2012 IEEE/ACM 45th International Symposium on Microarchitecture Workshops

Hardware Prefetchers Leak : A Revisit of SVF for Cache-Timing Attacks

Sarani Bhattacharya, Chester Rebeiro, and Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
India
{*sarani.bhattacharya, chester, debdeep*}*@cse.iitkgp.ernet.in*

***Abstract*—Micro-architectural features have an influence on security against cache attacks. This paper shows that modern hardware prefetchers enabled in cache memories to reduce the miss penalty, can be a source of information leakage with respect to cache-timing attacks. The work revisits the Side Channel Vulnerability Factor (SVF) proposed in ISCA'12 and shows how to adapt the metric to assess the vulnerability of a prefetcher in cache-timing attacks. We use the modified metric denoted Timing-SVF, to show that standard prefetchers based on sequential algorithms can leak information in cache timing attacks. The findings have been established by experimental validations on a standard 128 bit cipher, called CLEFIA, designed by Sony Corporation Ltd. and used for light weight cryptography.**

***Keywords*-cache timing attacks, hardware prefetching, SVF, side channel measurements.**

I. Introduction

Side-channel attacks are the biggest threat to modern cryptographic systems. These attacks allow malicious user to gain access to sensitive data by monitoring power consumption, timing, or electro-magnetic radiation of the microprocessor. Cache-attacks [1], [2], [3] are a class of side-channel attacks that target cryptographic implementations that use look-up tables as a means to boost performance. Depending on the side-channel used, cache-attacks are classified into three: trace, access, and timing. A *cache-trace* attack typically monitors power consumption of the processor to obtain individual memory access patterns, while a *cache-access* attack obtains the same information by means of a spy process that co-exists on the same host. A *cache-timing* attack on the other hand monitors the time taken to perform the entire encryption, and relies on statistical means to determine the secret key.

Cache-trace attacks are mostly applicable for small embedded platforms where power measurements can be easily performed. Cache-access and timing attacks can be mounted from one virtual environment to another thus making these attacks a threat to cloud computing services [4], [5]. Further, since timing measurements can be made over a network, systems connected to a network are vulnerable to such attacks [6], [7], [8]. Thus cache-timing attacks are a serious threat and systems should be built to mitigate this attack form.

Cache-timing attacks have a black box view of the cipher, as compared to a gray box for cache-trace and access attacks. Hence they require more complex analysis to extract information from the side-channels. A popular cache-timing attack was introduced by D. J. Bernstein in 2005, which is based on construction of a template. In his attack, which came to be known as *profiled cache-timing attacks*, the secret key of an OpenSSL[1] implementation of AES was revealed to an unauthorized client running in a neighboring machine. The attack was critically analyzed by Neve et al. [9] and Canteaut [10], both providing variations of the attack. In [11], Bernstein's attack was adapted for an implementation of CLEFIA [12], and enhanced in [13] to improve the success rate.

The main reason for Bernstein's attack to work is the non-constant execution time [14]. There are various strategies that have been adopted to provide constant time implementations. For example, the use of specialized instructions, such as Intel's AES-NI [15], which eliminate memory based look-up tables and thereby result in near constant time execution of the crypto-algorithm. However, these instructions can only be used for the AES block cipher, leaving a large number of other block ciphers still vulnerable. Due to this limitation, several other schemes have been proposed, such as bitslicing [16], which eliminates use of look-up tables and TRESOR [17], which keeps encryptions outside RAM. However bitslicing cannot be used for operating modes requiring feedback, while encrypting outside RAM may have significant performance overheads (if specialized instructions are not used). Our research is in an alternate direction. We aim to build platforms which inherently protect against cache attacks. This approach will not be restricted to certain ciphers, operating modes and will not compromise on performance. A step towards this direction is to be able to quantify the side-channel vulnerability of a computing platform. Works such as [18], [19] present metrics to perform this quantification. For example the side channel vulnerability factor (SVF), is a metric which uses phase detection and correlation techniques to quantify the vulnerabilities of the system to side-channel attacks. However all side-channel

[1] http://www.openssl.org

978-0-7695-4920-0/12 $26.00 © 2012 IEEE
DOI 10.1109/MICROW.2012.13

metrics proposed so far are mainly applicable to cache-trace and cache-access attacks, and cannot be directly applied to profiled cache-timing attacks like Bernstein's, where critical events like memory accesses are unknown.

The next step to developing a fully secure system is to identify components which cause leakage and quantify the leakage. Some features in the architecture may assist the attack while other features may hinder the attack. Further *one man's meat is another man's poison*. An architectural feature which hinders an attack may assist another form of attack. One such feature is hardware prefetching for cache memories. In this paper we show that hardware prefetching algorithms like the commonly used sequential prefetching can abet profiled cache-timing attacks, due to the non-constant time encryption that it causes. The contributions of the paper are as follows.

- We first show how SVF can be modified to measure the vulnerability of a platform to profiled cache-timing attacks.
- We analyze sequential prefetching algorithms and show that they can leak information.
- We consider a block cipher CLEFIA, which has been attacked by profiled cache-timing attacks in [13], [11] and show its vulnerability due to sequential prefetching.

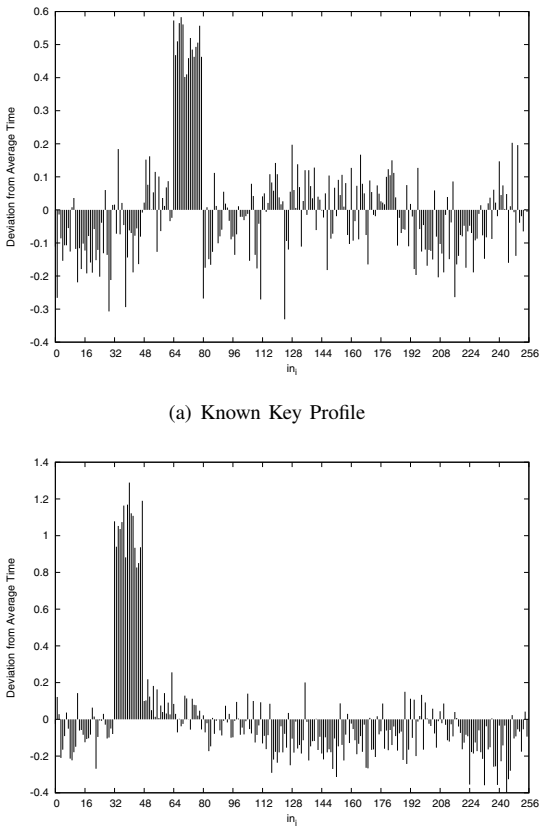
The structure of the paper is as follows. The second section contains the preliminaries for the paper, where profiled cache-timing attacks and prefetching are introduced. The third section shows the relationship between the number of cache misses and the timing for a block cipher. Section 4 presents the adaptation of the SVF metric for profiled cache-timing attacks. Section 5 shows how sequential prefetching can introduce non-constant time execution in a block cipher, and thereby causes leakage. Section 6 shows results for the cipher CLEFIA, and the final section has the conclusion and future directions of the work.

II. Preliminaries

This section provides the preliminaries that are required for the understanding of the paper. First the principle of profiled cache-timing attacks are presented. Then hardware prefetching in cache memories are introduced, and finally the SVF metric used to quantify side-channel vulnerability is discussed.

A. Profiled Cache-Timing Attacks

Like all other side-channel attacks, profiled cache-timing attacks also use a divide-and-conquer approach. The large secret key is broken down for example into bytes, and each byte is targeted independently. A profiled cache-timing attack targets the first operation where the targeted key byte is involved. This operation is typically an ex-or of the i^{th} plaintext byte (in_i) and the corresponding key byte (k_i). The attack comprises of three phases: learning, attack, and finally correlation. In the learning phase the attacker builds a timing profile for a known key. Then a timing profile for the unknown key is built, and the two profiles are correlated to reveal the secret key.



(a) Known Key Profile

(b) Unknown Key Profile

Figure 1. Timing Profile for OpenSSL AES on Intel Core 2 Duo

A typical timing profile obtained for AES is shown in Figure 1(a). It consists of all possible values of the i^{th} plaintext byte in_i as input on the x-axis, while the y-axis has the average execution time obtained when in_i is kept constant and all other plaintext bytes varied randomly. Figure 1(a) depicts the deviation from the average time instead of the actual timing. Figure 1(b) shows the timing profile for the unknown key. Except for a shift along the x-axis, the timing profiles in Figure 1(a) and 1(b) have similar characteristics.

The shift between the known key and the unknown key profiles are detected during the correlation phase of the attack. In this phase the timing profile for the unknown key is shifted and then correlated with the known key profile. This is done for all possible shifts, and the shift which results in the maximum correlation is likely to give away information about the secret key.

An obvious way to prevent the attack is to have constant time implementations [14]. The timing profile for a constant

Algorithm 1: *SP* : Sequential Prefetching Algorithm

Input: The access to memory block at address t_i
begin
  if *(t_i not in cache) or (t_i was prefetched and this is the first access to t_i)* **then**
    if t_{i+1} *not present in the cache* **then**
      prefetch t_{i+1}
    end
  end
end

time implementation would be a straight line parallel to the x-axis. For such implementations, there is no visible shift in the timing profiles and therefore reveals no information about the secret key.

B. Data Prefetching

A measure of the execution time can be estimated in terms of the memory accesses as follows [20]

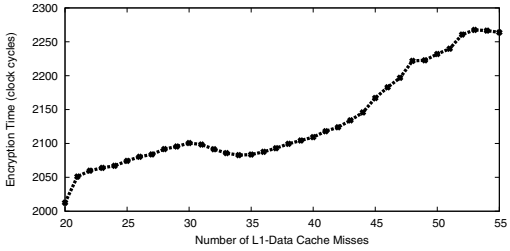
$$\tau = h * t_h + (1 - h) * t_m \ . \tag{1}$$

Here t_h is the time for a cache hit, h is the probability of a cache hit and t_m is the miss penalty. The cache miss overhead is significant therefore modern processors utilize micro-architectural acceleration techniques such as parallelization, out-of-order servicing of cache-misses, pipelining, non-blocking execution etc. One such technique is data prefetching which preloads the cache before the data is actually requested by the processor. The preloading is based on predictions from the current data being processed. There are two forms of prefetching: software prefetching and hardware prefetching. The software prefetching uses explicit prefetching instructions by the program that is run, while hardware prefetching has a dedicated prefetcher unit in the cache controller to automatically do the prefetching.

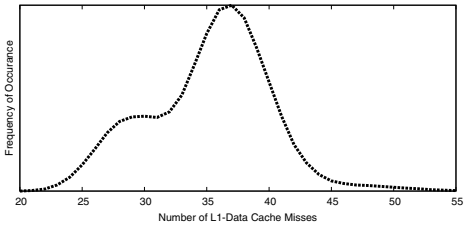
In this paper we consider a common form of hardware prefetching called sequential prefetching. In the standard sequential prefetching algorithm, an access to a memory block prefetches the next block into the memory. The algorithm is presented in Algorithm 1. If t_i is an memory address and t_{i+1} its subsequent address location, an access to the t_i would automatically prefetch t_{i+1}.

III. Cache Misses in a Block Cipher

For programs such as block cipher implementations using look-up tables, the memory operations are extensive (for example OpenSSL's AES implementation). Thus there exists a linear relation between the misses and the execution time of the program. Figure 2 shows the variation in the encryption time with the number of cache misses and the distribution of the cache misses for a typical block cipher using a 256 byte table. The experiments were done on a 2.8GHz Intel Core 2 Duo machine with a 32 KByte L1 data cache and running Ubuntu 10.04. The measurements were made using Intel's performance monitoring events and Linux's Perfmon library. It can be seen from the figure that the number of cache misses is Gaussian while the variation of the cache misses with the encryption time is linear. Due to these reasons, the execution time of the block cipher can be analyzed in terms of the number of cache misses.



(a) Encryption Time



(b) Distribution of Cache Misses

Figure 2. Encryption Time and Distribution vs Number of Cache Misses on Intel Core 2 Duo

We use the number of cache misses in a timing profile of a profiled cache-timing attack to analyze leakage. To quantify the leakage in the attack we use a recently published metric called the side-channel vulnerability factor (SVF). The next section discusses this metric and shows how it can be applied for profiled cache-timing attacks.

IV. Side Channel Vulnerability Factor in Profiled Cache-Timing Attacks

The side-channel vulnerability factor (SVF) [18] works on two vectors : the oracle trace and the side-channel trace. The oracle trace is the information which the attacker is attempting to observe, while the side-channel trace is built of information that is practically measured by the attacker. The SVF computes a similarity matrix for each trace and then uses the Pearson's correlation to detect patterns in the matrices. The value of the correlation coefficient represents the vulnerability of the attack platform. The closer this value is to one, the higher is the correlation between the traces and thus a more easily attackable target.

The SVF can be easily applied for side-channel attacks where the internal operations (such as memory accesses or table look-ups) are exposed to the adversary. The metric can be directly applied for cache-trace attacks and cache-access attacks. For example in a cache-trace attack, the oracle trace is a series of hits and misses corresponding to the memory accesses made by the cipher, while the side-channel trace

is the power consumption trace, which can be measured by the adversary. However, SVF cannot be directly applied for cache-timing attacks. In a cache-timing attack, the side-channel trace is the time for the complete encryption to execute. This is not a vector as required by SVF, but a single value. Moreover, there is no oracle trace easily available, due to the black box nature of the attack.

The SVF however can be enhanced and reused for profiled cache-timing attacks like [14]. A profiled cache-timing attack uses the correlation between the timing profile for a known key and the timing profile for the unknown key. There are two features in the timing profile that help the attacker. First the reproducibility of the timing profile and second the amount of variations in the profile. A system which reproduces timing profiles well would imply that the known key timing profile would highly correlate with the unknown key timing profile (except for the shift). About the variability, more variations in the timing profile would indicate more leakage. For example a constant time implementation would have zero variations in the timing profile hence have no leakage at all, while non-constant time encryptions would have variations in the timing profile and therefore leak information.

The enhancement to SVF should be able to capture both this information about the timing profile. To capture the reproducibility of timing profiles, we propose to build two timing profiles with the same key. We now have two vectors with the timing profiles. The first profile is used as the oracle trace, while the second profile is used as the side-channel trace. A good reproducibility of the timing profile would result in high SVF values, while poor reproducibility would result in low SVF values.

In order to capture the amount of variations and the reproducibility using a single metric, we shift the side-channel trace and compute the SVF. We do this for all possible shifts and thus would have as many SVF values as points on the timing profile. For CLEFIA for example, where a timing profile would have 256 points, there would be 256 SVF values. Next we compute Equation 2 to obtain a measure of the leakage. in the attack.

$$Timing\text{-}SVF = \sum_{\substack{\forall pairs\, of \\ i\, and\, i', i \neq i'}} |SVF(i) - SVF(i')| \quad (2)$$

The above equation sums up the absolute difference of every possible pair of SVF values (For AES there are $\binom{256}{2}$ such pairs). While the SVF detects just the reproducibility, Timing-SVF can also detect the amount of variations in the timing profile. Timing-SVF can have a minimum value of 0 indicating constant time implementation. As the variations in the timing profile increases, Timing-SVF would increase proportionally indicating a higher vulnerability of the system. If N are the number of points in the timing profile, Timing-SVF can have a maximum value of $2 \times \binom{N}{2} = N \times (N-1)$.

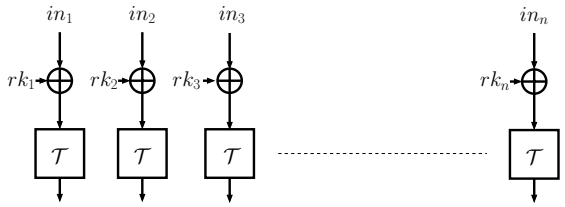


Figure 3. Memory Accesses in a Typical Block Cipher

V. Non-Constant time Execution in Sequential Prefetching

The memory accesses made by a typical block cipher is as shown in Figure 3. The inputs (in_1, in_2, in_3, ...) are ex-ored with the key material (rk_1, rk_2, rk_3, ...) and then used as an index to the look-up table. The table generally implements the s-box, which is the non-linear operation of the cipher. However, in some implementations the tables incorporate the s-box along with other linear operations of the cipher. In the system's memory, the table is split into blocks called *memory blocks*, and an access to an element in a memory block will load the entire block into a cache line, if a cache miss occurs. Thereafter, the memory block resides in the cache and all consecutive accesses to the memory block will result in a cache hit (unless the block is evicted by another process).

In Figure 3, the initial inputs are the plaintext while the remaining inputs are intermediate results of the cipher. If the plaintext is chosen uniformly at random, then the intermediate values of a good cipher (like all block ciphers used in practice today) will also be uniformly random. Thus the accesses made to the table are also at uniformly random indices. What this means is that every location in the table has an equal probability of being accessed and every memory block occupied by the table is equally likely to have a cache miss.

However, the uniformity in the probability of a cache miss across all memory blocks of the table does not hold when the cache memory supports prefetching. Depending on the prefetching algorithm used, the probability of obtaining a cache miss may vary from one memory block of the table to another. We explain the behavior with respect to the sequential prefetching algorithm (Algorithm 1). We initially assume that the table is isolated, which means that no memory access outside the table can prefetch a memory block of the table. Later we will relax this assumption.

With respect to the sequential prefetcher in Algorithm 1, the memory blocks of the table can be split into three categories (depicted in Figure 4).

- The first memory block of the table can never get prefetched because it would require a memory access to a block preceding the first block. This is outside

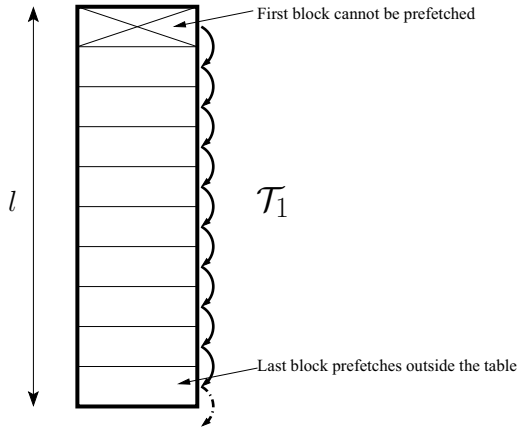


Figure 4. Prefetchable Blocks of a Table Used in a Block Cipher

the table boundary, and therefore by our assumption, cannot prefetch any part of the table.
- All blocks of the table except the last prefetch a memory block inside the table (a valid prefetching by our assumption).
- An access to the last block of the table prefetches a memory block outside the table boundary.

This classification of memory blocks results in non-constant time executions of the cipher and a timing profile which leaks information. In the following discussion we understand the impact of this on the timing profile.

The x-axis of the timing profile (Figure 1) consists of all possible values for the i^{th} byte of the input in_i. Since rk_i is a constant during the entire process of building the timing profile, the ex-or $in_i \oplus rk_i$, accesses all blocks of the look-up table. Further each point in the timing profile is built by placing in_i at a constant value (*i.e.* the memory block accessed by $in_i \oplus rk_i$ is fixed) and determining the average time when all other inputs are varied randomly. Assuming that the table occupies l memory blocks, one of three scenarios occur depending on which memory block is accessed by the fixed $in_i \oplus rk_i$.

- If the first memory block is fixed by $in_i \oplus rk_i$, then the second memory block is prefetched. Assuming no conflict misses and a clean cache at the start of encryption, then there are $l-2$ memory blocks of the table remaining, which can result in cache misses. All these $l-2$ memory blocks are also prefetchable.
- If the last memory block is fixed by $in_i \oplus rk_i$, the prefetching is done outside the table and will not affect the table accesses. In this case there are $l-1$ cache misses that can still occur and all these memory blocks can be prefetched except for the first.
- If any other memory block other than the first and last is fixed by $in_i \oplus rk_i$, then $l-2$ cache misses can occur and $l-3$ of these blocks can be prefetched.

Table I
CACHE MISSES IN A TABLE OF SIZE l USED IN A BLOCK CIPHER WITH SEQUENTIAL PREFETCHING ENABLED

Memory Block of table fixed	Number of Cache Misses Possible	Number of Sequential Prefetching Possible	Remarks on Average Time
First	$l-2$	$l-2$	t_1
Any except first and last	$l-2$	$l-3$	t_2
Last	$l-1$	$l-2$	t_3

Comparison of the Average Timings : $t_1 < t_2 < t_3$

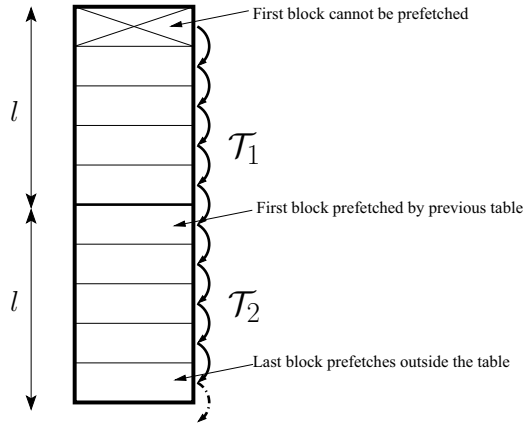


Figure 5. Prefetchable Blocks of two Adjacent Tables used in a Block Cipher

These observations are summarized in Table I along with the average timing expected. The first memory block would result in the fastest execution as it has the least number of cache misses possible, and all memory blocks can be prefetched. The last memory block results in slowest encryptions because it has the most number of cache misses. Thus prefetching causes non-constant encryption time to occur, which is captured in the timing profile.

Relaxing the Assumptions : The previous analysis was made under the assumption that the table is isolated, and other memory accesses made by the program do not affect the prefetching in the table. In reality however this assumption would not always hold. To show the effect we consider a block cipher implemented with two tables $\mathcal{T}_1$ and $\mathcal{T}_2$ placed side-by-side in memory (as shown in Figure 5). If $in_i \oplus rk_i$ accesses $\mathcal{T}_1$, then the last memory block prefetches the first memory block of $\mathcal{T}_2$. Conversely, if $in_i \oplus rk_i$ accesses $\mathcal{T}_2$, then the first memory block can be prefetched by the last memory block of $\mathcal{T}_1$. This can cause a reduction in the leakage as there are fewer variations in the encryption time.

VI. EXPERIMENTAL RESULTS

There are several factors that affect the execution time of a processor, and hence possible sources of leakage in profiled cache-timing attacks. The aim of our experiments however

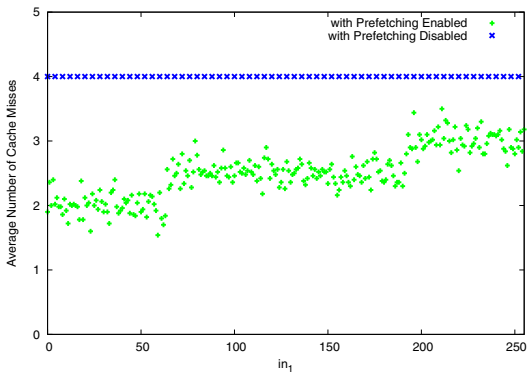


Figure 6. The Cache Profile for a Table of 256 Bytes

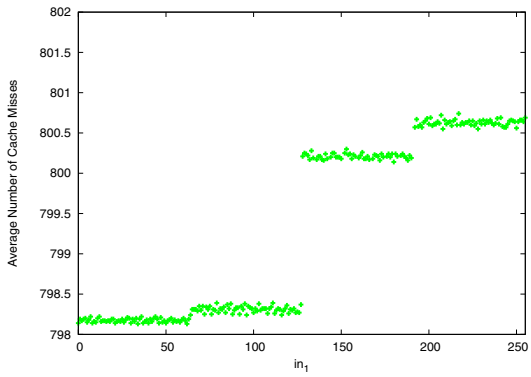


Figure 7. The CLEFIA Cache Profile

is to quantify the amount of leakage that occurs due to the sequential prefetching. To this accord we need to eliminate the other sources of leakage as best as possible.

The leakage can be from either the cache memory or other architectural features. Examples of the latter case is the dependence of the memory load time on the address [21]. To eliminate these forms of leakage we use the fact that the number of cache misses has a linear relationship with the timing (Figure 2), and analyze the cache misses instead of the timing.

We used the Cachegrind[2] tool as a simulation platform, which was modified to support the necessary prefetching algorithms. As a first step we used a table of 256 bytes accessed 36 times at random locations. This simulates table accesses in a block cipher. We executed this simulation on a direct mapped cache of $32KB$, with a 64 byte cache line. The large cache compared to the small table ensures minimum conflict misses. Figure 6 shows the cache-miss profile, which plots the average number of cache misses instead of timing, with respect to the value of the input in_1) obtained for this table with and without the prefetching. For cache-memories without prefetching, the number of cache misses is a constant, while with prefetching, variations in the cache misses are obtained. Since the simulated cache had a 64 byte cache line, the 256 byte table occupied 4 memory blocks. As predicted in Table I, the first and fourth memory block resulted in minimum and maximum cache misses respectively, while the second and third block had intermediate number of cache misses. *The profile with prefetching disabled had* $Timing\text{-}SVF = 0$*, while with prefetching enabled had* $Timing\text{-}SVF > 0$*, thus showing the leakage due to the prefetching*.

The next set of experiments were more realistic with an actual block cipher implementation. We chose to analyze the block cipher CLEFIA [22], which is the only cipher other than AES to be attacked using profiled cache-timing attacks [11], and unlike AES, does not have any specialized instructions. The implementation[3] of CLEFIA used two tables of 256 bytes each, with each table accessed 72 times. The cache memory simulated had two levels. The first level data cache was of $32KB$ and 8-way set associative while the $L2$ cache is of $6MB$ and 24-way set associative. Figure 7 shows the cache-profile obtained. We see characteristics in Figure 7 which are similar to Figure 6. Notably the minimum and maximum number of cache misses are in the extreme ends of the profile. *The Timing-SVF for CLEFIA with the sequential prefetching was found to be* 3237.

A variant of the sequential prefetching algorithm called prefetch on miss was also simulated. Unlike Algorithm 1, this algorithm prefetches the subsequent memory block only if a cache miss occurs. The Timing-SVF for CLEFIA for the prefetch on miss algorithm was found to be 8261, thus more leakage than the prefetching strategy of Algorithm 1. This experiment conveys the dependence of the prefetching algorithm on the leakage.

VII. Conclusions and Future Work

Micro-architectural features in a system has diverse effects on security applications. The paper shows that popular sequential prefetching algorithms can result in non-constant encryption time in a timing profile thereby a source of leakage these attacks. The SVF metric is enhanced in order to quantify the leakage in profiled cache-timing attacks, and the block cipher CLEFIA is used as a platform to demonstrate the claim.

As a future work, we would consider other prefetching algorithms such as arbitrary-stride prefetching, and compare their information leakage. The final aim is to build prefetching algorithms which can minimize or eliminate information leakage, without any overhead in performance.

[2]http://valgrind.org/docs/manual/cg-manual.html

[3]http://www.sony.net/Products/cryptography/clefia/download/data/clefia_ref.c

REFERENCES

[1] D. Page, "Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel," 2002.

[2] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of DES Implemented on Computers with Cache," in *CHES*, ser. Lecture Notes in Computer Science, C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., vol. 2779. Springer, 2003, pp. 62–76.

[3] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi, "Cryptanalysis of Block Ciphers Implemented on Computers with Cache," in *International Symposium on Information Theory and Its Applications*, 2002, pp. 803–806.

[4] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring Information Leakage in Third-Party Compute Clouds," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 199–212.

[5] M. Weiß, B. Heinz, and F. Stumpf, "A cache timing attack on aes in virtualization environments," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, A. D. Keromytis, Ed., vol. 7397. Springer, 2012, pp. 314–328.

[6] O. Aciiçmez, W. Schindler, and Çetin Kaya Koç, "Cache Based Remote Timing Attack on the AES," in *CT-RSA*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 4377. Springer, 2007, pp. 271–286.

[7] D. Brumley and D. Boneh, "Remote Timing Attacks are Practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[8] S. A. Crosby, D. S. Wallach, and R. H. Riedi, "Opportunities and Limits of Remote Timing Attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 3, 2009.

[9] M. Neve, J.-P. Seifert, and Z. Wang, "A Refined Look at Bernstein's AES Side-Channel Analysis," in *ASIACCS*, F.-C. Lin, D.-T. Lee, B.-S. Lin, S. Shieh, and S. Jajodia, Eds. ACM, 2006, p. 369.

[10] A. Canteaut, C. Lauradoux, and A. Seznec, "Understanding Cache Attacks," INRIA, Research Report RR-5881, 2006. [Online]. Available: http://hal.inria.fr/inria-00071387/en/

[11] C. Rebeiro, D. Mukhopadhyay, J. Takahashi, and T. Fukunaga, "Cache Timing Attacks on CLEFIA," in *INDOCRYPT*, ser. Lecture Notes in Computer Science, B. Roy and N. Sendrier, Eds., vol. 5922. Springer, 2009, pp. 104–118.

[12] Sony Corporation, "The 128-bit Blockcipher CLEFIA : Algorithm Specification," 2007.

[13] C. Rebeiro and D. Mukhopadhay, "Boosting Profiled Cache Timing Attacks with Apriori Analysis," *Information Forensics and Security, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[14] D. J. Bernstein, "Cache-timing Attacks on AES," Tech. Rep., 2005.

[15] Shay Gueron, "Intel Advanced Encryption Standard (AES) Instructions Set (Rev : 3.0)," 2010.

[16] C. Rebeiro, A. D. Selvakumar, and A. S. L. Devi, "Bitslice Implementation of AES," in *CANS*, ser. Lecture Notes in Computer Science, D. Pointcheval, Y. Mu, and K. Chen, Eds., vol. 4301. Springer, 2006, pp. 203–212.

[17] T. Müller, F. C. Freiling, and A. Dewald, "TRESOR Runs Encryption Securely Outside RAM," in *USENIX Security Symposium*. USENIX Association, 2011.

[18] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-Channel Vulnerability Factor: A Metric for Measuring Information Leakage," in *ISCA*. IEEE, 2012, pp. 106–117.

[19] L. Domnitser, A. Jaleel, J. Loew, N. B. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity Mitigation of Cache Side-Channel Attacks," *TACO*, vol. 8, no. 4, p. 35, 2012.

[20] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann, 2006.

[21] A. Fog, "The Microarchitecture of Intel and AMD CPU's, An Optimization Guide for Assembly Programmers and Compiler Makers," 2009.

[22] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-Bit Blockcipher CLEFIA (Extended Abstract)," in *FSE*, ser. Lecture Notes in Computer Science, A. Biryukov, Ed., vol. 4593. Springer, 2007, pp. 181–195.