Eliminating Remote Cache Timing Side Channels with Optimal Performance

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 common countermeasures against remote cache-based timing side channels, by reading constant data into caches before cryptographic functions and inserting padding instructions after these functions, respectively. These countermeasures destroy the relationship between the execution time and the cache misses/hits which are determined by the secret key, but introduce extra operations. However, the extra operations are sometimes unnecessary, and then bring great performance overheads. In this paper, we integrate these two countermeasure into a scheme that adopts WARM and DELAY alternatively and complementarily. Before each encryption/decryption, the constant data are warmed into caches if the last cryptographic function invocation is not finished at the expected speed. Because the effect of WARM may be broken by system activities, DELAY plays as another line of defense: after each encryption/decryption, padding instructions are inserted if the execution time may be exploited to extract the secret key. Moreover, when DELAY is necessary, WARM is performed as a part of inserted padding instructions, to further optimize the performance. The proposed WARM+DELAY scheme effectively eliminates remote cache timing side channels, and is applicable to different block ciphers and computing platforms. More importantly, we apply this scheme to AES, and analyze that it achieves *the optimal performance with the least extra operations*. We implement it on Linux with Intel Core CPUs to protect AES. Experimental results of the prototype 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.**

***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 any special devices. In particular, remote timing side channels allow an attacker without any system or physical privilege on the computer, to disclose the keys only by repeatedly invoking the encryption/decryption functions and measuring the execution time [3], [4], [7], [16]–[18].

Remote timing side channels passively exploiting the time difference of data cache misses and hits, called *remote cache timing side channels* in this paper, 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 similar basic operations. In the remainder, we emphasize encryption only, but the designs and conclusions are 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 necessary encryption/decryption functions, while other active cache-based side-channel attacks involve operations by a 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 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 mechanisms prevent cache-based timing attacks at different phases, but introduce extra operations and degrade the performance greatly. As there are different strategies to apply WARM and DELAY in the implementations of block ciphers, we attempt to integrate them into a scheme achieving

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

the optimal performance while effectively eliminating remote cache timing side channels. In particular, the following principles are enforced in the integrated scheme:

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 then information on the secret key might be leaked, the execution time is delayed as 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.

Different from existing cache side channel defenses, the proposed WARM+DELAY scheme eliminates remote cache timing side channels while achieves the optimal performance (that is, *no unnecessary lookup table read operations or inserted padding instructions*). In particular, we apply this scheme to AES, and analyze the situations that it produces the optimal performance. It is found that, in commodity computer systems, the proposed scheme achieves the optimal performance with different key sizes (128, 192, 256 bits) and different implementations (2KB, 4KB, 4.25KB, and 5KB lookup tables). For example, the protected AES-128 implementation with 2KB lookup tables [27] achieves the optimal performance, when (*a*) the probability that some entries of lookup tables are evicted from caches due to system activities during encryption, denoted as P_{evict}, is less than 0.2, and (*b*) the ratio of the delayed time of padding instructions to the time of reading constant tables from either caches or RAM, denoted as k, is less than 57.4. These conditions hold in commodity computer systems, which is confirmed by experimental results of the prototype. In fact, our protection scheme produces the optimal performance for various AES implementations (the lookup tables in size of 2K, 4KB, 4.25KB and 5KB) of different key sizes (128, 192 and 256 bits), when $P_{evict} < 0.2$ and $k < 42.41$. These conditions also hold in commodity systems.

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

We implement the WARM+DELAY scheme with AES-128. Experiment results demonstrate that, the execution time measured by attackers does not reflect the cache misses/hits during encryption. We confirm the optimization in different scenarios, by comparing it with different integration strategies of WARM and DELAY.

To the best of our knowledge, this work is the first protection against cache timing side channels with quantitative performance analysis. Our contributions are as follows:

- We integrate WARM and DELAY into a scheme that achieves the optimal performance with the least extra operations while effectively eliminates remote cache timing side channels. We apply the scheme to AES and analyze the optimal way to integrate WARM and DELAY with security. The optimal performance is also validated by experiments.
- The two basic elements are integrated in the way that the WARM+DELAY scheme is widely applicable. WARM and DELAY are independent of algorithms and implementations, work on various computer systems. These features are kept in the integrated scheme, which is applicable to various implementations with lookup tables of different block ciphers, and does not require any privileged operation on the computer system.

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

II. BACKGROUND AND RELATED WORKS

A. AES and Block Cipher Implementation

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

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

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

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

There are similar implementations of other block ciphers, consisting of table lookup operations and basic computations while no data-dependent branch in the execution path. For

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

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 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. Data in caches are loaded and evicted in cache lines. Each cache line (64 bytes) usually contains multiple lookup table entries, and accessing the entries in a same cache line is called an internal collision. The internal collision attack works for DES, 3DES and the first round of AES [9], [16], and is extended to the second and last rounds of AES [19]. This attack exploits the relationship between the encryption time and the number of cache hits. Let's take the first round of AES 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 access the lookup table entries in a same cache line, it results in shorter execution time. Then, the difference of some key bits is disclosed as the values (i.e., $p_{(i+4j)\%16} \oplus p_{(i+4k)\%16}$, $0 \leq j < k \leq 3$) counted most often, which is used to extract the secret key.

Bernstein's attack. It is based on the case that the whole AES execution time is correlated with the time for accessing the lookup tables, which depends on the lookup table index [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 infers the key of the victim server, with the obtained lookup table index and known plaintexts.

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

[[[[[–Linjq restart here.

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.

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 lookup tables.

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. We expect that 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 xxxx) are out of the scope of this paper. We attempt to eliminate remote cache timing side channels, with the following properties:

- It is applicable to various block ciphers and transparent to implementations. The scheme is algorithm-independent and works well for 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.

B. The WARM+DELAY Scheme

The scheme integrates 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. We attempt to optimize the performance while ensure the security. Therefore, we focus on the situation where large amounts of data are encrypted and the speed of encryption matters.

A naive scheme is to perform both WARM and DELAY every time the protected cryptographic function is invoked. But it introduces too many extra operations and the performance is seriously degraded, although the security is ensured. So we perform DELAY as a backup line of defense, or 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). Next, when we consider that the cryptographic function is invoked continuously, 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. Finally, we find that, the conditions to perform WARM and DELAY are similar. That is, when the execution time is greater than the expected time with all tables cached, DELAY is necessary to eliminate the risk

Algorithm 1 The WARM+DELAY scheme

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

and WARM is also necessary for the next encryption because probably some table entry is uncached. So, in our scheme, we perform WARM 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 other system activities. Given a block cipher, T_{NM} and T_W are constant on a certain computing platform.

In addition to encryption, 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 address of lookup tables is needed in WARM(). These operations are supported in commodity computer systems, 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 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 of CRYPT() 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 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.

In summary, 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, which are unknown to attackers.

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

In the following, we further explain that our scheme successfully prevent typical remote cache timing side-channel attacks. To perform the internal collision attacks [3], [19], the attackers need a collection of plaintexts with short (long) measured time due to cache hits (misses). However, protected by the proposed WARM+DELAY scheme, any plaintext is processed in the short time (i.e., T_{NM}) due to cache warm, while it is done in the long time (i.e., T_W or longer) due to system activities after DELAY. Therefore, the distribution of the collections depends on the 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. Analysis of Optimal Performance

A. Overview

In this section, we prove that the WARM+DELAY achieves the optimal performance with the least extra operations, by applying it to AES and analyzing the situations that it produces the optimal performance. In the scheme, the extra operations are introduced by WARM and DELAY, so the main point is to optimize the use of WARM and DELAY. In practice, 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().

We examine these principles with our proposed 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 [63]. 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 our scheme is optimal. That is, performing WARM() when the execution time of encryption lies in (T_{NM}, ∞) is the optimal in commodity systems.

Last, we extend the conclusion 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 proposed WARM+DELAY scheme achieves the optimal performance 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. 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). When the measured time is longer than T_W, encryption is disturbed by 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 the 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 $T_W - T_{NM}$). The proposed scheme 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 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 [63], 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.

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

Theorem 3. *If* $NT_{nc} < E(T_{delay})$ *for any* $N > 0$*, then loading all entries of the lookup table into caches in the* WARM() *operation provides better performance than any part of entries, where* $E(T_{delay}) = (1-(1-\frac{NC}{L})^{160})(T_W - T_{NM})$.

Proof: It takes T_{cl} to load data into a cache line in the WARM() operation; on the other hand, if some entry is uncached, it may trigger an extra DELAY() operation. Not loading N ($N > 0$) cache lines of table entries saves the execution time of WARM(), while the potential cost is the extra execution of DELAY().

Firstly, the cost of loading data into N cache lines in the WARM() operation is NT_{cl}.

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

]]]]]]]]]]]]]]

$$\begin{aligned} E(T_{delay}) &= P_{delay} * (T_W - T_{NM}) \\ &= (1-(1-\frac{NC}{2048})^{160})(T_W - T_{NM}) \end{aligned} \quad (1)$$

We assume that the size of a cache line is C Bytes, hence, the lookup table needs L/C cache lines.

We assume that each round of AES is independent in terms of cache access, so the inputs of each round are random. L_{2k} is accessed 16 times in each of AES rounds besides the last round. Meanwhile, the access of cache lines in L_{2k} is uniformly distributed based on the structure of the lookup table in the memory. Therefore, the probability that a certain cache line of L_{2k} is not accessed in all rounds is $P_1 = (1 - \frac{C}{L})^{160}$. Hence, the probability that N certain cache lines are not accessed in one execution is $P_N = (1 - \frac{NC}{L})^{160}$. Therefore, the probability of *invoking* DELAY() *while* N *cache lines of lookup tables are not in the cache* is shown in Equation 2.

$$P_{delay} = 1 - (1 - \frac{NC}{L})^{160} \quad (2)$$

The worst execution time is T_W, and the shortest execution time is T_{NM}, then the expected overhead (i.e. the cost) of not loading N cache line of lookup tables (and invoking DELAY()) is

$$\begin{aligned} E(T_{delay}) &= P_{delay} * (T_W - T_{NM}) \\ &= (1-(1-\frac{NC}{L})^{160})(T_W - T_{NM}) \end{aligned} \quad (3)$$

then the benefit of not loading N cache lines is $\Delta T_{warm} = N * t_{nc}$.

Therefore, we have: (1) If $\forall N > 0, \Delta T_{warm} < E(T_{delay})$, warming all cache lines is always better.

(2) if there $\exists N$ that $\Delta T_{warm} > E(T_{delay})$, not warming N cache lines provides better performance. (3) If $\exists N > 0, \Delta T_{warm} = E(T_{delay})$, warming or not warming N cache lines are equivalent. ■

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

Corollary 1. *in commodity computer systems, AES-128 with 2K table, Loading all the AES lookup tables into the cache is the best* WARM()*operation.*

for the Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM. In our experiment hardware, the size of a cache line C is 64 bytes. The time T_W is 2834 CPU cycles, while the shortest AES execution time T_{NM} is 309 cycles. The average time t_{nc} for loading one cache line of lookup table is 55.68 cycles. From Table I, we can see that the cost of not loading N ($N > 0$) cache lines of lookup tables is much higher than the benefit. Therefore, loading all the lookup tables into the cache provides the best performance in WARM().

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

Theorem 4. t_{delay} *is the time to perform* DELAY()*while* t_{warm} *is the time to perform* WARM(). *Let* $k = t_{delay}/t_{warm}$, *so in the condition that* $1 \le k \le k_0$(*k_0 is in Equation 7), performing* WARM() *when cache-miss occurs during the previous execution results the minimum extra time.*

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

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

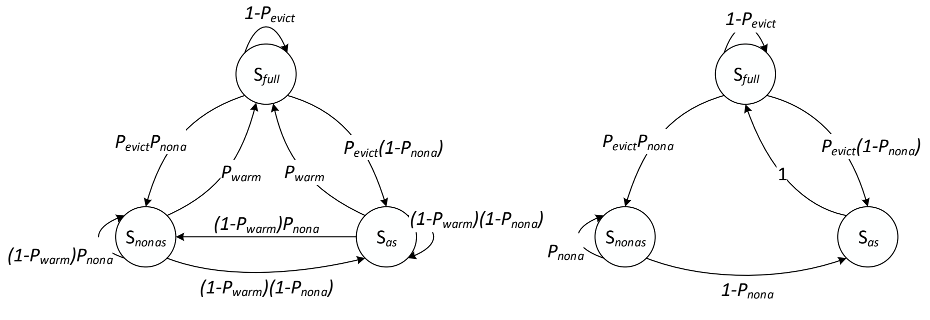
To prove the second case, we categorize the system state into three cases: (1) all the lookup tables are in the cache (S_{full}), (2) the lookup entries not in the cache are not accessed (S_{nonas}), and (3) the lookup entries not in the cache are accessed (S_{as}). We use the Markov model to analyse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

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

$$\Pi^c_{full} = \frac{1-P_{nona}}{1-P_{nona}+P_{evict}}, \Pi^c_{nonas} = \frac{P_{nona}P_{evict}}{1-P_{nona}+P_{evict}}, \Pi^c_{as} = \frac{P_{evict}(1-P_{nona})}{1-P_{nona}+P_{evict}} \quad (5)$$

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

N	1	2	3	5	10	15	20	25	30	32
$E(T_{delay})$	2509.29	2524.92	2524.99	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00
ΔT_{warm}	55.68	111.35	167.03	278.38	556.75	835.13	1113.5s	1391.88	1670.25	1781.60



(a) probabilistic warmup (b) conditional warmup

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

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

$$E(T^p) - E(T^c) = \Pi_{as}^p * t_{delay} - \Pi_{as}^c * t_{delay} + (1 - \Pi_{as}^p) * P_{warm} * t_{warm} \quad (6)$$

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

$$k_0 = \frac{(P_{delay} + P_{evict}) * (1 + P_{evict} - P_{evict} * P_{delay})}{P_{evict} * P_{delay} * (1 - P_{delay})} \quad (7)$$

∎

Corollary 2. *Performing* WARM() *when cache-miss occurs during the previous execution results the minimum extra time, in commodity computer systems.*

for the Lenovo ThinkCentre M8400t PC with an Intel Core i7-2600 CPU and 2GB RAM. From Equation 7, if we regard k_0 as the function of P_{evict}, we can get that k_0 is a monotony decrease function when $P_{delay} \geq 0.5$. That is when $P_{evict} = 1$, k_0 gets the minimum value. Also if we regard k_0 as the function of P_{delay}, we can get that k_0 is a monotony increase function when $P_{delay} \geq 0.5$ for arbitrary P_{evict}.

From Equation 2, we get $P_{delay} \geq 0.994$. So the minimum value $k_0 = 336.3$ when $P_{evict} = 1$.(The range of k_0 is listed in Table II when $P_{delay} = 0.994$) When P_{delay} gets larger, k_0 gets larger. In the environment described in Section IV, the maximum of t_{delay} is 2525.00 cycles, the minimum of t_{warm} is 124 cycles (accessing the lookup tables from caches). The maximum of t_{warm} is 1781.60 when all the lookup tables in RAM. Therefore, the range of k is: $1 < k < 20.36$ satisfying the condition of $E(T^p) > E(T^c)$. Therefore, the conditional WARM() provides better performance than the probabilistic WARM(), that is performing WARM() when cache-miss occurs during the previous execution results the minimum extra time

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

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

Process scheduling and interruption occur during the AES execution. In this case, $end - start > T_W$, 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 $E(T_{warm,N}) = N * t_{nc} + (32 - N) * t_c$, where t_c and t_{nc} denote the time for accessing one cache line from caches and RAM, respectively. If we do not perform WARM(), the next round of AES may need to access part(s) of lookup tables from RAM instead of caches, which will trigger DELAY(). The expected overhead, denoted as $E(T_{delay})$, is shown in Equation 3. Table III shows that performing WARM()achieves better performance.

Periodical schedule function is called without scheduling during AES execution. As no other process invoked, the lookup tables are not evicted. Therefore, WARM() is unnecessary although $T_{NM} < end - start \leq T_W$. 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 Theorems 3 and 4, we prove that performing WARM() when not all lookup tables are in caches, results the smallest extra time. However, in our scheme, we perform the WARM()

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

TABLE II: The values of k_0 corresponding to P_{evict}. ($P_{delay} = 0.994$)

P_{evict}	0	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
k_0	∞	3502.1	1835.4	1002.2	724.5	585.7	502.5	447.1	407.5	377.8	354.8	336.3

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

N	1	2	3	5	10	15	20	25	30	32
$E(T_{delay})$	2509.29	2524.92	2524.99	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00	2525.00
$E(T_{warm})$	175.80	227.60	279.4	383.00	642.00	901.00	1160.00	1419.00	1678.00	1781.60

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

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

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

D. Different Key Lengths and Implementations

The WARM+DELAY scheme provides the optimized performance for various implementations of AES [63], [64] with different key length, once $1 \leq k \leq k_0$. All the proofs above stands, with the only difference be the value of P_{delay} (detailed in Appendix B). P_{delay} depends on the size of lookup tables and the number of iterated rounds. For mbed TLS-1.3.10 [64] and OpenSSL-0.9.7i [63], the size of lookup tables are 4.25KB and 5KB. For OpenSSL-1.0.2c [63], the size of lookup tables is 4KB or 2KB. The number of rounds is 10, 12 and 14 for AES-128, AES-192 and AES-256,respectively. Table IV lists the corresponding minimum P_{delay}.

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

V. Implementation and Performance Evaluation

[[[[[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

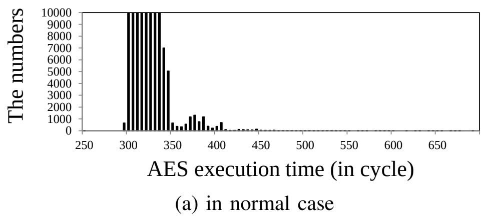
We apply the WARM+DELAY scheme to AES-128 implemented with a 2K lookup table, which is directly borrowed from OpenSSL-1.0.2c [63]. As we attempt to optimize performance while eliminate remote cache timing side channels, efficient GETTIME(), WARM() and DELAY() are finished; and the constant parameters (T_{NM} and T_W) are determined properly. Next, we show the implementation details about the WARM+DELAY scheme.

AES lookup table in caches. As mentioned above, we use the AES-128 implementation of OpenSSL-1.0.2c. However, even when all lookup tables are in the L1 cache, the execution time of encryption still has variations and these variations could be exploited to launched side-channel attackers [4], [25].]]]]]]]]]]] There are two reasons. One is the cache bank conflict. Cache bank is designed to improve the processor performance. Intel introduced a cache design consisting of multiple banks. Each of the banks serves part of the cache line specified by the offset in the cache line. The banks can operate independently and serve requests concurrently. But each cache bank can only serve one request at a time. So, multiple accesses to the same cache bank are slower than to different banks. For the cache bank conflict, we first disable the Hyper-Threading of the system to ensure the protected process itself not to produce concurrent access to the L1 cache. All the following experiments, Hyper-Threading is disabled.

Figure 2a shows the distribution of the execution time for 2^{30} random plaintexts. 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. We avoid the read-write conflict by exploiting the stack switch technique [65]. 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 Figure 2b.

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



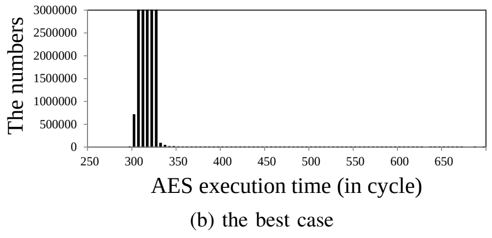


Fig. 2: The distribution of AES encryption time with 2KB lookup table in full warm condition.

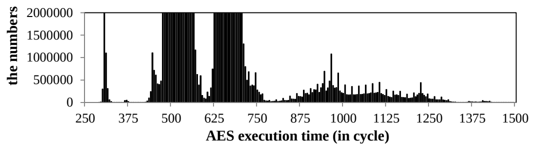


Fig. 3: The distribution of AES execution time only not warm one cache line.

constant.

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

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`.

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 2000 and the step is 50), as shown in Figure . 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 $t_{delay} = 2.995n + 12.886$, and the coefficient of determination is 0.999993. The precision is 3 cycles, much smaller than the noise of remote environments, so it cannot be exploited. Implementation of DELAY() is provided in Listing 1. If the input of DELAY() is less than zero, it simply returns. ????

¡¡¡¡¡¡¡ HEAD 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). ======= ¿¿¿¿¿¿¿ 2aebd8623132aa00e09dbc91ca48a734ff8debcf

Listing 1: The implementation of DELAY().

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

T_{NM}. T_{NM} is larger than the minimum AES execution time (no cache miss occurs), which avoids the unnecessary WARM() and DELAY() operations; and less than the AES execution that only one cache miss occurs. The average minimum AES execution time is measured by average 2^{30} AES execution time with the lookup tables all in L1D cache. In our environment it is 331 cycles. Fig. 3 shows the distribution of AES execution time for 2^{30} plaintexts while all lookup tables except one block of 64 bytes (i.e., one cache line) are loaded in L1D cache. Note that, this uncached entry may be unnecessary in an execution of AES encryption. In Fig. 3, the little data around 375 cycles are due to read-write conflict in the 2KB stack, and the data after 415 are caused by the cache miss. **(why rw conflict in stack)?????** We use the stack switch technique to eliminate the read-write conflict, at the same time it makes the AES execution time much more concentrated as shown in Fig. 2. This helps the determination of T_{NM} can be more accurate and reasonable. Also we should choose the value as large as possible to avoid the influence of the fluctuation around T_{NM} that might occur. Finally, we choose 355 cycles as T_{NM}. This value is chosen to ensure that all tables are in L1 caches and no fluctuation occurs around it. So no useful observation will be obtained by attackers, and no useful variations will be magnified.

T_W. before measure T_W, we flush both the data and instructions out of L1/2/3 caches, so T_W is the worst AES execution time and unrelated to the cache states. T_W is 2834 CPU cycles in this case. how to choose 2834? the average? the max? the min? or, from a figure?

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

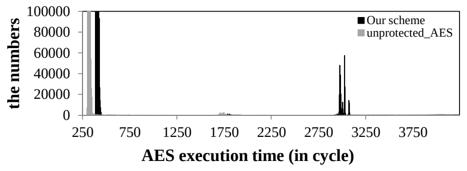


Fig. 4: The observed AES execution time with different plaintext.

application services. We find that in a real environment, the overhead of our scheme is acceptable.

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 4 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.

Figure 5 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 V, we find P_{evict} is less than 0.005 always, in which case the WARM+DELAY scheme is the optimal as proved in Section IV.

Performance of different defense methods. [[if possible, also in different scenarios as Fig. 6; more comprehensive]] Furthermore, we evaluate the performance of our scheme with different defense methods: AESNI, bitsliced AES implementation, compact table implementation **[[citation here, where the code from —-]]**. For each method, we we perform 2^{30} AES encryptions within random plaintexts. It is shown in Figure 6 that AESNI, the hardware implementation, has the best performance. The WARM+DELAY scheme has the best performance among all the software implementations.

TABLE V: 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

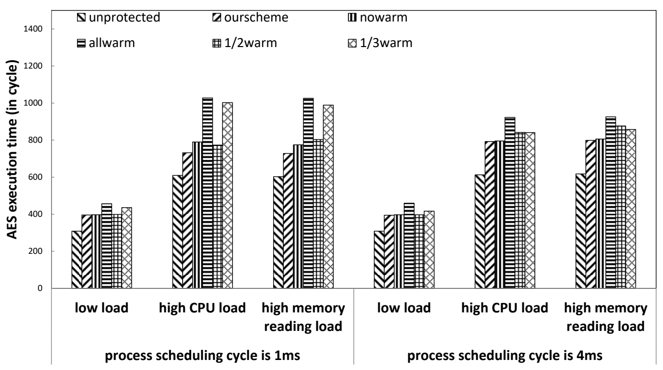


Fig. 5: AES execution performance in different scenarios.

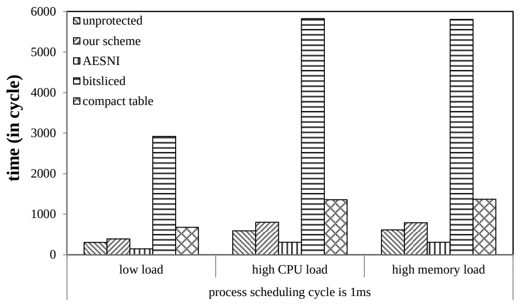


Fig. 6: Performance of different defense methods.

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

The HTTPS throughput is shown in Figure 7. When the unprotected AES is used, the throughput is XXX requests per second. When using our scheme, the throughput is xxxx requests per second. It is clearly that WARM+DELAY scheme has a low influence on the performance of TLS protocol. Furthermore, we compare the throughput using different defense methods involved in TLS protocol. Our defense scheme has the largest throughput among these methods.

VI. DISCUSSIONS

A. Instruction Cache

Firstly, the implementation of block ciphers is not subject to timing side-channel attacks based on instruction caches [66], because there is generally no branch in the execution path. The status of instruction caches affect the execution time, but is not related to the data (i.e., keys and plaintexts/ciphertexts). The instructions of block ciphers may be evicted from the caches

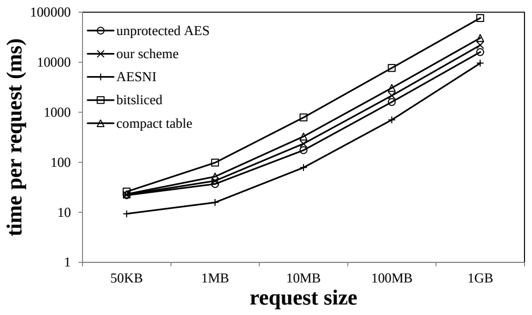


Fig. 7: Apache Benchmark result

due to system activities, which causes instruction cache misses and increases the execution time.

As the effect of instruction caches on the execution time is almost indistinguishable from that of data caches, we 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

Beside the timing attack, there are two other kinds of cache-based side-channel attacks: the trace-driven attack and the access-driven attack. Firstly, both of these attacks require special privileges on the target system. During the execution of encryption/decryption, the trace-driven attackers monitor the variations of electromagnetic fields or power to capture the profile of cache activities and deduce cache hits and misses [2], [27], [67], while the access-driven attackers run a spy process on the target server accessing shared caches, to infer the cache status periodically [5], [8], [68], [69]. The attackers with such privileges are not considered in our scheme, and will be considered in our future work.

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

will be our future work.

VII. Conclusion

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

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

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 THE RANGE OF k

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

By combing Equation 4, 5 and 6, we get Equation 8 and 9. We need to determine the range of k, which makes $E(T^p) - E(T^c) > 0$ (i.e., $y(P_{warm}) > 0$).

$$\begin{aligned} y(P_{warm}) = (P_{delay} + P_{evict}) * P_{warm}^2 \\ + P_{evict} * (P_{delay} + P_{evict})(1 - P_{delay}) * P_{warm} \\ - k * P_{delay} * P_{evict} * P_{warm} \\ + k * P_{delay}^2 * P_{evict} \ , \end{aligned} \tag{8}$$

$$E(T^p) - E(T^c) = \frac{y(P_{warm})}{(P_{warm} + P_{evict})(P_{delay} + P_{evict})} \tag{9}$$

The symmetry axis of $y(P_{warm})$ is $P_{warm} = P_a$, where P_a is denoted in Equation 10.

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

Equation 11 holds when $0.8 \leq P_{delay} \leq 1$, $0 \leq P_{evict} \leq 1$, which means $P_a \geq 0$.

$$\begin{aligned} k > 1 > & \frac{P_{evict}^2 + P_{evict}P_{delay} - P_{evict}P_{delay}^2 - P_{evict}^2 P_{delay}}{P_{evict}P_{delay}} \\ & = 1 + \frac{P_{evict}}{P_{delay}} - P_{delay} - P_{evict} \ . \end{aligned} \tag{11}$$

1) If $P_a > 1$ (i.e., Equation 12 holds), the minimum value of $y(P_{warm})$ is $y(1)$. When $k\epsilon(k_1, k_0]$, $y(P_{warm}) \geq 0$ holds.

$$k_1 = \frac{(P_{evict} + P_{delay})(2 + P_{evict} - P_{delay}P_{evict})}{P_{evict}P_{delay}} \ . \tag{12}$$

$$k_0 = \frac{(1 + P_{evict} - P_{evict}P_{delay})(P_{evict} + P_{delay})}{(1 - P_{delay})P_{evict}P_{delay}} \ . \tag{13}$$

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

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

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

$$\begin{aligned} f(1) = & \frac{1}{4(P_{evict} + P_{delay})}\{-(1 - P_{delay})^2 P_{evict}^4 \\ & + 2P_{evict}^3 P_{delay}^2 + 4P_{evict}P_{delay}^3 \\ & + P_{evict}^2 P_{delay}^2(4 - P_{delay}^2 - 2P_{evict}P_{delay})\}. \end{aligned} \tag{15}$$

$$\begin{aligned} f(k_1) = & (P_{evict} + P_{delay})\{2P_{delay} - 1 \\ & + P_{evict}P_{delay}(1 - P_{delay})\}. \end{aligned} \tag{16}$$

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

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

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

$$\begin{aligned} & P_{delay}^{128} = 1 - \\ & \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 36), \end{aligned} \tag{17}$$

$$\begin{aligned} & P_{delay}^{192} = 1 - \\ & \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 44), \end{aligned} \tag{18}$$

$$\begin{aligned} & P_{delay}^{256} = 1 - \\ & \frac{1}{\binom{4352/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4) \prod_{i=0}^{3} g(x_i, 52), \end{aligned} \tag{19}$$

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:

$$P_{delay}^{128} = 1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 36), \tag{20}$$

$$P_{delay}^{192} = 1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 44), \tag{21}$$

$$P_{delay}^{256} = 1 - \frac{1}{\binom{5120/C}{N}} \sum_{x_4=x_{4b}}^{x_{4e}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} f(x_4, 16) \prod_{i=0}^{3} f(x_i, 52), \tag{22}$$

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:

$$P_{delay}^{128} = 1 - \frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i, 40), \tag{23}$$

$$P_{delay}^{192} = 1 - \frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i, 48), \tag{24}$$

$$P_{delay}^{256} = 1 - \frac{1}{\binom{4096/C}{N}} \sum_{x_0=x_{0b}}^{x_{0e}} \sum_{x_1=x_{1b}}^{x_{1e}} \sum_{x_2=x_{2b}}^{x_{2e}} \prod_{i=0}^{3} f(x_i, 56), \tag{25}$$

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:

$$P_{delay}^{128} = 1 - (1 - \frac{NC}{2048})^{16*10}, \tag{26}$$

$$P_{delay}^{192} = 1 - (1 - \frac{NC}{2048})^{16*12}, \tag{27}$$

$$P_{delay}^{256} = 1 - (1 - \frac{NC}{2048})^{16*14}. \tag{28}$$

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] "OpenSSL:Cryptography and SSL/TLS Toolkit," https://www.openssl.org/.
[64] "SSL Library mbed TLS / PolarSSL," https://tls.mbed.org/.
[65] 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.
[66] O. Acıiçmez, "Yet another microarchitectural attack::exploiting I-cache," in *Computer security architecture, the 2007 ACM workshop on*. ACM, 2007, pp. 11–18.
[67] 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.
[68] 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.
[69] 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.