采用指令集扩展和随机调度的 AES 算法实现技术

孙迎红[1],童元满[2],王志英[2]
SUN Ying-hong[1],TONG Yuan-man[2],WANG Zhi-ying[2]

1.湖南涉外经济学院 计算机科学与技术系,长沙 410205
2.国防科学技术大学 计算机学院,长沙 410073
1.Department of Computer Science and Technology,Hunan International Economics University,Changsha 410205,China
2.School of Computer Science,National University of Defense Technology,Changsha 410073,China
E-mail:sun_yinghong@yahoo.com.cn

SUN Ying-hong,TONG Yuan-man,WANG Zhi-ying.AES implementation based on instruction extension and randomized scheduling.Computer Engineering and Applications,2009,45(16):106-110.

Abstract: Based on the random masking scheme,several fine grained masked primitives are defined.Then all the transformations in AES are decomposed to these primitives.And all the intermediate results are masked by different random values.To implement AES based on randomly masked primitives efficiently,three kinds of extended instructions are defined.Combined with random scheduling scheme,the whole execution flow of AES is presented.It is pointed out that this approach can prevent against first order and high order power analysis attack.Experiment results show that it has the advantage of security,performance and hardware complexity in comparison with several other countermeasures.

Key words: power analysis attack;Advanced Encryption Standard(AES);random mask;instruction extension

摘 要: 在随机掩码技术基础上,定义了若干细粒度的随机掩码操作,将 AES(Advanced Encryption Standard)算法中各种变换分解为细粒度随机掩码操作的序列,并使得所有的中间结果均被不同的随机量所掩码。为高效实现基于细粒度随机掩码操作分解的 AES 算法,定义了三种扩展指令,结合指令随机调度方法,给出了 AES 算法的完整实现流程,并指出这种实现技术可以抗一阶和高阶功耗攻击。实验结果表明,与其他典型防护技术相比,这种实现技术具有安全性、运算性能以及硬件复杂度等方面的综合优势。

关键词: 功耗攻击;高级加密标准;随机掩码;指令集扩展
DOI: 10.3778/j.issn.1002-8331.2009.16.031 **文章编号:** 1002-8331(2009)16-0106-05 **文献标识码:** A **中图分类号:** TP918

1 引言

密码算法的安全性包括两个方面的内容:一是密码算法关于密码分析技术的安全性即数学上的安全性,对当前广泛应用的密码算法一般都不易通过密码分析手段破解;二是关于密码算法实现方法的安全性,利用密码算法实现中的薄弱环节可实施旁路攻击以破解密钥,包括时间分析攻击、功耗攻击以及电磁辐射分析攻击等[1-2]。在各种旁路攻击技术手段中,相对而言功耗攻击的威胁最高[1]。

AES 算法是当前应用非常广泛的分组密码算法,因此安全芯片中通常包括 AES 算法部件。为有效抗功耗攻击,AES 算法部件的设计与实现必须采取有效的防护技术。由于随机掩码技术不仅适合软件实现,也适合于硬件实现,基于随机掩码的 AES 算法实现技术得到了深入而广泛的研究,国内外研究人员从各种不同角度提出了 AES 算法掩码实现技术[3-9],但是某些实现技术被证明是不安全的[10],基于掩码的 AES 算法实现技术仍然是一个值得研究的内容。

在随机掩码技术的基础上,本文定义了若干个随机掩码细粒度操作,并据此定义了三种扩展指令,结合指令随机动态调度方法,给出了 AES 算法的完整实现方法,分析这种实现技术的安全性。实验结果表明,与其他典型防护技术相比,这种实现技术具有安全性、运算性能以及硬件复杂度的综合优势。

2 基于随机掩码的细粒度操作

AES 算法的明文和密文分组为 128 字节,其输入、输出以及中间结果称为状态(State),以 4×4 的字节矩阵表示。在每一轮迭代中,AES 算法对 State 施加如下四种变换:SubBytes、ShiftRows、MixColumns 以及 AddRoundKey。SubBytes 是 AES 算法中唯一的非线性变换,通常也称为 S 盒。对字节 x,其 S 盒变换 $S(x)$定义为:

$$S(x)=L\cdot Inv(x)\oplus c, x\in \mathrm{GF}(2^8) \quad (1)$$

基金项目: 国家自然科学基金(the National Natural Science Foundation of China under Grant No.60706026)。
作者简介: 孙迎红(1980-),女,讲师,主要研究领域为计算机系统结构,信息安全;童元满(1982-),男,博士,主要研究领域为高性能计算,微处理器设计,信息安全;王志英(1956-),男,博士,教授,博士生导师,主要研究领域为高性能微处理器设计,信息安全。
收稿日期: 2008-09-23 **修回日期:** 2008-12-22

$$Inv(x)=\begin{cases}0, x=0\\ x^{-1}, \text{otherwise}\end{cases} \quad (2)$$

式(2)中 L 和 c 为 AES 算法中仿射变换的参数。当 x 不为 0 时,$Inv(x)$为 x 的逆。

本文将 AES 算法中所有的运算分解成小粒度的原子操作。为有效抗功耗攻击,令各中间结果被不同的随机量所掩码,基于随机掩码的细粒度操作如下所示。下面设 $x_1=u_1\oplus r_1, x_2=u_2\oplus r_2, u_1, r_1, u_2, r_2\in GF(2^8)$,其中 r_1 和 r_2 为随机量,x_1 和 x_2 为被掩码的中间结果。

(1)加法即按位异或(add)

掩码加法操作的运算过程为:

$t_1=(r_1\oplus r_2)\oplus r'$　$t_2=(x_1\oplus x_2)\oplus t_1=(u_1\oplus u_2)\oplus r'$

上式中 r'为新的随机量,这样预期结果$(u_1\oplus u_2)$被新的随机量 r'掩码。

(2)仿射变换(aff)

仿射变换是 S 盒中的线性部分,掩码仿射变换的运算过程为:

$t_1=L\cdot r_1\oplus r'$　$t_2=L\cdot x_1\oplus c$　$t_3=t_1\oplus t_2$

同样的,预期结果$(L\cdot u_1\oplus c)$被新的随机量 r'掩码。

(3)有限域 GF(2^8)上的平方操作(sqr)

本文以 x^{254} 即模幂的方式实现 $Inv(x)$,可将 0 和非 0 元素统一起来。模幂运算被拆分成模平方和模乘的操作序列,掩码模平方操作的运算过程为:

$t_1=r_1^2\oplus r'$　$t_2=x_1^2\oplus t_1=u_1^2\oplus r'$

同样的,预期结果(u_1^2)被新的随机量 r'掩码。

(4)有限域 GF(2^8)上的乘法操作(mul)

掩码乘法操作的运算过程为:

$t_1=r_1r_2\oplus r_3$　$t_2=x_1r_2\oplus t_1\oplus(r_3\oplus r_4)$

$t_3=x_2r_1\oplus t_2\oplus(r_4\oplus r_5)$　$t_4=x_1x_2\oplus t_3\oplus(r_5\oplus r')=u_1u_2+r'$

上式中,r_3、r_4、r_5 和 r'均为新的随机量,这样预期结果(u_1u_2)被 r'掩码。

(5)与常量的乘法操作(mulc)

当 x_1 与常量 a 相乘时,其运算过程为:

$t_1=ar_1\oplus r'$　$t_2=ax_1\oplus t_1=au_1\oplus r'$

同样的,预期结果(au_1)被新的随机量 r'掩码。

基于上述随机掩码的细粒度操作,AES 算法的一轮迭代的基本实现过程如图 1 所示,其中 ShiftRows 未被显式地说明其具体实现。同样的,AES 算法中密钥扩展部分也可转换为各掩码的细粒度操作的序列,具体实现在此不再赘述。

```
//SubBytes
for each byte of the State
        sqr,mul,sqr,mul,sqr,mul,sqr,
        mul,sqr,mul,sqr,mul,sqr    //Inv(x)
        aff      //仿射变换
end for
//MixColumns
for each byte of the State
        mulc,mulc,add,add,add
end for
//AddRoundKey
for each byte of the State
        add
end for
```

图 1　AES 算法中一轮迭代的基本实现过程

3　扩展指令

在上述基于随机掩码的细粒度操作中,加法即按位异或可以直接映射为微处理器中的异或指令,但仿射变换和有限域 GF(2^8)上的乘法并不与微处理器中指令相对应,而只能映射为某些指令的集合。为提高运算性能,本文对微处理器的指令集进行扩展,定义如下三条扩展指令。

(1)三操作数累加(xadd)

累加指令 xadd 为三操作数指令,其执行的操作为:rd←rs1⊕rs2⊕rs3,其中 rs1、rs2 和 rs3 为 3 个源操作数地址,rd 为目的操作数的地址,操作数的地址可能为寄存器或者内存地址。

(2)仿射变换(xaff)

仿射变换指令 xaff 为双操作数指令,其执行的操作为:rd←$L\cdot$rs1⊕rs2,其中 L 为 AES 算法中仿射变换的固定参数,rs1 和 rs2 为两个源操作数地址,rd 为目的操作数的地址,操作数的地址可能为寄存器或者内存地址。

(3)乘累加(xmac)

有限域 GF(2^8)上乘累加指令 xmac 为三操作数指令,其执行的操作为:rd←rs1⊗rs2⊕rs3,其中 rs1、rs2 和 rs3 为 3 个源操作数地址,rd 为目的操作数的地址,操作数的地址可能为寄存器或者内存地址。

基于扩展指令 xadd、xaff 和 xmac,可有效实现上述掩码的细粒度操作,比如掩码加法相当于 2 个累加指令 xadd,掩码仿射变换相当于 2 个 xaff 指令,掩码乘法操作相当于 4 个 xmac 和 3 个累加指令 xadd。在以 xmac 实现掩码平方操作时,rs1 和 rs2 相同;在常量乘法操作中,rs1 或 rs2 为一常量。扩展指令 xadd、xaff 和 xmac 对应的具体操作非常简单,硬件复杂度低,具体电路结构不再赘述;不涉及进位传递过程,运算延时小,不会成为提高时钟频率的瓶颈。

在扩展上述两条指令时,需结合原有的微处理器指令集体系结构仔细设计。当微处理器具有复杂指令集结构(CISC,Complex Instruction Set Computer)时,比如与 8051 兼容的 CPU,xadd、xaff 和 xmac 中源操作数和目的操作数的地址可以为存储器地址,也可以为寄存器,或为二者的结合;取舍的原则为使执行一个操作所需的时钟周期数和指令所占的存储空间最小。对于指令 xaff 而言,如果操作数地址可为存储器地址,则执行一个仿射变换所需的指令序列为:

```
mov  A,I1;   //I1 为源操作数地址,A 为累加寄存器,读出源操作数
xaff  A,rs2;   //A←L·A⊕rs2,实际的仿射变换
mov  D3,A;   //D3 为目的操作数地址,结果写入存储器
```

上述指令序列至少需 6 个字节存储空间,3 个时钟周期。

如果操作数地址为寄存器地址,则需包括如下指令序列:

```
mov  rs1,I1;   //I1 为源操作数地址,rs1 为寄存器,读出源操作数
mov  A,I2;    //I2 为源操作数地址,A 为累加器,读出源操作数
xaff  A,rs1;   //A←L·rs1⊕A,实际的仿射变换
mov  D,A;//D 为目的操作数地址,结果写入存储器
```

上述指令序列至少需 8 个字节存储空间,4 个时钟周期。

对指令 xmac 也可进行同样的分析,在此略去。

如果微处理器具有精简指令集结构(RISC,Reduced Instruction Set Computer)时,如 ARM、Sparc 以及 OpenRisc 等,

源操作数和目的操作数的地址均为寄存器;执行一个具体的仿射变换或乘累加操作之前，需从存储器中读出操作数至寄存器,最终将结果写入存储器。此时,仿射变换指令 xaff 即与普通的双操作数指令完全类似。对于扩展指令 xadd 和 xmac 而言,为保证其指令格式与执行过程和其他指令保持一致，可使 rd 和 rs3 为定制的特殊寄存器,类似于 8051 中累加寄存器,且 rd 和 rs3 相同(下面记为 A),这样 xadd 和 xmac 仅涉及两个通用寄存器读操作,无须修改微处理器的流水线。此时,执行一个乘累加操作所需的指令序列可能为:

```
load rs1,I1;  //I1 为源操作数地址,读出源操作数至寄存器 rs1
load rs2,I2;  //I2 为源操作数地址,读出源操作数至寄存器 rs2
load A,I3;    //I3 为源操作数地址,读出源操作数至特殊寄存器 A
xmac rs1,rs2; //A←rs1⊗rs2⊕A,特殊寄存器 A 无需显式指定
store D,A;    //D 为目的操作数地址,结果写入存储器
```

需要指出的是,基于随机掩码的细粒度操作将随机量作为操作数,假定随机量来源于安全 SoC 芯片内的真随机数发生器(TRNG,True Random Number Generator),真随机数发生器也会被分配一定的地址空间，当以随机量作为各指令的操作数时,并非从存储器中读取,而是直接从真随机数发生器读取,即读取被分配给真随机数发生器的地址。

4 指令随机调度方法

本文以软件方式实现 AES 算法的关键在于动态地生成随机指令序列,总体实现框架如图 2 所示,主要由三部分组成:基本块模板、指令窗口以及指令序列随机动态生成模块。

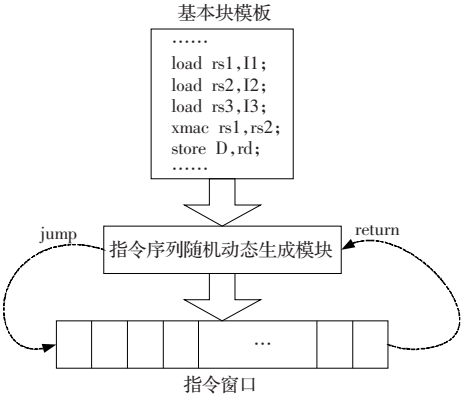

图 2 AES 算法软件实现的总体框架

4.1 基本块模板

如上所述,AES 算法中一轮迭代包括 SubBytes、ShiftRows、MixColumns 以及 AddRoundKey 等变换，各变换均可转换为基于随机掩码的细粒度操作序列，进而最终转换为 load/store 指令、按位异或 xor 以及扩展指令 xaff 和 xmac 的序列。

所谓基本块(basic block)就是若干条指令的组合,且不包含分支指令。基本块模块实际上就是实现某些具体操作的指令序列，主要包括 SubBytes 中对单个字节进行 S 盒变换、ShiftRows 中执行循环左移、MixColumns 中对任意一列进行处理以及 AddRoundKey 中对单个字节进行处理的指令序列。

举例来说,SubBytes 中对单个字节进行 S 盒变换所对应的细粒度操作序列为:

```
sqr,mul,sqr,mul,sqr,mul,sqr,mul,sqr,mul,sqr,mul,sqr,aff
```

在 RISC 体系结构下,掩码平方操作 sqr 所对应的基本块为:

```
load rs1,I1;     //读出源操作数 r1 至寄存器 rs1
load A,TRNG;     //从真随机数发生器读出一新的随机量至特殊寄存器 A
xmac rs1,rs1;    //A←rs1⊗rs1⊕A,相当于执行 t1=r1^2⊕r'
load rs2,I2;     //读出源操作数 x1 至寄存器 rs2
xmac rs2,rs2;    //A←rs2⊗rs2⊕A,相当于执行 t2=x1^2⊕t1=u1^2⊕r'
store D,A;       //结果写入存储器
```

MixColumns 中对任意一列进行处理所对应的细粒度操作序列为:

```
mulc,mulc,add,add,add   //计算第一行的结果
mulc,mulc,add,add,add   //计算第二行的结果
mulc,mulc,add,add,add   //计算第三行的结果
mulc,mulc,add,add,add   //计算第四行的结果
```

按照如图 1 所示的运算流程,通过复制基本块模板的指令序列,并重命名寄存器地址和存储器地址,可完整地实现 AES 算法。一个最小规模的基本块模块只需包括上述基于随机掩码的 5 个细粒度操作分别对应的指令序列即可。

4.2 指令窗口

所谓指令窗口就是用于暂存随机动态生成的指令序列的缓冲区。在 AES 算法运行过程中,指令序列随机动态生成模块动态产生一段可执行的代码并填充到指令窗口中,然后跳转到指令窗口对应的程序代码,执行完毕后返回到指令序列生成模块;重复上述动作,直到 AES 算法所有运算结束。

4.3 指令序列随机动态生成模块

指令序列随机动态生成模块是 AES 算法软件实现的关键部分,利用 AES 算法固有的并行性和一定的随机策略,动态地产生用于完成 AES 算法中所有运算的指令序列，使运算过程达到最大程度的随机化,并且充分挖掘内在并行性。

AES 算法中内在的并行性主要体现在以下几个方面:

SubBytes 中各字节的处理是完全可并行的,实际上就是各个字节 S 盒变换所对应的细粒度操作序列中同一位置的操作可并行执行。

MixColumns 中各列的处理是完全可并行的，且一列中各个字节的更新也是完全并行的;更新任意字节所需的细粒度操作也是部分可并行的,如两个常量乘法操作 mulc 可并行。

在基本块中,从存储器或真随机数发生器中读出相互之间不存在数据相关性的多个操作数至寄存器时,执行顺序是可调整的。

指令序列随机动态产生所必须遵循的原则有:

(1)不能破坏存在数据相关的指令的执行顺序;

(2)结合微处理器的流水线结构,尽可能地减少流水线气泡(或暂停),即使流水线满负荷工作;

(3)填充各细粒度操作对应的基本块到指令窗口时,必须保持其“原子性”,即保证基本块的指令序列在一次调度中就被全部写入指令窗口。

本文所采取的随机策略主要包括:

(1)各字节处理顺序随机化

如上述,AES 算法中各字节的处理是可完全并行的，因此本文随机确定各字节的处理顺序。在 8 位微处理器中,16 个字节的所有执行顺序总数为 16!,即 16 个字节的全排列;在 32 位微处理器中,基于 4 字节并行处理,所有执行顺序总数为 4!,即

24 种。为简化随机调度过程,可以按循环左移(右移)方式进行。设各字节编号为{0,1,2,3,…,15},且随机控制参数为 c ($0\leqslant c\leqslant 15$),则循环左移(右移)c 个位置,得到的结果即为各字节的执行顺序,所有可能的执行顺序为 16 种。比如当 c=2 时,则{2,3,4,…,15,0,1}即为循环左移后的执行顺序。同样的,按上述方式,在 32 位微处理器中,循环左移(右移)所能达到的执行顺序总数为 4。

根据本文的实现方式,各字节的处理是独立进行的,而并非按如图 1 所示的循环迭代方式逐个处理,这也相当于进行了完全的循环展开。

(2)寄存器随机重命名

基本块中各指令所使用的寄存器是可更改的,只需保证同一基本块中所有指令的寄存器统一的重命名以维持数据依赖关系即可。最简单的寄存器随机重命名策略为轮换方式,即将寄存器资源循环地分配给不同位置的基本块。

(3)存储地址随机化

在本文的实现方式中,除保存 AES 算法当前被掩码的状态及其随机掩码的存储空间外,还需要两组同样大小的存储空间以暂存中间运算结果,比如在 S 盒的运算过程中需要分别暂存 sqr 和 mul 的计算结果,在 MixColumns 的运算过程中需要分别暂存两个 mulc 的计算结果。设 AES 算法的当前被掩码的状态为 I',与之对应的随机掩码为 R($I'=I\oplus R$);另两组被掩码的中间结果及其掩码分别为 IA',RA,IB',RB($IA'=IA\oplus RA$,$IB'=IB\oplus RB$);I'、IA',以及 IB'和对应的随机掩码在存储器中存储位置相邻。在 AES 算法的运算过程中,可随机选择 I'、IA',以及 IB'的起始地址,最简单的随机策略即为在若干个(如 4 个)可能的地址中随机选择。

(4)指令执行顺序随机化

对于同一基本块中从存储器或 TRNG 中读取操作数的指令,以及不同基本块(处理不同的字节)中的指令,可随机地调整其顺序,以进一步随机化运算过程,并且减少流水线气泡。举例来说,在上文提到的掩码平方操作 sqr 所对应的基本块中,前两条 load 指令是可调换的;之后的 xmac 指令与这两条 load 指令存在数据相关,流水线中有可能出现一个气泡,此时可将后续的 load 指令提前,或者插入另一个字节相应的基本块中的 load 指令。后者实质上与软件流水机制是统一的,即从循环的不同迭代中取出相互之间不存在数据依赖的指令组成新的序列,以消除由单个迭代中因数据依赖引起的气泡。

为简化随机调度过程,可以仅调换同一基本块中的读操作数指令即 load 指令的执行顺序。

在上述各种随机策略中,最为关键的是各字节处理顺序随机化,其他三种随机化技术可以进一步地提高随机化程度,达到更好的防护效果;比如寄存器和存储地址随机重命名即可有效防止针对地址的 DPA 攻击。但为简化指令序列的随机动态生成,后三种随机策略可以省略以提高运算性能。

基于上述指令序列随机动态生成技术,即使对微处理器指令集进行了扩展,但并不需要编译器的参与即可完全实现 AES 算法。如果不要求扩展指令被其他程序使用,并不要求原有的工具链特别是编译器做出相应修改。

4.4　安全性分析

文献[10]给出了密码算法可被功耗攻击的形式化描述,其关键在于判断密码算法实现过程中的中间变量的概率分布是否与密钥相关。本文即根据文献[10]来分析所提出的 AES 算法实现的抗功耗攻击能力。

根据随机掩码的具体运算,可得出如下结论。

引理 1　设 u 为有限域 GF(2^8)中任意元素,c 为有限域 GF(2^8)中的一个非 0 常量,r 为服从集合{0,…,255}上均匀分布的随机量且与 u 相互独立,则变量 $u\oplus r$,$c\otimes(u\oplus r)$,$f_{(L,c)}(u\oplus r)$ 和 r^2 均服从均匀分布。

引理 2　设 u_1、u_2 为有限域 GF(2^8)中任意元素,r_1、r_2 为相互独立且服从集合{0,…,255}上均匀分布的随机量。令 $x_1=u_1\oplus r_1$,$x_2=u_2\oplus r_2$。变量 $x_1\otimes I_2$、$r_1\otimes x_2$、$x_1\otimes r_2$ 和 $r_1\otimes r_2$ 的概率分布均与密钥无关,且其分布与有限域 GF(2^8)中随机量 z 的概率分布相同,z 的概率分布为:

$$\Pr(z=i)=\begin{cases}511/2^{16}, i=0\\ 255/2^{16}, \text{otherwise}\end{cases} \tag{3}$$

引理 3　设 $x_1,x_2,\cdots,x_n$(n>0)为 n 个被掩码的中间结果,其中 $x_i=u_i\oplus r_i$,$0\leqslant i\leqslant n$,$r_1,\cdots,r_n$ 为相互独立且服从均匀分布的随机量。这 n 个被掩码的中间结果的联合概率分布与 $u_1,u_2,\cdots,u_n$ 无关。

上述三个引理的证明在此省略。根据上述引理,下面对 AES 算法实现中各中间结果是否可被功耗攻击进行分析。根据引理 1,掩码细粒度操作中各中间结果 t_1、t_2、t_3、t_4 均服从均匀分布。根据引理 2,在计算 t_1、t_2、t_3、t_4 时产生的更小范围内的所有中间结果中,比如掩码乘法操作 mul 中的 r_1r_2 和 x_1r_2 等,除 sqr 中的 x_1^2、mul 中的 r_1r_2、x_1r_2、x_2r_1 以及 x_1x_2 服从如式(3)所示的概率分布外,其他中间结果仍服从均匀分布。因此,AES 算法实现中任意单个中间变量的概率分布与密钥无关,即可以抗一阶功耗攻击。

由于 AES 算法实现中除细粒度操作内部的中间结果之外的所有中间结果均被不同的随机量掩码,根据引理 3,多个中间结果的联合概率分布与密钥无关。对于细粒度操作内部的中间结果而言,其联合概率分布同样与密钥无关。比如 add、aff、sqr 以及 mulc 中的 t_1 和 t_2 的联合概率分布与 u_1、u_2 无关。因此针对多个中间结果的高阶 DPA 攻击同样不可行。

5　实验结果

根据上文所述的 AES 算法实现技术,基于 8051 微处理器核,设计实现了三种扩展指令,并实现了 AES 完整运算流程(仅采用了第一种随机策略),实验结果如表 1 所示。需要指出的是,因为同一密钥可以用于多个分组的加解密,假定 AES 算法中的密钥扩展部分已预先处理,而并非与 AES 算法的迭代过程同时进行。

从表 1 中的数据可以看出,与其他的防护技术相比(其他实现的运算性能结果来源于文献[8-9]),本文的实现方式在运算性能方面具有一定优势,且所需的硬件开销较小,适合于在智能卡中实现。

指令窗口容量和基本块模板数目均会影响运算性能,其数目越大,指令随机调度的开销越小,运算性能越高。限于篇幅,在此不再详细给出指令窗口容量和基本块模板数目对运算性能影响的定量分析以及相关实验结果。表 2 中实验结果都是在两者均为 80 的情况下得到。

表 1 8051 智能卡中 AES 算法不同实现的比较

实现方式	时钟周期数	硬件开销	备注
文献[3]	293 500	/	存在零值攻击问题，也可能遭受二阶 DPA 攻击
文献[4]	168 000	3 050 Byte（ROM）	需要基于 ROM 的对数表和反对数表，存在零值攻击问题
文献[5]	208 000	3 400 Byte（ROM）	需要基于 ROM 的查找表，因使用相同的随机量对不同的中间结果掩码而可能遭受二阶 DPA 攻击
文献[8]	236 900+ 4 800×n	/	掩码与随机技术相结合，随机地插入无效伪操作，可能遭受二阶 DPA 攻击
文献[9]	256 000	3 100 Byte（ROM）	基于傅立叶变换的掩码技术，需要基于 ROM 的查找表，在布尔掩码和算术掩码之间转换时可能遭受二阶 DPA 攻击
本文	148 500	240 逻辑门	指令窗口容量和基本块模板数目均为 80

6 结束语

提出了基于随机掩码的 AES 算法抗高阶功耗攻击实现技术，定义了若干基于随机掩码的细粒度操作，将 AES 算法中不同的变换转换为细粒度操作的序列，并使得所有的中间结果均被不同的随机量所掩码；据此定义了三种扩展指令，结合指令随机调度方法，给出了 AES 算法的完整实现流程。根据所采用的随机掩码技术的特点，分析了 AES 算法具体实现的安全性，指出其可抗高阶功耗攻击。实验结果表明所提出的 AES 算法实现技术与同类实现技术相比具有综合优势。

参考文献：

[1] Mangard S.Securing implementations of block ciphers against side-channel attacks[D].Austria:Graz University of Technology,2004.

[2] Kocher P,Jaffe J,Jun B.Differential power analysis[C]//LNCS 1666:Advances in Cryptology(CRYPTO'99).Berlin Heidelberg:Springer-Verlag,1999:388-397.

[3] Akkar M,Giraud C.An implementation of DES and AES,secure against some attacks[C]//LNCS 2162:CHES 2001.Berlin Heidelberg:Springer-Verlag,2001:309-318.

[4] Trichina E,Korkishko L.Secure and efficient AES software implementation for smart cards[C]//LNCS 3325:WISA 2004.Berlin Heidelberg:Springer-Verlag,2004:425-439.

[5] Oswald E.A side-channel analysis resistant description of the AES S-Box[C]//LNCS 3557:FSE 2005.Berlin Heidelberg:Springer-Verlag,2005:413-423.

[6] BlÄomer J,Merchan J G,Krummel V.Provably secure masking of AES[C]//LNCS 3357:Selected Areas in Cryptography.Berlin Heidelberg:Springer-Verlag,2005:69-83.

[7] Oswald E,Schramm K.An efficient masking scheme for AES software implementations[C]//LNCS 3786:WISA 2005.Berlin Heidelberg:Springer-Verlag,2006:292-305.

[8] Herbst C,Oswald E,Mangard S.An AES smart card implementation resistant to power analysis attacks[C]//LNCS 3989:ACNS 2006,Berlin Heidelberg:Springer-Verlag,2006:239-252.

[9] Prouff E,Giraud C,Aumônier S.Provably secure S-Box implementation based on Fourier transform[C]//LNCS 4249:CHES 2006.Berlin Heidelberg:Springer-Verlag,2006:216-230.

[10] 童元满,王志英,戴葵,等.识别密码算法具体实现中潜在功耗攻击的理论分析方法[J].计算机辅助设计与图形学学报,2008,20(3):395-402.

（上接 105 页）

不可见。而本文算法利用了 HVS 特性，保证了有较好水印信号透明性的前提下，将水印信息嵌入到原始图像所有像素的最低 2 bit，获得了较高的峰值信噪比（$PSNR$>42 dB），所以水印信息的不可见效果非常好。

（2）水印信息是基于图像内容产生的，所以对于图像内容的任意改变都可判断成篡改，具有良好的篡改敏感性。

（3）针对水印系统的安全性而言，使用了调制加密和位置两个密钥增强了整个算法的安全性。水印信息是在基于密钥控制下生成的混沌序列调制加密生成的，所以整个系统的安全仅仅依赖于用户的密钥，算法可以完全公开。图像检测认证时不需要原始图像，只需要知道密钥就可以实现篡改检测和定位。另外，本算法可以抵抗伪认证攻击（即只改变图像内容而不改变水印信号的攻击，因为随着内容的改变，水印验证信号发生了变化）。

（4）水印信号是通过分块计算产生的，并且嵌入到空域中，所以算法的时间复杂度较好，算法计算过程简单，容易实现，在检测过程中不需要原始图像及原始水印的参与，并且对于篡改定位精度可高达 2×2 像素，同时具有较高的篡改检测概率，是一个安全且实用的图像认证算法。

参考文献：

[1] Zhu B B,Swanson M D,Tewfik A H.When seeing isn't believing[J].IEEE Signal Process Mag,2004,3:40-49.

[2] Wong P W,Memon N.Secret and public key image watermarking schemes for image authentication and ownership verification[J].IEEE Trans on Image Process,2001,10(10):1593-1601.

[3] Suthaharan S.Fragile image watermarking using a gradient image for improved localization and security[J].Pattern Recognit Lett,2004,25:1893-1903.

[4] 和红杰,张家树,田蕾.能区分图像或水印篡改的脆弱水印方案[J].电子学报,2005,33(9):1557-1561.

[5] 沃炎,韩国强,张波.一种新的基于特征的图像内容认证方法[J].计算机学报,2005,28(1):105-112.

[6] Wong P W.A public key watermark for image verification and authentication[C]//Proceedings of the IEEE International Conference on Image Processing,Chicago,USA,1998:455-459.

[7] Celik M U,Sharma G,Saber E.Hierarchical watermarking for secure image authentication with localization[J].IEEE Transactions on Image Processing,2002,11(6):585-595.

[8] Lin P,Hsieh C.A hierarchical watermarking method for image tamper detection and recovery[J].Pattern Recognition,2005,38(2):2519-2529.

[9] Ouda A H,El-sakka M R.Localization and security enhancement of block-based image authentication[C]//Proceedings of IEEE International Conference on Image Processing,vol 1,2005:673-676.

[10] 张宪海,杨永田.基于脆弱水印的图像认证算法研究[J].电子学报,2007,35(1):34-39.