CPU Elasticity to Mitigate Cross-VM Runtime Monitoring

Zeyu Mi, Haibo Chen, Yinqian Zhang, Shuanghe Peng, Xiaofeng Wang, and Michael Reiter

Abstract—In this paper, We present a new technique that offers lightweight, general, and elastic protection against Crum (Cross-VM runtime monitoring) attacks. Our protection, called CREASE (CPU Resource Elasticity as a Service), enables a VM (called *principal*) to purchase a higher clock rate from the cloud, through lowering the frequency of a malicious VM (called *peer*), to support its security-critical operations within a short period. During that period, the weakened peer becomes unable to catch up with the pace of the strengthened principal, therefore losing the capability to effectively collect its sensitive information. In the meantime, our approach can also make up for the performance impact on the peer through refunding schedule credits or service credits, in line with the service level agreement of today's cloud. At the center of our design is the novel application of on-demand frequency scaling and schedule quantum randomization, together with a situation-awareness mechanism that dynamically assesses the security risk posed by the peer. We analyzed the security guarantee of our design, implemented a prototype and evaluated it on a well-known Crum attack (an LLC side-channel attack) and various workloads. Our study shows that CREASE is effective at protecting the principal, with only a small impact on the peer's operations.

Index Terms—Cross-VM runtime monitoring, Defense, Cloud environment, CPU resource elasticity.

1 INTRODUCTION

With the rapid growth of public cloud services, the ways these systems protect their customers' information assets are increasingly under scrutiny. At the core of their security mechanisms are virtual machine (VM) based isolation, which separates different users' operations from each other. This protective partition, however, has been found to be penetrable by a series of studies: an attack VM can be strategically placed on the same physical machine running the target VM and the attacker can leverage a set of *side channels* to extract private information from the target. Most damaging among those attacks are a category of *Cross-VM Runtime Monitoring* (*Crum*) exploits, in which the attack VM continuously collects information from the target when it is running a program involving security-critical data like AES or RSA keys. What has been discovered is that such sensitive information can be inferred through sampling the caches within a processor [57], [58], [7], [25], [36], [23], [21], cross-processor channel [22] or memory bus [49] in the program runtime. Even more concerning is the report that the attack can be executed in only a few minutes [25].

Fighting Crum: challenges. Mitigating the Crum threat is hard, often requiring significant changes to hardware, operating systems (OS) or the software under the threat. More specifically, hardware solutions include cache partitioning [41], [16], [50], [51], [34], access randomization [51], [52], [35], cache decay interval randomization [27], inaccurate time keeping [37], and memory-trace obliviousness [33]. OS-based solutions include OS-supported cache partition [43], [46], [28], fine-grained timer elimination [48], enforced deterministic execution [8], [32], random noise injection [59], security-aware scheduling algorithms [47], and on-demand migration of protected VMs [39]. Other approaches aim at transforming a program to remove secret-dependent information flows [38], [13], [3], obfuscating execution traces by adding decoy execution [44], diversifying software programs for probabilistic defense [14] or detecting side-channel vulnerabilities via static analysis [17].

Despite sustained interest in the problem, none of the existing solutions demonstrate a huge potential for practical adoption. Most are either narrow, focusing on a specific type of threat (e.g., certain types of cache attacks), or one-size-fits-all, requiring platform changes that protect—and impose overheads on—all tenant VMs regardless of their need for it. While exceptions exist, those tend to be potentially burdensome to the tenants who adopt them, e.g., involving heavyweight program transformations [38], [13] or potentially disruptive VM migrations [39]. We therefore argue that there is a need for a Crum defense that is *general* in applying to a broad range of Crum threats; *elastic* in permitting only tenants who perceive a need for protection to pay for it, and only when they need it; and *lightweight*, so that adopting protection does not subject the tenant to substantial overheads.

Resource elasticity for protection. In this paper, we present a novel side-channel mitigation technique that we believe meets this need. Our approach, called CREASE (CPU Resource Elasticity as a Service), allows a VM running a critical task to purchase from the cloud provider a higher clock rate for a short period, before the task is completed, by lowering down the clock rate of its peer, another VM sharing the same machine, which is considered

- *Zeyu Mi and Haibo Chen are with Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University, Shanghai, China. E-mail: {yzmizeyu, haibochen}@sjtu.edu.cn. Contact author: Haibo Chen*
- *Yinqian Zhang is with Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA. E-mail: yinqian@cse.ohio-state.edu.*
- *Shuanghe Peng is with School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. E-mail: shhpeng@m.bjtu.edu.cn.*
- *Xiaofeng Wang is with Department of Computer Science, Indiana University at Bloomington, Bloomington, IN, USA. E-mail: xw7@indiana.edu.*
- *Michael Reiter is with Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA. E-mail: reiter@cs.unc.edu*

to behave suspiciously. The key idea here is that by temporarily speeding up the target VM and slowing down the attacker, the amount of information derivable from the side channels goes down quickly, due to the reduction in the attacker's sampling rate and the difficulty in staying synchronized with the target (which is made even harder by randomizing the scheduling quanta that are assigned). In the meantime, we can keep the cost of the protection low for the service user (e.g., just buying more CPU cycles for a few seconds) and easily handle the impact on the suspicious peer in line with the current service level agreements (SLAs) provided by today's commercial clouds: e.g., Amazon's SLA promises a monthly uptime percentage of 99.95% and when this cannot be met, service credits will be given to the user [5]. Using CREASE, the provider can easily stay profitable by setting the price for the protection service above the credits refunded to the owner of the peer when it is obliged to make up for the peer's performance degradation.

The approach can be made more effective when the target is aware of the runtime situation of the peer: e.g., whether the latter is aggressively probing the shared cache, as reported by the hypervisor. Note that this is *not* a detection mechanism. It is not meant to determine whether indeed the peer VM is malicious. Instead, all we want to do here is to inform the target of potential risks it faces, helping it decide on the amount of resources (here clock rate) it needs to acquire in order to hedge against the possible harm.

We analyzed the design of CREASE using the classic model of deletion-substitution channel, which demonstrates that the approach significantly reduces the side-channel leaks. We further implemented the design based on Xen and evaluated it against a known Crum attack (LLC cache attack), by studying the effectiveness of the elastic protection at different resource levels (clock rate and scheduling quanta), in terms of its impact on the accuracy of the data the adversary is capable of inferring cross-VM. Together with the situation data gathered from the PMU (performance monitoring unit), this information helps the target weigh the risk against the cost to choose a right level of protection. Our evaluation shows that CREASE can easily defeat the attack, with better performance (about 12.6% to 13.8%) brought to the target VM due to boosted frequency. For the suspicious VM under control, we found that it can still effectively serve its clients, with different levels of performance impacts (about 11.7% to 64.4% in a worst case), depending on the types of services (i.e., PARSEC [12] and CloudSuite [18]) and the level of protection requested. Important here is the fact that most CPU resource is not wasted in our approach, which is either spent on the target for having its job done quickly or used by the peer for serving its client. Besides, when there is no protection requested, CREASE incurs very low overhead (0.63% to 1.93%).

Contributions. The contributions of the paper are outlined as follows:

- *Elastic side-channel protection*. We propose a general Crum defense that is elastic, permitting the cloud user to purchase different levels of protection, at different cost, according to her perception of the threat from its co-resident VMs.
- *Clock-rate based, situation-aware techniques*. Our defense is novel in dynamically adjusting the target VM and its suspicious peer's clock rates to make a side-channel attack ineffective. This approach is further used together with a situation-assessment technique that evaluates whether the behavior of a target's peer (another VM) is risky enough to justify activating the protection, which helps minimize the performance impacts on both the target and the peer.
- *Implementation and evaluation*. We implemented our design on the Xen hypervisor [10] and evaluated it in a realistic cross-VM setting. Our study shows that the new techniques can effectively defeat known Crum exploits and incurs small overheads.

Roadmap. The rest of the paper is organized as follows: Section 2 provides the background information about our research; Section 3 elaborates the design and implementation of CREASE; Section 5 presents a security analysis of CREASE; Section 6 describes the evaluation of our technique; Section 7 discusses the limitations of our current system and potential future research; Section 8 reviews related prior research and Section 9 concludes the paper.

2 BACKGROUND

Elastic cloud business model. A key aspect of the cloud business model is that cloud vendors charge customers based on the capacity of the VMs they rent, instead of their actual resource usage. For example, Amazon EC2 prices its VM instances according to the different levels of computing resources they offer (CPU, memory and storage) [4], e.g., c4.large, m3.xlarge, d2.2xlarge, etc. The resource level here is the maximum capacity of these instances, which typically will *not* be reached during their daily operations. The gap between these two ends is exploited by the cloud vendor to make profit: oftentimes, the aggregated capacity of the VMs rented out far exceeds the computing power of the physical machine that hosts them. This property allows us to raise the clock rate of the VM under protection (called *principal*) and lower that of the peer VM, without degrading the performance of the service the peer provides most of the time.

In the case that the peer indeed runs CPU-intensive tasks, such as scientific computing, the additional resource acquired by the principal can have an observable impact on the peer's performance. This, however, fits squarely in the cloud's business model. Except the top-notch dedicated hosting service the cloud offers, all other service types are more or less best effort: for example, Amazon EC2 provides *spot instances* [6] with little guarantee about the resource availability; other non-dedicated Amazon instances are also expected to see occasional service disruption due to cross-VM interference and cloud operations (like maintenance). As a matter of fact, the SLAs of large cloud vendors only focus on the accessibility of VM instances and the guarantee of these instances' performance has never been mentioned. Ben-Yehuda et al. [1] presented a detailed deconstruction of Amazon's pricing model, with a special focus on spot instance.

Even when the peer's performance has been degraded to the level that its accessibility becomes an issue, another mechanism the vendors can use to make up for this temporary loss of the service is "service credits" [5], which is used to repay the VM's renter for the disruption of her service. This enables the vendor to reallocate the computing resources to the principal at a higher price compared with the service credits given back to the peer, when such a resource transfer becomes critical for protecting the principal's sensitive data.

Cross-VM runtime monitoring (Crum). In this paper, we aim to defend against a set of cross-VM side-channel attacks, which we call Crum attacks. In a Crum attack, the adversary controls a guest VM and instruments it to (1) interact *repeatedly* and *frequently* with one or multiple hardware or software resources shared with the principal, and (2) optionally interact with the principal's external interface (e.g., webserver requests) to trigger sensitive operations that make uses of the security-critical data like AES or RSA keys. The key assumption behind the Crum attacks is that although the adversary is assumed to have the ability to *synchronize* with the principal's sensitive computation via external interfaces, he needs to collect fine-grained side-channel observations that require repeated and frequent resource accesses.

Crum is broadly defined to include a large set of cross-VM side-channel attacks. For example, attacks using PRIME+PROBE [40], FLUSH+RELOAD [55] and FLUSH+FLUSH [20] on last-level caches (LLCs) are all examples of Crum attacks in which the adversary interacts with a shared LLC in ways that lead to information leakage. In fact, all known cross-VM side-channel attacks to date [57], [58], [7], [25], [49], [36], [23], [21], [22] can be described as a form of Crum attacks. However, as per-core side-channel attacks have been addressed effectively by minimum run time [47] that is already integrated into Xen schedulers, the focus of this paper is cross-core attacks that are largely unaddressed.

Threat model and assumptions. In this paper, we assume that the adversary is capable of co-locating his attack VM with the principal on the same physical machine, and also conducting Crum attacks to collect side-channel observations which will later be used to infer sensitive information of the victim. The cloud provider, together with its controlled hypervisor and hardware, are considered as trusted. The main goal of CREASE is to protect a VM's security-critical information (like cryptographic keys) from being leaked through cross-VM monitoring. However, CREASE does not aim to protect other covert information like *when* a principal VM is doing some critical operations, which barely adds to the principal's knowledge and has not been demonstrated as an effective way to steal security-critical information.

3 DESIGN

In this section, we present the technical details of CREASE, including its high-level idea and designs of its individual components.

3.1 Overview

Idea and architecture. As mentioned earlier, the key idea of our defense against the runtime monitoring attacks is to enable the principal to pay for a temporary reallocation of CPU resources, making its security-critical operations run faster while the monitoring activity from its suspicious peer, if any, goes slower. What is expected is that on one hand, the resource gap between the two parties renders a Crum attack hard to succeed (due to the reduction of the sampling rate and the increasing difficulty in keeping the attacker synchronized with the ongoing secret operations), on the other hand the computing power left with the peer can still sustain its legitimate service with reasonable performance. The latter is important because the less disruptive it could be, more affordable the protection becomes and more likely the technique will be adopted by the cloud service provider.

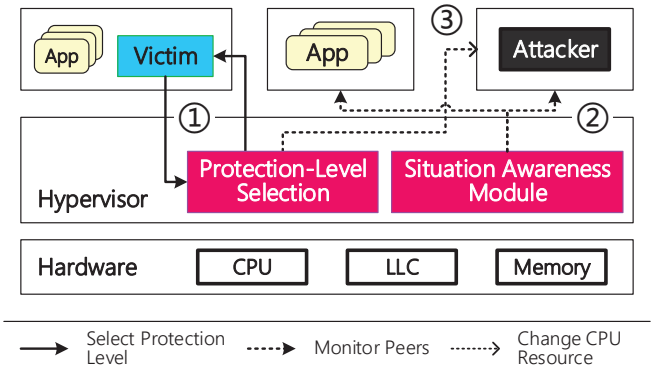


Figure 1: An overview of the architecture of CREASE.

For this purpose, we come up with a design as illustrated in Figure 1. At the center of the design is protection-level selection (PLS), through which the principal communicates with the hypervisor to identify the runtime situation of its peer, together with the prices and effectiveness of different levels of protection (reallocation of clock rate and quantum) it can purchase, and decide the protection it needs to use (Section 3.4). The protection level here is achieved through an on-demand CPU resource allocator (CRA) that performs dynamic CPU frequency scaling and quantum randomization across different cores (Section 3.2). To minimize the cost of running this security mechanism, a situation awareness module (SA) continuously collects and analyzes the information from the peer, such as the frequency of its cache use, helping the principal understand the security risk the peer poses (Section 3.3). Note that once the principal obtains the service, all other VMs on the same CPU socket are not allowed to speed up. This prevents the adversary from exploiting the mechanism to attack the principal, which shall only run security-critical operations upon acquiring a certain protection level.

An example. As an example, consider that the principal needs to run AES to encrypt a message. To protect its secret key against cache-channel attacks, the principal first talks to the hypervisor through its PLS component, which requires the hypervisor to continuously update to the principal the risks presented by the peer. To this end, the hypervisor runs the SA to gather the performance information from the peer, such as `LLC-load-misses` (measurement of the cache misses happening to the last level cache). Such information is further analyzed by a classifier that determines how likely the peer is indeed probing the principal's cache lines. The estimated risk and the prices for different protection levels are then continuously updated to the principal through the PLS. Once the principal decides to use a protection level (described by its own clock rate and quanta, and the peer's clock rate and quanta), the service is immediately purchased from the cloud provider through PLS APIs. These parameters are then enforced by the CRA during the execution of the encryption using a technique that combines real-time frequency scaling with CPU quantum adjustments.

3.2 On-Demand CPU Resource Allocation

Based on the observation that a VM's capacity planning is usually pessimistic to tolerate a possible load burst, the CRA dynamically steals the unused CPU power to boost the computation of a principal VM executing security-critical tasks. In this way, a principal can end up having much more computing power than

its malicious peer and thus the peer can hardly observe the correct sequence of secrets through cross-VM runtime monitoring. Here we elaborate the techniques that make this happen.

Dynamic clocking. The CRA enhances the principal's computing power by leveraging the dynamic overclocking feature in recent x86 multi-core processors. The basic assumption is that the thermal design power (TDP) of a CPU is the worst case power under the maximum workload and thus there is usually idle power that could be utilized to overclock a CPU core to a higher frequency. Even if there is no additional power available within a chip, one can manually decrease the voltage and frequency (and thus power) in some cores to provide headroom to overclock other cores. Dynamic overclocking is available on both Intel and AMD processors, which are called "Turbo Boost" and "Turbo Core" respectively (we use Intel's notion throughout this paper). For example, the basic frequency of our test processor is 2.3GHz and Turbo Boost allows a core to be accelerated to 3.0GHz or decelerated to 1.2GHz, creating a computing capability gap as large as 2.5 times.

Using such a feature, the CRA aims at empowering the security-critical VM (principal) to outrace its malicious peers, the attacking VMs, thereby thwarting any Crum attempt. This effort, however, turns out to be more complicated than it appears to be. Specifically, the clock rate of a CPU core *cannot* be directly elevated by the cloud provider: all one can do is just setting an upper limit on the overclocked frequency of a core, whose real speed is determined by a feedback controller during the runtime according to the available power budget and the workload of the core. In our experiment on an Intel Xeon CPU E5-2650 v3 processor, we found that the maximum frequency 3.0Ghz could only be reached when no more than 2 (out of 10) cores were running under the full loads (i.e., 100% vCPU usage); when there were five, only 2.6Ghz could be attained. On the other hand, Turbo Boost can slow down the peer by setting a lower frequency limit on its core (below the base frequency, 2.3Ghz in our experiment). Our strategy is to dynamically reduce the frequency of the peers to provide extra power budget for overclocking the principal. As security-critical operations like AES and RSA encryptions are usually CPU-hungry, the principal's core can usually be boosted to the maximum allowable frequency.

To select the peers for this purpose, our approach takes into account the risks posed by individual VMs and their workload. Specifically, those likely running an attack (based upon the way they use vCPUs) are further slowed down. Then, when more power budgets are still needed, the CRA looks into the performance profiles of other VMs to decide which ones can be scaled down without a significant impact on their operations (e.g., those not requiring real-time responsiveness). Our study shows that this strategy works effectively in speeding up the principal without exceedingly wasting the peer's computing resources in the evaluation section. In the meantime, it is important to note that when necessary, the peer can be further slowed down to serve the principal's security need.

Our current design can only speed up a single VM at a time. This strategy is put in place to defeat the attempt made by a malicious VM to also raise its clock rate for catching up with the principal under protection. In the case that multiple VMs within a CPU socket have legitimate needs to overclock at the same time, CREASE can handle their requests in a FIFO manner, or ask the VMs to bid for the extra CPU resource, or migrate a VM to another socket.

Dynamic quantum randomization. Dynamic overclocking creates a computing power gap between a principal VM and its peers such that an adversary can observe much less secret information. CRA further leverages dynamic quantum randomization to disturb the sequence alignment of secrets from the adversary.

A cloud provider usually overcommits resources (e.g., CPU) [11] for profit and thus there are multiple VMs running concurrently atop a physical CPU. However, commodity hypervisors like Xen and KVM usually use a mostly fixed scheduler quantum for its VMs (e.g., every 30ms for Xen). This could enable the malicious peer to synchronize its actions with the operations within the principal VM, a step critical for cross-VM information collection, e.g., determining the positions of the bits stolen from the principal's key. To make such a synchronization hard to succeed, the CRA further disturbs the peer's execution by randomizing the scheduling quantum of its virtual CPU core (vCPU). Specifically, our approach further refines the base scheduling quantum (e.g., 1ms in our case) and dynamically sets a random frequency (uniformly distributed between 1ms and 30ms) to the peer's vCPU at the end of a schedule quantum.

A side effect from the quantum randomization is that a benign VM co-running with an adversary VM will also suffer from some randomness of its scheduling timings. However, the performance impact of this randomization is limited, since the randomizing period is usually short (i.e., a principal VM performing security-critical operations). A further note is that using such a fine-grained schedule quantum does not necessarily lower the performance of a VM. Actually, recent studies show that the schedule quantum with a fine granularity may sometimes even significantly improve the performance of some workloads [53], [54], [2], due to better cache locality, reduced spinning time and faster responsiveness.

Note that our dynamic quantum randomization is fundamentally different from the minimum run time (MRT) guarantee used in [47], though both alter Xen scheduling quantum for side-channel protection. The MRT guarantee prevents CPU preemption, thus defeating L1 cache attacks where both the adversary and the victim are scheduled on the same core. Dynamic quantum randomization aims to desynchronize the adversary and the principal while they run on different cores.

Compensation. Dynamically boosting the execution of a principal VM may impact the performance of its peers. Such a performance loss can be compensated by CREASE, either through refunded "service credits" [5], as mentioned before, or through reallocation of CPU resources when available, particularly in the case that the service disruption is brief. Specifically, commercial hypervisors' schedulers have the capability of temporarily giving a vCPU a higher priority and longer schedule quantum and later making up for the slowdowns experienced by other vCPUs. For example, Xen's default Credit scheduler uses *credits* to determine the priority of a vCPU: if the remaining credits go below a threshold, the vCPU gets a low priority; otherwise, it gets a high priority and may be boosted with more schedule quantum. We integrated this mechanism into CREASE, which, after temporarily elevating the principal's resource level for protecting its security-critical operations, automatically repays affected peer VMs with more credits so that their performance could go up later. This approach

enables the cloud provider to make use of idle CPU cycles to compensate peers' performance losses, thereby saving the "service credits" it needs to give back.

Putting pieces together. With the integration of the dynamic CPU frequency scaling and schedule quantum randomization, the CRA can not only make it hard for the malicious peer to adequately sample the principal's operations during a Crum attack, but also prevent it from synchronizing its information gathering actions with the changes of the secret within the principal VM. Further, the compensation mechanism makes use of idle CPU cycles to compensate the peer's performance loss caused by the resource allocation protection, reducing the overhead of the whole approach.

3.3 Situation Awareness

Like any other security mechanisms, allocating CPU resources for protection does come with some cost, particularly, the performance loss that may happen to benign peers and the fee for purchasing the protection service. To keep the cost low, it is important to determine which protection level would be sufficient. For this purpose, CREASE includes a situation awareness module that continuously assesses the security risk posed by the peer, which is used by the PLS to decide whether more CPU resources need to be purchased to hedge against the risk. Note that the risk assessment here is *not* meant to accurately detect a Crum attack. Instead, it just uses a set of *necessary* conditions of the threats to filter out the situations where attacks are unlikely to happen, thereby reducing the cost for the protection.

To this end, the SA module continuously monitors the guest VMs for the signs of risky behaviors. Given the diversity of the Crum attacks, finding a generic behavior pattern to capture all these attacks is hard. Therefore in our research, we focus on a set of cross-VM cache side-channel attacks, including PRIME+PROBE [40], FLUSH+RELOAD [55] and FLUSH+FLUSH [20]. Should new types of side-channel attacks be discovered, the current scheme may be extended accordingly. In particular, these cache attacks will inevitably introduce anomalies in cache operations, on the instruction/data translation lookaside buffer (TLB), last-level cache (LLC) and others, which are documented by the performance monitoring units (PMU) within commercial processors. The SA module is designed to continuously gather the PMU events reported by the processor for each VM at the hypervisor level and analyze these events to identify potential security risks. For example, once the peer is found to cause a lot of LLC misses, suspicion may arise about whether it is performing a PRIME+PROBE attack. In practice, however, such a simple approach can be susceptible to the dynamics of workloads. Later in the section, we describe a more robust alternative.

Multi-grained event gathering with piggybacked polling. A challenge for using PMU for online monitoring is how to balance the granularity of the information collected and the overhead incurred. Specifically, PMU runs an interrupt mechanism to report events: when the number of accumulated events exceeds a predefined threshold, a PMU interrupt is issued. However, PMU interrupts may lead to frequent disruption of running services in a guest VM, especially given the fact that SA is an always-on service to monitor all guest VMs. On the other hand, a high PMU threshold could let suspicious peer behavior fall through the cracks, missing the opportunity to detect the risk in time.

To balance between these two ends, the SA combines proactive polling with reactive PMU interrupts for event gathering. Specifically, the SA is designed to periodically poll the PMU status, make an assessment of the security risk based upon the events and then reset the PMU counter for firing the interrupt when the risk is low. In this way, the PMU interrupt will only be triggered when events flood in within the polling interval, which is rare in the absence of an attack. The problem here, however, is that the PMU event counter is per-core based and it is infeasible for one CPU core to directly read from the PMU of another core. The most straightforward way to get the content of the core's counter is simply sending an inter- processor interrupt, which itself produces a large number of interrupts (and thus VM exits[1]). To address this issue, our approach leverages the normal VM exits, which happen periodically (every 140μs for the data-serving workload in CloudSuite, due to the events such as external interrupt, virtual APIC and VMCALL) during a VM's normal operations, to "piggyback" the access to the peer's PMU counter. Specifically, whenever the peer VM exits due to such a normal event, the SA checks its timing to determine whether a polling point is approaching: if so, it uses the exit to collect information from the peer's PMU and reset the timer; otherwise, it just waits for the right moment to send an interrupt. In our evaluation, we found that this strategy reduces the number of interrupts from 7685 to 7257 every second due to piggybacking for a peer is running the data-serving workload in CloudSuite. We confirmed that most interrupts due to event polling and PMU interrupts are eliminated during coarse-grained monitoring.

The design of the SA further takes into account the protection demands from VMs, using a multi-grained event gathering strategy to further reduce its overhead. Particularly, when no one is asking for protection, our approach becomes less aggressive in collecting PMU events (e.g., every 3ms in our implementation). Upon receiving a risk assessment request, it switches the gear, starts polling at the coarse granularity every 300μs and maintains the pace throughout the period for protecting the principal. In this way, we can minimize the interference with the peer's operations.

3.4 Protection Level Selection

In line with the service-oriented nature of cloud computing, CREASE provides the security service in a "pay-as-you-go" fashion by allowing a principal VM to decide whether to purchase a specific level of protection at a given price. Such on-demand protection acquisition happens through a set of protection level selection (PLS) APIs, through which the principal obtains the assessment of the peer's security risk and a quotation for different protection levels (i.e., the prices for the frequency/quantum for the principal and the frequency/quantum for the suspicious peer), and also informs the hypervisor of the choice of a protection level.

PLS APIs. Specifically, before running a security-critical task, the principal VM can communicate with the cloud provider through the following APIs:

- `risk_assessment`, which returns the security risks posed by peer VMs, e.g., those considered to execute a Crum attack on the principal.
- `get_price_info`, which returns a table of the prices for different protection levels.

1. VM exit is a control transition from a guest VM to the hypervisor.

- `disable\enable_protection`, which instructs the CRA to disable or enable the protection at a certain level (after the principal decides to purchase the corresponding service).

Such interfaces are provided directly to the user space programs in a VM. A tenant may simply choose a protection level by directly calling `disable\enable_protection` without risk assessment or getting a price quotation.

TOCTTOU. Note that it is possible that a malicious peer may mount an attack after a tenant has conducted the risk assessment and thus the selected protection level may not be sufficient to protect the principal. This threat can be addressed by periodically collecting risk reports from the SA and dynamically adjusting the protection levels in response to the perceived threat. Actually, the fine-grained monitoring when enabling protection allows the hypervisor to quickly adjust the protection level once observing any suspicious behavior. Further, along with specifying a designated protection level, a tenant can also specify a maximum protection level under high risk; the cloud provider can later charge the tenant with the actual protection level.

Pricing models. Similar to the way the current cloud services are priced, the charges for CREASE protection could also vary, in line with the Amazon's price models for instances (reserved instances, on-demand instances and spot instances) [1]. A VM can purchase protection through a fixed and guaranteed price (like a reserved instance) through a long-term contract along with a reserved VM instance, or a pay-per-use price with less availability guarantee for the protection service (like a on-demand instance), or a market-driven price through auctions [1] according to demand and supply (like a spot instance [6]). Under any above pricing models, CREASE is more cost-effective compared to other approaches like always-on protection or using a dedicated physical machine as it only charges a VM at a "pay-as-you-go" fashion. We leave the decision on the price models to the cloud provider and tenants.

Protection levels. At a specific protection level (i.e., the principal's frequency/quantum and the suspicious peer's frequency/quantum), it is important to help the principal understand the security implication of operating at this level. Quantitative estimation of side-channel risks is an open problem. Therefore, in this paper, we aim to quantify the security implication at each level by empirically evaluating the error rates encountered in the adversary's inference of the secret keys, when the principal and the adversary operate with different frequency gaps. To further understand the link between such an error rate estimation and the difficulty of successfully conducting side-channel attacks, we further derived two security metrics, channel capacity and key recovery effort, from the basic error rates estimation (Section 5). Although not the perfect estimation of side-channel risks, we believe our method represents the best effort towards such a goal. The analysis of error rate, channel capacity and key recovery effort will be provided to the principal, allowing it to select the right service given its risk attitudes (risk averse, risk neutral or risk seeking) and its valuations of the sensitive information under protection.

4 IMPLEMENTATION ISSUES

Decision-tree based risk assessment. A Crum attack exhibits some salient features. For example, in a last-level cache attack, since the program produces lots of cache misses with a small number of instruction and data, the peer's LLC miss ratio will go up while the instruction and data TLB miss ratio remains low. To capture such signals of the attack without being over-conservative (which results in purchasing the protection more frequently than needed), the SA runs a machine learning classifier to automatically assess whether an attack is being executed inside the peer. The classifier is built on top of five PMU attributes, including TLB miss ratio, iTLB miss ratio, LLC miss ratio, L2_prefetch miss ration and retired instruction number, which are collected within a short time frame (e.g., 300μs/3ms with/without protection requests). Its output is an assessment whether the peer poses a risk.

In our implementation, we utilized a decision-tree classifier due to its simplicity and performance. Specifically, the classifier is intuitive and works well with the features we observed from Crum attacks. Also, the decision tree is known for its high performance (9.3μs for 100,000 classifications in our experiment), which is important for the real-time protection provided by CREASE.

Our dataset includes a bad set with the attributes collected from 100 runs of the PRIME+PROBE and FLUSH+RELOAD attacks, and a good set with the data gathered from running different legitimate tasks, including PARSEC [12], CloudSuite [18], etc. The risk analysis model was trained in our research over 70% of the dataset. The classifier constructed from the training set was further evaluated over the rest 30% of the data and accurately identified 99.9596% of the attack records at a false detection rate of 0.02%.

When a protection level is not selected, CREASE assesses the risk of current vCPU with PMU data collected within 3ms, which is implemented by setting up a timer in Xen. The timer is triggered every 300μs when a protection level is selected.

Dynamic clocking and scheduling quantum randomization. The CPU frequency of one processor can be controlled by a model-specific register called IA32_PERF_CTRL_MSR (0x199). The value written in the bit range from 0 to 15 indicates the frequency level is required for the processor. The scheduling quantum is determined by the credit scheduler in the Xen hypervisor. Each time the scheduler function is called, we will choose a random value for the next vCPU.

5 SECURITY ANALYSIS

5.1 Deletion Channel Model

To understand the security guarantee offered by frequency scaling, we model Crum attacks as deletion-substitution channels.

A deletion-substitution channel is a type of communication channel model widely used in information theory and coding theory. In the model, the sender transmits a bit (which can be extended to non-binary values) each time, and the transmitted bit is dropped by the channel with a probability p_d or substituted by the channel with a different value with a probability p_s. An important property of the deletion channel is that the receiver does not know which bit was dropped by the channel, therefore perfectly reflecting the nature of the lack of synchronization in Crum attacks.

Modeling sensitive operations. In side-channel attacks, particularly the Crum attacks, the principal's sensitive operations process secret information in phases: in each phase, b bits of the secret is processed; another b bits are used after the operations move to a different phase. For example, in the square-and-multiply implementation of modulo exponentiation, which is widely used in asymmetric ciphers such as RSA, DSA and ElGamal, one bit of the exponent is processed each time. Therefore, $b = 1$ in this case. In the sliding-windows version of the square-and-multiply algorithm, in which a window of n (e.g., 5) bits is processed together, $b = n$. In symmetric ciphers, such as AES and DES, encryption of each block involves different combinations of plaintext and encryption keys, and therefore b is related to the size of the encryption key and the plaintext of the block.

Modeling attacks. In a Crum attack, the adversary collects the side-channel information of the principal's executions. To abstract away the complexity induced by imperfect side-channel measurements and non-synchronicity between the adversary and the principal, we model the side-channel attacks as a deletion-substitution channel. Particularly, we assume that in each phase of the principal's sensitive operation, the b-bit secret is transmitted through a noisy channel to the adversary who has probability p_d to miss the b bits all together and has probability p_s to mistaken the b-bit secret for another. When the channel is noise-free, where $p_d = 0$ and $p_s = 0$, the adversary can perfectly recover the b-bit secret. However, in practice, due to measurement noise or misalignment of the adversary's side-channel measurements with the principal's sensitive operations, usually $p_d > 0$ and $p_s > 0$.

Modeling frequency scaling and dynamic scheduling quantum randomization. By scaling up the execution frequency of the principal during sensitive operations and scaling down the execution frequency of the adversary, both p_d and p_s will be increased. The loss of synchronization is further amplified by dynamic scheduling quantum randomization, which ensures that dropping of signal by the deletion channel will not be detected by the adversary.

5.2 Security Metrics

As mentioned in prior sections, the primary security metric we will use to estimate side-channel risks at certain protection level is the error rate of side-channel inference. In this section, we will establish its connection to two other security concepts: channel capacity of the deletion-substitution channel and key recovery rates using a state-of-the-art sequence assembly algorithm.

Channel capacity. One commonly used approach to estimate the effectiveness of a side-channel attack is by adopting information theory and analyzing the capacity (or mutual information) of the corresponding covert channel [30], [9]. Although lower channel capacity does not directly imply security against side-channel attacks, it does quantify the efficiency of such information exfiltration. Therefore in this paper, we try to estimate an upper bound of the channel capacity given p_d and p_s we observe from empirical tests. However, computing of the channel capacity is an open problem for both binary or non-binary deletion-substitution channels. There have been a few studies on the upper bounds of the capacity of binary deletion channels [15], [19], which is a special case of the non-binary deletion-substitution channel model we have for Crum attacks. In a binary deletion channel, each bit of a signal takes a binary value and only bit deletion, rather than both deletion and substitution, is observed in the channel communication. Some of the capacity upper bounds is given in Table I of [15] and Table IV of [19]. In both studies, the upper bounds of channel capacity decrease monotonically with the rate of bit deletion. Some of these results are summarized here. The non-binary version of the deletion channel is given by Rahmati et al. [42], which shows the channel capacity upper bounds for a $2K$-ary deletion channel, C_{2K} is related to the channel capacity upper bounds for a binary deletion channel, C_2, by Equation 1.

$$C_{2K} \leq C_2(p_d) + (1 - p_d)log(K) \tag{1}$$

We are not aware of any non-trivial upper bound for the channel capacity of non-binary deletion-substitution channels. Therefore for the purpose of our study, we approximate the channel capacity of the non-binary deletion-substitution channels (denoted C_{ds}) with that of the non-binary deletion channels (denoted C_d). It is easy to see that $C_{ds} \leq C_d$, with the same value of p_d. Therefore, C_d is also an upper bound of the non-binary deletion-substitution channel we intend to study. Especially when p_d increases, as shown in Figure 3, p_d contributes much more in the error rate than p_s, and the substitution errors can be omitted without significantly loosen the upper bound. Particularly we list the channel capacity upper bounds for $b = 1, 2, 4, 8, 16, 32, 64, 128, 256$ that is computed using Equation 1 in Table 1.

Table 1: Upper bounds of channel capacity for $2K$-ary deletion-substitution channels

p_d	**1**	**2**	**4**	**8**	**16**	**32**	**64**
0.05	0.816	1.766	2.716	3.666	4.616	5.566	6.516
0.10	0.704	1.604	2.504	3.404	4.304	5.204	6.104
0.15	0.619	1.469	2.319	3.169	4.019	4.869	5.719
0.20	0.551	1.351	2.151	2.951	3.751	4.551	5.351
0.25	0.494	1.244	1.994	2.744	3.494	4.244	4.994
0.30	0.447	1.147	1.847	2.547	3.247	3.947	4.647
0.35	0.406	1.056	1.706	2.356	3.006	3.656	4.306
0.40	0.371	0.971	1.571	2.171	2.771	3.371	3.971
0.45	0.340	0.890	1.440	1.990	2.540	3.090	3.640
0.50	0.311	0.811	1.311	1.811	2.311	2.811	3.311
0.55	0.284	0.734	1.184	1.634	2.084	2.534	2.984
0.60	0.258	0.658	1.058	1.458	1.858	2.258	2.658
0.65	0.233	0.583	0.933	1.283	1.633	1.983	2.333
0.70	0.208	0.508	0.808	1.108	1.408	1.708	2.008
0.75	0.183	0.433	0.683	0.933	1.183	1.433	1.683
0.80	0.157	0.357	0.557	0.757	0.957	1.157	1.357
0.85	0.129	0.279	0.429	0.579	0.729	0.879	1.029
0.90	0.099	0.199	0.299	0.399	0.499	0.599	0.699
0.95	0.064	0.114	0.164	0.214	0.264	0.314	0.364

Key recovery effort. The drop of channel capacity significantly reduces the effectiveness of side-channel attacks. To show such effects in practical attacks, we also evaluate our defense by measuring the difficulty of recovering the secret from the deletion-substitution channel. Unlike covert channel communications, where the sender can use error-correcting encoding to assure reliability of communication, side-channel attacks can only recover from transmission errors using repeated attacks [57], [21].

We model the key recovery process as a sequence alignment procedure. Particularly, we assume the adversary can capture multiple traces of the sensitive operation with the same sequence of secret bits by repeatedly triggering the sensitive operation and conducting Crum attacks. These side-channel traces are noised versions of the secret sequence, with each b bits missing with probability p_d and misinterpreted with probability p_s. The task of the adversary is to recover the original secret sequence given the collected traces with errors.

To understand the link between error rates of key inferences and the ultimate key recovery efforts, we conducted an empirical evaluation with one of the state-of-the-art sequence alignment algorithms in bioinformatics [56]. In particular, we conducted a simulation of key recovery tests. For a specific error rate, we repeated a key recovery test 100 times. In each test, we first select a randomly generated 128 bit secret key and then generate 200 traces from this secret by injecting deletion and substitution errors uniformly at random in accordance with a given error rate. By running the algorithm specified in [56], with parameters slightly tuned for our binary key recovery purpose (the original one is for 4-ary genome assembly), one or multiple secret key candidates were recovered. The result of each test is the shortest edit distance between the original secret key and one of the key candidates recovered by the algorithm. The averages of such edit distances for 100 tests are calculated for different error rates, as shown in Figure 2. We found that when the error rate is below 10%, in some tests, the original secret is among the candidates, an observation in line with the prior studies [57], [21]. However, when the error rate grows above 10%, recovering the secret needs to correct errors on the right candidate, which is difficult: even given the candidate closest to the secret, with an edit distance n, the number of steps one needs to take to brute-force the secret are larger than $C_{128}^{n} \times 2^{n}$, roughly 10^{28} when the error rate is 0.35 (with a distance about 19). Of course, the longer the secret, the more difficult it becomes to recover it under the same error rate.

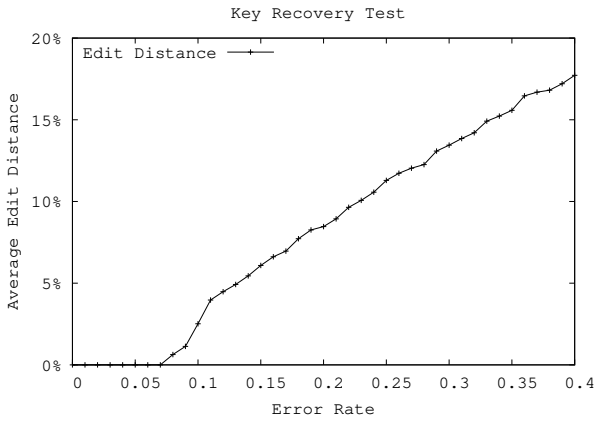


Figure 2: Average edit distance with different error rate.

Security guarantee of CREASE**.** CREASE mitigates Crum attacks by increasing both the deletion probability p_d and substitution probability p_s of the deletion-substitution channels, on which the attacker's cross-VM secret exfiltration relies. Dynamic scheduling quantum randomization breaks the synchronization of the communication channel, which further diminishes the efficiency of the attacks. A practical measure of these effects is the increased error rate of the side-channel measurements conducted by the attacker. As we have shown earlier in this section, higher error rate leads to loss of channel capacity and also increased difficulty of secret reconstruction. For instance, an increase of error rate from 5% to 35% will reduce the channel capacity of a 2-ary channel by half (i.e., 0.41) and increase the brute-force effort of reconstructing a 128-bit security key by 10^{28}. Although CREASE's protection for different types (e.g., RSA, AES) and length (e.g., 128 or 1024 bit) of the secrets are also different, we conservatively believe a 20 to 30% error rate will render the attack infeasible in practice. Therefore, while not completely eliminating the threats, CREASE significantly raises the bar of successful Crum attacks.

6 EVALUATION

We have implemented CREASE by extending the Xen hypervisor (version 4.5.1). The implementation adds around 1,200 lines of code to the Xen hypervisor. We also add a special hypercall for risk assessment and protection level selection. Currently, CREASE supports three protection levels, which correspond to different frequencies in the peers. For example, in our processor with a base frequency of 2.3Ghz, the low, medium and high protection levels correspond to 2.3Ghz, 1.8Ghz and 1.2Ghz of the peers' frequencies. For all protection levels, we turn on dynamic quantum randomization and run the principal VM to the maximum frequency (i.e., 3.0Ghz).

This section presents the evaluation results on the effectiveness and performance of CREASE. As CREASE requires changes to the hypervisor, we cannot directly run our experiments on a public cloud. Instead, we conducted all experiments on an in-house machine. Our test machine is equipped with two Intel Xeon CPU E5-2650 v3 processors, each of which had 10 cores with hyper-threading supported but disabled. Both L1 data and instruction caches are 32KB, 8-way set associative. Each core has a 256KB, 8-way L2 cache; all cores on one socket share a 25MB 20-way L3 cache. The cache line size is 64 bytes. We used Linux 3.18.12 as the guest OS for both Dom0 (the management VM) and DomU (the production VM). The number of Dom0 vCPUs is 4 and the vCPUs are pinned to 4 different physical cores (pCPUs). Each DomU has one vCPU pinned to a pCPU and has 4 GB memory. To reduce the interference from Dom0, the vCPUs in Dom0 and DomU were pinned to pCPUs in different sockets.

6.1 Effectiveness

Settings and attacks. To evaluate the effectiveness of CREASE, we used PRIME+PROBE [40] to attack the implementation of square-and-multiply in GnuPG version 1.4.13, which implements the Open PGP standard. The victim and the attackers resided in different VMs pinned in different pCPUs within a socket. The victim kept using GnuPG to decrypt a short file with a 2048-bit RSA public key. The attacker utilized the inclusiveness property of last-level cache in Intel processors to detect the cache line usage information of the victim. To exactly probe one cache set used by the victim without knowing the virtual-physical address mappings, the attacker utilized large pages available in today's cloud environments.

The key idea to attack the square-and-multiply implementation is to monitor the usage of the square function for the RSA key. When processing a "1" bit, the square function is followed by a modulo reduction and a multiply function , which is followed by another reduction. If the processed bit is "0", the square function is only followed by a reduction. Therefore, the time intervals between consecutive square functions help an attacker to recover the processed exponent.

We regarded the time intervals between the consecutive square functions in GnuPG as the target secret, and the time intervals recovered by the attacker as the stolen secret. Since there is noise encountered by the attacker, the stolen secret does not perfectly match with the targeted one. To measure the difference

Table 2: An error rate comparison of frequency scaling + 1∼30ms random quantum based upon 8 independent experiments

Experiment	#1	#2	#3	#4	#5	#6	#7	#8	Mean	STDDEV
No Protection	0.044499	0.027873	0.047922	0.033252	0.031296	0.053301	0.027873	0.038631	0.0380809	0.008
Random quantum	0.094423	0.099804	0.131115	0.154599	0.740705	0.154599	0.132583	0.08953	0.19966975	0.2201
2.3Ghz	0.123892	0.131063	0.140559	0.124134	0.125587	0.126605	0.118513	0.12583	0.127023	0.0065
2.3Ghz + random quantum	0.234217	0.180435	0.225786	0.670381	0.194292	0.307767	0.238287	0.190222	0.280173	0.1627
1.8Ghz	0.320122	0.325161	0.322593	0.325064	0.315132	0.326518	0.323514	0.323514	0.322702	0.0036
1.8Ghz + random quantum	0.574592	0.37245	0.470565	0.474538	0.554048	0.560167	0.445661	0.368574	0.477574	0.0811
1.2Ghz	0.543292	0.52958	0.536169	0.538253	0.531179	0.527157	0.537865	0.536993	0.535061	0.0053
1.2Ghz + random quantum	0.626871	0.564998	0.580261	0.751343	0.644992	0.701329	0.703038	0.583459	0.644536	0.0682

between the stolen secret and the real one, we calculated the edit distance [31] between them. The edit distance is the total number of insertion, deletion, and substitution operation required to transform a string to another; the error rate is estimated as the edit distance divided by the length of the secret.

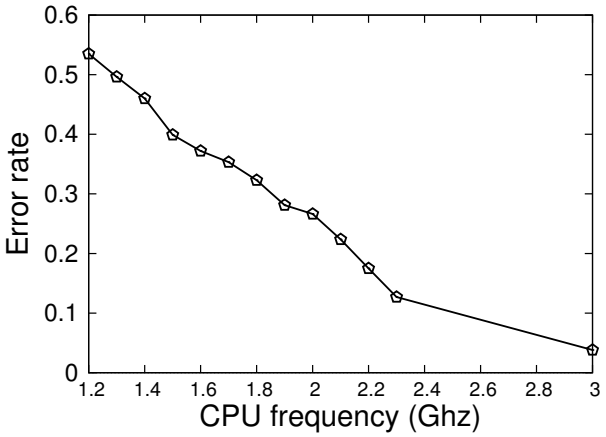


Figure 3: The error rate of stolen secrets for 2048-bit RSA key under different peer CPU frequencies

Defense and Results. Under CREASE's protection, the frequency of the attacker's physical CPU core is lowered while the principal VM's CPU frequency is temporarily boosted to a high value (e.g., 3.0Ghz). Figure 3 shows the relationship between the error rates of stolen secrets and the CPU frequency of the attacker. All results are average of 10 runs. The results demonstrate that the error rates of the stolen secret increase along with the decrease of the frequency of the attacker's pCPUs. When an attacker's CPU frequency drops to a minimum value (e.g., 1.2Ghz), the error rate reaches to 54%, which makes it nearly infeasible to recover the key according to our security analysis and prior studies [57], [21]. Yet, the error rate is as low as 3.8% without protection, by which an attacker can be fairly easy to recover the key.

To further demonstrate the effectiveness of dynamic quantum randomization, we co-ran two other VMs together with an attacking VM on the same physical CPU core to simulate an overcommitting scenario in a real cloud setting. The other two VMs were running the *data-analytics* workload from CloudSuite. We tested the effectiveness under the three protection levels.

As shown in Table 2, the combination of CPU frequency scaling and dynamic quantum randomization makes the rate error not only higher but also more unstable. The average error rate increases from 12.7% to 28.02% at 2.3Ghz (similarly for 1.8Ghz and 1.2Ghz) and the standard deviation increases from 0.0065 to 0.1627 at 2.3Ghz. The reason for the instability is that a smaller and random scheduling quantum gives less execution time for the attacker to probe the cache activities of the victim, which results in more missing pieces for the stolen secrets. Applying dynamic quantum randomization alone may increase the instability of the error rate, as shown in the third row of Table 2. However, the error rate is still small in most cases, which means the attacker can recover most of the sensitive data.

6.2 Performance

Performance of Principal VMs. We evaluated the performance of a protected VM by running six typical applications in turn on a protected VM. Three of the applications involve cryptographic operations: **GnuPG** (1.4.13), **Nginx** https connections (1.9.7 with OpenSSL 1.0.2d), and **scp** (in OpenSSH 4.0p1). We evaluated the performance of GnuPG by measuring the time to decrypt a 1GB file encrypted using the RSA encryption. **Nginx** is a web server which supports Transport Layer Security (TLS) protocol used in https connections. We ran a client in another VM residing in another physical server but within the same subnet. To evaluate the performance of https connections, we ran ApacheBench 2.3 in the client to request a static page of 612 Bytes (the default index.html file with **Nginx**). The reported results were averaged latencies of 500,000 requests. **Scp** is a network data transfer protocol built on top of secure shell (ssh). We transferred a 383 MB file using **scp** and evaluated the time to complete the total transfer.

We also ran other three applications from PARSEC [12] in the principal VM: **ferret**, **dedup** and **canneal**. They present CPU-bound applications with different memory and cache requirements. The input to these benchmarks was "native" and the number of threads was the same as the number of vCPUs in the VM.

All six applications were slightly modified to use CREASE's hypercall to select protection levels during runtime. We created ten VMs, one for the principal and nine for the peers. Four of the nine peers ran *data-serving* benchmarks, and five ran *data-analytics* benchmarks.

Figure 4 shows the performance for benchmarks in the principal VM. As expected, due to boosted frequency, all principal VMs run faster than normal runs. Yet, if an application chooses a higher protection level, the performance further increases slightly (from 12.55% to 12.69% and to 13.76% for low, medium and high protection levels), even though the principal VMs all run at the highest frequency (i.e., 3.0Ghz). The reason is that the slower the peers run, the less interference in the last level cache from the peers will be. As **scp** is involving copying a large file and thus is more memory and I/O intensive than CPU-intensive, boosting the CPU frequency only leads to 1.4% performance speedup. Nevertheless, this still creates a notable gap between a principal VM and the peers for security protection against Crum attacks.

Overhead of the peers. To evaluate the performance overheads of peers, we ran four CloudSuite benchmarks (*data-serving data-analytics*, *data-caching* and *media-stream*) and ten PARSEC benchmarks. The *data-serving* workload relies on YCSB 0.1.3 and Cassandra 0.7.3 to benchmark data store systems; we ran both the client and the server on the same VM for simplicity. The

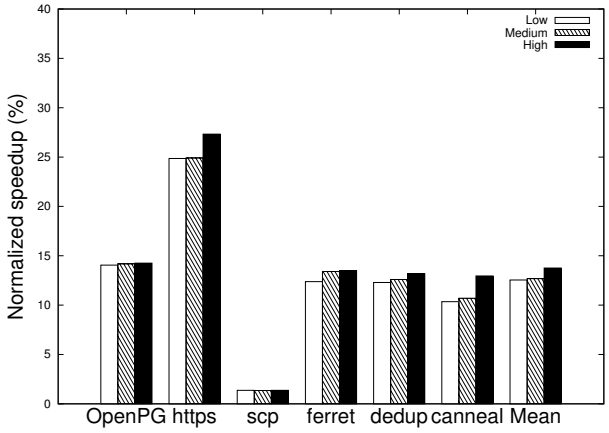


Figure 4: Performance speedup for principal applications

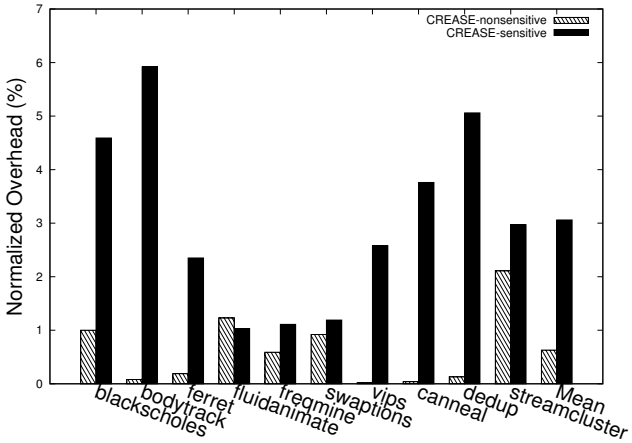


Figure 6: Performance overhead for peers running PARSEC

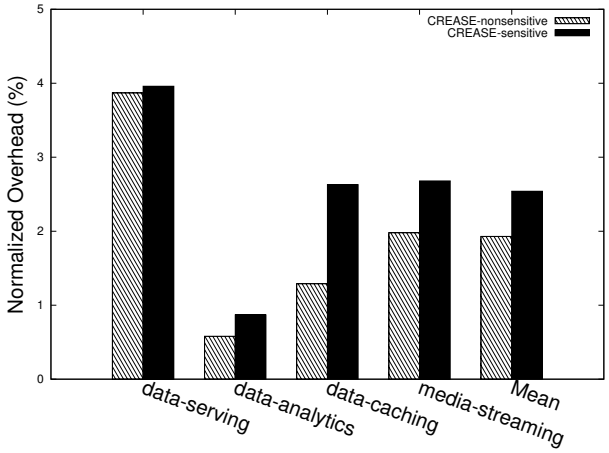


Figure 5: Performance overhead for peers running CloudSuite

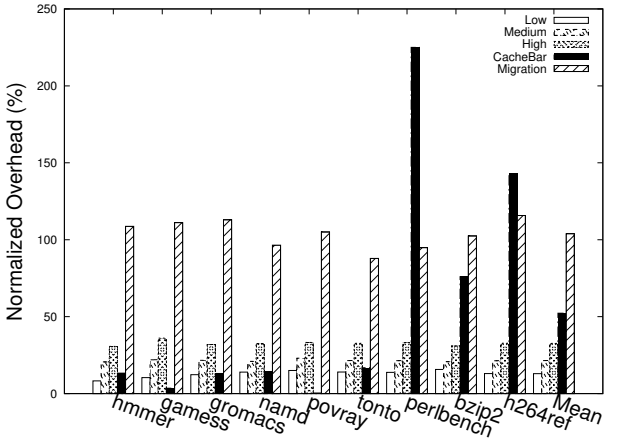


Figure 7: Performance comparison between CREASE and other state-of-the-art approaches using SPEC CPU 2006 when enabling protection

server stored 5,000,000 records and the client executed 500,000 (50% update operations) operations on all records. We evaluated the average update latency. The *data-analytics* workload used Hadoop 20.2 to perform Bayes classifier on a 162MB Wikipedia document; we evaluated the total execution of the classifier. The *data-caching* workload uses Memcached 1.4.15 to simulate the behavior of a Twitter caching server using a 1.2GB Twitter dataset. The metric of interest is the throughput calculated as the number of requests served per second. The *media-stream* workload uses the Darwin Streaming Server to benchmark video streaming behavior. We evaluated the average transferring speed. All the PARSEC benchmarks used the default configuration. The performance was evaluated when the principal VM execute sensitive operations (CREASE-sensitive) and non-sensitive operations (CREASE-nonsensitive). As there is no attack here, CREASE enabled the low protection level by default.

Figure 5 and Figure 6 show the overheads of peers running CloudSuite and PARSEC accordingly. The results show that when the principal executed non-sensitive operations, the average overhead is about 1.93% and 0.63% for CloudSuite and PARSEC accordingly. Due to the fine-grained monitoring when turning on protection, the average overhead of peers increases to 2.54% and 3.06% respectively.

Comparison with other approaches.

There are other approaches to defending against Crum attacks. CacheBar [60] is a state-of-art approach to mitigating the last level cache side-channel attacks. It leverages copy-on-access for physical pages shared among multiple security domains (for FLUSH+RELOAD attack), and manages cacheability for pages to limit the number of cache lines per cache set that an adversary can occupy simultaneously. It evaluates using SPEC CPU 2006 and reports their results in the paper. We evaluated CREASE using SPEC CPU 2006 benchmarks and compared it with CacheBar, as shown in Figure 7. CacheBar incurs larger overhead for some workloads (perlbench, bzip2 and h264ref) and the average overhead is 52.27%, which is larger than CREASE (32.66%).

Nomad [39] is a migration-based solution which migrates one victim VM to different physical machines when necessary. It assumes that cache-based side channel attack is capable of extracting a key in few minutes, which is not the case based on our evaluation. In fact, we observe that a 2048-bit key can be extracted within 30 milliseconds. Moreover, the migration-based defense cannot eliminate the threat if multiple attackers reside on different physical servers. We also evaluated the performance overhead of the migration-based solution, as shown in Figure 7. Its average overhead on SPEC CPU 2006 benchmarks is 103.91%, which is also larger than CREASE (32.66%).

Table 3: Error rates of colluding attacks on GnuPG (RSA's 2048-bit key) based upon 8 independent experiments

Experiment	#1	#2	#3	#4	#5	#6	#7	#8	Mean
Type 1 under 1.2Ghz	0.091443	0.093399	0.1022	0.101711	0.094377	0.08802	0.083619	0.090465	0.093154
Type 1 under 1.2Ghz and random quantum	0.279707	0.281663	0.207335	0.26357	0.266015	0.235208	0.280685	0.220049	0.254279
Type 1 under 50 CAP value	0.450367	0.452812	0.471394	0.482152	0.516381	0.451834	0.472861	0.535452	0.479157
Type 2 under 1.2Ghz	0.325183	0.300244	0.331051	0.343276	0.337408	0.303178	0.290954	0.353545	0.323105

7 DISCUSSION

Multi-core collusion. The SA evaluates the risk posed by the peer from the performance data collected from the PMU of its core. A potential security risk is that the adversary controlling multiple cores could distribute the attack workload across these cores, keeping the behavior of a single core inconspicuous while collectively gathering a sufficient amount of information for recovering the principal's secret. Currently, the colluding threat has not been thoroughly investigated in all prior studies and its implication is still less clear. Here we conduct a preliminary study to understand its impact on CREASE and evaluate the effect of possible defenses.

Basically, there could be two types of colluding models. The first one (type 1) aims to make up for the speed loss caused by lowered frequency by accumulating the processor power of multiple cores. Take PRIME+PROBE as an example, if the frequency of the attacker is much lower than the victim's, it is possible that the victim has already accessed the cache set before the attacker finishes priming the cache set. To quickly prime a cache set, an adversary can anticipate two probing programs residing on different cores, each of which priming half of the cache set. The concrete steps are as follows:

- *Step 1:* The main thread creates one priming thread and pins it to a new core. Both threads keep executing three operations: priming the half of the target cache set, waiting for a period of time, and probing the half cache set.
- *Step 2:* The main thread starts to prime the other half of the cache set with the same three operations. If the victim accesses the target data or instruction, one of the two threads would detect the access in the probe operation.

The first two rows in Table 3 show the error rates of stolen secrets without and with schedule quantum randomization when both CPU cores' frequencies are lowered to 1.2Ghz. Without schedule quantum randomization, this type of colluding attack can recover secret with a reasonable error rate. However, with schedule quantum randomization, the error rate gets significantly higher such that it is infeasible to recover the secret, due to the fact that the two cores are less likely to run together thanks to misaligned execution. One special case is the adversary VM monopolizes the two cores, where CREASE can set the limit to the accumulated resource for the VM, i.e., setting the *cap* of the VM to fix the maximum proportion of CPU it can use. With the *cap* setting as 50 (i.e., 50% ratio), the error rate will grow significantly to the level where it is hardly possible to recover any secret.

The second type of colluding attack (type 2) is using multiple cores to probe all cache sets, with the goal of collecting different sample sequences during monitoring and increasing the accuracy by comparing the sample sequences collected by different cores. To evaluate its effectiveness, we use one core to monitor the *square* function and the other to monitor the *multiply* function. For type 2 scenario, the sample sequence collected by two attackers should be combined to restore the actual behavior of the victim. However, simple combination is unable to produce the accurate result. Row 4 in Table 3 shows that the error rates of the type 2 model under 1.2Ghz, which is infeasible to recover the secret. Type 2 is less effective than type 1 because it is harder to align the secrets collected by different cores.

We also evaluated the case of using more CPU cores for colluding. Interesting, the error rates for both types of colluding attacks increase significantly compared to the 2-core case, e.g., the error rate increases from 9.3% to 48.7% when using 4 cores for the type 1 attack. The reason is that, with more cores colluding, it becomes extremely harder for such cores to synchronize and/or align the stolen secret pieces together. Further, we briefly studied the case of using multiple cores instead of multiple VMs to do the colluding, which turns out to be much harder since it is more difficult for multiple VMs to synchronize their operations and align secret pieces. Further, colocating more than one VMs with a victim VM is extremely hard given that the VM placement is controlled by the cloud provider [45].

Mitigating the colluding attacks: we only consider the defense in the type-1 scenario since this attack is easier to implement. Under this scenario, two attackers should keep priming the half cache set, which can be detected by our SA module. The basic idea of defending against such attacks is to leverage random scheduling quantum and prohibiting the attackers from co-running with each other. As shown in the second line in Table 3, random scheduling quantum can significantly increase the error rates for such attacks. Meanwhile, we can limit the accumulated CPU resources used by the two cores, such as setting the cap value of the vCPU.

Covert channel. One potential issue is that CREASE creates a covert channel between two VMs such that one VM can have insight into each other's CPU usage through the situation awareness module. However, as discussed in section 3.3, all that the situation-aware module tells the VM is nothing more than whether one of its peers apparently presents a side-channel threat, which is very thin and barely adding to the principal's knowledge, not to mention any utility for a cross-VM attack. Besides, even if a malicious principal is able to guess peers' current execution behaviors from the thin channel, it cannot know exactly what a specific peer's behavior is. Further, CREASE can easily detect this covert channel due to frequent risk assessment. On the other hand, if all these VMs intend to do is just colluding, they can easily utilize other mechanisms such as KSM, burst I/O, caches and memory burst.

Multiple principals. As discussed in section 3.1, our current implementation of CREASE allows running only a single principal (for a CPU socket) at a time. This prevents a malicious attacker to purchase more CPU frequency and quanta to perform Crum attack of other peers, which will not be executing security-critical operations without being elevated as a principal. Hence, a malicious VM trying to produce a speed gap between itself and other VMs only waste the attackers' money without being able to steal any useful information. If multiple VMs running on the same CPU socket do ask for protection at the same time, CREASE handles them through an unblocked FIFO queue. The VMs unable to be boosted will wait in the queue. Upon rejection, CREASE

will return a waiting time number indicating when the rejected VM can invoke the interface again to get boosted. To reduce the performance impact for the rejected VM, the waiting VMs will not be blocked, which means they can do other work. When the time is up, the VM can reapply again. To make up for the time cost of one request that is waiting in the queue, the cloud provider may grant a discount to it.

An attacker may issue malicious and frequent CPU requests to ask for protection, which blocks the victim needing to perform a critical operation. To mitigate the effects of such a DoS attack, CREASE limits the time duration a guest can boost itself. This is reasonable since a VM usually sends a CPU request for small operations like AES/RSA encryption. Under such situation, even if a malicious guest VM blocks the victim, the blocking time will not severely affect the victim. Moreover, if one VM is severely affected by such DoS attack, CREASE will migrate one of the API users to a different socket. We leave the details to handle such a case to our future research.

Financial attacks. Another possible attack is that an adversary may purposely generate noise workloads to trigger the SA to flag a false alarm to raise the risk level. The adversary may be pretending to perform Crum attack while being only interested in increasing costs for the competitor's VM. However, a higher level of security guarantee should come with a price. A company has the confidence to ask its customers for more money if it can guarantee to provide higher security compared with its competitors. Furthermore, if a suspicious VM constantly behaves like a Crum attacker and there is one VM in the same socket keeps invoking CREASE API, the cloud provider can treat such frequent alarms as an indication of malicious customers; the provider can migrate the VM away.

8 Related Work

Hardware solutions. New cache designs were proposed to defeat various cache-based side-channel attacks. These solutions include: using relaxed includsive cache [26], partitioning shared caches dynamically to isolate the principal and the adversary [41], [16], [50], [51], [29], introducing randomness into the cache usage [51], [52], [35], [27], and coarsening the fine-grained time keeping in modern processors with new ISA designs [37]. Although these approaches are promising, they are specific to CPU cache-based side-channels. Liu et al. [33] proposes a comprehensive defense against memory-traces (e.g., cache access patterns, memory bus access patterns) that involves both new hardware support and software modification. However, such a method induces as much as 195× slowdown. One major limitation of hardware-based approach is that the time window required to have them included in commercial hardware is very long, if not indefinite. Therefore, their adoption by public cloud providers for practical side-channel defenses is less likely, compared to our approach.

OS solutions. OS-based solutions refer to side-channel countermeasures that are implemented in the privileged software layers, such as operating systems or hypervisors. In the case of public clouds that we consider, they particularly mean hypervisor-based solutions. Our approach falls into this category. Similar to the hardware-solution counterparts, one direction of OS-supported approaches is to partition the shared caches [43], [46], [28], [34], [60]. Although effective against targeted attacks, these methods are too narrow in scope and failed to consider other types of side-channel attacks. Eliminating fine-grained timer by virtualizing RDTSC instructions [48] can be easily adapted to work in the cloud, but experiments show that such a method may render the operating system unstable. Moreover it fails to consider other sources for time keeping. Enforcing deterministic execution in cloud computing is another viable solution [8], [32], which unfortunately induces higher than acceptable performance overhead for practical use. Zhang et al. [59] have proposed to inject random noise particularly to defeat L1 cache attacks and Varadarajan et al. [47] studied security-aware scheduling algorithms to mitigate the per-core side-channel threats. These methods do not consider other vectors of attacks such as shared last-level caches. Moon et al [39] studied live VM migration based side-channel protection in public clouds. This method can be applied to defeat all side channels that exploit shared resources. Our approach also considers the willingness of cloud provider to adopt side-channel defense techniques by integrating the defense methods into the cloud pricing model, and therefore has better chance of adoption.

Software solutions. Software transformation is yet another category of side-channel defenses, including eliminating secret-dependent data flows and control flows [38], [13] and introducing decoy execution traces into the software program so that the true execution traces are obfuscated during the adversary's side-channel analysis [44]. These approaches effectively defeat all side-channel attacks but at the same time come at a very high performance cost (e.g., one or two magnitudes higher runtime overhead). Software diversification [14] offers probabilistic defense against cache side-channel attacks, by randomly and frequently switch between program replicas with the same semantics. This method reduces the performance overhead to acceptable level (e.g., around 100% overhead) but at the same time weakens the ability to provably defeat side-channel attacks. In general, these software approaches are promising in certain context, but due to the complexity of use and high performance overhead, it is unsuitable for cloud settings.

Other non-defensive approaches. Doychev et al. [17] present a method to detect side-channel vulnerability via static analysis. Irazoqui et al. [24] instead proposed to statically analyze programs for malicious side-channel attack code, but the requirement of examining all binary code is not suitable for public clouds.

9 Conclusions

This paper presents CREASE, a new technique to defend against cross-VM runtime monitoring. CREASE is shaped by a resource-elastic design that combines dynamic CPU frequency boosting and randomized scheduling to temporarily give more resources to a principal to outpace the peer. Further reducing the cost is the situation-awareness technique that continuously monitors peers without interfering their operations. Evaluation shows that CREASE is both effective and lightweight, incurring small overheads for both the principal and the peer.

Acknowledgment

This work is supported in part by the National Key Research & Development Program (No. 2016YFB1000500) and the National Natural Science Foundation of China (No. 61572314, 61303011).

REFERENCES

[1] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. *ACM Transactions on Economics and Computation*, 1(3):16, 2013.

[2] J. Ahn, C. H. Park, and J. Huh. Micro-sliced virtual processors to hide the effect of discontinuous cpu availability for consolidated systems. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 394–405. IEEE Computer Society, 2014.

[3] J. B. Almeida, M. Barbosa, G. Barthe, and F. Dupressoir. Verifiable side-channel security of cryptographic implementations: Constant-time MEE-CBC. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 163–184, 2016.

[4] Amazon Inc. Amazon ec instance types. http://aws.amazon.com/ec2/instance-types/.

[5] Amazon Inc. Amazon ec2 service level agreement (service credits). https://aws.amazon.com/ec2/sla/.

[6] Amazon Inc. Amazon spot instances. https://aws.amazon.com/ec2/spot/.

[7] G. I. Apecechea, M. S. Inci, T. Eisenbarth, and B. Sunar. Fine grain cross-VM attacks on Xen and VMware are possible! Technical Report 2014/248, IACR Cryptology ePrint Archive, Apr. 2014.

[8] A. Aviram, S. Hu, B. Ford, and R. Gummadi. Determinating timing channels in compute clouds. In *2010 ACM Workshop on Cloud Computing Security*, pages 103–108, 2010.

[9] M. Backes, B. Kopf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *30th IEEE Symposium on Security and Privacy*, pages 141–153. IEEE Computer Society, 2009.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[11] S. A. Baset, L. Wang, and C. Tang. Towards an understanding of oversubscription in cloud. In *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[12] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[13] B. Coppens, I. Verbauwhede, K. D. Bosschere, and B. D. Sutter. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *30th IEEE Symposium on Security and Privacy*, pages 45–60, 2009.

[14] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz. Thwarting cache side-channel attacks through dynamic software diversity. In *2015 ISOC Network and Distributed System Security Symposium*, 2015.

[15] S. Diggavi, M. Mitzenmacher, and H. D. Pfister. Capacity upper bounds for deletion channels. In *2007 IEEE International Symposium on Information Theory*, pages 1716–1720, June 2007.

[16] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Trans. Archit. Code Optim.*, 8(4), Jan. 2012.

[17] G. Doychev, D. Feld, B. Köpf, and L. Mauborgne. CacheAudit: A tool for the static analysis of cache side channels. In *22st USENIX conference on Security symposium*, 2013.

[18] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *ACM SIGPLAN Notices*, 47(4):37–48, 2012.

[19] D. Fertonani and T. Duman. Novel bounds on the capacity of binary channels with deletions and substitutions. In *2009 IEEE International Symposium on Information Theory*, pages 2552–2556, June 2009.

[20] D. Gruss, C. Maurice, and K. Wagner. Flush+ flush: A stealthier last-level cache attack. *arXiv preprint arXiv:1511.04594*, 2015.

[21] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar. Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud. Cryptology ePrint Archive, Report 2015/898, 2015. http://eprint.iacr.org/.

[22] G. Irazoqui, T. Eisenbarth, and B. Sunar. Cross processor cache attacks. Technical Report 2015/1155, IACR Cryptology ePrint Archive, 2015.

[23] G. Irazoqui, T. Eisenbarth, and B. Sunar. S$A: A shared cache attack that works across cores and defies VM sandboxing—and its application to AES. In *36th IEEE Symposium on Security and Privacy*, May 2015.

[24] G. Irazoqui, T. Eisenbarth, and B. Sunar. Mascat: Stopping microarchitectural attacks before execution. Cryptology ePrint Archive, Report 2016/1196, 2016.

[25] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar. Wait a minute! A fast, cross-VM attack on AES. In *Research in Attacks, Intrusions and Defenses – RAID 2014*, volume 8688 of *LNCS*, pages 299–319. Springer, 2014.

[26] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel. Ric: Relaxed inclusion caches for mitigating llc side-channel attacks. In *Design Automation Conference*, 2017.

[27] G. Keramidas, A. Antonopoulos, D. Serpanos, and S. Kaxiras. Non deterministic caches: a simple and effective defense against side channel attacks. *Design Automation for Embedded Systems*, 12:221–230, 2008.

[28] T. Kim, M. Peinado, and G. Mainar-Ruiz. STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud. In *21st USENIX conference on Security symposium*, 2012.

[29] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou. Architecting against software cache-based side-channel attacks. *IEEE Trans. Comput.*, 62(7), July 2013.

[30] B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *14th ACM Conference on Computer and Communications Security*, pages 286–296. ACM, 2007.

[31] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710, 1966.

[32] P. Li, D. Gao, and M. K. Reiter. Mitigating access-driven timing channels in clouds using StopWatch. In *43rd IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12, June 2013.

[33] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi. Ghostrider: A hardware-software system for memory trace oblivious computation. In *Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 87–101. ACM, 2015.

[34] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In *IEEE Symposium on High-Performance Computer Architecture*, Barcelona, Spain, mar 2016.

[35] F. Liu and R. B. Lee. Random fill cache architecture. In *47th IEEE/ACM Symposium on Microarchitecture*, pages 203–215, Dec. 2014.

[36] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *36th IEEE Symposium on Security and Privacy*, May 2015.

[37] R. Martin, J. Demme, and S. Sethumadhavan. Timewarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *39th Annual International Symposium on Computer Architecture*, pages 118–129, 2012.

[38] D. Molnar, M. Piotrowski, D. Schultz, and D. Wagner. The program counter security model: automatic detection and removal of control-flow side channel attacks. In *8th international conference on Information Security and Cryptology*, pages 156–168, 2005.

[39] S.-J. Moon, V. Sekar, and M. K. Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1595–1606. ACM, 2015.

[40] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology–CT-RSA 2006*, pages 1–20. Springer, 2006.

[41] D. Page. Partitioned cache architecture as a side-channel defence mechanism, 2005.

[42] M. Rahmati and T. Duman. An upper bound on the capacity of non-binary deletion channels. In *2013 IEEE International Symposium on Information Theory*, pages 2940–2944, July 2013.

[43] H. Raj, R. Nathuji, A. Singh, and P. England. Resource management for isolation enhanced cloud services. In *2009 ACM Cloud Computing Security Workshop*, pages 77–84, Nov. 2009.

[44] A. Rane, C. Lin, and M. Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. In *24th USENIX Conference on Security Symposium*, pages 431–446, 2015.

[45] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[46] J. Shi, X. Song, H. Chen, and B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *41st International Conference on Dependable Systems and Networks Workshops*, pages 194–199, 2011.

[47] V. Varadarajan, T. Ristenpart, and M. Swift. Scheduler-based defenses against cross-vm side-channels. In *23rd USENIX Conference on Security Symposium*, pages 687–702. USENIX Association, 2014.

[48] B. C. Vattikonda, S. Das, and H. Shacham. Eliminating fine grained timers in Xen. In *3rd ACM Workshop on Cloud Computing Security*, pages 41–46, 2011.

[49] Y. Wang, A. Ferraiuolo, and G. E. Suh. Timing channel protection for a shared memory controller. In *22nd IEEE International Symposium on High-Performance Computer Architecture*, 2014.

[50] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *22nd Annual Computer Security Applications Conference*, pages 473–482, 2006.

[51] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *34th annual international symposium on Computer architecture*, pages 494–505, 2007.

[52] Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In *41st annual IEEE/ACM International Symposium on Microarchitecture*, pages 83–93, 2008.

[53] C. Xu, S. Gamage, H. Lu, R. R. Kompella, and D. Xu. vturbo: Accelerating virtual machine i/o processing using designated turbo-sliced core. In *USENIX Annual Technical Conference*, pages 243–254, 2013.

[54] C. Xu, S. Gamage, P. N. Rao, A. Kangarlou, R. R. Kompella, and D. Xu. vslicer: latency-aware virtual machine scheduling via differentiated-frequency cpu slicing. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 3–14. ACM, 2012.

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

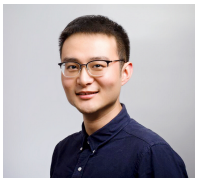
[56] Y. Ye. Identification and quantification of abundant species from pyrosequences of 16s rrna by consensus alignment. In *IEEE International Conference on Bioinformatics and Biomedicine*, page 153157, 2010.

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

[58] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-tenant side-channel attacks in PaaS clouds. In *21st ACM Conference on Computer and Communications Security*, Nov. 2014.

[59] Y. Zhang and M. K. Reiter. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *20th ACM Conference on Computer and Communications Security*, pages 827–837, Nov. 2013.

[60] Z. Zhou, M. K. Reiter, and Y. Zhang. A software approach to defeating side channels in last-level caches. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

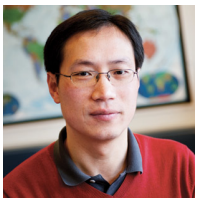


Zeyu Mi Zeyu Mi received the BS degree in software engineering in 2014, from Nanjing University, China. He is now a Ph.D student at the Institute of Parallel and Distributed Systems at School of Software, Shanghai Jiao Tong University. His research interests include system security and system virtualization.

Haibo Chen Haibo Chen received a Ph.D degree in computer science from Fudan University in 2009. He is currently a Professor at School of Software, Shanghai Jiao Tong University. He is a senior member of IEEE. His research interests are in operating systems and parallel and distributed systems.

Yinqian Zhang Yinqian Zhang is an assistant professor of the Department of Computer Science and Engineering of the Ohio State University. His research interest is computer system security, with particular emphasis on cloud computing security, OS security and side-channel security.

Shuanghe Peng Peng Shuanghe is a supervisor of master students in the Department of Information Security at School of Computer and Information Technology, Beijing Jiaotong University. Her research interests include System Security and Embedded Systems. Shuanghe has a PhD in computer application technology.



Xiaofeng Wang XiaoFeng Wang is a professor of Informatics and Computer Science at Indiana University Bloomington. His research focuses on system security and data privacy with a specialization on security and privacy issues in mobile and cloud computing, and privacy issues in dissemination and computation of human genomic data.

Michael Reiter Michael Reiter is the Lawrence M. Slifkin Distinguished Professor in the Department of Computer Science at the University of North Carolina at Chapel Hill (UNC). He received the B.S. degree in mathematical sciences from UNC in 1989, and the M.S. and Ph.D. degrees in Computer Science from Cornell University in 1991 and 1993, respectively. He joined AT&T Bell Labs in 1993 and became a founding member of AT&T Labs Research when NCR and Lucent Technologies (including Bell Labs) were split away from AT&T in 1996. He then returned to Bell Labs in 1998 as Director of Secure Systems Research. In 2001, he joined Carnegie Mellon University as a Professor of Electrical & Computer Engineering and Computer Science, where he was also the founding Technical Director of CyLab. He joined the faculty at UNC in 2007.