Cache Attacks: Past, Present and Future

X X X X
UL Transaction Security
{X X X X}@ul.com

December 1, 2014

Contents

Chapter 1

Introduction

1.1 Micro-Architectural-based Side-Channel Attacks

1.1.1 Brief discussion about Side-Channel Attacks

By definition, a Side-Channel Attack (SCA) is any attack based on information gathered from the physical implementation of a target cryptosystem. It can be for instance, timing information, power consumption, electromagnetic emanations, sound, etc.

Cryptographic implementations can leak sensitive information because of the physical properties and requirements of the cryptographic imlpementations and computational environments.

The initial focus of Side-Channel research was on smart card security. Smart cards are mainly used for storing secret/sensitive data and processing secure transaction. This interest in applying such attacks on smart cards is due to the fact that the measurements of side-channel information on smart cards are more easily observable compared to their application on devices with more computational resources.

1.1.2 Basics on CPU

CPUs are composed of many different elements/components. Each component provide a specific feature. One will see later that these features can be exploited to compromise the security of computational environments. The next paragraphs aims at giving some basic definitions regarding the Cache.

1.1.2.0.1 CPU Cache While the processor needs to read from or write to a location in the main memory, it first checks whether a copy of that data is in the cache. If so, it imeediately reads from or writes to the cache. This is much faster than reading from or writing to the main memory.

Nowadays modern CPUs have at least three independent caches: an *Instruction cache* (I-Cache) to speed up executable instruction fetch, a *Data* *cache* (D-Cache) to speed up data fetch and store, and also a *Translation Lookaside Buffer* (TLB) used to speed up virtual to-physical address translation for both executable isntructions and data.

While the data to be processed is already in the cache, the CPU immediately uses this data: a so-called Cache Hit has occured. On contrary, if the requested data is not yet in the cache, this is called a Cache Miss.

1.1.2.0.2 Cache entries Data is transferred into the cache in blocks of fixed size, called cache lines. It is the minimum amount of data that can be read from the main memory into the cache at once.

1.1.2.0.3 Replacement policy In case of a cache miss the cache may have to evict one of the existing entries. An heuristic is used to choose the entry to evict. This is called the replacement policy

1.1.2.0.4 Cache associativity Associativity is a trade-off while implementing caches. While the replacement policy allows to choose any entry in the cache to hold the copy, the cache is then called *Fully Associative* cache. At the other extreme, in case where each entry in the main memory are associated to just one location in the cache, then it is called a *Direct Mapped* cache.

Between both extremes, several cache implementations exist such as each entry in the main memory can go to any one of N slots in the cache. They are called *N-way set associative.*

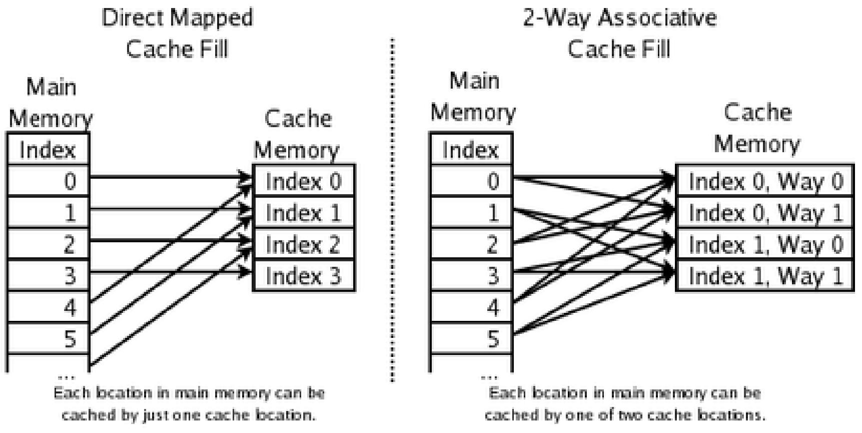

Figure 1.1: Cache associativity (`http://en.wikipedia.org/wiki/CPU_cache`)

1.1.3 Micro-Architectural Attacks

Micro-Architectural Attacks (MAA) can be considered as a special form of Side-Channel Analysis. It mainly exploits micro-architectural functionalities of processor implementations to compromise the security of computational environments even in the presence of sophisticated protection mechanisms such as sandboxing or virtualization.

According to the hardware components used to obtain a side-channel information, MAA actually exploit timing and access variations caused by those components. Basically there are currently 4 types of MAA in the literature:

Data Cache Timing Attack (DCTA) Also known as Cache Attacks, DCTA is the most mature type of attack in MAA and as depicted by the Table 2.1 many attack methods have already been proposed by researchers. Cache Attacks tend to reveal the data access patterns of cryptosystems, unlike the following ones which rather expose the execution flow of the ciphers.

Instruction Cache Analysis Attack (ICAA) Aciiçmez proposed this MA source in [2]. Instruction cache reveals fetching information of a program and so the execution flow. His attack targets RSA algorithm and he showd that by creating intentional conflicts between the instructions of a RSA process and a spy code and forcing the processor to evict the RSA instructions out of I-cache could enable him to obtani the knowledge of the program flow cache timing.

Branch Predication Analysis Aciiçmez demonstrated in [4] the first *Simple Branch Predicion Analysis* (SBPA). He targeted OpenSSL's Square & Multiply RSA implementation and he showed that the execution flow of the RSA process could be revealed by observing the Branch Target Buffer (BTB) state transistions during a single RSA operation.

Shared Function Units Attack (SFUA) Nowadays according to the Operating System (OS) and the Hardware design, many devices can run applications in parallel. Either it is quasi-parallel managed by the OS scheduler, or more or less explicitly parallel depending on the degree of additional hardware such as Dual Processors, Dual Cores, Simultaneous Multi Threading, etc. In Simultaneaous Multi-Threaded (SMT) processors, threads share a pool of *Functional Units* (FU) allocated dynamically to each process every cycle. Shared FUs can then be exploited such as one process can interfere with another one, leading to side channels.

These attacks rely on the fact that several applications can share the same processor resources. The shared usage between a spy and crypto process enables the spy process running in parallel to the victim process to extract sensitive information.

Chapter 2

Cache-based Timing Attack

2.1 Introduction

Cache-based attacks basically exploits the cache behavior of a cryptosystem by getting the execution time and/or the generated power consumption variations due to cache hits and misses. In the literature there are three different categories of Cache Attacks: *time-driven attacks*, *access-driven attacks* and *trace-driven attacks.*

Time-Driven Cache Attacks During a Time-Driven attack, an adversary is able to observe the overall time needed to perform certain computations, such as whole encryptions. From these timings he can make inferences about the overall number of cache hits and misses during an encryption.

Trace-Driven Cache Attacks In Trace-Driven attack, the adversary is able to obtain the traces of cache hits and misses for a sample of encryptions and recovers the secret key of a cryptosystem using this data. A "trace" is a sequence of cache hists and misses. E.g: M H H M, H M M H, etc. The attacker has to be able to profile the cache activity during a single encryption. Furthermore he has to know which memory access of the encryption algorithm causes a cache-hit.

Access-Driven Cache Attacks In Access-Driven attack the adversary is able to get more fine grained information about the cache behaviour. He should be able to determine the cache sets accessed by the cipher. For instance let a spy process S and a cryptographic victim process V run concurrently. They use the same cache. After letting V run for some amount of time and potentially letting it change the state of the cache, S observes the timings of its own memory accesses, which depend on the state of the cache.

Access-driven and Trace-driven cache attacks rely on more sophisticated knowledge about the implementation and the underlying hardware architecture. However, access-driven and trace-driven attacks require far less measurement samples than time-driven attacks. It seems that Trace and Access-driven attacks are highly platform dependent while Time-driven attacks are portable to different platforms.

2.1.1 Problem Analysis

A perfect memory system would provide at the working frequency of the CPU and would require constant time for each access. Unfortunately it is not the case for CPU architectures nowadays. A majority of block ciphers such as AES and DES store one (or several) precomputed arrays in memory to speed-up their execution by avoiding executing too much instructions. Bernstein [9] capitalized on the fact that many block ciphers, specifically AES, leak timing information during cache hits and misses. This/these array(s) may not necessarily be loaded into the cache before the encryption or decryption process starts. Therefore depending on the state of the cache different attacks may or may not be applicable.

2.1.1.1 Cache Misses categories

According to Hill in [22] as cach misses can occur at every cache level, the author categorized those cache misses into three categories.

Category	Description
Cold Start Misses	The first access to a block can not be in the cache, hence there must be a compulsory miss
Capacity Misses	In case the cache is too smal to hold all of the blocks needed during a process execution, misses occur on blocks that were discarded earlier
Conflict Misses	In case where the cache has sufficient space for the data, but the block can not be kept because the set is full, a conflict miss will then occur

2.1.1.2 Canteaut et al. Cache Attacks Classification

In [17], Canteaut et al. proposed to classify cache attacks according to the initial state of the cache:

Category	Description	Attack Dependencies
Reset Attacks	Arrays used by the crypto. cipher is not loaded into the cache before the attack begins	Cold Start Misses
Initialization attacks	The attacker is able to set the cache into a known state before the attack starts	Cold Start and Conflict Misses
Micro-Architecure attacks	All the required arrays are first loaded into the cache before the attack beginsi	Conflict Misses + Timing penalties due to CPU micro-architecture

The authors analyzed the different consequences of these three different cache states while starting an attacks.

Starting from an empty cache implies the complexity of resetting a cache memory. Therefore it depends on the target device. On small embedded systems such as smart cards, by simply removing the voltage supply the cache will be cleared due to its volatile nature. This

It is another history on more complex systems like computers. Resetting the cache will require to perform a lot of memory access on all cache blocks before any observation.

Starting from an initialized cache implies to initialize some chosen cache blocks with data from the lookup tables. This can be done by first flushing the cache memory and then by trigerring a fake encryptions with a known key in order to load known part of the table into the cache. This kind of attack requires then to get access to the cache memory.

Starting from a loaded cache prevents from having cache misses. However it does not remove timing variations as pointed out by Bernstein [9]. There a lot of different events that could occur and cause those timing variation. A deep knowledge of the target CPU would help in understanding the whole micro-architecture of the processor in order to have a better overview of what might cause the timing variations. According to Canteaut in [17] one of the many reasons of this issue are the conflict misses (cf. 2.1.1.1) that can occur. The latters mainly depend on the cache size, its associativity and on the replacement policy (cf. 1.1.2.0.2, 1.1.2.0.4 and 1.1.2.0.3).

2.1.1.3 Synchronous attacks Vs. Asynchronous attacks

While attacking the AES Osvik et al. described in [31] those two families of attacks.

Synchronous attacks – Some assumptions must be taken into account for this family of attack:

- known ciphertext/plaintext
- attacker's process run synchronously with the encryption on the same process
- consequently, the attacker has some interaction with the encryption code which allows him to obtain known plaintexts and execute code just before and just after the encryption

Furthermore, this kind of attack require two stages. First there is an online stage during which a set of random samples are obtained. Each sample consist of a known plaintext and the memory-access side-channel information gathered during the encryption of that plaintext. Then in a second stage, an offline step, the data is cryptanalyzed.

Asynchronous attacks – This family of attack is slightly different than Synchronous attacks in a way that the attacker has no more need to have interaction with the encryption code as described above.

- the attacker still executes his own program on the same processor as the encryption program
- there are no explicit interaction such as inter-process communication or I/O
- knowledge of the non-uniform distribution of the plaintexts or ciphertexts

This type of attack consists in ascertaining patterns of memory access performed by other processes just by performing and measuring accesses to its own memory. As claimed by the authors, it is very effective on certain platforms that embed CPUs that imlpement Simultaneous Multi-Threading (SMT).

2.2 Related works

Cache-based Timing attacks are the most analyzed in the literature. The aim of this chapter is to summarize through different analysis the different work that have been done in the literature, so it would be easier to get an overview of what have been done so far, under what specific environments and assumptions, and especially, what are the non-resolved issues so far.

The section that follows gives a non-exhaustive chronologically sorted overview of the different related works in the literature.

Table 2.1: Cache Attacks Timeline

1992	Hu first considered in [23] possibility of cross-process leakage via cache state in the context of intentional transmission via covert channels.
1998	Kelsey et al. mentionned in [26] the prospect of "attacks based on cache hit ratio in large S-box ciphers".
2002	Page extended Kelsey's work in [32] and described theoretical attacks on DES via cache misses and the ability to identify cache effects with a very high temporal resolution in side-channel traces .
2003	Brumley et al. adapted the attack principle introduced by Schindler [38] and demonstrated in [16] that side-channel attacks are a real danger not only to smart cards but also to widely used computer systems. A successful and practical remote timing attack on real applications over a local network have been performed and enables the authors to recover RSA private keys from an OpenSSL-based web server using packet timing information.
	Tsunoo et al. devised in [44] a timing-absed attack on MISTY1 and DES/3-DES exploiting the effects of internal cache collisions between the various memory lookups invoked by cipher.
2004	Bernstein proposed in [9] the first practical application of a remote time-driven cache attack against a server running AES.

2005 Bertoni et al. proposed in [11] an attack against an implementation of AES aiming at exploiting cache misses during encryption phase using power traces to identify where cache missess occur.

Aciiçmez et al. significantly improved Brumley et al. works [16] in [3].

Percival described in [33] a cache-based attack on RSA for processors with simultaneous multithreading.

Osvik et al. studied cache bahavior analysis in [31]. First they introduce a complex analysis of theoretical attack based on first and second round information. They also proposed the Evict+Time and Prime+Probe methods for extracting cache information.

Lauradoux presented in [27] new attack on the first round of AES based on power analysis, which combines both collision attacks and cahce attacks.

2006 Neve et al. refined Bernstein's approach in [30].

Canteaut et al. described in [17] a variant of Bernstein's attack which focuses on internal collisions and provide a more in-depth experimental analysis.

In his paper [36] Salembier tested Bernstein's attack [9] on different AMD CPUs in order to have an evaluation of the total aount of time required to run the entire attack.

Bonneau et al. improved Tsunoo [44] collision-based attack in [14] by attacking the first or last round of AES.

Bonneau et al. experimented in [13] a robust approach for attacking AES by not assuming a clean cachei. They demonstrated and claim that such an attack is pratical with a low number of samples and does not requite heavy computations .

2007 Aciiçmez presented in [2] a new Micro-Architectural Attack called Instruction Cache Analysis Attack. He experimented this new source of information leakage by presenting a simple pure software-based Instruction Cache attack on OpenSSL's RSA implementation.

Acimez et al. also proposed in [5] a similar attack extended to use second round information of the AES encryption.

Tiri et al. presented in [42] a model for statistical estimation of the effectiveness of AES cache attacks based on sizes of cache lines and lookup tables.

2008

Zhao et al. introduced in [25] a first two rounds access-driven attack on AES. They proposed an elimination techniue by guessing the key bytes and succeeded in recovering the full 128-bits AES key through the first round attack using ≈ 350 samples, and two round attack using ≈ 80 samples in few seconds.

2009

Ristenpart et al. [35] considered side-channel leakage in virtualization environments on the example of the Amazon EC2 cloud service. They used Prime+Probe technique for analyzing the timing side-channel.

Tromer et al. experimentally demonstrated on real systems (OpenSSL, dm-crypt) in [43] attacks which exploit inter-process information leakage through the state of the CPU's memory cache. They also demonstrated an extremely strong type of attack which requires knowledge of neither the plaintext nor ciphertext.

2010

Bogdanov et al. introduced in [12] an advanced time-driven attack and analyzed it on an ARM-based embedded system.

Zhao et al. improved trace-driven attack on AES and CLEFIA in [24] by considering S-box mis-alignment. 200 samples were enough for them to obtain the full 128-bits AES key within seconds.

Rebeiro et al. justified in [34] that cache timing attacks on AES are unable to force hits in the third round and concluded thata similar third round cache timing attack does not work.

2011

Gallais et al. introduced in [19] an improved adaptive plaintext, and presented a new known plaintext trace-driven cache-collision attacks against embedded AES implementations. They show that with ≈ 30 known plaintexts, the skey space of AES 128-bits could be reduced to 2^{30}.

Gullash et al. presented in [21] the first practical access-driven cache attack on AES in the asynchronous model. They introduced a noveal approach by using neural networks to handle noise surrounding key candidates.

Suzaki et al. described in [41] the exploitation of an OS-optimization, namely KSM, to recover user data and subsequently identify a user from a co-hosted guest OS hosted in a Linux Kernel-based Virtual Machine (KVM).

Brumley et al. demonstrated in [15] that the ladder computation operation in the ECDSA computation is vulnerable to a timing attack. They showed that it was possible to steal the private key used for server authentication in a TLS handshake implemented using OpenSSL.

2012

Weiß et al. adapted in [45] Bernstein's attack in a virtualized environments. They considered the attack of a specifically desgined authentication protocol between an app running on a TEE and a remote backend server. They claim that by overtaking he untrusted world with specific malware, it is possible to break the authentication protocol using Bernstein's timing attack.

Zhang et al. described in [47] an access-driven side channel attack by which a malicious virtual machine (VM) extracts fine-grained information from a victim VM running on the same physical computer. This attack is the rst such attack demonstrated on a symmetric multiprocessing system virtualized using a modern virtual machine manager (Xen).

Mowery et al. proved in [28] that any cache timing attack against x86 processors that does not subvert the prefetcher, physical indexing and massive memory requirements of modern programs will fail.

2013

Yarom et al. proposed in [46] the Flush+Reload cache side-channel attack technique that enables them to extract the private key from a victim program running GnuPG 1.4.13 after only a single signing or decryption round. It exploits a weakness in the Intelx86 processors: page sharing exposes processes to information leaks.

Spreitzer et al. tried to show in [40] the applicability of Bernstein's time-driven attack and Bogdanov's cache-collision attack on 3 differents state-of-the-art mobile devices. Their results show that though Bernstein's attack requires an enourmouse number of measurement samples it remains a threat for cryptographic implementation on mobile devices. Regarding Bogdanov's attack it seems that a CPU with a cache-line size wider than 32 bytes exarcerbates their attack.

Aly et al. investigated Bernstein's timing attack in [6] and introduced an additional timing information which is usd to correlate the timing profiles obtained in the study phase with the timing profiles obtained in the attack phase. By combining the timing information proposed by Bernstein and the minimum timing information, they claim to recover the whole secret key without a signle exhaustive key-search computation.

2014

Apecechea et al. exposed in [7] vulnerabilities in virtualized cloud servers by mounting Cross-VM cache attacks in Xen and VMware VMs targeting AES running in the target VM.

Benger et al. described in [8] a cache timing attack to extract information from the OpenSSL ECDSA implementation. The authors demonstrate that with as little as 200 signatures they are able to recover a 256-bits secp256k1 elliptic curve key. This is notably the same curve used in the Bitcoin protocol.

Saraswat et al. investigated in [37] the applicability of Bernstein's timing attack against remote servers.

Spreitzer et al. proposed in [39] a shift from Bernstein's threshold-based approach for the selection of potential key candidates towards the application of the optimal key-enumeration algorothm. Their approach significantly reduces the complexity of the remaining key-search phase.

2.3 The different attacked algorithms/software

Different algorithms/software have been theoretically or/and practically attacked using Cache-based analysis techniques. The aim of this part is to understand how the different algorithms have been compromised. Detailed description of each algorithms is not provided in details in this part. Instead, a straight-to-the-point approach is given.

2.3.1 AES

The most attacked one is the AES (rijndael) [**?**] mainly due to its specific implementation.

2.3.1.1 AES optimization

There are a lot of different implementations of the AES. E.g: Bernstein's Poly1305-AES [10], Gladman's AES optmization [20], OpenSSL [1], etc.

However the most common implementation is the one that was initially proposed by Daemen and Rijmen in their intial proposal. They described how to implement a fast AES for 32-bits (and greater word length) processors in [18]. They proposed to merge SubBytes, ShiftRows and the MixColumns into 16 lookups from 4 different tables. Each table would contain 256 32-bits entries. Those tables are also known as "*T-tables*". A total of 4Kb of memory are then required. In case when the 4Kb table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bits table (1Kb) by the use of circular rotates.

2.3.1.2 Inferred Cache Attacks

Combining the knowledge about the "*T-tables*" implementations and the CPU caches, leads to the concept of cache attacks on the AES. Its "*T-table*" implementations make use of key-dependent look-up indices to access the precomputed values of the round transformations. Furthermore, the different accesses are not performed in constant time because of the cache behaviour: either the data could be fetched from the CPU cache (cache hit occured) or from the main memory (cache miss). This creates some variations in the execution times. In the literature, different approaches have been used to perform cache timing attacks against AES.

2.3.1.3 Cache Timing Attack on AES: The Origin

Bernstein's Cache Timing Attack [9]– His attack uses timing analysis to reconstruct an AES key based on cache timing information. Even if the cache size is large enough to contain the 4Kb lookup tables, he took in account that other processes may run concurrently to the victim's process.

Those processes certainly also need the cache. Therefore some cache misses can occur while the victim's process is requesting the lookup tables content.

The index values used to access to the aforementioned tables consist of input bits XORed with key bits. By varying the input bits, the encryption timing would also vary accordingly.

His attack consists of three phases:

- the study phase which involves measuring the encryption time of multiple plaintexts P under a known zero-key K. The sum of all encryption times observed for plaintexts where the plaintext byte $P_i = b$ is stored in $Sum_T[i][b]$. Furthermore the number of encrypted plaintexts is stored within $n_P[i][b]$.
- the next phase is the attack phase which consists in collecting the exact same information as previously gathered but under an unknown key $K^{'}$. The gathered information are stored in $Sum_T{}^{'}[i][b]$ and $n_P{}^{'}[i][b]$.
- finally, one has the correlation phase when ithe *plaintext-byte signature* (noted $sig[j][b]$ and $sig^{'}[i][b]$) of the study phase and the attack phase are analogously computed.

$$sig[i][b] = \frac{Sum_T[i][b]}{n_P[i][b]} - \frac{\sum_{i=0}^{j}\sum_{b=0}^{c} Sum_T[i][b]}{\sum_{i=0}^{j}\sum_{b=0}^{c} n_P[i][b]} \tag{2.1}$$

From Bernstein's work a lot of investigations, improvements, theoretical and practical experiments have been performed 2.1 .

Drawbacks

- Bernstein's attack requires reference measurements of encryption under known key in an identical configurationi as the victim which are not often readily available.
- Due to signal-to-noise ratio timing encryption seems impractical on many real systems
- The attack requires a huge number of samples (analyzed encryptions)

2.3.1.4 First round attack

2.3.1.5 Second round attack

2.3.1.6 Final round attack

2.3.1.7 Collision-based attacks

2.3.2 RSA

2.3.3 DES

2.3.4 ECDSA

2.4 The victim's environments

2.4.1 Single Threaded Versus Multithreaded CPUs

Single-threaded means that there is only one thread executing the whole work of a given process. Therefore, the process must wait for the current execution of the thread to complete before it can perform another action. On the opposite we have Multi-threaded aspect of CPUs that enables to have several threads running in parallel either on a single core or on multiple core.

Nowadays a lot of devices embed multicore processors and one can notice two different approach for optimizing the processor's performance. On one hand we have optimizations that focus on reducing the latency of individual threads hence enhance the processor's Single-Threaded performance. On the other hand the other approach focus on reducing the latency of the applications' threads taken as a group hence optimizes the processor's multithreaded performonce.

The advantage of a single-threaded approach is that it minimizes the execution times of individual threads but has lower power efficiency.

However the multithreaded approach has an higher power efficiency and is more adapted to highly parallel applications.

2.4.1.1 Parallelization is only a matter of Scheduling

One of the most important difference between the two approaches is the way threads are scheduled so that they can (almost) run in parallel.

In a Single-Threaded architecture, processors are only able to execute a single thread at any given point in time. Therefore the execution time is allocated in time slices to the different threads. By switching between threads, the Scheduler gives the impression of having multiple threads executed simultaneously. Let this type of execution called *quasi-parallel execution.*

On the opposite processors based on Simultaneaous Multi-threaded architecture split a single physical processor into two logical processors by duplicating some sections of the micro-architecture responsible for architectural state. In this way the Scheduler can manage two threads that litteraly run simulataneously in parallel on the same processor.

2.4.1.2 Consequences

In a Single-Threaded configuration, the attacker must execute his own code while the crypto process (the encryption) is in progress. To achieve this, he might exploit the interrupt mechanism by predicting the Real-Time Clock (RTC) or the timer interrupts and yield the CPU to the encrypting process a few cycles before such an interrupt.

During the interrupt, the Scheduler is invoked and normally the attacker's process should regain the CPU. Therefore he will be able to analyze the state of the cache to see what the encrypting process accessed during those few cycles.

Otherwise on a Simultaneous Multi-threaded processor, it would be easier to run attacks because the attacker and the crypto processes can run simultaneously on different virtual cores therefore the attacker process can monitor the cipher execution. Memory accesses of both execution threads alter the cache states at the same time.

In [43] Tromer et al. pointed out that on multi-core processors, the lowest-level caches are usually private to each core. However it might happen that the cryptographic code exceeds these private caches and reaches caches that are shared among the cores. An asynchronous cross-core level attack can then be applicable.

2.4.2 Cache Attacks within a Virtualized Environment

2.4.2.1 Virtualization Concepts

Nowadays virtualization has a huge impact in the IT and networking worlds.

In a nutshell one can have many forms of virtualization distinguished primarily by the computing architecture layer. The two main different kinds of Virtualization are for instance:

Hardware Virtualization makes an OS think it is running on its own hardware hence it abstracts the hardware from the OS.

E.g: Xen [1], VMware [2].

Application virtualization makes an application thinks it is running in its own OS hence it abstracts the services and kernel from an application.

E.g: VMware ThinApp [3]

In a VM context, one has more layers of isolation between attacker and victim than in a cross-process setting.

2.4.2.2 Challenging environment

An attacker will have to overcome some challenges in this particular environment:

- he will have to ensure that his VM will execute regularly despite the coarse scheduling quanta used by the underlying Virtual Machine Monitor [4] (VMM).

[1]http://wiki.xen.org/wiki/Main_Page
[2]www.vmware.com/
[3]www.vmware.com/products/thinapp
[4]Piece of computer software that creates and runs VMs

- like any side-channel, he will have to deal with overcoming sources of noise in his gathered samples. They will all come from the various hardware and software features that may run concurrently.
- it is not impossible that attacker and victim process do not run on the same core

2.4.2.3 Not too isolated though

As one can see in the cache attacks timeline 2.1 since 2009 some (not that much) attempts of cache attacks on virtualized envrionments have been performed. Their main objective is to demonstrate that the specific isolation characteristic of virtualization environments can be circumvented using cache timing attack techniques.

In a cross-vm attack context In [35] Ristenpart et al. focused on where third-party cloud computing gives attackers "*novel*" abilities. They assume that a malicious party can run and control many VM instances in the cloud on the same physical hardware as the victim's VM.

Therefore, the attacker might manipulate shared physical resources to gather private information.

Their work was followed by Zhang et al. in [47] that presented an access-driven cache attack that enables them for the first time to extract fine-grain finformation from the victim's VM.

Furthermore, recently Apcechea et al. evaluated in [7] different crypto libraries (OpenSSL, PolarSSL, libgcrypt) against Bernstein's correlation attack when run in the most popular VMs used by cloud service providers: Xen and VMware VMs.

In an ARM-based platform Weißet al. evaluated in [45] different AES implementations (Bernstein, Barreto, OpenSSL, Gladman, Niyaz) against cache timing attack on a testbed based on an embedded ARM SoC with an L4 microkernel as virtualization layer. They provided detailed results in their paper.

2.5 Timing measurement techniques

In cache-based side-channel attackis, one of the most important ability to have from an attacker point of view would be the ability to measure time during cache access.

Several measurement "*instruments*" exist that enable to differentiate the cycles between a cahe hit or miss. According to the target processor it is possible to get an high resolution time. Most of the time in the literature, they use the ability of using some specific instructions or registers to get an accurate count of cycles. Otherwise one can use an external timer which is obviously less accurate and introduce more noises.

RDTSC ...

VTSC ...

HARDCLOCK ...

ARM Performance Monitor Control Register ...

PERFCTR ...

External Timer ...

...

2.6 Cache information extraction techniques

2.6.1 Evict+Time

This technique was first introduced by Osvik et al. in [31]. This technique assumes that the attacker ahs the ability to trigger an encryption and know when it has begun and ended. He also should have the ability to get knowledge of the (virtual) memory address of each table he wants access to.

Let a lookup table T_l. Its virtual address is denoted $V(T_l)$. W is a given memory address and y is an index used to access an element from a given table T_l. δ is the cache line size and S and B are respectively the number of cache sets and the number of bytes a cache line can hold.

The Evict+Time technique consists of 3 steps:

- Trigger an encryption of a chosen plaintext P
- Evict data from some memory addresses congruent to $V(T_l) + y \times B/\delta \mod S \times B$
- Time the execution of a second encryption of P

Drawbacks Since this technique relies on timing the triggerend encryption oepration, it is very snesitive to noise from for instance the instruction scheduler, conditional branches, cache contention etc.

2.6.2 Prime+Probe

As the previous technique, this one was also presented by Osvik et al. It aims at discovering the set of memory blocks read by the encryption a posteriori, by examining the state of the cache after encryption. By performing a single encryption on a given plaintext P, they gather measurement scores (timing) for each elements of each tables.

This technique is split in three steps, but first one needs to allocate a contiguous byte array $A[0, ..., (S \times W \times B - 1)]$ with start address congruent mod $S \times B$ to the start address of T_0.

- Read a value from every memory block in A (Prime)
- Trigger an encryption of P
- Time the memory access of A such as for each table l and element indexes y respectively varying such as $l = 0, ..., 3$ and $y = 0, \delta, 2\delta, ..., 256-\delta$, the memory addresses of $A[1024l + 4y + tSB]$ are read with $t = 0, ..., W - 1$.

Drawbacks The main issue with the above method is that the attacker does not know where the victim's lookup tables reside in memory. In case the attacker has the knowledge of the layout of the victim's lookup tables, this would enable hilm to try each possible table offset in turn and then aply the one-round **??** attack assuming this offset. The offset with the maximal candidate score is then picked up.

Another complications would e the distinction between physical and virtual memory addresses. When both victim and spy's process use the same shared library, it is possible to get the knowledge of the physical and virtual memory addresses.

2.6.3 Flush+Reload

It was first porposed by Yarom et al. in [46] and is an extension of Gullash et al.'s work [21]. They demonstrate that due to a weakness in the Intel x86 processors, page sharing could expose processes to information leaks. Their attack aims at exploiting this weakness to monitor access to memory lines in shared pages by targeting the Last Level Cache (LLC).

Actually they noticed that the processor's instruction *cflush* evicts the memory line from all the cache levels, including from the shared LLC.

Unlike the prior attacks that target the First Level Cache, their attack does not require the attack program and the victim to share the same execution core.

They run their attack between two unrelated processes in two different environment configuration: in a single operating system and then in separate virtual machines.

Their attack enables them to recover on average 97% of the secret key bits by observing a single signature or decryption round.

The attack is split in three phases:

- The spy process must first monitor flushed memory line
- then it waits for the victim to access the memory line before
- finally, the spy reloads the memory line then measures the time to load it

2.6.4 Prime+Trigger+Probe

Ristenpart et al introduced in [35] this cache usage measurement technique variant of the Prime+Probe that has been described previously in 2.6.2.

Their technique support the setting of time-shared virtual machines, i.e the shared computing resources among the different virtual machines instances.

To perform their cache usage measurement, one should first allocate a contiguous buffer B that should be large enough to fill a significant portion of the cache. Let s be the cache line size. The three-steps-attacl is as follows:

- B is read at s-byte offsets in order to ensure it is cached
- a loop must keep the CPU busy until its cycle counter jumps by a large value
- finaly, the time it takes to read B at s-byte offset is measured

2.7 Existing countermeasures

Countermeasures	Comments
Do not use cache at all	fetching from RAM is slower
Constant-time crypto software	
Flush the whole cache on any context switch	
Avoid key-dependent table lookups	implies computational overhead
Make high-resolution timers (e.g: rdtsc) inaccessible to processes	problem is that a lot of software packages already use rdtsc
Preload certain data while a process wkaes up	only cache hits can then be detected. No information about the secret key can be infered. Not an efficient solution for huge tables
Avoid T-tables implementation for AES	computational overhead
use hardware crypto processor that does not use caching	
Use HSM with a hardened cache-architecture that provides constatn encryption	
Use random cache warming	
Use non deterministic access ordering	
use Non deterministic cache placement	
Add more restrictions on the cflush instruction	
Prevent page sharing usage	
Switch-off memory de-duplication	

There are several rules that can be followed to avoid timings that are dependent on secret data. The *cryptocoding.net* codingr rules `https://cryptocoding.net/index.php/Coding_rules` describe detailed steps that can be taken to avoid those issues.

2.8 Cache attacks quick overview/comparisons

Year	Attack	Enc.	Victim	Samples	Key recov.
2003	Time-driven	DES	UP/600 MHz Pentium III	2^{24}	56-bits
2004	Time-driven	AES	UP/850 MHz Pentium III	2^{27}	128-bits
2005	Access-driven	AES	SMP/Pentium 4E	$2^{13.8}$	128-bits
2005	Access-driven	AES	SMP/Athlon 64	$2^{18.9}$	128-bits
2005	Trace-driven	RSA	SMT/2.8GHz Pentium 4	310/512 bits	-
2006	Time-driven	AES	1GHz Pentium 3	$2^{14.58}$	60/128bits (1st round)
2006	Time-driven	AES	1GHz Pentium 3	2^{15}	128-bits(Final rnd.)
				2^{13}	128-bits (Expanded Final rnd.)
			3.5 GHz Pentium 4 Xeon	2^{16}	128-bits (Final rnd.)
				$2^{13.6}$	128-bits (Expanded Final rnd.)
			0.9Ghz UltraSparc	2^{16}	128-bits (Final rnd.)
2006	Trace-driven	AES	-	$2^{3.9}$	128-bits (Final rnd.)
2007	Time-driven	AES	SMT/3.06 GHz Xeon	2^{18}	128-bits
2009	Access-driven (Evict+Time)	AES	2Ghz Athlon 64	$2^{13.2}$	128-bits
2009	Access-driven (Prime+Probe)	AES	2Ghz Athlon 64	$2^{14.87}$	128-bits
2012	Time-driven	AES (Barreto)	720 MHz Cortex-A8	$2^{20.61}$	72/128-bits
		AES (OpenSSL)	720 MHz Cortex-A8	$2^{20.93}$	-
		AES (Bernstein)	720 MHz Cortex-A8	$2^{18.6}$	128-bits
		AES (Gladman)	720 MHz Corteix-A8	$2^{20.93}$	0 bit
		AES (Niyaz)	720 MHz Cortex-A8	$2^{20.93}$	0 bit
2013	Trace-driven (Flush+Reload)	CRT-RSA	Core i5-3470	$2^{16.6}$	
			Xeon E5-2430	$2^{16.6}$	
2013	Time-driven	AES	1.3 GHz Cortex-A9 (Acer Iconia Tab A510)	2^{30} (study phase) 2^{27} (attack phase)	55/128-bits
			1Ghz Cortex-A8 (Google Nexus S)	2^{30} (study phase) 2^{29} (attack phase)	50/128-bits
			1.4Ghz Cortex-A9 (Samsung galaxy s3)	2^{30} (study phase) 2^{30} (attack phase)	67/128-bits

2.9 Cache attacks: open problems

2.9.1 Access-based cache attacks

- how to induce regular and frequent execution of the attacker's process/VM despite the hard-to-predict scheduler
- how to overcom sources of noise in the information available via the cache timing channel
- how to deal with core migrations which give rise to cache "readings" with no information of interest to the attacker. (attacker and victim use different cores)

2.9.2 Time-driven cache attacks

Timing attacks employ a threshold-based approach. This means that one fixes a threshold on the computed correlations and considers sub-key values as potential candidates iff the corresponding correlation is larger than the threshold. One may use different thresholds for the different sub keys, either because a profiling phase has shown different behaviors for different sub-keys because they are dynamically computed.

The threshold approach is simple to implement but has two major drawbacks:

- the actual key may not be found if one of its sub-key values led to a small correlation, then the key will never be tested in the search-phase and thus, the attack will provide no advantage over exhaustive search
- a loss of information since the ordering of kept sub-key values is not exploited in the search phase.

2.9.3 The case of mobile devices

- A thorough analysis of the noise behavior has not yet been done

Chapter 3

Cache Attacks Applied to Smart Cards

3.1 Cache Timing Attacks Versus Embedded Security

It is already well known nowadays that side-channel attacks are serious threat to embedded systems such as smart cards. Applied to symmetric key algorithms, the attacker was mainly limited to techniques based on information leakage via power consumption [?] and electromagnetic radiation [?].

Meanwhile one can notice that timing analysis was mainly applied to desktop and server PCs. This is due in part to the fact that lightweight embedded systems implement hardware implementations of symmetric key algorithms. Furthermore many lightweight platforms based on 8-bits or 16-bits CPUs run at very low frequencies. Consequently microarchitectural performance optimization such as caches are not necessary.

However as nowadays applications require more and more computing power, vendors decided to use 32-bits RISC ARM-based CPUs as a standard choice. Therefore, as a consequence, more and more security related features are being implemented in software instead of in hardware. It seems that ARM microprocessors do have cache memory and are therefore vulnerable to cache timing attacks.

Chapter 4

Cache Attacks Applied to Mobile Devices

4.1 Spreitzer et al.

As far as I know, in the literature, only Spreitzer et al. evaluated cache attacks on mobile devices [40].

They particularly show that T-table based implementations of the AES leak enough timing information on these devices so it is possible to recover parts of the secret key.

Instead of using lab testbeds, they run their attack on state-of-the-art Android-based mobile devices.

4.1.1 It almost works but...

Their work shows that it is possible to gather some possible key candidates using Bernstein's correlation phase though. Therefore one clearly observes that timing information is leaking. However as they claimed, the number of remainingkey bits is still too large for an exhaustive key search.

The remaining key-search phase corresponds to 2^{58} AES encryptions which is impractical. They tried to consider a first-round attack but the AES key space could not be reduced enough in order to perform an efficient exhaustive key search.

4.1.2 Forcing cache evictions in a realistic scenario does not help much

In order to trigger cache misses, an attacker needs to find a way to perform cache evictions. i.e memory accesses at constant cache locations must be performed.

The authors tried to mount their attack in a realistic scenario where applications (videos, slideshow) were launched on the mobile devices in order to trie to affect the cache. But it did not help in leaking more information and hence did not further reduce the key space.

4.1.3 Follow ups on their work

Second-round Attack It would be interesting to investigate a second-round attack on AES as suggested by Neve in its PhD thesis [29].

Collision attack Cache-collision attacks focus on the exploitation of collisions between lookup indices of intermediate state bytes. It is then possible to infer relations between key bytes. The authors observed on the ARM Cortex-A8 that the detection of diagonals that can lead to wide collissions is a challenging task.

They started with a reduced 3-round AES implementations and they succeeded in dissociating the encryption times of plaintexts that led to wide collisions and those that did not.

They also tried the attack on a 7-round AES implementation and they observed that the previous results were not anymore observable.

Bibliography

[1] Openssl's aes implementation.

[2] Onur Aciiçmez. Yet another microarchitectural attack:: Exploiting i-cache. In *Proceedings of the 2007 ACM Workshop on Computer Security Architecture*, CSAW '07, pages 11–18, New York, NY, USA, 2007. ACM.

[3] Onur Aciiçmez, Werner Schindler, and Çetin K. Koç. Improving brumley and boneh timing attack on unprotected ssl implementations. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 139–146, New York, NY, USA, 2005. ACM.

[4] Onur Aciicmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In *Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'07, pages 225–242, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] Onur Acimez, Werner Schindler, and etin K. Ko. Cache based remote timing attack on the aes. In *Topics in Cryptology CT-RSA 2007, The Cryptographers Track at the RSA Conference 2007*, pages 271–286. Springer-Verlag, 2007.

[6] Hassan Aly and Mohammed ElGayyar. Attacking aes using bernstein's attack on modern processors. In *AFRICACRYPT'13*, pages 127–139, 2013.

[7] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Fine grain cross-vm attacks on xen and vmware are possible! Cryptology ePrint Archive, Report 2014/248, 2014.

[8] Naomi Benger, Joop van de Pol, NigelP. Smart, and Yuval Yarom. ooh aah... just a little bit : A small amount of side channel can go a long way. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 75–92. Springer Berlin Heidelberg, 2014.

[9] Daniel J. Bernstein. Cache-timing attacks on aes. Technical report, 2005.

[10] Daniel K. Bernstein. Poly1305-aes for generic computers with ieee floating point, 2005.

[11] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. Aes power attack based on induced cache miss and countermeasure. In *ITCC (1)*, pages 586–591. IEEE Computer Society, 2005.

[12] Andrey Bogdanov, Thomas Eisenbarth, Christof Paar, and Malte Wienecke. Differential cache-collision timing attacks on AES with applications to embedded cpus. In *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, pages 235–251, 2010.

[13] Joseph Bonneau. Robust final-round cache-trace attacks against aes, 2006.

[14] Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against aes. cryptographic hardware and embedded systems. In *Lecture Notes in Computer Science series 4249*, pages 201–215. Springer, 2006.

[15] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *Proceedings of the 16th European Conference on Research in Computer Security*, ESORICS'11, pages 355–371, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[17] Anne Canteaut, Cédric Lauradoux, and André Seznec. Understanding cache attacks. Research Report RR-5881, 2006.

[18] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[19] Jean-François Gallais, Ilya Kizhvatov, and Michael Tunstall. Improved trace-driven cache-collision attacks against embedded aes implementations. In *Proceedings of the 11th International Conference on Information Security Applications*, WISA'10, pages 243–257, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] Brian Gladman. Gladman's aes implementation, 2008.

[21] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games – bringing access-based cache attacks on aes to practice. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 490–505, Washington, DC, USA, 2011. IEEE Computer Society.

[22] Mark Donald Hill. *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, EECS Department, University of California, Berkeley, Nov 1987.

[23] Wei-Ming Hu. Lattice scheduling and covert channels. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, SP '92, pages 52–, Washington, DC, USA, 1992. IEEE Computer Society.

[24] Xin jie ZHAO and Tao WANG. Improved cache trace attack on aes and clefia by considering cache miss and s-box misalignment, 2010. zhaoxinjieem@163.com 14645 received 2 Feb 2010, last revised 5 Feb 2010.

[25] Xin jie Zhao, Tao Wang, Dong Mi, Yuanyuan Zheng, and Zhaoyang Lun. Robust first two rounds access driven cache timing attack on aes. In *CSSE (3)*, pages 785–788. IEEE Computer Society, 2008.

[26] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *JOURNAL OF COMPUTER SECURITY*, pages 97–110. Springer-Verlag, 1998.

[27] Cdric Lauradoux. Collision attacks on processors with cache and countermeasures, 2005.

[28] Keaton Mowery, Sriram Keelveedhi, and Hovav Shacham. Are aes x86 cache timing attacks still feasible? In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 19–24, New York, NY, USA, 2012. ACM.

[29] Neve. Cache-based vulnerabilities and spam analysis, 2006.

[30] Michael Neve, Jean-Pierre Seifert, and Zhenghong Wang. A refined look at bernstein's aes side-channel analysis. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 369–369, New York, NY, USA, 2006. ACM.

[31] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[32] D. Page. Theoretical use of cache memory as a cryptanalytic side-channel, 2002.

[33] Colin Percival. Cache missing for fun and profit. In *Proc. of BSDCan 2005*, 2005.

[34] Chester Rebeiro, Mainack Mondal, and Debdeep Mukhopadhyay. Pinpointing cache timing attacks on aes. In *VLSI Design*, pages 306–311. IEEE, 2010.

[35] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[36] Robert G Salembier. Analysis of cache timing attacks against aes.

[37] Vishal Saraswat, Daniel Feldman, Denis Foo Kune, and Satyajit Das. Remote cache-timing attacks against aes. In *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*, CS2 '14, pages 45–48, New York, NY, USA, 2014. ACM.

[38] Werner Schindler. A timing attack against rsa with the chinese remainder theorem. In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, pages 109–124, London, UK, UK, 2000. Springer-Verlag.

[39] Raphael Spreitzer and Benoit Grard. Towards more practical time-driven cache attacks. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings.*, volume 8501 of *Lecture Notes in Computer Science*, pages 24 – 39. Springer, 2014.

[40] Raphael Spreitzer and Thomas Plos. On the applicability of time-driven cache attacks on mobile devices (extended version). *IACR Cryptology ePrint Archive*, 2013:172, 2013.

[41] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory deduplication as a threat to the guest os. In *Proceedings of the Fourth European Workshop on System Security*, EUROSEC '11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.

[42] Kris Tiri, Onur Aciiçmez, Michael Neve, and Flemming Andersen. An analytical model for time-driven cache attacks. In *Proceedings of the 14th International Conference on Fast Software Encryption*, FSE'07, pages 399–413, Berlin, Heidelberg, 2007. Springer-Verlag.

[43] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptol.*, 23(2):37–71, January 2009.

[44] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, and Maki Shigeri. Cryptanalysis of des implemented on computers with cache. In *Proc. of CHES 2003, Springer LNCS*, pages 62–76. Springer-Verlag, 2003.

[45] Michael Weiss, Benedikt Heinz, and Frederic Stumpf. A cache timing attack on aes in virtualization environments. In *14th International Conference on Financial Cryptography and Data Security (Financial Crypto 2012)*, Lecture Notes in Computer Science. Springer, 2012.

[46] Yuval Yarom and Katrina Falkner. Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 719–732, Berkeley, CA, USA, 2014. USENIX Association.

[47] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 305–316, New York, NY, USA, 2012. ACM.