一种 AVR 环境下 KLEIN 分组密码抗计时和缓存边信道攻击的快速保护方法

温雅敏[1] 黎凤霞[2] 龚 征[3,4] 唐韶华[2]

(广东财经大学数学与统计学院 广州 510320)[1] (华南理工大学计算机科学与工程学院 广州 510641)[2]
(华南师范大学计算机学院 广州 510631)[3] (上海市信息安全综合管理技术研究重点实验室 上海 200240)[4]

摘 要 随着物联网应用的广泛兴起,轻量级分组密码算法在资源受限环境下的应用前景得到了广泛关注。在物联网应用中,攻击者往往采用边界信道的方式对相应设备进行密钥恢复攻击。在 RFIDSec 2011 会议上,Gong 等人提出了一种新的适用于物联网资源环境下软件实现的轻量级分组密码算法 KLEIN[1]。从 AVR 微处理器的特点出发,基于 AVR 汇编语言给出了 KLEIN 分组加密算法的 bitslicing 实现。在实现过程中,分别基于读取和存储操作进行相应的优化,降低了算法在 MixNibbles 步骤中的计算复杂度,从而使得 KLEIN 算法能通过 bitslicing 方式对计时(Timing)和缓存(Cache)边信道攻击方式进行防御。从 AVR 平台的实际试验结果来看,优化后的 KLEIN 算法的 bitslicing 实现在 AVR 微处理器平台上具有实用性。

关键词 物联网,边信道攻击,轻量级分组密码,KLEIN,Bitslicing

中图法分类号 TP309.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.3.031

Fast Implementation of KLEIN for Resisting Timing and Cache Side-channel Attacks on AVR

WEN Ya-min[1] LI Feng-xia[2] GONG Zheng[3,4] TANG Shao-hua[2]

(School of Mathematics and Statistics, Guangdong University of Finance & Economics, Guangzhou 510320, China)[1]
(School of Computer Science and Engineering, South China University of Technology, Guangzhou 510641, China)[2]
(School of Computer Science, South China Normal University, Guangzhou 510631, China)[3]
(Shanghai Key Laboratory of Information Security Integrated Technology, Shanghai 200240, China)[4]

Abstract With the rapid development of IoT (Internet of Things) applications, lightweight block ciphers are widely focused in the applications of resource-constrained environments. In IoT applications, attackers often use side-channel information to recover secret keys. At RFIDSec 2011, Gong et al. proposed a new lightweight block cipher named KLEIN for the software implementation in resource-constrained environments. We proposed a bitslicing implementation of the KLEIN block cipher based on AVR ASM. In the implementation, look-up tables and logical operations are combined for reducing the computational costs in the MixNibbles step, which leads to a better balance between the algorithm's speed and storage. Our experiments on AVR show the bitslicing implementation of KLEIN is feasible for practical applications.

Keywords IoT, Side-channel attack, Lightweight block cipher, KLEIN, Bitslicing

1 引言

随着物联网(Internet of Things)应用的不断发展,相关信息安全问题也得到了越来越多的重视。轻量级密码学算法,特别是轻量级分组密码算法[1-5],由于可以在低功耗环境下保证数据的机密性和完整性,因此在物联网安全中起着非常重要的作用。由于密码学算法分析与设计技术的成熟,采用传统密码学分析方法对于相关应用的实际威胁往往较小,但攻击者可以通过能量、时间或存储上所产生的边信道信息进行攻击,在实际中往往更加具有威胁性[6,7]。

bitslicing 实现技术[8]最初用于 DES 分组加密算法的边信道(Side-channel)保护。在 bitslicing 实现中,每一个输入分组都将基于比特的方式来进行计算操作。由于每一个比特的处理均由相同的处理步骤得到最终输出,算法实现在抵抗计时(Timing)和缓存(Cache)边信道攻击[9-12]上具有非常好的安全性。虽然在安全性上得到提高,但由于 bitslicing 技术将

到稿日期:2014-04-18 返修日期:2014-07-20 本文受国家自然科学基金项目(61300204,61100201,61170080,U1135004),广东省自然科学基金(S2012040006711),广东省高等学校优秀青年教师培养计划项目(Yq2013051),广州市科技计划项目珠江科技新星专项(2014J2200006),广东省教育厅高校优秀青年创新人才培育项目(2012LYM_0066)资助。

温雅敏(1981-),女,博士,副教授,主要研究方向为密码学与信息安全,E-mail:yamin.wen@gmail.com;黎凤霞(1989-),女,硕士生,主要研究方向为多变量公钥密码体制;龚 征(1981-),男,博士,副教授,主要研究方向为信息系统安全,E-mail:cis.gong@gmail.com;唐韶华(1970-),男,博士,教授,博士生导师,主要研究方向为多变量公钥密码体制。

以往基于分组的计算操作转变为基于比特，在软件实现上将会大大增加计算复杂度，因此它往往只用于高性能并行化设备上的分组密码算法抗边信道攻击实现[6]。由于物联网设备价格低廉，设备的计算与存储开销均受限制，因此如何在相应的低功耗设备上给出分组密码算法的抗计时或缓存边信道攻击实现，在当前学术界与工业界仍是热点问题[13-16]。

爱特梅尔(ATMEL)公司设计并制造的 AVR 系列微处理器具有低功耗、成本低、开发环境友善等优点，在物联网领域得到了广泛的应用。本文从 AVR 微处理器的特点出发，基于 AVR 汇编语言给出了面向物联网应用的轻量级分组加密算法 KLEIN[1]的 bitslicing 实现及相关优化方法。在实现过程中，首先针对 S 盒的布尔函数标准型(Algebraic Normal Form，ANF)给出了 bitslicing 下的优化处理方法；其次分别基于读取和存储操作进行相应的优化，降低了算法在 MixNibbles 步骤上的计算复杂度，从而使得 KLEIN 算法能通过 bitslicing 方式对计时和缓存边信道攻击方式进行防御。从 AVR 平台的实际试验结果来看，优化后的 KLEIN 算法的 bitslicing 实现在 AVR 微处理器平台下具备可实用性。

2 KLEIN 算法简介

在 RFIDSec 2011 会议上，Gong 等人提出了一种新的面向软件实现的轻量级分组密码算法 KLEIN(在荷兰语中表示“mini”的意思)[1]。KLEIN 算法基于替换-置换结构(Substitution-Permutation Network)加以设计，分组长度固定为 64 比特。算法密钥长度可选择 64、80 和 96 比特，但需要不同迭代轮数(分别为 12、16 和 20 轮)，对应名称为 KLEIN-64/80/96。为了达到受限环境下软件实现的高效性，KLEIN 算法尽可能地采用了面向字节的处理模式。在算法的非线性模块上，KLEIN 采用了具有自反性质的 4 比特 S 盒，使得算法仅需要付出一个 S 盒的代价就能实现加解密运算。在扩散模块上，KLEIN 将 AES 的 MixColumns 函数变形为算法中的 MixNibbles 函数，同时与面向字节的循环左移函数 RotateNibbles 相结合。这种设计思路既保证了 KLEIN 算法的软硬件效率，又继承了 MixColumns 函数最大距离码(MDS)特性。在密钥调度模块上，KLEIN 选择了比较复杂的处理方法，从而保证基于 KLEIN 的哈希函数也具有较好的安全性[17]。KLEIN 算法描述如图 1 所示。

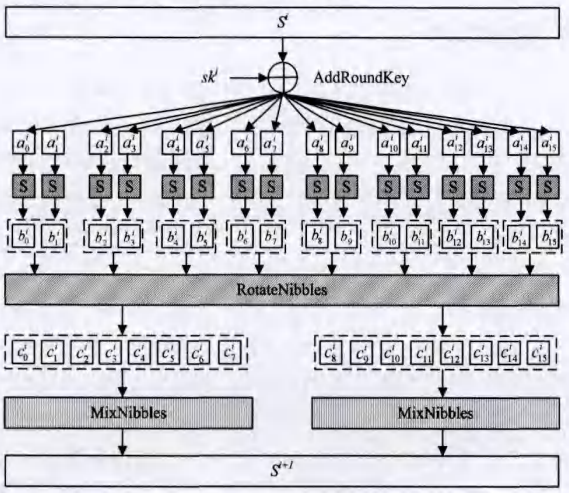

图 1 KLEIN 算法描述

通过典型传感器硬件平台 TelosB 和 IRIS 上大量的实验数据分析，表明 KLEIN 算法在软件实现上相比于现有轻量级分组密码算法具有一定的优势[1]。在硬件开销上，Synopsis 软件综合输出结果表明 3 种密钥长度的 KLEIN 算法的硬件实现开销都低于 1530 个门电路[1]。上述结果表明，KLEIN 算法在软硬件的实现上均适用于物联网软硬件环境。

3 基于 AVR 汇编的 KLEIN 分组密码算法 bitslicing 实现

3.1 4 比特 S 盒 bitslicing 方法

在分组密码算法中，S 盒往往作为唯一的非线性构件，实现算法在迭代若干轮数之后的抗密码学分析能力。在软件或硬件实现上，S 盒运算往往采用查表法(Look-Up Table，LUT)的形式，以加快加解密运算的速度。但在边信道攻击中，不同的 S 盒输入会对查表时间造成影响(例如排在 LUT 表前面的查询速度快一点，排在后面的速度相对会慢一点)。攻击者可以通过查表运算或缓存读取的时间差异所泄漏的边信道信息进行密钥恢复。因而在 S 盒的实现上，我们可以通过 bitslicing 的方式来避免查表所造成的时间或存储差异。常用的 S 盒 bitslicing 方法就是将其转换为布尔函数标准型(ANF)。由于 KLEIN 所用的 S 盒为 4 比特输入输出(见表 1)，因此其可以转换为 4 个布尔函数的形式。根据布尔函数标准型转换，我们可以得到 KLEIN 的 S 盒的 bitslicing 实现如下：

$$y_0 = 1 + x_0 + x_1 + x_0x_2 + x_1x_2 + x_0x_1x_2 + x_3 + x_1x_3 + x_0x_1x_3$$

$$y_1 = 1 + x_0 + x_2 + x_1x_2 + x_3 + x_1x_3 + x_0x_1x_3 + x_2x_3$$

$$y_2 = 1 + x_1 + x_2 + x_0x_2 + x_1x_2 + x_0x_1x_2 + x_0x_3 + x_0x_2x_3 + x_1x_2x_3$$

$$y_3 = x_1 + x_0x_2 + x_3 + x_0x_3 + x_0x_1x_3 + x_1x_2x_3$$

其中，(x_0, x_1, x_2, x_3)和(y_0, y_1, y_2, y_3)分别为输入值 X 和 Y 从低到高的 4 个比特(例如输入值为 4，那么 $x_0=0, x_1=0, x_2=1, x_3=0$，S 盒输出 $y_0=1, y_1=0, y_2=0, y_3=0$)。ANF 表达式中的加法与乘法运算均为二进制上的与(AND)运算和异或(XOR)运算。

表 1 KLEIN 4 比特 S 盒

输入	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
输出	7	4	A	9	1	F	B	0	C	3	2	6	8	E	D	5

如果采用上述方法来实现 S 盒，其处理速度将会比采用查表方法慢。在文献[1]中，Biham 等人提出了并行化处理的方法来提高 S 盒在 bitslicing 方式下的处理速度。其核心思想在于将 S 盒输入值当成与 S 盒输出长度相当的二维向量进行处理。举例说明，对于 KLEIN 的 S 盒而言，如果采用传统的布尔函数标准型方式进行处理，每次输入 4 个比特(x_0, x_1, x_2, x_3)，输出则为(y_0, y_1, y_2, y_3)。如果我们把(x_0, x_1, x_2, x_3)看作是 4 个 16 比特向量值而不是单独 1 个比特，那么由于 S 盒处理都是在 GF(2)上的运算，没有进位处理，因此输出值(y_0, y_1, y_2, y_3)也将变为向量值，这将降低 S 盒在 bitslicing 后的速度损失。

3.2 RotateNibbles 步骤 bitslicing 方法

在 KLEIN 算法中，RotateNibbles 步骤是将 S 盒输出值循环左移两个字节。由于该步骤在运算上与输入值无关，因

此在 bitslicing 上不需要进行特殊处理。所以在软件实现上，我们可以通过块移位和 S 盒置换操作一并处理的方式(例如，输入为 a,b,c,d 4 个 4 比特块，本来 S 盒输出为 $S[a]\|S[b]\|S[c]\|S[d]$，然后再进行 RotateNibbles 步骤，现改为直接输出 $S[c]\|S[d]\|S[a]\|S[b]$)，将 RotateNibbles 步骤和 S 盒操作步骤合并，从而在不改变算法及其安全性的情况下简化了 RotateNibbles 步骤的开销。由于在汇编代码实现中单独的块移位操作需要寄存器移位操作进行处理，因此该步骤虽然在技术上并无创新，但与标准实现相比，在实际中能略微降低 RotateNibbles 和 S 盒运算的整体时间开销。

3.3 MixNibbles 步骤 bitslicing 方法

在 KLEIN 算法中，MixNIbbles 步骤与 AES 的 MixColumns步骤在运算上是一致的，不同之处在于 AES 面向 128/192/256 比特进行处理，而 KLEIN 算法只需处理 64/80/96 比特。AES 的快速实现采用了 256 比特查表的方式，将 MixColumns 当中所用到的 GF(28)上的乘法运算均转换为若干次查表运算与异或运算相结合的结果[12]。在 MixNibbles 的 bitslicing 实现上，我们基于该矩阵运算的特点，给出了每一个字节运算的布尔函数表达式，如下：

$bij|0=a(i)4j|7\oplus a(i+1)4j|0\oplus a(i+1)4j|7\oplus a(i+2)4j|0\oplus a(i+3)4j|0$

$bij|1=a(i)4j|0\oplus a(i)4j|7\oplus a(i+1)4j|0\oplus a(i+1)4j|1\oplus a(i+1)4j|7\oplus a(i+2)4j|1\oplus a(i+3)4j|1$

$bij|2=a(i)4j|1\oplus a(i+1)4j|1\oplus a(i+1)4j|2\oplus a(i+2)4j|2\oplus a(i+3)4j|2$

$bij|3=a(i)4j|2\oplus a(i)4j|7\oplus a(i+1)4j|2\oplus a(i+1)4j|3\oplus a(i+1)4j|7\oplus a(i+2)4j|3\oplus a(i+3)4j|3$

$bij|4=a(i)4j|3\oplus a(i)4j|7\oplus a(i+1)4j|3\oplus a(i+1)4j|4\oplus a(i+1)4j|7\oplus a(i+2)4j|4\oplus a(i+3)4j|4$

$bij|5=a(i)4j|4\oplus a(i+1)4j|4\oplus a(i+1)4j|5\oplus a(i+2)4j|5\oplus a(i+3)4j|5$

$bij|6=a(i)4j|5\oplus a(i+1)4j|5\oplus a(i+1)4j|6\oplus a(i+2)4j|6\oplus a(i+3)4j|6$

$bij|7=a(i)4j|6\oplus a(i+1)4j|6\oplus a(i+1)4j|7\oplus a(i+2)4j|7\oplus a(i+3)4j|7$

其中，$aij|0$ 代表 MixNibbles 步骤中第 i 行、第 j 列所在字节的第 0 个比特。

3.4 基于读取特性的操作优化

在 AVR 微处理器环境下，要读取一个数组中的元素，计算其下标通常是复杂的，尤其在基于 3.3 小节方式转换后的 MixNibbles 步骤，下标形如$((i+1)\%4)*8+j*32$，在 $i+1$ 和 j 已知的情况下，仍然需要 2 次 lsr、10 次 lsr 和 1 次 add 操作。原来采用的方法是每读取一个数据只更新一个值，即：

```
CALCULATE i%4+j
READ a(i)4j|7
CALCULATE (i+1)%4+j
READ a(i+1)4j|0
READ a(i+1)4j|7
b_ij|0← a(i)4j|7 XOR a(i+1)4j|0 XOR a(i+1)4j|7
CALCULATE i%4+j
READ a(i)4j|0
READ a(i)4j|7
CALCULATE (i+1)%4+j
READ a(i+1)4j|0
bij|1←a(i)4j|0 XOR a(i)4j|7 XOR a(i+1)4j|0
```

这样会增加下标计算和取数的操作。优化后采取的方法是读取一个数据更新所有与此数据相关的值，即：

```
CALCULATE i%4+j
READ a(i)4j|7
bij|0←a(i)4j|7
bij|1←bij|1XOR a(i)4j|7
bij|3←bij|3 XOR a(i)4j|7
bij|4←bij|4 XOR a(i)4j|7
```

此方法以增加空间的开销为代价，需要保存 $bij|0$ 到 $bij|7$ 的值。与优化前的实现相比，需要额外 8 个寄存器进行存储，但该值是保存在寄存器中的，实验数据表明这并不会增加实际运行时的内存开销，且根据 MixNibbles 步骤的特点，可线性读取数据 a 的值。在 AVR 微处理器环境下，线性取值只需指针加 1：LD Rd,X+。

3.5 基于存储特性的操作优化

在 AVR 微处理器环境下，变量存放在 RAM 中，地址是 16 位的，寄存器是 8 位的，因此需使用两个寄存器分别保存高位地址和低位地址。而对数组变量进行读写操作，就必须修改高位地址寄存器和低位地址寄存器。采用原来的方法更新 bt_state[20]的值：

```
LowAddress←LowAddress+offset(0x14)
HighAddress←HighAddress+carry(0 or 1)
bt_state[HighAddress|LowAddress] ← 0x01
```

利用存储特性来调整数据位置，使得所有数组变量都不跨越低 8 位的地址空间(carry 为 0)，从而省去了所有对高位地址寄存器的操作。采取优化后的方法更新 bt_state[20]的值：

```
LowAddress←LowAddress+offset(0x14)
bt_state[HighAddress|LowAddress] ← 0x01
```

4 实验结果

为了测试 bitslicing 优化实现后的 KLEIN 算法的实用性，我们采用 ATMEL ATtiny45 系列微处理器作为实验平台，在平台上使用 AVR 汇编语言(编译环境 AVR Studio 4.12)来实现 KLEIN-80 加解密算法。ATtiny45 系列微处理器具有 4k 字节可编程 Flash ROM，256 字节 EEPROM，256 字节 SRAM，工作模式下主频可自适应调整，最大可为 20MHz。

KLEIN 算法在 AVR 微处理器上的实现开销比较如表 2 所列。

为了体现 3.4 节和 3.5 节中所提出的针对 AVR 汇编的优化方法的效果，将优化前后的 bitslicing KLEIN 算法也做了比较。各项性能数据均由 AVR Studio 4.12 测试给出，其中代码大小是加解密算法所占 Flash ROM 的字节数，内存开

销为所占 SRAM 的字节数，处理速度则是算法加、解密一个分组所需要的微处理器时钟数。

表 2　KLEIN 算法在 AVR 微处理器上的实现开销比较

实现算法	代码大小	RAM	ROM	完整轮数时钟周期
基本 KLEIN-64	1346	32	512	5562
基本 KLEIN-80	1378	34	512	7621
基本 KLEIN-96	1426	36	512	9799
优化前				
bitslicing KLEIN-64	2928	168	20	141059
bitslicing KLEIN-80	2954	170	20	188005
bitslicing KLEIN-96	2984	172	20	238141
优化后				
bitslicing KLEIN-64	2012	168	20	69035
bitslicing KLEIN-80	2038	170	20	91973
bitslicing KLEIN-96	2054	172	20	115339
Sbox 处理优化后				
bitslicing KLEIN-64	2780	160	20	62423
bitslicing KLEIN-80	2812	162	20	83157
bitslicing KLEIN-96	2896	164	20	104319

KLEIN 算法在 AVR 微处理器上的加解密速度比较如表 3 所列。

表 3　KLEIN 算法在 AVR 微处理器上的加解密速度比较

处理器主频	1MHz		8MHz		16MHz	
单次算法处理速度（单位：ms）	加密	解密	加密	解密	加密	解密
基本 KLEIN-64	5.562	7.923	0.695	0.99	0.348	0.495
基本 KLEIN-80	7.621	10.887	0.953	1.361	0.477	0.68
基本 KLEIN-96	9.799	14.159	1.225	1.77	0.612	0.885
优化前						
bitslicing KLEIN-64	141.059	43.713	17.632	5.484	8.816	2.732
bitslicing KLEIN-80	188.005	58.704	23.501	7.338	11.75	3.669
bitslicing KLEIN-96	238.141	73.61	29.768	9.201	14.884	4.601
优化后						
bitslicing KLEIN-64	69.035	33.489	8.629	4.186	4.315	2.093
bitslicing KLEIN-80	91.973	45.072	11.497	5.634	5.748	2.817
bitslicing KLEIN-96	115.339	56.57	14.417	7.071	7.209	3.536
Sbox 处理优化后						
bitslicing KLEIN-64	62.423	33.496	7.803	4.817	3.901	2.094
bitslicing KLEIN-80	83.157	45.079	10.395	5.635	5.197	2.817
bitslicing KLEIN-96	104.319	56.577	13.04	7.702	6.52	3.536

从表 2 与表 3 给出的性能数据可以看出，采用 bitslicing 方法来实现 KLEIN 算法大大增加了加解密所需的时间开销，但在实现的内存和代码大小开销上符合 AVR 硬件环境限制的要求。造成该测试结果的关键在于基于比特而不是字节来进行加解密运算操作，对速度的影响十分明显。尽管在速度上与面向字节的实现相比没有优势，但 bitslicing 实现能将每一次运算都规约到结果的一个比特。根据 Biham[8] 和 Konighofer[6] 的理论分析，bitslicing 算法后的 KLEIN 算法的加解密时钟周期将是固定值，对于不同的输入值的处理时间几乎相等，从而边信道攻击者无法从计算或存储的时间差异性上获得优势（由于 AVR 汇编一行代码即为一个时钟周期，基于 AVR 汇编的 KLEIN 分组密码 bitslicing 伪代码实现能从理论上证实该结论）。而未采用 bitslicing 保护技术实现的代码处理时间在输入值不同的情况下将会产生差异性。上述分析结果也体现在我们对加解密算法时钟周期的测试结果上。同时根据表 3 中具体工作频率下的加解密速度来看，bitslicing 后的 KLEIN 算法仍然在毫秒级别，可以满足 AVR 设备上的加解密要求。由于分组密码算法仅仅具有理论上的安全性，往往会造成实际中的安全性问题，因此上述 bitslicing 实现在需要抵抗计时和缓存攻击的边信道应用场景下具有很好的实用性。基于 AVR 汇编 KLEIN 分组密码 bitslicing 伪代码如下。

```
Encrypt:
Begin
round_key=key
    state=plain
    for i in [1..rounds] do
        call add_RoundKey
        call keySchedule
        state=sbox(state)
        call RotateNibbles
        call MixNibbles
    end for
    cipher=state^round_key
End
Decrypt:
Begin
    round_key=key
    for i in [1..rounds] do
        call keySchedule
    end for
    state=cipher ^ round_key
    for i in [0..rounds-1] do
        temp_state=state
        根据 temp_state 和 xtime 函数更新 u 和 v
        根据 u,v 和 temp_state 更新 temp_state
        根据 temp_state 更新 u 和 v
        v=xtime(v)
        根据 u,v 和 temp_state 更新 state
        state=sbox(state)
        逆转 round key
        state=state ^ round_key
    end for
    plain=state
End
add_RoundKey:
Begin
    state=state ^ round_key
End
keySchedule:
Begin
    更新 round_key 值
End
sbox:
INPUT state
OUTPUT tmp1
Begin
    根据 tmp1 初始化 X
    Y0=Y1=Y2=Y4=Y5=Y6=0x01
    READ X0
    Y0=Y0^X0
    Y1=Y1^X0
    READ X1
```

```
    Y0=Y0^X1
    Y2=Y2^X1
    Y3=Y3^X1
    同理读取 X2 到 X7 的值
    更新 Y0 到 Y7 的值
  根据 Y 更新 tmp1
End
RotateNibbles:
Begin
    state 循环左移两个字节
End
MixNibbles:
INPUT state
OUTPUT state
Begin
    把 8 字节 state 扩展为 64 字节 bt_state
    for j in [0..1] do
        for i in [0..3] do
            READ bt_state[(i*8+j * 32)+7]
            temp[i *8+j*32]=bt_state[(i*8+j*32)+7]
            temp[(i*8+j*32)+1]=bt_state[(i*8+j*32)+7]
            temp[(i*8+j*32)+3]=bt_state[(i*8+j*32)+7]

            temp[(i*8+j*32)+4]=bt_state[(i*8+j*32)+7]
        READ bt_state[((i+1)%4) * 8+j*32]
        temp[i * 8+j * 32] =temp[i * 8+j * 32]^ bt_state
[((i+1)%4) * 8+j*32]
        temp[(i * 8+j * 32)+1]=temp[(i * 8+j * 32)+1] ^
bt_state[((i+1)%4) * 8+j*32]
      同理读取 bt_state 各值,更新 temp 各值
    end for
    end for
    把 64 字节 temp 压缩为 8 字节 state
End
xtime:
Begin
    if((input & 0x80) == 0)
        output=(input << 1) ^ 0x00;
    else
        output=(input << 1) ^ 0x1B;
End
```

结束语 随着密码学在实际中的广泛应用,在现实中针对密码学算法特别是分组加密算法的攻击方法已由传统的密码学分析转为各种边信道分析方法。本文基于 bitslicing 的思想,将轻量级分组密码算法 KLEIN 的面向字节实现转换为能抵抗计时和缓存边信道攻击的 bitslicing 实现。针对 AVR 汇编的特性,我们给出了基于读取与存储特性的优化方法。优化后的 KLEIN 算法的 bitslicing 实现的加解密速度在 AVR 平台上可以满足实际应用的需要。在未来的工作中,我们将基于上述 bitslicing 实现,进一步研究抵抗其他类型边信道攻击的 KLEIN 算法实现及其优化方法。

参 考 文 献

[1] Gong Z, Nikova S, Law Y. KLEIN: A New Family of Lightweight Block Ciphers[C]// Proceeding of RFID Security and Privacy 2011. Berlin Heidlberg: Springer, 2011: 1-18

[2] Bogdanov A, Knudsen L R, Leander G, et al. PRESENT: An Ultra-Lightweight Block Cipher[C]// Proceeding of CHES 2007. Berlin Heidlberg: Springer, 2007: 450-466

[3] Nakahara, Jr J. Fast Variants of the MESH Block Ciphers [C]// Proceeding of Indocrypt 2004. Berlin Heidlberg: Springer, 2004: 162-174

[4] Nakahara, Jr J, Rijmen V, et al. The MESH Block Ciphers [C]// Proceeding of the International Workshop on Info. Security Applications, WISA 2003. Berlin Heidlberg: Springer, 2003: 458-473

[5] Poschmann A. Lightweight Cryptography- Cryptographic Engineering for a Pervasive World[D]. Germany: Ruhr-University Bochum. February 2009

[6] Konighofer R. A Fast and Cache-Timing Resistant Implementation of the AES[C]// Proceeding of CT-RSA 2008. Berlin Heidlberg: Springer, 2008: 187-202

[7] Moradi A, Poschmann A, Ling S, et al. Pushing the limits: A Very Compact and A Threshold Implementation of AES[C]// Proceeding of Eurocrypt 2011. Berlin Heidlberg: Springer, 2011: 69-88

[8] Biham E. A fast new DES implementation in software[C]// Proceeding of FSE 1997. Berlin Heidlberg: Springer, 1997: 260-272

[9] Neve M, Seifert J P. Advances on access-driven cache attacks on AES[C]// Proceeding of SAC 2006. Berlin Heidlberg: Springer, 2007: 147-162

[10] Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: The case of AES[C]// Proceeding of CT-RSA 2006. Berlin Heidlberg: Sprin-ger, 2006: 1-20

[11] Bonneau J, Mironov I. Cache-collision timing attacks against AES[C]// Proceeding of CHES 2006. Berlin Heidlberg: Springer, 2006: 201-215

[12] Aciic O, Schindler W, Koc C. Cache based remote timing attack on the AES[C]// Proceeding of CT-RSA 2007. Berlin Heidlberg: Springer, 2006: 271-286

[13] 赵新杰, 王韬, 矫文成, 等. 一种新的针对AES的访问驱动Cache攻击[J]. 小型微型计算机系统, 2009, 30(4): 797-800

[14] 赵新杰, 王韬, 郑媛媛. Camellia 访问驱动 Cache 计时攻击研究[J]. 计算机学报, 2010, 33(7): 1153-1165

[15] 赵新杰, 王韬, 郭世泽, 等. AES访问驱动Cache计时攻击[J]. 软件学报, 2011(3): 572-591

[16] 赵新杰, 王韬, 郭世泽, 等. 分组密码 Cache 攻击技术研究[J]. 计算机研究与发展, 2012, 49(3): 453-468

[17] 温雅敏, 龚征, 胡沐创, 等. 面向 ATtiny 微处理器的 KLEIN 分组密码算法实现[J]. 小型微型计算机系统, 2013, 34(7): 1641-1644