Towards the Optimal Performance of Integrating WARM and DELAY to Eliminate Remote Cache Timing Side Channels

Ziqiang Ma, Quanwei Cai, Jingqiang Lin *Senior Member, IEEE,* Bo Luo *Member, IEEE,* Jiwu Jing *Member, IEEE*

***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. They destroy the relationship between the execution time and the cache misses/hits which are determined by the secret key, but introduce extra operations and then bring remarkable performance overheads. 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 common countermeasures with the optimal performance while effectively eliminate remote cache side channels. To the best of our knowledge, this work is the first to systematically analyze the performance of integrating WARM and DELAY against cache timing side channels in commodity computer systems. We derive the optimization scheme to integrate WARM and DELAY, and apply it to AES. It is proven that, the integration scheme with appropriate parameters, achieves the optimal performance with the least extra operations. Finally, we implement it on Linux with Intel Core 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 system.**

***Index Terms*—AES, cache side channel, block cipher, performance, timing side channel.**

I. INTRODUCTION

In practical cryptographic algorithm implementations, the cryptographic keys could be leaked through side channels on timing [1]–[10], electromagnetic fields [11], power [12], [13], ground electric potential [14] or acoustic emanations [15], even when the algorithm is semantically secure. Among these vulnerabilities, timing side-channel attacks are launched easily without 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], [16]–[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 these implementations, table lookup is the primary time-consuming operation in encryption and decryption. The overall execution time is significantly influenced due to the number of cache misses in table lookups. Therefore, from the execution time, attackers are able to infer the inputs of table lookup operations which are determined by the secret keys (and also 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 also 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 computer system. Moreover, such remote attacks only require the least privilege to (remotely) invoke the encryption/decryption functions, while other active cache-based side-channel attacks involve operations by a malicious task that share caches with the victim cryptographic engine [5], [22]–[24].

Different methods have been proposed against cache timing side-channel attacks, by destroying the relationship between the secret keys and the execution time. Intuitively, the basic design is to perform encryption, *a*) with the lookup tables completely inside/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* the encryption operations.[1] With WARM and/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, especially in naive implementations. In this paper, we investigate the performance of symmetric cryptographic functions that implement WARM and DELAY to defend cache timing side channel attacks, and attempt to find the optimal strategy to integrate these

Ziqiang Ma, Quanwei Cai, Jingqiang Lin, Jiwu Jing are with Data Assurance and Communications Security Research Center, and also State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, CHINA; E-mail: {maziqiang, caiquanwei, linjingqiang, jingjiwu}iie.ac.cn. Bo Luo is with Department of Electrical Engineering and Computer Science, the University of Kansas, KS 66045, USA; E-mail: bluoku.edu.

Manuscript received April 19, 2005; revised August 26, 2015.

[1] Another choice is to perform encryption without caches; however, it is very inefficient. Moreover, a typical strategy of DELAY is to extend the encryption operation in constant time, equal to the execution time without using caches.

two complementary mechanisms. In particular, the following principles are analyzed and discussed in the integration:

1) Each encryption is protected by WARM or DELAY, so that the measured time reflects either the best case (i.e., all lookup tables 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 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 lookup tables 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.
3) When DELAY is performed, WARM is done as a part of inserted instructions for the next encryption. So, the cost of WARM is masked by padding instructions, and the performance is further improved.

The integration scheme does not require any privileged operation on the computer system. 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 it produces the optimal performance. It is proven that, the integration scheme with appropriate parameters, achieves *the optimal performance with the least extra operations*; that is, without any unnecessary lookup table read operations or inserted padding instructions. The optimal performance of the integrated WARM+DELAY scheme with different key sizes (128, 192, and 256 bits) and different implementations (2KB, 4KB, 4.25KB, and 5KB lookup tables), is investigated in commodity computer systems. For example, the protected AES-128 implementation with a 2KB lookup table [27] achieves the optimal performance, when the ratio of the extra cost caused by DELAY to the extra cost caused by WARM, denoted as G, is less than 167.7. 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 computer systems. Our contributions are as follows:

- We investigate the performance overheads of WARM and DELAY, and derive the scheme to integrate these countermeasures with the optimal performance while effectively eliminate remote cache side channels. Two basic countermeasures are algorithm-independent and implementation-transparent, and the integration does not require any privileged operation on the computer system. We apply the integration scheme to AES, and analyze the situations that it produces the optimal performance.
- We implement the WARM+DELAY integration scheme on Linux with Intel Core CPUs for AES-128, and experimentally confirm that it (a) eliminates cache timing side channels, (b) outperforms other different integration strategies of WARM and DELAY, and (c) works without any privileged operations on the system.

The remainder of this paper is organized as follows. Section II presents the background and related works. Section III discusses the WARM+DELAY scheme and analyzes its security. Section IV proves that the integration scheme achieves the optimal performance, which is verified in Section V. Section VI contains extended discussions. Section VII draws the conclusions.

II. BACKGROUND AND RELATED WORKS

A. AES and Block Cipher Implementation

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

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

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

These four tables are encoded into a 2KB lookup table in the compact AES 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 $T[x]$.

There are similar implementations of other block ciphers, consisting of table lookup operations and basic computations without data-dependent branches in the execution path, such as 3DES, Blowfish [28], CAST128 [29] in OpenSSH-7.4p1 [30].

B. Cache Side Channel

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.

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

1) Cache Timing Side Channel: Cache timing side-channel attacks on block cipher implementations 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 L1 caches, while an operation in RAM takes about 250 cycles [31]. Two typical remote cache-based side channels of block ciphers are outlined as follows.

Bernstein's attack [4]. This attack statically analyzes the relation between the overall execution time and lookup table indexes. The attacker firstly obtains the mapping between average maximum overall AES execution time and each table lookup index (i.e, $p_i \oplus k_i$, where p_i and k_i are the ith byte of the plaintext and key respectively, and $0 \leq i < 16$) on a duplicated server whose hardware and software configurations are the same as the victim one. For ith byte of the unknown key ($k_i^{'}$) on the victim server, it collects a large number of overall encryption times for different known plaintexts, obtains $p_i^{'}$ corresponding to the maximum average AES execution from 256 possible values, and infers $k_i^{'} = p_i^{'} \oplus (p_i \oplus k_i)$.

Internal collision attack. This attack is based on the fact that less time is needed for accessing the lookup table entries in the same cache line. Taking the attack on first round of AES [3] as an example, on the platform where each cache line (64 bytes) has 16 entries (4 bytes), the inputs of lookup table T_i $(0 \leq i \leq 3)$ are $p_{(i+4j)\%16} \oplus k_{(i+4j)\%16}$ where $0 \leq j \leq 3$. When any two inputs of T_i require accessing the lookup table entries in a same cache line, it results in shorter overall execution time, which is called an internal collision. The attacker makes a statistics of the average execution time for all possible $p_{(i+4j)\%16} \oplus p_{(i+4j')\%16}$ $(0 \leq j, j^{'} \leq 3)$. The lowest time at the value $p_{(i+4j)\%16} \oplus p_{(i+4j')\%16} = \delta$ represents that cache collision occurs, which means $k_{(i+4j)\%16} \oplus k_{(i+4j')\%16} = \delta \mod 16$. In this way, the higher 4 bits of secret key are extracted. This attack is extended to the second and last rounds of AES [19] to extract the full secret key without extra information. Also the internal collision attack works for DES, 3DES [9].

2) Active Cache Side Channel: In the local environment if the caches are shared between attackers and victims, the attackers can actively control the cache states and monitor the cache access patterns. Thereby the keys can be derived from the access patterns. This is called access-driven cache attacks. Attackers and victims can share either the L1 Cache or LLC (last level cache). Typical access-driven cache attacks include Evict+time [8], [10], Prime+Probe [10], [32] and Flush+Reload [24].

These attacks can be divided into two categories: synchronous and asynchronous attacks [8]. In synchronous attacks, attackers can use Evict+time and Prime+Probe methods to measure the cache access patterns. Using Evict+time, attackers first trigger the encryption, then evict the cache by accessing some different memory addresses and finally trigger the encryption again and time it. The attackers repeat these steps and infer the key information from the observations. In Prime+Probe, attackers fill the cache with a known array before trigger the encryption. Then they access the array again and find the cache access pattern from the different access time. These synchronous attacks only monitor the cache once per encryption. While in asynchronous attacks, attackers run the spy process along with the encryption. Also the Prime+Probe method can be used to monitor the cache states. Attackers fill the cache with a known array and repeatedly record the time of accessing the identical array while the victim is running, Another method is Flush+Reload which needs the system supports the shared memory. Flush+Reload method monitors other processes reading and executing access to shared memory pages. Using the attack, a spy program reads access to the shared memory pages, and learns when the victim executes the code in the monitored memory locations. From this information attackers can infer the victim's confidential data. The asynchronous attacks have higher precision which can monitor several times per encryption.

There have been proposed some other attack methods like Flush+Flush [33], Prime+abort [34]. All these attacks utilize the different methods to observe the cache access pattern and then attempt to retrieve the confidential data of the victims.

C. Control against Cache Timing Side Channels

Although eliminating timing side channels remains difficult [35], different countermeasures have been proposed on the levels of hardware, software and OS. These defense methods eliminate the execution time difference of cache hits/misses, introduce confusion to the execution time to obscure the difference, or both.

Some methods which are implemented on hardware to control the cache lack universality [35]–[37]. Some using particular implementation to make algorithms constant time are just suitable to specific algorithms [38]–[40]. Some introducing many extra operations to eliminate or confuse the cache misses incur significant performance overheads [5], [8], [26], [41].

Eliminating the execution time difference. The most direct method to defend against the cache side-channel attacks is to avoid using caches. Page suggests to disable caches in paper [42], which now is unrealistical. At present, several implementations without lookup tables are proposed. AES-NI is a widely used and useful method to resist the cache attacks, which is an hardware implementation introduced by Intel. Bitsliced implementations [8], [38], [39] of AES based on software can effectively defend against the cache timing attacks. However, they are application specific and hard to design. Besides, the software implementations are suffered from high performance overload compared with using lookup tables. There are also some methods using normal lookup tables that can avoid the cache misses such as utilizing the cache no-fill mode [8], [40] and loading the lookup tables into registers [8], [41]. But all their problems are the low performance and the effect to processes running simultaneously.

Cache warm is one way to eliminating the cache misses [8], [9], [42], which means to load all the lookup tables into the caches before the encryption. Using a compact table instead also can reduce the cache misses [5], [8]. Both the two methods would introduce some overload. Furthermore, they cannot avoid all the cache misses but just reduce them to a certain extent.

Another way to eliminate cache misses is cache partition. Cache coloring is an explicit method to partition the cache on OS level [35]. STEALTHMEM [43] allows each VM to load the sensitive data into its own locked cache lines. A hardware-based mechanism presented in [36], allows caches to be configured dynamically to match the need of a process. The new cache design [44], [45] uses partition-locked caches to prevent cache interference. Another cache design in [46] reduces cache miss rates by dynamic remapping and longer cache indices. SecDCP [47] changes the size of cache partitions at run time for better performance. Although they can effectively defend against the cache timing attacks, their deficiencies are obvious as these schemes have limited practical usage and cannot be deployed on ready-made commodity hardware.

Adding confusion to cache misses. Adding delay is a common way to make the attackers obtain the confusing time information. The delay can be added in several ways such as adding random delay [8], [42], and dynamic padding which means adding delay to a constant value [26], [35], [48], [49]. Besides these software methods, OS level defense methods are also proposed such as adding noise with periodic cache cleaning [37], also making encryption time to constant values by adding delay [50], [51], and using instruction-based scheduling [35], [52], [53]. These methods are algorithm independent and can effectively defend against the cache side attacks. But the major drawback of these methods is that they incur large performance overhead.

The author in paper [42] suggests to add some dummy instructions or access some extra arrays to confusion the encryption time. A fixed number Of clock cycles AES implementation [54] is proposed by adding dynamic delay to each round. Rescheduling the instructions to confuse the cache misses is carried out in both software level [42], [55] and OS level [56]. The masking technique can be used to the cache attack-resistant algorithm implementations [8], [57]. In addition, a modified random permutation table method raised in paper [58], and a hardware-based method PRC confuse the cache misses to the attackers. The combination of blinding and delay is used in paper [59] to reach the goal of confusion. These methods need modifications to the existing implementations making them suffer from the performance problem while have limited practical usage.

Another method to confuse the attackers' observations is modifying the precision of the time measured by attackers [42], [60], [61]. But it is useless in the remote environment because the attackers can use its own timers.

Combination methods. Some schemes combine the two strategies to defend against the cache timing attack. The scheme proposed in [62] consists of three parts: using compact tables, frequently randomizing tables, and pre-loading of relevant cache-lines. It makes cache misses occur as little as possible while makes the observations secret-independent. But the drawback is the performance overload as a result that three parts all would introduce some extra overhead.

Another scheme is hold in [25], which employs dynamic padding, isolating shared resources and lazily cleansing state to form a robust defense. It considers the performance problem and decreases the reduction of performance through careful design. But the scheme needs some privileges to modify the OS kernel.

The PRET hardware architecture [63], [64] replaces caches with scratchpad memories. Also in this architecture, it will delay the encryption to the worst case execution time.

TABLE I: symbols used in this paper

Symbol	Notation
T_{NM}	the AES execution time when no cache miss occurs
T_W	the worst AES execution time when all lookup tables are out of caches
$t1, t2$	the time that AES execution begins and ends respectively
T_{cl}	the time cost to load a cache line of data from RAM to L1D cache
$B_{nl}(N)$	the benefit of not loading data into N cache lines in the WARM operation
$D_{nl}(N)$	the expected overhead (or the time cost) due to not loading N cache lines of lookup tables
$P_{\bar{a}}(N)$	the probability that N certain cache lines are not accessed after R rounds of AES encryption
P_d	the probability that performing DELAY due to not loading N lines of lookup tables
X_n	the state of the lookup table in the caches
S_c	the state that all entries of the lookup tables are in caches
$S_{\bar{c},a}$	the state that some entries are uncached, and at least one of them is accessed during the AES encryption
$S_{\bar{c},\bar{a}}$	the state that some entries are uncached, but none of them is accessed during the AES encryption
P_e	The probability that eviction occurs
P_w	the probability of performing WARM operation
P_a	the probability that some uncached entries are accessed during the AES execution
$P_{j\|i}$	if the lookup table in the caches is in the state i, the probability that it will be in the state j after one step
$P_{a\|S_c}$	the probability that some uncached entries are accessed during the AES execution when in the state S_c
$P_{a\|S_{\bar{c},\bar{a}}}$	the probability that some uncached entries are accessed during the AES execution when in the state $S_{\bar{c},\bar{a}}$
Π_c	the probability that the system is in the state S_c in the stationary distribution
$\Pi_{\bar{c},\bar{a}}$	the probability that the system is in the state $S_{\bar{c},\bar{a}}$ in the stationary distribution
$\Pi_{\bar{c},a}$	the probability that the system is in the state $S_{\bar{c},a}$ in the stationary distribution
T_d	the extra cost introduced by DELAY operation
T_w	the extra cost introduced by WARM operation
T_w^{min}	the minimum value of T_w
G	T_d/T_w
M	T_w^{min}/T_d
F_G	represents an expression that G should be less than
P_1	the probability of performing WARM when a cache miss occurs during the pervious execution of encryption
P_2	the probability of performing WARM unless a cache miss occurs during the pervious execution of encryption

III. ELIMINATING REMOTE CACHE TIMING SIDE CHANNELS BY INTEGRATING WARM AND DELAY

In this section, we present the preliminaries of our work. First, we introduce the threat model and the objectives of this

paper. We then describe the WARM+DELAY defense scheme, and show its effectiveness through security analysis. This section provides a foundation for the performance analysis and optimization in Section IV. For ease of reference, we summarize the notations commonly used in the paper in Table I.

A. The Threat Model

In this paper we consider the *remote cache timing side-channel attacks* on block ciphers. The algorithms of block ciphers and their implementation details are publicly known, but the keys are kept secret, which are the target of the attacks. In particular, lookup tables are used in the implementation of block ciphers, and the execution time highly depends on the cache access status during table lookup.

In the threat model, we assume that the remote attackers know the software/hardware configurations of the target system, including the operating system, cache hierarchy, etc. However, the running states of the target 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 tables are cached, or stop using the function for a long time so that all tables are highly likely to be evicted from caches. They are also able to measure the overall execution time for each execution of the block cipher. We assume unprivileged attackers, who cannot run privileged processes on the target system to concurrently modify or probe the cache state of the system. Other cache timing side-channels by active attackers (such as Flush+Reload [24] and Prime+Probe [32]) are out of the scope of this paper. The attackers do not have physical access to the hardware. This threat model represents most of the typical attack scenarios of remote cache-timing side-channel attacks.

B. Design Goals

This work aims to provide an in-depth investigation of the *performance* of integrating WARM and DELAY to defend against the remote cache timing side-channel attacks launched by unprivileged attackers, as we presented in the threat model. As the baseline, the defense mechanism should satisfy the following conditions:

- The solution must be algorithm-independent, hence, it is universally applicable to various block ciphers and implementations that use lookup tables (so that they are vulnerable to cache-timing side channel attacks).
- The solution must be transparent to implementations that it does not require modification to the source code of the encryption/decryption functions.
- The solution requires no privileged operations – it should work well in user space or kernel space, to protect the block ciphers running in user mode and kernel mode, respectively.

Moreover, the objectives of our research are elaborated as follows:

- We attempt to develop the first mathematical 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 mathematical model, we examine the strategies of invoking WARM and DELAY, and present the theoretical boundaries for the optimal performance while ensuring security.
- In our model, the parameters are configurable. Therefore, by configuring appropriate parameters, optimal performance is achievable for different block-cipher implementations and computing platforms.

C. The WARM+DELAY Scheme

There are two basic countermeasures against remote cache timing side channels [4], [5], [8], [65]: *a*) WARM, load the lookup tables into caches *before* encryption; and *b*) DELAY, insert padding instructions *after* encryption. These mechanisms work complementarily and each has its own advantages: WARM improves the performance, but it cannot ensure security by itself because the effect may be broken by system activities; DELAY completely eliminates the timing side channels, but the performance is significantly degraded. So we want to integrate their advantages to form the defense scheme.

A naive integration is performing both WARM and DELAY operation every time the protected cryptographic function is invoked. But 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 tables cached (detailed in Section IV-B); similarly, WARM is performed only when some entries are not in caches(detailed in Section IV-C). As the scheme does not involve privileged operations, the conditions of WARM and DELAY are defined as regular timing.

Next, when we consider that the cryptographic function is invoked continuously for large amounts of data, the time of performing WARM varies and this variation may reflect the cache access of the last encryption. We could evict all tables from caches before WARM to eliminate the 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 tables cached, DELAY is necessary to mitigate the risk and WARM is also necessary for the next encryption because some table entry is 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 masked.

The WARM+DELAY scheme is described in Algorithm 1. For easy identification, in the following we use CRYPT, DELAY and WARM to denote the encryption function, delay operation and warm operation, respectively. In this algorithm, there are two parameters, T_W and T_{NM}. T_{NM} is the time period for one encryption execution, in the case that no cache

Algorithm 1 The WARM+DELAY scheme

Input: key, in
Output: out
1: **function** PROTECTEDCRYPT(key, in, out)
2: $t1 \leftarrow$ GETTIME()
3: CRYPT(key, in, out) // encrypt or decrypt
4: $t2 \leftarrow$ GETTIME()
5: **if** $t2 - t1 > T_{NM}$ **then**
6: WARM()
7: $t3 \leftarrow$ GETTIME()
8: **if** $t3 - t1 < T_W$ **then**
9: DELAY($T_W - t3 + t1$)
10: **end if**
11: **end if**
12: **end function**

misses occur during the execution, i.e. all lookup tables are in the caches. T_W is defined as the worst execution time for one encryption, in the case that none of table entries is cached before the execution and cache misses occur at most. These two parameters are measured when there is no concurrent interrupts, task scheduling or any other system activities. Given a block cipher, T_{NM} and T_W are constant on a certain computing platform.

In addition to CRYPT, three operations are introduced by this defense scheme: *a*) WARM, access lookup tables as constant data, *b*) DELAY, insert padding instructions, and *c*) GETTIME, get the current time as the conditions of WARM and DELAY. All these operations are algorithms-independent and implementation-transparent, except that the size and memory address 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 determine whether an entry of the lookup tables is in caches or not, which are generally unavailable on common computing platforms.

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

D. Security Analysis

The cache 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 data processed during encryption.

The execution time of CRYPT falls into three intervals, $(0, T_{NM}]$, (T_{NM}, T_W) and $[T_W, \infty)$. According to Algorithm 1, we make the following treatments, respectively.

- **Case 1**: The execution time of CRYPT is in $(0, T_{NM}]$. In fact, it is almost a fixed value for arbitrary data processed, as all tables are cached and the measured time is T_{NM}. When all table entries are in caches, the access time for any entry is constant, resulting in a fixed execution time. So DELAY is not performed in this case.
- **Case 2**: The execution time of CRYPT is in (T_{NM}, T_W). It will be delayed to T_W, which is the max execution time as all accessed lookup tables are uncached before encryption.
- **Case 3**: The execution time of CRYPT is in $[T_W, \infty)$. It results from task scheduling, interrupts or other system activities. As explained in Section III-A, the running states including the system activities, are random and unknown to the remote attackers, so the execution time is unavailable. In this case DELAY is not performed, too.

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

The value of T_{NM} and T_W are irrelevant to the data processed. Also the time greater than T_W is caused by random system activities and unrelated to the data processed. So the time in this two intervals cannot be exploited in cache timing side-channel attacks. Meanwhile, due to WARM operation, that the execution time may be greater than T_{NM} is because the system activities. Which intervals the execution time will lie in is not related to the data processed. Therefore, the time obtained by the remote attackers has no relation to the data processed. The relationship between the execution time and data processed during encryption is destroyed. Thus, the cache timing side-channel attacks can not succeed under the WARM+DELAY scheme.

In the following, we further explain that the integration scheme successfully prevent typical remote cache timing side-channel attacks in the literature.

To perform the internal collision attacks [3], [19], the attackers need compile a time table $t[i, j, p_i \oplus p_j]$ $(0 \leq i, j < 16)$ for different input p_i and p_j and extract the key byte information through the relation between execution time and inputs reflected by the table. When protected by the WARM+DELAY scheme, the time in the table obtained by attackers has no relation to the inputs and the time variation is due to system activities. So the table t is no longer related to the input data. The attackers cannot extract useful information from this table.

For Bernstein's attack [4], the attackers can build the right time pattern for the duplicated server. But when building the time pattern for the target server, it will be found that, the pattern reflects only the system activities and has nothing to do with the data processed. In result, the two patterns do not have the same meaning. So the pattern for the target server cannot be used to infer the keys.

IV. Towards the Optimal Performance of Warm+Delay

In this section, we apply the WARM+DELAY scheme introduced in Algorithm 1 to AES. We first introduce the practical conditions for the WARM+DELAY scheme to produce optimized long-term performance, and then present a statistical model to examine the WARM operations. We propose a strategy for the WARM+DELAY scheme, and quantitatively prove that the optimal performance is achieved through our proposed strategy.

A. 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 operations. Therefore, to achieve the optimal performance we should answer the following questions:

(1) Between WARM and DELAY, which is the preferred operation?
(2) What is the best strategy for the DELAY operations that would result the optimal performance in the long run? The strategy includes two aspects: (2.A) When a DELAY should be imposed? and (2.B) What is the optimal DELAY time to be imposed?
(3) What is the best strategy for the WARM operations? This strategy also includes two aspects: (3.A) When a WARM should be imposed? and (3.B) What is the optimal amount of data to be loaded during WARM?
When the WARM operation is performed, what is the best strategy that would result the best performance? In particular, what is the optimal amount of data to be loaded during WARM?

Note that in answering all the questions, we focus on the optimization of the global performance. That is, our objective is to optimize the overall performance of encrypting large mount of data (i.e. optimal performance in the long run), instead of the performance of encrypting just one block (short-term performance). In this section, we will provide a comprehensive quantitative analysis on how to achieve this long-term optimal.

The first question is relatively easy to answer. The WARM operation, which loads (all) lookup tables into caches, will speed up the encryption. Meanwhile, the DELAY operation, which adds delay to eliminate timing side-channels, increases the overhead of the encryption. Therefore, intuitively, the WARM operation is always preferred over DELAY in the WARM+DELAY scheme.

Next, in our design (as in other implementations of DELAY), we use a instruction loop to impose DELAY, hence, the cache state is not changed. That says, the DELAY operation only changes the execution time of the current AES encryption, while it will *not* affect the following AES executions. Therefore, to answer question (2): (2.A) we should only impose DELAY when it is required for security purposes (i.e., to impose 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 minimal length of delay in each DELAY operation).

Meanwhile, the WARM operation not only changes the execution time of the current AES, but also has a lasting effect on the subsequent AES encryptions. Intuitively, loading less data during one WARM operation may introduce less overhead for the current encryption, however, it might result more DELAY and WARM operations in future encryptions. 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 encryptions. This will be the main task of this section.

In the following subsections, we first examine the optimal conditions for DELAY operation. We prove that it provides the optimal performance while ensuring security if delaying to T_W in DELAY and performing it when the encryption time is in (T_{NM}, T_W). Then we investigate the WARM operation for AES-128 with 2KB lookup tables [66]. We prove that it is statistically the most efficient to load all lookup tables in WARM and perform it when 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 conditions 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.

B. The Optimal Conditions for DELAY Operation

Theorem 1. *To effectively eliminate the cache timing side channels, in the* DELAY *operation T_W is the minimal delay.*

Proof: From the attackers' point of view, T_{NM}, T_W, and any value greater than T_W are three types of measured time that are unexploitable. (i.e. the execution time does not reflect the cache misses/hits of *specific* lookup table entries)

If we delay the execution time to any value greater than T_W, though the security is guaranteed, the overhead is larger than that delays to T_W.

On the other hand, if we delay the execution time to any value less than T_W, a little information about the cache access leaks. Meanwhile if we delay it to a random value, which may be greater or less than T_W, the attackers could exclude all results greater than T_W and the other results are still exploitable. Such methods reduce the attack accuracy on each invocation, but does not eliminate the cache timing side channels completely. Therefore, delay to T_W 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_W) results in the least number of* DELAY *operations.*

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

When the execution time is in (T_{NM}, T_W), it may be due to cache misses, and/or system activities (but the overhead is less than $T_W - T_{NM}$). We cannot distinguish the exact reasons without special privileges. However, the attackers can distinguish them by repeatedly invoking the cryptographic function using the same input (and secret key). Therefore, DELAY is necessary in this case. ∎

In our WARM+DELAY scheme, the DELAY operation satisfies the Theorem 1 and 2. So the use of DELAY can achieve the optimal performance in the WARM+DELAY scheme.

C. The Optimal Conditions for WARM Operation

We apply the WARM+DELAY scheme to AES-128 with a 2KB lookup table [66], as described in Section II-A. We first determine the amount of data to be loaded in the WARM operation for the optimal performance. Then to explore the best occasion of performing WARM operation, we use Markov Chain to model different warm strategies and compare their performance. We show that, in this case, the WARM+DELAY scheme produces the best performance in commodity computer systems. The details about other key sizes (192 and 256 bits) and different implementations (4KB, 4.25KB, and 5KB lookup tables), are included in Section IV-D.

1. The Amount of Data To Be Loaded in WARM Operation. WARM operation needs to load the lookup table into the caches. It takes less time that loading parts of the table. But it also may introduce extra DELAY operation if the unloaded parts are accessed. The benefit of not loading N cache lines data in the WARM operation is denoted as $B_{nl}(N)$, while the expected overhead due to not loading N cache lines of table entries is denoted as $D_{nl}(N)$. The amount of data to be loaded in WARM operation can be 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 cache as T_{cl}. Then 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 $0 < N \leq \frac{L}{C}$, 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 $D_{nl}(N) = (1 - (1 - \frac{NC}{L})^{16R})(T_W - T_{NM})$.*

Proof: Not loading N ($\frac{L}{C} \geq N > 0$) 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 the execution of AES 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 as follows.

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

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

$$\begin{aligned} D_{nl}(N) &= P_d(N)(T_W - T_{NM}) \\ &= (1 - (1 - \frac{NC}{L})^{16R})(T_W - T_{NM}). \end{aligned} \quad (2)$$

On the other hand, the time cost to load a cache line of data from RAM to L1D cache is T_{cl}, so

$$B_{nl}(N) = NT_{cl}. \quad (3)$$

Therefore, we can conclude that (1) if $B_{nl}(N) < D_{nl}(N)$ for any $\frac{L}{C} \geq N > 0$, loading all entries of the lookup table into caches in the WARM operation provides the optimal performance; and (2) if there exists N satisfying that $B_{nl}(N) > D_{nl}(N)$, not loading N cache lines of table entries provides better performance. ∎

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

Proof: Firstly, with commodity hardware, the cache is enough to load 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 table of DES/3DES is 2KB. However, the typical L1D cache is 32KB or 64KB for Intel CPUs, and 16KB or 32KB for ARM CPUs.

In our experiments on a Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM, T_W, T_{NM}, and T_{cl} are $2834, 355, 55.68$ in CPU cycles, respectively (see Section V-A for details). Table II shows $B_{nl}(N)$ and $D_{nl}(N)$, when $R = 10$, $L =$ 2KB, and $C =$ 64B. We find that, the cost of not loading N ($32 \geq N > 0$) cache lines of lookup table is always much greater than the benefit. The result is applicable to other commodity computer systems. So by Theorem 3, loading all entries in WARM produces the optimal performance. ∎

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

2. State Transition of Warm Strategies. To determine the best warm opportunity, we should compare the costs introduced by different warm strategies. We use Markov Chain to analyze the different states of the lookup table in the L1 cache during the continuous invocations of AES encryption. Then the extra costs can be calculated based on the probability of each state in the stationary distributions. There are three states:

1) S_c: all entries of the lookup tables are in caches,
2) $S_{\bar{c},a}$: some entries are uncached, and at least one of them is accessed during the AES encryption,

TABLE II: 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
$B_{nl}(N)$	55.68	111.35	167.03	278.38	556.75	835.13	1113.50	1391.88	1670.25	1725.93	1781.60
$D_{nl}(N)$	2463.58	2478.92	2478.99	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00	2479.00

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 of the lookup table is estimated after each execution of AES encryption. The transition among the three states can be triggered by three events:

1) *Warm* event: performing WARM to load all the lookup tables into the cache which probability is denoted as P_w.
2) *Evict* event: the entries of lookup tables may be evicted from the cache if task scheduling, interrupts or any other system activities occur during the execution of encryptions. The probability that eviction occurs is denoted as P_e.
3) *Execution* event: For one AES encryption, some entries are uncached before encryption and it is possible that some of them is accessed during the encryption, and this probability is denoted as P_a. Actually, P_a equals P_d in Theorem 3.

The occurrence of these three events will change the state of lookup tables in the cache. We assume that during one state transition, each of the three events occurs once at most. Because the state changes after the *Execution* event, the *Execution* event occurs the last. Besides we assume that the *Warm* event occurs after the *Evict* event. It is reasonable because if the Warm event occurs, the Evict event before it has no effect on the state of lookup tables. In the following, we temporarily do not consider the effect of the partial warm caused by the *Execution* event.

We consider three specific warm strategies:

- conditional WARM: performing WARM when some cache misses occur during the pervious execution of encryption as done in WARM+DELAY scheme;
- less WARM: performing WARM with a probability $P_w^{\ominus} \in [0,1)$ when cache misses occur during the pervious execution of encryption;
- more WARM: performing WARM when some cache misses occur during the pervious execution of encryption and also performing WARM with a probability $P_w^{\oplus} \in [0,1]$ in other conditions.

The Markov Chain can be used to simulate the state transitions. The state of the lookup table in the caches denoted as X_n takes values in the state space $\{S_c, S_{\bar{c},a}, S_{\bar{c},\bar{a}}\}$. If the lookup table in the caches is in the state i, the probability that it will be in the state j after one step is given by $P_{j|i}$, where i and j are in the state space. That is

$$P\{X_{n+1} = j | X_n = i\} = P_{j|i} \quad , \quad i,j \in \{S_c, S_{\bar{c},a}, S_{\bar{c},\bar{a}}\}.$$

The stationary distributions for the three different strategies are as follow.

2.a. Conditional WARM. The state transition diagram of conditional WARM is shown in Fig. 1a. $P_{a|S_c}$ and $P_{a|S_{\bar{c},\bar{a}}}$ represent the probability that some entries of lookup tables which are uncached before encryption are accessed during the encryption. Due to the difference of the quantity of entries of lookup tables not in the cache, $P_{a|S_{\bar{c},\bar{a}}}$ is no less than $P_{a|S_c}$.

We use Π_c, $\Pi_{\bar{c},\bar{a}}$, $\Pi_{\bar{c},a}$ to represent the stationary distribution of the three states and the values are as follows.

$$\begin{cases} \Pi_c = \dfrac{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1-P_{e|S_c})}{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c})} \\ \Pi_{\bar{c},\bar{a}} = \dfrac{P_{e|S_c}(1-P_{a|S_c})}{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c})} \\ \Pi_{\bar{c},a} = \dfrac{P_{e|S_c}P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}}{P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1-P_{a|S_c})} \end{cases} . \tag{4}$$

2.b. Less WARM. The state transition diagram of less WARM is shown in Fig. 1b. $P_{a|S_c}$ and $P^{\ominus}_{a|S_{\bar{c},\bar{a}}}$ represent the probability that some entries of lookup tables which are uncached before encryption are accessed during the encryption. As the same in conditional WARM, $P^{\ominus}_{a|S_{\bar{c},\bar{a}}}$ is no less than $P_{a|S_c}$.

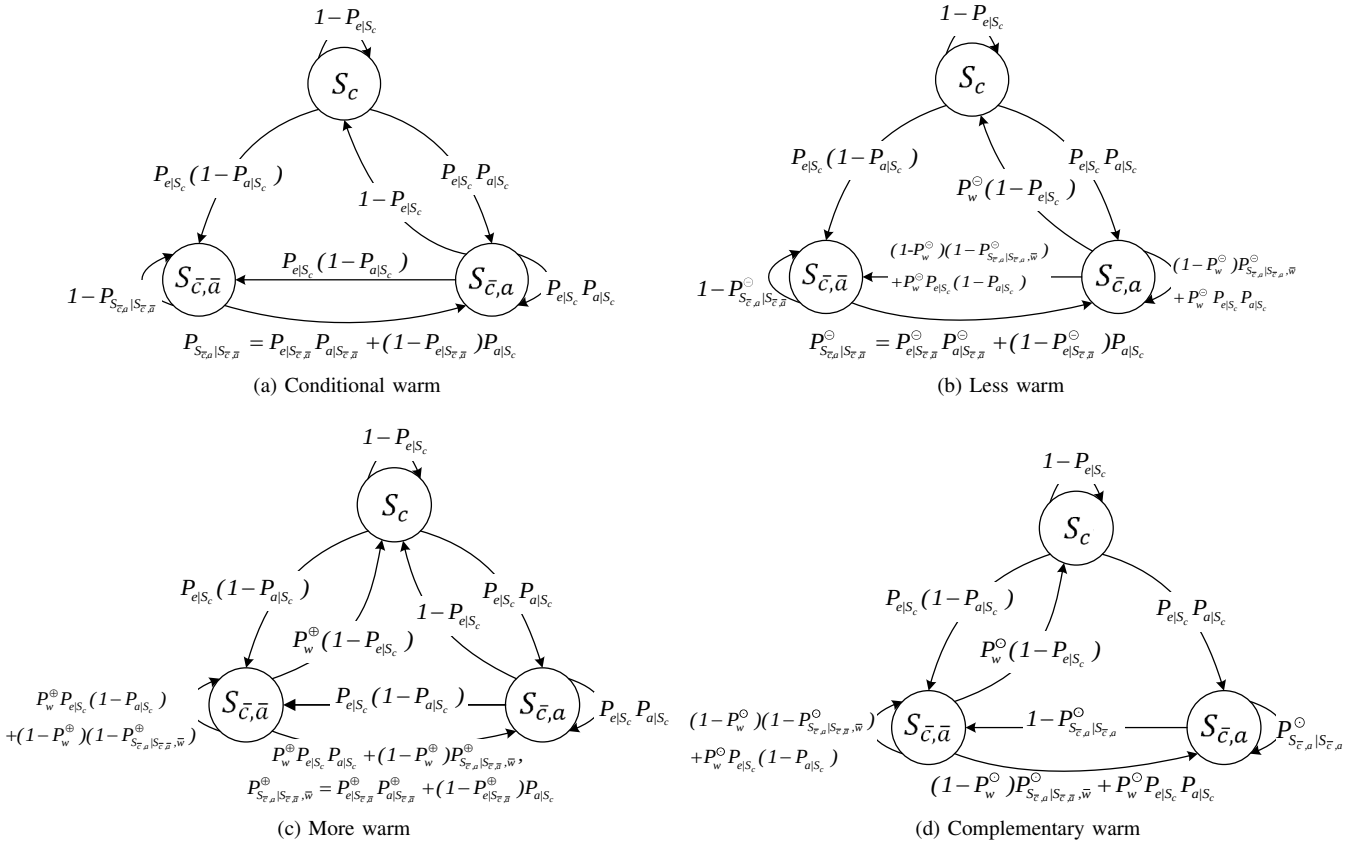
We use $\Pi_c^{\ominus}$, $\Pi_{\bar{c},\bar{a}}^{\ominus}$, $\Pi_{\bar{c},a}^{\ominus}$ to represent the stationary distribution of the three states and the values are as follows.

$$\begin{cases} \Pi_c^{\ominus} = \dfrac{(1-P_{e|S_c})P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}P_w^{\ominus}}{D_L} \\ \Pi_{\bar{c},\bar{a}}^{\ominus} = \dfrac{P_{e|S_c}(1-P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}})}{D_L} \\ \qquad + \dfrac{P_{e|S_c}(P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}} - P_{a|S_c})P_w^{\ominus}}{D_L} \\ \Pi_{\bar{c},a}^{\ominus} = \dfrac{P_{e|S_c}P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}}{D_L} \end{cases} , \tag{5}$$

where

$$\begin{aligned} D_L = {} & P_{e|S_c}(1 + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}}) \\ & + (1-P_{e|S_c})P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}P_w^{\ominus} \\ & + (P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}} - P_{a|S_c})P_{e|S_c}P_w^{\ominus}. \end{aligned}$$

2.c. More WARM. The state transition diagram of more WARM is shown in Fig. 1c. $P_{a|S_c}$ and $P^{\oplus}_{a|S_{\bar{c},\bar{a}}}$ represent the probability that some entries of lookup tables which are uncached before encryption are accessed during the encryption. As the same in conditional WARM, $P^{\oplus}_{a|S_{\bar{c},\bar{a}}}$ is no less than $P_{a|S_c}$.



(a) Conditional warm (b) Less warm

(c) More warm (d) Complementary warm

Fig. 1: The Markov state transition probability diagram.

We use $\Pi_c^{\oplus}$, $\Pi_{\bar{c},\bar{a}}^{\oplus}$, $\Pi_{\bar{c},a}^{\oplus}$ to represent the stationary distribution of the three states and the values are as follows.

$$\begin{cases} \Pi_c^{\oplus} = \dfrac{(1-P_{e|S_c})(P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},\bar{w}}(1-P_w^{\oplus})+P_w^{\oplus})}{D_M} \\ \Pi_{\bar{c},\bar{a}}^{\oplus} = \dfrac{P_{e|S_c}(1-P_{a|S_c})}{D_M} \\ \Pi_{\bar{c},a}^{\oplus} = \dfrac{P_{e|S_c}P_{a|S_c}P_w^{\oplus}}{D_M} + \dfrac{P_{e|S_c}P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},\bar{w}}(1-P_w^{\oplus})}{D_M} \end{cases}, \tag{6}$$

where

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

3. The Best Opportunity to Perform WARM Operation. We first compare the performance of conditional WARM with less WARM and more WARM strategies. Followed we investigate the relationship on the performance between conditional WARM and any warm strategies. Finally, we estimate whether the conditional WARM has the optimal performance in commodity computer systems.

We denote the extra cost introduced by DELAY and WARM as T_d and T_w, respectively. $T_d = T_W - T_{NM}$ is constant for different executions. T_w depends on the number of loaded cache lines of table entries.

3.a. Comparing Conditional WARM with Less WARM. By comparing the expected extra time introduced due to the two strategies, we can get the following theorem.

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

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

$$E(T^{\ominus}) - E(T) = \Pi_{\bar{c},a}^{\ominus} * T_d - \Pi_{\bar{c},a} * T_d \tag{7}$$

And then, we get that $E(T^{\ominus}) > E(T)$ for all $P_w^{\ominus} \in (0,1)$. So the extra time introduced by less WARM strategy is always larger than conditional WARM strategy. That is the conditional WARM strategy has better performance than the less WARM strategy. ■

3.b. Comparing Conditional WARM with More WARM. By comparing the expected extra time introduced due to the two

strategies, we can get the following theorem. First we denote:

$$
\begin{aligned}
&G = T_d/T_w \ , \\
&M = T_w^{min}/T_d \ , \\
&D_F = P_{e|S_c}(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c}P_{a|S_c})(P_{a|S_c} - 1) \\
&\quad + M(1 - P_{e|S_c})P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c})) \ , \\
&F_G = \frac{P_{e|S_c}(P_{a|S_c} - 1)(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c}))}{D_F} \ .
\end{aligned}
$$

Theorem 5. *Conditional* WARM *strategy provides better performance than the more* WARM *strategy if the following condition is satisfied:*

$$
\begin{cases} D_F \geq 0 \\ \textbf{\textit{or}} \ \ D_F < 0 \ and \ G < \min(F_G) \end{cases} .
$$

Proof: The process of proof is the same as Theorem 4 that comparing $E(T)$ with $E(T^{\oplus})$:

$$
\begin{aligned}
E(T^{\oplus}) - E(T) = \Pi_{\bar{c},a}^{\oplus} * T_d + \Pi_c^{\oplus} * P_w^{\oplus} * T_w^{min} \\
+ \Pi_{\bar{c},\bar{a}}^{\oplus} * P_w^{\oplus} * T_w - \Pi_{\bar{c},a} * T_d
\end{aligned} \tag{8}
$$

Through calculation, we can draw the result that $E(T^{\oplus}) \geq E(T)$ if

$$
\begin{cases} D_F \geq 0 \\ \textbf{or} \ \ D_F < 0 \ and \ G < \min(F_G) \end{cases} .
$$

More details are in Appendix A. ∎

3.c. The Warm Strategy with The Best Performance. By computing the expected extra time introduced by different warm strategies, we can get the following theorem.

Theorem 6. *the conditional* WARM *strategy that performing* WARM *when some cache misses occur during the previous execution of encryption results in the least extra time of* WARM *and* DELAY*, if the following condition is satisfied:*

$$
\begin{cases} D_F \geq 0 \\ \textbf{\textit{or}} \ \ D_F < 0 \ and \ G < \min(F_G) \end{cases} .
$$

Proof: Firstly, we consider an extreme case that is performing WARM with a probability $P_w^{\odot} \in [0, 1]$ just when cache misses do not occur during the pervious execution of encryption while not performing WARM when cache misses occur. This case is denoted as complementary WARM. It can be proved that conditional WARM strategy has better performance than complementary WARM strategy using the same method as in Theorem 4.

Generally, any warm strategies can be divided into the combination of less WARM, more WARM and complementary WARM. We denote the arbitrary warm strategy as W. In W, performing WARM with a probability $P_1 \in [0, 1]$ when a cache miss occurs during the pervious execution of encryption, while performing WARM with a probability $P_2 \in [0, 1]$ in other conditions. Similarly, we denote less WARM strategy, more WARM strategy and complementary WARM as W_L, W_M and W_{Co}. Then we can get the equation:

$$
W = x_1 W_L + x_2 W_M + x_3 W_{Co} \tag{9}
$$

where x_1, x_2 and x_3 satisfy:

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

In this way, the arbitrary warm strategy is divided. It has already been proved that conditional WARM has the best performance compared with less WARM strategy and complementary WARM. Meanwhile if satisfied the condition in Theorem 5 conditional WARM has better performance compared with more WARM strategy. So conditional WARM has better performance than warm strategies with any P_1 and P_2, if condition:

$$
\begin{cases} D_F \geq 0 \\ \textbf{or} \ \ D_F < 0 \ and \ G < \min(F_G) \end{cases}
$$

is satisfied. That is performing WARM when some cache misses occur during the previous execution of encryption results in the least extra time of WARM and DELAY. ∎

Corollary 2. *For the AES-128 implementation with a 2KB lookup table in commodity computer systems, performing* WARM *when cache-miss occurs during the previous execution, results in the least extra time of* DELAY *and* WARM.

Proof: For the AES-128 implementation with a 2KB lookup table, $P_{a|S_c}$ is larger than 0.994 from Equation 1. The range of $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$ is $[0.994, 1]$. For the Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM, T_d is 2479.00 cycles; the minimum of $T_w(T_w^{min})$ is 124 cycles when accessing the lookup table from L1D caches; and the maximum of T_w is 1781.60 cycles when all the lookup entries are not cached. Thus, $M = 0.0491$ and the rang of G is $[1.39, 19.99]$.

In this environment, the range of D_F is calculated as $[-0.000036, 0.0491]$. When $D_F > 0$, Theorem 6 is satisfied. While $D_F < 0$, F_G is a monotonous increasing function of $P_{a|S_c}$. F_G gets the minimum value 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:

$$
\begin{cases}
P_{e|S_c} = \dfrac{0.994M(1 - P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) + \sqrt{\Delta}}{0.006(1 - 0.006M)} \\
P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = \dfrac{0.07746P_{e|S_c}}{\sqrt{M(1 - P_{e|S_c})}} - 0.006P_{e|S_c} \\
if \ P_{e|S_c} > 1, P_{e|S_c} = 1 \\
if \ P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} > 1, P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1
\end{cases} , \tag{10}
$$

where

$$
\begin{aligned}
\Delta =& 0.988036M^2 - 1.976072M^2 P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \\
&+ (0.006M + 0.988M^2)(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^2 \ .
\end{aligned}
$$

By computation we get 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 gets the minimum value. The minimum value is 167.7. Therefore, G is smaller than the minimum value of F_G when $D_F < 0$. So based on Theorem 6 performing WARM when cache-miss occurs during the previous execution, results in the least extra time of DELAY and WARM. ∎

4. Remarks. In the analysis of WARM operation, we assume that some conditions are not considered. In this part, we discuss these conditions. First is effect of partial warm caused by the AES execution. Second is the influence of process scheduling and interruptions. Finally, we summarize the relation between the measured time and the cache states.

4.a. One AES Execution Is a Partial Warm Operation. When the system is in the state $S_{\bar{c},a}$, one AES execution is a partial warm operation, as in this case, cache misses are resulted to load the corresponding entries into the cache.

When the system state is $S_{\bar{c},a}$, conditional WARM strategy performs WARM with probability 1, while the general warm strategy performs WARM with a probability $P_1^{'} \in [0,1]$ greater than P_1 (in Theorem 6), due to partial warm by the AES execution. However, the condition that makes $E(T^{\oplus}) > E(T)$ hold is not changed. Moreover based on Theorem 6, although $P_1^{'}$ is different, the general warm strategy still can be divided. So Therefore, conditional WARM strategy still has better performance.

4.b. Process Scheduling and Interruption Occur during The AES Execution. In this case, $t2-t1 > T_W$, WARM will be triggered in our scheme. As the process scheduling and interruption occur, we assume that N cache lines of lookup tables are evicted from caches. Hence, the extra overhead introduced by using WARM to load all lookup tables is $E(T_w, N) = NT_{cl} + (32 - N) * T_c$, where T_c and T_{cl} denote the time for accessing one cache line from caches and RAM, respectively. If we do not perform WARM, the next round of AES may need to access part(s) of lookup tables from RAM instead of caches, which will trigger DELAY. The expected overhead is $D_{nl}(N)$ (Equation 2). Table III shows that performing WARM achieves better performance when $N > 0$. If $N = 0$, it means that the lookup tables are not evicted from the cache when the process scheduling and interruption occur. In this case WARM is not need. However from the overall execution we cannot distinguish whether the lookup tables are evicted. Thus performing WARM introduces some extra overhead.

4.c. Periodical Schedule Function Is Called without Scheduling during AES Execution. As no other process invoked, the lookup table is not evicted. Therefore, WARM is unnecessary although $T_{NM} < t2 - t1 \leq T_W$. However, WARM is a better choice, as we cannot predict whether the lookup table is in the cache, and the overhead introduced by accessing the elements in caches is concealed by DELAY.

4.d. The Relationship between The Cache State and Execution Time. In the above, we prove that performing WARM when not all lookup entries are in caches, results the smallest extra time. However, in our scheme, we perform the WARM operation according to the previous AES execution time, as we are technically unable to observe the cache state without introducing additional overhead. The relation between the cache state and execution time is as follows:

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

TABLE IV: the minimum value of the P_a in different cases.

table size	AES-128	AES-192	AES-256
5KB	0.85	0.88	0.90
4.25KB	0.907	0.944	0.966
4KB	0.924	0.954	0.973
2KB	0.994	0.998	0.999

D. Different Key Lengths and Implementations

All the theorems above still stand, with the only difference being the value of P_a, T_w and T_d. The WARM+DELAY scheme provides the optimal performance for various implementations of AES [66], [67] with different key length, if the conditions in previous theorems are satisfied. P_a (detailed in Appendix B) is the probability that some cache line size of lookup tables not in the cache are accessed. It depends on the structure and size of lookup tables and the number of iterated rounds. For mbed TLS-1.3.10 [67] and OpenSSL-0.9.7i [66], they have 5 lookup tables with the total size 4.25KB and 5KB. For OpenSSL-1.0.2c [66], the size of lookup tables is 4KB or 2KB. The number of rounds is 10, 12 and 14 for AES-128, AES-192 and AES-256,respectively. Table IV lists the corresponding minimum P_a. T_w and T_d are related to not only the different implementations and key length, but also the platform that running AES. Therefore, for different implementations and key length of AES, first we should determine P_a, T_w and T_d. Then we can use the previous theorems to determine whether the WARM+DELAY scheme provides the optimal performance. Furthermore, the WARM and DELAY can be configured to satisfy the theorems then get the optimal performance.

For other table-lookup block ciphers, we have to determine P_a, T_w and T_d according to the algorithm, the implementation and the running platform. Once the conditions in these theorems are satisfied , WARM+DELAY provides the optimal performance.

V. IMPLEMENTATION AND PERFORMANCE EVALUATION

We evaluate the scheme on a Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM. This CPU has 4 cores and each core has 32KB L1 data caches, and the operating system is 32-bit Linux with kernel version 3.6.2.

A. Implementation

We apply the WARM+DELAY scheme to AES-128 implemented with a 2K lookup table, which is directly borrowed from OpenSSL-1.0.2c [66]. As we attempt to optimize performance while eliminate remote cache timing side channels, efficient WARM, GETTIME, and DELAY are finished; and the constant parameters (T_{NM} and T_W) are determined properly. Next, we show the implementation details about the WARM+DELAY scheme.

TABLE III: The introduced time by performing warmup operation or not (in cycle).

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
$E(T_{warm})$	175.80	227.60	279.4	383.00	642.00	901.00	1160.00	1419.00	1678.00	1729.80	1781.60

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 lookup tables are in the L1D cache, the execution time of encryption still has variations and these variations could be exploited to launched side-channel attackers [4], [65].

The time variations result from accessing the L1D cache. There are two possible reasons. One is the cache bank conflict. Cache bank is a physical structure that divides the L1D cache 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 accesses to the same cache bank are slower than to different banks. Because the L1D cache is unshared, the L1D cache can not be accessed by the other cores on the same CPU, we disable the Hyper-Threading of the system to ensure the protected process itself not to produce concurrent access to the L1D cache for the cache bank conflict. All the following experiments, Hyper-Threading is disabled.

The other is read-write conflict that the load from L1 cache 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 this conflict occurs. We avoid the read-write conflict by exploiting the stack switch technique [68]. 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 can easily control the address. The beginning address of the array is made next to the lookup table module 4096, and this make the intermediate variables of AES execution use the remaining cache sets compared with the lookup table.

Finally, the distribution of the 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. In the implementation, we need to perform the following operations to achieve the high accuracy: (1) as the TSCs 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 of the computer, we disable this option in BIOS to ensure the clock cycle be a constant.

Besides, the cost of `RDTSCP` is 36 cycles which is much greater than the comparison operation. so, we perform GETTIME only if $t2-t1 < T_W$, instead of every time after WARM (see Line 7 in Algorithm 1).

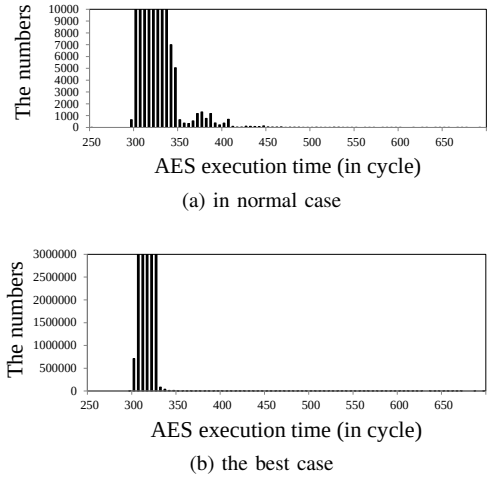


Fig. 2: The distribution of AES encryption time with 2KB lookup table in full warm condition.

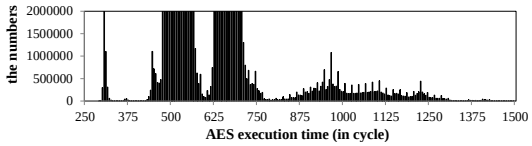


Fig. 3: The distribution of AES execution time only not warm one cache line.

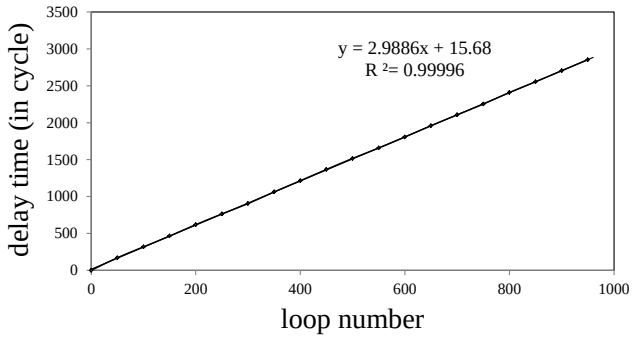


Fig. 4: AES execution performance in different scenarios.

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 the DELAY operation by executing XOR 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 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 (T_d) and the loop

number of xor instruction (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 precision is 3 cycles, much smaller than the noise of remote environments, so it cannot be exploited. Implementation of DELAY() is provided in Listing 1. We do not use a table of t_{delay} and n with the purpose of reducing the use of caches in our scheme.

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");
}
```

T_{NM}. T_{NM} is larger than the minimum AES execution time (no cache miss occurs), which avoids the unnecessary WARM() and DELAY() operations; and less than the AES execution that only one cache miss occurs. The average minimum AES execution time is measured by average 2^{30} AES execution time with the lookup tables all in L1D cache. In our environment it is 331 cycles. Fig. 3 shows the distribution of AES execution time for 2^{30} plaintexts while all lookup tables except one block of 64 bytes (i.e., one cache line) are loaded in L1D cache. Note that, this uncached entry may be unnecessary in an execution of AES encryption. We use the stack switch technique to eliminate the read-write conflict, at the same time it makes the AES execution time much more concentrated as shown in Fig. 2. This helps the determination of T_{NM} can be more accurate and reasonable. Also we should choose the value as large as possible to avoid the influence of the fluctuation around T_{NM} that might occur. Finally, we choose 355 cycles as T_{NM}. This value is chosen to ensure that all tables are in L1 caches and no fluctuation occurs around it. So no useful observation will be obtained by attackers, and no useful variations will be magnified.

T_W. T_W is the worst AES execution time and unrelated to the cache states. Before measuring, we flush both the data and instructions out of L1/2/3 caches. In actual measurement, we repeat the measurement for 10000 times, and calculate the average value of 100 greater measurements as T_W. In this case T_W is 2834 CPU cycles.

B. Performance Evaluation

In this section, we first demonstrate the result that with our scheme the distribution of AES execution time are separated into two parts: less than T_{NM} and no less than T_W, which meets our expectation. Then, we evaluate the performance of WARM+DELAY scheme in three different aspects. Firstly we compare our scheme with different WARM strategies to show that our scheme has the best performance among the different strategies using warm and delay operations. Secondly, we measure the performance of several different defense methods comparing with our scheme. It will show that our scheme has a better performance than other software-based methods.

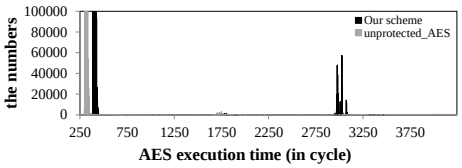


Fig. 5: The observed AES execution time with different plaintext.

Finally, we apply our defense scheme to Openssl and use an Apache web server as a HTTPS server to provide application services. We find that the overhead of our scheme is acceptable in a real environment.

The AES distribution with WARM+DELAY scheme. To show the security of WARM+DELAY scheme, we measure the distribution of AES execution time implemented with the WARM+DELAY scheme. We compare the distribution of the protected AES with the unprotected ones using 2^{30} different plaintexts.

Figure 5 shows the distributions of AES execution time for two cases. It is clearly that most of the execution time is less than T_{NM} or no less than T_W using the WARM+DELAY scheme. The time between T_{NM} and T_W in unprotected AES distribution is delayed to T_W. Also from this figure, the average execution time of our scheme is less than 1.29 times of the unprotected AES.

Performance of different warm strategies. We compare conditional WARM with WARM strategies which have different P_1 and P_2. The runtime environment is with low workload, high computing workload and high memory workload separately when the interval of OS scheduler is 1 ms and 4 ms respectively. In each case, we perform 2^{30} AES encryptions with random plaintexts. The AES encryption process and the concurrent workload run on the same CPU core. We use the benchmark SysBench to simulate computing and memory workload. For computing workload, we run SysBench in its CPU mode, which launches 16 threads to issue 10K requests to search the prime up to 300K. For memory workload, we adopt SysBench with 16 threads in its memory mode, which reads or writes 1KB block each time to operate the total 3GB data on one CPU core.

Figure 6 validates that the performance of conditional WARM strategy is the best. Moreover, we calculated P_{evict} under different workloads, according to the number of AES encryption whose execution time is greater than T_{NM}. When P_{evict} is the values in Table V, as proved in Section IV, the WARM+DELAY scheme is the optimal.

Comparison of different defense methods. Furthermore, we compare the performance of our scheme with different defense

TABLE V: The value of P_{evict} under different workload.

Interval of OS scheduler	Low workload	High CPU workload	High mem workload
1ms	1.87×10^{-4}	1.99×10^{-4}	2.20×10^{-3}
4ms	4.67×10^{-5}	7.52×10^{-5}	7.88×10^{-5}

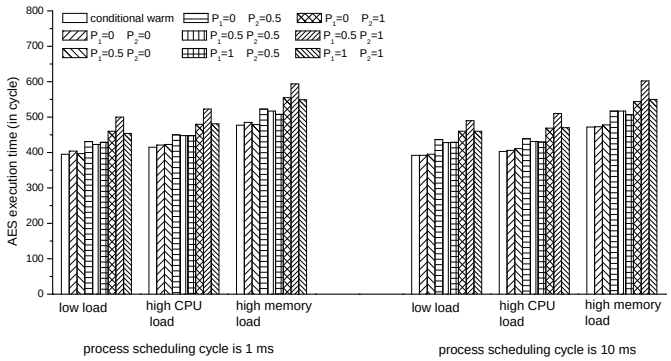


Fig. 6: AES execution performance in different scenarios.

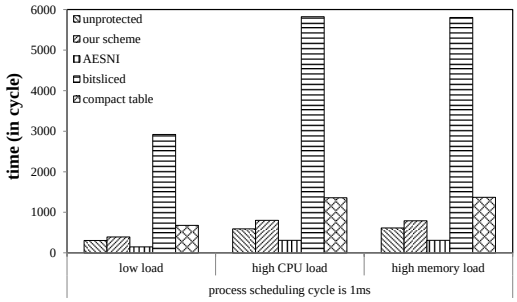


Fig. 7: Performance of different defense methods.

methods: AESNI, compact table implementation [66] and bitsliced AES implementation [69]. For each method, we perform 2^{30} AES encryptions with random plaintexts. It is shown in Figure 7 that AESNI, the hardware implementation, has the best performance. The WARM+DELAY scheme has the best performance among all the software implementations.

Performance in HTTPS. We applied our solution to protect the TLS connection protocol in OpenSSL. we use the Apache web server as the HTTPS server to provide application services. Apache serves several web pages of different sizes under HTTPS with TLSv1.2. The TLS cipher suit is ECDHE-RSA-AES128-SHA256. The client runs on another computer in 1Gbps LAN with the server. ApacheBench issues 10K requests with various levels of request size, and we measure the HTTPS server throughput.

The HTTPS throughput is shown in Figure 8. When the unprotected AES is used, the throughput is 27.2 requests per second for 1MB data. While using our scheme, the throughput is 23.5 requests per second for 1MB data. It is clearly that WARM+DELAY scheme has a low influence on the performance of TLS protocol. As the data of request increase, the influence on the performance of TLS protocol caused by WARM+DELAY scheme increases gradually. Furthermore, we compare the throughput using different defense methods involved in TLS protocol. Our defense scheme has the largest throughput among these methods.

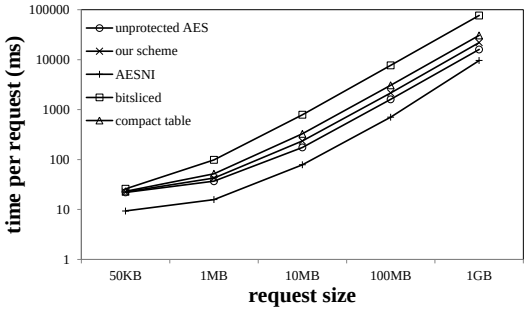


Fig. 8: Apache Benchmark result

VI. DISCUSSIONS

A. Instruction Cache

Firstly, the implementation of block ciphers is not subject to timing side-channel attacks based on instruction caches [70], because there is generally no branch in the execution path. 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. The status of instruction caches affect the execution time, but is not related to the data (i.e., keys and plaintexts/ciphertexts).

As the effect of instruction caches on the execution time is almost indistinguishable from that of data caches, we determine T_{NM} when all instructions of encryption/decryption are cached; and the value will be greater if it is done when the instructions are uncached. However, when the encryption/decryption functions are invoked and all the instructions are cached, such a greater T_{NM} will offer attack opportunities because the measured time may leak some information about data cache access. Similarly, we determine T_W as the execution time when all instructions are not in instruction caches (and all lookup tables are not in data caches). Otherwise, the measured time might also leak some information about data cache access.

B. Timing Variations with Fully Cached Lookup Tables

In the evaluation, we use the AES implementation with a 2KB lookup table. Based on the cache structure, the L1D cache of 32KB is divided into 64 cache sets and each set has 8 cache lines. The 2KB lookup table takes up one cache line of 32 cache sets. So we can declare a 2KB array taking up the reminder cache sets to be used as the stack, just making the address successive with the lookup table module 4096.

However, when using larger lookup tables, all cache sets will be occupied by default due to the continuity of lookup tables in the RAM. In these conditions, to avoid the impact of read-write of same cache sets and to be able to declare a 2KB array as the stack, first we should make the lookup tables only take up the 32 cache sets. This can be done by making the lookup tables discontinuous. When using 4KB lookup tables, we can make the first address of T_2 the same as T_0 module 4096, which makes them take up the same cache sets. So 4KB lookup tables only occupy 32 cache sets and the 2KB array can take up the other 32 cache sets. The same is true for 4.25KB

and 5KB lookup tables. We can make them only take up 32 cache sets and leave the others for the 2KB array.

C. Other Cache-based Timing Side Channels

Our WARM+DELAY scheme can be used in active cache side channel attacks. the synchronous attacks can be prevented by the WARM+DELAY scheme. These attacks monitor the cache once per encryption, so the attackers can not observe the inner cache states. With the WARM+DELAY scheme the cache access pattern is masked to the attackers. So the WARM+DELAY scheme is effective for the synchronous attacks. while the asynchronous attacks can not be defended because the attackers can change the cache states and observe the inner cache access pattern which the WARM+DELAY scheme is helpless.

Additionally, there is another type of cache attack called trace-driven attack. During the execution of encryption/decryption, the trace-driven attackers need direct physical contact to the victim machine to monitor the variations of electromagnetic fields or power to capture the profile of cache activities and deduce cache hits and misses [2], [27], [71], For trace-driven attacks, attackers can observe the inner cache access pattern of the encryption/decryption execution, so DELAY is not effective. While WARM is effective but not always executed. Therefore, the WARM+DELAY scheme can not eliminate but just mitigate the trace-driven attacks.

VII. CONCLUSION

We attempt to eliminate cache-based timing side channels with the optimal performance. The proposed WARM+DELAY scheme is the first one to prevent cache timing side-channel attacks, while achieves the optimal performance with the least extra operations. The scheme eliminates remote cache timing side channels, by integrating WARM and DELAY operations, two algorithm-independent and implementation-transparent mechanisms. We investigate the factors affecting the performance of the scheme. It is concluded that WARM should load all lookup tables into caches and DELAY should make the execution time to T_W. The optimal performance is achieved by performing WARM when cache-miss occurs during the previous execution and performing DELAY when the execution time of encryption is in (T_{NM}, T_W).

We implement the derived WARM+DELAY scheme on Linux with Intel Core CPUs for AES-128. Experimental results confirm that, (*a*) the execution time does not leak information about cache access, (*b*) the scheme can be configured to achieve the optimal performance, (*c*) the scheme outperforms other different integration strategies of WARM and DELAY, and (*d*) the implementation works without any privileged operations on the system.

ACKNOWLEDGMENT

Ziqiang Ma, Quanwei Cai, Jingqiang Lin, and Jiwu Jing were partially supported by National Natural Science Foundation of China (No. 61772518) and National Basic Research Program of China (973 Program No. 2013CB338001). Bo Luo was partially supported by ...

APPENDIX A
THE DETAIL PROOF OF THEOREM 4 AND THEOREM 5

In this section, we describe the proof details of Theorem 4 and Theorem 5.

For the following calculation, $0 \leq P_{a|S_c} \leq 1$, $0 \leq P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \leq 1$, $0 \leq P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \leq 1$, $0 \leq P^{\oplus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \leq 1$, $0 \leq P^{\ominus}_{w} \leq 1$, $0 \leq P^{\oplus}_{w} \leq 1$ and $0 \leq P_{e|S_c} \leq 1$.

A. The proof of Theorem 4

To prove the Theorem 4, we need to compare $E(T^{\ominus})$ with $E(T)$. So we get the Formula 7, and estimate if it is larger than 0. By combing Equation 4 and 5, we get Equation 11 and 12.

$$\begin{aligned} y(P^{\ominus}_{w}) = &- P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c}(P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{a|S_c} - P_a^{3^{\mathcal{L}}})) * P^{\ominus}_{w} \\ &+ P_{e|S_c} P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1 - P_{a|S_c}) + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \\ &- P_{e|S_c} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}(1 + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_a^{3^{\mathcal{L}}}) \end{aligned} \tag{11}$$

$$E(T^{\ominus}) - E(T) = \frac{y(P^{\ominus}_{w}) P_{e|S_c} T_d}{A_L * B_L} \tag{12}$$

where

$$\begin{aligned} A_L = &P_{e|S_c}(1 + P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}}) + (1 - P_{e|S_c}) P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} P^{\ominus}_{w} \\ &+ (P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}} - P_{a|S_c}) P_{e|S_c} P^{\ominus}_{w} \\ B_L = &P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + P_{e|S_c}(1 - P_{a|S_c}) \end{aligned}$$

In the Markov state transition diagram,

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

It can be seen from analysis that $P^{\ominus}_{a|S_{\bar{c},a}} > P_{a|S_c}$. So we can get $P^{\ominus}_{S_{\bar{c},a}|S_{\bar{c},a},\bar{w}} > P_{a|S_c}$. In this way, the function $y(P^{\ominus}_{w})$ is a monotony decrease function. When $P^{\ominus}_{w} = 1$, it gets the minimum 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 $y(P^{\ominus}_{w}) = 0$. Also A_L and B_L are greater than zero. Therefore $E(T^{\ominus}) - E(T) \geq 0$. That means the extra time introduced by less WARM strategy is always larger than conditional WARM strategy.

B. The proof of Theorem 5

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

$$\begin{aligned} E(T^{\oplus}) - E(T) = &\Pi^{\oplus}_{\bar{c},a} * T_d + \Pi^{\oplus}_{c} * P_w * T_w^{min} \\ &+ \Pi^{\oplus}_{\bar{c},\bar{a}} * P_w * T_w - \Pi_{\bar{c},a} * T_d \end{aligned} \tag{13}$$

We estimate if it is larger than 0. First, we find that $P^{\oplus}_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}$. Then by combing Equation 4 and 6, we get Equation 14 and 15.

$$y(P_w^{\oplus}) = B_M C_M P_w^{\oplus} + A_M C_M + G D_M \tag{14}$$

$$E(T^{\oplus}) - E(T) = \frac{y(P_w^{\oplus}) P_w^{\oplus} T_w}{E_M * F_M} \tag{15}$$

where

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

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

$$(C_M(1-P_{e|S_c})P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} + D_M)G + C_M P_{e|S_c}(1-P_{a|S_c}) \geq 0 \ . \tag{16}$$

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,

$$\begin{aligned}
D_F &= -(P_{e|S_c})^2 (P_{a|S_c})^2 \\
&\quad + (P_{e|S_c})^2 P_{a|S_c} + (1 - M(1-P_{e|S_c})) P_{e|S_c} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} P_{a|S_c} \\
&\quad + M(1-P_{e|S_c})(P_{e|S_c} + P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} - P_{e|S_c} P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}
\end{aligned} \tag{17}$$

Regard D_F as the function of $P_{a|S_c}$, and we can find that it is an quadratic function.

When $P_{a|S_c} = 1$,

$$D_F(P_{a|S_c}) = M(1-P_{e|S_c})(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})^2 \geq 0 \ .$$

Let $D_F(P_{a|S_c}) = 0$, and denote the smaller root of the equation as $P_{a|S_c,lroot}$. so when $P_{a|S_c} \geq P_{a|S_c,lroot}$, $D_F \geq 0$. When $P_{a|S_c} < P_{a|S_c,lroot}$, $D_F < 0$. Thus,

1) When $P_{a|S_c} \geq P_{a|S_c,lroot}$, $D_F \geq 0$. In this case, inequality 16 holds. That is $E(T^{\oplus}) - E(T) \geq 0$.
2) When $P_{a|S_c} < P_{a|S_c,lroot}$, $D_F < 0$. In this case, to make the inequality 16 holds,

$$G \leq \frac{C_M P_{e|S_c}(P_{a|S_c} - 1)}{D_F} \tag{18}$$

for all $P_{e|S_c} \in [0,1]$ and $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \in [0,1]$.

For the second case, denote

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

1) First, we regard F_G as the function of $P_{a|S_c}$ and calculate the derivative on $P_{a|S_c}$, and get:

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

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

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

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

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

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

So, when $0 \leq P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} \leq P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot}$,

$$\frac{d(F_G)}{d(P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}})} \leq 0 \ .$$

Therefore, when $P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}},rroot} \geq 1$, F_G is a monotony decrease 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) Third, we regard F_G as the function of $P_{e|S_c}$ and calculate the derivative on $P_{e|S_c}$, and get:

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

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

$$P_{e|S_c,rroot} = \frac{MP_{a|S_c}(1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) + \sqrt{\Delta}}{(1-P_{a|S_c})(1-M+MP_{a|S_c})} \tag{19}$$

where

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

So when $0 \leq P_{e|S_c} \leq P_{e|S_c,rroot}$,

$$\frac{d(F_G)}{d(P_{e|S_c})} \leq 0 \ .$$

Therefore, when $P_{e|S_c,rroot} \geq 1$, F_G is a monotony decrease 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 minimum value of F_G gets when

$$\begin{cases}
P_{e|S_c} = \dfrac{MP_{a|S_c}(1-P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}}) + \sqrt{\Delta}}{(1-P_{a|S_c})(1-M+MP_{a|S_c})} \\
P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = P_{e|S_c}\sqrt{\dfrac{1-P_{a|S_c}}{M(1-P_{e|S_c})}} - P_{e|S_c}(1-P_{a|S_c}) \\
if \ P_{e|S_c} > 1, P_{e|S_c} = 1 \\
if \ P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} > 1, P_{S_{\bar{c},a}|S_{\bar{c},\bar{a}}} = 1
\end{cases} \tag{20}$$

From the above, we can make the following conclusions:

1) When $D_F \geq 0$, $E(T^{\oplus}) - E(T) \geq 0$.
2) When $D_F < 0$, $E(T^{\oplus}) - E(T) \geq 0$ if G is smaller than the minimum value of F_G.

That is: $\forall D_F$, when $D_F < 0$, G is smaller than the minimum value of F_G. At this point conditional WARM strategy has better performance than the less WARM strategy.

APPENDIX B
THE P_a IN DIFFERENT AES KEY LENGTHES AND DIFFERENT AES IMPLEMENTATIONS

We assume the size of a cache line is C Byte. P_a represents that the probability that some of the N cache line size of lookup tables not in the cache are accessed. So when the AES implementation uses 4.25KB lookup tables, the P_a for AES-128, AES-192 and AES-256 are as follows respectively:

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

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

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

where

$$\begin{aligned}
f(x) &= \binom{256/C}{x} (1 - \frac{xC}{256})^{16} \\
g(x,y) &= \binom{1024/C}{x} (1 - \frac{xC}{1024})^{y} \\
x_0 + x_1 + x_2 + x_3 + x_4 &= N \\
x_{4b} &= \max(0, N - 4096/C) \\
x_{4e} &= \min(256/C, N) \\
x_{0b} &= \max(0, N - x_4 - 3072/C) \\
x_{0e} &= \min(1024/C, N - x_4) \\
x_{1b} &= \max(0, N - x_4 - x_0 - 2048/C) \\
x_{1e} &= \min(1024/C, N - x_4 - x_0) \\
x_{2b} &= \max(0, N - x_4 - x_0 - x_1 - 1024/C) \\
x_{2e} &= \min(1024/C, N - x_4 - x_0 - x_1)
\end{aligned}$$

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

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

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

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

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, the P_a for AES-128, AES-192 and AES-256 are as follows respectively:

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

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

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

where

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

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

$$P_a^{128} = 1 - (1 - \frac{NC}{2048})^{16*10},$$

$$P_a^{192} = 1 - (1 - \frac{NC}{2048})^{16*12},$$

$$P_a^{256} = 1 - (1 - \frac{NC}{2048})^{16*14}.$$

REFERENCES

[1] O. Acıiçmez, W. Schindler, and Ç. K. Koç, "Improving brumley and boneh timing attack on unprotected ssl implementations," in *CCS*. ACM, 2005, pp. 139–146.

[2] O. Acıiçmez and Ç. K. Koç, "Trace-driven cache attacks on AES (short paper)," in *ICICS*. Springer, 2006, pp. 112–121.

[3] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *CHES*. Springer, 2006, pp. 201–215.

[4] D. J. Bernstein, "Cache-timing attacks on AES," http://cr.yp.to/antiforgery/cachetiming-20050414.pdf, April 2005.

[5] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on AES to practice," in *S&P*. IEEE, 2011, pp. 490–505.

[6] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *CRYPTO*. Springer, 1996, pp. 104–113.

[7] M. Neve, J. P. Seifert, and Z. Wang, "A refined look at Bernstein's AES side-channel analysis," in *ASIACCS*. ACM, 2006, p. 369.

[8] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *CT-RSA*. Springer, 2006, pp. 1–20.

[9] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of DES implemented on computers with cache." in *CHES*. Berlin, Heidelberg: Springer, 2003, pp. 62–76.

[10] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on AES, and countermeasures." *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, 2010.

[11] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from PCs by radio: Cheap electromagnetic attacks on windowed exponentiation," in *CHES*. Springer, 2015, pp. 207–228.

[12] Y. Oren and A. Shamir, "How not to protect PCs from power analysis," *Rump Session, Crypto*, 2006.

[13] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo, "AES power attack based on induced cache miss and countermeasure," in *ITCC*. IEEE, 2005, pp. 586–591.

[14] D. Genkin, I. Pipman, and E. Tromer, "Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs," in *CHES*. Springer, 2014, pp. 242–260.

[15] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *CRYPTO*. Springer, 2014, pp. 444–461.

[16] Y. Tsunoo, "Cryptanalysis of block ciphers implemented on computers with cache," *preproceedings of ISITA 2002*, 2002.

[17] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *ESORICS*, 2011, pp. 355–371.

[18] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[19] O. Acıiçmez, W. Schindler, and Ç. K. Koç, "Cache based remote timing attack on the AES," in *CT-RSA*. Springer, 2007, pp. 271–286.

[20] A. C. Atici, C. Yilmaz, and E. Savas, "Remote cache-timing attack without learning phase." *IACR Cryptology ePrint Archive*, vol. 2016, p. 2, 2016.

[21] V. Saraswat, D. Feldman, D. F. Kune, and S. Das, "Remote cache-timing attacks against aes," in *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*. ACM, 2014, pp. 45–48.

[22] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in *CCS*. ACM, 2012, pp. 305–316.

[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 *CCS*. ACM, 2009, pp. 199–212.

[24] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A high resolution, low noise, l3 cache side-channel attack." in *USENIX Security*, 2014, pp. 719–732.

[25] B. A. Braun, S. Jana, and D. Boneh, "Robust and efficient elimination of cache and timing side channels," *arXiv preprint arXiv:1506.00189*, 2015.

[26] D. Zhang, A. Askarov, and A. C. Myers, "Predictive mitigation of timing channels in interactive systems," in *CCS*. ACM, 2011, pp. 563–574.

[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, pp. 191–204.

[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," Red Hat, Inc, Tech. Rep., 2007.

[32] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *S&P*. IEEE, 2015, pp. 605–622.

[33] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in *DIMVA*. Springer, 2016, pp. 279–299.

[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. Cock, Q. Ge, T. Murray, and G. Heiser, "The last mile: An empirical study of some timing channels on sel4," in *CCS*. ACM, 2014, pp. 570–581.

[36] D. Page, "Partitioned cache architecture as a side-channel defence mechanism." *IACR Cryptology ePrint archive*, vol. 2005, no. 2005, p. 280, 2005.

[37] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *CCS*. ACM, 2013, pp. 827–838.

[38] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *CHES*. Springer, 2009, pp. 1–17.

[39] R. Könighofer, "A fast and cache-timing resistant implementation of the AES," in *CT-RSA*, 2008, pp. 187–202.

[40] Y. H. Taha, S. M. Abdulh, N. A. Sadalla, and H. Elshoush, "Cache-timing attack against AES crypto system - countermeasures review," 2014.

[41] D. Jayasinghe, J. Fernando, R. Herath, and R. Ragel, "Remote cache timing attack on advanced encryption standard and countermeasures," in *ICIAFS*. IEEE, 2010, pp. 177–182.

[42] D. Page, "Defending against cache-based side-channel attacks," *Information Security Technical Report*, vol. 8, no. 1, pp. 30–44, 2003.

[43] 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, pp. 189–204.

[44] 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*. IEEE, 2009, pp. 393–404.

[45] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ISCA*. ACM, 2007, pp. 494–505.

[46] ——, "A novel cache architecture with enhanced performance and security," in *MICRO*. IEEE, 2008, pp. 83–93.

[47] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "Secdcp: secure dynamic cache partitioning for efficient timing channel protection," in *DAC*. IEEE, 2016, pp. 1–6.

[48] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *CCS*. ACM, 2010, pp. 297–307.

[49] C. Ferdinand, "Worst case execution time prediction by static program analysis," in *IPDPS*. IEEE, 2004, p. 125.

[50] J. V. Cleemput, B. Coppens, and B. De Sutter, "Compiler mitigations for time attacks on modern x86 processors," *TACO*, vol. 8, no. 4, p. 23, 2012.

[51] B. Coppens, I. Verbauwhede, K. D. Bosschere, and B. D. Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *S&P*. IEEE, 2009, pp. 45–60.

[52] 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*. Springer, 2013, pp. 718–735.

[53] V. Varadarajan, T. Ristenpart, and M. M. Swift, "Scheduler-based defenses against cross-vm side-channels." in *USENIX Security*, 2014, pp. 687–702.

[54] U. Herath, J. Alawatugoda, and R. Ragel, "Software implementation level countermeasures against the cache timing attack on advanced encryption standard," in *ICIIS*. IEEE, 2013, pp. 75–80.

[55] D. Jayasinghe, R. Ragel, and D. Elkaduwe, "Constant time encryption as a countermeasure against remote cache timing attacks," in *ICIAFS*. IEEE, 2012, pp. 129–134.

[56] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz, "Thwarting cache side-channel attacks through dynamic software diversity," in *NDSS*, 2015.

[57] J. Blömer, J. Guajardo, and V. Krummel, "Provably secure masking of AES," in *SAC*. Springer, 2004, pp. 69–83.

[58] J. Blömer and V. Krummel, "Analysis of countermeasures against access driven cache attacks on AES," in *SAC*. Springer, 2007, pp. 96–109.
[59] B. Kopf and M. Durmuth, "A provably secure and efficient countermeasure against timing attacks," in *CSF*. IEEE, 2009, pp. 324–335.
[60] P. Li, D. Gao, and M. K. Reiter, "Mitigating access-driven timing channels in clouds using stopwatch," in *DSN*. IEEE, 2013, pp. 1–12.
[61] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *ISCA*. IEEE, 2012, pp. 118–129.
[62] 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*, vol. 2006, p. 52, 2006.
[63] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2008, pp. 137–146.
[64] I. Liu and D. McGrogan, "Elimination of side channel attacks on a precision timed architecture," *Technical Report*, 2009.
[65] A. Canteaut, C. Lauradoux, and A. Seznec, "Understanding cache attacks," Ph.D. dissertation, INRIA, 2006.
[66] "OpenSSL:Cryptography and SSL/TLS Toolkit," https://www.openssl.org/.
[67] "SSL Library mbed TLS / PolarSSL," https://tls.mbed.org/.
[68] L. Guan, J. Lin, B. Luo, and J. Jing, "Copker: Computing with private keys without ram." in *NDSS*, 2014, pp. 23–26.
[69] "bitcoin-core/ctaes: Simple constant-time AES implementation," https://github.com/bitcoin-core/ctaes/.
[70] O. Acıiçmez, "Yet another microarchitectural attack::exploiting I-cache," in *Comp. Security Arch*. ACM, 2007, pp. 11–18.
[71] C. Lauradoux, "Collision attacks on processors with cache and countermeasures." in *Western European Workshop on Research in Cryptology, July 5-7, 2005, Leuven, Belgium (WEWoRC2005)*, vol. 5, 2005, pp. 76–85.

John Doe Biography text here.

Jane Doe Biography text here.