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

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

***Abstract*—Cache timing side channels allow a remote attacker to disclose the cryptographic keys, only 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 great 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 integrating WARM and DELAY, and apply the scheme 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 to protect AES. Experimental results confirm that, (*a*) the execution time does not leak information about cache access, (*b*) the scheme outperforms other different 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 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 semantically secure. Among these vulnerabilities, timing side-channel attacks are easily launched without special probe devices. In particular, remote timing side channels allow attackers, who do not have any system or physical privilege on 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, passively exploits the time difference of data cache misses and hits. They are widely found in various implementations of block ciphers [3], [4], [9], [16], [19]–[21]. In these implementations, table lookup is the primary time-consuming operation in encryption and decryption. Because accessing data in caches is much faster than those in RAM chips, cache misses and hits in table lookup remarkably influence the overall execution time of block ciphers. Therefore, from the execution time, the attackers are able to infer the inputs of table lookup operations that 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 emphasize encryption only in the remainder, but the discussions and conclusions are also applicable to decryption.

Compared with other side channels on 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 sharing caches with the victim cryptographic engine [5], [22]–[24].

Different methods are 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 all lookup tables inside/outside caches, or *b*) in a constant period of time. In particular, WARM and DELAY are two typical algorithm-independent and implementation-transparent mechanisms to eliminate cache timing side channels [4], [5], [8], [25], [26]. WARM fills cache lines by reading constant lookup tables *before* the encryption operations, and DELAY inserts padding instructions *after* the encryption operations;[1] so the execution time measured by the attackers does not reflect the cache misses/hits *during* encryption.

These two common mechanisms prevent cache-based timing attacks at different phases, but introduce extra operations and degrade the performance greatly. In this paper, we investi-

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] Performing encryption always without caches is another choice, but it is very inefficient. Moreover, a typical strategy of DELAY is to finish the encryption operation in constant time that is equal to the execution time without any caches.

gate the performance of cryptographic functions protected by WARM and DELAY, and attempt to find the best strategy to integrate these countermeasures. In particular, the following principles are analyzed and discussed in the integration:

1) Each encryption is protected by WARM or DELAY, i.e., it is performed with all lookup tables in caches or it is finished after a constant period of time. 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).
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. Note that, compared with the best case protected by WARM, the worst case by DELAY costs about eight times of CPU cycles. Only when the effect of WARM is broken by system activities, such as interrupts and task scheduling, and information on the secret key may be leaked, the execution time is delayed in 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.

We apply the optimization integration to protect AES, and analyze the situations 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 computer systems, which is confirmed by our experiments. The conditions for other AES implementations of different key sizes are not identical but also hold in commodity systems.

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.

To the best of our knowledge, this work 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.

$$T_0[x] = (2 \cdot S(x), S(x), S(x), 3 \cdot S(x))$$
$$T_1[x] = (3 \cdot S(x), 2 \cdot S(x), S(x), S(x))$$
$$T_2[x] = (S(x), 3 \cdot S(x), 2 \cdot S(x), S(x))$$
$$T_3[x] = (S(x), S(x), 3 \cdot S(x), 2 \cdot S(x))$$

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 while no data-dependent branch in the execution path. For example, 3DES, Blowfish [28], CAST128 [29] in OpenSSH-7.4p1 [30].

B. Cache Timing Side Channel

Caches, a small amount of high-speed memory cells located between CPU cores and RAM, are designed to temporarily

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

store the data recently accessed by CPU cores, avoiding accessing the slow RAM chips. When the CPU core attempts to access a data block, the operation takes place in caches if the data have been cached (i.e., cache hit); otherwise, the data block is firstly read from RAM into caches (i.e., cache miss) and then the operation is performed in caches.

Cache timing side-channel attacks 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 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.

– Linjq check here.

Internal collision attack. This attack exploits the relationship between the encryption time and the different input plaintexts(or ciphertexts). Taking the first round attack of AES [3]as an example, the plaintext and the round key are denoted in bytes as $p_0, p_1, ..., p_{15}$ and $k_0, k_1, ..., k_{15}$, respectively, and 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 access the lookup table entries in a same cache line, it results in shorter execution time, which is called an internal collision. Then, the attacker can make a statistics of the average execution time for all possible $p_{(i+4j)\%16} \oplus p_{(i+4k)\%16}$ $(0 \leq j, k \leq 3)$. The lowest time at the value $p_{(i+4j)\%16} \oplus p_{(i+4k)\%16} = \delta$ represents that cache collision occurs which means $k_{(i+4j)\%16} \oplus k_{(i+4k)\%16} = \delta$. In this way the secret key is extracted.

Since each cache line (64 bytes) contains multiple entries, the low 4 bits of k_i can not be determined using the first round attack. The attacker need further analysis or more data to extract the full key. This attack is extended to the second and last rounds of AES [19] that can extract the full secret key without extra information. Also the internal collision attack works for DES, 3DES [9].

Bernstein's attack. This attack is based on the case that the whole AES execution time is correlated with the time for accessing the lookup tables, which depends on input plaintexts [4]. 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 repeats the same experiment on the victim server and infers the key bytes of the victim server with the obtained lookup table index and known plaintexts ($k_i = p_i^{'} \oplus k_i^{'} \oplus p_i$).

C. Countermeasures against Cache Timing Side Channels

Although eliminating timing side channels remains difficult [32], 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 [32]–[34]. Some using particular implementation to make algorithms constant time are just suitable to specific algorithms [35]–[37].

Some introducing many extra operations to eliminate or confuse the cache misses incur significant performance overheads [5], [8], [26], [38].

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 [39], 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], [35], [36] 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], [37] and loading the lookup tables into registers [8], [38]. 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], [39], 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 [40] allows each VM to load the sensitive data into its own locked cache lines. A hardware-based mechanism presented in [33], allows caches to be configured dynamically to match the need of a process. The new cache design [41], [42] uses partition-locked caches to prevent cache interference. Another cache design in [43] reduces cache miss rates by dynamic remapping and longer cache indices. SecDCP [44] 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], [39], and dynamic padding which means adding delay to a constant value [26], [32], [45], [46]. Besides these software methods, OS level defense methods are also proposed such as adding noise with periodic cache cleaning [34], also making encryption time to constant values by adding delay [47], [48], and using instruction-based scheduling [32], [49], [50]. 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 [39] suggests to add some dummy instructions or access some extra arrays to confusion the encryption time. A fixed number Of clock cycles AES implementation [51] is proposed by adding dynamic delay to each round. Rescheduling the instructions to confuse the cache misses is carried out in both software level [39], [52] and OS level [53]. The masking technique can be used to the cache attack-resistant algorithm implementations [8], [54]. In addition, a modified random permutation table method raised in paper [55], and a hardware-based method PRC confuse the cache misses to the attackers. The combination of blinding and delay is used in paper [56] 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 [39], [57], [58]. 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 [59] 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 [60], 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 [61], [62] replaces 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 by Integrating Warm and Delay

[[[[[–Linjq restart here; I skip Section II.

This section firstly presents the threat mode and design goals. Then, the WARM+DELAY scheme is described, and we explain that it effectively eliminates remote cache timing side channels while keeps the potential to achieve the optimal performance.

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 system, cache hierarchy and other software/hardware configurations, are also publicly known; but the running states of the target system are unknown to the remote attackers due to non-deterministic interrupts, task scheduling and other system activities. In particular, the implementation of block ciphers is based on lookup tables, and the execution time depends mostly on the cache access of table lookup.

The attackers are able to invoke the encryption function arbitrarily and measure the time for each invocation. The attackers control the invocation, so they could continuously invoke the function to cache all lookup tables or invoke no function for a long time to let all tables be evicted from caches, with an extremely high probability.

We assume an unprivileged attacker that cannot run a processes on the target systems to concurrently modify or probe the cache state of the system, and they just measure the overall execution time of the block ciphers. The attackers do not have physical access to the hardware.

This work focuses on the remote cache timing side-channel attacks, which are launched with the least privileges (or capabilities) of attackers; other cache timing side-channels by active attackers (such as Flush+Reload [24] and Prime+Probe [63]) are out of the scope of this paper. We attempt to eliminate remote cache timing side channels, with the following considerations:

- It is applicable to various block ciphers and transparent to implementations. The scheme is algorithm-independent and works well for different implementations without modifying the source code.
- 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 block cipher implemented in user mode and kernel mode, respectively.
- The parameters are configurable to optimize the performance. Then, it produces the optimal performance by configuring appropriate parameters for different block-cipher implementations and computing platforms.
- We attempt to optimize the performance while ensure the security. So we focus on the situation where large amounts of data are encrypted and then the speed of encryption matters.

B. The WARM+DELAY Scheme

There are two basic countermeasures against remote cache timing side channels [4], [5], [8], [25]: *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.

A naive integration is to perform both WARM and DELAY 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); similarly, WARM is performed only when some entries are not in caches. As the scheme does not involve privileged operations, the conditions of WARM and DELAY are defined as regular timing.

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 $end - start > T_{NM}$ then
        Warm()
        $delay \leftarrow$ GetTime()
        if $delay - start < T_W$ then
            Delay($T_W - delay + start$)
        end if
    end if
end function
```

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. In this algorithm, there are two parameters, T_W and T_{NM}. T_{NM} is defined as the time period for one execution of encryption, in the case that all lookup tables are in caches before the execution. T_W is defined as the period for one execution of encryption, in the case that none of table entries is cached before the execution. 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 memory address of lookup tables is 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, not shown in Algorithm 1. 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.

C. Security Analysis

The remote attackers are (assumed to be) able to measure the time of each execution of ProtectedCrypt, but not internal Crypt. 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 cache lines, and then destroy the relationship between the execution time and data processed during encryption.

According to Algorithm 1, the measured time, i.e., the execution time of ProtectedCrypt, falls into two intervals, $(0, T_{NM}]$ and $[T_W, \infty)$. Next, 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 $(0, T_{NM}]$. 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.
- **Case 2**: The execution time is in (T_{NM}, T_W). It will be delayed to T_W, which is the execution time as all accessed lookup tables are uncached before encryption. That is, the encryption 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 of Crypt is greater than T_W. 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 attackers cannot find the actual execution time of encryption to infer the keys.

The timing information and also its differential information, are determined by non-deterministic system activities and unrelated to the secret keys. Cases 1 and 2 cannot be exploited in cache timing side-channel attacks, and in Case 3, the execution time is obscured by non-deterministic system activities. 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 is interrupted by non-deterministic system activities. So the differential information is determined only by the system events.

In the following, we further explain that the integration 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 is processed in the short time (i.e., T_{NM}) due to WARM, or done in the long time (i.e., T_W or longer) due to system activities after DELAY. Therefore, the distribution of the collection depends on non-deterministic system activities, but not the data processed; or, 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 the target servers are different from those of the duplicated server and unknown to the attackers, this pattern cannot be used to infer the keys in the target server.

IV. TOWARDS THE OPTIMAL PERFORMANCE OF WARM+DELAY

In this section, we prove that the WARM+DELAY scheme achieves the optimal performance with the least extra operations, by applying it to AES and analyzing the situations that it produces the optimal performance.

A. Overview

In the scheme, the extra operations are introduced by WARM, DELAY and GETTIME. The cost of GETTIME is fixed and necessary, so the overheads are determined by the cost of each WARM (or DELAY) operation, and the times of these operations. Therefore, the following principles need to be satisfied to achieve the optimal performance:

- In the DELAY operation, the imposed delay, or the execution time of padding instructions, is the shortest.
- The number of DELAY operations is minimum.
- In the WARM operation, only the minimum necessary data are loaded into caches.
- The number of WARM operations is minimum.
- Because WARM is much more efficient than DELAY,[3] WARM is preferred over DELAY.

Next, we examine these principles with the WARM+DELAY scheme, and derive the practical conditions which make it optimal. We first examine the performance of DELAY, and then analyze the WARM operation for 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 probably leads to more DELAY in the future). Then we identify the best condition for the WARM operation, and eventually derive the practical condition so that the WARM+DELAY scheme (with appropriate parameters) is optimal.

Last, 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. Performance Analysis of DELAY

Theorem 1. *In the* DELAY *operation, delay to T_W imposes the minimal overhead while effectively eliminates the cache timing side channels.*

Proof: From the attackers' point of view, there 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): T_{NM}, T_W, and any value greater than T_W. T_{NM} means that *all* accessed entries are in caches before encryption, and T_W corresponds to the longest execution path, when all lookup tables are not cached before encryption (and they are loaded into caches as the encryption operation is performed); so such results are the same for any input. When the measured time is longer than T_W, encryption is disturbed by unknown system activities (e.g. interrupts and task scheduling), and it is impossible for the attackers to exclude the disturbance and find the exact execution time of encryption.

If we delay the execution time to any value less than T_W, a little information about the cache access leaks. Or, 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 when the execution time of encryption is in (T_{NM}, T_W) results in the smallest 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. ■

C. Performance Analysis of WARM

We apply the WARM+DELAY scheme to AES-128 with 2KB lookup tables [64], as described in Section II-A. We show that, in this case, the WARM+DELAY scheme produces the best performance in commodity computer systems, that is, introduces the least extra WARM and DELAY operations. The details about other key sizes (192 and 256 bits) and different implementations (4KB, 4.25KB, and 5KB lookup tables), are included in Appendix B.

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

[3] It is true for any computing platform in practice; otherwise, caches are useless in performance improvement.

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 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: It takes $T_{cl} > 0$ to load data into a cache line in the WARM operation; on the other hand, if some entry is uncached, the execution time of AES encryption may become longer and then it triggers an extra DELAY operation. 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 $P_{\bar{a}ccess}(N)$, is $P_{\bar{a}ccess}(N) = (1 - \frac{NC}{L})^{16R}$.

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 listed as follows.

$$P_{delay} = 1 - P_{\bar{a}ccess}(N) = 1 - (1 - \frac{NC}{L})^{16R} \qquad (1)$$

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 the expected overhead (or the time cost) due to not loading N cache lines of table entries is

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

Finally, the benefit of not loading data into N cache lines in the WARM operation is

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

Finally, (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. ■

In the WARM+DELAY scheme, WARM is performed after encryption as a part of the DELAY operation, as shown in Algorithm 1. So the *actual* cost of loading data into N cache lines (or the benefit of not loading these data) in WARM is less than $B_{nl}(N)$. For example, when the time of CRYPT plus WARM is less than T_W (Line 8 in Algorithm 1), the cost of WARM is actually zero.

Therefore, the average cost of loading data into N cache lines in the WARM operation is *actually* less than NT_{cl}, so loading all entries in WARM still provides better performance than any part of entries if $B_{nl}(N) = NT_{cl} < D_{nl}(N)$ for any $\frac{L}{C} \geq N > 0$.

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 any part of entries.*

Proof: Firstly, with commodity hardware, the cache is enough to load all entries of the lookup table. For example, AES is implemented with the lookup table 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 I 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 is always much greater than the benefit. The result is applicable to other commodity computer systems. So loading all entries in WARM produces the optimal performance. ■

]]]]]]]]

Next, let us consider the different states of lookup tables in the L1 cache during the continuous invocations of AES encryption. The state of lookup tables is estimated after each execution of AES encryption. There are three states: (1) all entries of the lookup tables are in caches, denoted as S_c, (2) some entries are uncached, and at least one of them is accessed during the AES encryption, denoted as $S_{\bar{c},a}$, and (3) some entries are uncached, but none of them is accessed during the AES encryption, denoted as $S_{\bar{c},\bar{a}}$. $S_{\bar{c},a}$ 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 transition among the three states can be triggered by three events. First is the *Evict* event. If task scheduling, interrupts or any other system activities occur during the execution of encryptions, the entries of lookup tables may be evicted from the cache. The probability that eviction occurs is denoted as P_{evict}. Second is the *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 is P_{delay} in Theorem 3. Third is the *Warm* event. Performing WARM loads all the lookup tables into the cache which probability is denoted as P_{warm}.

Denote the execution time of DELAY and WARM as T_{delay} and T_{warm}, respectively. $T_{delay} = T_W - T_{NM}$ is constant for different executions, while T_{warm} depends on the number of loaded cache lines of table entries.

The occurrence of these three events will change the state of lookup tables in the cache. In the process of state transition, the state of lookup tables changes which means one AES executes. 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. Also 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 WARM+DELAY scheme, the warm strategy that perform WARM when some cache misses occur during the pervious execution of encryption is denoted as conditional WARM.

TABLE I: 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	1113.5s	1391.88	1670.25	1725.93	1781.60
$D_{nl}(N)$	2509.29	2524.92	2524.99	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00

While considering other two cases: (1)performing WARM with a probability $P^{\mathcal{L}}_{warm} \in [0, 1)$ when a cache miss occurs during the pervious execution of encryption; (2)performing WARM when some cache misses occur during the pervious execution of encryption and also performing WARM with a probability $P^{\mathcal{M}}_{warm} \in [0, 1]$ in other conditions. These two cases are denoted as less WARM and more WARM respectively.

For conditional WARM, the Markov state transition diagram is in Fig. 1a. P^{1}_{a}, P^{21}_{a} and P^{22}_{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^{21}_{a} and P^{22}_{a} is no less than P^{1}_{a}. Because we can not determine how many lookup entries are evicted, the ranges of P^{1}_{a}, P^{21}_{a} and P^{22}_{a} are the same. So we use P^{2}_{a} to represent $P^{2}_{evict}P^{21}_{a} + (1 - P^{2}_{evict})P^{22}_{a}$ and P^{2}_{a} has the same range with P^{1}_{a}.

We use $\Pi^{\mathcal{C}}_{c}$, $\Pi^{\mathcal{C}}_{\bar{c},\bar{a}}$, $\Pi^{\mathcal{C}}_{\bar{c},a}$ to represent the limit distribution of the three states when the Markov chain is in stable state, and the values are as follows.

$$\begin{aligned}
\Pi^{\mathcal{C}}_{c} &= \frac{P^{2}_{a}(1 - P_{evict})}{P^{2}_{a} + P_{evict}(1 - P^{1}_{a})} \ , \\
\Pi^{\mathcal{C}}_{\bar{c},\bar{a}} &= \frac{P_{evict}(1 - P^{1}_{a})}{P^{2}_{a} + P_{evict}(1 - P^{1}_{a})} \ , \\
\Pi^{\mathcal{C}}_{\bar{c},a} &= \frac{P_{evict}P^{2}_{a}}{P^{2}_{a} + P_{evict}(1 - P^{1}_{a})} .
\end{aligned} \tag{4}$$

For less WARM, the Markov state transition diagram is in Fig. 1b. P^{1}_{a}, $P^{21\mathcal{L}}_{a}$ and $P^{22\mathcal{L}}_{a}$ 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, $P^{21\mathcal{L}}_{a}$ and $P^{22\mathcal{L}}_{a}$ is no less than P^{1}_{a}. Also the ranges of P^{1}_{a}, $P^{21\mathcal{L}}_{a}$ and $P^{22\mathcal{L}}_{a}$ are the same. So we use $P^{2\mathcal{L}}_{a}$ to represent $P^{2\mathcal{L}}_{evict}P^{21\mathcal{L}}_{a} + (1 - P^{2\mathcal{L}}_{evict})P^{22\mathcal{L}}_{a}$ and $P^{2\mathcal{L}}_{a}$ has the same range with P^{1}_{a}. $P^{3\mathcal{L}}_{a}$ in the diagram has the same meaning with $P^{2\mathcal{L}}_{a}$.

We use $\Pi^{\mathcal{L}}_{c}$, $\Pi^{\mathcal{L}}_{\bar{c},\bar{a}}$, $\Pi^{\mathcal{L}}_{\bar{c},a}$ to represent the limit distribution of the three states when the Markov chain is in stable state, and the values are as follows.

$$\begin{aligned}
\Pi^{\mathcal{L}}_{c} &= \frac{(1 - P_{evict})P^{2\mathcal{L}}_{a}P^{\mathcal{L}}_{warm}}{DenominatorL} \ , \\
\Pi^{\mathcal{L}}_{\bar{c},\bar{a}} &= \frac{P_{evict}(1 - P^{3\mathcal{L}}_{a} + (P^{3\mathcal{L}}_{a} - P^{1}_{a})P^{\mathcal{L}}_{warm})}{DenominatorL} \\
\Pi^{\mathcal{L}}_{\bar{c},a} &= \frac{P_{evict}P^{2\mathcal{L}}_{a}}{DenominatorL} .
\end{aligned} \tag{5}$$

where

$$\begin{aligned}
DenominatorL = P_{evict}&(1 + P^{2\mathcal{L}}_{a} - P^{3\mathcal{L}}_{a}) \\
&+ (1 - P_{evict})P^{2\mathcal{L}}_{a}P^{\mathcal{L}}_{warm} \\
&+ (P^{3\mathcal{L}}_{a} - P^{1}_{a})P_{evict}P^{\mathcal{L}}_{warm}
\end{aligned}$$

For more WARM, the Markov state transition diagram is in Fig. 1c. P^{1}_{a}, $P^{21\mathcal{M}}_{a}$ and $P^{22\mathcal{M}}_{a}$ 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, $P^{21\mathcal{M}}_{a}$ and $P^{22\mathcal{M}}_{a}$ is no less than P^{1}_{a}. Also the ranges of P^{1}_{a}, $P^{21\mathcal{M}}_{a}$ and $P^{22\mathcal{M}}_{a}$ are the same. So we use $P^{2\mathcal{M}}_{a}$ to represent $P^{2\mathcal{M}}_{evict}P^{21\mathcal{M}}_{a} + (1 - P^{2\mathcal{M}}_{evict})P^{22\mathcal{M}}_{a}$ and $P^{2\mathcal{M}}_{a}$ has the same range with P^{1}_{a}.

We use $\Pi^{\mathcal{M}}_{c}$, $\Pi^{\mathcal{M}}_{\bar{c},\bar{a}}$, $\Pi^{\mathcal{M}}_{\bar{c},a}$ to represent the limit distribution of the three states when the Markov chain is in stable state, and the values are as follows.

$$\begin{aligned}
\Pi^{\mathcal{M}}_{c} &= \frac{(1 - P_{evict})(P^{2\mathcal{M}}_{a}(1 - P^{\mathcal{M}}_{warm}) + P^{\mathcal{M}}_{warm})}{DenominatorM} \ , \\
\Pi^{\mathcal{M}}_{\bar{c},\bar{a}} &= \frac{P_{evict}(1 - P^{1}_{a})}{DenominatorM} \ , \\
\Pi^{\mathcal{M}}_{\bar{c},a} &= \frac{P_{evict}P^{1}_{a}P^{\mathcal{M}}_{warm} + P_{evict}P^{2\mathcal{M}}_{a}(1 - P^{\mathcal{M}}_{warm})}{DenominatorM} .
\end{aligned} \tag{6}$$

where

$$\begin{aligned}
DenominatorM = P_{evict} &+ P^{\mathcal{M}}_{warm} + P^{2\mathcal{M}}_{a} - P^{2\mathcal{M}}_{a}P^{\mathcal{M}}_{warm} \\
&- P_{evict}(P^{\mathcal{M}}_{warm} + P^{1}_{a} - P^{1}_{a}P^{\mathcal{M}}_{warm})
\end{aligned}$$

Then we have the following theorems.

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

Proof: The expected extra time introduced by less WARM and conditional WARM are denoted as $E(T^{\mathcal{L}})$ and $E(T^{\mathcal{C}})$, respectively. So,

$$E(T^{\mathcal{L}}) - E(T^{\mathcal{C}}) = \Pi^{\mathcal{L}}_{\bar{c},a} * T_{delay} - \Pi^{\mathcal{C}}_{\bar{c},a} * T_{delay} \tag{7}$$

And then, we get that $E(T^{\mathcal{L}}) > E(T^{\mathcal{C}})$ for all $P^{\mathcal{L}}_{warm} \in (0, 1)$. 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. ■

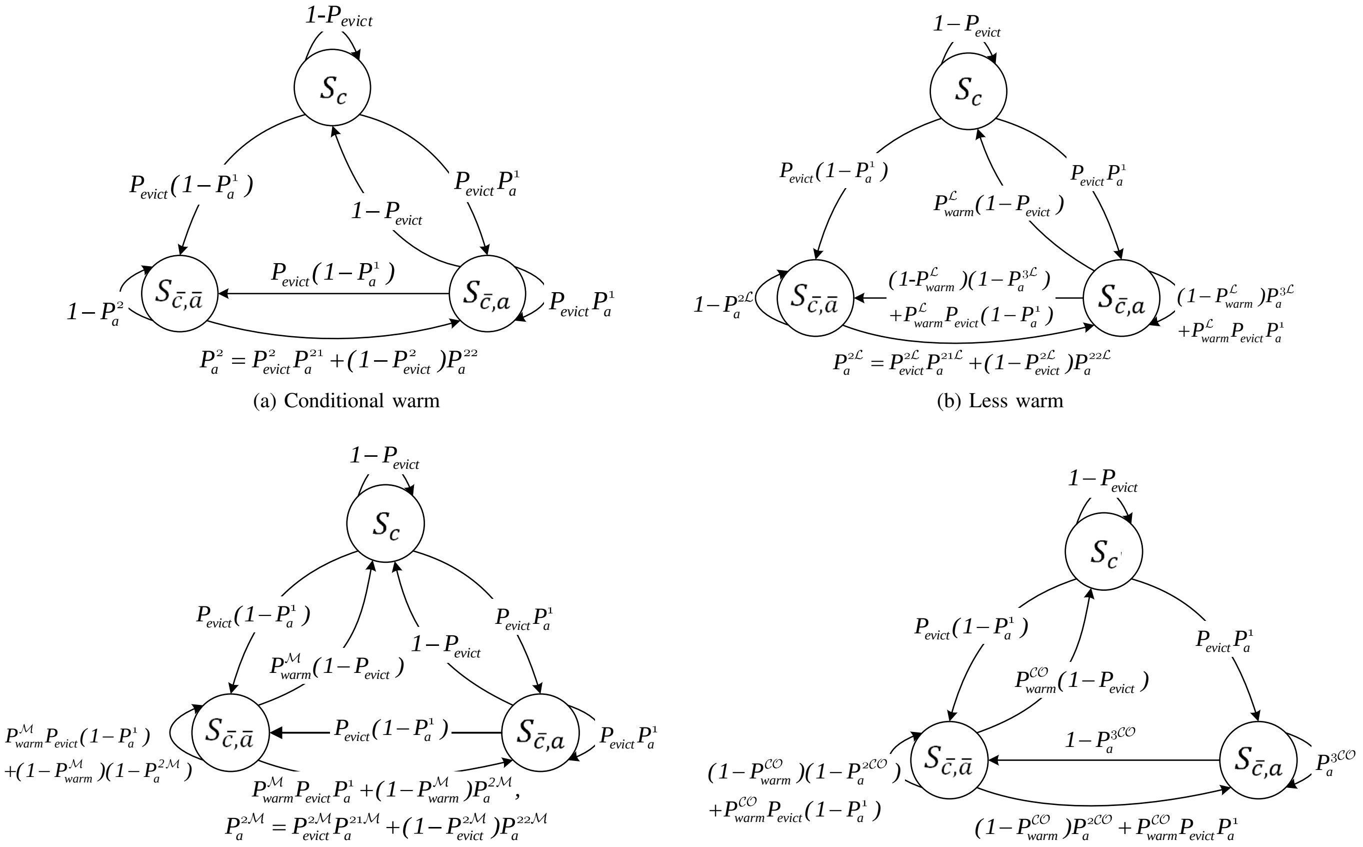
Let $g = T_{delay}/T_{warm}$ and $M = T_{warm_{min}}/T_{delay}$. We denote $P_{evict}(P^{2}_{a} - P_{evict}P^{1}_{a})(1 - P^{1}_{a}) - M(1 - P_{evict})P^{2}_{a}(P^{2}_{a} + P_{evict}(1 - P^{1}_{a}))$ as $GDenominator$. We denote $\frac{P_{evict}(1-P^{1}_{a})(P^{2}_{a}+P_{evict}(1-P^{1}_{a}))}{GDenominator}$ as $Gexpression$.

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

1) *$GDenominator \leq 0$;*
2) *or if $GDenominator > 0$, g is smaller than the minimum value of $Gexpression$.*

Proof: The expected extra time introduced by more WARM and conditional WARM are denoted as $E(T^{\mathcal{M}})$ and $E(T^{\mathcal{C}})$, respectively. Then,

$$\begin{aligned}
E(T^{\mathcal{M}}) - E(T^{\mathcal{C}}) = \Pi^{\mathcal{M}}_{\bar{c},a} * T_{delay} &+ \Pi^{\mathcal{M}}_{c} * P_{warm} * T_{warm_{min}} \\
&+ \Pi^{\mathcal{M}}_{\bar{c},\bar{a}} * P_{warm} * T_{warm} - \Pi^{\mathcal{C}}_{\bar{c},a} * T_{delay}
\end{aligned} \tag{8}$$


$1\text{-}P_{evict}$
S_c
$P_{evict}(1-P_a^1)$
$P_{evict}P_a^1$
$1-P_{evict}$
$S_{\bar{c},\bar{a}}$
$S_{\bar{c},a}$
$1-P_a^2$
$P_{evict}P_a^1$
$P_a^2 = P_{evict}^2 P_a^{21} + (1-P_{evict}^2)P_a^{22}$
$P_{warm}^{\mathcal{L}}(1-P_{evict})$
$1-P_a^{2\mathcal{L}}$
$(1\text{-}P_{warm}^{\mathcal{L}})(1-P_a^{3\mathcal{L}})$
$+P_{warm}^{\mathcal{L}}P_{evict}(1-P_a^1)$
$(1-P_{warm}^{\mathcal{L}})P_a^{3\mathcal{L}}$
$+P_{warm}^{\mathcal{L}}P_{evict}P_a^1$
$P_a^{2\mathcal{L}} = P_{evict}^{2\mathcal{L}}P_a^{21\mathcal{L}} + (1-P_{evict}^{2\mathcal{L}})P_a^{22\mathcal{L}}$
$P_{warm}^{\mathcal{M}}(1-P_{evict})$
$P_{warm}^{\mathcal{M}}P_{evict}(1-P_a^1)$
$+(1-P_{warm}^{\mathcal{M}})(1-P_a^{2\mathcal{M}})$
$P_{warm}^{\mathcal{M}}P_{evict}P_a^1 + (1-P_{warm}^{\mathcal{M}})P_a^{2\mathcal{M}}$,
$P_a^{2\mathcal{M}} = P_{evict}^{2\mathcal{M}}P_a^{21\mathcal{M}} + (1-P_{evict}^{2\mathcal{M}})P_a^{22\mathcal{M}}$
$S_{c'}$
$P_{warm}^{\mathcal{CO}}(1-P_{evict})$
$1-P_a^{3\mathcal{CO}}$
$P_a^{3\mathcal{CO}}$
$(1-P_{warm}^{\mathcal{CO}})(1-P_a^{2\mathcal{CO}})$
$+P_{warm}^{\mathcal{CO}}P_{evict}(1-P_a^1)$
$(1-P_{warm}^{\mathcal{CO}})P_a^{2\mathcal{CO}} + P_{warm}^{\mathcal{CO}}P_{evict}P_a^1$

(a) Conditional warm

(b) Less warm

(c) More warm

(d) Complementary warm

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

Also, we find that $P_a^2 = P_a^{2\mathcal{M}}$. Let $E(T^{\mathcal{M}}) - E(T^{\mathcal{C}}) > 0$, by a serials of calculation(See Appendix A for more details), we get the condition that

1) $GDenominator \le 0$;
2) or if $GDenominator > 0$, g is smaller than the minimum value of $Gexpression$.

While $Gexpression$ is a monotonous increasing function of P_a^1. But for P_a^2 and P_{evict}, it is not a monotonous function. The minimum value of $Gexpression$ achieves when P_a^1 achieves its minimum value, and P_a^2 and P_{evict} satisfy the following equations.

$$\begin{cases} P_{evict} = \dfrac{MP_a^1(1-P_a^2) + \sqrt{\Delta}}{(1-P_a^1)(1-M+MP_a^1)} \\ P_a^2 = P_{evict}\sqrt{\dfrac{1-P_a^1}{M(1-P_{evict})}} - P_{evict}(1-P_a^1) \\ if\ P_{evict} > 1, P_{evict} = 1 \\ if\ P_a^2 > 1, P_a^2 = 1 \end{cases} \tag{9}$$

where

$$\begin{aligned}\Delta =& M^2(P_a^1)^2(1-P_a^2)^2 \\ &+ M(P_a^2)^2(1-P_a^1)(1-M+MP_a^1)\end{aligned}$$

As an example, Fig. 2 shows the value of $Gexpression$ when $P_a^1 = 0.85$.

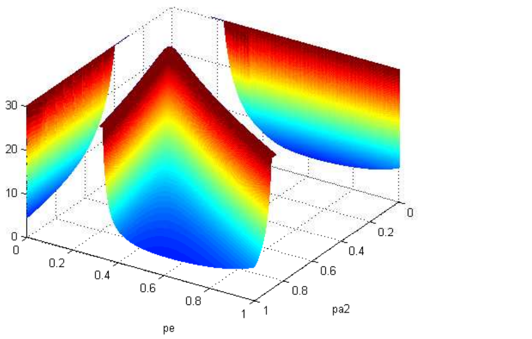


Fig. 2: the value of $Gexpression$ when $P_{a1} = 0.85$.

So in these conditions, the formula $E(T^{\mathcal{M}}) - E(T^{\mathcal{C}}) \ge 0$ holds. Therefore in these conditions conditional WARM strategy has better performance than the less WARM strategy. ■

Theorem 6. *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:*

1) *$GDenominator \le 0$;*
2) *or if $GDenominator > 0$, g is smaller than the minimum value of $Gexpression$.*

Proof: Firstly, we consider an extreme case that is performing WARM with a probability $P^{\mathcal{CO}}_{warm} \in [0, 1]$ just when cache misses do not occur during the pervious execution of encryption and not performing WARM when cache misses occur. This case is denoted as complementary WARM. In this case, when there is a cache miss in the previous, DELAY is needed. If the next encryption accesses any entry that is currently not in caches, performing WARM before hand introduces non extra overhead. However, if not performing WARM, as proved in Theorem 3, when some cache line size of lookup tables are not cached, not performing WARM introduces more extra time. Further more, complementary WARM strategy will introduce extra overhead when cache misses no occur. Therefore complementary WARM strategy has less performance than conditional WARM. On the other hand, the Markov state transition diagram of complementary WARM is in Fig. 1d. Using the same calculation process within Theorem 4, we can get that conditional WARM strategy has better performance than complementary WARM strategy.

Generally, arbitrary warm strategies can be divided into the combination of less WARM, more WARM and complementary WARM. We denote the general 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}. So we can get the equation:

$$W = x_1 W_L + x_2 W_M + x_3 W_{Co} \tag{10}$$

where x_1, x_2 and x_3 satisfy:

$$\begin{cases} x_2 P^{\mathcal{M}}_{warm} + x_3 P^{\mathcal{CO}}_{warm} = P_2 \\ x_1 P^{\mathcal{L}}_{warm} + x_2 = P_1 \\ x_1 + x_2 + x_3 = 1 \end{cases}$$

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

1) $GDenominator \leq 0$;
2) or if $GDenominator > 0$, g is smaller than the minimum value of $Gexpression$.

are 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_{a1} is larger than 0.994 from Equation 1. The range of P_{a2} is $[0.994, 1]$. For the Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM, T_{delay} is 2525.00 cycles; the minimum of $T_{warm}(T_{warm_{min}})$ is 124 cycles when accessing the lookup table from L1D caches; and the maximum of T_{warm} is 1781.60 cycles when all the lookup tables in RAM. Thus, $M = 0.0491$ and the rang of g is $[1.417, 20.363]$.

In this environment, when $GDenominator > 0$, $Gexpression$ is a monotonous increasing function of P^{1}_{a}. $Gexpression$ gets the minimum value when $P^{1}_{a} = 0.994$. Substitute the value into the equation 9 and get:

$$\begin{cases} P_{evict} = \dfrac{0.994M(1 - P^{2}_{a}) + \sqrt{\Delta}}{0.006(1 - 0.006M)} \\ P^{2}_{a} = \dfrac{0.07746 P_{evict}}{\sqrt{M(1 - P_{evict})}} - 0.006 P_{evict} \\ if \ P_{evict} > 1, P_{evict} = 1 \\ if \ P^{2}_{a} > 1, P^{2}_{a} = 1 \end{cases} \tag{11}$$

where

$$\begin{aligned} \Delta = & 0.988036M^2 - 1.976072M^2 P^{2}_{a} \\ & + (0.006M + 0.988M^2)(P^{2}_{a})^2 \end{aligned}$$

By computation we get that when $P_{evict} = 1$, $P^{2}_{a} = 1$ and $P^{1}_{a} = 0.994$, $Gexpression$ gets the minimum value. The minimum value is 167.7. Therefore, g is smaller than the minimum value of $Gexpression$ when $GDenominator > 0$. 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. ■

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 performing WARM with a probability $P_1^{'} \in [0, 1]$ greater than P_1, due to partial warm by the AES execution. However, based on Theorem 6, although $P_1^{'}$ is different, the general warm strategy still can be divided. So the conditions that make $E(T^{\mathcal{W}}) > E(T^{\mathcal{C}})$ holds is not changed. Therefore, conditional WARM strategy still has better performance.

Process scheduling and interruption occur during the AES execution. In this case, $end - start > T_W$, 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 $E(T_{warm,N}) = NT_{cl} + (32 - N) * T_c$, 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 II 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 WARMis not need. However from the overall execution we cannot distinguish whether the lookup tables are evicted. Thus performing WARM introduces some extra overhead.

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 $T_{NM} < end - start \leq T_W$. However, 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 the above, 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:

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

D. Different Key Lengths and Implementations

The WARM+DELAY scheme provides the optimal performance for various implementations of AES [64], [65] with different key length, if the conditions in Theorem 6 are satisfied. All the theorems above still stand, with the only difference be the value of P_a and M. 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 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 III lists the corresponding minimum P_a. M is the ratio of the minimum value of T_{warm} and T_{delay}. These two values 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 and M. Then we can find that, if the conditions in Theorem 6 are satisfied, the WARM+DELAY scheme provides the optimal performance.

For other table-lookup block ciphers, we have to determine P_a and M according to the algorithm, the implementation and the running platform. Once the conditions in Theorem 6 are satisfied , WARM+DELAY provides the optimal performance.

V. IMPLEMENTATION AND PERFORMANCE EVALUATION

[[[[[Linjq start here!!

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 [64]. 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.

]]]]]]]]]]]

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], [25].

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. 3a 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 [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 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. 3b.

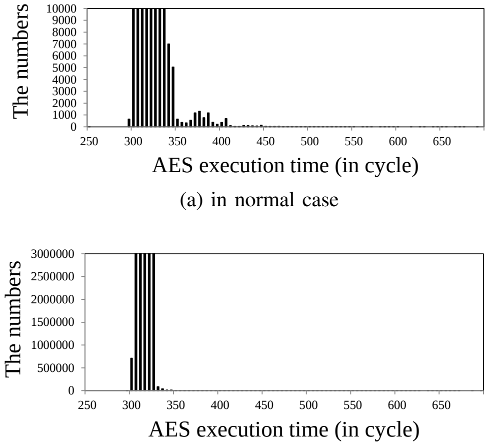
GETTIME. We adopt the instruction `RDTSCP` to implement GETTIME, to obtain the current time in high precision (clock cycles) with low cost. `RDTSCP` is a serializing call which prevents the CPU from reordering it. In the implementation, we need to perform the following operations to achieve the high 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,

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

N	1	2	3	5	10	15	20	25	30	31	32
$D_{nl}(N)$	2509.29	2524.92	2524.99	2525.00	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	1729.80	1781.60

TABLE III: the minimum value of the P_{delay} 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



(a) in normal case

(b) the best case

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

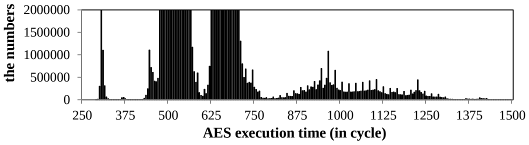


Fig. 4: The distribution of AES execution time only not warm one cache line.

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 $end - start < T_W$, instead of every time after WARM (see Line 7 in Algorithm 1).

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 number 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. 5. 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.

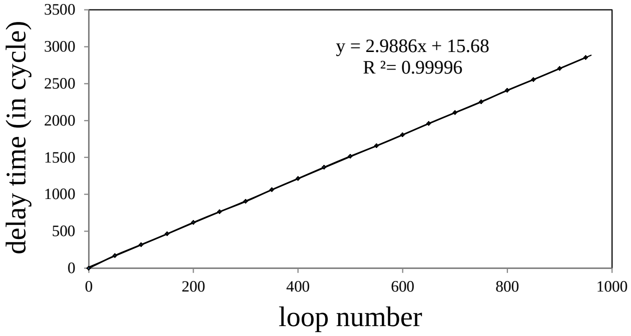


Fig. 5: AES execution performance in different scenarios.

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");
}
```

TNM. 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. 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. 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. 3. 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.

TW. Before measuring T_W, we flush both the data and

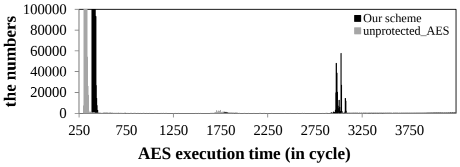


Fig. 6: The observed AES execution time with different plaintext.

instructions out of L1/2/3 caches, so T_W is the worst AES execution time and unrelated to the cache states. In actual measurement, we repeat the measurement for 10 times, and calculate the average value 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 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 the overhead of our scheme is acceptable in a real environment.

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 6 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 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 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.

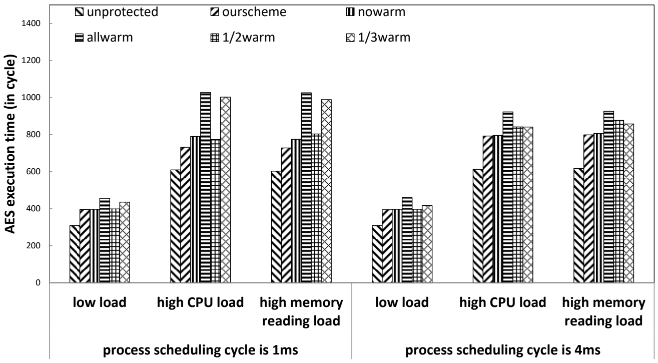


Fig. 7: AES execution performance in different scenarios.

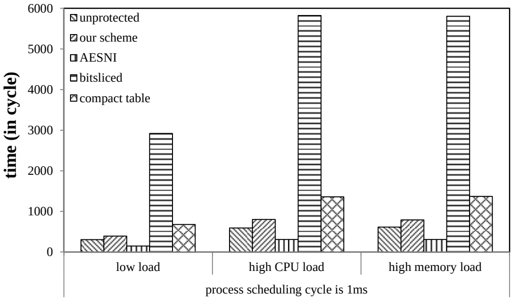


Fig. 8: Performance of different defense methods.

Figure 7 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 T_{NM}. From Table IV, 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, compact table implementation [64] and bitsliced AES implementation [67]. For each method, we we perform 2^{30} AES encryptions within random plaintexts. It is shown in Figure 8 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

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

Interval of OS scheduler	Low workload	High CPU workload	High mem workload
1ms	0.0028	0.0037	0.0041
4ms	0.0020	0.0026	0.0038

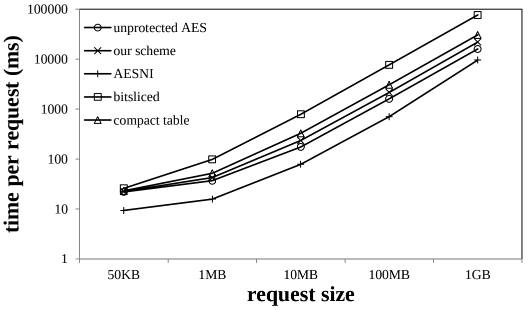


Fig. 9: Apache Benchmark result

server throughput.

The HTTPS throughput is shown in Figure 9. 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.

VI. DISCUSSIONS

A. Instruction Cache

Firstly, the implementation of block ciphers is not subject to timing side-channel attacks based on instruction caches [68], 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

Besides the timing attack, there are two other kinds of cache-based side-channel attacks: the trace-driven attack and the access-driven attack. However both of these attacks require the attacker with special abilities on the target system. 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], [69], while the access-driven attackers should have the ability to run a spy process on the target server accessing shared caches, to infer the cache status periodically [5], [8], [70], [71]. The attackers with such abilities 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 these attackers can observe the inner cache access pattern of the encryption/decryption execution. On the other hand, warm is to some degree effective for these attacks, provided that the cached lookup tables are not evicted by system activities or the local spy process.

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

achieve the optimal performance, by

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 outperforms other different integration strategies of WARM and DELAY, and (*c*) the implementation works without any privileged operations on the system.

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 optimal performance. Experimental results on the prototype system, confirm the security against remote cache timing side channels, and validate the performance optimization.

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 calculation, $0 \leq P_a^{1} \leq 1$, $0 \leq P_a^{2} \leq 1$, $0 \leq P_a^{2\mathcal{L}} \leq 1$, $0 \leq P_a^{2\mathcal{M}} \leq 1$, $0 \leq P_{warm}^{\mathcal{L}} \leq 1$, $0 \leq P_{warm}^{\mathcal{M}} \leq 1$ and $0 \leq P_{evict} \leq 1$.

A. The proof of Theorem 4

To prove the Theorem 4, we need to compare $E(T^{\mathcal{L}})$ with $E(T^{\mathcal{C}})$. So we get the Formula 7, and estimate if it is larger than 0. By combing Equation 4 and 5, we get Equation 12 and 13.

$$\begin{aligned} y(P_{warm}^{\mathcal{L}}) = & - P_a^{2}(P_a^{2\mathcal{L}} - P_{evict}(P_a^{2\mathcal{L}} + P_a^{1} - P_a^{3\mathcal{L}})) * P_{warm}^{\mathcal{L}} \\ & + P_{evict}P_a^{2\mathcal{L}}(1 - P_a^{1}) + P_a^{2\mathcal{L}}P_a^{2} \\ & - P_{evict}P_a^{2}(1 + P_a^{2\mathcal{L}} - P_a^{3\mathcal{L}}) \end{aligned} \tag{12}$$

$$E(T^{\mathcal{L}}) - E(T^{\mathcal{C}}) = \frac{y(P_{warm}^{\mathcal{L}})P_{evict}T_{delay}}{A_L * B_L} \tag{13}$$

where

$$\begin{aligned} A_L &= P_{evict}(1 + P_a^{2\mathcal{L}} - P_a^{3\mathcal{L}}) + (1 - P_{evict})P_a^{2\mathcal{L}}P_{warm}^{\mathcal{L}} \\ &\quad + (P_a^{3\mathcal{L}} - P_a^{1})P_{evict}P_{warm}^{\mathcal{L}} \\ B_L &= P_a^{2} + P_{evict}(1 - P_a^{1}) \end{aligned}$$

In the Markov state transition diagram,

$$P_a^{3\mathcal{L}} = P_{evict}^{3\mathcal{L}}P_a^{31\mathcal{L}} + (1 - P_{evict}^{3\mathcal{L}})P_a^{32\mathcal{L}}.$$

It can be seen from analysis that $P_a^{31\mathcal{L}} > P_a^{1}$ and $P_a^{32\mathcal{L}} > P_a^{1}$. So we can get $P_a^{3\mathcal{L}} > P_a^{1}$. In this way, the function $y(P_{warm}^{\mathcal{L}})$ is a monotony decrease function. When $P_{warm}^{\mathcal{L}} = 1$, it gets the minimum value. At this point, $P_a^{2\mathcal{L}} = P_a^{2}$ and $y(P_{warm}^{\mathcal{L}}) = 0$. Also A_L and B_L are larger than zero. Therefore $E(T^{\mathcal{L}}) - E(T^{\mathcal{C}}) \geq 0$. That means the extra time introduced by less WARM strategy is always larger than conditional WARM strategy.

B. The proof of Theorem 5

To prove the Theorem 4, we need to compare $E(T^{M})$ with $E(T^{C})$. So we get the Formula 8, and estimate if it is larger than 0.

First, we find that $P_{a2} = P_{a2}^{''}$. Then by combing Equation 4 and 6, we get Equation 14 and 15.

For the calculation, $0 \leq P_{a1} \leq 1$, $0 \leq P_{a2} \leq 1$, $0 \leq P_{a2}^{'} \leq 1$, $0 \leq P_{a2}^{''} \leq 1$, $0 \leq P_{Lwarm} \leq 1$, $0 \leq P_{Mwarm} \leq 1$ and $0 \leq P_{evict} \leq 1$.

$$\begin{aligned} y(P_{Mwarm}) = & A_M C_M P_{Mwarm}^2 + B_M C_M P_{Mwarm} \\ & - g D_M P_{Mwarm} \end{aligned} \tag{14}$$

$$E(T^{M}) - E(T^{C}) = \frac{y(P_{Mwarm})T_{warm}}{E_M * F_M} \tag{15}$$

where

$$\begin{aligned}
A_M &= Mg(1 - P_{a2}) \\
B_M &= MgP_{a2} + P_{evict}(1 - P_{a1}) \\
C_M &= \frac{P_{a2} + P_{evict}P_{a2} + P_{evict}(1 - P_{a1})}{P_{evict}(1 - P_{a1}} \\
D_M &= -P_{evict}P_{a1} + P_{evict}P_{a2} + P_{a2} \\
E_M &= (1 + P_{evict})(1 - (1 - P_{Mwarm})(1 - P_{a2})) \\
&\quad + P_{evict}(1 - P_{Mwarm})(1 - P_{a1}) \\
F_M &= P_{a2} + P_{evict}P_{a2} + P_{evict}(1 - P_{a1})
\end{aligned}$$

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

$$\begin{aligned}
&((P_{evict}P_{a2} + P_{a2} - P_{evict}P_{a1})(P_{evict}(1 - P_{a1}) - MP_{a2}) \\
&- MP_{evict}P_{a2})g \leq \\
&P_{evict}(1 - P_{a1})(P_{evict}P_{a2} + P_{a2} + P_{evict}(1 - P_{a1}))
\end{aligned} \tag{16}$$

We denote $(P_{evict}P_{a2} + P_{a2} - P_{evict}P_{a1})(P_{evict}(1 - P_{a1}) - MP_{a2}) - MP_{evict}P_{a2}$ as $GDenominator$.

1) If $GDenominator \leq 0$, then because the right of the inequality is greater than zero, the inequality holds.
   In this case, we regard $GDenominator$ as a function of P_{evict} and get:

$$\begin{aligned}
&f(P_{evict}) = (P_{a2} - P_{a1})(1 - P_{a1})P_{evict}^2 \\
&+ ((1 - M)P_{a2} - (1 - M)P_{a1}P_{a2} - MP_{a2}^2)P_{evict} \\
&- MP_{a2}^2
\end{aligned} \tag{17}$$

   It is an quadratic function and when $P_{evict} = 0$, $f(P_{evict}) < 0$. So (1) when $P_{a2} > P_{a1}$ and $P_{evict} < \min(1, P_{rootr})$, $f(P_{evict}) \leq 0$;(P_{rootr} is the greater root of equation $f(P_{evict}) = 0$.) (2) when $P_{a2} = P_{a1}$ and $P_{evict} < \min(1, \frac{M*P_{a1}}{1-M-P_{a1}})$, $f(P_{evict}) \leq 0$; ($1 - M - P_{a1} > 0$, if not,$f(P_{evict}) \leq 0$ holds.) (3) when $P_{a2} < P_{a1}$, the symmetry axis of $f(P_{evict})$ is $P_{evict} = P_{e\_axis}$, where P_{e_axis} is denoted in Equation 18.

$$P_{e\_axis} = -\frac{P_{a2}(1 - M - P_{a1} - MP_{a2} + MP_{a1})}{2(P_{a2} - P_{a1})(1 - P_{a1})} \tag{18}$$

   If $P_{e\_axis} \leq 0$, i.e. $P_{a2} \geq \frac{(1-M)(1-P_{a1})}{M}$, $f(P_{evict}) \leq 0$ holds. Otherwise, $P_{e\_axis} > 0$, i.e. $P_{a2} < \frac{(1-M)(1-P_{a1})}{M}$. In this case, if P_{rootl} exists, $P_{evict} < min(1, P_{rootl})$ and then $f(P_{evict}) \leq 0$. If P_{rootl} does not exist, $f(P_{evict}) \leq 0$ holds. (P_{rootl} is the smaller root of equation $f(P_{evict}) = 0$.)

2) If $GDenominator > 0$, then

$$g \leq \frac{P_{evict}(1 - P_{a1})(P_{a2} + P_{evict}P_{a2} + P_{evict}(1 - P_{a1}))}{GDenominator} \tag{19}$$

We denote $\frac{P_{evict}(1-P_{a1})(P_{a2}+P_{evict}P_{a2}+P_{evict}(1-P_{a1}))}{GDenominator}$ as $Gexpression$. First, we regard $Gexpression$ as the function of P_{a1} and calculate the derivative on P_{a1}, and get:

$$\begin{aligned}
\frac{d(Gexpression)}{d(P_{a1})} =& P_{evict}^2(MP_{a2} + P_{evict})(1 - P_{a1})^2 \\
&+ 2MP_{evict}(1 + P_{evict})P_{a2}^2(1 - P_{a1}) \\
&+ MP_{a2}^3(1 + P_{evict})^2
\end{aligned} \tag{20}$$

Therefore, $Gexpression$ is a monotony increase function of P_{a1}.

Second, we regard $Gexpression$ as the function of P_{a2} and calculate the derivative on P_{a2}, and get:

$$\begin{aligned}
\frac{d(Gexpression)}{d(P_{a2})} =& P_{evict}\bar{P}_{a1}\{M(1 + P_{evict})^2 P_{a2}^2 \\
&+ 2MP_{evict}\bar{P}_{a1}(1 + P_{evict})P_{a2} \\
&+ MP_{evict}^2\bar{P}_{a1}^2 \\
&- P_{evict}^2\bar{P}_{a1}(1 + P_{evict})\}
\end{aligned} \tag{21}$$

where

$$\bar{P}_{a1} = 1 - P_{a1}$$

Denote P_{a2r} as the greater root of equation $\frac{d(Gexpression)}{d(P_{a2})} = 0$. $M(1 + P_{evict})^2 > 0$ and when $P_{a2} = 0$, so $\frac{d(Gexpression)}{d(P_{a2})} < 0$, $P_{a2r} > 0$.

$$P_{a2r} = \frac{P_{evict}\sqrt{\frac{(1+P_{evict})\bar{P}_{a1}}{M}} - P_{evict}\bar{P}_{a1}}{1 + P_{evict}} \tag{22}$$

Therefore, if $P_{a2r} \geq 1$, then $\frac{d(Gexpression)}{d(P_{a2})} < 0$ and $Gexpression$ is a monotony decrease function of $P_{a2} \in [0, 1]$. And if $P_{a2r} < 1$, then $Gexpression$ first decreases then increase as the function of $P_{a2} \in [0, 1]$. Let $P_{a2r} = 1$, and we get:

$$\begin{aligned}
\bar{P}_{a1}P_{evict}^3 + (\bar{P}_{a1} - M(1 + \bar{P}_{a1})^2)P_{evict}^2 \\
- 2M(\bar{P}_{a1} + 1)P_{evict} - M = 0
\end{aligned} \tag{23}$$

It is a simple cubic equation on P_{evict}. We denote P_{e_r} as its greatest real root. If $P_{e\_r} < 1$, then, when $P_{evict} \geq P_{e\_r}$, $P_{a2r} \geq 1$. When $P_{evict} < P_{e\_r}$, $P_{a2r} < 1$. Otherwise, $P_{a2r} < 1$ holds. In general,if $P_{e\_r} < 1$, when $P_{evict} \geq P_{e\_r}$, $Gexpression$ is a monotony decrease function of $P_{a2} \in [0, 1]$. When $P_{evict} < P_{e\_r}$, $Gexpression$ first decreases then increase as the function of $P_{a2} \in [0, 1]$. And if not, $Gexpression$ first decreases then increase as the function of $P_{a2} \in [0, 1]$.

Third, we regard $Gexpression$ as the function of P_{evict} and calculate the derivative on P_{evict}, and get:

$$\begin{aligned}
\frac{d(Gexpression)}{d(P_{evict})} &= P_{a2}(\bar{P}_{a1} - M(\bar{P}_{a1} + P_{a2})^2)P_{evict}^2 \\
&- 2MP_{a2}^2(P_{a2} + \bar{P}_{a1})P_{evict} \\
&- MP_{a2}^3
\end{aligned} \tag{24}$$

If $P_{a2} = \sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1}$, $\frac{d(Gexpression)}{d(P_{evict})} < 0$. Function $Gexpression$ is a monotony decrease function on P_{evict}. Otherwise, $\frac{d(Gexpression)}{d(P_{evict})}$ is a simple cubic equation on P_{evict}. The symmetry axis of the function $\frac{d(Gexpression)}{d(P_{evict})}$ is:

$$P_{e\_axis} = \frac{MP_{a2}(\bar{P}_{a1} + P_{a2})}{\bar{P}_{a1} - M(\bar{P}_{a1} + P_{a2})^2} \tag{25}$$

(1) If $P_{a2} < \sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1}$, then $P_{e\_axis} > 0$.
When $P_{evict} = 0$, $\frac{d(Gexpression)}{d(P_{evict})} = -MP_{a2}^3 < 0$. When $P_{evict} = 1$,

$$\frac{d(Gexpression)}{d(P_{evict})} = P_{a2}(\bar{P}_{a1} - M(\bar{P}_{a1} + 2P_{a2})^2)$$

So when $P_{a2} < \frac{1}{2}(\sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1})$, $\frac{d(Gexpression)}{d(P_{evict})} > 0$. In this case, $Gexpression$ first decreases then increase as the function of $P_{evict} \in [0, 1]$. when $P_{a2} \geq \frac{1}{2}(\sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1})$, $\frac{d(Gexpression)}{d(P_{evict})} < 0$. In this case, $Gexpression$ is a a monotony decrease function of $P_{evict} \in [0, 1]$.
(2) If $P_{a2} > \sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1}$, then $P_{e\_axis} < 0$. When $P_{evict} = 0$, $\frac{d(Gexpression)}{d(P_{evict})} = -MP_{a2}^3 < 0$. So in this case, $Gexpression$ is a a monotony decrease function of $P_{evict} \in [0, 1]$.
In general, when $P_{a2} < \frac{1}{2}(\sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1})$, $Gexpression$ first decreases then increase as the function of $P_{evict} \in [0, 1]$; when $P_{a2} \geq \frac{1}{2}(\sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1})$, $Gexpression$ is a a monotony decrease function of $P_{evict} \in [0, 1]$.
In a conclusion, if $GDenominator > 0$, $Gexpression$ is a monotony increase function of P_{a1}; when $P_{evict} < \min(P_{e\_r}, 1)$, $Gexpression$ first decreases then increase as the function of $P_{a2} \in [0, 1]$; if $P_{e\_r} < 1$, when $P_{evict} \geq P_{e\_r}$, $Gexpression$ is a monotony decrease function of $P_{a2} \in [0, 1]$; when $P_{a2} < \frac{1}{2}(\sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1})$, $Gexpression$ first decreases then increase as the function of $P_{evict} \in [0, 1]$; when $P_{a2} \geq \frac{1}{2}(\sqrt{\frac{\bar{P}_{a1}}{M}} - \bar{P}_{a1})$, $Gexpression$ is a a monotony decrease function of $P_{evict} \in [0, 1]$.
Therefore, the minimum value of $Gexpression$ is get when P_{a1} achieves the minimum value, P_{a2} and P_{evict} achieve the maximum value.

From the above, we can make the following conclusions:

1) (1) when $P_{a2} > P_{a1}$ and $P_{evict} < \min(1, P_{rootr})$, (2) when $P_{a2} = P_{a1}$ and $P_{evict} < \min(1, \frac{M*P_{a1}}{1-M-P_{a1}})$ (if $1 - M - P_{a1} > 0$), (3) when $P_{a2} < P_{a1}$, $P_{a2} < \frac{(1-M)(1-P_{a1})}{M}$ and $P_{evict} < min(1, P_{rootl})$ (if P_{rootl} exists) or $P_{a2} \geq \frac{(1-M)(1-P_{a1})}{M}$, we have $GDenominator \leq 0$ and the Inequality 16 holds.
2) when conditions are opposite, i.e. $GDenominator > 0$, if g is smaller than the minimum value of $Gexpression$, the Inequality 16 holds.

At this point, $E(T^M) - E(T^C) \geq 0$ and conditional WARM strategy has better performance than the less WARM strategy.

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:

$$P_{delay}^{128} = 1 - \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 36), \tag{26}$$

$$P_{delay}^{192} = 1 - \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 44), \tag{27}$$

$$P_{delay}^{256} = 1 - \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 52), \tag{28}$$

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_{delay} for AES-128, AES-192 and AES-256 are as follows respectively:

$$P_{delay}^{128} = 1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 36), \tag{29}$$

$$P_{delay}^{192} = 1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 44), \tag{30}$$

$$\begin{aligned}P^{256}_{delay} =& 1- \\ &\frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4,16) \prod_{i=0}^{3} f(x_i,52),\end{aligned} \tag{31}$$

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_{delay} for AES-128, AES-192 and AES-256 are as follows respectively:

$$\begin{aligned}P^{128}_{delay} =& 1- \\ &\frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i,40),\end{aligned} \tag{32}$$

$$\begin{aligned}P^{192}_{delay} =& 1- \\ &\frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i,48),\end{aligned} \tag{33}$$

$$\begin{aligned}P^{256}_{delay} =& 1- \\ &\frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i,56),\end{aligned} \tag{34}$$

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_{delay} for AES-128, AES-192 and AES-256 are as follows respectively:

$$P^{128}_{delay} = 1-(1-\frac{NC}{2048})^{16*10}, \tag{35}$$

$$P^{192}_{delay} = 1-(1-\frac{NC}{2048})^{16*12}, \tag{36}$$

$$P^{256}_{delay} = 1-(1-\frac{NC}{2048})^{16*14}. \tag{37}$$

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] 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.

[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 *Cambridge Security Workshop on Fast Software Encryption (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] 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, "Partitioned cache architecture as a side-channel defence mechanism." *IACR Cryptology ePrint archive*, vol. 2005, no. 2005, p. 280, 2005.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

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

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

[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] "bitcoin-core/ctaes: Simple constant-time AES implementation," https://github.com/bitcoin-core/ctaes/.

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

[69] 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.

[70] 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.

[71] 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.

John Doe Biography text here.

Jane Doe Biography text here.