软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(7):1683−1694 [doi: 10.3724/SP.J.1001.2013.04329]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

RSA 踪迹驱动指令 Cache 计时攻击研究*

陈财森[1,2], 王 韬[1], 郭世泽[3], 周 平[1]

[1](军械工程学院 信息工程系,河北 石家庄 050003)

[2](装甲兵工程学院,北京 100072)

[3](北方电子设备研究所,北京 100083)

通讯作者: 陈财森, E-mail: caisenchen@163.com

摘 要: 指令 Cache 攻击是基于获取算法执行路径的一种旁路攻击方式.首先,通过分析原有 RSA 指令 Cache 计时攻击存在可行性不高且能够获取的幂指数位不足等局限性,建立了新的基于监视整个指令 Cache 而不只是监视特定指令 Cache 的踪迹驱动计时攻击模型;然后,提出了一种改进的基于 SWE 算法窗口大小特征的幂指数分析算法;最后,在实际环境下,利用处理器的同步多线程能力确保间谍进程与密码进程能够同步运行.针对 OpenSSL v.0.9.8f 中的 RSA 算法执行指令 Cache 计时攻击实验,实验结果表明:新的攻击模型在实际攻击中具有更好的可操作性;改进的幂指数分析算法能够进一步缩小密钥搜索空间,提高了踪迹驱动指令 Cache 计时攻击的有效性.对于一个 512 位的幂指数,新的分析算法能够比原有分析算法多恢复出大约 50 个比特位.

关键词: 指令 Cache 计时攻击;旁路攻击;RSA 密码算法;踪迹驱动;同步多线程

中图法分类号: TP309 **文献标识码**: A

中文引用格式: 陈财森,王韬,郭世泽,周平.RSA 踪迹驱动指令 Cache 计时攻击研究.软件学报,2013,24(7):1683−1694. http://www.jos.org.cn/1000-9825/4329.htm

英文引用格式: Chen CS, Wang T, Guo SZ, Zhou P. Research on trace drive instruction cache timing attack on RSA. Ruan Jian Xue Bao/Journal of Software, 2013,24(7):1683−1694 (in Chinese). http://www.jos.org.cn/1000-9825/4329.htm

Research on Trace Drive Instruction Cache Timing Attack on RSA

CHEN Cai-Sen[1,2], WANG Tao[1], GUO Shi-Ze[3], ZHOU Ping[1]

[1](Department of Information Engineering, Ordnance Engineering College, Shijiazhuang 050003, China)

[2](The Academy of Armored Forces Engineering, Beijing 100072, China)

[3](The Institute of North Electronic Equipment, Beijing 100083, China)

Corresponding author: CHEN Cai-Sen, E-mail: caisenchen@163.com

Abstract: The I-cache timing attack which exploits the instruction path of a cipher is one type of side channel attack. First, by analyzing the complications in the previous I-cache timing attacks on RSA algorithm because of how hard it has been to put them into practice, and how the number of the inferred bits is insufficient, this paper builds a new trace driven I-cache timing attack model via spying on the whole I-cache, instead targeting the instruction cache to which the special function mapped. Next, an improved analysis algorithm of the exponent based on the characteristic of the side of window in sliding window exponentiation (SWE) algorithm is proposed. Finally, an I-cache timing attack is implemented on RSA of OpenSSL v.0.9.8f in a practical environment, using a simultaneous multithreading processor to insure that the spy process and the cipher process can run in parallel. Experimental results show that the proposed attack model has strong applicability in real environments; the improved analysis algorithm of the exponent can further reduce the search space of the bits of the key, and improve the effectively of the trace driven I-cache timing attack. For a 512-bit exponent, it can recover about 50 bits of exponent more than the previous.

Key words: I-cache timing attack; side channel attack; RSA cryptographic algorithm; trace-driven; simultaneous-multithreading

* 基金项目: 国家自然科学基金(60772082); 河北省自然科学基金(08M010)
收稿时间: 2011-05-31; 定稿时间: 2012-09-29

旁路攻击是针对密码算法实现的一种新的攻击方式,利用密码算法实现过程中所泄露的信息推算密钥信息[1].典型的旁路攻击包括计时攻击、微架构攻击、功耗分析攻击、电磁分析攻击和故障攻击等.其中,微架构攻击的概念是由 Aciicmez 于 2007 年首次提出来的[2,3],主要关注于通用处理器部件与软件密码实现算法的安全性.与典型的旁路攻击不同,微架构攻击主要是针对现代计算机系统的微架构元器件的行为进行分析的,利用这些行为与密钥的相关性推算出密钥的相关信息.依据关注的处理器部件不同可分为数据 Cache 攻击[4]、指令 Cache 攻击[2,3,5,6]和分支预测攻击[7,8].其中,指令 Cache 攻击主要是利用监测指令 Cache 访问状态的变化情况获取算法的执行流,主要作用于算法操作序列依赖于密钥的密码算法,如本文关注的 RSA 算法.2005 年, Percival[4] 论证了多线程间共享 Cache 存储器的访问模式为间谍线程监视密码进程提供了入口,导致滑动窗口算法实现的 RSA 算法可能遭受数据 Cache 攻击,首次提出 RSA 算法存在遭受数据 Cache 攻击的可能性;2007 年,Aciicmez 通过监测 RSA 算法中关键操作函数指令(文中简称为关键指令)映射的指令 Cache 访问状态的变化情况,提出了两种可能的指令 Cache 攻击算法[2,5].同时,利用分支预测单元预测成功与预测失效产生的时间差异与 RSA 执行流程的相关性,首次提出针对 RSA 算法的分支预测分析算法[7];2010 年,Aciicmez 等人公布了将随机过程的方法应用于 DSA 指令 Cache 计时攻击的实验结果[9].目前,在 RSA 指令 Cache 攻击研究方面,Aciicmez 等人的研究成果在公开的文献中是比较显著的,是本文的主要研究基础.

原有 RSA 算法的指令 Cache 计时攻击是在密码算法关键指令对应的逻辑地址已知的前提下,构造与被监测关键指令映射到相同逻辑地址的间谍进程,使间谍进程与密码进程同步执行并不断循环执行自己的指令,测量间谍进程执行访问 Cache 的时间,利用获取的时间差异信息和指令 Cache 访问特性推导出密码进程的执行流,最终获取密钥信息.但是,原有算法的假设前提过于严格,在实际执行过程中,由于需要通过反编译目标程序的手段以获得被监测的关键指令对应的逻辑地址,而目标程序一般难以获得,因此实际攻击时可操作性不强.另外,针对采用滑动窗口算法实现的 RSA 算法指令 Cache 攻击在推导幂指数时,依据算法的执行流能够获取的幂指数位数目非常有限,不像数据 Cache 攻击还能够利用预计算表索引值与窗口大小的相关性进一步缩小幂指数的搜索空间,因此,需要寻找一种新的分析算法以降低攻击的复杂度.

本文在 Aciicmez 提出的 RSA 指令 Cache 攻击的基础上,通过分析原有攻击算法在实际攻击中的局限性,提出了一种可操作性更强的攻击方案,并给出能够进一步缩小密钥搜索空间的分析算法,充分利用获得的密码算法执行流信息,主要贡献为:

(1) 建立了一种基于整个指令 Cache 的踪迹驱动计时攻击模型,通过精心编写的间谍进程监视整个 L1 指令 Cache(L1 表示 1 级缓存)的访问状态变化情况,而不仅仅是被监视的关键指令所映射的指令 Cache 状态的变化,利用指令 Cache 访问命中和失效的特性推导出算法的关键操作执行流.这样,攻击者无需知道关键指令所映射的逻辑地址,只要构造一个能够与密码进程同步执行、不断执行访问填充整个 L1 指令 Cache 并测量访问每个 Cache 组的访问时间的间谍进程,就可以执行指令 Cache 攻击;

(2) 在原有的基于操作序列与幂指数位相关性的幂指数位分析算法的基础上,提出了一种新的基于窗口大小特征的分析算法,进一步提高了 Cache 计时信息的利用率,能够获取更多的指数比特位信息,从而缩小了密钥的搜索空间;

(3) 针对采用滑动窗口算法执行模幂运算的 RSA 算法,在同步多线程处理器上执行实际的攻击实验.实验结果表明:新的攻击模型可操作性更强,提高了指令 Cache 攻击的可行性;新的幂指数分析算法比原有分析算法能够获取更多的幂指数位,进一步缩小了密钥的搜索空间.

本文首先对指令 Cache 攻击的相关工作进行介绍,分析现有攻击方法的特点.第 2 节给出一种新的基于整个指令 Cache 的踪迹驱动计时攻击模型.第 3 节给出新模型下指令 Cache 计时数据的采集方法,在数据分析阶段提出一种基于窗口大小特征的分析算法.在实验及分析部分给出实验结果,并与原有攻击算法进行比较.最后进行总结.

1 相关工作

1.1 微架构攻击

旁路分析定义为利用如执行时间和功耗等旁路信息分析推导出密钥信息的一种攻击方式,其攻击对象主要是智能卡.最近几年,人们已开始研究在普通 PC 机平台上的旁路分析,并提出一种新的攻击方式——微架构分析(microarchitectural analysis,简称 MA)[2].它是旁路攻击研究的一个新领域,研究普通处理器的微架构元器件对软件密码系统安全性功能的影响作用.尽管如 Cache 等处理器部件的安全性缺陷早就在文献[4,5]中被提及,但是明确而广泛的基于处理器功能性的安全性攻击,最近才刚刚形成体系并迅速引起关注.目前,主要关注的微架构元器件有 Cache 和分支预测单元(branch prediction).其中,Cache 攻击是通过观测由 Cache 命中和失效产生的执行时间或功耗差异获取密码系统执行时的 Cache 行为,通过分析密码执行时的 Cache 状态统计信息来获取这些内存访问方式,统计信息包括 Cache 命中和失效的数目、被密码进程修改过的 Cache 行数目等等,非授权的恶意进程是无法直接获取这些信息的,但可以通过观测旁路泄露信息来估算这些信息;可分为数据 Cache 攻击和指令 Cache 攻击:数据 Cache 攻击与指令 Cache 攻击有着本质的不同,前者是基于数据路径攻击的,利用密码进程的数据访问方式,例如基于 S 盒的分组密码算法其查表的内存访问方式与密钥值的依赖关系[10];指令 Cache 攻击与分支预测分析(BPA)类似,主要被用于泄露密码进程的执行流.基于密码进程执行过程中会在共享的 Cache 和分支目标缓存中留下持续的变化踪迹,攻击者通过执行一个与密码进程同步执行的间谍进程跟踪这些踪迹,监视其状态的变化情况,就能够推测出密码系统的执行流,再利用执行流与密钥的依赖关系推导出密钥信息.

在实现过程中,主要利用处理器的同步多线程(SMT)能力,使得间谍进程能够与目标进程同步执行,并共享同一处理器中的 Cache 资源,利用相互之间持续访问私有数据或执行私有指令而造成 Cache 访问冲突,使得间谍进程能够监视密码进程的运行情况,导致安全系统的秘密或密钥泄露.尽管微架构攻击在普通处理器上也能执行,文献[11]指出,利用操作系统调度机制能够控制两个进程的轮流执行时机而实现 Cache 攻击,但是同步处理器的 SMT 能力为攻击提供了一定的便利.

1.2 RSA算法的指令Cache计时攻击

Cache 是位于内存与中央处理器之间的一个小的缓存器,它使处理器能够快速而方便地访问最频繁使用的数据和指令.当处理器需要从主存读取数据或指令时,它首先检测这些数据或指令是否存在于 Cache 中,如果存在(发生 Cache 命中),处理器立即读取这些数据或指令,而不需要访问主存;否则,处理器必须从主存中读取数据或指令(Cache 失效),同时将数据或指令的副本存储在 Cache 中.每一次 Cache 失效都会产生对更高一级存储器的访问,从而导致额外的存取延迟时间.Cache 计时攻击主要利用从 Cache 中读取数据或指令到 CPU 寄存器要比从 RAM 中读取要快的特性进行密码分析,是利用 Cache 的访问时间差异来推算密钥位的一种攻击方式.

密码系统具有数据依赖的内存访问模式,使得攻击者能够获取 Cache 命中和失效的特征信息.即使对特定的明文、密钥执行相同的指令,在执行过程中也会由于 Cache 行为的差异导致消耗的时间和功耗有所不同.Cache 攻击尝试利用这些差异信息来缩小密钥的搜索空间.由于数据 Cache 攻击存在的威胁性,OpenSSL 密码库[12]从 v0.9.8e 版本开始采取改进的固定窗口算法用于抵御文献[4]中提出的攻击方式,同时采用特殊方式处理 RSA 算法结构用于抵御数据 Cache 攻击的威胁.此外,还采用盲化操作用以抵御纯计时攻击.尽管如此,密码分析学家发现,OpenSSL 在指令 Cache 方面仍然存在遭受微架构攻击的可能.指令 Cache 攻击利用运行一个与密码进程同步的间谍进程监测指令 Cache 发生的变化,从而获取密码进程的执行流.

Aciicmez 分析了 RSA 算法中平方乘算法与窗口算法两种实现方式的执行流,依据执行流与密钥信息的依赖关系,主要包括:平方乘算法中,当幂指数位为 1 时,会比为 0 的情况多执行一个乘法操作;滑动窗口算法中,在每个窗口位滑动结束时,会执行一个乘法操作;以及固定窗口算法中的额外约减操作概率与预计算表中元素值的相关性,利用指令 Cache 的访问特性,提出了一种 RSA 算法指令 Cache 计时攻击的算法,其攻击模型描述如图 1 所示.

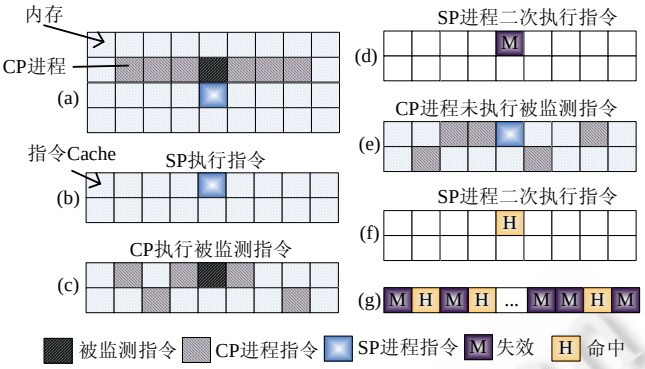

Fig.1 The previous trace driven I-cache timing attack model on the RSA algorithm

图 1 原有的针对 RSA 算法的踪迹驱动指令 Cache 计时攻击模型

图 1 中的被监测指令对应于 RSA 实现算法中的乘法操作指令或额外约减操作指令,CP 和 SP 分别是指密码进程和间谍进程.攻击者要判断被监测指令在密码进程执行过程中是否被执行过,具体操作步骤如下:

(1) 首先,构造间谍进程指令使之与被监测指令映射到相同的逻辑地址(如图 1(a)所示),并允许间谍进程在密码进程启动前能够执行代码,将 SP 进程指令载入指令 Cache 中(如图 1(b)所示),而且能够随时挂起,这要求控制好间谍进程的监视时机.这一技术,Neve 等人在文献[11]中已经针对 AES 最后一轮算法攻击进行了实现.文献[13]介绍过类似的思想,它利用操作系统调度机制改变 CPU 周期,从而控制间谍进程检测密码进程的指令执行时机,这些技术能够应用于微架构分析攻击;

(2) 然后,启动密码进程执行解密操作,操作过程中如果执行了被监测指令,则会访问被监视的 Cache 组区域,从而导致原先 SP 进程拥有在指令 Cache 中的垃圾指令被驱逐(如图 1(c)所示),导致 SP 进程二次执行指令时发生 Cache 失效,而总体执行时间增加(如图 1(d)所示);反之,如果 CP 进程没有执行被监测指令(如图 1(e)所示),则 SP 进程二次执行指令时发生 Cache 命中,而消耗较少的时间(如图 1(f)所示).通过判断 SP 每次执行所消耗的时间差异,可以判断被监测指令是否被执行;

(3) SP 进程以一定的频率循环执行私有指令并测量每次执行的时间,直到密码进程结束后才终止,从而可以采集密码进程的执行踪迹(如图 1(g)所示).即对应于算法的执行流,再依据执行流与密钥信息的依赖关系推导出密钥的值.

该攻击算法在理想情况下,攻击者只要采集 RSA 算法 1 次签名操作的执行踪迹就能推导出完整的密钥.

1.3 原有指令Cache计时攻击的局限性

原有的针对 RSA 算法的指令 Cache 攻击需要构造与被监测指令映射到相同逻辑地址的间谍进程,其假设前提是首先能够利用反编译的方式获取密码程序中被监视的操作函数所映射的逻辑地址,再构造会与该逻辑地址存在冲突的一段间谍进程代码,通过间谍进程执行时访问指令时的 Cache 命中和失效的行为来判断密码进程中被监测的操作函数是否被执行过,最后,利用这些特定函数的执行序列来推导出密钥的相关信息.但是,在实际执行攻击时可能存在一定的局限性:

(1) 原有攻击的假设前提过于严格.由于一般情况下被攻击目标是在远程服务器上提供验证服务的密码程序,会采取一定的安全措施,因此难以获取目标程序进行反编译,从而获取不到关键指令映射的逻辑地址.如果无法获取对应的逻辑地址,则原有攻击方法失效;

(2) 即使能够构造与被监测指令所映射的逻辑地址相同的间谍进程,但是,依据 Cache 映射的原理[14]可知,两个逻辑地址相同的指令只能保证其映射到的指令 Cache 组相同,并不能保证所映射的 Cache 行也相同,因此,当 Cache 结构为多路组相连情况时,被监测指令的执行并不能保证一定会驱逐出指令 Cache 中 SP 进程指令.这种情况下,就不能通过二次执行间谍指令的 Cache 命中和失效情况来判断密

码进程的关键指令是否执行过;如果要保证原有攻击可行的话,则需要编写能够填充被检测指令映射的 Cache 组中所有 Cache 行的间谍进程代码;

(3) 原有攻击是通过填充 NOP 指令使间谍进程与被监测指令映射到相同的逻辑地址,因此实际攻击时,间谍进程指令可能还会映射到其他指令映射的 Cache 组,导致攻击者无法区别间谍进程执行时的总体时间差异是来自于被监测指令的执行还是其他指令的执行,最终导致攻击者不能准确判断关键指令是否被执行;

(4) 针对采用滑动窗口算法实现的 RSA 算法指令 Cache 攻击在推导幂指数时,依据算法的执行流能够获取的幂指数位数目非常有限,不像数据 Cache 攻击还能够利用预计算表索引值与窗口大小的相关性以进一步缩小幂指数的搜索空间;如果所能获取的幂指数位数目不足以满足在多项式时间内恢复完整密钥的条件,则攻击失败.

以上分析的 4 点局限性使得原有的攻击与实际攻击之间还存在一定的距离.

2 基于整个指令 Cache 的踪迹驱动计时攻击模型

针对原有指令 Cache 攻击存在的局限性,受 RSA 踪迹驱动数据 Cache 计时攻击思想[4]的启发,本文提出一种监视整个指令 Cache 的踪迹驱动计时攻击模型.假设在被监测指令对应的逻辑地址未知的情况下执行攻击,构造的间谍进程不仅能填充被监测指令所映射的 Cache 组,还将其他所有的指令 Cache 组也都填充,从而监测整个指令 Cache 的访问情况,具体模型描述如图 2 所示.

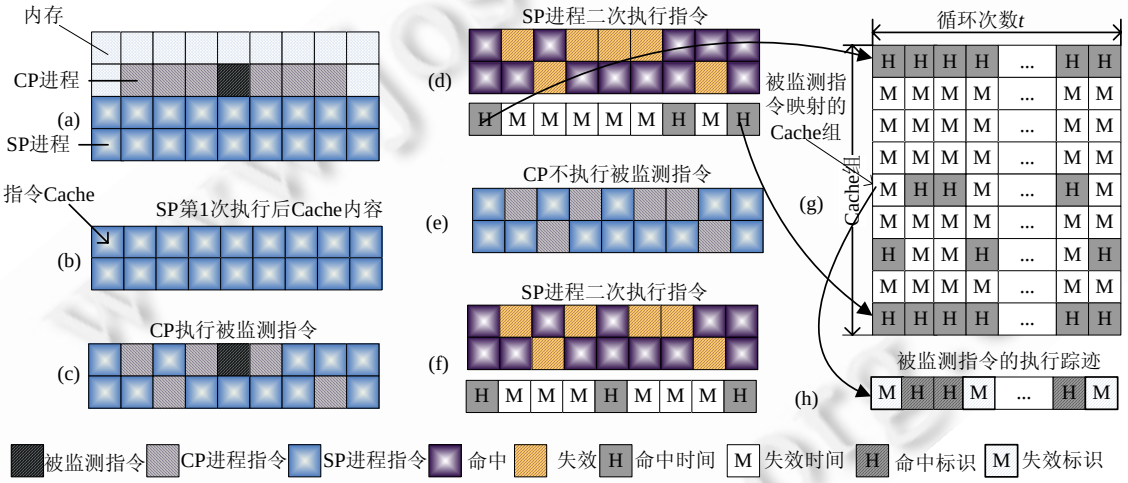

Fig.2 The trace driven timing attack model based on the whole I-cache

图 2 基于整个指令 Cache 的踪迹驱动计时攻击模型

为了方便描述,图 2 中对内存的结构和 Cache 结构进行了简化,被监测指令属于 CP 进程指令.根据执行环境中的 L1 指令 Cache 的结构和大小,利用填充 nop 空指令的方式编写间谍进程代码,使其执行时刚好能够填充整个指令 Cache(如图 2(a)所示).执行步骤大体上与原有模型差不多,但由于新的模型是监测整个指令 Cache 的访问状态,不只是被监测指令所映射的 Cache 组访问状态,因此,Cache 计时数据的采集及分析有所不同,具体操作步骤如下所示:

(1) SP 进程与 CP 进程在同一个处理器上同步执行,共享同一指令 Cache 资源,在 CP 进程启动前执行,将 SP 进程指令填充整个指令 Cache(如图 2(b)所示),并测量记录执行时访问每个 Cache 组中所有 Cache 行的时间;

(2) 启动 CP 进程开始执行解密/签名运算,执行的指令会访问被监视的 Cache 组区域,导致原先 SP 进程

拥有在指令 Cache 中的垃圾指令被驱逐.如果执行被检测指令(如图 2(c)所示),则其对应的 Cache 行中的 SP 进程指令被驱逐,导致 SP 进程二次执行时,访问被 CP 进程占用的 Cache 行时会发生失效(如图 2(d)所示);如果被检测指令不被执行(如图 2(e)所示),则 SP 进程二次执行时,访问相应的 Cache 行会发生命中(如图 2(f)所示).根据 Cache 组中的每个 Cache 行访问时间计算出每个 Cache 组的访问总时间,作为 Cache 组命中和失效的依据;

(3) SP 进程以一定的频率循环执行图 2(d)和图 2(f)所示的操作步骤,并测量每次执行时访问每个 Cache 组所需时间,直到密码进程结束后才终止,从而监视密码进程执行时的所有踪迹(如图 2(g)所示);

(4) 通过对被攻击密码算法内部实现方式的分析,利用实现算法的操作序列特征,如平方乘算法和滑动窗口算法中的平方和乘法操作序列,不可能连续出现两个乘法操作,根据所采集的算法执行踪迹可以判断出被检测指令所映射的 Cache 组位置(如图 2(g)所示),从而筛选出被检测指令的执行踪迹(如图 2(h)所示),即可知道算法的执行流,最终推导出密钥的值.

3 指令 Cache 计时分析算法

3.1 指令Cache计时信息的采集

本文在新的指令 Cache 计时攻击模型的基础上,结合文献[9]中的 Cache 计时数据分析方法,假设计时数据都是由一个个独立的 Cache 组访问时间组成的,同时参考 Percival 提出的数据 Cache 计时攻击中实现间谍进程与密码进程同步执行的方式[4].间谍进程执行自己的指令填充整个 I-Cache,并测量它重新执行代码映射到每个独立 Cache 组的时间,然后不断重复这一过程,采集一序列的指令 Cache 计时数据.

实际执行攻击时,需要依据所运行的处理器中指令 Cache 的结构来编写相应的间谍进程代码.我们给出一种通用的监测指令 Cache 的间谍进程,以实验中所采用的 Intel Core i3 处理器为分析对象,给出间谍进程代码示例. Core i3 为具有同步多线程能力的双核处理器,每个核的 L1 指令 Cache 的大小为 32KB,4 路组相连,Cache 行大小为 64 字节,共有 j=512 个 Cache 行,C=128 个 Cache 组.针对不同的 Cache 结构,间谍进程需要作相应的改变.我们设计一个个连续的 64byte 大小的代码块(刚好与 Cache 行的大小相同),分别标识为 $L=\{L_0,L_1,\ldots,L_{511}\}$,并将映射到相同 Cache 组的每个子集表示为 $l_i=\{L_j \in l: j \bmod C=i\}$,每个子集对应的代码块映射到同一 Cache 组,而且具有不同的逻辑地址标识,这些 Cache 组组成整个 Cache, $l=\bigcup_{i=0}^{c-1} l_i$.通过执行 l_i 子集的指令填充对应的第 i 个 Cache 组的所有 Cache 行,重复遍历执行所有的子集以填充整个指令 Cache.

间谍进程反复顺序执行这些 l_i 子集的代码,并测量每个独立的执行时间.例如,它从 l_0 的代码开始执行,$l_0=\{L_0,L_{128},L_{256},L_{384}\}$,并保存执行时间,将其标识为第 0 个 Cache 组的访问时间;然后再执行下一个 l_1 的代码:$l_1=\{L_1,L_{129},L_{257},L_{385}\}$,并测量执行时间,将其标识为第 1 个 Cache 组的访问时间;重复执行 l_i,$0 \leqslant i<128$,直到所有的子集都执行完毕.对于每个 l_i,我们都能得到一个唯一的测量时间,重复循环执行,直到密码进程执行结束,从而获取密码进程执行中泄露出的指令 Cache 旁路信息.依据实验中 Core i3 的 L1 指令 Cache 结构,本文编写的部分间谍进程代码示例如图 3 所示.

代码中 0x800 表示重复执行的次数,攻击时依据间谍进程循环 1 次的执行时间以及密码进程的整个执行时间进行相应的更改.其中,代码大部分由 nop 空指令组成,但是它们仅仅被用于填充而并没有执行.依据程序的局部性原理,当其中一个指令被执行时,会把包括被执行指令在内的一整块 Cache 行 64 字节大小的指令都载入 Cache 中.每个 nop 指令大小为 1 字节,每个代码块的其余指令加 59 个 nop 指令刚好与一个 Cache 行的大小相等.另外,rdstc 是时间戳指令,用于记录访问每个 Cache 组的执行时间,记录的数据存放在 eax 寄存器中,再通过 movnti 指令将结果保存在 exc 寄存器中.与文献[9]所采取的方式不同,采用支持 SSE2 指令集的 CPU 中的 movnti 指令替代 movb 指令,实现记录执行时间时通过 CPU cache 直接访问内存,减小了记录操作对测量时间带来的影响.

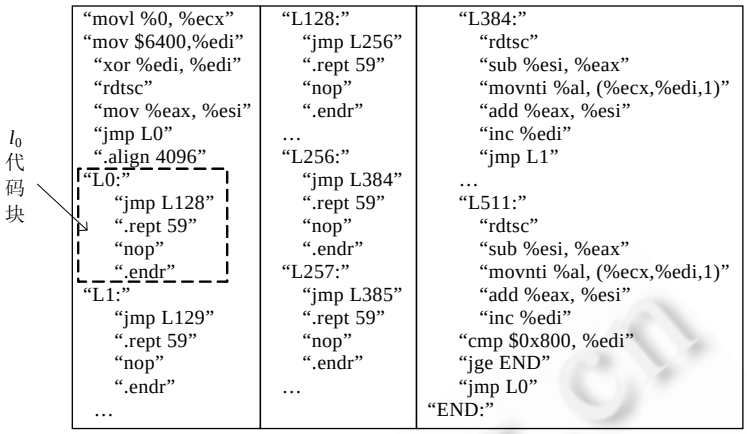

Fig.3 Outline of a generic I-cache spy process which can spy on the whole I-cache accessing status

图 3 监视整个指令 Cache 访问状态的部分间谍进程代码

3.2 改进的幂指数分析算法

根据攻击模型可知,通过间谍进程采集的指令 Cache 计时数据能够获取密码进程执行时被监测指令,即关键指令的执行流,从而推导出密码算法执行时依赖于密钥的操作序列.RSA 算法中的核心运算是模幂运算,无论是密钥还是密钥相关值,如中国剩余定理实现时的 d_p 和 d_q,都是对应于模幂运算的幂指数.因此,分析幂指数即分析密钥信息.本节在假设已经获取这些操作序列的前提下,研究如何分析出幂指数.

Aciicmez 提出的指令 Cache 分析算法[2]针对平方乘算法实现的 RSA 在执行攻击时,可以直接利用幂指数位为 1 时比为 0 的情况多执行一个乘法操作的特性,已经获取的平方和乘法操作序列,依据 SM→1,S→0 的对应关系恢复出完整幂指数,其中,S 表示平方操作,M 表示乘法操作;针对滑动窗口算法实现的 RSA 在执行攻击时,根据滑动窗口的滑动规则:每个窗口位都以 1 为开头,再以 1 为结尾,算法在每个窗口位滑动结束时才会执行一个乘法操作.当窗口大小 *win_size*=5 时,可以依据平方和乘法操作序列推导出部分幂指数位的值:如果发生乘法操作,可推导出对应的幂指数位为 1.从执行的平方乘法操作序列可以发现,如果连续出现 5 次以上的平方操作,则可以推导出窗口大小开始计算前滑过幂指数位为 0 的数目,具体分析幂指数位的过程如图 4 所示.图中以从左到右的滑动窗口算法为例,依据图 4 中的 4 个乘法操作可以推导出 4 个幂指数位为 1,由连续出现的 8 个平方操作可推导出 3 个幂指数位为 0.

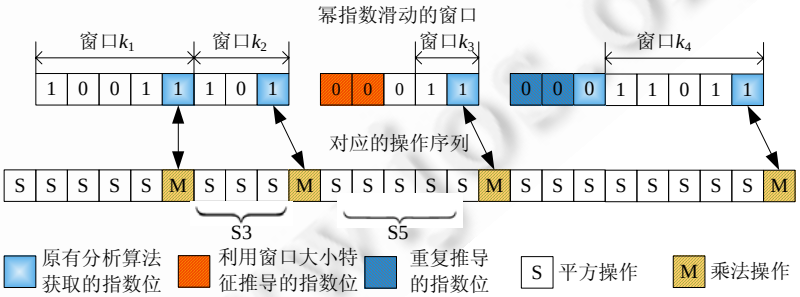

Fig.4 The relation between the the bits of the exponent and the sequence of multiplications and squarings

图 4 幂指数位与平方和乘法操作序列的映射关系

经过分析发现,对于 512 位的幂指数进行分析,原有的指令 Cache 攻击中的幂指数分析算法依据关键函数的操作序列所能获取的幂指数位数目非常有限,能够获取大约 200 个幂指数位,不像 RSA 数据 Cache 攻击[4]还

能够利用预计算表索引值与窗口大小的相关性,额外获取大约 110 个指数位,因而可以进一步缩小幂指数的搜索空间.如果所能获取的幂指数位数目不足以满足在多项式时间内恢复完整密钥的条件,则攻击失败.

本文利用滑动窗口算法中的窗口滑动规则:每次滑动窗口时,从指数位为 1 开始滑动,滑动数目与窗口大小相等的位,并以最右边的指数位为 1 的位置作为当前窗口的结束位置,计算当前窗口的大小 *now_size* 以及窗口值 *wvalue*.我们提出了一种新的基于窗口大小特征的分析算法:从滑动规则可知,如果从操作序列发现当前窗口大小小于 *win_size*,即可判断当前窗口之后存在(*win_size*−*now_size*)个为 0 的幂指数位.如图 4 所示,从操作序列中 3 个连续的平方操作“S3”可以判断出窗口 k_2 的大小最大为 3,因此可以推导出窗口 k_2 后面存在 2 个指数位为 0.新的分析算法可以充分利用已经推导的幂指数信息,例如,对于图中大小为 2 的窗口 k_2,由于对应的操作序列存在 5 个平方操作“S5”,因此,如果仅仅根据这一独立的信息并不能判断窗口大小为 2,而无法推导出窗口后面是否存在 3 个值为 0 的指数位.但是可以利用前面获取的 2 个幂指数为 0 的信息,将窗口 k_2 大小的最大值缩小为 3,从而运用新的推导算法可知,窗口 k_2 后面存在 2 个指数位为 0.在原有分析算法的基础上组合新的分析算法能够推导出更多的幂指数位.以图 4 为例,原有的分析算法能够获取 7 个指数位,增加新的分析算法后,除了 2 个重复推导的指数位以外,还能获取 2 个指数位,从而在 Cache 计时信息一定的情况下,提高了旁路信息的利用率,缩小了密钥的搜索空间.

4 实验结果及分析

4.1 实验配置与攻击步骤

OpenSSL 密码库中的 RSA 算法实现方式是滑动窗口算法还是固定窗口算法是可选择的.在本文提出的攻击模型的基础上,针对滑动窗口实现的 RSA 算法执行攻击实验,该攻击方案同样适用于采用平方乘算法实现的 RSA 算法,只是幂指数分析算法会有所差异,受篇幅所限,这里只给出针对滑动窗口算法的实验结果.依据 OpenSSL 实现平方和乘法操作时调用 BIGNUM 函数库中的函数不同,编写针对 RSA 算法指令 Cache 攻击的间谍程序,监视 RSA 签名算法的执行过程以获取其执行流,分析 RSA 签名密钥.在 Linux 操作系统中,利用处理器的同步多线程能力以及 fork 函数实现密码进程与间谍进程同步执行.为保证每一次算法执行时被监测指令都不在 Cache 中,间谍进程需要在每次操作之前“清空”Cache,将构造的指令填充整个指令 Cache.同时,为了便于研究,最小化系统中其他进程的运行,在具有超线程能力的 Core i3 处理器上执行攻击实验,具体实验环境配置见表 1.

Table 1 Environment conguration of the I-cache timing attack on RSA

表 1 针对 RSA 算法的指令 Cache 攻击实验环境配置

配置项	参数	
操作系统	Linux Fedora 8	
超线程是否开启	是	
OpenSSL	OpenSSL v.0.9.8f	
CPU	Intel Core i3 2.53 GHz	
内存	2 GB	
L1 Cache	Cache size: 32KB	Associative size: 4 way
	Cache line size: 64B	Number of cache sets: 128
L2 Cache	Cache size: 256KB	Associative size: 8 way
	Cache line size: 64B	

在 Linux 操作系统中,攻击进程启动间谍进程 SP,同时,利用 *fork*()函数创建一个带有独立虚拟地址空间的密码进程 CP,执行签名操作,当 *fork* 在 CP 中返回时,CP 现在的虚拟存储器刚好与调用 *fork* 时存在的虚拟存储器相同,使 SP 与 CP 共享同一指令 Cache.为了保证 SP 能够在 CP 执行前启动而在 CP 执行完毕后终止,当 AP 调用 *fork*()函数启动 CP 进程时,先调用高精度休眠函数 *nanosleep*,将休眠时间精确到毫微秒级,实验中设置的 *timespec* 的参数为{0,0x30000}.同时,依据 CP 的执行时间长短以及 SP 单次执行构造的间谍代码所需时间,设置 SP 循环采集的次数为 2 000 次.实验过程中,为了提高采集计时数据的精度,同时最小化其他系统进程,依据图 2

攻击模型的执行步骤以及第 3.1 节的指令 Cache 计时数据采集方法执行攻击实验.

4.2 结果分析与对比

实验结果主要分为两部分:一是利用获取的 Cache 计时信息推导出平方和乘法算法的操作序列;二是依据获取的操作序列恢复出幂指数的比特位信息.实验中,针对 OpenSSL 中 RSA 算法随机生成的 1 024 位签名密钥执行攻击.由于 RSA 算法为提高执行速度采用中国剩余定理实现,那么我们的攻击目标是获取幂指数 d_p 和 d_q,下面以指数 d_p 的攻击过程为例进行分析.

(1) 定位滑动窗口操作的起始位置

间谍进程采集的是整个密码进程执行过程中的 Cache 状态访问数据,但是恢复算法操作序列时只需要滑动窗口操作阶段对应的 Cache 计时数据,将该部分数据称为有效数据,其余为无效数据.尽管间谍进程执行循环采集 2 000 次,但是为了保证 SP 在 CP 执行前启动,CP 执行前的休眠时间所对应的数据为无效数据.此外,滑动窗口算法执行前的初始化操作、蒙哥马利转换操作等对应的数据也为无效数据.经过筛选,共采集大约 1 480 次有效数据,对 128 个 Cache 组数据,经过观察挑选出其中的 64 个 Cache 组数据进行分析.利用间谍进程采集 1 次 RSA 签名算法的执行踪迹,部分结果如图 5 所示.

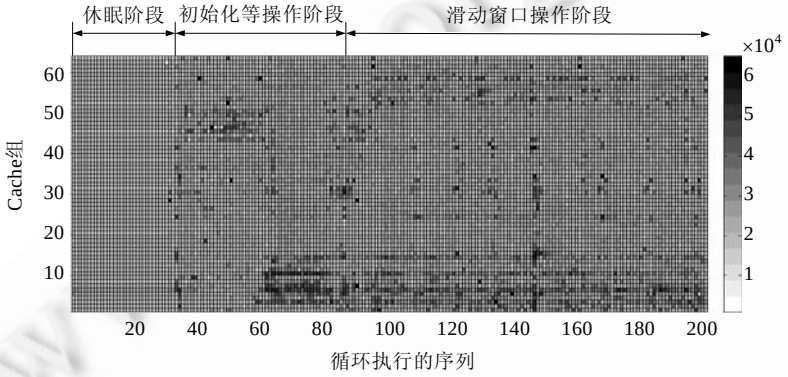

Fig.5 The pseudo-color map result of partial I-cache timing traces in our experiment

图 5 采集的指令 Cache 计时数据的伪彩色图表示结果

图 5 中用不同的颜色表示访问每个 Cache 组的时钟周期数,横坐标 X 表示间谍进程循环执行的序列,对应于采集的时间序列;纵坐标 Y 表示挑选的 64 个 Cache 组.从图中可以明显看出,CP 进程大约在 X=35 时开始启动执行签名操作,因为在休眠阶段只执行 SP 进程,基本上访问所有的 Cache 组都发生 Cache 命中,除了部分受系统进程干扰外.从 X=85 左右开始出现有规律的 Cache 访问踪迹,可以判断 CP 此时开始执行滑动窗口计算.利用同样的方法可以判断 CP 执行滑动窗口计算的结束位置以及整个模幂运算执行完毕的位置,从而筛选出对应于滑动窗口算法执行过程的计时信息集合 T_s.

(2) 平方和乘法操作序列恢复结果

如图 6 所示,对筛选出的有效数据进行分析,利用 Cache 访问时命中和失效的特性,可以观察平方和乘法操作函数指令映射的 Cache 组位置.利用图 6 中获取的 Cache 计时数据可以推导出共发生 7 次乘法操作,34 次平方操作,推导出操作序列为 MSSSSSSMSSSSMSSSSSMSSSSSMSSSSSSSMSSSSSM.

实验结果表明,攻击者无需知道密码进程关键指令映射的逻辑地址,仍然能够利用间谍进程获取算法的执行流,证明了新的攻击模型的有效性.用操作序列的完整率表示能够正确恢复的操作序列数目占总的操作数目的比率,两种攻击方案对应的样本量大小与恢复操作序列的完整率关系如图 7 所示.

实际攻击中受各种因素的影响,采集 1 次 RSA 签名的执行踪迹所能获取的操作序列数目很有限,但可以通过多次采集的方法提高获取操作序列的完整率.新的攻击方案不存在被监测指令的执行不会驱逐出指令 Cache

中 SP 进程指令的可能性,在相同样本量大小的条件下,能够获取更多的操作序列,获取的操作序列完整率比原有攻击方案高了 12%~20%.而原有的攻击方案受第 1.3 节中所分析的局限性影响,能够恢复的操作序列完整率相对有限.实验结果仍然存在噪声的影响,导致不能完整获取操作序列.

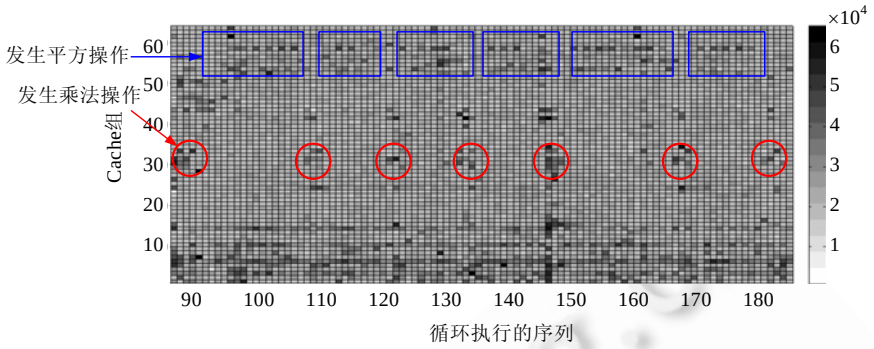

Fig.6 The pseudo-color map result of partial effective I-cache timing traces

图 6 指令 Cache 计时有效数据的伪彩色图表示结果

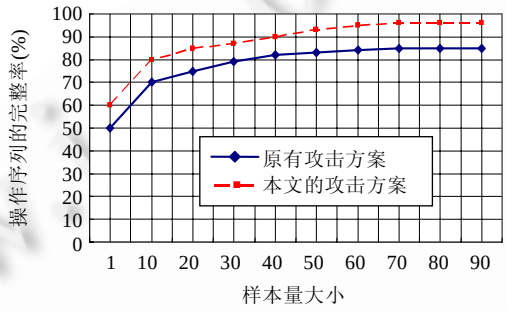

Fig.7 The relationship between the sample size and the integrality of the sequence of operations

图 7 重复操作的样本量大小与恢复操作序列的完整率关系

(3) 幂指数的分析结果

以图 6 中获取的平方和乘法操作序列为例进行幂指数分析.原有的分析算法能够推导出的幂指数比特位为 10XXXX1XXX1XXXX1XXXX100XXXX1XXXX1.其中,X 表示未知幂指数位,共获取 10 个比特位.利用第 3.2 节的分析算法获取的幂指数比特位为 10XXXX1XXX10XXX10XXX100XXXX1XXXX1,共获取 12 个比特位.对于随机生成的一个 512 位幂指数 d_p,共执行 90 次乘法操作、511 次平方操作,出现 70 次连续 5 个以上的平方操作,利用操作序列与幂指数相关性可推算出 d_p 中 90 个值为 1 的位,122 个值为 0 的位,即原有的分析算法共获取 212 个指数位;再利用窗口大小特征,可额外推导出 45 个值为 0 的幂指数位,最终获取 d_p 中 257 个离散的比特位.利用相同的方法可以获取幂指数 d_q 中大约 256 个比特位.利用格攻击算法[15]获取一半以上的幂指数信息,能够在多项式时间内恢复完整密钥.实验结果表明,新的幂指数分析算法提高了指令 Cache 计时攻击的有效性,而原有的分析算法仅能够获取 200 个左右的幂指数位,最终无法破解密钥.

4.3 讨 论

实验结果表明:新的攻击模型在相对宽松的假设前提下仍然可以获取算法的操作序列,新的分析算法能够进一步推导出更多的幂指数位,从而提高指令 Cache 计时攻击的可行性和有效性.同时,由于指令 Cache 计时攻击是基于密码算法执行流与密钥的相关性,而非数据访问模型,因此,即使有些密码算法采取了抵御数据 Cache 攻击的措施,也并不能保证对于指令 Cache 攻击仍然有效.实验中只是对采取滑动窗口算法的 RSA 算法进行分

析,对于执行流依赖于密钥的其他密码算法同样存在遭受指令 Cache 攻击的可能性,例如 ECDSA 算法;而对于执行流相对固定的分组密码算法,指令 Cache 攻击则存在一定的困难性,甚至不可能.

踪迹驱动计时攻击算法在理论上只需间谍进程采集一次密码算法执行的指令 Cache 计时踪迹就能够恢复出完整的密钥,攻击性较强,不像时序攻击需要采集百万次以上的数据.利用编写的间谍进程不仅能够执行本地攻击,而且利用植入木马的方式同样能够作用于远程攻击.

在未来的研究工作中,对于实际的攻击过程,重点关注以下几点:

(1) 提高数据分析精度.在同一个主机中,可能同时执行许多其他的进程,会对间谍进程采集的指令 Cache 计时数据引入一定的噪声,从而影响分析的精度,包括滑动窗口操作起始位置的判断、关键指令映射的 Cache 组的映射位置、平方和乘法操作数目以及序列顺序等等.因此,需要引入能够有效消除或减小噪声影响的分析方法,提高数据分析的精度;

(2) 充分利用计时信息,获取更多的幂指数比特位.原有的分析算法能够获取大约 200 个离散的比特位,目前仍然无法证明已知的 200 个幂指数位是否已经足够破解 RSA,而新的分析算法尽管能够使得攻击者在多项式时间内恢复完整密钥信息.但实际攻击中,噪声等因素的影响可能会使获取的幂指数位数目有所降低,使得整个攻击的复杂度很高.因此,需要寻找能够进一步推导幂指数比特位信息的算法,以提高攻击效率;

(3) 以本文的研究为基础,对其他公钥密码算法的指令 Cache 计时攻击安全性进行验证.尽管从理论上可以证明算法执行流依赖于密钥的密码算法都存在遭受指令 Cache 攻击的可能性,但其实际攻击的可行性仍然有待通过实际的攻击加以论证.

综上所述,新的攻击方案比原有的攻击方案具有更好的可行性及执行效率,但在实际攻击应用中仍有不少困难需要克服.在下一步的研究中,我们需要研究如何尽可能地排除或减小噪声的影响,比如引入随机过程的分析方法;寻找能够获取更多幂指数位的数据分析方法,同时给出能够有效抵御指令 Cache 攻击的措施.

5 总 结

本文对 RSA 算法的踪迹驱动指令 Cache 计时攻击进行了相关研究,分析了原有攻击算法在实际攻击中的困难性,建立了一种新的基于整个指令 Cache 的踪迹驱动计时攻击模型,并提出了能够获取更多幂指数位的分析算法,针对 OpenSSL 密码库的 RSA 算法执行攻击实验.实验结果表明:新的踪迹驱动计时攻击模型比原有的攻击模型具有更好的可操作性,从而提高了指令 Cache 攻击的可行性.新的幂指数分析算法在一定操作序列信息的情况下,能够推导出更多的幂指数比特位信息,使得攻击者有可能在多项式时间内恢复完整密钥,提高了针对 RSA 算法的指令 Cache 攻击的有效性.由于此类攻击比时序驱动攻击和访问驱动攻击所需的样本量要小、攻击性要强,同时,对于已经采取抵御数据 Cache 攻击措施的密码算法仍然有效,因此,对于此类攻击应引起充分的关注.

在实际攻击过程中,由于指令 Cache 计时攻击受外界因素影响较大,不同的执行环境、不同的密码库以及密码算法实现方式等因素都会影响攻击的执行效果,仍然存在噪声干扰大以及 Cache 组定位不准等问题.另外,如何寻找一种既能有效抵御此类攻击又能保证密码算法性能的防御措施,这都需要我们做进一步的工作.

References:

[1] Kocher PC. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Proc. of the CRYPTO'96. LNCS 1109, Berlin: Springer-Verlag, 1996. 104−113. [doi: 10.1007/3-540-68697-5_9]

[2] Aciicmez O, Gueron S, Seifert JP. Micro-Architectural cryptanalysis. IEEE Security and Privacy, 2007,5(4):62−64. [doi: 10.1109/MSP.2007.91]

[3] Aciicmez O. Yet another micro-architectural attack: Exploiting I-cache. In: Proc. of the ACM Workshop on Computer Security Architecture. New York: ACM Press, 2007. 11−18. [doi: 10.1145/1314466.1314469]

[4] Percival C. Cache missing for fun and profit. In: Proc. of the Technical BSD Conf. 2005. Ottawa, 2005. 1−13. http://www.daemonology.net/papers/htt.pdf

[5] Aciicmez O, Schindler W. A major vulnerability in RSA implementations due to micro-architectural analysis threat. In: Tal M, ed. Proc. of the 14th ACM Conf. on Computer and Communications Security (ACM CCS 2007). New York: ACM Press, 2008. 256−273.

[6] Acıicmez O, Schindler W. A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL. In: Proc. of the Topics in Cryptology (CT-RSA 2008). LNCS 4964, Berlin: Springer-Verlag, 2008. 256−273. [doi: 10.1007/978-3-540-79263-5_16]

[7] Aciicmez O, Koc. CK, Seifert JP. On the power of simple branch prediction analysis. In: Proc. of the 2nd ACM Symp. on Information, Computer and Communications Security (ASIACCS 2007). New York: ACM Press, 2007. 312−320. [doi: 10.1145/1229285.1266999]

[8] Aciicmez O, Koc CK, Seifert JP. Predicting secret keys via branch prediction. In: Proc. of the Topics in Cryptology (CT-RSA 2007). LNCS 4377, Berlin: Springer-Verlag, 2007. 225−242. [doi: 10.1007/11967668_15]

[9] Aciicmez O, Brumley BB, Grabher P. New results on instruction cache attacks. In: Proc. of the Cryptographic Hardware and Embedded Systems (CHES 2010). LNCS 6225, Berlin: Springer-Verlag, 2010. 110−124. [doi: 10.1007/978-3-642-15031-9_8]

[10] Tiri K, Aciicmez O, Neve M, Andersen F. An analytical model for time-driven cache attacks. In: Alex B, ed. Proc. of the Fast Software Encryption (FSE 2007). LNCS 4593, Berlin: Springer-Verlag, 2007. 399−413. [doi: 10.1007/978-3-540-74619-5_25]

[11] Neve M, Seifert JP. Advances on access-driven cache attacks on AES. In: Biham E, *et al*., eds. Proc. of the 13th Int'l Selected Areas of Cryptography (SAC 2006). LNCS 4356, Berlin: Springer-Verlag, 2007. 147−162. [doi: 10.1007/978-3-540-74462-7_11]

[12] The OpenSSL project: OpenSSL: The open source toolkit for SSL/TLS. 2013. http://www.openssl.org

[13] Tsafrir D, Etsion Y, Feitelson DG. Secretly monopolizing the CPU without superuser privileges. In: Proc. of the 16th USENIX Security Symp. 2007. 239−256. http://static.usenix.org/event/sec07/tech/full_papers/tsafrir/tsafrir_html/

[14] Bryant RE, O'Hallaron D, Wrote; Gong YL, Lei YC, Trans. Computer Systems: A Programmer's Perspective. 2nd ed., Beijing: China Machine Press, 2011. 408−433 (in Chinese).

[15] Coppersmith D. Finding a small root of a bivariate integer equation: Factoring with high bits known. In: Maurer U, ed. Proc. of the Advances in Cryptology EUROCRYPT'96. LNCS 1070, Berlin: Springer-Verlag, 1996. 178−189. [doi: 10.1007/3-540-68339-9_16]

附中文参考文献:

[14] Bryant RE, O'Hallaron D,著;龚奕利,雷迎春,译.深入理解计算机系统.第 2 版,北京:机械工业出版社,2011.408−433.

陈财森(1983－),男,福建漳州人,博士,CCF 学生会员,主要研究领域为信息安全对抗,公钥密码旁路分析.
E-mail: caisenchen@163.com

郭世泽(1969－),男,博士,研究员,博士生导师,主要研究领域为信息安全,密码学.
E-mail: tigerone-gsz@vip.sina.com

王韬(1964－),男,博士,教授,博士生导师,主要研究领域为信息安全对抗,密码旁路分析.
E-mail: twangdr@hotmail.com

周平(1988－),男,博士生,主要研究领域为公钥密码微架构分析.
E-mail: Zhou.Ping_01@hotmail.com