BEST: International Journal of Management, Information Technology and Engineering (BEST: IJMITE)
ISSN 2348-0513
Vol. 2, Issue 5, May 2014, 9-14

SOFTWARE AND HARDWARE DEFENSE METHODS AGAINST CACHE-BASED SIDE CHANNEL ATTACKS

DEEVI RADHA RANI[1] & S. VENKATESWARLU[2]

[1]Women Scientist, Department of CSE, KL University Vaddeswaram, Guntur, Andhra Pradesh, India

[2]Mentor, Professor Department of CSE, KL University Vaddeswaram, Guntur, Andhra Pradesh, India

ABSTRACT

Cryptographic algorithms implementing on a cryptographic device leak information through side channels. Cache behavior in modern processors can be used as a side channel and retrieve the key used in cryptographic implementations. Cache based side channel attacks are serious hazard against modern computers with cache memory. This paper surveys the software and hardware defense methods against cache-based side channel attacks and analyze the efficient defense method against cache based side channel attack.

KEYWORDS: Cryptographic Algorithms, Cache Based Side Channel Attack, Software Defense, Hardware Defense

INTRODUCTION

Cryptosystems implementing cryptographic algorithms leak information through side channels e.g., power consumption, electromagnetic emanation and execution time. The information leaked from side channels is used to retrieve the key of a cryptographic algorithm. Power, electromagnetic, and timing attacks are well-known side-channel attacks. Side channel attacks exploit the vulnerability of the cryptographic implementations, cache-based side channel attacks are one kind of such attacks. Cache based Side Channel Attacks are serious hazard against modern computers with cache-memory. The cache-based side channel attacks retrieve the keys by utilizing the relationship between the leaked information from the cache and the data-dependent table lookups which are employed in the cryptographic implementations [1]. Kocher predicted that RAM cache hits could produce timing characteristic in the implementations of various cryptographic algorithms if tables in memory were not used identically in every encryption [10]. Also, Page proposed a theoretical model for narrowing the possible values of secret information and pointed out that the cache could be used as a side channel [11].

OVERVIEW OF CACHE BASED SIDE CHANNEL ATTACK

According to the types of leaked information taken into use, such attacks fall into three categories: Time-driven attacks, which exploit the aggregate execution time over a large number of samples; trace-driven attacks, which analyze individual cache hits and misses to yield information; and access-driven attacks, where the cache accesses are spied by another process.

In this paper, the focus is on software Cache based side channel attacks: Access-driven and time-driven, which recover cipher keys by exploiting side channel information leaks caused by the implementation of cryptographic algorithms and data dependent behavior of cache memory. Access-driven attacks exploit the correlation between the secret key and the cache usage of a crypto thread/process. Since the cache is shared among multiple processes/threads, an attacker may derive the cache usage of the victim process by controlling a carefully crafted process, which runs together with the victim process [4].

Time-driven attacks measure the execution times of victim processes and exploit the correlation between the secret key and the number of cache misses which in turn determines the execution time to infer the key [4]. In spite of the differences of these strategies, the cache-based timing attacks all depend on the fact that the execution time of an encryption process is directly affected by the number of cache misses and share the goal of narrowing the possible values of the cipher key.

The time-driven attacks are easy to implement among the types of cache-based side channel attacks, as they require less leaked information. Bernstein first put this idea into reality and successfully carried out the attacks on the AES algorithm [12]. Bonneau proposed a different strategy which aimed at the last round of the AES algorithm rather than the first round. He assumed that the execution time of the AES encryption would cost more if less cache collision occurred during the runtime [13].

SOFTWARE DEFENSE AGAINST CACHE-BASED SIDE CHANNEL ATTACKS

There are several countermeasures to prevent the loss of information through a side channel breach of the cache behavior.

Turn-off Cache S-Box Access

Traditional method to turnoff cache access is to remove cache but it seems unrealistic since it removes all instructions depend on cache. So an alteration is made where only the instructions that access cache are removed. Turning off the cache for S-box access, essentially employing cache-bypass to always load data directly from memory. By eliminating the potential for cache-hits and cache-misses, reduces performance significantly but ensure that each access takes the same length of time. This kind of defense can easily be implemented in software need to modify the cryptographic algorithm in use.

Avoid Lookup Table

Constant Time Implementation which means that the execution time does not depend on the secret key or the input data. Avoid table lookups/S-boxes and use some form of computed non-linear transformation instead. This not only offers greater assurance of constant time access, but allows the potential for parallel execution of such transformations which are denied by the need for sequential memory access. Another constant time countermeasure is the bitslice implementation of the AES. It does not use any lookup tables with key or data-dependent address, i.e., no information can leak through the cache side channel. The implementation provides a very high performance.

Perform Cache Warming

Both time driven and trace driven cache based side channel attack differentiate cache miss and cache hits so removing this difference would be good defense method. To reduce or prevent the leakage of information it is also possible to warm up the cache [11]. For small S-boxes, the lookup tables, or parts of them, are loaded into the cache before execution begins, called pre-fetching or Cache Warming. There will be no cache misses if data is loaded to cache before execution and hence no leakage. This kind of defence can be implemented in software easily without modification in algorithm. However, this is only true if the S-box content is never evicted by other data or instructions and the S-box fits entirely into the cache: neither of these assumptions are guaranteed and hence the method can only be described as statistically sound. As a defense method this technique is good but in terms of performance it is not efficient.

Insert Dummy Operations

Inserting dummy operations in the algorithm increase the execution time. This is considered when attacks operate on behavior traces rather than timing information: with enough dummy loads inserted one cannot be sure if a given cache-hit or cache-miss is produced by real or faked execution. This method is not efficient since the randomization is simply noise that can be statistically removed. Additionally, since extra operations need to be serviced, the overall average execution time might increase by an unattractive factor. This is efficient and can easily implemented in software by just adding a single loop to the algorithm. Inserting dummy operations changes the execution time randomly but it is independent of cache activity.

Reorder Memory Accesses

Random reordering of memory accesses reduces the correlation between a captured behavior trace or execution timing and the input and algorithm. This can be achieved by using non-deterministic processor architecture but must be careful not to introduce potential hazards from the reordering.

Insert Delays

Insert actual random delays in the execution to randomize the overall execution time. This suffers from the same drawbacks of inserting random load operations in the sense that the statistical noise can be removed and will potentially increases the average execution time.

Perform Blinding and Bucketing

Combination of input blinding and bucketing is also the countermeasure which is secure and efficient against timing attacks. The blinding randomizes the input of the cryptographic device. The bucketing divides the distribution of the execution time into intervals and returns the result of each encryption at the end of corresponding interval.

HARDWARE DEFENSE AGAINST CACHE-BASED SIDE CHANNEL ATTACKS

- In [7], Page proposed the use of configurable cache architecture to provide hardware assisted defense. The cache is dynamically split into protected regions and can be specifically configured for an application. In partitioned caches, a part of the cache is allocated exclusively to the protected process in order to prevent information leakage. This may cause inefficient cache sharing since the cache partition is fixed statically. Partitioned cache architecture can be used as defense mechanism against cache based channel attack. A partitioned cache is added to devices which are vulnerable to side channel attacks. Partitioned cache segregates the cache behavior of one process to other. It prevents intra process interference i.e. it provides with enough space to store entire S-box in cache and locks when pre-loaded. Segregation does not allow to forcibly flush the cache. Also partitioned cache uses longer cache lines which make attack more difficult. In **PCache, a**s Partitions are Static, it prevents sharing leading to performance degradation. A process use only few cache lines in the partition and the unused lines are not available to other processes.
- In [3], the partition-locked cache (PLcache) is proposed to address cache sharing problem with a fine-grained locking control so that only the cache lines, which contain the critical data, are isolated. PLcache eliminate cache interference. The PLcache, with minimal hardware cost, can help the software developer achieve security without losing performance.

In **PLCache,** interested cache lines are locked by creating private partitions. These cache lines cannot be used by other cache accesses which are not belonging to private partitions. PL Cache is flexible cache partitioning mechanism which achieves less performance degradation. The total architecture of PLCache depends on hardware addition to cache and system inference for the selection of cache lines to be locked. 3 tags are added to the original cache line.

L	ID	LL	Original cache line

L specifies whether cache line is locked or not, ID specifies owner of the cache, LL specifies cache line locked if there is access to page or segment. 2 mechanisms namely Instruction Set Extension, Segment/Page-based Protection are used for controlling which cache lines should be locked. In Instruction Set Extension, new set of load/store instructions are added to the base ISA which gives control on what data to lock. New instructions are described below.

Table 1

Name	Description
ld.lock/ld.unlock	Same as load instruction with the additional action: If the memory access hits in the cache or causes a cache line to be fetched into the cache, the L bit of the cache line is set/cleared.
st.lock/st.unlock	Same as store instruction with the additional action: If the memory access hits in the cache or causes a cache line to be fetched into the cache, the L bit of the cache line is set/cleared.

In Segment/Page-based protection, regions of memory containing tables are marked as locked using function calls like lock_mem_region() and unlock_mem_region(). Handling Cache access is same as traditional cache except L bit is updated but it differs during cache miss replacement algorithm changes due to locked cache lines.

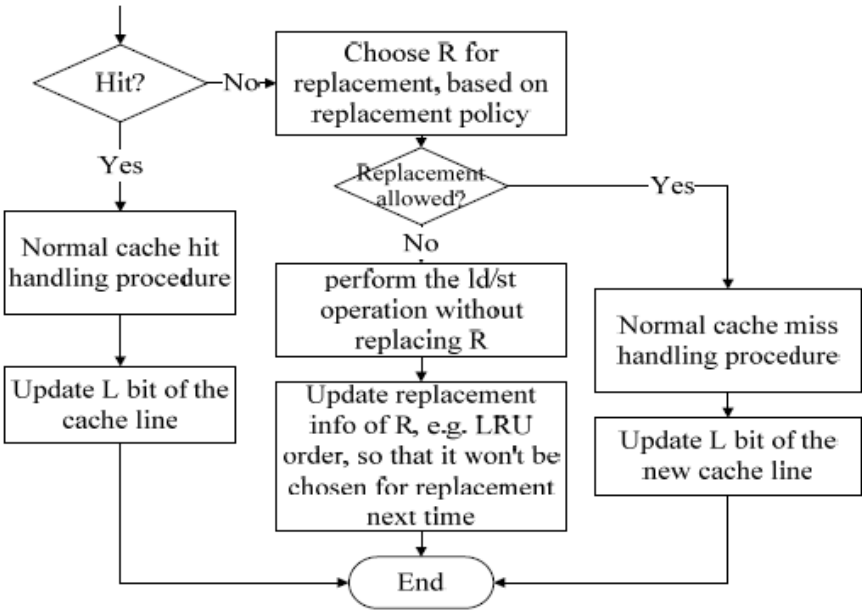

Figure 1: Cache Access Handling Procedure for Plcache

Let R be line driven by normal cache replacement algorithm and D be new data block being fetched into the cache. If D and R are not locked, D replaces R like normal cache miss. If D doesn't need to lock but R is locked, D cannot replace R. In this case, for load instruction D is returned to processor execution core and for store instruction data is return back to next level of memory without replacing R. If D needs to be locked, it is allowed to replace any line that is not locked or any locked line that belongs to the same process.

- Cache interference is the root cause of Cache-based attacks so novel general-purpose hardware solutions, the RPcache (Random Permutation Cache) randomize cache interference can be used. With a little more hardware, the RPcache can robustly provide both security and performance, even without input from the programmer. RPcache approach allows cache sharing, but randomizes the resulting interference, so no useful information can be inferred.

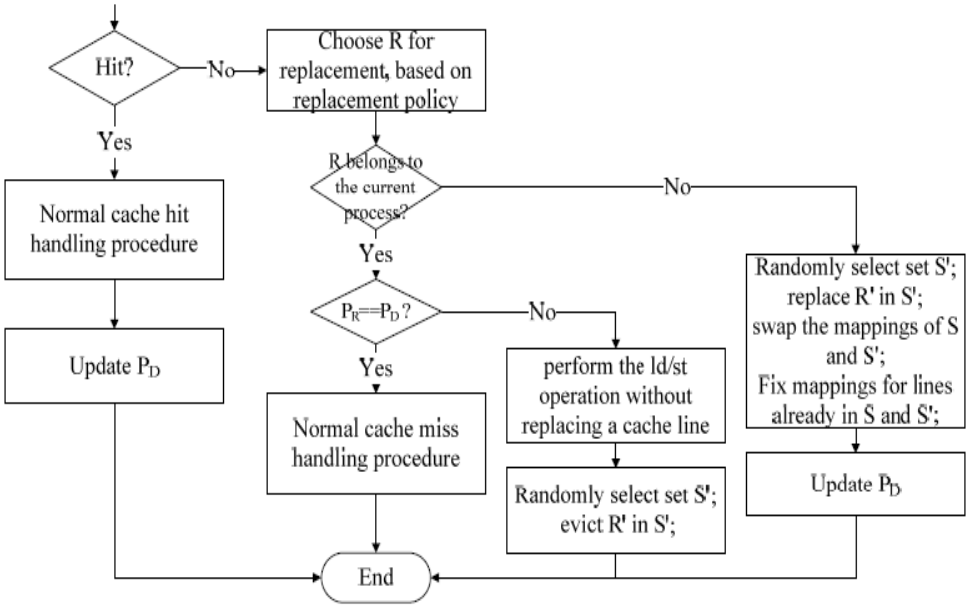

Figure 2: Cache Access Handling Procedure for Rpcache

- In [9], a precision timed architecture is put forward to achieve the timing-invariance features of hardware. The researchers claim that their mechanisms will completely eliminate the cache interference problem.

CONCLUSIONS

Cache based side channel attacks are serious threat against modern computers with cache memory. This paper presented various software and hardware defense methods against cache based side channel attacks. By analyzing various defense methods that can be implemented in software, producing non-linear mapping and replacing it with conventional S-box lookup tables would yield higher performance. Partition-locked cache is suggested to achieve the effect of cache partitioning with less performance degradation.

ACKNOWLEDGEMENTS

I would like to thank DST WOS-A for sponsoring me to do this research work and publish. I would also thank KL University for their support and facilities to carry out my research work.

REFERENCES

1. He Yuemei, Guan Haibing, Chen Kai, Liang Alei, "A New Software Approach to Defend against Cache-Based Timing Attacks," *Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on*, vol., no., pp.1,4, 19-20 Dec. 2009.
2. Jingfei Kong, Onur Aciicmez, Jean-Pierre Seifert, Huiyang Zhou, "Architecting Against Software Cache-based Side Channel Attacks," IEEE Transactions on Computers, 28 March 2012. IEEE computer Society Digital Library.

3. Zhenghong Wang and Ruby B. Lee. 2007, “New cache designs for thwarting software cache-based side channel attacks”, In *Proceedings of the 34th annual international symposium on Computer architecture* (ISCA '07). ACM, New York, NY, USA, 494-505.

4. Kong, J.; Aciicmez, O.; Seifert, J.-P.; Huiyang Zhou, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, vol., no., pp.393-404, 14-18 Feb. 2009.

5. O. Acıic¸mez and C¸ . K. Koc¸ “Trace-driven cache attacks on AES”, In *Information and Communications Security ICICS 2006*, LNCS 4307, pp. 112–121, Springer Verlag, 2006.

6. D. Page. Defending Against Cache Based Side-Channel Attacks. *Information Security Technical Report*, volume 8(1): 30-44, 2003

7. Page, D.: Partitioned cache as a side-channel defense mechanism. IACR Cryptology ePrint Archive, Report 2005/280 (August 2005)

8. Malte Wienecke, “Cache based Timing Attacks on Embedded Systems” available at http://www.emsec.rub.de/media/crypto/attachments/files/2010/04/ms_wienecke.pdf

9. Isaac Liu, David McGrogan, Elimination of Side Channel attacks on a Precision Timed Architecture. Technical Report, 2009. Available at: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-15.pdf.

10. Paul C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and other Systems. Lecture Notes in Computer Science, Springer, 1996.

11. D. Page, Theoretical Use of Cache Memory as a Cryptanalytic Side- Channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002. Available at: http://www.cs.bris.ac.uk/Publications/Papers/1000625.pdf.

12. Daniel J. Bernstein, Cache-timing Attacks on AES. Available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

13. Joseph Bonneau, Ilya Mironov, “Cache-Collision Timing Attacks Against AES**”,** In proceeding of Cryptographic Hardware and Embedded Systems - CHES 2006, LNCS 4249, pp. 201-215, Springer, 2006.