Software Engineering and Applications 软件工程与应用, 2018, *(*), **-**
Published Online **** 2018 in Hans. http://www.hanspub.org/journal/sea
http://dx.doi.org/10.12677/sea.2018.*****

Hans汉斯

Research on AES Cache Attack Technology Based on ARM Processor

*

Bo Li[1]

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

Email: leborn@buaa.edu.cn

Received: ******

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

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

*资助信息：请根据信息量的大小调整文本框的高度。确保资助信息的内容不会超过页面的下边距并占用下方空白

文章引用: 作者 1, 作者 2, 作者 3. 文章标题[J]. 软件工程与应用, 2018, *(*): ***-***.
DOI: 10.12677/**.2018.*****

Email: leborn@buaa.edu.cn

收稿日期: ******

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. 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[4]和 Kelsey[3]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.[5]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[6]对 MARS 和 Rijindael 进行了详细的分析。 Zhou YB[7]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[8]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[9]参考了 Bonneau[11]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[10]基于 Bernstein[12]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.[4] 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。

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 POSIX 计时所示：

表 1 POSIX 计时

```
1.  #include <time.h>
2.  int64 get _time(void)
3.  {
4.     struct timespec t;
5.     clock_gettime(CLOCK_MONOTONIC, &t);
6.     return t.tv_sec * 1000*1000*1000ULL + t1.tv_nsec;
7.  }
```

get_time 方法能够返回精确的系统时间，图 1 Cache 命中与 Cache 未命中实验通过 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。

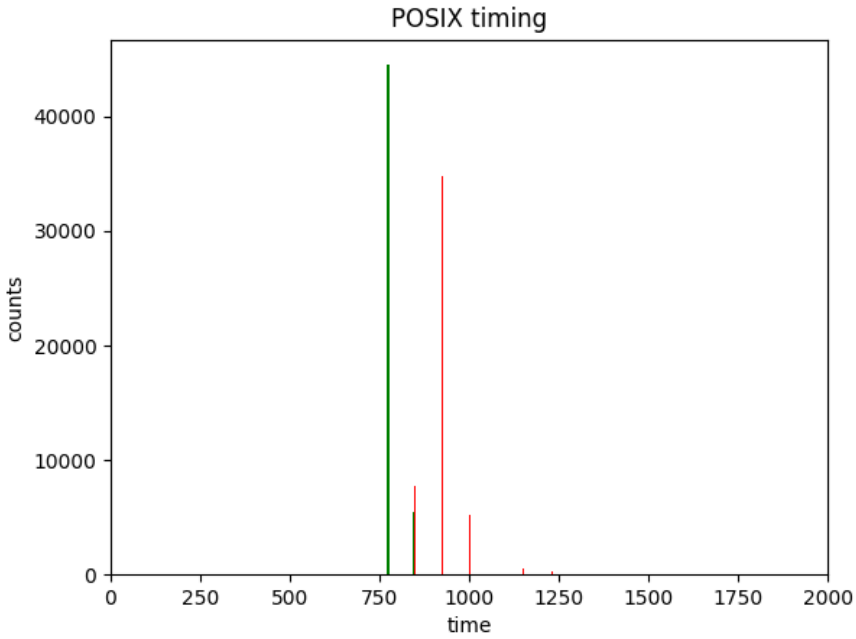

图 1 Cache 命中与 Cache 未命中实验

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。

表 2 获取高效驱逐策略

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.[1]提出的获取驱逐的思想，通过表 2 获取高效驱逐策略算法，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T_0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T_0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.[4]介绍的 Prime+Probe 攻击方式，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T_0、T_1、T_2、T_3中哪些区域的数据。

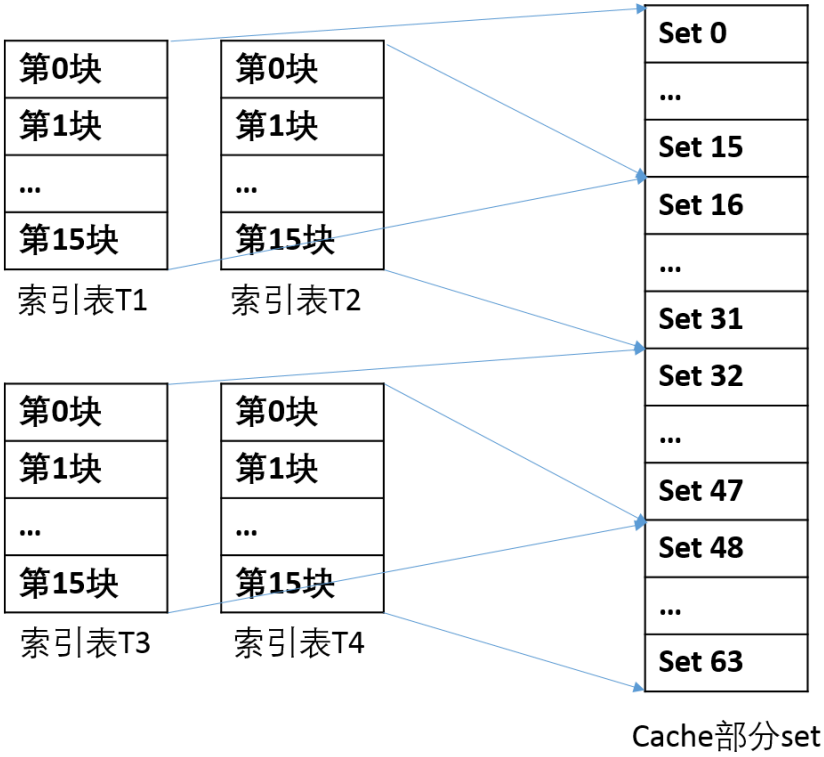

图 2 索引 Table 与 Cache 映射关系

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T_0、T_1、T_2、T_3 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。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T_0、T_1、T_2、T_3的首地址对应的 Cache Set 号为s_0、s_1、s_2、s_3，因此可以计算得到第一轮需要监测 Cache Set 号：$s_0 + (p_0 \oplus \hat{k}_0)/16$、$s_0 + (p_4 \oplus \hat{k}_4)/16$、$s_0 + (p_8 \oplus \hat{k}_8)/16$、$s_0 + (p_{12} \oplus \hat{k}_{12})/16$、$s_1 + (p_1 \oplus \hat{k}_1)/16$、$s_1 + (p_5 \oplus \hat{k}_5)/16$、$s_1 + (p_9 \oplus \hat{k}_9)/16$、$s_1 + (p_{13} \oplus \hat{k}_{13})/16$、$s_2 + (p_2 \oplus \hat{k}_2)/16$、$s_2 + (p_6 \oplus \hat{k}_6)/16$、$s_2 + (p_{10} \oplus \hat{k}_{10})/16$、$s_2 + (p_{14} \oplus \hat{k}_{14})/16$、$s_3 + (p_3 \oplus \hat{k}_3)/16$、$s_3 + (p_7 \oplus \hat{k}_7)/16$、$s_3 + (p_{11} \oplus \hat{k}_{11})/16$、$s_3 + (p_{15} \oplus \hat{k}_{15})/16$。

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 AES 第一轮攻击前 8 字节结果、图 4 AES 第一轮攻击后 8 字节结果所示。

图 3 AES 第一轮攻击前 8 字节结果

图 4 AES 第一轮攻击后 8 字节结果

图 3 AES 第一轮攻击前 8 字节结果、图 4 AES 第一轮攻击后 8 字节结果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 AES 第二轮访问索引所示。

表 3 AES 第二轮访问索引

$$n_2 = s(p_0 \oplus k_0) \oplus s(p_5 \oplus k_5) \oplus 2 \bullet s(p_{10} \oplus k_{10}) \oplus 3 \bullet s(p_{15} \oplus k_{15}) \oplus s(k_{15}) \oplus k_2$$
$$n_5 = s(p_4 \oplus k_4) \oplus 2 \bullet s(p_9 \oplus k_9) \oplus 3 \bullet s(p_{14} \oplus k_{14}) \oplus s(p_3 \oplus k_3) \oplus s(k_{14}) \oplus k_1 \oplus k_{15}$$
$$n_8 = 2 \bullet s(p_8 \oplus k_8) \oplus 3 \bullet s(p_{13} \oplus k_{13}) \oplus s(p_2 \oplus k_2) \oplus s(p_7 \oplus k_7) \oplus s(k_{13}) \oplus k_0 \oplus k_4 \oplus k_8 \oplus 1$$
$$n_{15} = 3 \bullet s(p_{12} \oplus k_{12}) \oplus s(p_1 \oplus k_1) \oplus s(p_6 \oplus k_6) \oplus 2 \bullet s(p_{11} \oplus k_{11}) \oplus s(k_{12}) \oplus k_{15} \oplus k_3 \oplus k_7 \oplus k_{11}$$

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$\hat{k}$中，并结合表 3 AES 第二轮访问索引，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k_0、k_5、k_{10}、k_{15}后四位的取值就能得到访问索引n_2，通过对k_3、k_4、k_9、k_{14}同时进行假设就能够得到访问索引n_5，通过对k_2、k_7、k_8、k_{13}同时进行假设，就能得到访问索引n_8，通过同时对k_1、k_6、k_{11}、k_{12}的后四位进行假设就能够得到访问索引n_{15}。

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$\hat{k}$，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$\hat{k}$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。

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 第二轮攻击结果$(\boldsymbol{k_0}, \boldsymbol{k_5}, \boldsymbol{k_{10}}, \boldsymbol{k_{15}})$、表 5 第二轮攻击结果$(\boldsymbol{k_4}, \boldsymbol{k_9}, \boldsymbol{k_{14}}, \boldsymbol{k_3})$、表 6 第二轮攻击结果$(\boldsymbol{k_8}, \boldsymbol{k_{13}}, \boldsymbol{k_2}, \boldsymbol{k_7})$、表 7 第二轮攻击结果$(\boldsymbol{k_{12}}, \boldsymbol{k_1}, \boldsymbol{k_6}, \boldsymbol{k_{11}})$所示。

表 4 第二轮攻击结果$(\boldsymbol{k_0}, \boldsymbol{k_5}, \boldsymbol{k_{10}}, \boldsymbol{k_{15}})$

k_0后四位取值	k_5后四位取值	k_{10}后四位取值	k_{15}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

表 5 第二轮攻击结果$(\boldsymbol{k_4}, \boldsymbol{k_9}, \boldsymbol{k_{14}}, \boldsymbol{k_3})$

k_4后四位取值	k_9后四位取值	k_{14}后四位取值	k_3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

表 6 第二轮攻击结果(k_8, k_{13}, k_2, k_7)

k_8后四位取值	k_{13}后四位取值	k_2后四位取值	k_7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

表 7 第二轮攻击结果$(k_{12}, k_1, k_6, k_{11})$

k_{12}后四位取值	k_1后四位取值	k_6后四位取值	k_{11}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 第二轮攻击结果$(\boldsymbol{k_0}, \boldsymbol{k_5}, \boldsymbol{k_{10}}, \boldsymbol{k_{15}})$中最后一行表示假设$(k_0, k_5, k_{10}, k_{15})$的值为(0,5,2,7)，最后一列表示该组合的可疑度，可见$(k_0, k_5, k_{10}, k_{15})$每个字节后四位取值为(0,5,2,7)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(0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7)，与第一轮攻击获取的前 4 位组合起来得到攻击结果 $\hat{k}$ = (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] 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.

[2] Irazoqui G, Inci M. S, Eisenbarth T et al, Lucky 13 Strikes Back[C]// ACM Symposium on Information, Computer and Communications Security. ACM, 2015.

[3] 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.

[4] 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.

[5] 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

[6] 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

[7] 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.

[8] 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).

[9] 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).

[10] Li B, Hu YP, Zhong MF. Time-Based Cache attacks on AES. Computer Engineering, 2008,34(17):141−143 (in Chinese with English abstract).

[11] 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]

[12] Bernstein DJ. Cache-Timing attacks on AES. 2005. http://cr.yp.to/papers.html\#cachetiming