**北京航空航天大学计算机学院**

**硕士学位论文开题报告**

**论文题目：**面向移动设备的cache攻击关键技术研究

**专 业：**软件工程

**研究方向：**软件工程

**研 究 生：**李勃

**学 号：**SY1506402

**指导教师：**姜博副教授

**北京航空航天大学计算机学院**

2016年12月10日

目 录

[1 论文选题背景和意义 1](#_Toc469874580)

[2 国内外研究现状及趋势 2](#_Toc469874581)

[2.1 国外研究现状 2](#_Toc469874582)

[2.2 国内研究现状 4](#_Toc469874583)

[3 论文研究内容及拟采取技术方案 4](#_Toc469874584)

[3.1 研究目标 4](#_Toc469874585)

[3.2 研究内容 5](#_Toc469874586)

[3.2.1 计时方式 5](#_Toc469874587)

[3.2.2 驱逐策略 6](#_Toc469874588)

[3.2.3 攻击方式 8](#_Toc469874589)

[3.2.4 keylogger攻击 8](#_Toc469874590)

[3.2.5 高级加密AES攻击 8](#_Toc469874591)

[3.2.6 可执行环境TEE攻击 10](#_Toc469874592)

[4 关键技术及难点 10](#_Toc469874593)

[4.1 确定驱逐策略 10](#_Toc469874594)

[4.2 探索异步攻击的方式 10](#_Toc469874595)

[4.3 探索对AES进行攻击的方式 11](#_Toc469874596)

[4.4 探索对可信执行环境TEE的攻击方式 11](#_Toc469874597)

[5 前期工作 11](#_Toc469874598)

[5.1 通过POSIX实现精确计时 11](#_Toc469874599)

[5.2 实现evict + reload和prime + probe 12](#_Toc469874600)

[5.3 访存操作对probe时间的影响 13](#_Toc469874601)

[5.4 Prime + probe异步aes攻击实验 15](#_Toc469874602)

[6 论文研究计划 19](#_Toc469874603)

[7 主要参考文献 20](#_Toc469874604)

# 论文选题背景和意义

随着移动互联网近些年突飞猛进的发展，手机等移动设备已经成为我们生活必不可少的一部分。它在给我们生活带来便利的同时，也给我们到来了潜在的威胁，尤其是在手机能方便管理资产，存储着各种隐私的今天。因此，随着科技的进步，安全问题也越发得到人们的重视。虽然手机等设备的开发厂商以及Android系统的开发公司采取了各种措施来保护用户的隐私安全，包括可信执行环境，虚拟内存管理，权限管理等。然而，由于电脑，手机等设备结构设计的缺陷，并随着安全领域不断的研究，一些手机设备的安全漏洞和攻击方式逐渐的显露出来，其中包括cache旁路攻击。Cache旁路攻击是通过探测程序执行时cache的hit或miss，访存时间的多少等信息来进行攻击的方法。其中攻击程序与被攻击程序没有直接的交互，他们并行执行在同一个核或不同的核上，访问各自的地址空间，因此攻击时不需要多余的权限。

在最近10年中，基于x86平台的CPU cache攻击得到了人们越来越多的重视。Kocher[1]和Kelsey[2]等人提出了通过分析高速缓冲存储器在运行时泄露出的旁路信息来破解计算机中的加密算法的方法。这一思想在计算机安全相关人员的重视下得到快速的发展。在最近几年不断提出了在x86平台上的cache攻击方法，并在监测用户键盘输入，AES T-table加密破解方面得到了实验，也证明了cache攻击的有效性。比如，Page[41]等人提出了一种针对DES加密技术的cache模型攻击的方法，将DES密钥的搜索空间从56位降低到32位。Tsunoo[26]通过分析DES查表索引以及Cache的访问特征首次实现了针对DES的Cache 攻击，并在一台600-MHz Pentium III 的个人计算机上，通过223个明文样本成功获取DES 全部密钥。Bernstein[28]在强制消除网络传输时延条件下实现了一种针对OpenSSL[29]中AES 的远程时序驱动计时攻击，其中加密服务端采集AES 加密时间。Bonneau[30]提出了一种利用密码程序在加密中的内部数据访问冲突导致的Cache 命中和失效信息来进行密码分析的方法。Percival[36]提出多线程间共享Cache 存储器访问引发的漏洞，它不仅提供了线程间一个简单、高带宽的隐秘通道，也给恶意线程监视其他线程提供了入口，使得恶意线程能够窃取加密密钥，他依次设计实现了一种针对RSA 的计时攻击。Osvik[37]借鉴Percival信息采集方法实现了多例针对AES 的Cache 计时攻击。Neve[38]将Osvik攻击[37]切入点转移到最后一轮，提出了一种新的AES 最后一轮访问驱动的Cache计时攻击方法。可见，通过这些漏洞，攻击者不仅可以获得用户键盘输入等的隐秘信息，还能获得价值更高的密钥等信息，进而威胁到用户的财产安全。

然而，由于Android等移动设备的体系结构与Intel x86结构有很大的区别，在指令集合cache组织方式以及cache替换策略等与cache攻击相关结构也有很大的不同。因此，直到最近为止，才有人提出在非root的手机上的有效的跨核 cache攻击方法。Moritz Lipp et al. [24] 提出了通过prime + probe， flush + reload， evict + reload以及flush + flush对ARM处理器的跨核攻击模型，并且不需要root权限。这些模型能够有效的探测到在被攻击程序运行时cache无意间泄露出来的信息，通过对这些信息进行统计分析，并将其作用于cache攻击的模型即可提取用户的私密信息。其中最典型的攻击模型为cache模板攻击，该模型在探测阶段不断的探测待攻击时间执行时共享库地址空间的加载情况，形成一个cache模板矩阵，该矩阵对应着某一事件执行时共享库各个地址的访问情况。在通过该模板矩阵进行攻击时，探测用户执行事件时共享库地址的加载情况，并与cache模板矩阵进行对比，进而分析出用户执行的操作。通过近些年的研究，cache旁路攻击已经被认可为一种强大的攻击方法，在研究攻击方法的同时也提出了一些修复漏洞的方法，包括在Android 6.0.1中修复的 /proc/self/pagemap 的访问权限。通过控制在非特权模式下用户线程对自己存储空间页表映射表的权限，能够有效的抑制攻击程序通过共享库执行cache模板攻击。

基于这些背景，本题旨在发掘更多在移动设备端的cache攻击方式，以及有效的攻击模式，主要针对用户输入、高级加密技术（AES）以及可信执行环境，探索并实现有效的cache攻击。挖掘与用户输入和AES，TEE相关的共享库可能泄露的漏洞，进一步针对攻击提出一些应对措施，从而促进移动设备端安全化进程，使用户能够更加安全，更加放心的使用手机。

# 国内外研究现状及趋势

## 国外研究现状

高速缓冲存储器Cache 的行为信息可能作为密码破解的旁路思想，最早是由Kocher[16]和Kelsey[23]等人提出。起初cache时间攻击针对的是加密算法[4][5][6][7][8][9][10]。例如，Bernstein[4]利用AES T-table实现的执行总时间来进行攻击。Percival[9]提出了能够更好的利用内存访问和cache访问时间差别的方法。此外，Osvik et al. 提出了Evict + Time以及Prime + Probe的概念来判断一个指定的cache set是否被待攻击程序占用。这两种方式均能让攻击者判断在待攻击程序运行期间哪些cache sets被占用了，并通过这些信息来攻击加密的实现[11，12，13，14]，还可以构建跨虚拟机的传输信道[15]。Yarom和Falkner[16]提出了Flush + Reload的方法，该方法利用了现代系统结构的三个基础概念来进行攻击。第一，攻击程序和被攻击程序之间共享内存的可利用性；第二，最后一级cache在多个核之间是共享的；第三，Intel平台使用的是inclusive的最后一级cache，意味着从最后一级cache中驱逐的数据也会从其他核的低级cache中驱逐。因此允许任何程序将在其他核上的其他线程或进程的数据驱逐出cache。尽管这种攻击方式的基础是由Gullasch et al.[17]提出的，Yarom和Falkner将其扩展到最后一级共享cache，用以支持跨核攻击。

因此，Flush + Reload允许攻击者判断具体哪条指令被执行或者哪些数据被被攻击程序访问。这种攻击方式能够获得更多更有用的信息，并已经被实现以攻击加密的实现[18][19][20]。此外，Gruss et al. [21] 阐述了基于Flush + Reload的利用单通道cache信息进行自动化攻击的可能性。除了攻击AES T-table类似的加密实现外，他们展现了如何推测键盘敲击的信息甚至通过利用cache边信道来构建按键敲击的日志信息。与之类似，Oren et al.[22]提出了利用cache攻击在Intel平台上攻击JavaScript实现可能性，并展示了如何推测访问的网站信息以及如何跟踪用户的鼠标活动。

Gruss et al. [23] 提出了替换Flush + Reload中的flush操作的Evict + Reload技术。他将flush操作替换为等价的evict操作。尽管这种方式没有在x86 CPUs中实现，Moritz Lipp et al. [24] 展示了其在ARM CPUs上的实现。最近，Flush + Flush [25] 方法被提出，不像其他技术，该方式并不依赖访存的信息，而是依赖刷新指令的时间来判断一个cache line是否被被攻击程序加载。Moritz Lipp et al.展示了在ARMv8-A刷新指令的执行时间依赖于数据是否在缓存中，因此，这种方式也可以用于实现cache攻击。

因为上面所描述的攻击方式都是基于Intel处理器的，因此类似的攻击被认为不能应用到手机等移动设备上，因为这些设备通常有不同的指令集，cache的组织方式也不一样[26]。因此，之前只有关于手机的同核攻击方面的阐述。例如，WeiB et al.[27] 调查了Bernstein cache时间攻击[28]，在一个使用ARM Cortex-A8的微型处理器上。随后WeiB et al. [29]在多核的情况下调查了这种时间攻击。就像WeiB et al.[30]所阐述的那样，在ARM处理器上，系统噪音会使得攻击困难。这一点Spreitzer和Plos[31]通过在不同的ARM Cortex-A8和ARM Cortex-A9手机上进行cache时间攻击也得到证明。他们的调查实验都验证了时间信息被泄露，但是需要花费大量的时间和大量的测量来进行cache攻击。

除了Bernstein攻击外，Bogdanov et al[32].提出了另一个针对AES T-table实现的cache攻击，其利用ARM 9处理器上冲突的特性。此外，有效的分析攻击[33]和电磁发散[34]被用来在ARM处理器上将cache访问可视化。Spreitzer和Plos[35]实现了在Android手机上攻击AES T-table实现的Evict + Time[36]方法。然而到目前为止，仅仅只有针对AES T-table实现的cache攻击。

## 国内研究现状

国内在密码Cache 计时攻击方面起步比较晚，主要对Cache 旁路攻击机理进行研究、对国外攻击进行复制或改进实验.文献[13]使用能量攻击对MARS 和Rijndael 进行了深入的分析;文献[14]对包括Cache 攻击在内近10 年的旁路攻击进展进行了系统介绍和分析;文献[15]介绍了基于Cache 的AES 攻击研究最新进展，分析了攻击的现实可行性，说明了反制攻击的措施建议;文献[16]参考Bonneau 攻击[7]思想，在Intel Celeron 1.99GHz 和Pentium 4 3.6GHz CPU 环境中，分别在221 和225 个随机明文样本条件下，5 分钟内成功恢复OpenSSL v.0.9.8a 库中128 位AES 密钥;文献[18]对Bernstein 攻击[5]在Pentium III，OpenSSL v.0.9.8a 和Miracl 环境下进行了改进实验.。

# 论文研究内容及拟采取技术方案

## 研究目标

本题的研究目标为在目标机（Lenovo K51c78）实现精确的计时方式，并得到快速高效cache驱逐策略，实现有效cache攻击策略，并实现针对用户输入的keylogger攻击、实现端到端的基于AES T-table的cache攻击，并探索对可信执行环境TEE的有效攻击方式。

上述所提到的Evict + Time、Prime + Probe、Flush + Reload以及Evict + Reload等方法虽然都有Intel x86平台下的实现。将他们与有效的时间源计时方式相结合，能够有效的将cache hits和cache misses的情况区分开，从而能够用于实现平台上的cache攻击。然而，由于ARM平台的指令集、cache组织结构以及cache的多级包含关系于x86平台有很大的区别。且cache的替换策略也与x86平台上的最近最少使用算法不一致，ARM所使用的伪随机替换算法增加了cache set的操作难度，驱逐操作的完整性在很大程度上是随机的，因此造成的系统噪音也是ARM移动平台cache攻击研究进度缓慢的原因之一。之前在x86平台上的攻击方式在Android客户端上不一定能发挥作用。此外，之前一些针对于ARM平台的cache攻击将被攻击程序和攻击程序集成到一个程序中，即由攻击程序触发待攻击的加密。由于这种方式能够有效的减少系统切换对占用了的cache set的影响，并且能够缩短驱逐出set到执行被攻击程序的时间，因此能够有效的减少系统噪音，得到更加明显的实验数据。然而在实际的攻击过程中，攻击程序和被攻击程序通常是不同的程序，攻击过程中引入的系统噪音会导致时间测量出现误差，需要对时间测量，驱逐策略等方面进行改进来优化对cache的操控。基于这些思想，本题主要的研究方向为面向移动设备的cache攻击技术研究，实现evict + reload以及prime + probe等工具，并实现对诸如key logger、高级加密（AES）或可信执行环境（TEE）的cache 攻击。

## 研究内容

### 计时方式

一个精确的计时方式是cache攻击的前提，它需要将cache hits和cache misses区分开来，供攻击者获取待攻击程序运行导致的cache状态的变化。并针对不同的攻击对象获取不同的相关信息，比如对共享库的攻击需要获取共享库相关地址在被攻击程序运行期间的cache缓存状况。然而获取这些状况的前提都是拥有能够准确区分cache hits和cache misses的能力。Moritz Lipp et al.[24]虽然提出了几种非特权的计时方式，包括perf\_event\_open、POSIX的clock\_gettime函数以及专用的线程计时器。但这些接口并非对所有Android版本，对所有处理器都开放，因此需要对待攻击机型确定能精确且能稳定测量的时间源，以此提供对cache攻击的支持。

除了通过读取寄存器获取CPU周期来度量时间外，还有其他3种可选方式来度量访存或访问cache的时间。

* clock\_gettime系统调用

"clock\_gettime"是基于Linux C语言的时间函数，他可以精确到纳秒。

#include<time.h>

int clock\_gettime(clockid\_t clk\_id， struct timespec \*tp);

clk\_id : 检索和设置的clk\_id指定的时钟时间。

CLOCK\_REALTIME:系统实时时间，随系统实时时间改变而改变，即从UTC1970-1-1 0:0:0开始计时，中间时刻如果系统时间被用户改成其他，则对应的时间相应改变

CLOCK\_MONOTONIC:从系统启动这一刻起开始计时，不受系统时间被用户改变的影响

CLOCK\_PROCESS\_CPUTIME\_ID:本进程到当前代码系统CPU花费的时间

CLOCK\_THREAD\_CPUTIME\_ID:本线程到当前代码系统CPU花费的时间

struct timespec

{

time\_t tv\_sec; /\* 秒\*/

long tv\_nsec; /\* 纳秒\*/

};

* Perf性能分析工具

Perf是内置于Linux内核源码树中的性能剖析(profiling)工具。它基于事件采样原理，以性能事件为基础，支持针对处理器相关性能指标与操作系统相关性能指标的性能剖析。常用于性能瓶颈的查找与热点代码的定位。

从Linux Kernel2.6.31版本开始，Linux内核开始提供\_\_NR\_perf\_event\_open的系统调用。使用这个系统调用我们可以像使用文件一样打开一个Performance counter，通过设置不同的参数让这个Performance Counter统计不同的软件或硬件事件，然后就可以向读文件一样来读取这些事件的统计结果。

static struct perf\_event\_attr attr;

attr.type = PERF\_TYPE\_HARDWARE;

attr.config = PERF\_COUNT\_HW\_CPU\_CYCLES;

fd = syscall(\_\_NR\_perf\_event\_open， &attr， 0， -1， -1， 0);

PERF\_COUNT\_HW\_CPU\_CYCLES：统计CPU周期数

* 线程计时模拟器

如果没有足够精确的计时接口可用，攻击者可以通过运行一个循环自增一个全局变量的线程来得到CPU周期的一个倍数估计值。因为一次循环自增操作可看做是由固定次数个时间周期组成的，所测时间的整数倍即为所对应的时钟周期，因此能够将cache hits和cache misses区分开来，从而也可以用于cache攻击。

### 驱逐策略

为了将地址从cache中驱逐到主存中，在Intel x86平台可以使用非特权的clflush指令。虽然ARM平台也提供了类似的cache操作工具，但在非特权模式下不允许使用。驱逐的第二种方式为相关地址的访问，主要原理为读取大量的能够映射到制定cache set的地址，以此来将该cache set中之前存储的数据替换到主存中。虽然读取大量的地址能够大概率的保证将关联set中的数据都驱逐出内存，但大量的访存操作不仅仅会增加驱逐所花的时间，存储相关地址的内存也会增大，而且由于cache伪随机替换策略的影响，驱逐干净cache set后很难了解cache set中存储哪些相关地址，会对probe阶段的探测结果产生影响，从而影响攻击结果。

除此之外，还需将L1 cache中相关set中的数据也驱逐到内存中。因此找到快速且可靠的驱逐方式是至关重要的。驱逐是否成功可以通过探测待驱逐的地址是否仍在cache中来判断。

Gruss et al. 发现了有三个因素对驱逐的成功率有影响，并将其作为可调整的驱逐策略参数：

1. 只有在能够映射到同一个cache set中的地址的cache hits和cache misses会对驱逐的成功率有非负的影响。这可以通过在相关地址中添加能够映射到其他cache set的地址来验证，并可以发现随机的非关联地址不会对平均的成功率产生影响。因此驱逐策略的有效性依赖于驱逐set的大小。
2. 此外，对于cache来说地址是不可区分的，因此访问模式被定义为一个地址系列，比如，其中每个标号代表一个不同的地址，这个系列定义了地址访问的一个先后次序，先访问等等。如果这个模式定义在一个循环中，则每个循环中访问的不同地址数会对驱逐策略的有效性产生影响。
3. ARM平台的cache替换策略倾向于驱逐最近添加到cache line中的数据，因此需要重复的访问相同的地址来保证地址被保存在cache中。比如，将驱逐序列从到缩短了超过33%的执行时间，并且增加了驱逐率。此外，在一定次数的重复之外，再增加访问次数不会增加驱逐率，或许还会更差。

基于这些观察，Gruss et al. 定义了依赖于cache以及cache的替换策略的三个可调整参数的驱逐模式，以供不同设备的调整以得到最佳的驱逐策略。

驱逐算法：

参数：待驱逐set m的N个相关地址

for i = 0; i < N – D; i++ do

for j = 0; j < A; j++ do

for k = 0; k < D; k++ do

Access(i + k)访问第i + k个相关地址

end for

end for

end for

N表示待驱逐set中可以存放的不同地址数

D表示每个循环访问的不同地址数

A表示每个循环每个地址的访问次数

因此，为了保证驱逐的成功率，需要对具体的设备做大量的实验以得到快速有效的驱逐模式。再此上才能保证行而有效的cache攻击。

### 攻击方式

在能够获取精确的系统时间来对驱逐操作计时以及快速高效可靠的驱逐方式的基础上，还需考虑攻击程序对待攻击程序的攻击交互方式。之前的一些实验结果是在攻击程序触发待攻击事件的条件下测量的，比如，攻击程序首先驱逐一个cache set，随后触发待攻击事件，最后再探测是否还占有cache set中的cache line来判断待攻击事件有没有占用到该cache set。在此情况下，通常能得到理想的实验数据，因为由一个程序触发攻击和待攻击动作，在攻击过程中保证攻击和待攻击程序在同一个核上运行（实际上是同一个程序），并且减少程序切换对随后需要访问的已占用set中cache line的部分甚至全部的驱逐。因此，在实际的攻击过程中需要考虑到系统切换和人为因素对cache set状态造成的影响。

在使用prime + probe操作对某使用AES加密算法实现的程序进行攻击的过程中，为了避免系统切换或人为因素对cache set状态造成的影响。由于攻击程序与加密程序不能合并在一个程序中，并且系统切换造成的影响无法消除（在攻击程序与加密程序切换执行的过程中一定需要操作系统的调度），在对每个set进行prime + probe操作测量访存时间时都引入系统调度这一因素，并将系统调度假定为一个常量，则所获取的时间为probe所测时间+系统调度导致的时间增量。由于每次set操作都引入了这一时间增量，cache hits和cache misses仍能通过threshold时间增量区分开来。

至于人为触发prime或probe操作对结果造成影响这一因素可以通过程序或脚本替代手工操作来解决。

### keylogger攻击

通过cache攻击能够在Android平台上获取使用共享库的用户输入信息，例如用户敲击键盘的输入。对用户键盘攻击时，首先剖析用户在输入特定按键时共享库地址的cache hits和cache misses的情况，并统计出一个cache攻击模板矩阵。每个按键均对应着一个矩阵，因此若矩阵可分，则可通过这些矩阵将用户输入区分开。在探测阶段则通过用户输入记录的cache hits和cache misses的矩阵推测出用户的输入。

为了执行对用户按键输入的cache模板攻击，攻击者需将按键相关的共享库或共享二进制文件映射到其地址空间中。通过使用共享库，攻击者能够绕过操作系统对其他app运行时数据访问的限制，并且能够在非特权模式的情况下展开攻击。

### 高级加密AES攻击

AES是美国国家标准技术研究所NIST旨在取代DES的21世纪的加密标准。AES加密数据块分组长度必须为128比特，密钥长度可以是128比特、192比特、256比特中的任意一个（如果数据块及密钥长度不足时，会补齐）。AES加密有很多轮的重复和变换。大致步骤如下：1、密钥扩展（KeyExpansion），2、初始轮（Initial Round），3、重复轮（Rounds），每一轮又包括：SubBytes、ShiftRows、MixColumns、AddRoundKey，4、最终轮（Final Round），但最终轮没有MixColumns。

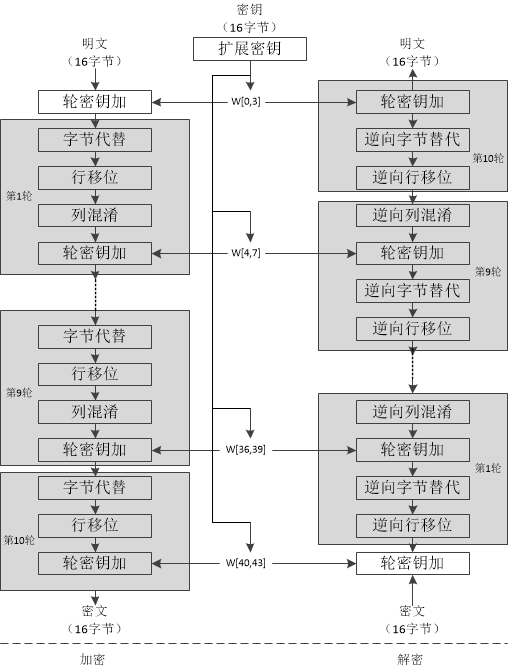


图1 AES加密步骤

Flush + Reload， Evict + Reload以及Prime + Probe可以用来对T-table实现的AES加密算法进行攻击。Moritz Lipp et al. [24] 阐述了跨核同步T-table AES的可能性，及由攻击者触发加密算法的执行。但异步的跨核攻击更具通用型，由于异步执行会引入不受控的内存访问，从而污染cache状态，因此实现的难度教同步cache攻击要大。本题旨在实现针对T-table AES的跨核的异步cache攻击的实现。

### 可执行环境TEE攻击

可信执行环境TEE（Trusted Execution Environment）是移动设备（包含智能手机、平板电脑、机顶盒、智能电视等）处理器上的一个安全区域，其可以保证加载到该环境内部的代码和数据的安全性、机密性以及完整性。TEE提供一个隔离的执行环境，提供的安全特征包含：隔离执行、可信应用的完整性、可信数据的机密性、安全存储等。总体来说，TEE提供的执行空间比常见的移动操作系统（如IOS、Android等）提供更高级别的安全性；比安全元素SE（Secure Element，如智能卡、SIM卡等）提供更多的功能。

移动设备上，TEE环境与移动OS并行存在，为丰富的移动OS环境提供安全功能。运行在TEE的应用称为可信应用（即TA，Trusted Apps），其可以访问设备主处理器和内存的全部功能，硬件隔离技术保护其不受安装在主操作系统环境的用户Apps影响。而TEE内部的软件和密码隔离技术可以保护每个TA不相互影响，这样可以为多个不同的服务提供商同时使用，而不影响安全性。

由于安全监控只能在监管上下文中调用，内核提供和用户空间与可执行环境的接口，比如，Alcatel One Touch Pop 2的可执行环境可以通过设备驱动调用QSEECOM或libQSEEComAPI.so来进行交互。可信执行环境的密钥管理组件提供一个生成基于硬件的RSA密钥，通过它能够在可信执行环境内签署或改变签署信息。Moritz Lipp et al. [24] 展示了Prime + Probe可以被用来区分一个给定的密钥是否是有效的。本题在时间充裕的情况下将探索对可信执行环境的cache攻击获取更多的信息。

# 关键技术及难点

## 确定驱逐策略

驱逐策略关系着cache set能够快速高效的驱逐，不仅仅要将L2 cache相关set中的所有line中的数据驱逐，也要讲L1 cache中的相关数据驱逐到内存中。然而由于大多数ARM处理器使用的L2 cache伪随机替换策略，要在连续的访存中估计cache set中的数据显得困难。虽然在不支持flush刷新的情况下可以使用相关地址来驱逐set，通过大量访问相关地址达到驱逐目的地址到内存的目的，但这不仅会增加时间和空间存储的开销，也会引入探测阶段的偏差（由于不清楚是哪些地址占用了set）。由此在确定机型上找到快速高效的驱逐策略是难点之一。

## 探索异步攻击的方式

Cache攻击的攻击程序与被攻击程序的交互方式是攻击能否成功的关键，然而在这个方向目前还没有太多相关的研究。Evict + Reload、Prime + Probe等方法只是探测cache set状态的工具，还需确定具体的攻击过程。由于在evict或prime过后需要被攻击程序执行待攻击的事件，之后再有reload或probe操作来获取cache的旁路信息，然而在这期间由于系统访存或攻击程序自身访存操作可能会对evict或prime阶段占用的cache set中的部分甚至全部line，导致探测阶段误认为待攻击程序访问了该cache set，因此如何处理系统噪音造成的影响也是攻击成功的关键。

## 探索对AES进行攻击的方式

由于AES加密过程涉及到4种操作：字节替代、行移位、列混淆和轮密钥加。加解密中每轮的密钥分别由初始密钥扩展得到。算法中16字节的明文、密文和轮密钥都以一个4\*4的矩阵表示。在对AES的T-table实现进行密钥攻击时，需要熟悉攻击过程以及每个过程中对那个table进行查找，并挖掘出可利用的内存位置，提取出AES可供利用的数学特性，这些均为AES攻击中的难点。

## 探索对可信执行环境TEE的攻击方式

由于TrustZone技术的主要方法是将设备的硬件和软件资源全部划分成安全区域和非安全区域，两个区域之间不能随意进行数据交换，非安全区域的进程禁止访问安全区域，以保证存储在安全区域的资源不被窃取。在硬件层面上，TrustZone 将物理处理器核虚拟成两个核，一个称为非安全核（Non-secure，NS），另一个称为安全核（Secure，S），通过CP15 的SCR 寄存器中的NS 位来表明当前所处的状态。非安全核只能访问自己的系统资源，安全核可以访问所有资源。在安全核和非安全核之间切换的机制称为监视器模式。非安全核的进程可以通过SMC 指令或者硬件异常机制进入监视器模式，由此获得安全核的服务。非安全核的进程只能获得安全核的服务，并不能访问安全核的数据。此外，TrustZone 技术在虚拟内存管理、Cache、外围总线等方面都做了努力来保证安全核的数据信息不被泄露。因此在可执行环境下探索通过cache攻击的方式是本题的难点。

# 前期工作

## 通过POSIX实现精确计时

"clock\_gettime"是基于Linux C语言的时间函数，他可以精确到纳秒。

#include<time.h>

int clock\_gettime(clockid\_t clk\_id， struct timespec \*tp);

clk\_id : 检索和设置的clk\_id指定的时钟时间。

CLOCK\_REALTIME:系统实时时间，随系统实时时间改变而改变，即从UTC1970-1-1 0:0:0开始计时，中间时刻如果系统时间被用户改成其他，则对应的时间相应改变

CLOCK\_MONOTONIC:从系统启动这一刻起开始计时，不受系统时间被用户改变的影响

CLOCK\_PROCESS\_CPUTIME\_ID:本进程到当前代码系统CPU花费的时间

CLOCK\_THREAD\_CPUTIME\_ID:本线程到当前代码系统CPU花费的时间

struct timespec

{

time\_t tv\_sec; /\* 秒\*/

long tv\_nsec; /\* 纳秒\*/

};

## 实现evict + reload和prime + probe

为了确定访存与访问cache的区别，并求出访存与访问cache的时间阈值。设计了这组实验，实验过程分为两组，第一组实验测量cache命中的时间，测量方法如下：

* 测量访cache时间：

access\_memory(address);//将address中的数据加载到cache中

uint64\_t time = get\_timing\_start()//获取当前系统时间

access\_memory(address);//访问address

uint64\_t delta = get\_timing\_end(session) - time;//计算访存时间

* 测量访内存时间：

evict(address);//将地址所在cache中的数据驱逐出去

uint64\_t time = get\_timing\_start()//获取当前系统时间

access\_memory(address);//访问address

uint64\_t delta = get\_timing\_end(session) - time;//计算访存时间

为了避免偶然因素对实验结果造成的影响，通过增加测量次数的方式来减少误差，分别对访cache和访内存进行50000次实验，并分别将其众数作为所测时间。实验结果如下：

Cache access time: 765ns

Memory access time: 1000ns

Threshhold: 883ns

## 访存操作对probe时间的影响

本节通过实验探究实际应用攻击场景下set的cache line miss对probe操作时间的影响。实验目标为Lenovo K51c78（ARM v8-a处理器， cache共有512个set，每个set为16路，每路大小为64B），被攻击程序为一个简单app，点击该app上的ACCESSSET按钮将执行针对相关set的访存操作，该程序的截图如2所示。



图2 被攻击程序界面

为了实现异步攻击，攻击程序为另一个app，该app基于POSIX计时方式，使用prime + probe攻击方式的，攻击程序的界面如图3所示。

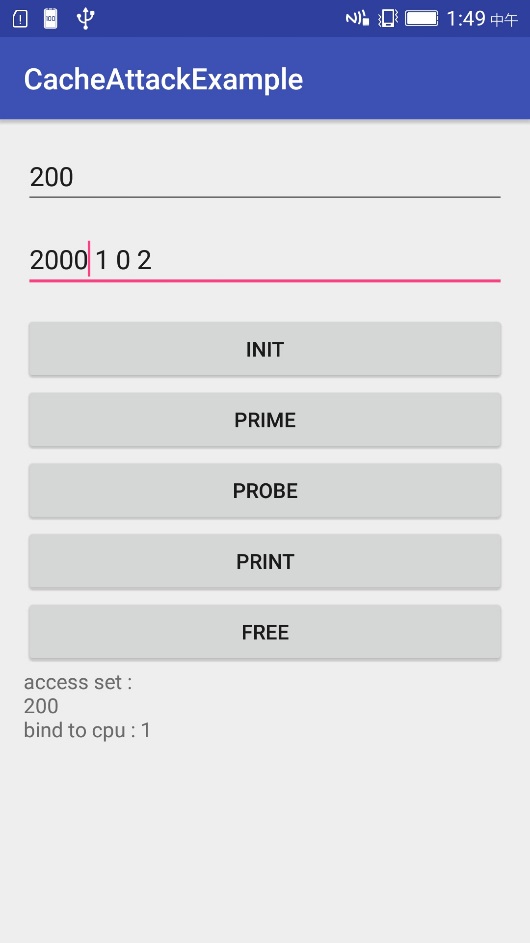


图3 攻击程序界面

点击prime操作后将对第一个输入栏中的set进行evict操作，点击probe操作将探测prime阶段读入cache set中的数据是否还在该set中。

具体的实验过程如下：

1. 设置cachesetaccess程序访问的set号，本实验设置为200，及点击accessset按钮后该app将通过一条访存语句占用该set的一个cache line。
2. 设置攻击程序的配置信息，在第一个edittext文本框中输入待探测的set号，第二个edittext文本框中输入重复探测的次数以及访问模式等信息。
3. 点击init按钮，攻击程序找到待探测set的相关地址并将其存储到数组中
4. 点击prime按钮，执行将制定set的eviction操作，将相关地址中的数据读入该set中
5. 切换到cachesetaccess程序，点击accessset按钮执行访存操作
6. 切换回攻击程序后点击probe按钮并返回probe时间
7. 重复过程4到6，知道重复次数达到所设置的值
8. 重复一次整个过程，不过这次切换到cachesetaccess后，不点击accessset按钮
9. 记录下实验结果并分析

由于需要排除系统切换或攻击程序本身访存操作对prime阶段占用的cache set的影响。因此需要每次对尽可能少的cache set做prime probe操作，并对多个set进行实验。本实验仅仅对单个set进行prime probe操作，set号为150，并将有访存操作的probe时间与没有访存操作的probe时间进行对比。实验结果如图4所示。

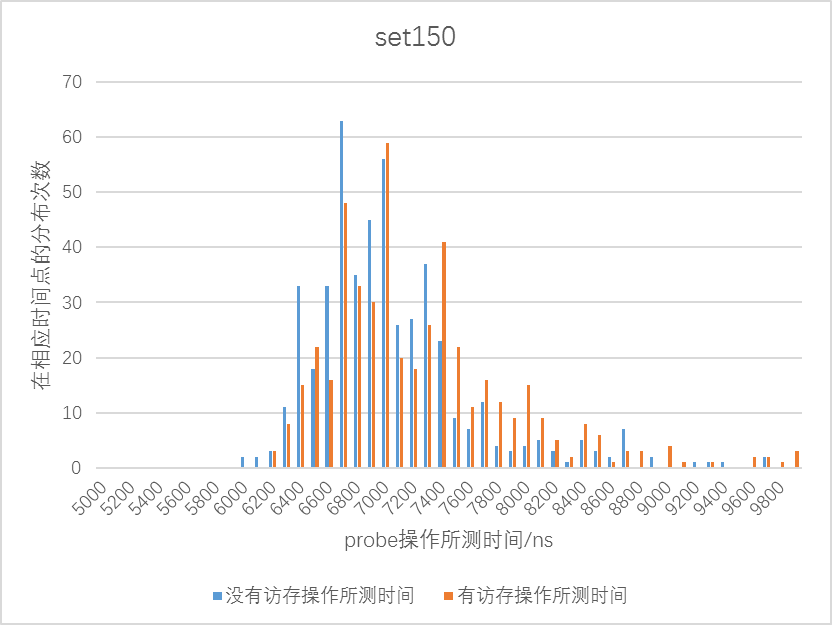


图4 set150实验数据

其中横坐标表示probe操作所花时间，单位为纳秒。纵坐标为probe操作所得时间在横坐标相应的时间点上统计的次数。其中蓝色条块表示没有执行访存操作所得结果，红色条块为相对的执行访存操作所测结果。根据图5.3.1可看出，红色条块的分布情况为蓝色条块分布向右偏移300ns的结果，及访存操作增加了probe操作的时间，且该时间大致等于一次cache miss（1000ns）与cache hit（765ns）的时间差。因此验证了当前实现的prime + probe攻击方式能够有效的被用于实现异步的cache 攻击。

## Prime + probe异步aes攻击实验

为验证prime + probe能够探测到AES操作所占用的cache set，本实验使用同上节实验相同的目标机和攻击程序，异步的对一个能够执行aes加密的app进行攻击实验。该app的界面截图如图5所示。

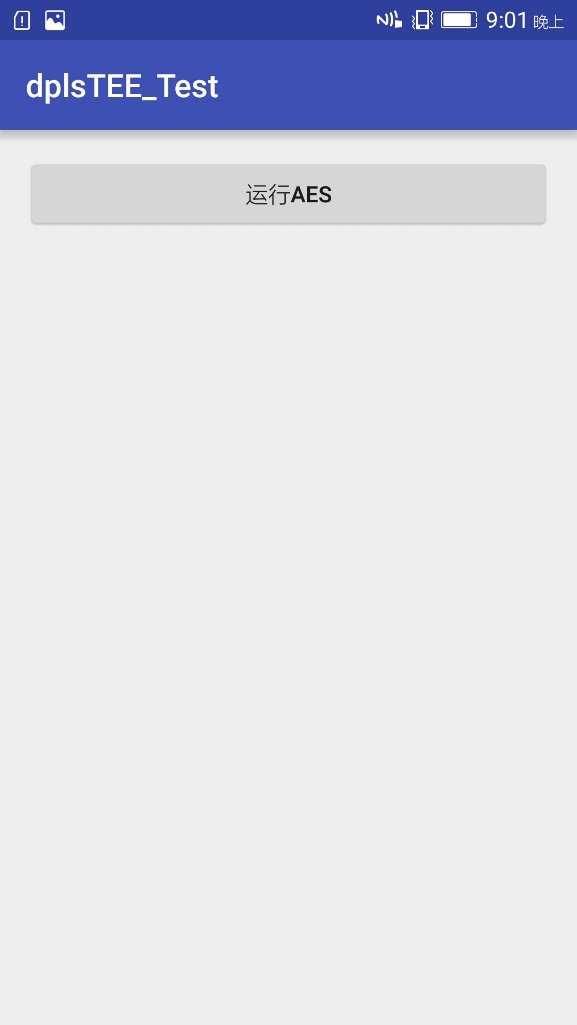


图5 AES程序

为了检测prime + probe攻击的有效性，针对多个set进行试验，通过在AES的查表过程中进行插桩，打印出执行AES加密操作时访问的cache set号，将set分为两类，一类为可能未占用set，表示AES操作没有显示说明占用的set，另一类为占用set，表示AES操作中通过插桩打印出执行过程中占用了的set。其中可能未占用set又可分为两类，未占用set和隐式占用set，因为在AES操作中仅仅打印出部分访问的set号，且AES或系统切换的访存操作访问的set并未打印出来。实验结果3个图如下所示，图6表示显示占用的set，图7和图8均表示可能未占用set。

图6 显示占用set

图7 可能未占用set

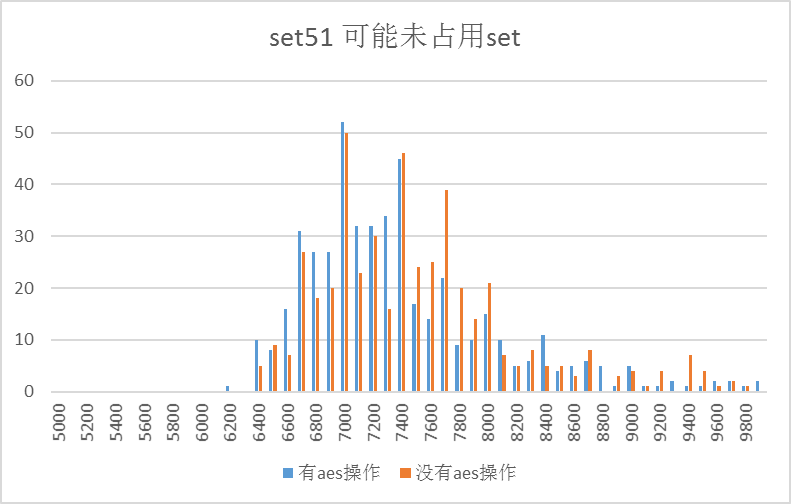


图8 可能未占用set

其中横坐标表示probe操作所花时间，单位为纳秒。纵坐标为probe操作所得时间在横坐标相应的时间点上统计的次数。其中蓝色条块表示没有执行访存操作所得结果，红色条块为相对的执行访存操作所测结果。通过图6可看出，prime + probe检测到了aes中的访存操作对cache造成的影响，其实验结果相较于没有aes操作的结果向后偏移了一个cache hit和cache miss的时间差。而图7中的有aes操作和没有aes操作的统计结果类似，表示在执行AES操作的过程中没有访问到该set，因此probe计时相同。图8中操作的set虽然为可能未访问的set，但有AES操作的结果相对于没有AES的统计结果有一个向右的偏移。出现这种情况是因为本实验并没有将执行AES中的所有访存操作访问的set都打印出来，而该set应该属于执行AES操作访问到的其他相关set。综述可得实现的prime + probe能够有效的对AES T-table进行异步攻击。

# 论文研究计划

表 1 研究计划表

|  |  |
| --- | --- |
| 规划 | 安排 |
| 2016年12月~2017年 2月 | 调研驱逐策略，并找到快速针对目标机有效的cache驱逐策略 |
| 2017年 3月~2017年 5月 | 调研相关技术并实现对keylogger的异步攻击 |
| 2017年 6月~2017年 8月 | 调研相关技术并实验对AES加密技术的异步攻击 |
| 2017年 8月~2017年12月 | 调研可信执行环境相关技术，并实现对TEE的攻击探索 |
| 2018年 1月~2018年 2月 | 整理资料并准备毕业答辩 |

# 主要参考文献

1. Gruss D, Maurice C, Mangard S. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript[M]// Detection of Intrusions and Malware, and Vulnerability Assessment. 2016.
2. Yarom Y, Falkner K. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack[C]// USENIX Security Symposium. 2014:719-732.
3. Irazoqui G, Eisenbarth T, Sunar B. Cross Processor Cache Attacks[C]// The, ACM. 2016:353-364.
4. Bernstein D J. Cache-timing attacks on AES[J]. Vlsi Design IEEE Computer Society, 2005, 51(2):218 - 221.
5. Kelsey J, Schneier B, Wagner D, et al. Side channel cryptanalysis of product ciphers[J]. Lecture Notes in Computer Science, 1998, 8(23):97-110.
6. Kocher P C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems[M]// Advances in Cryptology — CRYPTO ’96. Springer Berlin Heidelberg, 1999:104--113.
7. Neve M. Cache-based Vulnerabilities and SPAM analysis[J]. Doctor Thesis Ucl, 2006.
8. Neve M, Seifert J P, Wang Z. A refined look at Bernstein's AES side-channel analysis[C]// ACM Symposium on Information, Computer and Communications Security. ACM, 2006:369-369.
9. Dan P. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel.[J]. Journal of Arid Environments, 2002, 2002(10):393-446.
10. Tsunoo Y, Saito T, Suzaki T, et al. Cryptanalysis of DES Implemented on Computers with Cache.[J]. Proc of Ches Springer Lncs, 2003, 2779:62-76.
11. Irazoqui G, Eisenbarth T, Sunar B. S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing -- and Its Application to AES[J]. 2015:591-604.
12. Liu F, Yarom Y, Ge Q, et al. Last-Level Cache Side-Channel Attacks are Practical[J]. IEEE Symposium on Security & Privacy, 2015:605-622.
13. Osvik D A, Shamir A, Tromer E. Cache Attacks and Countermeasures: The Case of AES[J]. Lecture Notes in Computer Science, 2005, 2005:1-20.
14. Tromer E, Osvik D A, Shamir A. Efficient Cache Attacks on AES, and Countermeasures[J]. Journal of Cryptology, 2010, 23(1):37-71.
15. Maurice C, Neumann C, Heen O, et al. C5: Cross-Cores Cache Covert Channel[M]// Detection of Intrusions and Malware, and Vulnerability Assessment. 2015:46-64.
16. Kocher P C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems[C]// International Cryptology Conference on Advances in Cryptology. Springer-Verlag, 1996:104--113.
17. Gullasch D, Bangerter E, Krenn S. Cache Games -- Bringing Access-Based Cache Attacks on AES to Practice[J]. 2011, 2010(1):490-505.
18. Gülmezoğlu B, İnci M S, Irazoqui G, et al. A Faster and More Realistic Flush+Reload, Attack on AES[J]. 2015.
19. Irazoqui G, Inci M S, Eisenbarth T, et al. Know Thy Neighbor: Crypto Library Detection in Cloud[J]. Proceedings on Privacy Enhancing Technologies, 2015, 1(1):25-40.
20. Irazoqui G, Inci M S, Eisenbarth T, et al. Lucky 13 Strikes Back[C]// ACM Symposium on Information, Computer and Communications Security. ACM, 2015.
21. Gruss D, Spreitzer R, Mangard S. Cache template attacks: automating attacks on inclusive last-level caches[C]// Usenix Conference on Security Symposium. USENIX Association, 2015:897-912.
22. Oren Y, Kemerlis V P, Sethumadhavan S, et al. The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications[J]. Computer Science, 2015.
23. Kelsey J, Schneier B, Wagner D, et al. Side Channel Cryptanalysis of Product Ciphers.[C]// Computer Security - ESORICS 98, European Symposium on Research in Computer Security, Louvain-La-Neuve, Belgium, September 16-18, 1998, Proceedings. 1998:97--110.
24. Lipp M, Gruss D, Spreitzer R, et al. ARMageddon: Cache Attacks on Mobile Devices[J]. Mundo Electrónico, 2016, 6(1):págs. 60-65.
25. Gruss D, Maurice C, Wagner K, et al. Flush+Flush: A Fast and Stealthy Cache Attack[M]// Detection of Intrusions and Malware, and Vulnerability Assessment. Springer International Publishing, 2016.
26. Tsunoo Y, Saito T, Suzaki T, et al. Cryptanalysis of DES Implemented on Computers with Cache.[J]. Proc of Ches Springer Lncs, 2003, 2779:62-76.
27. Weiß M, Heinz B, Stumpf F. A Cache Timing Attack on AES in Virtualization Environments[M]// Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2012:314-328.
28. Bernstein D J. Cache-timing attacks on AES[J]. Vlsi Design IEEE Computer Society, 2005, 51(2):218 - 221.
29. OpenSSL the open-source toolkit for SSL/TLS. 2005. <http://www.openssl.org/>
30. Bonneau J, Mironov I. Cache-collision timing attacks against AES[J]. 2006, 4249:201-215.
31. Spreitzer R, Plos T. On the Applicability of Time-Driven Cache Attacks on Mobile Devices[C]// Network and System Security. 2013:656-662.
32. Bogdanov A, Eisenbarth T, Paar C, et al. Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs[M]// Topics in Cryptology - CT-RSA 2010. Springer Berlin Heidelberg, 2010.
33. Gallais J F, Kizhvatov I, Tunstall M. Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations[C]// Information Security Applications -, International Workshop, Wisa 2010, Jeju Island, Korea, August 24-26, 2010, Revised Selected Papers. 2010:243-257.
34. Gallais J F, Kizhvatov I. Error-Tolerance in Trace-Driven Cache Collision Attacks[J]. Cosade, 2011:222--232.
35. Spreitzer R, Plos T. Cache-Access Pattern Attack on Disaligned AES T-Tables[C]// Constructive Side-Channel Analysis and Secure Design. 2013:200-214.
36. Percival C. Cache missing for fun and profit[J]. Proc of Bsdcan, 2005.
37. Osvik D A, Shamir A, Tromer E. Cache Attacks and Countermeasures: The Case of AES[J]. Lecture Notes in Computer Science, 2005, 2005:1-20.
38. Neve M, Seifert J P. Advances on Access-Driven Cache Attacks on AES[C]// Selected Areas in Cryptography, International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers. 2006:147-162.
39. Aciicmez O. Advances in side-channel cryptanalysis : microarchitectural attacks[J]. 2006.
40. Acıiçmez O, Schindler W, Çetin K. Koç. Cache Based Remote Timing Attack on the AES[M]// Topics in Cryptology – CT-RSA 2007. Springer Berlin Heidelberg, 2007:271-286.
41. Dan P. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel.[J]. Journal of Arid Environments, 2002, 2002(10):393-446.