Cache Attack on AES for Android Smartphone\*

Bo Li

School of Computer Science and Engineering   
Beihang Univeristy  
Beijing, China  
(+86)82338059

1st author's E-mail address

Bo Jiang†

School of Computer Science and Engineering   
Beihang Univeristy  
Beijing, China  
(+86)82338059

jiangbo@buaa.edu.cn

**ABSTRACT**

In this paper, we describe the formatting guidelines for ACM SIG Proceedings.

**CCS Concepts**

• **Information systems➝Database management system engines**   • **Computing methodologies➝Massively parallel and high-performance simulations.**This is just an example, please use the correct category and subject descriptors for your submission*.* The ACM Computing Classification Scheme:

<http://www.acm.org/about/class/class/2012>. Please read the [HOW TO CLASSIFY WORKS USING ACM'S COMPUTING CLASSIFICATION SYSTEM](http://www.acm.org/publications/article-templates/CCS-HOWTO-v6-12Jan2015.docx) for instructions on how to classify your document using the 2012 ACM Computing Classification System and insert the index terms into your Microsoft Word source file.

**Keywords**

Keywords are your own designated keywords separated by semicolons (“;”).

# INTRODUCTION

With the rapid development of Internet in recent years, mobile phones and other mobile devices have become an indispensable part of our lives. It not only brings convenience to our life, but also brings us the potential threat, especially today when the mobile phone can easily manage assets, store a variety of privacy. Therefore, with the progress of science and technology, more and more attention has been paid to the security problems. Although the mobile phone and other equipment manufacturers and Android system development groups have taken various measures to protect the privacy of users, including trusted execution environment, virtual memory management, permissions management, however, with the continuous research of android security, security vulnerabilities of mobile phone equipment gradually revealed, including cache attacks. Cache attack is a method to attack by detecting the condition of cache hit or miss, or detecting how much time is needed to access the memory when the program is executing. There is no direct interaction between the attack program and the victim program when cache attack is scheduled, and they execute parallel on the same kernel or different kernel to access their respective address space, so the attacker does not need extra permissions. In the last 10 years, more and more attention has been paid to CPU cache attacks on the Intel platform. Kocher and Kelsey et al. Proposed a method to decipher the encryption algorithm in the computer by analyzing the bypass information leaked by the cache at runtime. This idea has been developed rapidly under the attention of computer security personnel. In recent years, the cache attack method has been put forward on the Intel platform, and the experiment has been carried out in monitoring user keyboard input, AES T-table encryption cracking, and also proved some effective methods of cache attack. For example, the possibility of cross-process leakage via cache state was first considered in 1992 by Hu in the context of intentional transmission via covert channels. In 1998, Kelsey et al. mentioned the prospect of “attacks based on cache hit ratio in large S-box ciphers.” In 2002, Page described theoretical attacks on DES via cache misses, assuming an initially empty cache and the ability to identify cache effects with very high temporal resolution in side-channel traces. He subsequently proposed several countermeasures for smartcards, though most of these require hardware modifications and are inapplicable. In 2002 and subsequently, Tsunoo et al. devised a timing-based attack on DES, exploiting the effects of collisions between the various memory lookups invoked internally by the cipher. Furthermore, Gruss et al. demonstrated the possibility to automatically exploit cache-based side-channel information based on the approach. Besides attacking cryptographic implementations like AES T-table implementations, they showed how to infer keystroke information and even how to build a keylogger by exploiting the cache side channel. Similarly, Oren et al. demonstrated the possibility to exploit cache attacks on Intel platforms from JavaScript and showed how to infer visited websites and how to track the user’s mouse activity.

However, the architecture of Android is different from the Intel CPUS, and instruction set, cache organization mode and cache replacement strategy are different too. Therefore, it was not proposed an effective cross core cache attacks on non-root mobile phones until recently. Moritz Lipp et al. proposes a cross core attack model for ARM processors through prime + probe, flush + reload, evict + reload, and flush + flush, and does not require root permissions. These models can effectively gain the private information by leaked cache information, through the statistical analysis of the information, and the model can act on the cache attack from the user's private information. However, there is no example of fully implementing AES attacks on the Android platform. Because the cache structure of mobile devices based on Android system is different from Intel , there will be some problems when applying the attack method of Intel platform to mobile devices. For example, the replacement policy of cache on the platform uses the replacement policy, so you can simply evict data of specified set to memory. However, since Android uses a pseudo random replacement policy, additional methods are needed to expel data from set specified in cache into memory. In addition, in order to obtain stable data access time, it is usually necessary to preheat the access memory or the cache operation. The previous attacks usually use the first access to cache or memory access time to measure cache hit or not, so it is easy to introduce errors, resulting in unsatisfactory results.

# Background

## Determining the Best Eviction Strategy

We can use the instruction to evict cache lines to memory on Intel platforms. Although some android devices provide the similar operating instructions, but it usually should be used in privileged mode. So a more general strategy is needed to evict the content of specified cache to memory. Continuous addresses access is a more general strategy, which reads some addresses that can be mapped into the same cache set, the new data will be read through before the cache to the data in the cache memory deported to the way for the realization of the specified set drive operation. Although we can read large amounts of address to guarantee that the related data of specified set are out of memory, but a large number of action of accessing memory will not only increase time, but also increase the related address storage memory capacity, and since the mobile device cache using pseudo random replacement strategy, continuous reading of a plurality of memory data does not guarantee that the content of cache can completely be evicted into memory. Therefore, it is very important to find a rapid and effective strategy for side channel attack.

Gruss found that there are three main factors can affect the efficiency of cache eviction. First is the cache set size, and second is the access order of different addresses, and the third is the times of repeat access. So they three are crucial to the eviction strategy. The eviction algorithms are implemented as follows:

|  |  |
| --- | --- |
| Eviction algorithms | |
| Input：  N：The number of addresses that can be stored in set  D：The number of different addresses accessed in each loop  A：The number of accessed times for certain address pre loop | |
| Output：None | |
| 1:  2:  3:  4:  5:  6: | 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)  end for  end for  end for |

Therefore, in order to ensure the success rate of eviction, it is necessary to do a lot of experiments on all the specific combination of the three arguments to obtain a rapid and effective mode of eviction. In this paper, we use a python script to detect the possible combination of strategies, and finally select the best combination of arguments as the target machine's eviction strategy.

## The Prime Probe Strategy

In order to obtain private-sensitive information through the cache, the attacker should have the ability to obtain cache status. The approach allow an adversary to determine which cache sets are used during the victim’s computations and have been exploited to attack cryptographic implementations and to build cross-VM covert channels. The approaches consist of following three basic steps.

:

1. Occupy specific cache sets.
2. Victim program is scheduled.
3. Determine which cache sets are still occupied.

In the first stage, the attack program evicts the data on specified sets of cache with pre-computed eviction strategy by reading data from multiple addresses which could map to the sets in a certain way. In the second stage, the attack program has been executed normally, and the access memory operation in the execution process may occupy some lines in the cache. The third stage, attack program detects how many data still in cache, if the data occupied in the first stage is still stored in cache, then it indicates that the victim program did not access the certain cache sets during attacked period, if some lines of specified cache sets was evicted into memory, we can say that the victim program did not access memory which map to specified cache sets. The following figures show the schematic diagram of the three stages of .

Figure 1. Prime

As shown in the diagram, for a specified set, the Prime phase maps multiple data to cache and the previous cached data is evicted into memory.

Figure 2. victim program executes

When the victim program is executed, the memory which was accessed may be mapped to one of the set of cache and take part of the line in the set.

Figure 3. Probe

As shown in figure, Probe phase, check whether the data putted into cache in the Prime phase is still in cache. The method of checking data is still in the cache is by comparing the access time, because if the data occupied in Prime stage is expelled into memory, it may need more time to access the memory which was occupied in Prime stage thanks to a cache miss.

## Precise Measurement of Time

An accurate timing method is crucial for cache attack, it needs the ability to distinguish cache hit and cache miss, and helps the attacker to obtain the cache state caused by the access memory operation of victim program. Different information is acquired for different attack objects, for example, the attack of shared library needs to obtain the cache status of the shared library during the running of the victim program. However, the premise to obtain these conditions is to have the ability to accurately distinguish between cache hit and cache miss. In order to distinguish the situation between cache hit and cache miss, timing sources or dedicated performance counters can be used. Moritz Lipp et al. has proposed several non-privileged timing methods, including perf\_event\_open syscall, POSIX clock\_gettime functions, and dedicated thread timer. However, these interfaces are not open to all Android versions, and all processors. Therefore, it is necessary to determine the effective timing method that can accurately and stably measure memory access time which provide support for the cache attacker.

Besides reading the register to obtain the CPU cycle to measure the time, there are three other optional ways to measure the time of access or access to cache. The first is clock\_gettime syscall, and clock\_gettime is the timing function based on the Linux C language, which can be accurate to nanoseconds. The second is the Perf performance analysis tool. The perf\_event\_open syscall is an abstract layer to access performance information through the kernel independently of the underlying hardware. For instance, PERF\_COUNT\_HW\_CPU\_CYCLES returns an accurate cycle count including a minor overhead due to the syscall. The availability of this feature depends on the Android kernel configuration. The third is a thread timing simulator. If no interface with sufficient accuracy is available, an attacker can run a thread that increments a global variable in a loop, providing a fair approximation of a cycle counter. Our experiments show that this approach works reliably on smartphones as well as recent CPUs. The resolution of this threaded timing information is as high as the other methods. Because one incremental operation can be considered to be composed by a fixed number of CPU time, it can distinguish between cache hit and cache miss, which can also be used for cache attack.

# Attacking AES Algorithms

## The First Round Attack

In this paper, we use the approach of to efficiently extract full key, and the approach is divided into the first round and the second round attack. The first round attack is based on the access index of the first round of T-table which can be calculated through the key and plaintext. Given a 16 byte key K = ( ,...., ), it will be extended to the 10 round internal keys for r=1,... 10 in AES encryption process. Each round key is divided into 4 words of 4 bytes each:. the first round of access index of T-table can be obtained by the plaintext and key through operation. The index which calculated by 0th, 4th, 8th, 12th plaintext and key bytes are about 0th T-table, while calculated by 1st, 5th, 9th, 13th are about 1st T-table, and 2nd, 6th, 10th, 14th about 2nd T-table, and 3rd, 7th, 11th, 15th about 3rd T-table.

In order to obtain the key of the AES, in this paper we guess the value of the key first and then verify that whether they are the exact true key bytes. Then, the K-S test is used to obtain the measurement scores which is used to determine whether two distributions are the same or not. The samples of the first distribution are the time which obtained after Prime followed by a Probe method. While the second samples obtain time which is obtained Prime followed by an AES encryption operation and then Probe operation calculating the time. At this point, if the K-S test finds that the two samples belong to the same distribution, we can say that victim program did not access specified set during the execution of AES.

## The Second Round Attack

The First-Round attack can only narrow each key byte down to one of sixteen possibilities, but the table lookups in the first round cannot reveal more information. In our case AES key contains 16 bytes, so there are still 64 remaining unknown bits still too much for exhaustive search. The second round attack mainly use the nonlinear mixing relationship between plaintext and key in the cipher

The indexes accessed in the second round of encryption are not direct like in the first round. We exploit the following equations, easily derived from the Rijndael specification, which give the indices in four of the table lookups in the 2nd round. We can get second round access index as follows:

⊕⊕⊕⊕2•⊕⊕3•⊕⊕⊕

⊕⊕⊕⊕3•⊕⊕⊕⊕⊕⊕

⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕1

⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕

Here, denotes the Rijndael S-box function, and denotes multiplication over GF(256).

Index is related with T-table 2, and is the index of T-table 1, of T-table 0, of T-table 3. These indexes are the key to achieve AES attack. In this paper, AES attack are performed by hypothesis testing which travels the possible values ​​of all keys and tests which combination is most suitable. Identification of a correct guess is done by a generalization of the hypothesis testing method used for the one-round attack. For each candidate guess and each sample, we evaluate using the candidates while fixing the unknown low bits of some key bytes to an arbitrary value.

We should guess the first possible combination of the key values, that is, for each byte in the key, from 0 to 255 possible, and 4 half bytes or 16 bits at the same time to guess. For each guess, you can get a measurement score m, according to the value of m can determine which guess is the highest probability of the combination, when you guess the AES key. The measurement score is determined by obtaining a correspondence relationship with the guess key k on the AES t-table. We can gain the index on T-table when AES encryption is executed by plaintext p XOR key k on certain index. Because the T-table access index reflect the cache set index, T-table access index may finally lead to the difference of time between Prime operation and Probe operation.

## The K-S statistical Test

In statistics, the Kolmogorov–Smirnov test (K–S test or KS test) is a nonparametric test of the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one-sample K–S test), or to compare two samples (two-sample K–S test). It is named after Andrey Kolmogorov and Nikolai Smirnov. The two Sample K-S test is one of the most useful and general nonparametric methods for comparing two samples.

This paper exploits the K-S test to determine whether the victim program accessed the specified sets of cache when attack is scheduled. First the attack programs evict and occupy specified cache sets in Prime period by reading data relevant to specified sets continuously, so the previously cached data is expelled into memory. After Prime is a Probe operation to detect whether the data cached in Prime stage is still stored in the cache. Because Prime operation followed by the Probe operation immediately, we can assume that the Prime stage cached data is still cached in the cache, and these data consist the first part of the samples. Then we schedule to get the second part of the sample data, this stage records the impact on sets when the victim program running. The mainly processes are: first, the victim programs evict and occupy the specified cache sets, then the victim programs are scheduled, the process may occupy some sets of the cache, finally the probe phase, the attack programs detect whether the cache data occupied in Prime phase is still cached in the cache, and the result can be returned by the length of access time on specified cache sets. The Probe access time can reveal the leaked information when victim program running of cache sets. At this point, two samples which are crucial in cache attack are gained. the first samples are collections of data of which Prime followed by the Probe operation, and the second sample is the subsequent memories access of victim program after Prime and before Probe stage. Assuming that the memory accessed during the run of the victim program is not mapped to the specified cache set, the access time of the two samples should be similar and the result of the K-S test should be able to reflect that the two samples belong to the same distribution and response to K-S value is that the returned value is small. On the country, if the victim program does not access the data that can be mapped to the specified set, the time in the second sample will be greater than the time in the first sample or there will be a right offset. So under the help of K-S test we can automatically check the result of two samples K-S value to detect whether two samples belong to a same distribution or not.

## Performance Optimization

# Experimental Study

In order to verify that based on K-S test can effectively carry out AES attack on smartphone. This paper mainly implements the side channel attack focus on AES on Lenovo K51c78 which have a 512KB L2 cache. The details property of Lenovo K51c78 is as follows.

Table . Lenovo K51c78

|  |  |  |  |
| --- | --- | --- | --- |
| **System** | **Cache** | **Cache set** | **Evict Strategy** |
| Android 5.0 | 512 KB | 512 | Pseudo-random |

The attack processes can be divided into three parts, including the preparation phase, the first round attack and the second round attack. The preparation phase mainly is focus on finding the exact time of cache hit and cache miss thresholds and finding fast and effective eviction strategies for the target-smartphone’s system and cache structure. Base on the plaintext p and the hypothesis key k the first round attack calculates the access index of the T-table and the sets index it could be mapped on the cache. We record the Probe time as the first K-S samples at the same time. The other samples are recorded on the condition where Prime operation directly after the Probe operation. The two samples are parameters of the K-S test, and the result of K-S test as the measurement scores can distinguish victim program’s specified cache sets hit from miss. In order to gain all key bit we firstly travels all possible value of AES bytes and then obtain the related measurement score. The measurement score reflects the suspicious of a certain key guess. So we can sort the measurement score, and the most suspicious key is the key the attack guess. In this paper, the victim AES key k is set to (0x00, 0x11, 0x22, 0x33, 0x44, 0x58, 0x66, 0x77, 0x00, 0x11, 0x11, 0x23, 0x44, 0x45, 0x66, 0x77). In order to reveal all AES key, the first round attack is performed and the first four bits of each byte are revealed successfully. The result is as follows:

Figure 4 AES first round attack result

The AES key used in the experiment is set to k = (0x00, 0x11, 0x22, 0x33, 0x44, 0x58, 0x66, 0x77, 0x00, 0x11, 0x11, 0x23, 0x44, 0x45, 0x66, 0x77). As shown in Figure 1, we can obviously get the first four bits of each key. In order to obtain all the byte information of the AES key, the target smartphone is subjected to a second round attack by using the AES key information revealed in the first round and the mathematical relationship between AES T-table access index and the plaintext as well as key. Through the hypothesis testing of the rest key bits, we calculate the corresponding cache set access index, and then get the K-S value about the corresponding index as the measurement score, lastly we sort the score and find the highest score of suspicion. The assumption key bytes related to the highest measurement score are exactly key values. The experiment results are shown below:

Figure 5 AES second round attack result

As the pre-set key k is set to (0x00, 0x11, 0x22, 0x33, 0x44, 0x58, 0x66, 0x77, 0x00, 0x11, 0x11, 0x23, 0x44, 0x45, 0x66, 0x77), we can see that the second round of attack completely reveal all AES key bits.

# Related Work

# Conclusion

Although the possibility of cross-process leakage via cache state was considered long age, and there are some implements about fully AES key recovery on Intel platform already. However, modern smartphones use one or more multi-core ARM CPUs that have a different cache organization and instruction set than Intel CPUs. So far, no fully AES key recovery have been demonstrated on non-rooted Android smartphone. In this work we demonstrated the powerful cache attacks based on K-S test and successfully recovered all AES key bytes. Furthermore, these attacks do not require any permission or privileges. In addition, we show a sample which carry out AES cache attacks and finally recover all key bytes.

# ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

# REFERENCES

1. Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= <http://doi.acm.org/10.1145/161468.16147>.
2. Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.
3. Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= <http://doi.acm.org/10.1145/332040.332491>.
4. Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.
5. Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.
6. Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.
7. Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= <http://doi.acm.org/10.1145/964696.964697>.
8. Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= <http://dx.doi.org/10.1016/j.jss.2005.05.030>.
9. Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= <http://doi.acm.org/10.1145/90417.90738>.

Authors’ background

\*This form helps us to understand your paper better, the form itself will not be published. Please fill in every author’s information.

\*Title can be chosen from: master student, Phd candidate, assistant professor, lecture, senior lecture, associate professor, full professor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Your Name | Position\* | Email | Research Field | Personal website |
| Bo Li |  |  |  |  |
| Bo Jiang | Associate Professor | gongbell@gmail.com | Mobile testing and security | http://jiangbo.buaa.edu.cn |