Eliminating Remote Cache Timing Side Channels with Optimal Performance

Michael Shell, *Member, IEEE,* John Doe, *Fellow, OSA,* and Jane Doe, *Life Fellow, IEEE*

***Abstract*—Cache timing side channels allow a remote attacker to recover the cryptographic keys, by repeatedly invoking the encryption/decryption functions and measuring the execution time. WARM and DELAY, are two common countermeasures against remote cache-based timing side channels, by reading constant data into caches before cryptographic operations and inserting padding instructions after these operations, respectively. These schemes destroy the relationship between the execution time and the cache misses/hits which are determined by the secret key, but introduces extra operations and performance overheads. In this paper, we integrate these two schemes into a framework that adopts WARM and DELAY complementarily. The proposed WARM+DELAY framework eliminates the remote cache timing side channels, and is applicable to any cryptographic algorithms and computing platforms. More importantly, we apply this framework to AES, and analyze that it achieves *the optimal performance with the least extra operations*. We implement this framework on Linux with Intel Core CPUs to protect AES. Experimental results of the prototype show that, (*a*) the execution time does not reflect cache access, (*b*) the WARM+DELAY framework outperforms different integration strategies of warm and delay, and (*c*) it works without any privileged operations on the system.**

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

I. INTRODUCTION

In the implementations of a cryptographic algorithm, 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 theoretically secure. Side channel attacks are using these vulnerabilities to obtain the sensitive information. Among all kinds of side channels, timing attack is widely used due to easy realization. Specially, remote timing side channels allow an attacker without any system or physical privilege on the computer, to recover the keys by just repeatedly invoking the encryption/decryption functions and measuring the execution time [3], [4], [7], [16]–[18], which is a big threat to the security system.

The remote timing side channels exploiting the time difference of cache misses and hits, called the *remote cache timing side channels* in this paper, are discovered in many cryptographic implementations of block ciphers [3], [4], [9], [16], [19]. In various implementations of block ciphers, table lookup operations are the primary time-consuming operations. Because accessing data in caches is much faster than those in RAM, cache misses and hits in these table lookup operations are reflected in the overall execution time of block ciphers. Different inputs of table look operations cause different cache access behavior so that the overall time differs, which are related to the secret keys, and then the information about keys leaks.

Remote timing cache side channels widely exists in the software implementations of cryptographic algorithms, and are practical to be exploited [4], [19]–[21]. Compared with other side channels on electromagnetic fields, power, ground electric potential and acoustic emanations, remote timing cache-based side-channel attacks do not require special equipments or extra physical access to the system. Moreover, they just need the ability to invoke the necessary encryptions/decryptions without any extra operations or spy processes running on the target system, compared with other cache-based side-channel attacks [5], [22]–[24].

Since little acquirements are needed, this type attack is easy to deploy and hard to defense. There already exists many defense methods to eliminate the remote cache side channels. The key point is to break the relationship between secret keys and the execution time by eliminating the cache misses or confusing the cache misses. Although the existing methods can offer some security guarantee, most of them have some drawbacks. Some methods which are implemented on hardware to control the cache lack universality. Some using particular implementation to make algorithms constant time are just suitable to specific algorithms. Some introducing many extra operations to eliminate or confuse the cache misses incur significant performance overheads. It is difficult to perform a universal and high efficient method to defend against the remote cache timing attacks.

In this paper, we present WARM+DELAY, a universal defense mechanism to eliminate remote cache timing side channels. Meanwhile we achieve the optimized performance both in theoretical analysis and actual implementation. Two elementary operations, *warm* (i.e., to fill cache lines by reading constant tables) and *delay* (i.e., to insert padding instructions after the encryption/decryption operations), have been proposed to prevent cache side-channel attacks [4], [5], [8], [25]. Our scheme combines these two operations as follows to reduce extra overheads, to gain the optimized performance while ensure security.

1) Destroy the relationship between the execution time and cache misses/hits, to ensure security. That is, the measured execution time reflects either the best case

M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

J. Doe and J. Doe are with Anonymous University.

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

(i.e., all lookup tables are cached), or the worst case (i.e., none of tables is in caches).

2) Prefer warm for the best case, to optimize the performance. Warm before executing the encryption/decryption operations, if not all tables are cached. However, cached data may be evicted by system activities and the effect of warm is destroyed; in this case, the time of encryption/decryption may reflect the data cache access, and then delay operation makes the execution to be the worst case.
3) If delay is performed, warm is also done for the next encryption/decryption operation to raise the performance. So, some useless padding instructions are utilized to perform warm instead. Note that warm also consumes CPU cycles.

The proposed WARM+DELAY scheme does not require any privileged operation on the system. The conditions to perform warm and delay, are defined as regular timing. Reading constant tables, padding instructions, and timing are commonly supported in computer systems. The scheme is independent of algorithms and implementations. It does not depend on any special design or feature of block ciphers, and it is applicable to different implementations, except that the lookup tables are accessible in the warm operations.

We apply this scheme to AES, and analyze the situations that it produces the optimized performance. It achieves the optimized performance for the AES-128 implementation with 2KB lookup tables [26], when (a) the probability that some entries of lookup tables are evict from caches by system activities during the executions of encryption/decryption, denoted as P_{evict}, is less than 0.2, and (b) the ratio of the delayed time to the time of reading constant tables from either caches or RAM, denoted as k, is less than 57.4. These conditions hold for continuous AES executions in commodity systems, and confirmed experimentally by the prototype. In fact, our scheme produces the optimized performance for various AES implementations (the lookup tables in size of 2K, 4KB, 4.25KB and 5KB) in different key length (128, 192 and 256 bits), when $P_{evict} < 0.2$ and $k < 42.41$. These conditions also hold in commodity systems.

We implement the WARM+DELAY scheme with AES. Experiment results on Linux with one Intel Core i7-2600 CPU and 2GB RAM, demonstrate that, the measured time does not reflect the data cache access. We also validate the optimization in different scenarios, by comparing it with different strategies of warm and delay. To the best of our knowledge, this work is the first protection against cache timing side channels with the analysis of optimized performance.

In summary, our contributions are listed below:

- We present WARM+DELAY, a defense mechanism efficiently combining warm and delay operations to eliminate remote cache timing side channels.
- We theoretically analyze the defense scheme and find the optimal way to use the warm and delay operation with security.
- The scheme is algorithm-independent and need no special privilege, so it can be widely used in all block ciphers with lookup tables.
- We apply the scheme into AES and verify the security and performance both in theory and experiments.

The remainder of this paper is organized as follows. Section II presents the background and related works. Section III describes the WARM+DELAY scheme and analyzes its security. Section IV proves that our scheme achieves the optimized performance, which is verified by the evaluation in Section V. Section VI contains extended discussions. Section VII draws the conclusion. Appendix A provides the proof of the optimized performance in details, while Appendix B analyzes various implementations of AES with different key lengths.

II. BACKGROUND AND RELATED WORKS

A. AES and Block Cipher

AES is a popular block cipher with 128-bit input 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 [1]. These steps except `AddRoundKey`, can be implemented as table lookup operations [26]. `AddRoundKey` consists of bitwise `XOR` operations on the state. The common implementation of the lookup tables needs four 1KB tables T_0, T_1, T_2 and T_3, which are used in all rounds.

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

Where S(x) stands for the result of an AES S-box lookup for the input value x. In this paper, we use the AES implementation with a 2KB lookup table in RAM. This lookup table is

$$T[x] = (2 \cdot S(x), S(x), S(x), 3 \cdot S(x), 2 \cdot S(x), S(x), S(x), 3 \cdot S(x))$$

The four lookup tables needed by AES computation are formed by this 2KB table.

There are similar implementations of other block ciphers: only table lookup and other basic operations, and no data dependent branch in the execution path. For example, 3DES, Blowfish [27], CAST128 [28] used in OpenSSH-7.4p1 [29].

B. Cache Timing 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. 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 this operation is performed in caches.

Cache timing side-channel attacks exploit the fact that accessing cached data is about two orders of magnitude faster

[1] The last round only performs `SubBytes`, `ShiftRows` and `AddRoundKey`

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 caches, while an operation in RAM takes about 250 cycles [30]. Two typical remote cache-based side channels for block ciphers are outlined in the following.

The internal collision attack. Each cache line contains multiple lookup table entries, and the internal collision is used to denote accessing the entries in the same cache line. The internal collision attack is applied to DES, 3DES and the first round of AES [9], [16], and extended to the second round and last round of AES [19]. This attack is based on the correlation between the encryption time and the average number of cache hits (or misses). For AES, the input plaintext and the first 128 bits of the key are denoted in bytes as $p_0, p_1, ..., p_{15}$ and $k_0, k_1, ..., k_{15}$ respectively, the inputs of the lookup table i $(0 \leq i \leq 3)$ in the first round, are $p_{(i+4j)\%16} \oplus k_{(i+4j)\%16}$ where $0 \leq j \leq 3$. When any two of $p_{(i+4j)\%16} \oplus k_{(i+4j)\%16}$ where $0 \leq j \leq 3$ refer to the entries of the lookup table in the same cache line, a shorter encryption time appears[2]. Then the difference of some key bits can be inferred as the values (i.e., $p_{(i+4j)\%16} \oplus p_{(i+4k)\%16}$, $0 \leq j < k \leq 3$) counted most often, which is used to extract the secret key.

The Bernstein's attack. Bernstein's attack [4] is based on the case that the whole AES execution time is correlated with the time for accessing the lookup tables, which depends on the lookup table index. The attacker collects a large number of encryption times for different plaintexts on the duplicated server whose hardware and software configuration is the same as the victim one, with a known key. For each byte of the key, the attacker obtains the lookup table index ($p_i \oplus k_i$ where $0 \leq i < 16$) corresponding to the maximum AES execution time, from the execution on the duplicate server. Then, the attacker infers the key of the victim server, with the obtained lookup table index and known plaintexts.

C. Countermeasure

Although eliminating timing side channels remains difficult [32], different countermeasures have been proposed on hardware, software and OS levels. All the defense methods can be divided into three categories: eliminating cache misses, adding confusion to cache misses and their combination.

Eliminating cache misses. The most direct method to defend against the cache side channel attacks is to avoid using caches. Page suggests to disable caches in paper [33], 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], [34], [35] 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], [36] and loading the lookup tables into registers [8], [37]. 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], [33], 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 [32]. STEALTHMEM [38] allows each VM to load the sensitive data into its own locked cache lines. A hardware-based mechanism presented in [39], allows caches to be configured dynamically to match the need of a process. The new cache design [40], [41] uses partition-locked caches to prevent cache interference. Another cache design in [42] reduces cache miss rates by dynamic remapping and longer cache indices. SecDCP technique [43] can change 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], [33], and dynamic padding which means adding delay to a constant value [32], [44]–[46]. Besides these software methods, OS level defense methods are also proposed such as adding noise with periodic cache cleaning [47], also making encryption time to constant values by adding delay [48], [49], and using instruction-based scheduling [32], [50], [51]. 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 [33] suggests to add some dummy instructions or access some extra arrays to confusion the encryption time. A fixed number Of clock cycles AES implementation [52] is proposed by adding dynamic delay to each round. Rescheduling the instructions to confuse the cache misses is carried out in both software level [33], [53] and OS level [54]. The masking technique can be used to the cache attack-resistant algorithm implementations [8], [55]. In addition, a modified random permutation table method raised in paper [56], and a hardware-based method PRC confuse the cache misses to the attackers. The combination of blinding and delay is used in paper [57] 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 [33], [58], [59]. But it is useless in the remote environment because the attackers can use its own timers.

Combination methods. Some schemes combine the two s-

[2] According to [9], a longer encryption time results for the implementation [31] of AES running on 600-MHz Pentium III CPUs.

trategies to defend against the cache timing attack. The scheme proposed in [60] 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 [61], 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.

PRET [62], [63], a hardware architecture, is designed to replace caches with scratchpad memories. Also in this architecture, it will delay the encryption to the worst case execution time.

III. Eliminating Remote Cache Timing Side Channels

This section presents the threat mode and design goals. Then, the defense scheme is described, and we explain that it effectively eliminate remote cache timing side channels.

A. Threat Model and Design Goals

We focus on the remote cache side-channel attacks on block ciphers. The algorithms of block ciphers are publicly known, but the keys are kept secret before the attacks are launched. The implementation details such as source code, operating systems, cache hierarchy and other hardware configurations, are also publicly known; but the running states are unknown to the remote attackers due to non-deterministic interrupts, task scheduling and other system activities. In particular, the implementation is based on lookup tables, and the execution time depends mostly on cache misses and hits.

For the attackers, we assume that they can arbitrarily invoke the encryption/decryption functions and measure the time for each invocation. That means, the attackers control the invocation, so they can continuously invoke the functions to cache all tables or invoke no function for a long time to let all tables be evicted, with a high probability. We assume that attackers cannot run any processes on the target systems, and they just can obtain the overall execution time of the block ciphers but cannot get the middle-state of the ciphers.

In our model, the target operating system and the underlying hardware are regarded as security. We expect that attackers do not have physical access to the hardware and can not run any processes on the target as well. We are only concerned with the remote cache timing side channels since they are quite easy to carry out and need little ability of attackers.

We believe that it is a realistic model because it is reasonable for an remote attacker to have no access to the target system directly. And if the target system is not trusted, it can be attacked by other attacks, which are not in our consideration.

Our scheme attempts to eliminate remote cache timing side channels, with the following properties:

- It is applicable to various block ciphers and transparent to implementations. The scheme is algorithm-independent and works well for any implementation without any modifying the source code of encryption/decryption.
- It requires no privileged operations on the system. All operations introduced by the defense scheme, are available in common computer systems. It works well either in user space or kernel space, to protect the cryptographic operations in user mode and kernel mode, respectively.
- The parameters are configurable to optimize the performance. Then, it provides optimized performance by configuring appropriate parameters for different algorithms and platforms.

Algorithm 1 The Warm+Delay scheme

```
Input: $key$, $in$
Output: $out$
function ProtectedCrypt($key$, $in$, $out$)
    $start \leftarrow$ GetTime()
    Crypt($key, in, out$)   // encrypt or decrypt
    $end \leftarrow$ GetTime()
    if $end - start > TWOCM$ then
        Warm()
        $delay \leftarrow$ GetTime()
        if $delay - start < WET$ then
            Delay($WET - delay + start$)
        end if
    end if
end function
```

B. The Warm+Delay Scheme

Our scheme integrates two strategies to eliminate remote cache timing side channels [4], [5], [8], [25]: *a*) Warm, load the lookup tables into caches before the encryption/decryption operation; and *b*) Delay, insert padding instructions after the encryption/decryption operation. These two operations cooperatively destroy the relationship between the time measured by attackers and the cache misses/hits during the execution of encryption/decryption. In practice, warm cannot completely eliminate the cache timing side channels by itself, because the effect of cache warm may be countered by system activities, which evict lookup tables from caches. Delay directly controls the time measured by attackers, but it degrades the performance significantly.

The Warm+Delay scheme is described in Algorithm 1. Given a block cipher, the execution time without cache misses (TWOCM) and the worst execution time (WET) are constant on a certain platform.

Three operations are introduced by this defense scheme: *a*) Warm(), cache warm by accessing lookup tables as constant data, *b*) Delay(), delay by padding instructions, and *c*) GetTime(), timing for the conditions of warm and delay. All these operations are algorithms-independent. Moreover, they are implementation-transparent, except that the address of lookup tables is needed in cache warm.

These operations are supported in common computer systems, either in user mode or kernel mode. The conditions of warm and delay are defined by timing. We do not query the status of cache lines to judge whether an entry of the lookup tables is in caches or not, which are unavailable on common platforms.

In this algorithm, there are two constant parameters, WET and TWOCM. WET is defined as the period for one execution of encryption/decryption, in the case that none of lookup tables is cached before the execution. TWOCM is defined as the maximal period for one execution of encryption/decryption, in the case that all lookup tables are in caches before the execution. These two parameters are determined when there is no concurrent interrupts, task scheduling or other system activities.

In general, PROTECTEDCRYPT() are invoked continuously in a loop, to process a great deal of data, and the performance matters in such cases. So WARM() takes effect in the next execution of encryption/decryption. Therefore, if PROTECTEDCRYPT() has been idle for a long time, we suggest an additional invocation of WARM() before PROTECTEDCRYPT(). Note that, even if this "initial" warm is not performed, the security is still ensured by delay and the impact of performance is negligible if the loop of PROTECTEDCRYPT() is long enough.

C. Security Analysis

The remote attackers are (assumed to be) able to measure the time of each execution of PROTECTEDCRYPT(). The cache timing side channels come from the relation between the measured time and the data processed; that is, different data (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 information about the status of cache lines, and then destroy the relationship between the execution time and data processed in encryption/decryption.

According to Algorithm 1, the measured time, i.e., the execution time of PROTECTEDCRYPT(), falls into two intervals $(0, TWOCM]$ and $[WET, \infty)$. We prove that, no information about the status of cache lines leaks through the time measured by attackers.

- **Case 1**: The execution time of CRYPT() is in $(0, TWOCM]$. It is almost a fixed value for arbitrary data processed, as all tables are cached. When all table entries are in caches, the access time for any entry is constant, resulting in a fixed execution time.[3]
- **Case 2**: The time of encryption/decryption is in $(TWOCM, WET)$. It will be delayed to WET, which is the execution time as all accessed lookup tables are uncached. That is, the encryption/decryption is executed equivalently without any caches. In this case, the cache timing side-channel attackers cannot infer the relation between the measured time and the data processed.
- **Case 3**: The execution time is greater than WET. It results from task scheduling or interrupts. As explained in Section III-A, the running states such as interrupts, task scheduling and other system activities, are unknown to the remote attackers, so the attackers cannot obtain the actual execution time of encryption/decryption to infer the keys.

In summary, Cases 1 and 2 cannot be exploited in cache timing side-channel attacks, and in Case 3, the actual execution time is obscured by unknown system activities. Moreover, the sequence of Cases 1, 2, and 3, cannot be exploited to construct cache timing side channels. In Case 1, the measured time is almost constant, and no exploitable differential information exists. Cases 2 and 3 happen only if encryption/decryption is interrupted by non-deterministic activities, and the differential information is determined by the system events but not the keys.

In the following, we explain that our scheme successfully prevent typical remote cache timing side channel attacks. To perform the internal collision attacks [3], [19], the attackers need a collection of plaintexts with short (long) measured time due to cache hits (misses). However, protected by the WARM+DELAY scheme, any plaintext may be processed in the short time (i.e., TWOCM) due to cache warm, while any plaintext may be done in the long time (i.e., WET) due to delay. Therefore, the collections depend on the non-deterministic system activities, but not the data processed; or, the collection contains random plaintexts and the attack results will be random.

When the attackers build the time pattern for Bernstein's attack [4], it will be found that, the pattern reflects only the system activities and has nothing to do with the data processed. Since the system activities of target servers are different from those of the duplicated server and unknown to the attackers, this pattern cannot be used to infer the keys.

IV. PERFORMANCE ANALYSIS

A. Overview

In this section, we show that our solution is optimal in the category of approaches that do not modify encryption algorithm code. Intuitively, all of the following principles need to be satisfied in order to achieve the optimal performance:

- In the DELAY() operation, the imposed delay, or the execution time with added delay, is the shortest.
- The number of DELAY() operations is minimum.
- In the WARM() operation, only minimum data is loaded into cache.
- The number of WARM() operations is minimum.
- WARM() is more efficient than DELAY(), WARM() is preferred over DELAY().

We examine these principles with our proposed scheme, and derive practical conditions which make it optimal. We first examine the performance of DELAY(), and then analyze the WARM() operation with AES-128 with 2KB lookup tables [64]. In the proof, we show that it is statistically the most efficient to load all lookup tables in WARM(), instead of loading parts of the tables (which leads to more DELAY() in the future). Then we identify the best timing for the WARM() operation. Eventually, we derive a practical condition, so that our scheme is optimal. That is, performing warm operation

[3] The impact of micro-architecture [25] on cache access is discussed in Section **??**.

when the execution time lies in $(TWOCM, \infty)$ is the optimal method in our experiment environment. Last, we discuss the impact of different key lengths and different implementations.

B. Performance Analysis of Delay

Theorem 1. *In the* Delay() *operation, delay to WET imposes the minimal overhead while effectively defending against timing side-channel attacks.*

Proof. From the attackers' viewpoint, there are three types of encryption times that are useless (i.e. the execution time is independent of the plaintext input): the shortest time, WET, and any value that is larger than WET. The shortest time means that all the lookup tables are loaded into the cache with no cache miss during encryption. The WET corresponds to the longest execution path, when all lookup tables are not in the cache. When the execution time is longer than WET, the encryption has been disturbed by the OS (e.g. scheduler or interrupt handler), and it is impossible for the attacker to find the real encryption time.

If we delay the encryption time to a random value or a discrete value less than WET, there still exists an observable relationship between the encryption time and the input. The remote attacker can find the relationship by excessively invoking the encryption. Therefore, delaying to WET imposes the shortest delay that is effective against the side-channel attacks. □

Theorem 2. *Performing the* Delay() *operation when the encryption time is in $(TWOCM, WET)$ results the smallest number of* Delay() *operations.*

Proof. If the AES execution time is less than TWOCM, it means no cache miss has occurred, so that the encryption time is independent of the input. When the observed encryption time is no less than WET, it means that the OS scheduler or interruption handler has been invoked during encryption – the attacker could not recover the actual encryption time to perform timing attacks, because the time for OS scheduler or interruption handler is unpredictable.

When the execution time is in $(TWOCM, WET)$, it is because of cache miss, OS scheduler, or interrupt handler (the overhead is less than WET). The proposed scheme cannot distinguish the actual reason. However, the attacker can distinguish it by repeatedly invoking the encryption using the same input, and further perform timing attacks. Therefore, Delay() is necessary. □

C. Performance Analysis of Warm

We apply the Warm+Delay scheme to AES-128 implemented as described in Section II-A. The Warm() operation is applied to the lookup tables, which consume 2KB in total. We will show that in this case our Warm+Delay scheme gets the best performance.

Theorem 3. *For the hardware of commodity computers, loading all the AES lookup tables into the cache is the best warm operation.*

TABLE I: Reduced and introduced time by not loading N cache lines (in cycle).

N	1	2	3	5	10	15	20	
$E(T_{delay})$	2509.29	2524.90	2524.99	2525.00	2525.00	2525.00	2525.00	25
ΔT_{warm}	55.68	111.35	167.03	278.38	556.75	835.13	1113.5	13

Proof. In AES, Warm() loads all lookup tables into cache. In the case that at least one corresponding cache line is *not* loaded, cache miss may happen, which will in turn trigger Delay() operation. Therefore, the benefit of not loading N $(N > 0)$ cache lines of lookup tables is the saved loading time in Warm(), while the potential cost is the added time due to Delay(). In the following, we will prove that the expected cost is greater than the benefit, regardless of N.

For AES with 2KB lookup table implementation, it just use one lookup table T_e with 2KB. We assume that the size of a cache line is C Bytes, hence, T_e needs $2048/C$ cache lines.

We assume that each round of AES is independent in terms of cache access. T_e is accessed 16 times in each of AES rounds besides the last round. Therefore, the probability that one cache line of T_e is not accessed in all rounds is $P_e = (1 - \frac{C}{2048})^{160}$. Hence, the probability that N cache lines are not accessed in one execution is P_e^N. Therefore, the probability of *invoking* Delay() *while N cache lines of lookup tables are not in the cache* is shown in Equation 1.

$$P_{delay} = 1 - (P_e)^N \tag{1}$$

We assume the execution time of WET is W, and the shortest execution time (TWOCM) is B, then the expected overhead (i.e. the cost) of not loading N cache line of lookup tables (and invoking Delay()) is

$$E(T_{delay}) = P_{delay} * (W - B) = (1 - ((1 - \frac{C}{2048})^{160})^N)(W - B) \tag{2}$$

The average time saved by not loading one cache line from RAM is denoted as t_{nc}, then the benefit of not loading N cache lines is $\Delta T_{warm} = N * t_{nc}$.

Therefore, we have: (1) if there exists N that $\Delta T_{warm} > E(T_{delay})$, not warming N cache lines provides better performance. (2) If $\forall N > 0, \Delta T_{warm} < E(T_{delay})$, warming all cache lines is always better. (3) If $\exists N > 0, \Delta T_{warm} = E(T_{delay})$, warming or not warming N cache lines are equivalent.

In our experiment hardware, the size of a cache line C is 64 bytes. The time W (i.e. WET) is 2834 CPU cycles, while the shortest AES execution time B is 309 cycles. The average time t_{nc} for loading one cache line of lookup table is 55.68. From Table I, we can see that the cost of not loading N $(N > 0)$ cache lines of lookup tables is much higher than the benefit. Therefore, loading all the lookup tables into the cache provides the best performance in Warm(). □ □

In our Warm+Delay scheme, we perform Warm()after the AES execution. If the execution time is less than the WET, we perform Warm() and then Delay(). Therefore, the time of Warm() becomes part of delay. In this case, the overhead of Warm()is further reduced. When this case occurs, the extra time reduced by not warming some cache line is zero, if the AES execution time plus Warm() time is less than WET. Otherwise, the extra overhead (e.g. overhead beyond WET) is still less than WARM(). In this situation, warming all cache lines is still the better choice.

Note, in commodity hardware, the cache is enough to load all lookup tables. For example, loading all AES lookup tables needs 5KB, while loading all tables for 3DES needs 2KB. However, the typical L1D cache size is 32 or 64KB for Intel CPU, and 16KB or 32KB for ARM CPU.

Theorem 4. *For commodity computers, performing* WARM() *when cache-miss occurred during the previous execution results the minimum extra time.*

Proof. To prove Theorem 4, we compare it with two other cache warm strategies:

- **Less** WARM() **operations**. When there is a cache miss in the previous AES execution, we perform WARM() with a probability less than 1. In this case, DELAY()is needed. If the next encryption accesses any entry that is currently not in caches, performing WARM() operation before hand introduces none extra overhead – without WARM() all these entries will still be loaded into caches during encryption. If the next encryption accesses some uncached entries, as proved in Thorem 3, when N cache line size of lookup tables are not cached, not performing WARM() introduces more extra time.
- **More** WARM() **operations**. Performing the WARM() operation with a probability P_{warm}, regardless of the cache state after the previous execution. Denote this strategy as *probabilistic* WARM(), while our scheme as *conditional* WARM().

To prove the second case, we categorize the system state into three cases: (1) all the lookup tables are in the cache (S_{full}), (2) the lookup entries not in the cache are not accessed (S_{nonas}), and (3) the lookup entries not in the cache are accessed (S_{as}). We use the Markov model to analyse both *probabilistic* WARM() and *conditional* WARM(). S_{nonas} and S_{as} indicate the states that one or more cache lines of lookup tables are evicted. Fig. 1 shown the state transition diagram. In the diagram, P_{nona} (equals to $1-P_{delay}$) is the probability that the entries not in the cache are not needed in one execution; while P_{evict} is the probability that some entries are evicted from the cache. We use Π^p_{full} (Π^c_{full}), Π^p_{nonas} (Π^c_{nonas}), Π^p_{as} (Π^c_{as}) to represent the limit distribution of the three states when the Markov chain in stable state for the probabilistic and conditional WARM() respectively, and the values are as follows.

$$\Pi^p_{full} = \frac{P_{warm}}{P_{evict}+P_{warm}}, \Pi^p_{nonas} = \frac{P_{nona}P_{evict}}{P_{evict}+P_{warm}}, \Pi^p_{as} = \frac{P_{evict}(1-P_{nona})}{P_{evict}+P_{warm}} \tag{3}$$

$$\Pi^c_{full} = \frac{1-P_{nona}}{1-P_{nona}+P_{evict}}, \Pi^c_{nonas} = \frac{P_{nona}P_{evict}}{1-P_{nona}+P_{evict}}, \Pi^c_{as} = \frac{P_{evict}(1-P_{nona})}{1-P_{nona}+P_{evict}} \tag{4}$$

TABLE II: The values of k_0 corresponding to P_{evict}.

P_{evict}	0	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.
k_0	∞	216.7	109.8	57.4	40.7	33.1	29.0	26.7	25.4	24

The expected extra time introduced by the probabilistic and conditional WARM() are denoted as $E(T^p)$ and $E(T^c)$, respectively. t_{delay} and t_{warm} are the time needed to perform *one* DELAY() and *one* WARM() operation[4]. Then,

$$E(T^p)-E(T^c) = \Pi^p_{as}*t_{delay}+(1-\Pi^p_{as})*P_{warm}*t_{warm}-\Pi^c_{as}*t_{delay} \tag{5}$$

We use $k = t_{delay}/t_{warm}$, so $E(T^p) > E(T^c)$ holds when $1 \leq k \leq k_0$.

$$k_0 = \frac{(P_{delay}+P_{evict})*(1+P_{evict}-P_{evict}*P_{delay})}{P_{evict}*P_{delay}*(1-P_{delay})} \tag{6}$$

From Equation 1, we get $P_{delay} > 0.907$. When P_{evict} varies, the range of k_0 is listed in Table II ($P_{delay} = 0.907$). When P_{delay} is larger, k_0 gets larger. For example, in the environment described in Section IV, the maximum of t_{delay} is 2525.00 cycles, the minimum of t_{warm} is 124 cycles (accessing the lookup tables from caches). The maximum of t_{warm} is 1781.60 when all the lookup tables in RAM. Therefore, the range of k is: $1 < k < 20.36$.Also, we confirmed experimentally in Section III that P_{evict} is always less than 0.005. Therefore, the conditional WARM() provides better performance than the probabilistic WARM(). □ □

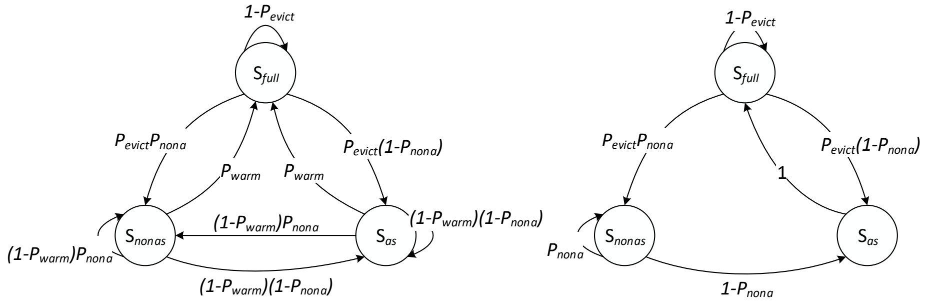
One AES execution is a partial warm operation. When the system is in the state S_{as}, one AES execution is a partial warm operation, as in this case, cache misses are resulted to load the corresponding entries into the cache.

When the system state is S_{as}, our scheme performs WARM() with probability 1, while the probabilistic WARM() performs WARM() with the probability P_{warm}, which is increased by the AES execution, a partial warm operation. However, according to Theorem 4, $E(T^p) > E(T^c)$ holds when $1 \leq k \leq k_0$, which is independent of P_{warm}. Therefore, our scheme is still better.

Process scheduling and interruption occur during the AES execution. In this case, $end - start > WET$, 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 $E(T_{warm,N}) = N * t_{nc} + (32 - N) * t_c$, where t_c and t_{nc} 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, denoted as $E(T_{delay})$, is shown in Equation 2. Table III shows that performing WARM()achieves better performance.

Periodical schedule function is called without scheduling during AES execution. As no other process invoked, the lookup tables are not evicted. Therefore, WARM() is unnecessary although $TWOCM < end - start \leq WET$. However,

[4] t_{delay} varies for different executions and t_{warm} varies when loading different size of lookup tables.



(a) probabilistic warmup

(b) conditional warmup

Fig. 1: The Markov state-transferring probability diagram.

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

N	1	2	3	5	10	15	20	25	30	32
$E(T_{delay})$	2509.29	2524.90	2524.99	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00
$E(T_{warm})$	175.80	227.60	279.4	383.00	642.00	901.00	1160.00	1419.00	1678.00	1781.60

TABLE IV: the minimum value of the P_{delay} in different cases.

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

WARM() is a better choice, as we cannot predict whether the lookup tables are in the cache, and the overhead introduced by accessing the elements in caches is concealed by DELAY().

The relationship between the cache state and execution time. In Theorems 3 and 4, we prove that performing WARM() when not all lookup tables 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:

- $end - start \leq TWOCM$, the system is in state S_{full} or S_{nonas}, no WARM() is needed according to Theorem 4.
- $end - start > WET$, the system is in state S_{as} or S_{nonas}, WARM() is needed according to Theorem 3.
- $TWOCM < end - start \leq WET$, the system is in the state S_{as} or S_{full}, WARM() is better according to previous analysis.

D. Other Key Lengths and Other Implementations

The WARM+DELAY scheme provides the optimized performance for various implementations of AES [64], [65] with different key length, once $1 \leq k \leq k_0$. All the proofs above stands, with the only difference be the value of P_{delay} (detailed in Appendix B). P_{delay} depends on the size of lookup tables and the number of iterated rounds. For mbed TLS-1.3.10 [65] and OpenSSL-0.9.7i [64], the size of lookup tables are 4.25KB and 5KB. For OpenSSL-1.0.2c [64], 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_{delay}.

For other table-lookup block ciphers, we have to determine the P_{delay} according to the algorithm and the implementation. Once $1 \leq k \leq k_0$ is satisfied , WARM+DELAY provides the optimized performance.

V. IMPLEMENTATION AND PERFORMANCE EVALUATION

The evaluation platform is a Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM. This CPU has 4 cores and each core has a 32KB L1 data cache. The operating system is 32-bit Linux with kernel version 3.6.2.

A. Implementation

We apply the WARM+DELAY scheme to AES-128. The implementation of AES employs the OpenSSL-1.0.2c [64] with a 2K lookup table. As we aim to provide the optimized performance while eliminating remote cache timing side channels, efficient GETTIME(), WARM() and DELAY() are finished; and the constant parameters (TWOCM and WET) are determined properly. Next, we show the implementation details about the WARM+DELAY scheme.

AES implementation. Even when all lookup tables are in the L1 cache, the encryption time still has some variations(Figure 2). There are two main reasons. One is the cache bank conflict. Each cache bank can only serve one request at a time. So, multiple accesses to the same cache bank are slower than to different banks. The other is that the load from L1 cache takes slightly more time if it involves the same set of cache lines as a recent store. These variations can be exploited by attackers [4], [25].

For the cache bank conflict, we first close the Hyper-Threading of the system to ensure the protected process itself not to produce concurrent access to the cache. Then in the remote environment, the attackers cannot run the attack process on the target, so the attack in [] can not be launched. We avoid the second cause by exploiting the stack switch technique [66]. 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 starting address of the array is made next to the lookup table module 4096, and this make the intermediate

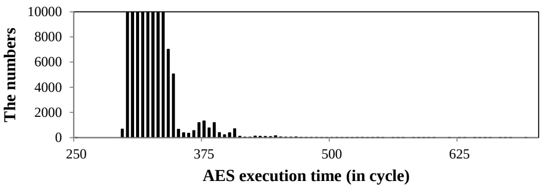
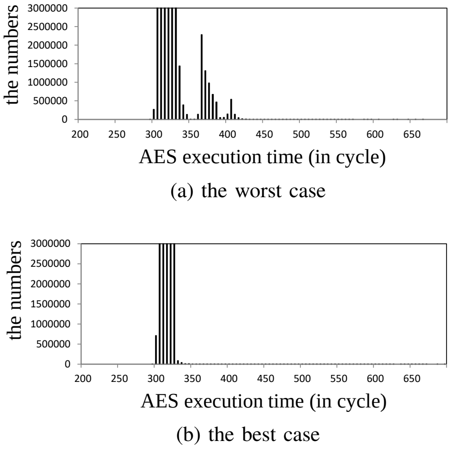


Fig. 2: The distribution of AES encryption time in full warm condition.



(a) the worst case

(b) the best case

Fig. 3: The distribution of AES encryption time with 2KB lookup table in full warm condition.

variables of AES execution use the remaining cache sets compared with the lookup table. In this case, the distribution of AES encryption for 2^{30} random plaintexts is shown in Figure 3b. Served as a contrast, the Figure 3a shows the distribution that the first address of the array is the same as the lookup table module 4096.

TWOCM. TWOCM 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. Figure 4 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. In Figure 4, the little data around 375 cycles are due to the above reasons, and the data after 415 are caused by the cache miss. Finally, we choose 355 cycles as TWOCM. 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.

WET. before measure WET, we flush both the data and instructions out of L1/2/3 caches, so WET is the worst AES execution time and unrelated to the cache states. WET is 2834 CPU cycles in this case.

WARM(). It accesses all blocks of the lookup table to load them into L1D cache and every time access one byte of each block of 64 bytes (i.e., the size of one cache line). In order to prevent the compiler optimization, the variables in the function are declared with keyword `volatile`.

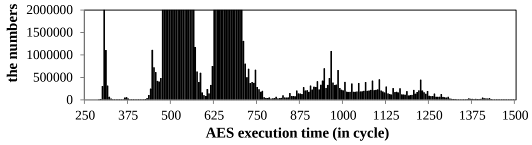


Fig. 4: The distribution of AES execution time only not warm one cache line.

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.

Listing 1: The implementation of DELAY().

```
volatile int delay(uint64_t t_delay){
    uint64_t n = (double)t_delay > 12.886 ?
            (uint64_t)((double)t_delay/2.995-4.302)
    for (; n>0; n--)
        asm volatile ("xor %%eax, %%eax;" : : : "%ea
}
```

DELAY(). The `usleep()` and `nanosleep()` are inappropriate, as they don't have enough accuracy, and they switch the state of AES execution process to TASK_INTERRUPTIBLE, which may make the lookup tables evicted from caches. We provide a new delay operation by executing xor instruction repeatedly, achieving a high precision without modifying the cache state. We measure the time cost for different loop number of the `xor` instruction, by invoking it 10^6 times with different loop numbers (from 0 to 2000 and the step is 50). The relation between the time delayed and the loop number of `xor` instruction is calculated through the least squares method. The equation is $t_{delay} = 2.995n + 12.886$, and the coefficient of determination is 0.999993. 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 1.1.

Besides, the cost of `RDTSCP` is 36 cycles which is larger than the comparison operation. To achieve better performance, we perform GETTIME() only when $delay - start < WET$, instead of every time after WARM(). In this way, if the input of DELAY() is less than zero, it simply returns.

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 TWOCM and no less than WET,

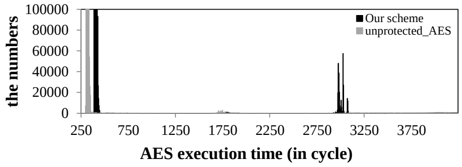


Fig. 5: The observed AES execution time with different plaintext.

which meets our expectation. Then, we evaluate the performance of WARM+DELAY scheme in three different aspects. Firstly we compare our scheme with different probabilistic 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. 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 in a real environment, the overhead of our scheme is insignificant.

The result of 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 TWOCM or no less than WET using the WARM+DELAY scheme. The time between TWOCM and WET in unprotected AES distribution is delayed to WET. 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 evaluate the probabilistic WARM() with the probability 0, 1/2, 1/3 and 1, and our scheme under low and high computing and memory workload when the interval of OS scheduler is 1ms and 4ms respectively. In each case, we perform 2^{30} AES encryptions for 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 reading 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 32KB block each time to operate the total 3GB data on one CPU core.

Figure 6 validates that the performance of WARM+DELAY scheme is the best. Moreover, we calculated P_{evict} under different workloads, according to the number of AES encryption whose execution time is greater than TWOCM. From Table V, we find P_{evict} is less than 0.005 always, in which case the WARM+DELAY scheme is the optimal as proved in Section IV.

Performance of different defense methods. Furthermore, we evaluate the performance of our scheme with different defense methods: AESNI, bitsliced AES implementation, compact table implementation. For each method, we we perform 2^{30} AES encryptions within 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.

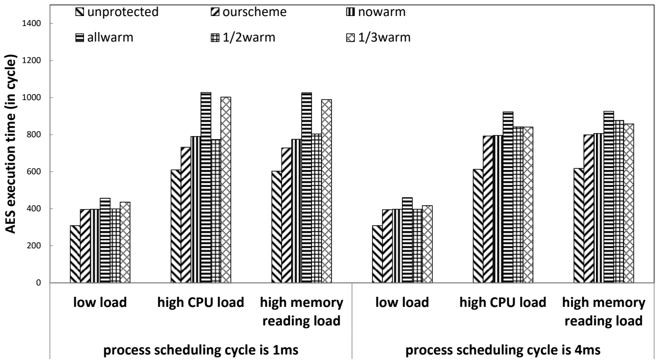


Fig. 6: AES execution performance in different scenarios.

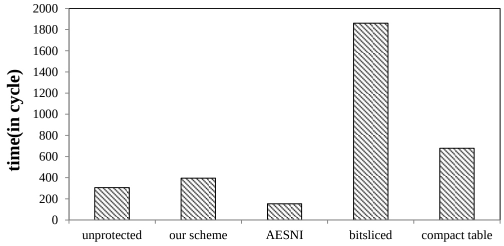


Fig. 7: Performance of different defense methods.

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 **??**. When the unprotected AES is used, the throughput is XXX requests per second. When using our scheme, the throughput is xxxx requests per second. It is clearly that WARM+DELAY scheme has a low influence on the performance of TLS protocol. Furthermore, we compare the throughput using different defense methods involved in TLS protocol. Our defense scheme has the largest throughput among these methods.

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

Interval of OS scheduler	Low workload	High CPU workload	High mem
1ms	0.0028	0.0037	0.00
4ms	0.0020	0.0026	0.00

VI. Discussions

A. Effect of Instruction Caches

Firstly, the implementation of block ciphers is not subject to timing side-channel attacks based on instruction caches [67], because there is generally no branch in the execution path. The status of instruction caches affect the execution time, but is not related to the data (i.e., keys and plaintexts/ciphertexts). The instructions of block ciphers may be evicted from the caches due to system activities, which causes instruction cache misses and increases the execution time.

As the effect of instruction caches on the execution time is indistinguishable from that of data caches, we determine TWOCM 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 TWOCM will offer attack opportunities because the measured time may leak some information about data cache access. Similarly, we determine WET 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 may also leak some information about data cache access.

B. Other Cache-based Side Channels

Beside the timing attack, there are two other kinds of cache-based side-channel attacks: the trace-driven attack and the access-driven attack. Firstly, both of these attacks require special privileges on the target system. During the execution of encryption/decryption, the trace-driven attackers monitor the variations of electromagnetic fields or power to capture the profile of cache activities and deduce cache hits and misses [2], [26], [68], while the access-driven attackers run a spy process on the target server accessing shared caches, to infer the cache status periodically [5], [8], [69], [70]. The attackers with such privileges are not considered in our scheme, and will be considered in our future work.

In principle, delay is not effective for trace-driven and access-driven attacks, because cache misses and hits exist during the execution of encryption/decryption. On the other hand, warm is effective for these attacks, provided that the cached lookup tables are not evicted by system activities or the local spy process.

VII. Conclusion

We propose the Warm+Delay scheme to eliminate the remote cache timing side channel attacks, for the block ciphers implemented based on lookup tables. Our scheme is applicable to all regular block ciphers, and transparent to implementations. The scheme works well in common computer systems, without any privileged operation.

We prove that, the Warm+Delay scheme destroys the relationship between the measured time and cache misses/hits, to ensure security. Then the scheme is applied to AES, and analyze the situations that it produces the optimized performance. Experimental results on the prototype system, confirm the security against remote cache timing side channels, and validate the performance optimization.

APPENDIX A
THE THE RANGE OF k

In this section, we describe how to calculate the range of k in details. From Table IV, we get $0.8 \leq P_{delay} \leq 1$. The conditions $0 \leq P_{warm} \leq 1$, $0 \leq P_{evict} \leq 1$,and $k > 1$ hold, obviously.

By combing Equation 3, 4 and 5, we get Equation 7 and 8. We need to determine the range of k, which makes $E(T^p) - E(T^c) > 0$ (i.e., $y(P_{warm}) > 0$).

$$\begin{aligned} y(P_{warm}) =& (P_{delay} + P_{evict}) * P_{warm}^2 \\ &+ P_{evict} * (P_{delay} + P_{evict})(1 - P_{delay}) * P_{warm} \\ &- k * P_{delay} * P_{evict} * P_{warm} + k * P_{delay}^2 * P_{evict} \ , \end{aligned} \tag{7}$$

$$E(T^p) - E(T^c) = \frac{y(P_{warm})}{(P_{warm} + P_{evict})(P_{delay} + P_{evict})} \tag{8}$$

The symmetry axis of $y(P_{warm})$ is $P_{warm} = P_a$, where P_a is denoted in Equation 9.

$$\begin{aligned} P_a =& \frac{1}{2(P_{evict} + P_{delay})}\{k * P_{evict}P_{delay} + P_{evict}^2 P_{delay} \\ &+ P_{evict}P_{delay}^2 - P_{evict}^2 - P_{evict}P_{delay}\} \ . \end{aligned} \tag{9}$$

Equation 10 holds when $0.8 \leq P_{delay} \leq 1$, $0 \leq P_{evict} \leq 1$, which means $P_a \geq 0$.

$$\begin{aligned} k > 1 >& \frac{P_{evict}^2 + P_{evict}P_{delay} - P_{evict}P_{delay}^2 - P_{evict}^2 P_{delay}}{P_{evict}P_{delay}} \\ =& \frac{P_{evict}}{P_{delay}} - P_{delay} + 1 - P_{evict} \ . \end{aligned} \tag{10}$$

1) If $P_a > 1$ (i.e., Equation 11 holds), the minimum value of $y(P_{warm})$ is $y(1)$. When $k\epsilon(k_1, k_0]$, $y(P_{warm}) \geq 0$ holds.

$$k > \frac{(P_{evict} + P_{delay})(2 + P_{evict} - P_{delay}P_{evict})}{P_{evict}P_{delay}} = k_1 \ . \tag{11}$$

$$k_1 < k \leq \frac{(1 + P_{evict} - P_{evict}P_{delay})(P_{evict} + P_{delay})}{(1 - P_{delay})P_{evict}P_{delay}} = k_0 \ . \tag{12}$$

2) If $0 \leq P_a \leq 1$, (i.e., $1 < k \leq k_1$), the minimum value of $y(P_{warm})$ is $y(P_a)$.

$$\begin{aligned} y(P_a) =& \frac{1}{4(P_{evict} + P_{delay})}\{-P_{evict}^2 P_{delay}^2 k^2 \\ &+ 2P_{evict}P_{delay}(P_{evict} + P_{delay})(2P_{delay} + P_{evict} - P_{evict}P_{delay})k \\ &- P_{evict}^2(P_{evict} + P_{delay})^2(1 - P_{delay})^2\}. \end{aligned} \tag{13}$$

$y(P_a)$ is a function of k (denoted as $f(k)$). As $-P_{evict}^2 P_{delay}^2 < 0$, the minimum of $f(k)$ is either $f(1)$ or $f(k_1)$, for $1 < k \leq k_1$. When $P_{delay} \in [0.8, 1]$, $f(1)$ and $f(k_2)$ is larger than 0, which means $y(P_a) > 0$

$$\begin{aligned} f(1) =& \frac{1}{4(P_{evict} + P_{delay})}\{-(1 - P_{delay})^2 P_{evict}^4 \\ &+ 2P_{evict}^3 P_{delay}^2 + 4P_{evict}P_{delay}^3 \\ &+ P_{evict}^2 P_{delay}^2(4 - P_{delay}^2 - 2P_{evict}P_{delay})\}. \end{aligned} \tag{14}$$

$$f(k_1) = (P_{evict}+P_{delay})(2P_{delay}-1+P_{evict}P_{delay}(1-P_{delay})). \tag{15}$$

From the above analysis, we find that when $1 < k \leq k_0$, $E(T^p) \geq E(T^c)$.

APPENDIX B
THE P_{delay} IN DIFFERENT AES KEY LENGTHES AND DIFFERENT AES IMPLEMENTATIONS

We assume the size of a cache line is C Byte. P_{delay} represents that the probability of performing delay operation with N cache line size of lookup tables not in the cache. So when the AES implementation uses 4.25KB lookup tables, the P_{delay} for AES-128 AES-192 and AES-256 are as follows respectively:

$$P_{delay}^{128} = 1 - (\frac{16}{17}(1 - \frac{C}{1024})^{36} + \frac{1}{17}(1 - \frac{C}{256})^{16})^N, \tag{16}$$

$$P_{delay}^{192} = 1 - (\frac{16}{17}(1 - \frac{C}{1024})^{44} + \frac{1}{17}(1 - \frac{C}{256})^{16})^N, \tag{17}$$

$$P_{delay}^{256} = 1 - (\frac{16}{17}(1 - \frac{C}{1024})^{52} + \frac{1}{17}(1 - \frac{C}{256})^{16})^N. \tag{18}$$

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

$$P_{delay}^{128} = 1 - (\frac{4}{5}(1 - \frac{C}{1024})^{36} + \frac{1}{5}(1 - \frac{C}{1024})^{16})^N, \tag{19}$$

$$P_{delay}^{192} = 1 - (\frac{4}{5}(1 - \frac{C}{1024})^{44} + \frac{1}{5}(1 - \frac{C}{1024})^{16})^N, \tag{20}$$

$$P_{delay}^{256} = 1 - (\frac{4}{5}(1 - \frac{C}{1024})^{52} + \frac{1}{5}(1 - \frac{C}{1024})^{16})^N. \tag{21}$$

When the AES implementation uses 4KB lookup tables, the P_{delay} for AES-128, AES-192 and AES-256 are as follows respectively:

$$P_{delay}^{128} = 1 - ((1 - \frac{C}{1024})^{40})^N, \tag{22}$$

$$P_{delay}^{192} = 1 - ((1 - \frac{C}{1024})^{48})^N, \tag{23}$$

$$P_{delay}^{256} = 1 - ((1 - \frac{C}{1024})^{56})^N. \tag{24}$$

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

$$P_{delay}^{128} = 1 - ((1 - \frac{C}{2048})^{16*10})^N, \tag{25}$$

$$P_{delay}^{192} = 1 - ((1 - \frac{C}{2048})^{16*12})^N, \tag{26}$$

$$P_{delay}^{256} = 1 - ((1 - \frac{C}{2048})^{16*14})^N. \tag{27}$$

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

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

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

[3] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *Cryptographic Hardware and Embedded Systems(CHES), International Workshop on*. 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 *Security and Privacy (S&P), 2011 IEEE Symposium on*. IEEE, 2011, pp. 490–505.

[6] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *16th International Cryptology Conference (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 *Information, Computer and Communications Security, ACM Symposium on*. ACM, 2006, pp. 369–369.

[8] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Cryptographers Track at the RSA Conference*. 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." *Proc of Ches Springer Lncs*, vol. 2779, pp. 62–76, 2003.

[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 *Cryptographic Hardware and Embedded Systems (CHES), 17th International Workshop on*. 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 *Information Technology: Coding and Computing (ITCC), 2005. International Conference on*, vol. 1. 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 *Cryptographic Hardware and Embedded Systems (CHES), 16th International Workshop on*. Springer, 2014, pp. 242–260.

[15] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *International Cryptology Conference(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 *Research in Computer Security, European Conference on*, 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 *Cryptographers Track at the RSA Conference*. 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 *Computer and communications security(CCS), 2012 ACM conference on*. 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 *Computer and communications security(CCS), 16th ACM conference on*. ACM, 2009, pp. 199–212.

[24] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A high resolution, low noise, l3 cache side-channel attack." in *23rd USENIX Security Symposium*, 2014, pp. 719–732.

[25] A. Canteaut, C. Lauradoux, and A. Seznec, "Understanding cache attacks," Ph.D. dissertation, INRIA, 2006.

[26] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[27] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," in *Cambridge Security Workshop on Fast Software Encryption (FSE)*, 1993, pp. 191–204.

[28] C. Adams, "Ietf rfc 2144: The CAST-128 encryption algorithm," 1997.

[29] "OpenSSH," http://www.openssh.com/.

[30] U. Drepper, "What every programmer should know about memory," Red Hat, Inc, Tech. Rep., 2007.

[31] "ANSI C Reference Code V2.0 (October 24, 2000). National Institute of Standards and Technology," http://csrc.nist.gov/CryptoToolkit/aes/rijndael/.

[32] D. Cock, Q. Ge, T. Murray, and G. Heiser, "The last mile: An empirical study of some timing channels on sel4," in *Computer and Communications Security(CCS), the 2014 ACM SIGSAC Conference on*. ACM, 2014, pp. 570–581.

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

[34] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *Cryptographic Hardware and Embedded Systems (CHES), 11th International Workshop on*. Springer, 2009, pp. 1–17.

[35] R. Könighofer, "A fast and cache-timing resistant implementation of the AES," in *The Cryptographers' Track at the RSA Conference (CT-RSA)*, 2008, pp. 187–202.

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

[37] D. Jayasinghe, J. Fernando, R. Herath, and R. Ragel, "Remote cache timing attack on advanced encryption standard and countermeasures," in *Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on*. IEEE, 2010, pp. 177–182.

[38] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud," in *USENIX Security symposium*, 2012, pp. 189–204.

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

[40] J. Kong, O. Acıiçmez, J. P. Seifert, and H. Zhou, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," in *High Performance Computer Architecture(HPCA). IEEE 15th International Symposium on*. IEEE, 2009, pp. 393–404.

[41] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Computer Architecture(ISCA), International Symposium on*. ACM, 2007, pp. 494–505.

[42] ——, "A novel cache architecture with enhanced performance and security," in *Microarchitecture, the 41st IEEE/ACM International Symposium on*. IEEE, 2008, pp. 83–93.

[43] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "Secdcp: secure dynamic cache partitioning for efficient timing channel protection," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[44] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *Computer and Communications Security(CCS), the 17th ACM SIGSAC Conference on*. ACM, 2010, pp. 297–307.

[45] C. Ferdinand, "Worst case execution time prediction by static program analysis," in *18th International Parallel and Distributed Processing Symposium (IPDPS), 2004*. IEEE, 2004, p. 125.

[46] D. Zhang, A. Askarov, and A. C. Myers, "Predictive mitigation of timing channels in interactive systems," in *Computer & communications security (CCS), ACM SIGSAC Conference on*. ACM, 2011, pp. 563–574.

[47] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Computer & communications security (CCS), ACM SIGSAC Conference on*. ACM, 2013, pp. 827–838.

[48] J. V. Cleemput, B. Coppens, and B. De Sutter, "Compiler mitigations for time attacks on modern x86 processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 23, 2012.

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

[50] 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 *European Symposium on Research in Computer Security*. Springer, 2013, pp. 718–735.

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

[52] U. Herath, J. Alawatugoda, and R. Ragel, "Software implementation level countermeasures against the cache timing attack on advanced encryption standard," in *Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on*. IEEE, 2013, pp. 75–80.

[53] D. Jayasinghe, R. Ragel, and D. Elkaduwe, "Constant time encryption as a countermeasure against remote cache timing attacks," in *Information and Automation for Sustainability (ICIAfS), 2012 IEEE 6th International Conference on*. IEEE, 2012, pp. 129–134.

[54] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz, "Thwarting cache side-channel attacks through dynamic software diversity," in *The 2015 Network and Distributed System Security Symposium (NDSS)*, 2015.

[55] J. Blömer, J. Guajardo, and V. Krummel, "Provably secure masking of AES," in *Selected Areas in Cryptography(SAC), International Workshop on*. Springer, 2004, pp. 69–83.

[56] J. Blömer and V. Krummel, "Analysis of countermeasures against access driven cache attacks on AES," in *Selected Areas in Cryptography (SAC), International Workshop on*. Springer, 2007, pp. 96–109.

[57] B. Kopf and M. Durmuth, "A provably secure and efficient countermeasure against timing attacks," in *Computer Security Foundations Symposium (CSF). IEEE*, 2009, pp. 324–335.

[58] P. Li, D. Gao, and M. K. Reiter, "Mitigating access-driven timing channels in clouds using stopwatch," in *Dependable Systems and Networks (DSN), 43rd Annual IEEE/IFIP International Conference on*. IEEE, 2013, pp. 1–12.

[59] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*. IEEE, 2012, pp. 118–129.

[60] 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.

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

[62] 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.

[63] I. Liu and D. McGrogan, "Elimination of side channel attacks on a precision timed architecture," *Technical Report*, 2009.

[64] "OpenSSL:Cryptography and SSL/TLS Toolkit," https://www.openssl.org/.

[65] "SSL Library mbed TLS / PolarSSL," https://tls.mbed.org/.

[66] L. Guan, J. Lin, B. Luo, and J. Jing, "Copker: Computing with private keys without ram." in *The 2014 Network and Distributed System Security Symposium (NDSS)*, 2014, pp. 23–26.

[67] O. Acıiçmez, "Yet another microarchitectural attack::exploiting I-cache," in *Computer security architecture, the 2007 ACM workshop on*. ACM, 2007, pp. 11–18.

[68] 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.

[69] M. Neve and J. P. Seifert, "Advances on access-driven cache attacks on AES," in *Selected Areas in Cryptography (SAC), International Workshop on*. Springer, 2006, pp. 147–162.

[70] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-vm attack on AES," in *Recent Advances in Intrusion Detection(RAID), International Workshop on*. Springer, 2014, pp. 299–319.

PLACE
PHOTO
HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.