**Research on AES Cache Attack Technology Based on ARM Processor \***

**Bo Li1**

1School of Computer and Engineering, Beihang University, Bejing, China

Email: leborn@buaa.edu.cn

Received: \*\*\*\*\*\*

Copyright © 2018 by author(s) and Hans Publishers Inc.

This work is licensed under the Creative Commons Attribution International License (CC BY).

<http://creativecommons.org/licenses/by/4.0/>

 C:\Users\178\Desktop\图片1.emf

1-1

**Abstract**

**Cache attack is a powerful attack tool that can access the user's private information based on the memory access mode revealed by the Cache, such as the user's keyboard input, encryption keys, etc. On Intel x86 platform, there have been Cache attack implementations aiming at AES, DES encryption algorithm, but on the Android platform, the structure of the Cache, instruction Set and Cache replacement strategy have a lot of differences from which of Intel x86, so Cache attack on mobile devices is difficult. This paper reduces the impact of random error of the experimental results on the Android platform by introducing hypothesis testing, eventually get all AES key bytes. Then this paper explores the asynchronous attack mode.**

**Keywords**

**Cache Attack; Aes Attack; Aes; Cache**

1-1

**基于ARM处理器的AES缓存攻击技术研究\***



**李 勃1**

1北京航空航天大学，北京，中国

Email: leborn@buaa.edu.cn

收稿日期: \*\*\*\*\*\*

1-1

**摘 要**

**Cache攻击是一种强大的攻击工具，能够根据Cache泄露的内存访问模式获取用户的私密信息，比如用户的键盘输入、加密的密钥等。在Intel x86平台上，已经有针对AES、DES加密算法的Cache攻击实现，但是在Android平台上，由于Cache结构、指令集、Cache替换策略等与Intel x86有很多差别，因此攻击难度较大。本文基于Android平台，以AES加密算法作为攻击对象，通过引入假设检验，降低随机误差对实验结果的影响，最终获取AES全部密钥。最后对AES异步攻击方式进行探索。**

**关键词**

**缓存攻击；AES攻击；AES；Cache；**

1-1

**1. 相关背景**

伴随着移动互联网突飞猛进式的增长，手机等移动设备已经成为我们日常生活中不可或缺的一部分。它在给我们的生活带来便利的同时也对我们的隐私带来了潜在的威胁，尤其是手机存储着包含账号、密码等用户私密信息的现在。然而，由于系统硬件的一些固有缺陷，一些针对计算机以及移动设备的攻击方式逐渐显现出来，其中就包括本文所研究的Cache攻击。其攻击方式为基于Cache hit与Cache miss的时间差异来探测被攻击程序执行过程中的内存以及Cache的访问情况，并依此获取被攻击程序私密信息。

起初，Intel x86平台上的Cache攻击得到了科研工作者的重视，并有很多学者从事这一方向的研究。Kocher[1]和Kelsey[2]提出了通过分析高速缓冲存储器Cache在运行时泄露出的信息来破解计算机加密算法的方法。基于这些思想，近些年不断的有人提出在Intel x86上进行Cache攻击的方法，并成功的实现了监听用户输入、破解加密算法等攻击，这也反向证明了Cache攻击是有效的。比如，Page et al.[3]提出了一种破解DES加密技术的Cache攻击模型，并将DES密钥的搜索空间从56位降低到了32位。

国内针对Cache攻击的研究起步较晚，Wu WL[4]对MARS和Rijindael进行了详细的分析。 Zhou YB[5]对进10年内的Cache旁路攻击进行了系统性的分析与介绍，之后Hou FY[6]对最新的针对AES加密算法的Cache攻击技术，分析了其用于攻击的可行性，并给出了预防措施。Deng GM[7]参考了Bonneau[8]的攻击思路，在Intel Celeron和Pentium环境下成功恢复了AES密钥，后来Li B[9]基于Bernstein[10]提出的思想在Pentium平台上针对OpenSSL进行了相关的改进实验。

然而，由于Android等移动设备与Intel x86设备有很大的区别，在指令集合、Cache组织方式以及Cache替换策略等与Cache攻击密切相关的结构也有很大的差异。因此，直到最近为止，才有人提出在非root的手机上执行有效的Cache攻击方法。 Moritz Lipp et al.[11]提出了通过在移动设备上进行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，将其作用于Cache攻击的模型即可提取用户的私密信息。

本文研究ARM处理器的移动设备平台上的Cache攻击关键技术，并在Lenovo k51c78平台上实现针对AES高级加密技术的Cache攻击。

**2. 相关技术**

**2.1. 计时方式**

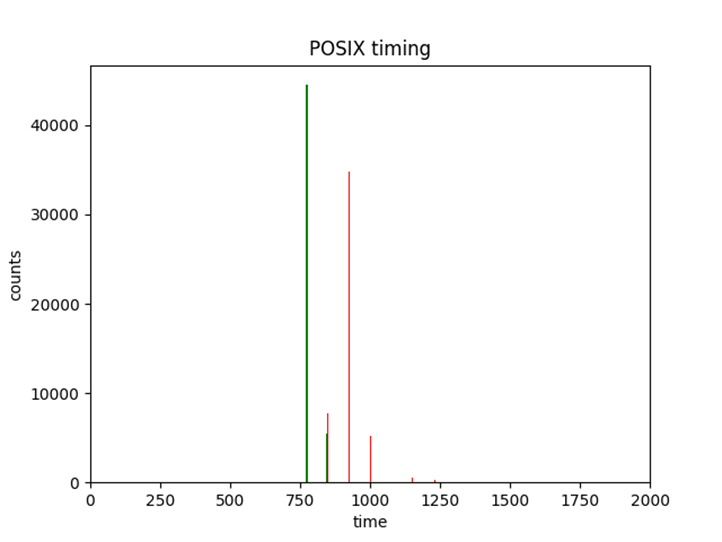
由于Cache攻击是通过监测访存与访缓存之间的时间差异来进行攻击的，因此首先需要能够在目标平台上获取到精确的计时方式。本文采用的是POSIX提供的clock\_gettime计时接口，它能够返回当前系统时间，精确到纳秒，调用方式如**表1**所示：

**Table 1.** POSIX timing

**表1.** POSIX计时

|  |  |
| --- | --- |
|  | #include <time.h> |
|  | int64 get \_time(void) |
|  | { |
|  | struct timespec t; |
|  | clock\_gettime(CLOCK\_MONOTONIC, &t); |
|  | return t.tv\_sec \* 1000\*1000\*1000ULL + t1.tv\_nsec; |
|  | } |

get\_time方法能够返回精确的系统时间，**图1**通过POSIX提供的接口测量访存和访问Cache的时间差距。该实验首先通过get\_time函数统计50000次Cache命中（蓝色柱状图）和Cache缺失（红色柱状图）的时间，然后将测量得到的时间统计到一张图中。其中图表的横坐标表示统计时间，单位为纳秒，纵坐标为在单个时间点上统计的次数。



**Figure 1.** Cache hit and miss experiment

**图1.** Cache命中与未命中实验

从上图可以看出，红色圆柱相对于蓝色圆柱有向右的偏移，反映到访问时间则是Cache命中需要时间比Cache缺失的时间短，也反映了POSIX计时方式的有效性。

**2.2. 驱逐策略**

为了获取Android平台上程序的内存使用情况，需要启动另外一个程序，通过不断的探测内存映射到的Cache Set，根据访存时间的多少判断该Set有没有被被攻击程序使用到。探测的方式为：1.攻击程序通过读取大量内存空间中数据将某个Cache Set中存储的数据驱逐到内存中，此时新读取的数据占据了该Cache Set中所有的Cache line；2.被攻击程序运行，其运行期间的访存操作可能会将数据缓存到待探测的Cache Set中；3.攻击程序检测第一阶段缓存到Cache中的数据是否还保存在Cache Set中，其判断依据是访问数据的时间，如果数据还在Cache中，则访问数据将从缓存中获取数据，因此时间较短。

**Table 2.** Evict Algorithm

**表2.** 驱逐策略算法

|  |
| --- |
| 驱逐算法： |
| 输入： |
| N：Cache Set的line数 |
| D：每个循环访问的地址数 |
| A：每个地址的访问次数 |
| 输出：无 |
| 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 |

为了获取高效的Cache驱逐策略，本文依据Gruss et al.[11]提出的**表2**所示的获取驱逐的思想，通过**表2**算法，在Android平台上对可能的驱逐策略组合进行测试，最终选择驱逐效果最好的参数组合进行攻击。

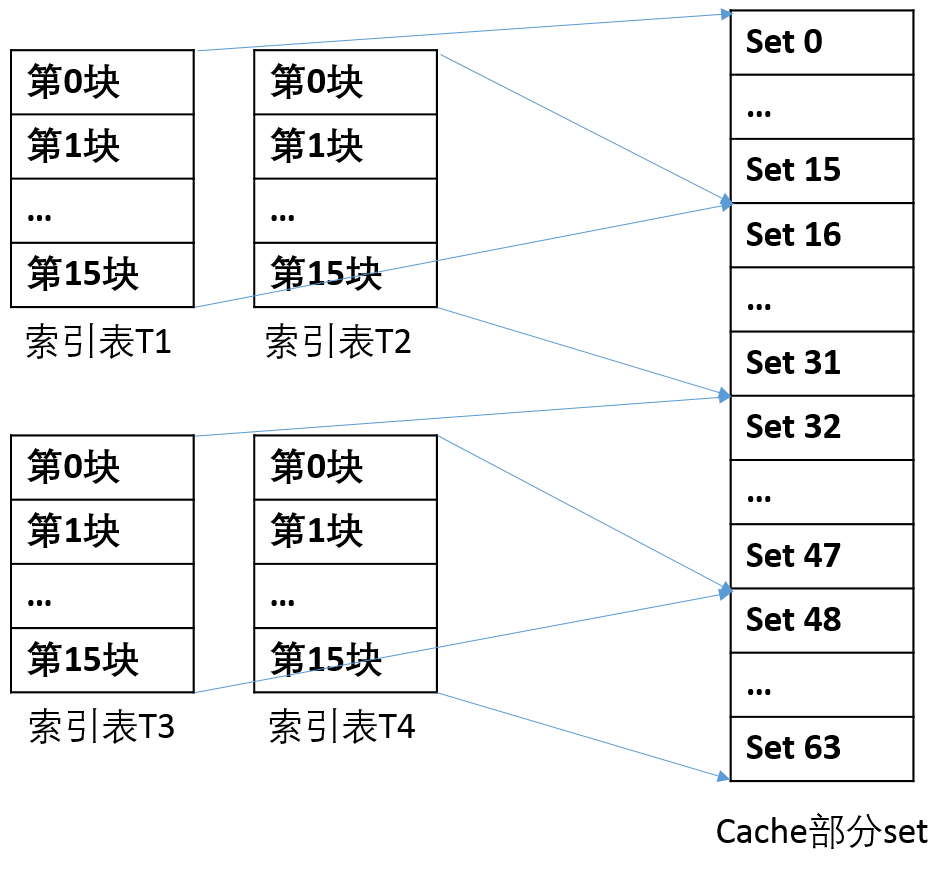
**3. 同步攻击**

**3.1. AES攻击**

AES执行过程中对各个Table的查表操作是对其进行Cache攻击的切入点，对于相同的密钥k，对于不同的明文p进行加密时，根据计算，会访问到各个Table中不同索引位置，如果Cache中没有缓存相应位置的数据时，会将该数据以及其周围的数据加载到其内存所对应的Cache Set的某一个line中，加载的数据块大小与Cache line大小相等。本文针对Lenovo k51c78测试机进行实验，其Cache line大小为64字节，Table中的每项数据为4字节。加密过程中每次访问Table中的数据时，会将其周围共16个数据项加载到同一个Cache line中。因此，如果中起始数据所在内存对应Cache中某一Set a中某一Cache line的起始位置，则中索引前16项的数据都将映射到Set a中，而索引第16至第32项数据将映射到Set a+1中，Table中索引为i的数据将映射到的 Set索引为a+(i/16)。

攻击程序通过对自身内存进行访问，能够获取到其他程序使用Cache的情况。在攻击过程中，通过Moritz Lipp et al.[11]介绍的Prime+Probe攻击方式，其中Prime+Probe为Cache攻击的基本策略之一，其分为两个步骤：Prime阶段通过读取能够映射到某个Set中的数据将该Set中之前的数据驱逐到内存中，Probe阶段则监测Prime阶段读取的数据是否还缓存在Cache中。

因此，攻击程序首先通过Prime方法读取其内存空间的数据占用Cache中指定Set中的所有line，然后调用AES加密程序，待加密程序执行完毕之后，执行Probe方法，根据Probe访问时间判断Prime阶段读取的数据是否还保存在Cache中。若均还在Cache中，则表示AES加密过程中的多轮查表操作没有访问到能够映射到该Set的数据。反之，若Probe阶段探测到之前读取的数据不在Cache中，则表明AES加密过程中的某一轮查表操作或其他访存操作访问到了能够映射到该Set的数据。为了简化攻击操作，假设AES加密过程中除了查表操作外不存在其他访存操作。因此，假设已知AES中各个索引表的起始地址对应的Cache Set索引，当攻击程序同时对Cache中的所有Set进行监控时，就能够监测到AES加密过程中访问了、、、中哪些区域的数据。



**Figure 2.** The map between AES table and Cache

**图2.** AES索引表与Cache映射情况

为了方便讨论，本文将以目标机Lenovo k51c78为例，讨论以其作为被攻击设备，设计针对AES加密算法的攻击的方案。该目标机的L2 Cache大小为512KB，及共有512个Set，Cache组织为16路组相联。因此每个Set共有16个Cache line，每个Cache line能缓存64字节的数据。被攻击的AES加密程序有、、、 4个主要的索引表，其中每个索引表包含256个数据项，可将其视为有256个元素的数据，其中每个数据项为4字节，并且假设已知索引表的首地址a对应的Cache Set索引i，为了方便讨论，假设a的取值满足a%64==0，表示索引table的首地址映射到Cache line的起始位置，效果如**图2**所示。因此，能够求得索引表中任意索引位置n对应的Cache Set位置，其表示当该位置数据被访问时其将被缓存的Cache Set号。由于攻击程序通过Prime+Probe只能探测到被攻击程序访问Cache Set的情况，并不能区分访问了Cache Set中哪些line，更不能获取Cache line中偏移的情况，因此通过Prime+Probe攻击程序并不能区分对索引表T0中索引0到索引15数据项的访问（对0到15项的任意一项的访问都将一整块的数据缓存到了Cache中）。

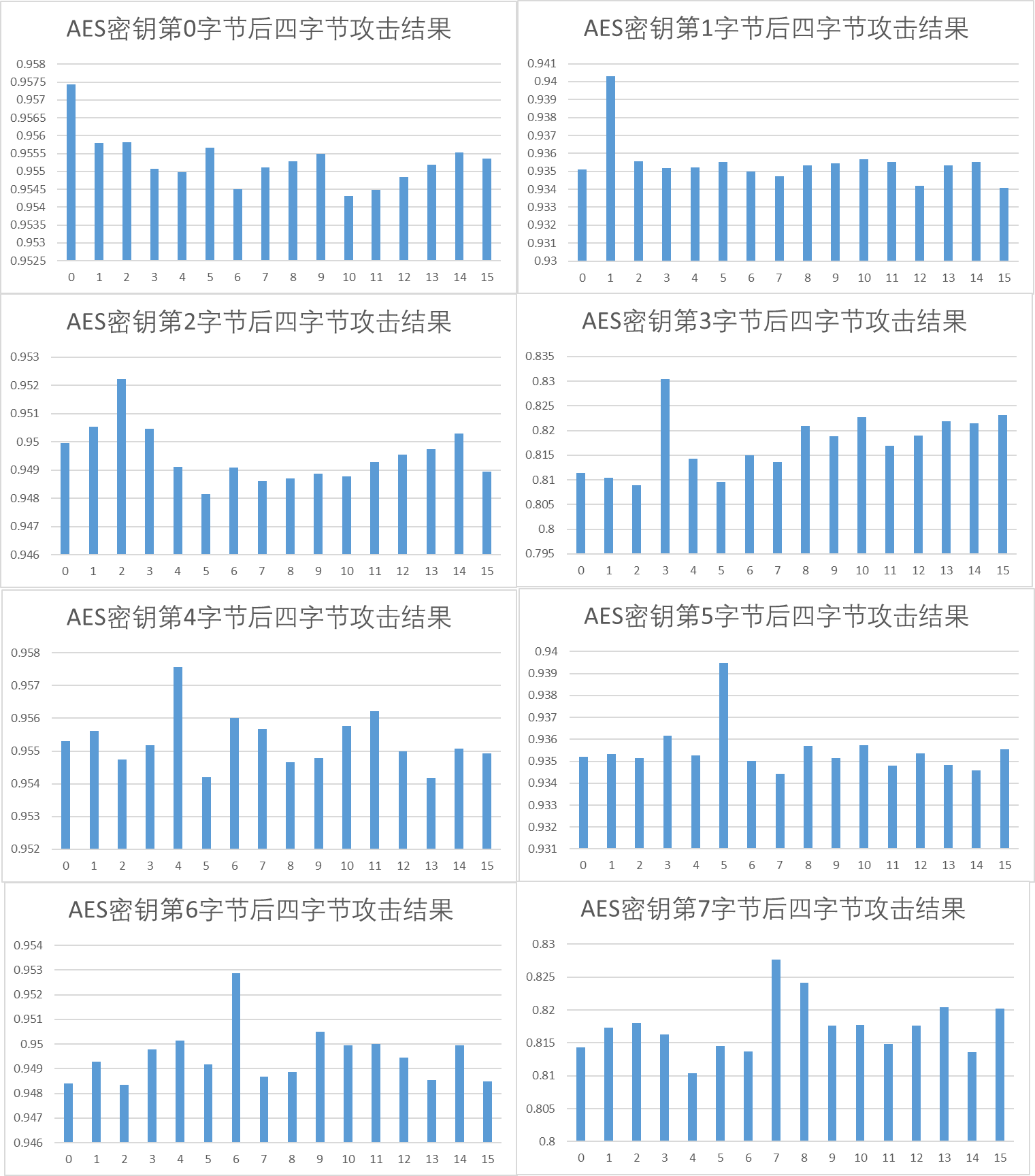
本文设计通过假设验证的方式来实现获取AES密钥，其破解过程分为两轮，第一轮获取密钥中每个字节的前4位，称为第一轮攻击，第二轮获取密钥中每个字节的后四位，称为第二轮攻击。

**3.1.1. 第一轮攻击**

第一轮攻击首先计算出AES加密过程中第一轮的查表索引，结合AES内存空间中各个查表索引Table的首地址，计算得到对AES进行攻击时需要检测的Cache Set。本文的实验中假设AES加密明文为p，密钥为k，并假设、、、的首地址对应的Cache Set号为、、、，因此可以计算得到第一轮需要监测Cache Set号：、、、、、、、、、、、、、、、。

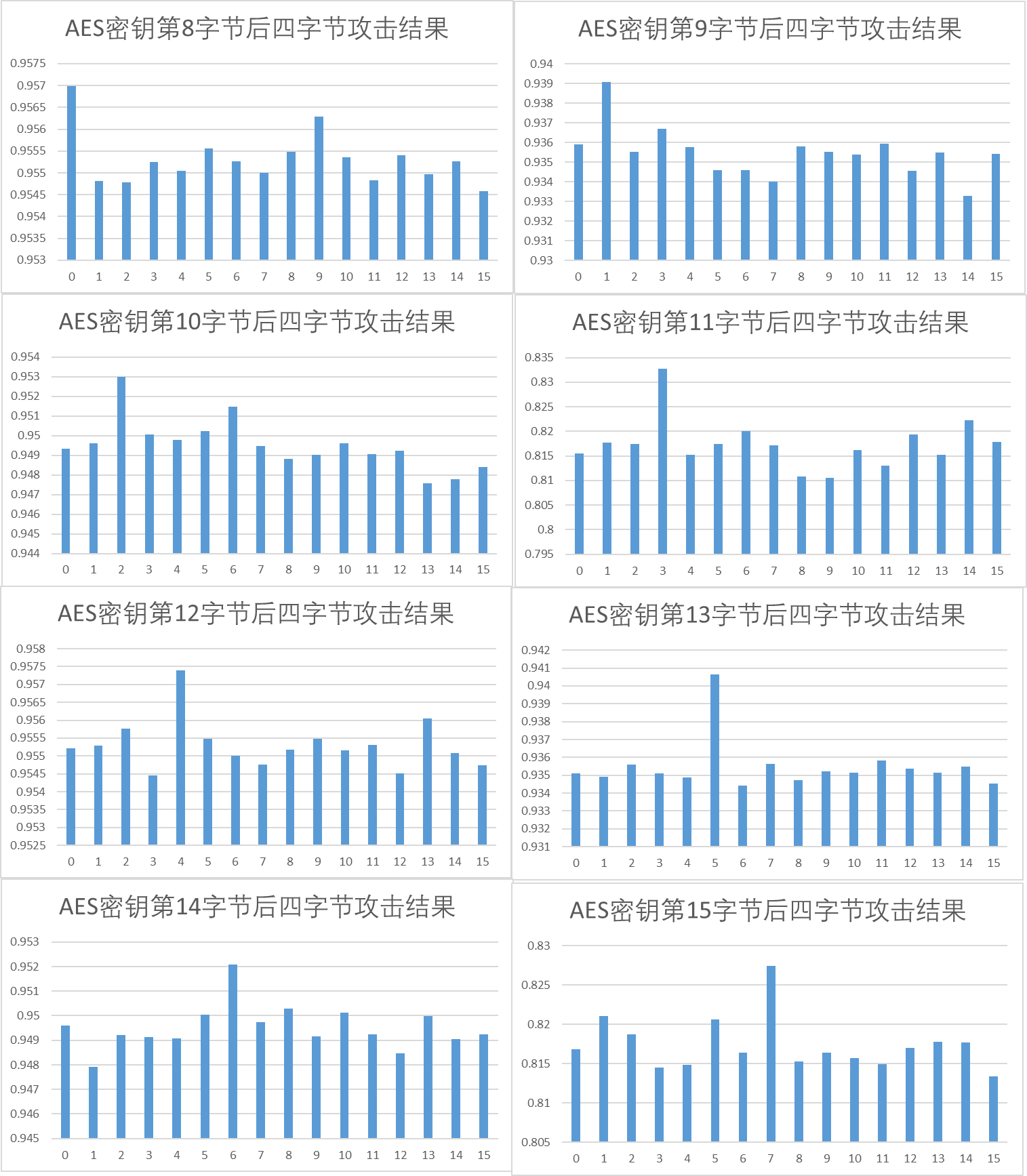
在获得第一轮攻击访问的Cache Set之后，通过攻击程序检测这些Set的访问情况。首先，通过Prime操作占用指定的Set，然后执行AES加密算法，最后通过Probe测量时间判断Prime阶段读入的数据是否还在Cache中。由于单次实验获取的时间作为度量分会映入计时方式导致的误差，因此可以针对每个明文进行多次实验，最后取平局值或通过KS检验与Cache hit情况下的时间进行度量。

本文通过假设检验方式破解用户密钥，首先假设密钥的取值，然后根据密钥的假设值以及明文计算得到一组访问索引，对索引进行计算可以得到对应的Cache Set访问索引，最后通过对这组Set进行监测，判断其在AES算法执行过程中的访问情况。本文在进行第一轮攻击的验证过程中，假设密钥k为k=(0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77)，根据第一轮攻击方式对其进行攻击获取密钥如**图3**、**图4**所示。



**Figure 3.** The result of AES first 8 bytes in first round attack

**图3.** AES第一轮攻击前8字节结果



**Figure 4.** The result of AES last 8 bytes in first round attack

图4. AES第一轮攻击后8字节结果

**图3**、**图4**中从左到右从上到下表示密钥k第0字节到第15字节的攻击结果，图中横坐标表示密钥字节前4位的假设值，纵坐标表示可疑度的高低，其中可疑度越高表示可能性越大，因此，本轮攻击得到的密钥的每个字节的前四位为：(0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7)。可见第一轮攻击能够获取到密钥所有字节的前4位。

**3.1.2. 第二轮攻击**

第二轮攻击过程与第一轮攻击过程类似，同样适用假设检验的方法。通过计算可以得到AES在第二轮中的访问索引如**表3**所示。

**Table 3.** The indices of AES in second round

**表3.** AES第二轮访问索引

|  |
| --- |
| ⊕⊕⊕⊕2•⊕⊕3•⊕⊕⊕ |
| ⊕⊕⊕⊕3•⊕⊕⊕⊕⊕⊕ |
| ⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕1 |
| ⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕ |

在进行攻击的过程中，首先将第一轮攻击的结果带入到假设的密钥中，并结合**错误!未找到引用源。**，然后通过假设检验的方式分组对密钥字节的后四位进行假设，并确保假设值能够遍历密钥的取值空间。通过同时假设、、、后四位的取值就能得到访问索引，通过对、、、同时进行假设就能够得到访问索引，通过对、、、同时进行假设，就能得到访问索引，通过同时对、、、的后四位进行假设就能够得到访问索引。

与第一轮攻击相同，在获取到AES查表索引后就能计算得到在执行Cache攻击中需要监测的Cache Set号。在攻击过程中，首先假设密钥值为，当对明文p进行加密时，通过计算得到访问的Cache Set号。然后攻击程序通过Prime+Probe检测该Set的访问情况，为了降低偶然因素对实验结果的影响，检测过程执行多次，最后通过KS检验将其与一组模板样本进行比对，最后得到该密钥的可疑度，其中模板样本中的数据为所有检测的地址都Cache命中的时间。其中KS检验为假设检验中的一种，可以判断两个分布是否为同一个分布，其返回0到1的值，越靠近0表示两个分布越相似。

与第一轮攻击相同，第二轮攻击使用假设检验的办法，并引入KS检验降低实验误差。本文实验设置密钥k为(0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77)，由于第一轮攻击获取了每个密钥字节的前4位，因此还有16\*4位数据等待获取。通过本节介绍的方法，获取得到的结果如**表4**、**表5**、**表6**、**表7**所示。

**Table 4.** The result of in second round

**表4.** 第二轮攻击结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 后四位取值 | 后四位取值 | 后四位取值 | 后四位取值 | 度量分 |
| 13 | 3 | 2 | 5 | 0.952152 |
| 1 | 4 | 10 | 8 | 0.952174 |
| 8 | 15 | 10 | 15 | 0.952371 |
| 8 | 5 | 8 | 9 | 0.952374 |
| 1 | 15 | 0 | 3 | 0.9524125 |
| 7 | 4 | 4 | 5 | 0.952766 |
| 0 | 5 | 2 | 7 | 0.9533185 |

**Table 5.** The result of in second round

**表5.** 第二轮攻击结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 后四位取值 | 后四位取值 | 后四位取值 | 后四位取值 | 度量分 |
| 10 | 4 | 8 | 10 | 0.9384495 |
| 4 | 7 | 6 | 11 | 0.9384925 |
| 1 | 13 | 0 | 1 | 0.938506 |
| 9 | 6 | 12 | 15 | 0.9385945 |
| 13 | 0 | 8 | 12 | 0.938629 |
| 2 | 14 | 10 | 6 | 0.938634 |
| 4 | 1 | 6 | 3 | 0.938652 |

**Table 6.** The result of in second round

**表6.** 第二轮攻击结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 后四位取值 | 后四位取值 | 后四位取值 | 后四位取值 | 度量分 |
| 4 | 8 | 7 | 11 | 0.9570985 |
| 11 | 7 | 12 | 15 | 0.95713 |
| 14 | 14 | 5 | 11 | 0.9571695 |
| 8 | 1 | 12 | 2 | 0.9572065 |
| 14 | 3 | 8 | 2 | 0.9572265 |
| 4 | 7 | 14 | 8 | 0.9574545 |
| 0 | 5 | 2 | 7 | 0.9578595 |

**Table 7.** The result of in second round

**表7.** 第二轮攻击结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 后四位取值 | 后四位取值 | 后四位取值 | 后四位取值 | 度量分 |
| 4 | 15 | 3 | 14 | 0.830743 |
| 1 | 9 | 6 | 10 | 0.8309515 |
| 5 | 15 | 0 | 12 | 0.8312265 |
| 7 | 14 | 5 | 3 | 0.831524 |
| 4 | 1 | 12 | 4 | 0.8315825 |
| 14 | 4 | 9 | 12 | 0.831855 |
| 4 | 1 | 6 | 3 | 0.8327175 |

以上各图表示Cache攻击第二轮攻击的结果，其中每个表的每一行的前4项表示假设密钥字节组合，比如**表4**中最后一行表示假设的值为，最后一列表示该组合的度量分，其中度量分数表示同一行4个密钥组合的可疑度，可疑度越高表示该行的密钥组合为真实的密钥值的可能性越高，因此可疑度最高的行就是Cache攻击估计的结果。可见每个字节后四位取值为的可疑度为0.9533185，由于每4个字节后四位组合共有65536种可能性，因此表中仅仅列出可疑度最高的7中组合。其中每个表中可疑度最高的组合就是攻击结果，因此可以获得AES每个字节的后四位为，与第一轮攻击获取的前4位组合起来得到攻击结果k ̂=(0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77)，与设置的密钥k相同，因此成功获取了全部密钥信息。

**4. 异步攻击设计**

上一章节介绍的攻击方式是同步攻击，与异步Cache攻击不同，同步攻击中被攻击程序对攻击程序来说是可控的，也就是说攻击程序能够控制被攻击程序何时执行，并知道被攻击程序何时执行完毕，以攻击AES高级加密算法为例，攻击程序能够触发AES加密程序的执行，并能够获取被加密的明文。但对于异步攻击来说，被攻击程序是完全不可控的，攻击程序只知道被攻击程序正在执行，因此攻击难度大大增大。

尽管上一章介绍的针对AES的同步攻击能够非常有效的恢复所有密钥，然而其需要攻击程序与被攻击的AES加密算法之间有交互，AES加密过程的明文对被攻击程序来说是可见的，且被攻击程序能够在加密过程之前或之后执行代码。本文探索针对AES高级加密技术的异步攻击，由于攻击程序不能控制加密的内容，攻击程序将在加密程序在相同的核上执行攻击程序，且他们之间没有显式的交互，攻击程序对加密程序仅有的了解是明文的分布情况，在本文中，假设明文是字母的组合。攻击程序将通过对自己内存空间的数据进行读取来获取攻击程序对Cache的使用情况，并依此破解AES的部分信息。

与同步攻击类似，异步攻击同样是通过假设验证的方式来破解AES密钥，首先假设AES密钥值为k，由于对于确定的明文p和密钥k就能够确定AES加密过程中访存情况，然而由于在异步攻击过程中，明文p是未知的，攻击者仅仅知道明文的分布情况，比如明文p全是字母，由于字母的ascii码十进制在141到172之间，转换16进制则在61到7A之前，因此如果假设明文仅仅为小写字母的组合，且分布完全随机的话，则明文的前四位为6的概率为15/26，前四位为7的概率为11/26。所以在第一轮攻击中可以假设明文前四位为6，及明文的十六进制设为p=0x6\*，其中\*从0~F中取值，假设密钥十六进制表示为k=0xnm，其中n和m均从0~F中取值，则k异或p则得到AES在执行加密过程中的对T-table的访问索引，AES加密过程中会经过10轮变换，每一轮都密钥变换都需要查表，其中第一轮查表索引i=p^k，其他轮的变换要相对复杂很多，因此当明文和密钥已知时，通过简单的异或操作就能求出第一轮查table的索引，当知道Table的首地址，以及首地址映射的Cache Set时，就能推测出第一轮查表操作访问的内存映射的Cache Set。在针对AES的Cache异步攻击工程中，由于明文为小写英文字母的组合，因此p=0x6\*，假设的k=0xnm，则第一轮访问索引为i=0x6\* ^ 0xnm。通常一个内存块大小64字节，一个table项大小为4字节，因此索引的后4位对应在字节块中的偏移，映射到Cache中则对应索引再Cache line中的索引，前4位则影响第一轮加密过程中访问table数据映射到的Cache Set号。当进行异步攻击时，由于明文p的前4位为0x6，当假设的密钥k遍历其取值空间时，计算0x6^0xn，并根据AES table首地址映射到的Set索引，相加就能得到预期的Cache访问情况，并验证该Set在aes运行期间的访问情况就能获取此轮假设密钥k值得度量分数，当获取到所有密钥取值的度量分时，进行排序，度量分最高的假设即为破解得到密钥值。

为了获取密钥假设值k的度量分，在执行异步攻击时，可以首先获取Cache Set的访问频率，并将其类比为代表密钥值k的可疑度，或称之为度量分m。对于Cache中的每一个Set，攻击程序通过一个循环不断的获取该Set的访问情况，监测该Set有没有在被被攻击AES加密过程中被访问到（具体的监测方式为：首先执行Prime操作，通过读取若干与该Set相关的地址占据该Set中的所有line，再执行Probe操作，检查在Prime阶段读取的数据是否还保存在该Set中，判断方式为比较Probe操作所花的时间，若所花的时间较短，则该Set没有被AES加密过程访问到，否则表示该Set被AES加密过程访问到了）。当只有攻击程序使用Cache时，所有的访存操作均能命中Cache，因此Probe获取的访问时间很快。然而，当被攻击的AES加密操作访问到某被监听的Set时，则会导致攻击程序映射到Cache中的某些line被驱逐到内存中，则该Set的监控时间相对较长。因此攻击者可以通过度量Set的被访问的次数来表示其度量分。

**4. 结束语**

移动设备安全关乎用户的隐私安全、财产安全甚至生命安全，然而由于Cache的设计缺陷，攻击者能够在不获取额外权限的情况下，通过对本地内存空间地址监测，获取到被攻击程序在运行期间的内存访问模式，并能够依此分析出用户的私密信息，包括用户的输入、加密算法密钥等。在本文的实验中，通过KS检验减少计时误差以及Cache替换策略对度量假设密钥可疑度的影响，成功获取了AES加密算法的全部密钥字节。因此，Cache相关漏洞应该得到系统设计人员以及软件开发人员的重视，以降低泄露用户私密信息的风险。

**参考文献 (References)**

1. 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.
2. 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.
3. Page D, Theoretical use of cache memory as a cryptanalytic side-channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol (2002). http://www.cs.bris.ac.uk/Publications/pub\_info.jsp?id=1000625
4. Wu WL, He YP, Feng DG, Qing SH. Power attack of mars and rijndael. Journal of Software, 2002,13(4):532−536 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/13/532.htm>
5. Zhou YB, Feng DG. Side-Channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. In: Proc. of the NIST Physical Security Workshop. 2005. 1−34.
6. Hou FY, Gu DW, Lin XY. Cache-Based attacks against AES: Research progress. Information Security and Communications Privacy, 2007, (8):41−43 (in Chinese with English abstract).
7. Deng GM, Zhao Q, Zhang P, Chen KY. Cache hit side channel attack based on AES. Computer Engineering, 2008,34(13):113−114 (in Chinese with English abstract).
8. Bonneau J, Mironov I. Cache-Collision timing attacks against AES. In: Goubin L, Matsui M, eds. Proc. of the Cryptographic Hardware and Embedded Systems (CHES 2006). LNCS 4249, Berlin: Springer-Verlag, 2006. 201−215. [doi: 10.1007/11894063\_16]
9. Li B, Hu YP, Zhong MF. Time-Based Cache attacks on AES. Computer Engineering, 2008,34(17):141−143 (in Chinese with English abstract).
10. Bernstein DJ. Cache-Timing attacks on AES. 2005. <http://cr.yp.to/papers.html\#cachetiming>
11. Lipp M, Gruss D, Spreitzer R, et al. ARMageddon: Cache Attacks on Mobile Devices[J]. Mundo Electrónico, 2015, 6(1):págs. 60-65.