A Formal Security Analysis of Even-Odd Sequential Prefetching in Profiled Cache-Timing Attacks

Sarani Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, India
sarani.bhattacharya @cse.iitkgp.ernet.in

Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Madras, India
chester @cse.iitm.ac.in

Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, India
debdeep @cse.iitkgp.ernet.in

ABSTRACT

Hardware cache prefetching has a profound impact on the memory access pattern of ciphers which are exploited in profiled cache-timing attacks. In this paper, we formally demonstrate that memory access patterns influenced by sequential prefetching and its variant, known as even-odd prefetcher has varying information leakage dependent on the alignment of the underlying tables used in the cipher implementation. This demonstrates that a suitable architecture choice for the hardware prefetcher combined with appropriate memory alignment in software can lead to prefetching architectures which are leakage resilient.

Keywords: Hardware Prefetcher, cache miss, quantifying information leakage, profiled cache-timing attacks.

1. INTRODUCTION

In 2005, D. J. Bernstein developed a profiled cache-timing attack on AES [2]. The attack had two phases, first a learning phase in which the attacker uses a known key (or an exact replica of the attack platform) to build a timing profile known as the *template*. Then, for the unknown key, the attacker builds another timing profile, correlates it with the template to obtain secret information about the cipher. Bernstein's attack was analyzed in [8] and [5], and adapted to attack CLEFIA in [10].

There have been several attempts made which analyze the reason for the attack to work. Bernstein in [2] attributed the non-constant encryption time to be the cause for leakage in profiled cache-timing attacks. Non-constant encryption times are due to look-up tables used in implementation of ciphers. Although there are methods to implement ciphers without tables, such as bit-slicing [4] and dedicated instructions [12], but they are limited to operating modes that do not support feedback or restricted to certain ciphers. Therefore, tables are mandatory in cipher implementations which operate in feedback modes and do not have dedicated instructions.

A formal treatment for profiled cache-timing attacks was introduced in [9]. Here, a method to quantify leakage in these attacks was developed. It was shown that hardware cache prefetching [6], a common feature in most modern microprocessors, resulted in non-uniform encryption time and therefore a cause of leakage in profiled cache-timing attacks. Two commonly used prefetching strategies namely sequential prefetching and arbitrary-stride prefetching [1] were analyzed in [9]. The results in [9] shows that sequential prefetching resulted in considerable leakage and the magnitude of leakage was affected by the size and number of tables used in the cipher implementation, as well as prefetching boundaries and relative placement of the tables in memory. Also in [3], it was shown that the popular sequential prefetching algorithm can result in non-constant encryption time in timing profile and thus results in information leakage. The paper [3] considers block cipher CLEFIA and its vulnerability due to sequential prefetching is demonstrated using a metric Timing SVF in the context of profiled cache-timing attacks.

An obvious way to prevent leakage due to prefetching is to disable the prefetcher. However, this may hamper the performance of applications. It is therefore required to have prefetching algorithms which do not compromise the security of the application. In this paper we analyze the leakage involved in a variant of sequential hardware prefetching algorithm termed as *even-odd sequential* (EOS), which is used in certain Intel microprocessors such as Intel Xeon E5345. We formally analyze this prefetching algorithm using metrics developed in [9] and show that leakage due to the EOS prefetcher depends on the size and alignment of the tables used in the cipher. Further, we show that for certain table alignments there is no leakage due to the EOS prefetcher. The results were verified with cachegrind [1]. We show that for a particular table alignment the leakage is always zero and for other alignments leakage reduces for large tables.

The contributions of the work are as follows:

- We identify a new prefetching algorithm: Even-Odd Sequential Prefetcher and formally analyze the prefetcher using combinatorial and provable techniques.

HASP 2016, June 18 2016, ,
© 2016 ACM. ISBN 978-1-4503-4769-3/16/06. . . $15.00
DOI: http://dx.doi.org/10.1145/2948618.2948624

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

- From the analysis we conclude that for a particular table alignment this prefetching technique results in zero leakage and we validate the result using cachegrind- the cache profiler.

The organization of the paper is as follows: in the next section we briefly introduce the profiled cache-timing attack and the common variants of hardware prefetching algorithm. In Section 3, a formal analysis of Even-Odd Sequential prefetcher is presented for four separate table alignments. Section 4 starts with the formulation of cache miss distribution followed by a validation step where the behavior of each predicted distribution is compared with empirical distributions from a cache profiler. Finally in Section 5, we compute and compare the leakage from the distributions using the Kullback-Leibler Divergence metric across the table alignments. Using this metric we show that for a particular table alignment the EOS prefetcher do not leak any information. In Section 6, we conclude the work that we present here.

2. PRELIMINARIES

In this section we initially provide a brief discussion on profiled cache-timing attacks and is followed by an introduction to hardware prefetching algorithms present in modern microprocessors.

2.1 Profiled Cache-Timing Attacks

The profiled cache-timing attack has 3 phases: learning, attack, and analysis phase. During the *learning phase*, the adversary uses a known key (or an exact replica of the attack platform) to build a *timing profile* for each key byte. The timing profile for the first key byte (k_0) of OpenSSL's AES[2] is shown in Figure 1(a). The profile has on the $x-$axis, all possible values corresponding to the plaintext byte p_0, and on the $y-$axis the average encryption time corresponding to when p_0 was fixed at a certain value and the remaining input bytes varied randomly. Note that Figure 1(a) shows the deviation from average encryption time, instead of the actual encryption time. The timing profile was build with around 2^{24} encryptions and called the *template* in this phase of the attack.

During the *attack phase*, the adversary builds a similar timing profile, only this time for the unknown key byte. Such a profile is shown in Figure 1(b). It can be noted that this profile is very similar to the template in Figure 1(a), except for a shift, which occurs due to the *equal images under different sub-keys* (EIS) [11] property of the cipher. It is this EIS property that results in the leakage of information about the secret key. During the *analysis phase* this shift is determined using correlation techniques in order to retrieve the unknown key byte. In similar manner timing profiles constructed for other bytes can be used to determine other parts of the key. However, we consider only the first byte. Similar analysis can be made for other key bytes as well.

In [2], constant-time cipher implementations were first discussed. In such a scenario, the timing profiles would have a straight line that overlaps with the x-axis, thereby every value of p_0 would result in exactly the same average encryption time and retrieving the unknown key byte would be impossible.

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

```
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
```
Algorithm 1: *SP* : Sequential Prefetching Algorithm

```
Input: Address of the memory block accessed ($t_i$)
begin
    if ($t_i$ is not present in cache) or ($t_i$ was prefetched and
      this is the first access to $t_i$) then
        if $t_i$ is even and $t_{i+1}$is not in cache then
            prefetch $t_{i+1}$
        end
        if $t_i$ is odd and $t_{i-1}$ not in cache then
            prefetch $t_{i-1}$
        end
    end
end
```
Algorithm 2: EOS: Even-Odd Sequential Prefetching

2.2 Hardware Cache Prefetching

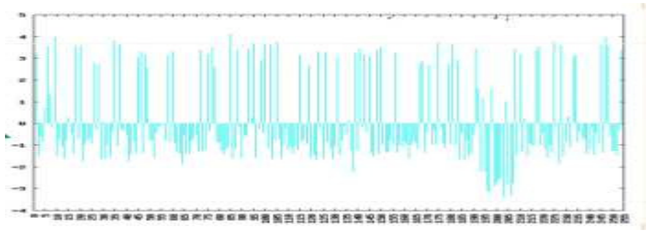
In a memory intensive program such as a block cipher implemented with look-up tables, the performance bottleneck is due to the penalty in servicing the cache misses. Modern cache-memories however have micro-architectural components in order to reduce the miss penalty. Hardware prefetching [6] is one such component in which data is automatically prefetched into the cache from the main memory before it is actually referenced. A prefetcher reduces cache miss penalty by predicting future memory accesses on the basis of the current memory access. But this prefetching techniques leads to serious leakages in profiled cache-timing attacks. In this work our focus is on a form of hardware prefetching known as the *even-odd sequential* (EOS) prefetching. In this section we summarize the standard sequential prefetching algorithm followed by the Even-Odd Sequential prefetching.

Sequential Prefetching : The sequential prefetching algorithm [1] loads the subsequent memory block as presented in Algorithm 1. In the algorithm t_i is the current memory location addressed while t_{i+1} is the adjacent memory location where t is the base address of the table and i is the index. In [9, 3], this prefetching algorithm was shown to have a major influence in the information leakage in block ciphers. The cause for the leakage is due to timing observations or more precisely the cache miss profiles shows a deviation due to the behavior of this prefetching algorithm.

Even-Odd Prefetching: The algorithm triggers the prefetch for the adjacent memory block where location is determined by the address of the current access. If the memory block accessed is even then the next block is prefetched, while if the memory block accessed is odd then the previous block is prefetched. This prefetcher algorithm essentially doubles the size of the cache line for the memory block accessed. Details of the prefetching algorithm is present in Algorithm 2.

3. MATHEMATICAL MODEL FOR CACHE MEMORY ACCESSES

Leakage in the profiled cache-timing attack is entirely due to accesses to the look-up tables used in the cipher imple-


(a) Timing Profile for the Known Key

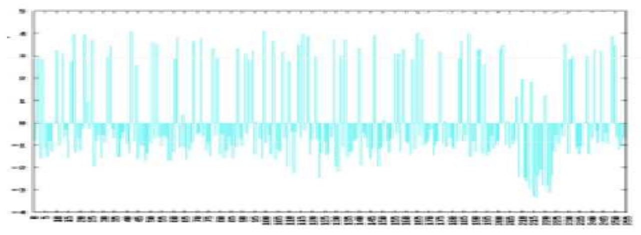

(b) Timing Profile for the Unknown Key

Figure 1: Timing Profiles for Known Key and Unknown Key for OpenSSL's AES Implementation on Intel Core 2 Duo. The x-axis has the possible values of the plaintext byte p_0 and the y-axis has the deviation of the encryption time from the average

mentation, thus we can analyze profiled cache-timing attacks in terms of the cache misses [13]. The execution time of the cipher implementation is directly proportional to the total number of cache hits and misses suffered by the implementation.

Consider a cipher which is implemented with a look-up table that occupies l memory blocks. Note that, the size of a memory block is equal to the cache line size. If the cache line has δ bytes then intuitively, the table exactly fits in $l \times \delta$ memory elements. Let us assume that during the cipher execution the table is accessed n_{max} number of times at random locations[3]. Since the accesses are random, a particular access to the lookup table results in a cache hit when any previous access resulted in an access to the same location in the table and has not been removed from the cache by the cache replacement policy. Moreover, recent micro-architectural features such as prefetchers prefetch blocks of memory in the cache memory, which has not yet been accessed by the implementation but has a probability of getting accessed in near future.

Thus, for an arbitrary access to a random location in lookup table can result in a cache hit or miss depending on two factors: whether the same location has been requested earlier and else whether some previous accessed memory request in turn prefetched the requested memory location. In this paper, we attempt to provide a formal analysis of cache profiling and leakage involved with a variant of sequential prefetcher named as Even-Odd Sequential prefetcher in the same lines of the modeling proposed in [9]. We perform the analysis in context to the profiled cache-timing attack scenario and we also claim that the underlying prefetching algorithm has a significant effect on the respective cache profiles. The steps involved in the formal analysis is as follows:

- Obtain the probability of a cache hit in the n^{th} access to the lookup-table, where $1 \leq n \leq n_{max}$.
- Obtain the conditional probability of a cache hit in the n^{th} memory access to the lookup-tables, with the EOS prefetcher.
- From the above steps, obtain the distribution of the number of cache misses. This distribution is Gaussian and therefore can be characterized by its mean and variance.
- Apply the Kullback-Leibler divergence to quantify the information leakage.

[3]The assumption of random accesses is justified as it is an essential requirement of block cipher constructions

The analysis is based on the assumption that no part of the table is present in the cache at the start of encryption.

3.1 Probability of a Cache Miss in the n^{th} Access in a Classical Cache

A *classical cache* is defined as a cache where the cache accesses are performed purely on request and are not affected by any prefetching algorithm. In order to have a cache hit in the n^{th} access to the table, there should be a collision with at-least one of the previous $n-1$ memory accesses. Let $A^C_{l,n}$ be a random variable that denotes the result of the n^{th} memory access to the table of size l in a system having a classical cache. $A^C_{l,n}$ can take values of either H or M respectively corresponding to a cache hit and a cache miss in the n^{th} memory access. The probability of obtaining a cache hit in the n^{th} access is given by [9],

$$\Pr[A^C_{l,n} = H] = \frac{1}{l^{n-1}} \sum_{i=0}^{n-2} \binom{n-1}{i} (l-1)^i$$

The probability of obtaining a cache miss in the n^{th} memory access is

$$\Pr[A^C_{l,n} = M] = 1 - \Pr[A^C_{l,n} = H]$$

3.2 Probability of a Cache Hit in the n^{th} Access in a Cache Supporting Prefetching

In a cache memory that supports hardware prefetching, a cache hit in the n^{th} memory access to a table may occur in two ways [9]. It may either be that the access collides with any of the previous memory accesses (the classical case) or the data have been prefetched. Let P denote the given prefetching strategy and $A^{C,P}_{l,n}$ the random variable denoting the result of the n^{th} access to the table of size l. The probability of obtaining a cache hit in the n^{th} access is

$$\begin{aligned}\Pr[A^{C,P}_{l,n} = H] = &\Pr[A^{C,P}_{l,n} = H \mid \mathit{collision}] \cdot \Pr[\mathit{collision}] \\ &+ \Pr[A^{C,P}_{l,n} = H \mid \overline{\mathit{collision}}] \cdot (1 - \Pr[\mathit{collision}])]\end{aligned}$$

For the case where a collision has occurred, the probability of a cache hit is exactly equal to the probability of cache hit in the classical case. For the case when no collision has occurred, a cache hit is obtained only if the data has been prefetched. Let $A^P_{l,n}$ denote a random variable which denotes probability of obtaining a cache hit in the n^{th} access exclusively due to prefetching. So the equation is rewritten as follows,

$$\Pr[A^{C,P}_{l,n} = H] = \Pr[A^C_{l,n} = H] + \Pr[A^P_{l,n} = H] \cdot (1 - \Pr[A^C_{l,n} = H])]$$

3.3 Conditional probability of the even-odd sequential prefetcher

In the attack proposed by Bernstein in [2], the correlation property over two timing profiles of a known key-known plaintext pair and an unknown key - known plaintext pair is used to determine the unknown key. So if a previous access is known to us, the probability of obtaining a cache hit gets affected since the requested memory block may have been already prefetched by any of the previous accesses. Thus we analyze the conditional probability of obtaining a cache hit conditioning on the previous occurrences of plaintext.

Let us assume that T_m is a random variable which denotes the block in the table accessed in the m^{th} access. The probability of hit of n^{th} access depends on the previous accesses, thus we aim to determine $\Pr[A^{EOSP}_{l,n} = H \mid T_m]$, where $A^{EOSP}_{l,n}$ is the random variable which denotes the cache event due to the memory element requested by the cipher at the the n^{th} access in presence of the Even-Odd Sequential prefetcher. Here we assume $m = 1$ thus conditioning on the first access. While we model the leakage for even-odd sequential prefetcher, it has also been observed that the conditional probability changes depending on the alignment of the tables in memory which is illustrated in the following discussion.

3.3.1 Effect of Table Alignment on probability distributions

The lookup table of the cipher implementation occupies l adjacent blocks in memory where each block contains same number of elements δ as that of a cache line. Thus the memory blocks of table lookup if indexed are l successive blocks alternatively having even and odd indices. As in this paper we model the leakage of the Even-odd Sequential prefetcher, the index of the accessed memory block either being even or odd, decides the memory block going to be prefetched by the prefetcher. Thus the alignment of table in memory does have an impact while analyzing the number of cache misses from table lookup.

The starting index of the lookup table of the cipher implementation may either be from an odd indexed memory location or from an even indexed location. Moreover, the number of memory blocks the table occupies will depend on the cipher specifications. Thus accordingly the lookup table may end in an odd indexed location or an even indexed location. There are clearly four cases: the starting index can be either even or odd, and depending on the length of the table the last index can again be even or odd, thus the analysis of each of the four cases requires separate attention.

Even-Odd Alignment: When the starting index of the table is even and the number of blocks in the table, l is also even. Thus the last index of the table is odd. During the cipher execution, if an even indexed block is accessed then the EOS prefetching algorithm will automatically prefetch the following odd indexed block(if not already in cache). Similarly, an access to the odd indexed block will result in a prefetch of its previous even index block. Thus in the even-odd alignment, each block in the table prefetches exactly one block of memory belonging to the same table. In other words, each block of the table can be prefetched by another block of the same table.

Odd-Odd Alignment: When the starting index of the table is odd and l is also odd. Thus the last block of the table has an odd index. Now, for this case, since the starting index of the table is odd, the memory block with even index responsible for prefetching the starting index is not a part of the lookup table. Thus the starting index cannot be prefetched by any memory block belonging to the table. The probability of cache misses of odd-odd alignment thus differs from the even-odd case. The remaining memory blocks apart from starting block in this odd-odd alignment, behaves exactly same as the blocks in even-odd alignment.

Odd-Even Alignment: When the starting index of the table is odd and l is also even. Thus the last block has index even. In this alignment, neither the starting odd indexed block of the table can be prefetched by any other block from the table, nor the last even indexed block can prefetch any block of the table. The remaining blocks apart from the first and last block has similar behavior as even-odd Alignment.

Even-Even Alignment: When the starting index of the table is even and l is also even. Thus the last block has index even. In this table alignment, only the last block of the table has no prefetch candidate inside the table. Apart from the last block, the probabilities remain same as the previous cases. Because of the fact that in the even-odd alignment all cache lines behave identically, we observe the cache hit-miss distribution does not depend on the previous cache accesses. We show formally next, this helps to make this configuration of the cipher tables not leak due to the hardware prefetcher and also compare the leakage to the other table alignments.

In the following analysis, we tackle the four cases separately as the probability equations for each of the cases will change as the table alignment changes.

3.3.2 Even-Odd Alignment

As the name suggests the table starts from an even location, length of the table is even and thus ends in an odd location. The probability of hit due to the prefetching is conditioned such that the first access is the last block of the table. So the probability of a cache hit given that the first access T_1 has occurred is calculated below.

The equation can be explained as all the memory blocks are prefetchable there are two cases: the probability that the n^{th} access is a hit due to the block being prefetched by the first access T_1 and the case where the block being prefetched by any other $n-2$ accesses other than the known first access T_1 and T_n. Let $EOSP$ be a function which returns the prefetched memory block ie. $EOSP(t_b) = t_{b+1}$ if t_b is even and $EOSP(t_b) = t_{b-1}$ if t_b is odd.

$$\begin{aligned}
Pr[A^{EOSP}_{l,n} = \mathcal{H} | T_1] = \\
&Pr[A^{EOSP}_{l,n} = \mathcal{H} | T_n = EOSP(T_1)] \cdot Pr[T_n = EOSP(T_1)] \\
&+ Pr[A^{EOSP}_{l,n} = \mathcal{H} | T_n \neq EOSP(T_1)] \cdot Pr[T_n \neq EOSP(T_1)]
\end{aligned}$$

There are two components in this equation.

- *When* $T_n = EOSP(T_1)$, it would certainly cause a cache hit. Also, since T_n cannot have a collision with T_1, it can only take $l-1$ different values and not l. Thus,

$$Pr[T_{l,n} = EOSP(T_1)] = \frac{1}{l-1}$$

- *When $T_n \neq EOSP(T_1)$.* This happens with probability $1-(1/(l-1))$. A hit in the n^{th} access occurs if and only if $T_n = EOSP(T_i)$ and $2 \le i \le n-1$. The probability with which this happens is given by the following equation.

$$\Pr[A_{l,n}^{EOSP} = H \mid T_n \neq EOSP(T_1)] = \frac{\alpha}{(l-2)(l-1)^{n-2}} \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$

,where α is the number of prefetchable blocks.

  - since $T_n \neq T_1$ and $T_n \neq EOSP(T_1)$. Thus, $\alpha = l-2$.

Thus combining two parts overall equation is written as,

$$\Pr[A_{l,n}^{EOSP} = H \mid T_1] = \frac{1}{(l-1)^{n-1}}[(l-1)^{n-2} + (l-2) \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}]$$

3.3.3 Odd-Odd Alignment

The table starts from an odd location and the table size is also odd, so ends in an odd location. Since the table starts from an odd location the first block of the table is prefetched by a block outside the table, thus is not prefetchable by the current table. Also, the first block being odd prefetches a block outside the table. Now, if the first access T_1 is conditioned then equation can be analyzed in two cases if $T_n = EOSP(T_1)$ and $T_n \neq EOSP(T_1)$

$$Pr[A_{l,n}^{EOSP} = \mathcal{H} | T_1] = Pr[A_{l,n}^{EOSP} = \mathcal{H} | T_n = EOSP(T_1)] \cdot Pr[T_n = EOSP(T_1)] + Pr[A_{l,n}^{EOSP} = \mathcal{H} | T_n \neq EOSP(T_1)] \cdot Pr[T_n \neq EOSP(T_1)]$$

There are two components in this equation.

- *When $T_n = EOSP(T_1)$*, it would certainly cause a cache hit. Also, since T_n cannot have a collision with T_1, it can only take $l-1$ different values and not l. Thus,

$$Pr[T_{l,n} = EOSP(T_1)] = \frac{1}{l-1}$$

- *When $T_n \neq EOSP(T_1)$.* This happens with probability $1-(1/(l-1))$. A hit in the n^{th} access occurs if and only if $T_n = EOSP(T_i)$ and $2 \le i \le n-1$. The probability with which this happens is given by the following equation.

$$\Pr[A_{l,n}^{EOSP} = H \mid T_n \neq EOSP(T_1)] = \frac{\alpha}{(l-2)(l-1)^{n-2}} \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$

,where α is the number of prefetchable blocks.

  - If $T_1 = t_1$ then $T_n \neq t_1$. Thus, $\alpha = l-1$.
  - If $T_1 \neq t_1$ then $T_n \neq T_1$, $T_n \neq EOSP(T_1)$ and $T_n \neq t_1$. Thus, $\alpha = l-3$.

3.3.4 Odd-Even Alignment

The table start from an odd location and have an even length and thus ends in an even location. Since the table starts from an odd location and ends in an even location the first block and the last block cannot be prefetched and the both of them prefetches a block outside the table. Thus the probability equation can be written as

$$\Pr[A_{l,n}^{EOSP} = H \mid T_1 = t_l \text{ or } T_1 = t_1] = \frac{(l-2)}{(l-1)^{n-1}} \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$

The above equation is explained in the following way as since $T_1 = t_l$ or t_1 and $T_n \neq EOSP(T_1)$, so the $n-2$ remaining accesses account for the summation and T_n can take $(l-2)$ values as $T_n \neq t_1$ and $T_n \neq t_l$.
When $T_1 \neq t_l$ and $T_1 \neq t_1$,

$$Pr[A_{l,n}^{EOSP} = \mathcal{H} | T_1 \neq t_l, T_1 \neq t_1] = Pr[A_{l,n}^{EOSP} = \mathcal{H} | T_n = EOSP(T_1)] \cdot Pr[T_n = EOSP(T_1)] + Pr[A_{l,n}^{EOSP} = \mathcal{H} | T_n \neq EOSP(T_1)] \cdot Pr[T_n \neq EOSP(T_1)]$$

There are two components in this equation.

- *When $T_n = EOSP(T_1)$*, it would certainly cause a cache hit. Also, since T_n cannot have a collision with T_1, it can only take $l-1$ different values and not l. Thus,

$$Pr[T_{l,n} = EOSP(T_1)] = \frac{1}{l-1}$$

- *When $T_n \neq EOSP(T_1)$.* This happens with probability $1-(1/(l-1))$. A hit in the n^{th} access occurs if and only if $T_n = EOSP(T_i)$ and $2 \le i \le n-1$. The probability with which this happens is given by the following equation.

$$\Pr[A_{l,n}^{EOSP} = H \mid T_n \neq EOSP(T_1), T_1 \neq t_l, T_1 \neq t_1] = \frac{\alpha}{(l-2)(l-1)^{n-2}} \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$

,where α is the number of prefetchable blocks.

  - If $T_1 \neq t_1$ then $T_n \neq T_1$, $T_n \neq EOSP(T_1)$, $T_n \neq t_l$(cannot be prefetched) and $T_n \neq t_1$(cannot be prefetched). Thus, $\alpha = l-4$.

3.3.5 Even-Even Alignment

For this case, the table starts from an even location, have an odd length and ends in an even location. So all the blocks in the table can be prefetched. But the last block prefetches a block outside the table. Thus the probability equation can be given as

$$\Pr[A_{l,n}^{EOSP} = H \mid T_1 = t_l] = \frac{1}{(l-1)^{n-2}} \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$

here, $T_1 = t_l$ and $T_n \neq EOSP(T_1)$, so the $(n-2)$ remaining accesses are responsible for the prefetching as in the previous cases. Since $T_n = t_l$ is possible here and $T_n \neq t_l$, so the $(l-1)$

blocks can be placed at location T_n When $T_1 \neq t_l$,

$$Pr[A^{EOSP}_{l,n} = \mathcal{H} | T_1 \neq t_l] =$$
$$Pr[A^{EOSP}_{l,n} = \mathcal{H} | T_n = EOSP(T_1)] \cdot Pr[T_n = EOSP(T_1)]$$
$$+ Pr[A^{EOSP}_{l,n} = \mathcal{H} | T_n \neq EOSP(T_1)] \cdot Pr[T_n \neq EOSP(T_1)]$$

There are two components in the following equation.

- *When* $T_n = EOSP(T_1)$, it would certainly cause a cache hit. Also, since T_n cannot have a collision with T_1, it can only take $l-1$ different values and not l. Thus,

$$Pr[T_{l,n} = EOSP(T_1)] = \frac{1}{l-1}$$

- *When* $T_n \neq EOSP(T_1)$. This happens with probability $1-(1/(l-1))$. A hit in the n^{th} access occurs if and only if $T_n = EOSP(T_i)$ and $2 \leq i \leq n-1$. The probability with which this happens is given by the following equation.

$$\Pr[A^{EOSP}_{l,n} = H \mid T_n \neq EOSP(T_1), T_1 \neq t_l] =$$
$$\frac{\alpha}{(l-2)(l-1)^{n-2}} \cdot \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$

  ,where α is the number of prefetchable blocks.

  - since $T_n \neq T_1$, $T_n \neq EOSP(T_1)$ and $T_n \neq t_l$(cannot be prefetched). Thus, $\alpha = l-3$.

This section formulates the probability distributions for the Even-Odd sequential Prefetcher and how it behaves when cache blocks are arranged in various table alignments. These probability distributions helps in the development of the cache miss distribution. In the next section using the Kullback-Leibler Divergence metric, we quantify the leakage for the various table alignments for cache blocks using the Even-odd Sequential Prefetcher.

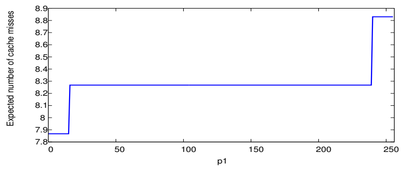


Figure 2: Cache miss distribution of sequential prefetcher

4. ESTIMATING CACHE MISS DISTRIBUTION AND ANALYZING LEAKAGE IN THE EOS PREFETCHER

In the previous section we analyzed the probability of cache accesses resulting in a hit with EOSP and this is eventually utilized to construct the cache miss distribution and also in quantifying information leakages using the Kullback-Leibler Divergence.

4.1 Cache Miss Distribution

A cipher implementation when executed repeatedly, the cache misses from these executions can be represented in a *cache miss distribution*, which is a normal distribution having a mean and a variance. In [9], the expected number of cache misses in the n^{th} access is calculated by,

$$E(A_{l,n}) = 0 \cdot (Pr[A_{l,n} = \mathcal{H}]) + 1 \cdot (Pr[A_{l,n} = \mathcal{M}])$$
$$= 1 - Pr[A_{l,n} = \mathcal{H}]$$

and the variance of cache misses in the n^{th} access is

$$V(A_{l,n}) = (1 - Pr[A_{l,n} = \mathcal{H}]) - (1 - Pr[A_{l,n} = \mathcal{H}])^2$$
$$= Pr[A_{l,n} = \mathcal{H}]^2 + Pr[A_{l,n} = \mathcal{H}]$$

The expectation of the number of cache misses after n memory accesses are given by the recurrence equation as,

$$E(M_n) = E(M_{n-1}) + E(A_{l,n})$$

and

$$V(M_n) = V(M_{n-1}) + V(A_{l,n}) + 2 \cdot Cov(M_{n-1}, A_{l,n})$$

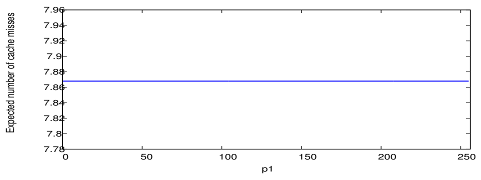
where M_n is the random variable denoting the number of cache misses after n memory accesses. In profiled cache-timing attack, whenever we attempt to predict a particular key k_i, the timing profile for the key k_i needs to be built. Since the timing information from a block cipher can be analyzed with the number of cache misses suffered by the execution, we are going to estimate the cache profile for k_i. Cache profile of k_i is the variation of the number of cache misses by varying input plaintext p_i. So for the key k_i, the cache miss distribution by varying the values of p_i gives the cache profile for that particular key k_i. Cache profiles for k_1 with the Even-Odd Sequential Prefetching styles with the size of table (in terms of number of memory blocks) $l = 16$, $n_{max} = 36$ and cache line size $\delta = 16$ for each of the four cases on the alignment of the tables are illustrated in Figure 3. Figure 3 suggests that there is no deviation in cache profile for the even-odd alignment as in Figure 3(a), while the remaining alignments illustrated in Figures 3(b),3(c),3(d) is having significant deviations. The correctness of our formal analysis and the respective simulation results are verified using a cache profiler in the following section.

4.2 Validation of the Cache Models

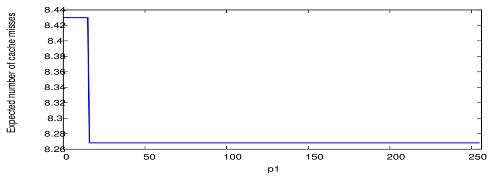
In this section we have checked the correctness of the mathematical formulations of cache models using a cache profiling tool *Cachegrind*[4]. The assumptions we formulated previously are provided by this tool platform. The source code of Cachegrind was patched to support the even-odd sequential algorithms described in Algorithms 2. The modified code of Cachegrind performs the even-odd sequential prefetching. The table sizes as discussed in Section 3 can either be even or odd. The cipher model used in the even sized tables uses parameters as $\Gamma = 1$, $l = 8$, $n_{max} = 8$. The odd sized tables uses parameters as $\Gamma = 1$, $l = 7$, $n_{max} = 8$. The alignments of the tables in each case can again start from an even or an odd location. The code is adjusted in each case accordingly. The cache modeled is 2-way set associative with $\delta = 32$ and of size $16KB$. The empirical distributions were obtained after sampling $10,000$ executions of the cipher.

Figure 4 shows the cache profiles for key byte k_1 obtained from Cachegrind and the predicted ones for the same cipher and same cache model with various table size and alignments. Figure 4(a) and Figure 4(d) has the same table sizes

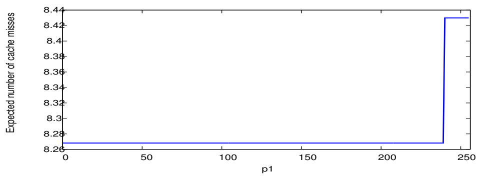
[4]http://valgrind.org/docs/manual/cg-manual.html



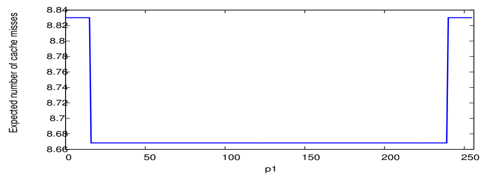
(a) EOSP with even-odd



(b) EOSP with odd-odd

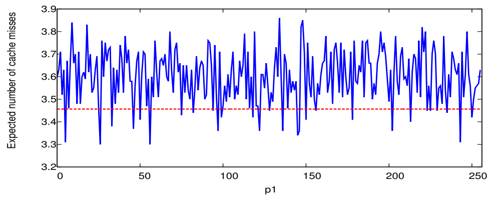


(c) EOSP with even-even

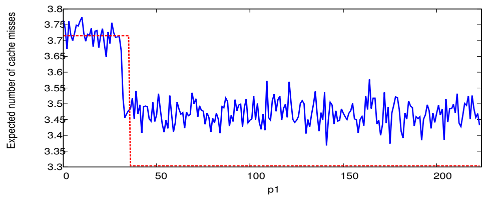


(d) EOSP with odd-even

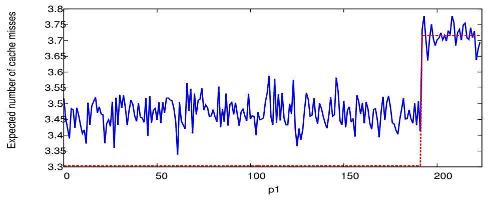
Figure 3: Cache Profiles for k_1 with Even-Odd Sequential Prefetching Styles for Different Table Alignments and $l = 16$, $n_{max} = 36$, $\delta = 16$ with $x-$axis having the conditioned value and y-axis the number of cache misses



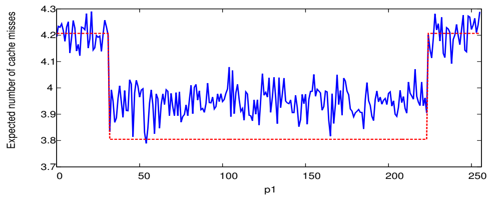
(a) Cache Misses with even-odd



(b) Cache Misses with odd-odd



(c) Cache Misses with even-even



(d) Cache Misses with odd-even

Figure 4: Predicted and Empirical Cache Profiles for k_1 for Cipher Model $\Gamma = 1$, $n_{max} = 8$

$l = 8$ but differs in their table alignments. While Figure 4(b) and Figure 4(c) have table size $l = 7$ again differing in their alignment. The expected number of cache misses in the predicted cache profile in Figure 3 and the empirical ones in Figure 4 follows the same pattern. In Figure 4, the empirical cache profile is plotted in *blue* lines while the predicted cache profile is plotted using the *red dotted* line using the same set of parameters as explained above and comparison of the cache profiles can be easily performed. It is clearly evident from the figures that the empirical cache profiles obtained from the cache profiler closely follows the predicted cache profile following our formal analysis.

5. INFORMATION LEAKAGE IN A TIMING PROFILE

In order to quantify the leakage due to hardware prefetchers in the context of profiled cache-timing attacks, the authors in [9] uses a metric to measure leakage in a timing profile: the symmetric version of Kullback-Leibler divergence [7]. Kullback-Leibler divergence is used as a measure of the difference between two probability distributions and in this paper we use the divergence in order to quantify the amount of *deviations* in the timing profile. Assuming a noise-free timing profile, every deviation from the average encryption time, results in information leakage. Therefore a profile with larger number of deviations would result in more information leakage.

The metric used is as follows: Equation 1 computes the symmetric KL divergence between two distribution $F_{k_{1,i}}$ and $F_{k_{1,i'}}$.

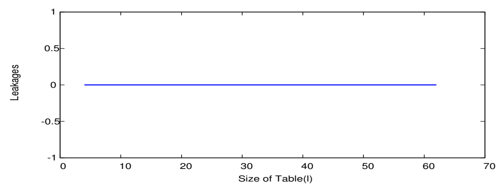
$$D(F_{k_{1,i}}, F_{k_{1,i'}}) = D(F_{k_{1,i}}||F_{k_{1,i'}}) + D(F_{k_{1,i'}}||F_{k_{1,i}}) \quad (1)$$

, where

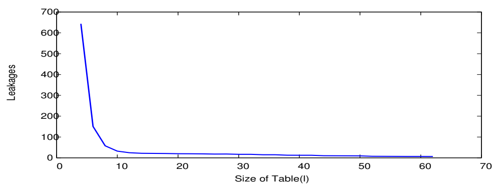
$$D(F_x||F_y) = \sum_j F_x(j) \log \frac{F_x(j)}{F_y(j)}$$

j ranges from minimum to maximum number of cache misses. The minimum number of cache misses is always 1 due to compulsory miss and the maximum can be either n or l(the greater between the two).

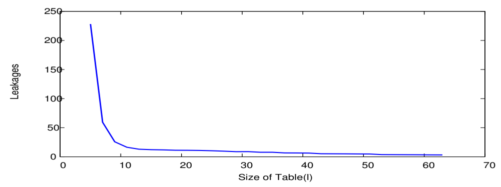
Total information leakage $(\mathfrak{D}(\mathsf{CP}(\mathsf{k}_1))$ for the cache profile for k_1, is the value of $D(F_{k_{1,i}}, F_{k_{1,i'}})$ and is added for every



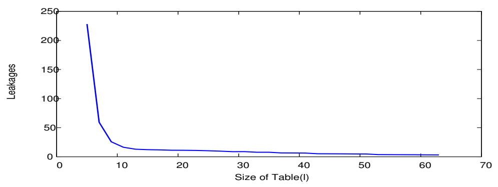
(a) Even-Odd Table alignment ($\Gamma = 1$)



(b) Odd-Even Table alignment ($\Gamma = 1$)



(c) Odd-Odd Table alignment ($\Gamma = 1$)



(d) Even-Even Table alignment ($\Gamma = 1$)

Figure 5: Leakage for k_1 with Various Table Alignments as the Table Size Increases ($n_{max} = 36$)

possible pair of distributions. Therefore,

$$\mathfrak{D}(\mathsf{CP}(\mathsf{k}_1)) = \sum_{\substack{\forall pairs\, of \\ i\, and\, i', i \neq i'}} D(F_{k_{1,i}}, F_{k_{1,i'}})$$

The Kullback Leibler divergence is zero when i and i' are on the same cache line as the shifts in the cache profile becomes uncertain. Thus higher the value of divergence higher is the leakage of information.

5.1 Analysis of Information Leakage

The leakages that we obtain by applying the metric on the cache profiles reveal that for the even-odd table alignment the information leaked is zero, because the line obtained is a straight line and any shift in the timing profile goes undetected in this case. On the contrary, the information leakages for the odd-even table alignment is the maximum among these four table alignments. The other two cases, namely, the odd-odd and the even-even table alignments show the same amount of information leakage which is higher than the even-odd alignment and lesser than the odd-even alignment.

As the size of the table(l) increases, the Kullback-Leibler divergence due to prefetching reduces. Figure 5 shows the variations in leakages for the various table alignments as the table size increases. The graphs plotting the information leakages for these four different table alignments as the size of table changes shows that for small table sizes the information leakage is huge and as the size increases the leakage reduces. Now, the information leakage for the even-odd table alignment is always zero and does not change though the table size changes. Also in Figure 5(d),5(c) the leakages in the even-even and odd-odd table alignments is lesser than the odd-even alignment. Thus if we can actually align the tables in even-odd position it will result in zero leakage or if we are having an odd-even alignment then try to have a large table size so as to make information leakage the minimum.

6. CONCLUSION

This paper considers a variant of sequential data prefetching algorithm named even-odd prefetching. The information leaked due to this prefetcher is formally analyzed for profiled cache-timing attacks. The analysis shows that the alignment of tables i.e., their starting and ending locations do have a great impact on the leakage of information. We have also observed that for a particular table alignment namely when the table start at an even memory block and ends at an odd memory block, there is no information leaked. This analysis shows that an implementer of a block cipher should consider the alignment of the tables in memory in order to eliminate leakage due to this prefetcher.

7. REFERENCES

[1] J. L. Baer. *Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors*. Cambridge University Press, 2010.

[2] D. J. Bernstein. Cache-timing Attacks on AES. Technical report, 2005.

[3] S. Bhattacharya, C. Rebeiro, and D. Mukhopadhyay. Hardware prefetchers leak: A revisit of SVF for cache-timing attacks. In *45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Workshops Proceedings, Vancouver, BC, Canada, December 1-5, 2012*, pages 17–23. IEEE Computer Society, 2012.

[4] E. Biham. A Fast New DES Implementation in Software. In E. Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[5] A. Canteaut, C. Lauradoux, and A. Seznec. Understanding Cache Attacks. Research Report RR-5881, INRIA, 2006.

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

[7] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

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

[9] C. Rebeiro and D. Mukhopadhyay. A formal analysis of prefetching in profiled cache-timing attacks on block ciphers. *IACR Cryptology ePrint Archive*, 2015:1191, 2015.

[10] C. Rebeiro, D. Mukhopadhyay, J. Takahashi, and T. Fukunaga. Cache Timing Attacks on CLEFIA. In B. Roy and N. Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2009.

[11] W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. R. Rao and B. Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[12] Shay Gueron. IntelÂő Advanced Encryption Standard (AES) Instructions Set (Rev : 3.0), 2010.

[13] K. Tiri, O. Aciiçmez, M. Neve, and F. Andersen. An analytical model for time-driven cache attacks. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2007.