Eliminating Remote Cache Timing Side Channels with Optimized Performance

No Author Given

No Institute Given

Abstract. Keywords: Timing side channels, cache, defense, optimized performance

1 Introduction

Side channels which not use the algorithm itself but use the vulnerability of the implementation is a serious threat to the cryptosystem. Power, electromagnetic and timing attacks are common used side channel attacks. In side channel attacks, the information obtained from one or more side channels is used to reveal the key of a cryptosystem. Timing side channel attacks are the easiest to be carried out because the time is measured simply and they don't require specialized equipment.

In this paper, we focus on a type of timing side channels called cache timing side channel which speculates the key of the cryptosystem taking advantage of the information leaks through the time variation due to the cache/RAM access during the whole execution process. In many symmetric cipher algorithms, there are many table lookup operations about the key. When accessing these data, the speed of accessing memory is much lower than cache, and the time variation can leak some information about the key which can be used to obtain the key.

Remote side channel attacks are those can be carried out remotely and don't require to contact with the cryptosystem or nearby. Also, the adversaries don't need the special permissions in the target cryptosystem. Compared with other side channels, the requirement of remote cache timing side channel is low but the influence is significant. The cache attacks were first described by D. Page in [14] and implemented in [29, 30] on attacking DES and Triple-DES. The first identified remote cache timing attack was implement by Bernstein in [5]. Afterwards, many remote cache timing attacks or can be used remotely were presented like [21, 22, 25, 8, 31, 34, 4, 7, 24, 22, 31, 3, 12, 6, 21].

Because of the widely existence and the serious threat, many defense methods are proposed to eliminate the cache side channel. D. Page first presented a serial of defense methods in [26] such as not using cache, modifying time accuracy, performing cache warm, adding time delay, adding redundant instruction, performing order skewing and physical shielding. Some of them are out of style but others enlighten the followers. In 2005, a hardware based defense method was presented by D. Page which allowed the cache to be configured dynamically to match the need of a given process [15]. Ernie Brickell [10] proposed a strategy based on compact tables while frequently randomizing tables and preloading of relevant cache lines. [23] uses adding random permutations to the lookup tables to protect the cryptographic algorithm. In [32, 33, 20, 18, 19],the authors presented new security-aware cache designs, the Partition-Locked cache (PLcache) and Random Permutation cache (RPcache) to defense the cache side channel. In [17], a system-level protection mechanism called STEALTHMEM was proposed which was also a hardware based solution. In [28], cache no-fill mode was used to ensure the security.

All the defense methods can be classified into two categories: hardware-based and software-based method. The hardware-based methods implement new cache structure, which are hardware dependent and not easily transplanted. The software-based methods are into two strategies: eliminating the cache miss and confusing the cache miss. Performing cache warm is to eliminate the cache miss while adding time delay, adding redundant instruction, performing order skewing, randomizing tables etc. are to confuse the cache miss. All the existing software-based methods have some deficiencies. Performing cache warm, adding time delay and redundant instruction have large extra overhead. Using compact table, performing order skewing or randomizing the tables need to modify the algorithm itself so they are impractical.

In our paper, we present a defense scheme to eliminate remote cache timing side channels while can have the optimum performance, which is a framework that can be used to all the symmetric algorithms implemented with lookup tables. The basic idea of our scheme is to combine performing cache warm and adding time delay. Through reducing the context of cache warm and the time of delaying and eliminating the unnecessary cache warm and delay operation on the premise of guarantee security, we achieve the optimum performance. Moreover, our scheme is independent with algorithms so that it can be easily transplanted. We implement our scheme on AES. Through the contrast experiments it is further proved the security and the optimality of our scheme.

2 Background

2.1 CPU Cache

The cache, a small amount of high-speed static RAM (SRAM) located between the CPU cores and the main memory, is used to temporarily store the data recently accessed by CPU, avoiding the high-speed CPU accessing the slow RAM. As the speed gap between RAM and CPU increases, multiple levels of caches are implemented, and lower-levels caches have smaller size but achieve higher speed.

Each level cache is divided into several cache sets which contains a predefined number of cache lines, the minimal storage cell of cache. A W-way set associative cache means that each cache set contains W cache lines. When CPU attempts to access an element, the cache control logic unit judges whether it is in the cache. “Cache hit” is used to denote that the element is in the cache, while “cache miss" means the memory block containing the element needs to be loaded into the cache from the main memory.

The cache hierarchy differs among different CPUs. For example, Intel Core2 Q8200 CPU which is used in our evaluation, contains two separate cache-sharing core sets. Each core has a level-one data (L1D) cache of 32 KB, 8-way set associative, 64-byte cache line size and an instruction cache of 32 KB. The two cores of each cache set share a unified L2 cache of 2 MB, 8-way set associative, 64-byte cache line size.

2.2 Overview of AES

Our scheme works for the symmetric encryption algorithm implementations based on the lookup tables. The symmetric encryption algorithms are widely implemented based on lookup tables, e.g., the NIST block encryption standards AES [13] and 3DES, Blowfish [27] used in GPG and SSH, Twofish [16] and MARS [11] the two of the five finalist ciphers in the AES selection program. To make the description clear, we use AES as an example.

AES supports block lengths of 128 bits and key lengths of 128, 192 and 256 bits. AES is a key-iterated block cipher, and the number of round transformations depends on the key lengths (10, 12, and 14 rounds for the key length of 128, 192, and 256 bits, respectively). Each round transformation (except the last one) of AES performs SubBytes, ShiftRows, MixColumns and AddRoundKey on an intermediate result (called the state), while the last round transformation only performs SubBytes, ShiftRows and AddRoundKey.

The implementation of AES described in [13], is widely used on 32-bit architectures. It speeds up the computation by combining all operations except the AddRoundKey, into lookup tables which can be pre-computed. The implementation uses four lookup tables T_0, T_1, T_2, T_3, the size of each is 1KB, in each round transformation exception the last one, and a 256 bytes lookup table T_4 in the last round transformation.

2.3 Remote Cache Timing Side Channel Attacks

Remote cache timing side channel attacks are based on the fact that the encryption time varies for different plaintexts due to the number of cache misses, more cache misses results in longer execution time. Different remote cache timing side channel attacks are proposed, and according the analysis on the execution time, they can be classified into three types: the statistical attack, the Bernstein's attack and the collision attack.

The statistical attack. Tsunoo [29, 30] first practically implements this method to attack DES and 3DES, and shows it's effective to all symmetric algorithms that adopt lookup tables. Tsunoo [29, 30] assumes the input of the S-box is $K_i \oplus P_i$, the ith byte from the key exclusive-ORs with the ith byte of the plaintext. The difference of the ith byte and jth byte $(K_i \oplus K_j)$ can be inferred from the values of P_i and P_j using either of the following relations, which is used to reduce the key search space. A shorter execution time reflects more cache hits which means the relation (1) holds, a longer execution time means the relation (2) holds.

$$P_i \oplus K_i = P_j \oplus K_j \Rightarrow P_i \oplus P_j = K_i \oplus K_j \tag{1}$$

$$P_i \oplus K_i \neq P_j \oplus K_j \Rightarrow P_i \oplus P_j \neq K_i \oplus K_j \tag{2}$$

The Bernstein's attack. Bernstein's attack uses the distribution of AES encryption time to recover each byte of the key. It contains three phases. The first one is the learning phases, in which the attacker constructs a duplicate server with the same CPU and AES implementation of the victim server, but with a known key, then encrypts a large number of 400, 600 and 800 byte random plaintexts to build a time pattern for the various bytes. The second phases is the attack phases, where the attacker builds the time pattern for the victim server whose AES key is unknown. In the last phases, the attacker analyses the time patterns of the duplicate and the victim server for each byte to produce a set of possible key bits.

The collision attack. Each cache line contains multiple lookup table entries, and the collision is used to denote accessing different index of lookup tables in the same cache line. When the inputs of the S-box introduce the collision, the encryption time will be smaller. The first round attack cannot recover all bits of the AES key as the exact index of lookup tables can't be determined, and Acıiçmez proposed a two round cache collision attack of AES which recovers all bits of the key [3].

3 Eliminating cache timing side channel

3.1 Threat Model and Goals

Our scheme aims to provide the optimal performance, while eliminating the remote cache timing side channel attacks on the symmetric encryption algorithm implementations based on the lookup tables. The implementations using directly instruction mapping or Intel AES-NI, which are immune to the remote cache timing side channel attacks, are not considered here.

We assume the attackers have the ability to perform all known remote cache timing side channel attacks on the target device which adopts the conventional cache memory. Therefore, the attackers are able to invoke the encryption function without any limit; have the necessary information of the target device including the cache size, the cache line size and set associativity; and can observe the aggregate profile of the encryption, i.e., total numbers of cache hits and misses. Moreover, as some attacks require the special initial state of the cache, our scheme allows the attackers to set the initial state of the cache as they need.

3.2 Overview of the defense scheme

In this section, we describe detail of our scheme in Algorithm 1, which provides the most efficient protection for the remote cache timing side channel attacks.

It doesn't need to modify the encryption algorithm code, and can be applied to all the symmetric encryption algorithm implementations based on the lookup tables. Moreover, as it runs in the user mode, our scheme can be deployed in different operating systems including the commercial ones.

For the software implementations, there are two strategies to eliminate the remote cache timing side channel attacks. One is cache warm which loads the lookup tables into the cache before encryption, the other is the cache miss confusion by introducing a time delay, adding dummy access to the lookup tables, changing the instructions order or performing a random permutation of the lookup tables, and so on. Our scheme uses the cache warm (line 5 in Algorithm 1), which loads the whole lookup tables into the cache, and the cache miss confusion together to eliminate the remote cache timing side channel attacks. Moreover, in order to achieve the algorithm independent and avoid modifying the implementations, we adopt the time delay to confuse the cache miss (line 14 in Algorithm 1).

Algorithm 1 Our Defense Scheme.

```
1: $state \leftarrow 1$
2: while 1 do
3:     $pStart \leftarrow$ RDTSCP()
4:     if $state == 1$ or $pStart - pEnd > THRESHOLD2$ then
5:         WARM()
6:         $state \leftarrow 0$
7:     end if
8:     $encStart \leftarrow$ RDTSCP()
9:     ENCRYPTION()
10:    $encEnd \leftarrow$ RDTSCP()
11:    if $encEnd - encStart > THRESHOLD1$ then
12:        $state \leftarrow 1$
13:        if $encEnd - encStart < WET$ then
14:            DELAY()
15:        end if
16:    end if
17:    $pEnd \leftarrow$ RDTSCP()
18: end while
```

To achieve the optimal performance, our scheme only performs the cache warm in the necessary cases, which includes: (a) the time durationbetween two encryption (`pStart - pEnd`) exceeds the `THRESHOLD2`, which means the process scheduling may be triggered to load the memory of other process into the cache; (b) the *state* equals to 1. The *state* is set to 1 when the encryption is invoked the first time (line 1 in Algorithm 1), or the encryption time is larger than the `THRESHOLD1` (line 11 in Algorithm 1) which means the needed lookup tables are evicted from the cache.

Our scheme invokes *Delay* (line 14 in Algorithm 1)only when the encryption time is larger than the `THRESHOLD1` but less than WET(worst execution time). The *Delay* introduces the time delay to make the encryption time equal to WET for cache miss confusion.

Moreover, although the design of our scheme is independent of the software and hardware, the implementation details vary according to the underlying hardwares and softwares. For example, we adopts the `retscp` instruction provided by Intel to get the timestamp efficiently. The thresholds (`THRESHOLD1` and `THRESHOLD2`) are related to the hardware, operating system and the implementation of the encryption algorithm, and can be calculated by performing the multiple encryption in the environment previously.

4 Analysis of the defense scheme

4.1 Security Analysis

In this section, we analyze the security of our defense scheme. From section **??**, we find that the remote cache timing side channel attacks rely on the relation between the overall encryption time and the data needed to protect, that is, different secret data (e.g., plaintext and key) requires to access different lookup table, resulting various access time as some lookup table entries are in the cache while others are not, which further makes the overall encryption time differ.

Our scheme combines the cache warm and the delay to make the overall encryption fall in two intervals $(0, THRESHOLD1]$ and $[WET, \infty)$, no matter what value the secret data is. We prove that the encryption time is independent of the secret data in the following cases:

- When the encryption time is in $(0, THRESHOLD1]$, it is a fixed value for arbitrary data. The cache warm loads all the lookup tables into the cache, the access time for each entry is a constant if no element is evicted from the cache before and during the encryption, which results a fixed encryption time. [1]
- If the encryption time is larger than $THRESHOLD1$ but less than WET, it will be delayed to WET, which equals to the case that all the accessed lookup tables are out of the cache. The attackers can never infer the relation between the encryption time and the data.
- If the encryption time is larger than WET, it means that the OS scheduling or interrupt occurs before or during the encryption. As explained in section 3.1, the attacker cannot get the internal state of the victim device nor predict the time variations for OS scheduler or interrupt handlers. Therefore the attackers fail to infer the actual encryption time from the observed one.
- The attacker fails to obtain any other information from which interval the encryption time falls in, as it is independent of the input data, but relied on whether the needed cache entries are evict, or OS scheduler and interrupt handlers are invoked, which is unpredictable for the remote attackers.

[1] The impact of micro-architecture [12] on cache access time is discussed in section 7.1.

In the following, we explain how our scheme protect the secret data against the known remote cache timing side channel attacks in detail.

To perform the statistical attack, the attackers need to obtain the plaintexts with short (long) encryption time due to the cache hit (miss). However, as the cache warm in our scheme loads all the lookup tables into the cache, all the plaintexts result the short encryption time if no needed entry is evicted before or during the encryption. For the case that some needed entries are evicted or OS scheduler and interrupt handler invoked, the encryption time of the plaintext that the remote attackers observed is no less than WET. As our scheme performs the cache warm once the last encryption time is longer than `THRESHOLD1` (less than WET), the remote attackers who has no ability to know the evicted cache, may get the short encryption time for the same plaintext. Therefore, from the view of the remote attacker, the encryption time is independent of the plaintexts.

For Bernstein's attack, the attackers need to analyse the time pattern of the duplicate and victom servers for the possible key bits. The useful time pattern for the Bernstein's attack, reflects the relation between encryption and plaintext. However, in our scheme, the observed encryption time varies due to whether the OS scheduler or interrupt handlers exist before or during the encryption, independent of the plaintext (some needed entries are evicted from the cache is also because of the OS scheduler or interrupt handlers).

The collision attack relies on the case that when the inputs of the S-box introduce the collision, the encryption time will be shorter. However, in our scheme, the necessary condition of a shorter encryption time is not the collision as the cache warm loads all the lookup tables in the cache, but no OS schedulers nor interrupt handlers exist before and during the encryption. So the collision attack doesn't work under our scheme.

4.2 Optimization Analysis

Our scheme achieves the optimized performance against remote cache timing side channel attacks, without modifying encryption algorithm code. To make the execution times of cryptographic operations independent of the input, there are two main approaches: static transformation, and extend all possible path to the long one [9]. The first one is achieved by modifying the implementation of the encryption algorithm for different platforms. The second one is adopted in our scheme (`delay`). Meanwhile, we adopt `warmup` in our scheme for better performance. We'll prove that our scheme which combines the `delay` and `warmup`, achieves optimal performance. There are five principles for better performance:

- The time delayed to by `delay` operation is the shortest.
- The number of `delay` operations is as small as possible.
- The contents loaded into the cache by the `warmup` operation are the least.
- The number of `warmup` operations is as small as possible.
- As the execution time of `warmup` is less than `delay`, `warmup` is preferred.

In this section, we use AES to prove that our scheme is optimal, which can be extended to other encryption algorithms implemented based on lookup tables.

Lemma 1. *The shortest time delayed to by* `delay` *operation is WET.*

Proof. For the remote attackers, there are three values of encryption time that are independent of the input: the shortest time, WET and the value larger than WET. The shortest time means that all the lookup tables are loaded into the cache, no cache miss happens. The WET corresponds to the longest execution path, in which case no cache hit occurs. When the encryption time is larger than WET, there are OS schedulers or interrupt handlers before or during the encryption, which is unpredicted by the remote attackers.

If the time delayed to is a random or discrete value less than WET, the observed encryption time still relies on the input. The remote attackers can find the relation between the observed encryption time and input by invoking the encryption a large number of times, to perform the timing attacks. Therefore, the shortest time delayed to by `delay` operation is WET.

Lemma 2. *Perform the* `delay` *operation when the encryption time is larger than* `THRESHOLD1` *but less than WET, results the smallest number of* `delay` *operations when we cannot distinguish the reason (cache miss or the OS scheduler and interrupt handler).*

Proof. If the AES execution time is less than `THRESHOLD1`, it means no cache miss occurs and the execution time is independent of the input. When the observed execution time is no less than WET, it means OS scheduler or interruption handler is invoked, which makes the remote attacker fail to infer the actual encryption time from the observed one to perform the timing attack, as the time for OS scheduler or interruption handler is unpredictable for them.

The case that the execution time is larger than `THRESHOLD1` but less than WET, is the result of cache miss, OS scheduler and the interrupt handler (the introduced extra time is less than WET). The remote attackers can distinguish it by invoking the encryption using the same input multiple times, and perform the timing attacks further, which makes the `delay` operation necessary.

Lemma 3. *Loading all the lookup tables into the cache is the best warmup operation.*

Proof. For AES, the contents that need to warm are the five lookup tables. Even when the warmup operation loads all the lookup tables except one cache line, the execution time varies for different input, which requires to perform the `delay` operation. In the following, we prove that the expected time reduced by not loading N cache line size of lookup tables, is smaller than the time increased due to the newly needed `delay` operation.

For AES, the size of each lookup table T_0, T_1, T_2, T_3 is 1KB, while the size of T_4 is 256 Byte. We assume the size of a cache line is C Byte, then each of T_0, T_1, T_2, T_3 has $1024/C$ cache lines, while T_4 has $256/C$ ones. The probability that one cache line loads the content of T_0, T_1, T_2, T_3 is 16/17, while it loads the content of T_4 is 1/17.

AES is an iterated encryption. It accesses each of T_0, T_1, T_2, T_3 four times in the first nine rounds, and accesses T_4 16 times in the last round. Therefore, the probability that one cache line of T_0, T_1, T_2 or T_3 is not accessed in one whole AES execution is $P_{0-3} = (1-\frac{C}{1024})^{36}$, and is $P_4 = (1-\frac{C}{256})^{16}$ for T_4. The expected probability that one cache line is not accessed in one whole AES execution is $P_e = \frac{16}{17}P_{0-3} + \frac{1}{17}P_4$, and N cache lines are not accessed in one execution is P_e^N. Therefore, the probability that `delay` operation is needed while not loading N cache line size of lookup tables, is $P_{delay} = 1 - (\frac{16}{17}P_{0-3} + \frac{1}{17}P_4)^N$.

We assume the execution time corresponding to WET is W, and the shortest execution time is B, then the expected extra time introduced by not loading N cache line size of lookup tables is

$$T_{delay} = P_{delay} * (W - B) = (1 - (\frac{16}{17}P_{0-3} + \frac{1}{17}P_4)^N)(W - B) \tag{3}$$

The time reduced by not loading one cache line size of lookup tables is denoted as t_{nc}, then the time reduced by not loading N cache line size is

$$\Delta T_{warm} = N * t_{nc} \tag{4}$$

In our environment, the size C of a cache line is 64 Byte. The time W corresponding to WET is ***, while the shortest time B is **. The time t_{nc} for loading one cache line size of data is **. From Table 1, we find that not loading some number of cache lines size of lookup tables, needs more extra time compared to the reduced one. Therefore, loading all the lookup tables into the cache is the best warmup operation.

The frequency of cache warm When discussing the frequency of the cache warm, we must consider that how to decrease the number of cache warm while the probability of generating delay is lowest due to the cost of delay is larger than the cache warm.

It is obviously that performing cache warm before each AES can ensure the lookup tables into the cache but often it is not necessary because after the previous AES execution, the lookup tables may be all in the cache. So in order to get the best performance, the choices of performing cache warm have several strategies:

1. never performing the cache warm.
2. performing cache warm before every AES execution.
3. performing cache warm with a probability
4. performing cache warm under certain conditions

Table 1: The reduced and introduced time by not loading N cache line size (in ms).

N	1	2	3	10	20	30	40	50	60	68
T_{delay}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
T_{warm}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The first method is not performing cache warm. So the AES execution time will change due to the different input plaintexts. In this situation, we employ the delay operation to guarantee the security. It can be inferred that this way has the most delay operations. The extra overhead is the time of delay operation.

The second method that performing the cache warm every time doing AES doesn't generate the case that need to delay. Based on the security analysis there still have probability to need the delay operation and it is inevitable. The problem of the method is that we do many cache warm operations unnecessary when the lookup tables are all in the cache.

The third method is performing the cache warm with probability P. Every time doing AES, there is probability of P to perform cache warm before. That is performing cache warm every few AES executions. At this time, for the AES that doesn't perform the cache warm, the state of whether the lookup tables are all in the cache is unknown because the process scheduling occurs leading the lookup tables not all in the cache. Therefore the AES execution time can be related with input plaintexts and attackers can collect this type of plaintexts and corresponding time to construct an attack. To eliminate the effect using the delay operation guarantees the security of AES when the AES execution time is larger than the computation threshold. This method adds some excess cache warm operation that is unnecessary and performs a delay operation at the place where just need the cache warm. Actually, the first and the second are particular circumstances of this method where $P = 0$ and $P = 1$.

The forth method is conditional warm that is performing the cache warm under the condition of a certain and particular. The key point is to choose when to perform the cache warm. Primarily, when first time to perform AES the lookup tables are certainly not in the cache. The cache warm operation must be carried out at this condition. In ideal conditions, the lookup tables are all in the cache when computing the subsequent AES. In reality, a lot of reasons make the lookup tables not all in the cache, which is analysed before. When the AES execution time is less than the computation threshold the lookup tables can be considered all in the cache. When the AES execution time is larger than the computation threshold, neither less than WET nor larger than WET the lookup tables can be all in the cache. Also, when AES computation is inconsecutive, the lookup tables can't be guaranteed not to be evicted from the cache.

AES computation is inconsecutive. In this case, the lookup tables can't be guaranteed all in the cache. So when the AES is inconsecutive before AES execution it should perform the cache warm to load the lookup tables into the cache. We set an interval threshold to estimate whether succession or not. When the value that equals the current time minus the time previous AES is over is larger than the interval threshold, it is considered that the AES is inconsecutive and the cache warm will be performed. The interval threshold is determined as follows: the largest interval time that makes the next AES execution time less than the computation threshold serves as the interval threshold. This threshold is alterable based on the computing environment. Different environments have different interval thresholds. In an ideal environment which has low load the interval threshold can be a large value. On the contrary, in a high load environment there will be a small interval threshold.

Actually, the function of this threshold is to improve the AES performance. Using this threshold can't ensure all lookup tables in the cache but just in a high probability. The security is guaranteed by the delay operation. Because the time of warm is less than the time of delay, adding this threshold can reduce the times of delay.

AES computation is consecutive and the execution is larger than WET. In this situation, the process scheduling or interruption occurs. The AES is interrupted during the execution but when it occurs is unknown. So after AES execution whether all the lookup tables are in the cache is not able to decide. It is possible that the next AES execution time is larger than the computation threshold due to the cache miss, which leads an extra delay operation. Therefore, whether this situation should perform the cache warm need to be evaluated.

For the five lookup tables, after one AES execution, the probability of one line of lookup tables accessed is

$$P_{delay} = 1 - (\frac{64}{68}P_1 + \frac{4}{68}P_2) \tag{5}$$

where P_1 is the probability of one line of $T_0 - T_3$ not accessed and P_2 is the probability of one line of T_4 not accessed.

If one line of lookup tables is not in the cache before AES computation. When not performing the cache warm, the probability of performing delay operation is P_{delay}. So the extra time generated by the delay operation is

$$T_{delay} = P_{delay} * t = (1 - (\frac{64}{68}P_1 + \frac{4}{68}P_2))(W - B) \tag{6}$$

And when performing the cache warm, the extra time generated by the cache warm operation is

$$T_{warm} = t_{nc} + (68 - 1)t_c \tag{7}$$

The rest can be done in the same manner. If N lines of lookup tables are not in the cache before AES computation. When not performing cache warm, the probability of performing delay operation is

$$P_{delay} = 1 - (\frac{64}{68}P_1 + \frac{4}{68}P_2)^N \tag{8}$$

And the extra time generated by the delay operation is

$$T_{delay} = P_{delay} * t = (1 - (\frac{64}{68}P_1 + \frac{4}{68}P_2)^N)(W - B) \tag{9}$$

And when performing the cache warm, the extra time generated by the cache warm operation is

$$T_{warm} = N * t_{nc} + (68 - N) * t_c \tag{10}$$

The results of T_{delay} and T_{warm} are in following form.

From the form it can be concluded that in this situation, when occurring the process scheduling during the AES, performing cache warm directly is better than not.

AES computation is consecutive and the execution is larger than computation threshold but less than WET. The delay operation is performed in this case to ensure the security. The delay function we use is implemented by some data addition between registers. So performing delay operation won't change the cache state which means the lookup tables will not be evicted from cache due to the delay. As we analysed before, in this case, the AES execution is not interrupted by other processes though the execution time is larger than the computation threshold. After the AES execution, the lookup tables probably are all in the cache. Therefore, when AES execution is in this situation, whether to perform the cache warm before next AES should be careful analysed. There are two main reasons for the AES execution changing: process scheduling occurs before AES execution and the schedule function is called but scheduling is not happened.

When scheduling occurs before AES execution, the state of lookup tables in the cache is not certain. We don't know how many lines of lookup tables are still in the cache. So we do the following calculation.

For the five lookup tables, after one AES execution, the probability of one line of lookup tables accessed is $P = 1 - (\frac{64}{68}P_1 + \frac{4}{68}P_2)$. If the cache is empty before the current AES, which means all the lookup tables are not in the cache, the probability of the next AES without cache warm need the delay operation is:

$$P_{delay} = \sum_{i=1}^{68} C_{68}^{i} P^{i} (1-P)^{68-i} (1-P^{i}) \tag{11}$$

So, if there are j lines of lookup tables in the cache before the current AES, the probability of the next AES without cache need the delay operation is

$$P_{delay,j} = \sum_{i=1}^{68-j} C_{68-j}^{i} P^{i} (1-P)^{68-j-i} (1-P^{i}) \tag{12}$$

Hence, if there are j lines of lookup tables in the cache before the current AES, the extra time generated by the delay operation for the next AES is:

$$T_{delay,j} = P_{delay,j} * t = \sum_{i=1}^{68-j} C_{68-j}^{i} P^{i} (1-P)^{68-j-i} (1-P^{i})(W-B) \tag{13}$$

For choosing the cache warm operation, if the cache is empty before the current AES, the extra time generated by the cache warm for the next AES is

$$T_{warm} = \sum_{i=1}^{68} C_{68}^{i} P^{i} (1-P)^{68-i} (i * t_{nc} + (68-i)t_{c}) \tag{14}$$

if there are j lines of lookup tables in the cache before the current AES, the extra time generated by the cache warm for the next AES is

$$T_{warm,j} = \sum_{i=1}^{68-j} C_{68-j}^{i} P^{i} (1-P)^{68-j-i} (i * t_{nc} + (68-i)t_{c}) \tag{15}$$

We compare the $T_{delay,j}$ with $T_{warm,j}$ when $j = 1, 2, ..., 68$. The $t_{nc} =$, $t_c =$, $W =$, $B =$, $P =$ which are measured by experiment. The result is in the form below.

From the form, $T_{delay,j}$ is larger than $T_{warm,j}$ when $j = 1, 2, ..., 68$. So in this situation that the process scheduling occurs before the current AES performing the cache warm before the next AES is better than not.

when the schedule function is called but scheduling is not happened, the cache state is not changed. All lookup tables are still in the cache in this case. Therefore the cache warm operation is unnecessary in this situation.

the two cases produce the opposite conclusion. The next we evaluate which one is the major factor, then we can have our strategy.

We use P to mark the probability that clock interrupt occurs during the AES execution and use Q to mark the probability that the current process is scheduled at the clock interrupt and the cache state changes. While we use P_{Δ} to represent the probability that the clock interrupt occurs between two AES executions. So, $P = \frac{t_{AES}}{4ms}$ and $P_{\Delta} = \frac{\Delta}{4ms}$. The t_{AES} means the AES execution time and the Δ means the interval time of two AES executions. 4 ms is the interval time of the clock interrupt of our experiment device.

So, the probability of performing the delay operation at the current AES is

$$P_{warm,j} = P(1-Q) + P_{\Delta}Q \tag{16}$$

The next AES, if choose the cache warm method, the extra time overhead is

$$T_{warm} = P(1-Q)t_{warm,min} + P_{\Delta}Qt_{warm} \tag{17}$$

where $t_{warm,min}$ means the time of performing cache warm when all lookup tables in the cache. The t_{warm} is variable due to the number of lines of lookup tables not in the cache is known.

and if choose not cache warm, the extra time overhead is

$$T_{delay} = P_{\Delta}Qt_{delay} = P_{\Delta}Q(W-B) \tag{18}$$

where t_{delay} is the extra overhead generated by the delay operation.

If $T_{warm} < T_{delay}$, we can obtain:

$$Q > \frac{P * t_{warm,min}}{P * t_{warm,min} + P_{\Delta}(t_{delay} - t_{warm})} \tag{19}$$

From experiments, $\Delta = 840 circles$, $t_{AES} = 587$, $t_{warm,min} = 315$ and the max of $t_{warm} = 2700$. So when $Q > 0.18$, $T_{warm} < T_{delay}$. At this moment we choose the cache warm to improve the AES performance. This means the scheduling occurs more frequently which is equivalent to a high load of the system. However, when the system has a low load, it is better to choose not performing the cache warm operation.

For the four methods of performing cache warm above, we choose the conditional warm method. Firstly, for the first method that not performing the cache warm, when the lookup tables are all in the cache, it is the same as the conditional warm method. However, when the delay is happened, there has probability to perform the cache warm for the conditional warm method, but the first method doesn't perform the cache warm. In this situation, the performance of the first method is worse than the conditional warm method due to the cost of cache warm is less. Secondly, the second method to perform the cache warm every time itself does many cache warm operations that originally are unnecessary when the lookup tables are all in the cache. When need to perform the delay operation, it is the same as conditional warm method. So, the whole performance of the second is worse than the conditional warm method. Lastly, for the third method that is to perform the cache warm with a probability P, we compute the expectation of the extra time overhead. At the same time we compute the expectation of the extra time overhead of the conditional warm method.

We set P as the probability to perform the cache warm and $\overline{P}$ as the probability that the lookup tables are not all in the cache but the AES execution is not affected which means the AES execution time is still below the computation threshold. While we set $\widetilde{P}$ as the probability of occurring the interruption including the process scheduling. There are three states of the lookup tables in the cache: all in the cache, part in the cache but the AES not being affected, part in the cache and the AES being affected. The state of the lookup tables before the AES as the change of the time satisfies the Markov chain.

For the probabilistic warm method, the state-transferring probability diagram is Figure 1(a).

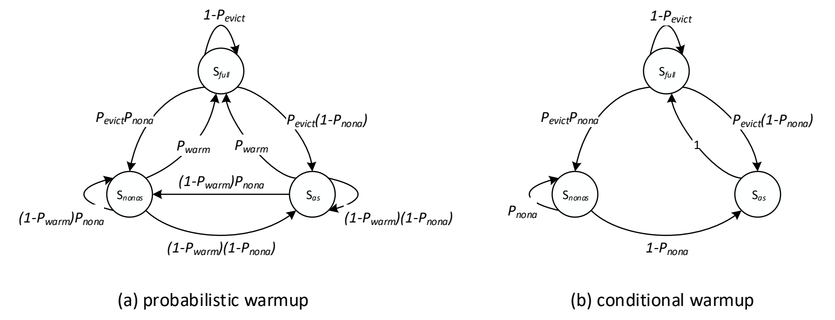


Fig. 1: the Markov state-transferring probability diagram of the probabilistic warmup and conditional warmup.

We use Π_1, Π_2, Π_3 to represent the limit distribution of the three states of the lookup tables when the Markov chain reaches stable state. So we can obtain the equation set as follows.

$$\begin{cases} 1 = \Pi_1 + \Pi_2 + \Pi_3 \\ \Pi_1 = \Pi_1(1-\widetilde{P}) + \Pi_2 P + \Pi_3 P \\ \Pi_2 = \Pi_1(\widetilde{P}\overline{P}) + \Pi_2(1-P)\overline{P} + \Pi_3(1-P)\overline{P} \end{cases} \tag{20}$$

Solving the equation set the result is :

$$\Pi_1 = \frac{P}{\widetilde{P}+P}\ ,\Pi_2 = \frac{\overline{P}\widetilde{P}}{\widetilde{P}+P}\ ,\Pi_3 = \frac{\widetilde{P}(1-\overline{P})}{\widetilde{P}+P}$$

Therefore, the extra time overhead of the probabilistic warm method is

$$T_P = \Pi_3 t_{delay} + P t_{warm} = \frac{\widetilde{P}(1-\overline{P})}{\widetilde{P}+P} t_{delay} + P t_{warm} \tag{21}$$

For the conditional warm method, the state-transferring probability diagram is Figure 1(b).

We use Π'_1, Π'_2, Π'_3 to represent the limit distribution of the three states of the lookup tables when the Markov chain reaches stable state with the the conditional warm method. So we can obtain the equation set as follows.

$$\begin{cases} 1 = \Pi'_1 + \Pi'_2 + \Pi'_3 \\ \Pi'_1 = \Pi'_1(1-\widetilde{P}) + \Pi'_3 \\ \Pi'_2 = \Pi'_1\widetilde{P}\overline{P} + \Pi'_2\overline{P} \end{cases} \tag{22}$$

Solving the equation set the result is :

$$\Pi'_1 = \frac{1-\overline{P}}{1-\overline{P}+\widetilde{P}}\ ,\Pi'_2 = \frac{\overline{P}\widetilde{P}}{1-\overline{P}+\widetilde{P}}\ ,\Pi'_3 = \frac{\widetilde{P}(1-\overline{P})}{1-\overline{P}+\widetilde{P}}$$

Therefore, the extra time overhead of the conditional warm method is

$$T_C = \Pi'_3(t_{delay} + t_{warm}) = \frac{\widetilde{P}(1-\overline{P})}{1-\overline{P}+\widetilde{P}}(t_{delay} + t_{warm}) \tag{23}$$

We mark the $T_P = P_1 t_{delay} + P_2 t_{warm}$ and $T_C = P'_1 t_{delay} + P'_2 t_{warm}$. So

$$P_1 = \frac{\widetilde{P}(1-\overline{P})}{\widetilde{P}+P}, P_2 = P$$
$$P'_1 = \frac{\widetilde{P}(1-\overline{P})}{1-\overline{P}+\widetilde{P}}, P'_2 = \frac{\widetilde{P}(1-\overline{P})}{1-\overline{P}+\widetilde{P}}$$

When $P = 1-\overline{P}$, it can be inferred that $P_1 = P'_1$, $P_2 > P'_2$. So $T_P > T_C$.

When $P > 1-\overline{P}$, it can be inferred that $P_1 < P'_1$, $P_2 > P'_2$. However, $P_2 > P_1$ and P_1 is close to zero, which means P_1 can be neglected and the t_{warm} is the main influence factor. So we can infer that $T_P > T_C$.

When $P < 1 - \overline{P}$, it can be inferred that $P_1 > P'_1$ but P_2 and P'_2 is incommensurable. However, $P_1 > P_2$ and P'_2 is close to zero, which means P'_2 can be neglected and the t_{delay} is the main influence factor. So we can infer that $T_P > T_C$.

Through the above analysis, the extra time overhead of conditional cache warm method is less than the probabilistic warm method, so the conditional cache warm method is better than the probabilistic warm method. Therefore, we choose the conditional warm method in our defense scheme by these comparisons.

5 Implementation

We implement our defense scheme into Linux kernel 3.9.2 for 32-bit x86 compatible platforms running on an Intel Core2 Q8200 CPU. To achieve our goals that make the AES security and have the best performance, the implementation must be accurate and elaborate. The timer function, cache warm function, delay function should be well designed. Also the choice of the two thresholds need to carefully selected.

5.1 The timer function

The timer function is the key point in our defense scheme. It is used to record the AES execution time and the interval time between two AES. The main extra time overhead of our defense scheme comes from the timer function. So the time cost by the timer function should be as little as possible. Meanwhile, because the AES execution time is very short, the accuracy of the timer function should be high enough.

There are several functions can be utilized to measure the time. However the C functions like `time()`, `clock()` just have a low accuracy to the level of 10 ms. The accuracy of `gettimeofday()` is fixed to the microsecond but the result of the function has a high fluctuation because of the overhead of the system call. The CPU instruction `RDTSC` also can be used to time and it can reach the clock cycle precision. So we choose to use the `RDTSC` to implement our timer function.

Using the `RDTSC` instruction we should solve three problems. First is the synchronization. The TSC on each kernel can't be guaranteed the synchronization. This problem can be solved by adding a patch[x86: unify/rewrite SMP TSC sync code]. The second problem is the clock cycle may be changed due to the energy-saving function of the computer. We can disable this function in the BIOS. So we can assume that the clock cycle is constant. The last is that `RDTSC` is a non-serializing call, so using the instruction alone will not prevent the CPU from reordering it. To solve the problem we can use the `CPUID` instruction before `RDTSC`. Because the `CPUID` is a serializing call, the order can be guaranteed. The timer function is on the below.

Listing 1.1: Timer function

```
inline uint64_t rdtsc()
{
    unsigned long a, d;
    asm volatile ("cpuid; rdtsc" : "=a" (a), "=d" (d) : : "ebx", "ecx");
    return a | ((uint64_t)d << 32);
}
```

Also, the Intel has a new instruction RDTSCP which is a serializing call. The RDTSCP has the same function and accuracy with the RDTSC instruction. Using RDTSCP doesn't need to use CPUID, so it has a better performance due to the high overhead of CPUID. The reason that we don't choose RDTSCP is the experiment platform we use doesn't support the instruction. If transplanting our defense scheme to the newer CPU, the RDTSC instruction can be replaced with RDTSCP to reach a higher performance.

5.2 The cache warm operation

The cache warm operation loads all five lookup tables into the cache. So the implementation can be just accessing all the five lookup tables once. Considering the cache structure, to improve the performance we can access one byte every 64 bytes to avoid repetitive access.

5.3 The delay operation

The delay operation prolongs the AES execution time to the WET. So the delay operation also need a high accuracy. The usleep() can be used to delay but it has several problems. The accuracy of usleep() can reach microsecond level but the overhead of function itself is very high. Also, during the sleepy time of the usleep() the cache state can be changed so that the state of the lookup tables in the cache is unpredictable. The two main reasons make it inapplicable in our scheme.

Therefore we should implement the delay function ourselves. We using two mov instructions in our delay function. the input of the function tells how many times to looping execute. The implementation of the delay function is on below.

Listing 1.2: Timer function

```
volatile int delay(uint64_t d_time){
        int t;
        t= d_time;
        int a= 10;
        int b=7;
        for(;t>0;t--){
                asm volatile (
                "mov %1 , %%eax; mov %%eax, %0;" :
                "=r" (b):
                "r" (a):
                    "%eax");
        }
        return b;
}
```

The relationship between the delay time and the input of the function is measured in Figure 2. It can be concluded that it is a linear relation. So through linear fitting we can obtain the equation

$$delay_t ime = 7.0029 * input + 8.5741 \tag{24}$$

Based on the equation, when we need to use the delay function, we can calculate to get the input of the function.

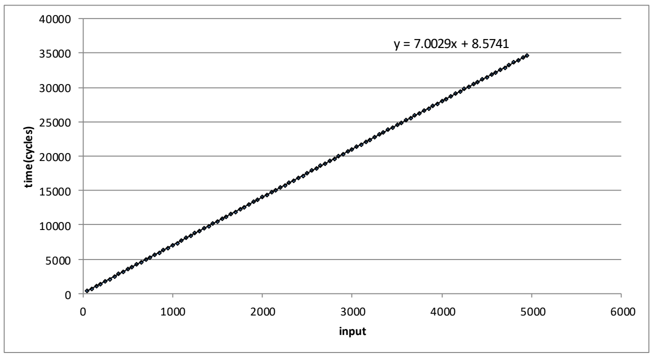


Fig. 2: The delay time changes with the different inputs.

5.4 The choice of the threshold

In our defense scheme there are two thresholds. The computation threshold is the most important. The choice of the computation threshold is based on the distribution of the AES execution time. The computation threshold should satisfy some requirements.

1. The threshold value must be close to the minimum AES execution time.
2. The value must less than the AES execution time that one line of lookup tables not in the cache.
3. The value must between the first two peaks of the distribution of AES.

In our experiment, we measure the shortest AES execution time is 587 cycles. So the computation threshold is set as 600 cycles in our defense scheme.

The interval threshold is used to represent the continuity of the AES execution. When the interval time of two AES executions is larger than the threshold, the cache warm function should be called. This threshold mainly is used to improve the performance of AES. So the choice of this threshold is based on the running load of the system. The high load corresponds to a low interval threshold because the cache state may be changed in a high possibility. Similarly the low load corresponds to a large interval threshold due to the low possibility of changing the cache state. The interval threshold can by measured by finding the largest interval time that makes the next AES execution time less than the computation threshold.

In our experiments for measuring the performance in next section we set the clock cycle as the interval threshold in a low load situation because the process scheduling function is called every clock cycle. And we measure the interval threshold in a high load situation as 2356 cycles.

6 Performance Evaluation

6.1 The performance comparison

Through the security and performance analysis, not only the interval threshold but also the delay strategy are based on the load of the execution environment. When comparing the performance between the original AES and AES using the defense scheme, the load situations should be considered separately. We do the comparative experiments in three situations: the low load system, the high computing load system and the high memory reading load system. We use SysBench to implement the high load environment. We run the SysBench in its CPU mode and make the benchmark launching 4 threads to issue 10K requests. Each request consists in calculation of prime numbers up to 30K. Running the SysBench circularly constructs the high computing load environment. In the similar way, to construct the high memory reading load environment, we circularly run the SysBench in its memory mode and make it launching 4 threads that each thread runs the one KB each read or write operation for 3 GB data.

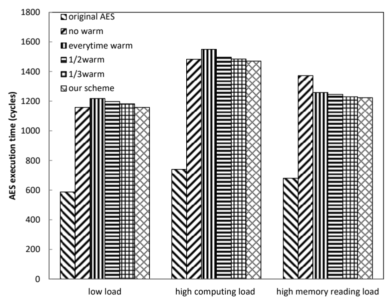
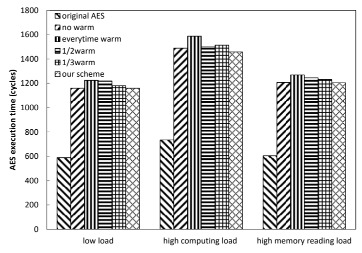


Fig. 3: AES execution performance with clock cycle of 4ms.

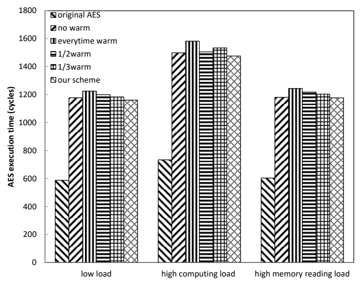
In each environment, we compute the original AES execution time, the AES execution time that using our defense scheme, the AES execution time that never performing the cache warm, the AES execution time that every time that performing the cache warm and the AES execution time that performing the cache warm with the probability of 1/2 and 1/3, which is corresponding to the four methods of the cache warm strategy. The experiments' results is in Figure 3.

From the figure the execution time of the AES with our defense scheme is 1.97 times of the original AES. The main influence factor of the performance is the four timers because the timers spend much time. Meanwhile, compared with other cache warm method our defense scheme is the most efficient.

The clock cycle of the system in the above experiment is 4ms. To find out the effect to the AES execution by the periodical process scheduling, we also do the same experiment on the system whose clock cycle is 10ms and 1ms. The result is in Figure 4.



(a) clock cycle is 1ms



(b) clock cycle is 10ms

Fig. 4: AES execution performance with different clock cycle

From the two figure, our defense scheme is the best one no matter what the clock cycle is.

7 Related Discussions

7.1 The full cache warm condition

In the paper [12], the author finds even the lookup tables are all in the cache the AES execution time can change due to the computer architecture. The peak of the distribution of AES execution time nearby the fastest value can be used to attack. We make an experiment that measure the distribution of AES execution time on different CPUs to find out the influence by the architecture. We perform the AES on Intel Core2 Q8200, Intel Core i5, Intel Core i7 and Xeon*** and find that the peak which can be utilized almost disappears and is smaller and smaller when the performance of the CPU is better.

For our defense scheme, if the computation threshold is chosen larger than the value of the peak, the effect of the peak is not eliminated. If the computation threshold is less than the value of the peak, the AES execution time affected by the peak is delayed to the WET. At this time the effect is eliminated. So when we choose the computation threshold we must consider this situation and make the computation threshold less than the value of the peak.

7.2 The instruction cache

The instruction cache has influence on the AES execution time. The computation threshold is the shortest AES execution time which is the execution time that all the AES instructions are in the cache. So influence of the interruption and the process scheduling is not just about the state of lookup tables in the cache, but also the instruction cache which can be reflected on the execution time. If the instructions of AES are not in the cache, the AES execution time will be larger than the computation time resulting in performing additional delay and cache warm operation. But when normally computing AES, the instruction is always in the cache and the change of instruction cache is usually accompanied by the change of data cache. Therefore we don't take into account the influence of the instruction cache in our analysis is reasonable.

7.3 Other attacks

There are two other kinds of cache attacks except the timing attack. The trace-driven attack utilizes to monitor the changes of the energy consumption or the electromagnetic radiation during an encryption so that the attackers can capture the profile of the cache activity to deduce the cache hits and misses. So the attackers can obtain the information of every cache line at every access. In this case the delay operation is useless. For the attacks in [13, 21, 2, 1], using our defense scheme most of the encryptions are under the condition that all lookup tables are in the cache. Only the process scheduling or interruption can change the state of the cache, However the process scheduling and interruption are the interference factors. So our defense scheme is efficient to trace-driven attack because there are no cache misses.

In access-driven attack, the adversary can execute a spy process on the target device in order to determine whether a cache line has been evicted or not. There are three kinds of accuracy for the spy process which are several AESs between two detections, just one AES between two detections and several detections during one AES. For our defense scheme just operates before and after the AES execution, So it is efficient for the former two accuracy. When the spy detects several times during AES, our scheme is useless.

8 Conclusion

We present a defense scheme to eliminate the cache timing side channel. Our scheme uses the cache warm and delay strategy to guarantee the security of the symmetric algorithm. Our scheme is a software implementation which can be easily used in different systems Also the cryptographic algorithms don't need to modify the code implementation which improves the universality of our scheme.

While analyzing the security of our scheme, we analyze the performance of the scheme in details. Our scheme has the best performance against the cache timing attack. We do a serial of contrast experiments and prove the optimality of our scheme.

References

1. Acıiçmez, O., Çetin Kaya Koç: Trace-driven cache attacks on aes. Information & Communications Security 2006, 112–121 (2006)
2. Acıiçmez, O., Çetin Kaya Koç: Trace-driven cache attacks on aes (short paper). In: Information and Communications Security, International Conference, ICICS 2006, Raleigh, Nc, Usa, December 4-7, 2006, Proceedings. pp. 112–121 (2006)
3. Acıiçmez, O., Schindler, W., Çetin K. Koç: Cache Based Remote Timing Attack on the AES. Springer Berlin Heidelberg (2006)
4. Aly, H., Elgayyar, M.: Attacking AES using bernstein's attack on modern processors. Springer Berlin Heidelberg (2013)
5. Bernstein, D.J.: Cache-timing attacks on aes. Vlsi Design IEEE Computer Society 51(2), 218 – 221 (2005)
6. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: Aes power attack based on induced cache miss and countermeasure. In: International Conference on Information Technology: Coding and Computing. pp. 586–591 Vol. 1 (2005)
7. Bonneau, J.: Robust final-round cache-trace attacks against aes. 2006 (2006)
8. Bonneau, J., Mironov, I.: Cache-collision timing attacks against aes. In: Cryptographic Hardware and Embedded Systems - CHES 2006, International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. pp. 201–215 (2006)
9. Braun, B.A., Jana, S., Dan, B.: Robust and efficient elimination of cache and timing side channels. Computer Science (2015)
10. Brickell, E., Graunke, G., Neve, M., Seifert, J.P.: Software mitigations to hedge aes against cache-based software side channel vulnerabilities. (2006)
11. Burwick, C., Gennaro, R., Halevi, S., Jutla, C., Matyas, S.M., Safford, D., Zunic, N.: The mars encryption algorithm. IBM (1999)
12. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding cache attacks (2006)
13. Daemen, J., Rijmen, V.: The design of rijndael: Aes, the advanced encryption standard. Springer-Verlag (2002)
14. Dan, P.: Theoretical use of cache memory as a cryptanalytic side-channel. Journal of Arid Environments 2002(10), 393–446 (2002)
15. Dan, P.: Partitioned cache architecture as a side-channel defence mechanism. Iacr Cryptology Eprint Archive (2005)
16. Kelsey, B.S.J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: A 128bit block cipher (1998)
17. Kim, T., Peinado, M., Mainar-Ruiz, G.: Stealthmem: system-level protection against cache-based side channel attacks in the cloud. In: Usenix Conference on Security Symposium. pp. 11–11 (2012)

18. Kong, J., Acıiçmez, O., Seifert, J.P., Zhou, H.: Hardware-software integrated approaches to defend against software cache-based side channel attacks. In: IEEE International Symposium on High PERFORMANCE Computer Architecture. pp. 393–404 (2009)
19. Kong, J., Acıiçmez, O., Seifert, J.P., Zhou, H.: Architecting against software cache-based side-channel attacks. IEEE Transactions on Computers 62(7), 1276–1288 (2013)
20. Kong, J., Acıiçmez, O., Seifert, J.P., Zhou, H.: Deconstructing new cache designs for thwarting software cache-based side channel attacks. In: ACM Workshop on Computer Security Architecture, Csaw 2008, Alexandria, Va, Usa, October. pp. 25–34 (2008)
21. Lauradoux, C.: Collision attacks on processors with cache and countermeasures. In: WEWoRC 2005 - Western European Workshop on Research in Cryptology, July 5-7, 2005, Leuven, Belgium. pp. 76–85 (2005)
22. Mairéad, O.., Hanlon, A., Tonge: Investigation of cache timing attacks on aes. School of Computing 51(2), 306 – 311 (2005)
23. Mer, J., Krummel, V.: Analysis of countermeasures against access driven cache attacks on AES. Springer Berlin Heidelberg (2007)
24. Neve, M., Seifert, J.P., Wang, Z.: Cache time-behavior analysis on aes (2006)
25. Neve, M., Seifert, J.P., Wang, Z.: A refined look at bernstein's aes side-channel analysis. In: ACM Symposium on Information, Computer and Communications Security. pp. 369–369 (2006)
26. Page, D.: Defending against cache-based side-channel attacks. Information Security Technical Report 8(1), 30–44 (2003)
27. Schneier, B.: Description of a new variable-length key, 64-bit block cipher (blowfish). In: Fast Software Encryption, Cambridge Security Workshop. pp. 191–204 (1993)
28. Taha, Y.H., Abdulh, S.M., Sadalla, N.A., Elshoush, H.: Cache-timing attack against aes crypto system - countermeasures review (2014)
29. Tsunoo, Y.: Cryptanalysis of block ciphers implemented on computers with cache. Preproceedings of Isita (2002)
30. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of des implemented on computers with cache. Proc of Ches Springer Lncs 2779, 62–76 (2003)
31. Tsunoo, Y., Tsujihara, E., Shigeri, M., Kubo, H., Minematsu, K.: Improving cache attacks by considering cipher structure. International Journal of Information Security 5(3), 166–176 (2006)
32. Wang, Z., Lee, R.B.: New cache designs for thwarting software cache-based side channel attacks. In: International Symposium on Computer Architecture. pp. 494–505 (2007)
33. Wang, Z., Lee, R.B.: A novel cache architecture with enhanced performance and security. In: Ieee/acm International Symposium on Microarchitecture. pp. 83–93 (2008)
34. Weiß, M., Heinz, B., Stumpf, F.: A Cache Timing Attack on AES in Virtualization Environments. Springer Berlin Heidelberg (2012)