Future Generation Computer Systems 49 (2015) 47–57

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

TEE: A virtual DRTM based execution environment for secure cloud-end computing☆

Weiqi Dai [a], Hai Jin [a,*], Deqing Zou [a], Shouhuai Xu [b], Weide Zheng [a], Lei Shi [a], Laurence Tianruo Yang [a,c]

[a] Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China

[b] Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA

[c] Department of Computer Science, St. Francis Xavier University, Antigonish, NS, Canada

HIGHLIGHTS

- We present the design, implementation and analysis of a novel system, called Trusted Execution Environment (TEE), for secure cloud-end.
- TEE can support a spectrum of application needs, ranging from pure cryptographic libraries to full-fledged trustworthy software.
- The novelty of our work is the virtualization of DRTM that can let vTPM determine the origin of TPM commands.

ARTICLE INFO

Article history:
Received 20 January 2014
Received in revised form
31 July 2014
Accepted 15 August 2014
Available online 26 August 2014

Keywords:
Virtual Machine Monitor (VMM)
Dynamic Root of Trust for Measurement (DRTM)
Cloud computing
Xen hypervisor

ABSTRACT

The Internet of Things (IoT) is the incoming generation of information technology. However, the huge amount of data collected by wireless sensors in IoT will impose a big challenge that can only be met by cloud computing. In particular, ensuring security in the cloud-end is necessary. Previous studies have mainly focused on secure cloud-end *storage*, whereas secure cloud-end *computing* is much less investigated. The current practice is solely based on Virtual Machines (VM), and cannot offer adequate security because the guest Operating Systems (OS) often can be compromised (e.g., by exploiting their vulnerabilities). This motivates the need of solutions for more secure cloud-end computing. This paper presents the design, implementation and analysis of a candidate solution, called Trusted Execution Environment (TEE), which takes advantage of both virtualization and trusted computing technologies simultaneously. The novelty behind TEE is the virtualization of the Dynamic Root of Trust for Measurement (DRTM).

1. Introduction

The Internet of Things (IoT) will be the future of the Internet [1]. However, data in IoT are massive, multi-sourced, heterogeneous, redundant, dynamic and sparse. It is both necessary and sufficient to utilize cloud computing [2,3] such that IoT moves and processes the data into a cloud, which must ensure both secure *storage* and secure *computing* because a single attack could cause the compromise of *multiple* customers' data. To reduce the cost, Virtual Machines (VM) have become essential for cloud computing and are indeed widely utilized in today's cloud-end computing practice. As shown in the example scenario of Fig. 1, the cloud provides the customers the "illusion" of their current self-managed systems through the so-called Trusted Virtual Domain (TVD) [4–7] (or its alternative [8]). While much progress has been made in secure cloud-end storage (e.g., [9–13]), the problem of secure cloud-end computing has yet to be tackled.

Because homomorphic cryptography (despite recent breakthrough [14]) is not efficient enough for most practical uses, sensitive data – even if encrypted for storage – have to be processed in computer memory in their plaintext form, and therefore can be attacked by exploiting vulnerabilities in the guest operating system (OS). This problem has been investigated mainly in the context

☆ This paper substantially extends the extended abstract published as [54].

* Corresponding author.
E-mail addresses: daiweiqi@gmail.com (W. Dai), hjin@hust.edu.cn (H. Jin), deqingzou@hust.edu.cn (D. Zou), shxu@cs.utsa.edu (S. Xu), zhwade@gmail.com (W. Zheng), foxshee@yahoo.com.cn (L. Shi), ltyang@stfx.ca (L.T. Yang).

http://dx.doi.org/10.1016/j.future.2014.08.005
0167-739X/

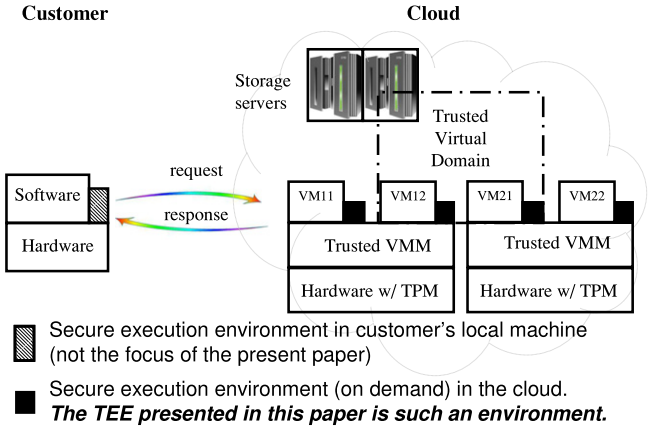

Fig. 1. A scenario of secure cloud computing.

of user-end computer, rather than cloud-end computer, via two approaches:

- Exploiting virtualization technology alone: Overshadow [15,16] advocates presenting an application with a normal (i.e., unencrypted) view of its resources while presenting the guest OS with only encrypted view (guest OS is still needed for handling the complex resource management). A similar approach is also investigated in [17]. However, these systems are not quite suitable for secure cloud-end computing because: (i) they still use the guest OS for managing resources, meaning that a malicious OS could *unfairly* allocate resources to sensitive applications. (ii) Overshadow's current implementation does not appear to authenticate the requests for using it, thus a malicious guest OS may replay the requests to run an encrypted application multiple times. (iii) The performance of Overshadow when utilized by multiple VMs on a single platform is not clear (because Overshadow was not motivated for secure cloud-end computing).
- Exploiting trusted computing technology alone: Flicker and its descendent [18–20] can guarantee strong security by exploiting the recently introduced processor feature known as Dynamic Root of Trust for Measurement (DRTM). These systems are not suitable for secure cloud-end computing because when a sensitive application is executed in the hardware-protected environment, other customers' VMs running on the same platform will be frozen.

In summary, the current VM-based cloud computing practice does not offer adequate security when sensitive applications run on cloud-end computers. Existing solutions that can resist attacks by compromised/malicious OS are neither motivated by, nor (quite) suitable for, secure cloud-end computing.

Our contributions. This paper makes the following contributions.

- We present the design, implementation and analysis of a novel system, called Trusted Execution Environment (TEE), for secure *cloud-end* computing as shown in Fig. 1.[1] TEE can support a spectrum of application needs, ranging from pure cryptographic libraries to full-fledged trustworthy software. Moreover, TEE can provide an "overlay" of TEE network that can be deemed as more trustworthy than the network of VMs within the same TVD (e.g., for cryptographic multi-party computation).

[1] Secure *customer-end* computing is not addressed in this paper. Nevertheless, we mention that a system like Flicker [18,19] as well as its descendent [21,20] can serve for that purpose.

Table 1
Acronyms (in alphabetical order).

D-CRTM	Dynamic Core Root of Trust for Measurement
DRTM	Dynamic Root of Trust for Measurement
LPC	Low Pin Count
NVM	TPM's Non-Volatile Memory
OS	Operating System
PCR	Platform Configuration Register
SRTM	Static Root of Trust (for) Measurement
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TVD	Trusted Virtual Domain
vD-CRTM	virtual D-CRTM
vDRTM	virtual DRTM
VM	Virtual Machine
VMM	Virtual Machine Monitor (Hypervisor)
vTPM	virtual TPM

- From a technical perspective, the novelty of our work is the virtualization of DRTM. This is a non-trivial task because, in particular: how should we deal with the problem that vTPM cannot determine the origin of TPM commands (because in the Xen hypervisor we experimented with only locality 0 used)?We resolve this issue by extending the locality control and management of vTPM.
- We have implemented the design reported in the paper, by using Xen paravirtualization. We argue for the security of TEE. Benchmark-based performance evaluation suggests that TEE-enabled cloud computing does not impose any significant performance penalty. We will make the source code of the system publicly available so that other researchers can adopt or adapt it for further research.

Limitations of TEE. TEE is limited in dealing with *privileged insider attacks* and *side-channel attacks*. The former can be launched by privileged administrators of cloud computers. The latter can even be launched by malicious cloud computing customers because VMs and TEEs on a single computer are likely rented to multiple customers [22]. The two limitations are inherent to the design choice that in order to accommodate simultaneous execution of multiple VMs and/or TEEs on a platform, there must be a trusted layer beneath them (no matter it is implemented in hardware or software). As such, the current VM-based cloud computing practice, Overshadow-like solutions, and TEE are equally subject to these attacks.

Paper organization. We discuss related prior work in Section 2, briefly review some background knowledge about Xen and trusted computing in Section 3, present the problem definition in Section 4, and describe the design, implementation, and analysis of TEE in Section 5. We conclude the paper in Section 6 with future research directions. We summarize major acronyms used in the paper in Table 1.

2. Related work

There are investigations that exploit VMMs for secure execution environment (e.g., [23–25,15,17,26–28]). Proxos [23] implements a safe execution environment while differentiating the system calls from an untrusted OS and those from a trusted OS. Terra [24] provides mechanisms for running applications in separate VMs and for VMMs to attest the software executing inside a VM. Chaos [25], which is implemented with Xen, encrypts process memory pages and uses trusted system calls to protect applications. Overshadow [15], which is implemented with VMWare, moves a step further by transparently ensuring that sensitive applications cannot be compromised by the malicious guest OS. A similar system [17] is implemented with Xen. On the other hand, there are efforts that exploit DRTM for secure execution environment.

Flicker [18,19] exploits DRTM to launch a secure execution environment that assumes a TCB as small as 250 lines of code. It arguably provides the highest security possible because it remains secure even if the BIOS, OS and DMA-enabled devices are all malicious. This is achieved (in part) by ensuring that when Flicker executes, everything else is frozen.

In comparison to the aforementioned investigations that exploit either virtualization or trusted computing, TEE takes advantage of both in a non-trivial fashion because of the following. First, unlike Flicker, TEE does not directly use DRTM; instead, it is based on vDRTM, which is introduced in the present paper. Second, execution environment solely based on vTPM is not sufficient for our purpose because (1) vTPM is fundamentally based on SRTM and thus cannot provide a secure system environment without rebooting the VM, and (2) vTPM cannot tell whether a command is from an untrusted guest OS or from a trusted execution environment. These are actually the reasons DRTM was introduced and explain why we should utilize DRTM. Third, integration of vTPM and Terra [24] is also not sufficient for our purpose because of the following. (1) TEE can be invoked on demand but Terra was not designed to operate in this fashion. (2) TEE can implement fine-grained protection and attestation but Terra cannot achieve this without paying the price because of the following dilemma. If one runs multiple sensitive applications in a single VM, then one cannot achieve fine-grained protection and attestation; if one runs a single sensitive application in a single VM, then it is difficult to share data between the VMs that are associated with their own vTPM (e.g., how should one vTPM unseal the data sealed by another vTPM? This causes a flexible or scalable key management problem, which TEE does not suffer from).

While there have been some initial efforts at exploring secure cloud computing [22,29], to our knowledge, no system like TEE (or vDRTM) has been reported in the public literature. For example, Santos et al. [30] briefly discussed a trusted cloud computing platform, whereby service providers offer a closed box execution of guest VMs while allowing the customers to determine whether the service is secure before the launch of their VMs. Their platform does not consider DRTM, and they present neither the design details nor any implementation. Sadeghi [31] envisioned a security architecture in which multiple VMs on a single platform can have access to an isolated "security critical application" execution environment, but did not explore a concrete system design and implementation. In particular, nothing is said about how the environment serves multiple VMs on a VMM.

3. Background

The Xen architecture. The Xen hypervisor, or simply Xen, is a popular VMM. It has a privileged domain, called Domain 0, that can access the physical I/O resources and manages the user domain VMs [32]. We use paravirtualized Xen and the following communication mechanisms (cf. [33] for details).

- Hypercalls. Hypercall allows a user domain (i.e., a guest OS or TEE in this paper) to trap into Xen to execute privileged operations (in Ring 0).
- Split device drivers. To provide virtual devices for domains, Xen adopts the so-called *split driver model*, which, roughly speaking, splits a physical device driver into two parts: a backend driver and multiple frontend drivers. The backend driver is a channel between the physical device and multiple frontend drivers, and the frontend drivers are associated with respective domains.

TPM. TPM is a chip embedded into a platform motherboard. Its main purpose is to establish a chain of trust during the bootstrapping of the OS through the 24 Platform Configuration Registers (PCRs), which store the platform integrity measurements in the form of 160-bit SHA-1 digests. The following TPM functions are relevant to our system.

- PCR extend operation. This operation allows the storage of an unlimited number of measurements in a specific PCR.

  $\mathrm{PCR}_{\mathrm{new}} = \mathrm{SHA\text{-}1}(\mathrm{PCR}_{\mathrm{old}} | \mathrm{measurement})$.

- Sealed storage. TPM sealing uses a non-migratable public key to encrypt the data, and bind the corresponding private key to the relevant PCRs. Security is ensured by that if the current platform state does not match the PCR values computed at the sealing time, the unsealing operation will fail and the data cannot be decrypted. An application running in TEE can use this mechanism to protect its confidential data from the guest OS.
- Attestation. The attestation function uses the private Attestation Identity Key (AIK) to sign the PCRs when attesting to a remote verifier, who can use the corresponding public key to verify the digital signature.

vTPM. vTPM [34] provides each VM on a single TPM-enabled platform with the functions offered by TPM. vTPM supports suspend and resume operations, as well as migration of a virtual TPM instance with its respective VM across platforms.

DRTM. Useful, secure bootstrapping ensured by TPM has had limited success (cf., e.g., [35–37]). The industry then introduced new CPU-based security technology to support DRTM. Unlike SRTM which forces the power-off before rebooting an (initially) trusted OS environment, DRTM can be launched at any moment in time and as often as needed, and can alleviate attacks such as TPM reset and BIOS attacks mentioned above. CPU vendors have implemented their hardware-based DRTM; Intel calls it Trusted Execution Technology (TXT) [38,39] while AMD calls it Secure Virtual Machine (SVM) [40]. The security guarantee offered by DRTM solely depends on the hardware, not even on the BIOS or bootloader. DRTM is invoked using a new CPU instruction (`SENTER` for Intel and `SKINIT` for AMD), which is the Dynamic Core of Root of Trust for Measurement (D-CRTM) for establishing a chain of trust during a late launch. Note that there are some differences between the implementations. When the `SKINIT` instruction is invoked, the only argument delivered is a physical memory address, which points to the region of protected code called Secure Loader Block (SLB). When the `SENTER` instruction is invoked, an Intel-signed code module, Authenticated Code Module (ACMod) [41], and an equivalent of AMDs SLB, Measured Launched Environment (MLE), are executed. Specifically, when the privileged instruction `SKINIT` (AMD) or `SENTER` (Intel) is executed, the following happens.

(i) CPU requests TPM to reset PCRs 17–23 to zero.
(ii) CPU measures SLB (AMD) or AC Module (Intel) and extends the measurement into PCR17.
(iii) Intel CPU measures MLE and extends the measurement into PCR18. AMD CPU skips this step.
(iv) CPU hands the control to SLB (AMD) or MLE (Intel).
(v) SLB or MLE creates the hardware-guaranteed trusted execution environment.

When the program running in the trusted execution environment exits, CPU returns the control to the next instruction in the program that invoked `SENTER`/`SKINIT`.

Locality. The introduction of DRTM has also brought the notion of locality in TPM specification version 1.2, in which memory mapped I/O is used. It reserves a range of physical memory that the host OS can map into its virtual address range. TPM memory map consists of five 4 kB pages, where each page corresponds to a locality. The purpose is to differentiate the sources of TPM-related requests: the trusted OS, the security domain the trusted OS created, the application running in that domain, or the CPU `SKINIT` or `SENTER` instruction. The specification assigns locality

0 for the legacy environment of SRTM (TPM specification version 1.1b), localities 1–3 for the trusted OS, the security domain, and the application, respectively, locality 4 for CPU's `SKINIT` [42]. Moreover, each locality is associated to a specific PCR; for example, PCR 17 is associated to locality 4 (the `SKINIT` instruction) and PCR 20 is associated to locality 1.

4. Problem definition

Why do we have to virtualize DRTM? Our solution to the problem of secure cloud-end computing aims to allow multiple customers or VMs on a commodity cloud-end platform to *simultaneously* enjoy DRTM-like secure execution environments, without requiring expensive extra hardware. Moreover, we want that the secure execution environments can be launched as needed and at any point in time for running sensitive applications while supporting attestations to remote parties. This immediately leads to the exploitation of both virtualization and trusted computing at the same time. In particular, this means that we have to virtualize DRTM because customers may need to *concurrently* invoke the secure execution environments *without* powering-off the computer.

Security assumptions. We assume:

- Attackers cannot launch sophisticated hardware attacks to break TPMs. This is a reasonable assumption because non-hardware attacks would be more relevant in cloud computing. Given that TPM is secure, it is also reasonable to assume that vTPM is secure as long as sufficient care is taken. Note that SRTM-based TPM (and thus vTPM) are vulnerable to TPM-reset and BIOS attacks (which may be caused by inappropriate implementation [43]), but these attacks can be defeated precisely using DRTM [37,44,45]. In the context of the present paper, these attacks are relevant to the bootstrapping of VMM but are defeated using DRTM (which is given in our system setting).
- VMM is secure. This assumption is made in most prior studies but is avoided in Flicker [18,19] (which however is not quite suitable for cloud computing). Nevertheless, VMM security is an active research area (e.g., [46]).
- The cryptosystems we use for authentication, digital signature, and encryption are secure in the standard sense [47]. We treat them as black-boxes.

Threat model. To accommodate the worst-case scenario, we consider the following attacks launched by an adversary who has corrupted some cloud customers and guest OS.

(1) Attack the cloud-end secure execution environment so as to compromise the honest customers' sensitive data.
(2) Attack the attestation of an honest customer's secure execution environment so as to cheat an honest remote verifier (e.g., the customer itself or a third party).
(3) Attack fairness so that honest customers' requests to launch secure execution environment are not faithfully accommodated, and that when an application runs in the secure execution environment, there is no or minimum side-effect on the other VMs on the same VMM.
(4) Abuse the secure execution environment to hide malicious activities (e.g., running malware) in the hope of evading from detection.

Functional requirements. The secure execution environment should satisfy the following functional requirements.

(5) Legacy code compatibility: It is desired that applications (as well as their underlying support software, if any) can run in the secure execution environment without forcing the developers to make modifications.

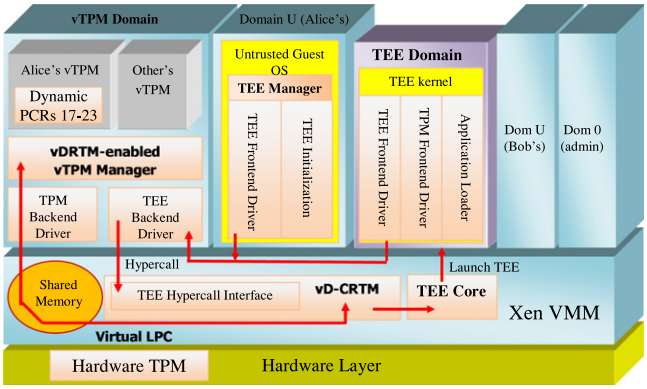

Fig. 2. TEE architecture has two parts: vDRTM (consisting of vDRTM-based vTPM Manager, Virtual LPC, and vD-CRTM) and TEE (consisting of TEE Manager, TEE Core, and TEE Domain). Arrows represent control flows.

(6) Fine-grained protection: This is useful because a whole application, or only a portion of it, may need to run in the secure execution environment. In the latter case, of course, the application needs to be modified to become aware of the secure execution environment.
(7) Low extra cost: The secure execution environment should only impose very low extra cost when compared with not using them (i.e. applications run in VMs). This makes TPM attractive (whereas co-processors are expensive).

5. TEE: design, implementation and analysis

5.1. TEE architecture

Overview. In order for a customer to run (on demand) a sensitive application in a secure execution environment that mimics the one enabled by DRTM, we virtualize DRTM. The resulting system architecture consists of two parts: vDRTM and TEE.

- vDRTM. It mimics the functions and services of DRTM while allowing multiple invocations, and has the following three components (which are highlighted in Fig. 2 and elaborated in Section 5.2).
  - vDRTM-enabled vTPM Manager (in vTPM Domain). It supports the control of *locality*, and is obtained by modifying the Xen vTPM Manager.
  - vD-CRTM (in VMM). It authenticates the hypercall for launching TEE, resets the corresponding vTPM's PCR 17–19 to zero, and extends the measurement of the TEE Kernel and the sensitive application into the PCR 17.
  - Virtual LPC. It is the virtual bus for communication between vD-CRTM and vTPM Manager. It mimics the LPC bus between CPU and TPM in a non-virtual system, which explains the term.
- TEE. It further consists of the following three components (see also Fig. 2).
  - TEE Domain. It is the trusted execution environment in which a sensitive application runs on top of TEE Kernel, which can be obtained by extending a desired or needed software system (ranging from pure cryptographic libraries to full-fledged OS as long as the system is deemed as trustworthy). The extension is to incorporate three modules:
    * TPM Frontend Driver. It allows TEE to have access to a vTPM. It is obtained by modifying the Xen TPM Frontend Driver.
    * TEE Frontend Driver. Through TEE Backend Driver (in vTPM domain), it allows TEE to send `exit` request to TEE Core (in Xen VMM) so as to destroy the TEE Domain and unpause the paused guest OS.

Table 2
Locality in DRTM and in vDRTM.

Localities	Used by (in DRTM)	Used by (in vDRTM)
Locality 4	Trusted hardware (DRTM)	vDRTM
Locality 3	Auxiliary components	Auxiliary components
Locality 2	"Runtime" environment for Trusted OS	TEE Kernel during launch
Locality 1	Trusted OS	TEE Domain after launch
Locality 0	Legacy environment for SRTM and its chain of trust	Untrusted guest OS

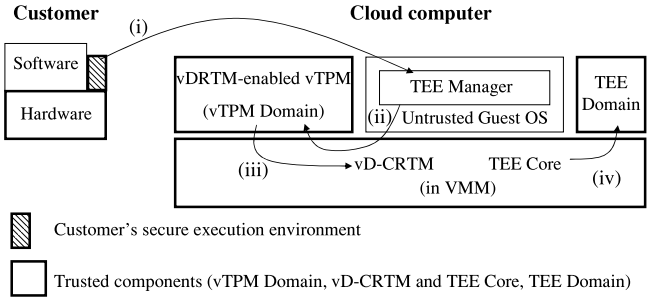

Fig. 3. High-level control flow for launching TEE: a running example (some detailed components are not shown for a better visual effect).

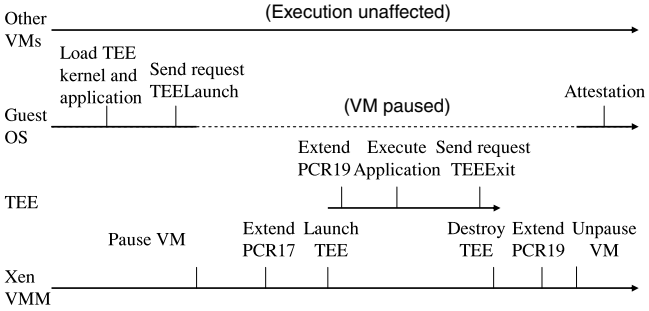

Fig. 4. Major events caused by TEE (the VM requesting TEE is paused when TEE runs; execution of the other VMs on the platform is unaffected).

- ∗ Application Loader. It loads (and decrypts, if needed) the sensitive application, and encrypts the output of the sensitive application (if any).

– TEE Manager (in guest OS). It is the interface between a customer and TEE, and consists of the following two modules (elaborated in Section 5.3).
  - ∗ TEE Initialization. This initializes the cryptographic mechanisms for authenticating the customers and the associated cryptographic keys.
  - ∗ TEE Frontend Driver. It loads both the TEE Kernel and the sensitive application into the guest OS memory space, and forwards to the vD-CRTM the request for launching TEE as well as the relevant parameters.
– TEE Core (in VMM). It uses Xen's `pause` command to pause the guest OS in which the TEE Manager initiated the request for launching TEE, launches the TEE Domain (TEE Kernel, which then loads the sensitive application), destroys the TEE Domain after the sensitive application exits, and uses Xen's `unpause` command to unpause the guest OS.

Running example. Before getting into the details, let us look at a running example (see also Fig. 3). A more detailed description of TEE life-cycle will be presented in Section 5.4.

(i) A customer sends TEE Manager in a VM (or guest OS) a request as well as parameters about the (encrypted) sensitive application.
(ii) TEE Manager allocates memory for the sensitive application and TEE Kernel, and requests the TEE Backend Driver (in vTPM Domain) to launch the TEE Kernel.
(iii) TEE Backend Driver (in vTPM Domain) uses a hypercall to request TEE Core to launch TEE.
(iv) TEE Core pauses the VM that issued the request; vD-CRTM authenticates the request and measures TEE Kernel and sends the measurement to vTPM Manager, which extends the measurement into PCR 17 in the VM's associated vTPM; TEE Core launches TEE Kernel, which loads the application. TEE Domain is then scheduled by Xen (as a replacement of the paused VM).

Exiting TEE Domain basically corresponds to the inversion of the above process. The major events caused by entering and exiting TEE are high-lighted in Fig. 4.

We note that the above process corresponds to the case that a customer runs a whole sensitive application in TEE Domain. It can be easily extended to accommodate the fine-grained protection that a portion of an application needs to run in TEE Domain. In this case, we can let the application contact the customer-end trusted execution environment, which then invokes TEE Domain in the cloud computer in the same fashion as illustrated above. The involvement of customer-end trusted execution environment is actually important because the calling application is not protected from the guest OS, which makes it possible that a malicious guest OS runs the sensitive portion of the application many times.

In what follows we elaborate vDRTM and TEE, and describe how they work together.

5.2. vDRTM

vDRTM Localities. Recall that TPM 1.2 uses five localities to distinguish the origin of a TPM command so as to determine whether it is trusted. As high-lighted in Table 2, we define five localities in vDRTM that are in parallel to their counterparts in DRTM. This is primarily implemented by vDRTM-enabled vTPM Manager, which ensures that only vD-CRTM is able to reset vTPMs PCRs 17–19. In particular, locality 4 is the only origin of command that can reset PCRs 17–19. This means that guest OS and TEE cannot reset these PCRs because they cannot use locality 4 (they can only use locality 0 and locality 1, respectively). Note that during the launch of TEE Kernel, it can use locality 2 and thus can release keys and data in the corresponding vTPM. TEE Kernel and sensitive application can use locality 1, and thus cannot release the keys and data associated to locality 2. Guest OS (locality 0) cannot release cryptographic keys associated to locality 1.

In our vDRTM implementation, we use flag `localityModifier` to indicate the currently active locality that can be used, and `TOSPresent`, which is inherited from TPM's specification version 1.2, to indicate the present TEE. As a result, when vDRTM serves a customer's request for launching TEE, it sets the `TOSPresent` flag to `TRUE`, which corresponds to the beginning of the chain of dynamic trust. The `TOSPresent` flag is very important because it can be used to defeat abnormal TPM reset (if reset by DRTM or vDRTM, PCRs 17–23 are set to 0 . . . 0; otherwise, they are set to 1 . . . 1).

vDRTM-enabled vTPM Manager. Recall that in the vTPM architecture [34], a vTPM instance is bound to a VM, and a vTPM Manager uses the vTPM instance number to communicate with the

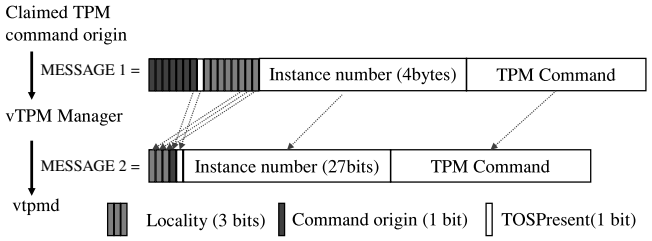

Fig. 5. Transferring locality.

corresponding vTPM instance (`vtpmd`). Unfortunately, for Xen Domain 0 running Linux-2.6.18-xen.hg we experimented with, the vTPM Frontend Driver and Backend Driver do not allow guest OS or VM to use any locality other than locality 0 because the flag `localityModifier` is always 0 and the flag `TOSPresent` is always `FALSE`. (Indeed, during the initialization of `vtpmd`, the two flags are set and never changed.) We found a patch [48] that can transfer vTPM locality information but only from TPM Frontend Driver to TPM Backend Driver. Unfortunately, it does not prevent TPM Frontend Driver from claiming an arbitrary locality to cheat the responding vTPM.

We solve the above problem by using the `TOSPresent` and `localityModifier` flags, and modifying and extending the TPM Backend Driver, vTPM Manager, and Xen vTPM. As shown in Fig. 5, for the first byte of MESSAGE1 received by vTPM Manager, the least significant bit is 0 means `TOSPresent = False` and 1 means `TOSPresent = True`, whereas the second least significant bit is 0 indicates that the message comes from TPM Frontend Driver and 1 indicates that the message comes from vD-CRTM. Moreover, the least significant three bits of the second byte indicate the locality (with 101–111 not used). vTPM Manager reformats the received message MESSAGE1 as shown in Fig. 5, and sends the reformatted message MESSAGE2 to `vtpmd`. This message reformatting operation is necessary because we want to avoid any change to the message format of `vtpmd` processes. Fortunately, this is made possible because vTPM Manager already needs to copy the 4 byte Instance number, but with the most significant five bits almost certainly unused (unless there are $>2^{27}$ vTPM instances on a single computer, which is almost certainly unlikely). Upon receiving MESSAGE2 from vTPM Manager, `vtpmd` extracts the relevant information as shown in Fig. 5 and executes Algorithm 1. Upon finishing the execution of Algorithm 1, `vtpmd` writes the response to vTPM Manager via the pipe:

`/var/vtpm/fifos/tpm_rsp_from_all.fifo`. Then, vTPM Manager gets the response, writes it to the shared memory, and uses a hypercall to notify vD-CRTM to get the response.

Algorithm 1 `vtpmd` locality processing

```
1: if received command origin == 1 then
2:   if received TOSPresent == 0 then
3:     set localityModifier = 000
4:   end if
5:   set TOSPresent as the received value
6: end if
7: if TOSPresent = 1 then
8:   set localityModifier as the received locality bits
9: end if
```

vD-CRTM. In parallel to D-CRTM's function reviewed in Section 3 (i.e., receiving the privileged instruction `SKINIT` or `SENTER` and measuring ACM or SLB and resetting the TPM, and extending the measurement into PCR 17), vD-CRTM is responsible for receiving and authenticating the hypercalls for launching TEE, measuring the TEE Kernel and application (possibly input as well) that will run in TEE Domain, commanding vTPM to reset PCRs 17–19, extending the aforementioned measurement into PCR 17, and finally handing over the control to TEE Core.

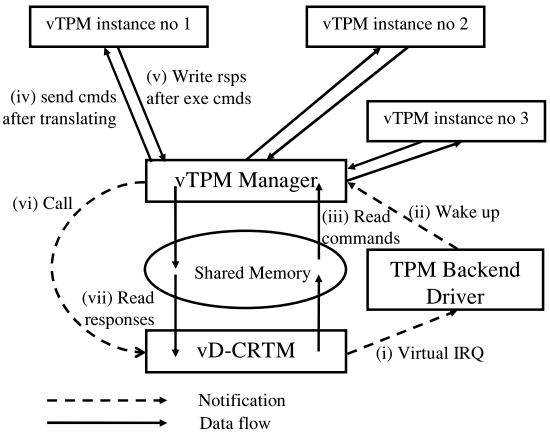

Fig. 6. Virtual LPC.

Virtual LPC. vDRTM must facilitate the communication between the vDRTM-enabled vTPM (in vTPM Domain) and vD-CRTM (in VMM), in a fashion similar to the communication between TPM and DRTM. A straightforward method is to establish a channel between vD-CRTM and each vTPM instance, which however will waste a significant amount of resource. Because vTPM Manager can already communicate with the vTPM instances, we build a virtual LPC between vD-CRTM and vTPM Manager, which is then multiplexed by all the vTPM instances.

Our Virtual LPC is implemented via shared memory between Xen and vTPM Manager. Virtual LPC is initialized by vTPM Manager, which uses the following hypercall to ask Xen to share one page of memory with it:

`void share_xen_page_with_guest(struct page_info *page, struct domain *d, int readonly)` Upon its creation, virtual LPC works as follows (see also Fig. 6).

(i) vD-CRTM writes a TPM command with `TOSPresent = true`, command origin, locality and Instance number (cf. Fig. 5) to the shared memory page, and then sends a virtual IRQ to TPM Backend Driver.
(ii) TPM Backend Driver wakes up vTPM Manager to read the message.
(iii) vTPM Manager determines if the command origin is TPM Frontend Driver or vD-CRTM Core. If it is vD-CRTM, vTPM Manager extracts the `TOSPresent`, locality, Instance number and TPM command, and translates MESSAGE1 into MESSAGE2 as shown in Fig. 5. If the command origin is incorrect (e.g., TPM Frontend Driver in the untrusted guest OS but outside the TEE Manager is compromised), vTPM Manager automatically corrects it.
(iv) vTPM Manager sends the received TPM command to `vtpmd` through pipe `/var/vtpm/fifos/tpm_cmd_to_%d.fifo`
(v) After `vtpmd` receives the command from the pipe, it extracts the locality information, `TOSPresent`, and command origin as shown in Fig. 5, and then executes Algorithm 1. Upon finishing the algorithm, `vtpmd` writes the response to vTPM Manager via pipe `/var/vtpm/fifos/tpm_rsp_from_all.fifo`
(vi) vTPM Manager gets the response, writes it to the shared memory, and uses our hypercall `vTPMManagerNotification` to notify vD-CRTM to get the response.
(vii) vD-CRTM reads the response from the shared memory.

5.3. TEE Manager

TEE Initialization. Each customer has two pairs of public–private keys; one for digital signatures and one for encryption. Let pk_{sign} denote the public key for verifying the customer's digital signatures, and pk_{enc} denote the public key by which the cloud computer can securely send information to the customer. We use two pairs of cryptographic keys for different purposes for the sake of prudent engineering. The measurement (hash) of these keys is securely stored by the cloud computing service provider. On the other hand, the cloud computing service provider generates a public key pk_{server} for the customer, which will be used for encrypting the sensitive application (as well as its input, if any) that needs to execute in a TEE environment. Note that the pk_{server} is unique to each customer for a security reason that will become clear later. Both the measurements (hash) of pk_{sign} as well as pk_{enc}, and the private key sk_{server} corresponding to pk_{server} are securely stored in the vTPM's non-volatile memory (NVM) associated with a VM when it is set up for the customer. Because the current Xen vTPM does not provide the NVM function, we utilized the implementation in the latest software-based TPMemulator's TPM_nv_storage.c [49]. The above design is specific to our prototype implementation, and can be adapted to accommodate any other desired designs. It is natural to assume that the above initialization operations, which are needed once per customer, are secure; in practice the initialization process would be vendor-specific.

TEE Frontend Driver. We need a mechanism for a customer to launch TEE through the relevant VM. Because we need to pause the VM before launching the TEE Domain, but Xen does not allow a domain to request to pause itself, we use Xen's backend–frontend mechanism to achieve this. Specifically, TEE Frontend Driver first allocates a piece of memory, loads the customer's public keys pk_{sign} and pk_{enc}, the TEE kernel and sensitive application and *client_nonce* (which is chosen by the customer for preventing the replay of past attestation) into this memory region, and then sends a request to TEE Backend Driver that will invoke a hypercall (which we named `TEELaunch`) to request vD-CRTM to launch TEE for the application and to share this memory region to TEE; this hypercall is the counterpart of AMD's `SKINIT` and Intel's `SENTER`. In the next subsection we will show how vDRTM interacts with TEE.

5.4. Putting the pieces together

Now we describe the detailed life-cycle of TEE by showing how vDRTM and TEE work together to authenticate a request, launch TEE, execute a sensitive application in TEE, exit TEE, and how the guest OS conducts remote attestation to prove that the request has been faithfully accommodated.

Step 1: Authenticating TEE launch request. Suppose we are about to authenticate the request for launching TEE (the request is delivered into vD-CRTM through the `TEELaunch` hypercall). To defeat replay attack, a fresh nonce is used as the challenge to the customer, which is a portion of the request that is digitally signed by the customer using the private key corresponding to pk_{sign}.

1. TEE Core uses Xen's `pause` to pause the VM that requested the TEE launch.
2. vD-CRTM verifies pk_{sign} and pk_{enc} against the measurement stored in the corresponding vTPM's NVM.
3. vD-CRTM checks the integrity of the TEE Kernel and the application in the shared memory against their certified hash.
4. If the verifications succeed, vD-CRTM uses virtual LPC to set `TOSPresent = true`, set `localityModifier = 4`, reset PCRs 17–19, and extend the hash of pk_{sign} and pk_{enc} to PCR 17.

Step 2: Preparing for launching TEE.

1. vD-CRTM lets the TEE Core take over the control. Because TEE inherits/reuses the vTPM associated with the VM, TEE Core asks the vTPM Manager to forward TEE's TPM commands to the guest OS' vTPM instance so as to allow TEE to use the instance.
2. vD-CRTM measures the TEE Kernel and extends the measurement into PCR 17.
3. TEE Core executes Xen's `create domain` to create a domain for the TEE Kernel and application.

Step 3: Execute sensitive application and exit.

1. Now TEE Kernel takes over the control and loads TEE Frontend Driver, TPM Frontend Driver, and Application Loader, which takes over the control, then reads sk_{server} from vTPM's NVM to decrypt the encrypted sensitive application, measures the application code (and its input), extends the measurement into PCR 19, resets PCR 20–22, then TPM Frontend Driver changes `localityModifier` to 1, which automatically modifies the current locality from 2 to 1, and finally transfers the control to the application.
2. When the application is finished and exits, the Application Loader takes over the control by letting TPM Frontend Driver changes `localityModifier` to 2, measuring the outputs and extends the measurement into PCR 19, copying the output (if any) and stored measurement log to the shared memory granted by TEE Frontend Driver (cf. Section 5.3). If the application has output, the Application Loader encrypts the output using pk_{enc} (so that the customer can obtain the output), erases the sensitive data in TEE, and sends through TEE Frontend Driver the request to TEE Backend Driver for exiting TEE.

Step 4: Resume guest OS.

1. TEE Backend Driver uses our `TEEExit` hypercall to inform vD-CRTM.
2. vD-CRTM extends *client_nonce* to PCR 17 so that the resumed guest OS cannot get any sensitive data that are sealed under PCR 17.
3. vD-CRTM sets `localityModifier` $= 0$ and `TOSPresent` $=$ `false`.
4. TEE Core destroys the TEE, and uses `unpause` to unpause the guest OS, which regains the control.

Step 5: Conduct remote attestation. To show that it has faithfully forwarded the TEE launch request, TEE Core sends command `TPMquote` to the vTPM, which then uses the attestation cryptographic key to sign PCRs 17 and 19. The verifier (the customer or a third party) can verify the signature using the corresponding key.

5.5. Security analysis of TEE

Assume that a customer has a secure execution environment in its local computer so that the private keys corresponding to pk_{sign} and pk_{enc} are never compromised, that TPMs and vTPMs are secure (this can be achieved using DRTM so that TPM reset and BIOS attacks are prevented), that VMMs are secure, and that the cryptosystems are secure. Note that we do *not* assume the TEE Manager is secure because it resides in guest OS, which could be malicious and thus compromise it. We show that the attacks specified in Section 4 can be defeated.

Attack against TEE so as to compromise the sensitive data. Because a successful attack at any stage of the life-cycle (relevant to TEE) of sensitive data can cause the compromise of the data, we show that the life-cycle is secure, including authentication of request, TEE launch, TEE runtime, and TEE teardown.

Authentication security. We argue that the authentication of requests (for launching TEE to execute sensitive applications) is secure, by showing that integrity of the customer authentication data is ensured and that integrity of the authentication procedure is assured.

- Integrity of customer authentication data. We claim that integrity of the customer authentication data, namely pk_{sign} and pk_{enc}, is ensured. Since we assumed that TEE Initialization is conducted in a secure environment, the hash of pk_{sign} and pk_{enc} is securely stored by cloud computing service provider. When loading them into vTPM's NVM (on demand), our TEE Initialization uses the `TPM_NV_DefineSpace` operation to set

  pcrInfoWrite → localityAtRelease
  = pcrInfoRead → localityAtRelease = 11110,

  which means that the corresponding area of NVM can be read/written by commands that use locality 1–4 (therefore, vD-CRTM and TEE which can use locality 4, 2 or 1), but cannot be read/written by commands that use locality 0 (i.e., the guest OS). As mentioned above, our vDRTM-enabled vTPM Manager can detect and fix lies about locality, and thus the guest OS can never tamper the integrity of pk_{sign} and pk_{enc} in vTPM's NVM.
- Integrity of the authentication procedure. Before vD-CRTM verifies the digital signature (which certifies the request) using pk_{sign}, TEE Core pauses the guest OS. Since vD-CRTM resides in VMM and is thus secure, the integrity of the authentication procedure is assured.

Security of TEE launch. We want to assure the integrity of TEE when it is launched. After vD-CRTM verifies the integrity of the request (including TEE Kernel, the sensitive application as well as its input), it sets `TOSPresent = true`, enters locality 4, and extends the measurement of the request into PCR 17 so that it can be used for remote attestation later. We observe that the above process cannot be performed by the guest OS because it can only use locality 0, locality 4 is required for writing to PCR 17, and the guest OS cannot use locality 4 without being detected (and fixed) by our vDRTM-enabled vTPM Manager. Since the guest OS has been paused, it cannot tamper the execution of TEE Kernel as well as the sensitive application.

Security of TEE Runtime. Since the guest OS has been paused, it cannot attack the newly launched TEE Domain. Other VM cannot attack the launched TEE Domain because they are isolated from the newly launched TEE by VMM. As such, the following operations will be done in a secure fashion. The TEE Kernel (more specifically the Application Loader) measures the sensitive application as well as its input and extends the measurement into PCR 19 for remote attestation later. If the application has output, secrecy of the output is ensured by encrypting it with a symmetric key k and AES-256 encryption (as an example), which means that the guest OS can only see the ciphertext. The k is encrypted using pk_{enc}, which allows the customer's local trusted execution environment to decrypt it using the corresponding private key. The ciphertext corresponding to the output is measured (hashed) so that the measurement is extended to PCR 19.

Security of TEE teardown. This has two aspects.

- Secure teardown of TEE Domain. Because the sensitive application, its input, its output, and k are erased before unpausing the paused guest OS, the guest OS is unable to get any useful information about these sensitive data after being unpaused.
- Secure switch of the vTPM used by TEE. We argue that the unpaused guest OS cannot attack the vTPM used by the just-destroyed TEE. After the Application Loader erases the application as well as its input and output, the TEE Frontend Driver (in TEE Kernel) uses a hypercall to request the TEE Backend Driver (in vTPM Domain) to exit TEE. Note that vD-CRTM extends *client_nonce* into PCR 17, which prevents future replay attack against remote attestation and simultaneously protects the information that has been sealed under PCR 17. After vD-CRTM sets `TOSPresent = false`, vTPM executes Algorithm 1 to set `locality = 0`. Since `TOSPresent = false`, vTPM will not accept any commands that attempt to set or modify `localityModifier` (until `TOSPresent` is set to `true` again by vD-CRTM for launching TEE). As a result, the guest OS always can only use locality 0, and thus cannot release any sensitive information associated to other localities.

Attack against attestation. We want to assure that if a vTPM attests that an application has been executed in TEE and an output was produced by the application, then that must have been the case. To break this assurance, there are three possible strategies. First, the attacker attempts to fake an attestation. This is infeasible because the vTPM's attestation key stays within the vTPM's security boundary. Second, the attacker attempts to cheat the vTPM into generating an attestation that is valid with respect to the *client_nonce*′ provided by the verifier. Because the guest OS can only use locality 0, it cannot extend the measurement of *client_nonce*′ into PCR 17. As such, the guest OS can obtain an attestation that is valid with respect to *client_nonce*′ only when *client_nonce*′ = *client_nonce*, where *client_nonce* is the last nonce whose measurement was extended into PCR 17 by vD-CRTM. This however happens with only a negligible probability. Therefore, the attack cannot succeed. Third, the attacker reuses or replays some past attestation. This is defeated by the use of a customer (or verifier) selected fresh *client_nonce*, which effectively prevents the attacker from reusing or replaying a past attestation.

Attack against fairness. We want to assure that a honest customer's request to run sensitive application in TEE is faithfully accommodated, and that the fact that one customer running TEE has little performance impact on the other customers' programs executing in their respective VMs or TEEs on the same platform.

- Unfaithful execution of sensitive application in TEE. A malicious guest OS may block an honest customer's request for executing sensitive application in TEE, while attempting to convince the customer that the application has been successfully executed. Note that the above attestation security assures that if vTPM attests the execution of a sensitive application, then that must have been the case (in part due to the use of fresh nonce *client_nonce* that is selected by the customer and must be included in the attestation). Since TEEs are scheduled by VMM, the request should be accommodated according to the VMM's policy. Therefore, the customer can always detect that unfaithful execution of his sensitive application.
  Note that it is possible that a malicious guest OS delays the forwarding of the request for executing sensitive application in TEE. However, this can be detected if the delay is significant. Moreover, this can be avoided by moving the TEE Manager from the user domain to, for example, a trusted domain (bootstrapped whenever VMM is up), which serves as the interface of all such requests (originated from customer's local trusted execution environment).
- Impact of TEE on other customers' VMs/TEEs. As we will report in the performance evaluation, the use of TEE does not have a significant performance impact on other customers. Intuitively, this is because in order to run an application in TEE, the corresponding VM is paused and then replaced by the TEE.

Abusing TEE. It would not be acceptable if a malicious party can abuse TEE to conduct malicious activities without being detected (e.g., running malware in TEE while evading from being detected).

There are two possibilities.

- Unauthorized use of TEE. If an unauthorized party can abuse TEE to conduct malicious activities, the problem is severe because we might not know who should be held accountable for the abuse. This can happen if a party can, using whatever means (e.g., compromising a guest OS as a stepping stone), successfully launch TEE without passing the authentication mechanism. For example, this is infeasible because in order to launch TEE, one must pass the authentication enforced by vD-CRTM, which is in VMM and secure. The "authentication security" discussed above assures that only authorized customers can launch TEE. As such, this type of attacks is prevented.
- Authorized customer abusing TEE. In the design of TEE we did not specify which applications can run in TEE because it is vendor-specific and orthogonal to the focus of the present paper. It would be ideal that only those certified (as trusted or trustworthy) applications are allowed to execute in TEE, which however may be over restrictive. As such, an authorized customer may be able to execute in TEE whatever he likes. Even though a TEE is isolated from other TEEs and VMs by VMM, it would be desired to at least mitigate such activities (e.g., the attacker can launch more stealthy side-channel attacks). To relieve this problem, there are two countermeasures depending on who provides the TEE Kernel. If the TEE Kernel is provided by the cloud computing service provider, we can extend the TEE Kernel to implement functions such as virus scanner or intrusion detection. If the TEE Kernel is provided by the customer (which is accommodated in our solution and advocated practice [50]), then we may need to run a monitoring program in, for example, Domain 0 so as to detect the execution of malicious program in TEE. This is in spirit parallel to the effort at monitoring the behavior within VMs because TEE actually resides outside any VMs. Monitoring cloud VMs is important and probably more difficult than in previously studied settings (e.g., VM introspection [51,52]) because the assumption – the VM and the VMM are owned by the same party – does not hold any more [53]. Thus, fully tackling this problem is left as future work.

5.6. Performance evaluation of TEE

To be useful, TEE should be efficient while having a low impact on the other VMs. We observe that TEE performance relies on the (size of) the TEE Kernel. To evaluate TEE's performance and impact on the other VMs, we use as a cloud computer an HP Compaq nc6400 with Intel Core Duo T2400 processor running at 1.82877 GHz, 2 GB RAM, and a v1.2 Infineon TPM. The VMM is vDRTM-enabled Xen v3.3.1 with Domain 0 running Ubuntu 8.04 (Linux 2.6.18.8-xen.hg kernel) as its guest OS, and vTPM Domain running Ubuntu 8.04 as its guest OS (the same kernel as Domain 0) allocated with one vCPU and 256 MB memory.

Performance of TEE. To evaluate the performance of TEE, we use a Domain U running Ubuntu 8.04 (Linux 2.6.18.8-xen.hg kernel) as Alice's VM, allocated with one vCPU and 256 MB memory. We use the TEE Manager to create a TEE Domain mini-OS (Linux 2.6.18.8-xen.hg kernel as the TEE kernel, which includes the TEE Frontend Driver, the TPM Frontend Driver and the Application Loader). The application is a standard AES-256 encryption program to encrypt a bitstring of 16 bytes for 256 $*$ 1024 times. The results reported in Table 3 are based on the average of 50 measurement runs, where time is measured by using the `RDTSC` instruction to count CPU cycles, which are then converted to milliseconds (ms) based on the machine's CPU speed (obtained by reading `/proc/cpuinfo`). We

Table 3
Time between launching TEE and resuming guest OS.

Operation	Time (ms)
TEE Frontend Driver sends the request	6.399
Check integrity of TEE Kernel and the application	7.806
Create TEE Domain	137
TEE executes application	18
Resume guest OS	7
Total	176

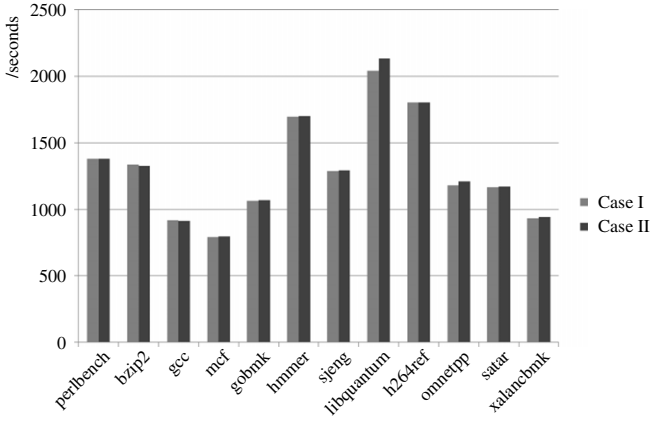

Fig. 7. Benchmark results.

draw the following observations.

- The time between TEE Frontend Driver sending the request and the creation of the TEE Domain is 14.205 ms. The TEE kernel is of 1,300,420 bytes and the AES encryption program is 22,605 bytes.
- It takes 173 ms to create the TEE Domain with one vCPU and 64 MB memory.
- The application running in TEE (mini-OS, one vCPU and 64 MB of memory) is a standard 256-bit AES encryption program. On average, the time consumed for encryption is 436.9 ms. To compare the application performance, we also performed the encryption test in Alice's DomU (Ubuntu 8.04, one vCPU and 256 MB of memory). The average encryption time is 452.6 ms. The application performances in TEE and DomA are almost the same. The application in TEE even runs a little faster because mini-OS is lighter than Ubuntu.
- Resuming guest OS takes 7 ms, which is much shorter than that of TEE creation because no authentication is needed.

Impact of TEE on the performance of the other VMs on the same platform. For this, we compare two cases.

- Case 1: Within Alice's VM we execute open source AES-256 encryption algorithm for encrypting 400 MB of data. Within Bob's VM we run 12 benchmarks from SPEC CINT 2006 (http://www.spec.org/cpu2006/CINT2006/) on top of Ubuntu 8.04 (guest OS) with 1 vcpu, 1024 MB of memory.
- Case 2: Alice uses TEE to execute the same encryption task, and we run the same benchmark program in Bob's VM.

As shown in Fig. 7, the only observable slow-downs caused by TEE are on `libquantum`, at the magnitude of about 4%, and on `omnetpp`, at the magnitude of 2%. The reason that TEE does not have significant impact on most benchmarks is that the VMM treats TEE as if it were a VM (in replacement of Alice's VM).

6. Conclusion and future work

We presented the design and implementation of TEE, a trusted execution environment for secure cloud(-end) computing because multiple users can simultaneously run their sensitive applications in their respective TEE's on a single computer platform. We presented a security argument about the security of TEE. Benchmark-based performance evaluation indicates that TEE does not have any impact on the performance of other VMs that do not use TEE, and that TEE does not have any significant performance side-effect when compared with running the same application in traditional VMs.

There are several interesting future research problems. First, how can we eliminate, or at least significantly mitigate, the threats of privileged insiders (cloud computer administrators) and the threats of side-channel attacks? Second, our argument for the security of TEE is informal. It is interesting to conceive some formal model that can be used to rigorously analyze security while accommodating the implementation details (e.g., locality control which has played an important role in our security argument).

Acknowledgments

This work is supported by National 973 Fundamental Basic Research Program under grant No. 2014CB340600 and National Science Foundation of China under grant No. 61272072. This paper substantially extended abstract published as [54].

References

[1] L. Atzori, A. Iera, G. Morabito, The Internet of things: a survey, Comput. Netw. 54 (15) (2010) 2787–2805.
[2] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (IoT): a vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.
[3] B. Gao, L. He, L. Liu, K. Li, S.A. Jarvis, From mobiles to clouds: developing energy-aware offloading strategies for workflows, in: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, IEEE Computer Society, 2012, pp. 139–146.
[4] J. Griffin, T. Jaeger, R. Perez, R. Sailer, L. van Doorn, R. Cáceres, Trusted virtual domains: toward secure distributed services, in: IEEE Workshop on Hot Topics in System Dependability, 2005.
[5] A. Bussani, J. Griffin, B. Jansen, K. Julisch, G. Karjoth, H. Maruyama, M. Nakamura, R. Perez, M. Schunter, A. Tanner, L. van Doorn, E.V. Herreweghen, M. Waidner, S. Yoshihama, Trusted virtual domains: secure foundation for business and it services, Tech. Rep., 2005.
[6] S. Cabuk, C. Dalton, H. Ramasamy, M. Schunter, Towards automated provisioning of secure virtualized networks, in: ACM CCS'07, 2007, pp. 235–245.
[7] S. Berger, R. Caceres, K. Goldman, D. Pendarakis, R. Perez, J. Rao, E. Rom, R. Sailer, W. Schildhauer, D. Srinivasan, S. Tal, E. Valdez, Security for the cloud infrastructure: trusted virtual data center implementation, IBM J. Res. Dev. 53 (4) (2009).
[8] P. Ruth, X. Jiang, D. Xu, S. Goasguen, Virtual distributed environments in a shared infrastructure, Computer 38 (5) (2005) 63–69.
[9] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS'07, 2007, pp. 598–609.
[10] G. Ateniese, R.D. Pietro, L. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, SecureComm'08, 2008, pp. 1–10.
[11] K. Bowers, A. Juels, A. Oprea, Proofs of retrievability: theory and implementation, Cryptology ePrint Archive, Report 2008/175, 2008. http://eprint.iacr.org/.
[12] K. Bowers, A. Juels, A. Oprea, Hail: a high-availability and integrity layer for cloud storage, Cryptology ePrint Archive, Report 2008/489, 2008. http://eprint.iacr.org/.
[13] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, Controlling data in the cloud: outsourcing computation without outsourcing control, in: Proceedings of ACM Cloud Computing Security Workshop, CCSW'09, 2008.
[14] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC'09, 2009, pp. 169–178.
[15] X. Chen, T. Garfinkel, E. Lewis, P. Subrahmanyam, C. Waldspurger, D. Boneh, J. Dwoskin, D. Ports, Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems, in: ACM ASPLOS XIII, 2008, pp. 2–13.
[16] D. Ports, T. Garfinkel, Towards application security on untrusted operating systems, in: HOTSEC'08.
[17] J. Yang, K. Shin, Using hypervisor to provide data secrecy for user applications on a per-page basis, in: Proceedings of the 4th International Conference on Virtual Execution Environments, VEE'08, 2008, pp. 71–80.
[18] J. McCune, B. Parno, A. Perrig, M. Reiter, A. Seshadri, Minimal TCB code execution, in: IEEE Symposium on Security and Privacy, SP'07, 2007, pp. 267–272.
[19] J. McCune, B. Parno, A. Perrig, M. Reiter, H. Isozaki, Flicker: an execution infrastructure for TCB minimization, in: ACM Eurosys'08, 2008, pp. 315–328.
[20] J. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, A. Perrig, TrustVisor: efficient TCB reduction and attestation, in: Proceedings of the IEEE Symposium on Security and Privacy, 2010.
[21] J. McCune, A. Perrig, M. Reiter, Safe passage for passwords and other sensitive data, in: NDSS'09, 2009.
[22] T. Ristenpart, E. Tromer, H. Shacham, S. Savage, Hey, you, get off of my cloud! Exploring information leakage in third-party compute clouds, in: Proceedings of ACM Conference on Computer and Communications Security, CCS'09, 2009.
[23] R. Ta-Min, L. Litty, D. Lie, Splitting interfaces: making trust between applications and operating systems configurable, in: OSDI'06, 2006.
[24] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh, Terra: a virtual machine-based platform for trusted computing, SIGOPS Oper. Syst. Rev. 37 (5) (2003) 193–206, SOSP'03.
[25] H. Chen, F. Zhang, C. Chen, R. Chen, B. Zang, P. Yew, W. Mao, Tamper-resistant execution in an untrusted operating system using a virtual machine monitor, Tech. Rep. 2007.
[26] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: a berkeley view of cloud computing, Tech. Rep., EECS Department, University of California, Berkeley, February 2009. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.
[27] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. de Lara, M. Brudno, M. Satyanarayanan, Snowflock: rapid virtual machine cloning for cloud computing, in: Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys'09, 2009, pp. 1–12.
[28] L. He, C. Huang, K. Duan, K. Li, H. Chen, J. Sun, S.A. Jarvis, Modeling and analyzing the impact of authorization on workflow executions, Future Gener. Comput. Syst. 28 (8) (2012) 1177–1193.
[29] M. Christodorescu, R. Sailer, D. Schales, D. Sgandurra, D. Zamboni, Cloud security is not (just) virtualization security, in: Proceedings of ACM Cloud Computing Security Workshop, CCSW'09, 2008.
[30] N. Santos, K. Gummadi, R. Rodrigues, Towards trusted cloud computing, in: Proceedings of the Workshop on Hot Topics in Cloud Computing, HotCloud'09, 2009.
[31] A. Sadeghi, Trusted computing—special aspects and challenges, in: Proceedings of 34th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'08, 2008, pp. 98–117.
[32] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: ACM SOSP'03, 2003, pp. 164–177.
[33] D. Chisnall, The Definitive Guide to the Xen Hypervisor, Prentice Hall, 2007.
[34] S. Berger, R. Cáceres, K. Goldman, R. Perez, R. Sailer, L. van Doorn, vTPM: virtualizing the trusted platform module, in: USENIX Security'06, 2006.
[35] J. Hendricks, L. van Doorn, Secure bootstrap is not enough: shoring up the trusted computing base, in: ACM SIGOPS European Workshop, 2004.
[36] R. Sailer, X. Zhang, T. Jaeger, L. van Doorn, Design and implementation of a TCG-based integrity measurement architecture, in: USENIX Security'04, 2004.
[37] B. Kauer, Oslo: improving the security of trusted computing, in: USENIX Security'07, 2007.
[38] D. Grawrock, The Intel Safer Computing Initiative: Building Blocks for Trusted Computing, Intel Press, 2006.
[39] Intel, Intel trusted execution technology mle developer's guide, June 2008. http://www.intel.com/technology/security/.
[40] AMD, Amd64 virtualization: secure virtual machine architecture reference manual, AMD Publication No. 33047 Rev. 3.01, May 2005.
[41] J. McCune, B. Parno, A. Perrig, M. Reiter, A. Seshadri, How low can you go?: recommendations for hardware-supported minimal TCB code execution, SIGARCH Comput. Archit. News 36 (1) (2008) 14–25.
[42] TCG, Pc client specific TPM interface specification (TIS). version 1.2, revision 1.00, July 2005. http://www.trustedcomputinggroup.org.
[43] A. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, M. Winandy, TCG inside?: a note on TPM specification compliance, in: Proceedings of the First ACM Workshop on Scalable Trusted Computing, STC'06, 2006, pp. 47–56.
[44] E. Sparks, A security assessment of trusted platform modules, Dartmouth Computer Science Technical Report TR2007-597, June 2007.
[45] J. Cihula, Trusted boot: verifying the xen launch, Fall 2007.
[46] D. Murray, G. Milos, S. Hand, Improving xen security through disaggregation, in: ACM VEE'08, 2008, pp. 151–160.
[47] O. Goldreich, The Foundations of Cryptography, vol. 1, Cambridge University Press, 2001.
[48] P. England, B. Lampson, J. Manferdelli, M. Peinado, B. Willman, A trusted open platform, Computer 36 (7) (2003) 55–62.
[49] M. Strasser, H. Stamer, J. Molina, Software-based TPM emulator for unix, March 2009. http://tpm-emulator.berlios.de/.
[50] J. Wei, X. Zhang, G. Ammons, V. Bala, P. Ning, Managing security of virtual machine images in a cloud environment, in: Proceedings of ACM Cloud Computing Security Workshop, CCSW'09, 2009.

[51] T. Garfinkel, M. Rosenblum, A virtual machine introspection based architecture for intrusion detection, in: NDSS'03, 2003.
[52] J. Pfoh, C. Schneider, C. Eckert, A formal model for virtual machine introspection, in: Proceedings of the 2nd Workshop on Virtual Machine Security, VMSec'09, 2009, pp. 1–10.
[53] L. Litty, H. Lagar-Cavilla, D. Lie, Computer meteorology: monitoring compute clouds, in: Proceedings of the 12th Workshop on Hot Topics in Operating Systems, HotOS'09, 2009.
[54] W. Dai, H. Jin, D. Zou, S. Xu, W. Zheng, L. Shi, TEE: a virtual DRTM based execution environment for secure cloud-end computing, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, ACM, 2010, pp. 663–665.

Weiqi Dai is working toward the Ph.D. degree in computer science and technology at Huazhong University of Science and Technology (HUST) in China. His research is focused on virtualization technology, trusted computing and cloud security.

Hai Jin is the Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. He is now the Dean of the School of Computer Science and Technology at HUST. Jin received his Ph.D. in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. He is a senior member of the IEEE and a member of the ACM. He has co-authored 15 books and published over 500 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Deqing Zou received the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST) in 2004. He is currently a professor of computer science at HUST. His main research interests include system security, trusted computing, virtualization, and cloud security.

Shouhuai Xu is a Professor in the Department of Computer Science, University of Texas at San Antonio. His research interests include security mechanisms and cybersecurity modeling & analysis. He earned his Ph.D. in Computer Science from Fudan University, China. More information about his research can be found at www.cs.utsa.edu/~shxu.

Weide Zheng received his BS degree (2009) and MS degree (2012) in Information Security from Huazhong University of Science and Technology (HUST), Wuhan, China. His main research interests include fault tolerance and security issues in cloud computing, virtualization and network. Currently, He is a software engineer at Baidu, Inc.

Lei Shi received his Master degree in Information Security from Huazhong University of Science and Technology (HUST), China and Master degree in Information Science from University of Michigan, USA.

Laurence Tianruo Yang graduated from Tsinghua University, China and got his Ph.D. in Computer Science from University of Victoria, Canada. He joined St. Francis Xavier University in 1999. He has published many papers in various refereed journals, conference proceedings and book chapters in these areas (including around 100 international journal papers such as IEEE Transactions on Computers, IEEE Journal on Selected Areas in Communications, IEEE Transactions on Parallel and Distributed Systems, IEEE Intelligent Systems, etc). He has been involved actively in conferences and workshops as a program/general/steering conference chair (mainly as the steering co-chair of IEEE UIC/ATC, IEEE CSE, IEEE HPCC, IEEE/IFIP EUC, IEEE ISPA, IEEE PiCom, IEEE EmbeddedCom, IEEE iThings, IEEE GreenCom, etc) and numerous conference and workshops as a program committee member. His current research includes parallel and distributed computing, embedded and ubiquitous/pervasive computing.