位存储（BST）和位加载（BLD）操作实现。

LSR：寄存器最高位补 0，高位向低位移动，最低位进入到标志寄存器的 C 位。

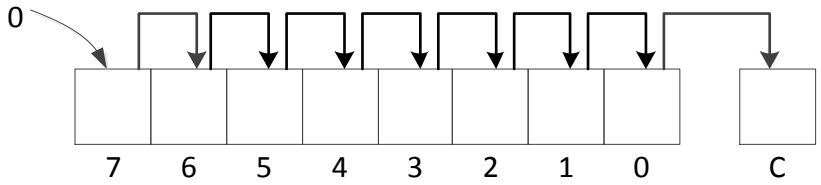

图7. LSR

ROR：C 位进入寄存器最高位，高位向低位移动，最低位进入到标志寄存器的 C 位。

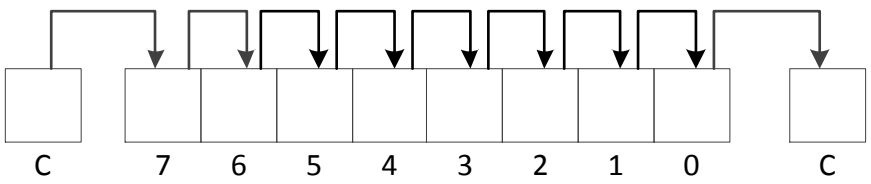

图8. ROR

BST：将寄存器的指定一个 bit 存放到标志寄存器的 T 位。

BLD：将寄存器的指定一个 bit 置为标志寄存器 T 位的值。

假设[R9,R8,R7,R6]4 个寄存器存放 32 bit 的值，则通过下面的 6 条指令可以实现循环右移 1 位。

```
lsr r9;
ror r8;
ror r7;
bst r6, 0;
ror r6;
bld r9, 7;
```

图9. S^{-1}

S^{-1} 需要 6 条指令，共 6 个 cycles，12 字节 flash。

2) S^{-3}：通过 3 次 S^{-1} 实现，共需要 18 条指令，18 个 cycles，36 字节 flash。

3) S^{-8}：和 S^{8} 的原理一样，不需要执行时间。

1.5 Modular addition

模 2^n（n 为一个分组所包含比特位的数目，本文中都是 32）的加法和一般的加法基本一致，不同之处在于它会舍弃最高位的进位。因此，对于 32 bits 的模 2^n 加法可以通过 4 条加法指令实现。

下面 6 条指令是实现[X3,X2,X1,X0]和[Y3,Y2,Y1,Y0]模 2^n 加法的指令。一共需要 4 个 cycles、8 字节 flash。

```
add X0, Y0;
adc X1, Y1;
adc X2, Y2;
adc X3, Y3;
```

图10. 模 2^n 加

1.6小结

表 2 是各个操作消耗的时间、RAM 和 Flash 汇总。

表2. 基本操作时间空间消耗

Operation	Cycles	RAM	Flash
XOR	4	0	8
AND	4	0	8
S^1	5	0	10

S^2	10	0	20
S^3	15	0	30
	14	0	20
S^8	0	0	0
S^{-1}	6	0	12
S^{-3}	18	0	36
S^{-8}	0	0	0
+	4	0	8

2. 算法比较

主要包括低 RAM（Minimal RAM）和高吞吐率（High-Throughput）两类。

1) **Minimal RAM**

Minimal RAM 侧重于消耗更少的 RAM。算法将轮密钥写在 Flash 中，加密时直接从 Flash 加载。

适用需要加密数据比较少的场景。由于数据量不是很大，从 Flash 加载比 RAM 每一轮只多 4 个 cycles；但数据基本不消耗 RAM，可以为其它应用节省很多空间。

2) High-Throughput

当数据量比较大时，继续从 Flash 加载将会消耗比较多的时间，因此将轮密钥放在 RAM 中。但初始密钥仍存放 Flash 中，经过密钥扩展程序计算后的轮密钥保存在 RAM 中供程序使用；或者直接将子密钥放在 Flash 中，真正开始加密数据时将子密钥从 Flash 加载到 RAM。同时，将循环进行展开，减少循环控制语句的执行时间。

适用于需要加密的数据量比较大的场景，通过空间来换取时间。

2.1 Simon Minimal RAM

包括从 RAM 加载明文，从 Flash 加载轮密钥，使用轮密钥进行加密以及将最终的密文放回 RAM 的所有时间；不包括明文的初始化。

1) 轮函数：一轮所需要的 cycles 是 42，指令条数是 34。表 3 是 Simon 一轮的执行时间及指令条数。

表3. 一轮 Simon 指令

Operation	Cycles	Instructions
K ← k	12	4
K ← K EOR Y	4	4

Y ← X	2	2
X ← S^1(X)	5	5
T ← X	2	2
T ← T & S^8(Y)	4	4
X ← S^1(X)	5	5
X ← X EOR T	4	4
X ← X EOR K	4	4

2) 循环控制

由于轮函数指令只有 34 条，小于 64，故循环控制只需要 3 条指令即可，分别是：增加已经循环的轮数；将已经循环的轮数和总轮数比较；条件跳转。

```
    inc currentRound;
    cp currentRound, totalRound;
brne loop;
```

图11. Simon Minimal RAM 循环控制

循环控制需要 3 条指令，共 3 cycles。

3) 明文加载密文写入

明文和密文都是 64 bits，每次加载（或写入）只能完成 8 bits，因此加载明文需要 8 条指令，写入密文需要 8 条指令，一共 16 条指令。从 RAM 加载数据、写入数据到 RAM 需要 2 个 cycles。故总时间为 32 cycles。

4) 初始化

初始化包括当前轮数置 0，总轮数设置，带进位加法目的寄存器置 0，加载明文、加载密钥的寄存器获取相应地址。如图 12，一共是 7 条指令。

```
clr currentRound ; set 0, have done rounds ; 1 cycle
ldi totalRound, 44; the total rounds ; 1 cycle
clr zero; 1 cycle
; move the address of plaintext to register X
ldi r26, low(plainText) ; 1 cycle
ldi r27, high(plainText) ;  1 cycle
ldi r30, low(keys) ; z is the current address of keys
ldi r31, high(keys) ;
```

图12. Simon Minimal RAM 初始化

由于系统在初始化的时候，所有寄存器默认是为 0 的。同时，数据段中存放数据地址从 0x0100 开始，代码段中存放数据的地址从 0x0000 开始，所以简化后的代码如下，只需要 2 条指令。

```
ldi totalRound, 44; the total rounds ; 1 cycle
ldi r27, 0x01;
```

图13.　简化初始化指令

但采用简化的代码会依赖系统初始化的结果，而且如果系统在运行一段时间后再调用本程序则可能出现问题。因此，为了保证结果的绝对正确性，采用最保守的写法，即 7 条指令。

5)　程序返回

程序返回需要一条 ret 指令，共 4 个 cycles。

- 总时间 T：

$$T = (42+3)\times 44+32+7+4 = 2023 cycles$$

- 加密效率 V：

$$V = T \div 8 \approx 253 cycles / byte$$

- Flash：

$$S_{flash} = (34+3+16+7+1)\times 2+44\times 4 = 298 bytes$$

- RAM：不考虑明文和密文占用的 RAM，RAM 消耗为 0。

2.2 Simon High-Throughput

和 Minimal RAM 不同，High-Throughput 加密是轮密钥是从 RAM 中加载的。轮密钥可以直接放在 Flash 中，在加密是先加载到 RAM；或者通过密钥扩展程序产生后直接放在 RAM 中。为了进一步减少时间，可以通过循环展开来减少循环控制语句的执行次数。文中进行了 4 轮循环展开。

2.4.1　Without Key Schedule

1)　一次循环

每一轮的密钥从 RAM 中加载，加载 32 bits 轮密钥需要 8 cycles，因此一轮的时间是(42-4=)38 cycles，指令条数仍为 34。由于进行了 4 轮展开，因此一次循环的总时间是(38*4=)152 cycles，指令条数是(34*4=)136。

2)　循环控制

展开 4 轮后，指令条数超过 64，需要借助无条件跳转语句来控制循环，循环控制指令需要 4，因此需要 4 cycles。

3)　其它

初始化、明文加载和密文写入已经程序返回的指令和 Minimal RAM 一样。

- 总时间 T：

$$T = (152+4)\times 11+32+7+4 = 1759 cycles$$

- 加密效率 V：

$$V = T \div 8 \approx 220 cycles / byte$$

- Flash：包括指令和数据。

$$S_{flash} = (136+4+16+7+1)\times 2+44\times 4 = 328 bytes$$

- RAM：由于不考虑明文和密文占用的 RAM，因此 RAM 消耗为轮密钥占用的空间。

$$S_{RAM} = 44\times 4 = 176 bytes$$

2.4.2 With Key Schedule

加密部分和 Without Key Schedule 的情况一样，只是增加了密钥扩展指令。

相比加密大量数据所需要的时间，密钥扩展的时间比较小。因此时间计算不不包括密钥扩展的时间，仍为 220cycles/byte。加密指令条数不变，需要的 Flash 仍为 328bytes。只是不需要 176bytes 来保存轮密钥，而是通过密钥扩展程序实现。

1) 扩展一轮，表 4 对应扩展 1 轮的指令描述，一共需要 63 条指令，共 63cycles

表4. 密钥扩展一轮

Operation	Cycles	Instructions
$K_i \leftarrow k_i$	4	4
$K_{i+1} \leftarrow k_{i+1}$	4	4
$K_{i+3} \leftarrow k_{i+3}$	5	5
$K_{i+3} \leftarrow S^{-3}(K_{i+3})$	18	18
$K_{i+3} \leftarrow K_{i+3}$ EOR K_{i+1}	4	4
$K_{i+1} \leftarrow K_{i+3}$	2	2
$K_{i+1} \leftarrow S^{-1}(K_{i+1})$	6	6
$K_{i+3} \leftarrow K_{i+3}$ EOR K_{i+1}	4	4
$K_i \leftarrow K_i$ EOR K_{i+3}	4	4
$K_i \leftarrow K_i$ EOR C	4	4
$K_i \leftarrow K_i$ EOR $(Z_3)_i$	4	4
$X \leftarrow K_i$	4	4

2) 循环控制：常量 Z 的周期是 62，分为 8 个字节存储在内存中，每循环 8 次后需要重新加载常量 Z，同时要控制循环次数不超过 44。整个控制共

需要 11 条指令。

3) 初始化
   - 寄存器：8 条指令；
   - 16 字节初始密钥由 Flash 加载到 RAM：9 条指令；
   - 常量 C 初始化：4 条指令；
   - 常量 Z 初始化：22 条指令。
4) 初始密钥：16 字节初始密钥保存在 Flash 中
5) 8 字节常量 Z 保存在 RAM 中

- Flash：包括指令和数据。

$$S_{flash} = 328 + (63+11+8+9+4+22) \times 2 + 16 = 578 bytes$$

- RAM：由于不考虑明文和密文占用的 RAM，因此 RAM 消耗为轮密钥占用的空间。

$$S_{RAM} = 44 \times 4 + 8 = 184 bytes$$

2.3 Speck Low-RAM

轮密钥放在 Flash 中，每轮从 Flash 加载轮密钥然后加密，最终将密文保存在 RAM。

1) 轮函数：一轮需要指令 33 条，共 41 cycles。

表5. 一轮 Speck 指令

Operation	Cycles	Instructions
K ← k	12	4
X ← $S^8(S^{-8}(X) + Y)$	4	4
K ← K EOR $S^{-8}(X)$	4	4
Y ← $S^3(Y)$	15	15
Y ← Y EOR K	4	4
X ← K	2	2

2) 循环控制：一轮指令条数小于 64，只需要 3 条指令即可控制循环。
3) 明文加载密文写入：明文加载、密文写入个 8 条指令，共 16 条指令，32 cycles。
4) 初始化：7 条指令，7 个 cycles。
5) 程序返回：一条 ret 指令，4 个 cycles。

- 总时间 T：

$$T=(41+3)\times 27+32+7+4=1231cycles$$

- 加密效率 V：

$$V=T\div 8\approx 154cycles/byte$$

- Flash：包括指令和数据。

$$S_{flash}=(33+3+16+7+1)\times 2+27\times 4=228bytes$$

- RAM：RAM 为 0。

2.4 Speck Faster Low-RAM

1) 循环一次：在一次循环中执行两次轮函数，可以消除一轮中最后的转移指令。因此，一次循环需要(39*2=)78cycles、指令(31*2=)62 条。整个加密需要 27 轮，每次循环执行 2 轮，最后 1 轮单独做。
2) 循环控制：一次循环共(32*2=)62 条指令，加上循环控制的 2 条指令，一共 64 条，而条件跳转语句的范围不能超过 63，因此需要借助无条件跳转语句。整个循环控制指令共 4 条，需要 4 cycles。
3) 其它：16 条明文加载密文写入指令，共 32cycles；7 条初始化指令，共 7cycles；1 条程序返回指令，共 4cycles；最后一轮单独执行，需要 31 条指令（可以直接将密文写入 RAM，而不用先转移，所以不是 33 条指令），共 39cycles。

- 总时间 T：

$$T=(39\times 2+4)\times 13+39+32+7+4=1148cycles$$

- 加密效率 V：

$$V=T\div 8\approx 144cycles/byte$$

- Flash：包括指令和数据。

$$S_{flash}=(62+4+16+7+1+31)\times 2=242bytes$$

- RAM：RAM 为 0。

2.5 Speck High-Throughput

没有密钥扩展程序，轮密钥直接放在 Flash。加密前，密钥转移到 RAM，加密过程中直接从 RAM 加载轮密钥。循环左移 3 位通过乘法指令实现。

1) 轮函数：由 1.3 节可知，乘法指令实现循环左移 3 位，需要 10 条指令，14cycles。乘法指令实现 Speck 一轮的情况如表 6：

表6. 乘法指令实现 Speck 一轮

Operation	Cycles	Instructions
K ← k	8	4
X ← $S^8(S^{-8}(X) + Y)$	4	4
K ← K EOR $S^{-8}(X)$	4	4
X ← $S^3(Y)$	14	10
X ← X EOR K	4	4

则乘法指令实现一轮需要 26 条指令，34cycles。由 3.1 节可知，乘法指令的结果并不是保存在原寄存器中。需要进行展开才能消除，通过 3 轮展开即可完全恢复由于乘法指令和减少移位指令带来的错位。这样实际上循环一轮需要执行轮函数 3 轮，指令(26*3=)78 条，时间(34*3=)102 cycles。

2) 循环控制：循环一轮的指令条数超过了 64，循环控制需要 4 条指令，共 4cycles。

3) 其它：16 条明文加载密文写入指令，共 32cycles；8 条初始化指令（乘法指令的被乘数 8 需要通过一条执行放到寄存器汇总），共 8cycles；1 条程序返回指令，共 4cycles。

- 总时间 T：

$$T = (34 \times 3 + 4) \times 9 + 32 + 8 + 4 = 998 cycles$$

- 加密效率 V：

$$V = T \div 8 \approx 125 cycles / byte$$

- Flash：包括指令和数据。

$$S_{flash} = (26 \times 3 + 4 + 16 + 8 + 1) \times 2 + 27 \times 4 = 322 bytes$$

- RAM：存放轮密钥。

$$S_{RAM} = 27 \times 4 = 108 bytes$$

2.6 小结

各种算法的执行时间，RAM 和 Flash 消耗情况如表 7。

表7. 算法时间空间对比

Algorithm	cycles/byte	RAM	Flash
Simon Minimal RAM	253	0	298

Simon High-Throughput	Without Key Schedule	220	176	328
	With Key Schedule	220	184	578
Speck Low-RAM		154	0	228
Speck Faster Low-RAM		144	0	242
Speck High-Throughput		125	108	322

图 14 为算法结果的图形表示。其中 SMR、SHT Without、SHT With、SLR、SFLR 和 SHT 分别对应 Simon Minimal RAM、Simon High-Throughput Without Key Schedule、Simon High-Throughput With Key Schedule、Speck Low-RAM、Speck Faster Low-RAM 和 Speck High-Throughput。

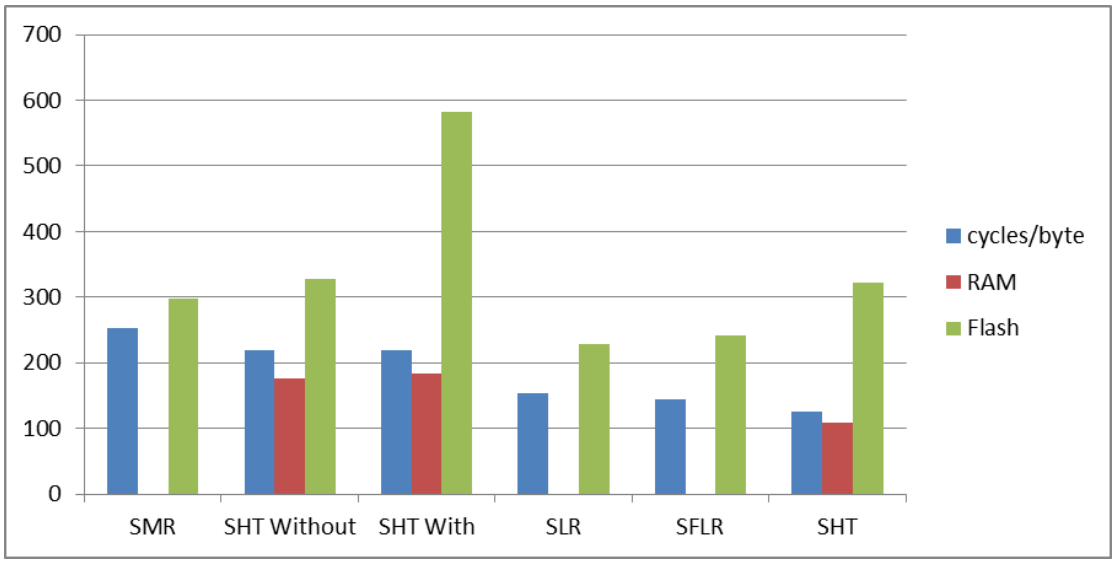

图14.　　算法时间空间对比

通过结果分析，可以得出以下结论：

- 加密数据量较少的情况下，使用 Minimal RAM；加密数据量较大的情况且 RAM、Flash 充足时，使用 High-Throughput 方法；
- 由于 Speck 加密轮数比 Simon 要小很多，使得它的性能要优于 Simon，加密速度上比 Simon 优 39.13%~43.18%；RAM 要优 38.64%；Flash 消耗要优 1.83%~18.79%；
- Simon 的密钥扩展程序过于复杂，使用密钥扩展比直接将轮密钥写入 Flash 消耗的 Flash 要多 76.22%；
- Minimal RAM 和 High-Throughput 两种方法的不足之处在于：无论是直接将轮密钥写在 Flash，还是通过扩展 Flash 中的初始密钥计算最终轮密钥，多次加密都是使用的相同的密钥，使用时间长了安全性会下降。