Not So Secure TSC

Jonas Juffinger, Sudheendra Raghav Neela, and Daniel Gruss

Graz University of Technology, Austria

Abstract. Modern cloud environments greatly facilitate efficiency as workloads of multiple tenants running on the same physical system, often grouped by functionality, e.g., function-as-a-service containers, or platform-as-a-service virtual machines. However, the sharing of physical hardware resources across mutually distrusting tenants comes with security implications. Consequently, cloud providers also offer higher security levels with trusted execution environments in the form of confidential virtual machines, e.g., Intel TDX and AMD SEV-SNP. Many works have shown that attackers can exploit microarchitectural side channels in cloud scenarios, including attacks on trusted-execution environments. However, the practical relevance of these attacks hinges on the co-location of the attacker with its victim on the same physical machine within the data center. Prior work has presented co-location detection techniques that work inside containers or on regular virtual machines. However, co-location detection techniques for confidential virtual machines, e.g., AMD SEV-SNP, have not been studied yet, as these have to be performed from within AMD SEV-SNP virtual machines to target the same physical host machine.

In this paper, we present the first co-location detection technique working on confidential virtual machine hosts. We exploit the new SecureTSC feature supported on recent AMD processors with SEV-SNP. SecureTSC is intended to provide a trusted timing source to confidential virtual machines that cannot be tampered with by the host. We systematically study the behavior of SecureTSC and show that it can be exploited to detect whether two confidential virtual machines are co-located. We demonstrate our co-location detection in a concrete scenario, where two confidential virtual machines attempt to co-locate with each other. Our attack uses a minimal network protocol, *TsCupid*, to determine whether any of the connected confidential virtual machines are co-located. We practically evaluate our attacks and show that we can detect co-location within 0.13 seconds in a fully parallelized way, minimizing the cost for an attack. Finally, we show that our attack cannot be mitigated without modifying the SecureTSC feature and propose a concrete design change that would fully prevent our attack.

Keywords: Side Channels, Timing Attack, Co-Location Attack, AMD SEV, Confidential Computing

1 Introduction

The efficiency and economic advantages of cloud computing and virtualization are rooted in the sharing of physical resources. Today, various types of cloud computing exist, often categorized into function-as-a-service, software-as-a-service, platform-as-a-service, and infrastructure-as-a-service. The idea for all types is similar: Place workloads of the same type but from different tenants on the same physical machine and share as many physical resources as possible while maintaining confidentiality, integrity, and availability, of the customer's data and workloads. As a consequence, a long line of work studied the security properties of workloads in cloud environments, in particular in terms of side-channel attacks [10, 44, 60, 68, 73, 74] as well as their mitigation [33, 42, 72]. These attacks typically monitor the usage of a shared hardware resource and exfiltrate secret information based on the observed behavior, e.g., timing [30], or access patterns [44].

Beyond distrusting other co-located tenants, customers of cloud computing may also distrust the cloud provider [59] or have legal restrictions on how and where their data may be processed [50]. Consequently, the concept of trusted execution environments was developed [8, 18] and implemented in billions of processors, e.g., Arm TrustZone [8], Intel SGX [15], AMD SEV [32], and Intel TDX [27]. Today, commercial cloud providers already offer confidential virtual machines, e.g., using AMD SEV or Intel TDX, typically in an platform-as-a-service model. However, while these confidential virtual machines promise higher security, they are still susceptible to side-channel attacks as demonstrated in many scientific works [12, 37–39, 62]. Moreover, the lack of a trusted timing source in trusted execution environments limits use cases that require a trusted timing source, especially those that rely on time for security-critical decisions [1, 7], including defenses against privileged attackers that resort to busy-looping timing threads instead [13, 49]. This changed with AMD's introduction of the SecureTSC feature[1], providing a trusted, non-malleable timing source to confidential virtual machines, allowing to detect manipulations from the untrusted outside.

While the side-channel attacks themselves may be practical, their practical relevance often hinges on the attack scenario. In particular, attacks on virtual machines of other tenants in modern public cloud environments, require co-location, which is challenging to achieve for multiple reasons: First, given the massive expansion rate of cloud computing, and growing sizes of data centers, the a priori probability of co-location is shrinking. Consequently, an attacker has to spawn more containers or virtual machines to possibly achieve co-location. Second, cloud providers have systematically eliminated prior channels for co-location detection [26], requiring attackers to resort to new and less reliable side-channel-based co-location detection [25]. Third, the resource provisioning of cloud providers limits the possible co-location targets: Cloud vendors typically dedicate entire servers to one type of workload, e.g., function-as-a-service or platform-as-a-service. Furthermore, resource provisioning systems monitor work-

[1] Intel introduced an equivalent feature dubbed "Virtual TSC" [29].

loads and dynamically move similar workloads to the same physical machine to optimize the utilization of hardware resources [43]. Consequently, to perform side-channel attacks on co-located confidential virtual machines, the attacker needs to detect co-location and mount the attack from inside a confidential virtual machine. However, co-location detection techniques for confidential virtual machines, e.g., AMD SEV-SNP, have not been studied yet.

In this paper, we present the first co-location detection technique working on confidential virtual machine hosts. We systematically study the behavior of the new SecureTSC feature supported on recent AMD processors with SEV-SNP support. Furthermore, we show that SecureTSC can be utilized as a reliable mechanism for co-location detection. We devise a minimal network protocol, `TsCupid`, which solves the challenges to this co-location detection, using a central coordination server, adjusting for various factors influencing the timestamp counter values and by threshold-comparing normalized timestamps across all connected confidential virtual machines.

We evaluate our co-location detection technique in a realistic scenario on an AMD Epyc 7313P (Zen 3, Milan microarchitecture), as public cloud vendors are still in the process of adopting SecureTSC and enabling it for customers. We focus on a scenario where two confidential virtual machines attempt to co-locate with each other. We demonstrate that, based on the SecureTSC values, we can correctly and reliably detect co-location on the same physical system within 0.13 seconds.

Finally, we show that our attack cannot be mitigated without modifying the SecureTSC feature. As long as the attacker has access to the unperturbed `rdtsc` value as a timing source, the scaling register, and the offset field, an attack remains possible. We propose a concrete design change that would fully prevent our attack while maintaining `rdtsc` as a trusted and non-malleable timing source.

In summary, this paper makes the following contributions:

- We present the first co-location detection technique on confidential virtual machines, exploiting the novel SecureTSC feature on AMD SEV-SNP processors.
- We develop a scalable methodology for co-location using our detection technique and analyze its costs on the AWS cloud and the Google Cloud Platform.
- We practically evaluate our co-location detection on an AMD Epyc 7313P system with SEV-SNP and SecureTSC enabled. We correctly detect co-location within 0.25 seconds with a success rate above 99.56 % with a 0.13 seconds detection threshold.
- We propose a design change for SecureTSC that fully mitigates our attack while maintaining the functionality of SecureTSC, by moving certain values and operations to AMD's security processor (AMD-SP).[2]

Outline. Section 2 provides background on trusted-execution environments, side channels, co-location detection, and SecureTSC. Section 3 presents the high-

[2] AMD-SP and PSP are sometimes used interchangeably in the literature and official documentation.

level idea of our co-location detection. Section 4 presents the design and implementation. Section 5 evaluates our co-location detection in a realistic scenario on an AMD Epyc 7313P machine with SEV-SNP and SecureTSC enabled. Section 6 discusses potential mitigations to prevent co-location detection. We conclude our work in Section 7.

Responsible Disclosure to AMD. We reported using SecureTSC for co-location detection to AMD on March 14th, 2024. We further showed our findings using the TsCupid protocol on April 29th, 2024. AMD acknowledged our findings and concluded that this was outside their current threat model for SecureTSC.

2 Background

In this section, we provide background on trusted execution environments, side-channel attacks and co-location detection, as well as the novel SecureTSC feature.

2.1 Security of Trusted Execution Environments

The trusted computing base is the minimal set of hardware and software components that a user or workload has to trust. If the trusted computing base is compromised, the workload or the user's data may be compromised as well. Traditionally, the operating system was part of the trusted computing base. Trusted execution environments aim to minimize the trusted computing base by restricting the capabilities of the operating system and even the system administrator. Arm TrustZone [8, 18] has been deployed for more than a decade in billions of mobile devices. TrustZone has an additional, fully isolated execution realm, running a separate trusted operating system and trusted applications (called trustlets). While architectural manipulation of TrustZone is only possible with the exploitation of bugs inside the TrustZone [20], microarchitectural attacks remain unmitigated [40, 51, 53, 71]. Intel introduced a trusted execution environment, SGX [15, 28], which splits applications into an untrusted part and a trusted part (called enclave). Architectural access is mitigated, and the memory is encrypted against physical attacks. However, again bugs inside an SGX enclave [35, 65], or side channels [11, 57] can still lead to exploitation. Furthermore, transient-execution attacks, e.g., Foreshadow [61] and ZombieLoad [55], as well as power side channels [34, 41] can still leak precise information from enclaves.

The current generation of trusted execution environments follows a different approach, focusing on virtual machines. AMD's Secure Encrypted Virtualization (SEV) [4, 32] has been available in commercial processors since 2020. More recently, Intel also introduced their alternative, Trusted Domain Extension (TDX) [27]. The scientific community has studied the security of AMD SEV thoroughly. Early works exploited unencrypted virtual machine states [23, 66], control over nested page tables [23, 47, 48], and insufficient encryption [19, 67].

More recent works demonstrated that side channels on the ciphertext can deterministically leak data [37, 39] as well as the possibility of power side channel attacks [64].

Side-channel attacks on trusted execution environments generally assume co-location on the same physical host. Since in a personal computer setting this is realistic, only few works studied the security implications of malicious workloads inside trusted execution environments [54, 56]. With confidential virtual machines, this also has implications on the threat model: Cloud vendors typically dedicate entire servers to one type of workload [43], e.g., a host running many AMD SEV virtual machines. Furthermore, resource provisioning systems monitor workloads and dynamically move similar workloads to the same physical machine to optimize the utilization of hardware resources [43], *i.e.*, again many similar virtual machines are grouped onto the same server. Consequently, to perform side-channel attacks on co-located confidential virtual machines, the attacker needs to detect co-location and mount the attack from inside a confidential virtual machine.

2.2 Co-location Detection Attacks

As Zhao et al. [75] point out, co-location detection is a pivotal step to side-channel attacks in the cloud. Consequently, a long line of research studied techniques to detect co-location in commercial clouds [9, 25, 26, 52, 58, 73, 74], for instance by analyzing IP addresses, hard disk performance, network latency, and cache covert channels on the L1 cache and the last-level cache. Inci et al. [25] report that most of the aforementioned techniques have been addressed with mitigations by public cloud providers. In particular, cloud providers are aware of the concerns around SMT side channels and have in some instances mitigated them in the past by disabling hyperthreading [60] or strictly schedule only workloads of the same virtual machine on two hyperthreads [3]. Inci et al. [25] also show and argue that some side channels remain possible, e.g., Prime+Probe on the last-level cache. However, as inclusive last-level caches do not scale with high core counts, more recent processor designs often have non-inclusive L3 caches, e.g., recent Intel server processors. Consequently, even if an attacker manages to craft a Prime+Probe eviction set evict data from the L3 cache, repeatedly iterating over Prime+Probe eviction sets may reach the L3 cache but will not evict data from other co-located workload's private L1 and L2 caches. AMD processors also split the last-level cache into isolated parts per core complex. Hence, while Flush+Reload via shared memory may still be possible within one core complex, cache attacks across core complexes are currently infeasible. Varadarajan et al. [63] showed that the effects of memory bus contention [69] can influence the performance on shared cloud systems in a way that allows for co-location detection. Zhao et al. [75] study the practicality of co-location attacks in function-as-a-service clouds and demonstrate that an attacker can reliably co-locate with a victim workload. Being specific to function-as-a-service clouds, their approach motivates further research into co-location techniques for other types of contemporary cloud systems.

2.3 AMD SecureTSC

Several works proposed to detect ongoing attacks on trusted execution environments requiring a trusted timing source [13, 49]. Focusing on Intel SGX, these works note that they have to rely on a timing thread as the `rdtsc` instruction is not available in SGX. Similarly, `rdtsc` can also be manipulated by the hypervisor in (confidential) virtual machines and, hence, even when available in a trusted execution environment cannot inherently be considered a trusted timing source. This situation changed with the novel SecureTSC feature introduced for AMD SEV-SNP [5, 16] and the Intel Virtual TSC feature [29] respectively. SecureTSC makes the `rdtsc` instruction a trusted and non-malleable timing source. A confidential virtual machine can enable SecureTSC via a bit in the `SEV_FEATURES` field in the VM Save Area. When enabled, all accesses to the timestamp counter, e.g., using `rdtsc` or `rdtscp`, or by reading the TSC Model-Specific Register (MSR), will be resolved directly by the hardware, without any way for the hypervisor to manipulate the value.[3] The SecureTSC timestamp counter is not affected by the P-State 0 frequency or by writes to the TSC MSR, which we confirm in our experiments.

Hypervisors can set the TSC frequency of virtual machines (guests). Even with SEV-SNP, the hypervisor still has control over the guest's TSC frequency, via the `DESIRED_TSC_FREQ` field in the `SNP_LAUNCH_START` request, sent to the AMD-SP while launching a guest. This value is stored in the Guest Context, a structure containing information, keys, and metadata associated with the guest. After being launched, the guest queries the AMD-SP with the `TSC_INFO` request. The response to this request contains the Guest TSC Scale, Guest TSC Offset, and TSC Factor. The Guest TSC Scale is calculated as the ratio of `DESIRED_TSC_FREQ` to the mean native frequency.

The Guest TSC Offset is an offset added to the guest TSC value on every read. According to official documentation [6], this value is decided by the AMD-SP. In our experiments, we did not receive a Guest TSC Offset other than 0 and we think that this is due to AMD-SP firmware not being ready. When a guest invokes an instruction to get the TSC value, the hardware first scales the TSC value with Guest TSC Scale and then adds the Guest TSC Offset.

As different cores and core complexes have different delays during boot time, especially compared to the bootstrap processor, there is a slight variance on the TSC values they observe. In all our measurements, we found no case with a difference of more than 2^{28} cycles, which corresponds to about 0.13 seconds at a frequency of 2 GHz.

[3] This may involve the AMD Secure Processor (AMD-SP).

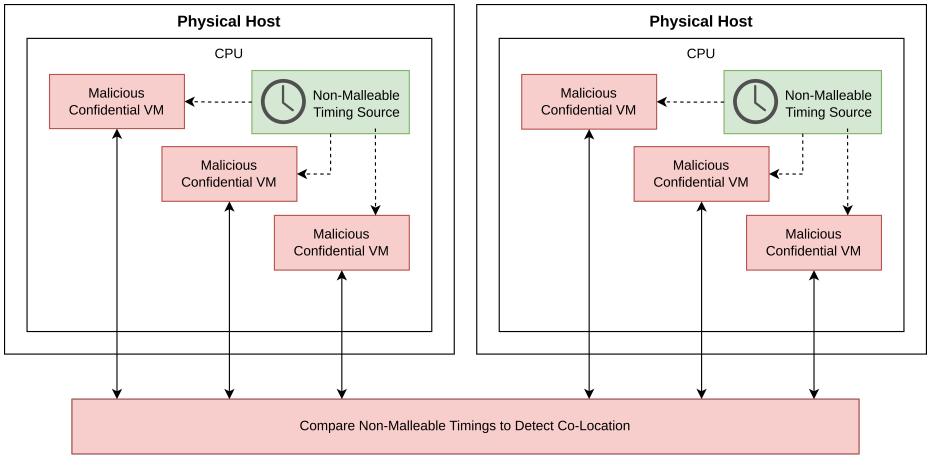

Fig. 1. Overview of our attack. The confidential virtual machines exchange the non-malleable timing information with a remote coordination server that evaluates the data and detects which of the virtual machines are co-located.

3 High-Level Idea

The high-level idea behind our attack is to exploit that servers in a data center have unique uptimes. If precise-enough measurements of the real uptime within the virtual machines are possible it is easy to detect co-located virtual machines by a central coordinator. Servers, even within one rack, are typically not started at precisely the same time but cascaded to avoid load spikes on the data center's power supply. Until now timing sources in virtual machines could easy be randomized by the hypervisor as it can freely adjust the offset and frequency of `rdtsc`, thwart an attack like this. Additionally, the hypervisor already continuously mangles with the `TSC` value to, for example, compensate for virtual machine exists. It was already shown that timing sources in virtual machines are too inaccurate for microarchitectural attacks [44].

Our attack exploits the novel SecureTSC feature, available for AMD SEV-SNP confidential virtual machines that prevents the hypervisor from mangling with the `TSC` value inside a virtual machine while it is running. Figure 1 illustrates our attack with the example of two physical machines in the same data center. The root cause enabling our attack is the availability of a non-malleable timing source, *i.e.*, a non-malleable `rdtsc` instruction and that servers have unique uptimes. Confidential virtual machines with SecureTSC enabled have access to the scale, offset, and value of the timestamp counter at any time. With these values, the confidential virtual machine, or an external system provided with this data, can compute the precise CPU uptime in timestamp counter cycles. Based on the precise CPU uptime, we then determine which virtual machines are co-located on the same physical machine. In contrast to prior work, using one-to-one channels to detect co-location, e.g., cache contention [25, 52, 73], memory bus contention [63, 69], CPU frequency [31] our side channel on SecureTSC is

a one-to-many channel: Instead of requiring time-consuming attempts between each two pairs of virtual machines to establish a channel for co-location detection, a remote coordination server can compare all non-malleable timestamp counter data at once and instantly determine the co-location status for all virtual machines involved. If only the co-location of any two virtual machines is required, the birthday problem shows that only a small number of virtual machines must be rented.

4 Design and Implementation of our Attack

In this section, we present the design and implementation of our attack. We show how an attacker can use values of SecureTSC to determine whether two or more SecureTSC-enabled AMD SEV-SNP virtual machines are co-located. We define the threat model for our attack and present an attack protocol, *TsCupid*, that we use to determine the co-location status of all connected virtual machines at once, using a single remote coordination network server.

4.1 Threat Model

The attacker has the goal to determine which of its confidential virtual machines are co-located on the same physical machine. With this knowledge, the attacker aims to mount subsequent attacks that benefit from the co-location of two or more confidential virtual machines, e.g., attacks on the cache directory which require multiple cores to exhaust the over-provisioned directory [70]. We assume the attacker has the resources to launch hundreds of confidential virtual machines, which are typically billed in short time frames (we evaluate the attack cost in Section 5.5). We also assume that starting a large number of confidential virtual machines eventually leads to co-location of some of them. Furthermore, we assume that an attacker has a coordination server. The coordination server can be one of the virtual machines within the same data center or an remote machine somewhere else.

Owners of AMD SEV-SNP confidential virtual machines can decide whether their machines are allowed to be live-migrated through the `MIGRATE_MA` field in the guest policy, a structure supplied by the guest owner that limits the actions that the hypervisor can take [6]. For this threat model, we assume that the AMD SEV-SNP confidential virtual machines disallow live migration. It must be noted that live migration of these confidential virtual machines is not yet supported [2, 22, 36]. Hence, live migration is no concern for our attack currently, in contrast to prior work, where the resource provisioning system (RPS) may have relocated virtual machines for load balancing [43]. However, we believe that our attack extends to the live-migration scenario as AMD SEV-SNP confidential virtual machines that allow live migration will know that they have been migrated by design [6].

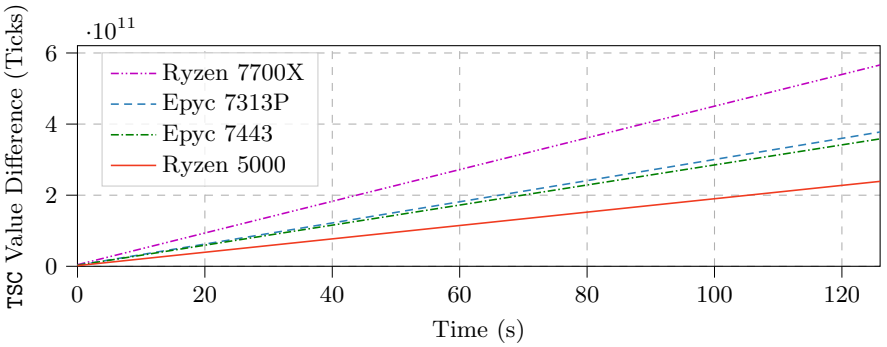

Fig. 2. The difference in `TSC` values reveals that different AMD CPUs increment their `TSC` at unrelated rates. For each machine, we plot the difference of a `TSC` read against the first `TSC` read. This was done on four machines for 128 s where a `TSC` read was performed every second.

4.2 Protocol for Co-location Detection

Basically, our co-location detection based on non-malleable timestamp counter values works by all virtual machines sending their current timestamp counter value to a central server. This central coordination server collects all timestamp counter values and compares them with each other. If two timestamp counter values from different virtual machines are identical, it means that the CPU they are running on has the same uptime and is, therefore, very likely, the same CPU. Hence, the virtual machines are deemed co-located.

4.3 Overview

However, this approach ignores multiple factors that negatively affect the co-location detection:

1. network latency and jitter that is possibly different on the virtual machines;
2. the high frequency of the timestamp counter;
3. the frequency of the timestamp counter can vary on different CPUs (see Figure 2);
4. the hypervisor can also specify a timestamp counter frequency *per* virtual machine (see Figure 3);
5. the timestamp counter offset field can be different *per* virtual machine.

The first point could be addressed by using an NTP-like transmission of the time value. NTP measures the round-trip time between two machines and under the assumption that both directions take approximately the same time, adds half of the round-trip time to the received time value [46]. But this solution does not address the other factors. Hence, we design a protocol, *TsCupid*, to take all five factors into account.

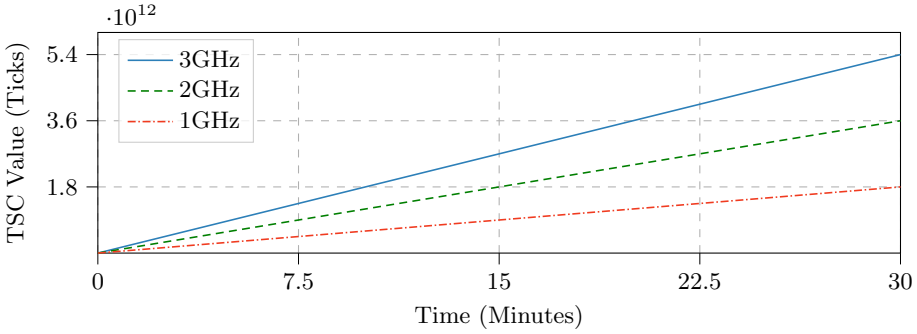

Fig. 3. SEV-SNP virtual machines running with different SecureTSC frequencies defined by the hypervisor. After virtual machine creation, the hypervisor can no longer change the frequency of the SecureTSC. The legend shows the configured frequency in GHz visible to the virtual machine.

At its core, TsCupid collects timestamp counter information, including the value, offset and frequency on a remote coordination server. Based on this information, we can infer a *normalized timestamp* that is offset- and frequency-corrected. Storing the normalized timestamp and the frequency, we can also extrapolate the current normalized timestamp for any of the confidential virtual machines without exchanging further messages. This allows the coordination server to compare the normalized timestamp values across all confidential virtual machines regardless of when they sent in the timestamp counter information.

The normalized timestamps are still affected by the high frequency of the timestamp counter (*i.e.*, multiple increments per nanosecond). Consequently, the coordination server will practically never receive two identical timestamps. Instead, TsCupid uses a threshold to determine which timestamps are to be considered identical and the virtual machines co-located.

4.4 Implementation

The process of co-location detection with TsCupid uses specific messages:

First, the protocol starts with an initialization message (`TSCINIT`) sent by the confidential virtual machine, which contains the timestamp counter frequency and timestamp counter offset. This information is sufficient for the server to establish a record for this confidential virtual machine and to extrapolate and match further messages received.

Second, the confidential virtual machine sends a message (`TSCVALUE`) containing the current timestamp counter value. When receiving the message, the server stores its own local timestamp, as illustrated in Figure 4. The purpose for this is to know precisely what local timestamp corresponds to the timestamp in the confidential virtual machine. However, the timestamp counter value transmitted by the confidential virtual machine cannot be used yet as the network jitter and latency have not been accounted for.

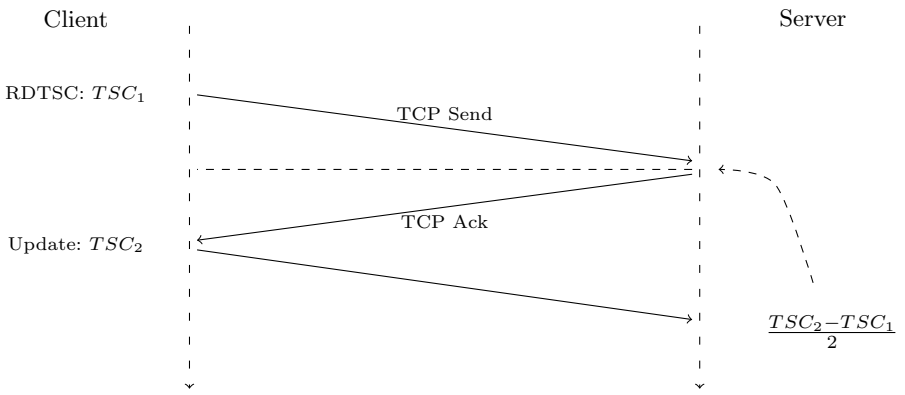

Fig. 4. TsCupid uses a NTP-style timestamp exchange. The round-trip time is taken into account by taking two measurements on the confidential virtual machine and one on the server. This way, the server can compute the precise timestamp counter value on the confidential virtual machine at precisely the point where it received the first message.

As the **last** step, the confidential virtual machine keeps track of when the TCP acknowledgement for the `TSCVALUE` message arrives. Immediately after this acknowledgement, the confidential virtual machine sends a message (`TSCROUNDTRIP`) with the current timestamp counter value to the coordination server. This allows the coordination server to compute the time between the two timestamps from the confidential virtual machine and, thus, infer the round-trip time for the messages. Based on the round-trip time, the attacker can infer the precise point in time between the two timestamps, which is exactly when it stored its own local timestamp, as illustrated in Figure 4.

TsCupid has two further messages to coordinate the confidential virtual machines. First, a message (`TSCMATCH`) to inform co-located machines of their co-location. The server can send this message at any point in time but typically will do so after receiving a `TSCROUNDTRIP` message which resulted in a matching normalized timestamp in its local database. The `TSCMATCH` message contains the number of co-located confidential virtual machines and further information that may be relevant to the co-located confidential virtual machines. Second, a message `TSCWAIT`, which instructs confidential virtual machines to idle for a specified amount of time before providing a new timestamp counter value, e.g., to allow for reduction of noise with multiple measurements.

The coordination server can now compare the normalized timestamps of two confidential virtual machines as follows: Confidential virtual machine A is running at timestamp counter frequency f_A, with the timestamp counter offset o_A, and values $t_{1,A}$ and $t_{2,A}$ transmitted by `TSCVALUE` and `TSCROUNDTRIP`. The coordination server C took the timestamp $t_{A,C}$ when the `TSCVALUE` message arrived, and has its own timestamp counter frequency the current timestamp counter value available as f_C and t_C. Another confidential virtual machine B is running

at timestamp counter frequency f_B, with o_B the timestamp counter offset, and values $t_{1,B}$ and $t_{2,B}$ transmitted by `TSCVALUE` and `TSCROUNDTRIP`. The normalized timestamp $n(A)$ for a confidential virtual machine A is computed as

$$n(A) = \left(\frac{t_{1,A} + t_{2,A}}{2} - o_A\right) + \frac{f_A}{f_C} \cdot (t_C - t_{A,C}) .$$

The coordination server can now compute whether A and B are co-located as

$$|n(A) - n(B)| < \varepsilon,$$

where ε is the detection threshold.

TsCupid stores all data to compute $n(x)$ for all confidential virtual machines in a database. Consequently, the coordination server can simply sort the data by $n(x)$ and compare the neighboring entries for being below the detection threshold ε. If the detection threshold ε is too small, the false negative rate increases, TsCupid does not correctly detect two co-located virtual machines. If ε is too large, the false positive rate increases, TsCupid erroneously labels virtual machines co-located.

5 Evaluation

In this section, we evaluate the effectiveness of our implementation of the *TsCupid* protocol on physical, local AMD SEV-SNP hardware. We show that we can effectively identify co-located SEV-SNP guest virtual machines (clients) by using our protocol. In Section 5.1, we describe our experimental hardware and software setup to study TsCupid. In Section 5.2 we evaluate the impact of network stress on our network's round trip time to aid our further noise evaluation. We run the experiments and discuss the results in Section 5.3. In Section 5.4, we look at sources of noise and challenges of TsCupid. Finally in Section 5.5, we calculate financial costs of deploying TsCupid in order to search for co-located machines.

5.1 Experimental Setup

Although SEV-SNP machines can be created on Google Cloud and Amazon's AWS, they are still in preview and do not support SecureTSC from the hypervisor side. Therefore, we evaluate TsCupid in our local network with a single machine, an AMD Epyc 7313P. The network has a link speed of 1 Gbit/s as we assume that this as a lower bound for real cloud environments. We launch 16 co-located clients with a TSC frequency of 2 GHz on our AMD Epyc 7313P. The central coordination server runs on another machine in the same network.

It is enough to evaluate our co-location detection with only a single AMD machine with only co-located virtual machines. Without SecureTSC support on any large cloud provider we can only estimate the probable differences in uptimes between different machines in a server farm. Therefore, we use a single machine to measure the minimum threshold ϵ required to correctly detect co-located

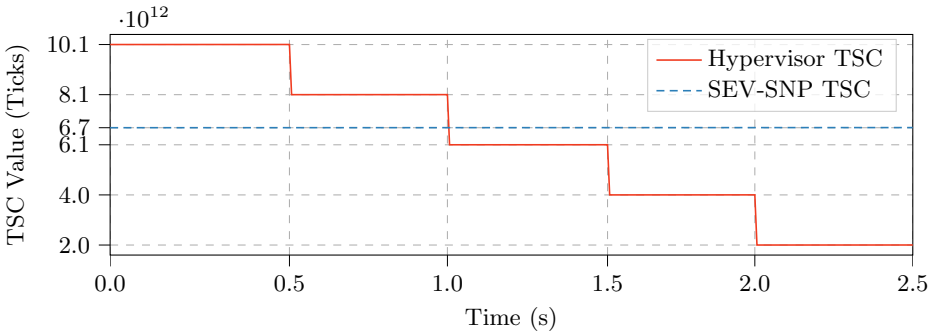

Fig. 5. Changing the `TSC` MSR on the hypervisor has no effect on the SecureTSC value returned by a read to the guest `TSC` MSR. Over a period of 2.5 s, the hypervisor's `TSC` MSR was decreased in intervals of fifths from 10.1×10^{12} ticks to 2.0×10^{12} ticks. The guest's `TSC` MSR stayed unaffected at a value of 6.7×10^{12} ticks.

machines. This corresponds to a co-location rate of 100 % as all our machines are co-located. With this information we can show that our found threshold ϵ of 0.13 ms is lower then the probable difference in machine uptimes. To test the correct functionality of our implementation of the client and server code and rule out any software bugs, we additionally tested our protocol on non-co-located clients and TSCupid correctly identified them as non-co-located.

Messages from the hypervisor take an average of 0.15 ms to reach the server, whereas messages from the clients take an average of 0.45 ms. We test 17 thresholds ε ranging from 8 µs to 0.5 s . For each threshold ε, we run 100 experiments each with and without network stress, totaling 3400 experiments.

We install AMD's kernel patches for the guest [16] and the hypervisor [17], to add SecureTSC functionality. Although the guest patches redirect attempts to read the TSC MSR directly with `rdmsr` by executing `rdtsc`, we found that directly reading the TSC MSR within a guest resulted in the same value returned by executing `rdtsc`. We do not know why AMD chose to implement their guest patches like this. Moreover, the guest TSC MSR is not affected by writes to the hypervisor TSC MSR as is shown in Figure 5 resolving the concerns by Alder et al. [1].

5.2 Impact of Network Stress on Round Trip Time

We run our experiments in two network settings, with and without a network stress program that applies pressure on the network by checking internet download speed from `fast.com`. The network stress program runs natively on the system we attack. The bandwidth to the internet from our local network is high enough to cause significant stress also within the 1 Gbit/s local network. The packet round trip times to Cloudflare's `1.1.1.1` public DNS resolving server are shown in Figure 6. Without network stress, it took an average of 4.14 ms with

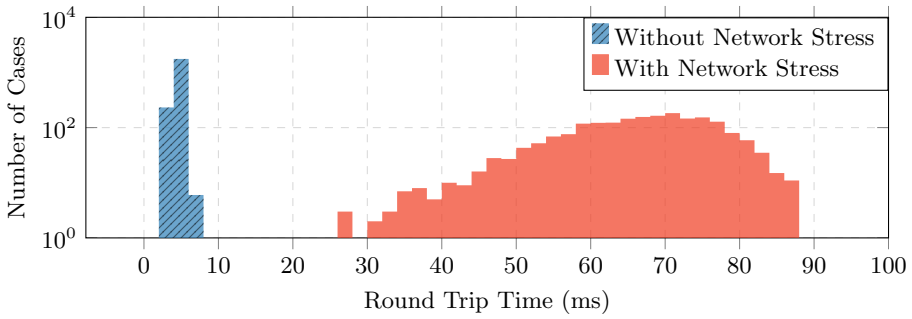

Fig. 6. Network stress has a discernible effect on packet round trip times. For 2000 packets, it took an average of 4.14 ms without network stress to make a round trip to Cloudflare's `1.1.1.1` public DNS resolving server from the system we attack. With network stress, it took an average of 66.46 ms to perform the same round trip.

a standard deviation of 0.31 ms. With network stress, packet round trip times took an average of 66.46 ms and a standard deviation of 10.0 ms.

5.3 Evaluation of TsCupid

We base TsCupid's success, *i.e.*, accuracy based on the percentage of clients where we correctly detected co-located for each threshold. We give TsCupid a maximum of 5 s before aborting the experiment. We provide a secondary metric for the time it takes to determine the maximum number of co-locations in each attempt. Figure 7 shows the percentage of co-locations detected and the time taken to detect them against the 17 thresholds. We determine that network stress has negligible consequences on the number of co-locations detected and some to little effect on the time taken to detect them. We believe that this is due to TsCupid being a minimal protocol with every message (except for `TSCMATCH`) contained in one TCP segment. The `TSCMATCH` message is sent after co-location was detected so network stress has no negative impact on it.

The results of all thresholds are shown in Figure 7. With a threshold ϵ of 0.13 seconds, we achieve 100 % (n=100, $\sigma_{\bar{x}}$=0) detection rate without network stress and 100 % (n=100, $\sigma_{\bar{x}}$=0) detection rate with network stress. This is far lower than typical power-up delays for servers in a data center [14, 24], making our co-location detection very robust.

For low thresholds, the detection time is considerable higher with network stress than without. This can be explained by the high jitter of network packages under high network load as shown in Figure 6. While we tested the ping round-trip time to the internet in those experiments, the jitter seems to be lower inside the network, as the time to detect is already close to 0 with a threshold of 4.2 ms an higher.

Under the assumption that a single AWS data center has 100 000 individual servers [45] and 10 % of them use AMD CPUs that use AMD SEV-SNP with

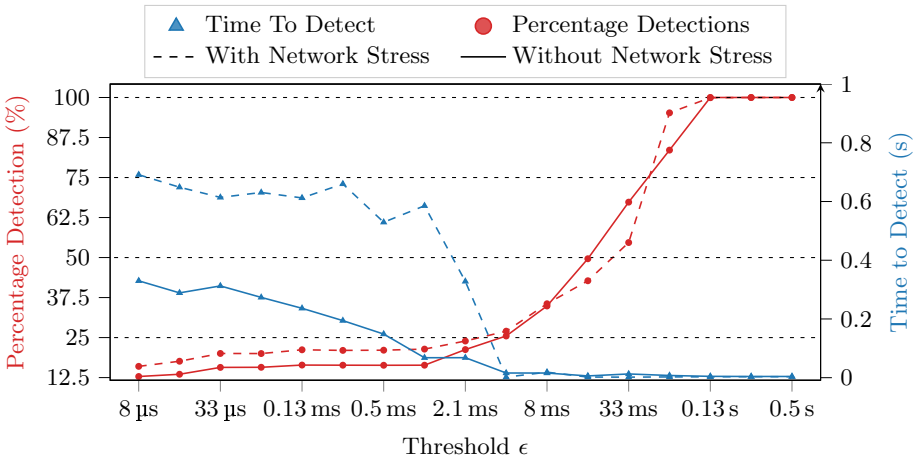

Fig. 7. The success rate of TsCupid with different TSC thresholds. A lower value allows for higher divergence of the TSC values. The detection percentage shows the percentage of all co-located virtual machines that were actually detected after a maximum of 5 s. The time to detect plots the time after which no new co-located virtual machines were detected. Network stress has only a small impact on the time to detect and negligible impact on the detection percentage.

SecureTSC, there are 10 000 target servers. For TSCupid to work two machines must not have an uptime more similar than the chosen threshold ϵ. Further assuming that these server are randomly started over the span of a year, we can calculate the average number of pairs below the threshold the following way:

$$\lambda = \frac{\#\text{ Machines} - 1}{\text{Interval length}} = \frac{9999}{3600\,\text{s} \cdot 24 \cdot 356} = 3.25 \times 10^{-4}\,\text{s}^{-1}$$

$$P(Gap < \epsilon) = 1 - e^{-\lambda \cdot \epsilon} = 1 - e^{-6.5 \times 10^{-4} \cdot 0.13} = 4.23 \times 10^{-5}$$

$$\#\text{ False positives} = P(Gap < \epsilon) \cdot \#\text{ Machines} = 0.42$$

On average there is less than one false positive, another machine that has been booted within 0.13 ms of another machine.

With a higher number of 50 000 target machines, started over the span of a year, there are on average 10.6 false positive machines. However, this is not a problem for an attacker. Following the use of TSCupid, potential co-located machines can use other methods [63, 25] to verify co-location. These one-to-one channels become viable for co-location detection after reducing the number of potentially co-located machines from, e.g., 50 000 to 10.

5.4 Sources of Noise & Challenges

One of the main assumptions of TsCupid is that the network latency between the clients and the server is symmetric, which is normally the case in local networks and within data centers. Based on this assumption we can compute the TSC of the client at the time the receiver received the message, as shown in Figure 4. Apart from an asymmetric network architecture, a potential challenge for an attacker is the hypervisor withholding network packets being sent out or received. A benign hypervisor may even inadvertently do this, as we observed that there were vastly different times between consecutive ping round trip times ranging from 0.1 ms to 4 ms. This is not surprising, as SEV-SNP machines are known to be subject to higher network latency [21]. We address this latency noise problem by running the protocol multiple times, if necessary, until the result is clear, *i.e.*, detection of co-location or non-co-location.

5.5 Financial Costs of TsCupid

On Google Cloud, the cheapest way to rent an AMD SEV-SNP machine is to rent an N2D series machine with two vCPUs and 1GB of RAM running for four hours, with a cost of $0.75. On Amazon's AWS, renting a Linux, M6A.Large instance running for one hour is the cheapest way to rent a machine capable of AMD SEV-SNP, with a cost of $0.105. As discussed in Section 5.3, a TsCupid server takes significantly less than an hour to determine if clients are co-located. Therefore, an attacker would need to spawn an instance, run the client code, and delete the instance within an hour.

If an attacker wants to co-locate any two virtual machines in a data center, they need to know the number of servers in a data center. Under the, for the attacker pessimistic, assumption that a single AWS data center has 100 000 individual servers [45] and half of them use AMD CPUs and support AMD SEV-SNP with SecureTSC, there are 50 000 target servers. Applying the birthday problem, the attacker needs to rent only 264 virtual machines so that any two are co-located with a probability of 50 %. To co-locate two virtual machines with 90 % probability, 480 virtual machines are required. On AWS, this would cost $27.72 and $50.40 respectively to spin up, whereas it would cost $198.00 and $360.00 on Google Cloud.

The probability of co-location further increases when taking into account that new virtual machines are probably not distributed evenly over all servers as some servers may already be completely occupied. This is significantly lower than prior co-location attacks that worked through one-to-one checks, e.g., contention covert channels.

6 Potential Mitigations

The TSC frequency, TSC offset, and `rdtsc` value are the three pieces a guest VM needs to calculate the CPU's up-time — the determiner of co-location. The

simplest way to prevent guests from computing the CPU up-time is to hide the Guest TSC Offset. Since letting the hypervisor control the Guest TSC offset could result in the hypervisor being able to tamper with the guest's `rdtsc` value, the only remaining possibility would be for the AMD-SP to control it. Therefore, we propose that AMD hide the `GUEST_TSC_OFFSET` field entirely.

It is also possible to fingerprint CPUs based on their TSC frequency alone [75]. This is relevant in SEV-SNP guest VMs as the `GUEST_TSC_SCALE` is a ratio of the hypervisor-set `DESIRED_TSC_FREQUENCY` and the mean native frequency, *i.e.*, the native frequency of the TSC on the CPU. Since both `GUEST_TSC_SCALE` and `DESIRED_TSC_FREQUENCY` are available to the guest within its guest context, it is trivial for malicious guest to calculate the mean native frequency and use it to fingerprint based on methods suggested by Zhao et al. [75]. Therefore, we propose that AMD hide either one of `DESIRED_TSC_FREQUENCY` or `GUEST_TSC_SCALE`. Moreover, we suggest that hypervisors use a randomly generated `DESIRED_TSC_FREQUENCY` for each guest that is launched.

One challenge to hiding `GUEST_TSC_OFFSET` and `DESIRED_TSC_FREQUENCY` is its impact on live migration. Before and after migrating, SEV-SNP guest VMs that use the SecureTSC feature are required to update their TSC offset by normalizing the current TSC value (incremented with `GUEST_TSC_FREQUENCY`) to `DESIRED_TSC_FREQUENCY` and adding this value to the current TSC value. This way, a guest doesn't experience a discontinuity in its `rdtsc` value before and after migrating.

Since this is the responsibility of the guest, hiding the `GUEST_TSC_OFFSET` and `DESIRED_TSC_FREQUENCY` would prevent this operation. Our proposal is to have the AMD-SP update the TSC offset before migration and have the guest store the updated value either in its VMSA or guest context. After migrating, the guest can send its offset to the AMD-SP which updates it in a similar fashion to the formula above. We believe that this mitigation isn't technically hard to implement as SEV-SNP introduces the Reverse Map Table, a hardware-supported feature where pages can be owned by the hypervisor, guests, or the AMD-SP. We propose moving the `GUEST_TSC_OFFSET`, `GUEST_TSC_SCALE`, and `DESIRED_TSC_FREQUENCY` to a page owned by the AMD-SP. However, this mitigation is only possible if AMD decides to implement it, as guests of the targeted guest threat model cannot defend themselves otherwise.

7 Conclusion

In this paper, we presented the first co-location detection technique on confidential virtual machine hosts. We showed that the new AMD SecureTSC feature, providing a trusted timing source to confidential virtual machines, can be utilized by an attacker to detect whether two confidential virtual machines are co-located, based on their trusted and non-malleable timestamp counter values. Our minimal network protocol, *TsCupid*, runs fully parallelized and allows to detect co-located machines across hundreds of confidential machines at once. It is the first one-to-many channel for co-location detection that is able to find co-

located virtual machines within 0.13 seconds while requiring only a comparably small number of virtual machines due to the birthday problem. In a data center with 50 000 AMD SEV-SNP machines an attacker only needs to spawn 480 virtual machines to get two co-located virtual machines with 90 % probability. We propose several concrete changes to the SecureTSC feature that would mitigate our attack.

Acknowledgments

We thank our anonymous reviewers for their valuable feedback on this work. We furthermore thank Stefan Gast, Andreas Kogler for valuable feedback and Martin Glasner for the BIOS random boot up delay talks. This research is supported in part by the European Research Council (ERC project FSSec 101076409), and the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85). Additional funding was provided by generous gifts from Red Hat and Google. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

1. Alder, F., Scopelliti, G., Van Bulck, J., Mühlberg, J.T.: About time: On the challenges of temporal guarantees in untrusted environments. In: SysTEX (2023)
2. Amazon AWS: AMD SEV-SNP Considerations (2024), `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html`
3. Amazon AWS: The EC2 approach to preventing side-channels (2024), `https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/the-ec2-approach-to-preventing-side-channels.html`
4. AMD: AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More (2020), `https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf`
5. AMD: AMD64 Architecture Programmer's Manual (2023)
6. AMD: SEV Secure Nested Paging Firmware ABI Specification (9 2023)
7. Anwar, F.M., Garcia, L., Han, X., Srivastava, M.: Securing time in untrusted operating systems with timeseal. In: RTSS (2019)
8. ARM: Security technology building a secure system using trustzone technology (2009), `https://developer.arm.com/documentation/PRD29-GENC-009492/c/TrustZone-Hardware-Architecture`
9. Bates, A., Mood, B., Pletcher, J., Pruse, H., Valafar, M., Butler, K.: On detecting co-resident cloud instances using network flow watermarking techniques. International Journal of Information Security **13**, 171–189 (2014)
10. Betz, J., Westhoff, D., Müller, G.: Survey on covert channels in virtual machines and cloud computing. Transactions on Emerging Telecommunications Technologies (2016)
11. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.R.: Software Grand Exposure: SGX Cache Attacks Are Practical. In: WOOT (2017)

12. Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., Lai, T.H.: SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In: EuroS&P (2019)
13. Chen, S., Zhang, X., Reiter, M.K., Zhang, Y.: Detecting Privileged Side-Channel Attacks in Shielded Execution with DéJà Vu. In: AsiaCCS (2017)
14. Cisco Systems Inc.: Cisco UCS C-Series Servers Integrated Management Controller GUI Configuration Guide for C22 M3, C24 M3, C220 M3 and C240 M3 Servers, Release 3.0 (2024), https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/c/sw/gui/config/guide/3_0/b_Cisco_UCS_C-series_GUI_Configuration_Guide_301/b_Cisco_UCS_C-series_GUI_Configuration_Guide_201_chapter_011.html#d71727e3886a1635
15. Costan, V., Devadas, S.: Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086 (2016)
16. Dadhania, N.A.: [PATCH v7 00/16] Add Secure TSC support for SNP guests (2023), https://lore.kernel.org/all/20231220151358.2147066-1-nikunj@amd.com/
17. Dadhania, N.A.: SecureTSC Hypervisor Patches (2023), https://github.com/nikunjad/linux/tree/snp-host-latest-securetsc_v5
18. Dai, W., Jin, H., Zou, D., Xu, S., Zheng, W., Shi, L.: TEE: A Virtual DRTM Based Execution Environment for Secure Cloud-End Computing. In: CCS (2010)
19. Du, Z.H., Ying, Z., Ma, Z., Mai, Y., Wang, P., Liu, J., Fang, J.: Secure encrypted virtualization is unsecure. arXiv:1712.05090 (2017)
20. Ge, X., Vijayakumar, H., Jaeger, T.: Sprobes: Enforcing kernel code integrity on the trustzone architecture. In: Workshop on Mobile Security Technologies (MoST) (2014)
21. Google: Confidential Computing concepts (2024), https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview
22. Google: Confidential Computing: Supported configurations (2024), https://cloud.google.com/confidential-computing/confidential-vm/docs/supported-configurations
23. Hetzelt, F., Buhren, R.: Security analysis of encrypted virtual machines. ACM SIGPLAN Notices **52**(7), 129–142 (2017)
24. HP: Setting the power-on delay (2024), https://support.hpe.com/hpesc/public/docDisplay?docId=sd00001068en_us&page=GUID-D7147C7F-2016-0901-0A72-000000000D92.html
25. Inci, M.S., Gulmezoglu, B., Eisenbarth, T., Sunar, B.: Co-location detection on the cloud. In: COSADE (2016)
26. Inci, M.S., Gulmezoglu, B., Irazoqui, G., Eisenbarth, T., Sunar, B.: Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud. Cryptology ePrint Archive, Report 2015/898 (2015)
27. Intel: Intel Trust Domain Extensions (2023), https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html
28. Intel: Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide (2024)
29. Intel: Intel Trust Domain Extensions Module Base Architecture Specification (2024), https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html
30. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Cross-VM Side Channels and Their Use to Extract Private Keys. In: Big Data and Cloud Computing (2014)
31. Kalmbach, M., Gottschlag, M., Schmidt, T., Bellosa, F.: TurboCC: A Practical Frequency-Based Covert Channel With Intel Turbo Boost. arXiv:2007.07046 (2020)

32. Kaplan, D., Powell, J., Woller, T.: AMD Memory Encryption (2016)
33. Kim, T., Peinado, M., Mainar-Ruiz, G.: StealthMem: system-level protection against cache-based side channel attacks in the cloud. In: USENIX Security (2012)
34. Kogler, A., Juffinger, J., Giner, L., Gerlach, L., Schwarzl, M., Schwarz, M., Gruss, D., Mangard, S.: Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels. In: USENIX Security (2023)
35. Lee, J., Jang, J., Jang, Y., Kwak, N., Choi, Y., Choi, C., Kim, T., Peinado, M., Kang, B.B.: Hacking in Darkness: Return-oriented Programming against Secure Enclaves. In: USENIX Security (2017)
36. Lendacky, T.: QEMU not working with virt-install (2022), `https://github.com/AMDESE/qemu/issues/3#issuecomment-1171302037`
37. Li, M., Wilke, L., Wichelmann, J., Eisenbarth, T., Teodorescu, R., Zhang, Y.: A systematic look at ciphertext side channels on AMD SEV-SNP. In: S&P (2022)
38. Li, M., Zhang, Y., Lin, Z., Solihin, Y.: Exploiting unprotected {I/O} operations in {AMD's} secure encrypted virtualization. In: USENIX Security (2019)
39. Li, M., Zhang, Y., Wang, H., Li, K., Cheng, Y.: CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In: USENIX Security (2021)
40. Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., Mangard, S.: ARMageddon: Cache Attacks on Mobile Devices. In: USENIX Security (2016)
41. Lipp, M., Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., Gruss, D.: PLATYPUS: Software-based Power Side-Channel Attacks on x86. In: S&P (2021)
42. Liu, F., Ge, Q., Yarom, Y., Mckeen, F., Rozas, C., Heiser, G., Lee, R.B.: Catalyst: Defeating last-level cache side channel attacks in cloud computing. In: HPCA (2016)
43. Makrani, H.M., Sayadi, H., Nazari, N., Khasawneh, K.N., Sasan, A., Rafatirad, S., Homayoun, H.: Cloak & Co-locate: Adversarial Railroading of Resource Sharing-based Attacks on the Cloud. In: Secure and Private Execution Environment Design (SEED) (2021)
44. Maurice, C., Weber, M., Schwarz, M., Giner, L., Gruss, D., Alberto Boano, C., Mangard, S., Römer, K.: Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In: NDSS (2017)
45. Miller, R.: Inside Amazon's Cloud Computing Infrastructure (2015), `https://www.datacenterfrontier.com/design/article/11431484/inside-amazon8217s-cloud-computing-infrastructure`
46. Mills, D., et al.: Network Time Protocol. Tech. rep., RFC-958, M/A-COM Linkabit (1985)
47. Morbitzer, M., Huber, M., Horsch, J., Wessel, S.: Severed: Subverting AMD's virtual machine encryption. In: EuroSec (2018)
48. Morbitzer, M., Proskurin, S., Radev, M., Dorfhuber, M.: SEVerity: Code Injection Attacks against Encrypted Virtual Machines. In: WOOT (2021)
49. Oleksenko, O., Trach, B., Krahn, R., Silberstein, M., Fetzer, C.: Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks. In: USENIX ATC (2018)
50. Pfarr, F., Buckel, T., Winkelmann, A.: Cloud computing data protection – a literature review and analysis. In: HICSS (2014)
51. Qiu, P., Wang, D., Lyu, Y., Qu, G.: VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-core Frequencies. In: CCS (2019)
52. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: CCS (2009)

53. Ryan, K.: Hardware-Backed Heist: Extracting ECDSA Keys from Qualcomm's TrustZone. In: CCS (2019)
54. Schwarz, M., Gruss, D., Weiser, S., Maurice, C., Mangard, S.: Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA (2017)
55. Schwarz, M., Lipp, M., Moghimi, D., Van Bulck, J., Stecklina, J., Prescher, T., Gruss, D.: ZombieLoad: Cross-Privilege-Boundary Data Sampling. In: CCS (2019)
56. Schwarz, M., Weiser, S., Gruss, D.: Practical Enclave Malware with Intel SGX. In: DIMVA (2019)
57. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S.: Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA (2017)
58. Shringarputale, S., McDaniel, P., Butler, K., La Porta, T.: Co-residency attacks on containers are real. In: CCSW (2020)
59. Sule, M.J., Li, M., Taylor, G.: Trust modeling in cloud computing. In: SOSE (2016)
60. Sullivan, D., Arias, O., Meade, T., Jin, Y.: Microarchitectural Minefields: 4K-aliasing Covert Channel and Multi-tenant Detection in IaaS Clouds. In: NDSS (2018)
61. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In: USENIX Security (2018)
62. Van Bulck, J., Moghimi, D., Schwarz, M., Lipp, M., Minkin, M., Genkin, D., Yuval, Y., Sunar, B., Gruss, D., Piessens, F.: LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In: S&P (2020)
63. Varadarajan, V., Zhang, Y., Ristenpart, T., Swift, M.: A Placement Vulnerability Study in Multi-Tenant Public Clouds. In: USENIX Security (2015)
64. Wang, W., Li, M., Zhang, Y., Lin, Z.: PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In: DIMVA (2023)
65. Weichbrodt, N., Kurmus, A., Pietzuch, P., Kapitza, R.: AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves. In: ESORICS (2016)
66. Werner, J., Mason, J., Antonakakis, M., Polychronakis, M., Monrose, F.: The severest of them all: Inference attacks against secure virtual enclaves. In: AsiaCCS (2019)
67. Wilke, L., Wichelmann, J., Morbitzer, M., Eisenbarth, T.: SEVurity: No Security Without Integrity–Breaking Integrity-Free Memory Encryption with Minimal Assumptions. In: S&P (2020)
68. Wu, Z., Xu, Z., Wang, H.: Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In: USENIX Security (2012)
69. Wu, Z., Xu, Z., Wang, H.: Whispers in the Hyper-space: High-bandwidth and Reliable Covert Channel Attacks inside the Cloud. ACM Transactions on Networking (2014)
70. Yan, M., Sprabery, R., Gopireddy, B., Fletcher, C., Campbell, R., Torrellas, J.: Attack directories, not caches: Side channel attacks in a non-inclusive world. In: S&P (2019)
71. Zhang, N., Sun, K., Shands, D., Lou, W., Hou, Y.T.: TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices. IACR Cryptology ePrint Archive, Report 2016/980 (2016)
72. Zhang, T., Zhang, Y., Lee, R.B.: CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. In: RAID (2016)
73. Zhang, Y., Juels, A., Oprea, A., Reiter, M.K.: HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In: S&P (2011)

74. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-Tenant Side-Channel Attacks in PaaS Clouds. In: CCS (2014)
75. Zhao, Z.N., Morrison, A., Fletcher, C.W., Torrellas, J.: Everywhere All at Once: Co-Location Attacks on Public Cloud FaaS. In: ASPLOS (2024)