Attack of the Knights: A Non Uniform Cache Side-Channel Attack

Farabi Mahmud
Texas A&M University
farabi@tamu.edu

Sungkeun kim
ksungkeun84@tamu.edu
Texas A&M University

Harpreet Singh Chawla
harpreetsc@tamu.edu
Texas A&M University

EJ Kim
ejkim@tamu.edu
Texas A&M University

Chia-Che Tsai
chiache@tamu.edu
Texas A&M University

Abdullah Muzahid
abdullah.muzahid@tamu.edu
Texas A&M University

ABSTRACT

For a distributed last-level cache (LLC) in a large multicore chip, the access time to one LLC bank can significantly differ from that to another due to the difference in physical distance. In this paper, we successfully demonstrate a new distance-based side-channel attack by timing the AES decryption operation and extracting part of an AES secret key on an Intel Knights Landing CPU. We introduce several techniques to overcome the challenges of the attack, including the use of multiple attack threads to ensure LLC hits, to detect vulnerable memory locations, and to obtain fine-grained timing of the victim operations. While operating as a covert channel, this attack can reach a bandwidth of 205 KBPS with an error rate of only 0.02%. We also observed that the side-channel attack can extract 4 bytes of an AES key with 100% accuracy with only 4000 trial rounds of encryption.

ACM Reference Format:
Farabi Mahmud, Sungkeun kim, Harpreet Singh Chawla, EJ Kim, Chia-Che Tsai, and Abdullah Muzahid. 2023. Attack of the Knights: A Non Uniform Cache Side-Channel Attack. In *Proceedings of Annual Computer Security Applications Conference (ACSAC) (ACSAC 2023).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The ever-shrinking feature size in CMOS process technology has enabled the integration of a large number of cores and caches onto a single chip [11, 31]. Large-scale multicores are equipped with a large last-level cache (LLC) to alleviate expensive off-chip accesses. As an example, AMD Ryzen comes with a 256 MB LLC whereas Intel Xeon Phi 7200 series has a 36 MB LLC [1, 3]. Such an LLC is distributed over multiple banks connected through a network-on-chip (NoC) to reduce access latency and improve core isolation, referred to as a Non-Uniform Cache Access (NUCA) architecture. With a distributed LLC, memory-level parallelism is improved by allowing concurrent requests to different banks. However, on the negative side, access latency to different banks may vary depending on the distance from the requesting core as seen in Figure 1. In this paper, we set out to investigate whether this latency difference can lead to security vulnerabilities in large-scale multicore machines.

When it comes to security vulnerabilities related to caches, side-channel attacks are the most notorious ones. Numerous cache side-channel attacks [5, 12, 39, 47, 53, 54, 57, 62, 67] were discovered and shown to be significant threats to data security, especially during the past few years. The most common form of cache side-channel attacks involves timing the cache access latency which depends on the state of the target cache lines. Take Flush+Reload [24, 26, 34, 68, 73, 74, 77] for example. The attacker tries to observe the target shared lines being accessed by the victim program, by detecting whether reloading the line incurs a cache hit or miss. Most existing cache-based side-channel attacks exploit the timing difference caused by cache states. Numerous defense techniques [22, 30, 33, 40, 43, 46, 58] have been proposed to eliminate such timing differences. Furthermore, most of the existing cache side-channel attacks have not been demonstrated or explored in Non-Uniform Cache Access (NUCA) architecture. Recently, Dai et al. [17] showed how network contention in NUCA architecture can be utilized to create timing differences and subsequently, a covert and side-channel. As we see more and more large-scale processor chips [2, 3, 7, 18, 38, 51, 66], attackers will turn their attention to similar attacks which have the potential to circumvent prior defenses. However, as these multicore chips scale up, contention-based NUCA attacks will become more difficult due to noises in the on-chip communication. In this paper, we showed that under such circumstances, there is still a possibility of a new side-channel attack in large-scale multicore chips that rely on physical distances in cache banks as opposed to network contention.

In this paper, we demonstrate a novel distance-based side-channel attack in large-scale NUCA architecture that may exist even when other cache-based side-channel attacks have been rendered ineffective. As a simplified case, let us consider Algorithm 1. It shows the pseudo-code of a side-channel attack that relies on physical distance in a NUCA machine. In this example, an address is determined by the victim according to a specific bit of a secret. The address can be mapped to the LLC bank of the nearest core when the bit is 0, or the farthest core when the bit is 1. Therefore, by timing the access latency, an attacker can infer the secret bit.

To further illustrate this scenario, Figure 1 shows access latency to the same address from different cores. We collected these latency numbers from the Intel Xeon Phi 7290 CPU. This CPU model belongs to Intel's Knights Landings line and has a multicore architecture with at least 64 cores (the CPU we tested has 72 cores). The figure shows that the LLC hit latency for the same address has a range between 280–350 cycles, and this pattern is generally stable

ACSAC 2023, December 2023, Austin, Texas, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

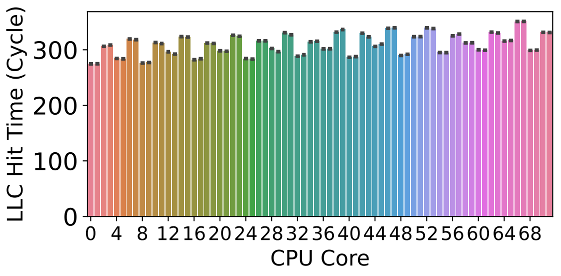


Figure 1: LLC hit time measured in cycles when accessing the same physical address (`0x1000000000`) from different cores (core 0 to 71) in the Intel Xeon Phi 7290 CPU. The latency numbers are averaged over 10,000 samples.

across cores. If an attacker can measure the access latency in the victim code (as shown in Algorithm 1), he/she will be able to guess the secret bit by telling whether the addresses fall into a near or far LLC bank. We refer to this as NUCA distance-based side-channel.

We demonstrate this side-channel in Intel Xeon Phi 7290 using AES code. We have addressed two major technical challenges for this attack, namely (i) the overlapping of memory accesses to LLC banks, and (ii) the difficulty of timing AES operations. To address these challenges, we have employed different techniques. We have used AdaBoost [29] model to differentiate between favorable and unfavorable cases in the presence of overlapping LLC accesses. With multiple samples and majority voting, we get 100% confidence in predicting the vulnerable access pattern as described in §6.3.2. To efficiently time only the region of interest within the decryption function, we have utilized multiple attack threads and time part of the AES decryption operation. Our proof-of-concept (PoC) attack code can accurately extract 4 bytes of any AES key with 4,000 decryption trials using a sequence of random plaintexts. We disclosed this vulnerability to Intel Product Incident Response Team (PSIRT). The team subsequently confirmed this vulnerability.

It should be noted that our attack is different from prior attacks. For example, in the case of Prime+Probe and Dai et al., because the attacks depend on sharing resources, the attack can be mitigated by partitioning or clustering the resources. Dai et al. can also be mitigated through software defenses by exploiting specific scheduling patterns in NoC routers. However, NUCA distance-based side-channel attacks cannot be completely mitigated by partitioning or clustering as long as the attacker is able to time the victim. The success of the attack depends on the capability of the attacker to time the victim, not the sharing of resources.

For responsible disclosure, we have revealed the attack details and provided the PoC attack code to Intel and received acknowledgment from them.

In summary, we make the following contributions:

- We demonstrated a *new* distance-based side-channel attack on NUCA architecture on an Intel Knights Landing CPU against a vulnerable AES implementation. The proof-of-the-concept code can accurately extract the 4 bytes of the AES key with only 4,000 decryption trials using a sequence of random plaintexts.

Algorithm 1: Pseudo code of a victim function which is vulnerable to a side-channel based on the difference of LLC access time due to distance.

```
Input: BitMask
Data: Secret
1 if (Secret & BitMask) == 1 then
2     $Addr = Addr_{Near}$ mapped to the LLC bank of the nearest core;
3 else
4     $Addr = Addr_{Far}$ mapped to the LLC bank of the farthest core;
5 Load($Addr$);
```

- We identified and addressed two technical challenges to performing a robust NUCA distance-based side-channel attack.
- We have developed several techniques including the use of a machine learning model to classify latency that consists of multiple cache accesses, use of a separate timing thread for fine-grained timing of the victim operations, and use of a remote thread to force L1D misses but LLC hits for the target cachelines.

The rest of the paper is organized as follows: §2 explains The necessary background of NUCA architecture, different cache side-channel attacks, AES algorithm; §3 explains the details of the target architecture; §4 explains the assumptions for the attack model; §5 gives detailed examples of attacks in a microarchitectural simulator and a real machine and shows other vulnerable algorithms; §6 shows different experiments to prove our claims; Finally, we discuss some possible defenses in §7.

2 BACKGROUND & RELATED WORKS

In this section, we describe the background of non-uniform cache access (NUCA) architecture, cache-based side-channel attacks, and NoC-based side-channel attacks.

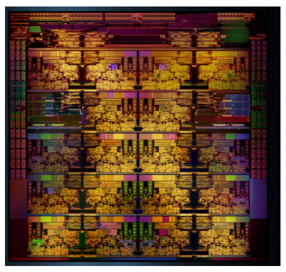
2.1 Non-Uniform Cache Access Architecture

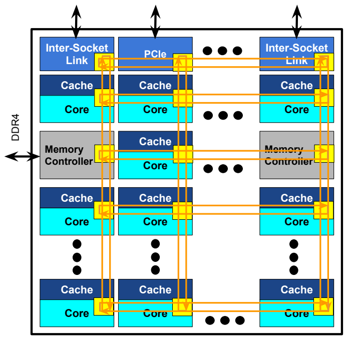
The Non-Uniform Cache Access (NUCA) Architecture is designed to reduce the memory access time by increasing the cache capacity and thus increasing the cache hit rate [1, 36, 52]. However, as the bandwidth and the number of ports are limited for individual physical cache banks, multicore processor chips adopt the design of having physically separated banks for shared last-level caches (LLC), which are interconnected with network-on-chip (NoC). As a result, cache access latency to different cache banks in a NUCA architecture shows disparity [41], as the memory operations involve network protocols and messaging within NoC.

In this paper, we particularly focus on the Mesh-based interconnect topology of the Skylake-SP and Skylake-X processors of Intel, which have 28 cores [51] as shown in Figure 2, and AMD's 32-core EPYC processors [60]. These processors use a different NoC design as the traditional ring-based architecture of Intel processors, and thus exhibit completely different access latency patterns from prior Intel processors.

2.2 Cache Side-channel Attacks

Existing cache side-channel attacks exploit the fact that the internal states of CPU caches, including TLB occupancy, cache eviction, cache replacement states, and memory controller buffers can be observed in the microarchitecture [21, 54, 71, 72]. The attackers that


(a) Intel Skylake Die.



(b) Intel Skylake Mesh Network.

Figure 2: NUCA Architecture in Intel's Skylake system. In the architectural diagram (b), each cache block consists of private L1, L2 and a shared L3/LLC bank.

use these side channels often time the latency of memory operations after the victims have accessed the TLB or the cache, and then use the information to infer the behaviors of the victims or even the secrets inside the victim programs.

For a side channel, the sender (or the "victim") and the receiver (or the "attacker") use the observable states of a system to encode and decode the information that they intend to communicate. Such communication typically requires a certain level of synchronization, especially if the sender and the receiver are communicating more than one bit. Synchronization is required between the sender sending the information by affecting the states of the system, and the receiver receiving the information by detecting the state changes and decoding the information. Between each bit being transmitted with the side channel, the participants also need to reset/refresh the state of the system because the receiver cannot distinguish the state changes caused by transmitting multiple bits from the sender.

Below, we explain some of the existing cache side-channel attack techniques, and how they transmit information:

- **Prime+Probe [5, 12, 39, 47, 53, 54, 57, 62, 67]:** This attack exploits the observable eviction in the shared cache when CPU needs to replace one of the attacker's cache lines with the cache line of the victim. To ensure the observable eviction, the attacker needs to first *prime* part of or the entirety of the cache, so that cache replacement will occur on one of the attacker's cache lines.
- **Flush+Reload [24, 26, 34, 68, 73, 74, 77]:** This attack depends on the attacker and the victim sharing the same data in the memory, and thus when the CPU brings the data into the cache for the victim, the attacker can access the same cache line afterward. Given that the cache line may be previously brought in for the attacker, the attacker will use instructions like CLFLUSH to remove the cache line prior to the potential operations in the victim. The sharing of data between the attacker and the victim can be a result of shared libraries or physical memory deduplication.
- **Evict+Reload:** Similar to Flush+Reload, Evict+Reload depends on the attacker and the victim sharing the same cache lines. The only difference is that the attacker first performs multiple memory accesses to evict the target cache line, instead of flushing the cache line using CLFLUSH. This technique is useful when CLFLUSH is not available on the specific architecture.
- **Flush+Flush [23]**: Flush+Flush also depends on the attacker and the victim sharing the cache line within the CPU, yet it exploits the difference of access latency due to the cache coherence protocol. In Flush+Flush, the attacker times the CLFLUSH instruction, aside from using it as a measure of resetting the cache states. If the target cache line was never accessed by a remote CPU core and thus was never brought into another private cache, the access latency will be much lower than that of the scenario in which the cache line is held in another private cache.

The mitigations of these side-channel attacks are mostly based on two approaches. The first approach prevents the attackers from observing the information decoded inside the timing of cache operations, such as adding extra latency to cache access [19, 75]. The second approach prevents the attacker and the victim from sharing resources such as the last-level cache and thus eliminates the medium which they can use for communication [42, 43, 46, 69].

2.3 Other NoC-based Side-Channel Attacks

In NUCA architecture, NoC has been used by attackers to detect the access latency after the eviction or while fetching cache lines, similar to Prime+Probe [59]. In order to mitigate such a side channel, prior work [59] proposes obfuscating the NoC access patterns by swapping routing algorithms when the attack is suspected.

Another type of explored side channels in NoC is the exploitation of NoC and cache contention, especially in a Ring interconnect topology [55]. These side channels exploit the interference of NoC traffics, and thus can be mitigated by isolating the network traffics of different processes or CPU cores [70].

2.4 Comparison with Existing Attacks

To show the difference between our NUCA distance-based side-channel attack and the existing cache side-channel attack (we use Prime+Probe [57] as an example) and NoC-based attack described (Dai et al. [17] as an example), we compare these attacks according to their attack models, attack assumptions, and the possible defenses.

Attack Models: The main difference between these attacks is in the way the sender (the "victim") transmits information to the receiver (the "attacker"). The cache side-channel attacks such as Prime+Probe primarily transmit information by evicting or populating specific cache lines. For the NOC attack by Dai et al., the information is transmitted through the interference of traffic on the shared NoC rings inside a ring-based Mesh interconnect. On the other hand, our attack does not transmit information through changes of cache states or interference, but rather through directly timing the memory access operations inside the victim to infer the secret from the difference in cache latency. This difference is fundamental to how this attack is performed as well as the possible defense to the attack.

Attack Assumptions: For the existing cache side channels (e.g., Prime+Probe), the attacker and the victim needs to share certain resources and as a result, causing the victim's behavior to be observable by the attacker. The NoC attack by Dai et al. requires the attacker to interfere with the traffic of the victim. Our NUCA attack does not require any contention or interference, but does require the attacker to be able to time the victim operations directly. In addition, we show that, end-to-end latency of victim operations can be too coarse-grained for the attacker to infer the victim's secret accurately, so the attack will require fine-grained timing information about the latency of individual memory access or at least groups of memory access. This is one of the important hurdles that the attacker needs to clear in order to exploit this side channel.

Possible Defenses: The existing cache side channels are typically mitigated by either un-sharing the resources or obfuscating the timing in the CPU cache. For the NoC attack by Dai et al., interference of NoC traffic can be mitigated by changing the scheduling policies within the NoC or isolating the NoC channels between the attacker and the victim. For our NUCA attack, the only effective way is to block the attacker from observing any meaningful patterns in the timing of the victim operations. Along the same line of thinking, oblivious RAM (ORAM) [65] can be used to remove the access patterns completely, and thus can be used to mitigate all three attacks simultaneously.

2.5 Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is a block cipher that is widely used. This was initially proposed in 1997 and later adopted as the standard by the National Institute of Standards and Testing [16]. Some ISAs started including modified instructions in their instruction sets to accommodate AES operations [25].

AES can have 128-bit, 192-bit, or 256-bit keys and can comprise of 10 rounds, 12 rounds, or 14 rounds of encryption/decryption respectively. Each round can be broken into different steps. These include -

(1) KeyExpansion - Different keys used in each round are calculated from the original cipher key using the predetermined AES Key Schedule
(2) AddRoundKey - During this step, Round Key is XOR'ed with the current state
(3) SubBytes - A lookup table is used to nonlinearly substitute each byte with some specific values
(4) ShiftRows - Last three rows of the state are shifted cyclically
(5) MixColumns - Mixing operation that combines four bytes in each column

In the first round the KeyExpansion and AddRoundKey steps are performed. In the next consecutive 9, 11, or 13 rounds (based on the number of bits in the cipher key), SubByte, ShiftRow, MixColumn and AddRoundKey steps are done. In the final round, only the SubByte, ShiftRow and AddRoundKey operations are executed. In our attack, we target the last round of decryption.

3 TARGET ARCHITECTURE DETAILS

In this section, we explain the architectural details of the target architecture that we have used to implement the attack. We also discuss how this makes other architectures vulnerable as well. The Intel Xeon Phi 7290 is based on the Intel Knights Landing family that was launched in 2016. Many HPC servers and cloud providers are still using Knights Landing processors in high performance computing including Cori [18] from National Energy Research Scientific Computing Center, Stampede-2 [64] from the University of Texas, Trinity [48] from Oak Ridge National Laboratory or Nurion [38] from Korea Institute of Science & Technology Information. However, the principle that is exploited in this attack, the latency difference due to mesh on chip network, is carried into current and future generations of Intel Skylake-SP [51], Cascade Lake [7] and even newer processors.

3.1 Memory Configuration

3.1.1 MCDRAM Configuration. To accommodate computationally intensive programs, Intel Knights Landing series processor started including high bandwidth on-package memory in the form of Multi Channel DRAM (MCDRAM). MCDRAM is a high bandwidth memory that supports multiple channels at the same time. Even though this memory is physically located within the same package, it is situated on a separate piece of silicon die. In Intel Xeon Phi 7290 we have 16Gb of MCDRAM on-package. We have three different memory modes depending on the usage of the MCDRAM. These three modes are -

(1) **Cache Mode** - MCDRAM used only as shared Cache memory 100% cache, 0% memory
(2) **Flat Mode** - MCDRAM used as 100% memory and 0% cache. No separate DDR memory is required.
(3) **Hybrid Mode** - MCDRAM is used as 25%/50% cache and the rest is used as memory.

In our experiments, we have used the MCDRAM in flat modes, however, this can be extended to any memory mode. The flat mode configuration of the MCDRAM uses L2 Cache as the last level cache. Since the attack depends on different L2 hit latencies, having more levels of cache would impact its feasibility. Therefore, we chose the flat mode configuration to simplify the experiment. However if we choose other modes, it will require careful calibration of latency thresholds. We leave the exploration of our attack in different memory modes for future works.

3.2 LLC Organization

The CPU has a floorplan shown as Figure 3, where its 72 physical cores. (or 288 physical threads with hyperthreading) are distributed across 38 tiles [32, 63]. It is known that not all tiles have active physical cores on them, and the physical CPU IDs—the IDs which are typically obtained through the Advanced Configuration and Power Interface (ACPI) and are recognized by OS—are arbitrarily assigned to a tile in an order which tends to alternate between the four quadrants. For our attack, we do not need to explicitly know the tiles mapping, however this is obtainable using methods shown in previous works [32].

3.2.1 MESIF Coherence Protocol. The CPU employs a directory-based cache coherence mechanism using the MESIF protocol [20] with a distributed directory system. Each tile includes a Caching/Home Agent (CHA) in charge of a portion of the directory. Two CPUs sharing the same tile use the same CHA and LLC. However, each CPU

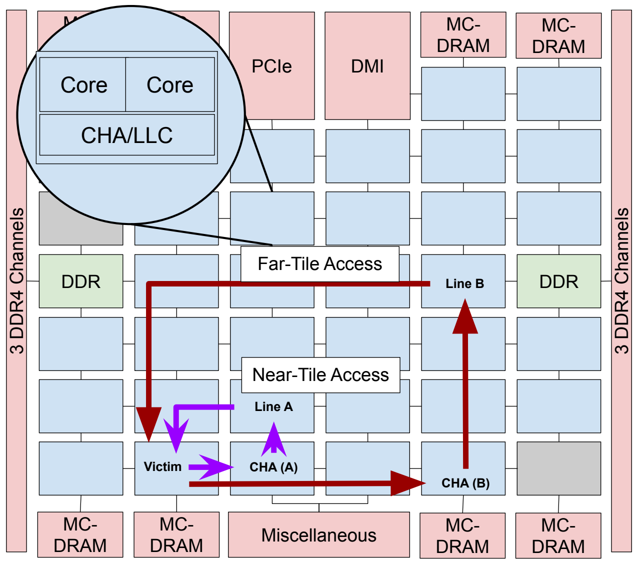


Figure 3: Knights Landing Floorplan Block Diagram [32]. Blue rectangles denote active tiles. Grey rectangles denote tiles with disabled cores. One tile is zoomed to show that it contains two cores and a CHA/LLC

has its own private L1 cache. Since, Intel Xeon Phi 7290 is configured in flat memory mode, L2 acts as the private (but shared among two cores on same tile) last level cache.

3.2.2 Resolving a LLC Request. Each time a core requests a cache line due to an L1 miss, a corresponding CHA (distributed tag directory) is queried based on the line address. If the cache line is present in the LLC bank of a tile, the CHA will instruct the tile to forward the data to the requester. Thus, two sources of latency contribute to the difference in LLC hit times - one due to difference in distance to the CHA location, and the other due to difference in distance to the forwarding tile. Even if two cache lines reside in the same forwarding tile, their LLC hit times can differ if two different CHAs handle the cache lines.

3.2.3 Clustering Mode. There are different clustering modes of the Intel Xeon Phi 7290 including All-to-All, Quadrant Clustering and Sub NUMA Clustering (SNC) modes.

(1) **All-to-All** In this mode, the overall address space is uniformly distributed using the hash function across all the tiles. So, an LLC request originating from any tile can have the required CHA in any of the other tiles. There are also no restrictions on which memory controller can access which set of tiles.
(2) **Quadrant/Hemisphere** In this mode, the memory request may originate from any tile. However, the tag directory *must* be located in the same quadrant/hemisphere of the memory controller. There exists a single address space shared across all the memory channels.
(3) **Sub NUMA Cluster (SNC-2/SNC-4)** This mode converts the processor to a 4 node NUMA(SNC-4) or 2 node NUMA(SNC-2) configuration. Each cluster has its address space interleaved within the quadrant (in SNC-4) or the hemisphere (in SNC-2). Memory request originating from the same cluster is served by the tag directory, memory controller within the same quadrant/ hemisphere. NUMA aware software can use this mode where each thread is pinned to a specific hemisphere(2)/quadrant(4).

We configure to an *All-to-All* cluster mode where a request can traverse the entire mesh to contact the tag directory, then forward it to the proper memory controller to fetch the required cache line. Both Quadrant/Hemisphere and SNC-2/SNC-4 exhibit lower differences in LLC Hit latencies between the nearest and the farthest nodes than the All-to-All mode. Quadrant/Hemisphere mode has lower difference because Tag Directory (TD) should be co-located in the same quadrant/hemisphere of the requestor, limiting the distance needed to travel by the request. On the other hand, SNC-2/SNC-4 mode shows even lower difference because the address space is limited within the cluster. That is why, we choose *All-to-All* mode. Both the cluster mode, and MCDRAM configuration can be obtained by using `numactl` application without any `root` privileges.

4 ASSUMPTIONS & THREAT MODEL

In this paper, we assume that the attacker's target is a NUCA architecture with a multi-hop mesh interconnection network. The attacker can identify vulnerable access patterns either through profiling, if they do not have access to the source code, or by recognizing it within the source code itself. Unlike other side-channel attacks based on contention within the NUCA architecture, the distance-based attack requires measuring the execution time of the victim functions that are known to access data in different cache banks in NUCA.

Attacker needs to know hardware configurations such as CPU Model Number, Cache Hierarchy and, MCDRAM configuration, and Clustering Mode. Note that all of these can be obtained from `/sys, /proc` and `numactl` without root privileges. The attacker can time the sensitive operations in the victim program by interacting with the victim, such as exchanging messages through the network or inter-process communication (IPC), detecting changes in shared variables, or exploiting other side-channels. In case of attacks using a shared variable, the attacker needs read permission on the shared variable through some libraries (like shared AES library) similar to previous attacks [13, 23, 71, 72, 74].

Restrictive defense mechanisms like not allowing hyperthreading of processes of different security domains can be assumed [49, 76]. However, attacker needs to be able to launch threads to different physical cores other than the victim's core. For the covert channel, we assume that the attacker (spy) and the victim process are strictly cooperating, so there is no need for a separate timer thread, rather the attacker is allowed to read timestamps directly by the victim process. For the side-channel, we assume that attacker do not have the permission to insert RDTSC or any other timing mechanism to directly monitor the victim process.

The attackers may or may not have access to an accurate timing function like RDTSC, but they can use alternatives such as counting threads. It is also assumed that the attacker does not possess root privileges. The hardware and the OS are assumed to be correct and trusted by the victim program.

5 IMPLEMENTATION

In this section, we describe the setup and steps for exploiting a NUCA distance-based side-channel. First, we show the attack on a simulated machine using the Gem5 simulator [9]. We did this to demonstrate the attack in a strictly controlled environment where it is easier to show the main components of an attack. Then, we demonstrate the attack example using the AES decryption function in a real machine. The attack is demonstrated on an Intel Xeon Phi 7290 CPU.

5.1 A NUCA Distance-based Side-channel in a Simulated Machine

To demonstrate the attack, we first explain it using a simulated machine in Gem5 [9]. To perform the attack, we need to identify two addresses that have a significant difference in LLC hit times. To accomplish this objective, we configure an `8x8` tiled architecture with 64 tiles. Each tile contains a core, a private L1I and L1D cache, and a shared LLC (L2) bank. The size of a cache line is 64B. Each L1D is a 2-way associative 4kB cache. Each LLC bank is an 8-way associative 2MB cache. 64 tiles are connected using an 8×8 mesh network. With this configuration, both the L1D cache and LLC use the bit[10:6] of the physical address for indexing. LLC bank is selected using bit[12:6] of the physical address.

An example attack code is shown in Figure 5. Before performing the attack, we allocate a large array (line 1 in Figure 5) and start accessing all cache lines belonging to the array (line 12-13). At the end of those accesses, we can infer that the initial entries of L1D cache will be evicted but they will still reside in LLC. This step is similar to *priming* the cache as used in prior cache side-channel attacks [54]. Next, we need a victim function that accesses two addresses (depending on the secret) such that the addresses reside in two different LLC banks.

For determining which addresses would be useful for the victim function, we analyze the LLC hit latency for all the entries of the array indexed from 0 to `511*64`. We found that the entry at the index `117*64` and `118*64` are mapped to the virtual address `0x4c7fc0` and `0x4c8000`, accordingly. These virtual addresses are mapped to the physical addresses `0xc6fc0` and `0xc7000`, respectively. Using the LLC bank selection bits, we can verify that entry at index `117*64` (virtual address `0x4c7fc0`, physical address `0xc6fc0`) is mapped to bank 63. On the other hand, the entry at index 118*64 is mapped to bank 0. With this mapping, we make sure that the victim function accesses either index `117*64` or `118*64` depending on the secret (Lines 5-6). As a result, when an attacker times the victim function (Lines 16-18), he/she can infer the secret one bit at a time by comparing the access latency with the access latency of bank 63 (or 0).

In this run, we have the attacking code running one of the 64 cores and the rest of the cores are running instances of Rodinia v3 [14] benchmark applications. This was done to make sure the attack is noise resilient. From Figure 4, we can clearly see that in the case of the green area (i.e. secret bit 0), the round trip latency of the load is mostly very high (95+ cycles). On the other hand when the secret bit is 1, we have round trip latency that is very low (around 40 cycles). In this way, by observing this round trip latency, the attacker can infer the secret successfully with high accuracy (>95%).

5.2 Attack on Intel Xeon Phi 7290

In this subsection, we describe the proof-of-concept (POC) implementation of our attack on the Intel Xeon Phi 7290 machine. To realize this attack, we need to identify Far– and Near– tile accesses and separate them. Moreover, we need to ensure LLC hits while having L1D misses.

5.2.1 *Identifying Far- and Near-Tile Accesses.*

To perform the attack, the attacker needs to identify addresses that are mapped to the CHA on a far tile or a near tile. To do that we use a strategy, called *Execute and Profile*. This strategy has two steps. First, it requires an attacker process to execute on the same tile (not necessarily the same physical core or thread) with the victim program. The attacker process then allocates a certain amount of virtual memory and accesses it. The attacker process uses a helper thread that accesses the addresses first to bring them to the LLC. The CPU tile of this LLC acts as the forwarding tile. When another thread of the attacker process accesses those addresses, LLC hits occur. Depending on the distance of the CHA associated with an address, different addresses have different LLC hit times. The attacker process profiles the hit times of different virtual addresses. Based on the LLC hit times, the attacker can identify two sets of virtual addresses - one that is mapped to the far tile's CHA and one that is mapped to the near tile's CHA. Let us denote these two sets as VA_{far} and VA_{near} respectively.

5.2.2 *Ensuring L1D Miss and LLC Hit.*

The attack also requires the accessed data to be a miss in L1D but a remote hit in LLC. If the access is an L1D hit or an LLC Miss, the latency will not depend on the distance in the NoC. Also, if the access is a hit in the local LLC, no communication with the CHA will happen. We guarantee the above scenario using two strategies: *First*, to ensure a remote LLC hit, the attacker can run on a separate core before the victim to load the target cachelines into a remote LLC. Once the cachelines are loaded, other cores will continue to forward from the same LLC slice upon L1D misses. *Second*, the attacker can either prime the L1D cache of the victim by running a thread on the victim core, or invalidating the L1D cache of the victim core using instructions like `PREFETCHW`. The latter is easier since the attacker is already running a thread on another core to keep the presence in LLC.

5.2.3 *Attack Target.*

Our attack requires two elements in the attack target: (1) An operation that accesses various cache locations depending on the secret. Here, we use an SSH engine which uses the vulnerable T-table implementation to decrypt incoming ciphertexts. (2) A fine-grained timing channel that allows the attacker's timing thread to measure part of the vulnerable function instead of the whole function. Here, we assume that the attacker has access to a shared `out` variable with the AES engine, while the rest of the AES engine is isolated from the attacker (so the attacker has no access to the secret key directly). This scenario is possible if the attacker and the AES engine reside in two separate processes but use System V shared memory to share the `in` and `out` buffers for decryption, or if the attacker is isolated from the AES engine in the same process using Intel MPK [15] or ARM Memory Domains [45] except for the `in` and `out` buffers.

5.2.4 *An Attack Example with AES.*

The traditional AES implementation uses a number of transformation tables, known as T tables, to

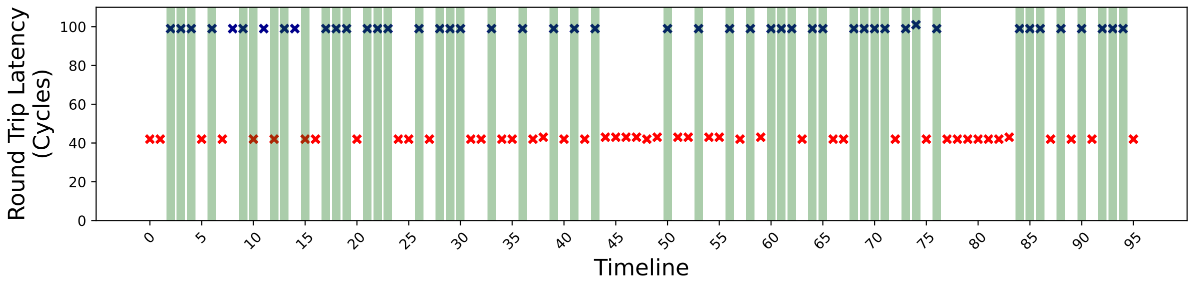


Figure 4: Round trip latency obtained from POC attack on a simulated machine. Green Region represents cases when the secret bit is 0, white region represents cases when the secret bit is 1. Blue Cross (x) represent data predicted to be bit 0 and red cross (x) represent data predicted to be bit 1. We have >=95% accuracy with other cores running Rodinia benchmark [14] applications

.

```
uint8_t arr[512 * 64], secret = 123;

void victim(unsigned int mask) {
    uint8_t s = secret & mask;
    if (s == 0) s ^= arr[117 * 64]; // LLC bank 63
    else        s ^= arr[118 * 64]; // LLC bank 0
}
int main() {
    unsigned long t1, t2, junk, mast = 1;
    for (int i = 0; i < 8; i++, mask <<= 1) {
        // Bring arr[117] and arr[118] to LLC
        for (int j = 0; j < 512; j++)
            junk ^= arr[j * 64];
        _mm_mfence();
        // Time the victim function
        t1 = __rdtscp(&junk);
        victim(mask);
        t2 = __rdtscp(&junk) - t1;
        // If the bit is 0, the latency > 100
        printf("BIT[%d]: %d\n", i, t2 > 100? 0:1);
    }
}
```

Figure 5: An example attack code to leak a secret. The secret data is accessed inside the victim function.

represent the computation and permutation of individual bytes during multiple rounds (9 rounds for AES-128, 11 rounds for AES-192, or 13 rounds for AES-256). These T tables have been the targets of exploitation on multiple side-channel attacks to leak the AES secret keys [8, 10, 27, 35]. We show a simplified version of the AES decryption code in Figure 7. Since AES is a block cipher, in each invocation, the `AES_decrypt` function will take a block of 128 bits as the input and decrypt it using the round key. `AES_decrypt` and `AES_encrypt` have very similar structures, except that they use two different sets of T tables, `Td0`–`Td4` and `Te0`–`Te4`, respectively, and that `AES_decrypt` has an extra round that uses only `Td4` at the end of decryption. Generally, the last round of `AES_decrypt` has been targeted by cache side-channel attacks because, in the last round, four elements of `Td4` and one element (four bytes) of the private key are XORed to produce the four bytes of the plaintext. For example, Line 26 - 31 does the following:

$$\text{S0} = \text{TD4[A]} \wedge \text{TD4[B]} \wedge \text{TD4[C]} \wedge \text{TD4[D]} \wedge \text{RK[1]}$$
$$\text{PUTU32(OUT,S0)}$$

where A, B, C, and D are indices of Td4. If the attacker knows the plaintext and Td4 values used, then the four bytes of the private key (i.e., `rk[1]`) can be derived by XORing them. The key challenge is to determine which Td4 values are used here.

The NUCA distance-based side-channel attack on AES is different from the FLUSH+RELOAD and similar attacks since it cannot infer exactly which cache line is accessed and brought into the cache by examining the cache contents. Instead, the attacker can only time the victim function, `AES_decrypt`, and use the latency to extract the bits inside the key. Specifically, this attack faces two major challenges: (1) **Overlapping of multiple cache loads**: An out-of-order CPU can issue multiple load instructions into the pipeline, and send multiple requests to the Load Store Queue (LSQ). Although the Total Store Ordering (TSO) model of most Intel CPUs forbids reordering of the load requests, requests can still be sent while the prior requests await responses. As a result, the latency of multiple load instructions without mutual dependency can overlap, and thus, the highest latency of individual loads will dominate the overall latency. (2) **Timing difficulty with multiple decryption rounds**: From the attacker's point of view, it is difficult to time the last round of decryption where only elements of `Td4` and the lower 4 bytes of the key are accessed. This is because timing the entire `AES_decrypt` function will include the time of earlier rounds of decryption making it impossible to determine how much time it takes only to access `Td4` entries.

To overcome the challenges, we formulate the attack as follows: The Attacker runs two threads – thread 1 runs a loop on the same core as the AES program to bring `Td0`–`Td3` into L1D, while thread 2 runs on another tile to keep the whole `Td4` inside LLC. Bringing the whole `Td0`–`Td3` tables into the L1D is possible because the total size of `Td0`–`Td3` is 4KB and Intel Xeon Phi 7290 has 32KB per core of L1D cache. Any future access to `Td0`–`Td3` entries does not cause any network traffic in the NoC and hence, `Td4` access times can be measured without any noise. We should note that `Td4` table contains 4 cache lines. Although all of them will be loaded in the LLC of thread 2's tile, the CHAs of those cache lines will likely be spread

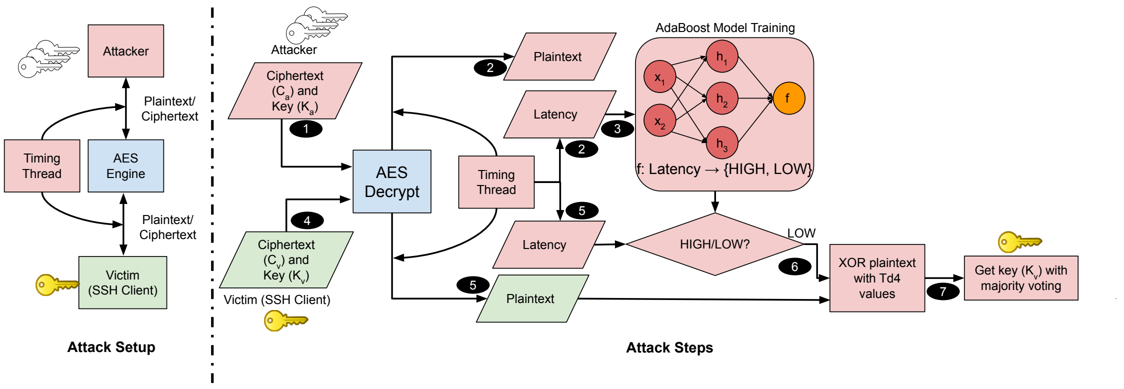


Figure 6: A step-by-step illustration of how an attacker can launch a NUCA distance-based side-channel attack on an AES code. The left side shows the setup assumed in the attack scenario.

```
1  static const u32 Td0[256] = ...;
2  static const u32 Td1[256] = ...;
3  static const u32 Td2[256] = ...;
4  static const u32 Td3[256] = ...;
5  static const u8 Td4[256] = {
6    0x52U, 0x09U, 0x6aU, 0xd5U, 0x30U, 0x36U, 0xa5U, 0x38U,
7    0xbfU, 0x40U, 0xa3U, 0x9eU, 0x81U, 0xf3U, 0xd7U, 0xfbU,
8    ...
9  };
10
11 void AES_decrypt(u32 *in, u32 *out, u32 *rd_key) {
12     u32 s0, s1, s2, s3, t0, t1, t2, t3;
13     s0 = in[0] ^ rk[0];
14     s1 = in[1] ^ rk[1];
15     s2 = in[2] ^ rk[2];
16     s3 = in[3] ^ rk[3];
17     ...
18     /* The last round */
19     s0 =
20         ((u32)Td4[(t0 >> 24)       ] << 24) ^
21         ((u32)Td4[(t3 >> 16) & 0xff] << 16) ^
22         ((u32)Td4[(t2 >>  8) & 0xff] <<  8) ^
23         ((u32)Td4[(t1      ) & 0xff])       ^
24         rk[0];
25     PUTU32(out    , s0);
26     s1 =
27         ((u32)Td4[(t1 >> 24)       ] << 24) ^
28         ((u32)Td4[(t0 >> 16) & 0xff] << 16) ^
29         ((u32)Td4[(t3 >>  8) & 0xff] <<  8) ^
30         ((u32)Td4[(t2      ) & 0xff])       ^
31         rk[1];
32     PUTU32(out + 4, s1);
33     ...
34 }
```

Figure 7: The vulnerable, fully unrolled (i.e., non-iterative) code for AES decryption in `aes_core.c` of OpenSSL 1.1.0f. The source code is simplified for brevity, and only shows the initial values of `Td4` and the last round of `AES_decrypt`.

apart - some of the CHA tiles will be near the victim's tile while others might be far apart as illustrated in Figure 8. Thus, when the victim accesses Td4 table from the LLC of the attacker, some cache lines will incur lower latency while others will incur higher latency.

`Timer` thread will repeatedly check for any change in `out` and `out+4` (Figure 7). We cannot rely on the timing of the whole function because there might be different LLC accesses within the AES_decrypt call which will add to the noise. As AES is a block cipher that uses different rounds of computation on the same table,

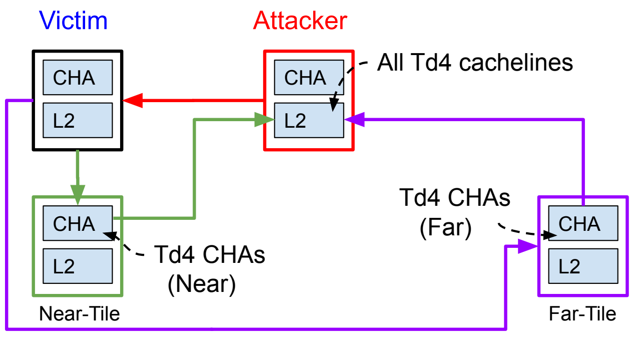


Figure 8: How the distance differences between access near-tile and far-tile CHAs are exploited by the AES attack. The attacker preloads all 4 Td4 cachelines in one core, so the LLC hit latency is determined by the distance to the CHAs.

timing the whole function complicates the secret extraction. However, by isolating the timing thread to only monitor the necessary region of interest within the function, we increase the accuracy of this attack as reflected in the results in §6. This technique can later be used to improve other cache side-channel attacks too. This is done through the `out` buffer provided as a parameter to the `AES_decrypt` function to collect the decrypted text. This `out` buffer is observable to the attacker. This helps the attacker to keep track of when `out` has been modified. The attacker can then start a timer (or start a counting loop).

The attacker stops the timer (or terminates the loop) when `out+4` is modified. The time between observing these two modifications to `out` entries will be directly proportional to the time for executing the statements from Line 26 to 31 in Figure 7. In other words, this is the time for four accesses to the `Td4` table (say, T_{26-31}). This method of measuring the `Td4` accesses derives from the SharedArrayBuffer method proposed in [62]. We can use PREFETCHW timer to monitor memory addresses without any write-access as shown in [28]. Results regarding using PREFETCHW timer implementation is given in §6.3.1. Due to the overlapping of memory loads between lines 26 and 31, the time T_{26-31} will be lower if the CHAs used in `Td4` accesses are all in near tiles or in L1D (let us call this scenario *LOW*) and higher if one or more accesses are to the CHAs in a far tile (let us call it *HIGH*). The attacker can use some simple threshold to determine whether T_{26-31} can be classified as *LOW* or *HIGH*.

Figure 6 shows the flow of our end-to-end attack. In Step 1, an attacker randomly generates N ciphertext and key pairs (C_a and K_a) in the `Attacker` thread ($N = 10M$ in our experiments). The attacker then uses AES Decryption to get plaintext for each round in Step 2. `Timer` thread measures the latency T_{26-31} during each decryption. The attacker classifies the latency with labels *LOW* or *HIGH*. The latency numbers and their labels are used to train an AdaBoost model. The model in essence is a more robust version of the threshold-based classification method that the attacker has already used.

The victim SSH Client is now allowed to use the AES engine to decrypt ciphertext C_v with its own secret key K_v (Step 4). Timer thread monitors these decryptions and measures the latency in Step 5. In Step 6 this latency is predicted to be either *LOW* or *HIGH* using the model trained in Step 3. If the latency is classified as *LOW*, then the attacker can XOR the plaintext with Td4 values associated with the LOW label to determine a set of possible values for K_v (more specifically, bytes 8-11 of K_v). The attacker repeats Step 4 - 7 multiple times and extracts possible values of K_v each time. The attacker uses a majority voting among the possible K_v values after T trials. Figure 15 shows the accuracy for extracting keys after different number of trails. Our experiments indicate that after $T = 4000$ trials, the attacker can extract 4 bytes of K_v with 100% accuracy. [1]

We have tested our code to work with OpenSSL 1.1.0f implementation of AES algorithm. AES introduced constant time code path without using T-table implementations to defend from previously known timing side-channel attacks [6]. However, because of extremely high performance overhead, the default behavior has been set to use T-table based implementation in recent (>3.0.0) versions [56]. So, even recent versions of software based AES with `no-asm no-hw` is vulnerable to this type of attack.

5.3 Generalizability of Attack

We explain the generalizability of the attack in terms of both hardware and software targets.

5.3.1 Vulnerable Hardware Platforms. Our attack requires a mesh-based NoC for the LLC where the LLC hit latencies are significantly different. We ran all of our experiments in Intel Xeon Phi 7290. To show generalizability, we also experimented on another platform, Intel 10700k CPU from Intel Comet Lake processor family which uses mesh interconnect [17]. This processor has 8 core count with 2 threads on each core. Figure 9 shows the results. It shows that the LLC hit latency to same address is uniquely different from different cores. This implies that similar vulnerabilities may exist in other mesh NoC-based platforms. However, we did not run end-to-end attacks on any other machine.

5.3.2 Vulnerable Software Targets. Our attack depends on two things: a) A data structure (buffer) that spans across multiple cachelines, and b) Secret dependent accesses to those cachelines. Due to CPU interleaving, different entries of these large data structures are allocated to different LLC slices and that's why we have secret dependent different latencies which can be used as a source to leak those secrets. Based on these two properties, we have explored the OpenSSL

[1] Our POC code is here - https://anonymous.4open.science/r/NUCA-side-channel-6C9D

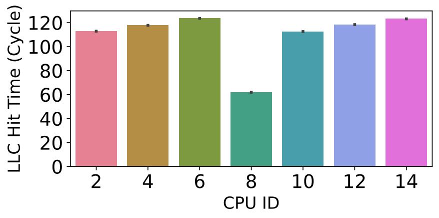


Figure 9: LLC Hit Latency for the same address from different CPU cores in Intel 10700k. Black error bars indicate 95% confidence interval over 10,000 trials after filtering out possible LLC misses. Requster is at CPU ID 0.

```
static const uint32_t S1[256] = {
    0x00636363, 0x007c7c7c, 0x00777777, 0x007b7b7b,
  ...
};

static void sl1(ARIA_u128 *o, const ARIA_u128 *x, const ARIA_u128
    *y)
{
    unsigned int i;
    for (i = 0; i < ARIA_BLOCK_SIZE; i += 4) {
        o->c[i    ] = sb1[x->c[i    ] ^ y->c[i    ]];
        o->c[i + 1] = sb2[x->c[i + 1] ^ y->c[i + 1]];
        o->c[i + 2] = sb3[x->c[i + 2] ^ y->c[i + 2]];
        o->c[i + 3] = sb4[x->c[i + 3] ^ y->c[i + 3]];
    }
}
```

Figure 10: The code implementation of ARIA in openssl 1.1.0f shows similar data structure S1 that spans multiple cacheline as well as function sl1 that accesses one of those entries depending on the secret

implementation of some popular cryptography algorithms from [4] and found many algorithms have these two properties and hence would be vulnerable to similar attack scenarios. Camellia [50] and ARIA [44] have a similar S-box structure that expands to multiple cachelines as shown in Figure 10, Another variable key length cryptographic algorithm Blowfish [61] has a similar data structure which spans multiple adjacent cachelines. These algorithms can potentially be vulnerable to the same attack.

6 EXPERIMENT

In this section we first describe the experiment results from Gem5 simulation. Then we will show the experiment results for covert-channel using the timing difference between farthest and nearest CHAs. Finally we will explain the results we have obtained using Intel Xeon Phi 7290 to implement the attack on a real machine.

6.1 Results from Gem5 simulation

6.1.1 Simulated Attack Latency Distribution. In Figure 11 we can that the latency distribution for LLC hits from the farthest and the nearest nodes. Timing distribution is clearly separable for LLC hit at the Near and the Far node. This clearly shows that the possibility of this side-channel to extract secrets.

With a specific set of machine where we can replicate similar behavior, we can easily extract the secrets by making sure that either the nearest or the farthest node is hit at LLC during the cache access.

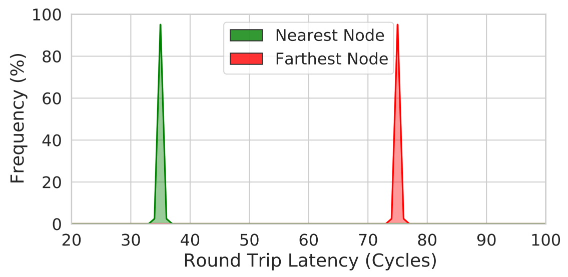


Figure 11: Simulated Attack Latency Distribution in Gem5. In this case we can clearly observe the latency difference due to the physical distance of nodes requesting the data from the source of the data

6.2 Covert-channel Experiments

In this section, we show the results that we have obtained by setting the attacker and victim thread in the same process. In a covert channel setup, the attacker and victim are trying to communicate covertly. This gives us the opportunity to have synchronization between attacker and victim process.

6.2.1 Bandwidth & Error Rate. Running a similar proof of concept code with all the optimizations and steps mentioned in Section 5 we get an accuracy of 99.98% on extracting secrets after 100,000 trials. In these trials, the channel could reach a bandwidth of 205KBPS while maintaining an Error Rate of less than 0.02%.

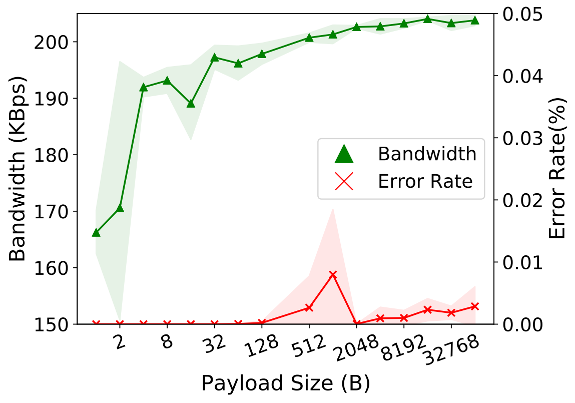


Figure 12: Bandwidth & Error Rate of covert channel with varying payload size

Figure 12 shows the Bandwidth and Error Rate of the covert channel implementation with varying size of payload. The shaded area shows 95% confidence interval.

6.3 Side-channel Results

6.3.1 PREFETCHW Timer. In this section, we discuss the possibility of using PREFETCHW for observing any change in the data following the examples of Adversarial Prefetch [28]. We used a spy thread to continuously execute PREFETCHW on a read-only memory address. When the victim thread who has write-access to that data, writes new data on that address, PREFETCHW takes much longer time >150 cycles to complete compared to <100 cycle time as observed in this experiment. The result latency distribution is shown in Figure 13 We can use this prefetch timer to identify whether *out*

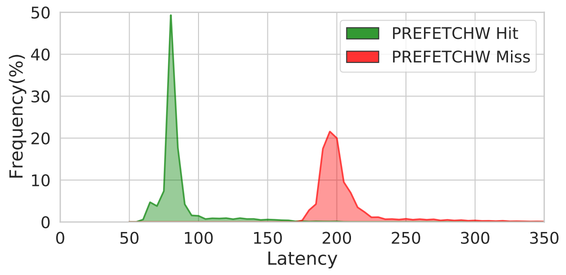


Figure 13: Hit or Miss latency distribution using PREFETCHW instructions. PREFETCHW takes much longer time to complete if a remote LLC modifies the data due to invalidations. To monitor a shared variable, we can utilize this difference.

shared variable was modified by the victim or not assuming that the attacker do not have write-access to the *out* variable.

6.3.2 Accuracy of ML Classifier. In our proof-of-concept (POC) code, we take multiple samples of T_{24-28} from decrypting a specific 16-byte plaintext and use majority voting (using AdaBoost Classifier [29]) to determine if one or more accesses fall to a far tile. Figure 14 shows that with AdaBoost Classifier, we can determine accesses to a far tile with 100% accuracy by using 40 or more samples. If one or more accesses happen to a far tile, we determine the potential lower 4 bytes of the key by XOR'ing. We keep doing this using random plaintexts and eventually, extract the lower 4 bytes of the key using a simple majority for each byte.

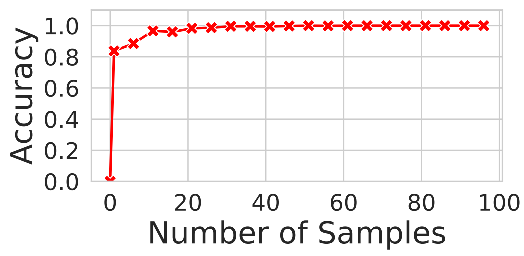


Figure 14: Accuracy of determining if one or more accesses fall to the far tile. We reach 100% accuracy by taking majority voting of 40 samples or more.

6.3.3 Key Extraction Accuracy. Figure 15 shows the accuracy for extracting keys with varying number of trails. Our experiments indicate that after $T = 4000$ trials, the attacker can extract 4 bytes of K_v with 100% accuracy.

7 POSSIBLE DEFENSE

In this section, we discuss about the possible defense mechanism against NUCA distance-based side-channel attacks. *First,* we need to identify the root cause of the attack and then we can add components/techniques to disable it. However, in the case of this attack,

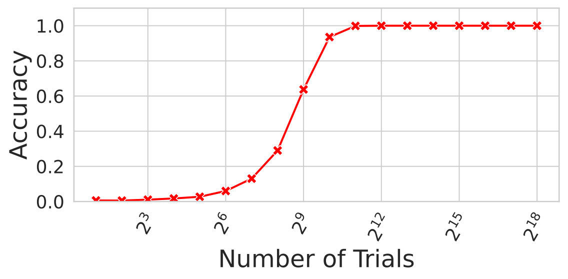


Figure 15: Key extraction accuracy with repeated decryption trials. We can extract 4 bytes of any random key with 100% accuracy by using only $\simeq 4000$ trials.

the source of the side-channel is because we have differences in the physical location of different CHAs representing different parts of the address space. In case of on-chip mesh networks, we have to take this physical distance into account while traveling hop by hop from the source node to the destination. With larger on chip networks like Intel Xeon Phi 7290 the number of hops to reach the farthest node only increases compared to the nearest nodes.

To have an architectural defense from this type of side-channel attack, we need to make sure that all the nodes are reachable within the same amount of time. This can be done in many network configurations like a Ring-based network where with some modification we can guarantee all the nodes can be reached within the same amount of time. Or, we can artificially add delays to the shorter path communication so that the attacker cannot infer the difference from timing the near and far node communications. In this way, all the nodes, regardless of their distance from the source, will send their response in the worst-case scenario. This will also harshly impact the performance. Our experiment results using the Booksim [37] simulator shows that in the 8×8 configuration, we have the packet network latency getting saturated at >100 cycles at an injection rate of 0.1. The experiment results can be seen here in Figure 16.

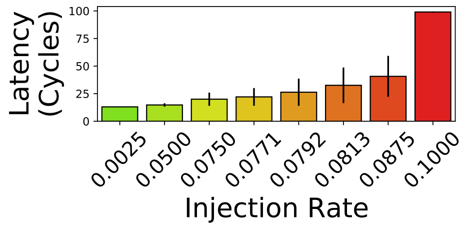


Figure 16: 8x8 on chip network simulated in BookSim [37] with uniform random traffic. The graph shows packet network latency with varying injection rate until the network gets saturated.

As we can observe from the Figure 16 at a higher injection rate, the network gets saturated. So at this latency (>100 cycles), the latency observed by the attacker would not be correlated with the distance from the attacker node but rather be compounded by the saturated network. Following this, we can design a defense mechanism that can work against NUCA distance-based side-channel attacks.

8 CONCLUSION

We have shown a novel NUCA distance-based side-channel attack in a simulated as well as a real machine. We demonstrated how to use this attack to break AES cryptographic algorithm in a Intel Xeon Phi 7290 machine. We used a combination of microarchitectural techniques with machine learning to overcome major challenges like identifying overlapping access while measuring access latencies of only region of interest of a victim function. With Gem5 simulator [9] we have shown the principle of this attack on the presence of other background application. We have also shown how a covert channel can be created using the same principles of non uniform distance between LLC banks. Our covert channel can transmit information at a rate of 205KB/s with a very low error rate of 0.02%. Finally, using the restricted environment in a side-channel setting, we could extract 4 bytes of the AES key with only 4,000 decryption trials. We leave the future works for extracting rest of the secret key by extending this or by using some other side-channel attacks.

REFERENCES

[1] Amd ryzen. https://www.amd.com/en/products/cpu/amd-epyc-7742.
[2] Ampere altra review. https://www.anandtech.com/show/16315/the-ampere-altra-review/3.
[3] Intel xeon phi. https://ark.intel.com/content/www/us/en/ark/products/series/75557/intel-xeon-phi-processors.html.
[4] Abood, O. G., and Guirguis, S. K. A survey on cryptography algorithms. *International Journal of Scientific and Research Publications 8*, 7 (2018), 495–516.
[5] Aciiçmez, O., and Schindler, W. A vulnerability in rsa implementations due to instruction cache analysis and its demonstration on openssl. In *Cryptographers' Track at the RSA Conference* (2008), Springer, pp. 256–273.
[6] Aly, H., and ElGayyar, M. Attacking aes using bernstein's attack on modern processors. In *Progress in Cryptology–AFRICACRYPT 2013: 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings 6* (2013), Springer, pp. 127–139.
[7] Arafa, M., Fahim, B., Kottapalli, S., Kumar, A., Looi, L. P., Mandava, S., Rudoff, A., Steiner, I. M., Valentine, B., Vedaraman, G., et al. Cascade lake: Next generation intel xeon scalable processor. *IEEE Micro 39*, 2 (2019), 29–36.
[8] Bernstein, D. J. Cache-timing attacks on aes.
[9] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., et al. The gem5 simulator. *ACM SIGARCH computer architecture news 39*, 2 (2011), 1–7.
[10] Bonneau, J., and Mironov, I. Cache-collision timing attacks against aes. In *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems* (Berlin, Heidelberg, 2006), CHES'06, Springer-Verlag, p. 201–215.
[11] Borkar, S. Design challenges of technology scaling. *IEEE micro 19*, 4 (1999), 23–29.
[12] Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., and Sadeghi, A.-R. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)* (2017).
[13] Briongos, S., Malagón, P., Moya, J. M., and Eisenbarth, T. Reload+refresh: Abusing cache replacement policies to perform stealthy cache attacks. In *Proceedings of the 29th USENIX Conference on Security Symposium* (2020), pp. 1967–1984.
[14] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)* (2009), Ieee, pp. 44–54.
[15] Corporation, I. Intel© 64 and ia-32 architectures software developer's manual, June, 2023. Accessed on 2023-09-18.
[16] Daemen, J., and Rijmen, V. Aes proposal: Rijndael.
[17] Dai, M., Paccagnella, R., Gomez-Garcia, M., McCalpin, J., and Yan, M. Don't mesh around:{Side-Channel} attacks and mitigations on mesh interconnects. In *31st USENIX Security Symposium (USENIX Security 22)* (2022), pp. 2857–2874.
[18] Doerfler, D., Austin, B., Cook, B., Deslippe, J., Kandalla, K., and Mendygral, P. Evaluating the networking characteristics of the cray xc-40 intel

knights landing-based cori supercomputer at nersc. *Concurrency and Computation: Practice and Experience 30*, 1 (2018), e4297.

[19] Godfrey, M., and Zulkernine, M. A server-side solution to cache-based side-channel attacks in the cloud. In *2013 IEEE Sixth International Conference on Cloud Computing* (2013), IEEE, pp. 163–170.

[20] Goodman, J., and Hum, H. Mesif: A two-hop cache coherency protocol for point-to-point interconnects (2004).

[21] Gras, B., Razavi, K., Bos, H., Giuffrida, C., et al. Translation leak-aside buffer: Defeating cache side-channel protections with tlb attacks. In *USENIX Security Symposium* (2018), vol. 216.

[22] Gruss, D., Lettner, J., Schuster, F., Ohrimenko, O., Haller, I., and Costa, M. Strong and efficient cache side-channel protection using hardware transactional memory. In *USENIX Security Symposium* (2017), pp. 217–233.

[23] Gruss, D., Maurice, C., Wagner, K., and Mangard, S. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016), Springer, pp. 279–299.

[24] Gruss, D., Spreitzer, R., and Mangard, S. Cache template attacks: Automating attacks on inclusive last-level caches. In *24th {USENIX} Security Symposium ({USENIX} Security 15)* (2015), pp. 897–912.

[25] Gueron, S. Advanced encryption standard (aes) instructions set. *Intel, http://softwarecommunity. intel. com/articles/eng/3788. htm, accessed 25* (2008).

[26] Gullasch, D., Bangerter, E., and Krenn, S. Cache games–bringing access-based cache attacks on aes to practice. In *2011 IEEE Symposium on Security and Privacy* (2011), IEEE, pp. 490–505.

[27] Gülmezoğlu, B., undefinednci, M. S., Irazoqui, G., Eisenbarth, T., and Sunar, B. A faster and more realistic flush+reload attack on aes. In *Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064* (Berlin, Heidelberg, 2015), COSADE 2015, Springer-Verlag, p. 111–126.

[28] Guo, Y., Zigerelli, A., Zhang, Y., and Yang, J. Adversarial prefetch: New cross-core cache side channel attacks. In *2022 IEEE Symposium on Security and Privacy (SP)* (2022), IEEE, pp. 1458–1473.

[29] Hastie, T., Rosset, S., Zhu, J., and Zou, H. Multi-class adaboost. *Statistics and its Interface 2*, 3 (2009), 349–360.

[30] Herdrich, A., Verplanke, E., Autee, P., Illikkal, R., Gianos, C., Singhal, R., and Iyer, R. Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2016), IEEE, pp. 657–668.

[31] Hoeneisen, B., and Mead, C. A. Fundamental limitations in microelectronics—i. mos technology. *Solid-State Electronics 15*, 7 (1972), 819–829.

[32] Horro, M., Kandemir, M. T., Pouchet, L.-N., Rodríguez, G., and Touriño, J. Effect of distributed directories in mesh interconnects. In *Proceedings of the 56th Annual Design Automation Conference 2019* (2019), pp. 1–6.

[33] Intel, C. Improving real-time performance by utilizing cache allocation technology. *Intel Corporation, April* (2015).

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

[35] Irazoqui, G., Inci, M. S., Eisenbarth, T., and Sunar, B. Wait a minute! a fast, cross-vm attack on aes. In *Research in Attacks, Intrusions and Defenses* (Cham, 2014), A. Stavrou, H. Bos, and G. Portokalidis, Eds., Springer International Publishing, pp. 299–319.

[36] Jaleel, A., Mattina, M., and Jacob, B. Last level cache (llc) performance of data mining workloads on a cmp-a case study of parallel bioinformatics workloads. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.* (2006), IEEE, pp. 88–98.

[37] Jiang, N., Becker, D. U., Michelogiannakis, G., Balfour, J., Towles, B., Shaw, D. E., Kim, J., and Dally, W. J. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)* (2013), IEEE, pp. 86–96.

[38] Kang, J.-S., Myung, H., and Yuk, J.-H. Examination of computational performance and potential applications of a global numerical weather prediction model mpas using kisti supercomputer nurion. *Journal of Marine Science and Engineering 9*, 10 (2021), 1147.

[39] Kayaalp, M., Ponomarev, D., Abu-Ghazaleh, N., and Jaleel, A. A high-resolution side-channel attack on last-level cache. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (2016), IEEE, pp. 1–6.

[40] Kessler, R. E., and Hill, M. D. Page placement algorithms for large real-indexed caches. *ACM Transactions on Computer Systems (TOCS) 10*, 4 (1992), 338–359.

[41] Kim, C., Burger, D., and Keckler, S. W. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems* (2002), pp. 211–222.

[42] Kim, T., Peinado, M., and Mainar-Ruiz, G. {STEALTHMEM}: System-level protection against cache-based side channel attacks in the cloud. In *21st {USENIX} Security Symposium ({USENIX} Security 12)* (2012), pp. 189–204.

[43] Kiriansky, V., Lebedev, I., Amarasinghe, S., Devadas, S., and Emer, J. Dawg: A defense against cache timing attacks in speculative execution processors. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2018), IEEE, pp. 974–987.

[44] Kwon, D., Kim, J., Park, S., Sung, S. H., Sohn, Y., Song, J. H., Yeom, Y., Yoon, E.-J., Lee, S., Lee, J., et al. New block cipher: Aria. In *Information Security and Cryptology-ICISC 2003: 6th International Conference, Seoul, Korea, November 27-28, 2003. Revised Papers 6* (2004), Springer, pp. 432–445.

[45] Limited, A. Arm$^{©}$ developer suite, 2001. Accessed on 2023-09-18.

[46] Liu, F., Ge, Q., Yarom, Y., Mckeen, F., Rozas, C., Heiser, G., and Lee, R. B. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE international symposium on high performance computer architecture (HPCA)* (2016), IEEE, pp. 406–418.

[47] Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy* (2015), IEEE, pp. 605–622.

[48] Lujan, J., Vigil, M., Kenyon, G., Sanbonmatsu, K., and Albright, B. Trinity supercomputer now fully operational. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2017.

[49] Marshall, A., Howard, M., Bugher, G., Harden, B., Kaufman, C., Rues, M., and Bertocci, V. Security best practices for developing windows azure applications. *Microsoft Corp 42* (2010), 12–15.

[50] Moriai, S., Kato, A., and Kanda, M. Addition of camellia cipher suites to transport layer security (tls). Tech. rep., 2005.

[51] Mujtaba, H. Intel skylake-x and skylake-sp mesh architecture for xcc "extreme core count" cpus detailed – features higher efficiency, higher bandwidth and lower latency.

[52] Muralimanohar, N., and Balasubramonian, R. Interconnect design considerations for large nuca caches. *ACM SIGARCH Computer Architecture News 35*, 2 (2007), 369–380.

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

[54] Osvik, D. A., Shamir, A., and Tromer, E. Cache attacks and countermeasures: the case of aes. In *Cryptographers' track at the RSA conference* (2006), Springer, pp. 1–20.

[55] Paccagnella, R., Luo, L., and Fletcher, C. W. Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical. In *30th USENIX Security Symposium (USENIX Security 21)* (Aug. 2021), USENIX Association.

[56] Pauli Dale, M. C. aes: make the no-asm constant time code path not the default. https://github.com/openssl/openssl/commit/1f7c5c56c7365fefd9cff9bea4d3d27346ca44d1, Jan 30,2022. Accessed on 2023-09-18.

[57] Percival, C. Cache missing for fun and profit, 2005.

[58] Qureshi, M. K. New attacks and defense for encrypted-address cache. In *Proceedings of the 46th International Symposium on Computer Architecture* (2019), pp. 360–371.

[59] Reinbrecht, C., Susin, A., Bossuet, L., Sigl, G., and Sepúlveda, J. Side channel attack on noc-based mpsocs are practical: Noc prime+ probe attack. In *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)* (2016), IEEE, pp. 1–6.

[60] Research, T. Amd optimizes epyc memory with numa. Available at https://www.amd.com/system/files/2018-03/AMD-Optimizes-EPYC-Memory-With-NUMA.pdf (2021/08/12), march 2018.

[61] Schneier, B. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption: Cambridge Security Workshop Cambridge, UK, December 9–11, 1993 Proceedings* (2005), Springer, pp. 191–204.

[62] Schwarz, M., Weiser, S., Gruss, D., Maurice, C., and Mangard, S. Malware guard extension: Using sgx to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2017), Springer, pp. 3–24.

[63] Sodani, A. Knights landing (knl): 2nd generation intel® xeon phi processor. In *2015 IEEE Hot Chips 27 Symposium (HCS)* (2015), IEEE, pp. 1–24.

[64] Stanzione, D., Barth, B., Gaffney, N., Gaither, K., Hempel, C., Minyard, T., Mehringer, S., Wernert, E., Tufo, H., Panda, D., et al. Stampede 2: The evolution of an xsede supercomputer. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. 2017, pp. 1–8.

[65] Stefanov, E., Dijk, M. V., Shi, E., Chan, T.-H. H., Fletcher, C., Ren, L., Yu, X., and Devadas, S. Path oram: An extremely simple oblivious ram protocol. *J. ACM 65*, 4 (Apr. 2018).

[66] Tam, S. M., Muljono, H., Huang, M., Iyer, S., Royneogi, K., Satti, N., Qureshi, R., Chen, W., Wang, T., Hsieh, H., et al. Skylake-sp: A 14nm 28-core xeon® processor. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)* (2018), IEEE, pp. 34–36.

[67] Tromer, E., Osvik, D. A., and Shamir, A. Efficient cache attacks on aes, and countermeasures. *Journal of Cryptology 23*, 1 (2010), 37–71.

[68] Van de Pol, J., Smart, N. P., and Yarom, Y. Just a little bit more. In *Cryptographers' Track at the RSA Conference* (2015), Springer, pp. 3–21.

[69] Wang, Z., and Lee, R. B. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th annual international symposium on Computer architecture* (2007), pp. 494–505.

[70] Wassel, H. M., Gao, Y., Oberg, J. K., Huffmire, T., Kastner, R., Chong, F. T., and Sherwood, T. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. *ACM SIGARCH Computer Architecture News 41*, 3 (2013), 583–594.

[71] Xiong, W., and Szefer, J. Leaking information through cache lru states. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2020), IEEE, pp. 139–152.

[72] Yao, F., Doroslovacki, M., and Venkataramani, G. Are coherence protocol states vulnerable to information leakage? In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2018), IEEE, pp. 168–179.

[73] Yarom, Y., and Benger, N. Recovering openssl ecdsa nonces using the flush+ reload cache side-channel attack. *IACR Cryptol. ePrint Arch. 2014* (2014), 140.

[74] Yarom, Y., and Falkner, K. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)* (2014), pp. 719–732.

[75] Zhang, D., Askarov, A., and Myers, A. C. Predictive mitigation of timing channels in interactive systems. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), pp. 563–574.

[76] Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), pp. 305–316.

[77] Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 990–1003.