A Formal Analysis of Prefetching in Profiled Cache-Timing Attacks on Block Ciphers

Chester Rebeiro and Debdeep Mukhopadhyay

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur 721302, India
{*chester,debdeep*}*@cse.iitkgp.ernet.in*

Abstract. Formally bounding side-channel leakage is important to bridge the gap between the theory and practice in cryptography. However, bounding side-channel leakages is difficult because leakage in a crypto-system could be from several sources. Moreover the amount of leakage from a source may vary depending on the implementation of the cipher and the form of attack. To formally analyze the security of a crypto-system against a form of attack, it is therefore essential to consider each source of leakage independently. This paper considers data prefetching, which is used in most modern day cache memories to reduce the miss penalty. To the best of our knowledge, we show for the first time that micro-architectural features like prefetching is a major source of leakage in profiled cache-timing attacks. We further quantify the leakage due to important data prefetching algorithms, namely sequential and arbitrary-stride prefetching. The analytical results, with supported experimentation, brings out interesting facts like the effect of placement of tables in memory and the cipher's implementation on the leakage in profiled cache-timing attacks.

Keywords. quantifying information leakage, formal modeling, cache memories, data prefetching, profiled cache-timing attacks

1 Introduction

A cache memory is a small high-speed memory that stores recently used data and instructions. The time required to access data present in the cache (a *cache hit*) is much lesser than when the data is not present in cache (a *cache miss*). In modern systems, the number of cache misses is reduced by *prefetching* data into the cache memory. However, not all cache misses are prevented and misses still occur. The differential behavior between a hit and a miss is manifested through side-channels such as timing, power, and electro-magnetic radiation.

In [21], it was prophesied that this non-uniform memory access behavior can be used to attack crypto-systems that are implemented with look-up tables stored in memory. The attack, which came to be known as *cache attacks*, was first formulated and simulated in [27]. Subsequently the attack was demonstrated on MISTY1 in [40] and extended to DES and 3-DES in [39]. AES was the next target with attacks in [6], [7], and [26]. Subsequently there have been several works that have analyzed, enhanced, and provided countermeasures for cache attacks.

Broadly, cache attacks can be classified into timing, trace, and access attacks. *Trace attacks* [2, 7, 14, 15, 29] require profiling of cache access patterns during encryption. Traces are generally obtained from power or electro-magnetic radiation

measurements and require sophisticated measuring equipment. *Access driven attacks* [25, 26, 28, 38] rely on spy processes to gain information about the cache access patterns of the cipher. *Timing attacks* such as [6, 8, 10, 30, 39, 40], on the other hand just require knowledge of the overall execution time of the cipher. Compared to trace attacks, timing measurements can easily be made. Moreover, unlike access attacks that are restricted to multi-user environments, timing attacks can target a wide range of platforms. These reasons, along with the threat that timing attacks may be mounted over the network [4, 9, 12], make timing attacks an important security concern.

Of all timing attacks, profiled cache-timing attacks are the most portable. These attacks do not require prior knowledge about the device architecture, however they assume a powerful adversary, who is capable of loading a key into the cryptographic device. Profiled attacks have two stages : a learning phase followed by an attack phase. In the *learning phase*, a template timing profile is generated for a known key. During the *attack phase*, a timing profile for the unknown key is generated. The two profiles are statistically correlated to yield a small set of candidate keys. Profiled cache-timing attacks were demonstrated on AES in [6] and CLEFIA in [30]. The reason for profiled cache-timing attacks to work is the variations in the encryption time [6, 10, 24]. These variations lead to information leakage that depends on the microprocessor architecture, operating system, and the cipher's implementation.

Formally bounding information leakage due to side-channels is important to analyze the security of a crypto-system. The first attempt to achieve this was in [23], where a model for physical observable cryptography was proposed. However the generality of the model made it impractical to implement. Simplifications using information theory and mutual information were introduced in [35] and [16]. Profiled side-channel attacks were also formally analyzed in [11, 31, 34]. The models introduced so far either generalized the leakage or approximated the leakage by hamming weights or distances. However, leakage is a function of several parameters and the magnitude of leakage of each parameter may differ. This is especially true for software implementations of ciphers, where leakage is influenced by numerous system specific parameters such as the branch prediction algorithm [3], hyperthreading [28], technology of the memory used [17], and cache architecture. Hamming weight or distance models will not apply in most of these cases. Depending on the type of cipher, its implementation, and the type of attack, the leakage contribution of each parameter may vary. For example, leakage from branch prediction is higher for public key ciphers compared to block cipher. For timing attacks on block ciphers implemented with look-up tables, the leakage from cache memories is the most dominant. To build provably secure crypto-systems, it is important to pinpoint the causes of leakage and quantify the amount of information leaked from each source. For timing attacks on block ciphers, a model for the leakage from cache memories was made in [37]. The work related leakage to the number of cache misses that occurred during the encryption. The cache memory modeled was that of a classical cache and did not consider the advanced techniques that reduce miss-penalty. Our work shows that the model proposed in [37] would fail for profiled cache-timing attacks. For such attacks, the leakages in cache memories are due to the micro-architectural acceleration techniques implemented in caches. Also, as stated earlier, each of these micro-architectural components have different amounts of leakage. The contributions of this work are as follows.

– We consider the leakage due to data prefetching in profiled cache-timing attacks. We show that the information leaked due to prefetching depends on several factors such as the size and number of look-up tables used in the cipher's implementation. We analyze the leakage of two prefetching algorithms : sequential and arbitrary-stride prefetching [42]. These algorithms or their variants are used in most modern day microprocessors.
– Unlike [37], where the mathematical models developed for cache misses were for classical cache memories, our models also consider data prefetching. We formally analyze the number of cache misses during an encryption in presence of the aforementioned prefetching algorithms.
– We propose a mathematical model to analyze the information leakage in a profiled cache-timing attack. The leakage, denoted $\mathfrak{D}$, is derived using the Kullback-Leibler divergence [22]. We use $\mathfrak{D}$ to study the effect of sequential and arbitrary-stride prefetching algorithms with different implementations of the cipher. To the best of our knowledge this is the first work which quantitatively analyzes the effect on security of a micro-architectural component of a computer.

The paper is organized as follows: Section 2 has the preliminaries about cache memories, block ciphers, and profiled cache-timing attacks. Section 3 lays the foundations and assumptions for the formal analysis. This section shows why encryption time can be analyzed in terms of the number of cache misses. Section 4 presents the mathematical models for the number of cache misses in presence of the two considered prefetching styles. Section 5 presents the model for analyzing profiled cache-timing attacks by using cache misses. The section also has empirical validations of the cache models developed. A formal model for profiled cache-timing attacks is presented in Section 6. An analysis of profiled cache-timing attacks for different implementation strategies is explained in Section 7 using the previously developed model. Other aspects influencing cache-timing attacks such as noise in measurements and leakages other than prefetching is presented in Section 8. The final section summarizes the work and presents potential future directions.

2 Preliminaries

In this section we first give a brief introduction about cache memories and prefetching. Then we describe a block cipher's structure from the cache attack point of view. We then describe formally the profiled cache-timing attack.

2.1 Cache Memories

Caches are high-speed memories, which are incorporated in computer architecture to reduce the memory access time. In systems that use cache memories, the main memory is divided into blocks referred to as *memory blocks*, while the cache is organized as *cache lines*. When a *cache miss* occurs, a block of memory gets loaded into a cache line. Unless evicted, all subsequent accesses to that memory block results in *cache hits*.

A cache miss has considerable overhead compared to a cache hit. On an Intel Core 2 Duo for example, it was found that a cache miss in the $L1$ data cache takes approximately 10 clock cycles more than a cache hit. To reduce this overhead, modern caches incorporate several techniques such as pipelining, parallelization, non-blocking, out-of-order loading, multiple-banks, critical word first,

Algorithm 1: *SP* : Sequential Prefetching Algorithm

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

Algorithm 2: *AP* : Arbitrary-stride Prefetching Algorithm

```
   Input: Address of instruction (I_A), Address of memory block accessed (t_i)
 1 begin
 2     prefetch_flag ← FALSE
 3     if first access to I_A then
 4         stride(I_A) ← 0  ;  state(I_A) ← 1
 5     else
 6         switch state(I_A) do
 7             case 1
 8                 state(I_A) ← 2
 9                 stride(I_A) ← t_i − lastaccess(I_A)
10             end
11             case 2
12                 if stride(I_A) = t_i − lastaccess(I_A) then
13                     prefetch_flag ← TRUE
14                 else
15                     stride(I_A) ← t_i − lastaccess(I_A)
16                 end
17             end
18         end
19         if prefetch_flag = TRUE and t_{i+stride(I_A)} is not present in cache then
20             prefetch t_{i+stride(I_A)}
21         end
22         lastaccess(I_A) ← t_i
23     end
24 end
```

early restart, and prefetching [19, 20]. Of all micro-optimizations, only prefetching works to reduce the number of cache misses. All other techniques reduce the overhead of the cache miss on the application's performance.

Prefetching : A prefetcher anticipates cache misses and fetches data from the memory before the processor requests for it. The prefetch is either invoked by the program (*software prefetching*) through explicit microprocessor instructions or gets triggered automatically (*hardware prefetching*). With respect to cache-attacks, software prefetching is of less concern as block cipher implementations do not generally use explicit prefetching instructions.

There are several techniques to decide 'when' and 'how' a hardware prefetch happens. Each technique has its own set of advantages and disadvantages and is generally suited to a particular class of applications. Broadly, the techniques are classified into two : *sequential prefetch* and *arbitrary-stride prefetch* [41, 42].

- *Sequential Prefetch (*SP*) :* A popular sequential prefetch algorithm is the *tagged prefetch* algorithm [41, 42]. The algorithm (described in Algorithm 1) triggers the prefetch for the adjacent memory block whenever it detects a cache miss or when a prefetch block is accessed for the first time. A variant of this algorithm, known as *prefetch-on-miss*, is used on some Intel platforms [18].
- *Arbitrary-stride Prefetch (*AP*):* [41, 42] This form of prefetching is also known as *data prefetch logic* and *hardware prefetching* on Intel platforms

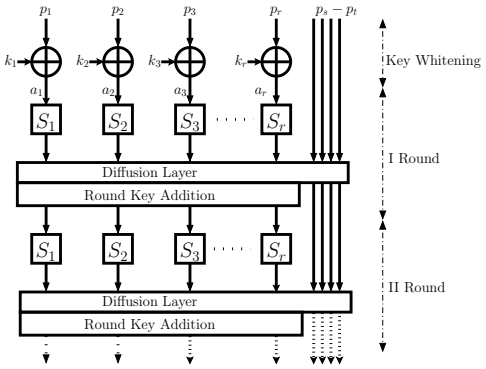


Fig. 1. Iterative Block Cipher Structure

[18] and are applicable to programs that use loops. In this prefetching algorithm, if a stride in the sequence of accesses from an instruction is detected, then a prefetch to the next memory address gets triggered. Although there are several forms of arbitrary-stride prefetching algorithms, we consider the *constant stride* variant [5] described in Algorithm 2. In this algorithm, if a stride is detected for accesses to memory locations t_i, t_{i+d}, and t_{i+2d} from the same instruction (I_A), then the memory block t_{i+3d} gets prefetched. The locations t_i, t_{i+d}, and t_{i+2d} form a *valid sequence*, while the *stride*, d, is the constant difference between the sequence. The stride can be either positive or negative, while the length of the sequence (denoted λ) is at-least 2.

2.2 Block Cipher Implementations

Figure 1 shows a typical structure of an iterated block cipher. The input (Λ) to each round of the cipher can be partitioned into two sets : inputs which undergo substitution (Λ_S) and those that don't (Λ_N). For SPN networks such as the AES specified in [13], $\Lambda_N = \phi$, while balanced and unbalanced Feistel networks have $|\Lambda_S| = |\Lambda_N|$ and $|\Lambda_S| \neq |\Lambda_N|$ respectively [32]. In all cases, the substitution transformations ($S_1, S_2, \cdots, S_r$) is preceded by a round key addition and followed by a diffusion layer.

Generalization of the Block Cipher Structure for Cache Attacks : Substitution is normally implemented by s-box look-up tables. These s-box accesses are key-dependent and are the main source of information leakage in cache attacks. The number of tables, size of tables (in terms of memory blocks), and number of accesses to a table vary from cipher to cipher and may also depend on the platform of implementation. This paper quantifies the effects of these parameters on the information leaked in profiled cache-timing attacks. Table 1 lists various parameters which can be used to characterize the memory accesses of a cipher. We define *cipher model* as a function of these parameters.

Table 2 gives the memory access characteristics for the block ciphers attacked by profiled cache-timing attacks. The AES implementation attacked in [6] considered the standard used in OpenSSL[1] library distributions. Although

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

Table 1. Memory Access Parameters of a Block Cipher

Γ	:	Number of tables used in the implementation
n_{max}	:	Number of key related look-ups per table during an encryption
γ	:	Number of rounds in the cipher
n_γ	:	Number of key related look-ups per table per round
δ	:	Number of table elements sharing a cache line
l	:	Number of memory blocks that are required to hold a table
t_1,t_2, $\cdots$, t_l	:	The contiguous memory block locations that hold the table

Table 2. Memory Access Model of Ciphers Attacked using the Profiled Cache Timing Attack

	$\mathbf{\Gamma}$	$\mathbf{n}_{max}$	γ	$\mathbf{n}_\gamma$	$\mathbf{l}$	δ
AES	4	36	9	4	16	16
CLEFIA	2	72	18	4	4	64

the number of rounds for AES is 10, the last round is not considered in the table as it uses a different look-up table. The implementation of CLEFIA [33] attacked in [30] was the reference implementation[2]. The attack on both AES and CLEFIA had a similar procedure. In the next part of this section we formally present the steps involved in a profiled cache-timing attack.

2.3 Profiled Cache-Timing Attacks

Consider an SPN cipher, $\mathcal{E}_\mathrm{K}$, having a secret key K and a structure similar to Figure 1. The memory access parameters for $\mathcal{E}_\mathrm{K}$ are as shown in Table 1. The secret key is chosen uniformly from the key set $\mathcal{K}^r$ and is split into r equal parts as $\mathrm{K} = (k_1|k_2|k_3|\cdots k_r)$. The block cipher function is $\mathcal{E}_\mathrm{K} : \mathcal{P}^r \rightarrow \mathcal{C}^r$. The plaintext $\mathrm{P} \in \mathcal{P}^r$ and ciphertext $\mathrm{C} \in \mathcal{C}^r$ are also split into r parts (for example $\mathrm{P} = (p_1|p_2|p_3|\cdots|p_r)$ as seen in Figure 1). Each part of k_i, p_i, and c_i are chosen from the set $\mathcal{K}$, $\mathcal{P}$, and $\mathcal{C}$ respectively and assumed to be of $\mathfrak{m}$ bits each.

Let a physical realization [23] of $\mathcal{E}_\mathrm{K}$, denoted as $\widetilde{\mathcal{E}_\mathrm{K}}$, be on a computer with cache memory. The side channel leakage of $\widetilde{\mathcal{E}_\mathrm{K}}$ is in the form of timing information (t). The goal of the side-channel adversary is to predict the key parts k_1, k_2, $\cdots$, k_r, from the timing information. We present the strategy for the profiled cache-timing attack on the key part k_1. The strategy described can be adopted for any of the other key parts.

To find k_1, the attack uses three phases : learning, attack, and finally analysis. In the *learning phase* a known key $\mathrm{K}^* = (k_1^*|k_2^*|\cdots|k_r^*)$ is considered and the timing profile for k_1^* is constructed as shown in Algorithm 3. The inner loop in the algorithm is generally executed around 2^{16} times.

For a given value of p_1^*, the algorithm invokes $\widetilde{\mathcal{E}_{\mathrm{K}^*}(\mathrm{P}^*)}$ several times and obtains the average encryption time (t) from the timing. The timing profile returned consists of $2^\mathfrak{m}$ average encryption times, which form the *timing profile for k_1^* for the known key* denoted $\mathsf{TP}(k_1^*)$.

During the *attack phase*, the key $\mathrm{K}^\# = (k_1^\#|k_2^\#|\cdots k_r^\#)$ is unknown. A timing profile ($\mathsf{TP}(k_1^\#)$) for $k_1^\#$ is constructed in a similar manner. Since the first

[2] http://www.sony.net/Products/cryptography/clefia

Algorithm 3: Timing Profile for k_1^*

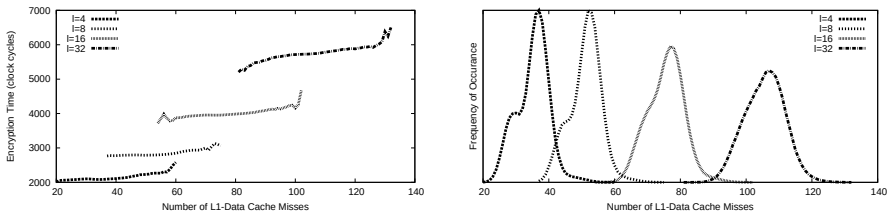
Output: The timing profile for k_1^*

begin
  forall $p_1^* \in \{0, 1, \cdots, 2^{\mathrm{m}} - 1\}$ **do**
    for *Large number of times* **do**
      $p_j^* \xleftarrow{\mathcal{R}} [0, 2^{\mathrm{m}})$, where $2 \leq j \leq r$;
      $\mathrm{P}^* = (p_1^* | p_2^* | p_3^* | \cdots | p_r^*)$;
      $(\mathrm{C}^*, \mathrm{t}) \leftarrow \widetilde{\mathcal{E}_{K^*}(\mathrm{P}^*)}$
    end
    Compute $AvgEncryptionTime_{k_1^*}[p_1^*]$ from the timing obtained
  end
  return $AvgEncryptionTime$
end

operation between p_1 and k_1 is ex-or (Figure 1), the average encryption times satisfy the EIS (equal images under different sub keys) property [31]. This means that for every p_1^* there exists a $p_1^{\#}$ such that $AvgEncryptionTime_{k_1^*}[p_1^*] \approx AvgEncryptionTime_{k_1^{\#}}[p_1^{\#} \oplus (k_1^* \oplus k_1^{\#})]$. Thus the $\mathsf{TP}(k_1^{\#})$ is a shifted version of $\mathsf{TP}(\mathsf{k}_1^*)$ and the amount of shift is equal to $(k_1^* \oplus k_1^{\#})$. During the *analysis* phase of the attack, this shift is extracted from the two timing profiles to reveal the unknown key part $k_1^{\#}$. The aim of this paper is to estimate the amount of information leaked in the timing profiles due to prefetching in cache memories.

3 Foundations and Assumptions

Profiled cache-timing attacks rely on the differences in encryption time of a cipher's execution. This paper analyzes these differences in terms of the number of cache misses that occur during the encryption. In this section we first empirically justify the fact that encryption time can in fact be analyzed from cache misses for various cipher models. Further, the section also states our assumptions about profiled cache-timing attacks. These assumptions are required in order to formally model the attack.



(a) Encryption Time (b) Distribution of Cache Misses

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

Relating Cache Misses and Timing : Figure 2(a) shows the variation in the encryption time with the number of cache misses [3] for SPN cipher models (Figure 1) with different table sizes. All other parameters of the cipher models are identical with values $n_{max} = 36$, $\Gamma = 4$, $\gamma = 9$, $n_\gamma = 4$, $\delta = 16$. From the figure, there is a linear relation between the execution time and the number of cache misses for all cipher models. Therefore the cipher's encryption time can be analyzed by the number of cache misses. Figure 2(b) shows the frequency distribution of the number of cache misses. The distribution is Gaussian irrespective of the table size.

Assumptions about the Platform and the Attack : The formal models developed in this work relies on the following assumptions on the attack and the attack platform.

- *The cache does not contain any information about the program at the start of encryption.* This assumption is valid as operations done after an encryption, such as transmission or storage of the ciphertext is highly likely to overwrite the cache contents. Thus each new encryption would start with a clean cache [6, 24].
- We assume that *during an encryption, data loaded into the cache is never flushed out.* This means that the execution of the cipher is independent of other processes running concurrently in the system. Also, there are no conflicts for the cache from within the encryption process. While this assumption holds for cipher models with tables much smaller than the size of the cache, for larger tables there would be noise added to the timing due to conflict misses. This noise makes it more difficult to attack.
- *The s-box accesses that occur during the execution of a cipher are at random locations and are independent of each other.* This assumption is a basic requirement of block ciphers.
- In profiled cache-timing attacks, the difference in cache misses between encryptions is of concern. We assume that *this difference in cache misses is only due to the accesses to the s-box tables used in the implementation.* Also, we assume that the table accesses are not influenced by any other part of the cipher's execution.

For caches that support prefetching, the effectiveness of prefetching depends on several system parameters such as the number of processing cores in the system, system load, etc. In order to formally analyze the profiled cache-timing attacks with prefetching we make a few assumptions. These assumptions are optimistic, that is, in reality, systems incorporating the described prefetching schemes would be less effective. The assumptions made are as follows:

- In practice, a trigger by the prefetcher does not imply that the memory block would *always* get loaded into cache. The actual availability of the prefetched block depends on several other factors such as the availability of system's resources, work-load of the processor etc. For example, in multi-core systems,

[3] The experiments were done on a $2.8GHz$ Intel Core 2 Duo machine with 32 KByte $L1$ data cache. The measurements were made using Intel's performance monitoring events and Linux's perfmon library. Since the figures show actual hardware measurements, the number of cache misses is higher than what would otherwise be expected. This is because of the spurious cache miss events that get counted during the measurements.

the L2 cache is a shared resource, therefore prefetching will not occur unless the L2 cache is available. In the model of the attack developed, however, it is assumed that *a prefetch trigger would always fetch a block of memory into cache*.
- In practice, the number of instructions the prefetcher can track is limited by the size of the table [5] which tracks memory access streams. However we assume that there is no such limit, and the prefetcher is able to track access patterns in infinitely many streams.

It may be noted that the assumptions made are realistic, as later we show that the theory developed based on them match the experimental results.

4 Mathematical Models for Cache Memory Accesses

In this section we develop mathematical models for the number of cache misses that occur during the execution of a cipher. The cipher is viewed as having Γ tables occupying l blocks each. Each table is accessed randomly n_{max} times during the execution. Various cache models are considered, starting from the classical cache memories to models which support sequential and arbitrary-stride prefetching.

Probability of a Cache Miss in a Classical Cache : In the architecture of a classical cache, a cache-miss results in a single block of data getting loaded into the cache. All subsequent accesses to any location within that memory block results in cache hits (ie. collisions). Consider a table $\mathcal{T}_1$, of size l, present at the contiguous memory locations t_1 to t_l and is accessed at n_{max} random locations. We determine the probability that the n^{th} access to the table results in a cache hit, where $1 \leq n \leq n_{max}$. Let $A^C_{l,n}$ be a random variable which denotes the outcome of the n^{th} memory access. $A^C_{l,n}$ takes the value $\mathcal{H}$ when a cache hit occurs and $\mathcal{M}$ when a cache miss occurs.

Theorem 1. *The probability that the n^{th} random access to table $\mathcal{T}_1$ results in a cache hit is given by*

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

Proof. Let the n^{th} access be to the memory location t_b (where $1 \leq b \leq l$). For $A^C_{l,n}$ to be a cache hit, t_b should have been accessed at-least once in the previous $n-1$ memory accesses. There are l^{n-1} different ways in which the previous $n-1$ memory accesses can be done. Out of these, the number of ways in which t_b gets accessed exactly j times is $\binom{n-1}{j}(l-1)^{n-1-j}$, and j varies from 1 to $n-1$. Theorem 1 follows. □

Corollary 1. *The probability that the n^{th} access to table $\mathcal{T}_1$ results in a cache miss is given by:*

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

4.1 Probabilistic Models for Cache Memories with Prefetching

In a cache memory that supports prefetching, a cache hit can occur either if the memory block being accessed collides with a previous access (the classical case) or has been prefetched earlier (the prefetching case). Consider a cache having some prefetching strategy P. Let the random variable $A_{l,n}^{C,P}$ denote the result of the n^{th} access to table $\mathcal{T}_1$. Then,

$$\begin{aligned} Pr[A_{l,n}^{C,P} = \mathcal{H}] = Pr[A_{l,n}^{C,P} = \mathcal{H}|collision] \cdot Pr[collision] \\ + Pr[A_{l,n}^{C,P} = \mathcal{H}|\overline{collision}] \cdot (1 - Pr[collision]) \end{aligned} \tag{2}$$

The probability of a collision is given by Equation 1 and would certainly result in a cache hit. The probability $Pr[A_{l,n}^{C,P} = \mathcal{H}|\overline{collision}]$ in Equation 2 depends completely on the prefetching strategy used. Let $A_{l,n}^{P}$ denote a random variable which takes the value $\mathcal{H}$ if and only if a cache hit in the n^{th} access occurs exclusively due to prefetching. Equation 2 can be rewritten as

$$Pr[A_{l,n}^{C,P} = \mathcal{H}] = Pr[A_{l,n}^{C} = \mathcal{H}] + Pr[A_{l,n}^{P} = \mathcal{H}] \cdot (1 - Pr[A_{l,n}^{C} = \mathcal{H}]) \tag{3}$$

In this section we analyze the probabilities of a cache hit with two prefetching strategies : sequential prefetching and arbitrary-stride prefetching [5]. These prefetching styles are common in modern processors.

Probability of a Cache Miss with Sequential Prefetching : For the prefetching algorithm described in Algorithm 1 and under the assumptions made in Section 3, the following observations can be made.

1. The first memory block in the table cannot be prefetched.
2. The last memory block in the table prefetches a block outside the boundary of the table.

These observations have an effect on the cache miss probabilities as will be seen in this section. Let $A_{l,n}^{S}$ be the random variable which takes the value $\mathcal{H}$ if and only if a cache hit in the n^{th} access to Table $\mathcal{T}_1$ occurs exclusively due to sequential prefetching. In Equation 3, P is replaced with S.

Theorem 2. *For the table $\mathcal{T}_1$, in a cache supporting sequential prefetching,*

$$Pr[A_{l,n}^{S} = \mathcal{H}] = \frac{l-1}{l} \cdot \frac{1}{(l-1)^{n-1}} \sum_{i=0}^{n-2} \binom{n-1}{i} (l-2)^i \tag{4}$$

Proof. If $\mathcal{T}_1$ is the only table used in the cipher, then all blocks in the table are prefetchable except the first block (ie. t_1). Thus the fraction of prefetchable blocks is $(l-1)/l$.

Assume n^{th} access is to memory block t_b. Since a collision is not allowed, t_b cannot occur in the previous $n-1$ memory accesses. Therefore, the only way to obtain a hit is when the previous block, ie. t_{b-1}, gets accessed at-least once. Theorem 2 follows by counting in a similar manner as in Theorem 1. □

Probability of a Cache Miss with Arbitrary-Stride Prefetching : Let $A_{l,n}^{A}$ be a random variable, which takes the value $\mathcal{H}$ if and only if a cache hit in the n^{th} access to table $\mathcal{T}_1$ occurs exclusively due to the arbitrary-stride prefetching described in Algorithm 2. In Equation 3, P is replaced with A.

We see that the probability with which $A^A_{l,n}$ takes the value $\mathcal{H}$ depends on the number of valid sequences occurring in the previous $n-1$ memory accesses. The valid sequences can have a positive or negative stride, which we respectively call *increasing sequence* and *decreasing sequence*. The following lemmas present different properties of the sequences.

Lemma 1. *Let $\lambda \geq 2$ be the minimum length of a valid sequence and t_b ($1 \leq b \leq l$) be the memory block in table $\mathcal{T}_1$ that gets accessed in the n^{th} memory access (where $n > \lambda$). The number of valid increasing sequences for t_b is*

$$S^+_b = \lceil (b-\lambda)/\lambda \rceil$$

Proof. Assuming that only increasing sequences are possible, a prefetch of t_b requires a sequence of at-least λ accesses that have a constant stride. This sequence can take values from t_1 to t_{b-1}. The maximum stride possible is thus $d_{max} = \lceil (b-\lambda)/\lambda \rceil$. All strides between 1 to d_{max} (both inclusive) form exactly one increasing sequence for t_b. Thus $S^+_b = d_{max}$. □

As an example, consider $b = 10$ and $\lambda = 3$, then the valid increasing sequences that can prefetch t_{10} are (t_7, t_8, t_9), (t_4, t_6, t_8), and (t_1, t_4, t_7), thus $S^+_b = 3$.

Lemma 2. *For the l sized table $\mathcal{T}_1$, the number of decreasing sequences for the memory block t_b is $S^-_b = S^+_{(l-b+1)}$.*

Proof. For every valid decreasing sequence for t_b of the form t_{b_1}, t_{b_2}, $\cdots$, t_{b_λ}, there exists an increasing sequence t_{l-b_1+1}, t_{l-b_2+1}, $\cdots$, $t_{l-b_\lambda+1}$ for the memory block t_{l-b+1}. □

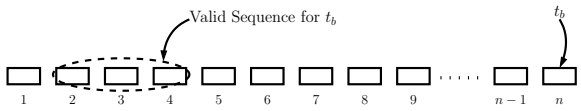
Continuing the previous example, if $l = 16$, then there are 3 valid decreasing sequences for t_7 because $S^-_7 = S^+_{10}$. These sequences are (t_{10}, t_9, t_8), (t_{13}, t_{11}, t_9), and (t_{16}, t_{13}, t_{10}).

Lemma 3. *If t_{b_i} ($1 \leq i \leq \lambda$) is the i^{th} element of a valid sequence for t_b, then for the pair t_b and t_{b_i} at most one sequence is possible. Moreover, $t_b = t_{b_i}$ mod $(\lambda + 1 - i)$.*

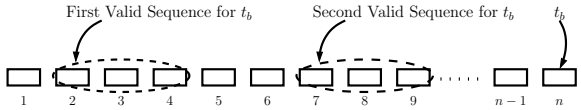
Proof. First Part : Assume, that two sequences are possible with t_{b_i} having strides d_1 and d_2 respectively. Then, $t_b = (\lambda+1-i)d_1 + t_{b_i} = (\lambda+1-i)d_2 + t_{b_i}$ Thus $d_1 = d_2$, and the two sequences are equal.

Second Part : If d is the stride of the sequence for t_b containing t_{b_i}, then $|t_b - t_{b_i}| = d(\lambda + 1 - i)$. Thus $t_b = t_{b_i} \bmod (\lambda + 1 - i)$. □

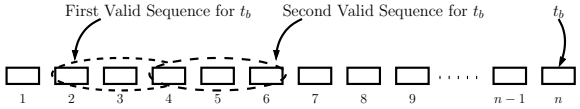
In order to prefetch t_b a valid sequence (either increasing or decreasing) for t_b should appear at-least once in $(n-1)$ memory accesses. Figure 3(a) shows a valid sequence appearing in the 2^{nd}, 3^{rd}, and 4^{th} memory accesses, which would prefetch t_b. Multiple valid sequences for t_b can also appear as shown in Figure 3(b). The sequences may also overlap as seen in Figure 3(c) forming a *chain* of sequences. For example, for t_{10} and $\lambda = 3$, a chain of length 1 is possible by the cascade of the sequences (t_1, t_4, t_7) and (t_7, t_8, t_9). This is a chain of length 1 because there is one overlap region. In a similar manner there can be chains of length greater than 1. The following lemma shows how many chains of length i are possible.



(a) Exactly one Valid Sequence is Present



(b) Two Valid Sequences are Present



(c) A Chain of Length 1

Fig. 3. Arbitrary Stride Prefetch of t_b Before the n^{th} Memory Access to Table $\mathcal{T}_1$ (for $\lambda = 3$)

Lemma 4. *For t_b, the number of chains of length i that are possible is*

$$chain(b, \lambda, i, n) = \begin{cases} 0 & \text{if } n < \lambda + (\lambda - 1)i \\ \lfloor \frac{S_b^+}{\lambda^i} \rfloor + \lfloor \frac{S_b^-}{\lambda^i} \rfloor & \text{otherwise} \end{cases}$$

Also, $i < max(\lfloor \log_\lambda S^+ \rfloor, \lfloor \log_\lambda S^- \rfloor)$.

Proof. A chain of length i will have $\lambda + (\lambda - i)$ elements. If the number of accesses made, n, is less than $\lambda + (\lambda - i)$, then no chains are possible. For any two overlapping sequences $(t_{b_1}^{(1)}, t_{b_2}^{(1)}, \cdots, t_{b_\lambda}^{(1)})$ and $(t_{b_1}^{(2)}, t_{b_2}^{(2)}, \cdots, t_{b_\lambda}^{(2)})$, without loss of generality $t_{b_1}^{(1)} = t_{b_\lambda}^{(2)}$ and no other elements are common. This means that an overlap can occur in exactly one location. This follows because if two sequences overlap in more than one location then the strides would be equal. If the strides of the cascaded sequences are d_1 and d_2 respectively,

$$\begin{aligned} t_{b_1}^{(1)} &= t_{b_\lambda}^{(2)} \\ t_b - d_1(\lambda) &= t_b - d_2 \\ d_1\lambda &= d_2 \end{aligned}$$

If a third sequence of stride d_3 is to be added; making a chain of length 2, then $d_2\lambda = d_3$, thus $d_1\lambda^2 = d_3$. In a similar manner for i cascades,

$$d_1\lambda^i = d_{i+1} \tag{5}$$

Since a stride can take values between 1 and d_{max} (both inclusive), for the chain of length i, Equation 5 would hold for $\lfloor d_{max}/\lambda^i \rfloor$ times. Moreover $d_{max} = S_b^+$,

therefore there are $\lfloor S_b^+/\lambda^i \rfloor$ possible chains of length i comprising of increasing sequences and the maximum value of i is $\lfloor log_\lambda S_b^+ \rfloor$.

An increasing sequence can never overlap with a decreasing sequence because all elements in an increasing sequence are less than t_b while all elements in a decreasing sequence are greater than t_b. Thus increasing and decreasing sequences have to be considered independently. The lemma follows. □

The number of ways to prefetch t_b before the n^{th} memory access can be determined by counting the number possible $n-1$ memory accesses without a valid sequence for t_b. If $V_{b,n-1}$ is the number of $n-1$ memory accesses which have at-least one valid sequence for t_b and $I_{b,n-1}$ is the number of $n-1$ memory accesses which have no valid sequence for t_b then

$$V_{b,n-1} = (l-1)^{n-1} - I_{b,n-1}$$

Lemma 5. *For $\lambda = 2$ and the $(n+1)^{th}$ memory access is to t_b, the following recurrence computes the value of $I_{b,n}$: $I_{b,0} = 1$, $I_{b,1} = (l-1)$, $I_{b,2} = (l-1)^2 - (S_b^+ + S_b^-)$, and for $n > 2$,*

$$I_{b,n} = (l-1)I_{b,n-1} - (S_b^+ + S_b^-)I_{b,n-2} + \sum_{i=1}^{max(\lfloor \log_2 S_b^+ \rfloor, \lfloor \log_2 S_b^- \rfloor)} (-1)^{i+1} \cdot I_{b,n-2-i} \cdot chain(b,2,i,n)$$

Proof. If $n = 0$ implies the first access is to t_b, therefore $I_{b,0} = 1$. For $n = 1$, all memory accesses to table $\mathcal{T}_1$ are possible except the one which causes a collision. Therefore $I_{b,1} = (l-1)$. For $n = 2$, there are $(l-1)$ possible first memory accesses and another $(l-1)$ possible second memory accesses. $I_{b,2}$ comprises of all those pairs of accesses, which does not prefetch t_b. That is, the first and second memory accesses do not form a sequence which leads to t_b. There are $(S_b^+ + S_b^-)$ ways such sequences can be formed, these are eliminated from $(l-1)^2$.

For $n > 2$, $I_{b,n}$ is computed applying inclusion-exclusion. First, we include all possible memory access sequences till $(n-1)$ which does not prefetch t_b, namely $I_{b,n-1}$, along with all possible ways to access the n^{th} memory access, namely $(l-1)$. However, here we have included all possible memory accesses till $(n-2)$ which does not prefetch t_b, and the $(n-1)^{th}$ and the n^{th} memory access could form a sequence. Hence the total number of cases which needs to be excluded is $I_{b,n-1}(S_b^+ + S_b^-)$. But, while excluding this we have considered accesses, which, till $(n-3)^{rd}$ memory access did not prefetch t_b, and the $(n-2)^{nd}$, $(n-1)^{st}$, and n^{th} memory access may form a chain which prefetches t_b. This counts to $I_{b,n-3}chain(b,2,1,n)$. Likewise we continue till the maximum possible length of chain. □

In a similar manner $I_{b,n}$, when $\lambda = 3$, can be obtained as shown in the following recurrence: $I_{b,0} = 1$, $I_{b,1} = (l-1)$, $I_{b,2} = (l-1)^2$, $I_{b,3} = (l-1)^3 - (S_b^+ + S_b^-)$ and

$$I_{b,n} = (l-1)I_{b,n-1} - (S_b^+ + S_b^-)I_{b,n-3} + \sum_{i=1}^{max(\lfloor \log_3 S_b^+ \rfloor, \lfloor \log_3 S_b^- \rfloor)} (-1)^{i+1} \cdot I_{b,n-4-i} \cdot chain(b,3,i,n) \text{ when } n > 3$$

Theorem 3. *The probability that the random variable $A^A_{l,n}$ takes the value $\mathcal{H}$ in the n^{th} access is given by*

$$Pr[A^A_{l,n} = \mathcal{H}] = \frac{1}{l(l-1)^{n-1}} \sum_{b=1}^{l} V_{b,n-1} \tag{6}$$

Proof. The n^{th} memory access could be to l memory blocks. Each of these memory blocks can be prefetched in $V_{b,n-1}$ ways. Since, prefetching requires no collision, therefore there are $(l-1)^{n-1}$ possible ways to do the first $n-1$ memory accesses. □

4.2 Conditional Probability Models for Cache Accesses

The probability of a cache miss in the n^{th} memory access is altered if one of the previous accesses is known. Let T_m be a random variable that denotes the memory block that gets accessed in the m^{th} access. T_m can take values t_1 to t_l. In this part of the section we develop models to determine the probability of a cache miss in the n^{th} memory access given T_m.

In the classical model of the cache, the probability of having a cache hit in the n^{th} access is independent of the condition T_m $(m < n)$. Therefore

$$Pr[A^C_{l,n} = \mathcal{H} \big| T_m] = Pr[A^C_{l,n} = \mathcal{H}]$$

Thus, for cache memories with some prefetching policy P, the conditional probability for a hit in the n^{th} access can be represented in a similar way as in Equation 3.

$$\begin{aligned} Pr[A^{C,P}_{l,n} = \mathcal{H} \big| T_m] =& Pr[A^{C,P}_{l,n} = \mathcal{H} | collision, T_m] \cdot Pr[collision | T_m] \\ &+ Pr[A^{C,P}_{l,n} = \mathcal{H} | \overline{collision}, T_m] \cdot (1 - Pr[collision | T_m]) \\ =& Pr[A^C_{l,n} = \mathcal{H}] + Pr[A^P_{l,n} = \mathcal{H} \big| T_m] \cdot (1 - Pr[A^C_{l,n} = \mathcal{H}]) \end{aligned} \tag{7}$$

As in the previous part of this section, we consider the sequential prefetching and arbitrary-stride prefetching algorithms to determine $Pr[A^P_{l,n} = \mathcal{H} \big| T_m]$ for the l sized table $\mathcal{T}_1$. Our equations are for $m = 1$. Similar techniques can be adopted for $m > 1$.

Conditional Probability of a Cache Miss with Sequential Prefetching : Let SP be a function which returns the prefetched memory block (ie. $SP(t_b) = t_{b+1}$ for $1 \leq b \leq l$). For sequential prefetching $Pr[A^S_{l,n} = \mathcal{H} \big| T_1]$ varies depending on the value of T_1. The following discussion explain this phenomenon.

When $T_1 = t_l$,

$$Pr[A^S_{l,n} = \mathcal{H} \big| T_1 = t_l] = \frac{(l-2)}{(l-1)^{n-1}} \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i} \tag{8}$$

For $SP(t_l)$, the block prefetched is outside the boundary of the table and has no effect on the probability. Therefore, $A^S_{l,n} = \mathcal{H}$ only if the required prefetch occurs in the remaining $n-2$ memory accesses. If $T_n = SP(t_b)$, then t_b should occur at-least once in the $n-2$ accesses. This accounts for the summation in Equation 8. Further, T_n can be prefetched only by $l-2$ different values since

$T_n \neq t_l$ (as this would cause a collision with T_1) and $T_n \neq t_1$ (as t_1 cannot be prefetched).

When $T_1 \neq t_l$,

$$Pr[A^S_{l,n} = \mathcal{H} \big| T_1 \neq t_l] = Pr[A^S_{l,n} = \mathcal{H} | T_n = SP(T_1)] \cdot Pr[T_n = SP(T_1)] + Pr[A^S_{l,n} = \mathcal{H} | T_n \neq SP(T_1)] \cdot Pr[T_n \neq SP(T_1)] \quad (9)$$

There are two components in Equation 9.

- *When* $T_n = SP(T_1)$, 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,
$$Pr[T_{l,n} = SP(T_1)] = \frac{1}{l-1}$$
- *When* $T_n \neq SP(T_1)$. This happens with probability $1 - 1/(l-1)$. A hit in the n^{th} access occurs if and only if $T_n = SP(T_i)$ and $2 \leq i \leq n-1$. The probability with which this happens is given by the following Equation.
$$Pr[A^S_n = \mathcal{H} \big| T_n \neq SP(T_1), T_1 \neq t_l] = \frac{\alpha}{(l-2)(l-1)^{n-2}} \sum_{i=1}^{n-2} \binom{n-2}{i} (l-2)^{n-2-i}$$
,where α is the number of prefetchable blocks.
  - If $T_1 = t_1$ then $T_n \neq t_1$ and $T_n \neq t_2$. Thus, $\alpha = l-2$.
  - If $T_1 \neq t_1$ then $T_n \neq T_1$, $T_n \neq SP(T_1)$, and $T_n \neq t_1$. Thus, $\alpha = l-3$.

Conditional Probability of a Cache Miss with Arbitrary-Stride Prefetching : In a cache supporting arbitrary-stride prefetching, a known value of T_m restricts the number of valid sequences, thus altering the probability of a cache miss in the n^{th} memory access ($m < n$). For example if $\lambda = 2$ and $T_3 = t_{10}$, then from lemmas 1 and 2, the number of valid increasing sequences for t_{10} is 4. They are (t_2, t_6), (t_4, t_7), (t_6, t_8), and (t_8, t_9). However, if T_1 is fixed to t_2 then only one of these sequences is possible thus reducing the probability that t_{10} gets prefetched. We now determine the probability of a hit due to prefetching when $m = 1$. This is given by,

$$Pr[A^A_{l,n} = \mathcal{H} | T_1] = \frac{1}{l-1} \sum_{b \in \{1,2,3,\cdots,l\} \text{ and } t_b \neq T_1} Pr[A^A_{l,n} = \mathcal{H} | T_1 \text{ and } T_n = t_b]$$

Depending on the value taken by T_1, the probability of prefetching t_b in the n^{th} memory access would vary.

- If $T_1 = t_a$ and $t_a \neq t_b \bmod \lambda$ then by Lemma 3 no valid sequence can be present at T_1. Thus, t_b can be prefetched only due to a valid sequence present in the remaining $n-2$ memory accesses. Therefore,
$$Pr[A^A_{l,n} = \mathcal{H} | T_1 = t_a, T_n = t_b \text{ and } t_a \neq t_b mod \lambda] = \frac{V_{b,n-2}}{(l-1)^{n-2}}$$
- If $T_1 = t_a$ and $t_a = t_b \bmod \lambda$ then at-most one valid sequence may be present at T_1 (by Lemma 3). Consequently, the number of chains that can be formed for b is reduced. For example, for $\lambda = 2$ and $T_4 = t_{10}$, the number of chains of length 1 that can be formed is 2, they are $((t_2, t_6)$ and $(t_6, t_8))$ and $((t_6, t_8)$

and (t_8, t_9)). If $T_1 = t_2$, then only one of these chains is possible. For the conditional case, we call the number of chains for b as $chain'$. The number of chains of length i is given in the following equation,

$$chain'(b, a, \lambda, i, n) = \begin{cases} chain(b, \lambda, i, n) & \text{if } n > \lambda + (\lambda - 1)i \\ 1 & \text{if } n = \lambda + (\lambda - 1)i \text{ and } d|\lambda^i \\ 0 & \text{otherwise} \end{cases}$$

In the equation, d is the stride for the sequence at T_1; $(d = (b - a)/\lambda)$. The equation uses the property that a chain of length i is only possible if $d|\lambda^i$. The chain would have a length $\lambda+(\lambda-1)i$. From the equation for $chain'$, we see that n plays a crucial role in determining the number of chains of length i. Specifically, if $n = \lambda + (\lambda - 1)i$, then the number of elements in the chain is equal to the number of memory accesses made, therefore the first element of the chain must coincide with T_1, which is fixed at t_a. Thus only one chain is possible. For $n > \lambda+(\lambda-1)i$, the chain does not have to coincide with T_1, thus all possible chains for b are applicable. This is given by $chain(b, \lambda, i, n)$. Let $I'_{b,a,n}$ be the number of ways memory accesses can be done in which $T_1 = t_a$, $T_{n+1} = t_b$, and there are no valid sequences for t_b in the preceding n memory accesses, ie. t_b is not prefetched. For $\lambda = 2$, $I'_{b,a,n}$ is given by the following recurrence: $I'_{b,a,0} = 1$, $I'_{b,a,1} = 1$, $I'_{b,a,2} = (l - 2)$, and

$$I'_{b,a,n} = (l-1)I'_{b,a,n-1} - (S_b^+ + S_b^-)I'_{b,a,n-2} + \sum_{i=1}^{\substack{max(\lfloor \log_2 S_b^+ \rfloor, \\ \lfloor \log_2 S_b^- \rfloor)}} (-1)^{i+1} \cdot I'_{b,a,n-2-i} \cdot chain'(b, a, 2, i, n) \text{ when } n > 2$$

Similarly for $\lambda = 3$, the recurrence for $I'_{b,a,n}$ is $I'_{b,a,0} = 1$, $I'_{b,a,1} = 1$, $I'_{b,a,2} = (l - 1)$, $I'_{b,a,3} = (l - 1)^2 - 1$, and,

$$I'_{b,a,n} = (l-1)I'_{b,a,n-1} - (S_b^+ + S_b^-)I'_{b,a,n-3} + \sum_{i=1}^{\substack{max(\lfloor \log_3 S_b^+ \rfloor, \\ \lfloor \log_3 S_b^- \rfloor)}} (-1)^{i+1} \cdot I'_{b,a,n-4-i} \cdot chain'(b, a, 3, i, n) \text{ when } n > 3$$

The probability of a cache hit in the n^{th} access due to prefetching in this case is

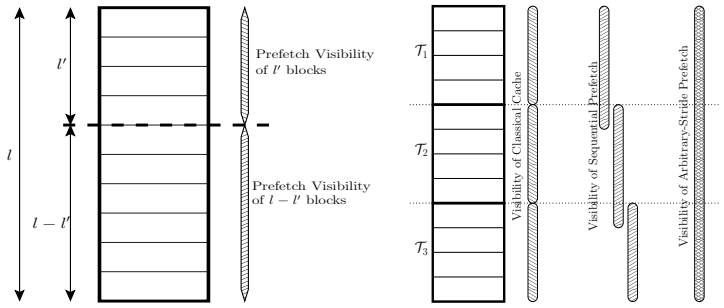
$$Pr[A^A_{l,n} = \mathcal{H} | T_1 = t_a, T_n = t_b \text{ and } t_a = t_b mod \lambda] = 1 - \frac{I'_{b,a,n-1}}{(l-1)^{n-2}}$$

4.3 Effect of Table Placement in Memory

The placement of tables in memory has an effect on the probability of obtaining a cache hit. There are two factors in the table's placement that influences the cache hit probability.

- The boundaries of prefetching
- The relative placement of the tables

Boundaries of Prefetching Some prefetchers are limited by prefetching boundaries in the memory. For example, the arbitrary-stride prefetching feature in Intel microprocessors can only prefetch within a memory page (4096 bytes) [20]. This means that even though a valid sequence appears in the memory accesses,



(a) Table Spread Across Boundary

(b) Tables Placed Side-by-Side

Fig. 4. Prefetch Visibility for Look-up Tables

prefetching will not occur if the memory block to be prefetched is in another page.

In order to determine the effect of prefetching in a table split between two memory pages, the part of the table in each page can be considered as an independent entity. Figure 4(a) shows a table of l memory blocks spread across two memory pages with l' memory blocks in one page and $l - l'$ blocks in another. Since accesses to a table in a cipher is uniformly random, each part of the table gets a fraction of the accesses that is proportional to its size. For some prefetching scheme P, the probability that the n^{th} memory access results in a cache hit due to a prefetched data is given by

$$Pr[A^P_{l,n} = \mathcal{H}] \approx \frac{l'}{l} Pr[A^P_{l',\lfloor \frac{nl'}{l} \rfloor} = \mathcal{H}] + \left(1 - \frac{l'}{l}\right) Pr[A^P_{(l-l'),n-\lfloor \frac{nl'}{l} \rfloor} = \mathcal{H}]$$

,where $A_{l',\lfloor \frac{nl'}{l} \rfloor}$ and $A_{(l-l'),\lfloor \frac{n(l-l')}{l} \rfloor}$ are the random variables denoting the outcome of the prefetching in each part of the table.

Relative Placement of Tables If a cipher is implemented with multiple tables and executed on a system with a classical cache architecture, an access to one table does not affect the probability of a hit in another table. However in cache architectures supporting prefetching, it is possible for memory accesses in one table to prefetch blocks of memory in another.

Figure 4(b) shows the ability (visibility) of a cache strategy to load blocks from other tables. Tables $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$, are assumed to be placed side-by-side in memory. For sequential prefetching, an access to the last memory block of one table may prefetch the first memory block of an adjacent table. Similarly, with the arbitrary-stride scheme, sequences may occur which would prefetch a memory block from any of the other tables. As seen in Figure 4(b) arbitrary-stride prefetching has the maximum visibility, however its effect on information leakage is less compared to sequential prefetching (as will be seen in Section 7). Therefore in this section we restrict discussion to sequential prefetching in multiple tables.

Consider that all the three tables in Figure 4(b) have l memory blocks and that they are accessed consecutively. That is, every access to the table $\mathcal{T}_1$ is

followed by an access to the table $\mathcal{T}_2$ and then $\mathcal{T}_3$. The random variable $A_{l,n}^{[j],C,S}$ denotes the outcome of the n^{th} memory access to table $\mathcal{T}_j$ for a cache with sequential prefetching policy. The random variable $T_n^{[j]}$ denotes the memory block accessed in the n^{th} memory access of table $\mathcal{T}_j$. This can take values $t_1^{[j]}, t_2^{[j]}, \cdots t_l^{[j]}$.

Since probability of a collision is independent of the number of tables used we obtain the following relation, which is similar to Equation 3.

$$Pr[A_{l,n}^{[j],C,S} = \mathcal{H}] = Pr[A_{l,n}^{C} = \mathcal{H}] + Pr[A_{l,n}^{[j],S} = \mathcal{H}] \cdot (1 - Pr[A_{l,n}^{C} = \mathcal{H}]) \quad (10)$$

$Pr[A_{l,n}^{[j],S} = \mathcal{H}]$ is the probability of a cache hit in the n^{th} access due to prefetching. For sequential prefetching this is given by the following lemma.

Lemma 6. *For $j > 1$,*

$$Pr[A_{l,n}^{[j],S} = \mathcal{H}] = Pr[A_{l,n}^{S} = \mathcal{H}] + \frac{1}{l} \cdot Pr[t_l^{[j-1]} \textit{ gets accessed}]) \quad (11)$$

If all accesses to $\mathcal{T}_{j-1}$ are random then $Pr[t_l^{[j-1]}$ gets accessed$] = 1-((l-1)^n/l^n)$ and if $T_1^{[j-1]} = t_l^{[j-1]}$ then this probability is 1. For $j = 1$, $Pr[A_{l,n}^{[1],S} = \mathcal{H}] = Pr[A_{l,n}^{S} = \mathcal{H}]$

Proof. For $\mathcal{T}_1$, just as in a single table, the first memory block cannot be prefetched (from the first observation Section 4.1). For tables $\mathcal{T}_j$ $(j > 1)$, which are placed after another table, the first memory block $t_1^{[j]}$, can be prefetched by the last memory block of the previous table (ie. $t_l^{[j-1]}$). The probability of $t_1^{[j]}$ getting accessed in the n^{th} memory access of t_j is $1/l$ and the probability that this access results in a cache hit is equal to the probability with which $t_l^{[j-1]}$ gets accessed at-least once in n memory accesses. □

When a condition is applied to one of the tables, it affects the outcome of the neighboring table's memory accesses. Table 3 summarizes the conditional probabilities for the n^{th} access in the three tables. To simplify notations in the table, $Pr[X = \mathcal{H}]$ is simply represented as $Pr[X]$.

Table 3. Conditional Probabilities for Sequential Prefetching in Three Tables ($\mathcal{T}_1$, $\mathcal{T}_2$ $\mathcal{T}_3$) Placed Side-by-Side

condition	$\mathbf{Pr[A_{l,n}^{[1],S}}$|**condition**$\mathbf{]}$	$\mathbf{Pr[A_{l,n}^{[2],S}}$|**condition**$\mathbf{]}$	$\mathbf{Pr[A_{l,n}^{[3],S}}$|**condition**$\mathbf{]}$
$T_1^{[1]} = t_l^{[1]}$	$Pr[A_{l,n}^{S}\|T_1 = t_l]$ (Equation 8)	$Pr[A_{l,n}^{S}] + \frac{1}{l}$ (Equation 11)	$Pr[A_{l,n}^{S}] + \frac{1}{l}X_1$ (Equation 11)
$T_1^{[1]} \neq t_l^{[1]}$	$Pr[A_{l,n}^{S}\|T_1 \neq t_l]$ (Equation 9)	$Pr[A_{l,n}^{S}] + \frac{1}{l}X_2$ (Equation 11)	$Pr[A_{l,n}^{S}] + \frac{1}{l}X_1$ (Equation 11)
$T_1^{[2]} = t_l^{[2]}$	$Pr[A_{l,n}^{S}]$ (Equation 4)	$Pr[A_{l,n}^{S}\|T_1 = t_l] + \frac{1}{l}X_1$ (Equation 8,11)	$Pr[A_{l,n}^{S}] + \frac{1}{l}$ (Equation 11)
$T_1^{[2]} \neq t_l^{[2]}$	$Pr[A_{l,n}^{S}]$ (Equation 4)	$Pr[A_{l,n}^{S}\|T_1 \neq t_l] + \frac{1}{l}X_1$ (Equation 9,11)	$Pr[A_{l,n}^{S}] + \frac{1}{l}X_2$ (Equation 11)
$T_1^{[3]} = t_l^{[3]}$	$Pr[A_{l,n}^{S}]$ (Equation 4)	$Pr[A_{l,n}^{S}] + \frac{1}{l}X_1$ (Equation 4,11)	$Pr[A_{l,n}^{S}\|T_1 = t_l] + \frac{1}{l}X_1$ (Equation 8,11)
$T_1^{[3]} \neq t_l^{[3]}$	$Pr[A_{l,n}^{S}]$ (Equation 4)	$Pr[A_{l,n}^{S}] + \frac{1}{l}X_1$ (Equation 4,11)	$Pr[A_{l,n}^{S}\|T_1 \neq t_l] + \frac{1}{l}X_1$ (Equation 9,11)
$X_1 = (l^n - (l-1)^n)/l^n$ $\quad$ $X_2 = (l^n - (l-1)^{n-1})/l^n$			

4.4 Estimating the Cache Miss Distribution

When a cipher is executed several times, the number of cache misses form a distribution as seen in Figure 2(b). This distribution is normal and can be defined by its expected value and variance. Further we see that the random variables $A^{C}_{l,n}$ (for the classical cache) and $A^{C,P}_{l,n}$ (for caches with prefetching) form a distribution of the number of cache misses in the n^{th} access. For this section, we generalize the notations $A^{C}_{l,n}$ and $A^{C,P}_{l,n}$ by $A_{l,n}$. The expected number of cache misses in the n^{th} access is,

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

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

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

Let M_n be a random variable denoting the number of cache misses after n memory accesses. Let the expectation and variance of M_n be denoted $E(M_n)$ and $V(M_n)$. These are given by the recurrences,

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

and

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

While the variance for the classical cache can be correctly predicted, the variance for the prefetching caches is approximated from that of the classical case. The expectation can be correctly predicted in all cases.

If the cipher uses more than 1 table then the expected number of cache misses in the cipher is simply the sum of each table's corresponding expectation. If the tables are placed so that accessing one table is independent of the others (for example, the tables are placed in different memory pages), then the variance of the number of cache misses is the sum of each table's corresponding variance. On the other hand if the accesses to the tables are not independent (example, the tables are placed side-by-side) then the covariance between the cache miss distributions of each table has to be considered.

5 Building Cache Profiles of a Block Cipher

In a profiled cache-timing attack, prediction of k_i requires the timing profile for k_i to be built (Algorithm 3). From the discussion in Section 3, timing information from a block cipher can be analyzed in terms of the number of cache misses during the cipher's execution. We therefore introduce the notion of a *cache profile* for k_i.

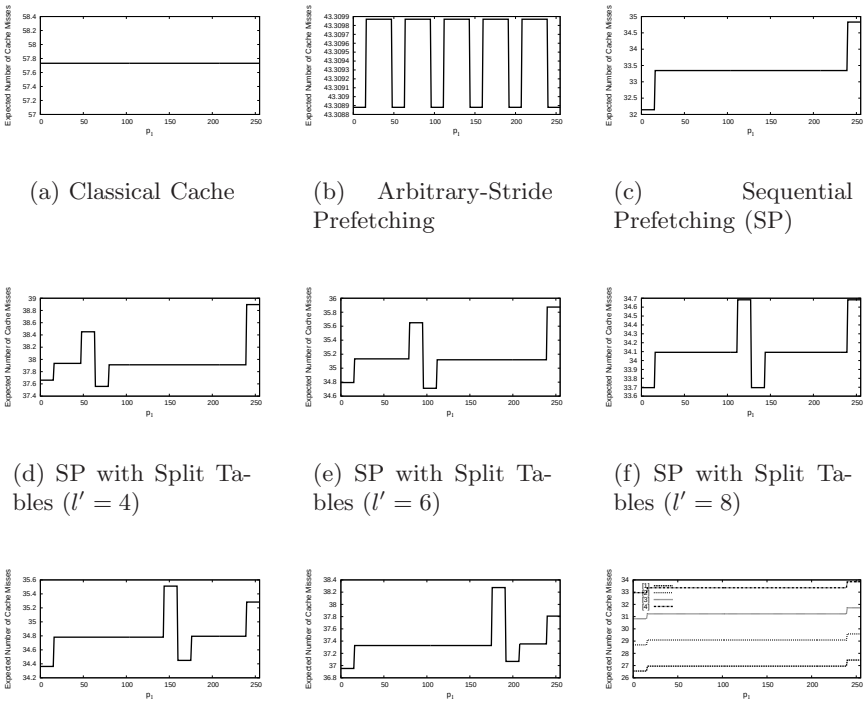
Definition 1. *The* **cache profile** *for k_i is the variations in the cache miss distribution with respect to the value of p_i when Algorithm 3 is computed.*

Algorithm 4 shows the estimation of the cache profile for k_1 and can be modified to retrieve the other keys. The algorithm takes as input the number of tables (Γ), the number of accesses per table (n_{max}) and the size of each table

Algorithm 4: Cache Profile for k_1

Input: $\mathcal{E}_k(\Gamma, n_{max}, l)$, Prefetching Style ($P$), Location of tables
Output: The Cache Profile for k_1 ($\mathsf{CP}(k_1)$)

```
1 begin
2     forall p_1 ∈ {0, 1, ··· , 2^m − 1} do
3         for n ∈ {1, ··· , n_max} do
4             Compute the conditional probabilities of a cache hit in the n^th
              memory access in all Γ tables given T_1 = p_1.
5         end
6         E[p_1] = expected value of cache misses after n_max memory accesses
          V[p_1] = variance of the cache misses after n_max memory accesses
7     end
8     return The table of ⟨E, V⟩
9 end
```



(a) Classical Cache

(b) Arbitrary-Stride Prefetching

(c) Sequential Prefetching (SP)

(d) SP with Split Tables ($l' = 4$)

(e) SP with Split Tables ($l' = 6$)

(f) SP with Split Tables ($l' = 8$)

(g) SP with Split Tables ($l' = 10$)

(h) SP with Split Tables ($l' = 12$)

(i) SP with Tables Side-by-Side

Fig. 5. Cache Profiles for k_1 with Different Prefetching Styles and $\Gamma = 4, l = 16, n_{max} = 36, \delta = 16$

(l) for the cipher $\mathcal{E}_{\mathrm{K}}$. The algorithm also takes the prefetching style P used in the cache memory, and the locations of the Γ tables in memory.

The cache profile returned by the algorithm comprises of a table with 2^{m} rows with each row consisting of a value of p_1 and the corresponding distribution of the number of cache misses that occurred during the encryption. Each distribution

Table 4. Comparison of Predicted and Empirical Distributions for the Cipher Model, $\Gamma = 1$, $l = 8$, $n_{max} = 36$

Prefetching Style	**Predicted Distribution**		**Empirical Distribution**	
	Expectation	Variance	Expectation	Variance
Classical Cache	7.934	0.0629	7.938	0.0591
Sequential Prefetching	4.492	0.6695	4.463	0.7302
Arbitrary-Stride Prefetching	7.734	0.2454	7.734	0.2622

is Gaussian (as seen in Figure 2(b)), therefore the table records the mean and variance of the distribution.

Depending on the type of prefetching, the cache profile would vary for the same cipher parameters. Figure 5 plots the cache profiles for $k_1 = 0$ with various prefetching styles and locations of the tables. The $x-$axis has the values of p_1 while the $y-$axis has the expected number of cache misses. Except for the classical cache (Figure 5(a)), all other prefetching styles show a variation in the expected number of cache misses during the encryption. In the arbitrary-stride prefetching, the variations in the cache profile is very less, while the variations are much more significant in the case of the sequential prefetching schemes. Figures 5(d), 5(e), 5(f), 5(g), and 5(h) show how the cache profile with sequential prefetching (5(c)) gets affected due to the tables placed across a prefetching boundary. All cache profiles except (5(i)) assume that each table is not affected by accesses to a neighboring table. Figure 5(i) shows cache profiles for each table when the tables are placed side-by-side with $\mathcal{T}_1$ placed first followed by $\mathcal{T}_2$, $\mathcal{T}_3$, and then $\mathcal{T}_4$. These variations in the cache profile aid an adversary in a profiled cache-timing attack.

5.1 Validation of the Cache Models

To check the correctness of the mathematical models developed in this section, we used the cache profiling tool *Cachegrind*[4]. Cachegrind provides an ideal platform for verification as most of the assumptions made in Section 3 are met. The source code of Cachegrind was patched to support the prefetching algorithms described in Algorithms (1) and (2). The predicted cache miss distribution was compared with empirical results obtained from Cachegrind for various cipher (Figure 1) and cache models. Table 4 shows a comparison of the distributions for various prefetching styles. The cipher model used in the table has parameters $\Gamma = 1$, $l = 8$, $n_{max} = 36$ with the accesses to the table at randomly chosen locations. The cache modeled is direct mapped with $\delta = 32$ and of size $16KB$. The empirical distributions were obtained after sampling $10,000$ executions of the cipher. Figure 6 shows the cache profiles for key byte k_1 obtained from Cachegrind and the prediction for the same cipher and cache model. Both the table and the cache profile show that the predicted distributions closely match empirical results. It was also found that for reasonably large cache memories ($> 4KB$), the size of the cache and the associativity have little impact on the distribution obtained.

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

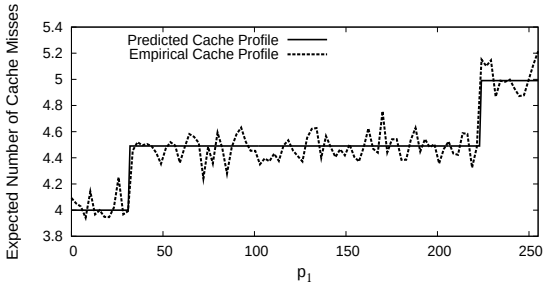


Fig. 6. Predicted and Empirical Cache Profiles for k_1 for Cipher Model $\Gamma = 1$, $l = 8$, $n_{max} = 36$

6 Formal Model for the Profiled Cache Timing Attack

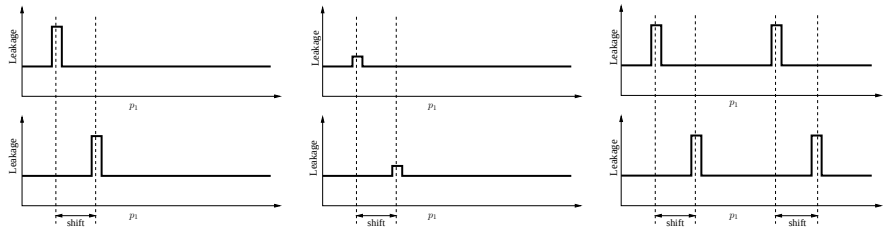
Just as in the case of timing profiles used in a profiled cache-timing attack (Section 2.3), the distribution of the cache misses for p_1 also satisfies the EIS property. Therefore for the known (K^*) and unknown ($K^\#$) keys,

$$CacheMissDistribution_{k_1^*}(p_1^*) = CacheMissDistribution_{k_1^\#}(p_1^\# \oplus (k_1^* \oplus k_1^\#))$$

This means that the cache profile for the unknown key ($\mathsf{CP}(k_1^\#)$) is a shifted version of the cache profile of the known key ($\mathsf{CP}(k_1^*)$). It is this shift that leaks information about the secret key from the cache profile.

The cache profile $\mathsf{CP}(\mathsf{k_1})$ comprises of 2^{m} distributions corresponding to each value of p_1. Let the i^{th} distribution be called $F_{k_{1,i}}$ for $(0 \leq i \leq 2^{\mathsf{m}} - 1)$. In the cache profile with a classical cache (Figure 5(a)), all 2^{m} distributions are equal, therefore a shift between the profiles $\mathsf{CP}(k_1^\#)$ and $\mathsf{CP}(k_1^*)$ is not observable and no information is leaked.

Consider the cache profile in Figure 7(a) in which all 2^{m} distributions are the same except for one. The deviating distribution being $F_{k_{1,\mu}}$. The shift in the profile is observable with an error whose standard deviation is say e. The magnitude of the error depends on the amount of deviation of $F_{k_{1,\mu}}$ compared to the other distributions. For example, the measurement of shift in the cache profile in Figure 7(b) would have higher error than that of Figure 7(a). If there are more



(a) Single, Large Deviating Distribution

(b) Single, Small Deviating Distribution

(c) Multiple Deviating Distributions

Fig. 7. Cache Profiles with Single or Multiple Deviating Distributions

variations in the distributions such as in Figure 7(c), then the error in deducing the shift would reduce. In general, if there are N variations in a cache profile then the standard deviation of the mean measurement would approximately reduce to $\frac{e}{\sqrt{N}}$ [36]. Thus, the uncertainty of the key $k_1^{\#}$ can be analyzed in terms of the variations in a cache profile.

Quantifying Information Leakage : The variations in the cache profile is measured by the distances between the distributions. The symmetric version of the Kullback-Leibler (KL) divergence [22] is used for this purpose. Equation 12 computes the symmetric KL divergence between two distribution $F_{k_{1,i}}$ and $F_{k_{1,i'}}$.

$$D(F_{k_{1,i}}, F_{k_{1,i'}}) = D(F_{k_{1,i}} || F_{k_{1,i'}}) + D(F_{k_{1,i'}} || F_{k_{1,i}}) \tag{12}$$

, where

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

The index of the summation, j, ranges from the minimum to the maximum number of cache misses. The minimum number of cache misses is 1 corresponding to the first compulsory miss in the table. The maximum number of cache misses is either n (the number of table accesses) or l (the number of memory blocks required by the table), whichever is greater.

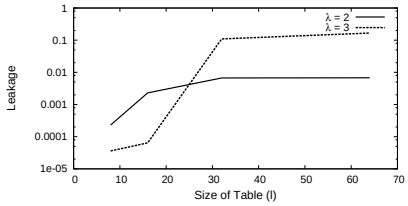
To compute the total information leakage ($\mathfrak{D}(\mathsf{CP}(\mathsf{k_1}))$ for the cache profile for k_1, the value of $D(F_{k_{1,i}}, F_{k_{1,i'}})$ is added for every possible pair of distributions. Therefore,

$$\mathfrak{D}(\mathsf{CP}(\mathsf{k_1})) = \sum_{\substack{\forall pairs\ of \\ i\ and\ i', i \neq i'}} D(F_{k_{1,i}}, F_{k_{1,i'}}) \tag{13}$$

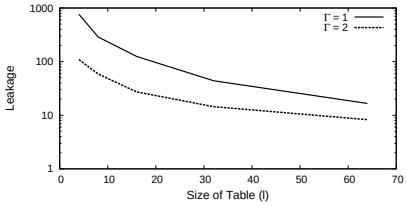
7 Analysis of Information Leakage

There are two parts in Equation 13: *(a)* the Kullback Leibler divergence and *(b)* the summation over all pairs of distributions in the cache profile. The Kullback Leibler divergence is zero when i and i' fall in the same cache line, leading to an uncertainty in detection of shifts in the cache profile. For information to be leaked, at-least one pair of i and i' from different cache lines should have a non-zero divergence. Higher the value of the divergence implies more information getting leaked. The summation in Equation 13 quantifies the case when the cache profile has more variations in the distributions. This leads to more information leakage.

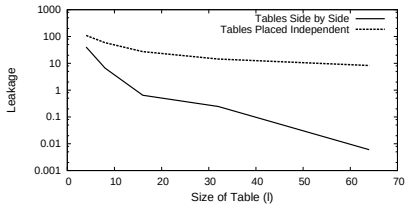
As the size of the table (l) increases, it was found that the Kullback Leibler divergence due to prefetching reduces. On the other hand the contribution of the summation in Equation 13 increases. In some cache configurations, the divergence is more pronounced thus leading to a decrease in leakage as l increases. In other cache configurations, the leakage increases with l due to the increase in the number of terms summed in Equation 13. Figure 8 shows the variation in leakages for various configurations as the size of the table(s) increases. From the figures we see that caches with sequential prefetching (Figure 8(b)) fall in the former category with prominent divergence. The arbitrary-stride prefetching falls in the latter category where the leakage reduces with an increasing l. Also, from these figures, it is seen that the sequential prefetching has much larger leakage than arbitrary-stride prefetching.



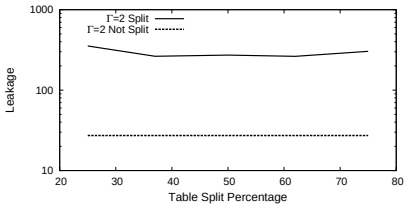
(a) Arbitrary-Stride Prefetching ($\Gamma = 1$)



(b) Sequential Prefetching (SP)



(c) SP with Relative Positions of Tables ($\Gamma = 2$)



(d) SP with Split Tables ($l = 16$, $\Gamma = 1$)

Fig. 8. Leakage for k_1 with Various Configurations as the Table Size Increases ($n_{max} = 36$)

Figure 8(b) shows that the leakage reduces as the number of tables (Γ) increase. Moreover, if the tables are placed side-by-side (Figure 8(c)), there is a further reduction in leakage. Figure 8(d) shows an increase in leakage when the table is split between prefetching boundaries.

We summarize the results as follows:

- For sequential prefetching, large tables have less leakage due to prefetching compared to small tables.
- For the arbitrary-stride prefetching, large tables leak more than small tables. However the magnitude of leakage is several times smaller compared to the sequential prefetching scheme.
- Larger number of tables in the cipher would result in lesser information leakage in profiled cache-timing attacks.
- Tables in which accesses are dependent on other tables have less leakage compared to tables in which the accesses are independent of other tables.
- Split tables have a higher leakage, and the more unequal the split, the larger the leakage.

8 Practical Aspects in Timing Profiles

In this section we discuss variations in timing profiles obtained during the execution of Algorithm 3. The difference $\mathfrak{t}_d$ between any pair of distributions in the resulting timing profile can be represented by the equation,

$$\mathfrak{t}_d = \mathfrak{t}_c + \mathfrak{t}_o + \mathfrak{t}_n$$

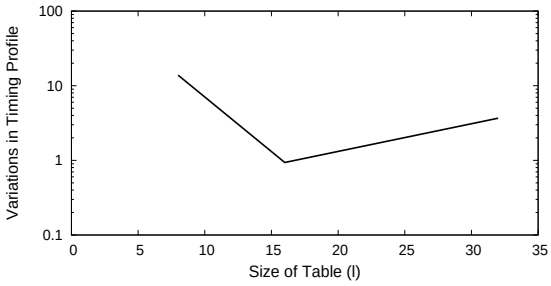


Fig. 9. Variations in $\mathfrak{D}$ for Timing Profiles as l increases on an Intel Core 2 Duo with $\Gamma = 4$, $n_{max} = 36$

There are three contributing components in $\mathfrak{t}_d$. The component $\mathfrak{t}_c$ denotes the variation due to cache misses between the distributions. This is the variation formally analyzed in the previous sections. Besides cache misses, other factors could also cause differences in the timing. This is represented by $\mathfrak{t}_o$. Some of these factors have been discussed in [6]. These contributory factors include, for example, memory reads taking longer time under certain conditions. The components $\mathfrak{t}_c + \mathfrak{t}_o$ in $\mathfrak{t}_d$ result in information leakage, while the third component $\mathfrak{t}_n$ is due to noise, which makes attacking the system more difficult.

The noise can be categorized into two: frequently occurring ($\mathfrak{t}_{nf}$) and sporadic ($\mathfrak{t}_{ns}$). Therefore $\mathfrak{t}_n = \mathfrak{t}_{nf} + \mathfrak{t}_{ns}$. Reasons for frequently occurring noise in the measurements could be several such as stalls in the processor owing to pipelines or buffers getting full. Generally these events affect all distributions in the timing profile to an equal extent therefore their effect gets canceled out when the difference between distributions is taken (so, $\mathfrak{t}_{nf} = 0$). Sporadic noise occur rarely during the execution but are of much larger magnitude. Examples of sporadic noise are due to TLB misses or page faults. Since this noise is sporadic, they affect distributions by an unequal extent and their effect on the difference of distributions cannot be canceled out (ie. $\mathfrak{t}_{ns} > 0$). Also, as the size of the tables increase, $\mathfrak{t}_{ns}$ (hence $\mathfrak{t}_n$) also increases. For example, with an increase in table size, TLB misses and pages faults are more likely.

To quantify the amount of variations in the timing profile, we use the same metric as in Equation 13 but the distributions $F_{k_{1,i}}$ and $F_{k_{1,i'}}$ in the equation are timing and not cache misses.

$$\mathfrak{D}(\mathsf{TP}(\mathsf{k}_1)) = \sum_{\substack{\forall pairs\ of \\ i\ and\ i', i \neq i'}} D(F_{k_{1,i}}, F_{k_{1,i'}}) \tag{14}$$

Figure 9 show the variations in this metric as the size of the tables used increases. Assuming that $\mathfrak{t}_o$ is a constant with respect to the table size, the analysis in Section 7 shows a decrease in $\mathfrak{t}_c$ as l increases. Also t_n increases with l. This explains the trend in Figure 9, where for smaller values of l there is more leakage and less noise compared to larger values of l. This makes implementations of ciphers with small tables more easier to break by profiled cache timing attacks than ciphers with large tables. This was experimentally verified by mounting cache profiled attacks on implementations of CLEFIA (using two 256 byte tables) and OpenSSL's AES implementation (using four 1024 byte tables). We found

that in 85% of cases the attack on CLEFIA gave the correct key, while only 60% of the cases the right AES key was obtained in the same attack environment.

9 Summary and Future Directions

This paper is the first reported attempt to quantify leakages due to a micro-architectural component in the system. The paper considers data prefetching; an important feature in most modern day cache memories. Two data prefetching algorithms, namely sequential prefetching and arbitrary-stride prefetching are analyzed. The results can be used as a guide to implementing ciphers with minimal leakage due to prefetching. The analysis shows that ciphers implemented with small tables leak more information due to prefetching compared to those with large tables. This leakage can be reduced by using several tables in the implementation. Further reduction in leakage can be obtained by placing tables side-by-side in memory. Finally, tables should not be split across prefetching boundaries as this would increase leakage.

There are several architecture and micro-architecture components in the system capable of leaking information. This paper has considered one such component. A future direction is to pin-point and quantify the leakage from other components in the system, thus leading to a complete analysis of leakage in the entire system. Since disabling prefetching may have an averse effect on the performance of other applications, another direction could be in the design of new prefetching strategies which either reduces or completely eliminates information leakage.

References

1. Abe, M. (ed.): Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings, Lecture Notes in Computer Science, vol. 4377. Springer (2006)
2. Aciiçmez, O., Çetin Kaya Koç: Trace-Driven Cache Attacks on AES (Short Paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS. Lecture Notes in Computer Science, vol. 4307, pp. 112–121. Springer (2006)
3. Aciiçmez, O., Çetin Kaya Koç, Seifert, J.P.: Predicting secret keys via branch prediction. In: Abe [1], pp. 225–242
4. Aciiçmez, O., Schindler, W., Çetin Kaya Koç: Cache Based Remote Timing Attack on the AES. In: Abe [1], pp. 271–286
5. Baer, J.L.: Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors. Cambridge University Press (2010)
6. Bernstein, D.J.: Cache-timing Attacks on AES. Tech. rep. (2005)
7. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: AES Power Attack Based on Induced Cache Miss and Countermeasure. In: ITCC (1). pp. 586–591. IEEE Computer Society (2005)
8. Bonneau, J., Mironov, I.: Cache-Collision Timing Attacks Against AES. In: Goubin, L., Matsui, M. (eds.) CHES. Lecture Notes in Computer Science, vol. 4249, pp. 201–215. Springer (2006)
9. Brumley, D., Boneh, D.: Remote Timing Attacks are Practical. Computer Networks 48(5), 701–716 (2005)
10. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding Cache Attacks. Research Report RR-5881, INRIA (2006), http://hal.inria.fr/inria-00071387/en/
11. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Çetin Kaya Koç, Paar, C. (eds.) CHES. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002)

12. Crosby, S.A., Wallach, D.S., Riedi, R.H.: Opportunities and Limits of Remote Timing Attacks. ACM Trans. Inf. Syst. Secur. 12(3) (2009)
13. Federal Information Processing Standards Publication 197: Announcing the Advanced Encryption Standard (AES) (2001)
14. Fournier, J.J.A., Tunstall, M.: Cache Based Power Analysis Attacks on AES. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP. Lecture Notes in Computer Science, vol. 4058, pp. 17–28. Springer (2006)
15. Gallais, J.F., Kizhvatov, I., Tunstall, M.: Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations. In: Chung, Y., Yung, M. (eds.) WISA. Lecture Notes in Computer Science, vol. 6513, pp. 243–257. Springer (2010)
16. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES. Lecture Notes in Computer Science, vol. 5154, pp. 426–442. Springer (2008)
17. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest We Remember: Cold Boot Attacks on Encryption Keys. In: van Oorschot, P.C. (ed.) USENIX Security Symposium. pp. 45–60. USENIX Association (2008)
18. Hegde, R.: Optimizing Application Performance on Intel Core Microarchitecture Using Hardware-Implemented Prefetchers. Intel Software Network, `http://software.intel.com/en-us/articles/optimizing-application-performance-on-intel-coret-microarchitecture-using-hardware-implemented-prefetchers/` (2008)
19. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 4th Edition. Morgan Kaufmann (2006)
20. Intel Corporation: Intel 64 and IA-32 Architectures Optimization Reference Manual (2009)
21. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side Channel Cryptanalysis of Product Ciphers. J. Comput. Secur. 8(2,3), 141–158 (2000)
22. Kullback, S., Leibler, R.A.: On Information and Sufficiency. Annals of Mathematical Statistics 22, 49–86 (1951)
23. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor, M. (ed.) TCC. Lecture Notes in Computer Science, vol. 2951, pp. 278–296. Springer (2004)
24. Neve, M.: Cache-based Vulnerabilities and SPAM Analysis. Ph.D. thesis, Thesis in Applied Science, UCL (2006)
25. Neve, M., Seifert, J.P.: Advances on Access-Driven Cache Attacks on AES. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 4356, pp. 147–162. Springer (2006)
26. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The Case of AES. In: Pointcheval, D. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 3860, pp. 1–20. Springer (2006)
27. Page, D.: Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel (2002)
28. Percival, C.: Cache Missing for Fun and Profit. In: Proc. of BSDCan 2005 (2005)
29. Rebeiro, C., Mukhopadhyay, D.: Cryptanalysis of CLEFIA Using Differential Methods with Cache Trace Patterns. In: Kiayias, A. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 6558, pp. 89–103. Springer (2011)
30. Rebeiro, C., Mukhopadhyay, D., Takahashi, J., Fukunaga, T.: Cache Timing Attacks on CLEFIA. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT. Lecture Notes in Computer Science, vol. 5922, pp. 104–118. Springer (2009)
31. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES. Lecture Notes in Computer Science, vol. 3659, pp. 30–46. Springer (2005)
32. Schneier, B., Kelsey, J.: Unbalanced feistel networks and block cipher design. In: Gollmann, D. (ed.) FSE. Lecture Notes in Computer Science, vol. 1039, pp. 121–144. Springer (1996)

33. Sony Corporation: The 128-bit Blockcipher CLEFIA : Algorithm Specification (2007)
34. Standaert, F.X., Koeune, F., Schindler, W.: How to Compare Profiled Side-Channel Attacks? In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D. (eds.) ACNS. Lecture Notes in Computer Science, vol. 5536, pp. 485–498 (2009)
35. Standaert, F.X., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 5479, pp. 443–461. Springer (2009)
36. Taylor, J.R.: An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements (Second Edition). University Science Books (1997)
37. Tiri, K., Aciiçmez, O., Neve, M., Andersen, F.: An analytical model for time-driven cache attacks. In: Biryukov, A. (ed.) FSE. Lecture Notes in Computer Science, vol. 4593, pp. 399–413. Springer (2007)
38. Tromer, E., Osvik, D.A., Shamir, A.: Efficient Cache Attacks on AES, and Countermeasures. Journal of Cryptology 23(2), 37–71 (2010)
39. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of DES Implemented on Computers with Cache. In: Walter, C.D., Çetin Kaya Koç, Paar, C. (eds.) CHES. Lecture Notes in Computer Science, vol. 2779, pp. 62–76. Springer (2003)
40. Tsunoo, Y., Tsujihara, E., Minematsu, K., Miyauchi, H.: Cryptanalysis of Block Ciphers Implemented on Computers with Cache. In: International Symposium on Information Theory and Its Applications. pp. 803–806 (2002)
41. Vanderwiel, S.P., Lilja, D.J.: When caches aren't enough: Data prefetching techniques. IEEE Computer 30(7), 23–30 (1997)
42. Vanderwiel, S.P., Lilja, D.J.: Data prefetch mechanisms. ACM Comput. Surv. 32(2), 174–199 (2000)