Journal of Computer Security 0 (0) 1
IOS Press

Towards the Optimal Performance of Integrating WARM and DELAY against Remote Cache Timing Side Channels on Block Ciphers

Ziqiang Ma [a,b,c], Quanwei Cai [a,b,*], Jingqiang Lin [a,b,c], Bo Luo [d] and Jiwu Jing [b,e]
[a] *State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China*
E-mails: maziqiang@iie.ac.cn, caiquanwei@iie.ac.cn, linjingqiang@iie.ac.cn
[b] *Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, China*
[c] *School of Cyber Security, University of Chinese Academy of Sciences, China*
[d] *Department of Electrical Engineering and Computer Science, University of Kansas, USA*
E-mail: bluo@ku.edu
[e] *School of Computer Science, University of Chinese Academy of Sciences, China*
E-mail: jing@is.ac.cn

Abstract. Cache timing side channels allow a remote attacker to disclose the cryptographic keys, by repeatedly invoking the encryption/decryption functions and measuring the execution time. WARM and DELAY are two algorithm-independent and implementation-transparent countermeasures against remote cache-based timing side channels for block ciphers. They destroy the relationship between the execution time and the cache misses/hits which are determined by the secret key, but bring remarkable performance overhead. In this paper, we investigate the performance of cryptographic functions protected by WARM and DELAY, and attempt to find the best strategy to integrate these two countermeasures with the optimal performance while effectively eliminate remote cache timing side channels for block ciphers implementations with lookup tables. To the best of our knowledge, this work is the first to systematically analyze the performance of integrating WARM and DELAY against cache side channels.We derive the optimal scheme to integrate WARM and DELAY, and apply it to AES. It is proven that the integration scheme achieves the optimal performance with the least extra operations on commodity systems. Finally, we implement it on Linux with Intel CPUs. Experimental results confirm that, (*a*) the execution time does not leak information on cache access, (*b*) the scheme outperforms other integration strategies of WARM and DELAY, and (*c*) the implementation works without any privileged operations on the computer.

Keywords: cache side channel, optimal performance, timing side channel, block cipher, AES

1. Introduction

In practical implementations of cryptographic algorithms, the cryptographic keys could be leaked through side channels on timing [1–11], electromagnetic fields [12], power [13, 14], ground electric

[*]Corresponding author. E-mail: caiquanwei@iie.ac.cn.

potential [15] or acoustic emanations [16], even when the algorithm is semantically secure. Among these vulnerabilities, *timing side-channel attacks* are launched without any special probe devices. In particular, *remote timing side channels* allow attackers, who have no system privilege or physical access to the computer, to discover the keys by repeatedly invoking the encryption/decryption functions and measuring the execution time [3, 4, 7, 17, 18].

One type of remote timing side channels, called *remote cache timing side channels* in this paper, exploits the time differences of data-cache hits and misses. Such cache timing side channels are widely found in the implementations of block ciphers [4, 9, 19–21]. In the implementations, table lookup is the primary time-consuming operation in encryption/decryption. The overall execution time is influenced by the number of cache misses/hits in table lookup. Therefore, from the execution time, attackers are able to infer the inputs of table lookup operations which are determined by the secret keys (and the known plaintext/ciphertext). Generally, for a block cipher, encryption and decryption are symmetric computations with same basic operations. So we only emphasize on encryption in the rest of the paper, but all discussions and conclusions are applicable to decryption.

Compared with other side channels, such as power, electromagnetic fields, ground electric potential and acoustic emanations, cache-based timing side-channel attacks do not require special equipments or extra physical access to the victim system. Moreover, such remote attacks only require the least privilege to (remotely) invoke the encryption functions, while other active cache side-channel attacks involve operations by a malicious task that shares caches with the victim cryptographic engine [5, 22–24].

Various methods are proposed against cache timing side-channel attacks for block ciphers, destroying the relationship between the secret keys and the execution time. The intuitive design is to perform encryption, *a*) with the lookup tables completely inside or outside caches, or *b*) in a constant period of time, regardless of cache utilization. WARM and DELAY are two typical algorithm-independent and implementation-transparent mechanisms to eliminate cache timing side channels [8, 25, 26]. WARM fills cache lines by reading constant tables *before* the encryption operations, and DELAY inserts padding instructions *after* encryption.[1] With WARM or DELAY, the execution time measured by attackers becomes irrelevant to the cache misses/hits *during* encryption.

These two common mechanisms prevent cache-based timing attacks at different phases, but they introduce extra operations and significantly degrade the performance. In this paper, we investigate the performance of symmetric cryptographic functions that adopt WARM and DELAY to defend against cache timing side channel attacks, and attempt to find *the optimal strategy to integrate these two mechanisms* to achieve the optimized performance. The following principles are analyzed and discussed in the integration:

(1) Each encryption is protected by WARM and/or DELAY, so that the measured time reflects either the best case (i.e., all table entries are cached by WARM), or the worst case (i.e., it is delayed to the execution time without any caches). Otherwise, the cache timing side channel attacks may still be launched.
(2) In order to optimize the performance, WARM is preferred to DELAY; i.e., finish encryption operations as many as possible, with all table entries in caches. Only when the effect of WARM is broken by system activities, such as interrupts and task scheduling, i.e., the information on the secret key may be leaked, the execution time is delayed to the worst case.

[1]Another choice is to perform encryption without caches, which is inefficient. In fact, a typical DELAY strategy is to extend the encryption operation in constant time, equal to the execution time without caches.

(3) When DELAY is performed, WARM is done as a part of inserted padding instructions for the next encryption. So, the cost of WARM is masked by padding instructions, and the performance is further improved.

The integrated scheme does not require any privileged operation on computer systems. The conditions to perform WARM and DELAY, are defined as regular timing. Reading constant tables, inserting padding instructions, and timing are commonly supported in computer systems without special privileges. The scheme is independent of algorithms and transparent to implementations. It does not depend on any special design or feature of block ciphers, and it is applicable to different implementations.

We apply the integrated WARM+DELAY scheme to protect AES, and analyze the conditions that produce the optimal performance. It is proven that, the integrated scheme with appropriate parameters, achieves *the optimal performance with the least extra operations* on commodity systems; that is, without any unnecessary table lookup operations or padding instructions. The optimal performance of WARM+DELAY with different key sizes (128, 192, and 256 bits) and different implementations (2KB, 4KB, 4.25KB, and 5KB lookup tables), is also investigated. For example, the protected AES-128 implementation with a 2KB lookup table [27] achieves the optimal performance, if the ratio of the extra execution time caused by DELAY to the extra time by WARM, is less than a certain threshold. These conditions hold in commodity computers, which is confirmed by our experiments. The conditions of the optimal performance for other AES implementations of different key sizes are not identical but also hold in commodity systems.

To the best of our knowledge, this is the first to systematically analyze the optimal performance of integrating WARM and DELAY against cache timing side channels in commodity systems. Our contributions are as follows:

- We investigate the overheads of WARM and DELAY, and derive the scheme to integrate these countermeasures with the optimal performance while effectively eliminate remote cache side channels for block ciphers.
- We implement the optimal integration scheme on Linux with Intel CPUs for AES-128, and experimentally confirm that it (*a*) eliminates cache timing side channels, (*b*) outperforms other different integration strategies, and (*c*) works without any privileged operation on the system.

The remainder of this paper is organized as follows. Section 2 presents the background and related works. Section 3 discusses the WARM+DELAY scheme and analyzes its security. Section 4 proves the optimal integration scheme, which is verified in Section 5. Section 6 contains extended discussions. Section 7 draws the conclusions.

2. Background and Related Works

2.1. AES and Block Cipher Implementation

AES is a block cipher with 128-bit blocks, and the key is 128, 192 or 256 in bits. AES encryption (or decryption) consists of several rounds of transformations and the number depends on the key length. Each round consists of `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` on the 128-bit state.[2] The transformations except `AddRoundKey`, are implemented as lookup operations on constant tables [27], and `AddRoundKey` consists of bitwise-XOR on the state.

[2]The last round of AES encryption performs only `SubBytes`, `ShiftRows` and `AddRoundKey`.

The straightforward AES implementation needs four 1KB tables in all rounds, T_0, T_1, T_2 and T_3, where $S(x)$ is the result of an AES S-box lookup for the input x.

$$\begin{cases} T_0[x] = (2 \cdot S(x), S(x), S(x), 3 \cdot S(x)) \\ T_1[x] = (3 \cdot S(x), 2 \cdot S(x), S(x), S(x)) \\ T_2[x] = (S(x), 3 \cdot S(x), 2 \cdot S(x), S(x)) \\ T_3[x] = (S(x), S(x), 3 \cdot S(x), 2 \cdot S(x)) \end{cases}$$

Four tables are encoded into a 2KB lookup table in the compact implementation, $T[x] = (2 \cdot S(x), S(x), S(x), 3 \cdot S(x), 2 \cdot S(x), S(x), S(x), 3 \cdot S(x))$. Note that, $T_0[x]$, $T_1[x]$, $T_2[x]$ and $T_3[x]$ are included in $T[x]$.

Implementations of other block ciphers, usually consist of table lookup operations and basic computations without data-dependent branches in the execution path, such as 3DES, Blowfish [28], CAST-128 [29] in OpenSSH-7.4p1 [30].

2.2. Cache Side Channels

Caches, a small amount of high-speed memory cells located between CPU cores and RAM, are designed to temporarily store the data recently accessed by CPU cores, avoiding accessing the slow RAM chips. When the CPU core attempts to access a data block, the operation takes place in caches if the data have been cached (i.e., cache hit); otherwise, the data block is firstly read from RAM into caches (i.e., cache miss) and then the operation is performed in caches.

Cache timing side-channel attacks exploit the fact that accessing cached data is about two orders of magnitude faster than those in RAM, to recover the keys based on the execution time. Typically, it takes 3 to 4 cycles for a read operation in L1D caches, while an operation in RAM takes tens or hundreds of cycles [31].

Cache side-channel attacks on cryptographic engines are roughly divided into three categories: trace-driven, time-driven and access-driven attacks. The trace-driven attackers probe the variation of electromagnetic fields or power, to capture the profile of cache activities and deduce cache hits and misses [2, 27]. In time-driven attacks, the adversary passively measures the execution time to disclose the secret keys. In these two categories of attacks, the cache states are changed by the victim cryptographic engine (and the system activities), while an access-driven attacker actively manipulates the cache states by running a malicious process that shares caches with the victim.

2.2.1. Time-driven Side Channel

Two typical remote cache time side channel attacks on block cipher implementations are outlined as follows.

Bernstein's attack. It statically analyzes the relation between the overall execution time and lookup table indexes [4]. The attacker firstly obtains the AES execution time for each possible value of $p_i \oplus k_i$ (where p_i and k_i are the ith byte of the plaintext and key, respectively) on a duplicated server whose hardware and software configurations are the same as the victim server. It finds the value of $p_i \oplus k_i$ corresponding to the maximal average AES execution time. Then for the ith byte of the unknown key (denoted as $k_i^{'}$) on the victim server, it collects a large number of encryption times for different known

plaintexts, and obtains $p_i^{'}$ corresponding to the maximal average AES execution time. The maximal time shall result from the same index in the two phases, so the attacker infers $k_i^{'} = p_i^{'} \oplus (p_i \oplus k_i)$.
Internal collision attack. This attack exploits the fact that less time is needed for accessing the lookup table entries in the same cache line. Each 64-byte cache line has 16 4-byte entries, and in the first round of AES [3], the inputs of lookup table T_i are $p_{i+4j} \oplus k_{i+4j}$ where $0 \leqslant i, j \leqslant 3$. When any two inputs of T_i access the lookup table entries in a same cache line (i.e., the high 4 bits of $p_{i+4j} \oplus k_{i+4j} \oplus p_{i+4j'} \oplus k_{i+4j'}$ is zero), it results in less execution time. The attacker measures the execution time for all possible $p_{i+4j} \oplus p_{i+4j'}$ $(0 \leqslant j, j^{'} \leqslant 3)$. The least time at the value $p_{i+4j} \oplus p_{i+4j'} = \delta$ means that $k_{i+4j} \oplus k_{i+4j'}$ has the same high 4 bits as δ. Thus, the information of key bytes is leaked. This attack is extended to the second and last rounds of AES [19] to fully extract the secret key. It also works for DES and 3DES [9].

2.2.2. Access-driven Cache Side Channel

When the caches are shared by the attacker and the victim, the attacker could actively control the cache states and monitor the cache access by the victim process. Then, the keys are derived from the access patterns.

There are two categories of active cache side channels: synchronous and asynchronous [8]. In a synchronous attack, the attacker first sets the cache states, triggers the encryption, and monitors the memory access by detecting the change of cache states after the encryption. While in an asynchronous attack, the attacker sets the cache states and detects the changes by the victim process running concurrently. The synchronous attack monitors the cache state once per encryption, while the asynchronous one does it for several times per encryption with higher performance.

In synchronous Evict+Time attacks [8, 10], the attacker sets the cache states as follows: triggering the encryption to load data into caches and explicitly evicting some from caches, which will be monitored. Then, the attacker triggers the encryption again, and records the time to infer whether the monitored addresses are accessed or not.

Prime+Probe [10, 32] is launched in both scenarios. The attacker sets the cache states by filling one cache set with its own data, called Prime. Then, it accesses these data and measures the access time, to determine whether the monitored cache set is used by the encryption after the Prime phase, called Probe. Different cache sets are monitored one by one in synchronous attacks, or the access pattern of a cache set is detected in an asynchronous attack.

Flush+Reload [24] is another asynchronous threat that exploits shared memory. The attack process first flushes the shared memory data out of caches, and then measures the access time of some addresses in the shared memory to find whether they are accessed by the concurrent encryption or not (called Reload). Flush+Flush [33] and Prime+Abort [34] work in the asynchronous mode and exploit different methods to observe the cache access patterns. Flush+Flush is a variant of Flush+Reload, which monitors the cache state by flushing some addresses in the shared memory instead of reading them. Prime+Abort accesses the Prime data within Intel TSX, and then if some of them is accessed by the victim during the encryption, an abort occurs. So the attacker studies the cache access pattern through the abort events.

2.3. Defense against Cache Timing Side Channels

Countermeasures against the remote cache timing attacks have been proposed at different levels, and some of them also are effective against the active attacks.
Eliminating the difference of execution time. The straightforward method is to perform encryption without caches, by disabling caches [35], exploiting the no-fill cache mode [8, 36] or loading lookup tables into registers [8]. But all of them result in very low performance. Bitsliced implementations [8,

37, 38] of AES without lookup tables, effectively defend against the cache timing attacks. However, they are specific to AES, and the performance is even lower.

AES-NI is a set of hardware instructions for AES, free of timing side channels. It is available only on Intel CPUs, and Intel provides the hardware support only for AES.

Loading lookup tables into caches before encryption [8, 9, 35], or using a compact table [5, 8], eliminates cache misses remarkably, but not completely. There are still time differences exploitable to disclose the keys.

Confusing the cache access. Delay (or inserting padding instructions) after encryption, confuses the cache access leaked through the execution time. Random delay [8, 35] or delay for the constant time of encryption [26, 39–41], at the user-space or system level [42–44], were proposed. Instruction-based task scheduling [40, 45, 46] also confuses the cache access. These methods are algorithm-independent and effectively defend against the timing side-channel attacks, but result in a large performance overhead.

[35] suggests to access extra data arrays, while the permutation of lookup tables and caches is proposed in [47]. Rescheduling the instructions achieves the same goal, and is enforced at different levels [35, 48]. The masking technique is used to the cache attack-resistant algorithm implementations [8, 49]. The combination of blinding and delay is designed [50] to confuse the cache access against the cache timing side channels. These algorithm-dependent designs are not implementation-transparent, and also suffer from the performance problem to some extent.

Breaking the precision of timing confuses a local attacker's observations [35, 51, 52], but it does not work against a remote attacker with its own timer.

Limiting the cache sharing. Cache partition prevents the interference of active cache side-channel attackers. Cache coloring partitions the caches at the operating system (OS) level [40]. STEALTH-MEM [53] allows each VM to load the sensitive data into its own locked cache lines. Partition-locked caches are designed [54, 55] to prevent the cache interference. Although they defend against the cache attacks, the usage of caches is significantly degraded and then such designs are rarely deployed on commodity systems.

CATalyst [56] and Cloak [57] employ CPU hardware features to limit the cache sharing. CATalyst [56] adopts Intel Cache Allocation Technology (CAT), a way-based cache-partitioning mechanism, to divide the caches into two partitions: non-secure one and secure one. The secure applications store sensitive data and codes in the secure cache partition which will never be evicted, against active cache side-channel attack. Cloak [57] uses hardware transactional memory (HTM) to provide non-shared caches for sensitive data and codes; that is, the sensitive codes and data are executed and accessed in HTM-backed transactions which will abort when attackers attempt to probe their cache access-patterns.

Integrated methods. Different methods are integrated to defend against the cache timing attacks. The scheme [58] integrates compact lookup tables, table permutation (or randomization), and cache line preloading, so that cache misses occur as little as possible while the relationship between the secret keys and the cache misses/hits is secret-independent. Dynamic instruction padding, isolated cache resources, and lazily-cleaned cache states are integrated against cache side-channel attacks [25]. These schemes are not implementation-transparent, or require special system privileges.

3. Eliminating Remote Cache Timing Side Channels by Integrating WARM and DELAY

This section first introduces the threat model and the objectives. Then we describe the WARM+DELAY scheme, and analyze its effectiveness. It provides a foundation for the analysis of performance optimization in next sections.

3.1. Threat Model

We consider the *remote cache timing side-channel attacks* on block ciphers which are implemented with lookup tables. The cryptographic algorithms and their implementation details are publicly known, but the keys are kept secret, which are the attack targets. The table-based implementations are very efficient and widely adopted by popular cryptographic engines, e.g., AES in OpenSSL and mbed TLS[59], 3DES and Blowfish [28] in GPG and SSH. In such implementations, no branches exist, and no operation except the table lookup operation results in the cache-based execution time variation.

This threat model is adapted in most of the typical scenarios of remote cache timing side-channel attacks. The remote attackers know the software/hardware configurations of the target system, including the OS, the cache hierarchy, etc. However, the running states of the system are unknown to the attackers, due to non-deterministic interrupts, task scheduling and other system activities. The attackers are able to invoke the encryption function arbitrarily – they could continuously invoke the function so that all lookup table entries are cached, or stop using the function for a long time so that all entries are highly likely to be evicted from caches. They are able to measure the overall execution time for each execution of the block cipher.

In the following analysis in Section 3 and Section 4, we first assume that the execution time variation is only caused by the cache hits/misses of lookup table entries, by ignoring the variation caused by the system architecture issues. Then, in Section 5.1, we demonstrate the mitigations to prevent the attacks exploiting the tiny time variation caused by these issues (e.g., cache replacement policy and cache structure) [4, 60].

The cache timing side-channels by active attackers are out of the scope of this paper. That is, the attackers cannot run privileged or unprivileged processes on the target system to modify or probe the cache state. The attackers do not have physical access to the hardware, either.

3.2. Objective

This work aims to provide an in-depth investigation of the *performance* of integrating WARM and DELAY against the remote cache timing side-channel attacks for block ciphers. As the baseline, the defense shall satisfy the following conditions:

- *Algorithm-independent.* It is generally applicable to block ciphers implemented with lookup tables (so that they are vulnerable to cache timing attacks).
- *Implementation-transparent.* It does not require modification to the source code of the encryption functions.
- *Unprivileged.* It requires no privileged operations. It works well in user space or kernel space, to protect the computations in user mode and kernel mode.

More importantly, the objectives of our research are elaborated as follows:

- We attempt to develop the first quantitative model to formally analyze the performance of block cipher implementations that integrate WARM and DELAY to defend against the cache timing side-channel attacks.
- Follow the model, we examine the strategies of integrating WARM and DELAY, and present the theoretical boundaries for the optimal performance while ensuring security.
- In the optimal strategy, the parameters are configurable. By configuring appropriate parameters, the optimal performance is achieved for different block-cipher implementations and computing platforms.

3.3. The WARM+DELAY *Scheme*

There are two basic common countermeasures against remote cache timing side channels [4, 5, 8, 60]: *a*) WARM, load the lookup tables into caches *before* encryption; and *b*) DELAY, insert padding instructions *after* encryption. They work complementarily and each has its own advantages: WARM improves the performance, but cannot completely ensure security by itself because the effect may be broken by system activities; DELAY completely eliminates the timing side channels, but significantly degrades the performance.

A naive integration scheme is to perform both WARM and DELAY operation every time the protected encryption function is invoked. It introduces too many unnecessary extra operations and the performance is seriously degraded, although the security is ensured. In order to achieve the optimal performance, we perform DELAY as the last line of defense, only when the effect of WARM is broken and the execution time may leak information on the secret key i.e., is not equal to the expected execution time with all table entries cached (detailed in Section 4.2); similarly, WARM is performed only when some entries are not in caches (detailed in Section 4.3). As the scheme does not involve privileged operations, the conditions of WARM and DELAY are defined as regular timing.

When we consider that the cryptographic function is invoked continuously for large amounts of data, the time of each WARM may vary and this variation reflects the cache state after the previous encryption. We could evict all tables from caches before WARM to eliminate this variation, but it brings another extra overhead. Fortunately, we find that, the conditions to perform WARM and DELAY are related. That is, when the execution time is greater than the expected time with all table entries cached (i.e., the execution time when no cache miss occurs), DELAY is necessary to mitigate the risk and WARM is also necessary for the next encryption because some table entries are probably uncached. So, WARM is performed as a part of inserted instructions by DELAY, if it is needed. This design further improves the performance, and the variation of WARM is also masked.

The WARM+DELAY scheme is described in Algorithm 1. There are two parameters, T_{nm} and T_{mm}. T_{nm} is the time period for one encryption operation, in the case that $\underline{n}$o cache $\underline{m}$isses occur during the execution, or all table entries are in caches. T_{mm} is defined as the time period with the $\underline{m}$aximal number of cache $\underline{m}$isses, i.e., the worst execution time in the case that none of table entries is cached before the encryption. Given an implementation of block ciphers, T_{nm} and T_{mm} are fixed values on a specific computing platform.

In addition to ENCRYPT, three operations are introduced in this scheme: *a*) WARM, access lookup tables as constant data, *b*) DELAY, insert padding instructions, and *c*) GETTIME, return the current time as the conditions of WARM and DELAY. These operations are algorithms-independent and implementation-transparent, except that the memory addresses of lookup tables are needed in WARM. All these operations are supported in commodity computer systems without special privileges, either in user mode or kernel mode. We do not query the status of any cache line to find whether a table entry is in caches or not, which are generally unavailable on common computing platforms.

In general, PROTECTEDENCRYPT are invoked continuously to process large amounts of data, where the performance matters. WARM takes effect in the *next* execution of encryption. Therefore, if PROTECTEDENCRYPT has been idle for a long time, we suggest an additional "initial" invocation of WARM before the loop of PROTECTEDENCRYPT. Note that, even if this initial WARM is not performed, the security is still ensured by DELAY afterward and the performance impact is negligible if the loop is long enough.

Algorithm 1 The WARM+DELAY scheme

```
Input: key, in
Output: out
 1: function PROTECTEDENCRYPT(key, in, out)
 2:     $t1 \leftarrow$ GETTIME()
 3:     ENCRYPT($key, in, out$)
 4:     $t2 \leftarrow$ GETTIME()
 5:     if $t2 - t1 > T_{nm}$ then
 6:         WARM()
 7:         $t3 \leftarrow$ GETTIME()
 8:         if $t3 - t1 < T_{mm}$ then
 9:             DELAY($T_{mm} - t3 + t1$)
10:         end if
11:     end if
12: end function
```

3.4. Security Analysis

The timing side channels result from the relation between the measured execution time and the data processed; that is, different data processed (i.e., keys and plaintexts/ciphertexts) determines the lookup table access, resulting in different measured time as some table entries are in caches and others are not. We ensure that the measured time does not leak any exploitable information about the status of lookup tables in caches, and then destroy the relationship between the execution time and the data processed in encryption.

The execution time of ENCRYPT falls into three intervals, $(0, T_{nm}]$, (T_{nm}, T_{mm}) and $[T_{mm}, +\infty)$. We make the following treatments according to Algorithm 1:

- **Case 1**: The execution time of ENCRYPT is in $(0, T_{nm}]$. It means that this execution is done when all table entries are cached. No cache miss occurs and no information is leaked. Thus, DELAY is not performed in this case.
- **Case 2**: The execution time is in (T_{nm}, T_{mm}). It will be delayed to T_{mm}, the execution time as all accessed table entries are uncached before encryption.
- **Case 3**: The execution time is in $[T_{mm}, +\infty)$. It results from task scheduling, interrupts or other system activities. As explained in Section 3.1, the running states are random and unknown to the remote attackers, so the execution time is unexploitable. In this case, DELAY is not performed, either.

When applying the WARM+DELAY scheme, as shown in Algorithm 1, the remote attackers are (assumed to be) able to measure each execution time of PROTECTEDENCRYPT, but not internal ENCRYPT. So the measured time, i.e., the execution time of PROTECTEDENCRYPT, falls into two intervals, $(0, T_{nm}]$ and $[T_{mm}, +\infty)$.

The values of T_{nm} and T_{mm} are irrelevant to the data processed, and determined by the implementation and computing platform. When the measured time is in $(0, T_{nm}]$, it means that all lookup tables are in caches. When the time is greater than T_{mm}, it is caused by random system activities and/or the DELAY operations which destroy the relationship between the time and the data processed. Although the execution time of WARM may leak the cache states after the previous encryption, it is only a part

Table 1

Symbols in the performance analysis.

Symbol	**Description**
T_{nm}	The encryption execution time when no cache miss occurs.
T_{mm}	The encryption execution time with the maximal number of cache misses.
T_{cl}	The time cost to load a cache line of data from RAM to L1D caches.
$B_{nl}(N)$	The benefit of not loading data into N cache lines in a Warm operation.
$D_{nl}(N)$	The expected overhead (or padding by Delay) due to not loading N cache lines in a Warm operation.
$P_{\bar{a}}(N)$	The probability that N certain cache lines are not accessed after R rounds of AES encryption.
T_d	The extra time cost introduced by Delay operation. $T_d = T_{mm} - T_{nm}$.
T_w	The extra time cost introduced by Warm operation.
S_c	The state that all entries of the lookup table are in caches.
$S_{\bar{c},a}$	The state that some entries are uncached, and at least one of them is accessed during encryption.
$S_{\bar{c},\bar{a}}$	The state that some entries are uncached, but none of them is accessed during encryption.
Π_c	The probability that the system is in S_c, in the stationary distribution.
$\Pi_{\bar{c},\bar{a}}$	The probability that the system is in $S_{\bar{c},\bar{a}}$, in the stationary distribution.
$\Pi_{\bar{c},a}$	The probability that the system is in $S_{\bar{c},a}$ in the stationary distribution.

of the measured time and masked by the system activities or Delay operations. So the cache timing side-channel attacks do not work under the integration scheme in Algorithm 1.

In the following, we further explain that the integration scheme successfully prevents typical remote cache timing side-channel attacks in the literature. For Bernstein's attack [4], although the attacker obtains the maximal AES execution time for each $p_i \oplus k_i$ on the duplicated server, the measured time of the victim server reflects only system activities of the victim server. Because random and unknown system activities are involved in the measured AES execution time, the maximal time is not determined by the lookup table index; that is, it does not hold that $k_i' \oplus p_i' = p_i \oplus k_i$.

To launch an internal collision attack [3, 19], the attackers need to determine the least execution time for all possible $p_{i+4j} \oplus p_{i+4j'}$ $(0 \leqslant i, j, j' \leqslant 3)$ and exploit the high 4 bits of $p_{i+4j} \oplus p_{i+4j'}$ to disclose the high 4 bits of $k_{i+4j} \oplus k_{i+4j'}$. When protected by the Warm+Delay scheme, the execution time measured by attackers is irrelative to the inputs of the lookup table and the time variation is caused by random and unknown system activities. So the least time does not mean an internal collision, that is, $p_i \oplus k_i = p_j \oplus k_j$ does not hold. Therefore, the attackers cannot extract useful information about the keys.

4. Towards the Optimal Performance of the Warm+Delay Scheme

In this section, we apply the Warm+Delay scheme in Algorithm 1 to AES, which is applied to other block ciphers as described in Section 4.5. We first discuss the factors for the integration scheme to produce the optimized long-term performance. Then we present the conditions for the Delay operations to get the optimal performance. Meanwhile we use a statistical model to examine the Warm operations. Finally, we propose a strategy for the Warm+Delay scheme, and quantitatively prove that the optimal performance is achieved through our strategy. We list the major symbols of the analysis in Table 1.

4.1. Overview

In the WARM+DELAY scheme, the extra operations are introduced by WARM, DELAY and GETTIME. Because the cost of GETTIME is static and always necessary, the additional overhead is determined by WARM and DELAY. Therefore, to achieve the optimal performance, we shall discuss the following questions:

(1) Of WARM and DELAY, which is the preferred operation?
(2) What is the best strategy for the DELAY operations that will produce the optimal performance in the long term? The strategy includes two aspects: (2.A) When shall DELAY be imposed? and (2.B) What is the optimal delay to be imposed?
(3) What is the best strategy for the WARM operations? It also includes two aspects: (3.A) When shall WARM be imposed? and (3.B) What is the optimal amount of lookup table entries to be accessed in WARM?

When attempting to answer these questions, we focus on the optimization of the long-term performance. That is, our objective is to optimize the overall performance of encrypting large amounts of data, instead of the performance of encrypting one block (i.e., short-term performance).

The first question is relatively easy to answer. The WARM operation, which loads lookup tables into caches, will speed up the following encryption. Meanwhile, the DELAY operation, which inserts padding instructions to eliminate timing side channels, increases the overhead. Therefore, intuitively, the WARM operation is preferred over DELAY.

In our analysis (as in other implementations), an instruction loop imposes DELAY which accesses no memory, so the cache state is not changed by DELAY. The DELAY operation only changes the execution time of the current encryption, while it will *not* affect the following executions. So the answers to Question (2) are: (2.A) We should only impose DELAY when it is required for security purposes (i.e., to impose the minimal number of DELAY operations); and (2.B) The length of the imposed delay should be just enough to ensure security, but not more than that (i.e., to impose the minimal padding in each DELAY operation).

Meanwhile, the WARM operation not only changes the execution time of the current encryption, but also has a lasting effect on the next one(s). Intuitively, loading less data during one WARM operation introduces a smaller overhead for the current encryption; however, it may result in more DELAY and WARM operations in the future. In this case, optimized short-term performance may cause worse long-term performance, which is not desirable. Therefore, to answer Question (3), we need to build a quantitative model for the impacts of a partial/full WARM on future encryption. And it is the main challenge of this work.

In the next, we first examine the optimal conditions for DELAY. We prove that, it provides the optimal performance while ensuring security if delaying to T_{mm} in DELAY and performing DELAY only when the encryption time is in (T_{nm}, T_{mm}). Then we investigate the WARM operation for AES-128 with 2KB lookup tables [61]. We prove that it is statistically the most efficient to load all lookup table entries in WARM and perform it if a cache miss occurs during the previous encryption. Finally, we extend the conclusions to different key lengths of AES and different implementations, and show that, the conlusions are applicable to these typical key lengths and implementations. That is, the derived WARM+DELAY scheme achieves the optimal performance for various AES implementations with lookup tables.

4.2. The Optimal Conditions for DELAY

Theorem 1. *To effectively eliminate the cache timing side channels, in the* DELAY *operation delaying to T_{mm} imposes the minimal delay.*

Proof. From the attacker's perspective, T_{nm}, T_{mm} and any value greater than T_{mm} are three types of measured time that are unexploitable; i.e., the measured time does not reflect the cache misses/hits of certain lookup table entries.

If we delay the execution time to any value greater than T_{mm}, though the security is guaranteed, the overhead is greater than that delays to T_{mm}. On the other hand, if we delay the time to any value less than T_{mm}, a little information about the cache access leaks. Meanwhile, if we delay it to a random/pseudorandom value, which may be greater or less than T_{mm}, the attackers could exclude all results greater than T_{mm} and the other results are still exploitable. Such methods reduce the attack accuracy, but do not e-liminate the timing side channels completely. Therefore, delaying to T_{mm} imposes the minimal overhead while effectively eliminates the cache timing side channels. □

Theorem 2. *Performing the* DELAY *operation if and only if the execution time of encryption is in (T_{nm}, T_{mm}), results in the least number of* DELAY *operations.*

Proof. As described above, if the encryption execution time is equal to or less than T_{nm}, no cache miss occurs, and if the time is equal to T_{mm} or greater, it is unexploitable. In these cases, the execution time is independent of keys and plaintexts/ciphertexts, so DELAY is unnecessary.

When the execution time is in (T_{nm}, T_{mm}), it may result from cache misses due to the access of the un-cached table entries or system activities (but the overhead is less than $T_{mm} - T_{nm}$). We cannot distinguish the exact reasons without special privileges. However, the attackers may eliminate the effect of system activities by repeatedly invoking the encryption function using the same input. Therefore, DELAY is necessary in this case. □

It is easy to verify that, the DELAY operation in the scheme in Algorithm 1 satisfies Theorems 1 and 2.

4.3. The Optimal Conditions for WARM

We apply the WARM+DELAY scheme to AES-128 with a 2KB lookup table [61], as described in Section 2.1. We first determine the amount of data to be loaded in WARM for the optimal performance. Then, in order to explore the best conditions of performing the WARM operation, we use Markov Chain to model different WARM strategies and compare their performance. We show that, the best conditions of performing the WARM operation (i.e. with the optimal performance) hold in commodity computer systems.

4.3.1. The Amount of Data Loaded in WARM

A WARM operation loads the lookup tables into caches. It takes less time if only a part of the table is loaded, but it may introduce extra DELAY operations if the unloaded entries are accessed. The benefit of not loading N cache lines of data in the WARM operation is denoted as $B_{nl}(N)$, while the expected overhead due to not loading N cache lines is denoted as $D_{nl}(N)$. So the amount of data to be loaded in WARM is determined by comparing $B_{nl}(N)$ with $D_{nl}(N)$.

We denote the size of a cache line as C, and the time cost to load a cache line of data from RAM to L1D caches as T_{cl}. We have the following theorem for an AES implementation of R rounds with an L-byte lookup table ($L \gg C$).

Theorem 3. *If $B_{nl}(N) < D_{nl}(N)$ for any $0 < N \leqslant \frac{L}{C}$, loading all entries of the lookup table into caches in the* WARM *operation provides better performance than loading any part of entries, where $B_{nl}(N) = NT_{cl}$ and $D_{nl}(N) = (1-(1-\frac{NC}{L})^{16R})(T_{mm}-T_{nm})$.*

Proof. Not loading N cache lines of table entries saves the execution time of WARM, while the potential cost is the longer execution of encryption and the extra execution of DELAY. Since $L \gg C$, we needs $\frac{L}{C}$ cache lines to hold the lookup table. In each round of AES encryption, the lookup table is accessed for 16 times. We assume that, in each round, the input of `SubBytes` is random and then each table entry is accessed uniformly. Hence, the probability that N certain cache lines are not accessed after R rounds of AES encryption, denoted as $P_{\bar{a}}(N)$, is $P_{\bar{a}}(N) = (1-\frac{NC}{L})^{16R}$.

If a table entry is uncached but accessed during encryption, the execution time is greater than T_{nm} and DELAY is performed. Therefore, if N cache lines of table entries are not in caches, the probability of extra DELAY is:

$$P_d(N) = 1 - P_{\bar{a}}(N) = 1 - (1-\frac{NC}{L})^{16R} \tag{1}$$

Table 2

Benefit and expected cost of not loading N cache lines of table entries (in CPU cycles).

N	1	2	3	5	10	15	20	25	30	31	32
$B_{nl}(N)$	54.55	109.10	163.65	272.75	545.50	818.25	1091.00	1386.25	1636.50	1691.05	1745.60
$D_{nl}(N)$	2463.58	2478.92	2478.99	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00

The execution time shall be delayed to T_{mm} according to Theorem 1, while it is T_{nm} if all entries are in caches. So,

$$D_{nl}(N) = (1-(1-\frac{NC}{L})^{16R})(T_{mm}-T_{nm}) \tag{2}$$

The time cost to load a cache line of data is T_{cl}, so $B_{nl}(N) = NT_{cl}$. Therefore, we conclude that (a) if $B_{nl}(N) < D_{nl}(N)$ for any $\frac{L}{C} \geqslant N > 0$, loading all entries of the lookup table into caches in the WARM operation provides better performance; and (b) if there exists N satisfying that $B_{nl}(N) > D_{nl}(N)$, not loading N cache lines of table entries does better. $\square$

Example 1. For the AES-128 implementation with a 2KB lookup table, loading all entries of the lookup table into caches in the WARM operation provides better performance than loading any part of entries.

We finished the experiments on a Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM. T_{mm}, T_{nm}, and T_{cl} are 2834.00, 355.00, and 54.55 CPU cycles, respectively (see Section 5.1 for details). Table 2 shows $B_{nl}(N)$ and $D_{nl}(N)$, when $R = 10$, $L = 2$KB, and $C = 64$B. The cost of not loading N ($32 \geqslant N > 0$) cache lines of the lookup table is always much greater than the benefit. Thus, according to Theorem 3, loading all entries in WARM produces better performance.

On commodity systems, the cache is enough for all entries of the lookup tables. For example, AES is implemented with the lookup tables of 2KB, 4KB, 4.25KB or 5KB, and the lookup tables of DES/3DES are 2KB. Meanwhile, the typical L1D cache is 32KB or 64KB for Intel CPUs, and 16KB or 32KB for ARM CPUs.

Moreover, in our WARM+DELAY scheme, WARM is performed as a part of the DELAY operation, as shown in Algorithm 1. So the cost of WARM is further reduced, ensuring that loading all entries of the lookup table in WARM is better.

4.3.2. State Transitions of Different WARM *Strategies*

We use Markov Chain to analyze the states of the lookup table in caches during the continuous invocations of AES encryption. Then the extra costs are calculated based on the probability of each state in the stationary distributions, to determine the optimal WARM strategy. There are three states after one AES execution (i.e. ENCRYPT in Algorithm 1):

(1) S_c: all entries of the lookup table are in caches;
(2) $S_{\bar{c},a}$: some entries are uncached, and at least one of them is accessed during the AES encryption;
(3) $S_{\bar{c},\bar{a}}$: some entries are uncached, but none of them is accessed during the AES encryption.

$S_{\bar{c},a}$ and $S_{\bar{c},\bar{a}}$ represent the states where one or more cache lines of table entries are uncached or evicted from caches.

The state is estimated after each execution of AES encryption. The transition among three states is triggered by three events:

(1) WARM: Performing WARM to load all table entries into caches, and the probability is P_w.
(2) EVICTION: Some table entries may be evicted from caches when task scheduling, interrupts or other system activities occur during the execution. The probability that such an event occurs is denoted as P_e.
(3) ACCESS-UNCACHED: For one AES execution, some entries are uncached before encryption and some of them are accessed during encryption, and the probability is denoted as P_a. Actually, P_a is the same as P_d in Theorem 3; i.e., ACCESS-UNCACHED results in DELAY.

Without loss of generality, we assume that during one state transition, each of the events occurs once at most.[3] We assume that, during one state transition, WARM occurs before EVICTION if both of them occur. It is reasonable, because an EVICTION event occurring before WARM has no effect on the state of the lookup table, and then can be ignored. Since the state is observed immediately after the AES execution, ACCESS-UNCACHED occurs in the last during one state transition (if occurs). In the following analysis, we firstly ignore the partial-warm effect of ACCESS-UNCACHED in the analysis of the different strategies, and in Section 4.3.4 we show that this effect does not influence our conclusions.

We consider three strategies:

- *Conditional* WARM: performing WARM when some cache misses occur during the previous encryption execution, as done in the WARM+DELAY scheme;

[3]Even if some occurs more than once actually, they can be viewed as one event with the strengthened impact on the state transition.

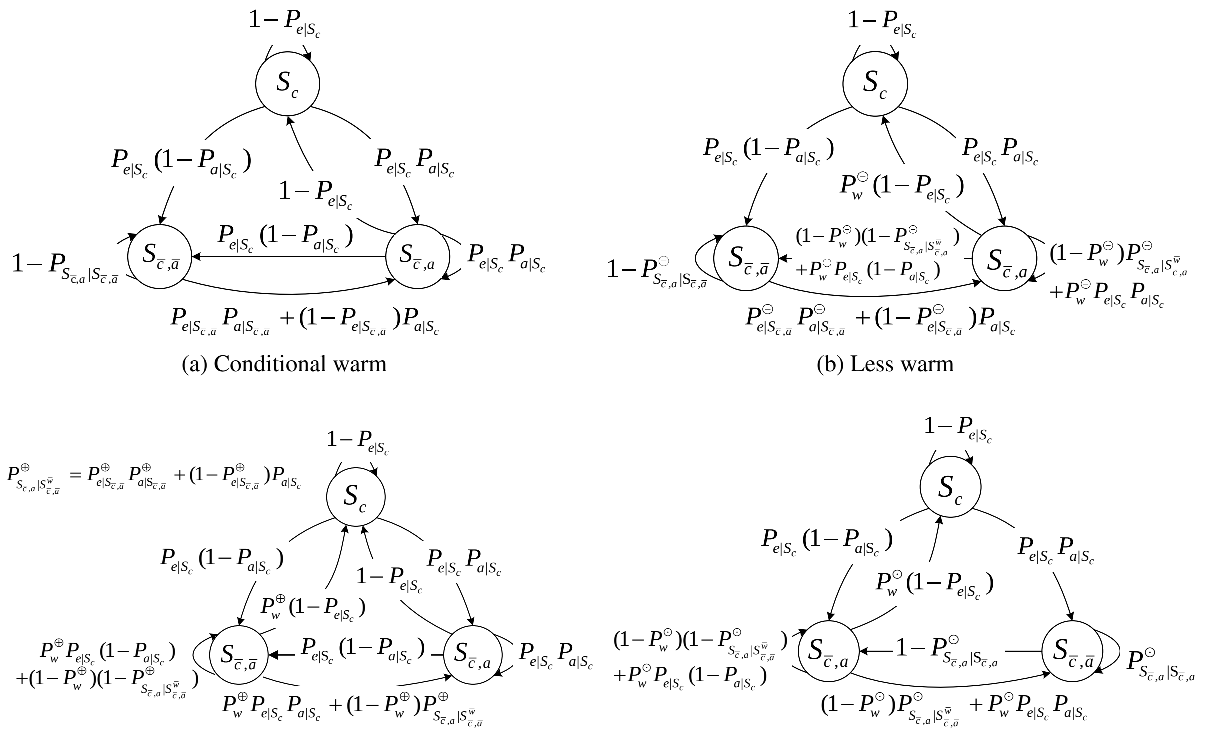
S_c
$S_{\bar{c},\bar{a}}$
$S_{\bar{c},a}$
$1-P_{e|S_c}$
$P_{e|S_c}(1-P_{a|S_c})$
$P_{e|S_c}P_{a|S_c}$
$1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$
$P_{e|S_{\bar{c},\bar{a}}}P_{a|S_{\bar{c},\bar{a}}}+(1-P_{e|S_{\bar{c},\bar{a}}})P_{a|S_c}$
(a) Conditional warm
$P_w^{\ominus}(1-P_{e|S_c})$
$(1-P_w^{\ominus})(1-P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}})+P_w^{\ominus}P_{e|S_c}(1-P_{a|S_c})$
$(1-P_w^{\ominus})P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}}+P_w^{\ominus}P_{e|S_c}P_{a|S_c}$
$1-P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$
$P^{\ominus}_{e|S_{\bar{c},\bar{a}}}P^{\ominus}_{a|S_{\bar{c},\bar{a}}}+(1-P^{\ominus}_{e|S_{\bar{c},\bar{a}}})P_{a|S_c}$
(b) Less warm
$P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}=P^{\oplus}_{e|S_{\bar{c},\bar{a}}}P^{\oplus}_{a|S_{\bar{c},\bar{a}}}+(1-P^{\oplus}_{e|S_{\bar{c},\bar{a}}})P_{a|S_c}$
$P_w^{\oplus}(1-P_{e|S_c})$
$P_w^{\oplus}P_{e|S_c}(1-P_{a|S_c})+(1-P_w^{\oplus})(1-P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}})$
$P_w^{\oplus}P_{e|S_c}P_{a|S_c}+(1-P_w^{\oplus})P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}$
(c) More warm
$P_w^{\ominus}(1-P_{e|S_c})$
$(1-P_w^{\ominus})(1-P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}})+P_w^{\ominus}P_{e|S_c}(1-P_{a|S_c})$
$1-P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a}}$
$P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a}}$
$(1-P_w^{\ominus})P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}+P_w^{\ominus}P_{e|S_c}P_{a|S_c}$
(d) Unusual warm

Fig. 1. The Markov state transition probability diagram.

- *Less* WARM: performing WARM with a probability $P_w^{\ominus} \in [0,1)$ when cache misses occur during the previous encryption execution;
- *More* WARM: performing WARM when some cache misses occur during the previous encryption execution, and also performing with a probability $P_w^{\oplus} \in (0,1]$ in other conditions.

Markov Chain is used to simulate the state transition. The state of the lookup table denoted as X_n, takes values in the state space $\{S_c, S_{\bar{c},a}, S_{\bar{c},\bar{a}}\}$. If it is in the state i, the probability that it will be in the state j after one step is denoted as $P_{j|i} = P\{X_{n+1} = j|X_n = i\}$, where $i, j \in \{S_c, S_{\bar{c},a}, S_{\bar{c},\bar{a}}\}$. Also, $P_{a|i}$ and $P_{e|i}$ represent the probabilities that ACCESS-UNCACHED and EVICTION occur in the state i, respectively, where $i \in \{S_c, S_{\bar{c},a}, S_{\bar{c},\bar{a}}\}$. Besides, we use $\ominus$ and $\oplus$ to distinguish the symbols of less and more WARM as the superscripts, respectively.

Next, the stationary distributions for the three different strategies are analyzed as follow.

Conditional WARM. The state transition of conditional WARM is shown in Fig. 1a. In the state S_c, WARM is not performed. If EVICTION does not occur, ACCESS-UNCACHED does not occur and the state remains, i.e., $P_{S_c|S_c} = 1 - P_{e|S_c}$. When EVICTION occurs, some table entries are evicted from caches. Then if these evicted entries are accessed during the AES execution, it turns to $S_{\bar{c},a}$ and $P_{S_{\bar{c},a}|S_c} = P_{e|S_c}P_{a|S_c}$; otherwise, it turns to $S_{\bar{c},\bar{a}}$ and $P_{S_{\bar{c},\bar{a}}|S_c} = P_{e|S_c}(1 - P_{a|S_c})$.

In the state $S_{\bar{c},a}$, WARM is performed to load all entries into caches, and then the state transition is similar to that in S_c. If EVICTION does not occur, it turns to S_c and $P_{S_c|S_{\bar{c},a}} = 1 - P_{e|S_c}$. When

EVICTION occurs, and if the evicted entries are accessed during encryption, the state remains the same and $P_{S_{\bar{c},a}|S_{\bar{c},a}} = P_{e|S_c}P_{a|S_c}$; otherwise, it turns to $S_{\bar{c},\bar{a}}$ and $P_{S_{\bar{c},\bar{a}}|S_{\bar{c},a}} = P_{e|S_c}(1-P_{a|S_c})$.

In the state $S_{\bar{c},\bar{a}}$, WARM is not performed because cache misses do not occur. Whether EVICTION occurs or not, some table entries are not in caches. If these uncached entries are accessed during encryption, it turns to $S_{\bar{c},a}$: (*a*) if EVICTION occurs also, the probability is $P_{e|S_{\bar{c},\bar{a}}}P_{a|S_{\bar{c},\bar{a}}}$; and (*b*) if not, the probability is $(1-P_{e|S_{\bar{c},\bar{a}}})P_{a|S_c}$. Thus, $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P_{e|S_{\bar{c},\bar{a}}}P_{a|S_{\bar{c},\bar{a}}} + (1-P_{e|S_{\bar{c},\bar{a}}})P_{a|S_c}$. If the uncached entries are not accessed, the state remains and $P_{S_{\bar{c},\bar{a}}|S_{\bar{c},\bar{a}}} = 1 - P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$.

We use Π_c, $\Pi_{\bar{c},\bar{a}}$, $\Pi_{\bar{c},a}$ to represent the stationary distribution of three states and the following equations hold:

$$\begin{cases} \Pi_c = \dfrac{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1-P_{e|S_c})}{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c})} \\[2ex] \Pi_{\bar{c},\bar{a}} = \dfrac{P_{e|S_c}(1-P_{a|S_c})}{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c})} \\[2ex] \Pi_{\bar{c},a} = \dfrac{P_{e|S_c}P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}}{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c})} \end{cases} \tag{3}$$

Less WARM. The state transition of less WARM is in Fig. 1b. In the state S_c, WARM is not performed. So the transition probability is the same as that in conditional WARM.

In the state $S_{\bar{c},a}$, WARM is performed with the probability $P^{\ominus}_w$. The state turns to S_c if WARM is performed and EVICTION does not occur, so $P^{\ominus}_{S_c|S_{\bar{c},a}} = P^{\ominus}_w P_{S_c|S_{\bar{c},a}} = P^{\ominus}_w(1-P_{e|S_c})$. The state remains if the uncached entries are accessed during the AES execution, whether WARM is performed or not. Thus, $P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a}} = (1-P^{\ominus}_w)P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} + P^{\ominus}_w P_{e|S_c}P_{a|S_c}$, where $P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} = P^{\ominus}_{e|S_{\bar{c},a}}P^{\ominus}_{a|S_{\bar{c},a}} + (1-P^{\ominus}_{e|S_{\bar{c},a}})P_{a|S_c}$ is the probability of the state transition from $S_{\bar{c},a}$ to $S_{\bar{c},a}$ when WARM is not performed. The state turns to $S_{\bar{c},\bar{a}}$ if the uncached entries are not accessed, and the probability is $P^{\ominus}_{S_{\bar{c},\bar{a}}|S_{\bar{c},a}} = (1-P^{\ominus}_w)(1-P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}}) + P^{\ominus}_w P_{e|S_c}(1-P_{a|S_c})$.

In the state $S_{\bar{c},\bar{a}}$, WARM is not performed. Whether EVICTION occurs or not, some entries are not in caches. So if these uncached entries are accessed during the AES execution, it turns to $S_{\bar{c},a}$, and the transition probability is $P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P^{\ominus}_{e|S_{\bar{c},\bar{a}}}P^{\ominus}_{a|S_{\bar{c},\bar{a}}} + (1-P^{\ominus}_{e|S_{\bar{c},\bar{a}}})P_{a|S_c}$. Otherwise, the state remains and $P^{\ominus}_{S_{\bar{c},\bar{a}}|S_{\bar{c},\bar{a}}} = 1 - P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$.

The stationary distribution of three states satisfies with the following equations:

$$\begin{cases} \Pi^{\ominus}_c = \dfrac{(1-P_{e|S_c})P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}P^{\ominus}_w}{D_L} \\[2ex] \Pi^{\ominus}_{\bar{c},\bar{a}} = \dfrac{P_{e|S_c}(1-P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}})}{D_L} + \dfrac{P_{e|S_c}(P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} - P_{a|S_c})P^{\ominus}_w}{D_L} \\[2ex] \Pi^{\ominus}_{\bar{c},a} = \dfrac{P_{e|S_c}P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}}{D_L} \end{cases} \tag{4}$$

where

$$D_L = P_{e|S_c}(1 + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}}) + (1 - P_{e|S_c})P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}P^{\ominus}_w + (P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} - P_{a|S_c})P_{e|S_c}P^{\ominus}_w.$$

More WARM. The state transition diagram of more WARM is shown in Fig. 1c. In the state S_c, WARM is performed with the probability $P^{\oplus}_w$ but has no effect. In the state $S_{\bar{c},a}$, WARM is performed to load all table entries into caches. The two situations are the same with the states of conditional WARM and also the transition probabilities.

In the state $S_{\bar{c},\bar{a}}$, WARM is performed with the probability $P^{\oplus}_w$. If WARM is performed, all entries are in caches and the state transition is the same as in the state S_c. The state turns to S_c with the probability $P^{\oplus}_{S_c|S_{\bar{c},\bar{a}}} = P^{\oplus}_w P_{S_c|S_c} = P^{\oplus}_w(1 - P_{e|S_c})$. If WARM is not performed, whether EVICTION occurs or not, some entries are not in caches. So if these uncached entries are accessed during encryption, it turns to $S_{\bar{c},a}$, and $P^{\oplus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P^{\oplus}_w P_{e|S_c}P_{a|S_c} + (1 - P^{\oplus}_w)P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}$, where $P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}$ is the probability of the state transition from $S_{\bar{c},\bar{a}}$ to $S_{\bar{c},a}$ when WARM is not performed. Otherwise, the state remains and $P^{\oplus}_{S_{\bar{c},\bar{a}}|S_{\bar{c},\bar{a}}} = P^{\oplus}_w P_{e|S_c}(1 - P_{a|S_c}) + (1 - P^{\oplus}_w)(1 - P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}})$.

The stationary distribution of three states satisfies with the following equations:

$$\begin{cases} \Pi^{\oplus}_c = \dfrac{(1 - P_{e|S_c})(P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}(1 - P^{\oplus}_w) + P^{\oplus}_w)}{D_M} \\ \Pi^{\oplus}_{\bar{c},\bar{a}} = \dfrac{P_{e|S_c}(1 - P_{a|S_c})}{D_M} \\ \Pi^{\oplus}_{\bar{c},a} = \dfrac{P_{e|S_c}P_{a|S_c}P^{\oplus}_w}{D_M} + \dfrac{P_{e|S_c}P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}(1 - P^{\oplus}_w)}{D_M} \end{cases} \tag{5}$$

where

$$D_M = P_{e|S_c} + P^{\oplus}_w + P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}(1 - P^{\oplus}_w) - P_{e|S_c}(P^{\oplus}_w + P_{a|S_c} - P_{a|S_c}P^{\oplus}_w).$$

4.3.3. The Best Strategy to Perform WARM *Operation*

We compare the performance of conditional WARM with the less WARM and more WARM strategies. Then, we investigate the relationship between conditional WARM and an arbitrary strategy. Finally, we estimate whether the conditions for conditional WARM to achieve the optimal performance are satisfied in commodity systems.

We denote the extra time cost introduced by DELAY and WARM as T_d and T_w, respectively. $T_d = T_{mm} - T_{nm}$ is constant, while T_w varies because the number of loaded cache lines changes if different WARM strategies are adopted.

Theorem 4. *The conditional* WARM *strategy provides better performance than the less* WARM *strategy.*

Proof. The expected extra time costs introduced by less WARM and conditional WARM are denoted as $E(T^{\ominus})$ and $E(T)$, respectively. DELAY is needed in $S_{\bar{c},a}$, so the extra time introduced by WARM is masked in the DELAY operation. In S_c and $S_{\bar{c},\bar{a}}$, WARM and DELAY are not needed for these two strategies. The difference between the extra time introduced by less WARM and conditional WARM is:

$$E(T^{\ominus}) - E(T) = \Pi^{\ominus}_{\bar{c},a} T_d - \Pi_{\bar{c},a} T_d \tag{6}$$

We find that $E(T^{\ominus}) - E(T) > 0$ for any $P^{\ominus}_w \in (0,1)$, and the detailed calculation is in Appendix A.1. So the extra time introduced by less WARM is always greater than conditional WARM, and conditional WARM is better. □

Next, we compare conditional WARM with more WARM. We use T^{min}_w to denote the minimum of T_w as the number of cached table entries varies, which is the time of WARM when all table entries have been in caches before WARM. The following symbols are defined: $G = T_d/T_w$, $M = T^{\min}_w/T_d$,

$$D_F = P_{e|S_c}(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c}P_{a|S_c})(P_{a|S_c} - 1) + M(1 - P_{e|S_c})P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c})),$$
$$F_G = \frac{P_{e|S_c}(P_{a|S_c} - 1)(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c}))}{D_F},$$

and $F^{\min}_G$ is the minimal value of F_G as $P_{e|S_c}$, $P_{a|S_c}$ and $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$ vary.

Theorem 5. *The conditional* WARM *strategy provides better performance than more* WARM*, if the following condition holds: (1) $D_F \geqslant 0$, or (2) $D_F < 0$ and $G < F^{\min}_G$.*

Proof. We compare $E(T)$ and $E(T^{\oplus})$ as in Theorem 4, where $E(T^{\oplus})$ is the expected extra time costs of more WARM. DELAY is needed in $S_{\bar{c},a}$, so the extra time introduced by WARM is masked in DELAY for these two strategies. In S_c and $S_{\bar{c},\bar{a}}$, WARM and DELAY are not needed for conditional WARM, but for the more WARM strategy, WARM is performed with $P^{\oplus}_w$. The extra time is T^{min}_w in S_c as all table entries are cached, and is T_w in $S_{\bar{c},\bar{a}}$. Thus,

$$E(T^{\oplus}) - E(T) = \Pi^{\oplus}_{\bar{c},a} T_d + \Pi^{\oplus}_{c} P^{\oplus}_w T^{min}_w + \Pi^{\oplus}_{\bar{c},\bar{a}} P^{\oplus}_w T_w - \Pi_{\bar{c},a} T_d \tag{7}$$

It is drawn that, $E(T^{\oplus}) \geqslant E(T)$, if (1) $D_F \geqslant 0$, or (2) $D_F < 0$ and $G < F^{\min}_G$. The details are in Appendix A.2. □

Theorem 6. *Conditional* WARM *that performs* WARM *if some cache miss occurs during the previous encryption, results in the least extra time cost of* WARM *and* DELAY*, if the following condition holds: (1) $D_F \geqslant 0$, or (2) $D_F < 0$ and $G < F^{\min}_G$.*

Proof. Firstly, consider an unusual strategy that performs WARM with a probability $P^{\odot}_w \in [0,1]$ only if cache misses do *not* occur during the previous encryption, while not WARM if any cache miss occurs. It is called *unusual* WARM, and the state transition is shown in Fig. 1d. It is proved that the conditional WARM strategy produces better performance than unusual WARM, following the same steps in Theorem 4.

Any WARM strategy W is generally described with P_1 and P_2: performs WARM with a probability $P_1 \in [0,1]$ when a cache miss occurs during the previous encryption, and with a probability $P_2 \in [0,1]$

in other states. Therefore, W is viewed as a combination of less WARM with a percentage x_1, more WARM with x_2 and unusual WARM with x_3, where:

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ x_2 P_w^{\oplus} + x_3 P_w^{\odot} = P_2 \\ x_1 P_w^{\ominus} + x_2 = P_1 \end{cases} . \tag{8}$$

It has been proved that conditional WARM always produces better performance than less WARM and unusual WARM. So conditional WARM performs better than any warm strategy, if it does better than more WARM, i.e., (1) $D_F \geqslant 0$, or (2) $D_F < 0$ and $G < F_G^{\min}$. □

Example 2. For the AES-128 implementation with a 2KB lookup table, performing WARM when cache misses occur during the previous encryption, results in the least extra time cost of DELAY and WARM.

First of all, $P_{a|S_c}$ is greater than 0.994 based on Equation 1, and the range of $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$ is $[0.994, 1]$. We measured T_d and T_w in the Lenovo ThinkCentre M8400t PC. T_d is 2479.00 CPU cycles, T_w^{min} is 88.00 cycles when accessing the lookup tables in the L1D cache, and the maximum of T_w is 1745.60 CPU cycles when all the lookup entries are not cached. Thus, $M = 0.0355$ and the rang of G is $[1.42, 28.17]$.

On this platform, the range of D_F is $[-0.000036, 0.0355]$. When $D_F \geqslant 0$, conditional WARM is the best according to Theorem 6. When $D_F < 0$, F_G is a monotone increasing function of $P_{a|S_c}$, so F_G reaches the minimum when $P_{a|S_c} = 0.994$. At this point, $P_{e|S_c}$ and $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$ satisfy:

$$\begin{cases} P_{e|S_c} = \dfrac{0.994M(1 - P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) + \sqrt{\Delta}}{0.006(1 - 0.006M)} \\ P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = \dfrac{0.07746 P_{e|S_c}}{\sqrt{M(1 - P_{e|S_c})}} - 0.006 P_{e|S_c} \\ \text{If } P_{e|S_c} \geqslant 1, P_{e|S_c} = 1 \text{ and } P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1 \\ \text{If } P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \geqslant 1, P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1 \end{cases} , \tag{9}$$

where

$$\Delta = 0.988M^2 - 1.976M^2 P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + (0.006M + 0.988M^2)(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^2.$$

We find that when $P_{e|S_c} = 1$, $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1$ and $P_{a|S_c} = 0.994$, F_G reaches the minimum and $F_G^{\min}$ is 167.67. So G is much less than $F_G^{\min}$ when $D_F < 0$.

Finally, according to Theorem 6, performing WARM when cache misses occur during the previous encryption, results in the least extra time of DELAY and WARM.

4.3.4. Remarks

In the above analysis, we ignore some effects on caches by encryption execution, task scheduling and interrupts, and these effects are discussed as below. We also summarize the relation between the measured time and the cache states.

One encryption execution is a partial WARM operation. When the system is in the state $S_{\bar{c},a}$, one encryption execution is a partial warm operation because table entries are accessed during the encryption and cache misses will load the corresponding entries into caches.

Therefore, when the system is in $S_{\bar{c},a}$, conditional WARM performs WARM with probability 1, while a general strategy actually performs WARM with a probability $P_1^{'}$ greater than P_1 in Theorem 6, due to the partial warm by the encryption execution. Although $P_1^{'}$ is different from P_1, the condition that leads to $E(T^{\oplus}) > E(T)$ does not change. So conditional WARM still produces better performance.

If task scheduling or interrupts occur during the encryption, WARM is performed always. In this case, the time of ENCRYPT is greater than T_{mm} (i.e., $t2 - t1 > T_{mm}$), WARM will be triggered in the conditional WARM strategy. When task scheduling or interrupts occur, we assume that N cache lines of the lookup table are evicted from caches. Hence, the extra overhead of WARM to load the lookup table is $W_{nl}(N) = NT_{cl} + (32 - N)T_{\bar{c}l}$, where $T_{\bar{c}l}$ and T_{cl} denote the time for accessing one cache line from caches and RAM, respectively. If we do not perform WARM, the next AES may need to access some part(s) of the lookup table from RAM instead of caches, which will trigger DELAY. The expected overhead is $D_{nl}(N)$ (see Equation 2).

Table 3 shows the results on the Lenovo ThinkCentre M8400t PC, and performing WARM achieves better performance when $N > 0$. If $N = 0$ (i.e., no table entry is evicted from caches), WARM is not needed while $t2 - t1 > T_{mm}$. However, we cannot find whether the table entry is evicted or not, unless more privileged operations are supported. So we have to perform WARM and it introduces a little extra overhead (i.e., access each cache line of data that are in caches) in this special case.

Table 3
The introduced time by performing warmup operation or not (in CPU cycles).

N	1	2	3	5	10	15	20	25	30	31	32
$D_{nl}(N)$	2463.58	2478.92	2478.99	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00
$W_{nl}(N)$	139.80	191.60	243.40	347.00	606.00	865.00	1124.00	1383.00	1642.00	1693.80	1745.60

If task scheduling is triggered but the encryption execution is not suspended and resumed immediately, WARM is still performed. Because the task is not switched, the time cost of scheduling is less than T_d. That is, $T_{nm} < t2 - t1 \leqslant T_{mm}$. In this case, WARM is unnecessary for no other task is executed and the lookup table is not evicted. However, WARM is still a better choice, as we cannot find whether a table entry is in caches or not, and the introduced overhead is concealed by DELAY.

The execution time does not exactly reflect the cache state. In the above, we prove that, WARM when not all lookup entries are in caches, results the least extra time. However, in the WARM+DELAY scheme, WARM is performed according to the previous execution time, as it is technically unable to observe the cache states. The relationship between the cache state and execution time is as follows:

- $t2 - t1 \leqslant T_{nm}$, the system is in S_c or $S_{\bar{c},\bar{a}}$, no WARM is needed according to Theorem 6.
- $t2 - t1 > T_{mm}$, it is in S_c, $S_{\bar{c},a}$ or $S_{\bar{c},\bar{a}}$, WARM is needed according to Theorem 3.
- $T_{nm} < t2 - t1 \leqslant T_{mm}$, it is in S_c or $S_{\bar{c},a}$, WARM is better according to the previous analysis.

Therefore, performing WARM according to the previous execution time is reasonable to obtain the best performance.

4.4. Different Key Lengths and Implementations

For other key lengths with different numbers of rounds and implementations with different sizes of tables, Theorems 1, 2, 3, 4, 5, and 6 still hold, but we need to re-calculate the equations (for example, in Theorem 3, T_{cl} is the maximum of T_w divided by the number of cache lines in the table). After the re-calculation, we will find whether the WARM+DELAY scheme provides the optimal performance or not.

T_w, T_{mm}, and T_{nm} will be measured on the platform. $P_d(N)$ in Theorems 3 (i.e., the probability that some cache lines of table entries not in caches are accessed, or called P_a in Section 4.3.2), depends on the structure of lookup tables. We list the calculation of P_a in Appendix B. Other variables and equations will be calculated, following the same steps as in the theorems.

4.5. Different Algorithms

For other block cipher implementations with lookup tables, Theorems 1, 2, 3, 4, 5, and 6 still hold. We still use S_c, $S_{\bar{c},a}$ and $S_{\bar{c},\bar{a}}$ to denote three states of the lookup table in caches during the continuous encryption/decryption. However, as described in Section 4.4, we need to re-calculate the parameters in Theorems 3, 4, 5, and 6 for different numbers of rounds and lookup table access in each execution of encryption/decryption, and various size of lookup tables, by using the parameters of cache line, $P_d(N)$, T_w, T_{mm}, and T_{nm} on each target platform, to find whether the WARM+DELAY scheme is optimal.

5. Performance Evaluation

We evaluate the scheme on a Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM. The CPU has 4 cores and each core has a 32KB L1 data cache, and the operating system is Linux v3.6.2 (32-bit).

5.1. Implementation

We apply WARM+DELAY to AES-128 implemented with a 2KB lookup table, directly borrowed from OpenSSL-1.0.2c [61]. As we attempt to optimize the performance, efficient WARM, GETTIME, and DELAY are finished as follows.

WARM. It loads all entries of the lookup table into the L1D cache by accessing one byte of each block of 64 bytes (i.e., the size of one cache line). In order to ensure these operations are not obsoleted due to the compiler optimization, the variables are declared with the keyword `volatile`.

However, even when all table entries are in L1D caches, the execution time of encryption still has variations and these variations could be exploited to launched side-channel attackers [4, 60]. The variations result from two types of time differences within L1D caches. One is the cache bank conflict. The L1D cache is divided physically into several banks. Each core has its own L1D cache and the banks are not across the L1D cache. Different banks can be concurrently accessed, but one bank only serves one request at a time. So, multiple access operations to the same cache bank are slower than those to different banks. Because the L1D cache is unshared and the L1D cache is not accessed by other cores, we disable Hyper-Threading of the system to ensure the protected process itself not to produce concurrent access to L1D caches with cache bank conflicts. In all the following experiments, Hyper-Threading is disabled.

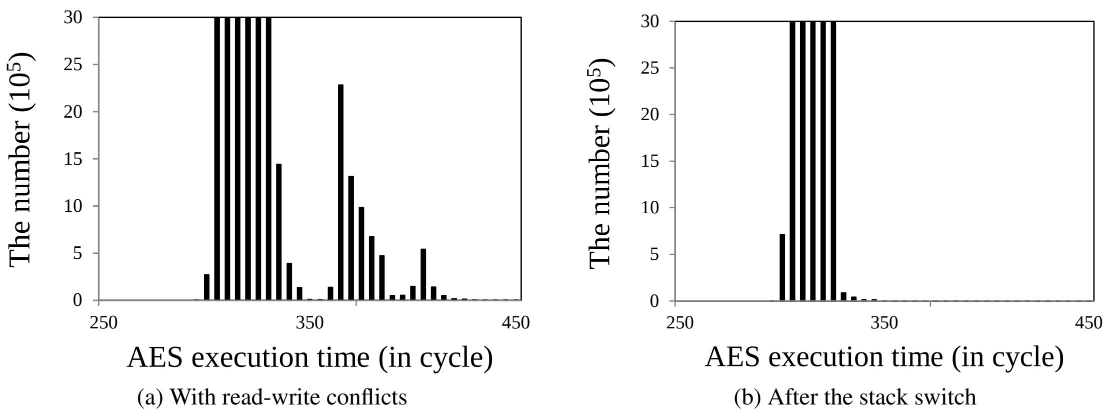

Fig. 2. The distribution of AES encryption time with a 2KB lookup table in L1D caches.

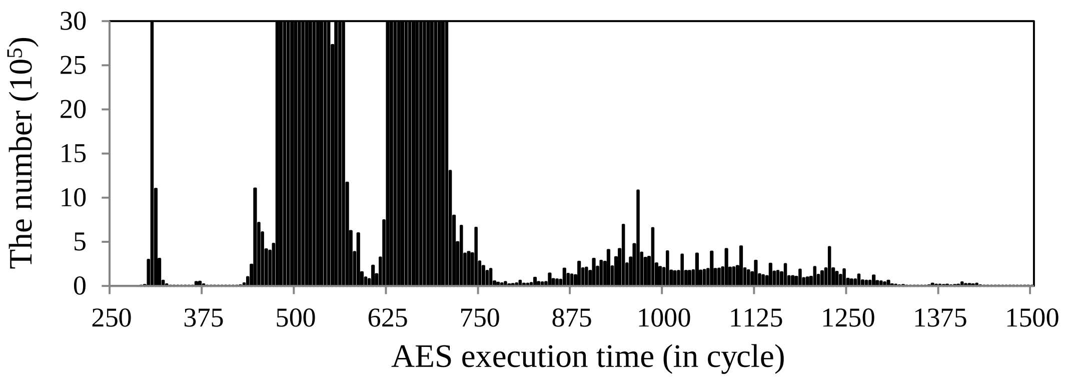

Fig. 3. The distribution of AES execution time, with one cache line of data uncached.

The other cause is the read-write conflict that the load from L1D caches takes slightly more time if it involves the same set of cache lines as a recent store. Fig. 2a shows the distribution of the AES execution time for 2^{30} random plaintexts when such a conflict occurs. We avoid the read-write conflict by switching the stack [62]. The aligned consecutive lookup table distributes in 32 cache sets due to the cache mapping rule while the total number of cache sets is 64 on our platform. We declare a 2KB global array as the stack, with which we easily control the address. The beginning address of the array is made next to the lookup table module 4096, and this makes the intermediate variables of AES execution use the remaining cache sets. Finally, the distribution of AES encryption time is shown in Fig. 2b.

GETTIME. We adopt the instruction `RDTSCP` to implement GETTIME, to obtain the current time in high precision (clock cycles) with low cost. `RDTSCP` is a serializing call which prevents the CPU from reordering it. We finished the following to achieve the high accuracy: (1) as the time stamp counters (TSC) on each core are not guaranteed to be synchronized, we install the patch [x86: unify/rewrite SMP TSC sync code] to synchronize the TSCs; (2) the clock cycle changes due to the energy-saving option, so we disable this option in BIOS to ensure the clock cycle be constant.

The cost of `RDTSCP` is 36 cycles which is much greater than the comparison operation and cannot be ignored. We perform GETTIME only if $t2 - t1 < T_{mm}$, instead of every time after WARM (Line 7 in Algorithm 1).

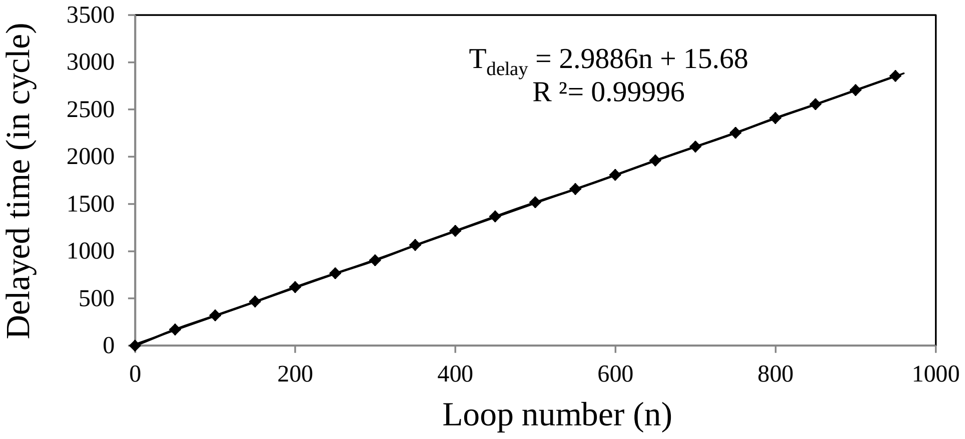

Fig. 4. The relation between the delayed time and the number of instruction loops.

DELAY. The system functions such as `nanosleep()` and `usleep()`, do not satisfy the resolution requirement, and they switch the process state to TASK_INTERRUPTIBLE, which may cause the lookup tables to be evicted from caches. On the contrary, we implement DELAY by executing the `xor` instruction repeatedly as follows, achieving a high resolution without modifying the cache state.

We measure the time for different loop numbers of the `xor` instruction, by invoking it for 10^6 times with different loop numbers (from 0 to 950 and the step is 50), as shown in Fig. 4. The relation between the time delayed and the loop number (n) is calculated through the least squares method. The result is $T_{delay} = 2.9886n + 15.68$, and the coefficient of determination is 0.99996. The implementation of DELAY() is provided in Listing 1. No table is used in this implementation, to reduce the use of caches.

Listing 1: The implementation of DELAY().

```
volatile int delay(uint64_t t_delay){
    uint64_t n = (double)t_delay>15.68 ? (uint64_t)((double) t_delay
        /2.9886-5.2466) : 0;
    for (; n>0; n--)
        asm volatile ("xor␣%%eax,␣%%eax;" : : : "%eax");
}
```

The fixed parameters (T_{nm} and T_{mm}) are set properly as follows. First of all, the implementation of block ciphers is not subject to timing side-channel attacks based on instruction caches [63], because there is no branches in the execution path. However, the instructions of block ciphers may be evicted from the caches due to system activities, which causes instruction cache misses and increases the execution time. As the effect of instruction caches on the execution time is almost indistinguishable from that of data caches, we measure T_{nm} when all instructions are cached, and T_{mm} as the execution time when all instructions are not in instruction caches (and all table entries are not in data caches). Otherwise, the measured time might leak some information about data cache access. Also, when determining the value of T_{nm} and T_{mm}, we need to take the impact of architecture into consideration.

T_{nm}. T_{nm} is greater than the minimal AES execution time when no cache miss occurs, which avoids unnecessary WARM() and DELAY() operations; and less than the execution time when only one cache miss occurs. The minimal AES execution time, is measured as the average time of 2^{30} AES execution with all table entries in the L1D cache, and it is 331 cycles in our environment.

Fig. 3 shows the distribution of AES execution time for 2^{30} plaintexts, when all table entries except one block of 64 bytes (i.e., one cache line) are in L1D caches. Note that, this uncached entry may be unnecessary in an execution of AES encryption (i.e., the results of less than 420 CPU cycles in Fig. 3). The stack switch makes the AES execution time much more concentrated, which helps the measurement of T_{nm} to be more accurate, as shown in Fig. 2. In order to avoid the influence of possible fluctuations around T_{nm} and exclude the impact of the system architecture (e.g., read-write conflicts of the memory other than the lookup tables), we choose T_{nm} as large as possible. Finally, we choose 355 cycles as T_{nm}, to ensure that all table entries are in L1D caches and no fluctuation occurs around it.

$\mathbf{T_{mm}}$. T_{mm} is the maximal encryption execution time when all table entries are out of caches. Before measuring it, we flush both the data and instructions out of L1/2/3 caches. We repeat the measurement for 10000 times, and calculate the average value of the 100 greatest measurements as T_{mm}. The result 2834 CPU cycles.

5.2. Performance Evaluation

This section shows that, with the WARM+DELAY scheme, the distribution of AES execution time are separated into two parts: less than T_{nm} and greater than T_{mm}, which meets our expectation. Then, we evaluate the performance in different aspects. We compare it with different WARM strategies to show that the integration scheme has the best performance among the different strategies integrating WARM and DELAY. We measure the performance of several different timing side-channel defenses. It is shown that WARM+DELAY produces better performance than other software-based methods, either running as an encryption service or integrated in an Apache HTTPS server.

The distribution of the encryption execution time with WARM+DELAY. We measure the AES execution time implemented with the WARM+DELAY scheme, and compare it with the unprotected version, for 2^{30} random plaintexts.

Fig. 5 shows the distributions of the AES execution time for two implementations, which will be observed by attackers. With WARM+DELAY, almost all results are in the range of (0, $T_{nm} + 2T_{GetTime}$] and [$T_{mm} + 2T_{GetTime}$, $+\infty$), where $T_{GetTime}$ is the cost of `RDTSCP`, because GETTIME is called at least twice (Lines 2 and 4 in Algorithm 1). A small set appears around the position of 1750 cycles due to task scheduling, which are unexploitable constant results. In this case, the average execution time of WARM+DELAY is less than 1.29 times of the unprotected implementation.

Performance of different WARM strategies. We compare conditional WARM with other strategies that are configured with different P_1 and P_2: perform WARM with $P_1 \in [0, 1]$ when a cache miss occurs during the previous encryption, and with $P_2 \in [0, 1]$ in other states. They are compared in different scenarios: low workload, high CPU workload and high memory workload.

With low workload, no other task except the evaluated AES implementation runs. We use the SysBench benchmark to simulate the CPU and memory workloads. We run SysBench in its CPU mode, which launches 16 threads to issue 10K requests to search the primes up to 300K, to produce the CPU workload. For the memory workload, we adopt SysBench with 16 threads in its memory mode, which reads or writes 1KB blocks each time to access total 3GB data on one CPU core. The evaluated AES implementation and the concurrent workload run on the same CPU core. The interval of task OS scheduling is set as 1 ms and 4 ms, respectively. In each case, we perform AES encryption with 2^{30} random plaintexts.

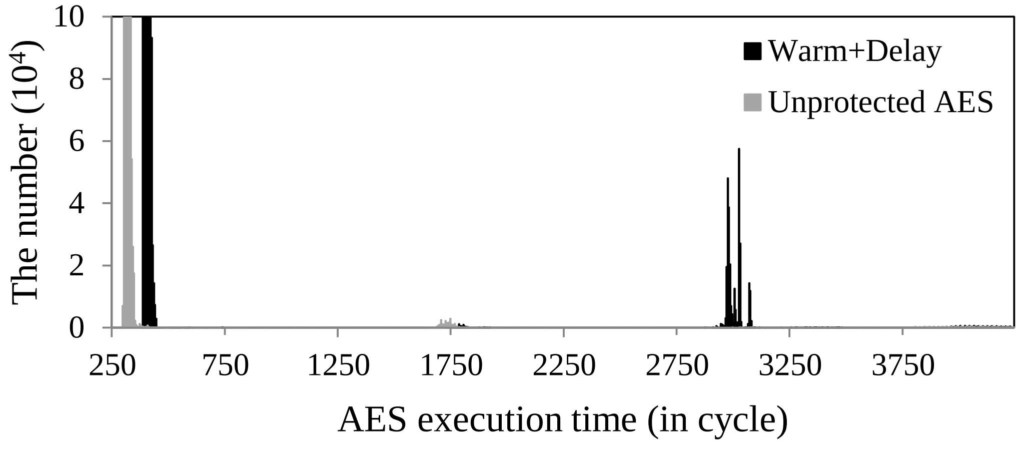

Fig. 5. The distribution of the observed AES execution time with different plaintexts.

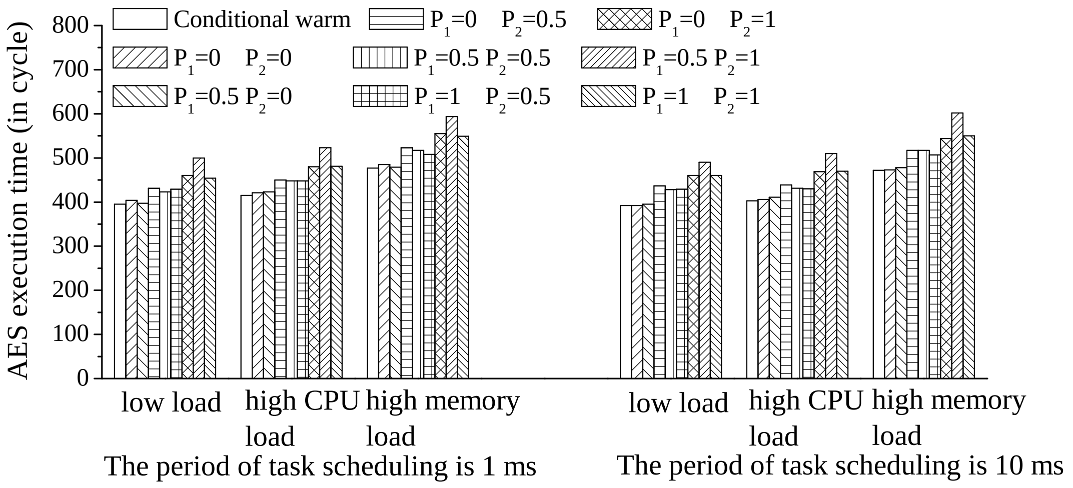

Fig. 6. AES execution performance of different WARM strategies.

Fig. 6 shows that, in any case, conditional WARM is the best, and the strategy with $P_1 = 0.5$ and $P_2 = 1$ is the worst.

Comparison with other defenses. WARM+DELAY is compared with other timing side-channel defenses: AES-NI, compact table [61] and bit-sliced implementation [64]. Each method encrypts random plaintexts for 2^{30} times.

Fig. 7 shows that, hardware AES-NI has the best performance, and the WARM+DELAY scheme is the best among all software implementations.

Performance when integrated in HTTPS servers. We applied each solution to protect the AES implementation in OpenSSL, which serves as the cryptographic engine in an Apache HTTPS server. Apache serves several web pages of different sizes with TLSv1.2. The TLS cipher suit is ECDHE-RSA-AES128-SHA256. The client runs on another computer in 1 Gbps LAN with the server. ApacheBench issues 10K requests with various request sizes, and we measure the HTTPS server throughput.

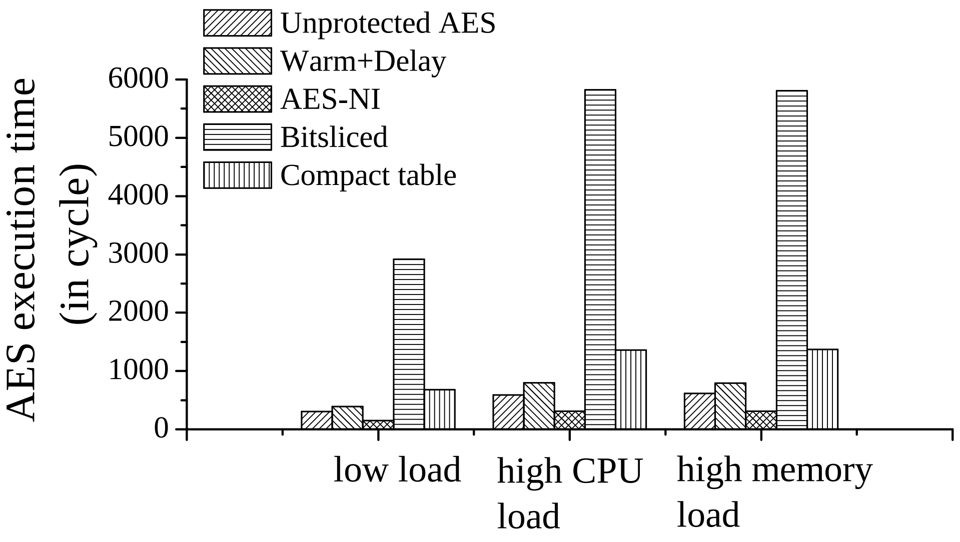

Fig. 7. Performance of different defense methods.

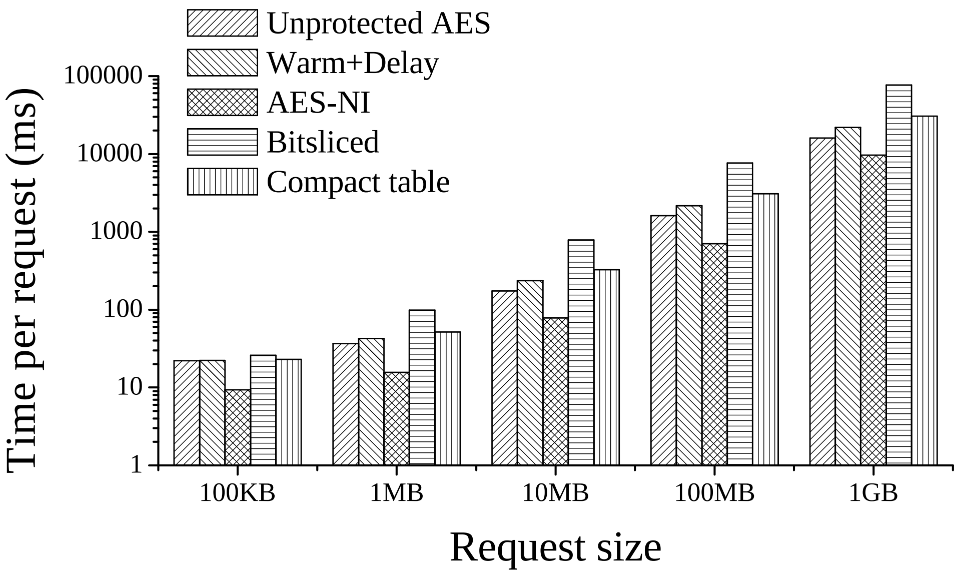

Fig. 8. Performance of different defense methods in Apache HTTPS servers

The HTTPS throughput is shown in Fig. 8. When the unprotected AES is used, the throughput is 27.2 requests per second for 1MB data. With WARM+DELAY, the throughput is 23.5 requests per second for 1MB data. WARM+DELAY scheme has a little influence on the performance. As the requested data increase, the performance influence caused by the WARM+DELAY scheme increases gradually. Meanwhile, in Fig. 8, WARM+DELAY is better than other countermeasures except hardware AES-NI.

6. Discussions

6.1. Timing Variations with Fully-Cached Lookup Tables

In the evaluation, we use the AES implementation with a 2KB lookup table. The L1D cache of 32KB, is typically divided into 64 cache sets and each set has 8 cache lines. The 2KB lookup table occupies exactly one cache line in each of 32 cache sets. So we declare a 2KB array using the other 32 cache sets as the stack, the address of which does not intersect with the lookup table module 4096.

However, when a larger lookup table is used, all cache sets will be occupied by default due to the continuity of lookup tables in memory. In these cases, to avoid the the read-write conflict of cache sets between the look table and the stack, we shall reserve the lookup table within 32 cache sets. It will be achieved by discontinuous table entries. For example, when using 4KB lookup tables (T_0, T_1, T_2, T_3), we allocate T_2/T_3 in the same memory address as T_0/T_1 module 4096, so that they occupy the same cache sets. Then the 4KB lookup tables occupy only 32 cache sets and the 2KB stack occupies the other 32 cache sets. This method works for 4.25KB or 5KB lookup tables.

6.2. Other Cache-based Timing Side Channels

The WARM+DELAY scheme is effective against remote cache timing side channels, and it also prevents synchronous access-driven attacks such as Evict+Time [8]. The synchronous access-driven attackers do not observe the cache states during the encryption execution but only monitor the cache state once per encryption, while the access pattern is masked by WARM+DELAY.

The WARM+DELAY scheme fails to prevent the trace-driven attacks and the asynchronous attacks. For the trace-driven attacks, as WARM+DELAY scheme only uses WARM to load all the lookup table entries before/after the encryption, the attackers who have the privilege to probe the cache state, infer the cryptographic key based on the cache access pattern obtained during the encryption execution. For the asynchronous attacks, in addition to probe the cache state, the attackers have the privilege to modify the cache states, which makes WARM+DELAY ineffective.

7. Conclusion

This work investigates the integration of DELAY and WARM, against remote cache-based timing side channels with the optimal performance (or with the least extra operations), for block cipher implementations with lookup tables. We theoretically derive the optimization strategy to integrate DELAY and WARM, and apply it to AES. We implement the derived integration scheme on Linux for AES-128 with a 2KB lookup table. Experimental results confirm that, (*a*) the execution time does not leak information about cache access during encryption, (*b*) the scheme achieves the optimal performance, outperforming other different integration strategies, and (*c*) the implementation works without any privileged operations on the system.

Acknowledgments

This work was partially supported by National Natural Science Foundation of China (No. 61772518) and the Information Construction Project of the 13th Five-year Plan of Chinese Academy of Sciences (XXH13505). We would like to thank the associate editor and the reviewers for the insightful comments.

Appendix A. Proofs of Theorems 4 and 5

We present the proof details of Theorems 4 and 5. In the following calculation, all probabilities are in $[0,1]$. $E(T^{\ominus})$, $E(T)$ and $E(T^{\oplus})$ are the expected extra time costs introduced by less WARM, conditional WARM and more WARM, respectively.

A.1. Proof of Theorem 4

By combining Equations 3 and 4 in Equation 6, we obtain Equation 10 as follows.

$$E(T^{\ominus}) - E(T) = \frac{Y(P_w^{\ominus})P_{e|S_c}T_d}{A_L B_L} \tag{10}$$

where

$$Y(P_w^{\ominus}) = -P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}\{P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c}(P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{a|S_c} - P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}})\}P_w^{\ominus}$$
$$+ P_{e|S_c}P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1 - P_{a|S_c}) + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c}P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1 + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}})$$
$$A_L = P_{e|S_c}(1 + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}}) + (1 - P_{e|S_c})P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}P_w^{\ominus} + (P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} - P_{a|S_c})P_{e|S_c}P_w^{\ominus}$$
$$B_L = P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c}).$$

$P^{\ominus}_{a|S_{\bar{c},a}}$ and $P_{a|S_c}$ are determined by the number of uncached table entries. The number of uncached table entries in $S_{\bar{c},a}$ is greater than in S_c, so $P^{\ominus}_{a|S_{\bar{c},a}} > P_{a|S_c}$. In the Markov state transition diagram, $P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} = P^{\ominus}_{e|S_{\bar{c},a}}P^{\ominus}_{a|S_{\bar{c},a}} + (1 - P^{\ominus}_{e|S_{\bar{c},a}})P_{a|S_c}$, and then we get $P^{\ominus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},a}} > P_{a|S_c}$. In this way, $Y(P_w^{\ominus})$ is monotone decreasing. When $P_w^{\ominus} = 1$, it gets the minimal value. At this point, $P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$ and $Y(P_w^{\ominus}) = 0$. A_L and B_L are positive, so $E(T^{\ominus}) - E(T) \geqslant 0$.

A.2. Proof of Theorem 5

DELAY is needed in $S_{\bar{c},a}$, so the extra time introduced by WARM is concealed in the DELAY operation. However, in the more WARM strategy, the extra time introduced by WARM is the minimum of T_w (T_w^{min}) in the state S_c, as all lookup entries are in caches. In the state $S_{\bar{c},\bar{a}}$, the extra time introduced by WARM is T_w. Therefore, the difference between the expected extra time introduced by more WARM and conditional WARM is:

$$E(T^{\oplus}) - E(T) = \Pi^{\oplus}_{\bar{c},a}T_d + \Pi^{\oplus}_{c}P_wT_w^{min} + \Pi^{\oplus}_{\bar{c},\bar{a}}P_wT_w - \Pi_{\bar{c},a}T_d \tag{11}$$

$P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}}$ means the probability of the state transition from $S_{\bar{c},\bar{a}}$ to $S_{\bar{c},a}$ when WARM is not performed in the more WARM strategy. If WARM is not performed in $S_{\bar{c},\bar{a}}$, the state transition is the same as with the conditional WARM strategy. So $P^{\oplus}_{S_{\bar{c},a}|S^{\bar{w}}_{\bar{c},\bar{a}}} = P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$. By combining Equations 3 and 5, we get the following equations.

$$E(T^{\oplus}) - E(T) = \frac{Y(P_w^{\oplus})P_w^{\oplus}T_w}{E_M F_M} \tag{12}$$

$$Y(P_w^{\oplus}) = B_M C_M P_w^{\oplus} + A_M C_M + G D_M \tag{13}$$

where

$$\begin{aligned}
A_M &= MG(1 - P_{e|S_c}) P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c}) \\
B_M &= MG(1 - P_{e|S_c})(1 - P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) \\
C_M &= P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c} - P_{a|S_c} P_{e|S_c} \\
D_M &= P_{e|S_c}(P_{a|S_c} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{a|S_c} P_{e|S_c}) - P_{e|S_c}((P_{a|S_c})^2 P_{e|S_c} + P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) \\
E_M &= P_{e|S_c} + P_w^{\oplus} + P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_w^{\oplus} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c}(P_w^{\oplus} + P_{a|S_c} - P_w^{\oplus} P_{a|S_c}) \\
F_M &= P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c})
\end{aligned}$$

Because E_M and F_M are greater than zero, to make $E(T^{\oplus}) - E(T) \geqslant 0$, $Y(P_w^{\oplus})$ should be equal or greater than zero. Therefore, we get $B_M C_M P_w^{\oplus} + A_M C_M + G D_M \geqslant 0$. If this inequality holds for all $P_w^{\oplus} \in [0,1]$, it is satisfied that

$$(C_M(1 - P_{e|S_c}) P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + D_M) G + C_M P_{e|S_c}(1 - P_{a|S_c}) \geqslant 0 \,. \tag{14}$$

We denote $C_M(1 - P_{e|S_c}) P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + D_M$ as D_F. Therefore,

$$\begin{aligned}
D_F = &-(P_{e|S_c})^2 (P_{a|S_c})^2 + (P_{e|S_c})^2 P_{a|S_c} + (1 - M(1 - P_{e|S_c})) P_{e|S_c} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} P_{a|S_c} \\
&+ M(1 - P_{e|S_c})(P_{e|S_c} + P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}
\end{aligned} \tag{15}$$

Thus,

(1) When $D_F \geqslant 0$, Inequality 14 holds. That is, $E(T^{\oplus}) - E(T) \geqslant 0$.
(2) When $D_F < 0$, to let Inequality 14 holds, G shall be less than $\frac{C_M P_{e|S_c}(P_{a|S_c} - 1)}{D_F}$ for $P_{a|S_c} \in [0,1]$, $P_{e|S_c} \in [0,1]$ and $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \in [0,1]$.

Next, we denote

$$F_G = \frac{C_M P_{e|S_c}(P_{a|S_c} - 1)}{D_F}$$

and compute the minimal value of F_G when $D_F < 0$.

(1) We regard F_G as the function of $P_{a|S_c}$ and calculate the derivative on $P_{a|S_c}$. Then we get

$$\frac{d(F_G)}{d(P_{a|S_c})} = (P_{e|S_c})^3 (1 - P_{a|S_c})^2 + M(1 - P_{e|S_c}) P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} (P_{e|S_c} + P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c} P_{a|S_c})^2$$

Therefore, $\frac{d(F_G)}{d(P_{a|S_c})} \geqslant 0$, so that F_G is a monotone increasing function of $P_{a|S_c}$. When $P_{a|S_c}$ fetches the minimal value, F_G gets its minimum.

(2) We regard F_G as the function of $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$. We calculate the derivative on $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$, and get

$$\frac{d(F_G)}{d(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})} = \{M(1-P_{e|S_c})(1-P_{a|S_c})(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c}))^2 - (P_{e|S_c})^2(1-P_{a|S_c})^2\}P_{e|S_c}$$

We denote $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot}$ as the greater root of the equation $\frac{d(F_G)}{d(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})} = 0$, and

$$P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot} = P_{e|S_c}\sqrt{\frac{1-P_{a|S_c}}{M(1-P_{e|S_c})}} - P_{e|S_c}(1-P_{a|S_c}).$$

When $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \in [0, P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot}]$, $\frac{d(F_G)}{d(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})} \leqslant 0$.
Therefore, when $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot} \geqslant 1$, F_G is a monotone decreasing function of $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$. When $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot} < 1$, F_G decreases at $[0, P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot}]$ and increases at $(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot}, 1]$.

(3) We regard F_G as the function of $P_{e|S_c}$. We calculate the derivative on $P_{e|S_c}$, and get

$$\frac{d(F_G)}{d(P_{e|S_c})} = P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1-P_{a|S_c})(1-M+MP_{a|S_c})(P_{e|S_c})^2 \\ - 2MP_{a|S_c}P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})P_{e|S_c} - M(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^3$$

When $P_{e|S_c} = 0$, $\frac{d(F_G)}{d(P_{e|S_c})} = -M(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^2 \leqslant 0$. Let $\frac{d(F_G)}{d(P_{e|S_c})} = 0$, and we get the greater root of this equation:

$$P_{e|S_c,rroot} = \frac{MP_{a|S_c}(1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) + \sqrt{\Delta}}{(1-P_{a|S_c})(1-M+MP_{a|S_c})} \tag{16}$$

where

$$\Delta = M^2(P_{a|S_c})^2(1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^2 + M(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^2(1-P_{a|S_c})(1-M+MP_{a|S_c})\ .$$

So when $P_{e|S_c} \in [0, P_{e|S_c,rroot}]$, $\frac{d(F_G)}{d(P_{e|S_c})} \leqslant 0$. Therefore, when $P_{e|S_c,rroot} \geqslant 1$, F_G is a monotony decreasing function of $P_{e|S_c}$. When $P_{e|S_c,rroot} < 1$, F_G decreases at $[0, P_{e|S_c,rroot}]$ and increases at $(P_{e|S_c,rroot}, 1]$.

Therefore, the minimal value of F_G gets when

$$\begin{cases} P_{e|S_c} = \frac{MP_{a|S_c}(1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) + \sqrt{\Delta}}{(1-P_{a|S_c})(1-M+MP_{a|S_c})} \\ P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P_{e|S_c}(\sqrt{\frac{1-P_{a|S_c}}{M(1-P_{e|S_c})}} - (1-P_{a|S_c})) \\ \text{if } P_{e|S_c} \geqslant 1, P_{e|S_c} = 1 \text{ and } P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1 \\ \text{if } P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \geqslant 1, P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1 \end{cases} \tag{17}$$

In summary,

(1) When $D_F \geqslant 0$, $E(T^{\oplus}) - E(T) \geqslant 0$.
(2) When $D_F < 0$, $E(T^{\oplus}) - E(T) \geqslant 0$ if G is smaller than the minimal value of F_G, where the minimal value satisfies the Equation set 17.

Finally, if (1) $D_F \geqslant 0$, or (2) $D_F < 0$ and $G < F_G^{\min}$, conditional WARM produces better performance than less WARM.

Appendix B. P_a for Different AES Key Lengthes and Implementations

The size of a cache line is C bytes. P_a represents the probability that some of the N cache line size of lookup tables not in caches are accessed. When the AES implementation uses 4.25KB lookup tables, P_a for AES-128, AES-192 and AES-256 is as follows:

$$P_a^{[128]} = 1 - \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 36),$$

$$P_a^{[192]} = 1 - \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 44),$$

$$P_a^{[256]} = 1 - \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 52),$$

where

$$f(x) = \binom{256/C}{x}(1 - \frac{xC}{256})^{16}$$

$$g(x, y) = \binom{1024/C}{x}(1 - \frac{xC}{1024})^{y}$$

$$x_0 + x_1 + x_2 + x_3 + x_4 = N$$

$$x_{4b} = \max(0, N - 4096/C)$$

$$x_{4e} = \min(256/C, N)$$

$$x_{0b} = \max(0, N - x_4 - 3072/C)$$

$$x_{0e} = \min(1024/C, N - x_4)$$

$$x_{1b} = \max(0, N - x_4 - x_0 - 2048/C)$$

$$x_{1e} = \min(1024/C, N - x_4 - x_0)$$

$$x_{2b} = \max(0, N - x_4 - x_0 - x_1 - 1024/C)$$

$$x_{2e} = \min(1024/C, N - x_4 - x_0 - x_1)$$

When the AES implementation uses 5KB lookup tables, P_a for AES-128, AES-192 and AES-256 is as follows:

$$P_a^{[128]} =1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 36),$$

$$P_a^{[192]} =1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 44),$$

$$P_a^{[256]} =1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 52),$$

where

$$\begin{aligned}
f(x,y) &= \binom{1024/C}{x}(1 - \frac{xC}{1024})^y \\
x_0 + x_1 + x_2 + x_3 + x_4 &= N \\
x_{4b} &= \max(0, N - 4096/C) \\
x_{4e} &= \min(1024/C, N) \\
x_{0b} &= \max(0, N - x_4 - 3072/C) \\
x_{0e} &= \min(1024/C, N - x_4) \\
x_{1b} &= \max(0, N - x_4 - x_0 - 2048/C) \\
x_{1e} &= \min(1024/C, N - x_4 - x_0) \\
x_{2b} &= \max(0, N - x_4 - x_0 - x_1 - 1024/C) \\
x_{2e} &= \min(1024/C, N - x_4 - x_0 - x_1)
\end{aligned}$$

When the AES implementation uses 4KB lookup tables, P_a for AES-128, AES-192 and AES-256 is as follows:

$$P_a^{[128]} =1 - \frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i, 40),$$

$$P_a^{[192]} =1 - \frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i, 48),$$

$$P_a^{[256]} = 1 - \frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i, 56),$$

where

$$f(x,y) = \binom{1024/C}{x}(1 - \frac{xC}{1024})^y$$
$$x_0 + x_1 + x_2 + x_3 = N$$
$$x_{0b} = \max(0, N - 3072/C)$$
$$x_{0e} = \min(1024/C, N)$$
$$x_{1b} = \max(0, N - x_0 - 2048/C)$$
$$x_{1e} = \min(1024/C, N - x_0)$$
$$x_{2b} = \max(0, N - x_0 - x_1 - 1024/C)$$
$$x_{2e} = \min(1024/C, N - x_0 - x_1)$$

When the AES implementation uses 2KB lookup table, P_a for AES-128, AES-192 and AES-256 is as follows:

$$P_a^{[128]} = 1 - (1 - \frac{NC}{2048})^{160},$$

$$P_a^{[192]} = 1 - (1 - \frac{NC}{2048})^{192},$$

$$P_a^{[256]} = 1 - (1 - \frac{NC}{2048})^{224}.$$

References

[1] O. Acıiçmez, W. Schindler and Ç.K. Koç, Improving Brumley and Boneh timing attack on unprotected SSL implementations, in: *ACM CCS*, 2005.
[2] O. Acıiçmez and Ç.K. Koç, Trace-driven cache attacks on AES, in: *ICICS*, 2006.
[3] J. Bonneau and I. Mironov, Cache-Collision Timing Attacks Against AES, in: *CHES*, 2006.
[4] D.J. Bernstein, Cache-timing attacks on AES, 2005, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.
[5] D. Gullasch, E. Bangerter and S. Krenn, Cache games: Bringing access-based cache attacks on AES to practice, in: *IEEE S&P*, 2011.
[6] P.C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, in: *CRYPTO*, 1996.
[7] M. Neve, J.P. Seifert and Z. Wang, A refined look at Bernstein's AES side-channel analysis, in: *ACM AsiaCCS*, 2006.
[8] D.A. Osvik, A. Shamir and E. Tromer, Cache Attacks and Countermeasures: The Case of AES, in: *CT-RSA*, 2006.
[9] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri and H. Miyauchi, Cryptanalysis of DES Implemented on Computers with Cache, in: *CHES*, 2003.

[10] E. Tromer, D.A. Osvik and A. Shamir, Efficient Cache Attacks on AES, and Countermeasures, *Journal of Cryptology* (2010).
[11] H.B. Ben Gras Kaveh Razavi and C. Giuffrida, Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks, in: *USENIX Security*, 2018.
[12] D. Genkin, L. Pachmanov, I. Pipman and E. Tromer, Stealing Keys from PCs by Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation, in: *CHES*, 2015.
[13] Y. Oren and A. Shamir, How not to protect PCs from power analysis, *Rump Session, CRYPTO* (2006).
[14] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero and G. Palermo, AES Power Attack Based on Induced Cache Miss and Countermeasure, in: *ITCC*, 2005.
[15] D. Genkin, I. Pipman and E. Tromer, Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs, in: *CHES*, 2014.
[16] D. Genkin, A. Shamir and E. Tromer, RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis, in: *CRYPTO*, 2014.
[17] B.B. Brumley and N. Tuveri, Remote timing attacks are still practical, in: *ESORICS*, 2011.
[18] D. Brumley and D. Boneh, Remote timing attacks are practical, *Computer Networks* **48**(5) (2005).
[19] O. Acıiçmez, W. Schindler and Ç.K. Koç, Cache based remote timing attack on the AES, in: *CT-RSA*, 2007.
[20] A.C. Atici, C. Yilmaz and E. Savas, Remote Cache-Timing Attack without Learning Phase, *IACR Cryptology ePrint Archive* (2016).
[21] V. Saraswat, D. Feldman, D.F. Kune and S. Das, Remote cache-timing attacks against AES, in: *CS2 Workshop*, 2014.
[22] Y. Zhang, A. Juels, M.K. Reiter and T. Ristenpart, Cross-VM side channels and their use to extract private keys, in: *ACM CCS*, 2012.
[23] T. Ristenpart, E. Tromer, H. Shacham and S. Savage, Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds, in: *ACM CCS*, 2009.
[24] Y. Yarom and K. Falkner, Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack, in: *USENIX Security*, 2014.
[25] B.A. Braun, S. Jana and D. Boneh, Robust and efficient elimination of cache and timing side channels, *arXiv:1506.00189* (2015).
[26] D. Zhang, A. Askarov and A.C. Myers, Predictive mitigation of timing channels in interactive systems, in: *ACM CCS*, 2011.
[27] J. Daemen and V. Rijmen, *The design of Rijndael: AES - The advanced encryption standard*, Springer Science & Business Media, 2013.
[28] B. Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish), in: *FSE*, 1993.
[29] C. Adams, IETF RFC 2144: The CAST-128 Encryption Algorithm, 1997.
[30] OpenSSH, http://www.openssh.com/.
[31] U. Drepper, What every programmer should know about memory, Technical Report, Red Hat, 2007.
[32] F. Liu, Y. Yarom, Q. Ge, G. Heiser and R.B. Lee, Last-level cache side-channel attacks are practical, in: *IEEE S&P*, 2015.
[33] D. Gruss, C. Maurice, K. Wagner and S. Mangard, Flush+ Flush: A fast and stealthy cache attack, in: *DIMVA*, 2016.
[34] C. Disselkoen, D. Kohlbrenner, L. Porter and D. Tullsen, Prime+ Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX, in: *USENIX Security*, 2017.
[35] D. Page, Defending against cache-based side-channel attacks, *Information Security Technical Report* (2003).
[36] Y.H. Taha, S.M. Abdulh, N.A. Sadalla and H. Elshoush, Cache-timing attack against AES crypto system - countermeasures review (2014).
[37] E. Käsper and P. Schwabe, Faster and timing-attack resistant AES-GCM, in: *CHES*, 2009.
[38] R. Könighofer, A Fast and Cache-Timing Resistant Implementation of the AES, in: *CT-RSA*, 2008.
[39] A. Askarov, D. Zhang and A.C. Myers, Predictive black-box mitigation of timing channels, in: *ACM CCS*, 2010.
[40] D. Cock, Q. Ge, T. Murray and G. Heiser, The Last Mile: An Empirical Study of Some Timing Channels on seL4, in: *ACM CCS*, 2014.
[41] C. Ferdinand, Worst case execution time prediction by static program analysis, in: *IPDPS*, 2004.
[42] Y. Zhang and M.K. Reiter, Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud, in: *ACM CCS*, 2013.
[43] J.V. Cleemput, B. Coppens and B. De Sutter, Compiler mitigations for time attacks on modern x86 processors, *TACO* (2012).
[44] B. Coppens, I. Verbauwhede, K.D. Bosschere and B.D. Sutter, Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors, in: *IEEE S&P*, 2009.
[45] D. Stefan, P. Buiras, E.Z. Yang, A. Levy, D. Terei, A. Russo and D. Mazières, Eliminating cache-based timing attacks with instruction-based scheduling, in: *ESORICS*, 2013.
[46] V. Varadarajan, T. Ristenpart and M.M. Swift, Scheduler-based Defenses against Cross-VM Side-channels, in: *USENIX Security*, 2014.

[47] J. Blömer and V. Krummel, Analysis of countermeasures against access driven cache attacks on AES, in: *SAC*, 2007.
[48] S. Crane, A. Homescu, S. Brunthaler, P. Larsen and M. Franz, Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity, in: *NDSS*, 2015.
[49] J. Blömer, J. Guajardo and V. Krummel, Provably secure masking of AES, in: *SAC*, 2004.
[50] B. Kopf and M. Durmuth, A Provably Secure and Efficient Countermeasure against Timing Attacks, in: *CSF*, 2009.
[51] P. Li, D. Gao and M.K. Reiter, Mitigating access-driven timing channels in clouds using StopWatch, in: *DSN*, 2013.
[52] R. Martin, J. Demme and S. Sethumadhavan, TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks, in: *ISCA*, 2012.
[53] T. Kim, M. Peinado and G. Mainar-Ruiz, STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud, in: *USENIX Security*, 2012.
[54] J. Kong, O. Acıiçmez, J.P. Seifert and H. Zhou, Hardware-software integrated approaches to defend against software cache-based side channel attacks, in: *HPCA*, 2009.
[55] Z. Wang and R.B. Lee, New cache designs for thwarting software cache-based side channel attacks, in: *ISCA*, 2007.
[56] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser and R.B. Lee, CATalyst: Defeating last-level cache side channel attacks in cloud computing, in: *HPCA*, 2016.
[57] F.S.O.O.I.H. Daniel Gruss Julian Lettner and M. Costa, Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory, in: *USENIX Security*, 2017.
[58] E. Brickell, G. Graunke, M. Neve and J.-P. Seifert, Software mitigations to hedge AES against cache-based software side channel vulnerabilities, *IACR Cryptology ePrint Archive* (2006).
[59] SSL Library mbed TLS / PolarSSL, https://tls.mbed.org/.
[60] A. Canteaut, C. Lauradoux and A. Seznec, Understanding cache attacks, PhD thesis, INRIA, 2006.
[61] OpenSSL: Cryptography and SSL/TLS Toolkit, https://www.openssl.org/.
[62] L. Guan, J. Lin, B. Luo and J. Jing, Copker: Computing with Private Keys without RAM, in: *NDSS*, 2014.
[63] O. Acıiçmez, Yet another MicroArchitectural Attack: Exploiting I-Cache, in: *ACM Workshop on Computer Security Architecture*, 2007.
[64] bitcoin-core/ctaes: Simple constant-time AES implementation, https://github.com/bitcoin-core/ctaes/.